

LEARNING COLLECTIONS OF FUNCTIONS

Emmanouil Antonios Platanios

APRIL 2020

CMU-ML-20-103

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

THESIS COMMITTEE

Tom Mitchell, Chair
Graham Neubig
Rich Caruana (Microsoft)
Eric Horvitz (Microsoft)

*Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy.*

Copyright © Emmanouil Antonios Platanios

This research has been supported in part by the Air Force Office of Scientific Research under award FA95501710218, and in part by the National Science Foundation under award IIS1250956.

Keywords: multi-task learning, self-reflection, semi-supervised learning, never-ending learning, accuracy estimation, crowdsourcing, bayesian models, deep learning, contextual parameter generation, higher-order learning.

Learning Collections of Functions

Committee:

1. Tom Mitchell
2. Graham Neubig
3. Rich Caruana
4. Eric Horvitz

Date of submission: April 2020

Date of defense: April 2020

Please cite this document as:

Emmanouil Antonios Platanios. “Learning Collections of Functions.”

Carnegie Mellon University PhD Thesis. 2020.

URN: CMU-ML-20-103

URL: <http://reports-archive.adm.cs.cmu.edu/anon/ml2020/CMU-ML-20-103.pdf>

This document is provided by:

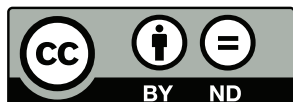
Machine Learning Department Reports,

SCS Technical Report Collection,

Carnegie Mellon University.

<http://reports-archive.adm.cs.cmu.edu>

reports@cs.cmu.edu



© 2020 by Emmanouil Antonios Platanios. This thesis is made available under a Creative Commons “Attribution — No Derivatives 4.0” license:

<https://creativecommons.org/licenses/by-nd/4.0/deed.en>.

*Dedicated to my loving family, who constantly inspire and motivate me to
push boundaries and pursue my dreams, no matter how far-fetched they may be.*

ABSTRACT

Human intelligence is magnificent. One of its most impressive aspects is how humans always seem able to learn new skills quickly and without much supervision by utilizing previously learned skills and forming connections between them. More specifically, human learning is often not about learning a single skill in isolation, but rather about learning collections of skills and utilizing relationships between them to learn more efficiently. Furthermore, these relationships may either be explicitly provided or implicitly learned, indicating high levels of abstraction in the learned abilities. On the other hand, even though machine learning has witnessed growing success across a multitude of applications over the past years, current systems are each highly specialized to solve one or just a handful of problems.

In this thesis, we argue that a computer system that learns to perform multiple tasks jointly and that is aware of the relationships between these tasks, will be able to learn more efficiently and effectively than a system that learns to perform each task in isolation. Moreover, the relationships between the tasks may either be explicitly provided through supervision or implicitly learned by the system itself, and will allow the system to self-reflect and evaluate itself without any task-specific supervision. This includes learning relationships in the form of *higher-order functions*—namely functions that compose, transform, or otherwise manipulate other functions—that can enable truly *multi-task* and *zero-shot learning*.

In the first part, we present a method that allows learning systems to evaluate themselves in an unsupervised manner by leveraging explicitly provided relationships between multiple learned functions. We refer to this ability as *self-reflection* and show how it addresses an important limitation of existing *never-ending learning* systems like the never-ending language learner (Mitchell et al., 2018). We then propose multiple extensions that improve upon this method, resulting in several robust algorithms for estimating the accuracy of classifiers from unlabeled data. In the second part, we consider more general multi-task learning settings and propose an abstract framework called *contextual parameter generation (CPG)*, which allows systems to generate functions for solving different kinds of tasks without necessarily having been shown any training data for these tasks. This framework generalizes existing approaches in multi-task learning, transfer learning, and meta-learning, and it further allows for learning arbitrary higher-order functions. It does so by formalizing the notion of a function representation and what it means for functions to operate on other functions or even on themselves. This new type of learning, which we refer to as *higher-order learning*, enables learning relationships between multiple functions in the form of higher-order functions, and is inspired by functional programming and category theory. Finally, we propose the *jelly bean world (JBW)*, a novel evaluation framework for never-ending learning systems.

ACKNOWLEDGEMENTS

First and foremost, I want to thank my advisor, Tom Mitchell, a brilliant and inspiring person without whom this thesis would not have been possible. Tom gave me not only freedom but also the necessary encouragement, to pursue my ideas even when they were unconventional and not necessarily directly related to his research interests. He encouraged me to pursue simple yet impactful ideas and offered me the support I needed to keep pushing during hard times. I remember one particular phrase that Tom said when I first started my PhD, and which I believe defines the goal of research very accurately and resonates strongly with my thoughts: *“When you were an undergraduate student your job was to answer questions well. Now, as a researcher, your goal is to ask the right questions.”* This phrase has stuck with me. Many thanks are also due to Graham Neubig, Rich Caruana, and Eric Horvitz for serving on my thesis committee, and to Yiannis Demiris, Christos Papavassiliou, Sotirios Chatzis, and Dimitrios Korkinof who cultivated my interest in research while at Imperial College London and motivated me to consider pursuing a PhD in the first place.

Throughout my PhD journey I have been fortunate to work with multiple accomplished professors and industry researchers including Alex Smola, Ashish Kapoor, Avrim Blum, Barnabàs Póczos, Eric Horvitz, Eric Xing, Graham Neubig, Hoifung Poon, and Rich Caruana. I want to especially thank Eric Horvitz for offering me a great opportunity to work together at Microsoft Research a few years back, and for his continuing support and valuable advice since. I will not forget the insightful, passionate, and visionary conversations we have had over the years. He is the kind of person with whom you sit down to discuss one idea and end up with ten new ideas by the end of the meeting, which is what successful research should be like. Alex Smola has also been very supportive and influential throughout my PhD journey and has offered me invaluable advice about both research and life, for which I will always be grateful. Alex is a very busy person who somehow always manages to find the time to sit down for a “quick” chat—whether at a coffee shop or his office—that typically ends up lasting for one to two hours, and that is overflowing with ideas and useful advice.

This thesis would not have been possible without my collaborators and co-authors. I want to especially thank Otilia Stretcu and Abulhair Saparov, two great and close friends with whom I have shared endless conversations over the last few years, and Maruan Al-Shedivat, George Stoica, Avinava Dubey, and Mrinmaya Sachan, who have also been great friends and co-authors. Moreover, my PhD would not have been successful without the multiple friends I made during this journey, including Brynn Edmunds and Daniel Bird, Chris Dann and Mariya Toneva, who aside from being good friends have also been great at organizing fun events for all of us, Willie Neiswanger, who is both a good friend and an inspiring person with an impressive amount of energy, as well as Aaditya Ramdas, Adarsh Prasad, Andrew Wilson, Ankur Parikh, Antonis Tasoglou, Arun Suggala, Ben Boecking, Ben Cowley, Ben Lengerich, Calvin McCarter, Chenghui Zhou, Dallas Card, Dan Schwartz, Danai Koutra, Gus

Xia, Ina Fiterau, Ivan Stelmakh, Jay-Yoon Lee, Jing Xiang, Junier Oliva, Kartik Goyal, Kirstin Early, Kirthevasan Kandasamy, Leila Wehbe, Leqi Liu, Lisa Lee, Manzil Zaher, Maria De-Arteaga, Matt Barnes, Micol Marchetti-Bowick, Nicole Rafidi, Nika Haghtalab, Nikos Katsipoulakis, Nikos Lappas, Peter Stojanov, Po-Wei Wang, Ritesh Noothigattu, Sashank Reddi, Simon Du, Vagelis Papalexakis, Vaishnavh Nagarajan, Veeru Sadhanala, Wei Dai, Xun Zheng, Yaoliang Yu, Yifei Ma, Yining Wang, and Yuxiang Wang. The never-ending learning research group also played an important role in helping me become familiar and contribute to a large ongoing project, when I first started my PhD. Specifically, I want to thank Jayant Krishnamurthy, who also helped me figure out what my next step is, Alan Ritter, Bill McDowell, Bishan Yang, Bhanava Dalvi, Bryan Kisiel, Derry Wijaya, Estevam Hruschka Jr, Forough Arabshahi, Igor Labutov, Justin Betteridge, Katie Mazaitis, Matt Gardner, Mehdi Samadi, Ndapakula Nakashole, Partha Talukdar, Richard Wang, Shashank Srivastava, and William Cohen.

This thesis would also have not been possible had it not been for the amazing environment of the machine learning department at Carnegie Mellon University. This is the result of the passionate work of many awesome professors and staff members, and especially so the work of Diane Stidle. Diane is an amazing person who not only takes care of almost everything, but is also one of the nicest people I know. I also want to thank Ameet Talwalkar, Andrej Risteski, Geoff Gordon, and Nina Balcan, with whom I have had multiple engaging and inspiring conversations.

On a separate note, I want to thank all the people who made my journey in developing TensorFlow Scala and other open source projects possible. Specifically, I am grateful to Alexandre Passos from Google who has always been open to discuss issues about the design of TensorFlow and resolve my concerns, and Skye Wanderman-Milne, also from Google, without whom control flow differentiation in TensorFlow Scala may have never happened. Finally, I am thankful to Chris Lattner, Brennan Saeta, Dan Zheng, Mingsheng Hong, Richard Wei, and the rest of the Swift for TensorFlow team, who included me in conversations about the project early on and encouraged me to be an active member of the team even though I was not actually working for Google.

The sponsors of this work have also been instrumental in making it possible. Specifically, I want to thank Doug Riecken from the Air Force Office of Scientific Research (AFOSR), the National Science Foundation (NSF), and Carnegie Mellon University for awarding me the Presidential Fellowship.

Last but not least, I want to thank my family who constantly inspire me and motivate me to push boundaries and pursue my dreams, no matter how far-fetched they may be. My parents instilled in me their passion for the sciences, and my brother's strength in fighting through difficult situations has been a constant source of inspiration and strength for me. I also thank my family for their unconditional love and patience at times when my research consumed all of my time and I would be embarrassingly slow at responding to their calls and messages. Finally, I want to thank my best friend and partner in crime, Otilia Stretcu, who has been there for me through both good and bad times and who has supported me in this journey, while being a constant source of happiness and love.

PUBLICATIONS

Parts of this thesis have previously appeared in the following publications:

- [1] **Emmanouil Antonios Platanios**, Avrim Blum, and Tom M. Mitchell. “Estimating Accuracy from Unlabeled Data.” In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2014.
- [2] Tom M. Mitchell, William W. Cohen, Estevam R. Hruschka Jr, Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhanava Dalvi, Matt Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir P Mohamed, Ndapakula Nakashole, **Emmanouil Antonios Platanios**, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. “Never-Ending Learning.” In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2015.
- [3] **Emmanouil Antonios Platanios**, Avinava Dubey, and Tom M. Mitchell. “Estimating Accuracy from Unlabeled Data: A Bayesian Approach.” In: *International Conference in Machine Learning (ICML)*. 2016, pp. 1416–1425.
- [4] **Emmanouil Antonios Platanios**, Ashish Kapoor, and Eric Horvitz. *Active Learning amidst Logical Knowledge*. 2017. arXiv: [1709.08850 \[cs.AI\]](#).
- [5] **Emmanouil Antonios Platanios**, Hoifung Poon, Eric Horvitz, and Tom M. Mitchell. “Estimating Accuracy from Unlabeled Data: A Probabilistic Logic Approach.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.
- [6] Tom M. Mitchell, William W. Cohen, Estevam R. Hruschka Jr, Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhanava Dalvi, Matt Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir P. Mohamed, Ndapakula Nakashole, **Emmanouil Antonios Platanios**, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. “Never-Ending Learning.” In: *Communications of the Association for Computing Machinery (CACM)* 61.5 (2018), pp. 103–115.
- [7] **Emmanouil Antonios Platanios**. *Agreement-based Learning*. 2018. arXiv: [1806.01258 \[cs.LG\]](#).
- [8] **Emmanouil Antonios Platanios**, Mrinmaya Sachan, Graham Neubig, and Tom M. Mitchell. “Contextual Parameter Generation for Universal Neural Machine Translation.” In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2018.
- [9] **Emmanouil Antonios Platanios** and Alex Smola. *Deep Graphs*. 2018. arXiv: [1806.01235 \[cs.LG\]](#).

- [10] **Emmanouil Antonios Platanios**, Otilia Stretcu, Graham Neubig, Barnabás Póczos, and Tom M. Mitchell. “Competence-based Curriculum Learning for Neural Machine Translation.” In: *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 2019.
- [11] Otilia Stretcu, Krishnamurthy Viswanathan, Dana Movshovitz-Attias, **Emmanouil Antonios Platanios**, Sujith Ravi, and Andrew Tomkins. “Graph Agreement Models for Semi-Supervised Learning.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.
- [12] **Emmanouil Antonios Platanios**, Maruan Al-Shedivat, Eric Xing, and Tom M. Mitchell. “Learning from Imperfect Annotations.” In: *Under Review*. 2020.
- [13] **Emmanouil Antonios Platanios**, Otilia Stretcu, Abulhair Saparov, and Tom M. Mitchell. “Learning When To Trust Your Neighbors: Robust Graph-Based Semi-Supervised Learning.” In: *Under Review*. 2020.
- [14] **Emmanouil Antonios Platanios***, Abulhair Saparov*, and Tom M. Mitchell. “Jelly Bean World: A Testbed for Never-Ending Learning.” In: *International Conference on Learning Representations (ICLR)*. 2020.
- [15] **Emmanouil Antonios Platanios***, Otilia Stretcu*, George Stoica*, Barnabás Póczos, and Tom M. Mitchell. “Contextual Parameter Generation for Knowledge Graph Link Prediction.” In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2020.
- [16] Otilia Stretcu, **Emmanouil Antonios Platanios**, Tom M. Mitchell, and Barnabás Póczos. “Learn to Walk Before You Run: Coarse-to-Fine Curriculum Learning for Classification.” In: *Under Review*. 2020.

Furthermore, working on this thesis resulted in the following open-source projects:

- [1] **Emmanouil Antonios Platanios**. *Makina*. <https://github.com/eaplatanios/makina>. 2015.
- [2] **Emmanouil Antonios Platanios**. *Symphony MT*. <https://github.com/eaplatanios/symphony-mt>. 2018.
- [3] **Emmanouil Antonios Platanios**. *TensorFlow Scala*. https://github.com/eaplatanios/tensorflow_scala. 2018.
- [4] **Emmanouil Antonios Platanios*** and Abulhair Saparov*. *Jelly Bean World*. <https://github.com/eaplatanios/jeally-bean-world>. 2018.
- [5] **Emmanouil Antonios Platanios**. *Swift ALE*. <https://github.com/eaplatanios/swift-ale>. 2019.
- [6] **Emmanouil Antonios Platanios**. *Swift Retro*. <https://github.com/eaplatanios/swift-retro>. 2019.
- [7] **Emmanouil Antonios Platanios**. *Swift RL*. <https://github.com/eaplatanios/swift-rl>. 2019.

CONTENTS

List of Figures	xvii
List of Tables	xxiv
List of Algorithms	xxvii
1 INTRODUCTION	1
1.1 Motivation	2
1.2 The Never-Ending Language Learner	4
1.3 Never-Ending Learning	5
1.4 Thesis Outline	7
1.5 How to Read this Thesis	10
I SELF-REFLECTION	11
2 ESTIMATING ACCURACY FROM UNLABELED DATA	13
2.1 Introduction	14
2.2 Problem Formulation	15
2.3 What Would a Human Do?	17
2.4 A Direct Approach	18
2.4.1 Key Idea	19
2.4.2 Optimization	24
2.5 Experiments	26
2.5.1 Results on Error Estimation	28
2.5.2 Results on Ground Truth Estimation	31
2.6 Limitations	32
2.7 Key Takeaways	34
3 A BAYESIAN APPROACH TO ESTIMATING ACCURACY	35
3.1 Introduction	35
3.2 A Simple Bayesian Model	36
3.3 A Coupled Bayesian Model	38
3.4 A Hierarchically Coupled Bayesian Model	42
3.5 Experiments	46
3.5.1 Results on Error Estimation	47
3.5.2 Results on Ground Truth Estimation	49
3.6 Key Takeaways	49
4 A LOGIC-BASED APPROACH TO ESTIMATING ACCURACY	51
4.1 Introduction	51
4.2 Probabilistic Logic	53
4.3 Proposed Logic Rules	55
4.3.1 Ensemble Rules	55
4.3.2 Constraint Rules	56
4.4 Inference	57

4.4.1	Probabilistic Soft Logic	58
4.4.2	Grounding	59
4.4.3	Optimization	60
4.5	Experiments	61
4.5.1	Evaluation Metrics	62
4.5.2	Results without Logical Constraints	62
4.5.3	Results with Logical Constraints	64
4.6	Key Takeaways	64
5	LEARNING FROM IMPERFECT LABELS	67
5.1	Introduction	68
5.2	Combining Bayesian Models and Deep Learning	71
5.2.1	Learning	72
5.2.2	Instance and Predictor Representations	74
5.2.3	Discussion	75
5.2.4	Extending to Multi-Label Settings	75
5.3	Experiments	76
5.3.1	Experimental Setup	76
5.3.2	Results	78
5.3.3	Ablation Study	79
5.3.4	Predictor Embeddings Visualization	81
5.4	Key Takeaways	82
6	LEARNING WHEN TO TRUST YOUR NEIGHBORS	83
6.1	Introduction to Graph Node Classification	84
6.1.1	Related Work	87
6.2	Proposed Algorithm	87
6.2.1	Model	88
6.2.2	Inference	89
6.2.3	Label and Confusion Priors	93
6.3	Experiments	95
6.3.1	Experimental Setup	97
6.3.2	Case Study #1: Enhancing Baselines	98
6.3.3	Case Study #2: Robustness Analysis	99
6.4	Key Takeaways	100
II	HIGHER-ORDER LEARNING	103
7	CONTEXTUAL PARAMETER GENERATION	105
7.1	Multi-Task Learning	106
7.1.1	A Brief History of Multi-Task Learning	107
7.1.2	The Current Landscape of Multi-Task Learning	108
7.1.3	When, How, and Why Does Multi-Task Learning Work?	111
7.2	Contextual Parameter Generation	113
7.2.1	Feeding Context as an Additional Input	115
7.2.2	Related Work	116
7.2.3	Another Perspective on Parameter Generation	117
8	CASE STUDIES FOR CONTEXTUAL PARAMETER GENERATION	119

8.1	Case Study #1: Parity Function	119
8.1.1	Recurrent Neural Networks	120
8.1.2	Multiplicative Recurrent Neural Networks	122
8.1.3	Contextual Recurrent Neural Networks	122
8.1.4	Experiments	124
8.1.5	Key Takeaways	124
8.2	Case Study #2: Machine Translation	125
8.2.1	Neural Machine Translation	126
8.2.2	Contextual Parameter Generation	129
8.2.3	Experiments	133
8.2.4	Related Work	137
8.2.5	Key Takeaways	138
8.3	Case Study #3: Knowledge Graph Link Prediction	139
8.3.1	Knowledge Graph Link Prediction	139
8.3.2	Problem Formulation	142
8.3.3	Limited Expressive Power	143
8.3.4	Contextual Parameter Generation	145
8.3.5	Experiments	150
8.3.6	Key Takeaways	156
8.4	Case Study #4: Jelly Bean World	156
8.4.1	Problem Setting	157
8.4.2	Reward Function	157
8.4.3	Models	158
8.4.4	Experiments	160
8.4.5	Key Takeaways	162
III EVALUATION		163
9	THE JELLY BEAN WORLD	165
9.1	Introduction	166
9.2	Design	168
9.2.1	Simulator	168
9.2.2	Environment	174
9.3	Learning Tasks	175
9.4	Experiments	176
9.4.1	Case Studies	176
9.4.2	Greedy Visual Agent	181
9.4.3	Relative Utility of Scent and Vision	182
9.4.4	Example of Spatial Non-Stationarity	184
10	CONCLUSION	187
10.1	Key Results	187
10.2	Future Work	188
10.3	Key Takeaways	189

IV	APPENDIX	191
A	COMPETENCE-BASED CURRICULUM LEARNING	193
A.1	Neural Machine Translation and Curriculum Learning	193
A.2	Competence-Based Curriculum Learning	195
A.2.1	Difficulty Metrics	196
A.2.2	Competence Functions	199
A.2.3	Scalability	201
A.3	Experiments	201
A.3.1	Setup	201
A.3.2	Results	202
A.3.3	Learning Rate Schedule	203
A.3.4	Implementation and Reproducibility	205
A.4	Related Work	205
A.5	Key Takeaways	206
B	ACTIVE LEARNING AMIDST LOGICAL KNOWLEDGE	209
B.1	Motivation	209
B.2	Related Work	210
B.3	Proposed Methods	211
B.3.1	A Simple Constraint: Mutual Exclusion	212
B.3.2	More General Logical Constraints	215
B.3.3	Computational Complexity	217
B.4	Experiments	217
B.4.1	Datasets	219
B.4.2	Evaluation Metrics	220
B.4.3	Results Analysis	220
B.5	Key Takeaways	222
C	NEURAL COGNITIVE ARCHITECTURES	223
C.1	Cognitive Architectures	224
C.2	The Hub-and-Spoke Theory	225
C.3	Proposed Architecture	226
C.4	Perception and Action Spokes	230
C.5	Reasoning Hub	231
C.6	Goal Contextualization	235
C.7	Learning Mechanisms	237
C.8	Next Steps	240
	BIBLIOGRAPHY	243

LIST OF FIGURES

Figure 1.1	Overview of the NELL learning algorithm. CPL and CML refer to different classifiers used by NELL. A detailed description is provided in Section 1.2	5
Figure 1.2	Overview of the work done as part of this thesis and of how the different parts relate to each other. Arrows indicate the influence and motivation for each project and dotted arrows indicate motivation alone without necessarily transferring parts of the proposed methods.	9
Figure 2.1	Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in Section 1.4	13
Figure 2.2	Illustration of the ways in which aggregating imperfect labels is an important problem. The left side shows the knowledge integration component of NELL, which was presented in Chapter 1 . The right side shows the two main ways in which training data can be collected for training large scale machine learning models.	15
Figure 2.3	Illustration of a motivating example on whether consistency among multiple functions (in this case humans) is related to their true accuracies.	17
Figure 2.4	Illustration of Equation 2.21	23
Figure 2.5	True errors (blue bars) versus errors estimated from unlabeled data using the DIRECT method (red bars), for four competing function approximations (ADJ, CMC, CPL, and VERB), to five different target function domains (i.e., bodypart, beverage, bird, person, and protein) using the NELL dataset. Note that each plot uses a different vertical scale to make it easier to observe the accuracy of the error rates estimates.	29
Figure 2.6	True errors (blue bars) versus errors estimated from unlabeled data using the DIRECT method (red bars), for eight competing function approximations (based on different story features), and three different target function domains (using neural activity from three different brain regions) using the brain dataset. Note that our error rate estimates from unlabeled data are quite close to the true error rates.	31
Figure 2.7	Accuracy of plain majority vote (MAJ) and majority vote weighted by the error rate estimates provided by the DIRECT method, for the two datasets we used in our experiments. A larger value corresponds to better performance.	32

Figure 2.8	Illustration of the three key limitations of the DIRECT approach. A detailed discussion of these limitations is provided in Section 2.6	33
Figure 3.1	Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in Section 1.4	35
Figure 3.2	Simple probabilistic model for error rate estimation using only unlabeled data.	37
Figure 3.3	Graphical model for coupled error rate estimation using only unlabeled data. The coupling comes from the use of a Dirichlet process prior to group problem domains and share information within each group. Note that, $\text{CRP}(\alpha)$ denotes the Chinese restaurant process (CRP) with concentration parameter α	41
Figure 3.4	Graphical model for hierarchical coupled error rate estimation using only unlabeled data. The hierarchical coupling comes from the use of a hierarchical Dirichlet process prior to cluster problem domains and functions and share information within each cluster. Note that, $\text{CRP}^d(\alpha)$ denotes a separate Chinese restaurant process (CRP) per domain d , with concentration parameter α	44
Figure 3.5	Mean squared error (MSE) of the estimated error rates for our methods (shown in red color) and the methods that we are competing against (shown in blue and green color). The lower the MSE (i.e., the shorter the bar), the better the result is. It is clear from these plots that our methods outperform the competing methods in all cases. Error bars have been omitted from these plots because they are negligibly small (~ 2 orders of magnitude smaller than the reported values). Note that these plots use a logarithmic scale.	48
Figure 3.6	Accuracy for ground truth estimation for our methods (shown in red color) and the methods that we are competing against (shown in blue and green color). Higher numbers imply better performance.	49
Figure 4.1	Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in Section 1.4	51

Figure 4.2	Proposed method overview. The classifier outputs and the logical constraints make up the system inputs. The representation of the logical constraints in terms of the function approximation error rates is described in Section 4.3 . In the logical constraints box, blue arrows represent subsumption constraints and red dashed lines represent mutually exclusion constraints. Given the inputs, the first step is grounding (i.e., computing all feasible ground predicates and rules that the system will need to perform inference over) and it is described in Section 4.4.2 . In the ground rules box, $\wedge$, $\neg$, and $\rightarrow$ correspond to the logical AND, OR, and IMPLIES operators, respectively. Then, inference is performed in order to infer the most likely truth values of the unobserved ground predicates, given the observed ones and the ground rules (described in detail in Section 4.4). The results include: (i) the estimated error rates, and (ii) the most likely target function outputs (i.e., combined predictions).	54
Figure 4.3	Results on the NELL dataset where no logical constraints are provided. For the first two rows, lower numbers imply better performance, but the opposite is true for the last row. . . .	62
Figure 4.4	Results on the brain dataset where no logical constraints are provided. For the first two rows, lower numbers imply better performance, but the opposite is true for the last row. . . .	63
Figure 4.5	Accuracy for ground truth estimation, for the two datasets we used in our experiments. Higher numbers imply better performance.	63
Figure 4.6	Illustration of the categories used in the two NELL datasets and the constraints between them. Each blue arrow represents a subsumption constraint, and each set of labels connected by a red dashed line represents a mutually exclusive set of labels. For example, <code>animal</code> subsumes <code>vertebrate</code> and <code>bird</code> , <code>fish</code> , and <code>mammal</code> are mutually exclusive.	64
Figure 4.7	Results on the NELL datasets where logical constraints are provided. For the first two columns, lower numbers imply better performance, but the opposite is true for the last column. . . .	65
Figure 5.1	Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in Section 1.4	67
Figure 5.2	Overview of the proposed inference algorithm and probabilistic model.	70
Figure 5.3	Illustration of our tensor factorization approach, modeling the interaction of instance difficulties with predictor competences.	72

Figure 5.4	Visualization of the learned predictor embeddings. Each dot in the plots represents a predictor projected on the 2D plane using UMAP (McInnes et al., 2018). In the left, the predictors are first clustered based on which instances they make mistakes on and then colored and shaped based on which cluster they belong to (in the embedding space, predictors that make similar mistakes tend to cluster together). In the middle, the predictors are colored based on their false positive rate. In the right, the predictors are colored based on their false negative rate.	81
Figure 6.1	Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in Section 1.4	83
Figure 6.2	Overview of GEM, an efficient graph-based semi-supervised learning algorithm.	85
Figure 6.3	Plots of accuracy at multiple ratios of the number of “disagreeing” edges to the total number of graph edges. These results are discussed in Section 6.3.3	100
Figure 7.1	Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in Section 1.4	105
Figure 7.2	The current landscape of multi-task learning	109
Figure 8.1	Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in Section 1.4	119
Figure 8.2	Illustration of the LSTM variants that we use in our parity function case study.	123
Figure 8.3	Results for the parity function experiments.	125
Figure 8.4	Overview of an NMT system, under our modular framework. Our main contribution lies in the parameter generator module (i.e., coupled or decoupled—each of the boxes with blue titles is a separate option). Note that g denotes a parameter generator network. In our experiments, we consider linear forms for this network. However, our contribution does not depend on the choices made regarding the rest of the modules; we could still use our parameter generator with different architectures for the encoder and the decoder, as well as with different kinds of vocabularies.	128
Figure 8.5	Illustration of the existing approaches to multilingual NMT.	130
Figure 8.6	Pairwise cosine distance for all language pairs in the IWSLT-15 and IWSLT-17 datasets. Darker colors represent more similar languages.	138

Figure 8.7	Overview of how our approach differs from prior work. Rather than stacking the relation with the source entity and transforming them jointly, the relation is used to generate the parameters of the model that is used to transform the source entity.	141
Figure 8.8	Toy example that cannot be modeled by additive interactions between entities and relations.	144
Figure 8.9	Overview of ConvE (top) and CoPER-ConvE (bottom).	149
Figure 8.10	Overview of MINERVA (top) and CoPER-MINERVA (bottom).	150
Figure 8.11	Time required for CoPER-ConvE to obtain its best performance on each dataset, as a fraction of the time it takes ConvE to achieve equivalent performance.	154
Figure 8.12	Heatmap showing the pairwise cosine distances between relation embeddings in NELL-995. Relations are grouped according to manually defined “types” which we have collected by manually annotating the dataset relations. Each block corresponds to a section denoted by these groups.	156
Figure 8.13	Example illustration of the jelly bean world. More details can be found in Chapter 9	158
Figure 8.14	Overview of the models we use in this case study. The gray blocks and arrows correspond to modules that are common across all models. The blue blocks and arrows correspond to modules that are only used by the REWARD AWARE model. The red blocks and arrows correspond to modules that are only used by the REWARD CONTEXTUAL model. The blocks and arrows that have red and blue stripes correspond to modules that used by both the REWARD AWARE and the REWARD CONTEXTUAL model. Arrows merging together correspond to concatenation of the respective tensors. All models are described in Section 8.4.3 . Note that, <code>gelu</code> refers to the Gaussian error linear units (GELUs; Hendrycks and Gimpel, 2016) activation function.	159
Figure 8.15	Experiment results along with key observations. The shaded vertical bars correspond to the two alternating reward functions that we use. The reward rate is computed using a 50-step window and the shaded bands correspond to standard error over 20 runs.	161
Figure 9.1	Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in Section 1.4	165
Figure 9.2	Overview of the modules comprising the Jelly Bean World.	168

Figure 9.3	Illustration of the procedural generation algorithm for the infinite world map. The 32×32 patches shown in white have already been sampled and those in gray have been sampled but not fixed in order to avoid boundary effects. The red line corresponds to an example path followed by an agent. Once the agent enters a patch that is not fixed, then that patch is sampled, along with its non-fixed neighboring patches in order to avoid boundary effects. For simplicity, no items are shown in this map.	171
Figure 9.4	Rendering of an agent’s perspective from the JBW visualizer.	171
Figure 9.5	Non-episodic experiment result. The reward rate is computed using a 100,000-step window and the shaded bands correspond to standard error over 20 runs. “Greedy Visual” refers to the reward rate obtained by the greedy visual agent described in Section 9.4.2	178
Figure 9.6	Non-stationary experiment results. The reward rate is computed using a 100,000-step window and the shaded bands correspond to standard error over 20 runs.	179
Figure 9.7	Multi-modal experiment results. The braces on the right specify the reward function used in each case. The reward rate is computed using a 100,000-step window and the shaded bands correspond to standard error over 20 runs. “Greedy Visual” refers to the reward rate obtained by the greedy visual agent baseline described in Section 9.4.2 (which, as explained in that section, is not able to handle visual occlusion).	180
Figure 9.8	Example showing one of the challenges the scent modality poses for agents.	182
Figure 9.9	Results of experiments showcasing the relative utility of scent and vision.	183
Figure 9.10	Visualization of an example environment with spatial non-stationarity. Each tile is colored according to its scent. JellyBeans are shown in blue, Bananas in green, and Onions in red. Walls are depicted as grey squares.	184
Figure A.1	Overview of the competence-based curriculum learning framework. During training, <i>difficulty</i> of each training example is estimated and a decision on whether to use it is made based on the current <i>competence</i> of the model.	194
Figure A.2	Example illustration of the preprocessing sequence used in the proposed algorithm. The histogram on the top is that of sentence lengths from the WMT-16 En→De dataset used in our experiments. Here sentence lengths represent an example difficulty scoring function, d . “CDF” stands for the empirical “cumulative density function” obtained from the histogram in the top plot.	196

Figure A.3	Example illustration of the training data “filtering” performed by our curriculum learning algorithm. At each training step: (i) the current competence of the model is computed, and (ii) a batch of training examples is sampled uniformly from all training examples whose difficulty is lower than that competence. In this example, we are using the sentence length difficulty heuristic shown in Equation A.1, along with the square root competence model shown in Equation A.8. . . .	198
Figure A.4	Plots of various competence functions with $c_0 = 0.01$ (initial competence value) and $T = 1,000$ (total duration of the curriculum learning phase). . . .	200
Figure A.5	Plots illustrating the performance of various models on the test set, as training progresses. Blue lines represent the baseline methods when no curriculum is used and red lines represent the same models when different versions of our curriculum learning framework are used to train them. The vertical lines represent the step in which the models attain the BLEU score that the baseline models attain at convergence. . . .	204
Figure B.1	Illustration of active learning in an interdependent multiple classifier setting. . . .	210
Figure B.2	Illustration of the categories used in the NELL datasets and the constraints between them. Each blue arrow represents a subsumption constraint, and each set of labels connected by a red dashed line represents a mutually exclusive set of labels. For example, <code>animal</code> subsumes <code>vertebrate</code> and <code>bird</code> , <code>fish</code> , and <code>mammal</code> are mutually exclusive. . . .	218
Figure B.3	Results. The red colored plots correspond to the proposed methods and the blue colored plots correspond to existing methods (apart from the constraint propagation step that we optionally added to all existing methods to enable a more fair comparison, and which is denoted by a -CP appended to the method name). For the first plot, the lower the bar, the better the result. For the rest of the plots, the higher the value of the curve per iteration, the better the result. We thus observe that the proposed methods outperform all existing methods for all of the experiments performed. . . .	221
Figure C.1	The original Hub-and-Spoke model (Rogers et al., 2004). . .	225
Figure C.2	Overview of the proposed neural cognitive architecture. . .	227

LIST OF TABLES

Table 2.1	The NELL categories used in our experiments.	27
Table 2.2	Mean absolute deviation (MAD) of individual, pairwise, and all function error rates for the NELL dataset, for all three proposed methods and for the cases where we use all of the available data samples and only 50 data samples per domain. The best score in each case is underlined and shown in red. .	29
Table 2.3	Mean absolute deviation (MAD) of individual, pairwise, and all function error rates for the brain dataset, for all three proposed methods, and for the cases where we use 4 classifiers and 11 classifiers. The best score in each case is underlined and shown in red.	31
Table 5.1	Statistics for the datasets we use in our experiments. “#Predictors” refers to the number of predictors in the dataset, “Average Redundancy” refers to the average number of predictions provided for each instance, “Average Accuracy” refers to the average predictor accuracy, and “Random Accuracy” refers to the accuracy obtained of a completely random predictor. . .	77
Table 5.2	Accuracy across varying levels of redundancy, for all datasets we used in our experiments. For each experiment, we report mean accuracy over 50 runs with different random initializations. We also compute standard error but it is generally too low and so it is not included in the table. The best results are underlined and shown in red color. The methods marked with a “★” are used for the ablation study of Section 5.3.3 . . .	79
Table 5.3	Results for LIA without marginal likelihood fine-tuning. . .	80
Table 6.1	Dataset statistics. We obtained Cora from Lu and Getoor (2003), CiteSeer from Sen et al. (2008), PubMed from Namata et al. (2012), and Disease from Chami et al. (2019). We use dataset train, validation, and tests splits provided by Yang et al. (2016). We decided to use these splits so that our results are comparable with numbers that were previously reported for existing methods.	95
Table 6.2	Test set accuracy for two baseline model architectures (discussed in Section 6.3.2). Shaded rows correspond to our method. The best result per row group is shown in bold red font and the best result across all rows is underlined.	96

Table 6.3	Test set accuracy for multiple existing methods, including the current state-of-the-art (discussed in Section 6.3.2). NGM was proposed by Bui et al. (2018), VAT and VATENT were proposed by Miyato et al. (2018), and we also previously proposed GAM (Stretcu et al., 2019). Shaded rows correspond to our method. The best result per row group is shown in bold red font and the best result across all rows is underlined.	97
Table 8.1	Number of learnable parameters for each model, for the IWSLT-17 experiments. “M” corresponds to “millions.”	134
Table 8.2	Comparison of the proposed approach (shaded rows) with the base pairwise NMT model (PNMT) and the Google multilingual NMT model (GML) for the IWSLT-15 dataset. The <i>Percent Parallel</i> row shows what portion of the parallel corpus is used while training; the rest is being used only as monolingual data. Results are shown for the BLEU and Meteor metrics. CPG* represents the same model as CPG, but trained without using auto-encoding training examples. The best score in each case is underlined and shown in red.	135
Table 8.3	Comparison of our proposed approach (shaded rows) with the base pairwise NMT model (PNMT) and the Google multilingual NMT model (GML) for the IWSLT-17 dataset. Results are shown for the BLEU metric only because Meteor does not support It, NL, and Ro. CPG ⁸ represents CPG using language embeddings of size 8. The “C ₄ ” subscript represents the low-rank version of CPG for controlled parameter sharing (see Section 8.2.2.1), using rank 4, etc. The best score in each case is underlined and shown in red.	136
Table 8.4	Dataset statistics. “#Train” denotes the number of questions used for training, N_e the number of distinct entities, N_r the number of distinct relations, $\bar{N}_a$ the average number of answers per question, and $\bar{d}$ the average degree of the graph nodes in the dataset.	151
Table 8.5	Results for multiple link prediction models. The results for ConvE, MINERVA, CoPER-ConvE, and CoPER-MINERVA are reported according to our own experiments. The rest of the results are taken from Das et al. (2018). All numbers are expressed as percentages. † denotes experiments performed using g_{linear} , and ‡ denotes those performed using g_{MLP} . “—” denotes missing results from the respective publications. Note that for MINERVA, we use the implementation provided by Lin et al. (2018), and report our results with the provided implementation for fair comparison with our CoPER extension. The best score in each case is underlined and shown in red.	153

Table 8.6	Results comparing different parameter generator networks. The results for ConvE, CoPER-PL-ConvE (using a parameter lookup generator function), and CoPER-ConvE are reported according to our own experiments. All numbers are expressed as percentages. $\dagger$ refers to the g_{linear} generator, while $\ddagger$ refers to the g_{MLP} generator. “—” denotes experiments outside our computational resource capabilities. The best score in each case is underlined and shown in red.	155
Table 9.1	Existing reinforcement learning environments positioned relative to our desired never-ending learning properties. . .	167
Table 9.2	Simulator configuration used in our experiments.	176
Table 9.3	Item types used in our experiments.	177
Table 9.4	Simulator configuration used for showing the relative utility of scent and vision.	182
Table 9.5	Item types for the simple environment where the scent modality dominates the vision modality.	183
Table 9.6	Simulator configuration used for our non-stationary environment.	184
Table 9.7	Item types for the non-stationary environment.	185
Table A.1	Number of parallel sentences in each dataset. “k” stands for “thousand” and “M” stands for “million.”	202
Table A.2	Summary of experimental results. For each method and dataset, we present the test set BLEU score of the best model based on validation set performance. We also show the relative time required to obtain the BLEU score of the best performing baseline model. For example, if an RNN gets to 26.27 BLEU in 10,000 steps and the SL curriculum gets to the same BLEU in 3,000 steps, then the plain model gets a score of 1.0 and the SL curriculum receives a score of $3,000/10,000 = 0.3$. Plain stands for the model trained without a curriculum and, for Transformers, Plain* stands for the model trained using the learning rate schedule shown in Equation A.10	203
Table B.1	Datasets used in our active learning experiments.	218
Table C.1	Example modalities. RNN stands for Recurrent Neural Network, CNN for Convolutional Neural Network, GAN for Generative Adversarial Network, MLP for Multi-Layer Perceptron, $\text{Scalar}[0, 1]$ for a single number in the interval $[0, 1]$, and $\text{Vector}[0, 1]$ for a vector containing numbers in the interval $[0, 1]$	230
Table C.2	Example uses of the problem compiler. We use c with different subscripts to denote context vectors representing primitives in the problem specification language, and g with different subscripts to denote transformation functions for context vectors (which could be defined as learnable neural networks, for example).	236

LIST OF ALGORITHMS

Algorithm 4.1	Grounding algorithm.	60
Algorithm 4.2	PSL consensus ADMM inference algorithm.	61
Algorithm 6.1	Initialization of the variational labels $\hat{\mathbf{y}}$	92
Algorithm 6.2	GEM inference algorithm.	93
Algorithm 9.1	Pseudocode for the greedy vision-based algorithm.	181
Algorithm A.1	Competence-based curriculum learning algorithm.	197

INTRODUCTION

Deep learning systems have become the de facto standard for solving prediction problems in a multitude of application areas including computer vision, natural language processing, and robotics. Driven by progress in deep learning, the machine learning community is now able to tackle increasingly more complex problems—ranging from multi-modal reasoning (Hu et al., 2017) to dexterous robotic manipulation (OpenAI et al., 2020)—many of which typically involve solving combinations of tasks. However, many real-world problems require integrating multiple, distinct modalities of information (e.g., image, audio, language) in ways that machine learning models cannot currently handle well. Most of these approaches are also limited in utilizing information learned from solving one problem to directly help in solving another—something at which human intelligence excels. There have been a few attempts to train a single model that solves multiple problems jointly (e.g., Kaiser et al., 2017), but the resulting systems generally underperform compared to those trained separately for each problem. Moreover, most of the existing approaches are also not capable of *never-ending learning* (NEL); namely a machine learning paradigm in which an algorithm learns from examples continuously over time, in a largely self-supervised fashion, and where its experience from past examples can be leveraged to learn future examples (Mitchell et al., 2018). Current machine learning systems typically underperform when the problems that need to be learned are not fixed a priori, but are rather dynamic and keep changing as part of the environment where the learning agents operate. For example, humans do not just learn to solve a fixed set of problems, but they rather adapt and by solving one problem, they become better able to tackle new problems that they may even have been previously unaware of. For example, after humans managed to build heart monitoring devices, new unsolved problems became available, such as discovering the relationship between heart rate or blood pressure and specific health problems. Furthermore, humans are capable of creating problems to learn, on their own, something that most current machine learning systems are not designed to achieve. Never-ending learning is thus something at which human intelligence excels, but machine learning is lacking. Due to the dynamic nature of the real world, to achieve true intelligence, a learning agent that interacts with it needs to be able to adapt in such a continuous fashion. In fact, such an ability is crucial for never-ending learning, because learning *forever* only really makes sense if the learning objectives are ever-evolving.

One of the most impressive aspects of human learning is how humans often seem able to learn new skills quickly and without much supervision, by utilizing previously learned skills and forming connections between them. For example, when humans learn to read they first learn to convert written language to spoken language by reading out loud. They then use their previously learned ability to understand spoken language in order to understand what they are reading. Eventually, the need to read out loud

vanishes and humans can understand written language directly (arguably, the human brain might simulate how words sound but there is no need for speech to be produced). This indicates that human learning is often not about learning a single skill in isolation, but rather about learning collections of skills and utilizing relationships between them to learn more efficiently. Furthermore, these relationships may either be explicitly provided or implicitly learned, indicating high levels of abstraction in the learned abilities. In this thesis, we argue that in order for computer systems to become capable of general intelligence they need to first become able to jointly learn multiple functions and account for relationships between them, that are either explicitly provided through supervision or implicitly learned. This includes learning relationships in the form of *higher-order functions*—namely functions that operate on other functions (e.g., compose, transform, or otherwise manipulate other functions)—that can enable truly *multi-task* and *zero-shot learning*. Intuitively, this can be attributed to the fact that a natural and human-inspired way to perform zero-shot learning is to take previously learned skills and combine them to form functions for solving new, previously unseen tasks. Thus, we aim to test the following hypothesis:

THESIS STATEMENT: *A computer system that learns to perform multiple tasks jointly and that is aware of the relationships between these tasks, will be able to learn more efficiently and effectively than a system that learns to perform each task in isolation. Moreover, the relationships between the tasks may either be explicitly provided through supervision or implicitly learned by the system itself, and will allow the system to self-reflect and evaluate itself without any task-specific supervision.*

Therefore, our goal is to show that learning collections of functions is more effective than learning single functions in isolation, and can also enable unsupervised evaluation which we call *self-reflection*. This is inspired by human learning and also motivated by our earlier work on never-ending learning. In the next section, we provide further insight into our motivation for this thesis. Then, we propose a definition for never-ending learning before describing the never-ending language learner (NELL), a system that we have been developing for multiple years and that also formed our original motivation for this work. Finally, we present the outline of this thesis in [Section 1.4](#), as well as advice on how to read it in [Section 1.5](#).

1.1 MOTIVATION

A long-standing goal in the fields of artificial intelligence and machine learning has been to develop algorithms that can be applied across domains and that can efficiently handle multiple problems, just like the human mind does. Even though research in this area, often called *multi-task learning*, has a long history (Caruana, 1997), there has recently been a resurgence of interest in fundamental questions related to:

- (i) algorithmic frameworks for multi-task learning, such as learning-to-learn or meta-learning (Thrun and Pratt, 1998; Finn et al., 2017; Franceschi et al., 2018) and never-ending or lifelong learning (Carlson et al., 2010; Mitchell et al., 2018),
- (ii) establishing best practices for building reliable systems that can handle multiple tasks at scale, such as federated learning for model personalization (Smith et al.,

- 2017) or multi-agent coordination (Cao et al., 2013; Samarakoon et al., 2018), and, last but not least,
- (iii) learning deep representations (Bengio et al., 2013) that support multi-tasking and enable transfer learning in multiple domains, such as computer vision (Yosinski et al., 2014) or natural language processing (Collobert and Weston, 2008; Peters et al., 2018; Devlin et al., 2019).

However, the notion of a *task* remains ill-defined and in fact, most existing machine learning systems that are branded as performing multi-task learning share an important common characteristic: all tasks share the same input space and are all being tackled using the same model (with a few exceptions; Ruder (2017) provides an extensive review). Furthermore, models that are pre-trained on large datasets and then fine-tuned in a task-specific manner, do not allow for information learned from one task to influence the learning of other tasks. These are all limitations that motivate this thesis and that we hope to address.

Our interest in these questions started while working on the never-ending language learner (NELL; Carlson et al., 2010; Mitchell et al., 2018). NELL is a system that learns to read the web and extract knowledge from websites, in a never-ending fashion. One of the core mechanisms employed in NELL is *co-training*, which was originally proposed by Blum and Mitchell (1998). Co-training is a semi-supervised learning algorithm where multiple models are being trained together and each model can use as training examples the most confident predictions made by the other models. If any of the models produces wrong but confident predictions, these can propagate to the other models and eventually hinder learning. This motivated us to develop several algorithms for estimating the accuracy of classifiers using only unlabeled data (Platanios et al., 2014, 2016, 2017b, 2020a). The key idea behind all these methods is that agreement among multiple models implies that the agreed upon prediction is more likely correct than wrong. However, we also observed that once we have multiple interacting tasks that are being learned jointly, we can perform accuracy estimation in a more robust manner by also accounting for inconsistencies between the tasks. For example, if one classifier predicts that “Pittsburgh” is a city and another one predicts that it is an animal, and we know that something cannot be both a city and an animal at the same time, then we can infer that at least one of these two classifiers must be wrong. Finally, this work pointed out an important pattern in how current machine learning systems are being trained. Training data is often obtained by collecting multiple noisy labels for samples through crowdsourcing which are then aggregated to produce a single “denoised” label per sample. To this end, we adapted our accuracy estimation methods resulting in a learning framework for general machine learning systems, that enables training from multiple imperfect labels directly—without requiring an explicit label aggregation step (Platanios et al., 2020a). Through this and other experiences we accumulated while working on NELL, we observed that: (i) learning multiple tasks jointly while also accounting for their interactions, and (ii) learning from multiple noisy sources of supervision, are both crucial to building successful never-ending learning systems. Given that NELL is so central to our motivation for this thesis, we provide a high-level overview of the system, in the next section.

1.2 THE NEVER-ENDING LANGUAGE LEARNER

The never-ending language learner (NELL; Carlson et al., 2010; Mitchell et al., 2015, 2018) is a system that learns to read the web and extract knowledge from websites, in a never-ending fashion. As mentioned in the previous section, one of the core mechanisms employed in NELL is *co-training*, which is a semi-supervised learning algorithm where multiple models are being trained together and each model can use as training examples the most confident predictions made by the other models. This allows us to utilize unlabeled data as the learning process is mostly self-supervised, thus enabling never-ending learning. At a high level, the NELL learning algorithm consists of the following steps:

1. NELL is initially provided a few facts that are known to be true (e.g., “New York” is a city)—the initial *knowledge base* (KB). It also has access to the world wide web—in practice, a large crawl of the web collected by Callan and Hoy (2009) and Callan (2012).
2. NELL consists of multiple classifiers, each one of which is targeted at modeling different kinds of information (e.g., context of noun phrases—if a phrase appears after the words “lives in,” then it is likely a city—or morphology of words—if a word ends with “-burgh,” then it is likely a city). During this step, these classifiers are all trained using the current version of the KB.
3. The classifiers are forced to make predictions over unlabeled data that NELL obtains from the world wide web. We refer to these predictions as *candidate beliefs* and they are known to be noisy. Especially early on in training, the classifiers will be far from perfect and thus many of these candidate beliefs are expected to be wrong.
4. NELL integrates the candidate beliefs into a small confident subset of beliefs that are treated as correct *facts* and are added back to the KB.
5. This process is repeated, starting at step 2, using the newly constructed KB as the source of training data for the classifiers.

This learning process is illustrated in Figure 1.1. The most crucial step in this process is step 4, which we refer to as *knowledge integration*, because adding wrong beliefs to the KB can result in degraded performance over time for the system. This is because these wrong predictions will be treated as ground truth in the next iteration and will be provided as training data to the classifiers. NELL is using multiple types of side information to reduce the likelihood of errors (e.g., logical constraints between the KB categories and relations). However, it relies on a set of manually curated heuristics for integrating this information and is thus not very robust. This motivated us to work on methods for estimating the accuracy of classifiers from unlabeled data and eventually, for directly learning from imperfect labels. This work forms Part I of this thesis and, as we explain in Chapter 2, the proposed algorithms for tackling this limitation of NELL have significant implications for several application areas of machine learning, beyond never-ending learning.

After having spent multiple years working on NELL and having gained a better understanding of what the main challenges are and what defines never-ending learning,

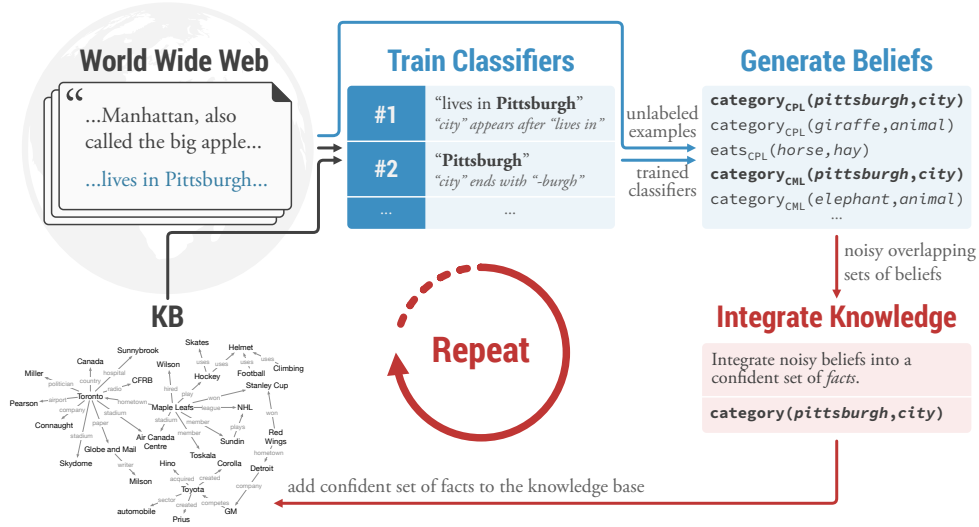


Figure 1.1: Overview of the NELL learning algorithm. CPL and CML refer to different classifiers used by NELL. A detailed description is provided in [Section 1.2](#).

we are now in a position to provide a more formal description of what we hope to achieve. In the next section, we present our definition of never-ending learning.

1.3 NEVER-ENDING LEARNING

As mentioned earlier, machine learning has witnessed growing success across a multitude of applications over the past years. However, despite these successes, current machine learning systems are each highly specialized to solve one or a small handful of problems. They have much narrower learning capabilities as compared to humans, often learning just a single function or model based on statistical analysis of a single dataset. One reason for this is that current machine learning paradigms are restricted and specialized to a particular problem and/or dataset. An alternative learning paradigm that more closely resembles the generality, diversity, competence, and cumulative nature of human learning is *never-ending learning* (Mitchell et al., 2018). The core thesis of never-ending learning is that *we will never truly understand machine learning until we can build computer programs that, like people: (i) learn many different types of knowledge or functions, (ii) from years of diverse, mostly self-supervised experience, (iii) in a staged curricular fashion, where previously learned knowledge enables learning further types of knowledge, and (iv) where self-reflection and the ability to formulate new representations and new learning tasks enable the learner to avoid stagnation and performance plateaus.*

Let us consider never-ending learning in the context of reinforcement learning, and let $s_t \in \mathcal{S}$ denote the state of the environment at time t , $a_t \in \mathcal{A}$ denote the action performed by the learning agent at time t , $\omega_t \in \Omega$ denote the observation of the world that the learning agent receives at time t , and $r_t \in \mathbb{R}$ denote the reward provided to the learning agent at time t . The distribution of the next state of the world

$s_t \sim T(s_{t-1}, a_{t-1})$ has the Markov property (i.e., it depends only on the previous state and action) and the initial state of the world is given by a distribution $s_0 \sim W$. The observation $\omega_t \sim O(s_t)$ depends only on the current state of the world (perhaps deterministically). The reward r_t is given by a function $R(s_{t-1}, a_{t-1}, s_t)$ of the current state, the previous state, and the previous action taken. The environment is a tuple containing all these elements $\mathcal{E} \triangleq (W, T, O, R)$. Then, the goal of reinforcement learning is to find a learning algorithm π that, given the history of previous observations, actions, and rewards, outputs the next action so that the obtained reward is maximized. We deliberately blur the distinction between the policy and the algorithm that learns the policy, which is why we call π a “learning algorithm.” As an example, if our goal is to maximize discounted future returns with discount factor γ , we want to find a π such that $\mathbb{E} [\sum_{t=0}^{\infty} \gamma^t r_t]$ is maximized, where the actions taken by the agent are provided by π , $a_t = \pi(a_0, \dots, a_{t-1}, \omega_0, \dots, \omega_t)$, and where the expectation is taken over the randomness in $\mathcal{E}$.

This formalism does not distinguish between learning algorithms that are highly specialized to a single task and learning algorithms that are capable of learning a wide variety of tasks and adapting to richer and more complex environments, which are hypothesized to be the hallmarks of *general intelligence* (see e.g., Lake et al., 2017). In order to more formally describe general intelligence, we posit that there exists an underlying measure of *complexity* of the environment $\mathcal{E}$ such that: (i) highly specialized and non-general learning algorithms can perform well in environments with low complexity, but (ii) environments with high complexity require successful learning agents to possess more general learning capabilities. It is in these more complex environments where we can characterize never-ending learning. We can formalize this notion of complexity by letting π^* be the (computable) learning algorithm that maximizes expected reward in an environment $\mathcal{E}$. Then, we define the complexity of $\mathcal{E}$ to be the length of the shortest program (i.e., Turing machine) that implements π^* :

$$\text{complexity}(\mathcal{E}) = \min\{|T| : T \text{ is a Turing machine that implements } \pi^*\}$$

$0 \quad \underbrace{\hspace{10em}}_{\pi^* \text{ exhibits specialized intelligence}} \quad \underbrace{\hspace{10em}}_{\pi^* \text{ exhibits generalized intelligence}} \rightarrow \text{complexity}(\mathcal{E})$

We can equivalently define $\text{complexity}(\mathcal{E}) = K(\pi^*)$, where $K(\cdot)$ is the *Kolmogorov complexity* and is related to the *minimum description length* and *minimum message length* (Kolmogorov, 1963; Nannan, 2010). As shown in the next page, the complexity of the environment $K(\mathcal{E})$ is bounded below by $K(\pi^*)$ minus a constant. Note here that we distinguish between $\text{complexity}(\mathcal{E})$ and $K(\mathcal{E})$ in order to handle cases where we can have very complex environments in terms of Kolmogorov complexity, but whose optimal agents are very simple.

Never-ending learning is in many respects similar to *lifelong learning*, also called *continual learning* (Chen and Liu, 2018). Like never-ending learning, lifelong learning is distinguished from multi-task learning by the never-ending nature of the learning problem. However, in never-ending learning, and unlike multi-task learning and lifelong learning, a well-defined set of tasks is not assumed a priori. Rather, never-ending learning is more similar to real-world settings in this respect, where the notion

KOLMOGOROV COMPLEXITY LOWER BOUND

We show that $K(\mathcal{E})$ must be at least $K(\pi^*)$ up to a constant. We can write an algorithm $\hat{\pi}$ that enumerates all possible sequences of environment states, observations, actions, and rewards from time t up to time T : $(s_t, \omega_t, a_t, r_t), \dots, (s_T, \omega_T, a_T, r_T)$. Then $\hat{\pi}$ computes the action that maximizes the expected reward $\mathbb{E}[\mathcal{R}(\{r_k\}_{k=t}^T)]$. Since the images $\text{im}(\hat{\pi}) \subseteq \mathcal{A}$ and $\text{im}(\pi^*)$ are discrete, and $\lim_{T \rightarrow \infty} \arg \max_{a_t} \mathbb{E}[\mathcal{R}(\{r_k\}_{k=t}^T)] = \arg \max_{a_t} \mathbb{E}[\mathcal{R}(\{r_k\}_{k=t}^\infty)]$, there is a sufficiently large finite T such that the action computed by $\hat{\pi}$ is the same as that computed by π^* . Note that $\hat{\pi}$ relies on a subroutine that simulates the environment $\mathcal{E}$ in order to first enumerate the environment states, and the subroutine to perform the optimization is independent of $\mathcal{E}$, and so $K(\hat{\pi}) = K(\mathcal{E}) + c$ for a constant c . Since $K(\pi^*) \leq K(\hat{\pi})$, we have that $K(\mathcal{E}) \geq K(\pi^*) - c$.

of a task or subtask naturally emerges from the complexity of the environment, and the distinction between tasks is not always so sharp. More specifically, in contrast to most popular reinforcement learning settings, never-ending learning focuses on environments with *high complexity*. In never-ending learning, agents can only exist in a single reset-free environment (i.e., we explicitly disallow the agent π from learning across multiple episodes or in multiple environments, which is closer to human learning). We require π to only have access to a single episode. During its lifetime, π can only use the information provided by its past observations $\{\omega_t\}$ and actions $\{a_t\}$ in a single world. Thus, never-ending learning explicitly removes the distinction between training and testing that is common to many other classical machine learning paradigms. Additionally, note that in the general reinforcement learning formalism, s_t can contain information about t , and the reward function R can be time-varying, thus rendering the environment non-stationary. In never-ending learning, we are interested in the full generality of non-stationary environments, as the stationarity assumption is not realistic in even simple adversarial and multi-agent settings. In [Part iii](#) we describe how to design and build environments that allow us to test for never-ending learning properties.

1.4 THESIS OUTLINE

This thesis is divided into three parts:

- **SELF-REFLECTION** ([PART I](#)): We design algorithms that enable learning systems to evaluate themselves in an unsupervised manner by leveraging explicitly provided relationships between multiple learned functions. We refer to this ability as *self-reflection* and show how it addresses the critical NELL limitation discussed in the previous section. In [Chapter 2](#), we design a method for estimating the accuracy of classifiers from unlabeled data that is based on an explicit relationship between the agreement rates of these classifiers and their error rates. This work also attempts to answer the question of *whether consistency implies correctness* and if so, under what conditions. Then, in [Chapter 3](#) we provide another method to solve the same problem that is also able to account for dependencies among the classifiers, and that is based on a probabilistic

graphical model. In [Chapter 4](#), we propose a method that is further able to account for logical constraints between the labels that the classifiers predict (e.g., a NP that refers to a city cannot also refer to an animal at the same time and therefore, if a classifier predicts that a given NP refers to a city and another one predicts that it refers to an animal, then at least one of them has to be wrong). In [Chapter 5](#), we propose yet another method for tackling this problem that offers a highly robust algorithm which further allows for learning directly from multiple noisy sources of supervision combined with self-supervision in an end-to-end fashion. Learning from imperfect labels is crucial to never-ending learning, as it is unreasonable to assume that a system will constantly have access to newly labeled training examples. This would be both very expensive and unreasonably demanding in terms of human labor. Therefore, a never-ending learning system will need to rely mostly on self-supervision and weak and potentially noisy supervision, similar to humans. Finally, in [Chapter 6](#), we show how this method and the underlying idea can also be used to tackle other problems that may at first seem completely disconnected from the original setting that motivated this work. Specifically, we show that it can be used for robust graph-based semi-supervised learning.

- **HIGHER-ORDER LEARNING** ([PART II](#)): We design algorithms that enable continuous learning of multiple problems that can grow in number over time. In [Chapter 7](#), we first provide some background on multi-task learning along with a brief history of the field, pointing out the most significant current limitations. Then, we propose an abstract framework called *contextual parameter generation* (CPG), which allows systems to generate functions for solving different kinds of tasks without necessarily having been shown any training data for these tasks. This framework not only generalizes existing approaches in multi-task learning, transfer learning, and meta-learning, but it also allows for learning arbitrary higher-order functions. It does so by formalizing the notion of a function representation and what it means for functions to operate on other functions or even on themselves. This new type of learning, which we refer to as *higher-order learning*, enables learning relationships between multiple functions in the form of higher-order functions, and is inspired by functional programming. Then, in [Chapter 8](#) we present strong empirical evidence for the power of CPG on a few different multi-task learning problems structured as a set of case studies with each one aimed at evaluating different aspects of CPG. The case studies we consider are focused on the following problems: (i) computing the parity function for sequences of arbitrary lengths ([Section 8.1](#)), (ii) machine translation ([Section 8.2](#)), (iii) knowledge graph link prediction ([Section 8.3](#)), and (iv) the jelly bean world ([Section 8.4](#)), which is proposed in [Part iii](#). In the last case study, we also provide the first instance of a *neural cognitive architecture* that is later used to motivate some of the future work we propose in [Chapter 10](#) and [Appendix C](#).
- **EVALUATION** ([PART III](#)): Building never-ending learning systems necessitates well-defined and robust ways to evaluate whether a system is indeed capable of never-ending learning. However, there are currently no ways to achieve

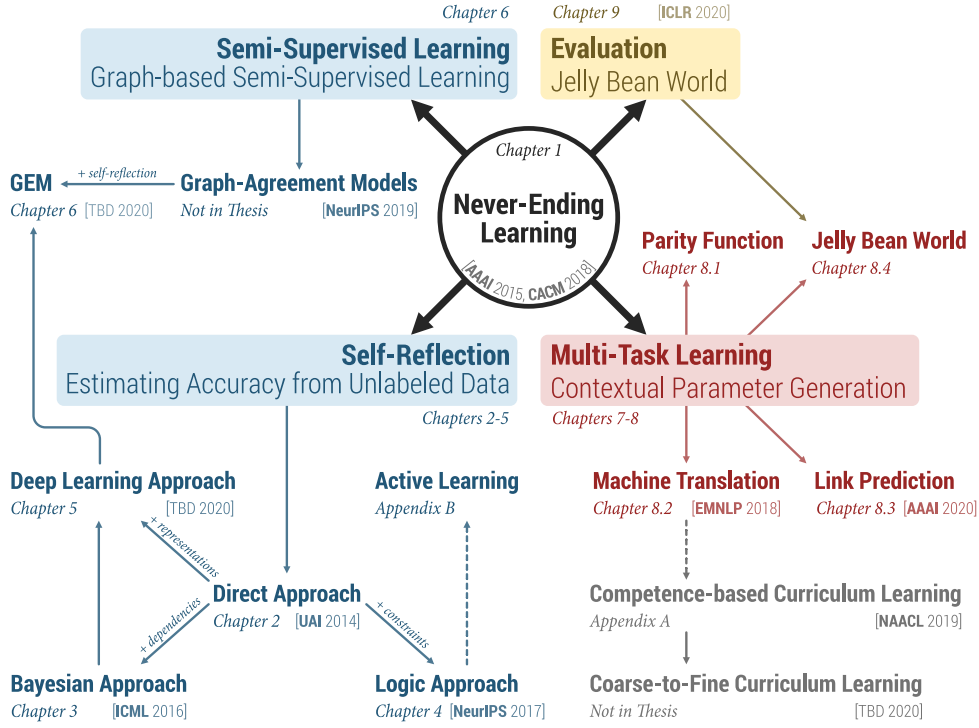


Figure 1.2: Overview of the work done as part of this thesis and of how the different parts relate to each other. Arrows indicate the influence and motivation for each project and dotted arrows indicate motivation alone without necessarily transferring parts of the proposed methods.

this. In [Chapter 9](#), we present the *jelly bean world (JBW)*, a novel framework that we designed for evaluating never-ending learning systems, and which allows us to control the kinds of problems the learning agents need to solve, and their interactions. We have designed the JBW in a way that renders never-ending learning necessary, and that allows us to test all parts of the never-ending learning thesis, in a controllable manner. We also use this framework to showcase the effectiveness of contextual parameter generation in settings where there exists a compositional structure over the tasks that are being learned.

An overview of the work done as part of this thesis and of how the different parts related to each other is shown in [Figure 1.2](#). Also, each chapter starts with an introductory paragraph and illustration that positions the content of that chapter relative to the rest of the work we present in this thesis. Finally, in [Chapter 10](#) we present a conclusion that summarizes the key results of this thesis. We also present the next steps we see in this line of work and the opportunities that this thesis opens up. Specifically, as the main next step we propose a novel family of architectures for never-ending learning, which we call *neural cognitive architectures* and which are inspired by human cognition. These architectures are presented in detail in [Appendix C](#). They are inspired from the *Hub-and-Spoke* model of human cognition (Rogers et al., 2004; Ralph et al., 2017) and account for human *goal-priming* (Custers and Aarts, 2005; Aarts et al., 2008; Takarada and Nozaki, 2018), by using CPG to emulate it. They contain perception and

action spokes (i.e., modules), and a common reasoning hub for all problems, that is independent of data modalities and which enables human-inspired capabilities such as *associative memory* (Fanselow and Poulos, 2005; Ranganath and Ritchey, 2012) and *world simulation*. Note that the design of a single universal neural cognitive architecture forms the most open-ended part of this thesis and it is meant to describe our way of thinking about the design space as a whole. Our main goal is provide some interesting research directions for which this thesis paves the way.

1.5 HOW TO READ THIS THESIS

This introduction provides an overview of the three parts that comprise this thesis. Each part is largely self-contained and mostly independent of the others. The same is generally true about each chapter except for the motivation that led to some of the work presented (e.g., the limitations of the work presented in [Chapter 2](#) motivate our work in [Chapters 3, 4, and 5](#)). The only exception to this is [Chapter 8](#), which we recommend reading after [Chapter 7](#), and [Section 8.4](#), which we recommend reading after [Chapter 9](#). Finally, the reader is not assumed to have detailed knowledge of machine learning or of the topics that are presented; we try to provide the necessary background, making this thesis accessible to graduate students.

Part I

SELF-REFLECTION

In [Chapter 2](#), we design a method for estimating the accuracy of classifiers from unlabeled data. This method is used to address a critical NELL limitation related to do with error propagation due to the co-training algorithm, which could degrade the long-term performance of the system and which was discussed in detail in [Chapter 1](#). Then, in [Chapters 3](#), [4](#), and [5](#), we propose three extensions that improve upon this method, resulting in several reliable algorithms for estimating the accuracy of classifiers from unlabeled data. The method described in [Chapter 5](#) offers a robust algorithm which also allows for learning directly from imperfect labels in an end-to-end fashion. Finally, in [Chapter 6](#), we show how this method and the underlying idea can also be used to tackle other problems that are very different from the original setting that motivated this work. Specifically, we show how it can be used for robust graph-based semi-supervised learning.

ESTIMATING ACCURACY FROM UNLABELED DATA

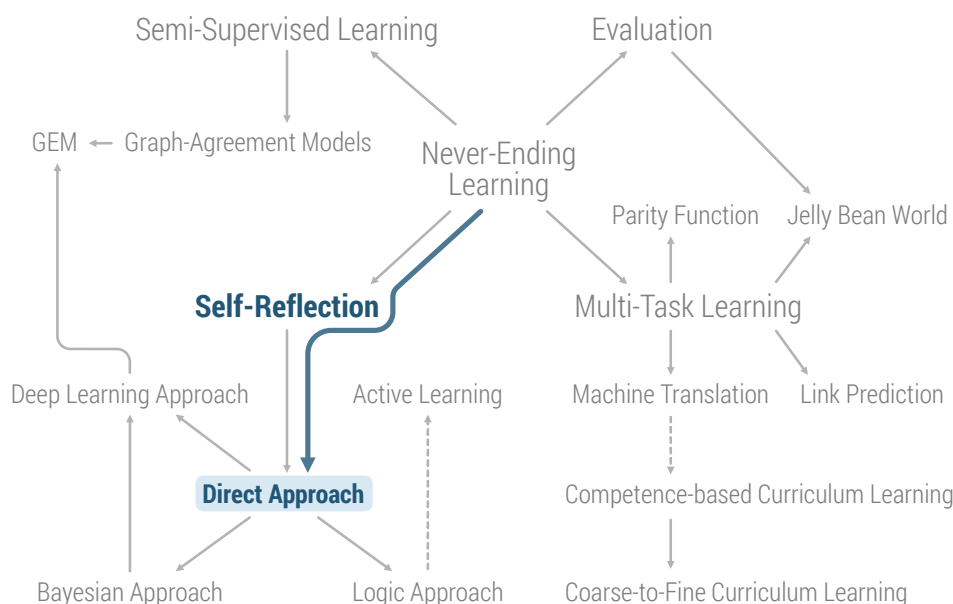


Figure 2.1: Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in [Section 1.4](#).

In [Chapter 1](#), we showed how never-ending learning forms the main motivation for this thesis. We also provided a high-level overview of the Never-Ending Language Learning (NELL) system, which is learning collections of functions jointly through co-training (a form of self-labeling). Self-labeling algorithms are highly prone to error propagation and that could ruin their long-term performance. Therefore, it is important to be able to estimate the accuracies of the functions they are learning using only unlabeled data; especially so in the case of NELL, which in some cases only has access to unlabeled data, and is learning continually in a never-ending fashion. Note that, although estimating accuracy without any labeled data may seem like an impossible task, we argue that humans do have a way of doing exactly that. In fact, in this chapter we present a direct approach to do this based on intuition about how we, as humans, would approach this problem. Our method showcases that learning collections of functions could allow us to perform completely unsupervised evaluation, which may at first sound impossible. Perhaps most importantly, we also show that under some independence assumptions, consistency among the learned functions (i.e., agreement in their predictions) implies correctness of their predictions.

2.1 INTRODUCTION

Estimating the accuracy of classifiers is central to machine learning and many other fields. Accuracy is defined as the probability of a system’s output agreeing with the true underlying output, and thus is a measure of the system’s performance. Most existing approaches to estimating accuracy are *supervised*, meaning that a set of labeled examples is required for the estimation. Being able to estimate the accuracies of classifiers using only unlabeled data is important for many applications, including: (i) any *autonomous learning system* that operates under no supervision (being able to perform this evaluation without supervision is also important for never-ending learning systems in general, because it offers a mechanism for ensuring that their performance does not deteriorate—or even plateau—with time), as well as (ii) *crowdsourcing* applications, where multiple workers provide answers to questions, for which the correct answer is unknown. The rising popularity and recent success of deep learning has resulted in machine learning systems that rely on large amounts of annotated training data (LeCun et al., 2015; Gulshan et al., 2016; Wu et al., 2016a; Esteva et al., 2017). The most common and scalable way to collect such large amounts of training data is through crowdsourcing (Howe, 2006). Crowdsourcing works well in simple settings where annotation tasks do not require domain expertise—for example, in object detection and recognition tasks in natural images and videos (e.g., Deng et al., 2009; Kovashka et al., 2016). However, annotation in specialized domains such as medical pathology requires a certain level of competency and expertise from the annotators which makes annotation expensive. Moreover, often times there is high rate of disagreement even between experts, which results in increasingly subjective and inconsistent labels (Elmore et al., 2015; Hutson et al., 2019). The two-sided motivation for this tackling this problem is illustrated in Figure 2.2.

A typical approach to dealing with subjectivity is to treat each annotation as simply noisy, collect multiple redundant labels per example (e.g., from different annotators), and then aggregate them using majority voting or other more advanced techniques (e.g., Dawid and Skene, 1979; Carpenter, 2008; Liu et al., 2012; Platanios, 2012; Zhou et al., 2015; Zhou and He, 2016) to obtain a single “ground truth” label. At the expense of redundancy, this results in better data quality and more accurate estimates of the ground truth. More recently, emerging systems for *data programming* and *weak supervision* also internally rely on label aggregation techniques similar to methods used for solving the crowdsourcing problem. For example, Snorkel (Ratner et al., 2017; Bach et al., 2019) is a popular data programming system that was designed for efficient and low-cost creation of large-scale labeled datasets using programmatically generated so-called *weak labels*. However, none of these systems solve label aggregation effectively in the presence of high subjectivity.

Given our initial motivation in the context of NELL, let us start by considering the problem of estimating the accuracy of classifiers using unlabeled data. Traditionally, one estimates accuracy of a function based on its performance over a set of labeled test examples. This chapter considers the question of under what conditions is it possible to estimate accuracy based instead on *unlabeled data*. We show that accuracy can be estimated exactly from unlabeled data in the case that at least three different

Example Task: Determine whether a noun phrase refers to a city or not.

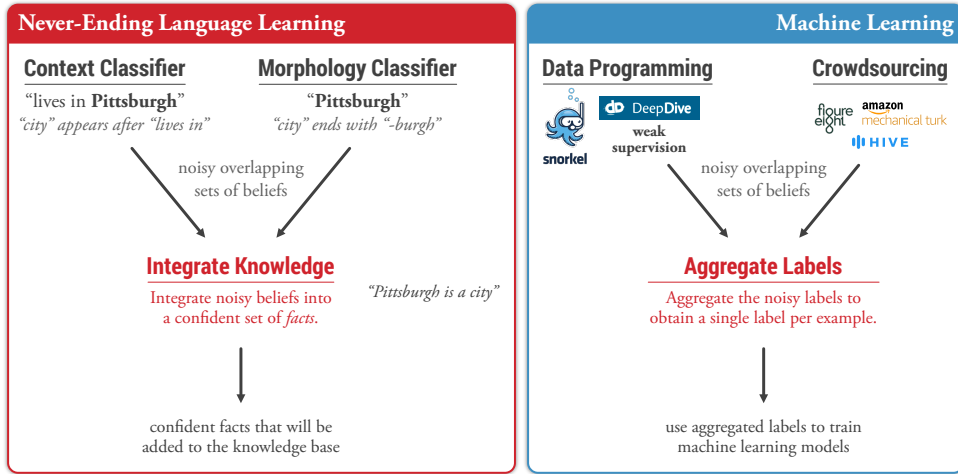


Figure 2.2: Illustration of the ways in which aggregating imperfect labels is an important problem. The left side shows the knowledge integration component of NELL, which was presented in [Chapter 1](#). The right side shows the two main ways in which training data can be collected for training large scale machine learning models.

approximations to the same function are available, so long as these functions make independent errors and have better than chance accuracy. More interestingly, we show that even if one does not assume independent errors, one can still estimate accuracy given a sufficient number of competing approximations to the same function, by viewing the degree of independence of these approximations as an optimization criterion. We also present experimental results demonstrating the success of this approach in estimating classification accuracies to within a few percentage points of their true value, in two diverse domains.

2.2 PROBLEM FORMULATION

We consider a “multiple approximations” problem setting in which we have several different approximations, $\hat{f}_1, \dots, \hat{f}_N$, to some target boolean classification function, $f : \mathcal{X} \rightarrow \{0, 1\}$, and we wish to know the true accuracies of each of these different approximations, using only unlabeled data. The multiple functions can be from any source—learned or manually constructed. One example of this setting that we consider here is taken from the Never Ending Language Learner (NELL; Mitchell et al., 2018). Out of NELL’s over 4,100 learning tasks, many involve learning classifiers that map noun phrases (NPs) to boolean categories such as `fruit`, `food`, and `vehicle`. For each such boolean classification function, NELL learns several different approximations based on different views of the NP. One approximation is based on the orthographic features of the NP (e.g., if the NP ends with the letter string “-burgh,” it may be a city), whereas another uses phrases surrounding the NP (e.g., if the NP follows the word sequence “lives in,” it may be a city). Our aim in this chapter is to find a way

to estimate the error rates of each of the competing approximations to f , using only unlabeled data (e.g., in the case of NELL, many unlabeled NPs).

Other researchers have considered variants of this “multiple approximations” setting. For example, Blum and Mitchell (1998) introduced the co-training algorithm which uses unlabeled data to train competing approximations to a target function by forcing them to agree on classifications of unlabeled examples. Others have used the disagreement rate between competing approximations as a distance metric to perform model selection and regularization (Bengio and Chapados, 2003; Schuurmans et al., 2006). Balcan et al., 2013 used disagreement along with an ontology to estimate the error of the prediction vector for multi-class prediction, from unlabeled data, under an assumption of independence of the input features given the labeling. Parisi et al., 2014 proposed a spectral method used to rank classifiers based on accuracy and combine their outputs to produce one final label, also under an assumption of independence of the input features given the labeling. Moreover, there has been work at developing more robust semi-supervised learning algorithms by using the concept of agreement rates (Collins and Singer, 1999) or some task specific constraints (Chang et al., 2007) to decide what should be added to the training dataset. However, very few have tried to directly estimate actual per function error rates using agreement rates. Dasgupta et al. (2001) PAC-bound the error rates using the pairwise agreement rates only, under the assumption that the functions make independent errors, and Madani et al. (2004) estimate the average error of two predictors using their disagreements. Donmez et al. (2010) are among the few to estimate per-function error rates from unlabeled data. Here, the authors estimate the prediction risk for each function under the assumption that the true probability distribution of the output labels is known. Much of the emphasis of their work is on methods that use the known label distribution to estimate the error rate even of a single classifier, but agreements are used as well, especially under the assumption of conditional independence. In contrast, we propose here a method for estimating actual function error rates from agreement rates, without making these assumptions. Finally, Collins and Huynh (2014) review many methods that have been proposed for estimating the accuracy of medical tests in the absence of a gold standard. This is effectively the same problem that we are considering, applied to the domains of medicine and biostatistics. They start by presenting a method for estimating the accuracy of tests, where the tests are applied in multiple different populations (i.e., different input data), while assuming that the accuracies of the tests are the same across the populations, and that the test results are independent conditional on the true “output label.” These are similar assumptions to the ones made by prior work, but the idea of applying the tests to multiple populations is new.

The main contributions of this chapter include: (1) formulating the problem of estimating the error rate of each of several approximations to the same function, based on their agreement rates over *unlabeled data*, as an optimization problem, (2) providing two different analytical methods that estimate error rates from agreement rates in this setting, one based on a set of simultaneous equations relating accuracies, agreements, and error dependencies, and a second, based on maximizing data likelihood, and (3) demonstrating the success of these two methods in two very different real-world problems. We consider the proposed methods a first step towards developing a self-

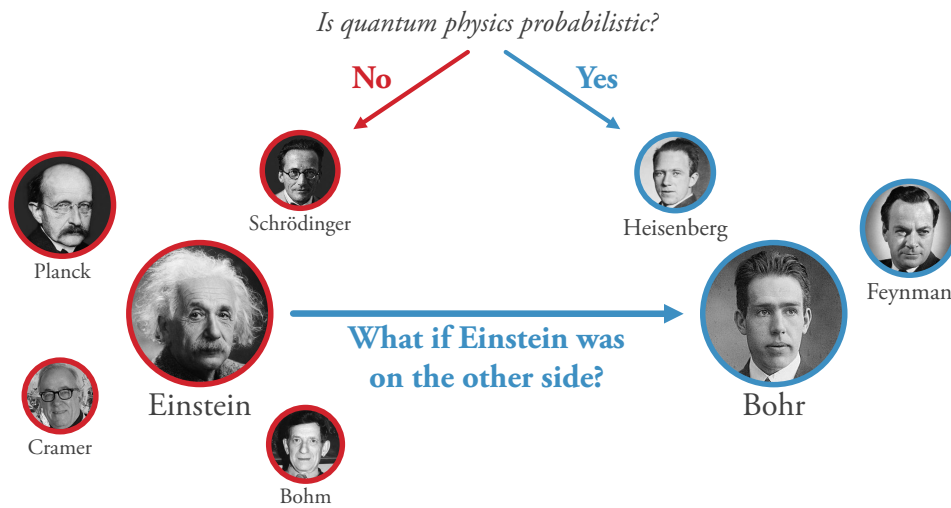


Figure 2.3: Illustration of a motivating example on whether consistency among multiple functions (in this case humans) is related to their true accuracies.

reflection framework for autonomous learning systems. Finally, we list three key limitations of the proposed methods that are addressed in the following chapters.

2.3 WHAT WOULD A HUMAN DO?

It can be observed from the previous section that most of the related work on estimating classifier accuracies with only unlabeled data is trying to relate agreement rates between different classifiers (which can be observed) with the accuracies of those classifiers. In fact, this is also the approach we take in this chapter. More specifically, one of our goals is to shed light on the more general question of *how the consistency among multiple functions is related to their true accuracies*. We are now going to present an example that will provide some intuition behind why one might want to use agreement rates as indicators of correctness, and what issues might arise if one does that.

Given that this thesis is largely inspired by human cognition and intelligence, let us think about a human would do in this situation. Let us consider a case where a person asks 10 different people a question that is related to politics and 8 of those people agree on an answer. One might immediately think that, since we have such a strong majority, the answer must be the correct one. However, one has to be careful. Let us assume that those 8 people that agree belong to the same political party and that the 2 people that gave a different answer belong to some other party. In this case, we might want to reconsider whether that answer is correct and the extent to which we trust it. Now, if 7 of the people from that party were in agreement and 1 person from the other party had also agreed with them, then maybe we should trust that answer even more. We therefore see that *the answer to the question of whether consistency implies correctness may have to do with how dependent the functions that agree with each other are*. One trivial example that reinforces this argument is when our multiple functions are in fact copies of the same function and thus fully dependent. In this

case, consistency among the functions gives us no information about their correctness (i.e., they are always consistent with each other in their responses). An illustration of an equivalent example to the one presented in this paragraph is shown in [Figure 2.3](#) (using a question related to quantum physics instead of politics).

One last thing to note about the example of the previous paragraph is that it raises a new question: *if functions that are highly dependent disagree, then what does that imply about the question being asked, or about the functions themselves?* In the case of asking people questions, such as in the example, it might imply that the question asked was subjective. We can extend this interpretation to classifiers by saying that maybe the provided classification problem is too hard, or the classifiers are too uncertain about their answers. This may be an interesting question to explore, but it is outside the scope of this thesis.

2.4 A DIRECT APPROACH

Based on the intuition gained from the example presented in the previous section, we now propose a direct method for estimating accuracies of classifiers from unlabeled data.¹ This method consists of matching the sample agreement rates of the functions with the exact formulas of these agreement rates written in terms of the functions' error rates, and it estimates the individual error rates for each function, as well as the joint error rates of all possible subsets of the functions, based on predictions they make over a sample of unlabeled instances $x_1, \dots, x_N$.

Let us denote the input data by $\mathbf{x} = \{x_1, \dots, x_N\}$ and the true binary output labels by $\mathbf{y} = \{y_1, \dots, y_N\}$. We assume that the input data $\mathbf{x}$ are sampled from some unknown distribution $p(\mathbf{x}) = \mathcal{D}$, and $y_i \in \{0, 1\}$, for $i = 1, \dots, N$. We consider M functions, $\hat{f}_1, \dots, \hat{f}_M$ which attempt to model the mapping from x_i to y_i , for $i = 1, \dots, N$. For example, each function may be the result of a different learning algorithm, or might use a different subset of the features of x_i as input. Furthermore, for convenience we also sometimes use the notation $\hat{y}_{ij} = \hat{f}_j(x_i)$.

Note that, in this case, the error event is defined as a function of some input x .

ERROR RATES. We define the error event $E_{\mathcal{A}}$ of a set of functions $\mathcal{A}$ as the event in which every function in $\mathcal{A}$ makes a mistake:

$$E_{\mathcal{A}}(x) = \bigcap_{j \in \mathcal{A}} [\hat{f}_j(x) \neq f(x)], \quad (2.1)$$

where $\cap$ denotes the set intersection operator and where $\mathcal{A}$ contains the indices of the functions. We define the error rate of a set of functions $\mathcal{A}$ (i.e., the probability that all functions in $\mathcal{A}$ make an error together) as:

$$e_{\mathcal{A}} = p_{x \sim \mathcal{D}}(E_{\mathcal{A}}(x)), \quad (2.2)$$

where $p_{x \sim \mathcal{D}}(\cdot)$ denotes the probability of an event that depends on x where x is drawn from distribution $\mathcal{D}$.

¹ This work has been published in (Platanios et al., 2014).

AGREEMENT RATES. We define the agreement rate $\alpha_{\mathcal{A}}$, for a set of functions $\mathcal{A}$ as the probability that all of the functions' outputs are the same:

$$\alpha_{\mathcal{A}} = \mathbb{P}_{\mathbf{x} \sim \mathcal{D}} \left(\{ \hat{f}_j(\mathbf{x}) = \hat{f}_k(\mathbf{x}), \forall i, j \in \mathcal{A} : j \neq k \} \right). \quad (2.3)$$

In contrast to the function error rates, this quantity can be directly estimated using only unlabeled data, by computing the following sample estimate:

$$\hat{\alpha}_{\mathcal{A}} = \frac{1}{N} \sum_{i=1}^N \mathbb{1}_{\{\hat{y}_{ij} = \hat{y}_{ik}, \forall j, k \in \mathcal{A} : j \neq k\}}, \quad (2.4)$$

where $\mathbb{1}_{\{\cdot\}}$ is the indicator function and has value 1 when the subscript condition is satisfied, and 0 otherwise. This is an unbiased estimate of the true agreement rate.

2.4.1 Key Idea

The key idea here is to notice that the agreement rate can be defined in terms of the error rates of the functions in $\mathcal{A}$. In order to understand how we can write the agreement rate in terms of error rates let us consider a simple example where $\mathcal{A} = \{j, k\}$ (i.e., consider only the pairwise agreement rate between functions f_j and f_k). The probability that two functions agree is equal to the probability that both make an error, plus the probability that neither makes an error (this is because these two events are mutually exclusive):

$$\alpha_{\{j, k\}} = \mathbb{P}_{\mathbf{x} \sim \mathcal{D}} \left(E_{\{j\}}(\mathbf{x}) \cap E_{\{k\}}(\mathbf{x}) \right) + \mathbb{P}_{\mathbf{x} \sim \mathcal{D}} \left(\bar{E}_{\{j\}}(\mathbf{x}) \cap \bar{E}_{\{k\}}(\mathbf{x}) \right), \quad (2.5)$$

where $\bar{\cdot}$ denotes the complement of a set. By using De Morgan's laws and the inclusion-exclusion principle we obtain an expression for the agreement rate between the two functions in terms of their individual error rates and their joint error rate (using the notation defined in [Equation 2.2](#)):

$$\alpha_{\{j, k\}} = 1 - e_{\{j\}} - e_{\{k\}} + 2e_{\{j, k\}}. \quad (2.6)$$

In the same way we can obtain the following general result for the agreement rate of a set of functions $\mathcal{A}$ of arbitrary size:

$$\alpha_{\mathcal{A}} = \mathbb{P}_{\mathbf{x} \sim \mathcal{D}} \left(\bigcap_{j \in \mathcal{A}} E_{\{j\}}(\mathbf{x}) \right) + \mathbb{P}_{\mathbf{x} \sim \mathcal{D}} \left(\bigcap_{j \in \mathcal{A}} \bar{E}_{\{j\}}(\mathbf{x}) \right), \quad \text{MUTUAL EXCLUSION} \quad (2.7)$$

$$= e_{\mathcal{A}} + 1 - \mathbb{P}_{\mathbf{x} \sim \mathcal{D}} \left(\bigcup_{j \in \mathcal{A}} E_{\{j\}}(\mathbf{x}) \right), \quad \text{DE MORGAN'S LAWS} \quad (2.8)$$

$$= e_{\mathcal{A}} + 1 + \sum_{k=1}^{|\mathcal{A}|} \left[(-1)^k \sum_{\substack{\mathcal{J} \subseteq \mathcal{A} \\ |\mathcal{J}|=k}} e_{\mathcal{J}} \right], \quad \text{INCLUSION-EXCLUSION} \quad (2.9)$$

where $\cup$ denotes the set union operator and $|\cdot|$ denotes the set cardinality. In the next section we examine the most basic case, assuming that the functions make independent errors and have error rates below 0.5. We show that in this case we can solve for the error rates *exactly*, provided that we have at least 3 different functions. In the subsequent section we examine the most general case, assuming that we have N functions that make errors with unknown inter-dependencies, and show that we can formulate this as a constrained numerical optimization problem whose objective function reflects a soft prior assumption regarding the error dependencies. Experimental results presented in [Section 2.5](#) demonstrate the practical utility of this approach, producing estimated error rates that are within a few percentage points of the true error rates, *using only unlabeled data*.

2.4.1.1 3 Functions That Make Independent Errors

When we have 3 functions that make independent errors we can replace the $e_{\{j,k\}}$ term in [Equation 2.6](#) with the term $e_{\{j\}}e_{\{k\}}$, for each pair, $\hat{f}_j$ and $\hat{f}_k$, of functions. In this case we have only 3 unknown variables (i.e., the individual function error rates) and we have $\binom{3}{2} = 3$ equations (i.e., [Equation 2.6](#) defined for $1 \leq i < j \leq 3$). Therefore, we can directly solve for each error rate in terms of the three observed agreement rates:

$$e_{\{j\}} = \frac{c \pm (1 - 2a_{\{k,l\}})}{\pm 2(1 - 2a_{\{k,l\}})}, \quad (2.10)$$

KEY IDEA

The significance of [Equations 2.6](#) and [2.9](#) is that they relate the different agreement rates $a_{\mathcal{A}}$, which are easily estimated from *unlabeled* data, to the true error rates $e_{\mathcal{A}}$ of the functions, which are difficult to estimate without labeled data. Note that if we have a system of such equations with rank equal to the number of error rates mentioned, then we can solve exactly for these error rates in terms of the observed agreement rates. This is not the case in general because, given a set of functions, $\hat{f}_1, \dots, \hat{f}_M$, we obtain $2^M - M - 1$ agreement rate equations (one for each subset of two or more functions) expressed in terms of $2^M - 1$ error rates (one for each non-empty subset of functions). However, if we assume that the errors made by the M individual functions are independent, then we can express all of the $2^M - 1$ error rates in terms of M single-function error rates (e.g., $e_{\{j,k\}} = e_{\{j\}}e_{\{k\}}$) and we can then solve exactly for all error rates (given the additional assumption that error rates are better than chance). Furthermore, if we are unwilling to make the strong assumption that errors of individual functions are independent, then we can instead solve for the set of error rates that minimize the dependence among errors. For example, among the infinite solutions to the underdetermined set of equations, we choose the solution that minimizes $\sum_{j,k} (e_{\{j,k\}} - e_{\{j\}}e_{\{k\}})^2$ (this idea can be easily extended to larger subsets than simply pairs of functions). The key idea in this paper is that the correspondence between easily-observed agreement rates and hard-to-observe error rates given by these equations can be used as a practical basis for estimating error rates from unlabeled data.

where $j \in \{1, 2, 3\}$, $k, l \in \{1, 2, 3\} \setminus j$ with $k < l$ and:

$$c = \sqrt{(2a_{\{1,2\}} - 1)(2a_{\{1,3\}} - 1)(2a_{\{2,3\}} - 1)}, \quad (2.11)$$

where, for a set B and an element of that set b , the notation $B \setminus b$ denotes the set containing all elements in B except b . In most practical applications the competing functions do not make independent errors. We now consider the more difficult problem of estimating the error rates from agreement rates, but without assuming independence of the function error events.

2.4.1.2 N Functions That Make Dependent Errors

When we have N functions that make dependent errors we rely on the agreement rate defined in Equation 2.9. We consider the agreement rates for all sets $\mathcal{A} = \{\mathcal{A} \subseteq \{1, \dots, N\} : |\mathcal{A}| \geq 2\}$ of functions (the agreement rate is uninformative for less than two functions) and we obtain $2^N - N - 1$ equations by matching Equation 2.9 to the sample agreement rate for each possible subset of functions. Moreover, the unknown variables are all the individual function error rates along with all of the possible joint function error rates (let us denote the vector containing all those variables by $\mathbf{e}$). This is a total of $2^N - 1$ unknown variables. The set of $2^N - N - 1$ equations involving $2^N - 1$ unknown variables yields an underdetermined system of equations with an infinite number of possible solutions. We therefore cast this problem as a constrained optimization problem where the agreement equations form constraints that must be satisfied and where we seek the solution that minimizes the following objective:

$$c(\mathbf{e}) = \sum_{\mathcal{A}: |\mathcal{A}| \geq 2} \left(e_{\mathcal{A}} - \prod_{j \in \mathcal{A}} e_{\{j\}} \right)^2. \quad (2.12)$$

We are effectively trying to minimize the dependence between the error events, while satisfying all of the agreement rates constraints. We already saw in Section 2.4.1.1 that, if we assume that the error events are independent, then we can obtain an exact solution. By defining our optimization problem in this way we are effectively relaxing this constraint by saying that we want to find the error rates that satisfy our constraints and that are, at the same time, as independent as possible (i.e., a measure of dependence is now used as a regularizer). Most existing methods trying to estimate function error rates using only unlabeled data assume that the error events are independent; the main novelty of this method lies in the fact that *we relax all these assumptions and thus make no hard assumptions about our functions*.

This can be seen from the fact that when the error events are independent we have $e_{\mathcal{A}} = \prod_{j \in \mathcal{A}} e_{\{j\}}$.

Note that we could also define different objective functions based on information we might have about our function approximations or based on different assumptions we might want to make. For example, one could try minimizing the sum of the squares of all the error rates (i.e., the ℓ_2 norm of $\mathbf{e}$) in order to obtain the most optimistic error rates that satisfy the agreement rates constraints. The novelty of our method partly lies in the formulation of the error rates estimation problem using only unlabeled data as a constrained optimization problem.

In this section we defined the model we are using for this method and the optimization problem we wish to solve. We refer to this method as the DIRECT method, due to its direct derivation from a simple observation on the relationship between agreement rates and error rates. In [Section 2.4.2](#) we define additional constraints that both this method and the maximum likelihood method (described in the next section) use.

2.4.1.3 Maximum Likelihood Formulation

We now consider an alternative formulation of the same ideas that we described in the previous section. We define a probabilistic model of the consistency in the functions' outputs, considering the most general case of having M functions that make potentially dependent errors. Let us denote the outputs of the functions on an independent and identically distributed sample of data, $x_1, \dots, x_N$, by $\hat{\mathbf{y}}_i = [\hat{y}_{i1}, \dots, \hat{y}_{iM}]$, for $i \in \{1, \dots, N\}$. The $\hat{\mathbf{y}}_i$'s are independent and therefore, we can define the data likelihood as:

$$\mathcal{L}(\mathbf{e}) = p_{\mathcal{D}}(\hat{\mathbf{y}}_1, \dots, \hat{\mathbf{y}}_N | \mathbf{e}) = \prod_{i=1}^N p_{\mathcal{D}}(\hat{\mathbf{y}}_i | \mathbf{e}), \quad (2.13)$$

where the parameter vector $\mathbf{e}$ contains all possible error event probabilities. More specifically, it contains all the $e_{\mathcal{J}}$, for all $\mathcal{J} \subseteq \{1, \dots, M\}$ and $|\mathcal{J}| \in \{1, \dots, M\}$. $\hat{\mathbf{y}}_i$ contains all the function outputs given the data sample x_i . Note that, for simplicity, we remove the explicit $x_i \sim \mathcal{D}$ notation in this section, and subscript p using only the data distribution $\mathcal{D}$. In order to compute $p_{\mathcal{D}}(\hat{\mathbf{y}}_i | \mathbf{e})$ we need to consider two cases:

1. All functions agree with each other (i.e., $\hat{\mathbf{y}}_i$ is a vector of all 1's or all 0's).
2. The functions can be split in two non-empty groups: those that output 1 and those that output 0.

The groups of functions with the same output can also be viewed as maximal cliques in a graph whose nodes correspond to the functions and whose edges correspond to agreement between function pairs (i.e., when there is agreement between two functions there is an edge connecting their corresponding nodes in the graph, and when there is no agreement between them there is no edge). By using this representation we call the first case the "one clique case" and the second case the "two cliques case." We are now going to consider these two cases separately.

ONE CLIQUE CASE. Let us denote the set of all function indices in the clique by $\mathcal{C}$ (i.e., $\mathcal{C} = \{1, \dots, M\}$). In this case, either all functions make an error or none of them does. Therefore, for the probability of the current sample we have that:

$$p_{x_i \sim \mathcal{D}}(\hat{\mathbf{y}}_i | \mathbf{e}) = p_{x_i \sim \mathcal{D}}\left(\bigcap_{j \in \mathcal{C}} E_{\{j\}}(x_i)\right) + p_{x_i \sim \mathcal{D}}\left(\bigcap_{j \in \mathcal{C}} \bar{E}_{\{j\}}(x_i)\right), \quad (2.14)$$

$$= e_{\mathcal{C}} + 1 - p_{x_i \sim \mathcal{D}}\left(\bigcup_{j \in \mathcal{C}} E_{\{j\}}(x_i)\right), \quad (2.15)$$

$$= e_{\mathcal{C}} + 1 + \sum_{k=1}^{|\mathcal{C}|} \left[(-1)^k \sum_{\substack{\mathcal{J} \subseteq \mathcal{C} \\ |\mathcal{J}|=k}} e_{\mathcal{J}} \right], \quad (2.16)$$

following a similar derivation to the one we used when defining the agreement rates in [Equation 2.9](#).

TWO CLIQUES CASE. Let us denote the set of function indices in the first clique by $\mathcal{C}_1$ and those in the second clique by $\mathcal{C}_2$. Then, we have two possible events:

1. All functions in $\mathcal{C}_1$ make an error and none of those in $\mathcal{C}_2$ make an error.
2. All functions in $\mathcal{C}_2$ make an error and none of those in $\mathcal{C}_1$ make an error.

Let $p_{\mathbf{x}_i \sim \mathcal{D}}^1(\hat{\mathbf{y}}_i | \mathbf{e})$ denote the probability of $\hat{\mathbf{y}}_i$ given that the first event occurs, and let $p_{\mathbf{x}_i \sim \mathcal{D}}^2(\hat{\mathbf{y}}_i | \mathbf{e})$ denote the probability of $\hat{\mathbf{y}}_i$ given that the second event occurs. It can be easily seen that these two events are mutually exclusive and so we have that:

$$p_{\mathbf{x}_i \sim \mathcal{D}}(\hat{\mathbf{y}}_i | \mathbf{e}) = p_{\mathbf{x}_i \sim \mathcal{D}}^1(\hat{\mathbf{y}}_i | \mathbf{e}) + p_{\mathbf{x}_i \sim \mathcal{D}}^2(\hat{\mathbf{y}}_i | \mathbf{e}). \quad (2.17)$$

Once again, following a similar derivation to the one we used when defining the agreement rates in [Equation 2.9](#), we have that:

$$p_{\mathbf{x}_i \sim \mathcal{D}}^1(\hat{\mathbf{y}}_i | \mathbf{e}) = p_{\mathbf{x}_i \sim \mathcal{D}} \left(\left[\bigcap_{j \in \mathcal{C}_1} E_{\{j\}}(\mathbf{x}_i) \right] \cap \left[\bigcap_{k \in \mathcal{C}_2} \bar{E}_{\{k\}}(\mathbf{x}_i) \right] \right), \quad (2.18)$$

$$= p_{\mathbf{x}_i \sim \mathcal{D}} \left(\left[\bigcap_{j \in \mathcal{C}_1} E_{\{j\}}(\mathbf{x}_i) \right] \cap \overline{\left[\bigcup_{k \in \mathcal{C}_2} E_{\{k\}}(\mathbf{x}_i) \right]} \right), \quad (2.19)$$

$$= e_{\mathcal{C}_1} + \sum_{k=1}^{|\mathcal{C}_2|} \left[(-1)^k \sum_{\substack{\mathcal{J} \subseteq \mathcal{C}_2 \\ |\mathcal{J}|=k}} e_{\{\mathcal{J} \cup \mathcal{C}_1\}} \right]. \quad (2.20)$$

For the second line we used one of De Morgan's laws and for the last line we used a modified form of the inclusion-exclusion principle. To understand the step we used to obtain the last equation, let us consider a simple case with example events A , B_1 and B_2 . It is clear from the Venn diagram in [Figure 2.4](#) that:

$$p(A \cap \overline{[B_1 \cup B_2]}) = p(A) - p(A \cap B_1) - p(A \cap B_2) + p(A \cap B_1 \cap B_2). \quad (2.21)$$

The shaded blue part in the diagram corresponds to this probability. The last step in [Equation 2.20](#) follows from extending this result using the inclusion-exclusion principle. Similarly, we derive the following expression for the second case:

$$p_{\mathbf{x}_i \sim \mathcal{D}}^2(\hat{\mathbf{y}}_i | \mathbf{e}) = e_{\mathcal{C}_2} + \sum_{k=1}^{|\mathcal{C}_1|} \left[(-1)^k \sum_{\substack{\mathcal{J} \subseteq \mathcal{C}_1 \\ |\mathcal{J}|=k}} e_{\{\mathcal{J} \cup \mathcal{C}_2\}} \right]. \quad (2.22)$$

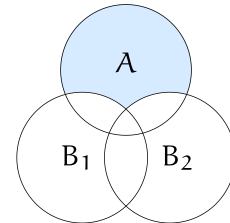


Figure 2.4: Illustration of [Equation 2.21](#).

Having defined the likelihood function it remains to describe the optimization problem that we need to solve to obtain the maximum likelihood estimate of $\mathbf{e}$. We use the negative logarithm of the likelihood as the objective function that we want to minimize. The details of how we solve this optimization problem are provided in [Section 2.4.2](#). We refer to this method as MLE.

2.4.1.4 Regularization

We now define a method which is a slight modification of the aforementioned MLE method, in that it uses a modified objective function. The objective function we considered in the MLE method is non-convex and hence may have multiple local maxima. In order to avoid getting stuck into one of these local maxima—or at least help with avoiding it—we add a regularization term. Following the same argument we used in constructing the objective function of the DIRECT method, we define the new objective function:

$$c(\mathbf{e}) = -\log \mathcal{L}(\mathbf{e}) + \lambda \sum_{\mathcal{A}: |\mathcal{A}| \geq 2} \left(e_{\mathcal{A}} - \prod_{j \in \mathcal{A}} e_{\{j\}} \right)^2, \quad (2.23)$$

λ can be interpreted as a function of the variance of this prior.

where λ is a hyperparameter whose value can be chosen arbitrarily. We call this the maximum a posteriori (MAP) method because the added term is equivalent to adding a Gaussian prior to the error rate estimates. As we shall see in [Section 2.5](#) the performance of this method depends heavily on the value of λ .

2.4.2 Optimization

In the previous sections we defined optimization problems that correspond to each of our methods. We use the TOMLAB Base Module v.7.7 “conSolve” solver for all methods. More details about this solver can be found at <https://tomopt.com/tomlab>. In the following sections we discuss: (i) some additional constraints that apply to all proposed methods, (ii) extensions of our approach to the case where multiple approximations are learned for each of several different target functions, and (iii) an approximation that can make our methods much more computationally efficient and in some cases even more accurate.

2.4.2.1 Error Rates Constraints

Our unknown variables include both the individual function error rates and the joint function error rates. We need to impose constraints on the values that the joint function error rates are allowed to take. These constraints follow from basic rules of probability and set theory; they represent bounding joint event probabilities using the corresponding marginal event probabilities. They are described by:

$$e_{\mathcal{A}} \leq \min_{j \in \mathcal{A}} e_{\mathcal{A} \setminus j}, \quad (2.24)$$

for $|A| \geq 2$. Furthermore, regarding the individual function error rates, it is easy to see that if we transform all e_j , for $j = 1, \dots, M$, to $1 - e_j$, the agreement rates remain unchanged. A similar result holds for the likelihood function. In order to make our models identifiable we add the constraint that $e_j \in [0, 0.5)$, for $j = 1, \dots, M$, which simply means that our functions perform better than chance, rendering it a reasonable constraint. Note that this identifiability issue can also be resolved by adding the slightly weaker constraint that the majority of the functions perform better than chance.

In order for our methods to work in the first place, this constraint must hold for most functions.

2.4.2.2 Dealing With Multiple Classification Problems

Up to this point we have assumed that there is a single target function and multiple approximations to this function. More generally though, we might have multiple target functions or problem settings, and a common set of learning algorithms used for learning each one. For example, this is the case in NELL where the different target functions correspond to different boolean classification problems (e.g., classifying a NP as a city, as a location, etc.). Multiple learning methods are utilized to approximate each target function (e.g., a classifier based on the NP orthography, a second classifier based on the NP contexts, etc.), so that each such classification problem or target function corresponds to an instance of our “multiple approximations” problem setting.

Of course we can apply our DIRECT method or our MLE method to estimate accuracies separately for each target classification problem (and that is what we actually did for our experiments in [Section 2.5](#)). However, when we have multiple target functions and multiple learning methods shared across each of them, there is an interesting opportunity to further couple the error rate estimates across these different target functions. In [Equations 2.12](#) and [2.23](#) we introduced terms to minimize the dependency between the error rates of competing approximations. In the case where we have multiple target functions, we can introduce additional terms to capture other relevant assumptions. For example, we could introduce a term to minimize the difference in error dependencies between two learning methods across multiple classification problems (e.g., we could minimize the difference in error dependencies between orthography-based and context-based classifiers trained for different classification problems). In fact, these interesting settings form the motivation for some of our work presented in [Chapter 3](#), as discussed in [Section 2.6](#).

2.4.2.3 Approximating High Order Error Rates

Once the agreement rate estimates (number of occurrences of each clique formation in the case of the MLE and the MAP methods) have been calculated, the execution time of the optimization procedure for all proposed methods does not depend on the number of available data samples, N . Even the execution time of the optimization procedure for the MLE method, which may at first sight seem to depend on N , does not actually depend on it because there is only a fixed number of possible clique combinations one can obtain for a given number of functions, M . This number is equal to 2^{M-1} . In a large data sample we will have a lot of repeated samples in terms of the maximal cliques that they result in. We can compute the log-likelihood term for each one of

these cliques once and multiply it by the number of samples in which they each appear. Thus, the execution time depends only on the number of functions, M . This can be easily seen by considering the number of unknown variables we have which is equal to $2^M - 1$. As will be shown in [Section 2.5](#), the performance of all methods, in terms how good the obtained function error rate estimates are, increases with an increasing number of functions, M . It is therefore not a good idea to try and reduce M . Therefore, we instead propose a way to reduce the execution time of the optimization procedure by approximating high order error rate terms, instead of estimating them directly.

We can estimate high order joint function error rates using low order function error rates by using the following formula:

$$e_{\mathcal{A}} = \frac{1}{|\mathcal{A}|} \sum_{j \in \mathcal{A}} e_{\mathcal{A} \setminus j} e_{\{j\}}, \quad (2.25)$$

for $|\mathcal{A}| > M_e$, where M_e is chosen arbitrarily. With a high value of M_e we obtain better estimates but execution time is larger, and vice-versa. This estimate is based on the fact that the higher the order of the function error rates, the less significant the impact of an independence assumption between them is. Furthermore, the only available information regarding high order error rates comes from high order sample agreement rates, $\hat{a}_{\mathcal{A}}$, which will likely be very noisy estimates of the true agreement rates. That is because there will be very few data samples where all of the functions in $\mathcal{A}$ agree and therefore the sample agreement rate will be computed using only a small number of data samples resulting in a noisy estimate of the true agreement rate. This motivates us to approximate high order error rates using low order error rates, instead of directly estimating them. In fact, in the case where the sample agreement rates are too noisy, this approximation might even increase the quality of the obtained error rate estimates. By approximating high order error rates as we described earlier, we are effectively ignoring the corresponding high order sample agreement rates (i.e., they are not used in our estimation) for the DIRECT method.

The “order” of an error rate, $e_{\mathcal{A}}$, or agreement rate, $\alpha_{\mathcal{A}}$, refers to the size/cardinality of set $\mathcal{A}$, or simply $|\mathcal{A}|$.

2.5 EXPERIMENTS

We perform experiments using two very different datasets, in order to explore the ability of our methods to estimate error rates in realistic settings without requiring any domain-specific tuning. For both datasets, we use a set of labeled data examples to perform our experiments. We use the data samples without their labels to estimate agreement rates, and subsequently estimate error rates using our methods. We then use the same examples with their labels to estimate each function’s true error rate, which we refer to as the “true error rate” of the function.

NELL DATASET. This dataset consists of data samples where we use four binary logistic regression (LR) classifiers to predict whether a NP belongs to a specific category in the NELL knowledge base (e.g., is “Monongahela” a river?). The domain in this case is defined by the category (e.g., beverage and river are two different domains) and the four classifiers we use are the following:

1. ADJ: A LR classifier that uses as features the adjectives that occur with the NP over millions of web pages.
2. CMC: A LR classifier that considers orthographic features of the NP (e.g., does the NP end with the letter string “-burgh?”).
3. CPL: A LR classifier that uses as features words and phrases that frequently appear together with the NP.
4. VERB: A LR classifier that uses as features verbs that appear with the NP.

We provide more details for these classifiers in our publication about NELL (Mitchell et al., 2018). Table 2.1 lists the NELL categories that we used as the domains in our experiments, along with the number of labeled examples available per category. Note that the NP features used by these four classifiers are somewhat independent given the correct classification label, and thus the classifiers should make somewhat independent errors.

Category	# Examples
animal	20,733
beverage	18,932
bird	19,263
bodypart	21,840
city	21,778
disease	21,827
drug	20,452
food	19,566
fruit	18,911
muscle	21,606
person	21,700
protein	21,811
river	21,723
vegetable	18,826

Table 2.1: The NELL categories used in our experiments.

BRAIN DATASET. Functional Magnetic Resonance Imaging (fMRI) data were collected while 8 subjects read a chapter from “Harry Potter and the Philosopher’s Stone”—a popular novel (Rowling, 2012)—one word at a time. The classification task is to find which of two 40-second long story passages correspond to an unlabeled 40 second time series of fMRI neural activity. For this binary classification task, we use eleven classifiers.

After the classifiers are trained on a portion of the data, each classifier uses different types of annotations of the text to predict the brain activity related to the two story passages. The held out 40-second fMRI segment is assigned the label of the story passage with the closest predicted activity. Each classifier uses a different type of word annotations in the text: (i) corpus derived semantic features borrowed from Murphy et al. (2012), (ii) occurrence or absence of dialog, (iii) motion actions, (iv) emotions, (v) non-motion actions, (vi) presence of different story characters, (vii) word length, (viii) local variance of word length, (ix) number of words in the sentence, (x) word part-of-speech tag, and (xi) grammatical role of a word in the sentence (computed automatically using a dependency parser). We run the classification using all 11 classifiers and distinct subsets of the data corresponding to 11 different brain regions (these regions represent our domains in this case). Note that we combine the data collected from the 8 subjects in a single dataset, and end up with 924 labeled examples per location. Different brain regions encode different types of information and thus, we expect the performance of the 11 classifiers to be different for each region. Estimating the classifiers’ error rates for each region should thus also help us determine where different types of information are encoded in the brain. Additional details about this dataset can be found in (Wehbe et al., 2014).

More details can be found at our code repository: <https://github.com/eaplatanios/makina>. Our experimental results for both datasets are presented and discussed in the

following two sections. As the performance metric we use the mean absolute deviation (MAD) between the true function error rates and the function error rates estimated from unlabeled data (i.e., we sum the absolute values of the element-wise differences of the true error rates vector and the estimated error rates vector). We compute the MAD for the individual function error rates alone, for the pairwise function error rates (i.e., for $|\mathcal{A}| = 2$) alone and for all function error rates together. Note the higher order error rates are quite small, because it is rare for every one of the competing functions to simultaneously make an error. Therefore, we consider the individual and pairwise error rates to be most diagnostic of how well our approach is working.

2.5.1 Results on Error Estimation

NELL DATASET. We initially apply the DIRECT method using only the ADJ, the CPL and the VERB classifiers, while assuming that they make independent errors. The method for estimating error rates in this context is described in [Section 2.4.1.1](#). In this case, we estimate only the individual function error rates. The resulting MAD is 2.82×10^{-2} ; that is, the average error estimate is within a few percent of the true error. Although encouraging, this MAD is poor in comparison to our less restricted methods described below, and indicates that the assumption that the classifiers make independent errors is incorrect in this case (and in most other cases as a matter of fact). Some of the obtained error rates are not even within the interval $[0, 0.5]$ and are thus obviously incorrect, since we know that the true error rates must lie in this interval. From now on, we consider only the more general case of N functions that make dependent errors, thus making no explicit independence assumptions.

[Table 2.2](#) presents results for all three of our methods used with the entire NELL dataset. It includes the results obtained when using all available data samples (i.e., the numbers shown in [Table 2.1](#)) and when using only 50 data samples per category. It is clear from this table that the more data samples we have the better our methods perform, presumably due to the more accurate estimates of the true agreement rates for the DIRECT method, and to the larger volume of evidence we have to incorporate into our likelihood, for the other two methods. Furthermore, we see that the DIRECT method performs significantly better than the other two methods. This could possibly be attributed to the fact that for this method we solve an easier optimization problem, whereas for the other two we solve a highly non-convex problem and we likely get stuck in local minima. Better numerical optimization solvers could potentially help with that. Finally, the MAP method performs better than the MLE method, presumably reflecting the correctness of our prior which attempts to minimize error dependencies across competing approximations. We did discover that the performance of the MAP method depends strongly on the choice of the λ parameter. In this case, we use $\lambda = 10$ simply because this value gives the regularization term the same order of magnitude as the log-likelihood term in the objective function. Moreover, now it becomes clear why the 2.82×10^{-2} MAD that we obtained when we assumed independent error events is quite a bad result. The DIRECT method manages to achieve an MAD that is almost 6 times better than that.

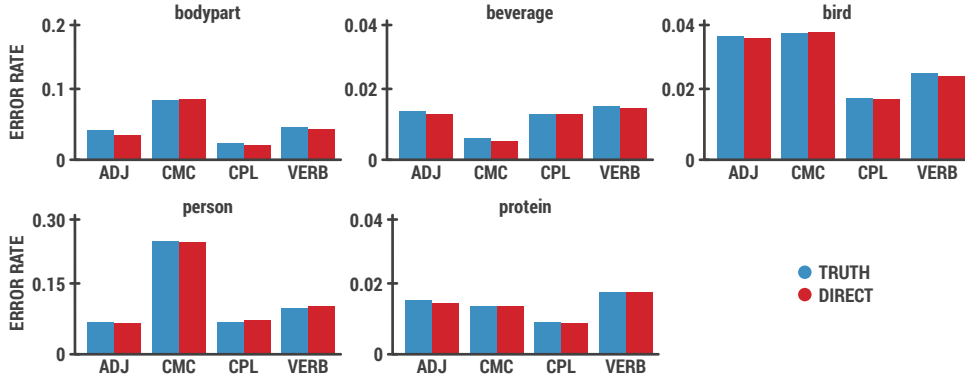


Figure 2.5: True errors (blue bars) versus errors estimated from unlabeled data using the DIRECT method (red bars), for four competing function approximations (ADJ, CMC, CPL, and VERB), to five different target function domains (i.e., bodypart, beverage, bird, person, and protein) using the NELL dataset. Note that each plot uses a different vertical scale to make it easier to observe the accuracy of the error rates estimates.

$\times 10^{-2}$	All Data Samples			50 Data Samples		
	Individual	Pairwise	All	Individual	Pairwise	All
DIRECT	<u>0.49</u>	<u>0.31</u>	<u>0.29</u>	<u>0.82</u>	<u>0.39</u>	<u>0.40</u>
MLE	2.77	2.19	1.84	20.06	19.96	15.42
MAP	1.54	1.30	1.08	13.11	15.17	11.14

Table 2.2: Mean absolute deviation (MAD) of individual, pairwise, and all function error rates for the NELL dataset, for all three proposed methods and for the cases where we use all of the available data samples and only 50 data samples per domain. The best score in each case is underlined and shown in red.

We also run an experiment using the approximation described in [Section 2.4.2.3](#) and setting $M_e = 2$ (i.e., considering only pairwise agreement rates). The individual functions MAD in this case is 0.52×10^{-2} , the pairwise one is 0.35×10^{-2} and the overall one is 0.31×10^{-2} . These results are worse than the ones we obtained without using this approximation, as expected, but they are still very good. This is important because it shows that this proposed approximation method is useful (there was a significant speedup as well—it takes 3 times less time).

From these results it is clear that the DIRECT method, which also happens to be the simplest and fastest of the three proposed methods, performs best for this dataset, without requiring any parameter tuning (as opposed to the MAP method). [Figure 2.5](#) shows a plot of the estimated error rates for the DIRECT method, along with the true error rates for five randomly selected NELL classification problems. This plot gives us an idea of how good the DIRECT estimates are, and helps us make sense of the reported MAD values. As seen in this plot, the DIRECT method is able to recover the ranking of the competing function approximations based on error rate exactly, irrespective of

the exact error estimate. This is in fact true for each of the 15 NELL target function classification problems that we used (i.e., not only for the five shown in this figure).

BRAIN DATASET. Table 2.3 presents results for the brain dataset, obtained when using 4 of the 11 competing function approximations (randomly selected to be classifiers 1, 3, 4, and 5) and when using all 11 of them. We observe that the more competing classifiers we use, the better the quality of the resulting estimates is. When using all 11 classifiers the DIRECT method performs significantly better than the other two methods. We have also included a plot of the estimated error rates for the DIRECT method, along with the true error rates, for three randomly selected brain regions (i.e., domains), in Figure 2.6. In this figure, we observe that we can also recover the ranking of the classifiers based on their error rates, using the DIRECT method, for this dataset.

Furthermore, we observe that for the case when we use 8 classifiers, the MLE method and the MAP method both perform poorly. This can probably be attributed to the optimization algorithm not being able to deal with these problems very well, due to their high dimensionality and non-convexity. These results could probably be improved by choosing a different optimization algorithm better suited for these problems. It is interesting to note that when we use only 4 classifiers, the MLE and the MAP methods perform slightly better than the DIRECT method in estimating the individual function error rates. However, they perform significantly worse when dealing with higher order error rates and so, overall, the DIRECT method still dominates. Note that, for the MAP method we selected $\lambda = 10$ for the same reasons as for the NELL dataset.

We also run an experiment using the approximation described in Section 2.4.2.3 and setting $M_e = 2$ (i.e., considering only pairwise agreement rates). The MAD for the individual function error rates in this case is 4.40×10^{-2} , the pairwise one is 4.06×10^{-2} and the overall one is 1.90×10^{-2} . These results are only slightly better than the ones we obtained without using this approximation. This is important because it shows once again that this approximation method is useful (there was a

INDEPENDENCE ASSUMPTION WEAKNESS

In order to make it more clear that the independence assumption is not very appropriate even in the case of NELL where a significant amount of effort has been put into having the NELL classifiers make independent errors, we provide here a measure of that dependence. We compute the following quantity for each domain:

$$\frac{1}{Z} \sum_{j,k} \left| \frac{e_{\{j,k\}}}{e_{\{j\}}e_{\{k\}}} - 1 \right|, \quad (2.26)$$

where Z is the total number of terms in the sum, and we average over all domains. This quantity gives us a measure of the average dependence of the function error rates across all domains. If the functions make independent errors, then this quantity should be equal to 0. We computed this quantity for the NELL dataset using the sample error rates, which are an estimate of the true error rates, and we obtained a value of 8.1770, which is far from 0. This indicates why our methods—and especially the DIRECT method—do so much better than the exact solution when assuming independent errors.

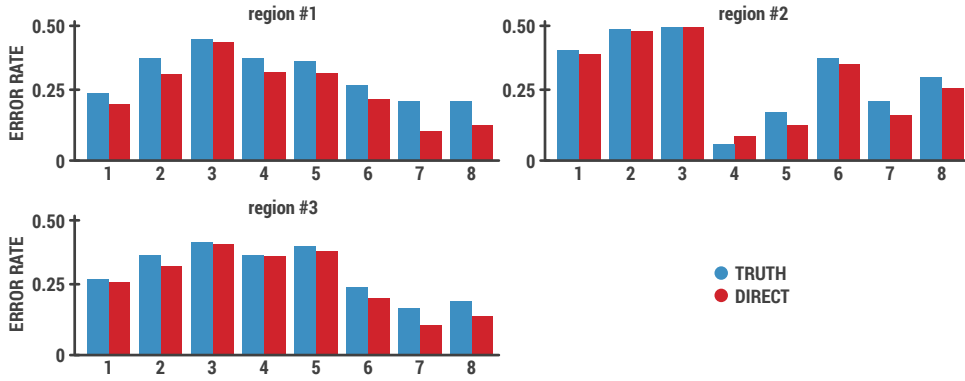


Figure 2.6: True errors (blue bars) versus errors estimated from unlabeled data using the DIRECT method (red bars), for eight competing function approximations (based on different story features), and three different target function domains (using neural activity from three different brain regions) using the brain dataset. Note that our error rate estimates from unlabeled data are quite close to the true error rates.

$\times 10^{-2}$	4 Classifiers			11 Classifiers		
	Individual	Pairwise	All	Individual	Pairwise	All
DIRECT	10.97	<u>6.60</u>	<u>6.50</u>	<u>4.36</u>	<u>4.14</u>	<u>2.01</u>
MLE	10.60	8.34	7.64	32.02	12.33	4.50
MAP	<u>9.61</u>	18.19	11.16	27.95	18.60	7.26

Table 2.3: Mean absolute deviation (MAD) of individual, pairwise, and all function error rates for the brain dataset, for all three proposed methods, and for the cases where we use 4 classifiers and 11 classifiers. The best score in each case is underlined and shown in red.

significant speedup as well—it takes 8 times less time). The better accuracy could possibly be attributed to two factors: (i) the problem is of much lower dimensionality and so the optimization algorithm might be dealing better with it, and (ii) the high order sample agreement rates might have been bad estimates of the true agreement rates due to insufficient data and so they might have affected our methods negatively. Given the significant speedup and slight performance gain, this is the method that we shall refer to as DIRECT, in the following chapters.

2.5.2 Results on Ground Truth Estimation

A natural question to ask at this point is whether or not our method can also be used to estimate the ground truth labels directly, instead of just the error rates. As mentioned in the introduction of this chapter, a common approach to estimating ground truth labels from multiple imperfect labels is to perform a majority vote for each instance. We can use our estimated error rates to make the majority vote more robust by assigning more weight to functions that have lower error rates. The results obtained when doing this are shown in Figure 2.7. Our method helps boost the performance of majority voting. The gains are much smaller for the brain dataset, as opposed to the NELL dataset. This is most likely due to the fact that the brain

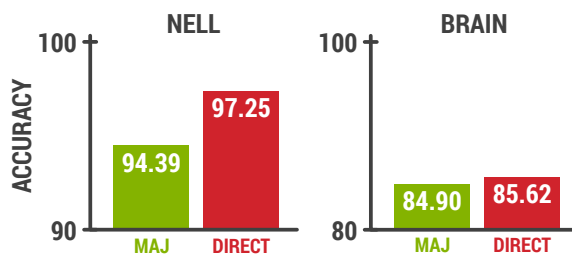


Figure 2.7: Accuracy of plain majority vote (MAJ) and majority vote weighted by the error rate estimates provided by the DIRECT method, for the two datasets we used in our experiments. A larger value corresponds to better performance.

dataset functions are very dependent and so modeling the function dependencies in some way may be crucial to improving performance. This is discussed further in the next section, and an improved method that is able to better handle this case will be introduced in the next chapter.

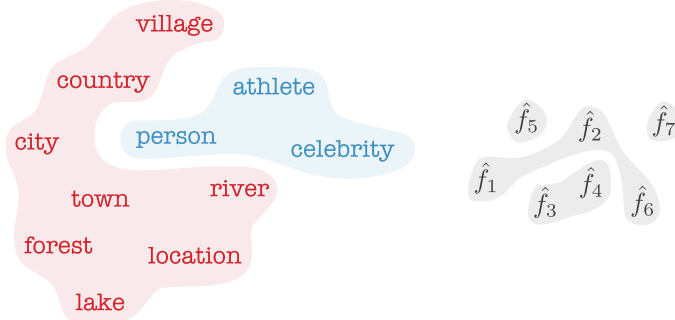
2.6 LIMITATIONS

The DIRECT approach presented in this chapter has three key limitations:

1. Dependencies: We are currently directly modeling the relationship between agreement rates and error rates and we are effectively relaxing a strong independence assumption using the objective function shown in [Equation 2.12](#). However, as we saw in [Section 2.5.2](#) this may not be sufficient to do well in cases where the functions are highly dependent. This is also evident from the fact that our method performs worse for the brain dataset than for the NELL dataset. Therefore, it is important to find ways to better model dependencies between the functions. It is also important to be able to model dependencies between the different domains (i.e., classification problems). This is because, for some of them we may have a lot of data and for others very little, but it may also be easy to figure out which ones are similar to each other, so that we can share information across them. Modeling all these kinds of dependencies could therefore prove useful to further improving the performance and robustness of our method. This limitation is addressed in [Chapter 3](#).
2. Logical Constraints: In the case of NELL, we have a lot of side information about the domains in the form of logical constraints (i.e., the KB ontology is fixed and known a priori). Such constraints could prove very useful for accuracy estimation. For example, if we know that the categories `athlete` and `country` are mutually exclusive, then we also know that if two functions say that a specific NP refers to both an `athlete` and a `country`, then at least one of the two functions has to be wrong. In fact, the pre-existing knowledge integrator in NELL made use of such constraints using heuristic rules and this resulted in improved performance. Therefore, the main question here is whether we can

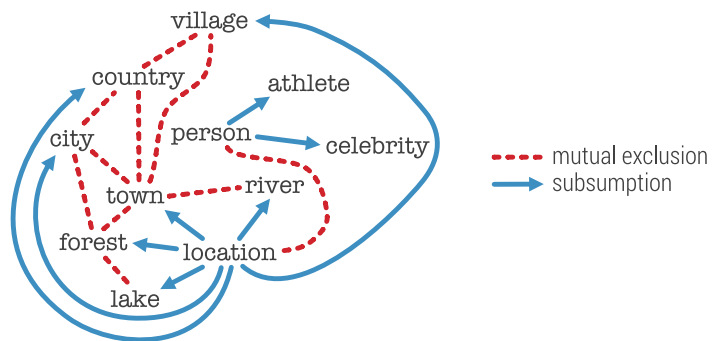
Limitation #1 Dependencies

between *tasks* and between *function approximations*



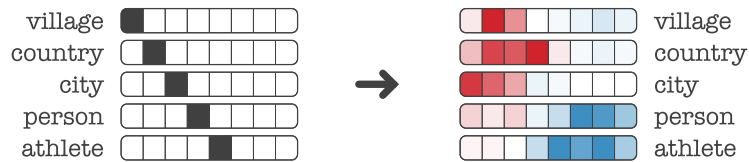
Limitation #2 Logical Constraints

between *tasks*



Limitation #3 Representations

for *tasks*



and similarly for *instances* and *function approximations*

Figure 2.8: Illustration of the three key limitations of the DIRECT approach. A detailed discussion of these limitations is provided in [Section 2.6](#).

use such constraints in the context of our agreement-based accuracy estimation methods. This question is addressed in [Chapter 4](#).

3. Representations: So far we have treated the input instances, x_i , as indicator variables and the DIRECT method has no way of using any information that may be available about them (e.g., in the case of NPs, it would be easy to construct feature vectors using neural word embedding approaches). The same is true about the function approximations. In some cases, these function approximations may even be human annotators (e.g., in Amazon Mechanical Turk). Ideally we should be able to use information about them to infer dependencies and become better able to share information effectively. Deep learning offers very powerful representation learning methods and so, in [Chapter 5](#), we propose an approach that makes use of such methods and addresses the limitations listed in this section. Furthermore, it makes it possible to merge the label integration phase with the model training phase, resulting in a single end-to-end approach for learning from imperfect labels.

2.7 KEY TAKEAWAYS

In this chapter, we introduced the concept of estimating the error rate of each of several approximations to the same function, based on their agreement rates over *unlabeled data* and we provided three different analytical methods to do so. Our experimental results are encouraging and suggest that function agreement rates are indeed very useful in estimating function error rates and thus, answer our motivating question on whether consistency implies correctness: *consistency does imply correctness, under certain independence assumptions*. We consider this work to be a first step towards developing a *self-reflection framework* for autonomous learning systems. However, we also identified three key limitations of the proposed methods that will be addressed in the following chapters.

Moreover, in this chapter, we considered a setting where we only have access to unlabeled data. Even though this is useful for NELL, it is also important to be able to handle labeled data that may be available. It is unclear how such information could be incorporated to the proposed DIRECT method (it could possibly be done by adding a regularizer forcing the estimated error rates to be close to the sample error rates we compute using the labeled data). However, all methods presented in the following chapters have been designed such that they can be used directly in semi-supervised settings, as well as in unsupervised settings.

A BAYESIAN APPROACH TO ESTIMATING ACCURACY

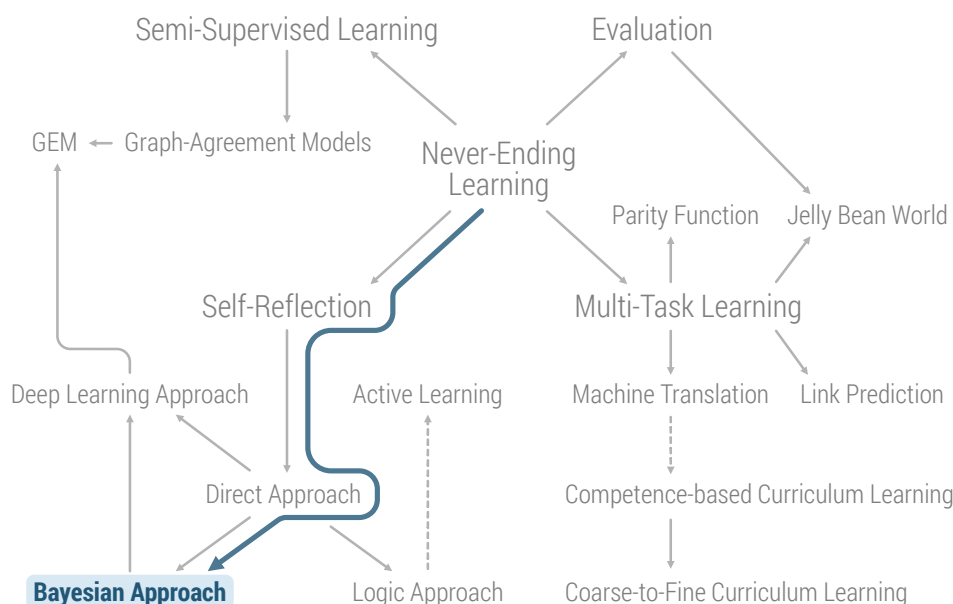


Figure 3.1: Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in [Section 1.4](#).

In the previous chapter, we showed that learning collections of functions allows us to perform completely unsupervised evaluation. Specifically, we proposed a method inspired by how we, as humans, would approach this problem. Even though the proposed method performs reasonably well in practice, we also identified and discussed some of its limitations in [Section 2.6](#). In this chapter, we propose a method that addresses the first one of these limitations; namely, the inability of the direct approach to directly model the dependencies between the learned functions. As we shall show, this method is based on Bayesian modeling and it manages to successfully address this limitation.

3.1 INTRODUCTION

The direct approach of [Chapter 2](#) relaxes the independence assumption for the function approximations. However, we observed that this may not be sufficient to do well in cases where the functions are highly dependent. In such cases, it is important to better model dependencies between the functions. Furthermore, it is important to model

dependencies between the different domains (i.e., classification problems). This is because, for some of them we may have a lot of data and for others very little, but it may be easy to figure out which ones are similar to each other and share information between them. Modeling these kinds of dependencies could prove useful to improving the performance and robustness of our methods.

In this chapter, we first propose to model the problem using a simple and elegant probabilistic graphical model that allows us to encode assumptions we make about our data in an intuitive manner, and can be extended to encode additional assumptions.¹ The proposed approach allows us to infer the posterior distribution of a single label for each data sample jointly with the accuracies of our classifiers (thus also removing the need for the separate weighted majority vote step that we used in [Section 2.5.2](#)), and it is also able to handle missing data, which are data samples for which a classifier might not have predicted any label. This can happen when the classifier does not have any features for those data samples, for example, and is not uncommon in practice. In fact, it happens quite frequently in the case of NELL. In order to model dependencies between the functions and the domains, we propose two nonparametric extensions of the simple model that allow sharing information among the different functions and domains, and that are especially useful in the case of limited data. Finally, we present experimental results demonstrating the success of the new approaches in the same experimental setting we considered in [Chapter 2](#).

Dawid and Skene (1979) were the first to formulate this problem in terms of a graphical model and Moreno et al. (2015) proposed a nonparametric extension applied to crowdsourcing. The state-of-the-art approach before the work presented in this chapter, is the work of Tian and Zhu (2015) and it also comes from the area of crowdsourcing. The authors proposed an interesting max-margin majority voting scheme for combining classifier outputs. As we show in [Section 3.5](#), the methods presented in this chapter are able to outperform their approach. The goal of this chapter is to present novel methods for estimating the error rate of each function approximation using only unlabeled data, while also jointly inferring the posterior distribution of the response of function f while accounting for these error rates.

3.2 A SIMPLE BAYESIAN MODEL

We consider the “multiple approximations” problem setting that was described in [Section 2.2](#). We have several different approximations, $\hat{f}_1, \dots, \hat{f}_N$, to some target boolean classification function, $f : \mathcal{X} \rightarrow \{0, 1\}$, and we wish to know the true accuracies of each of these different approximations, using only unlabeled data. We also wish to know the most likely single label, meaning the most likely response of the true underlying function f . For convenience, we sometimes use the notation $\hat{y}_{ij} = \hat{f}_j(x_i)$. We define the following generative process describing how our data samples were generated:

1. Let us make the assumption that there is an underlying distribution from which the labels of all the data instances are sampled. We first draw $p \sim \text{Beta}(\alpha_p, \beta_p)$, representing the prior probability for the true label being equal to 1.

¹ This work has been published in (Platanios et al., 2016).

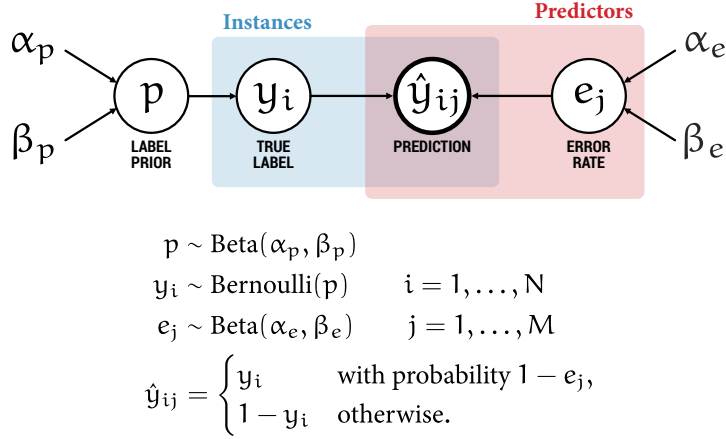


Figure 3.2: Simple probabilistic model for error rate estimation using only unlabeled data.

2. For each data example, x_i where $i = 1, \dots, N$, we draw a label $y_i \sim \text{Bernoulli}(p)$. This label is the true label that corresponds to $f(x_i)$.
3. Let us further assume that there is another underlying distribution from which the error rates of our function approximations are sampled. For each function approximation, $\hat{f}_j$ where $j = 1, \dots, M$, we draw an error rate $e_j \sim \text{Beta}(\alpha_e, \beta_e)$.
4. Finally, we can assume that each function approximation takes the sampled label for each example and flips it with probability equal to its error rate (thus making an error). It then outputs the resulting label. Thus, for each data instance x_i , and function approximation $\hat{f}_j$, we draw an output label $\hat{y}_{ij}$, according to the following distribution:

$$\hat{y}_{ij} = \begin{cases} y_i & \text{with probability } 1 - e_j, \\ 1 - y_i & \text{otherwise.} \end{cases} \quad (3.1)$$

This output label corresponds to $\hat{f}_j(x_i)$.

We emphasize the last step in the generative process, where with probability equal to the function error rate, the correct label is flipped and the function approximation makes an error. An illustration of this model is shown in [Figure 3.2](#). Note that, at inference time we are only provided a set of unlabeled data instances $x_1, \dots, x_N$ and the function approximations $\hat{f}_1, \dots, \hat{f}_M$.

INFERENCE. In order to perform inference for this simple model we use *Gibbs sampling*, a well-known Markov chain Monte Carlo (MCMC) sampling method (Geman and Geman, 1984). The conditional probabilities we use during sampling are defined as follows:

$$P(p \mid \cdot) = \text{Beta}(\alpha_p + \sigma_y, \beta_p + N - \sigma_y), \quad (3.2)$$

$$P(y_i \mid \cdot) \propto p^{y_i} (1 - p)^{1 - y_i} \pi_i, \quad (3.3)$$

$$P(e_j \mid \cdot) = \text{Beta}(\alpha_e + \sigma_j, \beta_e + S - \sigma_j), \quad (3.4)$$

where:

$$\sigma_y = \sum_{i=1}^S y_i, \quad (3.5)$$

$$\sigma_j = \sum_{i=1}^N \mathbb{1}_{\{\hat{y}_{ij} \neq y_i\}}, \quad (3.6)$$

$$\pi_i = \prod_{j=1}^M e_j^{\mathbb{1}_{\{\hat{y}_{ij} \neq y_i\}}} (1 - e_j)^{\mathbb{1}_{\{\hat{y}_{ij} = y_i\}}}, \quad (3.7)$$

and $\mathbb{1}_{\{\cdot\}}$ evaluates to one if its subscript statement is true and to zero otherwise. We sequentially sample from these three distributions, by sampling each random variable while keeping the others fixed to their last sampled values. The distribution of the samples we obtain is guaranteed to converge to the true posterior distribution of our random variables, given that we obtain a large enough number of samples.

Note that it is easy to handle missing data when using this model (in contrast to prior work), as we can model the missing data as latent variables which themselves can be inferred in the Gibbs sampling algorithm. The conditional probability for $\hat{y}_{ij}$, in case it needs to be sampled, is as follows:

$$P(\hat{y}_{ij} \mid \cdot) \propto e_j^{\mathbb{1}_{\{\hat{y}_{ij} \neq y_i\}}} (1 - e_j)^{\mathbb{1}_{\{\hat{y}_{ij} = y_i\}}}. \quad (3.8)$$

3.3 A COUPLED BAYESIAN MODEL

Up to this point we have assumed that there is a single target function and multiple approximations to that function. More generally though, we might have multiple target functions or problem settings and a common set of learning algorithms used for learning each of them. For example, this is the case in NELL, where the different target functions correspond to different boolean classification problems (e.g., classifying a NP as a city, as a location, etc.). The same is true for crowdsourcing when we may be asking multiple questions to human annotators. Multiple learning methods are utilized to approximate each one of these target functions (e.g., a classifier based on the NP morphology, a second classifier based on the context in which an NP appears

IMPLICIT USE OF AGREEMENT RATES

Most of the prior work proposes using agreement rates between the function approximations in order to estimate the error rates of these functions. By looking at Equations 3.3, 3.4, 3.5, 3.6, and 3.7, we can see that our method is also implicitly using agreement rates in order to estimate the function error rates. We are using the agreement between the function outputs and the true underlying labels in order to infer both the error rates of our functions and these labels. This fundamental connection further supports the argument made in Chapter 2 relating agreement and correctness, in that under certain conditions, agreement of several functions implies correctness of these functions.

in, etc.), so that each such classification problem corresponds to an instance of our “multiple approximations” problem setting.

It is reasonable to assume that there are some structural dependencies between our function approximations that could result in similar behavior (i.e., similar error rate across multiple domains). This is not an unreasonable assumption because the classifiers use the same set of features across all domains. If this is indeed the case, then sharing information across domains might prove useful. This forms the main motivation for the extension to the model proposed earlier, that we present in this section. The main idea is that we want to cluster the domains based on the distribution of the error rates of the function approximations. However, we do not know the number of clusters needed and this is why we resort to Bayesian non-parametrics; we want to infer the necessary number of clusters “automatically.” More specifically, we are going to use a Dirichlet process (DP) prior. Note that, Moreno et al. (2015) propose clustering the classifiers while only considering a single domain, instead of clustering the domains as we do. In the following paragraphs we provide an introduction to DPs and introduce our new and improved model.

DIRICHLET PROCESS. The Dirichlet process is a distribution over discrete probability measures (i.e., atoms):

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k}, \quad (3.9)$$

with countably infinite support, where the finite-dimensional marginals are distributed according to a finite Dirichlet distribution (Ferguson, 1973). It is parametrized by a base probability measure H , which determines the distribution of the atom locations, and a concentration parameter $\alpha > 0$ that is proportional to the inverse variance of the atom locations. The DP can be used as the distribution over mixing measures in a nonparametric mixture model. In the DP mixture model (Antoniak, 1974), the data samples $\{x_i\}_{i=1}^N$ are assumed to be generated according to the following process:

$$G \sim \text{DP}(\alpha, H), \quad \theta_i \sim G, \quad x_i \sim f(\theta_i). \quad (3.10)$$

While the DP allows for an infinite number of clusters a priori, any finite dataset will be modeled using a finite, but random, number of clusters.

MODEL. For the definition of our model we are going to use the Chinese restaurant process (CRP) representation of the DP (Blackwell and MacQueen, 1973), because this form is most appropriate for deriving the Gibbs sampling equations we use to perform inference. Following from the intuition provided in the beginning of Section 3.3, we now have a problem setting in which we have several different domains, $d = 1, \dots, D$, where for each domain d , we have a set of function approximations, $\hat{f}_1^d, \dots, \hat{f}_M^d$, to some target boolean classification function, $f^d : \mathcal{X} \rightarrow \{0, 1\}$. We wish to know the true accuracies of each of these different approximations, using only unlabeled data, as well as the most likely single label, meaning the most likely response of the true underlying function f . To this end, we define the following generative process:

1. Draw an infinite number of potential error rates, $\Phi_{\mathbf{l}}$, for our function approximations. For each $\Phi_{\mathbf{l}}$ and for $j = 1, \dots, M$, draw an error rate $[\Phi_{\mathbf{l}}]_j \sim \text{Beta}(\alpha_e, \beta_e)$.
2. For each domain $d = 1, \dots, D$:
 - i. Draw $p^d \sim \text{Beta}(\alpha_p, \beta_p)$, representing the prior probability for the true underlying function output being equal to 1 for domain d .
 - ii. For each data instance x_i^d , where $i = 1, \dots, N^d$, draw a label $y_i^d \sim \text{Bernoulli}(p^d)$. This corresponds to the true label $f^d(x_i^d)$.
 - iii. Draw a cluster assignment, $z^d \sim \text{CRP}(\alpha)$.
 - iv. For each function approximation, $\hat{f}_j^d$, define the error rate as $e_j^d = [\Phi_{z^d}]_j$.
 - v. For each data instance x_i^d and function approximation $\hat{f}_j^d$, draw an output label $\hat{y}_{ij}^d$ from the following distribution:

$$\hat{y}_{ij}^d = \begin{cases} y_i^d & \text{with probability } 1 - e_j^d, \\ 1 - y_i^d & \text{otherwise.} \end{cases} \quad (3.11)$$

This output label corresponds to $\hat{f}_j^d(x_i^d)$.

An illustration of this model is shown in [Figure 3.3](#). Note that, at inference time we are only provided D sets of unlabeled data $\{x_1^d, \dots, x_{N^d}^d\}_{d=1}^D$, one for each domain, along with the function approximations $\{\hat{f}_1^d, \dots, \hat{f}_M^d\}_{d=1}^D$.

INFERENCE. In order to perform inference for this model we also use Gibbs sampling. For sampling from the DP we use the approach described by Neal (2000). In order to get fast convergence, we first marginalize out of the conditional probabilities $\Phi_{\mathbf{l}}$ and sample the rest of the variables sequentially for a few iterations (i.e., we perform *collapsed Gibbs sampling*). We then start sampling the $\Phi_{\mathbf{l}}$ along with the other random variables, using the original conditional probabilities.

COLLAPSED GIBBS SAMPLING. The conditional probabilities we use during the collapsed sampling phase are as follows:

$$P(p^d | \cdot) = \text{Beta}(\alpha_p + \sigma_y^d, \beta_p + N^d - \sigma_y^d), \quad (3.12)$$

$$P(y_i^d | \cdot) \propto (p^d)^{y_i^d} (1 - p^d)^{1 - y_i^d} \mathcal{B}(A_i^d, B_i^d), \quad (3.13)$$

$$P(z^d = k | \cdot) \propto \begin{cases} Z_k^d \mathcal{P}_k^d & \text{if } Z_k^d > 0, \\ \alpha \mathcal{P}_{\text{new}}^d & \text{otherwise,} \end{cases} \quad (3.14)$$

where:

$$A_i^d = \alpha_{z^d}^d + \sum_{\hat{i}=1}^N \sum_{j=1}^M \mathbb{1}_{\{\hat{y}_{ij}^d \neq y_i^d\}}, \quad (3.15)$$

$$B_i^d = \beta_{z^d}^d + \sum_{\hat{i}=1}^N \sum_{j=1}^M \mathbb{1}_{\{\hat{y}_{ij}^d = y_i^d\}}, \quad (3.16)$$

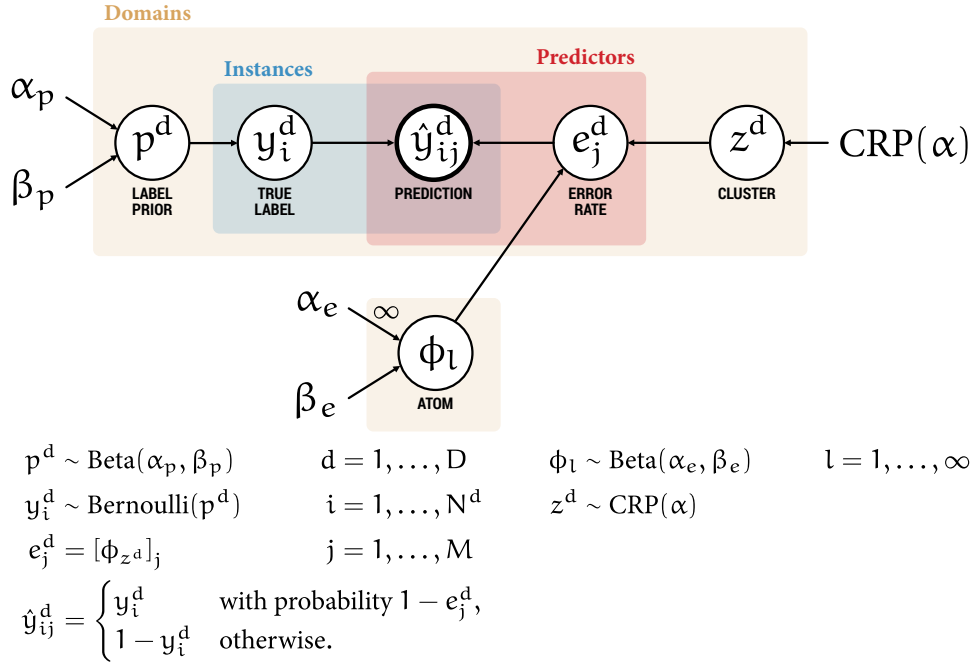


Figure 3.3: Graphical model for coupled error rate estimation using only unlabeled data. The coupling comes from the use of a Dirichlet process prior to group problem domains and share information within each group. Note that, $\text{CRP}(\alpha)$ denotes the Chinese restaurant process (CRP) with concentration parameter α .

$$\alpha_k^d = \alpha_e + \sum_{\substack{\hat{d}=1 \\ \hat{d} \neq d}}^D \mathbb{1}_{\{z^{\hat{d}}=k\}} \sigma^{\hat{d}}, \quad (3.17)$$

$$\beta_k^d = \beta_e + \sum_{\substack{\hat{d}=1 \\ \hat{d} \neq d}}^D \mathbb{1}_{\{z^{\hat{d}}=k\}} (N^{\hat{d}} - \sigma^{\hat{d}}), \quad (3.18)$$

$$\sigma_y^d = \sum_{i=1}^{N^d} y_i^d, \quad (3.19)$$

$$\sigma_j^d = \sum_{i=1}^{N^d} \mathbb{1}_{\{\hat{y}_{ij}^d \neq y_i^d\}}, \quad (3.20)$$

$$\sigma^d = \sum_{j=1}^M \sigma_j^d, \quad (3.21)$$

$$Z_k^d = \sum_{\substack{\hat{d}=1 \\ \hat{d} \neq d}}^D \mathbb{1}_{\{z^{\hat{d}}=k\}}, \quad (3.22)$$

$$\mathcal{P}_k^d = \frac{\mathcal{B}(\alpha_k^d + \sigma^d, \beta_k^d + N^d - \sigma^d)}{\mathcal{B}(\alpha_e + \alpha_k^d, \beta_e + \beta_k^d)}, \quad (3.23)$$

$$\mathcal{P}_{\text{new}}^d = \frac{\mathcal{B}(\alpha_e + \sigma^d, \beta_e + N^d - \sigma^d)}{\mathcal{B}(\alpha_e, \beta_e)}, \quad (3.24)$$

and $\mathcal{B}(\cdot, \cdot)$ is the Beta function.

UNCOLLAPSED GIBBS SAMPLING. After the first phase, we start sampling the error rates along with the rest of the variables and store the samples we obtain. During this phase, we use the following conditional probabilities:

$$P(p^d | \cdot) = \text{Beta}(\alpha_p + \sigma_y^d, \beta_p + N^d - \sigma_y^d), \quad (3.25)$$

$$P(y_i^d | \cdot) \propto (p^d)^{y_i^d} (1 - p^d)^{1 - y_i^d} \pi_i^d \quad (3.26)$$

$$P(z^d = k | \cdot) \propto \begin{cases} Z_k^d \mathcal{R}_k^d & , \text{ if } Z_k^d > 0, \\ \alpha \mathcal{P}_{\text{new}}^d & , \text{ otherwise,} \end{cases} \quad (3.27)$$

$$P([\phi_k]_j | \cdot) = \text{Beta}(\Phi_j^\alpha, \Phi_j^\beta), \quad (3.28)$$

where:

$$\pi_i^d = \prod_{j=1}^N (e_j^d)^{\mathbb{1}_{\{y_{ij}^d \neq l_i^d\}}} (1 - e_j^d)^{\mathbb{1}_{\{y_{ij}^d = l_i^d\}}}, \quad (3.29)$$

$$\mathcal{R}_k^d = \prod_{j=1}^N (e_j^d)^{\sigma_j^d} (1 - e_j^d)^{N^d - \sigma_j^d}, \quad (3.30)$$

$$\Phi_j^\alpha = \alpha_e + \sum_{d=1}^D \mathbb{1}_{\{z^d = k\}} \sigma_j^d, \quad (3.31)$$

$$\Phi_j^\beta = \beta_e + \sum_{d=1}^D \mathbb{1}_{\{z^d = k\}} (N^d - \sigma_j^d), \quad (3.32)$$

and the remaining quantities are defined in the same way as for the collapsed sampling phase. In the case of missing data we use [Equation 3.8](#).

3.4 A HIERARCHICALLY COUPLED BAYESIAN MODEL

As mentioned in [Section 2.6](#), the dependencies between our function approximations can be important for estimating error rates using unlabeled data. So far in our models we share little information across these functions when estimating their error rates. One natural extension to our coupled error estimation model, which allows sharing more information across functions, is to use a hierarchical Dirichlet process (HDP) prior, originally proposed by Teh et al. (2006). This prior would allow us to first cluster the domain (i.e., as we are doing in the DP model) and then, for each domain cluster, to also cluster the classifiers, and share the classifier clusters between different

domain clusters. In the following paragraphs we provide an introduction to HDPs and introduce our hierarchical coupled error estimation model.

HIERARCHICAL DIRICHLET PROCESS. Hierarchical Dirichlet processes, originally proposed by Teh et al. (2006), extend DPs to be able to model grouped data. The HDP is a distribution over probability distributions G^m , $m = 1, \dots, M$, each of which is conditionally distributed according to a DP. These distributions are coupled using a discrete common base measure, which is also distributed according to a DP. Each distribution G^m can be used to model a collection of observations $\{x_i^m\}_{i=1}^{N_m}$, as follows:

$$\begin{aligned} G &\sim \text{DP}(\gamma, H), & G^m &\sim \text{DP}(\alpha, G), \\ \theta_i^m &\sim G^m, & x_i^m &\sim f(\theta_i^m). \end{aligned} \quad (3.33)$$

Each observation within a group is a draw from a mixture model, and mixture components can be shared between groups. The intuition behind this property of the HDP is that, due to the base measure of the child DPs being discrete, they necessarily share atoms. Thus, the mixture models in the different groups may share mixture components, as desired.

MODEL. To extend our model for coupled error rate estimation and allow sharing of information across functions by using an HDP, as described at the beginning of [Section 3.4](#), we define the following generative process:

1. Draw an infinite number of potential error rates, $\phi_l \sim \text{Beta}(\alpha_e, \beta_e)$, for our function approximations.
2. For each domain $d = 1, \dots, D$:
 - i. Draw $p^d \sim \text{Beta}(\alpha_p, \beta_p)$, as in the coupled error estimation model.
 - ii. For each data example, x_i^d where $i = 1, \dots, N^d$, draw a label $y_i^d \sim \text{Bernoulli}(p^d)$, as in the coupled error estimation model. This corresponds to the true label $f^d(x_i^d)$.
 - iii. Draw an infinite number of potential cluster assignments for each function approximation, $k_m^d \sim \text{CRP}(\gamma)$.
 - iv. For each function approximation, $j = 1, \dots, M$:
 - a. Draw a cluster assignment, $z_j^d \sim \text{CRP}^d(\alpha)$, from the CRP corresponding to the current domain.
 - b. Define the error rate as $e_j^d = \phi_{t_j^d}$, where $t_j^d = k_{z_j^d}^d$.
 - c. For each data example, x_i^d , draw an output label, $\hat{y}_{ij}^d$, according to the following distribution:

$$\hat{y}_{ij}^d = \begin{cases} y_i^d & \text{with probability } 1 - e_j^d, \\ 1 - y_i^d & \text{otherwise.} \end{cases} \quad (3.34)$$

This output label corresponds to $\hat{f}_j^d(x_i^d)$.

An illustration of this model is shown in [Figure 3.4](#).

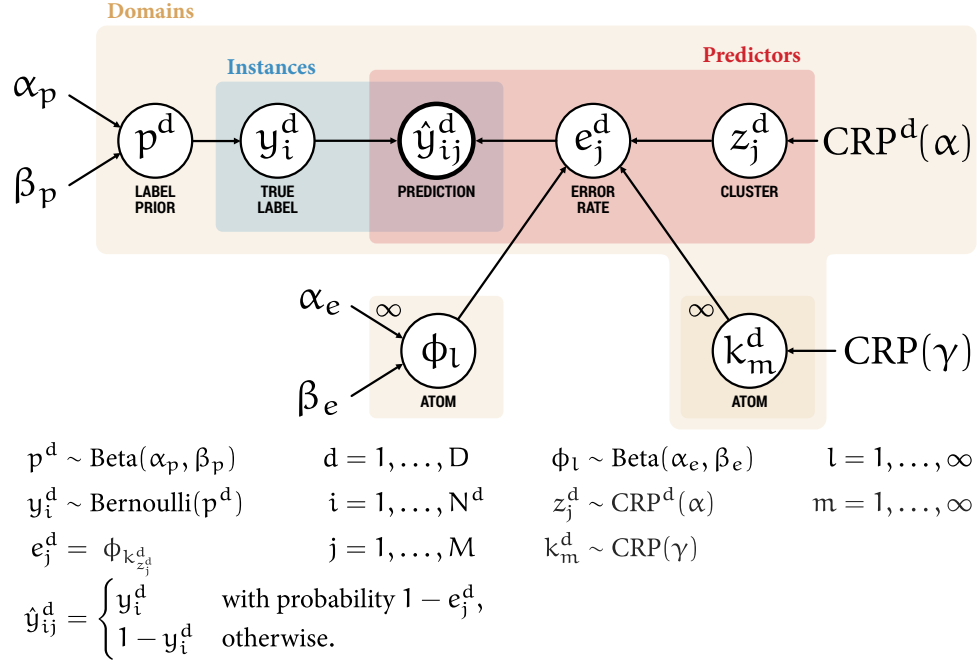


Figure 3.4: Graphical model for hierarchical coupled error rate estimation using only unlabeled data. The hierarchical coupling comes from the use of a hierarchical Dirichlet process prior to cluster problem domains and functions and share information within each cluster. Note that, $\text{CRP}^d(\alpha)$ denotes a separate Chinese restaurant process (CRP) per domain d , with concentration parameter α .

INFERENCE. In order to perform inference for this model we also use Gibbs sampling. For sampling from the HDP we use the approach described by Teh et al., 2006. Similar to the coupled model, for the initial sampling phase we use collapsed Gibbs sampling.

COLLAPSED GIBBS SAMPLING. The conditional probabilities we use during the collapsed sampling phase are as follows:

$$P(p^d | \cdot) = \text{Beta}(\alpha_p + \sigma_y^d, \beta_p + N^d - \sigma_y^d), \quad (3.35)$$

$$P(y_i^d | \cdot) \propto (p^d)^{y_i^d} (1 - p^d)^{1 - y_i^d} L_i^d, \quad (3.36)$$

$$P(z_j^d = t | k_t^d = k, \cdot) \propto Z_{kt}^d \mathcal{P}_{jk}^d, \quad (3.37)$$

$$P(k_t^d = k | \cdot) \propto Z_k^d \mathcal{P}_{kt}^d, \quad (3.38)$$

where:

$$L_i^d = \prod_{k=1}^K \mathcal{B}(A_{ik}^d, B_{ik}^d), \quad (3.39)$$

$$A_{ik}^d = \alpha_k^d + \sum_{j=1}^M \sum_{i=1}^N \mathbb{1}_{\{k_{z_j^d}^d = k\}} \mathbb{1}_{\{\hat{y}_{ij}^d \neq y_i^d\}}, \quad (3.40)$$

$$B_{ik}^d = \beta_k^d + \sum_{\hat{i}=1}^N \sum_{j=1}^M \mathbb{1}_{\{k_{z_j^d}^d = k\}} \mathbb{1}_{\{y_{ij}^d = y_i^d\}}, \quad (3.41)$$

$$\alpha_k^d = \alpha_e + \sum_{\substack{\hat{d}=1 \\ \hat{d} \neq d}}^D \sum_{j=1}^N \mathbb{1}_{\{k_{z_j^{\hat{d}}}^{\hat{d}} = k\}} \sigma_j^{\hat{d}}, \quad (3.42)$$

$$\beta_k^d = \beta_e + \sum_{\substack{\hat{d}=1 \\ \hat{d} \neq d}}^D \sum_{j=1}^N \mathbb{1}_{\{k_{z_j^{\hat{d}}}^{\hat{d}} = k\}} (N^{\hat{d}} - \sigma_j^{\hat{d}}), \quad (3.43)$$

$$\alpha_{jk}^d = \alpha_k^d + \sum_{\substack{j=1 \\ j \neq j}}^N \mathbb{1}_{\{k_{z_j^d}^d = k\}} \sigma_j^d, \quad (3.44)$$

$$\beta_{jk}^d = \beta_k^d + \sum_{\substack{j=1 \\ j \neq j}}^N \mathbb{1}_{\{k_{z_j^d}^d = k\}} (N^d - \sigma_j^d), \quad (3.45)$$

$$Z_{kt}^d = \begin{cases} \sum_{\substack{j=1 \\ j \neq j}}^N \mathbb{1}_{\{z_j^d = t\}} & \text{if } t \text{ occupied,} \\ \alpha \sum_{\substack{\hat{d}=1 \\ \hat{d} \neq d}}^D \sum_{\hat{t}} \mathbb{1}_{\{k_{z_{\hat{t}}}^{\hat{d}} = k\}} & \text{if } t \text{ unoccupied and } k \text{ exists,} \\ \alpha\gamma & \text{if } t \text{ unoccupied and } k \text{ is new,} \end{cases} \quad (3.46)$$

$$\mathcal{P}_{jk}^d = \frac{\mathcal{B}(\alpha_{jk}^d + \sigma_j^d, \beta_{jk}^d + N^d - \sigma_j^d)}{\mathcal{B}(\alpha_{jk}^d, \beta_{jk}^d)}, \quad (3.47)$$

$$Z_k^d = \begin{cases} \sum_{\substack{\hat{d}=1 \\ \hat{d} \neq d}}^D \sum_{\hat{t}} \mathbb{1}_{\{k_{z_{\hat{t}}}^{\hat{d}} = k\}} & , \text{ if } k \text{ exists,} \\ \gamma & , \text{ if } k \text{ is new,} \end{cases} \quad (3.48)$$

where, noting that our previous definitions for α_{jk}^d and β_{jk}^d also apply to sets over functions, $\mathcal{J}$, in the following way:

$$\alpha_{\mathcal{J}k}^d = \alpha_k^d + \sum_{\substack{j=1 \\ j \notin \mathcal{J}}}^N \mathbb{1}_{\{k_{z_j^d}^d = k\}} \sigma_j^d, \quad (3.49)$$

$$\beta_{\mathcal{J}k}^d = \beta_k^d + \sum_{\substack{j=1 \\ j \notin \mathcal{J}}}^N \mathbb{1}_{\{k_{z_j^d}^d = k\}} (N^d - \sigma_j^d), \quad (3.50)$$

we have:

$$\mathcal{P}_{kt}^d = \frac{\mathcal{B}(\alpha_{\mathcal{J}k}^d + \sum_{j \in \mathcal{J}} \sigma_j^d, \beta_{\mathcal{J}k}^d + \sum_{j \in \mathcal{J}} (N^d - \sigma_j^d))}{\mathcal{B}(\alpha_{\mathcal{J}k}^d, \beta_{\mathcal{J}k}^d)}, \quad (3.51)$$

where $\mathcal{J} = \{j : z_j^d = t\}$.

UNCOLLAPSED GIBBS SAMPLING. After the first phase, we start sampling the error rates along with the rest of the variables and store the samples we obtain. During this phase, we use the following conditional probabilities:

$$P(p^d | \cdot) = \text{Beta}(\alpha_p + \sigma_y^d, \beta_p + N^d - \sigma_y^d), \quad (3.52)$$

$$P(y_i^d | \cdot) \propto (p^d)^{y_i^d} (1 - p^d)^{1 - y_i^d} \pi_i^d \quad (3.53)$$

$$P(z_j^d = t | k_t^d = k, \cdot) \propto Z_{kt}^d \mathcal{R}_{jk}^d, \quad (3.54)$$

$$P(\phi_k | \cdot) = \text{Beta}(\Phi^\alpha, \Phi^\beta), \quad (3.55)$$

$$P(k_t^d = k | \cdot) \propto Z_k^d \mathcal{R}_{kt}^d, \quad (3.56)$$

where:

$$\pi_i^d = \prod_{j=1}^N (e_j^d)^{\mathbb{1}_{\{y_{ij}^d \neq i_i^d\}}} (1 - e_j^d)^{\mathbb{1}_{\{y_{ij}^d = i_i^d\}}}, \quad (3.57)$$

$$\mathcal{R}_{jk}^d = (e_j^d)^{\sigma_j^d} (1 - e_j^d)^{N^d - \sigma_j^d}, \quad (3.58)$$

$$\mathcal{R}_{kt}^d = \prod_{j \in \mathcal{J}} (e_j^d)^{\sigma_j^d} (1 - e_j^d)^{N^d - \sigma_j^d}, \quad (3.59)$$

$$\Phi^\alpha = \alpha_e + \sum_{d=1}^D \sum_{j=1}^N \mathbb{1}_{\{k_{z_j^d}^d = k\}} \sigma_j^d, \quad (3.60)$$

$$\Phi^\beta = \beta_e + \sum_{d=1}^D \sum_{j=1}^N \mathbb{1}_{\{k_{z_j^d}^d = k\}} (N^d - \sigma_j^d), \quad (3.61)$$

where $\mathcal{J} = \{j : z_j^d = t\}$ and the remaining quantities are defined in the same way as for the collapsed sampling phase. In the case of missing data we use [Equation 3.8](#).

3.5 EXPERIMENTS

For our experiments, we use the same datasets that we used in [Section 2.5](#). However, we now compare to more baseline methods:

1. MAJ: This method consists of simply taking the most common label among the classifier outputs (i.e., the majority vote) as the combined label.
2. DW: This method was proposed by Dawid and Skene (1979). Results for this method have been omitted from our figures because they are several orders of magnitude worse than the ones that we have included.
3. GIBBS-SVM and GD-SVM: These methods were proposed by Tian and Zhu (2015).
4. DIRECT: This is the method we presented in [Chapter 2](#).

5. cBCC: This method was proposed by Moreno et al. (2015) and we adapted it to model error rates instead of the full confusion matrix.

Some of these methods do not explicitly estimate error rates, but rather combine the classifier outputs to produce a single label. For them we produce an estimate of the error rate by comparing the predicted labels to the ground truth labels. Also, we now report the mean squared error (MSE) of the error rate estimates when compared to the true error rates, rather than the mean absolute deviation (MAD). This is because this metric better emphasizes differences in cases when the differences are small. Lower MSE indicates better performance.

For all experiments and all three models, we use the following Gibbs sampling inference procedure: (i) we sample 4,000 samples that we throw away (i.e., burn-in samples), (ii) we sample 2,000 samples and keep every 10th sample in order to reduce the dependencies between the collected samples that are introduced by the sequential nature of the sampling procedure, and (iii) we obtain our error rate and label estimates by averaging over the collected samples. We repeat each experiment 10 times and report the mean of the evaluation metrics. We also compute the standard deviation of these metrics but we decided to omit it from the result figures because it is about 2 orders of magnitude smaller than the mean value. We use the following hyperparameters:

- Labels Prior: α_p and β_p are both set to 1 (uniform and uninformative prior).
- Error Rates Prior: α_e is set to 1 and β_e is set to 10. We have chosen these values in order to “avoid” the identifiability problem that was described in Chapter 2. This prior encodes our assumption that more than half of our classifiers have error rate lower than 0.5.
- DP and HDP Concentration Parameters: We carry out several experiments with many logarithmically spaced values for α and γ (all combinations of pairs of values were considered for the HDP) and we compute the log-likelihood for a held-out dataset, for each experiment. The results that we report for the DP and HDP models correspond to the experiment that results in the highest log-likelihood value for the held-out dataset.

The held-out dataset consists of a random sample that contains 10% of the available data.

More details can be found at our code repository: <https://github.com/eaplitanios/makina>. The results for all experiments are summarized in Figure 3.5 and are discussed in the following sections. In what follows, we use the following abbreviations: BAYES refers to our error estimation model of Section 3.2, C-BAYES refers to our coupled error estimation model of Section 3.3, and finally, HC-BAYES refers to our hierarchically coupled error estimation model of Section 3.4.

3.5.1 Results on Error Estimation

Our results on error rate estimation are shown in Figure 3.5. It is clear that HC-BAYES consistently outperforms the competing methods. In the presence of dependencies across domains and classifiers, we expect that C-BAYES and HC-BAYES perform better than BAYES when we have a small amount of data. This is because that is when sharing

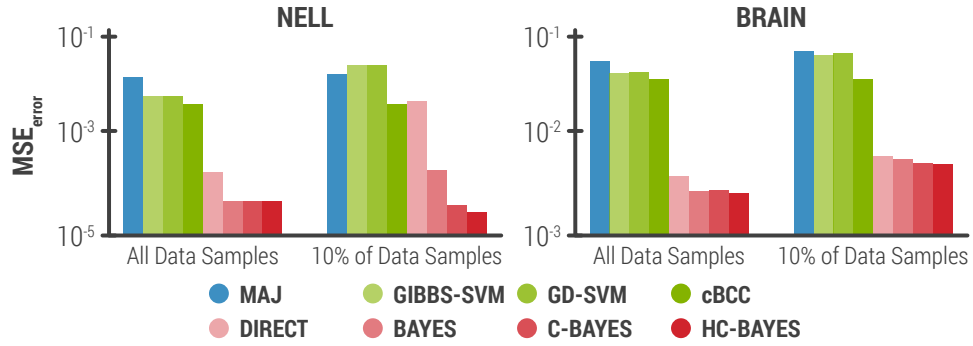


Figure 3.5: Mean squared error (MSE) of the estimated error rates for our methods (shown in red color) and the methods that we are competing against (shown in blue and green color). The lower the MSE (i.e., the shorter the bar), the better the result is. It is clear from these plots that our methods outperform the competing methods in all cases. Error bars have been omitted from these plots because they are negligibly small (~ 2 orders of magnitude smaller than the reported values). Note that these plots use a logarithmic scale.

information becomes useful. When we have a large amount of data, we expect that the performance of the simple BAYES model will be similar to its extensions, since for C-BAYES and HC-BAYES, the atoms may not be clustered as there may be enough data per atom to render information sharing unnecessary. These expectations are evidently met in our results.

For all experiments where we use all data samples we see that the three proposed methods perform equivalently well and always beat the competing methods. The fact that the coupling introduced by the nonparametric extensions does not offer an improvement in performance can be attributed to the fact that for all these experiments we have enough data for each error rate to be modeled independently (i.e., without being clustered). The three proposed models do not perform identically. This is probably due to the fact that they use different priors, thus enabling different levels of information sharing. In the case of limited data samples (i.e., 10% of the data samples), HC-BAYES performs best, followed by C-BAYES. This supports our argument that coupled error estimation methods are more powerful in cases where there is only a limited amount of available data. Despite the fact that this is not really the case with the full NELL dataset—or other web-scale projects—it is a common scenario that is encountered with other types of data, such as data in neuroscience and biology. Note that, in such cases, even unlabeled data can be very hard and expensive to obtain. Finally, note that cBCC is the closest method to ours and also beats competing methods in most cases. However, our methods are always able to outperform cBCC. This is probably due to the fact that this model only allows for information sharing among different classifiers (i.e., it clusters the classifiers), but none of our datasets involves a high number of classifiers.

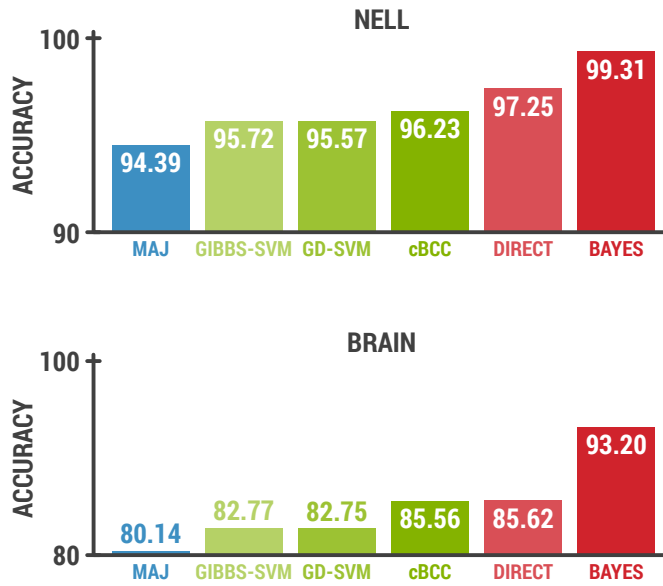


Figure 3.6: Accuracy for ground truth estimation for our methods (shown in red color) and the methods that we are competing against (shown in blue and green color). Higher numbers imply better performance.

3.5.2 Results on Ground Truth Estimation

Our methods are also able to directly estimate the most likely ground truth labels. Therefore, it is natural to also perform the experiment of [Section 2.5.2](#). Our results for this experiment are shown in [Figure 3.6](#). It is interesting to see that DIRECT manages to still outperform the other baselines. Most importantly though, BAYES is able to provide a further boost over DIRECT; especially so for the brain dataset. This is important because, as discussed in [Section 2.6](#), the highly dependent classifiers in the brain dataset formed the main motivation for the models presented in this chapter.

Note that we only use the simple BAYES model here because it performs similarly to C-BAYES and HC-BAYES when using the full datasets.

3.6 KEY TAKEAWAYS

In this chapter, we presented a way to address one of the key limitations of our DIRECT approach for estimating accuracies from unlabeled data. The core idea behind the newly proposed methods can also be extended to model the full confusion matrix for general discrete labels (i.e., instead of just binary labels that we consider in this chapter). This can be useful in several applications where one needs to know how the error rates decompose into precision and recall, for example. In fact, in [Chapter 5](#) we shall return to the BAYES method and extend it to model the full confusion matrix and also address another limitation of the DIRECT approach. The resulting method outperforms the current state-of-the-art for this problem, while also having multiple applications beyond accuracy estimation and learning from imperfect labels.

A LOGIC-BASED APPROACH TO ESTIMATING ACCURACY

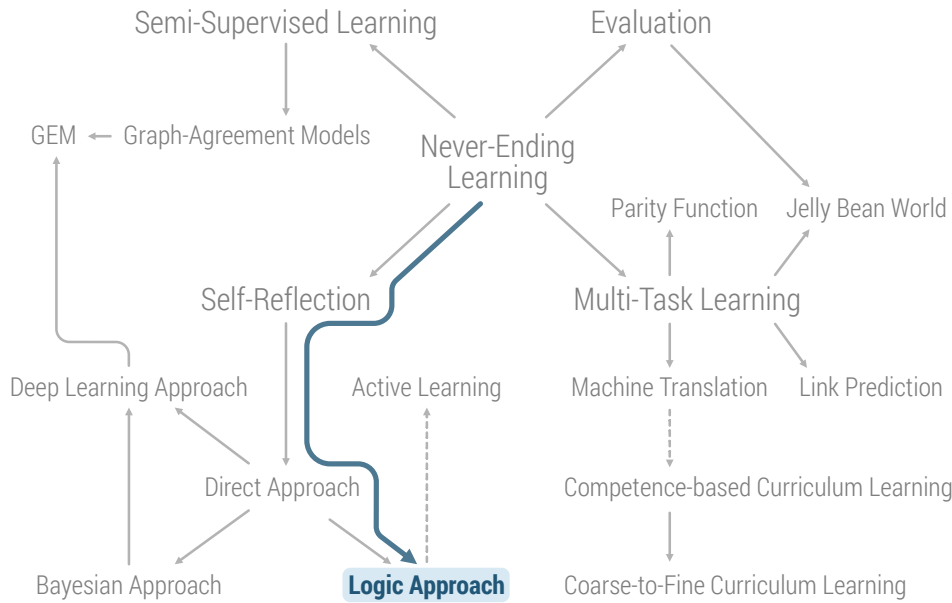


Figure 4.1: Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in [Section 1.4](#).

In [Chapter 2](#), we showed that learning collections of functions allows us to perform completely unsupervised evaluation, by proposing a direct approach to do this based on intuition about how we, as humans, would approach this problem. However, in [Section 2.6](#), we also identified and discussed some limitations of this approach. In the previous chapter we addressed the first one of these limitations: the direct approach cannot directly model the dependencies between the learned functions. In this chapter, we propose a method that addresses the second limitation: the previously proposed approaches cannot use a weak form of supervision in the form of logical constraints between the function predictions, which is often already available or easy to obtain. As we shall show, the proposed method is based on probabilistic logic and it successfully addresses this limitation.

4.1 INTRODUCTION

In the previous chapters we established why estimating the accuracy of classifiers is central to machine learning and many other fields. The two main motivating examples

in our case have been never-ending learning systems like NELL, and the use of crowd-sourcing for collecting training data for machine learning systems, at scale. In both these cases it is common to encounter tasks which involve making several predictions that are tied together by *logical constraints*. In fact, such learning tasks are abundant in machine learning and multi-task learning, which forms one of the focus areas of this thesis. As an example, we may have two classifiers in NELL which predict whether noun phrases represent animals or cities, respectively, and we know that something cannot be both an animal and a city at the same time (i.e., the two categories are mutually exclusive). In such cases, we observe that if the predictions of the system violate at least one of the constraints, then at least one of the system’s components must be wrong. This chapter extends this intuition and presents an *unsupervised* approach (i.e., only *unlabeled data* are needed) for estimating accuracies of classifiers, that is able to use information provided by such logical constraints. Furthermore, the proposed approach is also able to use any available labeled data, thus also being applicable to *semi-supervised* settings. Most importantly, this chapter addresses a key limitation of the DIRECT approach, that was described in [Section 2.6](#).

We once again consider the “multiple approximations” problem setting that was described in [Section 2.2](#). We have several different approximations, $\hat{f}_1^d, \dots, \hat{f}_{M^d}^d$, to a set of target boolean classification functions, $f^d : \mathcal{X} \mapsto \{0, 1\}$ for $d = 1, \dots, D$, and we wish to know the true accuracies of each of these different approximations, using only unlabeled data, as well as the response of the true underlying functions, f^d . For convenience we sometimes use the notation $\hat{y}_{ij}^d = \hat{f}_j^d(x_i)$. Each value of d characterizes a different *domain* (or problem setting) and each domain can be interpreted as a class or category of objects. Similarly, the function approximations can be interpreted as classifying inputs as belonging to these categories or not. We consider the case where we may have a set of logical constraints defined over the domains. In contrast to prior work, we allow the function approximations to provide soft responses in the interval $[0, 1]$ (as opposed to only allowing binary responses—i.e., they can now return the probability for the response being 1), thus allowing modeling their “certainty.” As an example of this setting, to which we will often refer throughout this chapter, let us consider a part of NELL, where the input space of our functions $\mathcal{X}$ is the space of all possible noun phrases (NPs). Each target function f^d returns a boolean value indicating whether the input NP belongs to a category, such as *city* or *animal*, and these categories correspond to our domains. There also exist logical constraints between these categories that may be hard (i.e., strongly enforced) or soft (i.e., enforced in a probabilistic manner). For example, *city* and *animal* may be mutually exclusive (i.e., if an object belongs to the category *city*, then it is unlikely that it also belongs to the category *animal*). In this case, the function approximations correspond to different classifiers (potentially using a different set of features / different views of the input data), which may return a probability for a NP belonging to a class, instead of a binary value. Our goal is to estimate the accuracies of these classifiers using only unlabeled data. In order to quantify accuracy, we define the error rate of classifier j in domain d as $e_j^d \triangleq p_{x \sim \mathcal{D}}(\hat{f}_j^d(x) \neq f^d(x))$, for the binary case, for $j = 1, \dots, M^d$, where $\mathcal{D}$ is the true underlying distribution of the input data. Note

that accuracy is equal to one minus error rate. This definition may be relaxed for the case where $\hat{f}_j^d(x) \in [0, 1]$ represents a probability:

$$e_j^d \triangleq \hat{f}_j^d(x)p_{\sim \mathcal{D}}(f^d(x) \neq 1) + (1 - \hat{f}_j^d(x))p_{x \sim \mathcal{D}}(f^d(x) \neq 0), \quad (4.1)$$

which resembles the expected probability of error. Even though our work is motivated by the use of logical constraints defined over the domains, we also consider the setting where there are no such constraints.¹ The proposed method consists of:

1. defining a set of logic rules for modeling the logical constraints between f^d and $\hat{f}_j^d$, in terms of the error rates e_j^d and the known logical constraints, and
2. performing probabilistic inference using these rules as priors, in order to obtain the most likely values of the e_j^d and the f^d , which are not observed.

The key intuition behind our method is that, if the constraints are violated for the function approximation outputs, then at least one of these function approximations has to be making an error. For example, in the case of NELL, if two function approximations respond that a NP belongs to the city and the animal categories, respectively, then at least one of them has to be making an error. We define the form of the logic rules in Section 4.3 and then describe how to perform probabilistic inference over them in Section 4.4. An overview of the proposed method is shown in Figure 4.2. In the next section we introduce the notion of *probabilistic logic*, which fuses classical logic with probabilistic reasoning and forms the backbone of our method.

4.2 PROBABILISTIC LOGIC

In *classical logic*, we have a set of *predicates* (e.g., $\text{mammal}(x)$ indicating whether x is a mammal, where x is a variable) and a set of *rules* defined in terms of these predicates (e.g., $\text{mammal}(x) \rightarrow \text{animal}(x)$, where “ $\rightarrow$ ” can be interpreted as “IMPLIES”). We refer to predicates and rules defined for a particular instantiation of their variables as *ground predicates* and *ground rules*, respectively (e.g., $\text{mammal}(\text{whale})$ and $\text{mammal}(\text{whale}) \rightarrow \text{animal}(\text{whale})$). These ground predicates and rules take boolean values (i.e., are either true or false—for rules, the value is true if the rule holds). Our goal is to infer the most likely values for a set of unobserved ground predicates, given a set of observed ground predicate values and logic rules. In *probabilistic logic*, we are instead interested in inferring the probabilities of these ground predicates and rules being true, given a set of observed ground predicates and rules. Furthermore, the truth values of ground predicates and rules may be continuous and lie in the interval $[0, 1]$, instead of being boolean, representing the probability that the corresponding ground predicate or rule is true. In this case, boolean logic operators, such as AND ($\wedge$), OR ($\vee$), NOT ($\neg$), and IMPLIES ($\rightarrow$), need to be redefined. In the next section we assume their classical logical interpretation.

¹ This work has been published in (Platanios et al., 2017b).

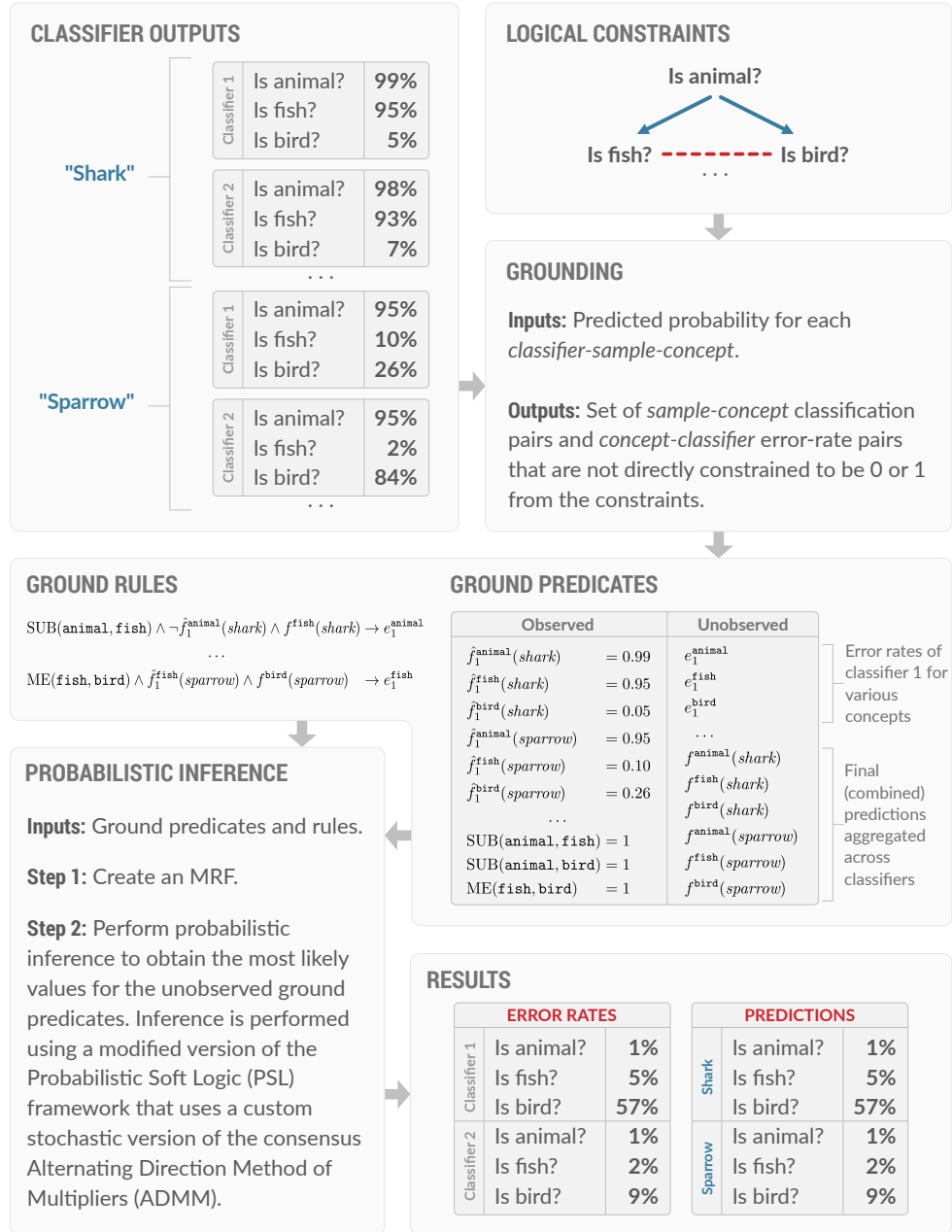


Figure 4.2: Proposed method overview. The classifier outputs and the logical constraints make up the system inputs. The representation of the logical constraints in terms of the function approximation error rates is described in Section 4.3. In the logical constraints box, blue arrows represent subsumption constraints and red dashed lines represent mutually exclusion constraints. Given the inputs, the first step is grounding (i.e., computing all feasible ground predicates and rules that the system will need to perform inference over) and it is described in Section 4.4.2. In the ground rules box, $\wedge$, $\neg$, and $\rightarrow$ correspond to the logical AND, OR, and IMPLIES operators, respectively. Then, inference is performed in order to infer the most likely truth values of the unobserved ground predicates, given the observed ones and the ground rules (described in detail in Section 4.4). The results include: (i) the estimated error rates, and (ii) the most likely target function outputs (i.e., combined predictions).

4.3 PROPOSED LOGIC RULES

As described earlier, our goal is to estimate the true accuracies of each of the function approximations, $\hat{f}_1^d, \dots, \hat{f}_{M^d}^d$ for $d = 1, \dots, D$, using only unlabeled data, as well as the response of the true underlying functions, f^d . We now define the logic rules that we perform inference over in order to achieve this goal. The rules are defined in terms of the following predicates, for $d = 1, \dots, D$:

- Function Approximation Outputs: $\hat{f}_j^d(x)$, defined over all approximations $j = 1, \dots, M^d$, and inputs $x \in \mathcal{X}$, for which the corresponding function approximation has provided a response. Note that the values of these ground predicates lie in $[0, 1]$ due to their probabilistic nature (i.e., they do not have to be binary, as in related work), and some of them are observed.
- Target Function Outputs: $f^d(x)$, defined over all inputs $x \in \mathcal{X}$. Note that in contrast to the semi-supervised setting, in the purely unsupervised setting, none of these ground predicate values are observed.
- Function Approximation Error Rates: e_j^d , defined over all approximations $j = 1, \dots, M^d$. Note that none of these ground predicate values are observed. The primary goal of the proposed method is to infer their values.

The goal of the logic rules we define is two-fold: (i) combine the function approximation outputs in a single output value, and (ii) account for the logical constraints between the domains. We aim to achieve both goals while accounting for the error rates of the function approximations. We first define a set of rules that relate the function approximation outputs to the true underlying function outputs. We call this set of rules the *ensemble rules* and we describe them in the following section. We then discuss how to account for logical constraints between the domains.

4.3.1 Ensemble Rules

The first set of rules specifies a relation between the target function outputs $f^d(x)$ and the function approximation outputs $\hat{f}_j^d(x)$, independent of the logical constraints:

$$\hat{f}_j^d(x) \wedge \neg e_j^d \rightarrow f^d(x), \quad \neg \hat{f}_j^d(x) \wedge \neg e_j^d \rightarrow \neg f^d(x), \quad (4.2)$$

$$\hat{f}_j^d(x) \wedge e_j^d \rightarrow \neg f^d(x), \quad \text{and} \quad \neg \hat{f}_j^d(x) \wedge e_j^d \rightarrow f^d(x), \quad (4.3)$$

for $d = 1, \dots, D$, $j = 1, \dots, M^d$, and $x \in \mathcal{X}$. In words, the first set of rules state that if a function approximation is not making an error, then its output should match the output of the target function. The second set of rules state that if a function approximation is making an error, then its output should not match the output of the target function.

It is interesting to note that the ensemble rules effectively constitute a weighted majority vote for combining the function approximation outputs, where the weights are determined by the error rates of the approximations. These error rates are implicitly computed based on agreement between the function approximations. This is related

the DIRECT method we presented in [Chapter 2](#). There we tried to answer the question of whether consistency in the outputs of the approximations implies correctness, and directly used the agreement rates of the approximations in order to estimate their error rates. Thus, there exists an interesting connection between DIRECT and this logic-based approach in that here we also implicitly use agreement rates to estimate error rates, and, as we show in [Section 4.5](#), our results—even though significantly improving upon DIRECT—reinforce our earlier claim.

4.3.2 Constraint Rules

The space of possible logical constraints is huge; we do not deal with every possible constraint in this chapter. Instead, we focus our attention on two types of constraints that are abundant in structured prediction problems in machine learning, and which are motivated by the use of our method in the context of NELL:

- Mutual Exclusion: If domains d_1 and d_2 are mutually exclusive, then $f^{d_1} = 1$ implies that $f^{d_2} = 0$. For example, in the NELL setting, if a NP belongs to the city category, then it cannot also belong to the animal category.
- Subsumption: If d_1 subsumes d_2 , then if $f^{d_2} = 1$, we must have that $f^{d_1} = 1$. For example, in the NELL setting, if a NP belongs to the cat category, then it must also belong to the animal category.

This set of constraints is sufficient to model most ontology constraints between categories in NELL, as well as a big subset of the constraints generally used in practice.

A set of ME domains can be reduced to pairwise ME constraints for all pairs in that set.

MUTUAL EXCLUSION RULE. We first define the predicate $\text{ME}(d_1, d_2)$, indicating that domains d_1 and d_2 are mutually exclusive. This predicate has value 1 if domains d_1 and d_2 are mutually exclusive and value 0 otherwise, and its truth value is assumed

IDENTIFIABILITY

Let us consider flipping the values of all error rates (i.e., setting them to one minus their value) and the target function responses. Then, the ensemble logic rules would evaluate to the same value as before (i.e., satisfied or unsatisfied). Therefore, the error rates and the target function values are not identifiable when there are no logical constraints. As we will see in the next section, the constraints may sometimes help resolve this issue as the corresponding logic rules often do not exhibit this kind of symmetry. However, for cases where this symmetry exists we can resolve it by assuming that most of the function approximations have error rates better than chance (i.e., < 0.5). This can be done by considering the following two rules: $\hat{f}_j^d(x) \rightarrow f^d(x)$, and $\neg \hat{f}_j^d(x) \rightarrow \neg f^d(x)$, for $d = 1, \dots, D$, $j = 1, \dots, M^d$, and $x \in \mathcal{X}$. Note that all these rules imply is that $\hat{f}_j^d(x) = f^d(x)$; that is, they represent the prior belief that the function approximations are correct. As will be discussed in [Section 4.4](#), in probabilistic frameworks where rules are weighted with a real value in $[0, 1]$, these rules will be given a weight that represents their significance (or strength). In such a framework, we can consider using a smaller weight for these prior belief rules compared to the remainder of the rules, which would simply correspond to a *regularization* weight. Note that this weight can be a tunable or even a learnable parameter.

to be observed for all values of d_1 and d_2 . Furthermore, note that it is *symmetric*, meaning that if $\text{ME}(d_1, d_2)$ is true, then $\text{ME}(d_2, d_1)$ is also true. We define the mutual exclusion logic rule as follows:

$$\text{ME}(d_1, d_2) \wedge \hat{f}_j^{d_1}(x) \wedge f^{d_2}(x) \rightarrow e_j^{d_1}, \quad (4.4)$$

for $d_1 \neq d_2 = 1, \dots, D$, $j = 1, \dots, M^{d_1}$, and $x \in \mathcal{X}$. In words, this rule says that if $f^{d_2}(x) = 1$ and domains d_1 and d_2 are mutually exclusive, then $\hat{f}_j^{d_1}(x)$ must be equal to 0, as it is an approximation to $f^{d_1}(x)$ and ideally we want $\hat{f}_j^{d_1}(x) = f^{d_1}(x)$. If that is not the case, then $\hat{f}_j^{d_1}$ must be making an error.

SUBSUMPTION RULE. We first define the predicate $\text{SUB}(d_1, d_2)$, indicating that domain d_1 subsumes domain d_2 . This predicate has value 1 if domain d_1 subsumes domain d_2 , and 0 otherwise, and its truth value is always observed. Note that, unlike mutual exclusion, this predicate is not symmetric. We define the subsumption logic rule as follows:

$$\text{SUB}(d_1, d_2) \wedge \neg \hat{f}_j^{d_1}(x) \wedge f^{d_2}(x) \rightarrow e_j^{d_1}, \quad (4.5)$$

for $d_1, d_2 = 1, \dots, D$, $j = 1, \dots, M^{d_1}$, and $x \in \mathcal{X}$. In words, this rule says that if $f^{d_2}(x) = 1$ and d_1 subsumes d_2 , then $\hat{f}_j^{d_1}(x)$ must be equal to 1, as it is an approximation to $f^{d_1}(x)$ and ideally we want $\hat{f}_j^{d_1}(x) = f^{d_1}(x)$. If that is not the case, then $\hat{f}_j^{d_1}$ must be making an error.

Having defined all of the logic rules that comprise our model, in the next section we describe how to perform inference. Inference in this case comprises determining the most likely truth values of the unobserved ground predicates, given the observed predicates and the set of rules that comprise our model.

4.4 INFERENCE

In [Section 4.2](#) we introduced the notion of probabilistic logic and defined our model in terms of probabilistic predicates and rules. In this section we discuss in more detail the implications of using probabilistic logic, and the way in which we perform inference. There exist various probabilistic logic frameworks, each making different assumptions. In what is arguably the most popular such framework, *Markov logic networks (MLNs)* which were proposed by Richardson and Domingos (2006), inference is performed over a constructed *Markov random field* (MRF; Hammersley and Clifford, 1971) based on the model logic rules. Each potential function in the MRF corresponds to a ground rule and takes an arbitrary positive value when the ground rule is satisfied and the value 0 otherwise (the positive values are often called *rule weights* and can be either fixed or learned). Each variable is boolean-valued and corresponds to a ground predicate. MLNs are thus a direct probabilistic extension to boolean logic. It turns out that due to the discrete nature of the variables in MLNs, inference is NP-hard and can thus be very inefficient. Part of our goal in this chapter is for our method to be applicable at a

very large scale (e.g., for systems like NELL). We thus resort to *probabilistic soft logic* (PSL; Bröcheler et al., 2010; Pujara et al., 2013), which can be thought of as a convex relaxation of MLNs. Note that the model proposed in the previous section be used with various probabilistic logic frameworks. Our decision to use PSL is motivated by scalability. One could just as easily perform inference for our model using MLNs, or any other such framework.

4.4.1 Probabilistic Soft Logic

In PSL, models, which are composed of a set of logic rules, are represented using *hinge-loss Markov random fields* (HL-MRFs; Bach et al., 2013). In this case, inference amounts to solving a convex optimization problem. Variables of the HL-MRF correspond to soft truth values of ground predicates. Specifically, a HL-MRF f is a probability density over m random variables, $\mathbf{Y} = \{Y_1, \dots, Y_m\}$ with domain $\mathbf{D} = [0, 1]^m$, corresponding to the unobserved ground predicate values. Let $\mathbf{X} = \{X_1, \dots, X_n\}$ be an additional set of variables with known values in the domain $[0, 1]^n$, corresponding to observed ground predicate values. Let $\phi = \{\phi_1, \dots, \phi_k\}$ be a finite set of k continuous potential functions of the form $\phi_j(\mathbf{X}, \mathbf{Y}) = (\max\{\ell_j(\mathbf{X}, \mathbf{Y}), 0\})^{p_j}$, where ℓ_j is a linear function of $\mathbf{X}$ and $\mathbf{Y}$, and $p_j \in \{1, 2\}$. We will see how these functions relate to the ground rules of the model. Given the above, for a set of non-negative free parameters $\lambda = \{\lambda_1, \dots, \lambda_k\}$ (i.e., the equivalent of MLN rule weights), the HL-MRF density is defined as:

$$f(\mathbf{Y}) = \frac{1}{Z} \exp \left\{ - \sum_{j=1}^k \lambda_j \phi_j(\mathbf{X}, \mathbf{Y}) \right\}, \quad (4.6)$$

where Z is a normalizing constant so that f is a proper probability density function. Our goal is to infer the most probable explanation (MPE), which consists of the values of $\mathbf{Y}$ that maximize the likelihood of our data (as opposed to performing marginal inference which aims to infer the marginal distribution of these values). This is equivalent to solving the following convex problem:

$$\min_{\mathbf{Y} \in [0, 1]^m} \sum_{j=1}^k \lambda_j \phi_j(\mathbf{X}, \mathbf{Y}). \quad (4.7)$$

Each variable X_i or Y_i corresponds to a soft truth value (i.e., $Y_i \in [0, 1]$) of a ground predicate. Each function ℓ_j corresponds to a measure of the *distance to satisfiability* of a logic rule. The set of rules we use is what characterizes a particular PSL model. The rules represent prior knowledge we have about the problem we are trying to solve. For our model, these rules were defined in Section 4.3. As mentioned above, variables are allowed to take values in the interval $[0, 1]$. We thus need to define what we mean by the truth value of a rule and its distance to satisfiability. For the logical operators

AND ($\wedge$), OR ($\vee$), NOT ($\neg$), and IMPLIES ($\rightarrow$), we use the definitions from Łukasiewicz Logic (Klir and Yuan, 1995):

$$P \wedge Q \triangleq \max\{P + Q - 1, 0\}, \quad (4.8)$$

$$P \vee Q \triangleq \min\{P + Q, 1\}, \quad (4.9)$$

$$\neg P \triangleq 1 - P, \text{ and} \quad (4.10)$$

$$P \rightarrow Q \triangleq \min\{1 - P + Q, 1\}. \quad (4.11)$$

Note that these operators are a continuous relaxation of the corresponding boolean operators, in that for boolean-valued variables, with 0 corresponding to FALSE and 1 to TRUE, they are equivalent. By writing all logic rules in the form:

$$B_1 \wedge B_2 \wedge \cdots \wedge B_s \rightarrow H_1 \vee H_2 \vee \cdots \vee H_t, \quad (4.12)$$

it is easy to observe that the distance to satisfiability (i.e., 1 minus its truth value) of a rule evaluates to:

$$\max\{0, \sum_{i=1}^s B_i - \sum_{j=1}^t H_j + 1 - s\}. \quad (4.13)$$

Note that any set of rules of first-order predicate logic can be represented in this form (Bröcheler et al., 2010), and that minimizing this quantity amounts to making the rule “more satisfied.”

In order to complete our method description it remains to describe: (i) how to obtain a set of ground rules and predicates from a set of logic rules of the form presented in section Section 4.3 and a set of observed ground predicates, and define the objective function of Equation 4.7, and (ii) how to solve the optimization problem of this equation to obtain the most likely truth values for the unobserved ground predicates. These two steps are described in the following two sections.

4.4.2 Grounding

Grounding is the process of computing all possible groundings of each logic rule to construct the inference problem variables and the objective function. As already described in Section 4.4.1, the variables $\mathbf{X}$ and $\mathbf{Y}$ correspond to ground predicates and the functions ℓ_j correspond to ground rules. The easiest way to ground a set of logic rules would be to go through each one and create a ground rule instance of it, for each possible value of its arguments. However, if a rule depends on n variables and each variable can take m possible values, then m^n ground rules would be generated. For example, the mutual exclusion rule of Equation 4.4 depends on d_1 , d_2 , j , and X , meaning that $D^2 \times M^{d_1} \times |X|$ ground rule instances would be generated, where $|X|$ denotes the number of values that X can take. The same applies to predicates. $\hat{f}_j^{d_1}(X)$ would result in $D \times M^{d_1} \times |X|$ ground instances, which would become variables in our optimization problem. This approach would thus result in a huge optimization problem rendering it impractical when dealing with large scale problems such as NELL. The key to scaling up the grounding procedure is to notice that many of the

Algorithm 4.1: Grounding algorithm.

Inputs: $\hat{f}_j^d(x)$, for $d = 1, \dots, D$, and $j = 1, \dots, M^d$, which are observed.
 Set of pairwise mutual-exclusion constraints $ME = \{d_1^i, d_2^i\}_{i=1}^{C_m}$.
 Set of subsumption constraints $SUB = \{d_1^i, d_2^i\}_{i=1}^{C_s}$.

- 1 Create empty set G_p of ground predicates.
- 2 Create empty set G_l of ground rules.
- 3 **foreach** *observed* $\hat{f}_j^d(x)$ **do**
- 4 Add $\hat{f}_j^d(x)$, e_j^d , and $f^d(x)$ to G_p .
- 5 Add $\hat{f}_j^d(x) \wedge \neg e_j^d \rightarrow f^d(x)$ and $\neg \hat{f}_j^d(x) \wedge \neg e_j^d \rightarrow \neg f^d(x)$ to G_l .
- 6 Add $\hat{f}_j^d(x) \wedge e_j^d \rightarrow \neg f^d(x)$ and $\neg \hat{f}_j^d(x) \wedge e_j^d \rightarrow f^d(x)$ to G_l .
- 7 Add $\hat{f}_j^d(x) \rightarrow f^d(x)$ and $\neg \hat{f}_j^d(x) \rightarrow \neg f^d(x)$ to G_l .
- 8 **foreach** $(d_1, d_2) \in ME$ **do**
- 9 **if** $d_1 = d$ **then**
- 10 Add $f^{d_2}(x)$ to G_p .
- 11 Add $ME(d_1, d_2) \wedge \hat{f}_j^{d_1}(x) \wedge f^{d_2}(x) \rightarrow e_j^{d_1}$ to G_l .
- 12 **else if** $d_2 = d$ **then**
- 13 Add $f^{d_1}(x)$ to G_p .
- 14 Add $ME(d_2, d_1) \wedge \hat{f}_j^{d_2}(x) \wedge f^{d_1}(x) \rightarrow e_j^{d_2}$ to G_l .
- 15 **foreach** $(d_1, d_2) \in SUB$ **do**
- 16 **if** $d_1 = d$ **then**
- 17 Add $f^{d_2}(x)$ to G_p .
- 18 Add $SUB(d_1, d_2) \wedge \neg \hat{f}_j^{d_1}(x) \wedge f^{d_2}(x) \rightarrow e_j^{d_1}$ to G_l .

Output: Set G_p of ground predicates and set G_l of ground rules.

possible ground rules are always satisfied (i.e., have distance to satisfiability equal to 0), irrespective of the values of the unobserved ground predicates that they depend on. These ground rules would therefore not influence the optimization problem solution and can be safely ignored. Since in our model we are only dealing with a small set of predefined logic rule forms, we devised a heuristic grounding procedure that only generates ground rules and predicates that may influence the optimization. Our grounding algorithm is shown in [Algorithm 4.1](#) and is based on the idea that a ground rule is only useful if the function approximation predicate that appears in its body is observed. It turns out that this approach is orders of magnitude faster than existing state-of-the-art solutions such as the grounding solution used by Niu et al. (2011).

4.4.3 Optimization

For large problems, the objective function of [Equation 4.7](#) will be a sum of potentially millions of terms, each one of which only involving a small set of variables. In PSL, the method used to solve this optimization problem is based on the consensus *alternating directions method of multipliers* (ADMM; Boyd et al., 2011). The approach consists of handling each term in that sum as a separate optimization problem using copies

Algorithm 4.2: PSL consensus ADMM inference algorithm.

Inputs: Observed ground predicate values $\mathbf{X}$.
 Objective terms ℓ and $\mathbf{p}$.
 Rule weights λ .
 Parameter ρ .
 Mapping $\mathcal{G}$ from variable copy indices to consensus indices.

- 1 Initialize all $\mathbf{Y}$ (consensus variables) and α_j (Lagrange multipliers) for $j = 1, \dots, k$, randomly.
- 2 Initialize the variable copies $\mathbf{y}_j$ for $j = 1, \dots, k$, corresponding to each subproblem, randomly.
- 3 **while not converged do**
- 4 **for** $i = 1, \dots, k$ **do**
- 5 $\alpha_j \leftarrow \alpha_j + \rho(\mathbf{y}_j - \mathbf{Y}_{\mathcal{G}(j,:)})$.
- 6 $\mathbf{y}_j \leftarrow \arg \min_{\mathbf{y}_j} [\lambda_j [\max\{\ell_j(\mathbf{X}, \mathbf{y}_j)\}]^{p_j} + \frac{\rho}{2} \|\mathbf{y}_j - \mathbf{Y}_{\mathcal{G}(j,:)} + \frac{1}{\rho} \alpha_j\|_2^2]$.
- 7 **for** $i = 1, \dots, \text{length}(\mathbf{Y})$ **do**
- 8 $\mathbf{Y}_i \leftarrow \frac{\sum_{\mathcal{G}(j,d)=i} ([\mathbf{y}_j]_d + \frac{1}{\rho} [\alpha_j]_d)}{\sum_{\mathcal{G}(j,d)=i} 1}$.
- 9 Project $\mathbf{Y}_i$ on the interval $[0, 1]$.

Output: Inferred ground predicate values $\mathbf{Y}$.

of the corresponding variables, while adding the constraint that all copies of each variable must be equal. This allows for solving the subproblems completely in parallel and is thus scalable. The algorithm is summarized in Algorithm 4.2. More details on this algorithm and on its convergence properties can be found in the latest PSL paper by Bach et al. (2017). We propose a stochastic variation of this consensus ADMM method that is much more scalable and can be used for systems like NELL.

During each iteration, instead of solving all subproblems and aggregating their solutions in the consensus variables, we sample $K \ll k$ subproblems to solve. The probability of sampling each subproblem is proportional to the distance of its variable copies from the respective consensus variables. The intuition and motivation behind this approach is that at the solution of the optimization problem, all variable copies should be in agreement with the consensus variables. Therefore, prioritizing subproblems whose variables are in greater disagreement with the consensus variables might facilitate faster convergence. Indeed, this modification to the inference algorithm allowed us to apply our method to the NELL dataset and obtain results within minutes instead of hours.

4.5 EXPERIMENTS

We initially perform experiments with exactly the same setup as in the previous chapter (described in Section 3.5), except that we now use more evaluation metrics than we did before. We refer to the logic-based method that was introduced in this chapter as LOGIC for the remainder of this section. Same as before, our implementation as well as the experiment datasets are available at <https://github.com/eaplatanios/makina>.

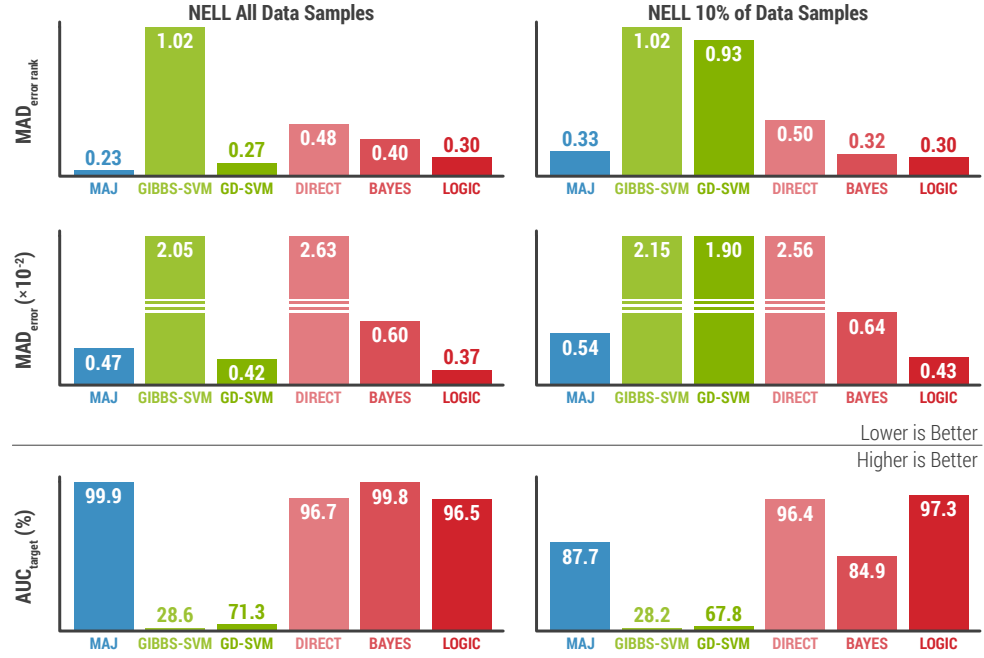


Figure 4.3: Results on the NELL dataset where no logical constraints are provided. For the first two rows, lower numbers imply better performance, but the opposite is true for the last row.

4.5.1 Evaluation Metrics

- Error Rank MAD: We rank the function approximations by our error rate estimates and by their true error rates to produce two vectors containing the ranks. We then compute the mean absolute deviation (MAD) between the two vectors, where by MAD we mean the ℓ_1 norm of the vectors' difference (same as in Section 2.5).
- Error MAD: MAD between the vector of our error rate estimates and the vector of the true error rates, where each vector is indexed by the function approximation index (same as in Section 2.5).
- Target AUC: Area under the precision-recall curve for the inferred target function values relative to the ground truth labels.

We compute these metrics for each domain and then average across domains to eventually obtain a single score for each method.

4.5.2 Results without Logical Constraints

The results for the NELL dataset are shown in Figure 4.3 and the results for the brain dataset are shown in Figure 4.4. MAJ performs well when a lot of data is available. However, LOGIC manages to outperform all alternative approaches in almost all cases. This makes it clear that our method can also be used effectively in cases where there are no logical constraints. In order to also provide results that are more comparable to those of the previous chapters, we also perform the experiment described in Sections 2.5.2

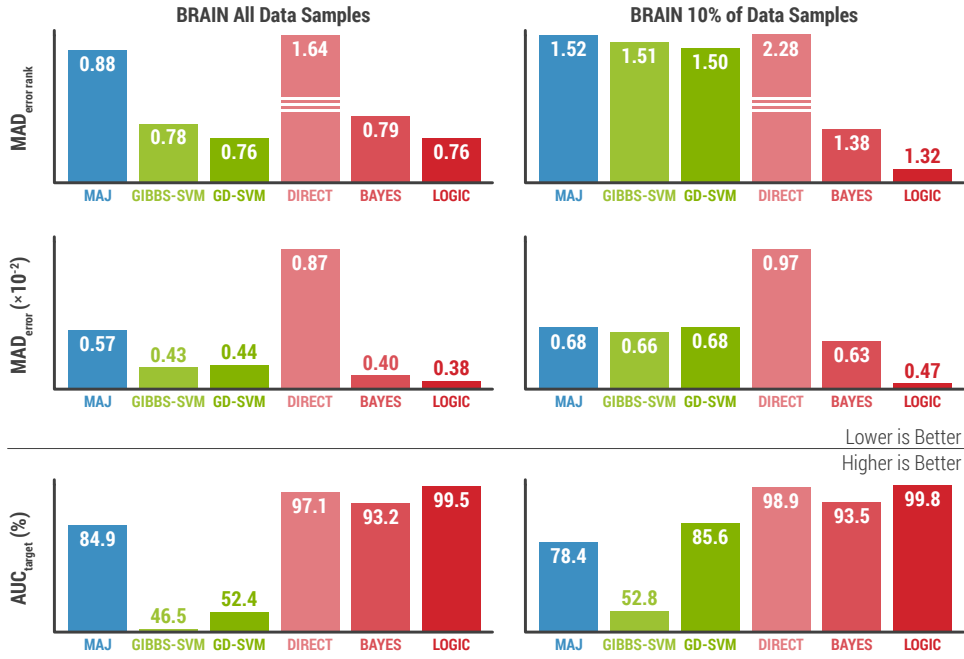


Figure 4.4: Results on the brain dataset where no logical constraints are provided. For the first two rows, lower numbers imply better performance, but the opposite is true for the last row.

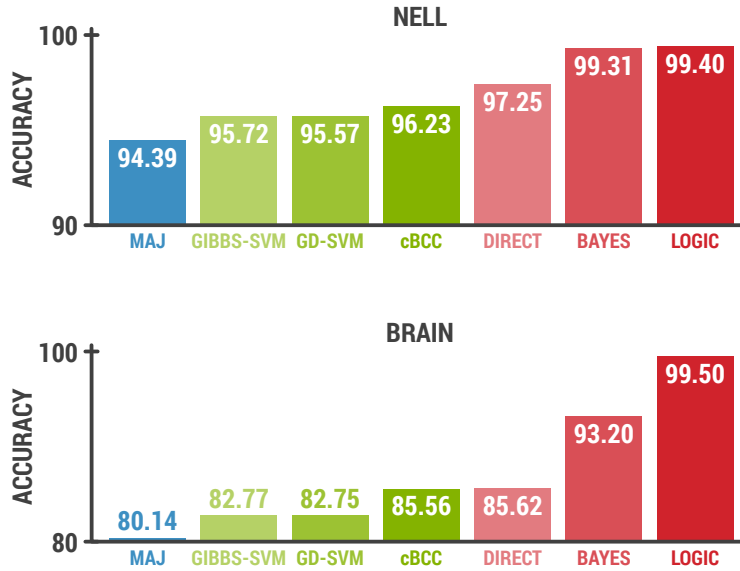


Figure 4.5: Accuracy for ground truth estimation, for the two datasets we used in our experiments. Higher numbers imply better performance.

and 3.5.2. Our results for this experiment are shown in Figure 4.5. It is interesting to see how LOGIC is able to outperform all our previous methods even in the case where no logical constraints are provided. However, our goal when developing LOGIC

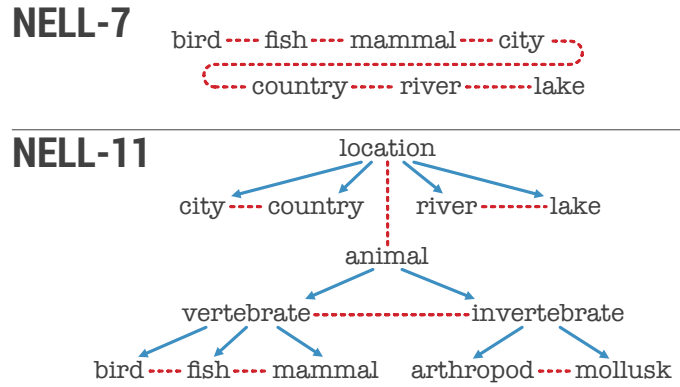


Figure 4.6: Illustration of the categories used in the two NELL datasets and the constraints between them. Each blue arrow represents a subsumption constraint, and each set of labels connected by a red dashed line represents a mutually exclusive set of labels. For example, animal subsumes vertebrate and bird, fish, and mammal are mutually exclusive.

was to be able to benefit from the information provided by such constraints. This is addressed in further experiments that are described in the following section.

4.5.3 Results with Logical Constraints

Given that our method can now handle logical constraints between domains (i.e., as opposed to DIRECT and BAYES), we also put together two larger datasets than previously. The goal in both datasets is to classify whether noun phrases (NPs) belong to certain categories (which correspond to domains in this case). The logical constraints used for each one are shown in Figure 4.6. For both of the datasets, we have a total of 553,940 NPs and 6 classifiers, which act as our function approximations and are described by Mitchell et al. (2015). Note that not all of the classifiers provide a response every input NP and thus we only evaluate against methods that are able to handle missing labels.

The results for these experiments are shown in Figure 4.7. It is clear that our method outperforms all existing methods, including the state-of-the-art, by a significant margin. Both the MADs of the error rates and the AUCs of the label prediction are significantly better. This is the desirable behavior for our method as it indicates that it is able to effectively use the additional information provided by the logical constraints. Note also that the largest execution time of our method among all datasets was about 10 minutes when using a 2013 15-inch MacBook Pro. The second best performing method, which is the hierarchically coupled Bayesian model of Chapter 3, required about 100 minutes. This highlights the scalability of this approach.

4.6 KEY TAKEAWAYS

In this chapter, we introduced a new approach to estimate accuracy from unlabeled data of several function approximations to several underlying true functions. In contrast to previous efforts, our approach is able to use the information provided by logical constraints that may exist between the outputs of the underlying functions. The

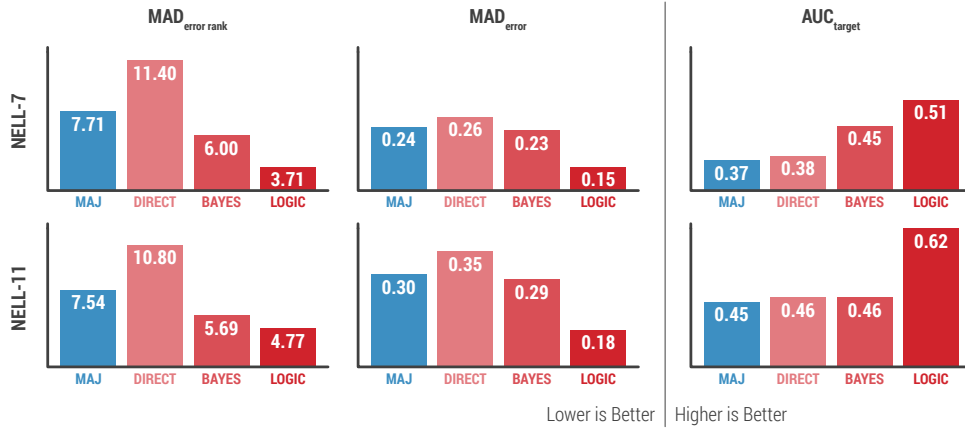


Figure 4.7: Results on the NELL datasets where logical constraints are provided. For the first two columns, lower numbers imply better performance, but the opposite is true for the last column.

proposed method also enables inference of the most likely outputs of the underlying true functions. Furthermore, it is capable of scaling to cases with many functions and millions of observations, due to the use of the efficient PSL framework for performing inference combined with a heuristic grounding algorithm and a stochastic variation of consensus ADMM. In order to explore the ability of the proposed approach to estimate error rates in realistic settings without domain-specific tuning, we performed an extensive experimental evaluation using four different datasets. Our methods were shown to outperform the current state-of-the-art, in both the tasks of estimating error rates and inferring the most likely single label, using only unlabeled data. The logic-based approach presented in this chapter also inspired us to design an active learning algorithm for cases where we have similar logical constraints between labels. This algorithm, along with some theoretical guarantees is presented in [Appendix B](#).

In the following chapter we present a method that addresses the last remaining limitation of the DIRECT approach and further improves performance by learning representations of the data instances that are being labeled and of the predictors which provide imperfect annotations. The new method enables end-to-end learning of machine learning models directly from imperfect labels (i.e., without the need for a separate label aggregation phase). This is also important for most large-scale machine learning systems since they are often trained using crowdsourced data.

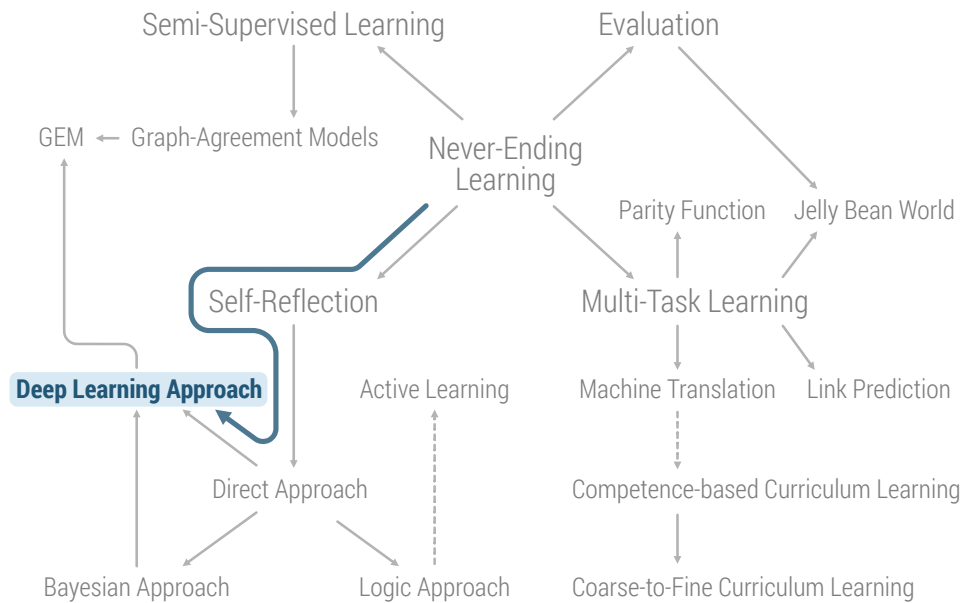


Figure 5.1: Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in [Section 1.4](#).

In [Chapter 2](#), we showed that learning collections of functions allows us to perform completely unsupervised evaluation, by proposing a direct approach to do this based on intuition about how we, as humans, would approach this problem. We then improved upon this method in [Chapters 3](#) and [4](#) by addressing some of its limitations, that we identified in [Section 2.6](#). In this chapter, we propose a method that addresses the last limitation: the inability of our previous approaches to learn and use powerful representations for the functions that are being learned. As we shall show, the proposed method is based on combining our ideas from [Chapter 3](#) with the power of neural networks in learning abstract representations over arbitrary data modalities, and it manages to successfully address this last limitation. We also show how it can be applied to the crowdsourcing setting, that we briefly discussed in [Chapter 1](#), and outperform current state-of-the-art methods for aggregating crowdsourced data.

5.1 INTRODUCTION

In the previous chapters we focused on never-ending learning and proposed multiple methods for estimating the accuracy of classifiers from unlabeled data. We have shown why this problem is central to machine learning and many other fields. Another motivating use case for the methods we proposed is that of crowdsourcing. The rising popularity and recent success of deep learning has resulted in machine learning systems that rely on large amounts of annotated training data (LeCun et al., 2015; Gulshan et al., 2016; Wu et al., 2016a; Esteva et al., 2017) and the most common and scalable way to collect such large amounts of training data is through crowdsourcing (Howe, 2006). Crowdsourcing works well in settings where annotation tasks do not require domain expertise—for example, in object detection and recognition tasks in natural images and videos (e.g., Deng et al., 2009; Kovashka et al., 2016). However, annotation in specialized domains such as medical pathology requires a certain level of competency and expertise which makes annotation expensive. Moreover, often times there is a high rate of disagreement even between experts, which results in increasingly subjective and inconsistent labels (Elmore et al., 2015; Hutson et al., 2019).

A typical approach to dealing with subjectivity is to treat each annotation as simply noisy, collect multiple redundant labels per example (e.g., from different annotators), and then aggregate them using majority voting or other more advanced techniques (e.g., the methods we proposed in the previous chapters or other related work such that of Dawid and Skene, 1979; Carpenter, 2008; Liu et al., 2012; Platanios, 2012; Zhou et al., 2015; Zhou and He, 2016) to obtain a single “ground truth” label. At the expense of redundancy, this results in better data quality and more accurate estimates of the ground truth. More recently, emerging systems for *data programming* and *weak supervision* also rely internally on label aggregation techniques similar to methods used for solving the crowdsourcing problem. Snorkel (Ratner et al., 2017; Bach et al., 2019) is one popular such system that was designed for efficient and low-cost creation of large-scale labeled datasets using programmatically generated, so-called *weak labels*. However, as we show in our empirical evaluation in Section 5.3, none of these systems solve label aggregation effectively in the presence of high subjectivity. We argue that to become more effective, these methods need to make use of meta-data and other types of information that may be available about the data instances and the annotators labeling them. This is also the main limitation of the methods we proposed in the previous chapters that we aim to address.

To this end, we propose a novel approach that allows us to train accurate predictive models of the ground truth *directly on the non-aggregated imperfectly labeled data*.¹ Our method merges the two steps of: (i) aggregating subjective, weak, or noisy annotations, and (ii) training machine learning models. At training time, along with learning a model that predicts the ground truth, we also learn models of the difficulty of each example and the competence of each annotator in a generalizable manner (i.e., these models can make predictions for previously unseen examples and annotators). This can also enable us to more effectively assign annotators to examples, thus driving the

¹ The work we present in this chapter has been published in (Platanios et al., 2020a).

cost of crowdsourcing down while improving the quality of the resulting datasets. The proposed approach can be effectively used for training on crowdsourced data as well as on weakly labeled data, and can also be used within frameworks such as Snorkel (Ratner et al., 2017; Bach et al., 2019) to significantly improve their performance. More specifically, it can:

1. Learn truth estimators: Learn functions that represent the underlying ground truth, while imposing almost no constraints (as opposed to prior work). In fact, we are able to leverage the capacity of deep neural networks along with the interpretability provided by Bayesian models, in order to obtain highly expressive estimators of the underlying truth.
2. Learn quality estimators: Learn functions that estimate the quality of each annotator. When annotators can be described by some features (e.g., age, gender, location, etc. of an Amazon Mechanical Turk annotator, instead of just an integer identifier), our quality estimators are able to generalize to new, previously unseen, predictors. In the case where the predictors can be described using features (instead of just an indicator—e.g., the age and location of an Amazon Mechanical Turk annotator), we are even able to generalize our quality estimators to new, previously unseen, predictors. Previous work only considered estimating accuracies of a fixed set of predictors, without being able to leverage any information we might have about them. Furthermore, in contrast to previous work, we are also able to predict the per-instance-predictor pair qualities (i.e., the probability that a specific observation from a predictor, is corrupted), by learning dependencies between the instances and the predictors (e.g., our method can determine whether a human annotator is an expert for a subset of queries, instead of just estimating his/her overall accuracy). Furthermore, in contrast to previous work, we are also able to predict the per-instance predictor competencies (i.e., our method can determine whether a human annotator is

INTERESTING CONNECTIONS

The problems of ensemble learning, aggregating and denoising crowdsourced data, and estimating accuracy from unlabeled data, all share the same underlying core problem: *learning from imperfect labels*. More specifically, there is a common setting among all these problems where: (i) there exists an underlying ground truth, (ii) we only get to observe multiple, possibly overlapping, noisy views of that truth, and (iii) we want to be able to estimate that truth. The noisy views can have arbitrary form, such as: (i) human annotators in a crowdsourcing platform, that may make mistakes (e.g., Zhou et al., 2015), or (ii) classifiers that have already been trained (e.g., Platanios et al., 2014, 2016, 2017b) like the settings we considered in the previous chapters. To give a concrete example, consider the problem of medical pathology diagnostics, where learning-based models are becoming increasingly popular (e.g., Gulshan et al., 2016). Training models by imitating expert decisions is not as straightforward in such a scenario: the true diagnosis is unknown a priori, while the diagnostic concordance between experts is often far from perfect (Elmore et al., 2015). If we assume that the expert decisions are the ground truth, the model may overfit to their mistakes. Therefore, this practical setup requires a principled learning framework that takes into account potential discrepancies or disagreements in the observations.

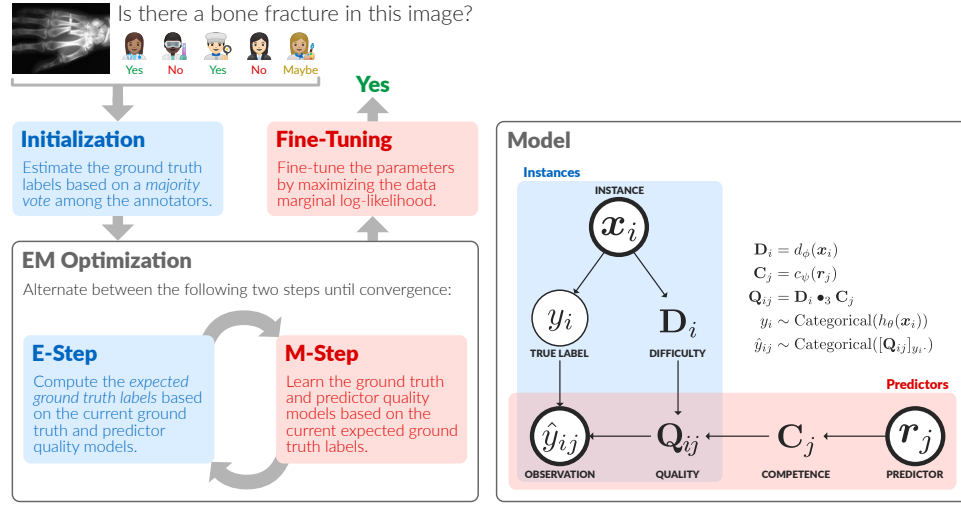


Figure 5.2: Overview of the proposed inference algorithm and probabilistic model.

an expert for a subset of queries, instead of just estimating his/her overall accuracy), which is done by learning dependencies between the instances and the annotators. Finally, our approach is able to distinguish between different types of errors by estimating the full confusion matrix for each pair of instances and predictors.

3. Be easily extended: The truth and quality estimators can take arbitrary functional forms and fully leverage the expressivity of deep neural networks.

Both human annotators and machine learning classifiers may sometimes be unable to make predictions about certain aspects of the ground truth (e.g., human annotators may be unsure about what the correct answer to a question is). The proposed method is formulated in a way that allows it to be extended such that it can also *learn decision function estimators* for the annotators (i.e., estimators that predict whether an annotator will be able to provide a prediction for a given data instance). These estimators can have significant implications for data annotation systems where the cost of querying annotators is high (e.g., when the annotators are highly qualified, such as doctors or other kinds of domain experts). This is because it allows for better matching annotators to instances, thus reducing the required amount of annotation redundancy. An overview of the proposed approach and model is shown in Figure 5.2.

The proposed method generalizes the approaches of Zhou et al. (2015) and Khetan et al. (2018), as well as the approaches we presented in the previous three chapters. Similar to Chapter 3, we define a generative process for our observations. However, our model is able to handle categorical labels, as opposed to just binary. Similar to Zhou et al. (2015), we define the confusion matrix for each instance-predictor pair as a function of instance difficulty and predictor competence. However, we explicitly learn the difficulty and competence as functions, which allows us to generalize to previously unseen instances and annotators. Interestingly, the inference algorithm we propose for our generative probabilistic model has a similar form to that of Zhou et al. (2015), except for the explicit learning of the ground truth, difficulty, and competence

The idea of modeling instance difficulties and annotator competencies has been studied before by Carpenter (2008) and Platanios (2012), among others.

functions. Thus, we also show that the algorithm of Zhou et al. (2015) can be re-derived as an *expectation-maximization* (EM; Dempster et al., 1977) inference algorithm for a generative model, simplifying the argument of the original paper. Finally, similar to Khetan et al. (2018), we use a parametric function to model the ground truth, and go a step further by proposing to use parametric functions to model the instance difficulties and predictor competences. Thus, our approach enables estimation of which annotators are likely to perform better on which instances, potentially enabling more effective allocation of annotators and thus annotation cost reductions.

In contrast to prior work, our method also allows for *end-to-end learning*. Prior methods do not allow for this as they implicitly separate ground truth inference (i.e., label aggregation) from model training. More specifically, previously one would have to train a machine learning model in two stages: (i) infer the ground truth labels from the provided annotations, and (ii) train a machine learning model on the inferred labels. Our approach merges these two stages and allows us to train machine learning models directly on the imperfect annotations. In Section 5.3.3, we conduct an ablation study that showcases the performance gains resulting from end-to-end learning.

5.2 COMBINING BAYESIAN MODELS AND DEEP LEARNING

We denote the observed data by $\mathcal{D} = \{\mathbf{x}_i, \hat{\mathbf{y}}_i\}_{i=1}^N$, where $\hat{\mathbf{y}}_i = \{\mathcal{M}_i, \{\hat{\mathbf{y}}_{ij}\}_{j \in \mathcal{M}_i}\}$. $\mathcal{M}_i$ is the set of predictors that made predictions for instance $\mathbf{x}_i$, and $\hat{\mathbf{y}}_{ij}$ is the output of predictor $\hat{f}_j$ for instance $\mathbf{x}_i$. Our goal is to learn functions that represent the underlying ground truth and predictor qualities, given our observations $\mathcal{D}$. Note that, predictor qualities generalize the notion of error rates from Chapter 3.

GROUND TRUTH. We define the ground truth as a function $h_\theta(\mathbf{x}_i)$ that is parameterized by θ and that approximates the true distribution of a label given $\mathbf{x}_i$. In our setting, $h_\theta(\mathbf{x}_i) \in \mathbb{R}_{\geq 0}^C$ and $\sum_j [h_\theta(\mathbf{x}_i)]_j = 1$, where C is number of values the label can take (i.e., assuming categorical labels). More specifically, $[h_\theta(\mathbf{x}_i)]_k \triangleq p(y_i = k | \mathbf{x}_i)$, where we use square brackets and subscripts to denote indexing of vectors, matrices, and tensors. For example, h_θ could be a deep neural network that would normally be trained in isolation using, for example, the cross-entropy loss function. In our method the network is trained using the EM algorithm, as described in the following section.

PREDICTOR QUALITIES. We define the predictor qualities as confusion matrices $\mathbf{Q}_{ij} \in \mathbb{R}_{\geq 0}^{C \times C}$, for each instance $\mathbf{x}_i$ and predictor $\hat{f}_j$, where $\sum_l [\mathbf{Q}_{ij}]_{kl} = 1$, for all $k \in \{1, \dots, C\}$. $[\mathbf{Q}_{ij}]_{kl}$ represents the probability that predictor $\hat{f}_j$ outputs label l given that the true label of instance $\mathbf{x}_i$ is k . We define these confusion matrices in a way that generalizes the successful approach of Zhou et al. (2015):

$$\mathbf{Q}_{ij} = \mathbf{D}_i \bullet_i \mathbf{C}_j, \quad (5.1)$$

where $\bullet_i$ represents an inner product along the i^{th} dimension of the two tensors, and:

- $\mathbf{D}_i = d_\phi(\mathbf{x}_i)$ represents the *difficulty* tensor for instance $\mathbf{x}_i$, where d is a function parameterized by ϕ , $\mathbf{D}_i \in \mathbb{R}^{C \times C \times L}$, and L is a latent dimension (it

We also perform a normalization step such that all elements of $\mathbf{Q}_{ij}$ are non-negative and such that each row sums to 1 (thus making each row a valid probability distribution).

is a hyperparameter of our model). $[\mathbf{D}_i]_{kl-}$ is an L -dimensional embedding representing the likelihood of confusing $\mathbf{x}_i$ as having label l instead of k , when k is its true label.

- $\mathbf{C}_j = c_\psi(\mathbf{r}_j)$ represents the *competence* tensor for predictor $\hat{f}_j$, where c is a function parameterized by ψ , $\mathbf{r}_j$ is some representation of $\hat{f}_j$ (e.g., could be a one-hot encoding of the predictor, in the simplest case), and $\mathbf{C}_j \in \mathbb{R}^{C \times C \times L}$. $[\mathbf{C}_j]_{kl-}$ is an L -dimensional embedding representing the likelihood that predictor $\hat{f}_j$ confuses label k for l , when k is the true label.

Using $L > 1$ allows the instance difficulties and predictor competences to encode more information. An intuitive way to think about this is that we are embedding difficulties and competencies in a common latent space, which can be thought of as jointly clustering them. This is very similar to how matrix factorization methods are used for collaborative filtering in recommender systems.

Our goal is to learn the functions h_θ , d_ϕ , and c_ψ , given the observations $\mathcal{D}$. In order to do this, we propose the following generative process for our observations. For $i = 1, \dots, N$, we first sample the true label for $\mathbf{x}_i$, $y_i \sim \text{Categorical}(h_\theta(\mathbf{x}_i))$. Then, for $j \in \mathcal{M}_i$, we sample the predictor output $\hat{y}_{ij} \sim \text{Categorical}([\mathbf{Q}_{ij}]_{y_i-})$, where $[\mathbf{Q}_{ij}]_{y_i-}$ represents the y_i^{th} row of $\mathbf{Q}_{ij}$. In the next section, we propose an algorithm for learning the parameters θ , ϕ , and ψ .

A potentially interesting extension would be to also learn the predictors' decision making functions, so that we can explicitly model missing observations.

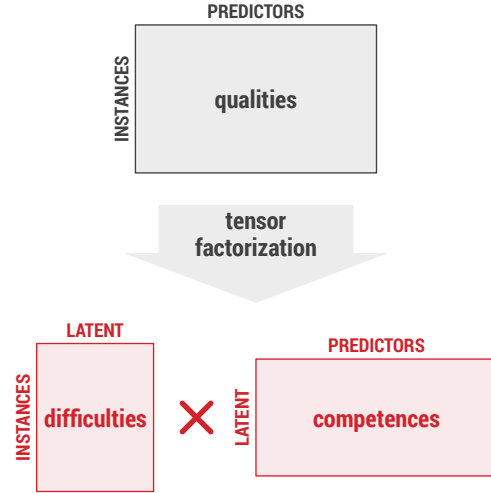


Figure 5.3: Illustration of our tensor factorization approach, modeling the interaction of instance difficulties with predictor competences.

5.2.1 Learning

A widespread approach for performing learning with probabilistic generative models, is to maximize the likelihood of the observed data with respect to the model parameters. Let $\mathbf{y} = \{y_i\}_{i=1}^N$. The likelihood of a single observation, $\hat{y}_{ij}$, can be derived as follows:

$$p(\mathcal{D}, \mathbf{y}) = \prod_{i=1}^N p(y_i) \prod_{j \in \mathcal{M}_i} p(\hat{y}_{ij} | y_i), \quad (5.2)$$

where $p(\hat{y}_{ij} | y_i)$ depends on $\mathbf{Q}_{ij}$. There are two main approaches in which we can maximize the likelihood function of Equation 5.2: (i) marginalize out the y_i latent variables and then maximize the resulting function with respect to θ , ϕ , and ψ , or (ii) use the EM algorithm which was originally proposed by Dempster et al. (1977). It has previously been observed that the EM algorithm can perform much better than approach (i) for mixture models (Bishop, 2006). This is because the latter tends to get

The marginalization approach also consistently underperformed EM in our experiments.

stuck in bad local optima. Since our model resembles a Bernoulli mixture model with the latent assignments being defined by the y_i s, we decided to use the EM algorithm. We derive the EM algorithm steps for our model, as follows:

E-STEP. We need to compute the expectation of y_i given $\mathcal{D}$ and $\mathbf{y}_{\setminus i}$ (which denotes all of $\mathbf{y}$ except for y_i), for all $i = 1, \dots, N$, and we know that:

$$p(y_i | \mathcal{D}, \mathbf{y}_{\setminus i}) \propto p(\mathcal{D}, \mathbf{y}) = \prod_{s=1}^N p(y_s) \prod_{j \in \mathcal{M}_s} p(\hat{y}_{sj} | y_s). \quad (5.3)$$

Therefore, by removing all terms that do not depend on y_i and normalizing, we obtain the following expectation, which we compute during this step, while keeping θ , ϕ , and ψ fixed:

$$\mathbb{E}_{\mathbf{y}|\mathcal{D}} \{ \mathbb{1}_{\{y_i=k\}} \} = \frac{\lambda_i^k}{\sum_{l=1}^C \lambda_i^l}, \quad (5.4)$$

where $\mathbb{1}_{\{\cdot\}}$ evaluates to one if its subscript statement is true and to zero otherwise, and:

$$\lambda_i^k = [h_\theta(\mathbf{x}_i)]_k \prod_{j \in \mathcal{M}_i} [\mathbf{Q}_{ij}]_{k\hat{y}_{ij}}. \quad (5.5)$$

Note that $\mathbf{Q}_{ij}$ is a function of both ϕ and ψ . For brevity, in what follows we use the following notation: $\tilde{y}_i^k = \mathbb{E}_{\mathbf{y}|\mathcal{D}} \{ \mathbb{1}_{\{y_i=k\}} \}$.

M-STEP. We maximize the following log-likelihood function with respect to θ , ϕ , and ψ , while using the values of $\tilde{y}_i^k$ computed in the last E-step:

$$\mathcal{L} = \prod_{i=1}^N p(\tilde{y}_i) \prod_{j \in \mathcal{M}_i} p(\hat{y}_{ij} | \tilde{y}_i) \Rightarrow \quad (5.6)$$

$$\log \mathcal{L} = \sum_{i=1}^N \log p(y_i = \tilde{y}_i) + \sum_{i=1}^N \sum_{j \in \mathcal{M}_i} \log p(\hat{y}_{ij} | y_i = \tilde{y}_i) \Rightarrow \quad (5.7)$$

$$\log \mathcal{L} = \sum_{i=1}^N \sum_{k=1}^C \tilde{y}_i^k \log[h_\theta(\mathbf{x}_i)]_k + \sum_{i=1}^N \sum_{j \in \mathcal{M}_i} \sum_{k=1}^C \tilde{y}_i^k \log[\mathbf{Q}_{ij}]_{k\hat{y}_{ij}}. \quad (5.8)$$

The training procedure for learning the parameters θ , ϕ , and ψ consists of iterating over the E-step and the M-step shown above, until convergence, where convergence can be measured by computing the change in the parameter values across learning iterations. It is important to note that EM finds local optima of the likelihood function, and so the starting point may play an important role. Also, as we explained in [Chapter 3](#), there exists an inherent symmetry in our model that can be problematic, if left unhandled. The likelihood of the observed data is the same if we flip the true underlying labels and the predictor qualities (i.e., set $y_i^{\text{flipped}} = 1 - y_i$ and $\mathbf{Q}_{ij}^{\text{flipped}} = 1 - \mathbf{Q}_{ij}$). We would like to somehow encode the prior assumption that most of the predictors

are correct most of the time. One way to do this is by choosing the starting point of the EM algorithm carefully.

INITIALIZATION. In the E-step shown in Equation 5.4, we compute the expected values of the true underlying labels, y_i . We can encode the symmetry-breaking assumption by replacing the first E-step with a majority vote among the predictors:

$$\tilde{\mathbb{E}}_{\mathbf{y}|\mathcal{D}} \{ \mathbb{1}_{\{y_i=k\}} \} = \frac{\sum_{j \in \mathcal{M}_i} \mathbb{1}_{\{\hat{y}_{ij}=k\}}}{|\mathcal{M}_i|}, \quad (5.9)$$

where $|\mathcal{M}_i|$ denotes the size of the set $\mathcal{M}_i$. We initialize the EM algorithm by replacing the first E-step with this majority vote approximation. As we show in our experiments, this helps us avoid the aforementioned symmetry, and thus we refer to this initialization scheme as *symmetry-breaking initialization*. In the case where the predictors provide us with $p(\hat{y}_{ij} = k)$ instead of a single categorical value, we can still use this initialization scheme by replacing $\mathbb{1}_{\{\hat{y}_{ij}=k\}}$ with $p(\hat{y}_{ij} = k)$, in Equation 5.9.

REGULARIZATION. Similar to Zhou et al. (2015) we propose to regularize our model by maximizing the following regularized objective function:

$$\log \mathcal{L} - \alpha \frac{1}{2} \sum_j \sum_{k=1}^C \sum_{l=1}^C \sum_{m=1}^L [C_j]_{klm}^2 - \beta \frac{1}{2} \sum_i \sum_{k=1}^C \sum_{l=1}^C \sum_{m=1}^L [D_i]_{klm}^2, \quad (5.10)$$

where α and β are defined as follows:

$$\alpha = \gamma C^2, \quad (5.11)$$

$$\beta = \alpha \frac{\#\{\text{labels per predictor}\}}{\#\{\text{labels per instance}\}}, \quad (5.12)$$

and γ is a hyper-parameter, set to 0.25 in our experiments (same as Zhou et al. (2015)).

MARGINAL LIKELIHOOD FINE-TUNING. In our experiments we found that maximizing the marginal likelihood function after EM converges tends to improve performance. We refer to this step as *marginal likelihood fine-tuning*. More specifically, after the values of the parameters θ , ϕ , and ψ , converge to fixed values across multiple EM steps, we solve the following maximization problem using these fixed values as the initial point:

$$\max_{\theta, \phi, \psi} \sum_{\mathbf{y}} p(\mathcal{D}, \mathbf{y}) \Leftrightarrow \max_{\theta, \phi, \psi} \sum_{i=1}^N \sum_{j \in \mathcal{M}_i} \log \sum_{k=1}^C [h_{\theta}(\mathbf{x}_i)]_k [\mathbf{Q}_{ij}]_{k\hat{y}_{ij}}. \quad (5.13)$$

5.2.2 Instance and Predictor Representations

A major advantage of the proposed approach over prior work is that we learn models of the ground truth and the predictor qualities as functions of some representations (i.e.,

representations of the data instances, $\mathbf{x}_i$, and of the annotators, $\mathbf{r}_j$). It is thus important to define these representations. For many problems, the representations of the data instances can be defined in the same manner as was previously done when performing supervised learning (e.g., we can directly use raw pixel values to represent images). However, predictor representations are introduced here for the first time. A simple approach would be to use a one-hot encoding of the predictors. However, this would not allow for any amount of information sharing across predictor (e.g., what if two predictors are very similar). We already know from our previous work in [Chapter 3](#) that modeling the dependencies between predictors can be crucial to performance. One way to allow for this is to learn vector embedding representations for the predictors, which would be implicitly equivalent to clustering them. Ideally, one would want to use any available information about these predictors (e.g., Amazon Mechanical Turk annotators could be described by their age, location, etc.). Unfortunately, we could not find any public datasets that provide such information about the predictors/annotators, and hence in our empirical study we use embedding representations.

5.2.3 Discussion

The approach we have proposed in this chapter can be thought of as introducing a new loss function for training the model h_θ using multiple imperfect labels per training instance, each coming from a different source. This new loss function introduces latent variables that represent the ground truth labels, as well as a couple of auxiliary models that are learned, and which represent the instance difficulties and predictor competences. We also proposed an EM-based algorithm to minimize this new loss function as well as an initialization scheme and a regularization scheme to help prevent overfitting. Perhaps most interestingly, a key difference between this approach and previous work is that we are able to explicitly learn functions that output the likelihood that a predictor will label a specific instance correctly. This enables using our method to perform crowdsourcing more actively by assigning annotators to instances they are more likely to label correctly, thus helping reduce redundancy and drive costs down.

5.2.4 Extending to Multi-Label Settings

Our method can be easily extended to handle settings where we have multiple categorical labels that may be assigned to each instance. In this case, the model per label is defined in the same way as previously, except that now the functions h_θ , d_ϕ , and c_ψ also take as input a representation for the label (e.g., a label embedding). This allows us to share information across labels and can be thought of as a generalization of our BAYES method from [Chapter 3](#), where information is shared by clustering the labels. Furthermore, it allows us to use the proposed method in extreme classification settings (e.g., Prabhu and Varma, 2014) or settings where the number of labels is not fixed and known *a priori* and can keep increasing (e.g., face recognition; Weinberger and Saul, 2009; Liu et al., 2016). This is made possible by learning label representations and then letting the difficulty and competence functions also receive as input pairs of labels and return a vector instead of a three-dimensional tensor. In the next section,

we show how learning label representations can significantly enhance the robustness and performance of our approach.

5.3 EXPERIMENTS

Unfortunately, the datasets we used for our experiments in Chapters 2, 3, and 4 are quite old and it is impossible to obtain features for the data instances. This is because we only have access to integer identifiers of the data instances but no information about which instances the numbers refer to (e.g., which noun phrases). For this reason, we evaluate the newly proposed approach on multiple datasets from the crowdsourcing domain, all of which have ground truth labels, (multiple) subjective annotations for each example, as well as information on who provided each annotation (i.e., an annotator integer identifier):

1. Blue Birds (BB) (Welinder et al., 2010): Classify whether a bird in a photo is an “Indigo Bunting” or a “Blue Grosbeak.”
2. Word Similarity (WS) (Snow et al., 2008): Classify whether a pair of words is similar or dissimilar.
3. RTE (Snow et al., 2008): Classify whether a sentence entails another sentence.
4. Medical Causes (MC) (Dumitrache et al., 2018): Classify whether a medical term *causes* another medical term, given a sentence that contains the two terms (e.g., “pancreatic adenocarcinoma causes weight loss”).
5. Medical Treats (MT) (Dumitrache et al., 2018): Classify whether a medical term *treats* another medical term, given a sentence that contains the two terms (e.g., “aspirin treats pain”).

The last two datasets are in fact part of a single dataset on medical relations, and we are thus able to perform experiments using both the single task formulation of our algorithm and the multi-task formulation. As we discuss in the end of this section, this allows us to show how our approach can be used to share information across labels and improve the quality of the learned models. Statistics for these datasets are provided in Table 5.1. Note that, no associated features for the annotator identifiers were provided and thus we are unable to evaluate the usefulness of annotator features (we instead learn embeddings for the annotators). Unfortunately we were unable to obtain any crowdsourcing datasets with associated annotator meta-data. This is probably due to the fact that no prior method is able to make use of such information. However, we do make use of instance features for all the datasets. In cases where such features are not readily available, we compute them manually using pre-trained machine learning models. We are making all the features and annotator identification information publicly available in a standardized format. More details are provided in our code and data repository which is available at <https://github.com/eaplatanios/noisy-labels>.

5.3.1 Experimental Setup

We perform experiments using the following two variants of our approach:

Dataset	#Instances	#Predictors	Average Redundancy	Average Accuracy (%)
Blue Birds (Welinder et al., 2010)	108	39	39	63.56
Word Similarity (Snow et al., 2008)	30	10	10	81.33
RTE (Snow et al., 2008)	800	164	10	84.13
Medical Causes (Dumitrache et al., 2018)	3,984	408	15	32.40
Medical Treats (Dumitrache et al., 2018)	3,984	408	15	38.88

Table 5.1: Statistics for the datasets we use in our experiments. “#Predictors” refers to the number of predictors in the dataset, “Average Redundancy” refers to the average number of predictions provided for each instance, “Average Accuracy” refers to the average predictor accuracy, and “Random Accuracy” refers to the accuracy obtained of a completely random predictor.

- LIA: A version of our method which uses instance and predictor features specific to each dataset. When features are not available for the instances and/or the predictors, we learn embeddings of size 16 which are initialized randomly and optimized along with the other model parameters during the M-step (see Section 5.2.1).
- LIA-ML: A multi-label variant of the aforementioned method. This method is only used with the medical relations datasets. In this case, we consider all 14 medical relations included in the dataset jointly and only evaluate on the two for which the ground truth labels are provided (i.e., “causes” and “treats”). We use this method variant in order to show how our approach can effectively share information across labels.

In both instances of LIA, h_θ and d_ϕ are multi-layer perceptrons (MLPs) with 4 layers of 16 hidden units each, with the only exception being the medical relations dataset where we use 32 units for each layer. c_ψ is always modeled as a linear function. Note that for both the embedding sizes and the MLP sizes, we did not perform an extensive search to choose these values; we rather performed a small grid search and selected the number that resulted in the highest validation data likelihood. We compare against the following baselines for ground truth estimation:

- MAJ: Simple majority voting. We use *soft* majority voting whenever possible. That is, we use soft labels (probabilities or confidence scores) whenever the predictors provide them, instead of always thresholding them to obtain discrete labels.
- MMCE: Regularized minimax conditional entropy by Zhou et al. (2015), which has been shown to outperform alternatives. We consider it the current state-of-the-art for crowdsourcing.
- Snorkel: A method originally designed for aggregating annotations of programmatic weak predictors proposed by Ratner et al. (2017), which is part of a popular software package that also allows for subsequent training of machine learning models on the aggregated data.
- MeTaL: Successor to Snorkel, proposed by Ratner et al. (2018). For both this method and for Snorkel we use the original implementation provided by the authors, which can be found at <https://github.com/HazyResearch/snorkel>.

Aside from these baselines, we also perform experiments using the following custom methods that we designed for the purpose of performing an ablation study (the study is presented in [Section 5.3.3](#)):

- LIA-E: In order to evaluate the usefulness of instance features, we learn embeddings of size 16 for the instances instead of using their features.
- MAJ*-E: A two-step method that resembles how machine learning models are currently being trained when using crowdsourced data. First, we estimate ground truth using MAJ. Next, we train the h_θ model used in LIA-E directly on the aggregated labels.
- MAJ*: Same as MAJ*-E, except that we use the model h_θ of LIA (i.e., one that makes use of instance features instead of learning instance embeddings).
- MMCE*-E: Same as MAJ*-E, but with MMCE used for label aggregation.
- MMCE*: Same as MAJ*, but with MMCE used for label aggregation.

During each M-step we use the AMSGrad optimizer (Reddi et al., 2018) to maximize the log-likelihood function with the learning rate set to 0.001. We perform 1,000 optimization iterations using a batch size of 1,024. Overall, we perform 10 EM iterations (all models converged within that limit) with warm starting (i.e., the model parameters are always initialized to the values obtained during the previous M-step). When using LIA with image instances, we use as image features the activations of the last layer of a pre-trained ResNet-101 Convolutional Neural Network (CNN). Similarly, for all text instances we use as text features the representations provided by a pre-trained BERT model (Devlin et al., 2019).

We evaluate all methods by computing the *accuracy* of the predicted instance labels. This is a common metric for evaluating crowdsourcing methods and it also implicitly measures the quality of the confusion matrices predicted by our model. This is because these confusion matrices heavily influence the supervision provided to the ground truth model h_θ , while training. Furthermore, instead of just computing accuracy over the full datasets, we also measure how performance varies as a function of *redundancy*—the maximum number of annotations provided per instance. In order to limit redundancy for existing datasets we randomly sample subsets of the provided annotations. Performing well in low redundancy settings is important because it can result in significantly reduced crowdsourcing costs.

5.3.2 Results

Our results are presented in [Table 5.2](#). LIA methods consistently outperform alternative approaches. In certain cases (e.g., in Blue Birds) we are able to boost accuracy over the best alternative method by 14%, thus establishing a new state-of-the-art for this dataset. In the multi-task setting, where we train the LIA-ML model to jointly infer the ground truth labels for both Medical Causes (MC) and Medical Treats (MT) while sharing the representations of instance difficulties and annotator competencies. We observe that multi-task training boosts performance with more than 8% absolute gain (or over 20% relative gain) over its single task counterpart, outperforming the baselines with over 25% relative gain. Finally, our approach can obtain the performance

			Accuracy (%)										
			MAJ	MAJ*-E	MAJ*	MMCE	MMCE*-E	MMCE*	Snorkel	MeTaL	LIA-E	LIA	LIA-ML
Dataset & Redundancy	BB	2	63.9	64.4	65.1	66.9	63.2	63.2	63.8	63.0	65.1	71.5	—
		5	71.1	70.9	70.9	73.9	74.2	73.2	72.4	71.5	73.0	76.2	—
		10	74.4	75.9	75.4	76.5	76.9	77.0	76.0	83.0	78.1	83.1	—
		20	76.4	76.2	76.1	78.5	77.7	77.7	76.0	87.0	77.2	90.0	—
		39	75.9	78.4	78.4	79.6	78.8	78.8	76.0	89.0	78.9	93.0	—
	WS	2	82.8	87.2	87.7	80.1	79.3	80.0	76.2	76.0	87.7	88.7	—
		5	87.1	91.4	91.3	87.0	84.4	84.0	76.3	85.1	87.7	92.7	—
		10	88.6	93.3	93.3	90.2	80.0	80.0	76.3	93.1	87.7	96.3	—
	RTE	2	72.8	72.8	74.5	75.3	77.3	73.2	65.2	61.0	76.7	78.0	—
		5	84.8	84.0	84.8	88.5	88.9	85.3	79.1	72.4	84.3	89.1	—
		10	90.0	90.4	89.9	92.7	92.7	87.1	90.0	78.0	91.6	93.1	—
	MC	2	26.8	24.2	26.5	29.1	29.3	22.0	27.1	25.3	25.0	29.5	30.1
		5	24.1	23.6	24.2	24.5	24.6	21.3	24.0	21.0	24.0	30.9	36.4
		10	24.1	23.6	24.1	24.4	24.6	20.1	24.0	20.0	23.6	30.5	34.1
	MT	2	33.8	35.7	34.2	35.3	38.6	34.2	33.3	22.1	34.0	38.6	40.8
5		34.2	34.1	33.6	36.8	37.0	35.0	34.0	21.0	33.0	38.3	46.1	
10		34.2	34.3	35.2	38.5	38.3	36.3	35.0	03.1	33.8	42.1	45.4	

Table 5.2: Accuracy across varying levels of redundancy, for all datasets we used in our experiments. For each experiment, we report mean accuracy over 50 runs with different random initializations. We also compute standard error but it is generally too low and so it is not included in the table. The best results are underlined and shown in red color. The methods marked with a “*” are used for the ablation study of Section 5.3.3.

of the best alternative method using up to 4 times less redundancy, which can have significant implications for the cost of crowdsourcing, especially when the annotations require domain expertise (e.g., in healthcare). We note that Snorkel and MeTaL tend to perform well overall, but sometimes fail entirely (often performing on par with or worse than majority voting, which has also been observed by others, e.g., <https://github.com/HazyResearch/snorkel/issues/1073>). MeTaL also suffers from calibration issues, as it often achieves very low accuracy while having reasonable mean average precision. Data programming systems could thus benefit significantly by integrating our method in their pipeline, tying together the label aggregation and model training phases.

5.3.3 Ablation Study

Our main contributions are: (i) end-to-end learning by fusing the label aggregation and model training phases, and (ii) allowing for instance and annotator features to inform label aggregation.² In this section, we show how each one of these contributions is important in its own right by performing experiments where we introduce each one on their own, while keeping everything else constant.

² As mentioned in the beginning of this section, we were unable to obtain any crowdsourcing datasets with associated annotator meta-data and thus in our experiments we only use instance features.

END-TO-END LEARNING. The best way to test the effectiveness of end-to-end learning is to compare end-to-end approaches with two-stage approaches where: (i) we first aggregate labels, and (ii) we then train machine learning models using the aggregated labels. To this end, we introduced the baseline methods marked with a “*” in Table 5.2. The results indicate that the two-stage approach underperforms the base label aggregation method for both MAJ and MMCE. This is most likely due to the fact that in both these cases the model being learned cannot inform the label aggregation stage. In contrast, LIA is able to outperform all two-stage approaches because it allows for exactly this. Note that MMCE models in instance difficulty and annotator competence similar to LIA, with the exception that it does not use instance features and it does not allow for end-to-end learning of ground truth predictors. Also note that LIA-E does not use instance features, but is still able to outperform MAJ*-E and MMCE*-E in many cases, indicating that end-to-end learning is effective and accounts for at least part of the performance gains achieved by LIA. Finally, we also observe that MMCE* completely fails in some cases (e.g., in the Medical Causes (MC) and Medical Treats (MT) datasets). We do not have a good understanding as to why this happens, but the fact that LIA does not suffer in these cases indicates that end-to-end learning is more robust to low quality annotations.

INSTANCE FEATURES. The methods which use instance features are LIA, MAJ*, and MMCE*. To test for the usefulness of these features, we provide variants of these methods (labeled LIA-E, MAJ*-E, and MMCE*-E, respectively, in Table 5.2) that use indicator features instead and learn instance embeddings. We observe that for MAJ and MMCE the results are inconclusive (the feature-based methods outperform the alternative in about half of the experiments). However, in the context of end-to-end learning, we observe that LIA consistently outperforms LIA-E by a significant margin. This indicates that instance features are indeed useful—especially so in the context of end-to-end learning where they can inform the label aggregation phase.

It is important to also mention that results pertinent to this ablation study for the Word Similarity (WS) dataset were a bit unstable with many models effectively failing to learn anything meaningful (specifically MMCE*-E, MMCE*-M, and LIA-E). Our best explanation for this is that the Word Similarity (WS) dataset is very small with only 30 instances and is thus highly prone to overfitting.

Finally, in order to evaluate the effect of the marginal likelihood fine-tuning approach presented in Section 5.2.1, we also perform experiments using LIA without this fine-tuning phase. The results are shown in Table 5.3 and it is clear that marginal likelihood fine-tuning results in better performance. Also, in order to sanity check that LIA is able to predict the qualities of the predictors accurately, we also perform a synthetic

Accuracy (%)			
Dataset & Redundancy	BB	2	71.9 $\pm$ 0.9
		5	74.9 $\pm$ 0.4
		10	76.9 $\pm$ 0.5
		20	77.5 $\pm$ 0.4
		39	79.1 $\pm$ 0.2
	WS	2	88.7 $\pm$ 0.8
		5	91.0 $\pm$ 0.6
		10	93.3 $\pm$ 0.0
	RTE	2	63.8 $\pm$ 0.1
		5	67.5 $\pm$ 0.1
		10	69.6 $\pm$ 0.1
	MC	2	24.0 $\pm$ 0.2
		5	23.9 $\pm$ 0.2
		10	23.0 $\pm$ 0.2
	MT	2	38.1 $\pm$ 0.3
		5	38.5 $\pm$ 0.2
		10	39.2 $\pm$ 0.1

Table 5.3: Results for LIA without marginal likelihood fine-tuning.

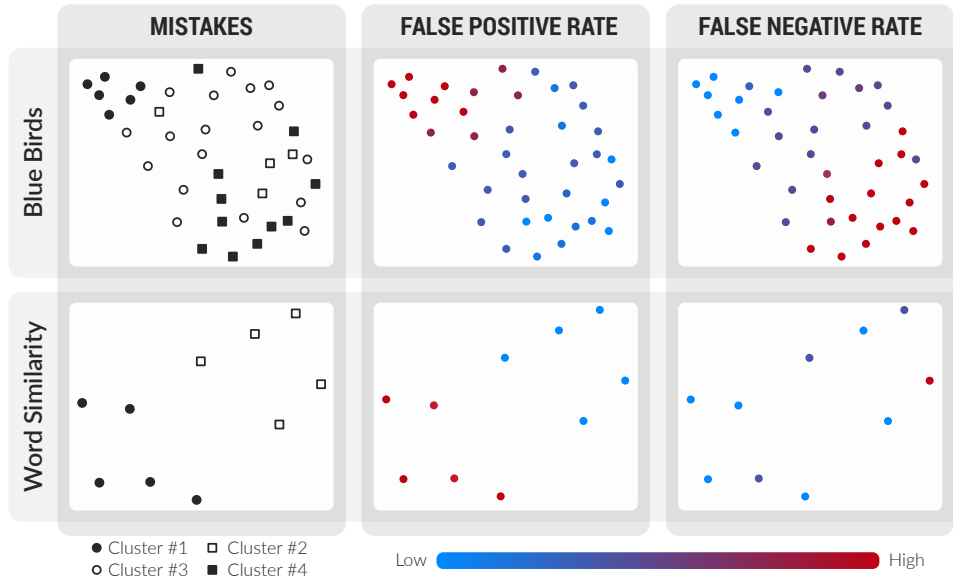


Figure 5.4: Visualization of the learned predictor embeddings. Each dot in the plots represents a predictor projected on the 2D plane using UMAP (McInnes et al., 2018). In the left, the predictors are first clustered based on which instances they make mistakes on and then colored and shaped based on which cluster they belong to (in the embedding space, predictors that make similar mistakes tend to cluster together). In the middle, the predictors are colored based on their false positive rate. In the right, the predictors are colored based on their false negative rate.

experiment. More specifically, we added an “always correct” and an “always wrong” oracle to all datasets used in our experiments. It turns out the predicted qualities for the two oracles are the highest and lowest among all predictors, respectively. This indicates that our model is indeed capable of uncovering such highly competent and incompetent predictors.

5.3.4 Predictor Embeddings Visualization

To evaluate whether the learned predictor embeddings are meaningful in some way, we perform dimensionality reduction using UMAP (McInnes et al., 2018), plot them in Figure 5.4, and color predictors in three different ways, which can help us understand the information captured by the manifold:

1. Mistakes Cluster: To cluster predictors, we represent each with a one-hot vector that indicates the instances it made mistakes on, and then run agglomerative clustering using ℓ_1 distance as the distance metric. On the plot, each cluster is associated with a unique shape.
2. False Positive Rate: Each predictor is colored based on its false positive rate.
3. False Negative Rate: Each predictor is colored based on its false negative rate.

We have provided figures for the Blue Birds and Word Similarity datasets which are the only ones for which all predictors provide annotations for all instances (it is unclear

how to properly compute the mistakes clustering distance metric when some or most of the annotations are missing). From these plots, it is clear that the learned predictor embeddings encode both the expertise of the corresponding predictor as well as the likelihood of making a false positive or false negative mistake.

5.4 KEY TAKEAWAYS

In this chapter, we have introduced a learning framework for: (i) training deep models directly on data with imperfect annotations, and (ii) modeling the processes that produced the labels. Our approach improves upon the classical and widely used two-stage setup (first aggregate and denoise the labels and then train the model) by merging the two stages. As a result, we are able to train models end-to-end using multiple noisy labels, while estimating the difficulties of the examples and learning accurate representations for the annotators that produced the labels. Experimental results on multiple small and large scale publicly available crowdsourcing datasets indicate that our method results in significant gains in accuracy (up to 25% relative gain over the current state-of-the-art approaches for aggregating noisy labels). Moreover, it turns out that training the model to predict multiple related labels simultaneously improves the learned representations and results in further gains in the predictive performance of the model. Finally, we performed an ablation study to evaluate the effect of both end-to-end learning and instance features and showed that both contribute to the performance gains achieved by the proposed method.

Perhaps most interestingly, the proposed learning framework more closely resembles the human learning process, than the classical semi-supervised learning paradigm. Humans are extremely good at learning from imperfect supervision. There are a few future directions of interest for this work that are outside the scope of this thesis. It would be interesting to explore generalizations of our models to non-boolean, discrete-valued functions, or even to real-valued functions, as humans also seem very capable of handling such cases. Furthermore, this work could form the first step towards developing a *self-reflection* framework for autonomous learning systems—defined as the ability of a system to reflect on its own learning process and improve—and also enabling more effective *active learning*.

Note that, the methods presented in this part of the thesis are not limited to crowdsourcing or estimating accuracies of classifiers from unlabeled data. In fact, the underlying ideas can be used more generally in the context of unsupervised or semi-supervised learning. For example, based on the method presented in this chapter we developed a method for performing graph node classification in a graph-based semi-supervised learning setting, that we present in the following chapter.

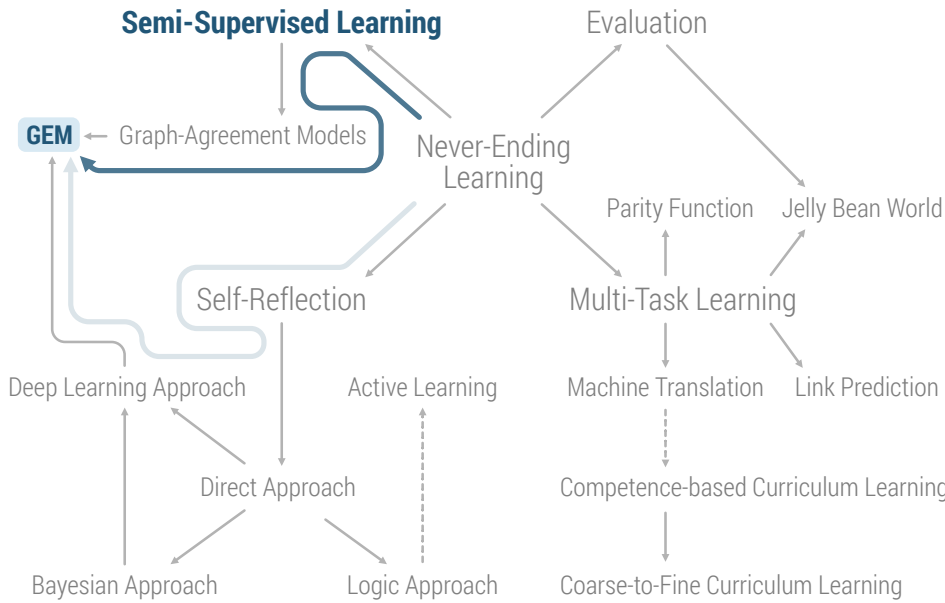


Figure 6.1: Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in [Section 1.4](#).

In the previous three chapters, we showed that learning collections of functions allows us to perform completely unsupervised evaluation. Specifically, we focused on never-ending learning and crowdsourcing and proposed multiple methods for estimating the accuracy of classifiers from unlabeled data and for aggregating crowdsourced annotations. As previously mentioned, these methods rely on a common idea that is central to machine learning and many other fields. This implies that our algorithms should also be applicable to other areas of machine learning. In this chapter, we show how we can adapt the algorithm presented in [Chapter 5](#) to solve the problem of graph-based node classification, which may at first seem to be completely different than the previous problems we considered.¹

Graph-based algorithms have gained popularity due to their success in solving semi-supervised learning problems. They often rely, either implicitly (e.g., graph convolutional networks [GCNs]) or explicitly (e.g., label propagation methods), on the assumption that a graph node is in some sense similar to its neighbors. However, this

¹ The work we present in this chapter has been published in (Platanios et al., 2020b).

assumption is often too strong for real-world graphs. Inspired by our earlier work, we propose to treat the labels of a node's neighbors as multiple noisy predictors of its own label. To this end, in this chapter we propose a probabilistic model which allows us to learn two functions: (i) a node label predictor for each node, and (ii) a model which, given a node and its neighbor, learns to predict how their labels are related in the form of a confusion matrix. We propose an inference algorithm for performing joint end-to-end learning of these two functions, called GEM. We conduct experimental evaluation of GEM on 4 graph node classification datasets of varied difficulty and show gains of up to 55% relative improvement in accuracy over the baseline methods. Moreover, GEM is able to boost the performance of multi-layer perceptrons (MLPs) to match that of GCNs, despite the fact that MLPs ignore all graph structure information, and are significantly more computationally and memory efficient than GCNs. GEM is inspired and is a direct application of the ideas presented in the preceding chapters.

6.1 INTRODUCTION TO GRAPH NODE CLASSIFICATION

Artificial Intelligence has witnessed an impressive leap in the last decade. These advances were made possible by increases in computing power and also, importantly, via the availability of large amounts of data. However, labeled examples are difficult or impractically expensive to obtain in real-world settings. On the other hand, unlabeled data is often easy and cheap to obtain. Semi-supervised learning is a paradigm that aims to take advantage of both labeled and unlabeled data in order to train models with better generalization capabilities. Moreover, real-world data is often structured (e.g., data samples are interconnected), and this structure can be leveraged to further improve our prediction models. In many cases, this structure can be represented using graphs as their flexibility allows us to represent various kinds of relationships, such as knowledge graphs, social networks, protein interactions networks, chemical bond networks, citation networks, etc. In this chapter, we propose GEM, an efficient graph-based semi-supervised learning algorithm inspired from our earlier work presented in Chapters 2, 3, 4, and 5, that improves the generalization capabilities of learned models by leveraging both unlabeled data and the graph structure, and that can be used to enhance any baseline model as a blackbox component, whether or not it leverages graph structure information itself.

We focus on the problem of graph node classification. This problem has received significant attention over the years and many different approaches have been developed to solve it, ranging from the earlier label propagation methods (e.g., Zhu et al., 2003; Zhou et al., 2004; Hein and Maier, 2006) to the more recent graph neural networks (e.g., Kipf and Welling, 2016; Gilmer et al., 2017; Hamilton et al., 2017). We provide a more complete review of these methods in Section 6.1.1. The current state-of-the-art methods for this problem are graph-based neural networks (Kipf and Welling, 2016; Veličković et al., 2018) combined with variants of label propagation (e.g., our work in Stretcu et al., 2019), because they are able to utilize information provided by the graph structure and the unlabeled nodes. These methods rely, either implicitly (e.g., graph convolutional networks [GCNs]) or explicitly (e.g., label propagation methods), on the assumption that a graph node is in some sense similar to its neighbors. However, we previously

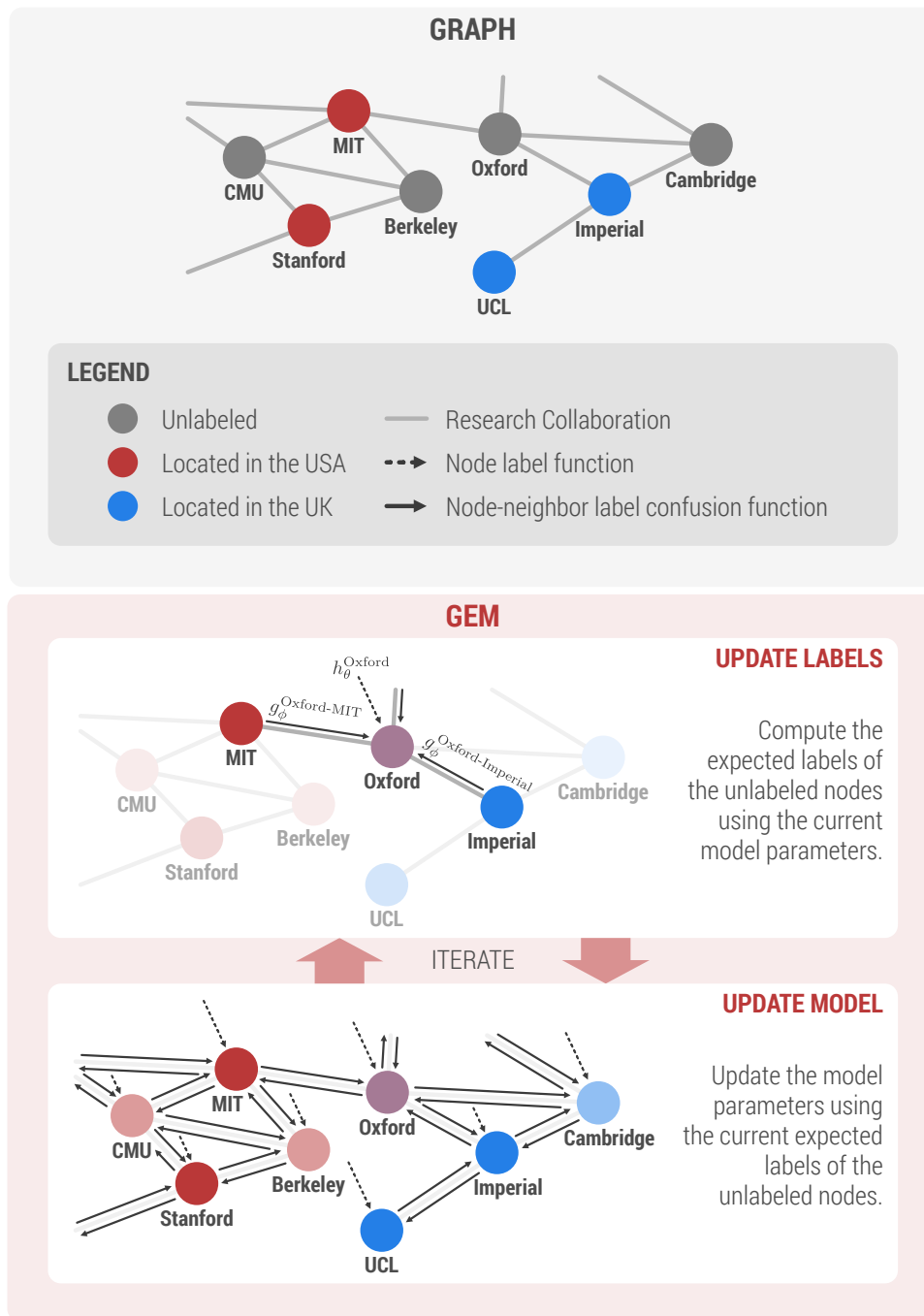


Figure 6.2: Overview of GEM, an efficient graph-based semi-supervised learning algorithm.

showed that this assumption is often too strong for real-world graphs (Stretcu et al., 2019). In fact, it is false for about 20%-30% of all node pairs in some of the most frequently used benchmark datasets. Rather, we make the weaker assumption that the labels of a node's neighbors can provide information about that node's label, even when their labels differ.

Our method is inspired by the observation that this setting bears a striking resemblance to that of crowdsourcing where multiple annotators provide “noisy” labels for an instance, and methods are developed to aggregate these labels into a single estimate of the underlying ground truth (see e.g., Zhou et al., 2015; Platanios et al., 2016; Khetan et al., 2018; Platanios et al., 2020a). For this reason, we posit that the neighbors of a node act as multiple noisy predictors of the node’s label, and we design a method for aggregating these labels into a single estimate of the underlying true label. To do so, we propose a *generative* approach that models the distribution of the node labels based on their features and their neighbors’ labels and features. Unlike existing methods, which learn attention over neighbors (Veličković et al., 2018) or agreement between node labels and their neighbors’ labels (Stretcu et al., 2019), we learn the full *confusion matrix* between node labels and their neighbors’ labels. More concretely, instead of learning a single score in $[0, 1]$ that specifies the probability that the label of a node i agrees with the label of its neighbor j like we did in our previous work (Stretcu et al., 2019), or the attention node i should put on its neighbor j like Veličković et al. (2018), we learn a probability distribution over node i ’s labels conditioned on the label of its neighbor j . This formulation is more expressive because when the neighbors’ labels disagree with the node label, we can estimate how the two labels depend on each other. In contrast to most prior work, our method also models the joint distribution of labels over all nodes in a graph, enabling it to capture more complex dependencies between the node labels.

We formulate our generative model as a *Markov random field* (MRF; Hammersley and Clifford, 1971) and propose a *variational inference* (Jordan et al., 1999) algorithm for learning the model parameters and inferring the missing node labels. We perform an extensive empirical analysis showing that our method, GEM, is able to improve the state-of-the-art accuracy on several established graph node classification datasets, and is also robust to graphs with “noisy” edges (i.e., edges between nodes with different labels), while popular existing methods such as GCNs are not. In summary, GEM has the following properties:

1. It can enhance any graph node classification model as a blackbox component, including neural networks, so long as it is trainable via continuous optimization (e.g., gradient descent methods).
2. It is *robust* to “noisy” edges, improving the performance of baseline models even when most of the graph edges connect nodes with different labels.
3. It incurs no additional cost when making predictions, as opposed to GCNs, for example, which have high computational and memory cost at inference time. More specifically, we show that GEM can be used to train a simple multi-layer perceptron to obtain better generalization accuracy than GCN.
4. It obtains state-of-the-art results on several benchmark datasets.

A high-level overview of GEM is shown in [Figure 6.2](#).

6.1.1 Related Work

Graph-based semi-supervised learning has been an active research area for more than two decades. Blum and Chawla (2001) were among the early proponents of the idea that graph nodes which are connected by an edge should have the same label, and proposed a solution based on the min-cut algorithm. Several approaches extended this *label propagation* idea (e.g., Zhu et al., 2003; Zhou et al., 2004; Belkin et al., 2006; Hein and Maier, 2006), by propagating labels along graph edges, such that the final label of a node will effectively be a majority vote over its neighbors' labels. These methods are often implemented by regularizing the predicted labels using the graph Laplacian without taking advantage of any node features that may be available (e.g., in the form of meta-data). Chapelle et al. (2009) and Bengio et al. (2006) provide an extensive review of such methods.

Recent approaches have focused on combining graph-based methods with the power of neural networks in learning abstract representations. Graph neural networks learn node representations as a function of their neighbors' representations, which are aggregated using some form of a convolution operation (e.g., Bruna et al., 2014; Duvenaud et al., 2015; Kipf and Welling, 2016; Ying et al., 2018), message-passing (e.g., Gilmer et al., 2017), or trainable aggregation functions (e.g., Hamilton et al., 2017; Gao and Ji, 2019). Veličković et al. (2018) and Thekumparampil et al. (2018) extended these approaches by using attention over the graph edges. Another line of work uses the graph as a form of regularization when training a neural network over the node features. For example, Bui et al. (2018) combine label propagation and deep learning by using a loss function that encourages neighboring nodes to have similar representations—as encoded by a neural network. We previously extended this approach by also learning which pairs of neighbors should have the same label, thus being able to handle graphs with “noisy” edges (Stretcu et al., 2019). Other graph-based regularization approaches include the work of Yang et al. (2016), Verma et al. (2019), Weston et al. (2008), and Weston et al. (2012). The work of He et al. (2007) and Ma et al. (2019) is similar to the approach we present in this chapter, in that they model the joint distribution of the labels of all nodes in the graph. Similarly, Qu et al. (2019) model the joint distribution of the node labels using a conditional random field. Their approach, termed the “Graph Markov Neural Network,” is also trained using a form of variational inference, but the variational family of distributions they consider is defined by a separate graph neural network. As described in the following section, our approach is significantly different in that we explicitly model the relationship of the labels of neighboring nodes and use a mean field approximation when performing variational inference.

6.2 PROPOSED ALGORITHM

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph where $\mathcal{V}$ is a set of nodes (or vertices) and $\mathcal{E}$ is a set of edges represented as pairs of node indices. Each node i in the graph is associated with a feature vector $\mathbf{v}_i \in \mathbb{R}^M$ and a label $y_i \in \{1, \dots, C\}$ that may or may not be observed. Furthermore, let $\mathcal{V}_L \subset \mathcal{V}$ be a subset of the graph nodes for which y_i is observed. Our task is to learn a function that maps all nodes to their corresponding

labels. Note that we consider a setting in which every node has a single label y_i , but our method can easily be extended to multi-label settings. For simplicity we also initially assume a *transductive* setting where we are given the whole graph at training time with some of the labels missing. However, as we shall show, the proposed method is also directly applicable to *inductive* settings where the learned node classification function is evaluated on nodes not seen at training time. We propose a probabilistic model which allows us to learn two functions:

LABEL PRIOR. We define the ground truth prior as a function $h_\theta(\mathbf{v}_i)$ that is parameterized by θ and that approximates the true distribution of the label given a representation of the i^{th} node, $\mathbf{v}_i$. In our setting, $h_\theta(\mathbf{v}_i) \in \mathbb{R}_{\geq 0}^C$ and $\sum_j [h_\theta(\mathbf{v}_i)]_j = 1$, where C is the number of values the label can take (i.e., assuming categorical labels). More specifically, $[h_\theta(\mathbf{v}_i)]_k$ can be interpreted as the probability that $y_i = k$ given the node features $\mathbf{v}_i$, where we use square brackets and subscripts to denote indexing of vectors, matrices, and tensors. As an example, h_θ could be a GCN that would normally be trained in isolation using the cross-entropy loss function. In our method, h_θ is trained jointly with the confusion network (described in the next paragraph) by performing probabilistic inference over the latent variable model that we propose in [Section 6.2.1](#).

CONFUSION PRIOR. We define the node pair confusion prior as the matrix $g_\phi(\mathbf{v}_i, \mathbf{v}_j) \in \mathbb{R}_{\geq 0}^{C \times C}$ for each pair of neighboring nodes $(i, j) \in \mathcal{E}$, where $\sum_k [g_\phi(\mathbf{v}_i, \mathbf{v}_j)]_{kl} = 1$, for all $l \in \{1, \dots, C\}$. $[g_\phi(\mathbf{v}_i, \mathbf{v}_j)]_{kl}$ can be interpreted as the probability that node i has label k given that its neighbor j has label l .

Our goal is to learn the functions h_θ and g_ϕ given the provided graph $\mathcal{G}$ and the labeled nodes $\mathcal{V}_L$, and to more accurately infer the labels of the unlabeled nodes. To this end, we propose a new probabilistic latent variable model in [Section 6.2.1](#), along with an efficient inference algorithm in [Section 6.2.2](#). This model does not depend on the specific functional forms of h_θ and g_ϕ and so, in [Section 6.2.3](#) we present the actual parametric function forms for h_θ and g_ϕ that we used in our experiments. In order to simplify notation, in what follows we assume that h_θ and g_ϕ both convert their outputs to log space, and we use the following shorthand notation: $h_k^i = [h_\theta(\mathbf{v}_i)]_k$ and $g_{kl}^{ij} = [g_\phi(\mathbf{v}_i, \mathbf{v}_j)]_{kl}$.

6.2.1 Model

The label prior function, h_θ , effectively gives us a prior over node labels, while the confusion function, g_ϕ , describes how the labels of neighboring nodes are related. Since these functions only allow us to model pairwise interactions between the labels of neighboring nodes, a natural way to define a probability distribution over the node labels is using a *Markov random field* (MRF; Hammersley and Clifford, 1971; Murphy, 2012). Therefore, we propose the model:

$$p(\mathbf{y} \mid \theta, \phi) \propto \exp \left\{ \sum_{i=1}^{|\mathcal{V}|} h_{y_i}^i + \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} g_{y_i y_j}^{ij} \right\}, \quad (6.1)$$

where $\mathcal{N}_i$ is the set of neighbors of node i . A standard approach for performing inference with probabilistic generative models is to maximize the likelihood of the observed data with respect to the model parameters. However, the normalizing constant of this distribution is intractable and thus, even if all node labels were observed, we would still need to use an approximate inference method to learn the values of θ and ϕ that maximize the likelihood. Moreover, in real-world semi-supervised learning settings most of the labeled nodes are unobserved, and have to be treated as latent variables which would require another inference method to estimate them. In the next section, we propose an efficient approximate inference method for learning θ , ϕ , and the unlabeled nodes. We then provide considerable empirical evidence demonstrating the strength of our method.

6.2.2 Inference

One approach for learning θ and ϕ is to marginalize out the unobserved y_i variables and then maximize the likelihood function defined in Equation 6.1 with respect to θ and ϕ . However, this approach is computationally intractable due to the combinatorial sums that would need to be computed. An alternative approach that avoids this marginalization would be to use the *expectation maximization (EM)* algorithm originally proposed by Dempster et al. (1977). In fact, it was previously observed that in many cases the EM algorithm can perform much better than the marginalization approach (Bishop, 2006), as the latter tends to get stuck in bad local optima. However, the likelihood function defined in Equation 6.1 does not factorize with respect to each node's label (due to the confusion terms that “tie” the labels of neighboring nodes in the graph together), and this makes a naive application of the EM algorithm also intractable. For this reason, we use *variational inference* instead, which is a generalization of the EM algorithm and allows us to obtain an efficient approximate inference algorithm (Jordan et al., 1999). In the rest of this section, we derive our inference algorithm in detailed steps. A summary of the algorithm, which ends up being simple and intuitive, is shown in Algorithm 6.2.

6.2.2.1 Variational Inference

The basic idea of variational inference is to posit a family of tractable distributions $q(\mathbf{y}, \theta, \phi)$ called the *variational family*, and to find the distribution within this family that is *closest* to the distribution of interest, which in our case is the likelihood function $p(\mathbf{y} \mid \theta, \phi)$ of Equation 6.1. The measure of closeness that variational inference minimizes is the Kullback-Leibler divergence between the $q(\mathbf{y}, \theta, \phi)$ and $p(\mathbf{y} \mid \theta, \phi)$:

$$\begin{aligned} \mathcal{KL}(q \parallel p) &\triangleq \mathbb{E}_q \left\{ \log \frac{q(\mathbf{y}, \theta, \phi)}{p(\mathbf{y} \mid \theta, \phi)} \right\}, \\ &= \mathbb{E}_q \{ \log q(\mathbf{y}, \theta, \phi) \} - \mathbb{E}_q \{ \log p(\mathbf{y} \mid \theta, \phi) \} \triangleq \mathcal{L}, \end{aligned} \quad (6.2)$$

where the expectation is taken with respect to $q(\mathbf{y}, \theta, \phi)$.² In order to make inference tractable we propose to use the common *mean field approximation* for the node labels, while keeping θ and ϕ together:

$$q(\mathbf{y}, \theta, \phi) = q(\theta, \phi) \prod_{i=1}^{|\mathcal{V}|} q(y_i), \quad (6.3)$$

where $q(\theta, \phi) = \mathbb{1}\{\theta = \tilde{\theta}, \phi = \tilde{\phi}\}$ is a delta function where all probability mass is placed at $(\tilde{\theta}, \tilde{\phi})$, and $q(y_i)$ is a categorical distribution with parameters $\tilde{y}_i$.

Variational inference is a coordinate descent algorithm, where at every iteration, we consider each variational distribution (i.e., $q(y_i)$ and $q(\theta, \phi)$) in isolation while keeping all other variational distributions fixed, and minimize the KL divergence with respect to the isolated variational distribution. To minimize the KL divergence with respect to $q(y_i)$, we write the variational derivative of $\mathcal{L}$ (which defined in Equation 6.2) with respect to $q(y_i)$ and solve for $q(y_i)$ such that the variational derivative is zero:

$$\frac{d\mathcal{L}}{dq(y_i)} = -\log q(y_i) - 1 + \mathbb{E}_{q(\mathbf{y}_{-i}, \theta, \phi)}\{\log p(\mathbf{y} | \theta, \phi)\} = 0. \quad (6.4)$$

Solving for $q(y_i)$ we find the distribution $q^*(y_i)$ that minimizes the KL divergence:

$$q^*(y_i) \propto \exp \left\{ \mathbb{E}_{q(\mathbf{y}_{-i}, \theta, \phi)}\{\log p(\mathbf{y} | \theta, \phi)\} \right\}. \quad (6.5)$$

We use this equation to derive the variational updates for each variable. Letting $\tilde{y}_k^i = q^*(y_i = k)$, we have that:

$$\begin{aligned} \log \tilde{y}_k^i &= \mathbb{E}_q \{ \log p(\mathbf{y} | \theta, \phi) \}, \\ &= h_k^i + \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbb{E}_q \{ g_{ky_j}^{ij} + g_{y_jk}^{ji} \} + K, \\ &= h_k^i + \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \sum_{l=1}^C [g_{kl}^{ij} + g_{lk}^{ji}] \tilde{y}_l^j + K \Rightarrow \\ \tilde{y}_k^i &\propto \exp \left\{ h_k^i + \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \sum_{l=1}^C [g_{kl}^{ij} + g_{lk}^{ji}] \tilde{y}_l^j \right\}, \end{aligned} \quad (6.6)$$

where the expectations are taken with respect to $q(\mathbf{y}_{-i}, \theta, \phi)$, and K is a constant.

In the case where the variational distribution is a delta function, such as $q(\theta, \phi)$ in our model, the entropy term $\mathbb{E}_q\{\log q(\theta, \phi)\}$ in the objective function is zero. Thus, the distribution that optimizes $\mathcal{L}$ with respect to $q(\theta, \phi)$, while keeping all other variational distributions fixed is:

$$q^*(\theta, \phi) = \mathbb{1}_{\{\theta=\tilde{\theta}, \phi=\tilde{\phi}\}}, \quad (6.7)$$

² We deviate from the standard definition of $\mathcal{L}$ in that the variational objective is normally defined as the negation of that quantity.

where $\mathbb{1}_{\{\cdot\}}$ is the indicator function and has value 1 when the subscript condition is satisfied, and 0 otherwise, and:

$$\tilde{\theta}, \tilde{\phi} = \arg \max_{\theta, \phi} \mathbb{E}_{q(\mathbf{y})} \{\log p(\mathbf{y} \mid \theta, \phi)\}, \quad (6.8)$$

and where the expectation is taken with respect to $q(\mathbf{y})$, and $p(\mathbf{y} \mid \theta, \phi)$ is defined in Equation 6.1. As mentioned earlier, this quantity is intractable to compute due to its normalizing constant. We could potentially approximate it using *Markov chain Monte Carlo (MCMC)* methods (Gelfand and Smith, 1990), but this would result in a very expensive update for θ and ϕ . Therefore, we chose to use the *pseudolikelihood approximation* that has often been previously used successfully for inference in MRFs (e.g., Hyvärinen, 2006; Vishwanathan et al., 2006; Sutton and McCallum, 2007):

$$p(\mathbf{y} \mid \theta, \phi) \approx \prod_{i=1}^{|\mathcal{V}|} p(y_i \mid \mathbf{y}_{\mathcal{N}_i}, \theta, \phi), \quad (6.9)$$

where:

$$p(y_i \mid \mathbf{y}_{\mathcal{N}_i}, \theta, \phi) \propto \exp \left\{ h_{y_i}^i + \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} [g_{y_i y_j}^{ij} + g_{y_j y_i}^{ji}] \right\}. \quad (6.10)$$

Therefore, we have that:

$$\begin{aligned} \mathbb{E}_q \{ \log p(\mathbf{y} \mid \theta, \phi) \} \approx \\ \sum_{i=1}^{|\mathcal{V}|} \sum_{k=1}^C h_k^i \tilde{y}_k^i + \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \sum_{l=1}^C [g_{kl}^{ij} + g_{lk}^{ji}] \tilde{y}_k^i \tilde{y}_l^j - \mathbb{E}_q \{ \log Z_i \}, \end{aligned} \quad (6.11)$$

where:

$$Z_i = \sum_{k=1}^C \exp \left\{ h_k^i + \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \sum_{l=1}^C [g_{kl}^{ij} + g_{lk}^{ji}] y_l^j \right\}, \quad (6.12)$$

and we approximate its expectation with respect to $q(\mathbf{y}_{\mathcal{N}_i})$ using sampling:

$$\mathbb{E}_{q(\mathbf{y}_{\mathcal{N}_i})} \{ \log Z_i \} \approx \frac{1}{N_{\text{samples}}} \sum_{\mathbf{y}_{\mathcal{N}_i} \sim q(\mathbf{y}_{\mathcal{N}_i})} \log Z_i. \quad (6.13)$$

Since Z_i is specific to each node, it is efficient to compute, taking time $\mathcal{O}(\delta_i C^2)$ where δ_i is the degree of node i in $\mathcal{G}$. We compute the variational update for θ and ϕ by plugging this approximation into Equation 6.8 and using an off-the-shelf gradient ascent algorithm.

6.2.2.2 Initialization

It is important to note that variational inference finds local optima of the likelihood function, and so the initial point can play an important role. Therefore, having defined the variational updates for $\mathbf{y}$, θ , and ϕ , it remains to describe how they are initialized. As we discuss in the next section, h_θ and g_ϕ are neural networks in all our experiments. We thus initialize $\tilde{\theta}$ and $\tilde{\phi}$ randomly, using the popular variance scaling initialization method proposed by Glorot and Bengio (2010). For the initialization of $\tilde{\mathbf{y}}$, we use a simple form of *label propagation* (Zhu et al., 2003), since it is efficient and tends to perform well in practice. The exact initialization algorithm for $\tilde{\mathbf{y}}$ is shown in Algorithm 6.1.

Algorithm 6.1: Initialization of the variational labels $\tilde{\mathbf{y}}$

Inputs: Graph $\mathcal{G}$ that contains some labeled nodes and some unlabeled nodes.

```

1 Initialize  $\tilde{\mathbf{y}}$  as a  $|\mathcal{V}| \times C$  matrix filled with zeros.
2 labeled  $\leftarrow$  Empty set of nodes.
3 nodesToLabel  $\leftarrow$  Empty set of nodes.
  // Start with the labeled nodes.
4 foreach node  $i \in \mathcal{G}.\text{labeledNodes}$  do
5    $\tilde{\mathbf{y}}_{\mathcal{G}, \text{labelOf}(i)}^i \leftarrow 1$ 
6   labeled.add( $i$ ); nodesToLabel.remove( $i$ )
7   foreach  $j \in \mathcal{G}.\text{neighborsOf}(i)$  s.t.  $j \notin \text{labeled}$  do
8     nodesToLabel.add( $j$ )
  // Proceed with the unlabeled nodes.
9 while nodesToLabel is not empty do
10  foreach node  $i \in \text{nodesToLabel}$  do
11    scores  $\leftarrow$  Zero-initialized array of size  $C$ .
12    foreach  $j \in \mathcal{G}.\text{neighborsOf}(i)$  s.t.  $j \in \text{labeled}$  do
13      for  $k \in \{1, \dots, C\}$  do
14        scores[ $k$ ] +=  $\tilde{\mathbf{y}}_k^j$ 
15    for  $k \in \{1, \dots, C\}$  do
16       $\tilde{\mathbf{y}}_k^i = \text{scores}[k] / \text{scores.sum}()$ 
17  foreach node  $i \in \text{nodesToLabel}$  do
18    labeled.add( $i$ ); nodesToLabel.remove( $i$ )
19    foreach  $j \in \mathcal{G}.\text{neighborsOf}(i)$  s.t.  $j \notin \text{labeled}$  do
20      nodesToLabel.add( $j$ )

```

Output: $\tilde{\mathbf{y}}$.

6.2.2.3 Complete Algorithm

The complete inference algorithm that allows us to learn θ and ϕ , as well as infer any missing node labels, is shown in Algorithm 6.2. We refer to our algorithm as GEM in

Algorithm 6.2: GEM inference algorithm.

Inputs: Graph $\mathcal{G}$ that contains some labeled nodes and some unlabeled nodes.

- 1 Initialize $\tilde{\theta}$ and $\tilde{\phi}$ using the variance scaling initialization method proposed by Glorot and Bengio (2010).
- 2 Initialize $\tilde{\mathbf{y}}$ using Algorithm 6.1.
- 3 **while not converged do**
- 4 **foreach** node $i \in \text{shuffle}(\mathcal{G}.\text{unlabeledNodes})$ **do**
- 5 **for** $k \in \{1, \dots, C\}$ **do**
- 6 // Use the most up-to-date $\tilde{\mathbf{y}}$'s for the following update.
 Update $\tilde{\mathbf{y}}_k^i$ using Equation 6.6.
- // Use an off-the-shelf gradient ascent algorithm for this step.
- 7 Update $\tilde{\theta}$ and $\tilde{\phi}$ using Equations 6.8 and 6.11.
- 8 Check for early stopping and terminate if necessary.

Output: $\tilde{\mathbf{y}}$, $\tilde{\theta}$, and $\tilde{\phi}$.

the subsequent sections. It terminates when either: (i) the change in the parameter values across learning iterations becomes negligible, or (ii) an *early stopping* condition is satisfied. The specific early stopping condition we used in our experiments is when the accuracy of $\hat{\mathbf{y}}^i = \arg \max \tilde{\mathbf{y}}^i$, when compared to the true node labels over a validation dataset, has not improved over the last 10 iterations. A similar early stopping condition is used when updating $\tilde{\theta}$ and $\tilde{\phi}$ using gradient ascent. Intuitively, what this algorithm does is that in each iteration: (i) we estimate the expected values of the unobserved node labels given our current model parameters, and (ii) we update our model parameters to maximize the likelihood of these expected values. These two steps are analogous to the E and M steps of the EM algorithm. Note that this is a form of a coordinate ascent method for maximizing the likelihood function of Equation 6.1.

6.2.3 Label and Confusion Priors

The inference algorithm we derived in the previous section is valid regardless of the functional form we choose for h_θ and g_ϕ . We evaluate the algorithm using a number of functional forms, each motivated empirically as well as to facilitate fair comparison with existing methods. For all forms, we consider a decomposition of g_ϕ that is inspired from the decomposition presented in Chapter 5:

$$g_\phi(\mathbf{v}_i, \mathbf{v}_j) = \mathbf{D}_i \bullet \mathbf{C}_j, \quad (6.14)$$

where $\bullet$ here represents an inner product along the last dimension of the two tensors. $\mathbf{D}_i = d_{\phi_0}(\mathbf{v}_i)$ represents the latent representation of the contribution of node i to the confusion matrix, where d is a function parameterized by ϕ_0 . $\mathbf{D}_i$ is a real-valued tensor with shape $C \times C \times L$, where L is a latent dimension as well as a hyperparameter of our model. $[\mathbf{D}_i]_{kl}$ is an L -dimensional embedding representing the likelihood of confusing node i as having label l instead of k , when k is its true

label.³ $C_j = c_{\phi_1}(\mathbf{v}_j)$ is defined similarly for neighbor j of node i . Under this definition, we have $\phi = \{\phi_0, \phi_1\}$. Note that we intentionally allow for the case where $d_{\phi_0} \neq c_{\phi_1}$ so that our model can represent asymmetric relationships between the node labels.⁴ Using $L > 1$ allows the model to represent more complex relationships. An intuitive way to think about this is that we are embedding nodes and their neighbors into a common latent space and jointly clustering them. In fact, this is very similar to the manner in which matrix factorization methods behave in collaborative filtering for recommender systems. To define h_θ , d_{ϕ_0} , and c_{ϕ_1} , we consider two variants:

- Coupled GEM: We define h_θ , d_{ϕ_0} , and c_{ϕ_1} such that they share most of their parameters. Given a baseline model f_ψ that learns feature representations over graph nodes, we define:

$$h_\theta(\mathbf{v}) = h_{\text{dense}}(f_\psi(\mathbf{v})), \quad (6.15)$$

$$d_{\phi_0}(\mathbf{v}) = d_{\text{dense}}(f_\psi(\mathbf{v})), \quad (6.16)$$

$$c_{\phi_1}(\mathbf{v}) = c_{\text{dense}}(f_\psi(\mathbf{v})), \quad (6.17)$$

where h_{dense} is a densely connected layer that outputs a distribution over classes; d_{dense} and c_{dense} are densely connected layers that output $C \times C \times L$ tensors. Note that the parameters ψ are shared across both h_θ and g_ϕ . Also, note that we use different dense layers for mapping nodes and their neighbors to confusion matrices and thus g_ϕ is not necessarily symmetric. We refer to this variant as GEM in [Section 6.3](#).

- Decoupled GEM: We define h_θ , d_{ϕ_0} , and c_{ϕ_1} such that they do not share any of their parameters. As with the coupled GEM approach, given a baseline model f_ψ that learns feature representations over graph nodes, we define:

$$h_\theta(\mathbf{v}) = h_{\text{dense}}(f_{\psi_h}(\mathbf{v})), \quad (6.18)$$

$$d_{\phi_0}(\mathbf{v}) = d_{\text{dense}}(f_{\psi_g}(\mathbf{v})), \quad (6.19)$$

$$c_{\phi_1}(\mathbf{v}) = c_{\text{dense}}(f_{\psi_g}(\mathbf{v})), \quad (6.20)$$

where in this case h_θ and g_ϕ uses different instances of f_ψ . In our experiments, we typically fix the hyperparameters and architecture of f_{ψ_h} in order to make our results comparable to previously reported numbers. We train with multiple sizes for f_{ψ_g} and select the one that results in the highest validation dataset accuracy. In [Section 6.3.2](#), we explore how f_{ψ_g} affects performance, given a fixed f_{ψ_h} . We refer to this variant as GEM* in [Section 6.3](#).

Both of the aforementioned variants rely on a baseline model, f_ψ , that learns feature representations over graph nodes. In our experiments we consider two such models:

- MLP: A simple multi-layer perceptron that receives as input the features of a node and outputs a distribution over the labels. Note that this model does not

³ The “—” subscript represents taking all elements of a tensor along the corresponding dimension.

⁴ The model is able to learn graphs in which $g_{a,b}$ and $g_{b,a}^\top$ are not necessarily equal.

Dataset	#Nodes	#Edges	#Classes	#Features	#Train	#Validation	#Test
Cora	2,708	5,429	7	1,433	140	500	1,000
CiteSeer	3,327	4,732	6	3,703	120	500	1,000
PubMed	19,717	44,338	3	500	60	500	1,000
Disease	1,044	1,043	2	1,000	748	42	254

Table 6.1: Dataset statistics. We obtained Cora from Lu and Getoor (2003), CiteSeer from Sen et al. (2008), PubMed from Namata et al. (2012), and Disease from Chami et al. (2019). We use dataset train, validation, and tests splits provided by Yang et al. (2016). We decided to use these splits so that our results are comparable with numbers that were previously reported for existing methods.

use any graph structure information. We will demonstrate that training it as part of the larger model in GEM can bring its performance on par with methods that exploit information about the graph structure. When reporting results, we use a subscript after the method name (e.g., MLP_8). This simply denotes the number of hidden units used for the corresponding MLP.

- GCN: A graph convolutional network, originally proposed by Kipf and Welling (2016). In contrast to the MLP, this model does exploit information about the graph structure and significantly benefits from doing so. However, it is much more computationally expensive, since in order to produce a prediction, it needs to process both the features of a node and those of its neighbors (and potentially even more hops if it has multiple layers). Furthermore, even though GCNs can exploit information about the graph structure, we will show that their performance can also be further enhanced with GEM. GCNs have gained a lot of popularity in recent years, but as we show in Section 6.3, their performance can be matched—or even outmatched in certain cases—by MLPs that were trained using GEM.

6.3 EXPERIMENTS

In order to evaluate the proposed approach we perform two case studies. First, we run experiments using multiple baseline methods along with the same methods enhanced by GEM. We show that GEM consistently improves the performance of all baselines, in some cases by *up to 55%* relative accuracy, and outperforms the existing state-of-the-art methods for graph node classification. As part of this case study we also demonstrate: (i) models that do not incorporate information about the graph structure (e.g., a simple multi-layer perceptron) can be enhanced to perform as well as models that do (e.g., graph convolutional networks), by using GEM to train them, and (ii) the relationship between the choice of the confusion model, g_ϕ , and the performance of GEM. In the second case study, we evaluate how robust GEM and other methods are to random noise added to the graph (where “noise” here corresponds to edges connecting nodes with different labels), and discuss some interesting observations.

Model	Dataset			
	Cora	CiteSeer	PubMed	Disease
MLP				
MLP ₈	53.1	47.7	70.8	50.3
⊥+ GEM	68.9	53.2	76.3	50.8
⊥+ GEM*	77.9	68.2	73.2	63.0
MLP ₁₆	55.3	50.3	71.7	51.2
⊥+ GEM	75.8	63.6	74.9	57.1
⊥+ GEM*	75.1	69.1	74.3	63.0
MLP ₁₂₈	56.6	51.5	71.8	59.1
⊥+ GEM	79.7	74.5	79.1	81.1
⊥+ GEM*	82.6	74.6	82.9	81.5
MLP ₁₀₂₄	53.1	52.2	70.2	57.1
⊥+ GEM	76.9	70.4	80.9	82.3
⊥+ GEM*	82.7	72.0	81.9	83.5
Varying the size of g_ϕ				
MLP ₁₂₈	56.6	51.5	71.8	59.1
⊥+ GEM* ₁₆	78.7	61.2	70.0	68.5
⊥+ GEM* ₃₂	79.7	71.5	74.7	70.1
⊥+ GEM* ₆₄	81.9	71.5	79.6	74.4
⊥+ GEM* ₁₂₈	82.6	74.6	82.9	78.7
⊥+ GEM* ₂₅₆	81.8	70.9	76.8	81.5
GCN				
GCN ₈	77.2	60.0	76.5	50.8
⊥+ GEM	78.3	60.5	80.2	57.5
⊥+ GEM*	81.2	69.5	79.1	60.6
GCN ₁₆	79.8	62.2	75.1	52.8
⊥+ GEM	81.7	66.2	77.8	57.5
⊥+ GEM*	82.2	70.8	79.3	77.2
GCN ₁₂₈	77.7	67.1	74.7	61.1
⊥+ GEM	78.4	67.2	78.4	87.0
⊥+ GEM*	84.4	74.5	81.2	81.9
GCN ₁₀₂₄	78.4	65.8	76.8	72.8
⊥+ GEM	82.1	73.3	79.9	90.6
⊥+ GEM*	85.1	72.4	81.8	89.8
Varying the size of g_ϕ				
GCN ₁₂₈	77.7	67.1	74.7	61.1
⊥+ GEM* ₁₆	80.7	63.2	75.4	63.4
⊥+ GEM* ₃₂	81.8	71.8	81.2	72.3
⊥+ GEM* ₆₄	82.3	73.0	80.3	78.6
⊥+ GEM* ₁₂₈	84.4	74.5	77.8	81.9
⊥+ GEM* ₂₅₆	80.7	71.2	76.3	77.5

Table 6.2: Test set accuracy for two baseline model architectures (discussed in [Section 6.3.2](#)). Shaded rows correspond to our method. The best result per row group is shown in bold red font and the best result across all rows is underlined.

Model	Dataset			
	Cora	CiteSeer	PubMed	Disease
ManiReg (Belkin et al., 2006)	59.4	60.1	70.7	—
SemiEmb (Weston et al., 2012)	59.0	59.6	71.7	—
LP (Zhu et al., 2003)	68.0	45.3	63.0	—
DeepWalk (Perozzi et al., 2014)	67.2	43.2	65.3	—
ICA (Lu and Getoor, 2003)	75.1	69.1	73.9	—
Planetoid (Yang et al., 2016)	75.7	64.7	77.2	—
Chebyshev (Defferrard et al., 2016)	81.2	69.8	74.4	—
MLP _[250,100] +NGM (Bui et al., 2018)	—	—	75.9	—
MoNet (Monti et al., 2017)	81.7	—	78.8	—
GCN ₁₆ (Kipf and Welling, 2016)	81.5	70.3	79.0	69.7
GAT ₈ (Veličković et al., 2018)	83.0	72.5	79.0	70.4
HGCN (Chami et al., 2019)	79.9	—	80.3	74.5
GCN ₁₆ +O-BVAT (Deng et al., 2019)	83.6	74.0	79.9	—
GMNN (Qu et al., 2019)	83.7	73.1	81.8	—
G-U-Net (Gao and Ji, 2019)	84.4	73.2	79.6	—
MLP ₁₂₈	56.6	51.5	71.8	59.1
⌞+ NGM	77.7	67.8	73.6	—
⌞+ VAT	56.5	56.1	73.1	—
⌞+ VATENT	24.1	46.7	70.1	—
⌞+ GAM	80.7	73.0	82.8	—
⌞+ GAM*	70.7	70.3	71.9	—
⌞+ GEM	79.7	74.5	79.1	81.1
⌞+ GEM*	82.6	74.6	82.9	81.5
GCN ₁₂₈	77.7	67.1	74.7	61.1
⌞+ NGM	81.4	68.9	76.2	—
⌞+ VAT	79.0	69.5	76.8	—
⌞+ VATENT	83.4	69.8	75.0	—
⌞+ GAM	86.2	73.5	86.0	—
⌞+ GAM*	84.2	71.3	77.0	—
⌞+ GEM	78.4	67.2	74.7	61.1
⌞+ GEM*	84.4	74.5	81.2	81.9
GCN _{1,024}	78.4	65.8	76.8	72.8
⌞+ NGM	82.0	70.5	68.9	—
⌞+ VAT	81.8	69.3	76.3	—
⌞+ VATENT	64.0	50.5	72.1	—
⌞+ GAM	86.0	73.6	81.6	—
⌞+ GAM*	82.4	71.9	81.2	—
⌞+ GEM	82.1	73.3	79.9	90.6
⌞+ GEM*	85.1	72.4	81.8	89.8

Table 6.3: Test set accuracy for multiple existing methods, including the current state-of-the-art (discussed in Section 6.3.2). NGM was proposed by Bui et al. (2018), VAT and VATENT were proposed by Miyato et al. (2018), and we also previously proposed GAM (Stretcu et al., 2019). Shaded rows correspond to our method. The best result per row group is shown in bold red font and the best result across all rows is underlined.

6.3.1 Experimental Setup

For both case studies we use three datasets that have become the de facto standard for evaluating graph node classification algorithms (for details, refer to our previous

work, Stretcu et al., 2019) which we obtained from Yang et al. (2016), as well as one dataset recently used by Chami et al. (2019). The datasets are listed in Table 6.1 along with some statistics. For Cora, CiteSeer, and PubMed, the node features are normalized word frequency vectors representing scientific articles, the graph edges correspond to citations between these articles, and the node labels indicate the research area of the article. For Disease, the node features indicate susceptibility of a human subject to a disease, the graph edges correspond to a disease spreading between two subjects, and the node labels correspond to whether or not the subject is infected. During each θ and ϕ update step we use the Adam optimizer (Kingma and Ba, 2015) to solve the maximization problem shown in Equation 6.8, with the learning rate set to 0.001, a maximum of 1,000 optimization iterations, and a batch size of 128. We evaluate all methods by computing the *accuracy* of the inferred node labels. We provide more details in our code repository (available at <https://github.com/eaplatanios/gem>).

6.3.2 Case Study #1: Enhancing Baselines

We present results for multiple configurations of the baseline methods and their GEM variants in Table 6.2. For GEM* we performed experiments using the following sizes for f_{ψ_g} : 16, 32, 64, 128, and 256. Table 6.2 shows the results obtained with the model that produces the highest validation accuracy, as well as all results for two configurations (MLP₁₂₈ and GCN₁₂₈, where we use a subscript on GEM* to denote the size of f_{ψ_g}). We observe that in all cases GEM is able to enhance the underlying baseline method, *achieving relative improvements of up to 55% for MLP₁₀₂₄*. We also observe that the baseline GCN models consistently outperform the baseline MLP models, which is expected given that they are able to exploit information about the graph structure. However, and perhaps most interestingly, some of the MLP models trained with GEM are able to outperform the baseline GCN models. This is interesting because it shows that the graph structure information is indeed very valuable when training our models, but may not be necessary when performing inference, as we may be able to learn a relationship between the graph structure and the node features. This is very important in practice because MLPs are both significantly more computationally and memory efficient than GCNs. Another interesting observation is that unboundedly increasing the size of the baseline model causes performance to deteriorate beyond some point. However, GEM is able to mitigate this deterioration, resulting in increased robustness to the choice of hyperparameters. Finally, we observe that varying the size of f_{ψ_g} has a significant impact on the performance of GEM. GEM is still able to consistently improve the performance of the baseline method (except for a couple instances when the confusion model size is too small). However, choosing the right size for f_{ψ_g} can result in further gains. Table 6.2 also shows that, if tuned properly, GEM* always outperforms the coupled GEM model.

In Table 6.3, we also present results comparing our GEM-enhanced baselines to previously proposed methods for graph node classification, including the current state-of-the-art. We see that in most cases GEM establishes a new state-of-the-art for this problem. It is only outperformed in a couple of instances by GAM (Stretcu et al., 2019), which was designed with a motivation similar to ours.

6.3.3 Case Study #2: Robustness Analysis

We developed GEM such that it can better handle graph edges between nodes that have differing labels, as opposed to other existing methods, such as label propagation (Zhu et al., 2003). In this section, we demonstrate this empirically. In this case study we fix the model size and study how performance varies as we introduce “disagreeing” edges to the graph. For each graph, we use uniform rejection sampling to add random “disagreeing” edges and we take snapshots of these “noisy” graphs at multiple ratios of the number of “disagreeing” edges to the total number of edges. The results are shown in Figure 6.3. We use a model size of 128 for both the MLP and the GCN, and we report results averaged across 5 runs using different random seeds. We make the following important observations from Figure 6.3:

1. The GEM-enhanced models consistently outperform the baseline models, across all proportions of “disagreeing” edges.
2. The GCN performance degrades dramatically as we increase the proportion of “disagreeing” edges, eventually converging to random predictions. This behavior is expected since GCN assumes that all edges are equally important and averages the contributions obtained from the neighbors of a node. This means that, *in complete graphs (fully-connected), GCNs will predict the same label for all nodes*. GEM helps make GCNs a bit more robust early on, but as the graph gets more connected it is unable to correct for GCN’s behavior, due to the aforementioned issue.
3. MLPs are completely robust to the proportion of “disagreeing” edges, which is also expected since they do not consider the edges of the graph. However, GEM takes the graph structure into account, and so, when enhancing MLPs with GEM, we notice that we consistently get a performance improvement, but that the improvement is sometimes smaller as the proportion of “disagreeing” edges increases.
4. The Disease dataset exhibits a very interesting effect: we see that the performance of GEM-enhanced models increases when increasing the proportion of “disagreeing” edges, eventually reaching 100% accuracy. Investigating this further revealed that this is due to the fact that in the Disease dataset, ~72% of the nodes are labeled (i.e., the majority of them) and so increasing graph connectivity has the following effects: (i) GEM is able to learn a more accurate confusion predictor, g_ϕ , and (ii) this confusion predictor can be used to more robustly infer the labels of the unlabeled nodes, since now they are connected to more labeled nodes. This observation highlights that GEM may also be very useful for settings where there is a small number of unlabeled nodes and a large number of labeled nodes. We do not observe the same behavior in the other datasets, most likely because the number of labeled nodes in those datasets is very small and so the newly added graph edges rarely connect to any of the labeled nodes.

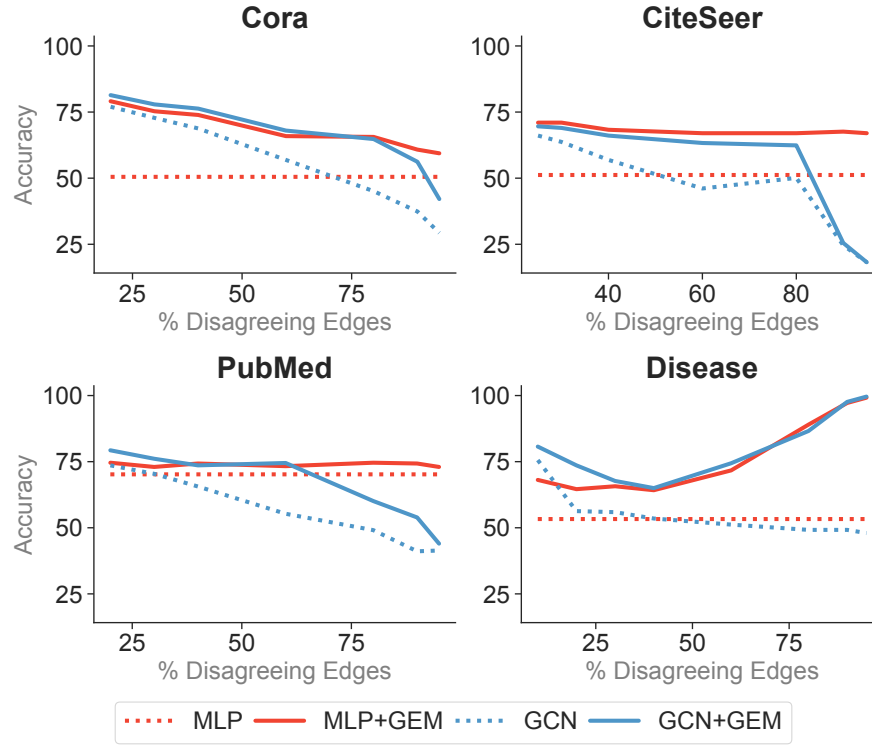


Figure 6.3: Plots of accuracy at multiple ratios of the number of “disagreeing” edges to the total number of graph edges. These results are discussed in [Section 6.3.3](#).

6.4 KEY TAKEAWAYS

We have proposed GEM, a new algorithm for graph-based semi-supervised learning. The specific problem we considered is graph node classification. Existing algorithms for solving this problem often rely on the assumption that a node is in some sense similar to its neighbors. However, this assumption is often too strong for real-world graphs. GEM addresses this problem by borrowing intuition from the area of crowdsourcing and from the work we presented in [Chapter 5](#). The key underlying idea is that we considered the labels of a node’s neighbors as noisy predictors of the node’s label. To this end, we introduced a probabilistic model which allows us to learn two functions: (i) a label predictor for each node, and (ii) a model which, given a node and its neighbor, learns to predict how their labels are related in the form of a confusion matrix. We also proposed a scalable variational inference algorithm for learning both of these functions in an end-to-end manner. Experimental results on multiple graph node classification datasets indicate that GEM results in significant gains in accuracy (up to 55% relative gain over baseline methods), thus establishing new state-of-the-art performance for the graph node classification problem. Furthermore, we performed a robustness analysis to examine how GEM performs when dealing with graphs where multiple or even most of the edges connect nodes with different labels. GEM was able to slow down the performance degradation of GCNs and to consistently enhance the performance of MLPs, even under large amounts of noise.

Most importantly, the main reason we ventured into this problem of graph node classification was so that we can showcase the usefulness and applicability of the ideas and methods we presented in the previous chapters, beyond the areas of never-ending learning and crowdsourcing. GEM provides evidence that the algorithms presented in [Part i](#) of this thesis are indeed useful and applicable in areas beyond never-ending learning and crowdsourcing.

Part II

HIGHER-ORDER LEARNING

In [Chapter 7](#), we provide some background on multi-task learning along with a brief history of the field, pointing out what the current limitations are. Then, we propose a novel abstraction, *contextual parameter generation*, that allows us to address these limitations. Finally, in [Chapter 8](#), we present results on a few different multi-task learning problems that include machine translation and knowledge graph link prediction, and showcase the power and usefulness of contextual parameter generation. We also provide a novel *neural cognitive architecture* that is later used to motivate the future work we propose in [Chapter 10](#) and [Appendix C](#).

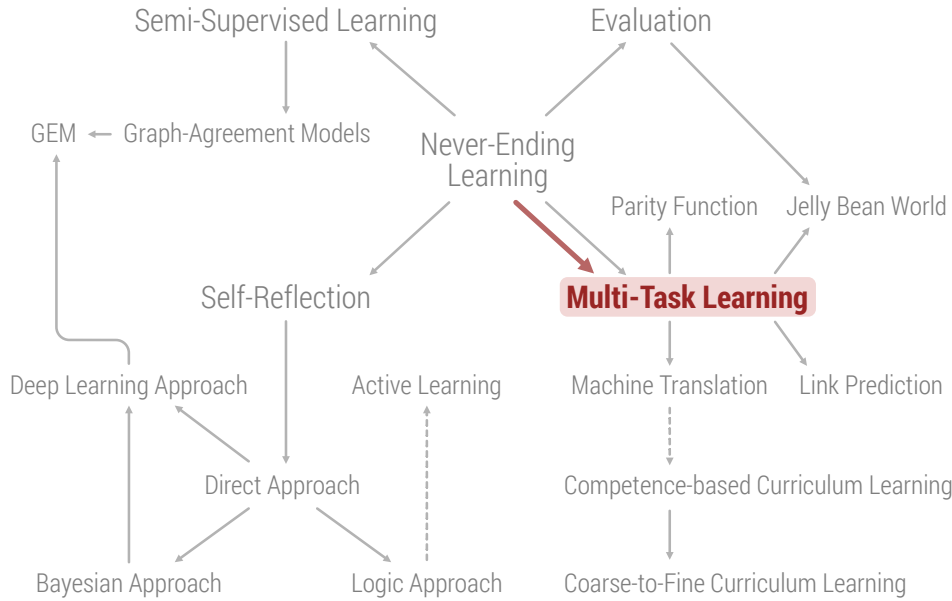


Figure 7.1: Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in [Section 1.4](#).

In the first part of this thesis, we focused on self-reflection in the context of learning collections of functions. The methods we proposed revolved mainly around estimating accuracies of classifiers from unlabeled data. However, learning collections of functions is also inherently related to *multi-task learning*. In fact, as discussed in [Chapter 1](#), multi-task learning is an important aspect of general learning and intelligence. To this end, in this chapter we propose a novel abstraction, *contextual parameter generation*, that enables large-scale multi-task learning and is able to significantly outperform existing methods for multi-task learning. Before we present our approach, we provide a short introduction to multi-task learning along with a brief history that aims to help better position and motivate our work. We also describe its relationship to other areas like *meta-learning* and *transfer learning*. Empirical evidence for the power and usefulness of contextual parameter generation is provided in the next chapter through a series of case studies.

7.1 MULTI-TASK LEARNING

Machine learning has witnessed growing success across a multitude of applications over the past years. However, despite these successes, current machine learning systems are each highly specialized to solve one or a small handful of problems. They have much narrower learning capabilities as compared to humans, often learning just a single function or model based on statistical analysis of a single dataset. One reason for this is that current machine learning paradigms are restricted and specialized to a particular problem and/or dataset, motivated by the fact that we typically care about optimizing for a particular application-specific metric. While these paradigms often allow us to achieve acceptable performance, they also fail in cases where we may have very limited amounts of training data and where the tasks that are being learned are complex (e.g., consisting of multiple sub-tasks). Furthermore, there may be multiple auxiliary training signals available that we are simply ignoring by being focused on a single metric. These signals can be thought of as other related tasks and by sharing information among them we can potentially enable our models to generalize better on the original task. Moreover, we can even go a step further and design models that learn to perform multiple tasks using the same shared representations, and where by learning one task they become better at learning others. This general approach is called *multi-task learning (MTL)* and it has been of interest to the machine learning community for many years (e.g., Caruana, 1997), mainly due to its multi-faceted nature and significance.

GENERAL INTELLIGENCE. One of the goals of this thesis is to take us a step closer to artificial general intelligence and in [Chapter 1](#) we argued that agents with this capability should not learn how to perform a single task repetitively, but rather how to perform a general variety of tasks, and how to switch between them and/or combine them to better perform other tasks (e.g., by composing them). We posited that, at a sufficiently high level of task complexity, optimal learning agents will be required—either explicitly or implicitly—to perform abstract reasoning over concepts and make informed decisions about the environment in which they are deployed. Therefore, MTL is likely an integral step towards generally intelligent systems.

HUMAN INTELLIGENCE. When we, as humans, learn new tasks, we often apply knowledge that we have acquired by learning other tasks to learn faster (e.g., curriculum in school—from a pedagogical perspective we often first learn tasks that provide us with the necessary skills and background to learn more complex tasks later). This motivates research in MTL because: (i) it seems like an effective way to learn and may thus be useful to imitate, and (ii) being able to reproduce this aspect of human learning in machine learning may help us better understand human learning. This also points out that multi-task learning is related to *curriculum learning* which can be traced back to the work of Elman (1993) and Krueger and Dayan (2009). This connection also motivated us to do some work in curriculum learning, part of which is presented in [Appendix A](#). The two curriculum learning methods we developed were published in (Platanios et al., 2019) and (Stretcu et al., 2020).

MACHINE LEARNING. MTL can be viewed as a form of *inductive transfer*, which aims to improve a machine learning model by introducing an *inductive bias* making it prefer certain hypotheses over others (Caruana, 1997). For example, ℓ_1 and ℓ_2 regularization—which are often used when training machine learning models—are both common forms of inductive bias, the first one leading to a preference for sparse solutions and the second one to a preference for smooth models (Ruder, 2017). In the case of MTL, learning to perform auxiliary tasks can provide a useful inductive bias for when learning to solve a specific target task.

7.1.1 A Brief History of Multi-Task Learning¹

NETtalk by Sejnowski (1987) is perhaps the earliest instance of multi-task learning. The authors built a neural network for reading English text. Their main motivation was to reduce computational cost by using a single neural network to learn 26 distinct tasks (mainly due to the computational constraints of the time). Most of the network hidden layers were shared among all tasks, but separate output layers were used for each task. The authors may well have not been aware that they published the first system using what would later be called multi-task learning. A few years later, Dietterich et al. (1990) noticed that they could not get decision trees to match the performance of the NETtalk neural network. According to them, one of the most likely explanations was that NETtalk was being trained on all tasks at the same time, while the same was not true for their decision tree approach. In fact, one of the systems that gathered a lot of attention at the time, ALVINN—one of the first autonomous land vehicle projects by Pomerleau (1989)—was also getting significant accuracy improvements by having a single neural network produce multiple outputs.

Around the same time, Suddarth and Kergosien (1990) tried injecting logical rules in neural networks and Abu-Mostafa (1990) tried providing hints in terms of what the network outputs should look like (e.g., the invariance hint which stated that the network output must be the same for certain input pairs). Both approaches resulted in performance improvements. One year later Pratt et al. (1991) showed how neural network training times can be significantly reduced by *pre-training* networks to solve one task and then *fine-tuning* on others. Dietterich and Bakiri (1994) proposed using error correcting output codes for what can also be effectively considered a multi-task learning problem. Silver and Mercer (1996) investigated the question of how we can transfer task knowledge between models learning to solve different problems, based on a measure of task relatedness. Caruana (1995, 1997) then extended these ideas by showing that a network learning multiple related tasks at the same time can use these tasks as an inductive bias for each other and thus learn faster. Caruana also identified various mechanisms by which multi-task learning improves generalization and showed that many of them apply to other machine learning methods too, beyond neural networks (e.g., k-nearest-neighbors and decision trees). Meanwhile, Breiman and Friedman (1997) proposed the “Curds & Whey” method for multivariate linear regression, a simple multi-task learning problem. A few years earlier, Mitchell and

These are some of the first instances of transfer learning, which is a type of multi-task learning.

¹ This section is mostly borrowed from a talk Rich Caruana gave during the “Adaptive and Multi-Task Learning” workshop at the International Conference on Machine Learning (ICML), 2019.

Thrun (1993) started exploring directions in explanation-based neural networks which, similar to the previous ideas, also exhibited a form of *inductive transfer*. Also, de Sa (1993) was looking into minimizing disagreement among different model outputs as an auxiliary task; in many ways similar to the ideas we presented in (Platanios, 2018a). Notably, this was also the time when the idea of meta-learning (i.e., learning-to-learn) first appeared with the heavily influential work of Thrun and Mitchell (1994)—which has recently become more popular than it has ever been—and Schmidhuber (1995). Thrun (1996) went on to defend his thesis focused on meta-learning and lifelong learning and to also propose methods for clustering tasks in order to perform selective cross-task transfer of knowledge (Thrun and O’Sullivan, 1998). Meanwhile, Baxter (1995) also defended his PhD thesis on learning internal representations that can be used across multiple tasks, as part of which he also developed some initial theory around multi-task learning. Then, Munro and Parmanto (1997) started looking into combining multi-task learning and ensemble learning, and Ring (1997) did some of the earliest work in continual learning—arguably yet another form of multi-task learning—which he defined as “the constant development of increasingly complex behaviors; the process of building more complicated skills on top of those already developed.” Later on, Blum and Mitchell (1998) published their work on co-training which was very influential in the development of never-ending learning by Mitchell et al. (2018) and in turn in the writing of this thesis.

Further progress was made during the following decade. Ben-David et al. (2002) proposed a theoretical framework for learning from a pool of disparate data sources. As Bayesian models were getting popular in machine learning, Bakker and Heskes (2003) proposed a Bayesian framework for multi-task learning, and similarly for kernel methods (e.g., Jebara, 2004; Lawrence and Platt, 2004; Evgeniou et al., 2005; Yu et al., 2005). Finally, since 2008 there has been a resurgence of neural network approaches which has enabled possibilities that were previously completely unattainable. Most of the core ideas behind multi-task learning however have remained the same, with the main things changing being the scale and scope of the problems being tackled. A brief overview of what we like to call the *current landscape of multi-task learning* is presented in Figure 7.2 and a more detailed discussion is provided in the following section. This recent work has served to provide evidence for the importance and usefulness of these ideas stemming from novel research work over the past few decades.

7.1.2 The Current Landscape of Multi-Task Learning

Most—if not all—of the existing work on multi-task learning can be classified in two main categories (Ruder, 2017):

HARD PARAMETER SHARING. This dates back to the work of Caruana (1995) and is arguably the most common approach used for multi-task learning to this day. The key idea is that some of the model parameters are shared entirely by all tasks. A simple example is that of a neural network with some shared hidden layers followed by task-specific output layers (e.g., Ramsundar et al., 2015). Note that this simple example assumes that the input features are the same across all tasks, but that does

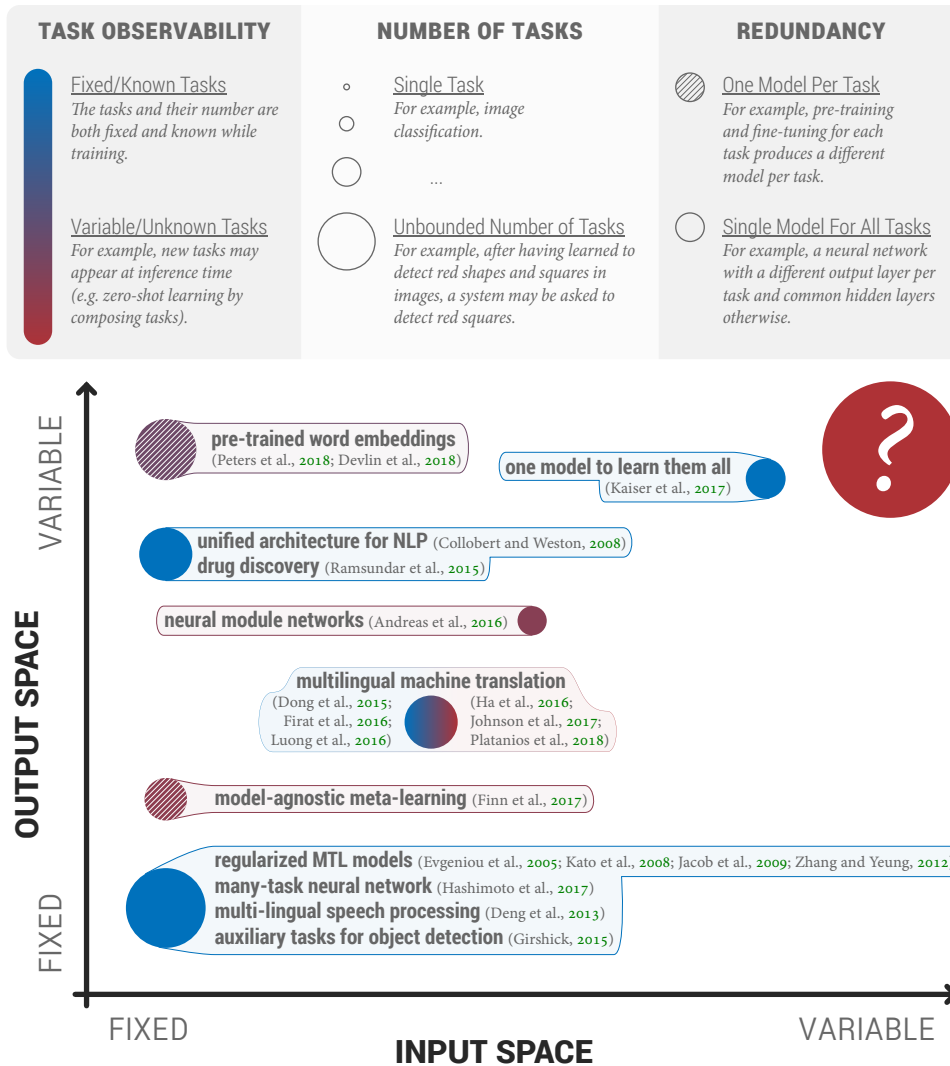


Figure 7.2: The current landscape of multi-task learning. The big red circle with the question mark on the top right represents what we hope to achieve with contextual parameter generation.

not need to be the case in general. There has been a lot of influential work in hard parameter sharing and even though we cannot include it all in this paragraph, we have included some of the most influential examples. Baxter (1997) was the first to provide theoretical results showing how hard parameter sharing greatly reduces the risk of overfitting. Hashimoto et al. (2016) proposed an approach that also takes into account a pre-defined hierarchical structure over the tasks they consider. Andreas et al. (2016) proposed sharing various low-level modules that are “wired together” differently for each task. Kaiser et al. (2017) used separate encoder and decoder networks for different data modalities that are shared across all tasks that utilize these modalities. This allowed them to handle tasks with different input modalities. Hard parameter sharing also includes methods where a single model is supervised with multiple

objectives in order to improve generalization (e.g., Girshick, 2015). Finally, other examples of hard parameter sharing include the work of Collobert and Weston (2008), Deng et al. (2013), Dong et al. (2015a), Firat et al. (2016a), Luong et al. (2016), and Johnson et al. (2017).

SOFT PARAMETER SHARING. The key idea in soft parameter sharing is that we use a separate model per task, but the parameters of these models are constrained to be similar to each other, to some extent. For example, Argyriou et al. (2007) assume that the models for each task share a small set of features. To this end, they propose a method for finding this common set of features by using a specific regularization term on the model parameters. Assuming all models have the same number of parameters, we can take their parameter vectors and stack them side-by-side to form a matrix so that each row corresponds to a feature and each column corresponds to a task. The authors then use the ℓ_1 norm of the per-row ℓ_2 norm of this matrix as their regularization term. Yuan and Lin (2006) replace the ℓ_2 norm with an ℓ_∞ norm. These approaches can be classified as a form of soft parameter sharing as each model ends up with its own set of parameters, but these parameters are constrained to be similar in some way. There also exist some theoretical results on the benefits of these block-sparse regularization methods (e.g., Lounici et al., 2009). However, there is evidence to suggest that these methods often underperform simpler per-task regularization approaches (e.g., Negahban and Wainwright, 2011). Jalali et al. (2010) propose a slightly modified version of these regularization schemes that attempts to avoid the aforementioned underperformance issues by mixing block-sparse regularization with per-task regularization. Other work employing similar methods includes the approaches of Duong et al. (2015) and Yang and Hospedales (2016). However, soft parameter sharing is an abstract idea that comes in many alternative forms. Evgeniou and Pontil (2004), Evgeniou et al. (2005), and Jacob et al. (2009) relax the constraint that some features are shared across all tasks by instead imposing a clustering constraint on the parameter vectors. In fact, multiple methods followed in the same direction, some even imposing Bayesian clustering priors, including work by Thrun and O’Sullivan (1996), Heskes (2000), Bakker and Heskes (2003), Lawrence and Platt (2004), Yu et al. (2005), Ando and Zhang (2005), Xue et al. (2007), Daumé (2009), Kim and Xing (2009), and Zhang and Yeung (2010). Online variants of some of these approaches were also proposed by Cavallanti et al. (2008) and Saha et al. (2011). Furthermore, soft parameter sharing has recently proven to be very successful in deep learning (e.g., Ruder et al., 2019). *Pre-training and fine-tuning* is one such example. Devlin et al. (2019) propose BERT, a large-scale natural language processing model that was pre-trained as a language model on a massive text corpus and then separately fine-tuned on a multitude tasks, achieving state-of-the-art performance in all of them. This is a form of soft parameter sharing as a single massive model is pre-trained and then copies of this model are instantiated and fine-tuned for each task. This common initialization has a large impact on performance and serves as a strong prior for the parameters of all task-specific models. A similar argument can be made for a lot of the recently successful *meta-learning* methods like the model-agnostic

meta-learning (MAML) algorithm by Finn et al., 2017. This algorithm also learns a common initialization for all task-specific models.

In Section 7.2, we introduce *contextual parameter generation (CPG)* which is inspired by all this work and offers a generalized framework under which we can obtain both hard and soft parameter sharing. However, before that we provide some insight as to when, how, and why multi-task learning works, in order to better motivate our work on CPG and better explain how it addresses several of the limitations of current work in multi-task learning.

7.1.3 When, How, and Why Does Multi-Task Learning Work?

Intuitively, we expect multi-task learning to help in settings where we have tasks that are somehow related and we have sufficient training data for some of them, but little or no training data at all for others. In such cases, we expect that we may be able to leverage the task relationships in order to share information between the “data-rich” and the “data-poor” tasks. A practical example for this setting is that of low-resource machine translation that is discussed in detail in the following chapter. On the other hand, in situations where we have an abundance of training data for all tasks it may be more beneficial to train separate models in isolation because multi-task learning always involves making assumptions about the task relationships that may not be true, and may thus hurt performance. In order to better understand when multi-task learning can help, we now discuss several empirical observations that Caruana (1995, 1997) made while tackling multiple real-world problems:

- Pooling versus Isolation: It has been observed that oftentimes, training using multiple objective functions together can help improve performance over training on each objective separately. Intuitively, one way to explain this is that some tasks can act as regularizers for other tasks, helping machine learning models achieve better generalization capabilities.
- Inputs versus Outputs: It has been observed that sometimes it may be interesting to convert certain quantities from being inputs to a machine learning model, to being target outputs. For example, let us consider a model that predicts how long a patient will stay in a hospital. We could use some of the patients test results as inputs to the model, but Caruana (1997) observed that it sometimes may be beneficial to convert these results to outputs that the model is trying to predict, along with the hospital stay duration. This is a very interesting observation and shows how converting a single-task problem to a multi-task problem may sometimes be beneficial in unexpected ways.
- Task Interference: Tasks may interfere in both positive and negative ways. We say that tasks interfere positively when models trained to perform them jointly do better than when trained to perform each one in isolation. However, the opposite effect has also been observed. Sometimes training models to perform multiple tasks jointly hurts their performance. We refer to this effect as *negative transfer* or *catastrophic interference* and it was first described by Sharkey and

Sharkey (1995). In fact, it is also one of the biggest challenges related to multi-task learning and we argue that it is the result of excessive parameter sharing. In order to resolve this issue we need methods that are able to trade-off between hard parameter sharing, soft parameter sharing, and no parameter sharing at all; the latter being useful for cases where the tasks may not be related at all. The method we describe in the next section tackles this issue directly.

- Task Decomposition: When learning complex tasks it may be beneficial to break them down into sub-tasks that can be directly supervised and learn all subtasks in a multi-task manner. Our work on coarse-to-fine curriculum learning (Stretcu et al., 2020) is a related example, as is the work by Abu-Mostafa (1990) on hints for neural networks.
- Attention Focusing: Multi-task learning can often help teach models where to focus their attention. Let us consider an example by Caruana (1997) to showcase this. Consider an autonomous vehicle. When learning to steer, models may often ignore lane markings because they are usually a small part of the input image. However, if a model learning to steer is also required to learn to recognize the lane markings, then it will learn to attend to that part of the input image. This may in turn help the model learn the steering task better.
- Task Learning Rates: When training a model on multiple tasks it is often the case that for some tasks training converges faster than for other tasks. This can result in overfitting as it is unclear how to stop training early for some tasks but not all, and this is necessary because we are training a single model for all of the tasks. Even though there have been some domain-specific attempts to tackle this problem (e.g., the work of Kendall et al., 2018, which also applies to the next paragraph), there are no elegant general-purpose workarounds. Most often researchers will either: (i) tune the learning rate for each task such that the objective for all tasks converge at around the same rate (although this can be computationally expensive as multiple training runs are needed), or (ii) save “snapshots” of the model when each task converges and then use a different “snapshot” for each task at inference time.
- Multiple Metrics: A common instance of hard parameter sharing is when we have a single model but we care about multiple performance metrics and can actually supervise this model with multiple signals. In this case, we may be optimizing multiple loss functions with respect to a single model. This can often help in practice, but care must be taken with respect to how we “balance” the different loss terms, as they may be at completely different scales and some may be more dominant than others during the different phases of training.
- Training Data Size: As mentioned in the beginning of this section, the amount of training data available for each task can significantly affect the benefits offered by multi-task learning approaches. The more training data that is available, the less significant the benefits of multi-task learning are.

From these observations and the discussion in the previous section, it is evident that a key challenge of multi-task learning is controlling how much information is shared between the tasks. For example, when tasks are very similar we want to transfer information from well-supervised tasks to the ones for which we have very limited or no supervision at all. However, when tasks are very dissimilar we want to avoid transferring information between them, as this could result in catastrophic interference. This is the trade-off that hard parameter sharing and soft parameter sharing methods face. In that case, information sharing is done through parameter sharing. In the next section, we present a novel approach to multi-task learning that is a generalization of both hard and soft parameter sharing and aims to enable better learning of this trade-off. Furthermore, this new approach is designed such that if we know a priori how some tasks are related, we can encode that information as part of the model that is being learned to solve them.

7.2 CONTEXTUAL PARAMETER GENERATION

Let us refer to parameterized functions as *networks*. Let us also denote a network by a lowercase English letter with a lowercase Greek letter subscript (e.g., f_θ), where the Greek letter refers to the network parameters. Therefore, given some input x , the network output is simply defined as:

$$y = f_\theta(x). \quad (7.1)$$

Most deep learning models can be seen as networks. For example, we can have a convolutional neural network (CNN) that takes images as input, transforms them using convolutional filters (i.e., parameters), and produces distributions over labels (e.g., cat or dog). Research in deep learning has resulted in multiple network architectures that can successfully learn to solve various problems, and that each make different assumptions about its input space. For example, CNNs assume that there is some periodical structure in the input space and are translation invariant, whereas recurrent neural networks (RNNs) assume that each part of an input sequence can be processed using the same network parameters.

Never-ending learning requires a system to be able to perform multiple tasks; perhaps even previously unseen tasks that can be formulated in terms of other previously learned tasks. This means that traditional multi-task neural network architectures that use a different output layer for each task (e.g., Caruana, 1997) cannot be used in this context. That is because the set of tasks the system is learning to perform is not known a priori, when the neural network architecture is chosen. This motivates us to treat tasks as separate inputs. More specifically, we assume that a description of the task the system is performing at each time is provided as an additional input to the system.

An additional benefit of feeding the task as an additional input to the system is that it enables *zero-shot learning*. Let us clarify this with a simple example. Consider a multi-task learning setting where the objective is to learn to convert between different temperature units. Specifically, we want our model to be able to convert temperature values from °C to °F and between any other combinations of °C, °F, and °K. In this

case, a task can be described as a pair of temperature units: the source temperature unit and the target temperature unit. If a typical multi-task learning system gets to observe training examples for the tasks ($^{\circ}\text{C}$, $^{\circ}\text{F}$) and ($^{\circ}\text{K}$, $^{\circ}\text{F}$), then it will not be able to generalize well for the ($^{\circ}\text{C}$, $^{\circ}\text{K}$) task. However, if the system is able to learn embeddings for the three separate temperature units and represent a task using a pair of such embeddings that is fed to the system as an additional input, then it is much more likely that it will be able to generalize well. As we shall show in the following paragraphs, feeding the task representation as an additional input to the system also has some disadvantages that led us to propose contextual parameter generation. In [Section 8.2](#) we present an extensive case study on the problem of multilingual machine translation, which generalizes the simple example of this paragraph.

We argue that, for most existing neural network architectures, it is hard or even impossible to encode assumptions about the contexts (e.g., tasks) in which they are used, to share information across these contexts, and to “personalize” them for each context, when that context is simply provided as an additional model input. As we discuss in the end of this section, this limitation could be attributed to the fact that most existing architectures are only able to represent additive interactions between their inputs. Previously, there has been some success in encoding this kind of assumptions using probabilistic graphical models (PGMs). When working with PGMs, researchers typically first define a prior probabilistic model over how the data observations are generated and then perform inference to obtain a posterior distribution over the model parameters and possibly also latent variables. These generative models are often hierarchical, meaning that the parameters of the distribution from which the observations are sampled, are also often sampled themselves from a higher-level distribution. This results in an interesting type of information sharing across all the different distributions, and has been behind many successful models, such as latent Dirichlet allocation (Blei et al., 2003) and hierarchical Dirichlet processes (Teh et al., 2006). There have been efforts to combine such approaches with neural networks (e.g., Tran et al., 2019), but they are often expensive and impractical for large scale problems. Furthermore, in order to make probabilistic inference tractable they often limit model expressivity.

This motivated us to develop a method called *contextual parameter generation* (CPG; Platanios et al., 2018). The core idea behind this method is that, given a network f_{θ} , instead of learning θ directly while training, we define it as:

$$\theta = g_{\phi}(c), \quad (7.2)$$

where c is a description of the context in which we are applying the model. For example, if we are encoding text written in English as part of a multilingual machine translation model, the context could simply be a one-hot encoding of the English language. The parameters we learn during training are just those of g_{ϕ} , which we refer to as the parameter generation network. This allows us to share information across instances of f_{θ} used in different contexts. While we previously had to learn and use different parameters for each context in which f_{θ} is used, they are now all generated as a function of the context. For example, instead of using different encoders for text

written in English and text written in German, we can now use one encoder and simply generate its parameters as a function of their language. Note that we can simply define g_ϕ as a lookup table over different contexts, and this would reduce to the previous setting in which there is no information sharing. However, the CPG formulation allows us to impose arbitrary information sharing structures by manipulating the functional form of the parameter generation network, g_ϕ . For example, we could learn embeddings for all language families and have all Romance language embeddings be defined as linear transforms of the corresponding Romance family embedding. When performing multi-task learning, we can think of each task as a context in which a network processes its inputs. Given a representation of this context, we can generate the parameters of a single universal network that is used for all tasks. The way in which contexts are defined and processed to generate parameters can thus allow for controlled information sharing across multiple tasks. We refer to networks that employ CPG as *contextualized networks*, and we let them optionally have some of their parameters be generated by a CPG component, and some be directly learned (e.g., we may not want to generate the parameters of a batch normalization layer using CPG). Contextualized networks also have better generalization properties than plain networks because they can be used with previously unseen contexts, as long as the new contexts can be composed out of previously seen contexts.

7.2.1 Feeding Context as an Additional Input

A natural question to ask is why could we not just feed the context as another network input. Here, we provide a few reasons to justify contextual parameter generation as a better alternative to this.

1. Structured Information Sharing: Similar to the motivation for PGMs described earlier in this section, CPG provides a structured way to share information across contexts. We shall see practical examples of this in the following chapter.
2. Additive Interactions: Without loss of generality, let us split the input x to a neural network in two parts, x_0 and x_1 . For example, assuming x is a vector, then x_0 and x_1 are vectors such that when concatenated, they form x . Most neural network architectures currently in use only allow for interactions of the following form:

$$y = f_\theta(h_{\phi_0}^0(x_0) + h_{\phi_1}^1(x_1)), \quad (7.3)$$

where f , h^0 , and h^1 are arbitrary functions, and y is the output of the neural network. For example, consider the following function types:

- Element-wise Operations: Functions that are applied to each element of a given tensor x , independently. Neural network activation functions are a common example (e.g., $f(x) = \tanh(x)$).
- Linear Transforms: Functions of the form $f(x) = Wx + b$, where x is a vector and W and b are parameters that are learned.
- Convolution Neural Networks (CNNs): Functions that are compositions of convolutions, element-wise operations, and linear transformations.

Note that convolutions are also linear transformations with a parameter sharing structure that is informed by an invariance assumption.

- Recurrent Neural Networks (RNNs): Functions that slice an input tensor along some dimension and then apply the same, often stateful, function on each slice. This function is most often defined as a composition of convolutions, element-wise operations, and linear transformations.

This form is very restrictive. For example, it cannot be used to represent simple if-then-else rules such as “if $x_0 = 2$, then $2x_1$ else $5x_1$.” This is especially important for multi-task learning because we can think of x_0 as the description of some task and we might want to condition on this task while processing the rest of the model inputs. This would be the case for a mixture of experts model, for example. In order to represent this kind of interaction we have to explicitly encode them in the neural network architecture. Ideally, we want the model to be able to learn these interactions on its own—if they are necessary—instead of having them be hardcoded as part of the model architecture. CPG in fact allows for multiplicative or even polynomial interactions between $h_{\phi_0}^0(x_0)$ and $h_{\phi_1}^1(x_1)$, which would allow us to represent if-then-else rules. Multiplicative interactions have also been shown to be useful in other settings (e.g., Lengerich et al., 2020). In fact, self-attention methods (Vaswani et al., 2017) have been very successful recently and they rely on a simple form of multiplicative interactions which can also be seen as a special instance of contextual parameter generation (the queries and the keys in self-attention determine the weights that will be used to multiply with the values).

3. Deployment: CPG allows us to generate a context-specific model and later use it without involving the parameter generation network. This can be beneficial for deployment. For example, if we only care about English-to-German translation due to an upcoming vacation trip, we could have Google Translate generate a translation model for this language pair and then store that model on a mobile device for offline use.
4. Modeling Assumptions: Different neural network architectures make different assumptions. For example, CNNs assume spatial invariance for the inputs (meaning that they contain repeating patterns in different locations). This assumption is unlikely to hold for an arbitrary context vector and so it is unreasonable to just concatenate an arbitrary context vector with an image and feed the result to a CNN. CPG avoids this problem because the architecture modification it entails does not affect the assumptions made by a network about the input data.

7.2.2 Related Work

Ha et al. (2018) are probably the first to introduce a similar idea to that of having one network (called a *hypernetwork*) generate the parameters of another. However, in their work, the input to the hypernetwork are structural features of the original network (e.g., layer size and index). Al-Shedivat et al. (2017) propose a related method where

a neural network generates the parameters of a linear model. Their focus is mostly on interpretability (i.e., knowing which features the network considers important). Dumoulin et al. (2018) provide a comprehensive review of some more related work from other fields, such as computer vision, that was published concurrently to our work. Furthermore, CPG is generic enough so that many existing methods can be formulated as CPG variants. One such example is model-agnostic meta-learning (MAML) by Finn et al. (2017), where a model is pre-trained over a large number of tasks and then fine-tuned on new tasks drawn from the same task distribution. In this case, the parameter generation network consists of taking a gradient descent step, using only the new task's data. Along a slightly different direction, Kang et al. (2011) propose clustering the tasks and learning a single shared model per task cluster. However, this approach is a bit too constraining as rarely can two tasks ever be considered exactly identical. Kumar and Daumé (2012) extend this approach by proposing a simple version with the parameter vectors being a linear combination of a small set of basis task vectors.

7.2.3 *Another Perspective on Parameter Generation*

Recently, neural architecture search (NAS; Elsken et al., 2019) has enabled machine learning practitioners to automate the tuning of neural network architectures. Empirically, there is strong evidence that a neural network's architecture strongly affects its generalization performance. However, designing architectures is most often done manually and doing it well requires a lot of experience. Therefore, NAS is an important area of research as it aims to automate this process. Interestingly, CPG can directly be used for architecture search. Consider a setting where we have an extremely high-capacity fully connected network, along with a parameter generator for this network that always generates sparse parameters (i.e., most of the neural network weights are set to zero). The parameter generator in this case can be thought of as an architecture generator and learning a good generator for a given problem amounts to solving a NAS problem. More generally, CPG is not just about parameter generation, but rather about function generation. Considering a large parametric function class and generating parameters for functions in this class, which is what CPG does, is a special instance of function generation which is, in its own right, very powerful.

In the following chapter, we present several empirical case studies showcasing the strengths of the CPG abstraction. Then, in [Section 10.2](#) we present some remaining open questions related to CPG that are outside the scope of this thesis.

CASE STUDIES FOR CONTEXTUAL PARAMETER GENERATION

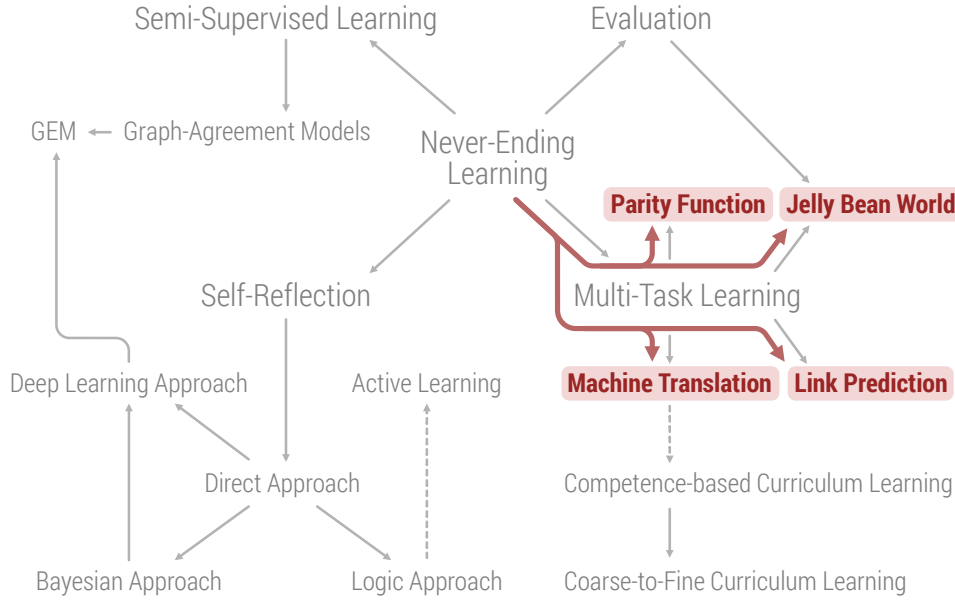


Figure 8.1: Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in [Section 1.4](#).

In the previous chapter we proposed a novel abstraction, *contextual parameter generation*, that enables large-scale multi-task learning. In this chapter, we present multiple diverse case studies that showcase the power and usefulness of contextual parameter generation. The first case study is aimed at providing some insight into the increased expressive power of contextual parameter generation (discussed in [Section 7.2.1](#)). The rest of the case studies in this chapter are focused on multi-task learning and show how contextual parameter generation establishes new state-of-the-art performance for multiple problems.

8.1 CASE STUDY #1: PARITY FUNCTION

The n -variable parity function is defined as the Boolean function $p^n : \{0, 1\}^n \mapsto \{0, 1\}$ with the property that $p^n(x) = 1$ if and only if the number of ones in the vector $x \in \{0, 1\}^n$ is odd. Formally, we define the parity function as:

$$p^n(x) = x_1 \oplus x_2 \oplus \cdots \oplus x_n, \quad (8.1)$$

The “exclusive-or” logical operator evaluates to 1 when its two arguments have different values (i.e., one of them is equal to 0 and the other to 1), and to 0 otherwise.

where $\oplus$ denotes the “exclusive-or” logical operator (also known as XOR). We refer to the problem of learning the parity function as the parity problem. Learning the XOR function is the simplest possible parity problem (i.e., where $n = 2$) and we shall refer to it as the XOR problem. In fact, this problem was used by Minsky and Papert (1969) to test some of the first learning algorithms. It was also later shown by Rumelhart et al. (1985) that it can be solved by a simple feed-forward neural network with a single hidden layer containing 2 units. Rumelhart et al. (1985) also showed that the parity problem for a fixed value of n can be solved by a similar network containing n units. We consider the harder problem of learning the parity function for bit sequences of arbitrary lengths (i.e., n is not fixed), while training only on short sequences. This will allow us to test the generalization capabilities of our models. In order for our models to be able to handle sequences of arbitrary lengths, we shall only consider *recurrent neural networks (RNNs)* for this case study. In the following sections, we provide some background on RNNs, discuss their limitations, and show how contextual parameter generation can address some of these limitations. We also provide some interesting relationships between contextual parameter generation and existing work in *multiplicative RNNs* and even *hidden Markov models (HMMs)*.

8.1.1 Recurrent Neural Networks

A recurrent neural network is an adaptation of feed-forward neural networks that enables modeling sequential data. The core idea behind RNNs can be traced back to the work of Rumelhart et al. (1986). The input to an RNN is typically a sequence of data points, $\mathbf{x} = \{\mathbf{x}_1, \dots, \mathbf{x}_T\}$, and the RNN goes through each element in that sequence, processes it, and updates some hidden state. This hidden state can then be used to make predictions for downstream tasks. The hidden state is typically a high-dimensional vector and the function that updates it is non-linear (often a feed-forward network), resulting in RNNs being able to learn powerful representations. For example, given a sequence of input vectors $\{\mathbf{x}_1, \dots, \mathbf{x}_T\}$, the standard RNN computes a sequence of hidden states $\{\mathbf{h}_1, \dots, \mathbf{h}_T\}$ and a sequence of outputs $\{\mathbf{o}_1, \dots, \mathbf{o}_T\}$, by iterating over the following equations for $t = 1, \dots, T$:

$$\mathbf{h}_t = \tanh(W_h \mathbf{x}_t + U_h \mathbf{h}_{t-1} + \mathbf{b}_h), \quad (8.2)$$

$$\mathbf{o}_t = W_o \mathbf{h}_t + \mathbf{b}_o, \quad (8.3)$$

where $\mathbf{h}_0$ is initialized to a pre-specified value (e.g., zeros), and W_h , U_h , $\mathbf{b}_h$, W_o , and $\mathbf{b}_o$ are model parameters that are learned during training. Even though it may initially seem trivial to train RNNs using gradient descent methods, it has been shown that training over long sequences is in fact hard (see e.g., Bengio et al., 1994). The gradients of the model parameters tend to explode or vanish numerically thus often making training impossible. Furthermore, the standard RNN model is generally not able to “remember” information over long spans of the input sequences.¹ There have been multiple proposals on how to address this issue (e.g., echo state networks by

¹ This can sometimes be avoided by using more expensive Hessian-free optimizers (Martens and Sutskever, 2011) to train RNNs.

Jaeger and Haas, 2004), but the most successful and dominant one is arguably that of Hochreiter and Schmidhuber (1997), who proposed long short-term memory (LSTM) networks.

LSTMs are perhaps the most popular RNN variant and have been used successfully in a multitude of applications (e.g., Graves and Schmidhuber, 2009; Ha et al., 2016; Luong et al., 2016; Johnson et al., 2017; Ha et al., 2018). In contrast to the standard RNN, the hidden state of an LSTM consists of two components: (i) h_t , similar to the standard RNN, and (ii) c_t , a cell that operates as a longer term memory. Formally, it is defined by the following equations:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \quad \text{FORGET GATE} \quad (8.4)$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \quad \text{INPUT GATE} \quad (8.5)$$

$$u_t = \tanh(W_u x_t + U_u h_{t-1} + b_u), \quad \text{UPDATE GATE} \quad (8.6)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \quad \text{OUTPUT GATE} \quad (8.7)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot u_t, \quad \text{CELL STATE} \quad (8.8)$$

$$h_t = o_t \odot \tanh(c_t), \quad \text{HIDDEN STATE} \quad (8.9)$$

where $\odot$ represents the Hadamard (i.e., element-wise) product, and $W_f, U_f, b_f, W_i, U_i, b_i, W_u, U_u, b_u, W_o, U_o, b_o$ are model parameters that are learned during training. At a high level, the forget gate decides what to “erase” from the cell state and the update gate decides how to update the cell state based on the current input that has been processed by the input gate. LSTMs have been shown to be highly successful in multiple applications and researchers have also proposed multiple variations to the standard LSTM formulation (e.g., Gers and Schmidhuber, 2000; Cho et al., 2014; Koutnik et al., 2014; Yao et al., 2015). However, multiple comparisons have shown that all variants perform pretty much equivalently (see e.g., Jozefowicz et al., 2015; Greff et al., 2016). Therefore, the rest of this section will focus on standard LSTMs.

Even though LSTMs have been used successfully for many applications, they still suffer from certain problems. As we show in [Section 8.1.4](#), LSTMs can sometimes fail to learn even very simple functions such as the parity function. The main issue is that by increasing the number of hidden units in an LSTM we can make it able to train on long sequences, but the resulting network will often not be able to generalize to longer sequences than what it was trained on. We already showed in [Section 7.2.1](#) that most current neural network architectures are only able to represent additive interactions between their inputs. LSTMs are not different in that they can only represent additive interactions between their hidden states and input vectors. In addition to this, it has been previously observed by Sutskever et al. (2011) that the performance of recurrent neural networks can be significantly improved by allowing for multiplicative interactions between the hidden states and input vectors.

8.1.2 Multiplicative Recurrent Neural Networks

In order to address the limitation discussed in the previous section, Sutskever et al. (2011) proposed factorizing the standard RNN of Equations 8.2 and 8.3 as follows:

$$\mathbf{m}_t = \mathbf{W}_m \mathbf{x}_t \odot \mathbf{U}_m \mathbf{h}_{t-1}, \quad (8.10)$$

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{m}_t + \mathbf{b}_h), \quad (8.11)$$

$$\mathbf{o}_t = \mathbf{W}_o \mathbf{h}_t + \mathbf{b}_o. \quad (8.12)$$

This is called the *multiplicative RNN* as the hidden state and input vector now interact multiplicatively, as shown in Equation 8.10. Multiplicative RNNs were able to improve the performance of standard RNNs on the character-level language modeling task (Sutskever et al., 2011; Mikolov et al., 2012). However, Cooijmans et al. (2016) showed that they still fall short of the more popular LSTM networks. To this end, Krause et al. (2016) proposed to apply the same modification on the standard LSTM (shown in Equations 8.4 to 8.9) resulting in the *multiplicative LSTM* formulation:

$$\mathbf{m}_t = \mathbf{W}_m \mathbf{x}_t \odot \mathbf{U}_m \mathbf{h}_{t-1}, \quad \text{MULTIPLICATIVE TERM} \quad (8.13)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{m}_t + \mathbf{b}_f), \quad \text{FORGET GATE} \quad (8.14)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{m}_t + \mathbf{b}_i), \quad \text{INPUT GATE} \quad (8.15)$$

$$\mathbf{u}_t = \tanh(\mathbf{W}_u \mathbf{x}_t + \mathbf{U}_u \mathbf{m}_t + \mathbf{b}_u), \quad \text{UPDATE GATE} \quad (8.16)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{m}_t + \mathbf{b}_o), \quad \text{OUTPUT GATE} \quad (8.17)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t, \quad \text{CELL STATE} \quad (8.18)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \quad \text{HIDDEN STATE} \quad (8.19)$$

On a similar note, Wu et al. (2016b) proposed *multiplicative integration*, which can be thought of as an alternative formulation of multiplicative RNNs. The core idea relies on the fact that the standard RNN and the standard LSTM have a common aspect: the input $\mathbf{x}_t$ and the hidden state $\mathbf{h}_{t-1}$ are always combined using additive functions of the form $\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}$. In fact, most existing RNN designs rely on update functions of this form. Wu et al. (2016b) proposed using the Hadamard (i.e., element-wise) product instead of the addition operation, $\mathbf{W}\mathbf{x}_t \odot \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}$, and showed how this simple change helps alleviate the exploding and vanishing gradients problem. The authors applied this modification to both the standard RNN and the standard LSTM networks, and they show how, under certain conditions, their standard RNN variant can be seen as a nonlinear extension of hidden Markov models (Rabiner, 1986).

8.1.3 Contextual Recurrent Neural Networks

A recurrent neural network update function should be “specializable” based on the current hidden state. Ideally, we imagine a setting where we can generate a different update function for different hidden states. This would allow representing arbitrary ways of conditioning on the current hidden state of the recurrent network. Contextual parameter generation (CPG) seems like a natural fit to this situation as we can see how

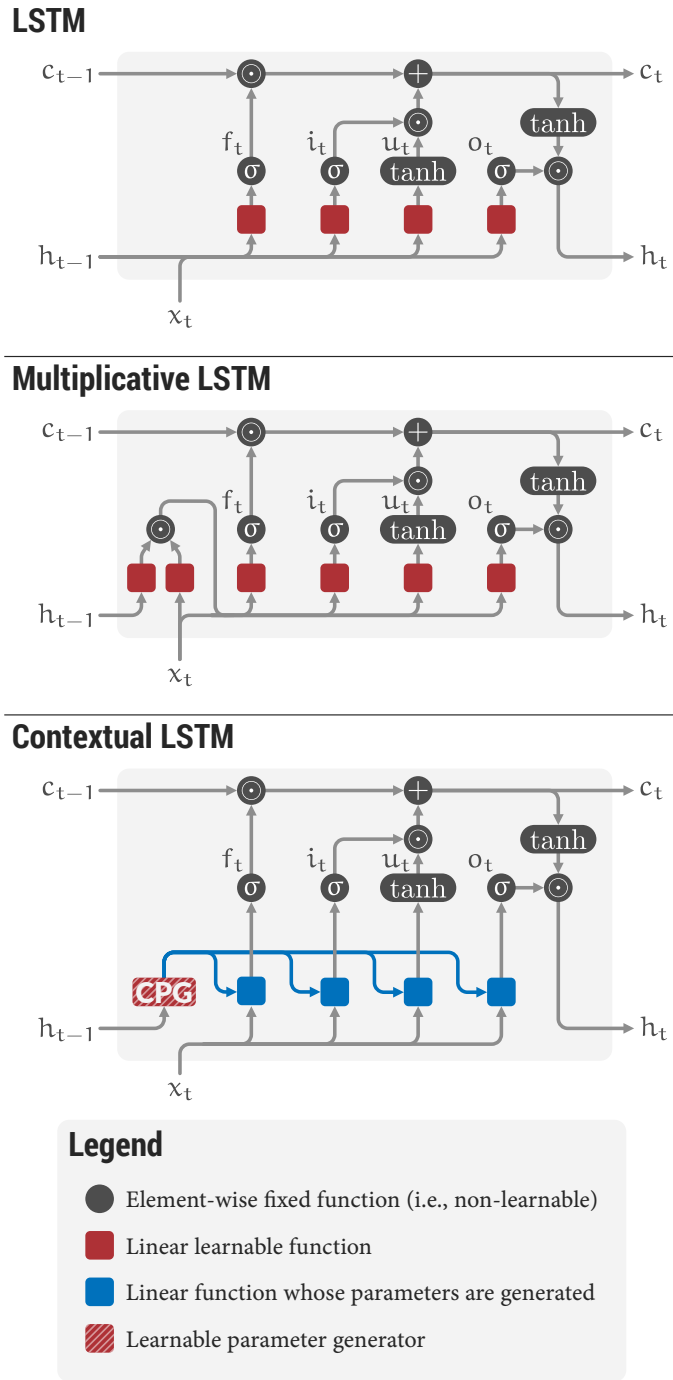


Figure 8.2: Illustration of the LSTM variants that we use in our parity function case study.

generating the parameters of the update function using the current hidden state as the context is equivalent to the ideal setting we just described. Making the connection to probabilistic graphical models, contextual RNNs can be thought of as an analogue to *hidden Markov models* (HMMs; Rabiner, 1986). We now present a contextual variant

of LSTMs, as a special instance of contextual RNNs. In the following section we show how a contextual LSTM can learn the parity function with significantly better generalization accuracy than both plain LSTMs and multiplicative LSTMs.

Let us consider an LSTM and stack all its parameters, $W_f, U_f, b_f, W_i, U_i, b_i, W_u, U_u, b_u, W_o, U_o,$ and b_o , in a single vector θ . The contextual LSTM considers these parameters a function of h_{t-1} instead of making them directly learnable. Here we shall simply consider a linear function:

$$\theta = W_{\text{generator}} h_{t-1}. \quad (8.20)$$

We refer to this model as the *contextual LSTM*. An illustration of the different kinds of RNNs that we have presented in this section is shown in [Figure 8.2](#).

8.1.4 Experiments

In order to evaluate the power of contextual LSTMs, we perform an experimental study using the parity function problem that was described in the beginning of this section. Specifically, during training all networks are presented with bit sequences of length 1, 2, or 3, sampled uniformly at random, along with the parity value for each sequence. After training, the networks are evaluated by computing their accuracy on random bit sequences of length 1 to 100. For each model we use three instances, each with a different number of parameters, so we can better evaluate the generalization capabilities of each one, as we make them increasingly more prone to overfitting. Specifically, we evaluate on models with 100, 1,000, and 16,000 parameters, where the number of parameters is controlled by setting the hidden state size appropriately.

Our results are shown in [Figure 8.3](#). The first observation we make is that the plain LSTM model is completely unable to generalize to sequences of length larger than 3, which is the length of the longest training sequence. Furthermore, we observe that increasing the model size (i.e., the number of parameters) generally results in worse generalization performance, which is expected. The multiplicative LSTM is able to improve the generalization performance of the plain LSTM by a big margin, but the contextual LSTM outperforms all alternative methods by a significant margin. In fact, we observe that over all model sizes the contextual LSTM is able to generalize reasonably well and achieve very good performance even when tested on bit sequences with 100 bits. In summary, the addition of multiplicative units makes the parity function learnable and contextualization makes our learners better able to generalize. The latter is also partially true because the functional form of the contextual LSTM is closer to the functional form of the true underlying function (i.e., the parity function).

8.1.5 Key Takeaways

This case study offers some initial evidence for the strength of the CPG abstraction. An interesting byproduct of this section is a new RNN cell, the contextual LSTM cell, that could prove very powerful in practice. However, evaluating this cell extensively is beyond the scope of this thesis. Next, we focus on two large scale case studies on

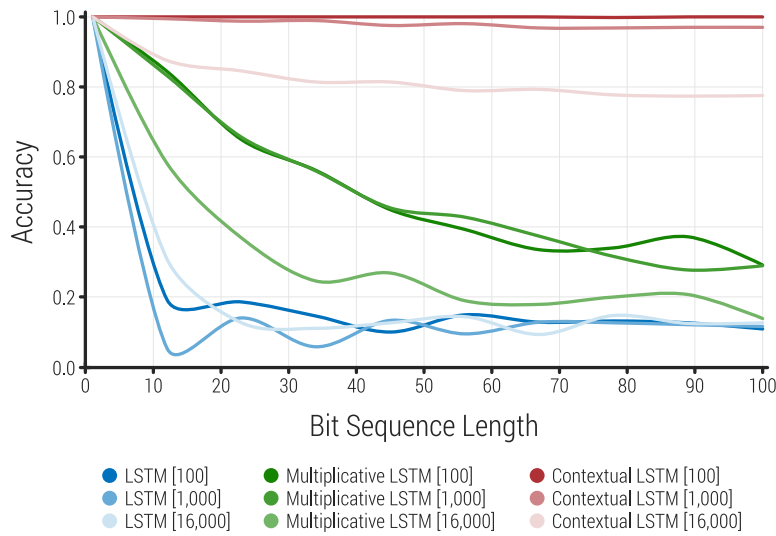


Figure 8.3: Results for the parity function experiments.

real world data which showcase how CPG can be used to perform multi-task learning more effectively than alternative approaches.

8.2 CASE STUDY #2: MACHINE TRANSLATION

An interesting and challenging multi-task learning problem is that of *multilingual machine translation*. This is because the tasks of translating between different pairs of languages are not independent and a lot of information can be shared between them. However, as we shall see, using a single model to translate between arbitrary pairs of languages does not perform well in practice. This makes multilingual machine translation a great setting in which to try and apply our contextual parameter generation abstraction. It is also the first setting in which we actually applied contextual parameter generation, and one in which we obtained some very encouraging results. We first present some background on multilingual machine translation. We then propose a simple modification to existing neural machine translation (NMT) models that enables using a single *universal* model to translate between multiple languages while allowing for language specific parameterization, and that can also be used for *domain adaptation*. Our approach requires no changes to the model architecture of a standard NMT system, but instead introduces a new component, the *contextual parameter generator* (CPG), that generates the parameters of the system (e.g., weights of a neural network). This parameter generator accepts source and target language embeddings as input, and generates the parameters for the encoder and the decoder, respectively. The rest of the model remains unchanged and is shared across all languages. We show how this simple modification enables the system to use monolingual data for training and also perform *zero-shot* translation. We further show it is able to surpass state-of-the-art performance for both the IWSLT-15 and IWSLT-17 datasets and

that the learned language embeddings are able to uncover interesting relationships between languages.²

8.2.1 Neural Machine Translation

Neural Machine Translation (NMT) directly models the mapping of a source language to a target language without any need for training or tuning any component of the system separately. This has led to a rapid progress in NMT and its successful adoption in many large-scale settings (Crego et al., 2016; Wu et al., 2016a). A typical NMT system comprises of an encoder, a decoder, and an attention mechanism (Bahdanau et al., 2015). The encoder maps the source sentence to a fixed-size vector representation in the case of the basic encoder-decoder framework (Sutskever et al., 2014), or a set of vectors in the attention-based variants. The decoder then uses the encoded representation to generate the target sentence word-by-word. The encoder-decoder abstraction makes it conceptually feasible to build a system that maps any source sentence in any language to a vector representation, and then decodes this representation into any target language. Thus, various approaches have been proposed to extend this abstraction to multilingual MT (Dong et al., 2015b; Firat et al., 2016a; Ha et al., 2016; Luong et al., 2016; Johnson et al., 2017).

Prior work in multilingual NMT can be broadly categorized into two paradigms. The first, *universal NMT* (Ha et al., 2016; Johnson et al., 2017), uses a single model for all languages. Universal NMT lacks any language-specific parameterization, which is an oversimplification and detrimental when we have very different languages and limited amounts of training data. As verified by our experiments, the method of Johnson et al. (2017) suffers from high sample complexity and thus underperforms in limited data settings. The universal model proposed by Ha et al. (2016) requires a new coding scheme for the input sentences which results in large vocabulary sizes that are difficult to scale. The second paradigm, *per-language encoder-decoder* (Firat et al., 2016a; Luong et al., 2016), uses separate encoders and decoders for each language. This does not allow for sharing of information across languages, which can result in overparameterization and can be detrimental when the languages are similar.

In this section, we strike a balance between these two approaches, proposing a model that has the ability to learn parameters separately for each language, but also share information between similar languages. We propose using a new *contextual parameter generator* (CPG) which (a) generalizes all of these methods, and (b) mitigates the aforementioned issues of *universal* and *per-language encoder-decoder* systems. It learns language embeddings as a context for translation and uses them to generate the parameters of a shared translation model for all language pairs. Thus, it provides these models the ability to learn parameters separately for each language, but also share information between similar languages. The parameter generator is general and allows any existing NMT model to be enhanced in this way. In addition, it has the following desirable features:

² The work we present in this section has been published in (Platanios et al., 2018).

1. Simple: Similar to the methods of Johnson et al. (2017) and Ha et al. (2016), and in contrast to those of Luong et al. (2016) and Firat et al. (2016a), it can be applied to most existing NMT systems with a minor modification, and it is able to accommodate attention layers seamlessly.
2. Multilingual: It enables multilingual translation using the same single model as before (i.e., the pairwise translation model).
3. Semi-supervised: It can use monolingual data.
4. Scalable: It reduces the number of parameters by employing extensive, yet controllable, sharing across languages, thus mitigating the need for large amounts of training data, similar to the method of Johnson et al. (2017). It also allows for the decoupling of languages, avoiding the need for a large shared vocabulary, same as for the method of Ha et al. (2016).
5. Adaptable: It can adapt to support new languages, without complete retraining.
6. State-of-the-art: It achieves better performance than pairwise NMT models as well as the universal model of Johnson et al. (2017). In fact, it surpasses state-of-the-art performance.

In order to provide the necessary background, we now define the multi-lingual NMT setting and then introduce a modular framework that can be used to define and describe most existing NMT systems. This will help us distill previous contributions and introduce ours.

8.2.1.1 *Setting*

We assume that we have a set of source languages S and a set of target languages T . The total number of languages is $L = |S \cup T|$. We also assume we have a set of $C \leq |S| \times |T|$ pairwise parallel corpora, $\{P_1, \dots, P_C\}$, each of which contains a set of sentence pairs for a single source-target language combination. The goal of multilingual NMT is to build a model that, when trained using the provided parallel corpora, can learn to translate well between any pair of languages in $S \times T$. The majority of related work only considers pairwise NMT where $|S| = |T| = 1$.

8.2.1.2 *NMT Modules*

Most NMT systems can be decomposed to the following modules, which are also illustrated in Figure 8.4:

PREPROCESSING PIPELINE. The data preprocessing pipeline handles tokenization, cleaning, normalizing the text data and building a *vocabulary*, i.e., a two-way mapping between preprocessed sentences and sequences of word indices that will be used for the translation. A commonly used approach for defining the vocabulary is to use byte-pair encoding (BPE) which generates subword unit vocabularies (Sennrich et al., 2016b) and eliminates out-of-vocabulary words, often resulting in better translation quality.

ENCODER/DECODER. The *encoder* takes in indexed source language sentences and produces an intermediate representation that can later be used by a *decoder* to generate

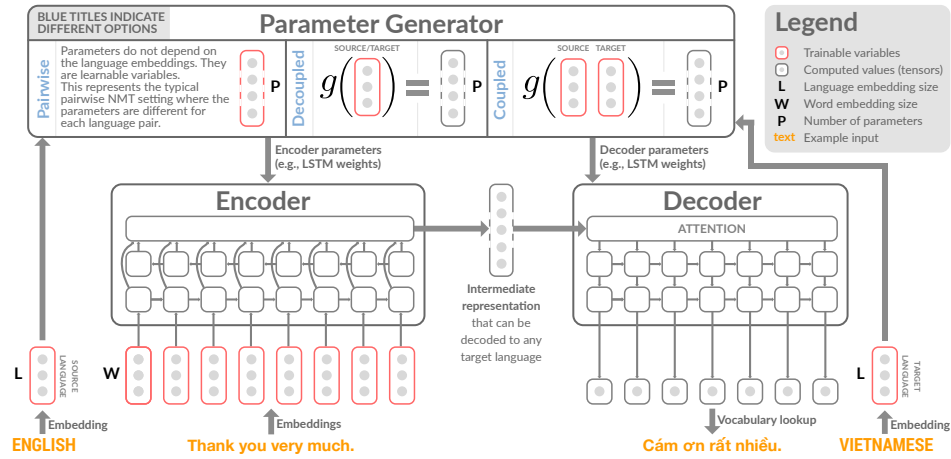


Figure 8.4: Overview of an NMT system, under our modular framework. Our main contribution lies in the parameter generator module (i.e., coupled or decoupled—each of the boxes with blue titles is a separate option). Note that g denotes a parameter generator network. In our experiments, we consider linear forms for this network. However, our contribution does not depend on the choices made regarding the rest of the modules; we could still use our parameter generator with different architectures for the encoder and the decoder, as well as with different kinds of vocabularies.

sentences in a target language. Generally, we can think of the encoder as a function $f^{(\text{enc})}$ parameterized by $\theta^{(\text{enc})}$. Similarly, we can think of the decoder as another function $f^{(\text{dec})}$ parameterized by $\theta^{(\text{dec})}$. The goal of learning to translate can then be defined as finding the values of $\theta^{(\text{enc})}$ and $\theta^{(\text{dec})}$ that result in the best translations. A large amount of previous work proposes novel designs for the encoder/decoder module. For example, using attention over the input sequence while decoding (Bahdanau et al., 2015; Luong et al., 2015) provides significant gains in translation performance.³

PARAMETER GENERATOR. All modules defined so far have previously been used when describing NMT systems and are thus easy to conceptualize. However, in previous work, most models are trained for a given language pair and it is not trivial to extend them to work for multiple pairs of languages. We introduce here the concept of a *parameter generator* for machine translation, which makes it easy to define and describe multilingual NMT systems. This module is responsible for generating $\theta^{(\text{enc})}$ and $\theta^{(\text{dec})}$ for any given source and target language. Different parameter generators result in different numbers of learnable parameters and can thus be used to control information sharing between languages. Next, we describe related work in terms of our parameter generator (an illustration is also shown in Figure 8.5):

- **Pairwise:** In the simple and commonly used pairwise NMT setting (Crego et al., 2016; Wu et al., 2016a), the parameter generator would generate separate

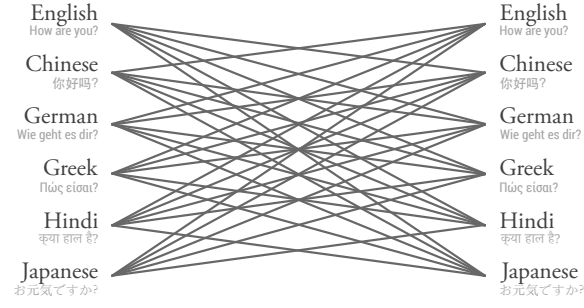
³ Note that depending on the vocabulary that is used and on whether it is one shared vocabulary across all languages, or one vocabulary per language, the output projection layer of the decoder (which produces probabilities over words) may be language dependent, or common across all languages. In our experiments we use separate vocabularies and thus this layer is language-dependent.

- parameters, $\theta^{(\text{enc})}$ and $\theta^{(\text{dec})}$, for each pair of source-target languages. This results in no parameter sharing across languages and thus $\mathcal{O}(\text{ST})$ parameters.
- Per-Language: In the case of Dong et al. (2015a), Luong et al. (2016) and Firat et al. (2016a), the parameter generator would generate separate encoder parameters $\theta^{(\text{enc})}$, for each source language, and separate decoder parameters $\theta^{(\text{dec})}$, for each target language. This leads to a reduction in the number of learnable parameters for multilingual NMT from $\mathcal{O}(\text{ST})$ to $\mathcal{O}(\text{S} + \text{T})$. Dong et al. (2015b) train multiple models as a one-to-many multilingual NMT system that translates from one source language to multiple target languages. On the other hand, Luong et al. (2016) and Firat et al. (2016a) perform many-to-many translation. However, Luong et al. (2016) only report results for a single language pair and do not attempt multilingual translation. Firat et al. (2016a) propose an attention mechanism that is shared across all language pairs. We generalize these ideas with the parameter generator network which is described later.
 - Universal: Ha et al. (2016) and Johnson et al. (2017) propose using a single common set of encoder-decoder parameters for all language pairs. While Ha et al. (2016) embed words in a common semantic space across languages, Johnson et al. (2017) learn language embeddings that are in the same space as the word embeddings. Here, the parameter generator would provide the same parameters $\theta^{(\text{enc})}$ and $\theta^{(\text{dec})}$ for all language pairs. It would also create and keep track of learnable variables representing language embeddings that are prepended to the encoder input sequence. As we observe in our experiments (discussed in Section 8.2.3), this system fails to perform well when the training data is limited. Finally, we believe that embedding languages in the same space as words is not intuitive. In our approach, languages are embedded in a separate space.

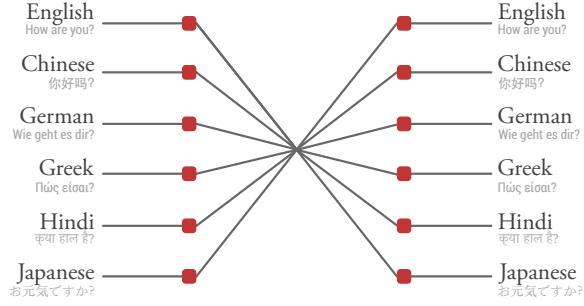
In contrast to all these related systems, we provide a simple, efficient, yet effective alternative—a parameter generator for multilingual NMT that enables semi-supervised and zero-shot learning. We also learn language embeddings, similar to Johnson et al. (2017), but in our case they are separate from the word embeddings and are treated as a *context* for the translation. This notion of *context* is used to define parameter sharing across various encoders and decoders, as discussed in the previous chapter.

8.2.2 Contextual Parameter Generation

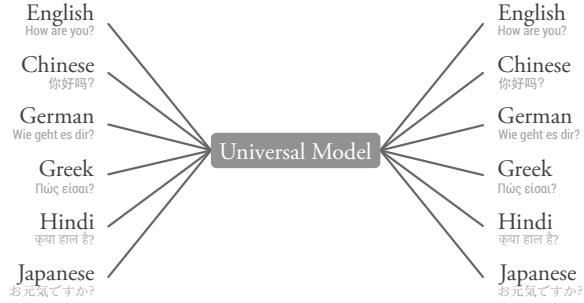
We propose to use contextual parameter generators as a new way to share information across different languages and control the amount of sharing. Let us denote the source language for a given sentence pair by ℓ_s and the target language by ℓ_t . Then, when using the contextual parameter generator, the parameters of the encoder are defined as $\theta^{(\text{enc})} = g^{(\text{enc})}(\mathbf{l}_s)$, for some function $g^{(\text{enc})}$, where $\mathbf{l}_s$ denotes an embedding for the source language ℓ_s . Similarly, the parameters of the decoder are defined as $\theta^{(\text{dec})} = g^{(\text{dec})}(\mathbf{l}_t)$ for some function $g^{(\text{dec})}$, where $\mathbf{l}_t$ denotes an embedding for the target language ℓ_t . Our generic formulation does not impose any constraints on the functional form of $g^{(\text{enc})}$ and $g^{(\text{dec})}$. In this case, we can think of the source language ℓ_s as a context for the encoder. The parameters of the encoder depend on



(a) Pairwise models.



(b) Per-language encoder/decoder.



(c) Universal model.

Figure 8.5: Illustration of the existing approaches to multilingual NMT.

its context, but its architecture is common across all contexts. We can make a similar argument for the decoder, and that is where the name of this parameter generator comes from. We can even go a step further and have a parameter generator that defines $\theta^{(\text{enc})} = g^{(\text{enc})}(l_s, l_t)$ and $\theta^{(\text{dec})} = g^{(\text{dec})}(l_s, l_t)$, thus coupling the encoding and decoding stages for a given language pair. In our experiments we stick to the previous *decoupled* form, because unlike the method of Johnson et al. (2017), it has the potential to lead to a common *interlingua*. Concretely, because the encoding and decoding stages are decoupled, the encoder is not aware of the target language while encoding sentences. Thus, we can take an encoded intermediate representation of a sentence and translate it to any target language. This is because, in this case, the intermediate representation is independent of any target language. This makes for a stronger argument that the intermediate representation produced by our encoder

could be approaching a universal interlingua, more so than methods that are aware of the target language when they encode sentences.

8.2.2.1 Parameter Generator Network

We refer to the functions $g^{(\text{enc})}$ and $g^{(\text{dec})}$ as *parameter generator networks*. Even though our proposed NMT framework does not rely on a specific choice for $g^{(\text{enc})}$ and $g^{(\text{dec})}$, here we describe the functional form we use in our experiments. Our goal is to provide a simple form that works, and for which we can reason about. For this reason, we decided to define the parameter generator networks as simple linear transforms, similar to the factored adaptation model of Michel and Neubig (2018), which was only applied to the bias terms of the output softmax:

$$g^{(\text{enc})}(l_s) = W^{(\text{enc})}l_s, \quad (8.21)$$

$$g^{(\text{dec})}(l_t) = W^{(\text{dec})}l_t, \quad (8.22)$$

where $l_s, l_t \in \mathbb{R}^M$, $W^{(\text{enc})} \in \mathbb{R}^{P^{(\text{enc})} \times M}$, $W^{(\text{dec})} \in \mathbb{R}^{P^{(\text{dec})} \times M}$, M is the language embedding size, $P^{(\text{enc})}$ is the number of parameters of the encoder, and $P^{(\text{dec})}$ is the number of parameters of the decoder. Another way to interpret this model is that it imposes a low-rank constraint on the parameters. As opposed to our approach, in the base case of using multiple pairwise models to perform multilingual translation, each model has $P = P^{(\text{enc})} + P^{(\text{dec})}$ learnable parameters for its encoder and decoder. Given that the models are pairwise, for L languages we have a total of $L(L - 1)$ learnable parameter vectors of size P . On the other hand, using our contextual parameter generator we have a total of L vectors of size M (one for each language), and a single matrix of size $P \times M$. Then, the parameters of the encoder and the decoder, for a single language pair, are defined as a linear combination of the matrix columns.

CONTROLLED PARAMETER SHARING. We can further control parameter sharing by observing that the encoder/decoder parameters often have some “natural grouping.” For example, in the case of recurrent neural networks we may have multiple weight matrices, one for each layer, as well as attention-related parameters. Based on this observation we now propose a way to control how much information is shared across languages. The language embeddings need to represent all of the language-specific information and thus may need to be large. However, when computing the parameters of each group, only a small part of that information is relevant. Let $\theta^{(\text{enc})} = \{\theta_j^{(\text{enc})}\}_{j=1}^G$ and $\theta_j^{(\text{enc})} \in \mathbb{R}^{P_j^{(\text{enc})}}$, where G denotes the number of groups. Then, we define:

$$\theta_j^{(\text{enc})} \triangleq W_j^{(\text{enc})}P_j^{(\text{enc})}l_s, \quad (8.23)$$

where $W_j^{(\text{enc})} \in \mathbb{R}^{P_j^{(\text{enc})} \times M'}$ and $P_j^{(\text{enc})} \in \mathbb{R}^{M' \times M}$, with $M' < M$, and similarly for the decoder parameters. We can see now that $P_j^{(\text{enc})}$ is used to extract the relevant information (size M') for parameter group j , from the larger language embedding (size M). This allows us to control parameter sharing across languages in the following way: if we want to increase the number of per-language parameters (i.e., the language embedding size) we can increase M while keeping M' small enough so that the total

number of parameters does not explode. This would not have been possible without the proposed low-rank approximation for $W^{(\text{enc})}$, that uses the parameter grouping information.

ALTERNATIVE OPTIONS. Given that our proposed approach does not depend on the specific choice of the parameter generator network, it might be interesting to design models that use side-information about the languages that are being used (such as linguistic information about language families and hierarchies). This is outside the scope of this thesis, but may be an interesting future direction.

8.2.2.2 *Semi-Supervised and Zero-Shot Learning*

The proposed parameter generator also enables semi-supervised learning via back-translation. Concretely, monolingual data can be used to train the shared encoder/decoder networks to translate a sentence from some language to itself.⁴ This is possible and can help learning because of the fact that many of the learnable parameters are shared across languages. Furthermore, zero-shot translation, where the model translates between language pairs for which it has seen no explicit training data, is also possible. This is because the same per-language parameters are used to translate to and from a given language, irrespective of the language at the other end. Therefore, as long as we train our model using some language pairs that involve a given language, it is possible to learn to translate in any direction involving that language.

8.2.2.3 *Domain Adaptation*

Let us assume that we have trained a model using data for some set of languages, $\ell_1, \ell_2, \dots, \ell_m$. If we obtain data for some new language ℓ_n , we do not have to retrain the whole model from scratch. In fact, we can fix the parameters that are shared across all languages and only learn the embedding for the new language (along with the relevant word embeddings if not using a shared vocabulary). Assuming that we had a sufficient number of languages in the beginning, this may allow us to obtain reasonable translation performance for the new language, with a minimal amount of training.⁵ Furthermore, other types of “domain” information can be encoded as part of the context as well. For example, the context can also contain information about the person speaking or writing, which can be used to perform extreme adaptation for personalized translation (Michel and Neubig, 2018).

8.2.2.4 *Number of Parameters*

For the base case of using multiple pairwise models to perform multilingual translation, each model has $P + 2WV$ parameters, where $P = P^{(\text{enc})} + P^{(\text{dec})}$, W is the word embedding size, and V is the vocabulary size per language (assumed to be the same across languages, without loss of generality). Given that the models are pairwise, for

⁴ This is similar to the idea of auto-encoders by Vincent et al. (2008).

⁵ This is due to the small number of parameters that need to be learned in this case. To put this into perspective, in most of our experiments we used language embeddings of size 8.

L languages, we have a total of $L(L-1)(P+2WV) = \mathcal{O}(L^2P + 2L^2WV)$ learnable parameters. For our approach, using the linear parameter generator network presented in [Section 8.2.2.1](#), we have a total of $\mathcal{O}(PM + LWV)$ learnable parameters. Note that the number of encoder/decoder parameters has no dependence on L now, meaning that our model can easily scale to a large number of languages. In [Table 8.1](#) we show the number of learnable parameters, per model, for one of our experiments.

8.2.3 Experiments

In this section, we describe our experimental setup along with our results and key observations. For all our experiments we use as the base NMT model an encoder-decoder network which uses a bidirectional LSTM for the encoder, and a two-layer LSTM with the attention model of Bahdanau et al. (2015) for the decoder. The word embedding size is set to 512. This is a common baseline model that achieves reasonable performance and we decided to use it as-is, without tuning any of its parameters, as extensive hyperparameter search is outside the scope of this chapter. During training, we use a label smoothing factor of 0.1 (Wu et al., 2016a) along with the AMSGrad optimizer (Reddi et al., 2018) with its default parameters in TensorFlow and a batch size of 128 (due to GPU memory constraints). Optimization is stopped when the validation set BLEU score was maximized. The order in which language pairs are used while training is as follows: we first sample a language pair uniformly at random, and then sample a batch for this pair uniformly at random.⁶ During inference, we employ beam search with a beam of size 10 and the length normalization scheme of Wu et al. (2016a). We want to emphasize that we did not run experiments with other architectures or configurations, and thus this architecture was not chosen because it was favorable to our method, but rather because it was a frequently mentioned baseline in existing literature. All experiments are run on a machine with a single Nvidia V100 GPU and 24 GBs of system memory. Our most expensive experiment took about 10 hours to complete, which would cost about \$25 on a cloud computing service such as Google Cloud or Amazon Web Services, thus making our results reproducible even by independent researchers.

The goal of our experiments is to show how, by using a simple modification of this model: (i) we can achieve significant improvements in performance while at the same time (ii) being more data and computation efficient and (iii) enabling support for zero-shot translation. To this end, we perform three types of experiments:

1. Supervised: In this experiment, we use full parallel corpora to train our models. Plain pairwise NMT models (PNMT) are compared to the same models modified to use our proposed decoupled parameter generator. We use two variants: (i) one which does not use auto-encoding of monolingual data while training (CPG*), and (ii) one which does (CPG). More details can be found in [Section 8.2.2.2](#).

⁶ We did not observe any “forgetting” effect, because we keep “revisiting” all language pairs throughout training. It may be interesting to explore other sampling schemes, but it is outside the scope of this chapter.

	PNMT	GML	CPG ⁸	CPG ⁸ _{C4}	CPG ⁸ _{C2}	CPG ⁸ _{C1}	CPG ⁶⁴ _{C8}	CPG ⁵¹² _{C8}
#Params	832M	113M	199M	156M	135M	124M	199M	199M

Table 8.1: Number of learnable parameters for each model, for the IWSLT-17 experiments. “M” corresponds to “millions.”

2. Low-Resource: Similar to the supervised experiments except that we limit the size of the parallel corpora used in training. However, for GML and CPG the full monolingual corpus is used for auto-encoding training.
3. Zero-Shot: In this experiment, our goal is to evaluate how well a model can learn to translate between language pairs that it has not seen while training. For example, a model trained using parallel corpora between English and German, and English and French, will be evaluated in translating from German to French. PNMT can perform zero-shot translation in this setting using pivoting. This means that, in the previous example, we would first translate from German to English and then from English to French (using two pairwise models for a single translation). However, pivoting is prone to error propagation incurred when chaining multiple imperfect translations. The proposed CPG models inherently support zero-shot translation and require no pivoting.

For the experiments using the CPG model without controlled parameter sharing, we use language embeddings of size 8. This is based merely on the fact that this is the largest model size we could fit on one GPU. Whenever possible, we compare against PNMT, GML by Johnson et al. (2017),⁷ and other state-of-the-art results.

8.2.3.1 Datasets

We use the following datasets:

- IWSLT-15: Used for supervised and low-resource experiments only (this dataset does not support zero-shot learning). We report results for Czech (Ch), English (En), French (Fr), German (De), Thai (Th), and Vietnamese (Vi). This dataset contains ~90,000-220,000 training sentence pairs (depending on the language pair), ~500-900 validation pairs, and ~1,000-1,300 test pairs.
- IWSLT-17: Used for supervised and zero-shot experiments. We report results for Dutch (Nl), English (En), German (De), Italian (It), and Romanian (Ro). This dataset contains ~220,000 training sentence pairs (for all language pairs except for the zero-shot ones), ~900 validation pairs, and ~1,100 test pairs.

We preprocess the data using a modified version of the Moses tokenizer by Koehn et al. (2007) that correctly handles escaped HTML characters. We also perform some Unicode character normalization and cleaning. While training, we only consider sentences up to length 50. For both datasets, we generate a per-language vocabulary that consists of the 20,000 most frequently occurring words while ignoring words that appear less than 5 times in the whole corpus.

⁷ We use our own implementation of GML in order to obtain a fair comparison, in terms of the whole MT pipeline. We have modified it to use the same per-language vocabularies that we use for our approaches, as the proposed shared BPE vocabulary fails to perform well for the datasets we use.

		BLEU				Meteor			
		PNMT	GML	CPG*	CPG	PNMT	GML	CPG*	CPG
100% Parallel Data	En→Cs	14.89	15.92	16.88	<u>17.22</u>	19.72	20.93	21.51	<u>21.72</u>
	Cs→En	24.43	25.25	26.44	<u>27.37</u>	27.29	27.46	28.16	<u>28.52</u>
	En→De	25.99	15.92	26.41	<u>26.77</u>	44.72	42.97	45.97	<u>46.30</u>
	De→En	30.93	29.60	31.24	<u>31.77</u>	30.73	29.90	30.95	<u>31.13</u>
	En→Fr	38.25	34.40	38.10	<u>38.32</u>	57.43	53.86	57.42	<u>57.68</u>
	Fr→En	37.40	35.14	37.11	<u>37.89</u>	34.83	33.14	34.34	<u>34.89</u>
	En→Th	23.62	22.22	26.03	<u>26.33</u>	—	—	—	—
	Th→En	15.54	14.03	16.54	<u>26.77</u>	21.58	21.02	22.78	<u>23.05</u>
	En→Vi	27.47	25.54	28.33	<u>29.03</u>	—	—	—	—
	Vi→En	24.03	23.19	25.91	<u>26.38</u>	27.59	26.96	28.23	<u>28.79</u>
Mean		26.26	24.12	27.30	<u>27.80</u>	32.98	32.03	33.67	<u>34.01</u>
10% Parallel Data	En→Cs	5.71	8.18	8.40	<u>9.49</u>	12.18	14.97	15.25	<u>15.90</u>
	Cs→En	6.64	14.56	14.81	<u>15.38</u>	13.02	20.04	19.98	<u>20.87</u>
	En→De	11.70	14.60	15.09	<u>16.03</u>	29.98	33.74	34.88	<u>36.19</u>
	De→En	18.10	19.02	19.77	<u>20.25</u>	22.57	23.27	23.65	<u>24.40</u>
	En→Fr	24.47	25.15	24.00	<u>25.79</u>	44.10	44.84	44.95	<u>46.22</u>
	Fr→En	23.79	25.02	24.55	<u>27.12</u>	26.28	26.61	26.20	<u>28.18</u>
	En→Th	7.86	15.58	<u>18.41</u>	17.65	—	—	—	—
	Th→En	7.13	9.11	<u>10.19</u>	10.14	13.91	16.32	16.78	<u>16.92</u>
	En→Vi	18.01	17.51	<u>18.92</u>	18.90	—	—	—	—
	Vi→En	6.69	16.00	16.28	<u>16.86</u>	13.39	21.01	21.34	<u>22.28</u>
Mean		13.01	16.47	17.04	<u>17.76</u>	21.93	25.10	25.38	<u>26.37</u>
1% Parallel Data	En→Cs	0.49	1.25	1.57	<u>2.38</u>	4.60	6.24	6.28	<u>8.38</u>
	Cs→En	1.10	1.76	1.87	<u>4.60</u>	6.29	7.13	7.08	<u>11.15</u>
	En→De	1.22	4.13	4.06	<u>6.46</u>	12.23	18.29	17.61	<u>23.83</u>
	De→En	1.46	3.42	3.86	<u>7.49</u>	7.58	8.79	8.95	<u>13.73</u>
	En→Fr	2.88	7.74	7.41	<u>12.45</u>	13.88	21.29	21.80	<u>30.36</u>
	Fr→En	4.05	5.22	5.06	<u>11.39</u>	9.58	9.86	9.83	<u>16.34</u>
	En→Th	1.22	5.72	8.01	<u>9.26</u>	—	—	—	—
	Th→En	1.42	1.66	1.65	<u>3.37</u>	6.08	7.22	5.89	<u>8.74</u>
	En→Vi	5.35	5.61	5.48	<u>8.00</u>	—	—	—	—
	Vi→En	2.01	3.57	3.64	<u>6.43</u>	7.86	8.76	8.48	<u>12.04</u>
Mean		2.12	4.01	4.26	<u>7.18</u>	8.51	10.95	10.74	<u>15.58</u>

Table 8.2: Comparison of the proposed approach (shaded rows) with the base pairwise NMT model (PNMT) and the Google multilingual NMT model (GML) for the IWSLT-15 dataset. The *Percent Parallel* row shows what portion of the parallel corpus is used while training; the rest is being used only as monolingual data. Results are shown for the BLEU and Meteor metrics. CPG* represents the same model as CPG, but trained without using auto-encoding training examples. The best score in each case is underlined and shown in red.

8.2.3.2 Results

Our results for the IWSLT-15 experiments are shown in Table 8.2. It is clear that our approach consistently outperforms both the corresponding pairwise model, PNMT, and GML. Furthermore, its advantage grows larger in the low-resource setting (up to 5.06 BLEU score difference, or a $2.4\times$ increase), which is expected due to the extensive parameter sharing in our model. For this dataset, there exist some additional published state-of-the-art results not shown in Tables 8.2 and 8.3. Huang et al. (2018)

		BLEU							
		PNMT	GML	CPG ⁸	CPG ⁸ _{C4}	CPG ⁸ _{C2}	CPG ⁸ _{C1}	CPG ⁶⁴ _{C8}	CPG ⁵¹² _{C8}
Supervised	De→En	21.78	21.25	22.56	20.78	22.09	21.23	21.50	22.38
	De→It	13.16	13.84	14.73	14.34	14.43	13.84	14.34	14.11
	De→Ro	10.85	11.95	12.24	12.37	12.72	10.37	11.32	11.94
	En→De	19.75	17.06	19.41	19.04	18.42	17.04	17.46	19.29
	En→It	27.70	25.74	27.57	27.11	28.21	26.26	27.26	27.48
	En→NL	24.41	22.46	24.47	25.15	24.64	23.94	24.48	24.50
	En→Ro	19.23	18.60	20.83	20.96	18.69	17.23	20.20	20.86
	It→De	14.39	12.76	14.61	15.06	14.15	13.12	14.18	14.69
	It→En	29.84	27.96	30.62	30.10	29.44	29.22	29.56	30.18
	It→NL	16.74	16.27	17.99	18.11	18.05	17.13	17.71	17.99
	NL→En	26.30	24.78	26.31	26.17	25.74	26.15	26.33	26.20
	NL→It	16.03	16.10	16.81	17.50	17.03	16.81	16.89	17.09
	NL→Ro	12.84	12.48	14.01	14.44	12.56	11.79	12.38	13.66
	Ro→De	12.75	12.21	13.58	13.66	13.02	12.62	12.96	13.63
Ro→En	24.33	22.88	23.83	23.88	24.20	23.58	24.65	23.57	
Ro→NL	13.70	14.11	15.34	15.51	15.11	14.65	15.29	15.19	
	Mean	18.99	18.15	19.68	19.75	19.28	18.44	19.16	19.74
Zero-Shot	De→NL	12.75	12.50	12.74	12.80	11.65	12.41	12.67	12.75
	It→Ro	9.97	9.57	10.57	10.17	10.42	9.65	10.69	10.32
	NL→De	11.32	10.47	11.52	11.20	11.28	10.89	11.63	11.45
	Ro→It	11.69	10.82	11.51	11.40	11.66	11.42	11.78	11.27
	Mean	11.43	10.84	11.59	11.39	11.25	11.09	11.69	11.44

Table 8.3: Comparison of our proposed approach (shaded rows) with the base pairwise NMT model (PNMT) and the Google multilingual NMT model (GML) for the IWSLT-17 dataset. Results are shown for the BLEU metric only because Meteor does not support It, NL, and Ro. CPG⁸ represents CPG using language embeddings of size 8. The “C4” subscript represents the low-rank version of CPG for controlled parameter sharing (see Section 8.2.2.1), using rank 4, etc. The best score in each case is underlined and shown in red.

We were unable to find reported state-of-the-art results for the rest of the language pairs.

report a BLEU score of 28.07 for the En→Vi task, while our model is able to achieve a score of 29.03. Furthermore, Ha et al. (2016) report a BLEU score of 25.87 for the En→De task, while our model is able to achieve a score of 26.77. Our results for the IWSLT-17 experiments are shown in Table 8.3.⁸ Again, our method consistently outperforms both PNMT and GML, in both the supervised and the zero-shot settings. Furthermore, the results indicate that our model performance is robust to different sizes of the language embeddings and the choice of M' for controllable parameter sharing. It only underperforms in the degenerate case where $M' = 1$. It is also worth noting that, in the fully supervised setting, GML—the current state-of-the-art in the multilingual setting—underperforms the pairwise models. The presented results provide evidence that our proposed approach is able to significantly improve performance, without requiring extensive hyperparameter tuning.

⁸ Note that, our results for IWSLT-17 are not comparable to those of the official challenge report (Cettolo et al., 2017) as we use less training data, a smaller baseline model, and our evaluation pipeline potentially differs. However, the numbers presented for all methods in this paper are comparable, as they are all obtained using the same baseline model and evaluation pipeline.

8.2.3.3 *Language Embeddings*

An important aspect of our model is that it learns language embeddings. In [Figure 8.6](#) we show pairwise cosine distances between the learned language embeddings for our fully supervised experiments. There are some interesting patterns which indicate that the learned language embeddings are reasonable. For example, we observe that German (De) and Dutch (Nl) are most similar for the IWSLT-17 dataset, with Italian (It) and Romanian (Ro) coming second. Furthermore, Romanian and German are the furthest apart for that dataset. These relationships agree with linguistic knowledge about these languages and the families in which they belong. We see similar patterns in the IWSLT-15 results but we focus on IWSLT-17 here, because it is a larger, better quality, dataset with more supervised language pairs. These results are encouraging for analyzing such embeddings to discover relationships between languages that were previously unknown. For example, perhaps surprisingly, French (Fr) and Vietnamese (Vi) appear to be significantly related for the IWSLT-15 dataset results. This is likely due to French influence in Vietnamese because of the occupation of Vietnam by France during the 19th and 20th centuries (Marr, 1981).

8.2.4 *Related Work*

Interlingual translation (Richens, 1958) has been the object of many research efforts. For a long time, before the move to NMT, most practical machine translation systems only focused on individual language pairs. Since the success of end-to-end NMT approaches such as the encoder-decoder framework (Cho et al., 2014; Sutskever et al., 2014; Bahdanau et al., 2015), recent work has tried to extend the framework to multilingual translation. An early approach was Dong et al. (2015a) who performed one-to-many translation with a separate attention mechanism for each decoder. Luong et al. (2016) extended this idea with a focus on multi-task learning and multiple encoders and decoders, operating in a single shared vector space. The same architecture was used by Caglayan et al. (2016) for translation across multiple modalities. Zoph and Knight (2016) flipped this idea with a many-to-one translation model, however requiring the presence of a multi-way parallel corpus between all the languages, which is difficult to obtain. Lee et al. (2017) used a single character-level encoder across multiple languages by training a model on a many-to-one translation task. Closest to our work are more recent approaches, already described in the beginning of this section (Firat et al., 2016a; Ha et al., 2016; Johnson et al., 2017), that attempt to enforce different kinds of parameter sharing across languages.

Parameter sharing in multilingual NMT naturally enables semi-supervised and zero-shot learning. Unsupervised learning has been previously explored with key ideas such as back-translation (Sennrich et al., 2016a), dual learning (He et al., 2016a), common latent space learning (Lample et al., 2018), etc. In the vein of multilingual NMT, Artetxe et al. (2018) proposed a model that uses a shared encoder and multiple decoders with a focus on unsupervised translation. The entire system uses cross-lingual embeddings and is trained to reconstruct its input using only monolingual data. Zero-shot translation was first attempted by Firat et al. (2016b) who performed

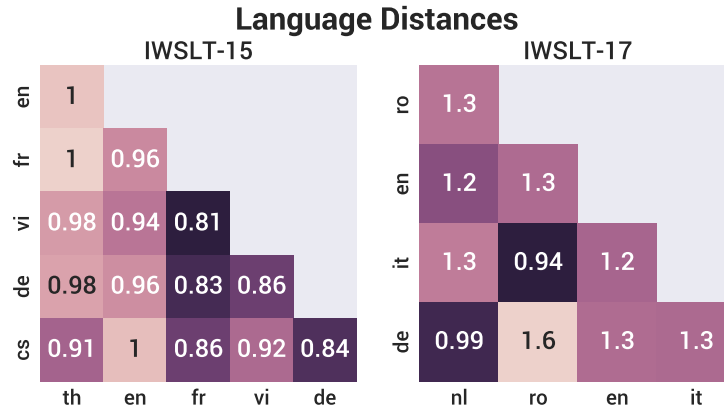


Figure 8.6: Pairwise cosine distance for all language pairs in the IWSLT-15 and IWSLT-17 datasets. Darker colors represent more similar languages.

zero-shot translation using their pre-trained multi-way multilingual model, fine-tuning it with pseudo-parallel data generated by the model itself. This was recently extended using a teacher-student framework by Chen et al. (2017). Later, zero-shot translation without any additional steps was attempted in (Johnson et al., 2017) using their shared encoder-decoder network. An iterative training procedure that leverages the duality of translations directly generated by the system for zero-shot learning was proposed by Lakew et al. (2017). For extremely low resource languages, Gu et al. (2018) proposed sharing lexical and sentence-level representations across multiple source languages with a single target language, and Cheng et al. (2016) proposed joint training of the source-to-pivot and pivot-to-target NMT models.

8.2.5 Key Takeaways

We have presented here a novel contextual parameter generation approach to neural machine translation. Our resulting system, which outperforms other state-of-the-art systems, uses a standard pairwise encoder-decoder architecture. However, it differs from earlier approaches by incorporating a component that generates the parameters to be used by the encoder and the decoder for the current sentence, based on the source and target languages, respectively. We refer to this component as the contextual parameter generator. The benefit of this approach is that it dramatically improves the ratio of the number of parameters to be learned, to the number of training examples available, by leveraging shared structure across different languages. Thus, our approach does not require any extra machinery such as back-translation, dual learning, pivoting, or multilingual word embeddings. It rather relies on the simple idea of *treating language as a context within which to encode/decode*. We also showed that the proposed approach is able to achieve state-of-the-art performance without requiring any tuning. Finally, we performed a basic analysis of the learned language embeddings, which showed that cosine distances between the learned language embeddings reflect well known similarities among language pairs such as German and Dutch. In the next section we present yet another application of this idea in an entirely different domain.

8.3 CASE STUDY #3: KNOWLEDGE GRAPH LINK PREDICTION

Another interesting and challenging multi-task learning problem is that of *knowledge graph link prediction*. Given a question that consists of a source entity and a relation (e.g., Shakespeare and BornIn), the goal is to predict the most likely answer entity (e.g., England). This can be viewed as an interesting multi-task learning problem where different relations are treated as different tasks. Recent approaches tackle this problem by learning entity and relation embeddings. However, they often constrain the interaction between these embeddings to be additive, as opposed to multiplicative (i.e., the embeddings are concatenated and then processed by a sequence of linear functions and element-wise non-linearities). We show that this type of interaction significantly limits representational power. For example, such models cannot handle cases where a different projection of the source entity is used for each relation. We propose to use contextual parameter generation to address this limitation. More specifically, we treat relations as the context in which source entities are processed to produce predictions by using relation embeddings to generate the parameters of a model operating over source entity embeddings. This allows models to represent more complex interactions between entities and relations. Note that this also goes back to our argument about additive and multiplicative interactions in [Section 7.2.1](#). We apply our method on two existing link prediction methods, including the current state-of-the-art, resulting in significant performance gains and establishing a new state-of-the-art for this task. These gains are achieved while also reducing training time by up to 28 times.⁹

8.3.1 Knowledge Graph Link Prediction

Many real-world applications ranging from search engines to conversational agents such as Amazon’s Alexa and Apple’s Siri rely on the ability to infer new facts from existing knowledge. A common means of representing such knowledge is via *knowledge graphs* (KGs). In KGs, facts are represented by entity-relation-entity triples, (e_s, r, e_t) , which encode factual relationships between graph nodes. An example triple of this form could be (Shakespeare, BornIn, England), which specifies that Shakespeare was born in England. For each triple, we refer to e_s and e_t as the source and target entities, respectively, and we refer to r as the relation between e_s and e_t . A collection of triples is called a *knowledge graph* because the triples implicitly form a graph where entities correspond to graph nodes and relations to graph edges. There are many existing large-scale KGs, both automatically generated—e.g., the never-ending language learner that was discussed in [Section 1.2](#)—and human-curated—e.g., Freebase by Bollacker et al. (2008). However, an important limitation is that they are often incomplete. For example, Freebase is missing the place of birth for 71% of the people that exist in its graph (West et al., 2014). Despite this deficiency, many missing links are inferrable from existing knowledge in the KG. For instance, knowing who Shakespeare’s parents were

⁹ The work we present in this chapter has been published in (Platanios* et al., 2020b). The link prediction problem is very relevant to the Never-Ending Language Learner (NELL) that was introduced in [Section 1.2](#). However, this section assumes no prior knowledge and can be read independently of the thesis introduction.

and where they lived could be used to infer the most likely place where Shakespeare was born. This motivates the task of *link prediction*, which is typically formulated as either question answering—inferring answers to questions of the form $(e_s, r, ?)$ —or fact checking—evaluating the validity for statements of the form (e_s, r, e_t) . While each formulation offers a different approach to link prediction, question answering can be thought of as a generalization of fact checking. This is because, in the worst case, answers can be produced by enumerating all possible entities and applying a fact checking model on each one of them. Thus, in this paper we propose a novel method to tackle link prediction using the question answering formulation, although our core contribution can also be applied to fact-checking methods. The proposed method consistently outperforms the current state-of-the-art across multiple datasets.

The study of link prediction has gathered substantial attention in the past years, and many methods have been proposed to solve it. A significant boost in performance was observed when recent methods such as ConvE (Dettmers et al., 2018), MINERVA (Das et al., 2018), and MultiHop-KG (Lin et al., 2018) combined KGs with the expressive power of neural networks. All these approaches consist of: (i) learning finite dimensional continuous vector representations (i.e., embeddings) for both the entities and the relations in the KG, and then (ii) processing them (e.g., using a neural network) in order to infer missing links in the KG. Different models process these embeddings through potentially very different types of architectures (e.g., convolutional networks or recurrent neural networks). However, they all have something in common: entity and relation representations are combined in a way that only allows for additive interactions between them (e.g., they may be concatenated and then projected using a linear transformation). In this work, we show how this type of interaction between entities and relations significantly limits expressive power, and we propose a novel method to address this limitation. More specifically, we propose to treat the relations as the *context* in which source entities are interpreted and transformed to produce target entities. Concretely, we use the relation embeddings to generate the parameters of a model operating over entities, which outputs a distribution over correct answers. The proposed method, CoPER (Contextual Parameters from Embedded Relations), has the following properties:

1. Abstract: It can be used to enhance the representational power of several existing link prediction methods.
2. Simple: It can be formulated as a simple transformation for qualifying models, that can be implemented with only about 10 lines of code.
3. Scalable: It speeds up convergence by up to 28 times.
4. State-of-the-Art: It outperforms competing methods by a significant margin on several established datasets.

Figure 8.7 shows an illustration of the proposed method. Existing approaches to perform link prediction can be classified in two categories: *single-hop* and *multi-hop* methods.

SINGLE-HOP METHODS. Given a question, single-hop methods predict answers by learning source entity and relation embeddings, and then jointly transforming

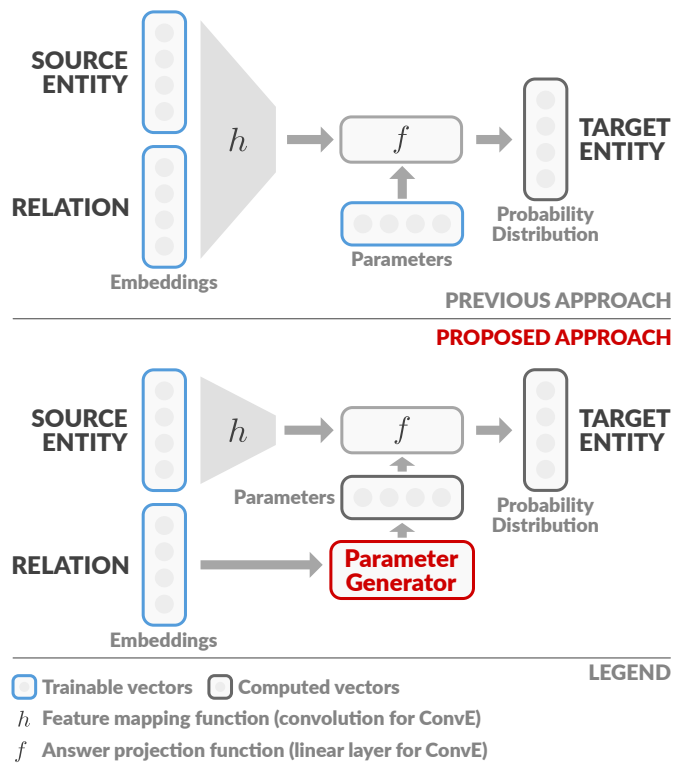


Figure 8.7: Overview of how our approach differs from prior work. Rather than stacking the relation with the source entity and transforming them jointly, the relation is used to generate the parameters of the model that is used to transform the source entity.

them to answer entities using a fixed amount of computation. TransE by Bordes et al. (2013) produces answer entity embeddings by adding the source entity and relation embeddings together. DistMult by Yang et al. (2015) extends TransE by instead multiplying the source entity and relation embeddings element-wise. ComplEx by Trouillon et al. (2016) boosts DistMult’s performance by extending the model to use complex numbers instead of real numbers (while retaining DistMult’s original architecture), allowing for more expressive relationships between entities and relations. TransR by Lin et al. (2015) infers answer entity embeddings by first projecting the source and potential answer entity embeddings to relation space via the query relation, and then applying TransE on the result. CTransR, by the same authors, extends TransR by clustering distinct source and target entity pairs from triples into groups, and learning distinct relation vectors for each group, thereby sharing information between group-correlated relations. TransD by Ji et al. (2015) develops on CTransR by accounting for entity types, resulting in greater method flexibility. ITransF by Xie et al. (2017) improves upon TransE by encouraging the sharing of statistic regularities between the projection matrices of relations using a sparse attention mechanism. DistMult, TransR, CTransR, and ITransF empirically illustrate the benefits of multiplicative interactions between entities and relations over additive ones, and serve as an inspiration for our work. Furthermore, as shall become clear by the end of this chapter, they all form special cases of the general

formulation that we propose. The current state-of-the-art model, ConvE by Dettmers et al. (2018), estimates a distribution over possible answers by first concatenating the source entity and relation embeddings and then feeding them through a convolutional neural network. Similar to TransE, the concatenation of entity and relation embeddings only allows for an additive interaction between the two. As we describe in Section 8.3.3 and briefly mentioned in Section 7.2.1, this significantly limits expressive power.

MULTI-HOP METHODS. Multi-hop approaches determine answers by finding paths connecting source entities to target entities, and consist mostly of path ranking methods (Lao et al., 2011; Gardner et al., 2013) and neural models (Guu et al., 2015; Nee-lakantan et al., 2015; Toutanova et al., 2016; Das et al., 2018; Lin et al., 2018). Given a question of the form $(e_s, r, ?)$, these methods aim to find sequences of relations that start at e_s and when composed, are semantically equivalent to r . NeuralLP by Yang et al. (2017) learns end-to-end differentiable relation paths between source entities and targets. Similarly, NTP- λ by Rocktäschel and Riedel (2017) proposes an end-to-end differentiable backward chaining model to learn effective sequences between source entities and answers. MINERVA by Das et al. (2018) proposes a history-dependent reinforcement learning approach to KG link prediction. MultiHop-KG by Lin et al. (2018) extends MINERVA by employing a pretrained single-hop method to shape the rewards used by MINERVA, which allows the model to use a more granular and informative reward policy when traversing paths. Both of these approaches process entities and relations additively by concatenating and transforming them to obtain the next path entity. Similar to their single-hop counterparts, this limits their expressivity.

8.3.2 Problem Formulation

Before describing our method, we introduce the notation that we will be using for the remainder of this case study. Let e_s , r , and e_t denote one-hot encoded representations of the source entity, relation, and target entity of a KG triple. A common approach to learning abstract representations of entities and relations is to learn vector embeddings. This provides for a simple yet effective method of sharing information. The transformation from one-hot encodings to vector embeddings is modeled as follows. Let N_e and N_r denote the total number of distinct entities and relations in the KG, respectively. Given a set of entities, $E = \{e_i\}_{i=1}^{N_e}$, and a set of relations $R = \{r_i\}_{i=1}^{N_r}$, we define the following embedding matrices: $E \in \mathbb{R}^{D_e \times N_e}$ and $R \in \mathbb{R}^{D_r \times N_r}$, where D_e and D_r correspond to the entity and relation embedding sizes, respectively. E and R are both trainable parameters. Given a question of the form $(e_s, r, ?)$, the corresponding source entity and relation embeddings are $e_s = Ee_s$ and $r = Rr$, where $e_s \in \mathbb{R}^{D_e}$ and $r \in \mathbb{R}^{D_r}$, respectively. Multiple existing single-hop link prediction methods (as well as a single hop in multi-hop methods) can be described in terms of the following abstract model:

$$e_s = Ee_s, \quad \text{ENTITY EMBEDDING} \quad (8.24)$$

$$r = Rr, \quad \text{RELATION EMBEDDING} \quad (8.25)$$

$$z = h_\phi(e_s, r, \dots), \quad \text{MERGE} \quad (8.26)$$

$$\mathbf{a} = f_{\theta}(\mathbf{z}, \dots), \quad \text{PREDICTION} \quad (8.27)$$

where $\mathbf{z}$ is a latent representation of the merged entity and relation embeddings. The merge is performed using the *merge function* h , parameterized by ϕ . Then, the answer $\mathbf{a}$ is predicted from $\mathbf{z}$ using the *prediction function* f , which is parameterized by θ . Depending on the model, $\mathbf{a}$ can be the embedding of the target entity $\hat{e}_t$, a probability distribution over target entities, or the probability that the fact (e_s, r, e_t) is true. Note that the functions h_{ϕ} and f_{θ} may also take other model-dependent arguments, such as e_t , which we represent using “...” In fact, in recent methods that have shown improved link prediction accuracies (Das et al., 2018; Dettmers et al., 2018; Lin et al., 2018), f_{θ} and h_{ϕ} are neural networks that consist of convolution and/or recurrent layers. The top part of Figure 8.7 shows an illustration of this formulation.

While multiple existing link prediction methods fit under this formulation, we use ConvE (Dettmers et al., 2018) as a running example, since it was both the state-of-the-art at the time of writing this chapter, and one of the baseline methods used in our experiments in Section 8.3.5. In ConvE, we have:

$$\mathbf{z} = \text{Conv2D}(\text{Reshape}([\mathbf{e}_s; \mathbf{r}])), \quad \text{MERGE} \quad (8.28)$$

$$\hat{e}_t = f_{\theta}(\mathbf{z}), \quad \text{PREDICTION} \quad (8.29)$$

where $[\mathbf{e}_s; \mathbf{r}] \in \mathbb{R}^{D_e + D_r}$ represents the result of stacking the entity and relation embeddings together, followed by a reshape into a $W \times H$ rectangular matrix, where W and H are model hyperparameters such that $WH = D_e + D_r$. This matrix is then passed through a 2D convolution layer to obtain the merged representation $\mathbf{z}$. The prediction function f_{θ} is defined as a linear layer, where θ denotes its weight matrix and bias vector combined, followed by a dropout layer. An illustration of the ConvE model is shown in the top part of Figure 8.9. For further details, we refer the reader to the work of Dettmers et al. (2018). Given that h_{ϕ} may also include other entity-relation merge operations, more complex models such as MINERVA or MultiHop-KG can also be expressed in terms of this abstraction.

8.3.3 Limited Expressive Power

Most existing neural methods consist of the following steps: (i) learn entity and relation embeddings, (ii) concatenate the source entity and relation embeddings, and (iii) perform a sequence of linear transformations and element-wise non-linear operations on them (as shown in Equations 8.24, 8.25, 8.26, and 8.27), in order to obtain the target entities. We now use an example to explain why this only explicitly allows for additive interactions between the source entity and relation, and why this is significantly limiting the model’s expressive power. Consider a simple merging function where the source entity and relation are first concatenated into a single vector as $[\mathbf{e}_s; \mathbf{r}]$, and then projected through a linear layer:

$$h_{\phi}(\mathbf{e}_s, \mathbf{r}) = \phi \cdot [\mathbf{e}_s; \mathbf{r}], \quad (8.30)$$

where $\phi \in \mathbb{R}^{D_z \times (D_e + D_r)}$ and D_z is the size of $\mathbf{z}$. If we refer to the first D_e columns of ϕ as ϕ_e , and the last D_r columns as ϕ_r (i.e., $\phi = [\phi_e; \phi_r]$), then we can write $h_\phi(\mathbf{e}_s, \mathbf{r}) = \phi_e \mathbf{e}_s + \phi_r \mathbf{r}$. Note that, in this case the elements of the entity embedding $\mathbf{e}_s$ and the relation embedding $\mathbf{r}$ only interact in an additive way (i.e., the output $\mathbf{z}$ is a linear combination of the elements in $\mathbf{e}_s$ and $\mathbf{r}$ and it does not support more complex interactions, such as multiplicative or polynomial interactions). The same is true for the merging function in ConvE through the convolution operation (only some of the elements of ϕ are shared), as well as the merging functions of MINERVA and Multihop-KG (further explained in Section 8.3.4.3). The main implication of this is that relations cannot influence the projection matrices used to transform the entities.

Let us demonstrate this important limitation by considering the example shown in Figure 8.8, illustrating 4 KG facts: (e_0, r_0, e_2) , (e_0, r_1, e_3) , (e_1, r_0, e_3) , (e_1, r_1, e_2) . Suppose we want to encode these facts using a model in the form of Equation 8.30:

$$\mathbf{e}_2 = \phi_e \mathbf{e}_0 + \phi_r \mathbf{r}_0, \quad (8.31)$$

$$\mathbf{e}_3 = \phi_e \mathbf{e}_0 + \phi_r \mathbf{r}_1, \quad (8.32)$$

$$\mathbf{e}_3 = \phi_e \mathbf{e}_1 + \phi_r \mathbf{r}_0, \quad (8.33)$$

$$\mathbf{e}_2 = \phi_e \mathbf{e}_1 + \phi_r \mathbf{r}_1. \quad (8.34)$$

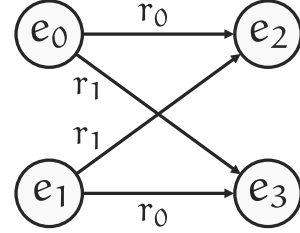


Figure 8.8: Toy example that cannot be modeled by additive interactions between entities and relations.

Subtracting Equation 8.32 from Equation 8.31, and Equation 8.34 from Equation 8.33, we have that:

$$(\mathbf{e}_2 - \mathbf{e}_3) = \phi_r (\mathbf{r}_0 - \mathbf{r}_1), \quad (8.35)$$

$$(\mathbf{e}_3 - \mathbf{e}_2) = \phi_r (\mathbf{r}_0 - \mathbf{r}_1), \quad (8.36)$$

which leads to a degenerate solution where: (i) $\mathbf{e}_3 = \mathbf{e}_2$, and (ii) $\phi_r = 0$ or $\mathbf{r}_0 = \mathbf{r}_1$. If we instead subtract Equation 8.33 from Equation 8.31, and Equation 8.34 from Equation 8.32, we achieve similar degenerate solutions where: (i) $\mathbf{e}_2 = \mathbf{e}_3$, and (ii) $\phi_e = 0$ or $\mathbf{e}_1 = \mathbf{e}_0$. Therefore, additive models are unable to handle this toy example.

While this is a toy example, it illustrates a more general problem. For instance, consider a case where we want to learn a different expert model for each relation. This means that given a source entity and relation, the relation determines which expert to use when processing the source entity. This example is important because related work in other areas has shown that mixtures of experts (our toy example is in fact a very simple form of a mixture of experts) can result in significant performance gains (Lengerich et al., 2018). Methods that combine entities and relations additively cannot learn such a mixtures of experts. Ideally, we want our model to be expressive enough such that it can learn functions that are conditioned on the relation, such as the above mixture of experts example. This example is important because learning a separate model for each relation may sometimes be impossible. A common pattern for certain KGs is that for some relations we have a lot of training data, but for most we have very little. In such cases, we want to be able to leverage the fact that many relations are similar by sharing information between them. Note also that increased

expressive power alone is not sufficient as it can result in overfitting. We propose to use contextual parameter generation which allows us to increase expressive power in a manner that, as we show using an extensive empirical evaluation in [Section 8.3.5](#), is useful to the link prediction problem. Importantly, as we show in [Section 8.3.4.2](#) the proposed approach is able to handle the aforementioned toy example as well as more general mixtures of experts.

8.3.4 Contextual Parameter Generation

We now present a new approach that addresses the limitations regarding additive interactions raised in the previous section. Our method, termed CoPER—*Contextual Parameters from Embedded Relations*—can be used to enhance multiple existing additive link prediction methods by enabling them to learn more expressive relationships between entities and relations. At the core of CoPER lies the key idea that relations define how source entities are processed in order to produce answer entities. Specifically, when answering a question $(e_s, r, ?)$, the target entity e_t can be obtained through a transformation of the source entity e_s and the parameters of this transformation are determined by the relation r .

In [Figure 8.7](#), we show how a baseline model, expressed using Equations [8.24](#), [8.25](#), [8.26](#), and [8.27](#), can be transformed using CoPER. In this baseline model, the embeddings of e_s and r are merged through the additive operation h (e.g., concatenation followed by convolution), and then transformed using f (e.g., a neural network) whose parameters are learned (e.g., through backpropagation). In CoPER, operation h is only applied to e_s , while r is used to *generate* the parameters of f . Thus, the parameters of f are no longer learned directly, but are rather the output of a new model component, the contextual parameter generator (CPG). In the following section, we propose and compare different potential architectures for the CPG module. We then explain how the proposed modification can have a large impact in the kinds of KG relationships our models can represent.

8.3.4.1 Parameter Generator Network

The contextual parameter generation (CPG) module is a function that takes as input a relation r and outputs the parameters θ of some other function f . Let $g: \{1, 2, \dots, N_r\} \mapsto \mathbb{R}^{D_\theta}$ be our parameter generation function, where N_r is the number of relations in the KG and $\theta \in \mathbb{R}^{D_\theta}$. We now present three simple functional forms for g that we also use for our experiments.

PARAMETER LOOKUP TABLE. The simplest approach is to output an entirely different set of parameters θ for each relation. This results in the following form:

$$g_{\text{lookup}}(\mathbf{r}) = W_{\text{lookup}} \mathbf{r}, \quad (8.37)$$

where $\mathbf{r}$ here is a one-hot encoded vector representation of the relation, and $W_{\text{lookup}} \in \mathbb{R}^{D_\theta \times N_r}$ is the only learnable parameter of g_{lookup} . Interestingly, this is comparable to DistMult and TransR, as each of these methods also uses relations to define distinct

mappings over entity embeddings. However, the problem with this simple formulation is that information sharing across relations can only happen through the shared entity embeddings. This makes the model prone to overfitting, especially for relations for which we have limited training data. Moreover, in certain KGs, many of the relations may be similar (e.g., `bornIn` and `livesIn`), and it may be beneficial for them to share information. This motivates a different approach for generating parameters.

LINEAR PROJECTION. Instead of using one-hot representations for the relations, we can instead learn embeddings:

$$g_{\text{linear}}(\mathbf{r}) = W_{\text{linear}} \mathbf{R} \mathbf{r} + \mathbf{b}, \quad (8.38)$$

where we use the embedding lookup equation, $W_{\text{linear}} \in \mathbb{R}^{D_\theta \times D_r}$, bias term $\mathbf{b} \in \mathbb{R}^{D_\theta}$, D_r is the relation embedding size, and both W_{linear} and $\mathbf{R}$, are trainable model parameters. Intuitively, the learned relation embeddings represent a linear combination of D_r different values for θ , allowing for sharing information between the relations.

MULTI-LAYER PERCEPTRON. Most of our experiments are performed using g_{linear} , with which we achieve state-of-the-art results. However, we observed that g_{linear} sometimes underperforms when using small datasets. We argue that the most likely reason is due to W_{linear} becoming too big relative to the original number of parameters. This is because, if we originally had D_θ trainable variables, g_{linear} now has $D_\theta \times D_r$ parameters, which is significantly larger. Limiting the value of D_r is not necessarily a solution as a small D_r can significantly constrain the capacity of our model. We therefore propose a third variant of the generator network using a multi-layer perceptron:

$$g_{\text{MLP}}(\mathbf{r}) = \text{MLP}(\mathbf{R} \mathbf{r}). \quad (8.39)$$

This can be thought of as a low-rank approximation to g_{linear} .

These are only three proposals for the parameter generator network and, as we show next, even a simple network such as g_{linear} already significantly increases representational power. Note however that the idea behind CoPER is more general and can be extended to more complex architectures. Moreover, in contrast to CTransR and TransD which also learn correlations between relations, CPG learns unrestricted relationships between them: based on the choice of CPG module, the network can learn arbitrary relation interactions. Furthermore, its abstract design enables it to fully benefit from the expressive power of neural networks.

8.3.4.2 Enhanced Expressive Power

Through the parameter generation component, CoPER enables link prediction methods to directly model more complex interactions between the entity and relation embeddings. A CPG module as simple as g_{linear} combined with any typical neural network architecture for f_θ (from a single linear layer to many complex layers fol-

lowed by element-wise non-linearities) allows the model to represent multiplicative interactions between source entities and relations. For ease of explanation, we will illustrate this increase in representational power with a simple form for h_ϕ and f_θ . However, it is easy to extend our example to more complex architectures. According to the CoPER formulation, h_ϕ now only operates on e_s , preprocessing the source entity embedding before passing it to the predictor function f_θ . For simplicity, we can assume that no preprocessing is necessary and so $h_\phi(e_s) = e_s$. We can further assume that f is a simple linear projection, $f_\theta(x) = \theta x$. The parameters θ are given by $\theta = g_{\text{linear}}(r) = WRr + b = Wr + b$. Therefore, we have that:

$$\hat{e}_t = f_\theta(h_\phi(e_s)) = f_\theta(e_s) = \theta e_s = (Wr + b)e_s. \quad (8.40)$$

This result shows that the relation and entity embeddings now interact in a multiplicative way, which means the relation itself can affect the weights with which we multiply the entity embedding. This is more expressive than an additive interaction, as it now allows us to represent dependencies such as conditionals (i.e., “if” statements), mixtures of experts, and even the toy example we present in [Figure 8.8](#).

TOY EXAMPLE. Going back to our toy example for which additive interactions are not sufficient, we now show that a CPG module as simple as g_{linear} or g_{lookup} can handle that example. Applying the predictor derived in [Equation 8.40](#) to the KG in [Figure 8.8](#), the following equations must hold for the toy example to be representable by the model:

$$e_2 = (Wr_0 + b)e_0, \quad (8.41)$$

$$e_3 = (Wr_0 + b)e_1, \quad (8.42)$$

$$e_2 = (Wr_1 + b)e_1, \quad (8.43)$$

$$e_3 = (Wr_1 + b)e_0. \quad (8.44)$$

Subtracting [Equation 8.41](#) from [Equation 8.42](#), and [Equation 8.43](#) from [Equation 8.44](#), we have that:

$$e_3 - e_2 = (Wr_0 + b)(e_1 - e_0), \quad (8.45)$$

$$e_3 - e_2 = (Wr_1 + b)(e_0 - e_1). \quad (8.46)$$

Avoiding the degenerate solution where $e_0 = e_1$, we have:

$$W(e_1 - e_0)(r_0 + r_1) + 2b(e_1 - e_0) = 0. \quad (8.47)$$

This equation has an infinite number of solutions. Note that although we showed here that CPG leads to multiplicative interactions between e_s and r for a particular choice of $f_\theta(x) = \theta x$, our conclusions will stand for most neural network architectures, from multilayer perceptrons to convolutional to recurrent neural networks, since they usually involve such a projection step on the inputs.

8.3.4.3 CoPER and Existing State-of-the-Art Models

We discussed how CoPER can generally be used to extend link prediction models that only allow for additive interactions. We will now show how it can be applied to two specific models, ConvE and MINERVA, which are representative of the two main lines of recent neural link prediction methods: single-hop and multi-hop methods. While these models may have distinct complex architectures with multiple types of neural network layers, each integrates entities and relations additively in several key components of their networks. We substitute each of these interactions with our CPG module to alleviate the aforementioned limitations. Figure 8.9 shows a parallel between the original ConvE model and its CoPER-enhanced version, CoPER-ConvE. Figure 8.10 shows the same comparison for MINERVA.

CONVE. The original ConvE model can be described in terms of our abstract framework, as shown in Equations 8.28 and 8.29. In CoPER-ConvE, the first preprocessing steps in the pipeline (reshape and convolution) are only applied to the entity embedding, while the relation is used to generate the parameters of the projection layer:

$$\mathbf{z} = \text{Conv2D}(\text{Reshape}(\mathbf{e}_s)), \quad (8.48)$$

$$\theta = g(r), \quad (8.49)$$

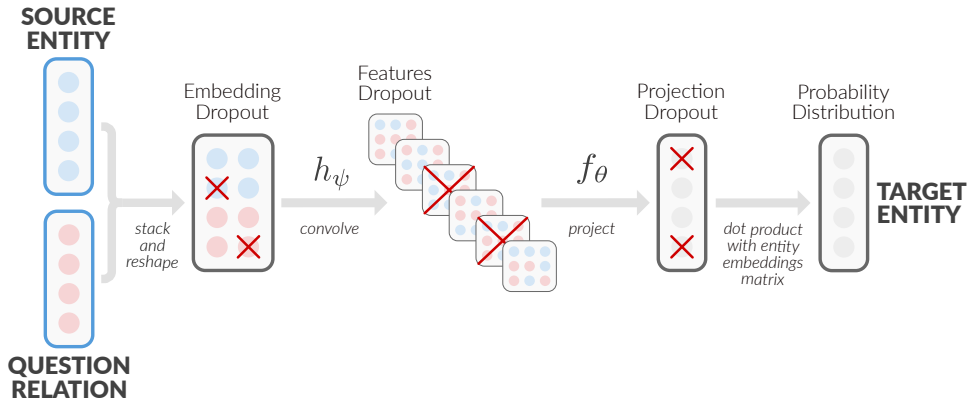
$$\hat{\mathbf{e}}_t = f_\theta(\mathbf{z}) = \theta_1 + \theta_{2:D_\theta} \mathbf{z}, \quad (8.50)$$

where $\theta = [\theta_1; \theta_2]$ is the parameter vector produced by the parameter generator, with its first element θ_1 used as the bias in the projection layer of f_θ , and the other $D_\theta - 1$ elements used as the weight matrix.

MINERVA. MINERVA is a deterministic multi-hop question-answering model that answers questions of the form $(e_s, r, ?)$ by finding paths in the graph that connect e_s to the predicted answer $\hat{e}_t$. The model defines states as the entities in the KG, and actions as tuples (r, e) that consist of an outgoing relation and its destination entity, specifying a hop to a neighboring node in the KG. Given a question (e_s, r_q, e_t) , MINERVA traverses the KG along its relations from e_s to the most likely target entity e_t . Each step along the graph path iteratively accumulates a history of entities and relations visited, which is aggregated and then stored in the hidden state of a long short-term memory (LSTM) network (Hochreiter and Schmidhuber, 1997), as illustrated in Figure 8.10. The hidden state of the LSTM is updated as follows:

$$\mathbf{h}_i = \text{LSTM}(\mathbf{h}_{i-1}, [\mathbf{e}_i; \mathbf{r}_{i-1}]), \quad \text{MERGE} \quad (8.51)$$

where $\mathbf{h}_i$ denotes the accumulated history representation at the i^{th} time step, $\mathbf{h}_{i-1}$ is the hidden state of the LSTM (i.e., the history representation) at the previous step, $\mathbf{r}_{i-1}$ denotes the embedding representation of the relation taken in the previous step leading to state $\mathbf{e}_i$ (represented by an entity embedding), and $[\cdot; \cdot]$ represents vector concatenation. Additionally, $[\mathbf{e}_i; \mathbf{r}_{i-1}]$ denotes the LSTM input. Because the LSTM module consists of a series of input projections, $\mathbf{e}_i$ and $\mathbf{r}_{i-1}$ are additively incorporated into the agent's history.



ConvE

CoPER-ConvE

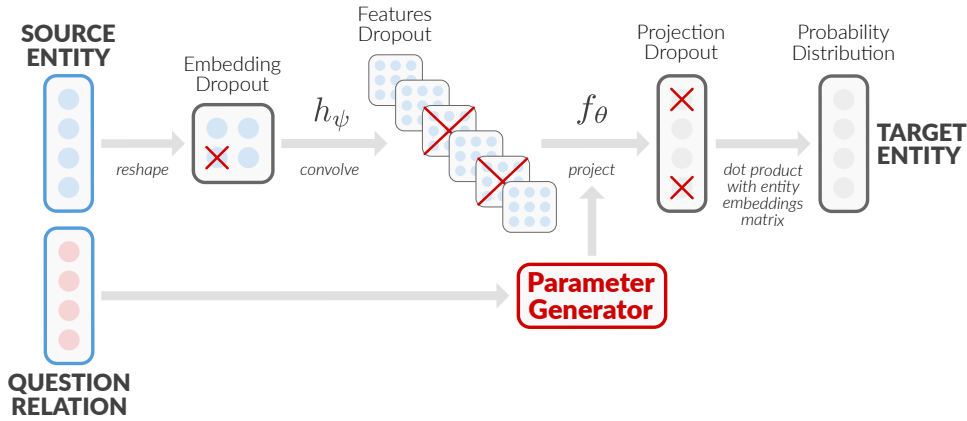


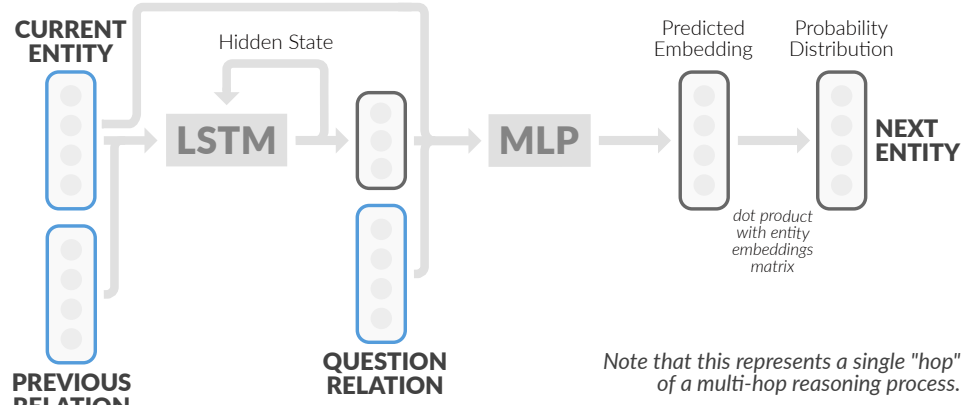
Figure 8.9: Overview of ConvE (top) and CoPER-ConvE (bottom).

At every time step, once the traversal history has been accumulated, the agent next determines the subsequent action to take as follows:

$$\mathbf{o}_i = \text{MLP}([\mathbf{h}_i; \mathbf{e}_i; \mathbf{r}_q]), \quad \text{MERGE} \quad (8.52)$$

$$\mathbf{a}_j = \text{Categorical}(\mathbf{A}_i \mathbf{o}_i), \quad \text{PREDICTION} \quad (8.53)$$

where $\mathbf{A}_i$ denotes the embedding representations of each available action from $\mathbf{e}_i$, $\mathbf{o}_i$ represents the multi-layer perception (MLP) output, Categorical denotes a categorical distribution decision function—such as a network policy—which operates over action distribution logits given by $\mathbf{A}_i \mathbf{o}_i$ —and $\mathbf{a}_j$ is the selected action. Since each action is a tuple (r, e) , as explained above, we represent $\mathbf{a}_j$ as the concatenation of the respective relation embedding and an entity embedding. $\mathbf{A}_i$ denotes then the matrix containing vector representations of all available actions from each state. Importantly, we observe that the entity and relation embeddings are concatenated as input to the MLP, which also induces an additive interaction between them in this component. Thus, in both



MINERVA

CoPER-MINERVA

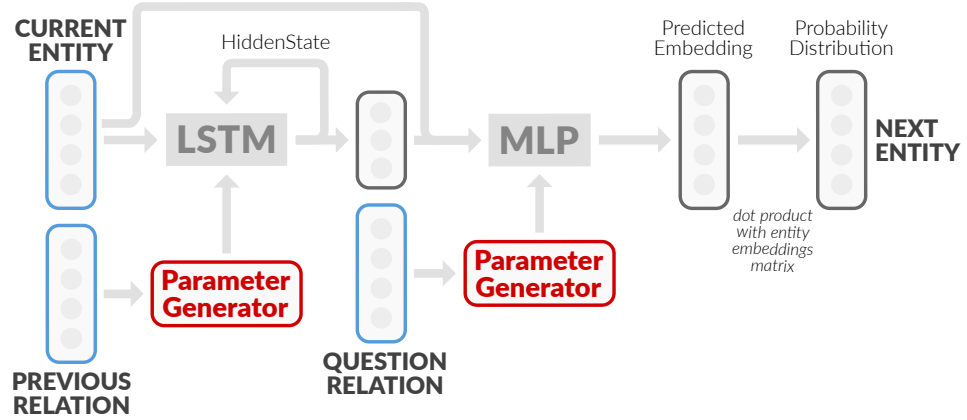


Figure 8.10: Overview of MINERVA (top) and CoPER-MINERVA (bottom).

components where entity and relation information is processed in MINERVA, it is done additively. As illustrated by our toy example, this limits expressive power.

In CoPER-MINERVA, we replace these additive steps with parameter generators, as illustrated in the bottom of Figure 8.10. In the first case, the embedding of the previous relation in a step, r_{i-1} , is used as input to a parameter generator that outputs the parameters of the LSTM component. In the second case, the query relation embedding, r , is used to generate the parameters of the MLP, which operates over the step history and the current entity representations. The rest of the model remains unchanged.

8.3.5 Experiments

In this section, we empirically evaluate the performance of CoPER on several established link-prediction datasets.

Dataset	#Train	N_e	N_r	$\bar{N}_a$	$\bar{d}$
UMLS	5,216	135	46	7.83	26.59
Kinship	8,544	104	25	6.14	82.15
FB15k237	272,115	14,541	237	3.03	17.87
WN18RR	86,835	40,945	11	1.41	2.19
NELL-995	154,213	75,492	200	3.57	4.07

Table 8.4: Dataset statistics. “#Train” denotes the number of questions used for training, N_e the number of distinct entities, N_r the number of distinct relations, $\bar{N}_a$ the average number of answers per question, and $\bar{d}$ the average degree of the graph nodes in the dataset.

8.3.5.1 Datasets

We adopt the following datasets used in prior literature:

1. UMLS: Unified medical language systems was proposed by Kok and Domingos (2007) and it contains links between various entities from the medical domain.
2. Kinship: Alyawarra Kinship contains kinship relationships between members of the Alyawarra tribe from Central Australia.
3. WN18RR: A subset of WordNet (Fellbaum, 1998) which contains lexical and semantic relationships between lexical entries, and that was provided by Dettmers et al. (2018).
4. FB15k-237: A subset of FreeBase (Bollacker et al., 2008) that contains entity-relation type links between data parsed from Wikipedia, and that was provided by (Toutanova and Chen, 2015).
5. NELL-995: A subset of NELL (Mitchell et al., 2015) that contains factual information extracted from the web, and that was provided by Xiong et al. (2017).

Table 8.4 displays summary statistics for each dataset. We keep our train, validation, and test dataset partitions consistent with those of prior literature to ensure fair comparisons. Specifically, we use the published datasets from Das et al. (2018) and Lin et al. (2018). Similar to prior work, we augment our training data with inverse relations (i.e., for each example, (e_s, r, e_t) , we introduce (e_t, r^{-1}, e_s)).

8.3.5.2 Metrics

We report results using two metrics: Hits@k and mean reciprocal rank (MRR). Both assess how a model ranks the correct answer compared to all other possible answers. Hits@k, also known as recall-at-k, is defined as the proportion of times the correct answer is ranked among the top-k answers, according to the probabilities predicted by the model. Similar to prior work, we report the average Hits@1 and Hits@10 over the test dataset. MRR is defined as the average value of the reciprocated rank of the correct answer for each test instance. Therefore, MRR is a measure of the overall quality of a model’s predictions. Note that, these evaluation metrics were also previously used for ConvE, MINERVA, and MultiHop-KG.

8.3.5.3 *Baselines*

We evaluate CoPER-ConvE and CoPER-MINERVA against their base models, ConvE and MINERVA, and multiple other link prediction methods. To ensure a fair comparison between CoPER-ConvE and its unaltered baseline, we reimplemented ConvE in our setup, and retained the hyperparameters originally reported by Dettmers et al. (2018). Our implementation either matches or improves upon the previously published results (possibly due to the use of negative sampling which is described in the following paragraph). For CoPER-MINERVA, we construct our CoPER framework within the MINERVA implementation provided by Lin et al. (2018). For both our CoPER extensions, we vary the relation embedding size (originally set to 200) based on the number of relations in each dataset, which stems from our observation that datasets with few relations (e.g., Kinship or WN18RR) perform better with smaller embeddings. We choose the dropout parameters by performing a grid search in the interval $[0,1]$ based on the validation set Hits@1 score. Regarding the parameter generation module, we perform experiments using both g_{linear} and g_{MLP} , which are defined in Equation 8.38 and Equation 8.39, respectively. For the MLP, we use a single hidden layer with a rectified linear unit (ReLU) activation and choose the number of hidden units by also performing a grid search. We train our models using the binary cross-entropy loss function. For each positive training example, we sample 10 negatives as described in the following paragraph, and use a label smoothing factor of 0.1. We use the AMSGrad optimizer by Reddi et al. (2018) for all experiments, with a learning rate of 0.001 and a batch size of 512. All our experiments were performed on a machine with a single Nvidia Titan X GPU. All hyperparameter values and CPG architectures utilized in our experiments can be found in our repository at <https://github.com/otiliastr/coper>.

NEGATIVE SAMPLING. We train CoPER by minimizing the binary cross-entropy between the predicted distribution over answer entities, and the true answer entities. However, one challenge is that the training data only contains triples of the form (e_s, r, e_t) and there is no clear way to obtain negative examples of the form “this entity is not a correct answer to this question.” A common approach to address this challenge is to consider all answers that appear in the training set as correct answers and all other entities in the KG as wrong answers. However, this approach suffers from two main problems. First, it can become very expensive for large KGs as all the entities are involved in the computation of the loss function for each training example. Furthermore, it is not necessarily correct because some of the entities treated as incorrect answers during training are the very answers we are asked to infer at test time. To alleviate these two issues, we use an alternative approach based on that of Bordes et al. (2013) and Yang et al. (2015). Instead of considering all possible alternative answers as wrong, we uniformly sample a fixed number of alternative entities, for each positive triple, and use them as negative training examples. Our experiments indicate that this negative sampling approach boosts performance and significantly improves computational efficiency.

Dataset	Metric	Models								
		DistMult	CompEx	NeuralLP	NTP- λ	MINERVA	MultiHop-KG	ConvE	CoPER-MINERVA	CoPER-ConvE
UMLS	Hits@1	82.1	82.3	64.3	84.3	75.3	90.2	92.89	77.76 [†]	<u>95.46</u> [‡]
	Hits@10	96.7	99.5	96.2	<u>100.0</u>	96.7	99.2	99.70	97.43 [†]	99.70 [‡]
	MRR	86.8	89.4	77.8	91.2	84.1	94.0	95.35	85.44 [†]	<u>97.08</u> [‡]
Kinship	Hits@1	48.7	75.4	47.5	75.9	60.5	78.9	74.21	66.20 [†]	<u>83.62</u> [‡]
	Hits@10	90.4	98.0	91.2	87.8	92.4	98.2	97.86	94.23 [†]	<u>98.42</u> [‡]
	MRR	61.4	83.8	61.9	79.3	72.0	86.5	83.04	76.00 [†]	<u>89.52</u> [‡]
WN18RR	Hits@1	43.1	41.0	37.6	—	41.3	41.8	41.86	42.66 [†]	<u>44.05</u> [‡]
	Hits@10	52.4	51.0	<u>65.7</u>	—	51.3	51.7	52.17	50.99 [†]	56.12 [‡]
	MRR	46.2	44.0	46.3	—	44.8	45.0	45.19	46.51 [†]	<u>48.33</u> [‡]
FB15k237	Hits@1	32.4	15.8	16.6	—	22.3	<u>32.7</u>	30.30	29.49 [†]	32.18 [‡]
	Hits@10	60.0	42.8	34.8	—	44.9	56.4	60.83	50.39 [†]	<u>62.92</u> [‡]
	MRR	41.7	24.7	22.7	—	29.2	40.7	40.51	36.51 [†]	<u>42.56</u> [‡]
NELL-995	Hits@1	55.2	64.3	—	—	63.96	65.6	67.04	65.52 [†]	<u>72.15</u> [‡]
	Hits@10	78.3	86.0	—	—	82.35	84.4	87.96	83.24 [†]	<u>88.35</u> [‡]
	MRR	64.1	72.6	—	—	70.97	72.7	75.42	72.46 [†]	<u>78.68</u> [‡]

Table 8.5: Results for multiple link prediction models. The results for ConvE, MINERVA, CoPER-ConvE, and CoPER-MINERVA are reported according to our own experiments. The rest of the results are taken from Das et al. (2018). All numbers are expressed as percentages. [†] denotes experiments performed using g_{linear} , and [‡] denotes those performed using g_{MLP} . “—” denotes missing results from the respective publications. Note that for MINERVA, we use the implementation provided by Lin et al. (2018), and report our results with the provided implementation for fair comparison with our CoPER extension. The best score in each case is underlined and shown in red.

8.3.5.4 Results

Our experiment results are shown in Table 8.5. We observe that CoPER-ConvE outperforms ConvE on all datasets, with up to 9.41% Hits@1 performance gain over ConvE on Kinship. Moreover, we find that CoPER-ConvE achieves superior performance over all other existing methods on these datasets, often by a significant margin. Notably, we observe a 4.7% Hits@1 gain for Kinship over the best existing method and a 4.09% Hits@1 gain for NELL-995. To the best of our knowledge, CoPER-ConvE establishes a new state-of-the-art for this problem.

We also examine the effect of CoPER on training time. Since CoPER-ConvE is the variant with the best results across all datasets, we perform this analysis for ConvE and CoPER-ConvE. Given that CoPER-ConvE consistently outperforms ConvE in terms of Hits@1 we compare the number of iterations that each method requires to reach the best Hits@1 value that ConvE achieves (e.g., when both ConvE and CoPER reach 92.89% Hits@1 on UMLS). Then, we calculate the ratio: $\frac{\text{\#iterations CoPER}}{\text{\#iterations ConvE}}$. For instance, if a baseline model requires 10,000 steps to attain its best performance while its CoPER variant takes 3,000 steps to achieve identical performance, then our metric would

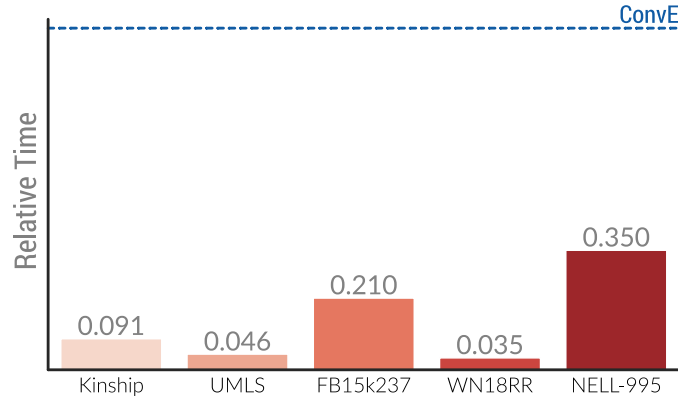


Figure 8.11: Time required for CoPER-ConvE to obtain its best performance on each dataset, as a fraction of the time it takes ConvE to achieve equivalent performance.

be: $\frac{3,000}{10,000} = 0.3$. Analogously, this would correspond to a $\frac{1}{0.3} = 3.33$ factor of training speed gain. Our results, illustrated in Figure 8.11, show that CoPER-ConvE always requires much fewer training iterations to achieve its best performance than ConvE, yielding a speedup of between 2.9 to 28.6 times.

Recall that, in Section 8.3.4.1 we proposed to use g_{linear} and g_{MLP} for the parameter generation network, instead of the arguably more straightforward parameter lookup table. This was motivated by suggesting that using g_{lookup} would more likely result in overfitting, especially in cases where there is too little training data per relation. To examine the impact of relation information sharing through the contextual parameter generator, we conduct an experiment comparing our best performing g_{lookup} CoPER models against our best performing g_{linear} or g_{MLP} models. As g_{lookup} does not enable this kind of information sharing, pitting it against the other generators enables us to explicitly analyze the importance of information sharing through the generator across our benchmark datasets. Table 8.6 illustrates our results from these experiments, with the addition of ConvE, for reference. In the table, CoPER-PL refers to g_{lookup} , while CoPER refers to the best performing g_{linear} or g_{MLP} model. Models using g_{linear} are marked with “ $\dagger$,” while models using g_{MLP} are marked with “ $\ddagger$.” Note also that, training CoPER-PL-ConvE on NELL-995 is infeasible with our resource capabilities due to the memory required to store the parameter lookup table and entity embeddings. Based on these experiments, we observe a strong correlation with performance disparity between CoPER-PL and CoPER and dataset size and sparsity (please refer to Table 8.4 for the dataset statistics). These results suggest that as datasets become smaller and denser, generator functions such as g_{linear} and g_{MLP} become crucial to maintaining strong performance. Moreover, while the performance discrepancy between CoPER-PL and CoPER is less prominent for larger and sparser datasets, we observe that sharing information through the generator remains highly important. Finally, we also compare CoPER with TransR and TransD, which are simple models that allow for multiplicative interactions, and CoPER consistently outperforms the other methods.

Dataset	Metric	Models		
		ConvE	CoPER-PL-ConvE	CoPER-ConvE
UMLS	Hits@1	92.89	73.82	<u>95.46</u> [‡]
	Hits@10	<u>99.70</u>	99.09	<u>99.70</u> [‡]
	MRR	95.35	85.17	<u>97.08</u> [‡]
Kinship	Hits@1	74.21	74.90	<u>83.62</u> [†]
	Hits@10	97.86	96.63	<u>98.42</u> [†]
	MRR	83.04	83.22	<u>89.52</u> [†]
WN18RR	Hits@1	41.86	<u>44.10</u>	44.05 [†]
	Hits@10	52.17	51.20	<u>56.12</u> [†]
	MRR	45.19	46.63	<u>48.33</u> [†]
FB15k237	Hits@1	30.30	30.72	<u>32.18</u> [†]
	Hits@10	60.83	60.04	<u>62.92</u> [†]
	MRR	40.51	40.52	<u>42.56</u> [†]
NELL-995	Hits@1	67.04	—	<u>72.15</u> [†]
	Hits@10	87.96	—	<u>88.34</u> [†]
	MRR	75.42	—	<u>78.68</u> [†]

Table 8.6: Results comparing different parameter generator networks. The results for ConvE, CoPER-PL-ConvE (using a parameter lookup generator function), and CoPER-ConvE are reported according to our own experiments. All numbers are expressed as percentages. [†] refers to the g_{linear} generator, while [‡] refers to the g_{MLP} generator. “—” denotes experiments outside our computational resource capabilities. The best score in each case is underlined and shown in red.

8.3.5.5 Relation Embeddings

One of our purported benefits of the contextual parameter generators described in Section 8.3.4.1 is that relation information can be shared through parameter generators using models such as g_{linear} , enabling relations to leverage knowledge from their similar counterparts. To illustrate this, we analyze the resulting relation embeddings after training CoPER-ConvE with g_{linear} on the NELL-995 dataset. In Figure 8.12 we provide a visualization of the pairwise cosine distances between the learned relation embeddings. Specifically, the plot illustrates the pairwise cosine similarities between each of the 200 relations in the NELL-995 dataset. Prior to plotting, the relations are manually grouped based on the types of entities they are defined over. We observe a clear block-diagonal structure in the resulting heatmap, which indicates that the learned relation embeddings reflect similarities that are indeed meaningful. For ease of understanding, we have labeled several blocks with their respective relation group “type.” All material relevant to our “type” assignment and code for this visualization can be found in our code repository at <https://github.com/otiliastr/coper>.

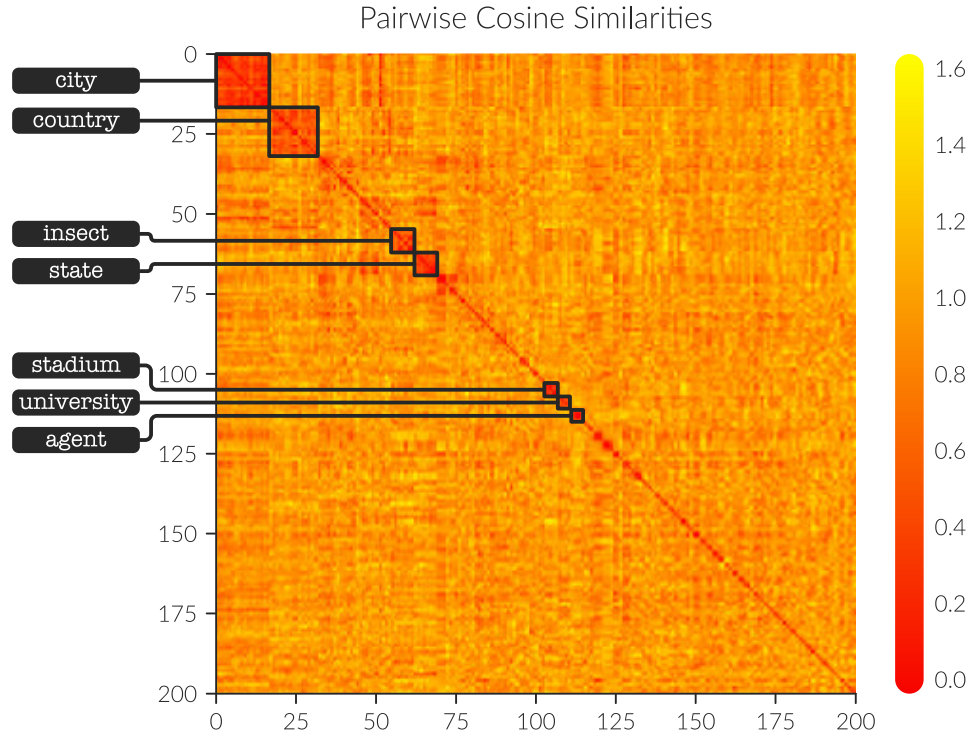


Figure 8.12: Heatmap showing the pairwise cosine distances between relation embeddings in NELL-995. Relations are grouped according to manually defined “types” which we have collected by manually annotating the dataset relations. Each block corresponds to a section denoted by these groups.

8.3.6 Key Takeaways

We proposed CoPER, a novel framework that improves upon the current state-of-the-art methods for the task of knowledge graph link prediction. CoPER treats relations as the context in which source entities are processed to predict target entities. We showed how this significantly increases the expressive power of link prediction models by allowing them to represent multiplicative interactions between entities and relations. We also exhibited the flexibility of our approach by extending both a single-hop and a multi-hop link prediction model, achieving new state-of-the-art performance for this task, while significantly speeding up convergence time over unaltered methods by up to $28\times$. This case study thus provides further evidence on the usefulness of contextual parameter generation for multi-task learning. In the next section, we present another case study that instead focuses more on using CPG for never-ending learning where there exists some compositional structure over the tasks that are being learned.

8.4 CASE STUDY #4: JELLY BEAN WORLD

This case study is based on the evaluation framework we describe in [Chapter 9](#) and so we recommend reading that chapter first. However, we shall try to make this

section relatively self-contained. Our main goal is to show how contextual parameter generation can be used to handle multi-task settings where the task being learned changes with time. More specifically, we consider a reinforcement learning function with a non-stationary reward function.

8.4.1 Problem Setting

The jelly bean world (JBW) is an *infinite* two-dimensional grid world where a learning agent can navigate around and collect items of different types. Each grid cell may contain up to one item of a specific type and the item types determine how the items are distributed around the world (e.g., there may be multiple onions distributed uniformly and few jelly beans and bananas that tend to appear in clusters). A simplified example illustration is shown in [Figure 8.13](#). The learning agent has two *perception modalities*:

- Vision: Each agent has a *visual range* property that specifies how far they can see. Vision is represented as a three-dimensional tensor, where the first two dimensions correspond to the width and the height of the agent’s visual field, and the third dimension corresponds to the color dimensionality. The visual field is always centered at the agent’s current position and the color observed at each cell within the visual field is the sum of the color vectors of all items and agents positioned at that map location.
- Scent: Scent is represented as a fixed-dimensional vector, where each dimension can be used to model orthogonal/unrelated scents. Each agent and each item has a pre-specified scent vector that is provided as part of the world configuration (similar to their colors). At each time step, agents can perceive the scent at their current grid position. The physics of scent are described by a simple diffusion difference equation on the world grid.

More details about both modalities are provided in [Section 9.2.1](#). In the simplified setting of this case study, at each time step the learning agent decides whether to turn right, turn left, or move forward. When an agent steps into a grid cell that contains an item, it automatically collects that item. The JBW configuration we use for the experiments of this case study is the same one that is used in [Section 9.4](#) and is summarized in [Tables 9.2](#) and [9.3](#).

8.4.2 Reward Function

Our goal in this case study revolves around non-stationary reward functions and thus we decided to use a composite reward function that alternates between the following two reward functions every 100,000 steps:

1. Collect[JellyBean] $\wedge$ Avoid[Onion]: The agent receives 1 reward point for each JellyBean it collects and loses 1 reward point for each Onion it collects.
2. Avoid[JellyBean] $\wedge$ Collect[Onion]: The agent loses 1 reward point for each JellyBean it collects and receives 1 reward point for each Onion it collects.

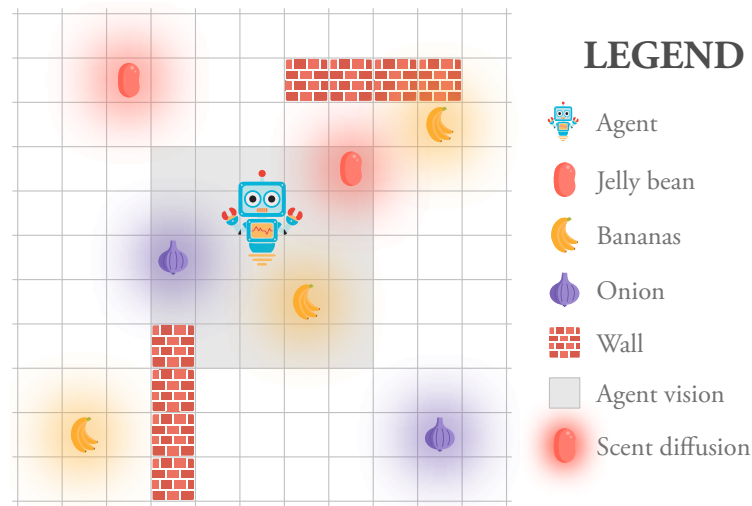


Figure 8.13: Example illustration of the jelly bean world. More details can be found in [Chapter 9](#).

We refer to this as a cyclical or periodic reward function schedule and it is also used in [Section 9.4](#) for further experiments. The non-stationary nature of this reward function renders this a hard learning problem as the learning agent will need to be able to adapt to a changing reward function every 100,000 steps. This is not something that “traditional” reinforcement learning models and training algorithms are generally able to handle as they often relying on using a decaying learning rate when learning which makes adaptation to changing rewards impossible. At the same time, as discussed in [Chapter 1](#), non-stationarity is often encountered in real world learning problems that humans encounter.

8.4.3 Models

We define a simple baseline model as follows:

1. The agent’s visual field is processed by a convolutional neural network (CNN) to produce a fixed-size vector. We refer to this network as the *vision sensor*.
2. The scent at the current cell is processed by a multi-layer perceptron (MLP) to produce a fixed-size vector. We refer to this network as the *scent sensor*.
3. The outputs of the vision sensor and the scent sensor are concatenated and fed to a long short-term memory (LSTM) network (Hochreiter and Schmidhuber, 1997) that produces another fixed-size vector. We refer to this network as the *reasoning module*.
4. The output of the reasoning module is processed by two separate linear layers to produce: (i) a distribution over actions for the agent to take in this step, and (ii) a score (or *value* for the current state. The latter is necessary because, as we explain in the next paragraph, we are using an *actor-critic* algorithm to train all our models. We refer to the last two linear layers as the *action networks*.

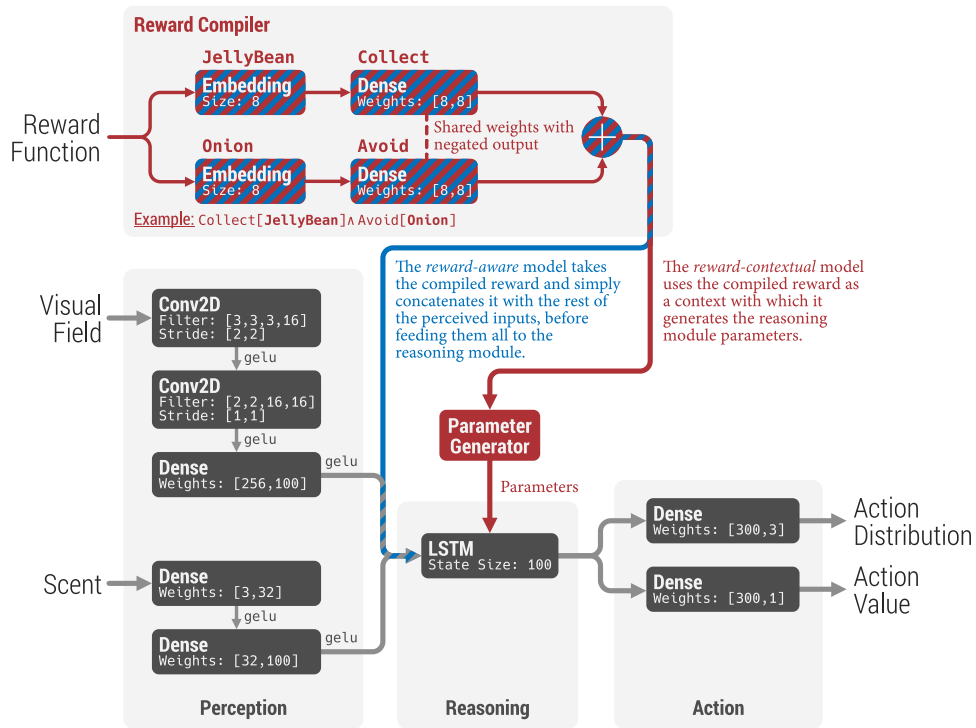


Figure 8.14: Overview of the models we use in this case study. The gray blocks and arrows correspond to modules that are common across all models. The blue blocks and arrows correspond to modules that are only used by the REWARD AWARE model. The red blocks and arrows correspond to modules that are only used by the REWARD CONTEXTUAL model. The blocks and arrows that have red and blue stripes correspond to modules that used by both the REWARD AWARE and the REWARD CONTEXTUAL model. Arrows merging together correspond to concatenation of the respective tensors. All models are described in [Section 8.4.3](#). Note that, `gelu` refers to the Gaussian error linear units (GELUs; Hendrycks and Gimpel, 2016) activation function.

Note that this model is not aware at all of the current reward function and thus, it can only adapt to a changing reward function by noticing how its collected rewards change with time. In the following sections, we refer to this model as PLAIN.

One way to improve upon this model is to make it aware of the reward function by feeding it as an additional input. This can be achieved by *compiling* the current reward function to a single vector embedding that is then concatenated with the outputs of the vision and scent sensors before being fed to the reasoning module. Our *reward compiler* component is described in [Section 8.4.3.1](#). We refer to this variant of the baseline model as the REWARD AWARE model.

Given our discussion in [Section 7.2.1](#), a natural way to employ contextual parameter generation in this setting would be to use the compiled reward function embedding as a context based on which the parameters of the reasoning module are generated. Note that, we decided not to have the agent use the reward function to generate the parameters of the perception (i.e., vision and scent sensors) and the action modules because, intuitively, we expect these modules to perform the same transformations

irrespective of which reward function is being used. In fact, there is a lot more underlying motivation for this decision, as well as for the grouping into perception, reasoning, and action modules. This motivation is described in detail in [Appendix C](#), where we propose *neural cognitive architectures*; a new type of neural network architectures that are inspired by human cognition and by the findings of this thesis. As we shall see in [Appendix C](#), the reward compiler that is described in the following section is also an integral part of neural cognitive architectures. In the following sections, we refer to this CPG-based model as the REWARD CONTEXTUAL model.

All models are trained using proximal policy optimization (PPO); a popular on-policy reinforcement learning algorithm proposed by Schulman et al. (2017).

8.4.3.1 Reward Compiler

We compile the reward function into a function that, when evaluated, produces a single vector embedding that encodes the current reward function. For the purposes of this case study, we use a very simple compiler:

- Item types are encoded in learnable embeddings of size 8.
- The Collect predicate is compiled to a learnable linear transformation that can be applied to item type embeddings. The output size of this linear transformation is also 8. The output of the compiled function is, on its own, a valid reward function embedding.
- The Avoid predicate is compiled to the same function as the Collect predicate, with its output negated.
- The conjunction of two predicates is compiled to a function that first compiles the two predicates and then adds the resulting reward function embeddings. We chose to use this simple functional form because it is invariant to the order in which the two predicates are specified. This is desirable because the reward function conjunction operation is also invariant to the order in which the two reward functions are specified.

For example, consider the reward function “Collect[JellyBean] $\wedge$ Avoid[Onion],” which is also used as an example in [Figure 8.14](#). This reward function would be compiled to the following vector value:

$$g_{\text{Collect}}(c_{\text{JellyBean}}) - g_{\text{Collect}}(c_{\text{Onion}}), \quad (8.54)$$

where $c_{\text{JellyBean}}$ and c_{Onion} are embeddings of the jelly bean and onion items, respectively, and g_{Collect} is a linear transformation. More details on the topic of reward compilation are provided in [Section C.6](#).

8.4.4 Experiments

For all experiments we evaluate performance using the *reward rate* metric, which is defined as the amount of reward obtained per step, computed over a moving window. The size of this window is 50,000 steps. This is an appropriate metric for this

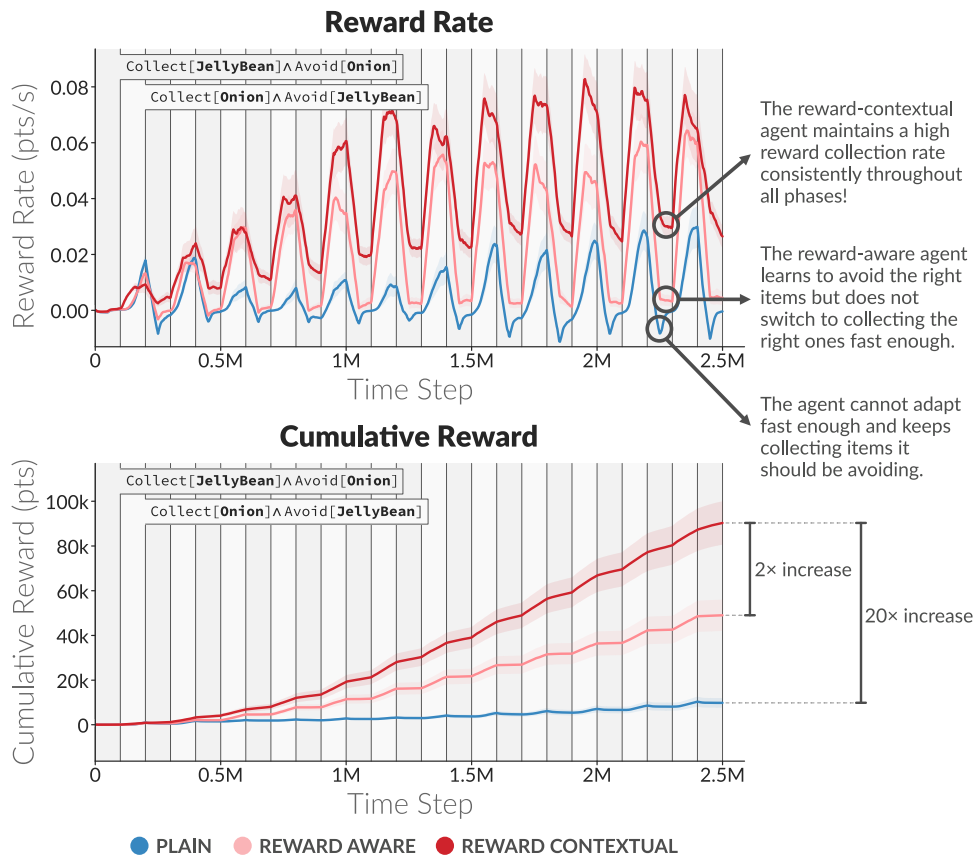


Figure 8.15: Experiment results along with key observations. The shaded vertical bars correspond to the two alternating reward functions that we use. The reward rate is computed using a 50-step window and the shaded bands correspond to standard error over 20 runs.

task as we want to measure the improvement in the ability of an agent to learn (i.e., the gradient of the reward rate), while also making sure the agent does not get stuck (i.e., the reward rate goes to zero). We also report the cumulative reward obtained by each agent as it helps better visualize the performance differences between the different models. Furthermore, we let the agents interact with the environment for a total of 2.5 million steps and perform 20 separate experiment runs for each agent, each one using a different random seed. We report the mean of the evaluation metrics over these runs, as well as the standard error. The experiments are implemented using Swift for TensorFlow¹⁰ and the code to reproduce them is available at <https://github.com/eaplatanios/jelly-bean-world>.

Our results are shown in Figure 8.15. In summary, we observe that the PLAIN model is unable to efficiently alternate between different learning problems and is effectively learning each problem from scratch whenever the reward function changes. This matches our expectations, given that this model does not observe the current reward function. REWARD AWARE is able to improve significantly upon PLAIN and we observe that it learns to avoid the right items when the reward function switches (i.e., its reward

¹⁰ <https://www.tensorflow.org/swift>.

rate does not become negative each time the reward function changes). However, it is still slow to switch to the correct behavior and maximize its reward rate. On the other hand, we observe that REWARD CONTEXTUAL learns to efficiently switch between tasks and consistently outperforms both PLAIN and REWARD AWARE. More specifically, it manages to obtain about $20\times$ higher cumulative reward than PLAIN and $2\times$ higher cumulative reward than REWARD AWARE.

8.4.5 Key Takeaways

This case study provides further evidence that contextual parameter generation is an effective way to perform multi-task learning. Specifically, it shows that it is better able to condition on the current task than methods that simply receive the current task as an additional input. This verifies the intuition and concrete arguments we provided in [Section 7.2.1](#). The contextualized architecture we proposed for this case study is also a simple instance of type of neural network architectures—*neural cognitive architectures*—that we propose in [Appendix C](#). Therefore, the results we obtained here provide additional support for that proposal.

Part III

EVALUATION

In Parts [i](#) and [ii](#) we proposed methods that enable and better support never-ending learning systems. However, building computer programs with these properties necessitates well-defined and robust ways to evaluate whether they are indeed capable of never-ending learning, and there are currently no ways to achieve this. To this end, in this part we propose a novel evaluation framework—the *jelly bean world* (JBW; [Chapter 9](#))—that can enable and facilitate research towards the goal never-ending learning. We have designed the JBW to be highly versatile, enabling evaluation of systems that have any number of the never-ending learning abilities mentioned in [Chapter 1](#).

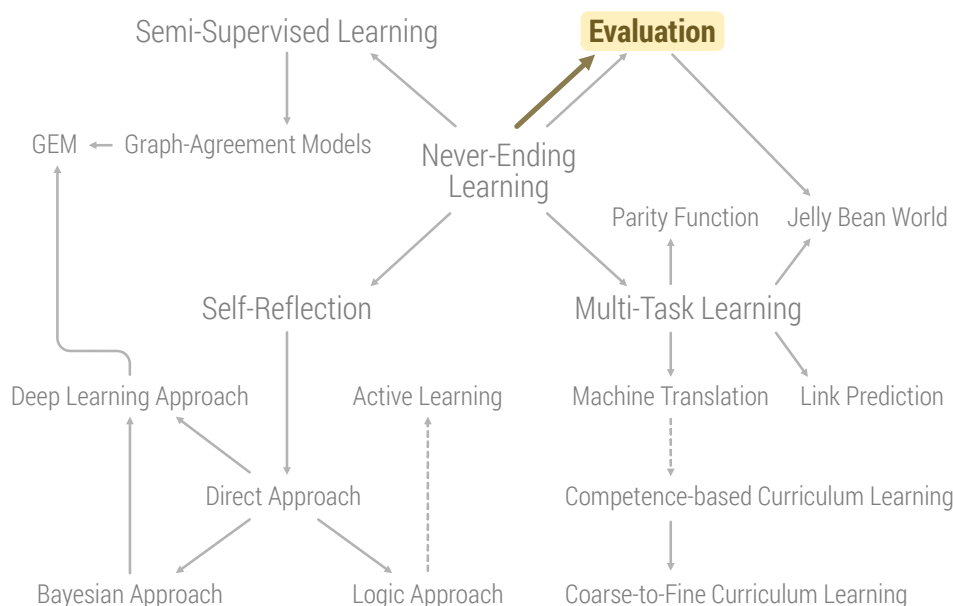


Figure 9.1: Illustration of how this chapter is positioned with respect to the rest of this thesis. The content of this chapter is shown in color, while the rest of the outline is shown in gray. The full outline is discussed in detail in [Section 1.4](#).

Throughout this thesis so far we have focused on proposing novel methods to tackle various problems related to learning collections of functions. As mentioned in [Chapter 1](#), this work was mainly motivated by never-ending learning, and building computer programs that are capable of never-ending learning necessitates well-defined and robust ways to evaluate for the desired properties. However, there are currently no ways to achieve that. There only exists one large-scale case study on never-ending learning with the Never-Ending Language Learner (NELL) by Mitchell et al. (2018), which uses the internet as the environment with which the system interacts (more details are provided in [Section 1.2](#)). While the internet does have significant complexity, it is unwieldy to use as a testbed. It is very difficult to focus on a particular aspect of the system or the environment, or to tweak the algorithm and restart experiments to observe the effects of changes. Furthermore, oftentimes tasks require manual annotation which can be very expensive. Thus, a good testbed for never-ending learning (and machine learning more generally) needs to provide the experimenter with a high degree of control. To this end, in this chapter we present a novel framework

for evaluating never-ending learning systems, and more generally systems aimed at learning collections of functions.

9.1 INTRODUCTION

We propose a novel evaluation framework—the *Jelly Bean World (JBW)*—that can enable and facilitate research towards the goal never-ending learning.¹ Before introducing the JBW, we summarize the properties that a testbed for never-ending learning should have:

1. Non-Episodic: It should disallow learning agents from resetting the environment and “retrying.” The testbed should also force them to only learn within a single environment (i.e., not transfer information across environments). This is in contrast with most popular reinforcement learning environments and, as we show in [Section 9.4](#), poses significant challenges to existing algorithms.
2. Non-Stationary: The testbed should allow for easy experimentation with non-stationary environments, where the reward can depend on time. Such reward functions are an easy way to increase the complexity of the world (for more details on our definition of world complexity please refer to [Section 1.3](#)).
3. Multi-Task: It should support settings in which reward is maximized not by learning how to perform a single task repetitively, but by learning how to perform a general variety of tasks, and learning how to switch between them and/or combine them to better perform other tasks (e.g., by composing them). We posit that, at a sufficiently high level of task complexity, optimal learning agents will be required—either explicitly or implicitly—to perform abstract reasoning and make informed decisions about actions in the environment.
4. Multi-Modal: It should support multiple data modalities that agents receive as input. These modalities should not contain the same information, but rather be complementary to each other so that the agents are forced to learn from diverse types of experiences. Multi-modality provides yet another way to increase the complexity of the world.
5. Controllable: It should be easy for experimenters to modify the complexity and richness of the learning problems in the testbed, make changes to it, and restart it (e.g., as opposed to NELL).
6. Efficient: It should run on readily available hardware and allow for quick experimentation. Ideally, we should not have to wait for days, weeks, or months (e.g., NELL) to obtain results.
7. Reproducible: It should make it easy to reproduce results and experiments, which would facilitate scientific research. This also requires that it allows for seamlessly saving and loading state and for reproducing results outside the environment in which they were first obtained. The testbed should also not require access to specialized hardware, which can be expensive.

Many of these properties are means to increase the complexity of the world. These properties are in fact very closely related to the characteristics of “AGI Environments,

¹ This chapter has been published in (Platanios* et al., 2020a).

Environment	Non-Episodic	Non-Stationary	Multi-Task	Multi-Modal	Controllable	Efficient
Atari (Bellemare et al., 2013) and Retro Games (Pfaud et al., 2018)	✗ Games end when the player wins or loses	✗ The game mechanics are stationary	✗ Each game has a single fixed reward function	✗ Agents only observe the game video frames	✗ Modifying the task complexity/richness is not possible	~ Can run on small machines but models are slow to train
Continuous Control (Duan et al., 2016), (Todorov et al., 2012)	✓ Some tasks are non-episodic (e.g., swimmer)	✗ Stationary rewards and environments (i.e., physics)	✓ Some of the tasks have interesting hierarchical structures	✗ Agents only observe positional information and joint angles	✗ The tasks and environments are non-configurable	~ Efficiency varies widely across tasks
Evolutionary Robotics (Mouret and Doncieux, 2012)	✗ Episodic in a finite world	✗ Stationary environments (i.e., physics)	✗ Only navigation goals	✓ Multiple different kinds of sensors	✓ Configurable using XML	✓ Fast 2D simulation written in C++
BabyAI (Chevalier-Boisvert et al., 2018)	✗ Episodic in a finite world	✗ Fixed set of levels and rewards	✓ Handful of tasks to perform	✓ Agents given visual input and instructions	✗ Existing levels are not configurable	✓ Built on fast MiniGrid simulator
Adversarial Games like Go (Silver et al., 2017), StarCraft (Vinyals et al., 2019), and Dota (OpenAI, 2019)	✗ Games end when the player wins or loses	✓ Non-stationary (without assumptions about the adversaries)	✗ Each game has a single fixed reward function	✓ Agents observe the game video frames and the game state	~ There is limited control over things like the adversary's competence	✗ Experiments are typically extremely computationally expensive
DeepMind Lab (Beattie et al., 2016)	✗ Levels have a time limit	~ Levels have different rewards but same physics	~ Levels have predefined rewards	✗ Agents only observe the game video frames	✗ The complexity of each level is fixed	~ Requires rendering of a 3D world
Malmö (Johnson et al., 2016) and MineRL (Guss et al., 2019)	~ Tasks have a pre-specified time limit but that is typically very long	✗ Stationary rewards and map generation is based on Perlin noise	✓ Supports 6 complex tasks but also allows for new ones	✓ Agents observe the game video frames and the game state	✗ Modifying the task complexity/richness is difficult and expensive	✗ Requires rendering of a 3D world and slow training of large models
Jelly Bean World (Proposed Here)	✓ Agents live "forever" in an infinite open world	✓ The rewards and the world can both be non-stationary	✓ Composable and dynamic tasks are supported	✓ Vision and scent are designed to be complementary	✓ Modifying the task complexity/richness is very easy	✓ Experiments can run efficiently on small machines

Table 9.1: Existing reinforcement learning environments positioned relative to our desired never-ending learning properties.

Tasks, and Agents” outlined by Laird and Wray III (2010) and later refined by Adams et al. (2012). The proposed JBW has all of these properties. It aims to provide an easy way to create sufficiently *complex* environments allowing researchers to experiment with never-ending learning, while remaining simple enough to control the problem and enable rapid prototyping. The JBW is a two-dimensional grid world with simple physics, but is extensible enough to admit a wide variety of complex and inter-related tasks. We present a comparison with related work in Table 9.1, showcasing the ways in which the JBW is a novel and highly versatile evaluation framework. The JBW is written in C++ and we provide C, Python, and Swift APIs. The source code is available at <https://github.com/eaplatanius/jelly-bean-world>. It is also worth noting that the JBW has already been used as the primary testbed for the instruction of the “Deep Reinforcement Learning” and the “Never-Ending Learning” graduate courses at Carnegie Mellon University.

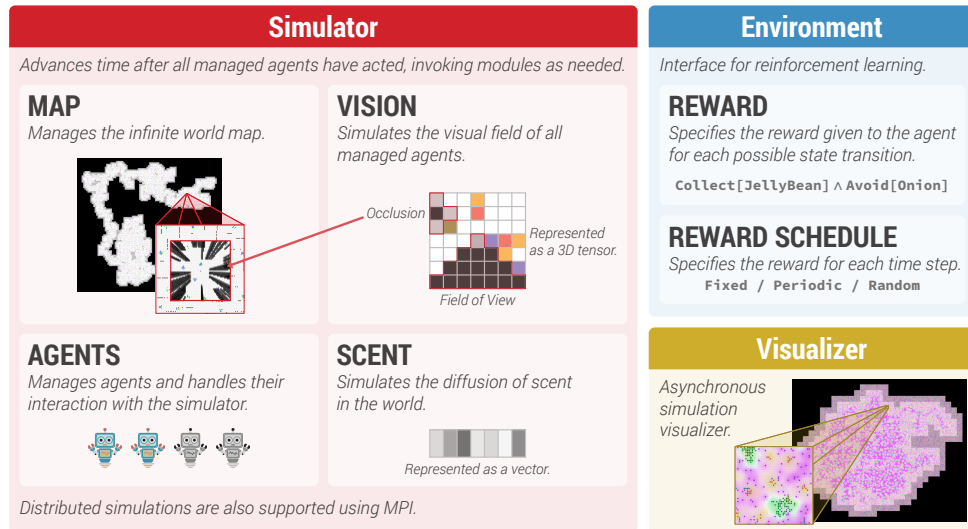


Figure 9.2: Overview of the modules comprising the Jelly Bean World.

9.2 DESIGN

The Jelly Bean World (JBW) consists of the following main modules, which are also illustrated in [Figure 9.2](#):

1. **Simulator**: Comprises the central component of the Jelly Bean World. The rest of the modules only interact with the simulator.
2. **Environment**: Provides a simple interface for performing reinforcement learning experiments in the never-ending learning setting as well as utilities for evaluating never-ending learning systems.
3. **Visualizer**: Provides the ability to visualize and debug the behavior of learning agents. The visualizer is completely asynchronous and can be attached, reattached, and detached to and from existing simulator instances, without affecting the simulations.

9.2.1 Simulator

The simulator manages a *map* and a set of *agents*. At a high-level, the map is an *infinite* two-dimensional grid where each grid cell can contain items (e.g., jelly beans and onions) and/or agents. Each item has a *color* and a *scent* that agents can perceive. Each agent has a direction and a position, and can navigate the world map and collect or drop items. The action space of each agent is to: turn, move, collect items, drop items, or do nothing, and it is configurable and can be constrained by the user. These constraints are described later in this section. Time in the simulator is discrete, and all agent-map interactions are *turn-based*, meaning that the simulator will first wait for all managed agents to request an action and will then simultaneously execute all actions and advance the current time. Thus, the simulator also controls the passage of time. In the following paragraphs, we describe each simulator component in detail.

MAP. In order to truly support never-ending learning, we have designed the JBW map to be *infinite*, meaning that it has no boundaries and agents can keep exploring it forever. To achieve this, the map is a procedurally-generated two-dimensional grid. We simulate it by dividing it into a collection of disjoint $(P \times P)$ -sized patches and only generating patches when an agent moves sufficiently close to them. The map also contains items of various types which are distributed according to a pairwise-interaction point process over the two-dimensional grid (Baddeley and Turner, 2000). Specifically, for a collection of items $\mathbf{I} \triangleq \{I_0, \dots, I_m\}$, where $I_i = (x_i, t_i)$, $x_i \in \mathbb{Z}^2$ is the position of the i^{th} item, $t_i \in \mathcal{T}$ is its type, and $\mathcal{T}$ is the set of all item types:

$$p(\mathbf{I}) \propto \exp \left\{ \sum_{i=0}^m f(I_i) + \sum_{j=0}^m g(I_i, I_j) \right\}, \quad (9.1)$$

where $f(I_i)$ is the *intensity* of item I_i and $g(I_i, I_j)$ is an *interaction* term between I_i and I_j , which are provided as part of the item's type. The intensity function characterizes the (log) probability of the existence of an item I_i independent of other items in the world. The interaction function can be understood as a description of the log probability of the existence of an item I_i given the existence of all other items I_j . For example, the interaction function can be used to increase the log probability of an item when it appears near other items, producing a clustering effect. Since the world is subdivided into $(P \times P)$ -sized patches, the maximum distance of interaction between items is P (this allows for efficiently generating map patches).

ITEM TYPES. Each item type $t \in \mathcal{T}$ defines the following configurable properties:

- **Color:** Fixed-size vector specifying the item color.
- **Scent:** Fixed-size vector specifying the item scent.
- **Occlusion:** Occlusion of an item, which is relevant to the vision modality that is described later in this section.
- **Intensity Function:** Maps from item locations to real values, where a higher value implies a higher probability of an item appearing in that location (i.e., corresponds to $f(I_i)$ in Equation 9.1).
- **Interaction Functions:** Collection of functions that map from pairs of item locations to real values, where a higher value implies a higher probability of that item pair appearing in those two locations (i.e., corresponds to $g(I_i, I_j)$ in Equation 9.1). The collection contains one function for each item type.

The number of item types and their properties are configurable. Note that each item type also specifies additional properties that are described later in this section. The JBW currently only supports a small number of implemented intensity and interaction functions to control the distribution of items in the world. However, it is very straightforward to implement new customized intensity and interaction functions. Let $(x, y) \in \mathbb{Z}^2$ be a position and $t \in \mathcal{T}$ be an item type. Intensity functions are indexed by item type, and so each item type is assigned its own intensity function: $f((x, y), t) \triangleq f_t(x, y)$. The JBW currently supports the following intensity functions:

Intensity Functions	
Zero	$f_t(x, y) = 0$
Constant[v]	$f_t(x, y) = v$
RadialHash[Δ, s, c, k]	$f_t(x, y) = c - k \cdot \hat{M}(\sqrt{x^2 + y^2}/s + \Delta)$, where $\hat{M} : \mathbb{R} \mapsto [0, 1]$ is the linear interpolation of $M(\lfloor t \rfloor)/(2^{32} - 1)$ and $M(\lfloor t \rfloor + 1)/(2^{32} - 1)$, and $M : \mathbb{Z} \mapsto \mathbb{Z}$ is the last mixing step of the 32-bit MurmurHash function (Appleby, 2008). This provides pseudorandomness to the intensity function.

For interaction functions, let (x_1, y_1) be the input position of the first item, t_1 be the type of the first item, (x_2, y_2) be the position of the second item, and t_2 be the type of the second item. Interaction functions are indexed by pairs of item types: $g((x_1, y_1), t_1), ((x_2, y_2), t_2)) \triangleq g_{t_1, t_2}((x_1, y_1), (x_2, y_2))$. The JBW currently supports the following interaction functions:

Interaction Functions	
Zero	$g_{t_1, t_2}((x_1, y_1), (x_2, y_2)) = 0$
PiecewiseBox[U, V, u, v]	$g_{t_1, t_2}((x_1, y_1), (x_2, y_2)) = \begin{cases} u, & \text{if } d < U, \\ v, & \text{if } U \leq d < V, \\ 0, & \text{otherwise,} \end{cases}$ where $d = (x_1 - x_2)^2 + (y_1 - y_2)^2$.
Cross[$U, V, u, v, \alpha, \beta$]	$g_{t_1, t_2}((x_1, y_1), (x_2, y_2)) = \begin{cases} u, & \text{if } d = 0, D \leq U, \\ \alpha, & \text{if } d \neq 0, D \leq U, \\ v, & \text{if } d = 0, U < D \leq V, \\ \beta, & \text{if } d \neq 0, U < D \leq V, \\ 0, & \text{otherwise,} \end{cases}$ where $d = \min\{ x_1 - x_2 , y_1 - y_2 \}$ and $D = \max\{ x_1 - x_2 , y_1 - y_2 \}$.
CrossHash[$s, c, k, \delta, u, v, \alpha, \beta$]	Same as Cross[$U, V, u, v, \alpha, \beta$] except for the fact that $U = c + k \cdot \hat{M}(x_1/s)$ and $V = U + \delta$, where $\hat{M}(\cdot)$ is defined as in the RadialHash intensity function.

Even though this is a small set of intensity and interaction functions it allows for creating worlds with many interesting features (e.g., we use the Cross interaction function to create contiguous wall segments that are axis-aligned, and the PiecewiseBox interaction function to create irregularly shaped clusters of trees forming forests).

PROCEDURAL GENERATION. When the simulator is instantiated the map is empty (i.e., no patches have been generated). Whenever a new agent is added to the simulator, a patch centered at its location is generated. In addition, whenever an existing agent moves sufficiently close to a region where no patch exists, a new patch is generated. The patch generation process consists of two main steps: (i) add a new empty $(P \times P)$ -sized patch to the collection of map patches (note that the new patch will be neighboring at least one existing patch and that all patches are disjoint), and (ii) fill the new patch with items. The second step is performed by using Metropolis-Hastings (MH) (Robert and Casella, 2010) to sample the items that the new patch contains, from the distribution defined in Equation 9.1. The proposal density we use is defined as follows:

- (i) add a new item $I_{m+1} = (x_{m+1}, t_{m+1})$ with probability $1/(2P^2 \cdot |\mathcal{T}|)$ (i.e., uniform in position and type), and
- (ii) remove an existing item I_i with probability $1/2m$ where m is the current number of items in the patch.

Before sampling, the patch is initialized by randomly selecting an existing patch and copying its items into the new patch. This is intended to facilitate rapid mixing of the Markov chain, and reduce the number of MH iterations. Note that if we use small patches and only sample new patches as the agents visit them, boundary effects may be observed due to the missing neighboring patches further away from the agent. For this reason, while sampling each new patch, we actually also sample all missing neighboring patches, but do not finalize them (i.e., they are still considered missing and may be resampled later on). This helps prevent boundary effects during the procedural generation process. An example illustration is shown in [Figure 9.3](#).

Each item has a color and a scent that is specified by its type and can be perceived by agents. The JBW thus supports two perception modalities, *vision* and *scent*. These modalities are complementary and agents can benefit by learning to combine them, as we explain at the end of this section.

VISION. Each agent has a *visual range* property that specifies how far they can see. Vision is represented as a three-dimensional tensor, where the first two dimensions correspond to the width and the height of the agent's visual field, and the third dimension corresponds to the color dimensionality. The visual field is always centered at the agent's current position and the color observed at each cell within the visual field is the sum of the color vectors of all items and agents positioned at that map location. Agents also have a *field of view* property that specifies their field of view angle (i.e., 180° denotes that the agent can only see the forward-facing half of the visual field, whereas 360° denotes that the agent can see the whole visual field). The part of the visual field that is outside an agent's field of view is masked out and appears black to the agent. Another important aspect of vision is that items also have an *occlusion* property as part of their type. This is used to simulate partial or complete *visual occlusion*. If an

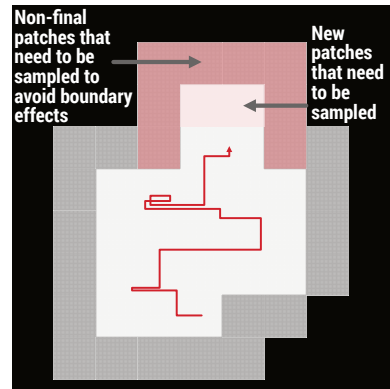


Figure 9.3: Illustration of the procedural generation algorithm for the infinite world map. The 32×32 patches shown in white have already been sampled and those in gray have been sampled but not fixed in order to avoid boundary effects. The red line corresponds to an example path followed by an agent. Once the agent enters a patch that is not fixed, then that patch is sampled, along with its non-fixed neighboring patches in order to avoid boundary effects. For simplicity, no items are shown in this map.

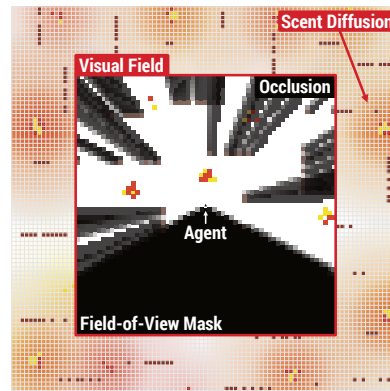


Figure 9.4: Rendering of an agent's perspective from the JBW visualizer.

item with occlusion 1 is in an agent's visual field, then the colors behind that item are not visible to the agent. An example is shown in [Figure 9.4](#).

Specifically, to compute the color of a cell with respect to the agent's field of view, let the cell position be (x, y) and consider a circle of radius $\frac{1}{2}$ centered at (x, y) . Project this circle onto the circle of radius 1 centered at the agent position. Let θ denote this projection (an arc). Let θ_{fov} be the arc on the agent's circle centered on a point in the current agent direction. The length of θ_{fov} is specified by the field of view parameter in the configuration. The color of cell $c_{x,y}$ is then computed as:

$$c_{x,y} = \hat{c}_{x,y} \cdot \frac{|\theta_{fov} \cap \theta|}{|\theta|}, \quad (9.2)$$

where $\hat{c}_{x,y}$ is the original color of the cell. In order to compute how much a cell at position (x, y) is occluded, we consider a circle of radius $\frac{1}{2}$ centered at (x, y) , and project this circle onto the circle of radius 1 centered at the agent position. Let θ denote this projection (an arc). Each item in the agent's visual field is similarly projected onto the agent's circle, each producing an arc θ_i . The color cell $c_{x,y}$ is then computed as:

$$c_{x,y} = \hat{c}_{x,y} \cdot \max \left\{ 1 - \sum_i o_i \frac{|\theta_i \cap \theta|}{|\theta|}, 0 \right\}, \quad (9.3)$$

where $\hat{c}_{x,y}$ is the original color of the cell, and o_i is the occlusion parameter of the i^{th} item, as specified by the item's type. If a cell is affected by both the field of view and visual occlusion, the above effects are composed (both multiplicative factors are applied to the original color).

SCENT. Scent is represented as a fixed-dimensional vector, where each dimension can be used to model orthogonal/unrelated scents. Each agent and each item has a pre-specified scent vector that is provided as part of the world configuration (similar to their colors). At each time step, agents can perceive the scent at their current grid position. The physics of scent are described by a simple diffusion difference equation on the world grid. We define the scent at location (x, y) at time t as:

$$S_{x,y}^t = \underbrace{C_{x,y}^t}_{\text{current scent}} + \underbrace{\lambda S_{x,y}^{t-1}}_{\text{previous scent}} + \underbrace{\alpha \left(S_{x-1,y}^{t-1} + S_{x+1,y}^{t-1} + S_{x,y-1}^{t-1} + S_{x,y+1}^{t-1} \right)}_{\text{neighboring cells diffused scent}}, \quad (9.4)$$

where λ is the rate of decay of the scent at each location, α is the rate of diffusion of the scent from neighboring grid cells, and:

$$C_{x,y}^t = \sum_{I \in \mathcal{I}_{x,y}^t} \text{scent}(I) + \sum_{A \in \mathcal{A}_{x,y}^t} \text{scent}(A), \quad (9.5)$$

where $\mathcal{I}_{x,y}^t$ is the set of all items at time t and location (x, y) , and $\mathcal{A}_{x,y}^t$ is the set of all agents at time t and location (x, y) . Our simulator ensures that the scent (or lack thereof) diffuses correctly, even as items are created, collected, dropped, and

destroyed. It does so by keeping track of the creation, collection, drop, and destruction times of each item in the world. Note also that, while simulating this diffusion, we also take into account the non-fixed patches that have been sampled in order to avoid boundary effects.

VISION-SCENT COMPLEMENTARITY. Vision and scent are complementary. Vision has high precision, in the sense that the agent can see the actual color of each grid cell in its visual field and can thus relatively accurately determine what items may exist in that cell. However, it has low recall—the agent can only see as far as its visual range allows and it has no visual information about the rest of the map. On the other hand, scent has low precision—the scent at the current cell is a linear combination of the scents of all items in the world and it may be very difficult to learn to interpret it and use it effectively. However, scent has high recall—the scent at the current cell contains information about items in a much larger range. Thus, learning to use both modalities will be beneficial to agents. In Sections 9.4 and 9.4.3, we provide some experimental results supporting this argument.

CONSTRAINTS. The simulator enforces multiple constraints on the actions that agents are allowed to take. We have designed the following small set of constraints with the goal of providing a computationally efficient way to support arbitrarily complex tasks and learning problems:

- Agent Collision: This occurs when multiple agents attempt to move to the same location at the same time. This conflict can be resolved in one of three ways: (i) allow multiple agents to occupy the same location, (ii) first-come-first-serve (only allow the first agent who made a move request for that location to actually move—this is the current default), or (iii) randomly choose one of the agents and satisfy their request (ignoring the requested action of the others).
- Item Blocking Movement: Item types may specify that they block agent movement (e.g., a `Wall` item type). This means that agents are not allowed to move to locations with items of that type.
- Item Collection Requirements: Item types may specify that in order to collect items of that type, an agent has to have first collected a number of other items (e.g., collecting `Wood` may be allowed only if an `Axe` has already been collected).
- Item Collection Costs: Similar to the collection requirements, item types may specify that in order to collect items of that type, an agent has to drop or destroy a specific number of other items (e.g., collecting an `Axe` may require destroying a piece of `Metal` and a piece of `Wood` that the agent has previously collected).

INTERFACE. Users interact with the simulator programmatically. The JBW provides functions to add or remove agents from the world, query the current vision and scent perception of each agent, and to direct agents to perform actions. Users can choose to add multiple agents to the world, thus enabling experimentation with multi-agent settings. Multi-agent interactions provide another controllable source of complexity in the JBW. Users can then request actions for each agent in the simulation (i.e., turn, move, do nothing, etc.). Once all agents have requested actions, the simulator executes

these actions and advances time, appropriately updating the state of the world. Users can also specify callback functions which are invoked when time is advanced.

SERVER/CLIENT SUPPORT. The JBW also provides a TCP server-client interface where the simulator can be setup to run as a server. Users (i.e., clients) can then connect to the server, and interact with the simulator by sending messages to the server. This allows use cases such as a class setting where students can each control an agent in a common simulated world, or perhaps a hackathon where participants can compete in a common world. It also allows debugging and visualization tools to be attached and detached to and from running simulator instances, without affecting the simulations. In fact, this is how our visualizer, which is described in the end of this section, communicates with the simulator.

PERSISTENCE. Simulations in the JBW can be saved to and loaded from files, which can then be distributed across platforms. This facilitates *reproducibility*. The simulator guarantees uniform random number generation behavior across all platforms and machines (e.g., in distributed settings). The state of the pseudorandom number generator is also saved and loaded along with the simulation.

PERFORMANCE. The JBW is implemented in optimized C++, with performance being highly prioritized in both its design and its implementation. This allows for less time and hardware resources to be spent simulating the world and more time and resources to be allocated for machine learning algorithms. Additionally, the JBW is perceptually quite simple, being a two-dimensional grid world with limited vision and scent inputs. This allows machine learning algorithms to focus less on perceptual information processing and more on abstract information processing, which we think is a hallmark of never-ending learning. As a rough indication of performance, on a single core of an Intel Core i7 5820K (released in 2014) at 3.5 GHz, the JBW can generate 8.56 patches per second, each of size 64×64 (i.e., 35,062 grid cells), using the configuration presented in [Section 9.4](#).

VISUALIZATION. Visualization can be instrumental when developing, debugging, and evaluating never-ending learning systems. To this end, we have implemented a real-time visualizer using Vulkan² in which the user can see any part of the simulated JBW, at any scale and simulation rate. The visualizer utilizes the simulator server-client interface to visualize simulations running in different processes or on remote servers, in a fully asynchronous manner. Rendering is multithreaded to provide a smooth and responsive user interface. Finally, the visualizer can be attached to and detached from existing simulation server instances, without affecting the running simulations.

9.2.2 Environment

Environments manage simulator instances and provide an interface for performing reinforcement learning experiments using the JBW. We provide implementations of

² Information on Vulkan can be found at <https://www.khronos.org/vulkan/>.

the JBW environments for OpenAI Gym (Brockman et al., 2016) in Python and for Swift RL (Platanios, 2019) in Swift. JBW environments support *batching* by design, with support for parallel execution of the multiple simulator instances being managed (i.e., one simulator for each batch entry). Perhaps the most important aspect of JBW environments is that they require the user to specify a *reward schedule* to use for each experiment. This schedule effectively defines the tasks that the agents are learning to perform. A reward schedule provides a function that, given a simulation time, returns a *reward function* to use at that time. A reward function returns a scalar reward value, given the current and previous states of the agent and the world (e.g., the world map). For example, a simple reward function could be one that gives the agent one reward point for each JellyBean it collects. We provide a simple domain-specific language (DSL) for composing and combining multiple reward functions in arbitrary ways, to allow for the design of composable learning tasks. This enables endless possibilities in the realms of multi-task learning, curriculum learning, and more generally never-ending learning. Currently environments are limited to single agent reinforcement learning settings, but we plan to support multi-agent settings in the future (this is easy because the JBW simulator already supports multiple agents for each simulation).

9.3 LEARNING TASKS

Learning tasks can be defined in terms of *reward functions* and *reward schedules*, which were defined in Section 9.2.2. The JBW allows researchers to easily define their own reward functions and schedules, but it also provides a few primitives and ways to compose them in order to effortlessly allow for quick experimentation and prototyping. In fact, all learning tasks used in our experiments were defined using these primitives. The currently supported primitives are:

Reward Function Compositions	
$r_1 \wedge r_2$	Applies both r_1 and r_2 and returns the sum of their rewards.

Reward Functions	
Action[v]	Give v to agents when they take an action (i.e., not a no-op).
Collect[i, v]	Give v to agents for each item of type i that they collect.
Avoid[i, v]	Give $-v$ to agents for each item of type i that they collect.
Explore[v]	Give v to agents each time they move further away from their starting position in the world map.

Reward Schedules	
Fixed[r]	The reward function is always fixed to r , and is thus stationary.
Curriculum[$\{r_i, t_i\}_{i=1}^R$]	Use reward function r_1 for the first t_1 steps, then r_2 for t_2 steps, ..., and keep using r_R after the list of reward functions is exhausted.
Cyclical[$\{r_i, t_i\}_{i=1}^R$]	Use reward function r_1 for the first t_1 steps, then r_2 for t_2 steps, ..., and then repeat after the list of reward functions is exhausted.

We note that Collect is a sparse reward function, whereas Action and Explore are not. For conciseness, we omit the v argument in reward functions when it is set to 1.

9.4 EXPERIMENTS

The goal of this section is to show how the non-episodic, non-stationary, multi-modal, and multi-task aspects of the JBW make it a challenging environment for existing machine learning algorithms, through a few example case studies. For all experiments we use the simulator configuration and item types shown in Tables 9.2 and 9.3. Due to space, the case studies focus on the single-agent setting. We use different agent models depending on which modalities are used in each experiment. If vision is used, then the visual field is passed through a convolution layer with stride 2, 3×3 filters, and 16 channels, and another one with stride 1, 2×2 filters, and 16 channels. The resulting tensor is flattened and passed through a dense layer with size 512. If scent is used, then the scent vector is passed through two dense layers: one with size 32, and one with size 512. If both modalities are being used, the two hidden representations are concatenated. In all cases, we use the Gaussian error linear units (GELUs; Hendrycks and Gimpel, 2016) as the nonlinearities. Finally, the result is processed by a long short-term memory (LSTM) network (Hochreiter and Schmidhuber, 1997) which outputs a value for the agent’s current state, along with a distribution over actions. Learning is performed using proximal policy optimization (PPO); a popular on-policy reinforcement learning algorithm proposed by Schulman et al. (2017). The experiments are implemented using Swift for TensorFlow.³

Map	Scent Dimensionality	3
	Color Dimensionality	3
	Patch Size	64×64
	MH Sampling Iterations	10,000
	Scent Decay (λ)	0.4
	Scent Diffusion (α)	0.14
Agent	Color	■ [0.00, 0.00, 0.00]
	Scent	■ [0.00, 0.00, 0.00]
	Action Space	MoveForward TurnLeft TurnRight
	Visual Range	8
	Field-of-View	experiment-specific

Table 9.2: Simulator configuration used in our experiments.

9.4.1 Case Studies

For all experiments we evaluate performance using the *reward rate* metric, which is defined as the amount of reward obtained per step, computed over a moving window. The size of that window varies per experiment and is reported together with the results. This is an appropriate metric for this task as we want to measure the improvement in the ability of an agent to learn (i.e., the gradient of the reward rate), while also making sure the agent does not get stuck (i.e., the reward rate goes to zero). Whenever possible, we also report the results obtained by the greedy vision-based agent described in Section 9.4.2. The greedy agent makes additional assumptions about the world, and thus, doesn’t generalize to more complex environments, it provides a lower bound on the optimal reward rate. Note that a perfect upper bound cannot be obtained as that would require solving an NP-hard discrete optimization problem.

³ <https://www.tensorflow.org/swift>.

JellyBean: Jelly beans appear close to bananas.	
Scent	 [1.64, 0.54, 0.40]
Color	 [0.82, 0.27, 0.20]
Occlusion	0.0
Blocks Agents	False
Intensity	Constant[1.5]
Interactions	JellyBean : PiecewiseBox[10,100,0,-6]
	Banana : PiecewiseBox[10,100,2,-100]
	Wall : PiecewiseBox[50,100,-100,-100]
Banana: Bananas appear close to jelly beans and away from walls.	
Scent	 [1.92, 1.76, 0.40]
Color	 [0.96, 0.88, 0.20]
Occlusion	0.0
Blocks Agents	False
Intensity	Constant[1.5]
Interactions	JellyBean : PiecewiseBox[10,100,2,-100]
	Banana : PiecewiseBox[10,100,0,-6]
	Wall : PiecewiseBox[50,100,-100,-100]
Onion: Onions appear scattered all over the world.	
Scent	 [0.68, 0.01, 0.99]
Color	 [0.68, 0.01, 0.99]
Occlusion	0.0
Blocks Agents	False
Intensity	Constant[1.5]
Interactions	None
Wall: Walls tend to be contiguous and axis-aligned.	
Scent	 [0.00, 0.00, 0.00]
Color	 [0.20, 0.47, 0.67]
Occlusion	1.0 in experiments with occlusion, 0.0 otherwise
Blocks Agents	True
Intensity	Constant[-12]
Interactions	Wall : Cross[20,40,8,-1000,-1000,-1]
Tree: Trees cluster together in irregular shapes.	
Scent	 [0.00, 0.47, 0.06]
Color	 [0.00, 0.47, 0.06]
Occlusion	0.1 in experiments with occlusion, 0.0 otherwise
Blocks Agents	True
Intensity	Constant[2]
Interactions	Tree : PiecewiseBox[100,500,0,-0.1]
Truffle: Truffles appear in forests and are very rare.	
Scent	 [8.40, 4.80, 2.60]
Color	 [0.42, 0.24, 0.13]
Occlusion	0.0
Blocks Agents	False
Intensity	Constant[0]
Interactions	Truffle : PiecewiseBox[30,1000,-0.3,-1]
	Tree : PiecewiseBox[4,200,2,0]

Table 9.3: Item types used in our experiments.

CASE STUDY #1: NON-EPISODIC.

The goal of this case study is to show that the JBW allows for experimenting with never-ending learning agents and to also show how current machine learning methods—e.g., PPO (Schulman et al., 2017) used to train an LSTM-based agent—are failing to effectively perform never-ending learning. For this experiment we reward the agent for collecting JellyBeans and avoiding Onions. We let agents interact with the JBW for 10 million steps. Our results are shown in Figure 9.5. The agents seem to be learning effectively for the first 1 million steps, but start to underperform later on, eventually getting stuck and being unable to collect any reward. This is the case for multiple different learning agents that we experimented with; both using different models and using different learning algorithms, such as Deep Q-Networks (DQNs) proposed by Mnih et al. (2013). After connecting the visualizer to observe what happens we see that all agents either: (i) get stuck in an area of the map that they have already explored and exhausted of jelly beans, or (ii) get stuck constantly rotating and not moving to new grid cells at all. This indicates that the JBW is indeed challenging for current machine learning methods when it comes to never-ending learning.

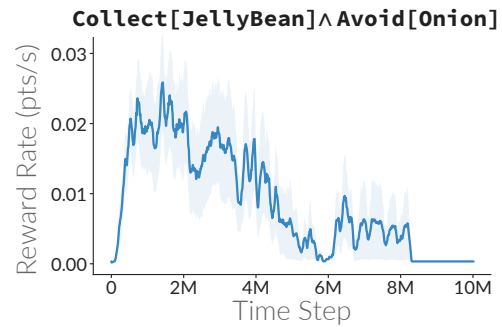


Figure 9.5: Non-episodic experiment result. The reward rate is computed using a 100,000-step window and the shaded bands correspond to standard error over 20 runs. “Greedy Visual” refers to the reward rate obtained by the greedy visual agent described in Section 9.4.2.

CASE STUDY #2: NON-STATIONARY. The goal of this case study is to demonstrate that the JBW allows for experimenting with non-stationary and multi-task learning problems. To this end, we perform two experiments: (i) one using a cyclical/periodic reward function schedule where every 100,000 steps we alternate between the $\text{Collect}[\text{JellyBean}] \wedge \text{Avoid}[\text{Onion}]$ and $\text{Avoid}[\text{JellyBean}] \wedge \text{Collect}[\text{Onion}]$ reward functions, and (ii) one testing a couple of curriculum reward schedules for eventually learning to $\text{Collect}[\text{JellyBean}] \wedge \text{Avoid}[\text{Onion}]$. The results are shown in Figure 9.6 and we observe that current standard machine learning approaches are not able to efficiently alternate between different learning problems and are effectively learning each problem from scratch whenever they switch, eventually ending up unable to learn either one. We also observe that agents who first learn to collect jelly beans and then switch to the full reward function are able to learn to collect jelly beans and avoid onions faster than agents that first learn to avoid onions or face the final learning problem directly from the beginning. Eventually all agents perform similarly, but this case study showcases how the JBW enables research in curriculum learning.

CASE STUDY #3: MULTI-MODAL. The goal of this case study is to: (i) show how computationally efficient features, such as the field of view mask and visual occlusion, allow for increasing the learning problem complexity in a controllable manner and, perhaps most importantly, (ii) show how the perception modalities of the JBW are

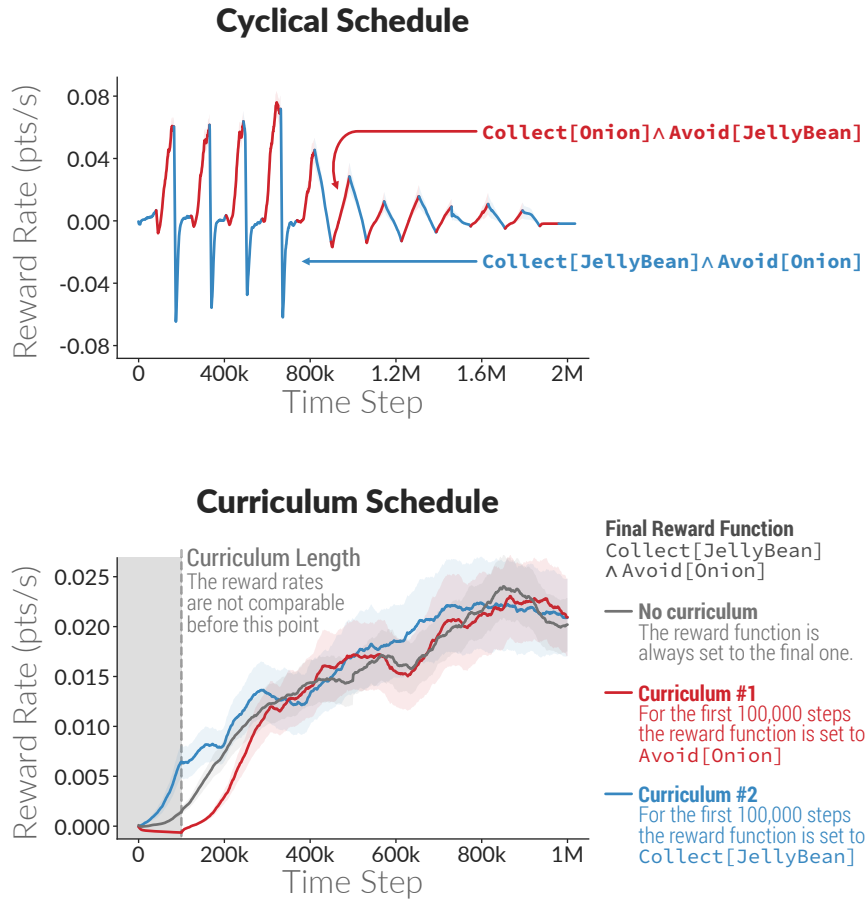


Figure 9.6: Non-stationary experiment results. The reward rate is computed using a 100,000-step window and the shaded bands correspond to standard error over 20 runs.

complementary. We thus perform three experiments. For the first two we use the fixed reward function $\text{Collect}[\text{JellyBean}]$ and for the last one we use $\text{Collect}[\text{Onion}]$. We change the reward function in order to show how easy it is to experiment using different tasks in the JBW. In the first experiment, we vary the field of view of the agents. The results are shown in the top plot of Figure 9.7. We see that decreasing the field of view allows us to make the learning task harder, while maintaining the same computational cost for the environment. Similarly, in the second experiment we measure the effect that visual occlusion has on performance. The results are shown in the middle plot of Figure 9.7 and we observe that enabling visual occlusion makes the learning task harder. Finally, with the third experiment our goal is to show that vision and scent are complementary. The results are shown in the bottom plot of Figure 9.7. We see that “vision” agents do better than “scent” agents, indicating that vision is perhaps an easier perception modality to use in the context of this learning task. Surprisingly though, the “vision” agents also do better than the “vision+scent” agents. This indicates a limitation of the model because, even though scent contains useful information that vision does not, the agents seem to get confused by it and

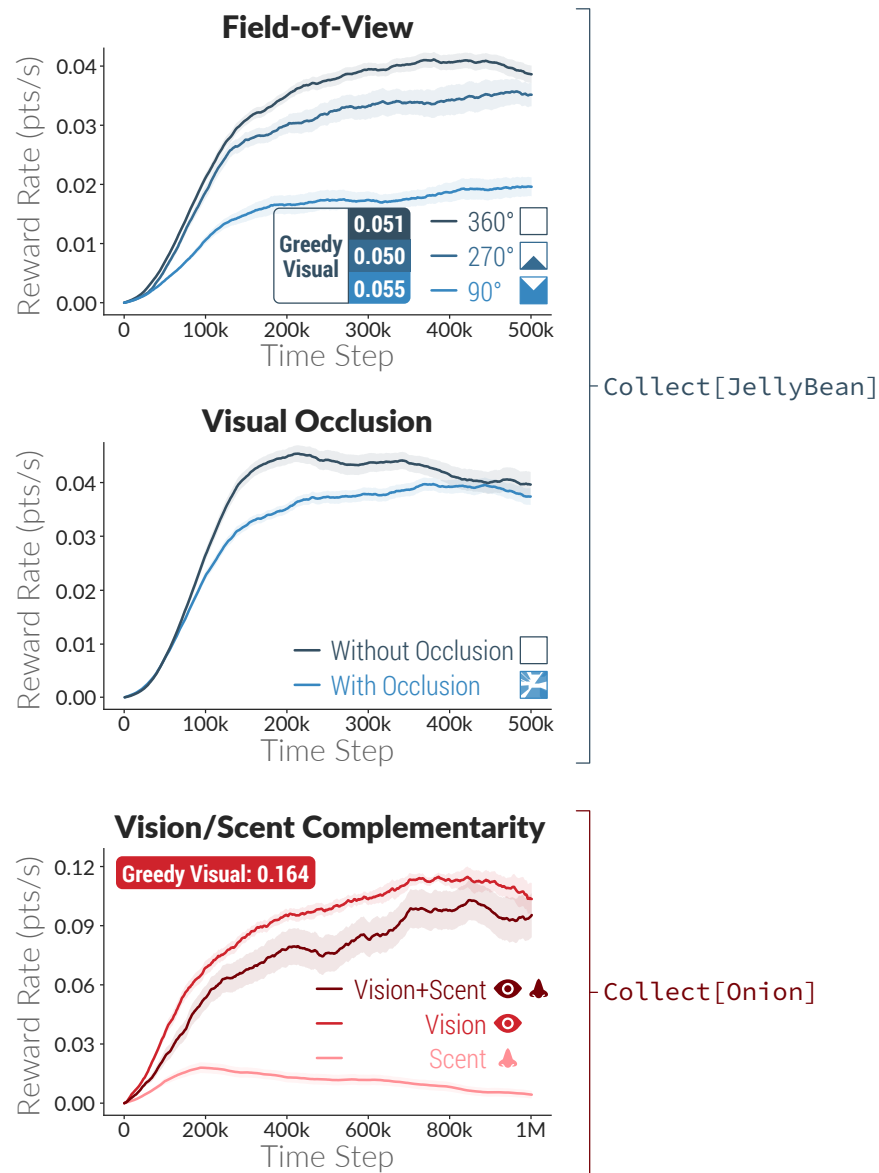


Figure 9.7: Multi-modal experiment results. The braces on the right specify the reward function used in each case. The reward rate is computed using a 100,000-step window and the shaded bands correspond to standard error over 20 runs. “Greedy Visual” refers to the reward rate obtained by the greedy visual agent baseline described in [Section 9.4.2](#) (which, as explained in that section, is not able to handle visual occlusion).

do not seem able to use it properly. It also shows the need for better multi-modal algorithms and the utility of the JBW in testing such algorithms. The relative utility of scent and vision depends on the environment and the agent model. We demonstrate this in [Section 9.4.3](#) by providing a different configuration where “scent” agents are able to outperform both “vision” and “vision+scent” agents.

9.4.2 Greedy Visual Agent

As a benchmark and for the sake of comparison, we also implemented a simple greedy agent that searches its visual field for cells of a particular color and then computes the shortest path to those cells. This algorithm makes the assumption that reward is maximized simply by collecting items of a single color, ad infinitum. It also assumes that this color is known a priori. Additionally, it assumes the color of obstacles (items that block agent movement or that should be avoided as part of the reward function) is known a priori, and is distinct from the color of items that provide reward. The shortest path it computes is such that it never goes through any such obstacles. The algorithm is shown in pseudocode in [Algorithm 9.1](#).

Algorithm 9.1: Pseudocode for the greedy vision-based algorithm.

Inputs: Color of rewarding items c_r and color of obstacles c_w .
Visual field ω_t .

```

1 bestPath  $\leftarrow$  null.
2 shortestPath  $\leftarrow$  ShortestPath( $\omega_t, c_r, c_w$ ).
3 if bestPath = null or |shortestPath| < |bestPath| then
4   | bestPath  $\leftarrow$  shortestPath
5 if bestPath = null then
6   | if the cell in front of the agent has color  $\gamma c_w$  for some  $\gamma > 0$  then
7     |   return MoveForward
8   | else
9     |   return one of {TurnLeft, TurnRight} sampled at random
10 else
11   | nextAction  $\leftarrow$  DequeueFrom(bestPath)
12   | if bestPath has no further actions then
13     |   bestPath  $\leftarrow$  null
14   | return nextAction
Output: Next action to take.

```

The function ShortestPath is simply Dijkstra's algorithm on a directed graph where each vertex corresponds to a unique agent position and direction within its visual field ω_t , and each edge corresponds to a possible action that transitions between agent states (Dijkstra, 1959). Let c_r be the color of items that provide reward, and c_w be the color of items that block agent movement. The algorithm returns a shortest path from the agent's current position and direction to a cell that has a color γc_r for any $\gamma > 0$, while avoiding cells that have color γc_w for any $\gamma > 0$ (we match any color in the direction of the vectors c_r and c_w in order to detect partially occluded items). If no such path exists, ShortestPath returns null. In the case where the agent's field of view is limited, ShortestPath only returns paths that pass through cells within the agent's field of view. Also, in the experiment where the agent must additionally avoid Onion items, ShortestPath avoids them in the same way that it avoids obstacles: it avoids cells that have color γc_o for any $\gamma > 0$, where c_o is the color of Onions.

In environments with visual occlusion, if items with high occlusion are arranged in a line (such as a wall), and the agent is adjacent to the wall and facing it, the portions of the wall further from the agent will be occluded by the portion of the wall closer to the agent. Since we currently do not distinguish between empty cells and completely occluded cells, `ShortestPath` will return paths that may pass through the wall. If no other paths are returned, the agent will continuously try to move through the wall and make no progress.

9.4.3 Relative Utility of Scent and Vision

The relative difficulty of utilizing scent or vision to effectively navigate in the JBW environments depends on the environment configuration as well as the method used by the learning agent. For example, if the learning agent does not possess memory, it cannot remember the scent of any previously visited tile, and thus it will not be able to determine the direction from which the scent is diffusing. Therefore, such methods

will not benefit from the information provided by the scent modality. In addition, scent is not necessarily blocked by items that block movement (e.g., walls). Thus, in environments with such items, scent is more difficult to utilize, since the simple strategy of following paths of monotonically increasing scent could lead to a wall. The agent could be fooled to believe an item is nearby when, in fact, it is behind a wall. This is illustrated in [Figure 9.8](#) and is possibly one of the main reasons the “vision+scent” agent underperforms the “vision” agent in [Figure 9.7](#). In the future, we would also like to support items that can block or reflect scent, which can result in interesting dynamics and make it potentially easier for agents to avoid this challenge.

In order to showcase how scent can provide useful information, we also designed a simpler environment configuration that does not contain any walls. The configuration for this world is shown in [Tables 9.4](#) and [9.5](#). Given the task of collecting JellyBeans, we expect the scent modality to be very useful as long as the agent has some sort of memory. The results of using an LSTM-based agent are shown in [Figure 9.9](#). We observe that the “scent” agent outperforms both the “vision” and the “vision+scent” agents. This is the opposite pattern of what we observe in [Figure 9.7](#). Also, note that since the configurations differ, the reward rates between the two figures are not directly comparable. This can partly be explained

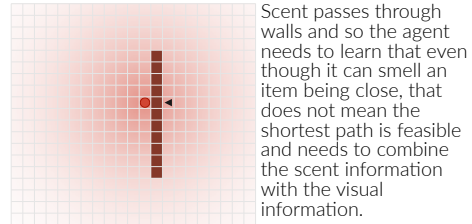


Figure 9.8: Example showing one of the challenges the scent modality poses for agents.

Map	Scent Dimensionality	3
	Color Dimensionality	3
	Patch Size	32×32
	MH Sampling Iterations	4,000
	Scent Decay (λ)	0.4
	Scent Diffusion (α)	0.14
Agent	Color	■ [0.00, 0.00, 0.00]
	Scent	■ [0.00, 0.00, 0.00]
	Action Space	MoveForward TurnLeft TurnRight
	Visual Range	5
	Field-of-View	60°

Table 9.4: Simulator configuration used for showing the relative utility of scent and vision.

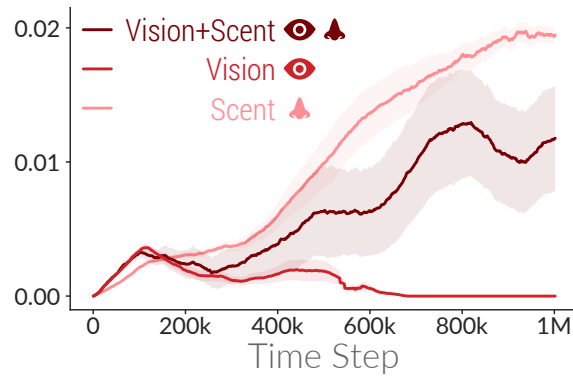


Figure 9.9: Results of experiments showcasing the relative utility of scent and vision.

JellyBean: Jelly beans appear close to bananas.	
Scent	■ [0.0, 0.0, 1.0]
Color	■ [0.0, 0.0, 1.0]
Occlusion	0.0
Blocks Agents	False
Intensity	Constant[-5.3]
Interactions	JellyBean : PiecewiseBox[10,200,0,-6]
	Banana : PiecewiseBox[10,200,2,-100]
	Onion : PiecewiseBox[200,0,-100,-100]
Banana: Bananas appear close to jelly beans.	
Scent	■ [0.0, 1.0, 0.0]
Color	■ [0.0, 1.0, 0.0]
Occlusion	0.0
Blocks Agents	False
Intensity	Constant[-5.3]
Interactions	JellyBean : PiecewiseBox[10,100,2,-100]
	Banana : PiecewiseBox[10,200,0,-6]
	Onion : PiecewiseBox[200,0,-6,-6]
Onion: Onions appear scattered, away from jellybeans and bananas.	
Scent	■ [1.0, 0.0, 0.0]
Color	■ [1.0, 0.0, 0.0]
Occlusion	0.0
Blocks Agents	False
Intensity	Constant[-5]
Interactions	JellyBean : PiecewiseBox[200,0,-100,-100]
	Banana : PiecewiseBox[200,0,-6,-6]

Table 9.5: Item types for the simple environment where the scent modality dominates the vision modality.

by the fact that, in this case, the visual field of the agent is more restricted, thereby limiting the agent’s reliance on vision in its learning. Ideally agents should be able to use each perception modality optimally and not be “confused” by the fact that one of them may be harder to utilize than the other. These experiments showcase how the JBW can be used to evaluate multi-modal machine learning algorithms.

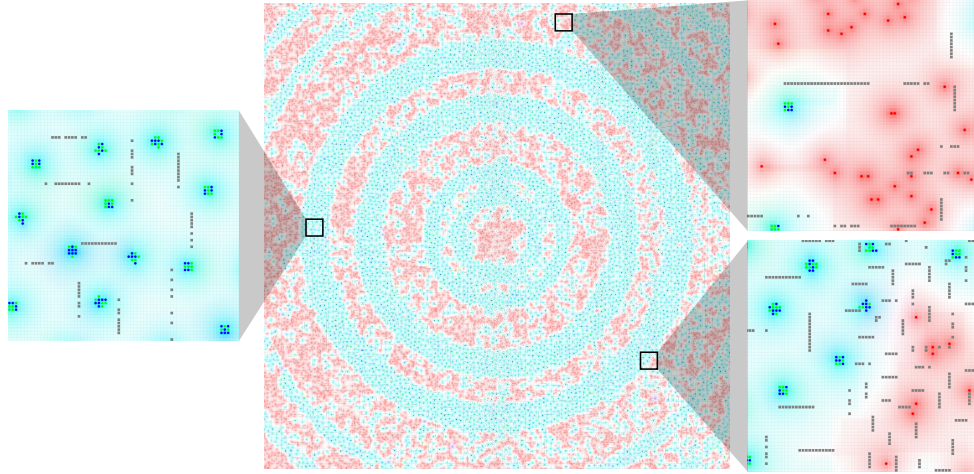


Figure 9.10: Visualization of an example environment with spatial non-stationarity. Each tile is colored according to its scent. JellyBeans are shown in blue, Bananas in green, and Onions in red. Walls are depicted as grey squares.

9.4.4 Example of Spatial Non-Stationarity

To demonstrate the ability to generate spatially non-stationary worlds in JBW, we provide a configuration that makes use of non-stationary intensity and interaction functions. The configuration is shown in Tables 9.6 and 9.7, and a visualization is provided in Figure 9.10. JellyBeans and Onions appear together in clusters and, since this configuration uses the non-stationary intensity function `RadialHash`, these clusters are arranged in concentric circles around the origin that are irregularly spaced. `RadialHash` uses a hash function to induce a pseudorandom relationship between the distance to the origin and the likelihood of finding such clusters. Walls in this environment also have a non-stationary distribution. In some regions, they are smaller and more frequent, whereas in other regions they are longer and appear more sporadically.

Map	Scent Dimensionality	3
	Color Dimensionality	3
	Patch Size	32×32
	MH Sampling Iterations	4,000
	Scent Decay (λ)	0.4
	Scent Diffusion (α)	0.14
Agent	Color	■ [0.00, 0.00, 0.00]
	Scent	■ [0.00, 0.00, 0.00]
	Action Space	MoveForward TurnLeft TurnRight
	Visual Range	5
	Field-of-View	experiment-specific

Table 9.6: Simulator configuration used for our non-stationary environment.

JellyBean: Jelly beans appear close to bananas.	
Scent	■ [0.0, 0.0, 1.0]
Color	■ [0.0, 0.0, 1.0]
Occlusion	0.0
Blocks Agents	False
Intensity	RadialHash[500,60,-3.0,14]
Interactions	JellyBean : PiecewiseBox[10,200,0,-6]
	Banana : PiecewiseBox[10,200,2,-100]
	Onion : PiecewiseBox[200,0,-100,-100]
Banana: Bananas appear close to jelly beans.	
Scent	■ [0.0, 1.0, 0.0]
Color	■ [0.0, 1.0, 0.0]
Occlusion	0.0
Blocks Agents	False
Intensity	RadialHash[500,60,-3.0,14]
Interactions	JellyBean : PiecewiseBox[10,100,2,-100]
	Banana : PiecewiseBox[10,200,0,-6]
	Onion : PiecewiseBox[200,0,-6,-6]
Onion: Onions appear scattered, away from jellybeans and bananas.	
Scent	■ [1.0, 0.0, 0.0]
Color	■ [1.0, 0.0, 0.0]
Occlusion	0.0
Blocks Agents	False
Intensity	Constant[-5]
Interactions	JellyBean : PiecewiseBox[200,0,-100,-100]
	Banana : PiecewiseBox[200,0,-6,-6]
Wall: Walls tend to be contiguous and axis-aligned.	
Scent	■ [0.0, 0.0, 0.0]
Color	■ [0.5, 0.5, 0.5]
Occlusion	1.0 in experiments with occlusion, 0.0 otherwise
Blocks Agents	True
Intensity	Constant[0]
Interactions	Wall : CrossHash[60,4,25,2,20,-200,-20,1]

Table 9.7: Item types for the non-stationary environment.

CONCLUSION

In the introduction of this thesis we set out to test the following hypothesis:

THESIS STATEMENT: *A computer system that learns to perform multiple tasks jointly and that is aware of the relationships between these tasks, will be able to learn more efficiently and effectively than a system that learns to perform each task in isolation. Moreover, the relationships between the tasks may either be explicitly provided through supervision or implicitly learned by the system itself, and will allow the system to self-reflect and evaluate itself without any task-specific supervision.*

In the following section we summarize our key results, discuss how they relate to this hypothesis, and provide evidence to support it. Then, we propose multiple avenues of research that this thesis opens up and that we consider interesting and potentially highly influential for future work.

10.1 KEY RESULTS

In [Chapter 2](#), we introduced the concept of estimating the error rate of each of several approximations to the same function, based on their agreement rates over *unlabeled data* and we provided three different analytical methods to do so. Our experimental results are encouraging and suggest that function agreement rates are indeed very useful in estimating function error rates and thus, answer our motivating question: *consistency does imply correctness, under certain independence assumptions*. We consider this work to be a first step towards developing a *self-reflection framework* for autonomous learning systems. However, in [Chapter 2](#) we also identified three key limitations of the proposed methods that were addressed in the following chapters. In [Chapter 3](#), we presented a method that is able to account for dependencies among the classifiers, and that is based on a probabilistic graphical model. In [Chapter 4](#), we proposed a method that is further able to account for logical constraints between the labels that the classifiers predict (e.g., a NP that refers to a city cannot also refer to an animal at the same time and therefore, if a classifier predicts that a given NP refers to a city and another one predicts that it refers to an animal, then at least one of them has to be wrong). In [Chapter 5](#), we proposed yet another method for tackling this problem that offers a highly robust algorithm and allows for learning directly from multiple noisy sources of supervision combined with self-supervision in an end-to-end fashion. Interestingly, the last method also achieves state-of-the-art performance among existing methods for aggregating crowdsourced labels. Then, in [Chapter 6](#), we showed how this method and the underlying idea can also be used to tackle other problems that may at first seem completely disconnected from the original setting that motivated this work. Specifically, we showed that it can be used for robust graph-based semi-supervised learning, and achieve state-of-the-art results

in this setting as well. This concluded [Part i](#) of this thesis and provided evidence that relationships between multiple functions that a system is learning can allow it to self-reflect and evaluate itself without any task-specific supervision, thus validating the second clause of our thesis statement.

In [Chapter 7](#), we provided a brief history of multi-task learning along with the current landscape of multi-task learning, and then proposed *contextual parameter generation* (CPG) as an effective abstraction for sharing information among multiple tasks that are being learned jointly. CPG also allows us to explicitly encode relationships between the tasks being learned. Then, in [Chapter 8](#) we presented multiple case studies for evaluating the effectiveness and usefulness of CPG. These case studies provided evidence that, when using CPG, learning to perform multiple tasks jointly while being aware of the relationships between these tasks, results in faster training and significantly improved performance, thus validating the first clause of our thesis statement and concluding [Part ii](#) of this thesis.

Finally, in [Chapter 9](#) we presented the *jelly bean world* (JBW), a novel framework that we designed for evaluating never-ending learning systems, and which allows us to control the kinds of problems the learning agents need to solve, and their interactions. We have designed the JBW in a way that renders never-ending learning necessary, and that allows us to test all parts of the never-ending learning thesis, in a controllable manner. In [Section 8.4](#), we also used this framework to showcase the effectiveness of contextual parameter generation in settings where there exists a compositional structure over the tasks that are being learned.

We have thus not only verified that our thesis statement holds true in multiple settings, but we have also proposed a novel evaluation framework that ought to allow researchers to evaluate other new approaches related to our thesis statement. However, multiple questions remain unanswered and, perhaps most importantly, our work has resulted in many more new questions that we hope will help stimulate further research in this direction. In the next section, we discuss some of these questions along with some related directions for future work.

10.2 FUTURE WORK

In our opinion, successful research should pave the way for further progress and research. To this end, we now present some future directions for the work presented in this thesis:

- In [Part i](#) we provided strong empirical evidence that consistency does imply correctness under certain conditions. However, we only have strong theoretical guarantees for the case when we have multiple noisy function approximations that make conditionally independent errors and where the majority of them have accuracies that are better than random guessing. We have no theoretical guarantees that this is true in general and no theoretical understanding of when it is true, other than the fact that it should work when the function approximations make conditionally independent errors. Therefore, an interesting direction

for future work would be to provide a theory that underlies the methods we proposed in [Part i](#) of this thesis.

- Similarly, in [Part ii](#) we proposed contextual parameter generation and obtained strong empirical evidence that it works well in practice. However, we have no good theoretical understanding as to why this happens. We only know that it has to do with how we share information across the different tasks. Therefore, understanding contextual parameter generation better, including when and why it works or does not work, is an interesting direction for future work.
- In [Part ii](#) we also briefly discussed that oftentimes there exists a compositional structure among the tasks being learned. In fact, composition lies at the core of category theory, a field that attempts to unify all of mathematics. It even attempts to unify mathematics with other fields. Therefore, understanding composition and how to efficiently learn composable functions could be of great interest to the machine learning and artificial intelligence communities.
- In [Section 7.2.3](#) we briefly described how contextual parameter generation could be used as a method for neural architecture search. Exploring this question would be an interesting direction for future work.
- Our work in this thesis has inspired us to propose a novel type of neural architectures for never-ending learning that we refer to as *neural cognitive architectures*. In fact, the model we proposed in [Section 8.4](#) is a simple instance of a neural cognitive architecture. We have briefly explored the design space of neural cognitive architectures based on our findings from this thesis and we present an extensive proposal in [Appendix C](#). Implementing and evaluating neural cognitive architectures as a step towards artificial general intelligence is of great interest to us, but it is outside the scope of this thesis.

10.3 KEY TAKEAWAYS

There are a few key takeaways from this thesis. First of all, our work in [Part i](#) shows that consistency among multiple function approximations is directly related to the correctness of these approximations. This relationship is governed by the dependencies that exist among these function approximations. The self-reflection methods we proposed enable using unlabeled data for training machine learning systems in ways that it was not possible before. In [Chapter 5](#) we presented a setting where we train a deep learning model using only noisy labels, and in [Chapter 6](#) we showed how a similar method can be used to train models in a semi-supervised graph node classification setting. Similarly, in [Part ii](#) we proposed a novel abstract framework for multi-task learning called contextual parameter generation (CPG). We showed how it significantly outperforms existing methods for multi-task learning and provided some intuition as to why this happens. As we discussed in [Chapter 1](#), multi-task learning is a natural big next step for the machine learning community and we believe that CPG may play an important role in making it possible. Specifically, we believe that designing neural cognitive architectures and compositional multi-task learning methods will move the community in the right direction. Finally, we hope that the questions raised by this thesis and that remain unanswered will lead to impactful research in the future.

Part IV

APPENDIX

A system that is capable of large-scale multi-task learning must be capable of learning to perform a potentially very large number of tasks, some of which may be related. For example, a common pattern that we encounter in human learning is that some of the tasks that humans perform can be broken down into smaller sub-tasks that they have previously learned to perform. We refer to this kind of structure as *task compositionality*. In order to be able to learn large numbers of tasks humans tend to use various forms of *curriculum learning*, meaning that they learn tasks in a particular order that helps accelerate learning. This turns out to be especially important for certain kinds of tasks (e.g., solving math problems). Inspired by this characteristic of human learning we propose *competence-based curriculum learning*, a new curriculum learning paradigm that can be used to enable accelerate learning for computer systems.

For this chapter, we use machine translation as the example target application for curriculum learning. This is motivated by the fact that machine translation offers good and established benchmarking tasks for evaluating models, while at the same time offering difficult and active research problems. Moreover, as we explain in the following section, the current state-of-the-art models for machine translation are notoriously hard to train, which makes them an ideal target for curriculum learning approaches. As we shall see, it is also quite easy to define heuristics for judging how hard different sentences are to translate, and this can be used in the context of curriculum learning. Note that, machine translation is also especially convenient for us because of the infrastructure we developed for our earlier machine translation work that was presented in [Section 8.2](#). In the next section, we provide some background on machine translation and prior work attempting to apply curriculum learning in this domain. We then present our general-purpose curriculum learning framework, before concluding with an extensive experimental evaluation of the proposed framework in the context of machine translation.¹

A.1 NEURAL MACHINE TRANSLATION AND CURRICULUM LEARNING

Neural Machine Translation (NMT; Kalchbrenner and Blunsom, 2013; Bahdanau et al., 2015) now represents the state-of-the-art approach adopted in most machine translation systems (Crego et al., 2016; Wu et al., 2016a; Bojar et al., 2017), largely due to its ability to benefit from end-to-end training on massive amounts of data. In particular, the recently-introduced self-attentional Transformer architectures by Vaswani et al. (2017) are rapidly becoming the de-facto standard in NMT, having demonstrated both superior performance and training speed compared to previous architectures using recurrent neural networks (RNNs; Kalchbrenner and Blunsom,

¹ The work presented in this chapter has been previously published in (Platanios et al., 2019).

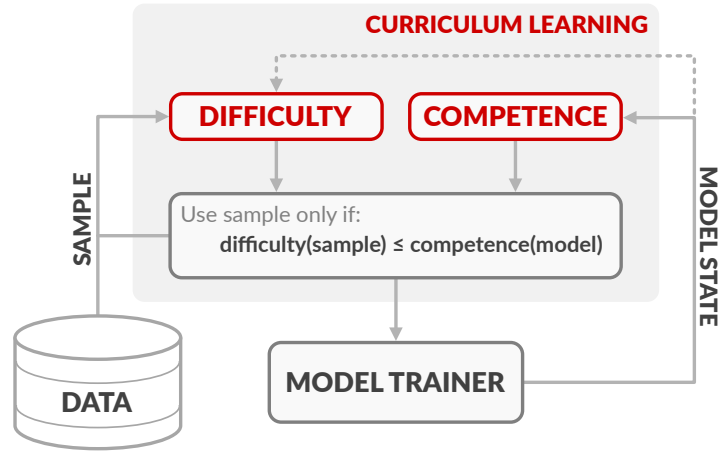


Figure A.1: Overview of the competence-based curriculum learning framework. During training, *difficulty* of each training example is estimated and a decision on whether to use it is made based on the current *competence* of the model.

2013; Sutskever et al., 2014). However, large scale NMT systems are often hard to train, requiring complicated heuristics which can be both time-consuming and expensive to tune. This is especially true for Transformers which have been shown to consistently outperform RNNs (Popel and Bojar, 2018), but they do so while relying on a number of heuristics such as specialized learning rates and large-batch training.

In this chapter, we tackle this problem by proposing a curriculum learning framework for training NMT systems that reduces training time, reduces the need for specialized heuristics or large batch sizes, and results in overall better performance. It allows us to train both RNNs and, perhaps more importantly, Transformers, with relative ease. The proposed approach is based on the idea of teaching algorithms in a similar manner to humans, starting with easy concepts and moving to more difficult ones later on. This idea can be traced back to the work of Elman (1993) and Krueger and Dayan (2009). The main motivation is that training algorithms can perform better if training data is presented in a specific order, starting from *easy* examples and moving on to more *difficult* ones, as the learner becomes more *competent*. In the case of machine learning, it can also be thought of as a means to avoid getting stuck in bad local optima early on in training. An overview of the proposed framework is shown in Figure A.1.

Notably, we are not the first to examine curriculum learning for NMT, although other related works have been met with mixed success. Kocmi and Bojar (2017) explore the impact of several curriculum heuristics on training a translation system for a single epoch, presenting the training examples in an easy-to-hard order based on sentence length and vocabulary frequency. However, their strategy introduces all training examples during the first epoch and it is not clear how this affects learning in the following epochs with official evaluation results (Bojar et al., 2017) indicating that the final performance may be hurt when using this strategy. Contemporaneously to our work, Zhang et al. (2018) further propose to split the training examples into a predefined number of bins (5, in their case), based on various difficulty metrics. A

manually designed curriculum schedule then specifies the bins from which the model samples training examples. Experiments demonstrate that benefits of curriculum learning are highly sensitive to several hyperparameters (e.g., learning rate, number of iterations spent in each phase, etc.), and largely provide benefits in convergence speed as opposed to final model accuracy.

In contrast to these previous approaches, we define a continuous curriculum learning method—instead of a discretized regime—with only one tunable hyperparameter (the duration of curriculum learning). Furthermore, as opposed to previous work which only focuses on RNNs, we also experiment with Transformers which are notoriously hard to train (Popel and Bojar, 2018). Finally, unlike any of the previous work, we show that our curriculum approach helps not only in terms of convergence speed, but also in terms of the learned model performance. In summary, our method has the following desirable properties:

1. Abstract: It is a novel, generic, and extensible formulation of curriculum learning. A number of previous heuristic-based approaches, such as that of Kocmi and Bojar (2017), can be formulated as special cases of our framework.
2. Simple: It can be applied to existing NMT systems with only a small modification to their training data pipelines.
3. Automatic: It does not require any tuning other than picking the value of a single parameter, which is the length of the curriculum (i.e., for how many steps to use curriculum learning, before easing into normal training).
4. Efficient: It reduces training time by up to 70%, whereas contemporaneous work of Zhang et al. (2018) reports reductions of up to 46%.
5. Improved Performance: It improves the performance of the learned models by up to 2.2 BLEU points, where the best setting reported by Zhang et al. (2018) achieves gains of up 1.55 BLEU after careful tuning.

A.2 COMPETENCE-BASED CURRICULUM LEARNING

We propose *competence-based curriculum learning*, a training framework based on the idea that training algorithms can perform better if training data is presented in a way that aligns with the model’s current *competence*. More specifically, we define the following two concepts that are central to our framework:

DIFFICULTY. A value that represents the difficulty of a training example and that may depend on the current state of the learner. For example, sentence length is an intuitive difficulty metric for natural language processing tasks. The only constraint is that difficulty scores are comparable across different training examples (i.e., the training examples can be ranked according to their difficulty).

COMPETENCE. A value between 0 and 1 that represents the progress of a learner during its training. It is defined as a function of the learner’s state. More specifically, we define the competence $c(t)$ at time t (measured in terms of training steps) of a learner as the proportion of training data it is allowed to use at that time. The training examples are ranked according to their difficulty and the learner is only allowed to

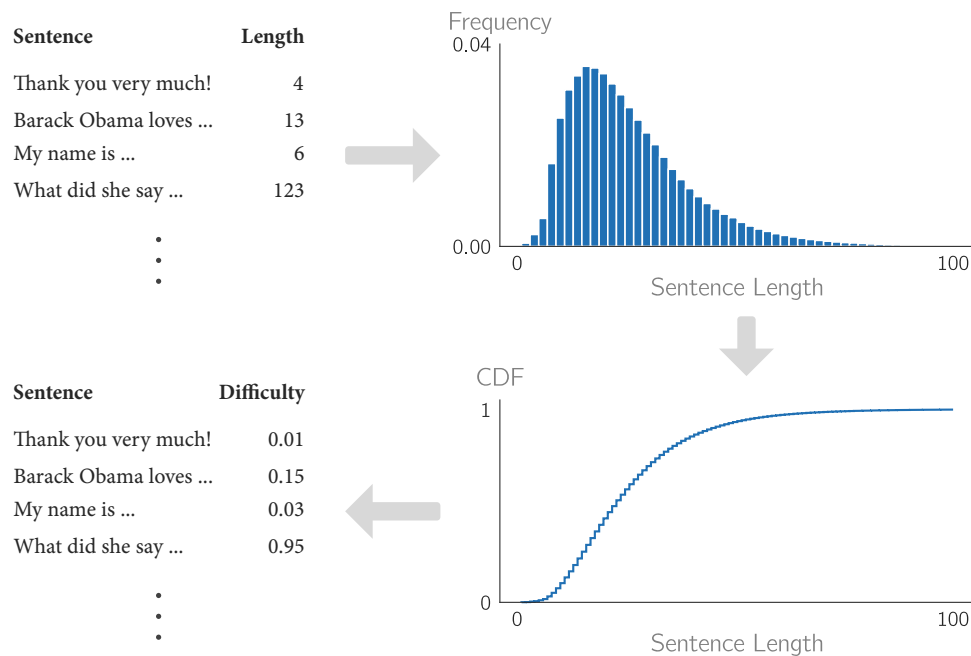


Figure A.2: Example illustration of the preprocessing sequence used in the proposed algorithm. The histogram on the top is that of sentence lengths from the WMT-16 En→De dataset used in our experiments. Here sentence lengths represent an example difficulty scoring function, d . “CDF” stands for the empirical “cumulative density function” obtained from the histogram in the top plot.

use the top $c(t)$ portion of them at time t .

Using these two concepts, we propose the algorithm shown in [Algorithm A.1](#). A high-level overview is shown in [Figure A.1](#), an example visualization of the first two steps is shown in [Figure A.2](#), and an example of the interaction between difficulty and competence is shown in [Figure A.3](#). Note that, at each training step, we are not changing the relative probability of each training example under the input data distribution, but we are rather constraining the domain of that distribution based on the current competence of the learner. Eventually, once the competence becomes 1, the training process becomes equivalent to the one that does not use a curriculum, with the main difference that the learner should now be more capable to learn from the more difficult examples. Given the dependence of this algorithm on the specific choices of the difficulty scoring function d and the competence function c , we now describe our instantiations for training NMT models.

A.2.1 Difficulty Metrics

There are many possible ways to define the difficulty of translating a sentence. We consider two heuristics inspired by what we, as humans, may consider difficult when translating, and by factors which can negatively impact the optimization algorithms used when training NMT models. In the rest of this section, we denote our training

Algorithm A.1: Competence-based curriculum learning algorithm.**Inputs:** Dataset, $\mathcal{D} = \{s_i\}_{i=1}^M$, consisting of M examples.Model trainer, $\mathcal{T}$, that uses batches of training data for each update.Difficulty scoring function, d .Competence function, c .

- 1 Compute the difficulty, $d(s_i)$, for each $s_i \in \mathcal{D}$.
- 2 Compute the cumulative density function of the difficulty scores. This results in one difficulty score per example, $\bar{d}(s_i) \in [0, 1]$. Illustrated in [Figure A.2](#).
- 3 **for** *training step* $t = 1, \dots$ **do**
- 4 Compute the model competence, $c(t)$.
- 5 Sample a data batch B_t uniformly from all $s_i \in \mathcal{D}$, such that $\bar{d}(s_i) \leq c(t)$. Illustrated in [Figure A.3](#).
- 6 Invoke the trainer, $\mathcal{T}$, using B_t as input.

Output: Trained model.

corpus as a collection of M sentences, $\{s_i\}_{i=1}^M$, where each sentence is a sequence of words: $s_i = \{w_0^i, \dots, w_{N_i}^i\}$.

SENTENCE LENGTH. We argue that it is harder to translate longer sentences, as longer sentences require being able to translate their component parts, which often consist of short sentences. Furthermore, longer sentences are intuitively harder to translate due to the propagation of errors made early on when generating sentences in the target language. Therefore, a simple way to define the difficulty of a sentence $s_i = \{w_0^i, \dots, w_{N_i}^i\}$ is as follows:

$$d_{\text{length}}(s_i) \triangleq N_i. \quad (\text{A.1})$$

Note that, we can compute this difficulty metric on either the source language sentence or the target language sentence. We only consider the source sentence in this chapter.²

WORD RARITY. Another aspect of language that can affect the difficulty of translation is the frequency with which words appear. For example, humans may find rare words hard to translate because we rarely ever see them and it may be hard to recall their meaning. The same can be true for NMT models where: (i) the statistical strength of the training examples that contain rare words is low and thus the model needs to keep revisiting such words in order to learn robust representations for them, and (ii) the gradients of the rare word embeddings tend to have high variance; they are overestimates of the true gradients in the few occasions where they are non-zero, and underestimates otherwise. This suggests that using word frequencies may be a

² NMT models typically first pick up information about producing sentences with the correct length. It can be argued that presenting only short sentences first may lead to learning a strong bias for the sentence lengths. In our experiments, we did not observe this to be an issue as the models kept improving and predicting sentences of the correct length, throughout training.

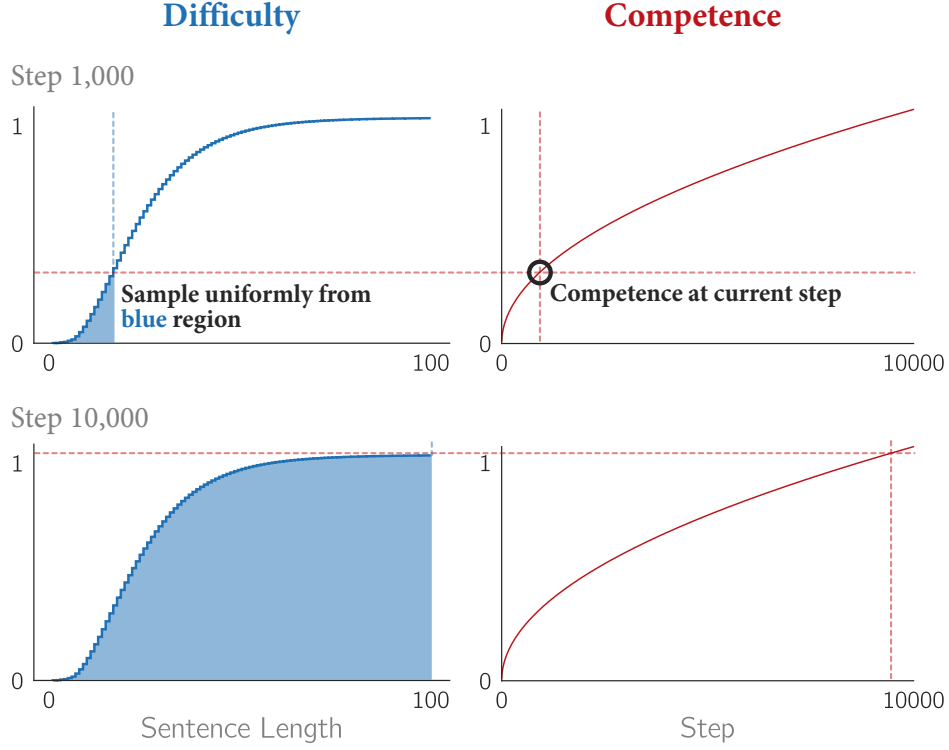


Figure A.3: Example illustration of the training data “filtering” performed by our curriculum learning algorithm. At each training step: (i) the current competence of the model is computed, and (ii) a batch of training examples is sampled uniformly from all training examples whose difficulty is lower than that competence. In this example, we are using the sentence length difficulty heuristic shown in Equation A.1, along with the square root competence model shown in Equation A.8.

helpful difficulty heuristic. Given a corpus of sentences, $\{s_i\}_{i=1}^M$, we define relative word frequencies as:

$$\hat{p}(w_j) \triangleq \frac{1}{N_{\text{total}}} \sum_{i=1}^M \sum_{k=1}^{N_i} \mathbb{1}_{\{w_k^i = w_j\}}, \quad (\text{A.2})$$

where $j = 1, \dots, \#\{\text{unique words in corpus}\}$ and $\mathbb{1}_{\{\cdot\}}$ evaluates to one if its subscript statement is true and to zero otherwise. Next, we need to decide on how to aggregate the relative word frequencies of all words in a sentence to obtain a single difficulty score for the sentence. Previous research has proposed various pooling operations, such as minimum, maximum, and average (Zhang et al., 2018), but they show that they do not work well in practice. We propose a different approach. Ultimately, what may be most important is the overall likelihood of a sentence as it contains information about both word frequency and, implicitly, about sentence length. An approximation to this likelihood is the product of the unigram probabilities, which is related to previous work in the area of active learning (Settles and Craven, 2008). This product can be thought of as an approximate language model (assuming words are sampled

independently) and also implicitly incorporates information about the sentence length that was proposed earlier (longer sentence scores are products over more terms in $[0, 1]$ and are thus likely to be smaller). We thus propose the following difficulty heuristic:

$$d_{\text{rarity}}(s_i) \triangleq - \sum_{k=1}^{N_i} \log \hat{p}(w_k^i), \quad (\text{A.3})$$

where we use logarithms of probabilities to prevent numerical errors. Note that negation is used because we define less likely (i.e., more rare) sentences as more difficult.

These are just two example difficulty metrics and it is easy to conceive others, such as the occurrence of homographs (Liu et al., 2018) or context-sensitive words (Bawden et al., 2018), the examination of which we leave for future work.

A.2.2 Competence Functions

For the purposes of this chapter, we propose two simple functional forms for $c(t)$ and justify them with some intuition. More sophisticated strategies that depend on the loss function, the loss gradient, or on the learner’s performance on held-out data, are possible, but we do not consider them in this chapter.

LINEAR. This is a simple way to define $c(t)$. Given an initial value $c_0 \triangleq c(0) \geq 0$ and a slope parameter r , we define:

$$c(t) \triangleq \min(1, tr + c_0). \quad (\text{A.4})$$

In this case, new training examples are constantly being introduced during the training process, with a constant rate r (as a proportion of the total number of available training examples). Note that we can also define $r = (1 - c_0)/T$, where T denotes the time after which the learner is fully competent, which results in:

$$c_{\text{linear}}(t) \triangleq \min\left(1, t \frac{1 - c_0}{T} + c_0\right). \quad (\text{A.5})$$

ROOT. In the case of the linear form, the same number of new and more difficult examples are added to the training set at all times t . However, as the training data grows in size, it gets less likely that any single example will be sampled in a training batch. Thus, given that the newly added examples are less likely to be sampled, we propose to reduce the number of new training examples per unit time as training progresses in order to give the learner sufficient time to assimilate their information content. More specifically, we define the rate in which new examples are added as inversely proportional to the current training data size:

$$\frac{dc(t)}{dt} = \frac{P}{c(t)}, \quad (\text{A.6})$$

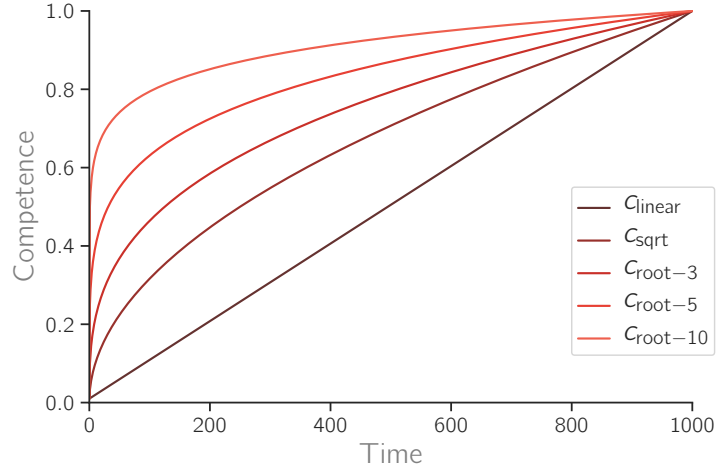


Figure A.4: Plots of various competence functions with $c_0 = 0.01$ (initial competence value) and $T = 1,000$ (total duration of the curriculum learning phase).

for some constant $P \geq 0$. Solving this simple differential equation, we obtain:

$$\int c(t)dc(t) = \int Pdt \Rightarrow c(t) = \sqrt{2Pt + D}, \quad (\text{A.7})$$

for some constants P and D . Then, we consider the following constraint: $c_0 \triangleq c(0) = \sqrt{D} \Rightarrow D = c_0^2$. We also have that $c(T) = 1 \Rightarrow P = (1 - c_0^2)/2T$, where T denotes the time after which the learner is fully competent. This, along with the constraint that $c(t) \in [0, 1]$ for all $t \geq 0$, results in the following definition:

$$c_{\text{sqrt}}(t) \triangleq \min \left(1, \sqrt{t \frac{1 - c_0^2}{T} + c_0^2} \right). \quad (\text{A.8})$$

In our experiments, we refer to this specific formulation as the “square root” competence model. If we want to make the curve *sharper*, meaning that even more time is spent per example added later on in training, then we can consider the following more general form:

$$c_{\text{root-}p}(t) \triangleq \min \left(1, \sqrt[p]{t \frac{1 - c_0^p}{T} + c_0^p} \right), \quad (\text{A.9})$$

for $p \geq 1$. We observed that best performance is obtained when $p = 2$ and then, as we increase p , the performance converges to that obtained when training without using any curriculum. Plots of the competence functions discussed in this section are shown in [Figure A.4](#).

A.2.3 Scalability

Our method can be easily used to train large-scale NMT systems. This is because it mainly consists of a preprocessing step of the training data that computes the difficulty scores. The implementation we are releasing with this thesis computes these scores in an efficient manner by building a graph describing their dependencies, as well as whether they are sentence-level scores (e.g., sentence length), or corpus-level (e.g., CDF), and using this graph to optimize their execution. Using only 8GB of memory, we can process up to 20k sentences per second when computing sentence rarity scores, and up to 150k sentences per second when computing sentence length scores.

A.3 EXPERIMENTS

For our experiments we use three of the most commonly used datasets in NMT, that range from a small benchmark dataset to a large-scale dataset with millions of sentences. Statistics about these datasets are shown in [Table A.1](#). We perform experiments using both RNNs and Transformers. For the RNN experiments we use a bidirectional LSTM for the encoder, and an LSTM with the attention model of Bahdanau et al. (2015) for the decoder. The number of layers of the encoder and the decoder are equal. We use a 2-layer encoder and a 2-layer decoder for all experiments on the IWSLT datasets, and a 4-layer encoder and a 4-layer decoder for all experiments on the WMT dataset, due to this dataset’s significantly larger size. For the Transformer experiments we use the “Base” model proposed by Vaswani et al. (2017). It consists of a 6-layer encoder and decoder, using 8 attention heads and 2,048 units for the feed-forward layers. The multi-head attention keys and values depth is set equal to the word embedding size. The word embedding size is 512 for all experiments. Furthermore, for the Transformer experiments on the two smaller datasets we do not use any learning rate schedule, and for the experiments on the largest dataset we use the default Transformer schedule. A detailed discussion on learning rate schedules for Transformers is provided towards the end of this section. All of our experiments are conducted on a machine with a single Nvidia V100 GPU, and 24 GBs of system memory.

A.3.1 Setup

During training, we use a label smoothing factor of 0.1 (Wu et al., 2016a) along with the AMSGrad optimizer (Reddi et al., 2018) using its default parameters in TensorFlow and a batch size of 5,120 tokens (due to GPU memory constraints). During inference, we employ beam search with a beam size of 10 and the length normalization scheme of Wu et al. (2016a).³

³ We emphasize that we did not run experiments with other architectures or configurations, and thus our baseline architectures were not chosen because they were favorable to our method, but rather because they were frequently mentioned in existing literature.

Dataset	# Train	# Dev	# Test
IWSLT-15 En→Vi	133k	768	1268
IWSLT-16 Fr→En	224k	1080	1133
WMT-16 En→De	4.5M	3003	2999

Table A.1: Number of parallel sentences in each dataset. “k” stands for “thousand” and “M” stands for “million.”

CURRICULUM HYPERPARAMETERS. We set the initial competence c_0 to 0.01, in all experiments. This means that all models start training using the 1% easiest training examples. The curriculum length T is effectively the only hyperparameter that we need to set for our curriculum learning methods. In each experiment, we set T in the following manner: we train the baseline model without using any curriculum and we compute the number of training steps it takes to reach approximately 90% of its final BLEU score. We then set T to this value. This results in T being set to 5,000 for the RNN experiments on the IWSLT datasets, and 20,000 for the corresponding Transformer experiments. For WMT, we set T to 20,000 and 50,000 for RNNs and Transformers, respectively. Furthermore, we use the following notation and abbreviations when presenting our results:

- Plain: Trained without using any curriculum.
- SL: Curriculum with sentence length difficulty.
- SR: Curriculum with sentence rarity difficulty.
- Linear: Curriculum with the linear competence shown in [Equation A.5](#).
- Sqrt: Curriculum with the square root competence shown in [Equation A.8](#).

DATA PREPROCESSING. Our experiments are performed using the machine translation library we developed for our work presented in [Section 8.2](#) (Platanios, 2018b). We also use the same data preprocessing approach as we did for [Section 8.2](#). While training, we consider sentences up to length 200. Similar to [Section 8.2](#), for the IWSLT-15 experiments we use a per-language vocabulary which contains the 20,000 most frequently occurring words while ignoring words that appear less than 5 times in the corpus. For the IWSLT-16 and WMT-16 experiments we use a byte-pair encoding (BPE) vocabulary (Sennrich et al., 2016b) trained using 32,000 merge operations, similar to the original Transformer paper by Vaswani et al. (2017).

A.3.2 Results

We present a summary of our results in [Table A.2](#) and we also show complete learning curves for all methods in [Figure A.5](#). The evaluation metrics we use are the test set BLEU score and the time it takes for the models using curriculum learning to obtain the BLEU score that the baseline models attain at convergence. We observe that Transformers consistently benefit from our curriculum learning approach, achieving gains of up to 2 BLEU, and reductions in training time of up to 70%. RNNs also benefit, but to a lesser extent. This is consistent with our initial motivation, which stems from the observation that training RNNs is easier and more robust than training Transformers. Furthermore, the square root competence model consistently outperforms the linear

		RNN					Transformer					
		Plain	SL Curriculum		SR Curriculum		Plain	Plain*	SL Curriculum		SR Curriculum	
			Clinear	Csqrt	Clinear	Csqrt			Clinear	Csqrt	Clinear	Csqrt
BLEU	En→Vi	26.27	26.57	27.23	26.72	26.87	28.06	29.77	29.14	29.57	29.03	29.81
	Fr→En	31.15	31.88	31.92	31.39	31.57	34.05	34.88	34.98	35.47	35.30	35.83
	En→De	26.53	26.55	26.54	26.62	26.62	–	27.95	28.71	29.28	29.93	30.16
Time	En→Vi	1.00	0.64	0.61	0.71	0.57	1.00	1.00	0.44	0.33	0.35	0.31
	Fr→En	1.00	1.00	0.93	1.10	0.73	1.00	1.00	0.49	0.44	0.42	0.39
	En→De	1.00	0.86	0.89	1.00	0.83	–	1.00	0.58	0.55	0.55	0.55

Table A.2: Summary of experimental results. For each method and dataset, we present the test set BLEU score of the best model based on validation set performance. We also show the relative time required to obtain the BLEU score of the best performing baseline model. For example, if an RNN gets to 26.27 BLEU in 10,000 steps and the SL curriculum gets to the same BLEU in 3,000 steps, then the plain model gets a score of 1.0 and the SL curriculum receives a score of $3,000/10,000 = 0.3$. Plain stands for the model trained without a curriculum and, for Transformers, Plain* stands for the model trained using the learning rate schedule shown in Equation A.10.

model, which fits well with our intuition and motivation for introducing it. Regarding the difficulty heuristics, sentence length and sentence rarity both result in similar performance. For the two small datasets, RNNs converge faster than Transformers in terms of both the number of training iterations and the overall training time. This is contrary to other results in the machine translation community (e.g., Vaswani et al., 2017), but could be explained by the fact that we are not using any learning rate schedule for training Transformers. However, RNNs never manage to outperform Transformers in terms of test BLEU score of the final model. Furthermore, to the best of our knowledge, for IWSLT-15 we achieve state-of-the-art performance. The highest previously reported result was 29.03 BLEU (which is in fact obtained by our method, presented in Section 8.2), in a multi-lingual setting. Using our curriculum learning approach we are able to achieve a BLEU score of 29.81 for this dataset. Overall, we have shown that *our curriculum learning approach consistently outperforms models trained without any curriculum, in both limited data settings and large-scale settings.*

A.3.3 Learning Rate Schedule

In all of our IWSLT experiments so far, we have used the default AMSGrad learning rate of 0.001 and intentionally avoid using any learning rate schedules. However, Transformers are not generally trained without a learning rate schedule. Such schedules typically use a warm-up phase, which means that the learning rate starts at a very low value and keeps increasing until the end of the warm-up period, after which a decay rate is typically used. In order to show that our curriculum learning approach can act as a principled alternative to such highly tuned learning rate schedules, we now present the results we obtain when training our Transformers using the following learning rate schedule:

$$\text{lr}(t) \triangleq d_{\text{embedding}}^{-0.5} \min \left(t^{-0.5}, t \cdot T_{\text{warmup}}^{-1.5} \right), \quad (\text{A.10})$$

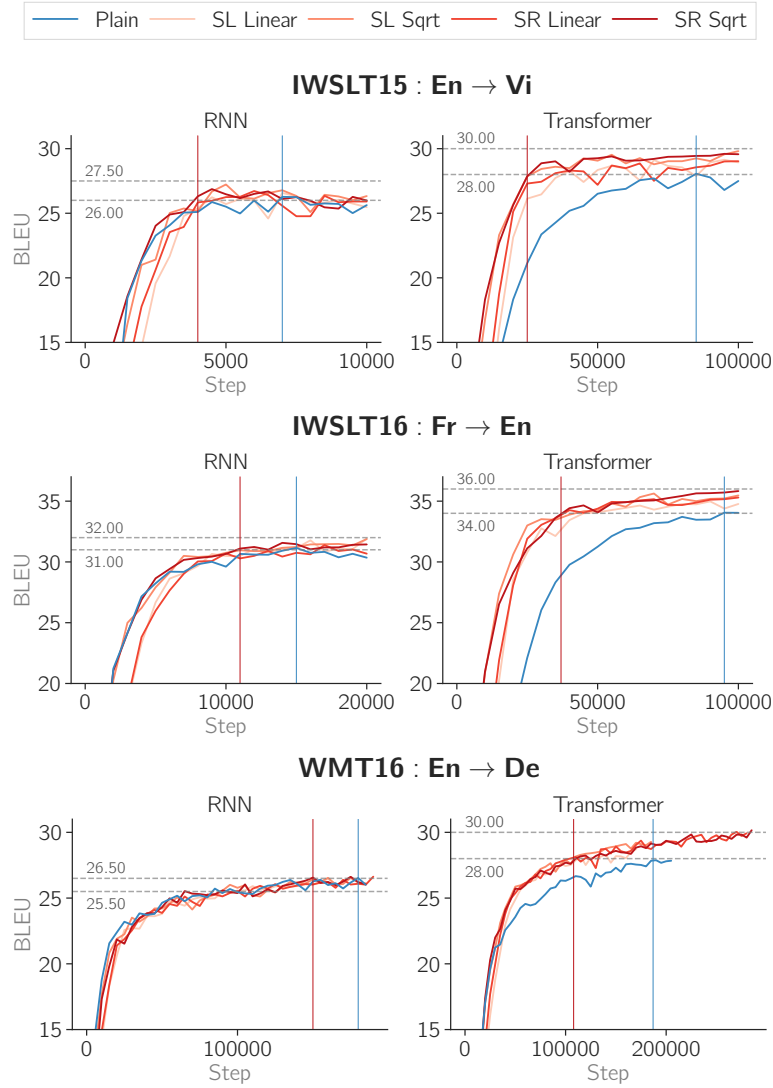


Figure A.5: Plots illustrating the performance of various models on the test set, as training progresses. Blue lines represent the baseline methods when no curriculum is used and red lines represent the same models when different versions of our curriculum learning framework are used to train them. The vertical lines represent the step in which the models attain the BLEU score that the baseline models attain at convergence.

where t is the current training step, $d_{\text{embedding}}$ is the word embedding size, and T_{warmup} is the number of warmup steps and is set to 10,000 in these experiments. This schedule was proposed in the original Transformer paper (Vaswani et al., 2017), and was tuned for the WMT dataset. The results obtained when using this learning rate schedule are also shown in Table A.2, under the name Plain*. In both cases, our curriculum learning approach obtains a better model in about 70% less training time. This is very important, especially when applying Transformers in new datasets, because such learning rate heuristics often require careful tuning. This tuning can be both very expensive and

time consuming, often resulting in very complex mathematical expressions, with no clear motivation or intuitive explanation (Chen et al., 2018). Our curriculum learning approach achieves better results in significantly less time, while only requiring one parameter (the length of the curriculum). Note that even without using any learning rate schedule, our curriculum methods were able to achieve performance comparable to the Plain* in about twice as many training steps. Plain was not able to achieve a BLEU score above 2.00 even after five times as many training steps, at which point we stopped these experiments.

A.3.4 *Implementation and Reproducibility*

We are releasing an implementation of our proposed method and experiments built on top of the machine translation library we developed as part of our work in Section 8.2 (Platanios, 2018b), using TensorFlow Scala (Platanios, 2018c). The implementation is available at <https://github.com/eaplatanios/symphony-mt>. Furthermore, all experiments were run on a machine with a single Nvidia V100 GPU, and 24 GBs of system memory. Our most expensive experiments—the ones using Transformers on the WMT-16 dataset—take about 2 days to complete, which would cost about \$125 on a cloud computing service such as Google Cloud or Amazon Web Services, thus making our results reproducible, even by independent researchers.

A.4 RELATED WORK

The idea of teaching algorithms in a similar manner as humans, from easy concepts to more difficult ones, has existed for a long time (Elman, 1993; Krause et al., 2016). Machine learning models are typically trained using stochastic gradient descent methods, by uniformly sampling mini-batches from the pool of training examples, and using them to compute updates for the model parameters. Deep neural networks, such as RNNs and Transformers, have highly non-convex loss functions. This makes them prone to getting stuck in saddle points or bad local minima during training, often resulting in long training times and bad generalization performance. Bengio et al. (2009) propose a curriculum learning approach that aims to address these issues by changing the mini-batch sampling strategy. They propose starting with a distribution that puts more weight on easy examples, and gradually increase the probability of more difficult examples as training progresses, eventually converging to a uniform distribution. They demonstrate empirically that such curriculum approaches do indeed help decrease training times and sometimes even improve generalization.

Perhaps the earliest attempt to apply curriculum learning in MT was made by Zou et al. (2013). The authors employed a curriculum learning method to learn Chinese-English bilingual word embeddings, which were subsequently used in the context of phrase-based machine translation. They split the word vocabulary in 5 separate groups based on word frequency, and learned separate word embeddings for each of these groups in parallel. Then, they merged the 5 different learned embeddings and continued training using the full vocabulary. While this approach makes use of some of the ideas behind curriculum learning, it does not directly follow the original

definition introduced by Bengio et al. (2009). Moreover, their model required 19 days to train. There have also been a couple of attempts to apply curriculum learning in NMT that were discussed in the beginning of this chapter.

There also exists some relevant work in areas other than curriculum learning. Zhang et al. (2016) propose training neural networks for NMT by focusing on hard examples rather than easy ones. They report improvements in BLEU score, while only using the hardest 80% training examples in their corpus. This approach is more similar to boosting by Schapire (1999), rather than curriculum learning, and it does not help speed up the training process; it rather focuses on improving the performance of the trained model. The fact that hard examples are used instead of easy ones is interesting because it is somewhat contradictory to that of curriculum learning. Also, in contrast to curriculum learning, no ordering of the training examples is considered.

Perhaps another related area is that of active learning where the goal is to develop methods that request for specific training examples. Haffari et al. (2009), Bloodgood and Callison-Burch (2010), and Ambati (2012) all propose methods that can be used to solicit training examples for MT systems, based on the occurrence frequency of n-grams in the training corpus. The main idea is that if an n-gram is very rare in the training corpus, then it is difficult to learn to translate sentences in which it appears. This is related to our sentence rarity difficulty metric and points out an interesting connection between curriculum learning and active learning.

Regarding training Transformer networks, Shazeer and Stern (2018) perform a thorough experimental evaluation of Transformers, when using different optimization configurations. They show that a significantly higher level of performance can be reached by not using momentum during optimization, as long as a carefully chosen learning rate schedule is used. Such learning rate schedules are often hard to tune because of the multiple seemingly arbitrary terms they often contain. Furthermore, Popel and Bojar (2018) show that, when using Transformers, increasing the batch size results in a better model at convergence. We believe this is indicative of very noisy gradients when starting to train Transformers and that higher batch sizes help increase the signal-to-noise ratio. We show that our proposed curriculum learning method offers a more principled and robust way to tackle this problem allowing us to train Transformers to state-of-the-art performance, using small batch sizes and without the need for peculiar learning rate schedules, which are typically necessary.

A.5 KEY TAKEAWAYS

We have presented a novel competence-based curriculum learning approach for training neural machine translation models. Our resulting framework is able to boost the performance of existing NMT systems, while at the same time significantly reducing their training time. It differs from previous approaches in that it does not depend on multiple hyperparameters that can be hard to tune, and it does not depend on a manually designed discretized training regime. We define the notions of competence, for a learner, and difficulty, for the training examples, and propose a way to filter training data based on these two quantities. Interestingly, we show that our method makes training Transformers faster and more reliable, but has a much smaller effect

in training RNNs. Perhaps most importantly though, the curriculum learning framework we presented in this chapter is not specific to machine translation, but is rather more generally applicable to machine learning. Our main goal was to show how a curriculum learning framework—that is inspired by human learning—can result in faster training for machine learning systems. Ultimately, we hope that this framework will become an integral part of the machine learning workflow.

Tasks which involve learning several classifiers whose outputs are tied together by logical constraints are abundant in machine learning. As an example, we may have two classifiers in the Never Ending Language Learning (NELL) project (Carlson et al., 2010; Mitchell et al., 2015, 2018) which predict whether noun phrases represent animals or cities, respectively. In this case, the outputs of the two classifiers are mutually exclusive. In Chapter 4 we presented a method which allows us to evaluate such classifiers using only unlabeled data, by leveraging the information provided by these logical constraints. In this chapter, we address the issue that many such tasks hinge on the training of a large number of classifiers in situations where obtaining labeled data is expensive. The difficulty of acquiring labels leads to the common approach (highlighted in Figure B.1) of performing an initial training of classifiers with a small number of labeled examples, and then iteratively identifying the most valuable additional labels to acquire, followed by re-training of the classifiers. We seek methods that are capable of performing such *active learning* (Settles, 2012)—an instance of *semi-supervised learning*. To this end, we propose methods for active learning that share a common underlying goal: efficient identification of the most valuable labels to acquire in the presence of *logical constraints* among the outputs of the classifiers being trained. Examples of such constraints include mutual exclusion (e.g., in multi-class/one-vs-all classification) and subsumption (e.g., in hierarchical classification) among target variables, which were also discussed in Chapter 4. In active learning for mutual exclusion and subsumption, we need to consider the complexities of behavior arising in the interactions among the linked classifiers. We shall provide theoretical justification for the proposed methods that resonates with intuition. As we shall show, our results challenge the core idea behind *uncertainty guided sampling*, a method commonly used in practice.¹

B.1 MOTIVATION

We motivate our work with challenges in information extraction, where noun phrases are mapped to various categories (e.g., `animal`, and `bird`) and relations between them. It is easy to see how these categories and relations can be tied through logical constraints. For example, one might say that `animal` and `location` are mutually exclusive, and `animal` subsumes `bird`. We consider examples highlighted by our work on the NELL project (Mitchell et al., 2018). NELL currently performs over 4,100 learning tasks and it is thus too expensive to obtain enough labeled data for each task separately. The ability to rank examples by the utility of discovering their labels would enable the system to more efficiently allocate and use resources available for labeling. Our goal

¹ The work presented in this chapter has been previously published in (Platanios et al., 2017a).

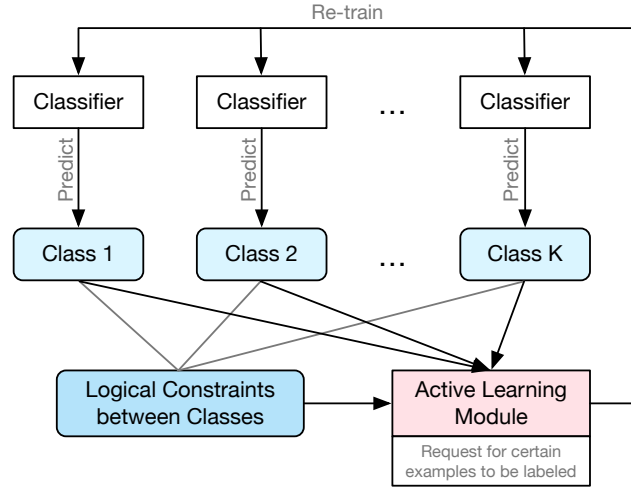


Figure B.1: Illustration of active learning in an interdependent multiple classifier setting.

in this chapter is to provide systems such as NELL with this ability, such that their learning rate would be significantly increased with respect to the resources available for labeling.

B.2 RELATED WORK

The literature covers many projects in the realms of active learning (Settles, 2012; Ruvolo and Eaton, 2013) and decision theory (e.g., the core concept of *value of information* and its use in guiding the collection of examples; Kapoor et al., 2007; Krause and Guestrin, 2009). Related work on computing the value of information for inference that leverages structural information includes an effort showing how the topology of influence diagrams could be used to assert an ordering over the value of information for variables (Poh and Horvitz, 1996). However, most existing approaches to active information gathering for machine learning are directed at collecting single labels for one classifier. Furthermore, even approaches that deal with settings involving multiple labels do not make use of logical constraints that may exist among labels (Reichart et al., 2008; Zhao et al., 2015). Work in the area of semi-supervised learning makes it clear that such constraints are present in many practical settings and that they can indeed prove useful if used appropriately (Chang et al., 2007; Chang et al., 2008; Mitchell et al., 2015, 2018). There have been a few approaches that make use of such constraints. First, we note that query-by-committee (QBC) can be viewed as a special case of our framework, where the logical constraint is that committee members must agree. Settles and Craven (2008) propose approaches to perform active learning for sequence labeling tasks, including uncertainty sampling and QBC. Culotta and McCallum (2005) consider adding constraints to such tasks. To the best of our knowledge, they are the first to consider general constraints. In distinction to this prior work, we do not focus on the difficulty of each labeling task. We consider a wider range of tasks. Culotta and McCallum present only one instantiation of our more general formulation. Luo et al. (2013) use uncertainty sampling, where probabilities

are computed using classifiers that account for constraints. Roth and Small (2008) score instances using the margin of learned classifiers and in the case of binary classification, margin-based active learning is equivalent to uncertainty sampling. Bilgic et al. (2010) consider dependencies among input instances and their labels. They cluster instances and look at disagreements of two kinds of classifiers over the clusters. Other methods that use “side” information in active learning include those of Kapoor and Baker (2009), Wallace et al. (2010), and Angeli et al. (2014).

Our method considers the important and common case where there are logical constraints over the output space, such as mutual exclusion and subsumption. The ubiquitous nature of such logic relationships creates a need for them to be addressed “head on.” The previously mentioned related work only deals with other kinds of probabilistic constraints. Harpale (2012) and Zhang (2010) have considered this setting, but they fail to provide theoretical justifications or deep experimental support. Furthermore, both approaches can be seen as separate instantiations of our more general framework, for which we also provide a formal analysis along with an extensive experimental evaluation.

B.3 PROPOSED METHODS

We now provide a description of our methods for performing active learning. The methods select examples to be labeled before each re-training step (i.e., the red box in Figure B.1). Let us consider a setting where we have a set of binary labels $Y_k^i \in \{0, 1\}$, for $k = 1, \dots, K$ and $i = 1, \dots, N$, for a provided set of instances $X^1, \dots, X^N$. Y_k^i denotes whether instance X^i belongs to class k . For example, X^i could represent a particular noun phrase (NP) and Y_k^i a label for that NP, indicating whether it is a city or not. There exists a set of logical constraints among the K labels which determine whether an assignment of values to these labels, for each instance, is valid or not. Let the marginal probability of each label being positive be defined as:

$$p_k^i \triangleq P_{X^i \sim \mathcal{D}}(Y_k^i = 1), \quad (\text{B.1})$$

for $k = 1, \dots, K$ and $i = 1, \dots, N$, where $\mathcal{D}$ is the distribution of the instances $X^1, \dots, X^N$, and $P(\cdot)$ denotes the probability of some event. Given a set of observed labels (which could be empty) and these marginal probabilities, we want to determine which label² to request in the active learning process in order to gain the most information. Thus, we use a *scoring function* to score each unobserved label based on how much information is gained by observing it, and we then pick the label with the highest score. We note that *information gain* can be defined in many ways depending on the task at hand and the evaluation metric that is being used. Our approach is initially motivated by the loose and possibly naïve definition of information gain as the expected number of labels one obtains after asking for a single label.

² Note that the word “label” here refers to a particular label-instance pair (i.e., we ask for a single label of a single instance at a time). This is the convention we use throughout this chapter.

We note that the common strategy of *uncertainty guided sampling* for allocating labeling effort uses the entropy of a label as its scoring function. That is:

$$\mathcal{S}_{\text{entropy}}(Y_k^i) \triangleq -p_k^i \log p_k^i - (1 - p_k^i) \log(1 - p_k^i). \quad (\text{B.2})$$

This function can be thought of as scoring each label based only on its own uncertainty ignoring any dependencies among the labels. The proposed methods make use of logical constraints among the labels, thus considering key dependencies. We therefore expect them to perform better in practice.

B.3.1 A Simple Constraint: Mutual Exclusion

Let us first consider a simple, yet powerful and common logical constraint among labels: *mutual exclusion*. We consider a setting where, for each value of i (i.e., instance), all labels (i.e., $Y_1^i, \dots, Y_K^i$) are mutually exclusive with each other. This means that, for each instance, at most one label can be positive. It is easy to see that, if we discover that a label for a specific instance is positive, then all other labels must be negative for that instance. However, if the answer is negative, then we cannot infer the value of any other label. Thus, intuitively we see that it might make sense to ask for the label with the highest marginal probability of being equal to 1 (i.e., the Y_k^i with the highest probability p_k^i). We now discuss this approach and provide theoretical justification for this intuition. We start by suggesting the following scoring function:

$$\mathcal{S}_{\text{probability}}(Y_k^i) \triangleq p_k^i. \quad (\text{B.3})$$

For our theoretical justification, we shall ignore the instance superscript, i , and consider the case where we only have a single instance X . We shall propose a theorem related to this scoring function, but we first state a lemma that will be used in the forthcoming proof.

Lemma B.1. *Let $x \in [0, 1]$, and $c \in [0, 1-x]$. Then, the following function is monotonic with respect to x : $f(x) = (1 - x - c) \log(1 - x - c) - (1 - x) \log(1 - x)$.*

Proof. $\partial f(x)/\partial x = \log(1 - x) - \log(1 - x - c)$ and since the logarithm is a monotonic function, we know that $\partial f(x)/\partial x \geq 0$. Thus, $f(x)$ is monotonic. $\square$

We now propose a theorem which demonstrates that the scoring function of [Equation B.3](#) is indeed the optimal choice when dealing with a set of mutually exclusive labels and when using the definition of information-theoretic information gain.

Theorem B.1. *Given a set of mutually exclusive labels, the scoring function of [Equation B.3](#) induces the same ranking of labels as that induced by the information-theoretic information gain.*

Proof. Due to the mutual exclusion constraint, we have that:

$$P_{X \sim \mathcal{D}}(\{Y_k = 0 \text{ for } k = 1, \dots, K\}) = 1 - \sum_{k=1}^K p_k. \quad (\text{B.4})$$

For notational convenience, let us henceforth denote this quantity by p_0 and also omit the $X \sim \mathcal{D}$ subscript from the probability operator notation. Now, note that:

$$P(\mathbf{y}_{-k}) = P(\mathbf{y}_{-k} \cup Y_k = 1) + P(\mathbf{y}_{-k} \cup Y_k = 0), \quad (\text{B.5})$$

$$P(\mathbf{y}_{-k}) = \begin{cases} 0, & \text{if } \mathbf{y}_{-k} \text{ has more than one 1s,} \\ p_l, & \text{if } y_l = 1 \text{ for } l \neq k, \\ p_k + p_0, & \text{otherwise,} \end{cases} \quad (\text{B.6})$$

where $\mathbf{y}_{-k}$ refers to an assignment of values to all labels Y_l , where $l = 1, \dots, K$, and $l \neq k$, and y_l refers to an assignment of Y_l . Let us also denote the information-theoretic information gain of variable Y_k by $\mathcal{I}(Y_k)$. We then have that, if $p_k \geq p_l$ for some $k \neq l$, then:

$$\mathcal{I}(Y_k) - \mathcal{I}(Y_l) = \mathcal{H}(\mathbf{Y}_{-k}) - \mathcal{H}(\mathbf{Y}_{-k} | Y_k) - \mathcal{H}(\mathbf{Y}_{-l}) + \mathcal{H}(\mathbf{Y}_{-l} | Y_l), \quad (\text{B.7})$$

$$\mathcal{I}(Y_k) - \mathcal{I}(Y_l) = \mathcal{H}(\mathbf{Y}_{-k}) + \mathcal{H}(Y_k) - \mathcal{H}(\mathbf{Y}_{-l}) - \mathcal{H}(Y_l), \quad (\text{B.8})$$

where $\mathcal{H}(Y_k)$ corresponds to the entropy of the Y_k variable, $\mathcal{H}(\mathbf{Y}_{-k})$ corresponds to the entropy of all variables Y_l , where $l = 1, \dots, K$ and $l \neq k$, and $H(p) \triangleq -p \log p - (1-p) \log(1-p)$. From [Equation B.6](#), we have that:

$$\mathcal{H}(\mathbf{Y}_{-k}) = - \sum_{\mathbf{y}_{-k}} P(\mathbf{y}_{-k}) \log P(\mathbf{y}_{-k}), \quad (\text{B.9})$$

$$\mathcal{H}(\mathbf{Y}_{-k}) = - \sum_{l \neq k} p_l \log p_l - (p_k + p_0) \log(p_k + p_0), \quad (\text{B.10})$$

where the first sum is over all possible assignments of the corresponding variables. Given that $k \neq l$ in [Equation B.8](#), we have that:

$$\mathcal{H}(\mathbf{Y}_{-k}) - \mathcal{H}(\mathbf{Y}_{-l}) = p_k \log p_k - p_l \log p_l, \quad (\text{B.11})$$

$$+ (p_l + p_0) \log(p_l + p_0), \quad (\text{B.12})$$

$$- (p_k + p_0) \log(p_k + p_0). \quad (\text{B.13})$$

Thus, it follows that:

$$\mathcal{I}(Y_k) - \mathcal{I}(Y_l) = (p_l + p_0) \log(p_l + p_0) - (1 - p_k) \log(1 - p_k), \quad (\text{B.14})$$

$$- (p_k + p_0) \log(p_k + p_0) + (1 - p_l) \log(1 - p_l), \quad (\text{B.15})$$

$$\mathcal{I}(Y_k) - \mathcal{I}(Y_l) = (1 - p_k - c) \log(1 - p_k - c) - (1 - p_k) \log(1 - p_k), \quad (\text{B.16})$$

$$- (1 - p_l - c) \log(1 - p_l - c) + (1 - p_l) \log(1 - p_l), \quad (\text{B.17})$$

and due to [Lemma B.1](#) with:

$$c = \sum_{\substack{m=1 \\ m \neq k, l}}^K p_m, \quad (\text{B.18})$$

we have that $\mathcal{J}(Y_k) - \mathcal{J}(Y_l) \geq 0$. This inequality implies that the ranking of labels induced by the information gain $\mathcal{J}(Y_k)$ is the same as the ranking induced by using the scoring function $\mathcal{S}_{\text{probability}}(Y_k^i)$. The proof is thus complete. $\square$

One of the most interesting consequences of [Theorem B.1](#) is that we now have a very efficient way to rank labels based on their information gain. Also note that, more often than not, classification systems are evaluated based on the area under the precision-recall curve (AUC). Intuitively, the AUC increases with the number of “gold” labels (i.e., labels that are guaranteed to be correct). We highlight the fact that the probability scoring function of [Equation B.3](#) is motivated by picking the label that is most likely to provide the greatest number of “gold” labels (i.e., labels that are fixed to 0).

We want to emphasize the relationship between using [Equation B.3](#) as the scoring function, as proposed in [Theorem B.1](#), and using entropy (i.e., [Equation B.2](#)) as the scoring function, as is done in uncertainty guided sampling. The following proposition and corollary of [Theorem B.1](#) describe this relationship more precisely.

Proposition B.1. *When:*

$$\arg \max_{\substack{k=1,\dots,K, \\ i=1,\dots,N}} p_k^i = \arg \min_{\substack{k=1,\dots,K, \\ i=1,\dots,N}} |p_k^i - 0.5|, \quad (\text{B.19})$$

the probability scoring function of [Equation B.3](#) is equivalent to the entropy scoring function of [Equation B.2](#), which is used in uncertainty guided sampling.

Proof. [Equation B.19](#) follows directly by noticing that, for $p_k^i \in [0, 1]$:

$$\arg \max_{\substack{k=1,\dots,K, \\ i=1,\dots,N}} \mathcal{S}_{\text{entropy}}(Y_k^i) \equiv \arg \min_{\substack{k=1,\dots,K, \\ i=1,\dots,N}} |p_k^i - 0.5|. \quad (\text{B.20}) \quad \square$$

Corollary B.1. *In the case of a single instance X the probability scoring function of [Equation B.3](#) and the entropy scoring function of [Equation B.2](#) are equivalent.*

Proof. The mutual exclusion constraint, $\sum_{k=1}^K p_k \leq 1$, implies that the condition of [Proposition B.1](#) is always satisfied. The corollary follows. $\square$

It is easy to observe that when we have several instances $X^1, \dots, X^N$ and we compare the scores of each label-instance pair, then the two scoring functions are no longer necessarily equivalent. Also note that, as the number of labels grows, the marginals are more likely to have smaller magnitudes and thus the condition of [Proposition B.1](#) is more likely to be satisfied.

INTERESTING FACT

The scoring function $\mathcal{S}_{\text{probability}}(Y_k^i)$ assigns a higher score to labels that are more certainly positive than to more uncertain labels. This is in contrast to uncertainty guided sampling and thus highlights the importance of [Theorem B.1](#). It also demonstrates that positive examples can, in some cases, be much more useful and informative than negative examples. Sharma and Bilgic (2013) discuss the sources of uncertainty and reinforce our argument about the ineffectiveness of uncertainty sampling for some kinds of logical constraints.

A DIFFERENT APPROACH. We now introduce a new concept that, when combined with the earlier motivation, gives rise to a new scoring function. The key intuition lies in the scenario where the discovery of a label being positive implies that all other labels are negative. This discovery may not be as valuable if the negative labels are already inferred to have low marginal probability. Instead, we propose to consider the *degree of surprise* of discovering that these labels are negative. For a label with marginal probability of being positive p_k , the amount of surprise can be defined in several ways. A function $\mathcal{S} : [0, 1] \mapsto \mathbb{R}$ is called a *surprise function* if it is decreasing and $\mathcal{S}(1) = 0$. A couple examples of such *surprise functions* are shown here:

- Logarithmic: $\mathcal{S}_{\log}(p_k) \triangleq -\log p_k$. This is equivalent to the *self-information* of event $Y_k = 1$ and was first referred to as a surprise measure by Tribus (1961).
- Linear: $\mathcal{S}_{\text{linear}}(p_k) \triangleq 1 - p_k$. This is related to the 0-1 loss which was originally proposed by Roy and McCallum (2001).

Using this definition of a surprise function we define a new scoring function for the mutual exclusion case as follows:

$$S_{\text{ME}}(Y_k) \triangleq p_k \underbrace{\sum_{c=1}^K \mathcal{S}(p_c) \mathbb{1}_{\{c=k\}} \mathcal{S}(1 - p_c) \mathbb{1}_{\{c \neq k\}}}_{\text{Surprise of setting } Y_k=1} + (1 - p_k) \underbrace{\mathcal{S}(1 - p_k)}_{\text{Surprise of setting } Y_k=0}, \quad (\text{B.21})$$

where $\mathcal{S}(\cdot)$ is an arbitrary surprise function and $\mathbb{1}_{\{\cdot\}}$ evaluates to one if its subscript statement is true and to zero otherwise. Note that the first term is the product of the probability of Y_k being equal to 1 and the sum of surprise “experienced” by fixing the value of Y_k to 1 (i.e., after propagating the mutual exclusion constraint, we sum over the surprises of all other labels being set to 0 and Y_k being set to 1). The second term is similarly defined as the product of the probability of Y_k being equal to 0 and the surprise of fixing Y_k to that value. No other variables are considered in this surprise value as no other label value is fixed, because of the mutual exclusion constraint. Note that this is substantially different than the entropy scoring function in that it is measuring “surprise” rather than uncertainty.

B.3.2 More General Logical Constraints

The scoring function of Equation B.21 and the underlying intuition can be easily extended to more general logical constraints than mutual exclusion. An example of a more general logical constraint is *subsumption*. In this case, each label can have a set of parent and child labels, and a label being set to 1 implies that its parent label is also set to 1. To extend the method introduced in the previous section, we need a function for propagating a fixed label-value pair through the constraints. Let this function be defined as $\mathcal{F}(Y_k = v) \triangleq \{(Y_{c_i}, v_i) : \text{if } Y_k = v, \text{ then } Y_{c_i} = v_i\}$, where $c_i \in \{1, \dots, K\}$ is a label index, and $v_i \in \{0, 1\}$ is the value of Y_{c_i} fixed by propagating the fixed label-value pair (Y_k, v) through the constraints. We can now define our scoring function for general logical constraints as follows:

$$\mathcal{S}_{\text{constraints}}(Y_k) \triangleq p_k \underbrace{\sum_{(Y_{c_i}, v_i) \in \mathcal{F}(Y_k=1)} S(Y_{c_i}, v_i)}_{\text{Surprise of setting } Y_k=1} + (1-p_k) \underbrace{\sum_{(Y_{c_i}, v_i) \in \mathcal{F}(Y_k=0)} S(Y_{c_i}, v_i)}_{\text{Surprise of setting } Y_k=0}, \quad (\text{B.22})$$

where $S(Y_{c_i}, v_i) = \mathcal{S}(p_{c_i})^{\mathbb{1}_{\{v_i=1\}}} \mathcal{S}(1-p_{c_i})^{\mathbb{1}_{\{v_i=0\}}}$.

FORMAL JUSTIFICATION. We have not derived a result for the general scoring function similar to that of [Theorem B.1](#). However, we can use the information-theoretic information gain to generate an interesting result, akin to our justification for using the scoring function of [Equation B.3](#) in the setting of mutual exclusion. We note that the information gain for the case with general logical constraints can be defined as a sum. The first term of this sum is the entropy of the label whose information gain is being computed. When the logarithmic surprise function is used with the scoring function of [Equation B.22](#), then our scoring function contains this entropy term, as well as an approximation of some other terms (but not all) of the complete information gain sum. More specifically, we have that (note that, in this derivation we ignore terms that are constant across all label variables because they do not affect the ranking of the labels induced by the information gain):

$$\begin{aligned} \mathcal{I}(Y_k) &= \mathcal{H}(Y_{-k}) - \mathcal{H}(Y_{-k} \mid \mathcal{H}_k), \\ &= \mathcal{H}(Y_{-k}) + \mathcal{H}(Y_k) - \mathcal{H}(Y), \\ &= \mathcal{H}(Y_k) - \mathcal{H}(Y_k \mid Y_{-k}), \\ &= \mathcal{H}(Y_k) + \sum_{y_k} \left[\sum_{y_{-k}} P(y_{-k}) P(y_k \mid y_{-k}) \log P(y_k \mid y_{-k}) \right], \\ &= \mathcal{H}(Y_k) + \sum_{y_k} \left[\sum_{y_{-k}} \underbrace{[P(y) \log P(y) - P(y_k, y_{-k}) \log P(y_{-k})]}_{\text{Constant}} \right], \\ &= \mathcal{H}(Y_k) - \sum_{y_k} P(y_k) \sum_{y_{-k}} P(y_{-k} \mid y_k) \log P(y_{-k}), \\ &= \mathcal{H}(Y_k) - \sum_{y_k} P(y_k) \sum_{\substack{y_{-f}, \\ y_f = \mathcal{F}(y_k)}} \underbrace{P(y_{-f}, y_{f \setminus k} \mid y_k)}_{P(y_{-f} \mid y_f)} \log P(y_{-f}, y_{f \setminus k}), \\ &= \mathcal{H}(Y_k) - \sum_{y_k} P(y_k) \sum_{\substack{y_{-f}, \\ y_f = \mathcal{F}(y_k)}} \left[\underbrace{P(y_{-f} \mid y_f)}_{\text{Sums to 1}} \log P(y_f) + \right. \\ &\quad \left. P(y_{-f} \mid y_f) \log P(y_{-f} \mid y_{f \setminus k}) \right], \\ &= \underbrace{\mathcal{H}(Y_k)}_{\text{Entropy}} - \sum_{y_k} P(y_k) \left[\underbrace{\log P(y_f)}_{\text{Constraints}} + \sum_{\substack{y_{-f}, \\ y_f = \mathcal{F}(y_k)}} \underbrace{P(y_{-f} \mid y_f) \log P(y_{-f} \mid y_{f \setminus k})}_{\text{Remainder}} \right], \end{aligned}$$

where $f \setminus k$ is the set of label indices in f excluding k . Note that the entropy scoring function of Equation B.2 only considers the term denoted by “Entropy” in this sum. When the logarithmic surprise function is used, the general scoring function of Equation B.22 contains an approximation to the terms denoted by “Constraints,” where the joint is written as the product of the marginals. This result shows that the general scoring function is, in some sense, a better heuristic than the entropy scoring function.

B.3.3 Computational Complexity

We now consider the real-world use of an active learning system, where a request is made for a label of a particular instance. Note that if we were to use the information-theoretic information gain as our scoring function, then the cost of computing it would be linear in N and exponential in K . Our proposed scoring functions reduce this cost. The entropy scoring function of Equation B.2 has a computational cost linear in the number of labels and the number of instances (i.e., because we need to compute it for all labels); its cost is $O(NK)$. The probability scoring function of Equation B.3 has the same cost. The mutual exclusion scoring function has cost $O(NK^2)$. Finally, ignoring the cost of the constraint propagation function, the general scoring function of Equation B.22 has a computational cost of $O(NK^2)$, because the highest number of labels that can be fixed is K . Note that the constraint propagation function can have a cost exponential in K in the worst case. However, there are special cases where the cost is not as high. For example, with either the mutual exclusion or the subsumption constraint the cost is linear in K . When mutual exclusion is combined with subsumption, we can alternate between all our constraints, one by one, and keep propagating them, until no fixed label can be further propagated. If the number of constraints is C , the cost of this propagation operation is $O(CK)$. This is the most complex scenario that we consider in our experiments and it covers most of the practical use cases in multi-task applications that motivated this work.

B.4 EXPERIMENTS

In the following paragraphs, we describe the setup of our experiments, including the datasets and the evaluation metrics that we use, and our results along with corresponding analyses. All the datasets and code for the experiments are available at <https://github.com/eaplatanios/makina>. We first define the names that we use to refer to different methods when plotting the results:

- RANDOM: Uses a random scoring function (i.e., it generates a random number between 0 and 1 each time it is invoked).
- Entropy: Uses the entropy scoring function of Equation B.2.
- RANDOM-CP: RANDOM which also propagates labels through the constraints.
- ENTROPY-CP: ENTROPY which also propagates labels through the constraints.
- PROBABILITY-CP: Uses the probability scoring function of Equation B.3 and also propagates labels through the constraints.

Dataset	#Classes	#Features	Balanced	#Training	#Testing	#New/Iteration
SatImage	6	36	×	3,104	1,331	100
Shuttle	7	9	×	30,450	13,050	1,000
Segment	7	19	✓	400	1,910	100
PenDigits	10	16	✓	7,494	3,498	100
Letter	26	16	✓	15,000	5,000	1,000
NELL-7	7	180,878	×	214	14,693	500
NELL-11	11	180,878	×	242	14,693	1,000
NELL-15	15	180,878	×	2,656	18,016	2,000

Table B.1: Datasets used in our active learning experiments.

NELL-7

bird ---- fish ---- mammal ---- city ---- country ---- river ---- lake

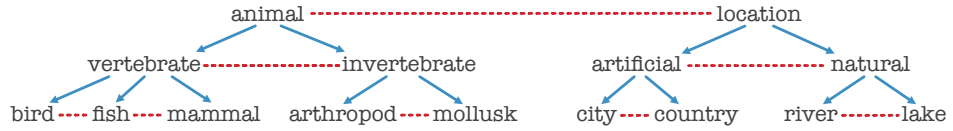
NELL-11**NELL-15**

Figure B.2: Illustration of the categories used in the NELL datasets and the constraints between them. Each blue arrow represents a subsumption constraint, and each set of labels connected by a red dashed line represents a mutually exclusive set of labels. For example, `animal` subsumes `vertebrate` and `bird`, `fish`, and `mammal` are mutually exclusive.

- LOG-CP: Uses the constraints scoring function of Equation B.22 with the logarithm surprise function and also propagates labels through the constraints.
- LINEAR-CP: Same as LOG-CP, but uses $\mathcal{S}_{\text{linear}}$ instead of the $\mathcal{S}_{\text{log}}$.

We apply the same experimental setup to all datasets. Each dataset consists of a set of positive examples for each label. For each experiment, we split the dataset into train and test subsets. For each label, we train a binary logistic regression classifier using the AdaGrad optimization algorithm of Duchi et al. (2011), with a batch size of 100 samples. Our experimental pipeline consists of the following steps:

1. We initially train a classifier for each label independently using the train portion of the dataset. We consider all positive examples for the corresponding label, along with a set of negative examples of the same size, sampled from the remaining set of examples in the train dataset that are not labeled as positive for this label.
2. We repeat the following steps until all test examples have been labeled:

- i. Request a set of M examples from the test dataset to be manually labeled,³ sequentially. For all the methods that include CP in their name, after each example is obtained, we propagate all logical constraints. The examples fixed by this process are considered manually labeled. Note that, since the datasets differ in size, M can vary across each dataset. Please refer to [Table B.1](#) for the values of M used for each dataset (labeled as “#New/Iteration”). Each method’s scoring function determines which examples are selected for labeling. The label-instance pair with the highest score is selected for labeling.
- ii. Move all the labeled examples from the test to the train dataset.
- iii. Re-train the classifiers for all labels using the updated train dataset. Training for the classifiers is initialized at the previously learned classifiers to reduce convergence time.
- iv. Evaluate progress using a set of metrics over the full dataset (i.e., the train and test parts of the dataset, combined). Note that even though it may seem unorthodox to evaluate on the full dataset, it is actually meaningful for settings like NELL. In fact, that is how NELL is evaluated as we care about the accuracy of its whole knowledge-base, irrespective of how the label of each instance was obtained.

A NOTE ON MARGINAL PROBABILITIES. Note that all the methods and results presented in [Section B.3](#) rely on marginal probabilities. In our experiments, we use classifiers to estimate these marginals and sometimes they may not be very accurate. This is the main reason we subsample a number of negative examples equal to the number of positive examples. Otherwise, our logistic regression classifiers would be biased towards low estimates of the probabilities, which would cause the entropy and our proposed scoring functions to perform very similarly, as shown in [Section B.3.1](#). This was indeed the case when we ran experiments without subsampling the negative examples, to test this hypothesis. This problem can also be alleviated by using better calibrated classification models.

B.4.1 Datasets

We now provide the list of datasets used in our experiments:

- SatImage: Classify a satellite image region (Feng et al., 1993).
- Shuttle: Classify a space shuttle in one of seven classes (Feng et al., 1993).
- Segment: Classify a small outdoor image region (Feng et al., 1993).
- PenDigits: Classify a handwritten digit (Alimoglu and Alpaydin, 1996).
- Letter: Classify an image as a letter of the English alphabet (Frey and Slate, 1991).
- NELL-7: Classify noun phrases as belonging to a certain category or not. The categories considered in this dataset are Bird, Fish, Mammal, City, Country, Lake, and River (i.e., the category represents the label in this case). The only constraint

³ Note that by “example” we mean a label-instance pair and so, all possible label instance pairs from the test dataset are considered at this stage.

considered in this case is that all these categories are mutually exclusive. We use the same set of features as that used by the coupled pattern learner (CPL) in NELL.

- NELL-11: Perform the same task as NELL-7, but additionally consider the categories `Animal`, `Location`, `Artificial Location`, and `Natural Location`. Also include the subsumption constraints shown in Figure B.2, while ignoring the categories not included in this dataset.
- NELL-15: Perform the same task as NELL-7, but with the categories and constraints illustrated in Figure B.2.

Table B.1 provides details on the statistics and experimental setup for each dataset. The NELL datasets were obtained from <https://rtw.ml.cmu.edu/rtw/resources> and the rest of the datasets were obtained from <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html>. Note that, for all datasets except for NELL-11 and NELL-15, the only constraint used is a mutual exclusion constraint between all labels.

B.4.2 Evaluation Metrics

Central to our evaluation is the *average area under the curve (average AUC)* metric. At each iteration and for each label, we compute the AUC over the whole dataset (i.e., the train dataset and test dataset combined). Then, we compute a weighted average of the AUCs for each label, where each label’s contribution is weighted by the number of positive examples for that label. This weighted average is what we refer to as the average AUC. It is easy to see that the average AUC is a non-decreasing function with respect to iteration number.⁴ We use the following three metrics to evaluate the proposed methods:

- Iterations until Average AUC=1: Number of iterations until the average AUC reaches value 0.999.
- Average AUC: Average AUC value across active learning iterations.
- Number of Fixed Labels: Number of labels that are effectively fixed (i.e., added to the train dataset), after each iteration. This measure is not always equal to the requested number of labels because of the constraint propagation step.

B.4.3 Results Analysis

Our results are shown in Figure B.3. We first note that the proposed methods *consistently outperform the other methods by a significant margin, for all datasets and all evaluation metrics*. For the datasets that only consider a single mutual exclusion constraint, PROBABILITY-CP always performs best with respect to the number of iterations until the average AUC reaches a value of 1. This is not unexpected; as we showed in Section B.3, this method can be considered optimal. Furthermore, an interesting observation is that for the average AUC plots, in all cases where we only have a single mutual exclusion constraint, despite seeing underperformance in early iterations,

⁴ This nice property is the reason we use the combined dataset as opposed to using just the test dataset.

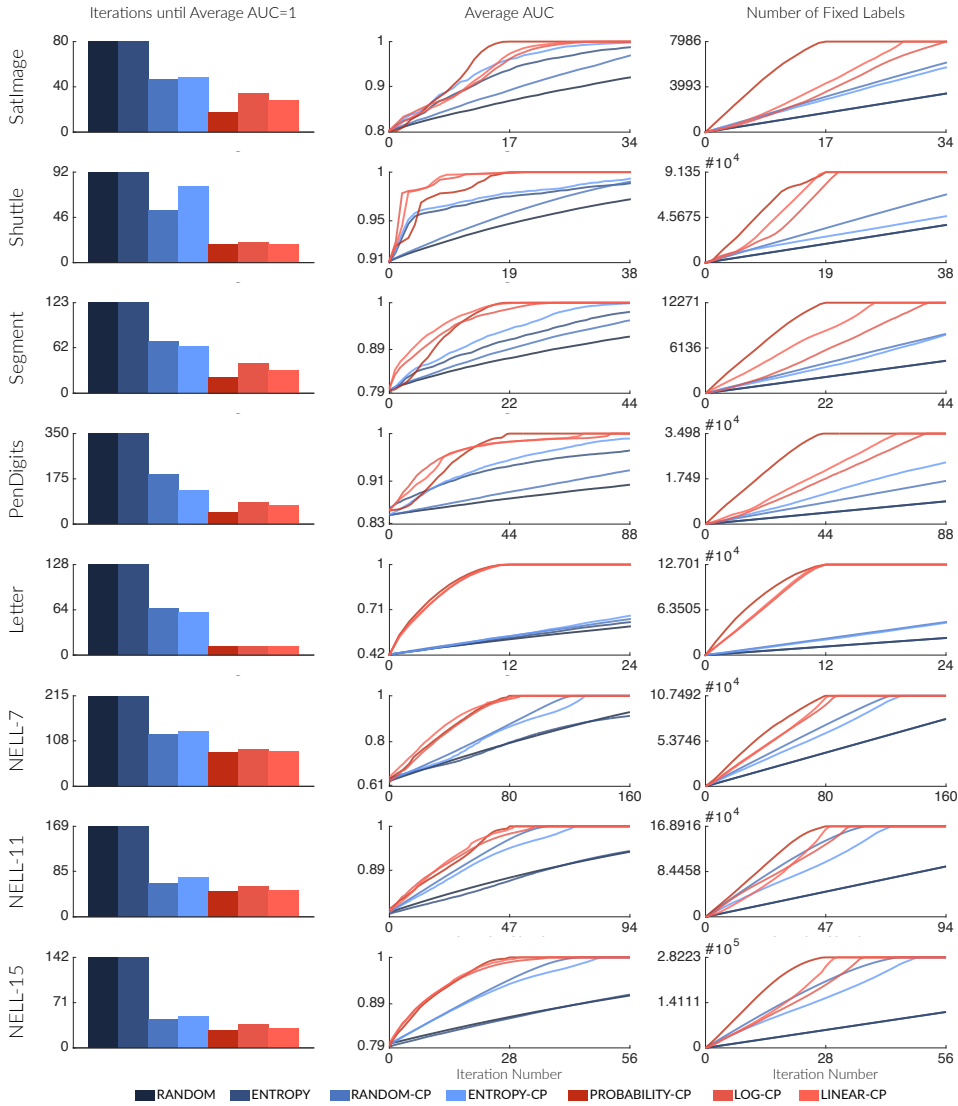


Figure B.3: Results. The red colored plots correspond to the proposed methods and the blue colored plots correspond to existing methods (apart from the constraint propagation step that we optionally added to all existing methods to enable a more fair comparison, and which is denoted by a -CP appended to the method name). For the first plot, the lower the bar, the better the result. For the rest of the plots, the higher the value of the curve per iteration, the better the result. We thus observe that the proposed methods outperform all existing methods for all of the experiments performed.

PROBABILITY-CP still reaches $AUC = 1$ faster. This may be based on the fact that this method first selects label-instance pairs with probability very close to 1, which ultimately turn out to be positive. However, after a few iterations, the method experiences a boost and outperforms all other methods. As for the number of fixed labels per iteration, PROBABILITY-CP also significantly outperforms the competing methods. This provides validation for our intuition discussed in Section B.3, that the method would “fix” more labels when the mutual exclusion constraint is propagated. As for

the two datasets where we also have subsumption constraints and multiple mutual exclusion constraints, we observe that the proposed methods consistently outperform all other methods, as expected. However, we did not expect for PROBABILITY-CP to be performing as well as LOG-CP and LINEAR-CP. We do not yet have an understanding about this finding, but we consider it an interesting and encouraging result for the proposed methods. We note that there are a few datasets with only a single mutual exclusion constraint, where PROBABILITY-CP is actually slower than the other two proposed methods, early on in the average AUC curve. Thus in some scenarios, there is some value in using these two methods. Finally, it is also interesting that the constraint propagation step alone provides a significant performance boost to all methods.

B.5 KEY TAKEAWAYS

We have proposed methods for performing active learning efficiently in the presence of logical constraints between the outputs of multiple classifiers. Our approaches resonate with underlying intuitions and challenge the core idea behind uncertainty guided sampling. Furthermore, we provided a theoretical justification for using the proposed methods. In a set of experiments, we found that the methods consistently outperformed competing approaches across eight diverse datasets and thus appear to be promising for practical applications. Moreover, our experiments showed that the proposed methods can be used to speed up the learning process in NELL. Per our knowledge, this is the first attempt to describe and carefully study methods for performing active learning when there are logical constraints among outputs of multiple classifiers. We are excited about numerous future directions for this work. Our first priority is to pursue additional theoretical results for the general setting with arbitrary logical constraints. We want to also explore methods for a setting where all labels for a particular data instance are requested at each iteration; this use case is useful to systems like NELL where the label space is extremely sparse. Furthermore, we want to explore ways in which we can use accuracy estimates for the trained classifiers—such as those proposed in [Part i](#)—in order to make the active learning procedures more robust. Implementing efficient computation of the value of information for multiple interdependent classifiers would be a step towards autonomous learning systems with the ability to reflect more deeply about their pursuit of information.

Cognitive architectures were first introduced by Newell (1990) who argued that the human mind functions as a single system, and proposed the notion of a *unified theory of cognition* (UTC). They often consist of constructs that reflect assumptions about human cognition and that are based on facts derived from psychology experiments (e.g., problem solving, decision making, routine action, memory, learning, skill, perception, motor behavior, language, motivation, emotion, imagination, and dreaming). In fact, Newell believed that cognitive architectures are the way to answer one of the ultimate scientific questions: “How can the human mind occur in the physical universe?”. Most existing work on UTCs has focused on symbolic approaches, such as the Soar architecture (Laird, 2012) and the ACT-R (Anderson et al., 2004) system. However, such approaches limit a system’s ability to perceive information of arbitrary modalities, require a significant amount of human input, and are restrictive in terms of the learning mechanisms they support (supervised learning, semi-supervised learning, reinforcement learning, etc.). For this reason, researchers in machine learning have shifted their focus towards methods like deep learning.

Deep learning systems have become the de facto standard for solving prediction problems in a multitude of application areas including computer vision, natural language processing, and robotics. Driven by progress in deep learning, the machine learning community is now able to tackle increasingly more complex problems—ranging from multi-modal reasoning (Hu et al., 2017) to dexterous robotic manipulation (OpenAI et al., 2020)—many of which typically involve solving combinations of tasks. However, many real-world problems require integrating multiple, distinct modalities of information (e.g., image, audio, language) in ways that machine learning models cannot currently handle well. Most of these approaches are also not able to utilize information learned from solving one problem to directly help in solving another—something at which human intelligence excels. There have been some limited attempts to train a single model that solves multiple problems jointly (e.g., Kaiser et al., 2017), but the resulting systems generally underperform those trained separately for each problem. Moreover, most of the existing approaches are also not capable of never-ending learning (NEL); namely a machine learning paradigm in which an algorithm learns from examples continuously over time, in a largely self-supervised fashion, and where its experience from past examples can be leveraged to learn future examples (Mitchell et al., 2018). Current machine learning systems fail when the problems that need to be learned are not fixed a priori, but are rather dynamic and keep changing as part of the environment where the learning agents operate. For example, humans do not just learn to solve a fixed set of problems, but they rather adapt and by solving one problem, they become better able to tackle new problems that they may even have been previously unaware of. For example, after humans managed to build heart monitoring devices, new unsolved problems became available, such as

discovering the relationship between heart rate or blood pressure and specific health problems. Furthermore, humans are capable of creating problems to learn, on their own, something that current machine learning systems are not designed to achieve. Never-ending learning is thus also something at which human intelligence excels. Due to the real world's dynamic nature, to achieve true intelligence, a learning agent that interacts with the real world needs to be able to adapt in such a continuous fashion. In fact, such an ability is crucial for never-ending learning, because learning *forever* only really makes sense if the learning objectives are ever-evolving.

The work presented in this thesis enables us to bridge the gap between UTCs, deep learning, and never-ending learning. To this end, we propose a *neural cognitive architecture* that allows for a tighter coupling between problems, as well as a higher-level of abstraction over distinct modalities of information. In this section, we only provide a proposed design for such an architecture that utilizes the methods and ideas that were presented in this thesis. This proposal is meant to describe our way of thinking about the design space for this problem as a whole. We provide no implementation or experimental evaluation as that is outside the scope of this thesis.

C.1 COGNITIVE ARCHITECTURES

Cognitive architectures can be broadly divided in *symbolic*, *subsymbolic*, and *hybrid* architectures. Symbolic systems rely on sets of rules and reason over discrete spaces (e.g., using first-order logic). Subsymbolic systems specify no such rules a priori and rely instead on emergent properties of several distinct processing units (e.g., neural networks). Hybrid approaches are a combination of symbolic and subsymbolic approaches. Symbolic processing makes reasoning interpretable, meaning that humans can see and understand how the system reasons about different problems, and it also often results in better generalization ability due to the constraints imposed by the symbolic language being used. However, it also limits the models' ability to perceive information of arbitrary modalities and further requires a significant amount of human input. Most past work on UTCs has focused on symbolic systems. In the following paragraphs, we describe two such successful systems.

SOAR. Laird (2012) designed Soar, a general cognitive architecture for developing systems that exhibit intelligent behavior, that has been in use since 1983. The design of Soar can be seen as an investigation of an approximation to *complete rationality*, which would imply the ability to use all available knowledge for every task that the system encounters. The primary principle at the base of Soar's design is that "all decisions are made through the combination of relevant knowledge at runtime. In Soar, every decision is based on the current interpretation of sensory data, the contents of working memory created by prior problem solving, and any relevant knowledge retrieved from long-term memory." Soar relies on multiple learning mechanisms (chunking, and reinforcement, episodic, and semantic learning), and on many representations of long-term knowledge (procedural knowledge productions, semantic memory, and episodic memory).

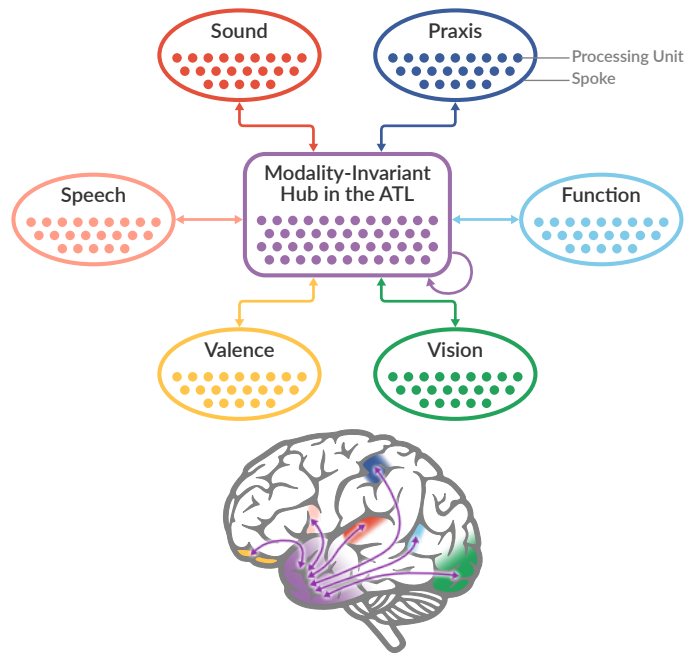


Figure C.1: The original Hub-and-Spoke model (Rogers et al., 2004).

ACT-R. Anderson et al. (2004) propose an alternative cognitive architecture, ACT-R, aimed at simulating and understanding human cognition. ACT-R consists of constructs that reflect assumptions about human cognition and that are based on facts derived from psychology experiments. An important feature of ACT-R that distinguishes it from other UTCs is that it directly allows researchers to compare the system’s performance to that of human participants.

In this section, we propose a novel cognitive architecture that also reflects assumptions about human cognition—inspired from the high-level design of the aforementioned systems—but that is subsymbolic and enables the use of neural networks and end-to-end training. It contains components that correspond to perception, action, reasoning, memory, world simulation, and learning.

C.2 THE HUB-AND-SPOKE THEORY

Rogers et al. (2004) proposed the *Hub-and-Spoke* theory of human cognition, which assimilates two important ideas: (i) multi-modal experiences provide the main “ingredients” for constructing concepts and they are encoded in modality-specific cortices, or *spokes*, that are distributed across the brain, and (ii) cross-modal interactions between the modality specific spokes are mediated by a single trans-modal *hub* that is located bilaterally in the anterior temporal lobes (ATLs) of the human brain. A visualization is shown in Figure C.1. This model of the human brain serves as one of the main inspirations for the high-level design of the proposed architecture.

C.3 PROPOSED ARCHITECTURE

We propose a novel *neural cognitive architecture* (NCA) for general learning and intelligence. The proposed architecture is inspired from the *Hub-and-Spoke* model for human cognition (Rogers et al., 2004; Ralph et al., 2017), as well as human *goal priming* (Custers and Aarts, 2005; Aarts et al., 2008; Papies, 2016; Takarada and Nozaki, 2018). It consists of the following parts:

- Perception and Action Spokes: Sensing input data consists of converting them to a common reasoning space, that is independent of the data modality. Much of the complexity of models like BERT¹ (Devlin et al., 2019), lies in perception, rather than reasoning. In fact, for BERT, reasoning often consists of a single linear layer, while perception consists of a Transformer (Vaswani et al., 2017). Similarly, taking an action consists of converting a common reasoning representation to some output data. This can include taking actions in some environment, or generating data of some structure (e.g., probabilistic distribution over labels).
- Reasoning Hub: Reasoning is performed in a latent space that is independent of the data modalities and the problem being solved. We argue that this is necessary for general learning and intelligence, as it allows for flexible sharing of information across different modalities and problems. Moreover, memory and simulations of the external world are all defined over the same latent space, abstracting away details about the perceived data that are not relevant to reasoning. Reasoning is described in detail in [Section C.5](#).
- Goal Contextualization: The problems that the system is learning to solve are processed such that they can contextualize any part of the neural cognitive architecture. This allows for the behavior of the system to vary across different problems, while still sharing information between them. The contextualization mechanism we propose to use was described in [Chapter 7](#) and was evaluated on multiple case studies in [Chapter 8](#). It further allows the system to generate its own target problems that it learns to solve. This is perhaps the most novel aspect of the proposed architecture and, as shown in the following paragraphs, derives its inspiration from human *goal priming* in psychology, and is described in more detail in [Section C.6](#).

An overview of the architecture is shown in [Figure C.2](#). This neural cognitive architecture is inspired from our work in this thesis, as well as work in multiple other areas:

DEEP LEARNING. Deep neural networks are very effective at learning abstract representations for arbitrary data modalities, that can then be used to perform multiple diverse tasks (e.g., Simonyan and Zisserman, 2015; He et al., 2016b; Peters et al., 2018; Devlin et al., 2019). The typical deep learning workflow is that for each problem researchers build large deep neural models that pool together information from different

¹ BERT is the current state-of-the-art model for a multitude of natural language processing tasks.

Note: Sensors and effectors can be defined in a compositional manner, and thus may not consist of a single modality network (e.g., when combining image and text inputs).

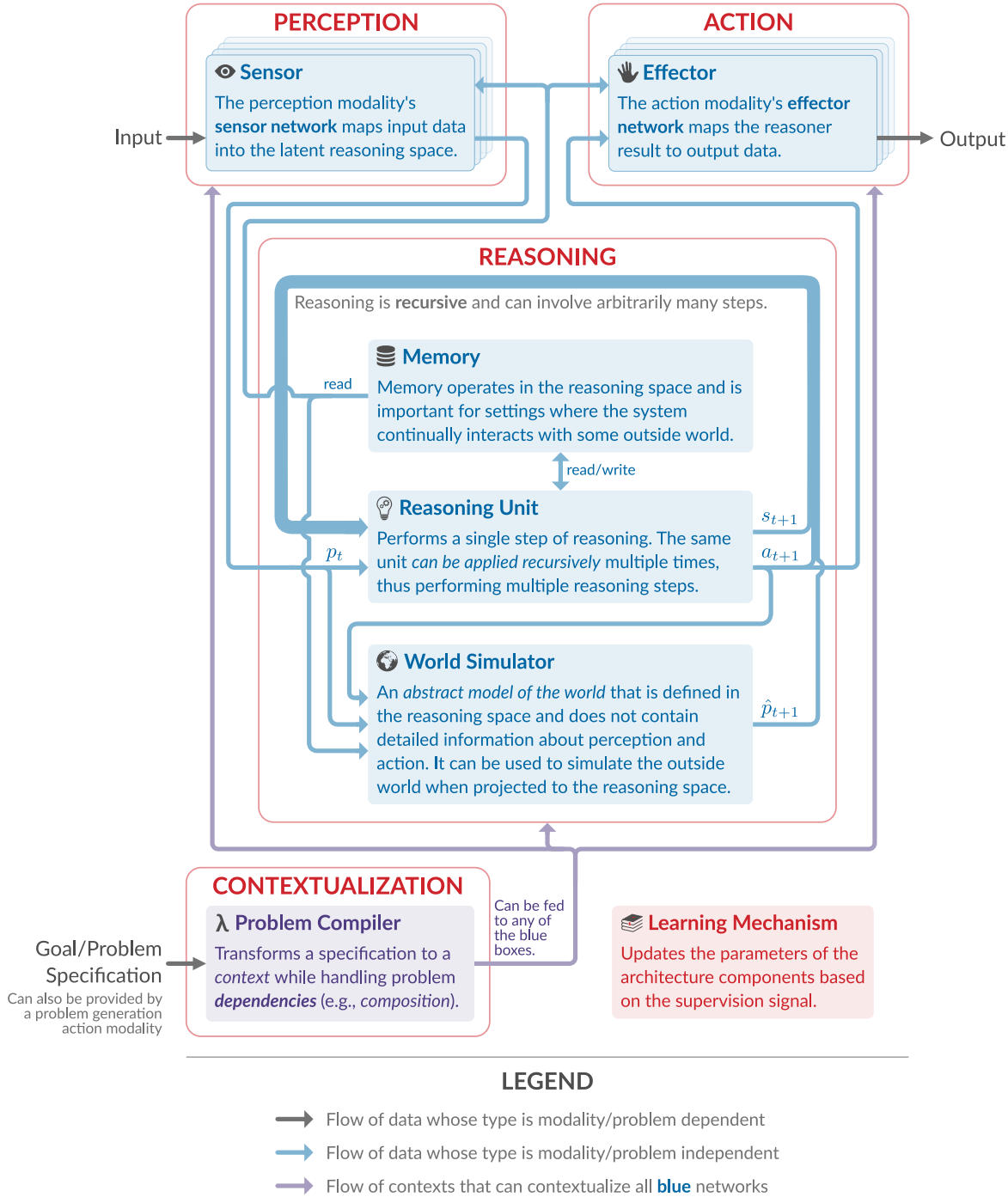


Figure C.2: Overview of the proposed neural cognitive architecture.

sources and that are trained independently of each other. An alternative approach is to pre-train large models in a problem-independent manner and then fine-tune them for each problem (e.g., Finn et al., 2017; Peters et al., 2018; Devlin et al., 2019). However, most of these approaches do not allow for information learned from solving one problem to directly help solve another—something at which human intelligence excels. For example, BERT (Devlin et al., 2019) is pre-trained as a language model and is then fine-tuned separately for problems such as question answering and textual entailment. Therefore, learning to answer questions well does not affect how well BERT reasons about textual entailment. This motivates us to find ways to couple the learning of multiple problems in a way that results in *constructive interference* between the different problems, meaning that learning to solve one well, helps the system learn to solve others. It further motivates us to treat *perception* (i.e., learning informative representations of the input data) and *reasoning* (i.e., learning to solve each task in the latent space of learned representations) separately, as most deep neural networks that are trained end-to-end to solve multiple tasks effectively do that, and most of their complexity is often related to perception (e.g., in BERT problem-specific reasoning is performed by a linear layer).

KERNEL METHODS. Before deep learning was popular, some of the most successful machine learning methods were making use of *kernels* (Hofmann et al., 2008), by formulating learning problems in a reproducing kernel Hilbert space (RKHS) of functions defined on the data domain, expanded in terms of a kernel. These kernels effectively project the data to a space where reasoning is modeled as a linear transformation. Such a projection can be thought of as a perception module, in terms of our formulation. Given the success of kernel methods, this further motivates separating the treatment of perception and reasoning.

NEUROSCIENCE. Neuroscientists have also observed that information processing in the human brain goes from low-level (i.e., sensory input processing) to high-level (i.e., reasoning). There is ample evidence to support this both for both auditory (Kaas and Hackett, 1998; Rauschecker, 1998; Romanski et al., 1999; Wessinger et al., 2001; Zatorre and Belin, 2001; Warren and Griffiths, 2003; Zatorre et al., 2004) and visual information (Mishkin et al., 1983; Felleman and Van, 1991). Furthermore, there has been evidence that the development of primary visual cortical networks is more rapid than the development of primary motor networks in humans (Gervan et al., 2011). This motivates the idea that perception is a low-level functionality that is not necessarily problem-specific and that can be learned before learning to reason and take actions. In addition to this, there is evidence that the brain relies on a set of canonical neural computations that are reused for different problems (Carandini and Heeger, 2012). For example, normalization of neural responses is one such operation that is thought to underlie multiple other operations such as the representation of odours, the modulatory effects of visual attention, the encoding of value, and the integration of multi-sensory information. This also supports the idea of abstracting over reasoning, by making the operations used to perform various tasks common across all tasks and finding other ways to specialize them.

PSYCHOLOGY. There has been significant evidence that *priming* is characteristic of human behavior (Tulving et al., 1982; Bargh and Chartrand, 2014; Weingarten et al., 2016). Priming is a technique where exposure to one stimulus influences the brain’s response to a subsequent stimulus. For example, the word “dog” is recognized more quickly after having seen the word “animal.” Priming can be perceptual, semantic, and conceptual. Perhaps most importantly for this thesis is *goal priming* (Custers and Aarts, 2005; Aarts et al., 2008; Papies, 2016; Takarada and Nozaki, 2018). Goal primes are cues that trigger goal-directed cognition and behaviour. Here, a goal refers to a state or behaviour that has reward value and therefore motivates a person to pursue it. For example, priming the concept of drinking can increase soda consumption (Veltkamp et al., 2008), or priming the goal of impression formation leads to better memory organization and recall compared to a mere memorization goal (Chartrand and Bargh, 1996). Goal contextualization in our architecture is the computational equivalent of goal priming, in that having specific goals changes the way in which the different architecture parts function.

BENEFITS OF MODULARITY. An important outcome of the Hub-and-Spoke architecture design is reducing the per-problem sample complexity. This means reducing the amount of training data required to learn to solve each problem. This is because, for multiple existing machine learning models, most of the model complexity lies in perception (e.g., BERT). This becomes more prevalent in reinforcement learning systems playing video games where they receive as input the raw pixel values of video frames as they are being rendered while playing, and they are tasked with learning to extract information from these raw values (Bellemare:2013:arcade; e.g., Bhonker et al., 2016; Vinyals et al., 2018). Such systems require massive amounts of training data to learn and we argue that this is mostly due to their perception components. If these components were shared across multiple problems then their effective per-task sample complexity would be reduced significantly. In fact, Parisotto et al. (2015) show that pre-training agents on some arcade games, oftentimes helps them learn faster when deployed to play other, new, arcade games. Thus, assuming we can share the perception component across different problems, we only need problem-specific training data for the reasoning component. Moreover, due to the shared reasoning hub, the per-problem sample complexity can be further reduced, because the same reasoning component is used for solving all problems. An interesting setting is one where the perception component can be trained using supervised tasks with differentiable loss functions, and, at the same time, be shared with reinforcement learning (RL) tasks where the reward function is unknown and certainly not differentiable. We believe that this would significantly reduce the sample complexity of the RL tasks. The jelly bean world of Chapter 9 offers a great testbed for testing this hypothesis.

The proposed architecture components reflect assumptions about human cognition that are based on facts derived from psychology experiments, thus rendering this architecture, a cognitive architecture. In the following sections we describe the different architecture components in more detail. Finally, in Section C.7, we describe how learning is performed. Note that, not all architectural components that we describe in the following sections are necessary for all learning problems. Therefore, for some

Modality Examples			
Data Type	Sensor Network	Effector Network	Description
String	BERT Encoder	RNN Decoder	Text
Image	CNN	Deep Convolutional GAN	Image
Scalar[0,1]	—	MLP→Sigmoid	Binary Distribution
Vector[0,1]	—	MLP→Softmax	Categorical Distribution

Table C.1: Example modalities. RNN stands for Recurrent Neural Network, CNN for Convolutional Neural Network, GAN for Generative Adversarial Network, MLP for Multi-Layer Perceptron, `Scalar[0,1]` for a single number in the interval $[0, 1]$, and `Vector[0,1]` for a vector containing numbers in the interval $[0, 1]$.

problems, some of the components may be ignored (e.g., a world simulator may not be relevant for a text classification task).

C.4 PERCEPTION AND ACTION SPOKES

We define perception and action spokes using two kinds of data modalities: (i) *perception modalities* that represent data types that a model can receive as input, and (ii) *action modalities* that represent data types that a model can produce as output. Each kind of modality has a different specification:

- **Perception Modalities:** Input space modalities are defined as tuples (`DataType`, `SensorNetwork`), where `DataType` is the type of data supported by this modality (e.g., `String`), `SensorNetwork` is a contextualized network that takes inputs of type `DataType` and produces vectors of size L_s , and L_s is the reasoning input representation size. Given some data of type `DataType` (e.g., a string of characters with type `String`), and optionally, a context (described in the next section), the `SensorNetwork` produces a vector of size L_s , that the reasoning module can understand.
- **Action Modalities:** Output space modalities are defined as tuples (`DataType`, `EffectorNetwork`), where `DataType` is the type of data supported by this modality (e.g., scalar number in the interval $[0, 1]$), `EffectorNetwork` is a contextualized network that takes as input vectors of size L_e and produces outputs of type `DataType` (e.g., a linear transformation followed by a sigmoid activation function), and L_e is the reasoning output representation size.

Note that a modality can act as both a perception and an action modality, as long as both a *sensor* and an *effector* network are provided. In this case, we also allow the sensor and the effector networks to optionally share some or all of their parameters. Examples of various modalities are shown in Table C.1. Modalities are defined such that, for any given input (or output) data type, there is a single matching perception (or action) modality that will be used.

Due to their generic definition, modalities can be *composed*. For example, given perception modalities $\mathcal{P}_1$ and $\mathcal{P}_2$, we can construct a pair modality `Pair[ $\mathcal{P}_1$ ,  $\mathcal{P}_2$ ]`,

whose data type is a pair of $\mathcal{P}_1.\text{DataType}$ and $\mathcal{P}_2.\text{DataType}$, and whose sensor network is a function of the two modalities' sensor networks. For example:

$$x \mapsto \text{Pool}(\mathcal{P}_1.\text{SensorNetwork}(x[0]), \mathcal{P}_2.\text{SensorNetwork}(x[1])). \quad (\text{C.1})$$

Compositionality gives the proposed NCA high expressive power with respect to the kinds of data it can handle. Compositionality, more generally (e.g., also at the problem space), is a core aspect of the proposed architecture and is discussed in more detail in [Section C.6](#). At the core of how compositionality is being handled lies contextual parameter generation, an abstraction that was described in [Chapter 7](#).

COMMUNICATION AND LANGUAGE. An interesting direction that we also wish to explore in the long term is to add support for a modality that corresponds to communication with other agents (i.e., an artificial learned language). This modality would act as both a perception and an action modality and we could define its data type as a fixed-size vector that contains numeric values, for example. We can test for the ability of agents to learn a language and communicate effectively by conducting experiments in a multi-agent setting where solving certain problems requires coordination and collaboration. This is related to the work of Sukhbaatar et al. (2016) and Andreas et al. (2017).

C.5 REASONING HUB

The reasoning component of the proposed architecture consists of a few parts. At the core lies the *reasoning unit*. This unit transforms the perception component output to an input for the action component and is represented as a contextualized network. It is generally accepted that not all problems require the same amount of reasoning (Kahneman and Egan, 2011). For example, solving an algebra problem requires more *thinking* than recalling your own name. Therefore, we argue that the ability to reason for arbitrary amounts of time, depending on the problem being solved, is an important aspect of general learning and intelligence. Most existing machine learning approaches do not allow for a variable amount of reasoning, as the amount of computation is predefined and fixed as part of the network architecture. The few attempts that do allow for this have been limited to very specific problems and have only shown small gains over preexisting fixed computation time approaches (Graves, 2016; Dehghani et al., 2019). In order to enable this capability in the proposed neural cognitive architecture, we decided to make the reasoning unit *recursive*, meaning that its output can optionally be fed back as input again, to recurse over the reasoning transformation. Each application of the reasoning transformation can be thought of as a reasoning step. The reasoning unit also outputs a decision on whether or not to stop, so that it can stop reasoning and produce an output at some point. The recursive nature of this unit introduces several challenges with respect to how it should be trained. Our initial plan is to incur a *pondering cost*, which is proportional to the number of reasoning steps used, and add that cost to the loss function used to train the reasoning unit.

This is not equivalent to using a recurrent neural network because the number of recursion steps is not predetermined.

RECURSION. More formally, at each time step t , the reasoning unit performs the following transformation:

$$[a_{t+1}, s_{t+1}, \text{STOP}_{t+1}] = R(p_t, a_t, s_t), \quad (\text{C.2})$$

where R represents the reasoning unit transformation, a_t represents the reasoning unit output at time t , s_t represents the internal state of the unit at time t , p_t represents the reasoning unit input at time t which comes from the perception component (note that if the system operates in a real-time environment, this may be different across different reasoning steps), and STOP_t is a boolean flag representing the decision of the reasoning unit about whether or not to stop reasoning at time t . Finally, a_T is fed to the action component, where T is such that $\text{STOP}_T = \text{True}$. Enhancing the reasoning unit with a state significantly increases its modeling capacity. For example, it could even perform a search with backtracking support (e.g., dynamic programming). This initial approach is inspired by the work of Graves (2016).

MEMORY. In designing general learning architectures, we need to allow for an explicit way for learning systems to remember experiences. This can happen implicitly, through the learned model parameters (assuming high capacity networks), but it can also be modeled explicitly by equipping the agent with a memory component. Cognitive architectures often use some form of memory that is symbolic, such as a knowledge-base (KB) that contains learned facts. We propose to add a memory component to our architecture, where all memories are represented in the latent reasoning space, rather than being grounded in the perception or action modality data types. This allows the memory to abstract away details about the data that are not relevant to the reasoning process. The way memory is added to our architecture is through the reasoning unit, which is enhanced such that it can read from and write to memory, while performing the reasoning transformation. More formally, we define the memory component in terms of two functions, $M_{\text{READ}} : K \mapsto V$ and $M_{\text{WRITE}} : (K, V) \mapsto ()$, where K and V correspond to the memory key and value types, respectively, “ $\mapsto$ ” is used to denote the function input and output types, and “ $()$ ” represents the “void” type, meaning that the function returns no values, and is only used for its side effects. Possible design choices for the memory include memory networks (Sukhbaatar et al., 2015), or even KBs defined over the latent reasoning space.

We propose to start with a simple, yet novel,² attention-based memory mechanism. In this case, the memory is defined as a pair of matrices, $M_k \in \mathbb{R}^{M \times D_k}$, and $M_v \in \mathbb{R}^{M \times D_v}$, where M is the memory size, D_k is the dimensionality of the keys, and D_v is the dimensionality of the values stored in the memory. M_k contains the memory keys and M_v contains the corresponding memory values. Let us refer to the K -valued input of M_{READ} and M_{WRITE} as the *query*. Queries are defined as vectors of size D_k . When a component wants to access a value stored in memory, it needs to provide a query “describing” that value. We also define an *indexing* function, $I : K \mapsto \Delta^M$, where Δ^M denotes the M -simplex, which contains all vectors of size M whose elements are in $[0, 1]$ and sum to 1. Intuitively, the indexing function maps from a query to a

Note that, the querying mechanisms are also learned, similar to the indexing mechanism.

² Novel because we are not aware of prior work that learns a memory indexing mechanism.

distribution over memory locations.³ The indexing function that we propose to use initially is the scaled dot-product attention by Vaswani et al. (2017):

$$I(q) = \text{Softmax} \left(\frac{qM_k^T}{\sqrt{D_k}} \right), \quad (\text{C.3})$$

which effectively measures the similarity between the query and all the memory keys. Then, the memory read function is defined as (in pseudocode):

$$M_{\text{READ}}(q) : \text{return } I(q)M_v, \quad (\text{C.4})$$

which returns a convex combination of all stored values, based on the computed index. The memory write function is similarly defined as:

$$M_{\text{WRITE}}(q, v) : M_v := \lambda I(q)v + (1 - \lambda I(q))M_v, \quad (\text{C.5})$$

where “:=” is used to denote assignment, and λ is an M -sized vector with values in $[0, 1]$ that denotes the strength of the write operation. If λ is closer to 1, then old values are forgotten faster. λ can be set adaptively, based on how often each value is being read. For example, it can be set closer to 1 for values that are rarely read. The learnable parameters of this learning mechanism consist of the parameters of I , and the memory keys, M_k . We propose to initialize M_v with zeros.

Allowing the memory indexing mechanism to be learnable by using separate keys and values⁴ enables *associative learning and memories*, which have been shown to be important aspects of human cognition (Fanselow and Poulos, 2005; Ranganath and Ritchey, 2012). In psychology, associative memory is defined as the ability to learn and remember the relationship between unrelated items (e.g., remembering the name of someone or the aroma of a particular perfume). This is enabled by our indexing mechanism because it allows for two unrelated values to have similar keys. This is mainly because we learn keys separately from the values they correspond to. Furthermore, the proposed memory mechanism also allows for a natural way of *forgetting*, where the keys of unused values change while learning to the point where they may be used for storing other unrelated values instead.

We also allow for the sensor and effector networks to optionally read from this memory. This can be important in cases where perception depends on past experiences. Tulving et al. (1982) provides some evidence supporting that this has been observed to be true of human perception (it is known as *priming* in psychology literature).

WORLD SIMULATOR. An important aspect of human reasoning is simulating the external world. Jay Wright Forrester, the father of system dynamics, described a mental model as: “*The image of the world around us, which we carry in our head, is just a model. Nobody in his head imagines all the world, government or country. He has only selected concepts, and relationships between them, and uses those to represent the real system.*” (Forrester, 1971). There is significant evidence of the importance of simulation

³ This can be thought of as a soft, probabilistic, version of memory indexing.

⁴ As opposed to indexing by comparing queries to values as done in memory networks.

in neuroscience (Singer et al., 2018). For example, Nijhawan (1994) shows that to strike a cricket ball one must estimate its future location, rather than where it is now. Bialek et al. (2001) show that prediction has the fundamental theoretical advantage that a system which parsimoniously predicts future inputs from their past, and that generalizes well to new inputs, is likely to contain representations that reflect their underlying causes. Furthermore, they show that much of sensory processing involves discarding irrelevant information, such as that which is not predictive of the future, to arrive at a representation of what is important in the environment for guiding action. Another related line of work is in the importance of auditory feedback (i.e., when we hear ourselves speaking). The study of neural mechanisms underlying audio-vocal integration has shown that auditory feedback may be used for updating internal representations of mappings between voice feedback and speech motor control. One of the earliest demonstrations of the role of auditory feedback in voice control is the Lombard effect, where people raise their voice amplitude to overcome environmental noise (Lombard, 1911; Lane and Tranel, 1971). A related phenomenon is side-tone amplification, in which people increase their voice loudness when their self-perceived loudness is too quiet to achieve a communication goal, and vice versa (Lane and Tranel, 1971). Given this strong evidence from neuroscience, we argue that in an interactive setting, where the learning agent keeps interacting with an outside world—which may also include other agents—being able to simulate that world can be very important. For example, this ability could enable a search over the potential implications its decisions can have on that outside world.

We thus propose to add a *world simulator* component to our neural cognitive architecture. Formally, the simulator S performs the following prediction:

$$\hat{p}_{t+1} = S(p_t, a_{t+1}), \quad (C.6)$$

where $\hat{p}_{t+1}$ is a prediction estimate of p_{t+1} . Furthermore, we allow the world simulator to read from memory (as defined in the previous paragraph), but not write to it. Intuitively, the world simulator is trying to predict the next perception input, given the current perception input and action output, while operating only in the latent reasoning space. Similar to the memory component, this allows the simulator to abstract away information that is not relevant to the problems the system is learning to solve. This type of world simulation in a latent reasoning space is also supported by neuroscientific evidence (e.g., Keller et al., 2012).

Recently, Ha and Schmidhuber (2018) proposed using an RNN-based world simulator for playing games in an RL setting. They use a variational auto-encoder (VAE) to compress the input images to a smaller vector representation and then learn a model that simulates the environment in this vector space. This differs from our proposal in that, we are simulating the world in the latent reasoning space that our system learns. This should help us obtain a representation, that has higher information content that is relevant for the reasoner.

C.6 GOAL CONTEXTUALIZATION

Even though deep learning methods are very effective at learning representations for arbitrary data modalities, they are often treated as black-box methods offering little control over how information is shared across different tasks, and over what exactly the networks are learning. For example, we can rarely guarantee that a network will generalize well to new tasks, and we often also have to keep training the network with new problem-specific data, in order for it to generalize better. Furthermore, deep learning approaches often render generalizing to new tasks, for which we might have no data at all, impossible. However, most real-world problems can be defined in terms of simpler problems (e.g., translating sentences relies on first being able to translate single words). Therefore, we argue that the ability to represent problems in a way such that they can be transformed and composed out of other problems, is an important aspect of general learning and intelligence. As discussed in [Chapter 7](#), this motivated our work on contextual parameter generation (CPG), and forms the basis of *contextualization*. In the proposed neural cognitive architecture, contextualization plays the important role of emulating the *goal priming* mechanism that is inherent in human intelligence and learning. We now describe how this is achieved, in three parts: (i) we first describe how problems (or goals) are specified through some language, (ii) we then define an architectural component that *compiles* the problem specification to a representation that can be used to contextualize other parts of the NCA by using CPG, and (iii) we describe how this allows for the learning system to generate its own target problems (or goals) that it aims to learn.

As shown in [Figure C.2](#), we also allow the sensor and effector networks to be contextualized because perception and action are often not independent of the problem being solved. This is motivated by the fact that priming in humans can be perceptual, semantic, and conceptual (Bargh and Chartrand, 2014). From a machine learning perspective, we have also shown the usefulness of contextualizing equivalents of perception and action modules, when we proposed using CPG for universal neural machine translation in [Section 8.2](#).

PROBLEM SPECIFICATION. We first need a representation for problems. We propose to use a fixed language for this representation, which could take multiple forms:

- **Fixed-Size Vector:** Problems could be represented as continuous-valued, fixed-size, vectors (e.g., Snell et al., 2017; Wang et al., 2017b; Grover et al., 2018). For example, given a fixed number of pre-specified problems the system may learn vector embeddings to represent them. The main disadvantage of this approach is that the vector representations of learning problems may not be interpretable.
- **Natural Language:** This could be a problem description that is provided as input to the system (e.g., “Identify human faces in the input image.”). This is the approach taken, for example, by McCann et al. (2018).
- **Structured Language:** This could be first-order logic (e.g., “Collect[JellyBean] $\wedge \neg$ Collect[Onion]”), or more general (e.g., “If[JellyBean] Then[Collect] Else[Avoid]”, or even a Python program).

Problem Compiler Examples		
Specification	Compiled Form	Explanation
Classify[City]	$g_{\text{Classify}}(c_{\text{City}})$	Predict if the input (e.g., "Washington") is a city.
Classify[City $\wedge$ $\neg$ Person]	$g_{\text{Classify}}(g_{\wedge}(c_{\text{City}}, g_{\neg}(c_{\text{Person}})))$	Predict if the input is a city and not a person.
Caption[Image, English]	$g_{\text{Caption}}(c_{\text{English}})$	Generate a short English sentence describing the input (e.g., generate captions for images).
Translate[English, German]	$g_{\text{Translate}}(c_{\text{English}}, c_{\text{German}})$	Translate the input from English to German. This is interesting because the modularity of our architecture means that this problem specification could even be used to translate images containing text, for example.

Table C.2: Example uses of the problem compiler. We use c with different subscripts to denote context vectors representing primitives in the problem specification language, and g with different subscripts to denote transformation functions for context vectors (which could be defined as learnable neural networks, for example).

PROBLEM COMPILATION. Given a problem specification, we need to define a *compiler* that takes it as input and produces a composition of learnable functions that, when evaluated, results in a single structured representation for the problem (e.g., a set of vectors). This representation can then be used to contextualize different parts of the proposed architecture (e.g., sensor or effector networks, or parts of the reasoner that are discussed in the next section). Given that the representation can potentially be a set of vectors, we could use different parts of that structure to contextualize different parts of the architecture. For example, text sensor networks could be contextualized using an embedding of the language in which the text is written. Note that, contextualizing networks is optional as it is sometimes not necessary.

The choice of the problem compiler is important. For fixed-size vectors and natural language specifications the compiler could be as simple as just a neural network (e.g., a multi-layer perceptron, a recurrent neural network, or a Transformer network). However, for other structured languages the compiler would be something more similar to programming languages compilers. Some examples of representations and their corresponding compiled forms are shown in [Table C.2](#). Following from the previous section examples, given a problem specification that is written as a Python program, we could also compile it into a composition of learnable functions.

This definition of problem specifications and problem compilers allows us to make the contextualization mechanism very flexible and extensible by introducing operators that compose compiled forms in arbitrary ways. For example, we could have two problem specifications, each with their own compiler, and a separate operator that allows us to merge the two compiled forms, resulting in a single final context vector. A simple form of a problem compiler was used in [Section 8.4](#).

PROBLEM GENERATION. An important aspect of human learning is that, even though nature provides us some reward signals for our actions (e.g., eating resolves hunger), we often “invent” new problems that we learn to solve. We could argue that this is a way of structuring much larger overarching problems into multiple subproblems. This human behavior aspect is very interesting and, at the same time, not really tackled by current machine learning systems. Therefore, we propose to let our learning system “invent” problems on its own. For this paragraph, we will use a reinforcement learning setting where a learning agent can perceive certain things about the environment in which it “lives” and take actions. Oftentimes, the agent receives a reward, but it may not know why. Thus, in such a setting, it would make sense for the agent to try and “invent” problems to solve, that would result in higher collected rewards. We propose to introduce an additional action modality that allows the agent to generate problem specifications, which are then directly fed in the problem compiler, and can contextualize multiple parts of the architecture.

In this case, we assume that no problem specification is provided to the agent as input.

For the fixed-size vector specification format, this could be implemented by having the effector network output a vector representing the problem. Perhaps more interestingly though, we could define a structured language that only depends on the agent’s perception and action modalities. This would allow the agent to generate arbitrary problem specifications that only depend on what it is able to perceive and how it can act. For example, given a perception modality that identifies the types of items in the environment, and an action modality that can collect items, we could define the problem specification language to be:

$$(\neg)\text{Collect}[\langle\text{Item}\rangle](\wedge(\neg)\text{Collect}[\langle\text{Item}\rangle])^*,$$

where $\neg$ denotes the logical NOT operation, $\wedge$ the logical AND operation, parenthesis denote optional parts, $\langle\text{Item}\rangle$ denotes any item type that can be sensed by the item identification perception modality, and $*$ denotes that the term in parenthesis preceding it can be repeated zero or more times. Note that $\text{Collect}[\cdot]$ acts as a logic predicate that can be applied on any item type. An example specification in this language that was used in [Section 8.4](#) and [Chapter 9](#), is $\text{Collect}[\text{JellyBean}]\wedge\neg\text{Collect}[\text{Onion}]$. We propose to formalize this problem generation mechanism and allow learning systems to decide on the problems they are learning to solve.

C.7 LEARNING MECHANISMS

The architecture components presented so far depend on parameters that need to be learned (e.g., the weights of the neural network layers that are used). Learning consists of setting the values of these parameters so that the system as a whole can solve the target problems. We assume that all components are formulated as functions that are differentiable with respect to their parameters.⁵ Under this assumption, we define our learning mechanism as follows:

⁵ Note that this is a very general assumption that holds for most deep learning models, and a lot of machine learning models, more generally.

1. Each action modality can optionally provide a *feedback mechanism*. Let us denote the output of the modality's effector network as a function $f_\theta(x)$, where x represents all inputs that it depends on. In this case, f represents the composition of all architecture modules that participated in producing this output (i.e., this includes the reasoning module, the goal contextualization module, and the relevant perception sensor networks). Then, we define the feedback mechanism as a function h of $f_\theta(x)$ and the external environment. For example, if $f_\theta(x)$ is producing a distribution over classes (for a multi-class classification problem), h could be defined as:

$$h(f_\theta(x), y) = f_\theta(x) - y, \quad (C.7)$$

where y represents a one-hot representation of the true class assignment provided by the environment. The main constraint on h is that it should produce an output that can be multiplied with $\nabla_\theta f_\theta(x)$.

2. Whenever an action modality produces an output and a corresponding feedback signal is returned from the environment, a gradient-based parameter update is performed along the following direction:

$$\mathcal{D}_\theta \triangleq \underset{\substack{\downarrow \\ \text{External}}}{h(f_\theta(x), \mathcal{E})} \underset{\substack{\downarrow \\ \text{Internal}}}{\nabla_\theta f_\theta(x)}, \quad (C.8)$$

where $\mathcal{E}$ represents the external environment. Note that the first part—shown in **blue**—is provided from the external environment, whereas the second part—shown in **red**—can be computed internally from the learning system itself. This separation is interesting from a human cognition perspective because, intuitively: (i) a human would know how to tweak their brain to move their hand further forward (**internal update**), while (ii) the external environment could tell them that to achieve a particular goal they would need to move their hand forward (**external update**). The model update could be a stochastic gradient descent step:

$$\theta_{t+1} = \theta_t + \lambda_t \mathcal{D}_{\theta_t}, \quad (C.9)$$

where λ_t represents the learning rate. This update could also be more elaborate by using the Adam (Kingma and Ba, 2015) or AMSGrad (Reddi et al., 2018) optimizer, for example.

Equation C.8 is interesting because it can be used to unified multiple different learning paradigms, such as supervised, semi-supervised, unsupervised, and reinforcement learning, under one formulation. For example:

- **Supervised Learning:** In this case, the gradient-based updates are computed by differentiating a loss function, $\mathcal{L}(f_\theta(x), \mathcal{E})$. This fits in our formulation by defining the feedback mechanism using the chain rule of differentiation:

$$h(f_\theta(x), \mathcal{E}) = \frac{\partial \mathcal{L}(f_\theta(x), \mathcal{E})}{\partial f_\theta(x)}. \quad (C.10)$$

For example, for the ℓ_2 loss we have $h(f_\theta(x), y) = f_\theta(x) - y$, and for the cross-entropy classification loss we have $h(f_\theta(x), y) = y/f_\theta(x)$.

- Semi-Supervised Learning: Can often also be formulated in terms of minimizing a differentiable loss function and thus [Equation C.10](#) also applies here.
- Unsupervised Learning: In this case, $h(f_\theta(x), \mathcal{E})$ does not depend on $\mathcal{E}$ at all and could be defined internally as well. More specifically, h could be used to perform some sort of *self-reflection*, as discussed in [Part i](#).
- Reinforcement Learning: In the case of Q-learning (Watkins and Dayan, 1992), we can have an action modality that predicts the Q-function value (Mnih et al., 2013) and then the learning mechanism can use a supervised learning feedback function h to learn it using the rewards provided by the environment. In the case of policy gradient methods (Sutton et al., 2000), h can be defined as the advantage function being used, or even as some function of the advantage for more complex methods (Mnih et al., 2016; Schulman et al., 2017; Wang et al., 2017a). More interestingly, if we want to use experience replay as done by (Mnih et al., 2013), we could develop a variant where: (i) the perception and action modality parameters are fixed and we are training only the problem compiler and the reasoning modules, and (ii) the stored experiences that are replayed are not represented in the original data space, but rather in the more abstract and compact reasoning space. This has the significant advantage of being able to store a lot more experiences, as memory is typically the bottleneck when using experience replay. Furthermore, we would only be storing information that is relevant to reasoning.

Our learning mechanism manages all feedback mechanisms and determines how to apply the corresponding updates and what learning rate to use for each one. Initially, we plan to use the same learning rate for all parameters and feedback mechanisms with exponential decay over time. However, our definition allows us to use potentially different learning rates for each parameter and for each learning goal (defined by corresponding feedback mechanisms). Next, we plan to integrate the ideas presented in [Part i](#) to this learning mechanism. In the long term, we would like to explore other interesting directions such as *staged learning*.

STAGED LEARNING. The aforementioned reinforcement learning example on experience replay demonstrates the idea of staged learning. In staged learning, we freeze the learning of the perception and action modalities early on during training (e.g., by significantly lowering the corresponding learning rate), and then focus more on training the reasoning module. As discussed in the beginning of this appendix, this would be more similar to how human learning works. Assuming that the perception and action modality networks have already been trained using a diverse set of learning goals, freezing them should allow for the reasoning module to tackle new learning goals in a fixed latent space, determined by these pretrained networks. We believe that this will result in significantly faster training times.

MIXED-PARADIGM LEARNING. As shown earlier, our learning mechanism is a generalization of multiple existing learning paradigms thus allowing us to mix them

together by simply intertwining their gradient-based updates. For example, we can take a gradient descent step towards minimizing a supervised cross-entropy classification loss, and then take a gradient descent step that improves the current Q-function estimate, in a reinforcement learning setting. This introduces multiple challenges that we will have to overcome, including, but not limited to:

- How do we balance the gradient contribution from each learning problem?
- How do we set the per-learning-goal and per-parameter learning rates?
- How do we make the learning mechanism scale?
- How do we properly batch the training data?

Other learning paradigms, such as active learning and curriculum learning, can also be supported by designing appropriate perception and action modalities.

C.8 NEXT STEPS

A never-ending learning system must be highly modular and allow for the addition and removal of modules without requiring a complete retraining from scratch. For this reason, we propose to implement the architecture in a highly modular manner, with each module being completely independent of the rest and having a fixed, well-defined, and generic interface. This will allow for adding and removing perception and action modalities and for extending the problem specification language, without requiring a complete retraining from scratch every time such a modification is made. Furthermore, each module will be solely responsible for persisting its state, so that we can keep extending the architecture and avoiding training restarts, as much as possible. The next steps in this work would be to implement simple variants of this architecture and test them on the jelly bean world of [Chapter 9](#) as well as some real-world problems. The goal of these evaluations would be to provide convincing evidence for the never-ending learning capabilities of the proposed architecture. Moreover, unlike NELL (Mitchell et al., 2018), we aim for this system to fully avoid complete training restarts throughout its lifetime. Finally, as an even longer term goal, we want to explore directions where the latent reasoning representation is also extensible without requiring complete training restarts.

BIBLIOGRAPHY

- [1] Etienne Lombard. "Le signe de l'elevation de la voix." In: *Ann. Mal. de L'Oreille et du Larynx* (1911), pp. 101–119.
- [2] Richard H. Richens. "Interlingual Machine Translation." In: *The Computer Journal* 1.3 (1958), pp. 144–147.
- [3] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs." In: *Numerische Mathematik* 1 (1 1959). 10.1007/BF01386390, pp. 269–271. ISSN: 0029-599X.
- [4] Myron Tribus. *Thermostatics and Thermodynamics*. D. Van Nostrand Company, 1961.
- [5] A. N. Kolmogorov. "On Tables of Random Numbers." In: *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)* 25.4 (1963), pp. 369–376. ISSN: 0581572X.
- [6] Marvin Minsky and Seymour Papert. "An Introduction to Computational Geometry." In: *Cambridge tracts., HIT* (1969).
- [7] Jay W. Forrester. "Counterintuitive Behavior of Social Systems." In: *Technological Forecasting and Social Change* 3 (1971), pp. 1–22.
- [8] John M. Hammersley and Peter E. Clifford. "Markov Random Fields on Finite Graphs and Lattices." In: (1971).
- [9] Harlan Lane and Bernard Tranel. "The Lombard Sign and the Role of Hearing in Speech." In: *Journal of Speech and Hearing Research* 14.4 (1971), pp. 677–709.
- [10] David Blackwell and James B. MacQueen. "Ferguson Distributions Via Polya Urn Schemes." In: *The Annals of Statistics* 1.2 (Mar. 1973), pp. 353–355.
- [11] Thomas S. Ferguson. "A Bayesian Analysis of Some Nonparametric Problems." In: *The Annals of Statistics* 1.2 (Mr 1973), pp. 209–230.
- [12] Charles E. Antoniak. "Mixtures of Dirichlet Processes with Applications to Bayesian Nonparametric Problems." In: *The Annals of Statistics* 2.6 (1974), pp. 1152–1174.
- [13] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm." In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1977), pp. 1–38.
- [14] A. P. Dawid and A. M. Skene. "Maximum Likelihood Estimation of Observer Error-Rates Using the EM Algorithm." In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979), pp. 20–28.
- [15] David G. Marr. "Language and Literacy." In: 1981, pp. 136–189.

- [16] Endel Tulving, Daniel L. Schacter, and Heather A. Stark. "Priming Effects in Word-Fragment Completion are Independent of Recognition Memory." In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 8.4 (1982), p. 336.
- [17] Mortimer Mishkin, Leslie G. Ungerleider, and Kathleen A. Macko. "Object Vision and Spatial Vision: Two Cortical Pathways." In: *Trends in Neurosciences* 6 (1983), pp. 414–417.
- [18] Stuart Geman and Donald Geman. "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 6 (Nov. 1984), pp. 721–741. ISSN: 0162-8828.
- [19] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Internal Representations by Error Propagation*. Tech. rep. California University of San Diego La Jolla Institute for Cognitive Science, 1985.
- [20] Lawrence R. Rabiner. "An Introduction to Hidden Markov Models." In: *IEEE Acoustics, Speech, and Signal Processing Magazine (ASSP)* 3.1 (1986), pp. 4–16.
- [21] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-Propagating Errors." In: *Nature* 323.6088 (1986), pp. 533–536. DOI: [10.1038/323533a0](https://doi.org/10.1038/323533a0). URL: <https://doi.org/10.1038/323533a0>.
- [22] Terry Sejnowski. "NET Talk: A Parallel Network that Learns to Read Aloud." In: *Complex Systems* 1 (1987), pp. 145–168.
- [23] Dean A. Pomerleau. "ALVINN: An Autonomous Land Vehicle in a Neural Network." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 1989, pp. 305–313.
- [24] Yaser S. Abu-Mostafa. "Learning from Hints in Neural Networks." In: *Journal of Complexity* 6.2 (1990), pp. 192–198.
- [25] Thomas G. Dietterich, Hermann Hild, and Ghulum Bakiri. "A Comparative Study of ID₃ and Backpropagation for English Text-to-Speech Mapping." In: *Machine Learning Proceedings*. Elsevier, 1990, pp. 24–31.
- [26] Alan E. Gelfand and Adrian F. M. Smith. "Sampling-Based Approaches to Calculating Marginal Densities." In: *Journal of the American Statistical Association* 85.410 (1990), pp. 398–409.
- [27] Allen Newell. *Unified Theories of Cognition*. Cambridge, MA, USA: Harvard University Press, 1990. ISBN: 0-674-92099-6.
- [28] S. C. Suddarth and Y. L. Kergosien. "Rule-Injection Hints as a Means of Improving Network Performance and Learning Time." In: *EURASIP Workshop*. 1990.
- [29] Daniel J. Felleman and D. C. Essen Van. "Distributed Hierarchical Processing in the Primate Cerebral Cortex." In: *Cerebral Cortex* 1.1 (1991), pp. 1–47.
- [30] Peter W. Frey and David J. Slate. "Letter Recognition Using Holland-Style Adaptive Classifiers." In: *Machine Learning* 6 (1991), p. 161.

- [31] Lorien Y. Pratt, Jack Mostow, Candace A. Kamm, and Ace A. Kamm. "Direct Transfer of Learned Information Among Neural Networks." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. Vol. 91. 1991, pp. 584–589.
- [32] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning." In: *Machine Learning* 8.3-4 (1992), pp. 279–292.
- [33] Virginia de Sa. "Minimizing Disagreement for Self-Supervised Classification." In: *Connectionist Models Summer School* (1993), pp. 300–307.
- [34] Jeffrey L. Elman. "Learning and Development in Neural Networks: The Importance of Starting Small." In: *Cognition* 48.1 (1993), pp. 71–99.
- [35] C. Feng, A. Sutherland, R. King, S. Muggleton, and R. Henery. "Comparison of Machine Learning Classifiers to Statistics and Neural Networks." In: *Proceedings of the Third International Workshop in Artificial Intelligence and Statistics*. 1993, pp. 41–52.
- [36] Tom M. Mitchell and Sebastian B. Thrun. "Explanation-Based Neural Network Learning for Robot Control." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 1993, pp. 287–294.
- [37] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent is Difficult." In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166.
- [38] Thomas G. Dietterich and Ghulum Bakiri. "Solving Multiclass Learning Problems via Error-Correcting Output Codes." In: *Journal of Artificial Intelligence Research (JAIR)* 2 (1994), pp. 263–286.
- [39] Romi Nijhawan. "Motion Extrapolation in Catching." In: *Nature* (1994).
- [40] Sebastian B. Thrun and Tom M. Mitchell. *Learning One More Thing*. Tech. rep. Carnegie Mellon University, 1994.
- [41] Jonathan Baxter. *Learning Internal Representations*. 1995.
- [42] Rich Caruana. "Learning Many Related Tasks at the Same Time with Back-propagation." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 1995, pp. 657–664.
- [43] George J. Klir and Bo Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice-Hall, Inc., 1995. ISBN: 0-13-101171-5.
- [44] Jürgen Schmidhuber. "On Learning How to Learn Learning Strategies." In: (1995).
- [45] Noel E. Sharkey and Amanda J. C. Sharkey. "An Analysis of Catastrophic Interference." In: *Connection Science* (1995).
- [46] Fevzi Alimoglu and Ethem Alpaydin. "Methods of Combining Multiple Classifiers Based on Different Representations for Pen-based Handwritten Digit Recognition." In: *Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAINN)*. 1996.

- [47] Tanya L. Chartrand and John A. Bargh. "Automatic Activation of Impression Formation and Memorization Goals: Nonconscious Goal Priming Reproduces Effects of Explicit Task Instructions." In: *Journal of Personality and Social Psychology* 71.3 (1996), p. 464.
- [48] Kim Leng Poh and Eric Horvitz. "A Graph-Theoretic Analysis of Information Value." In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 1996, pp. 1–10.
- [49] Daniel L. Silver and Robert E. Mercer. "The Parallel Transfer of Task Knowledge Using Dynamic Learning Rates Based on a Measure of Relatedness." In: *Learning to Learn*. Springer, 1996, pp. 213–233.
- [50] Sebastian B. Thrun. "Explanation-Based Neural Network Learning." In: *Explanation-Based Neural Network Learning*. Springer, 1996, pp. 19–48.
- [51] Sebastian B. Thrun and Joseph O'Sullivan. "Discovering Structure in Multiple Learning Tasks: The TC Algorithm." In: *Proceedings of the International Conference on Machine Learning (ICML)*. 1996.
- [52] Jonathan Baxter. "A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling." In: *Machine Learning* 28.1 (1997), pp. 7–39. DOI: [10.1023/A:1007327622663](https://doi.org/10.1023/A:1007327622663).
- [53] Leo Breiman and Jerome H. Friedman. "Predicting Multivariate Responses in Multiple Linear Regression." In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 59.1 (1997), pp. 3–54.
- [54] Rich Caruana. "Multitask Learning." In: *Machine Learning* 28.1 (1997), pp. 41–75.
- [55] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory." In: *Neural Computation* 9.8 (1997), pp. 1735–1780.
- [56] Paul W. Munro and Bambang Parmanto. "Competition among Networks Improves Committee Performance." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 1997, pp. 592–598.
- [57] Mark B. Ring. "CHILD: A First Step Towards Continual Learning." In: *Machine Learning* 28.1 (July 1997), pp. 77–104. ISSN: 1573-0565. DOI: [10.1023/A:1007331723572](https://doi.org/10.1023/A:1007331723572). URL: <https://doi.org/10.1023/A:1007331723572>.
- [58] Avrim Blum and Tom M. Mitchell. "Combining Labeled and Unlabeled Data with Co-Training." In: *Conference on Computational Learning Theory (COLT)*. 1998, pp. 92–100. DOI: [10.1145/279943.279962](https://doi.org/10.1145/279943.279962).
- [59] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.
- [60] Jon H. Kaas and Troy A. Hackett. "Subdivisions of Auditory Cortex and Levels of Processing in Primates." In: *Audiology and Neurotology* 3.2-3 (1998), pp. 73–85.
- [61] Josef P. Rauschecker. "Cortical Processing of Complex Sounds." In: *Current Opinion in Neurobiology* 8.4 (1998), pp. 516–521.

- [62] Sebastian B. Thrun and Joseph O’Sullivan. “Clustering Learning Tasks and the Selective Cross-Task Transfer of Knowledge.” In: *Learning to Learn*. Springer, 1998, pp. 235–257.
- [63] Sebastian Thrun and Lorien Pratt. *Learning to Learn*. Springer, 1998.
- [64] Michael Collins and Yoram Singer. “Unsupervised Models for Named Entity Classification.” In: *Joint Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*. 1999, pp. 1–11.
- [65] Michael I. Jordan, Zoubin Ghahramani, Tommi Jaakkola, and Lawrence K. Saul. “An Introduction to Variational Methods for Graphical Models.” In: *Machine Learning* 37.2 (1999), pp. 183–233.
- [66] Lizabeth M. Romanski, Joseph F. Bates, and Patricia S. Goldman-Rakic. “Auditory Belt and Parabelt Projections to the Prefrontal Cortex in the Rhesus Monkey.” In: *Journal of Comparative Neurology* 403.2 (1999), pp. 141–157.
- [67] Robert E. Schapire. “A Brief Introduction to Boosting.” In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. Vol. 2. Morgan Kaufmann Publishers Inc., 1999, pp. 1401–1406.
- [68] Adrian Baddeley and Rolf Turner. “Practical Maximum Pseudolikelihood for Spatial Point Patterns.” In: *Australian & New Zealand Journal of Statistics* 42.3 (2000), pp. 283–322. DOI: [10.1111/1467-842X.00128](https://doi.org/10.1111/1467-842X.00128).
- [69] Felix A Gers and Jürgen Schmidhuber. “Recurrent Nets that Time and Count.” In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN)*. Vol. 3. IEEE. 2000, pp. 189–194.
- [70] Tom Heskes. “Empirical Bayes for Learning to Learn.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2000.
- [71] Radford M. Neal. “Markov Chain Sampling Methods for Dirichlet Process Mixture Models.” In: *Journal of Computational and Graphical Statistics* 9.2 (June 2000), pp. 249–265.
- [72] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. “Policy Gradient Methods for Reinforcement Learning with Function Approximation.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2000, pp. 1057–1063.
- [73] William Bialek, Ilya Nemenman, and Naftali Tishby. “Predictability, Complexity, and Learning.” In: *Neural Computation* 13.11 (2001), pp. 2409–2463.
- [74] Avrim Blum and Shuchi Chawla. “Learning from Labeled and Unlabeled Data using Graph Mincuts.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2001.
- [75] Sanjoy Dasgupta, Michael L Littman, and David McAllester. “PAC Generalization Bounds for Co-training.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2001, pp. 375–382.

- [76] Nicholas Roy and Andrew McCallum. "Toward Optimal Active Learning through Sampling Estimation of Error Reduction." In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2001, pp. 441–448.
- [77] C. M. Wessinger, J. VanMeter, Biao Tian, J. Van Lare, J. Pekar, and Josef P. Rauschecker. "Hierarchical Organization of the Human Auditory Cortex Revealed by Functional Magnetic Resonance Imaging." In: *Journal of Cognitive Neuroscience* 13.1 (2001), pp. 1–7.
- [78] Robert J. Zatorre and Pascal Belin. "Spectral and Temporal Processing in Human Auditory Cortex." In: *Cerebral Cortex* 11.10 (2001), pp. 946–953.
- [79] Shai Ben-David, Johannes Gehrke, and Reba Schuller. "A Theoretical Framework for Learning from a Pool of Disparate Data Sources." In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2002, pp. 443–449. ISBN: 1-58113-567-X. DOI: [10.1145/775047.775111](https://doi.org/10.1145/775047.775111).
- [80] Bart Bakker and Tom Heskes. "Task Clustering and Gating for Bayesian Multi-task Learning." In: *Journal of Machine Learning Research (JMLR)* 4. May (2003), pp. 83–99.
- [81] Yoshua Bengio and Nicolas Chapados. "Extensions to Metric-Based Model Selection." In: *Journal of Machine Learning Research (JMLR)* 3 (Mar. 2003), pp. 1209–1227.
- [82] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. "Latent Dirichlet Allocation." In: *Journal of Machine Learning Research (JMLR)* 3. Jan (2003), pp. 993–1022.
- [83] Qing Lu and Lise Getoor. "Link-based Classification." In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2003, pp. 496–503.
- [84] Jason D. Warren and Timothy D. Griffiths. "Distinct Mechanisms for Processing Spatial Sequences and Pitch Sequences in the Human Auditory Brain." In: *Journal of Neuroscience* 23.13 (2003), pp. 5799–5804.
- [85] Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. "Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions." In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2003, pp. 912–919.
- [86] John R. Anderson, Daniel Bothell, Michael D. Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. "An Integrated Theory of the Mind." In: *Psychological Review* 111.4 (2004), p. 1036.
- [87] Theodoros Evgeniou and Massimiliano Pontil. "Regularized Multi-Task Learning." In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM. 2004, pp. 109–117.
- [88] Herbert Jaeger and Harald Haas. "Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication." In: *Science* 304.5667 (2004), pp. 78–80.

- [89] Tony Jebara. “Multi-Task Feature and Kernel Selection for SVMs.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. ACM, 2004, p. 55.
- [90] Neil D. Lawrence and John C. Platt. “Learning to Learn with the Informative Vector Machine.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. ACM, 2004, p. 65. ISBN: 1-58113-838-5. DOI: [10.1145/1015330.1015382](https://doi.org/10.1145/1015330.1015382).
- [91] Omid Madani, David M. Pennock, and Gary W. Flake. “Co-Validation: Using Model Disagreement on Unlabeled Data to Validate Classification Algorithms.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2004, pp. 1–8.
- [92] Timothy T. Rogers, Lambon Ralph, A. Matthew, Peter Garrard, Sasha Bozeat, James L. McClelland, John R. Hodges, and Karalyn Patterson. “Structure and Deterioration of Semantic Memory: A Neuropsychological and Computational Investigation.” In: *Psychological Review* 111.1 (2004), p. 205.
- [93] Robert J. Zatorre, Marc Bouffard, and Pascal Belin. “Sensitivity to Auditory Object Features in Human Temporal Neocortex.” In: *Journal of Neuroscience* 24.14 (2004), pp. 3637–3642.
- [94] Dengyong Zhou, Olivier Bousquet, Thomas N. Lal, Jason Weston, and Bernhard Schölkopf. “Learning with Local and Global Consistency.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2004, pp. 321–328.
- [95] Rie Kubota Ando and Tong Zhang. “A Framework for Learning Predictive Structures from Multiple Tasks and Unlabeled Data.” In: *Journal of Machine Learning Research (JMLR)* 6 (2005), pp. 1817–1853.
- [96] Aron Culotta and Andrew McCallum. “Reducing Labeling Effort for Structured Prediction Tasks.” In: *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI) - Volume 2*. 2005, pp. 746–751.
- [97] Ruud Custers and Henk Aarts. “Positive Affect as Implicit Motivator: On the Nonconscious Operation of Behavioral Goals.” In: *Journal of Personality and Social Psychology* 89.2 (2005), p. 129.
- [98] Theodoros Evgeniou, Charles A. Micchelli, and Massimiliano Pontil. “Learning Multiple Tasks with Kernel Methods.” In: *Journal of Machine Learning Research (JMLR)* 6.Apr (2005), pp. 615–637.
- [99] Michael S. Fanselow and Andrew M. Poulos. “The Neuroscience of Mammalian Associative Learning.” In: *Annual Review of Psychology* 56 (2005), pp. 207–234.
- [100] Kai Yu, Volker Tresp, and Anton Schwaighofer. “Learning Gaussian Processes from Multiple Tasks.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. ACM, 2005, pp. 1012–1019. ISBN: 1-59593-180-5. DOI: [10.1145/1102351.1102479](https://doi.org/10.1145/1102351.1102479).

- [101] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. "Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples." In: *Journal of Machine Learning Research (JMLR)* 7.Nov (2006), pp. 2399–2434.
- [102] Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. "Label Propagation and Quadratic Criterion." In: *Semi-Supervised Learning*. 2006. Chap. 11.
- [103] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.
- [104] Matthias Hein and Markus Maier. "Manifold Denoising." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2006.
- [105] Jeff Howe. "The Rise of Crowdsourcing." In: *Wired Magazine* 14.6 (2006), pp. 1–4.
- [106] Aapo Hyvärinen. "Consistency of Pseudolikelihood Estimation of Fully Visible Boltzmann Machines." In: *Neural Computation* 18.10 (2006), pp. 2283–2292.
- [107] Matthew Richardson and Pedro Domingos. "Markov Logic Networks." In: *Machine Learning* 62.1-2 (Feb. 2006), pp. 107–136.
- [108] Dale Schuurmans, Finnegan Southey, Dana Wilkinson, and Yuhong Guo. "Metric-Based Approaches for Semi-Supervised Regression and Classification." In: *Semi-Supervised Learning*. 2006, pp. 1–31.
- [109] Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. "Hierarchical Dirichlet Processes." In: *Journal of the American Statistical Association* 101.476 (Dec. 2006), pp. 1566–1581.
- [110] S. V. N. Vishwanathan, Nicol N. Schraudolph, Mark W. Schmidt, and Kevin P. Murphy. "Accelerated Training of Conditional Random Fields with Stochastic Gradient Methods." In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2006, pp. 969–976.
- [111] Ming Yuan and Yi Lin. "Model Selection and Estimation in Regression with Grouped Variables." In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1 (2006), pp. 49–67.
- [112] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. "Multi-Task Feature Learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2007, pp. 41–48.
- [113] Ming-Wei Chang, Lev Ratinov, and Dan Roth. "Guiding Semi-Supervision with Constraint-Driven Learning." In: *Annual Meeting of the Association of Computational Linguistics (ACL)*. Prague, Czech Republic, June 2007, pp. 280–287.
- [114] Jingrui He, Jaime G Carbonell, and Yan Liu. "Graph-Based Semi-Supervised Learning as a Generative Model." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2007.

- [115] Ashish Kapoor, Eric Horvitz, and Sumit Basu. “Selective Supervision: Guiding Supervised Learning with Decision-Theoretic Active Learning.” In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2007, pp. 877–882.
- [116] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. “Moses: Open Source Toolkit for Statistical Machine Translation.” In: *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*. 2007, pp. 177–180.
- [117] Stanley Kok and Pedro Domingos. “Statistical Predicate Invention.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2007, pp. 433–440.
- [118] Charles Sutton and Andrew McCallum. “Piecewise Pseudolikelihood for Efficient Training of Conditional Random Fields.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2007, pp. 863–870.
- [119] Ya Xue, Xuejun Liao, Lawrence Carin, and Balaji Krishnapuram. “Multi-Task Learning for Classification with Dirichlet Process Priors.” In: *Journal of Machine Learning Research (JMLR)* 8 (2007), pp. 35–63.
- [120] Henk Aarts, Ruud Custers, and Martijn Velkamp. “Goal Priming and the Affective-Motivational Route to Nonconscious Goal Pursuit.” In: *Social Cognition* 26.5 (2008), pp. 555–577.
- [121] Austin Appleby. *MurmurHash*. <https://sites.google.com/site/murmurhash/>. 2008.
- [122] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. “Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge.” In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’08. ACM, 2008, pp. 1247–1250. DOI: [10.1145/1376616.1376746](https://doi.org/10.1145/1376616.1376746).
- [123] Bob Carpenter. “Multilevel Bayesian Models of Categorical Data Annotation.” In: *Unpublished manuscript* 17.122 (2008), pp. 45–50.
- [124] Giovanni Cavallanti, Nicolò Cesa-Bianchi, and Claudio Gentile. “Linear Algorithms for Online Multitask Classification.” In: *Journal of Machine Learning Research (JMLR)* 11 (2008), pp. 2901–2934.
- [125] Ming Chang, Lev Ratinov, Nicholas Rizzolo, and Dan Roth. “Learning and Inference with Constraints.” In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2008, pp. 1513–1518.
- [126] Ronan Collobert and Jason Weston. “A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. Helsinki, Finland: ACM, 2008, pp. 160–167. ISBN: 978-1-60558-205-4. DOI: [10.1145/1390156.1390177](https://doi.org/10.1145/1390156.1390177). URL: <http://doi.acm.org/10.1145/1390156.1390177>.

- [127] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. “Kernel Methods in Machine Learning.” In: *The Annals of Statistics* (2008), pp. 1171–1220.
- [128] Roi Reichart, Katrin Tomanek, Udo Hahn, and Ari Rappoport. “Multi-Task Active Learning for Linguistic Annotations.” In: *Annual Meeting of the Association for Computational Linguistics (ACL)*. 2008, pp. 861–869.
- [129] D. Roth and K. Small. “Active Learning for Pipeline Models.” In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 2008.
- [130] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. “Collective Classification in Network Data.” In: *AI Magazine* 29.3 (2008), pp. 93–93.
- [131] Burr Settles and Mark Craven. “An Analysis of Active Learning Strategies for Sequence Labeling Tasks.” In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Oct. 2008, pp. 1070–1079.
- [132] Rion Snow, Brendan O’Connor, Daniel Jurafsky, and Andrew Y. Ng. “Cheap and Fast—But is it Good? Evaluating Non-Expert Annotations for Natural Language Tasks.” In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics. 2008, pp. 254–263.
- [133] Martijn Veltkamp, Henk Aarts, and Ruud Custers. “On the Emergence of Deprivation-Reducing Behaviors: Subliminal Priming of Behavior Representations turns Deprivation into Motivation.” In: *Journal of Experimental Social Psychology* 44.3 (2008), pp. 866–873.
- [134] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. “Extracting and Composing Robust Features with Denoising Autoencoders.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2008, pp. 1096–1103.
- [135] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. “Deep Learning via Semi-Supervised Embedding.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2008, pp. 1168–1175.
- [136] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. “Curriculum Learning.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2009, pp. 41–48. ISBN: 978-1-60558-516-1. DOI: [10.1145/1553374.1553380](https://doi.org/10.1145/1553374.1553380).
- [137] J. Callan and M. Hoy. *ClueWeb09 Dataset*. <http://boston.lti.cs.cmu.edu/Data/clueweb09/>. 2009.
- [138] Olivier Chapelle, Bernhard Scholköpfung, and Alexander Zien. “Semi-Supervised Learning.” In: *IEEE Transactions on Neural Networks* 20.3 (2009), pp. 542–542.
- [139] Hal Daumé. “Bayesian Multitask Learning with Latent Hierarchies.” In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2009, pp. 135–142.

- [140] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “Imagenet: A Large-Scale Hierarchical Image Database.” In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009, pp. 248–255.
- [141] Alex Graves and Jürgen Schmidhuber. “Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2009, pp. 545–552.
- [142] Gholamreza Haffari, Maxim Roy, and Anoop Sarkar. “Active Learning for Statistical Phrase-based Machine Translation.” In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. Association for Computational Linguistics, 2009, pp. 415–423. ISBN: 978-1-932432-41-1.
- [143] Laurent Jacob, Jean-Philippe Vert, and Francis R. Bach. “Clustered Multi-Task Learning: A Convex Formulation.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2009, pp. 745–752.
- [144] Ashish Kapoor and Simon Baker. “Which Faces to Tag: Adding Prior Constraints into Active Learning.” In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Sept. 2009.
- [145] Seyoung Kim and Eric P. Xing. “Tree-Guided Group Lasso for Multi-Task Regression with Structured Sparsity.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2009.
- [146] Andreas Krause and Carlos Guestrin. “Optimal Value of Information in Graphical Models.” In: *Journal of Artificial Intelligence Research (JAIR)* 35 (2009), pp. 557–591.
- [147] Kai A. Krueger and Peter Dayan. “Flexible Shaping: How Learning in Small Steps Helps.” In: *Cognition* 110.3 (2009), pp. 380–394.
- [148] Karim Lounici, Massimiliano Pontil, Alexandre B. Tsybakov, and Sara A. van de Geer. “Taking Advantage of Sparsity in Multi-Task Learning.” In: *Proceedings of the Conference on Computational Learning Theory (COLT)*. 2009.
- [149] Kilian Q. Weinberger and Lawrence K. Saul. “Distance Metric Learning for Large Margin Nearest Neighbor Classification.” In: *Journal of Machine Learning Research (JMLR)* 10.Feb (2009), pp. 207–244.
- [150] Mustafa Bilgic, Lilyana Mihalkova, and Lise Getoor. “Active Learning for Networked Data.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2010.
- [151] Michael Bloodgood and Chris Callison-Burch. “Bucking the Trend: Large-Scale Cost-Focused Active Learning for Statistical Machine Translation.” In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2010, pp. 854–864.
- [152] Matthias Bröcheler, Lilyana Mihalkova, and Lise Getoor. “Probabilistic Similarity Logic.” In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2010, pp. 73–82.

- [153] Andrew Carlson, Burr Settles, Justin Betteridge, Bryan Kisiel, Estevam R Hruschka Jr, and Tom M. Mitchell. "Toward an Architecture for Never-Ending Language Learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2010, pp. 1–8.
- [154] Pinar Donmez, Guy Lebanon, and Krishnakumar Balasubramanian. "Unsupervised Supervised Learning I: Estimating Classification and Regression Errors without Labels." In: *Journal of Machine Learning Research (JMLR)* 11 (Apr. 2010), pp. 1323–1351.
- [155] Xavier Glorot and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks." In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2010, pp. 249–256.
- [156] Ali Jalali, Sujay Sanghavi, Chao Ruan, and Pradeep K. Ravikumar. "A Dirty Model for Multi-Task Learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2010, pp. 964–972.
- [157] John E. Laird and Robert E. Wray III. "Cognitive Architecture Requirements for Achieving AGI." In: *Conference on Artificial General Intelligence (AGI)*. Atlantis Press. 2010.
- [158] Volker Nannen. *A Short Introduction to Kolmogorov Complexity*. 2010. arXiv: [1005.2400](https://arxiv.org/abs/1005.2400) [cs.LG].
- [159] Christian Robert and George Casella. *Introducing Monte Carlo Methods with R*. Springer New York, 2010. ISBN: 978-1-4419-1582-5.
- [160] Byron C. Wallace, Kevin Small, Carla E. Brodley, and Thomas A. Trikalinos. "Active Learning for Biomedical Citation Screening." In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2010, pp. 173–182.
- [161] Peter Welinder, Steve Branson, Pietro Perona, and Serge J. Belongie. "The Multidimensional Wisdom of Crowds." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2010, pp. 2424–2432.
- [162] Yi Zhang. "Multi-Task Active Learning with Output Constraints." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2010.
- [163] Yu Zhang and Dit-Yan Yeung. "A Convex Formulation for Learning Task Relationships in Multi-Task Learning." In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2010.
- [164] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers." In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122.
- [165] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." In: *Journal of Machine Learning Research (JMLR)* 12 (2011), pp. 2121–2159.

- [166] Patricia Gervan, Andrea Berencsi, and Ilona Kovacs. “Vision First? The Development of Primary Visual Cortical Networks is More Rapid than the Development of Primary Motor Networks in Humans.” In: *PloS one* 6.9 (2011).
- [167] Daniel Kahneman and Patrick Egan. *Thinking, Fast and Slow*. Vol. 1. Farrar, Straus and Giroux New York, 2011.
- [168] Zhuoliang Kang, Kristen Grauman, and Fei Sha. “Learning with Whom to Share in Multi-Task Feature Learning.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2011, pp. 521–528.
- [169] Ni Lao, Tom M. Mitchell, and William W. Cohen. “Random Walk Inference and Learning in a Large Scale Knowledge Base.” In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics. 2011, pp. 529–539.
- [170] J. Martens and Ilya Sutskever. “Training Recurrent Neural Networks with Hessian-Free Optimization.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2011.
- [171] Sahand N. Negahban and Martin J. Wainwright. “Simultaneous Support Recovery in High Dimensions: Benefits and Perils of Block ℓ_1/ℓ_∞ -Regularization.” In: *IEEE Transactions on Information Theory (TIT)* 57.6 (2011), pp. 3841–3863.
- [172] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. “Tuffy: Scaling Up Statistical Inference in Markov Logic Networks Using an RDBMS.” In: *Proceedings of the VLDB Endowment (PVLDB)* 4.6 (Mar. 2011), pp. 373–384. ISSN: 2150-8097. DOI: [10.14778/1978665.1978669](https://doi.org/10.14778/1978665.1978669).
- [173] Avishek Saha, Piyush Rai, Hal Daumé, and Suresh Venkatasubramanian. “On-line Learning of Multiple Tasks and Their Relationships.” In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2011.
- [174] Ilya Sutskever, James Martens, and Geoffrey E. Hinton. “Generating Text with Recurrent Neural Networks.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2011, pp. 1017–1024.
- [175] Sam S. Adams, Itamar Arel, Joscha Bach, Robert Coop, Rod Furlan, Ben Goertzel, J. Storrs Hall, Alexei V. Samsonovich, Matthias Scheutz, Matthew Schlesinger, Stuart C. Shapiro, and John F. Sowa. “Mapping the Landscape of Human-Level Artificial General Intelligence.” In: *AI Magazine* 33.1 (2012), pp. 25–42.
- [176] Vamshi Ambati. “Active Learning and Crowdsourcing for Machine Translation in Low Resource Scenarios.” AAI3528171. PhD thesis. 2012. ISBN: 978-1-267-58215-7.
- [177] J. Callan. *ClueWeb12 Dataset*. <http://lemurproject.org/clueweb12/>. 2012.
- [178] Matteo Carandini and David J. Heeger. “Normalization as a Canonical Neural Computation.” In: *Nature Reviews Neuroscience* 13.1 (2012), p. 51.
- [179] Abhay Harpale. “Multi-Task Active Learning.” PhD thesis. 2012.

- [180] Georg B. Keller, Tobias Bonhoeffer, and Mark Hübener. “Sensorimotor Mismatch Signals in Primary Visual Cortex of the Behaving Mouse.” In: *Neuron* 74.5 (2012), pp. 809–815.
- [181] Abhishek Kumar and Hal Daumé. “Learning Task Grouping and Overlap in Multi-Task Learning.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2012.
- [182] John E. Laird. *The Soar Cognitive Architecture*. MIT Press, 2012.
- [183] Qiang Liu, Jian Peng, and Alexander T. Ihler. “Variational Inference for Crowdsourcing.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2012, pp. 692–700.
- [184] Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan Cernocky. “Subword Language Modeling with Neural Networks.” In: *Preprint 8* (2012). URL: <http://www.fit.vutbr.cz/imikolov/rnnlm/char.pdf>.
- [185] Jean-Baptiste Mouret and Stéphane Doncieux. “Encouraging Behavioral Diversity in Evolutionary Robotics: An Empirical Study.” In: *Evolutionary Computation* 20.1 (2012), pp. 91–133.
- [186] B. Murphy, P. Talukdar, and Tom M. Mitchell. “Learning Effective and Interpretable Semantic Models using Non-Negative Sparse Embedding.” In: *Proceedings of the International Conference on Computational Linguistics (COLING)*. 2012.
- [187] Kevin P. Murphy. “Undirected Graphical Models (Markov Random Fields).” In: *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning. MIT Press, 2012. Chap. 19. ISBN: 978-0-262-01802-9.
- [188] Galileo Namata, Ben London, Lise Getoor, and Bert Huang. “Query-Driven Active Surveying for Collective Classification.” In: *10th International Workshop on Mining and Learning with Graphs*. 2012.
- [189] Emmanouil Antonios Platanios. *How to Grade a Test Without Knowing the Answers—A Bayesian Graphical Model for Adaptive Crowdsourcing and Aptitude Testing*. 2012. arXiv: [1206.6386](https://arxiv.org/abs/1206.6386) [cs.LG].
- [190] Charan Ranganath and Maureen Ritchey. “Two Cortical Systems for Memory-Guided Behaviour.” In: *Nature Reviews Neuroscience* 13.10 (2012), p. 713.
- [191] J.K. Rowling. *Harry Potter and the Sorcerer’s Stone*. Harry Potter US. Pottermore Limited, 2012. ISBN: 9781781100271.
- [192] Burr Settles. *Active Learning*. Vol. 6. Morgan & Claypool Publishers, June 2012.
- [193] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A Physics Engine for Model-Based Control.” In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2012, pp. 5026–5033.
- [194] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. “Deep Learning via Semi-Supervised Embedding.” In: *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 639–655.

- [195] Stephen H. Bach, Bert Huang, Ben London, and Lise Getoor. “Hinge-Loss Markov Random Fields: Convex Inference for Structured Prediction.” In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2013.
- [196] Maria-Florina Balcan, Avrim Blum, and Yishay Mansour. “Exploiting Ontology Structures and Unlabeled Data for Learning.” In: *Proceedings of the International Conference on Machine Learning (ICML)* (2013), pp. 1112–1120.
- [197] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. “The Arcade Learning Environment: An Evaluation Platform for General Agents.” In: *Journal of Artificial Intelligence Research (JAIR)* 47 (2013), pp. 253–279.
- [198] Yoshua Bengio, Aaron Courville, and Pascal Vincent. “Representation Learning: A Review and New Perspectives.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 35.8 (2013), pp. 1798–1828.
- [199] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. “Translating Embeddings for Modeling Multi-Relational Data.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2013, pp. 2787–2795.
- [200] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. “An Overview of Recent Progress in the Study of Distributed Multi-Agent Coordination.” In: *IEEE Transactions on Industrial Informatics* 9.1 (2013), pp. 427–438.
- [201] Li Deng, Geoffrey Hinton, and Brian Kingsbury. “New Types of Deep Neural Network Learning for Speech Recognition and Related Applications: An Overview.” In: *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 8599–8603.
- [202] Matt Gardner, Partha Pratim Talukdar, Bryan Kisiel, and Tom M. Mitchell. “Improving Learning and Inference in a Large Knowledge-Base Using Latent Syntactic Cues.” In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2013, pp. 833–838.
- [203] Nal Kalchbrenner and Phil Blunsom. “Recurrent Continuous Translation Models.” In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2013, pp. 1700–1709.
- [204] Wenjie Luo, Alex Schwing, and Raquel Urtasun. “Latent Structured Active Learning.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2013, pp. 728–736.
- [205] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: [1312.5602](https://arxiv.org/abs/1312.5602) [cs.LG].
- [206] Jay Pujara, Hui Miao, Lise Getoor, and William W. Cohen. “Knowledge Graph Identification.” In: vol. 8218. Chapter 34. 2013, pp. 542–557.
- [207] Paul Ruvolo and Eric Eaton. “Active Task Selection for Lifelong Machine Learning.” In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2013.

- [208] Manali Sharma and Mustafa Bilgic. “Most-Surely vs. Least-Surely Uncertain.” In: *IEEE International Conference on Data Mining (ICDM)*. 2013.
- [209] Will Y. Zou, Richard Socher, Daniel Cer, and Christopher D. Manning. “Bilingual Word Embeddings for Phrase-Based Machine Translation.” In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2013, pp. 1393–1398.
- [210] Gabor Angeli, Julie Tibshirani, Jean Y. Wu, and Christopher D. Manning. “Combining Distant and Partial Supervision for Relation Extraction.” In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1556–1567.
- [211] John A. Bargh and Tanya L. Chartrand. “The Mind in the Middle: A Practical Guide to Priming and Automaticity Research.” In: (2014).
- [212] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. “Spectral Networks and Locally Connected Networks on Graphs.” In: *International Conference on Learning Representations (ICLR)*. 2014.
- [213] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. *Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: [1406.1078 \[cs.CL\]](#).
- [214] John Collins and Minh Huynh. “Estimation of Diagnostic Test Accuracy Without Full Verification: A Review of Latent Class Methods.” In: *Statistics in Medicine* 33.24 (June 2014), pp. 4141–4169.
- [215] Jan Koutník, Klaus Greff, Faustino Gomez, and Jürgen Schmidhuber. *A Clockwork RNN*. 2014. arXiv: [1402.3511 \[cs.NE\]](#).
- [216] Fabio Parisi, Francesco Strino, Boaz Nadler, and Yuval Kluger. “Ranking and combining multiple predictors without labeled data.” In: *Proceedings of the National Academy of Sciences* (Jan. 2014), pp. 1–28.
- [217] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “DeepWalk: Online Learning of Social Representations.” In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2014, pp. 701–710.
- [218] Emmanouil Antonios Platanios, Avrim Blum, and Tom M. Mitchell. “Estimating Accuracy from Unlabeled Data.” In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2014.
- [219] Yashoteja Prabhu and Manik Varma. “FastXML: A Fast, Accurate and Stable Tree-Classifer for Extreme Multi-Label Learning.” In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM. 2014, pp. 263–272.
- [220] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2014, pp. 3104–3112.

- [221] Leila Wehbe, Brian Murphy, Partha Talukdar, Alona Fyshe, Aaditya Ramdas, and Tom M. Mitchell. “Simultaneously Uncovering the Patterns of Brain Regions Involved in Different Story Reading Subprocesses.” In: *PLOS ONE* 9.11 (Nov. 2014). Ed. by Kevin Paterson, e112575. ISSN: 1932-6203. DOI: [10.1371/journal.pone.0112575](https://doi.org/10.1371/journal.pone.0112575). URL: <http://dx.plos.org/10.1371/journal.pone.0112575>.
- [222] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. “Knowledge Base Completion via Search-Based Question Answering.” In: *Proceedings of the International Conference on World Wide Web (WWW)*. 2014, pp. 515–526.
- [223] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. “How Transferable are Features in Deep Neural Networks?” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2014, pp. 3320–3328.
- [224] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate.” In: *International Conference on Learning Representations (ICLR)*. 2015.
- [225] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. “Multi-Task Learning for Multiple Language Translation.” In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, pp. 1723–1732.
- [226] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. “Multi-Task Learning for Multiple Language Translation.” In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, pp. 1723–1732. URL: <https://www.aclweb.org/anthology/P/P15/P15-1166.pdf>.
- [227] Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. “Low Resource Dependency Parsing: Cross-Lingual Parameter Sharing in a Neural Network Parser.” In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. 2015, pp. 845–850.
- [228] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. “Convolutional Networks on Graphs for Learning Molecular Fingerprints.” In: (2015), pp. 2224–2232.
- [229] Joann G. Elmore, Gary M. Longton, Patricia A. Carney, Berta M. Geller, Tracy Onega, Anna N. A. Tosteson, Heidi D. Nelson, Margaret S. Pepe, Kimberly H. Allison, Stuart J. Schnitt, Frances P. O’Malley, and Donald L. Weaver. “Diagnostic Concordance among Pathologists Interpreting Breast Biopsy Specimens.” In: *Journal of the American Medical Association (JAMA)* 313.11 (2015), pp. 1122–1132.

- [230] Ross Girshick. “Fast R-CNN.” In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448.
- [231] Kelvin Guu, John Miller, and Percy Liang. “Traversing Knowledge Graphs in Vector Space.” In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2015, pp. 318–327.
- [232] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. “Knowledge Graph Embedding via Dynamic Mapping Matrix.” In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, pp. 687–696.
- [233] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. “An Empirical Exploration of Recurrent Network Architectures.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2015, pp. 2342–2350.
- [234] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: *International Conference for Learning Representations (ICLR)*. 2015.
- [235] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep Learning.” In: *Nature* 521.7553 (2015), p. 436.
- [236] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. “Learning Entity and Relation Embeddings for Knowledge Graph Completion.” In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2015, pp. 2181–2187. ISBN: 0-262-51129-0.
- [237] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. “Effective Approaches to Attention-based Neural Machine Translation.” In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2015, pp. 1412–1421.
- [238] Tom M. Mitchell, William W. Cohen, Estevam R. Hruschka Jr, Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhanava Dalvi, Matt Gardner, Bryan Kiesel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Tahir P Mohamed, Ndapakula Nakashole, Emmanouil Antonios Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. “Never-Ending Learning.” In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2015.
- [239] Pablo G Moreno, Antonio Artés-Rodríguez, Yee Whye Teh, and Fernando Perez-Cruz. “Bayesian Nonparametric Crowdsourcing.” In: *Journal of Machine Learning Research (JMLR)* 16 (Aug. 2015), pp. 1–21.
- [240] Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. “Compositional Vector Space Models for Knowledge Base Completion.” In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, July 2015, pp. 156–

166. DOI: [10.3115/v1/P15-1016](https://doi.org/10.3115/v1/P15-1016). URL: <https://www.aclweb.org/anthology/P15-1016>.
- [241] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. *Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning*. 2015. arXiv: [1511.06342](https://arxiv.org/abs/1511.06342) [cs.LG].
- [242] Bharath Ramsundar, Steven Kearnes, Patrick Riley, Dale Webster, David Konerding, and Vijay Pande. *Massively Multitask Networks for Drug Discovery*. 2015. arXiv: [1502.02072](https://arxiv.org/abs/1502.02072) [stat.ML].
- [243] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” In: *International Conference on Learning Representations (ICLR)*. 2015.
- [244] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. “End-to-End Memory Networks.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2015, pp. 2440–2448.
- [245] Tian Tian and Jun Zhu. “Max-Margin Majority Voting for Learning from Crowds.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2015.
- [246] Kristina Toutanova and Danqi Chen. “Observed versus latent features for knowledge base and text inference.” In: *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*. 2015, pp. 57–66.
- [247] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. “Embedding Entities and Relations for Learning and Inference in Knowledge Bases.” In: *International Conference on Learning Representations (ICLR)*. 2015.
- [248] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. *Depth-Gated LSTM*. 2015. arXiv: [1508.03790](https://arxiv.org/abs/1508.03790) [cs.NE].
- [249] Shiquan Zhao, Jian Wu, Victor S. Sheng, Chen Ye, PengPeng Zhao, and Zhiming Cui. “Weak Labeled Multi-Label Active Learning for Image Classification.” In: *ACM International Conference on Multimedia*. 2015, pp. 1127–1130.
- [250] Dengyong Zhou, Qiang Liu, John C. Platt, Christopher Meek, and Nihar B. Shah. *Regularized Minimax Conditional Entropy for Crowdsourcing*. 2015. arXiv: [1503.07240](https://arxiv.org/abs/1503.07240) [cs.LG].
- [251] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. “Neural Module Networks.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 39–48.
- [252] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. *DeepMind Lab*. 2016. arXiv: [1612.03801](https://arxiv.org/abs/1612.03801) [cs.AI].
- [253] Nadav Bhonker, Shai Rozenberg, and Itay Hubara. *Playing SNES in the Retro Learning Environment*. 2016. arXiv: [1611.02205](https://arxiv.org/abs/1611.02205) [cs.LG].

- [254] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. arXiv: [1606.01540 \[cs.LG\]](#).
- [255] Ozan Caglayan, Walid Aransa, Yaxing Wang, Marc Masana, Mercedes García-Martínez, Fethi Bougares, Loïc Barrault, and Joost van de Weijer. “Does Multi-modality Help Human and Machine for Translation and Image Captioning?” In: *Proceedings of the First Conference on Machine Translation*. Vol. 2. 2016.
- [256] Yong Cheng, Yang Liu, Qian Yang, Maosong Sun, and Wei Xu. *Neural Machine Translation with Pivot Languages*. 2016. arXiv: [1611.04928 \[cs.CL\]](#).
- [257] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. *Recurrent Batch Normalization*. 2016. arXiv: [1603.09025 \[cs.LG\]](#).
- [258] Josep Maria Crego, Jungi Kim, Guillaume Klein, Anabel Rebollo, Kathy Yang, Jean Senellart, Egor Akhanov, Patrice Brunelle, Aurelien Coquard, Yongchao Deng, Satoshi Enoue, Chiyo Geiss, Joshua Johanson, Ardas Khalsa, Raoum Khiari, Byeongil Ko, Catherine Kobus, Jean Lorieux, Leidiana Martins, Dang-Chuan Nguyen, Alexandra Priori, Thomas Riccardi, Natalia Segal, Christophe Servan, Cyril Tiquet, Bo Wang, Jin Yang, Dakun Zhang, Jing Zhou, and Peter Zoldan. *SYSTRAN’s Pure Neural Machine Translation Systems*. 2016. arXiv: [1610.05540 \[cs.CL\]](#).
- [259] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016, pp. 3844–3852.
- [260] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. “Benchmarking Deep Reinforcement Learning for Continuous Control.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2016, pp. 1329–1338.
- [261] Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. “Multi-Way, Multilingual Neural Machine Translation with a Shared Attention Mechanism.” In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. 2016, pp. 866–875. URL: <http://www.aclweb.org/anthology/N16-1101>.
- [262] Orhan Firat, Baskaran Sankaran, Yaser Al-Onaizan, Fatos T. Yarman-Vural, and Kyunghyun Cho. “Zero-Resource Translation with Multi-Lingual Neural Machine Translation.” In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2016.
- [263] Alex Graves. *Adaptive Computation Time for Recurrent Neural Networks*. 2016. arXiv: [1603.08983 \[cs.NE\]](#).
- [264] Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. “LSTM: A Search Space Odyssey.” In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (2016), pp. 2222–2232.

- [265] Varun Gulshan, Lily Peng, Marc Coram, Martin C. Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, Ramasamy Kim, Rajiv Raman, Philip C. Nelson, Jessica L. Mega, and Dale R. Webster. “Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Translation.” In: *Journal of the American Medical Association (JAMA)* 316.22 (Dec. 2016), pp. 2402–2410. ISSN: 0098-7484.
- [266] Thanh-Le Ha, Jan Niehues, and Alexander Waibel. “Toward Multilingual Neural Machine Translation with Universal Encoder and Decoder.” In: *Proceedings of the 13th International Workshop on Spoken Language Translation (IWSLT)*. 2016. URL: https://workshop2016.iwslt.org/downloads/IWSLT_2016_paper_5.pdf.
- [267] Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. “A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks.” In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2016.
- [268] Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tieyan Liu, and Wei-Ying Ma. “Dual Learning for Machine Translation.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016, pp. 820–828.
- [269] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition.” In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.
- [270] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2016. arXiv: [1606.08415 \[cs.LG\]](#).
- [271] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. “The Malmo Platform for Artificial Intelligence Experimentation.” In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2016, pp. 4246–4247.
- [272] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks.” In: *International Conference on Learning Representations (ICLR)*. 2016.
- [273] Adriana Kovashka, Olga Russakovsky, Li Fei-Fei, and Kristen Grauman. “Crowdsourcing in Computer Vision.” In: *Foundations and Trends® in Computer Graphics and Vision* 10.3 (2016), pp. 177–243.
- [274] Ben Krause, Liang Lu, Iain Murray, and Steve Renals. *Multiplicative LSTM for Sequence Modeling*. 2016. arXiv: [1609.07959 \[cs.NE\]](#).
- [275] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. “Large-Margin Softmax Loss for Convolutional Neural Networks.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. Vol. 48. 2016, pp. 507–516.

- [276] Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. “Multi-task Sequence to Sequence Learning.” In: *International Conference on Learning Representations (ICLR)*. 2016. URL: <https://arxiv.org/abs/1511.06114>.
- [277] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. “Asynchronous Methods for Deep Reinforcement Learning.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2016, pp. 1928–1937.
- [278] Esther K. Papies. “Health Goal Priming as a Situated Intervention Tool: How to Benefit from Nonconscious Motivational Routes to Health Behaviour.” In: *Health Psychology Review* 10.4 (2016), pp. 408–424.
- [279] Emmanouil Antonios Platanios, Avinava Dubey, and Tom M. Mitchell. “Estimating Accuracy from Unlabeled Data: A Bayesian Approach.” In: *International Conference in Machine Learning (ICML)*. 2016, pp. 1416–1425.
- [280] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Improving Neural Machine Translation Models with Monolingual Data.” In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2016, pp. 86–96.
- [281] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Neural Machine Translation of Rare Words with Subword Units.” In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*. 2016, pp. 1715–1725.
- [282] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. “Learning Multiagent Communication with Backpropagation.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016, pp. 2244–2252.
- [283] Kristina Toutanova, Victoria Lin, Wen-tau Yih, Hoifung Poon, and Chris Quirk. “Compositional Learning of Embeddings for Relation Paths in Knowledge Base and Text.” In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL Volume 1: Long Papers)*. Association for Computational Linguistics, 2016, pp. 1434–1444. DOI: [10.18653/v1/P16-1136](https://doi.org/10.18653/v1/P16-1136). URL: <https://www.aclweb.org/anthology/P16-1136>.
- [284] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. “Complex embeddings for simple link prediction.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. Vol. 48. 2016, pp. 2071–2080.
- [285] Evan Weingarten, Qijia Chen, Maxwell McAdams, Jessica Yi, Justin Hepler, and Dolores Albarracín. “From Primed Concepts to Action: A Meta-Analysis of the Behavioral Effects of Incidentally Presented Words.” In: *Psychological Bulletin* 142.5 (2016), p. 472.

- [286] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. *Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation*. 2016. arXiv: [1609.08144](#) [cs.CL].
- [287] Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan R. Salakhutdinov. "On Multiplicative Integration with Recurrent Neural Networks." In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016, pp. 2856–2864.
- [288] Yongxin Yang and Timothy M. Hospedales. *Trace Norm Regularised Deep Multi-Task Learning*. 2016. arXiv: [1606.04038](#) [cs.LG].
- [289] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. *Planetoid Github Repository*. <https://github.com/kimiyoung/planetoid>. Accessed: 2020-01-28. 2016.
- [290] Dakun Zhang, Jungi Kim, Josep Crego, and Jean Senellart. *Boosting Neural Machine Translation*. 2016. arXiv: [1612.06138](#) [cs.CL].
- [291] Yao Zhou and Jingrui He. "Crowdsourcing via Tensor Augmentation and Completion." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2016, pp. 2435–2441.
- [292] Barret Zoph and Kevin Knight. "Multi-Source Neural Translation." In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. 2016, pp. 30–34.
- [293] Jacob Andreas, Anca Dragan, and Dan Klein. "Translating Neuralese." In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL Volume 1: Long Papers)*. Vol. 1. 2017, pp. 232–242.
- [294] Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. "Hinge-Loss Markov Random Fields and Probabilistic Soft Logic." In: *Journal of Machine Learning Research (JMLR)* 18.1 (Jan. 2017), pp. 3846–3912. ISSN: 1532-4435.
- [295] Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Raphael Rubino, Lucia Specia, and Marco Turchi. "Findings of the 2017 Conference on Machine Translation (WMT17)." In: *Proceedings of the Second Conference on Machine Translation*. 2017, pp. 169–214.

- [296] Mauro Cettolo, Marcello Federico, Luisa Bentivogli, Jan Niehues, Sebastian Stüker, Katsuhito Sudoh, Koichi Yoshino, and Christian Federmann. “Overview of the IWSLT 2017 Evaluation Campaign.” In: *Proceedings of the 14th International Workshop on Spoken Language Translation (IWSLT)*. 2017.
- [297] Yun Chen, Yang Liu, Yong Cheng, and Victor O. K. Li. “A Teacher-Student Framework for Zero-Resource Neural Machine Translation.” In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL Volume 1: Long Papers)*. 2017, pp. 1925–1935.
- [298] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. “Dermatologist-level classification of skin cancer with deep neural networks.” In: *Nature* 542.7639 (2017), p. 115.
- [299] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. Vol. 70. 2017, pp. 1126–1135.
- [300] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. “Neural Message Passing for Quantum Chemistry.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2017, pp. 1263–1272.
- [301] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 1024–1034.
- [302] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. *Learning to Reason: End-to-End Module Networks for Visual Question Answering*. 2017. arXiv: [1704.05526](https://arxiv.org/abs/1704.05526) [cs.CV].
- [303] Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. “Google’s Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation.” In: *Transactions of the Association for Computational Linguistics (TACL)*. Vol. 5. 2017, pp. 339–351. URL: <https://www.aclweb.org/anthology/Q/Q17/Q17-1024.pdf>.
- [304] Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. *One Model to Learn them All*. 2017. arXiv: [1706.05137](https://arxiv.org/abs/1706.05137) [cs.LG].
- [305] Tom Kocmi and Ondřej Bojar. “Curriculum Learning and Minibatch Bucketing in Neural Machine Translation.” In: *Proceedings of the International Conference Recent Advances in Natural Language Processing (RANLP)*. 2017, pp. 379–386. DOI: [10.26615/978-954-452-049-6_050](https://doi.org/10.26615/978-954-452-049-6_050).
- [306] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. “Building Machines that Learn and Think Like People.” In: *Behavioral and Brain Sciences* 40 (2017). DOI: [10.1017/S0140525X16001837](https://doi.org/10.1017/S0140525X16001837).

- [307] Surafel M. Lakew, Quintino F. Lotito, Negri Matteo, Turchi Marco, and Federico Marcello. “Improving Zero-Shot Translation of Low-Resource Languages.” In: *Proceedings of the International Workshop on Spoken Language Translation (IWSLT)*. 2017, pp. 113–119.
- [308] Jason Lee, Kyunghyun Cho, and Thomas Hofmann. “Fully Character-Level Neural Machine Translation without Explicit Segmentation.” In: 5 (2017), pp. 365–378.
- [309] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M Bronstein. “Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs.” In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 5425–5434.
- [310] Emmanouil Antonios Platanios, Ashish Kapoor, and Eric Horvitz. *Active Learning amidst Logical Knowledge*. 2017. arXiv: [1709.08850 \[cs.AI\]](#).
- [311] Emmanouil Antonios Platanios, Hoifung Poon, Eric Horvitz, and Tom M. Mitchell. “Estimating Accuracy from Unlabeled Data: A Probabilistic Logic Approach.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.
- [312] Matthew A. Lambon Ralph, Elizabeth Jefferies, Karalyn Patterson, and Timothy T. Rogers. “The Neural and Computational Bases of Semantic Cognition.” In: *Nature Reviews Neuroscience* 18.1 (2017), p. 42.
- [313] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. “Snorkel: Rapid Training Data Creation with Weak Supervision.” In: *Proceedings of the VLDB Endowment (PVLDB)* 11.3 (2017), pp. 269–282.
- [314] Tim Rocktäschel and Sebastian Riedel. “End-to-End Differentiable Proving.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 3788–3800.
- [315] Sebastian Ruder. *An Overview of Multi-Task Learning in Deep Neural Networks*. 2017. arXiv: [1706.05098 \[cs.LG\]](#).
- [316] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: [1707.06347 \[cs.LG\]](#).
- [317] Maruan Al-Shedivat, Avinava Dubey, and Eric P. Xing. *Contextual Explanation Networks*. 2017. arXiv: [1705.10301 \[cs.LG\]](#).
- [318] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. “Mastering the Game of Go without Human Knowledge.” In: *Nature* 550.7676 (2017), pp. 354–359. DOI: [10.1038/nature24270](#). URL: <https://doi.org/10.1038/nature24270>.

- [319] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S. Talwalkar. “Federated Multi-Task Learning.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 4424–4434.
- [320] Jake Snell, Kevin Swersky, and Richard Zemel. “Prototypical Networks for Few-Shot Learning.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 4077–4087.
- [321] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. “Attention is All you Need.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 5998–6008.
- [322] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. “Sample Efficient Actor-Critic with Experience Replay.” In: *International Conference on Learning Representations (ICLR)*. 2017.
- [323] Ziyu Wang, Josh S. Merel, Scott E. Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. “Robust Imitation of Diverse Behaviors.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 5320–5329.
- [324] Qizhe Xie, Xuezhe Ma, Zihang Dai, and Eduard Hovy. “An Interpretable Knowledge Transfer Model for Knowledge Base Completion.” In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 950–962. DOI: [10.18653/v1/P17-1088](https://doi.org/10.18653/v1/P17-1088). URL: <https://www.aclweb.org/anthology/P17-1088>.
- [325] Wenhan Xiong, Thien Hoang, and William Y. Wang. “DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning.” In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2017, pp. 564–573. DOI: [10.18653/v1/D17-1060](https://doi.org/10.18653/v1/D17-1060).
- [326] Fan Yang, Zhilin Yang, and William W. Cohen. “Differentiable Learning of Logical Rules for Knowledge Base Reasoning.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017, pp. 2319–2328.
- [327] Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. “Unsupervised Neural Machine Translation.” In: *International Conference on Learning Representations (ICLR)*. 2018.
- [328] Rachel Bawden, Rico Sennrich, Alexandra Birch, and Barry Haddow. “Evaluating Discourse Phenomena in Neural Machine Translation.” In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT Volume 1: Long Papers)*. Association for Computational Linguistics, June 2018, pp. 1304–1313.

- [329] Thang D. Bui, Sujith Ravi, and Vivek Ramavajjala. “Neural Graph Learning: Training Neural Networks Using Graphs.” In: *ACM International Conference on Web Search and Data Mining (WSDM)*. ACM. 2018, pp. 64–71.
- [330] Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. “The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation.” In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL Volume 1: Long Papers)*. Association for Computational Linguistics, 2018, pp. 76–86.
- [331] Zhiyuan Chen and Bing Liu. *Lifelong Machine Learning, Second Edition*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2018. DOI: [10.2200/S00832ED1V01Y201802AIM037](https://doi.org/10.2200/S00832ED1V01Y201802AIM037).
- [332] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. *BabyAI: First Steps Towards Grounded Language Learning With a Human In the Loop*. 2018. arXiv: [1810.08272](https://arxiv.org/abs/1810.08272) [cs.AI].
- [333] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durgkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. “Go for a Walk and Arrive at the Answer: Reasoning over Paths in Knowledge Bases Using Reinforcement Learning.” In: *International Conference on Learning Representations (ICLR)*. 2018.
- [334] Tim Dettmers, Minervini Pasquale, Stenetorp Pontus, and Sebastian Riedel. “Convolutional 2D Knowledge Graph Embeddings.” In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. Feb. 2018, pp. 1811–1818.
- [335] Anca Dumitrache, Lora Aroyo, and Chris Welty. “Crowdsourcing Ground Truth for Medical Relation Extraction.” In: *ACM Transactions on Interactive Intelligent Systems (TIIS)* 8.2 (2018), p. 12.
- [336] Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. “Feature-wise Transformations.” In: *Distill* (2018). <https://distill.pub/2018/feature-wise-transformations>. DOI: [10.23915/distill.00011](https://doi.org/10.23915/distill.00011).
- [337] Luca Franceschi, Paolo Frasconi, Saverio Salzo, and Massimiliano Pontil. *Bilevel Programming for Hyperparameter Optimization and Meta-Learning*. 2018. arXiv: [1806.04910](https://arxiv.org/abs/1806.04910) [stat.ML].
- [338] Aditya Grover, Maruan Al-Shedivat, Jayesh K. Gupta, Yura Burda, and Harrison Edwards. *Learning Policy Representations in Multiagent Systems*. 2018. arXiv: [1806.06464](https://arxiv.org/abs/1806.06464) [cs.MA].

- [339] Jiatao Gu, Hany Hassan, Jacob Devlin, and Victor O. K. Li. “Universal Neural Machine Translation for Extremely Low Resource Languages.” In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. 2018, pp. 344–354.
- [340] David Ha, Andrew Dai, and Quoc V. Le. “HyperNetworks.” In: *International Conference on Learning Representations (ICLR)*. 2018. URL: <https://openreview.net/forum?id=rkpACe1lx>.
- [341] David Ha and Jürgen Schmidhuber. *World Models*. 2018. arXiv: [1803.10122 \[cs.LG\]](#).
- [342] Po-Sen Huang, Chong Wang, Sitao Huang, Dengyong Zhou, and Li Deng. “Towards Neural Phrase-Based Machine Translation.” In: *International Conference on Learning Representations (ICLR)*. 2018.
- [343] Alex Kendall, Yarin Gal, and Roberto Cipolla. “Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 7482–7491.
- [344] Ashish Khetan, Zachary C. Lipton, and Anima Anandkumar. “Learning from Noisy Singly-Labeled Data.” In: (2018).
- [345] Guillaume Lample, Ludovic Denoyer, and Marc’Aurelio Ranzato. “Unsupervised Machine Translation Using Monolingual Corpora Only.” In: *International Conference on Learning Representations (ICLR)*. 2018.
- [346] Ben Lengerich, Andrew Maas, and Christopher Potts. “Retrofitting Distributional Embeddings to Knowledge Graphs with Functional Relations.” In: *Proceedings of the International Conference on Computational Linguistics*. Association for Computational Linguistics, Aug. 2018, pp. 2423–2436.
- [347] Xi Victoria Lin, Richard Socher, and Caiming Xiong. “Multi-Hop Knowledge Graph Reasoning with Reward Shaping.” In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2018, pp. 3243–3253.
- [348] Frederick Liu, Han Lu, and Graham Neubig. “Handling Homographs in Neural Machine Translation.” In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. Association for Computational Linguistics, June 2018, pp. 1336–1345.
- [349] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. *The Natural Language Decathlon: Multitask Learning as Question Answering*. 2018. arXiv: [1806.08730 \[cs.CL\]](#).
- [350] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2018. arXiv: [1802.03426 \[stat.ML\]](#).

- [351] Paul Michel and Graham Neubig. “Extreme Adaptation for Personalized Neural Machine Translation.” In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL Short Papers)*. 2018, pp. 312–318.
- [352] Tom M. Mitchell, William W. Cohen, Estevam R. Hruschka Jr, Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhanava Dalvi, Matt Gardner, Bryan Kiesel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir P. Mohamed, Nandakumar Nakashole, Emmanouil Antonios Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. “Never-Ending Learning.” In: *Communications of the Association for Computing Machinery (CACM)* 61.5 (2018), pp. 103–115.
- [353] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. “Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2018).
- [354] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. *Deep Contextualized Word Representations*. 2018. arXiv: [1802.05365](https://arxiv.org/abs/1802.05365) [cs.CL].
- [355] Vicki Pfau, Alex Nichol, Christopher Hesse, Larissa Schiavo, John Schulman, and Oleg Klimov. *OpenAI Gym Retro*. <https://openai.com/blog/gym-retro/>. 2018.
- [356] Emmanouil Antonios Platanios. *Agreement-based Learning*. 2018. arXiv: [1806.01258](https://arxiv.org/abs/1806.01258) [cs.LG].
- [357] Emmanouil Antonios Platanios. *Symphony MT*. <https://github.com/eaplatanios/symphony-mt>. 2018.
- [358] Emmanouil Antonios Platanios. *TensorFlow Scala*. https://github.com/eaplatanios/tensorflow_scala. 2018.
- [359] Emmanouil Antonios Platanios, Mrinmaya Sachan, Graham Neubig, and Tom M. Mitchell. “Contextual Parameter Generation for Universal Neural Machine Translation.” In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2018.
- [360] Martin Popel and Ondřej Bojar. “Training Tips for the Transformer Model.” In: *The Prague Bulletin of Mathematical Linguistics* 110.1 (2018), pp. 43–70.
- [361] Alex Ratner, Braden Hancock, Jared Dunnmon, Roger Goldman, and Christopher Ré. “Snorkel MeTaL: Weak Supervision for Multi-Task Learning.” In: *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning* (2018), p. 3.
- [362] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. “On the Convergence of Adam and Beyond.” In: *International Conference on Learning Representations (ICLR)*. 2018.

- [363] Sumudu Samarakoon, Mehdi Bennis, Walid Saady, and Merouane Debbah. *Distributed Federated Learning for Ultra-Reliable Low-Latency Vehicular Communications*. 2018. arXiv: [1807.08127 \[cs.IT\]](#).
- [364] Noam Shazeer and Mitchell Stern. “Adafactor: Adaptive Learning Rates with Sublinear Memory Cost.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 4596–4604.
- [365] Yosef Singer, Yayoi Teramoto, Ben D. B. Willmore, Jan W. H. Schnupp, Andrew J. King, and Nicol S. Harper. “Sensory Cortex is Optimized for Prediction of Future Input.” In: *eLife* 7 (2018), e31557.
- [366] Yudai Takarada and Daichi Nozaki. “Motivational Goal-Priming With or Without Awareness Produces Faster and Stronger Force Exertion.” In: *Scientific Reports* 8.1 (2018), p. 10135.
- [367] Kiran K. Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. *Attention-based Graph Neural Network for Semi-Supervised Learning*. 2018. arXiv: [1803.03735 \[stat.ML\]](#).
- [368] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. “Graph Attention Networks.” In: *International Conference on Learning Representations (ICLR)* (2018).
- [369] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M. Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Yuhuai Wu, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. <https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/>. 2018.
- [370] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. “Graph Convolutional Neural Networks for Web-Scale Recommender Systems.” In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2018, pp. 974–983.
- [371] Xuan Zhang, Gaurav Kumar, Huda Khayrallah, Kenton Murray, Jeremy Gwinup, Marianna J. Martindale, Paul McNamee, Kevin Duh, and Marine Carpuat. *An Empirical Exploration of Curriculum Learning for Neural Machine Translation*. 2018. arXiv: [1811.00739 \[cs.CL\]](#).
- [372] Stephen H. Bach, Daniel Rodriguez, Yintao Liu, Chong Luo, Haidong Shao, Cassandra Xia, Souvik Sen, Alex Ratner, Braden Hancock, Houman Alborzi, Rahul Kuchhal, Chris Ré, and Rob Malkin. “Snorkel Drybell: A Case Study in

- Deploying Weak Supervision at Industrial Scale.” In: *International Conference on Management of Data (SIGMOD)*. ACM. 2019, pp. 362–375.
- [373] Ines Chami, Rex Ying, Christopher Ré, and Jure Leskovec. “Hyperbolic Graph Convolutional Neural Networks.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.
 - [374] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. “Universal Transformers.” In: *International Conference on Learning Representations (ICLR)*. 2019.
 - [375] Zhijie Deng, Yinpeng Dong, and Jun Zhu. “Batch Virtual Adversarial Training for Graph Convolutional Networks.” In: *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data*. 2019.
 - [376] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding.” In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (NAACL-HLT Long and Short Papers)*. Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: [10.18653/v1/N19-1423](https://doi.org/10.18653/v1/N19-1423). URL: <https://www.aclweb.org/anthology/N19-1423>.
 - [377] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. “Neural Architecture Search: A Survey.” In: *Journal of Machine Learning Research (JMLR)* 20.55 (2019), pp. 1–21. URL: <http://jmlr.org/papers/v20/18-598.html>.
 - [378] Hongyang Gao and Shuiwang Ji. “Graph U-Nets.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2019.
 - [379] William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. “MineRL: A Large-Scale Dataset of Minecraft Demonstrations.” In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. July 2019, pp. 2442–2448. DOI: [10.24963/ijcai.2019/339](https://doi.org/10.24963/ijcai.2019/339).
 - [380] Miles Hutson, Olga Kanzheleva, Caitlin Taggart, Bilson Campana, and Quang Anh Duong. “Quality Control Challenges in Crowdsourcing Medical Labeling and Diagnosis.” In: (2019).
 - [381] Jiaqi Ma, Weijing Tang, Ji Zhu, and Qiaozhu Mei. “A Flexible Generative Framework for Graph-based Semi-supervised Learning.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.
 - [382] OpenAI. *OpenAI Five*. <https://openai.com/five/>. 2019.
 - [383] Emmanouil Antonios Platanios. *Swift RL*. <https://github.com/eaplatanios/swift-rl>. 2019.
 - [384] Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabás Póczos, and Tom M. Mitchell. “Competence-based Curriculum Learning for Neural Machine Translation.” In: *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 2019.

- [385] Meng Qu, Yoshua Bengio, and Jian Tang. “GMNN: Graph Markov Neural Networks.” In: *Proceedings of the International Conference on Machine Learning (ICML)*. 2019.
- [386] Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. “Latent Multi-Task Architecture Learning.” In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2019.
- [387] Otilia Stretcu, Krishnamurthy Viswanathan, Dana Movshovitz-Attias, Emmanouil Antonios Platanios, Sujith Ravi, and Andrew Tomkins. “Graph Agreement Models for Semi-Supervised Learning.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.
- [388] Dustin Tran, Dusenberry Mike, Mark van der Wilk, and Danijar Hafner. “Bayesian Layers: A Module for Neural Network Uncertainty.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019, pp. 14633–14645.
- [389] Vikas Verma, Meng Qu, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. *GraphMix: Regularized Training of Graph Neural Networks for Semi-Supervised Learning*. 2019. arXiv: [1909.11715](https://arxiv.org/abs/1909.11715) [cs.LG].
- [390] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M. Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, et al. “AlphaStar: Mastering the Real-Time Strategy Game StarCraft II.” In: *DeepMind Blog* (2019).
- [391] Benjamin Lengerich, Sarah Tan, Chun-Hao Chang, Giles Hooker, and Rich Caruana. “Purifying Interaction Effects with the Functional ANOVA: An Efficient Algorithm for Recovering Identifiable Additive Models.” In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2020.
- [392] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. “Learning Dexterous In-Hand Manipulation.” In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20. DOI: [10.1177/0278364919887447](https://doi.org/10.1177/0278364919887447). eprint: <https://doi.org/10.1177/0278364919887447>. URL: <https://doi.org/10.1177/0278364919887447>.
- [393] Emmanouil Antonios Platanios, Maruan Al-Shedivat, Eric Xing, and Tom M. Mitchell. “Learning from Imperfect Annotations.” In: *Under Review*. 2020.
- [394] Emmanouil Antonios Platanios, Otilia Stretcu, Abulhair Saparov, and Tom M. Mitchell. “Learning When To Trust Your Neighbors: Robust Graph-Based Semi-Supervised Learning.” In: *Under Review*. 2020.
- [395] Emmanouil Antonios Platanios*, Abulhair Saparov*, and Tom M. Mitchell. “Jelly Bean World: A Testbed for Never-Ending Learning.” In: *International Conference on Learning Representations (ICLR)*. 2020.

- [396] Emmanouil Antonios Platanios*, Otilia Stretcu*, George Stoica*, Barnabás Póczos, and Tom M. Mitchell. “Contextual Parameter Generation for Knowledge Graph Link Prediction.” In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2020.
- [397] Otilia Stretcu, Emmanouil Antonios Platanios, Tom M. Mitchell, and Barnabás Póczos. “Learn to Walk Before You Run: Coarse-to-Fine Curriculum Learning for Classification.” In: *Under Review*. 2020.

COLOPHON

This document was typeset using a modified version of the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst’s seminal book on typography “*The Elements of Typographic Style*.” `classicthesis` is available for both $\text{\LaTeX}$ and $\text{\LyX}$:

<https://bitbucket.org/amiede/classicthesis/>

Final Version as of May 12, 2020 (Version).