# Bandit Methods and Selective Prediction in Deep Learning

**Author**: Weicheng Ye


Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Mathematics at the Department of Mathematical Sciences in Carnegie Mellon University

**Supervisor**: Professor Shlomo Ta'asan

Pittsburgh, April 2020

# Acknowledgments

An advisor is unique in one's life. At the very outset, I would like to extend my sincere gratitude to my advisor, Shlomo Ta'asan, for his influence over my Ph.D. life, as a mentor, supporter, advice giver, and friend. I appreciate all that he has done for me, his energy in work, his enthusiasm for research, and his consistent support especially in the face of failures and difficulties. I am especially grateful for the freedom that he has allowed in my research, as well as his unique perspective on mathematical modeling, which will no doubt shape my philosophy as I move forward.

Over the years, many people have also helped shape my research style and interests, and I have learned different skills from each one of them. I would like to thank my collaborators: Debsoumya Chakraborti, Dawei Geng, Ilqar Ramazanli, Tianyu Wang, Dr. Dangxing Chen, and Dr. Cynthia Rudin. Special thanks are given to the faculty at Carnegie Mellon University, in particular Professor Bill Hrusa, Professor Dmitry Kramkov, Professor Giovanni Leoni, Professor Steve Shreve, Professor Noel Walkington, for introducing me to this fascinating world of vision and learning.

I have also greatly benefited from fellows and friends for whom I have great respect and admiration. Thank you to DaQi Chen, Zixuan Chen, Mochong Duan, Giovanni Gravina, Hongyu Li, Han Nguyen, Antoine Remond-Tiedrez, Yufeng Shen, Son Van, Zeyang Ye, and Linan Zhang, for discussions, feedback, and friendship. Thank you for making life less lonely and more enjoyable.

Last but far from least, thank you, my wonderful parents, Yan Jiang and Jun Ye, for their love, wisdom, and encouragement over the many years. They are the first people to believe in me, for always supporting my education, for letting me know that no matter where I wander I can always come home. This thesis is dedicated to them.

**Weicheng Ye**

# Abstract

This thesis consists of three parts involving two areas in the field of machine learning: deep learning and reinforcement learning.

Stochastic Lipschitz bandit algorithms are methods that govern exploration-exploitation trade-offs and have been used for a variety of important task domains, including zeroth-order optimization. In the first part, we present a framework for Lipschitz bandit methods that adaptively learns partitions of context- and arm-space. Due to this flexibility, the algorithm can efficiently optimize rewards and minimize regret, by focusing on the most relevant portions of the space. Tuning hyperparameters is a crucial problem in deep learning that is required in every single model. Although the deep neural network with high complexity can fit the large-scale data very well, it uses plenty of hyperparameters. Our algorithms can achieve state-of-the-art performance in challenging real-world tasks such as neural network hyperparameter tuning.

Model-free Reinforcement Learning algorithms, such as $Q$-learning, require less space and are more expressive to use compared to model-based approaches. Upper Confidence Bound (UCB) exploration policy improves the exploration bonuses in the $Q$-learning framework. In the second part of this work, we study the regret bound of the $Q$-learning algorithm with UCB exploration in the scenario of compact state-action metric space. We develop a $Q$-learning algorithm with UCB exploration policy that adaptively discretizes the continuous state-action space and iteratively updates $Q$-values. In addition, the algorithm can efficiently optimize rewards and minimize the cumulative regret. We provide a rigorous analysis of bounding the regret with concentration of measure analysis and combinatorial approaches. This is the first result of this kind to the best of our knowledge.

Data gathered from real-world applications often suffer from corruption. The low-quality data will hinder the performance of the learning system in terms of the classification accuracy, model building time, and interpretability of the classifier. Selective prediction, also known as prediction with a reject option, reduces the error rate by abstaining from prediction under uncertainty while keeping coverage as high as possible. In the third part of this work, we propose sophisticated threshold learning algorithms integrated with selective prediction that can estimate the intrinsic rejection rate of the dataset. Correspondingly, we provide a rigorous framework to generalize the estimation of data corruption rate. To leverage the advantage of multiple learning methods, we extend learning algorithms to a hierarchical two-stage system. Our methods have advantages of being noise-robust, intelligible, and flexible with any network architecture. The empirical results show that our algorithms can accomplish state-of-the-art performance in real-world datasets in many classification and regression problems.

# Contents

**1 Introduction** **1**

1.1 Machine Learning Overview . . . 1
1.2 Deep Neural Network . . . 6
1.3 Convolutional Neural Network . . . 7
1.4 Back-propagation . . . 10
1.4.1 Loss Functions . . . 10
1.4.2 Back-propagation with Gradient Descent . . . 11
1.4.3 Adam Optimizer . . . 13
1.5 Regularization in Deep Learning . . . 14
1.5.1 Bias-Variance Trade-off . . . 15
1.5.2 $L_1$ and $L_2$ Regularization . . . 16
1.5.3 Dropout . . . 16
1.5.4 Early Stopping . . . 17
1.6 Batch Normalization . . . 17
1.7 Reinforcement Learning . . . 18
1.7.1 Multi-Armed Bandit . . . 18
1.7.2 Q Learning . . . 19
1.8 Thesis Overview . . . 21

**2 Practical Lipschitz Stochastic Bandits** **24**

2.1 Introduction . . . 24
2.2 The TreeUCB framework . . . 26
2.2.1 Partition to UCB Strategy . . . 27
2.2.2 The TreeUCB algorithm . . . 27
2.2.3 The Contextual TreeUCB algorithm . . . 29
2.3 Main Analysis . . . 31
2.3.1 Supported Propositions . . . 32
2.3.2 Point Scattering Inequalities . . . 36
2.3.3 Provable Guarantee for TreeUCB Algorithm . . . 39
2.4 Some Applications of Point Scattering Inequalities . . . 41
2.4.1 The UCB1 algorithm . . . 41
2.4.2 The `UniformMesh` algorithm in $[0,1]^d$ . . . 44
2.4.3 The contextual `UniformMesh` in $[0,1]^d$ . . . 44
2.5 Empirical Study . . . 48
2.5.1 Tree Fitting Rule in Experiments . . . 49

2.5.2 Synthetic Data . . . 49
2.5.3 Neural Network Tuning . . . 50
2.6 Conclusion . . . 51
2.7 Supplementary . . . 52

**3 The Analysis of Performance Measure in Q Learning 56**
3.1 Preliminaries . . . 56
3.1.1 Markov Decision Process . . . 56
3.1.2 Model-based and Model-free Reinforcement Learning . . . 57
3.2 Introduction . . . 58
3.3 Notations and Settings . . . 59
3.3.1 $Q$-Learning . . . 59
3.3.2 Lipschitz Assumption . . . 60
3.3.3 Performance Measure . . . 61
3.3.4 Iterative Updating Rule . . . 61
3.4 Performance Measure Algorithms and Theorems . . . 63
3.5 Main Analysis . . . 64
3.5.1 Concentration of Measure . . . 64
3.5.2 Assumptions for Concentration . . . 68
3.5.3 Upper Bound of Cumulative Regret . . . 69
3.6 Regret Bound in Covering Dimension . . . 71
3.7 Conclusion of Main Theorem . . . 74
3.7.1 Proof of Main Theorem . . . 74
3.7.2 Conclusion . . . 75
3.8 Supplementary Discussion . . . 76

**4 Learning Algorithms in Selective Prediction 78**
4.1 Introduction . . . 78
4.2 Preliminaries . . . 80
4.2.1 Chow's Rule – Optimal Rejection Rule . . . 80
4.2.2 Tortorella's Rule – Reject Rule in Binary Classification . . . 81
4.2.3 Selective Net . . . 83
4.3 Threshold Learning with Selective Prediction . . . 84
4.3.1 Advantage of Deep Learning with Selective Prediction . . . 84
4.3.2 Selective Net with Automatic Learning (SNAL) . . . 86
4.3.3 Threshold Learning with Reject Option (TLRO) . . . 88
4.4 Concentration of Measure Results . . . 89

4.5 Hierarchical Two-Stage Learning Algorithm . . . . . 91
4.5.1 Advantage of Two-Stage System . . . . . 92
4.5.2 Algorithm . . . . . 93
4.6 Empirical Study . . . . . 95
4.6.1 Datasets . . . . . 95
4.6.2 Baseline Models . . . . . 96
4.6.3 Cats vs. Dogs dataset . . . . . 98
4.6.4 Synthetic-noise Cats vs. Dogs Dataset . . . . . 99
4.6.5 CIFAR-10 Dataset . . . . . 100
4.6.6 Concrete Compressive Strength Dataset . . . . . 101
4.6.7 Two-Stage System . . . . . 102
4.7 Conclusion . . . . . 103
4.8 Supplementary . . . . . 104

**References** **114**

# List of Figures


1 Examples of handwriting digits (MNIST) . . . . . 4
2 Examples of pattern recognition (CIFAR-10) . . . . . 5
3 Examples of house number recognition (SVHN) . . . . . 6
4 Neural Network . . . . . 8
5 Higher correlation of images on nearby pixels . . . . . 8
6 Filters in CNN . . . . . 9
7 Path from GD vs. Path from SGD . . . . . 13
8 Bias-Variance Trade-off . . . . . 15
9 Upper Left: Himmelblau for the non-contextual task. Upper Right: Goldstein for the non-contextual task. Lower Left: Himmelblau for the contextual task. Lower Right: Goldstein for the contextual task. All methods use 1 as exploration parameter. (C)TUCB$_1$, (C)TUCB$_2$, (C)TUCB$_3$ corresponds to TUCB (CTUCB) with $\eta = 0.02$, $\eta = 0.01$, $\eta = 0.005$ correspondingly ($\eta$ as discussed at the beginning of this section). . . . . 51
10 For MNIST, each plot is averaged over 10 runs. For SVHN and CIFAR-10, each plot is averaged over 5 runs. The implementation of TUCB here uses sci-kit-learn package. In the left-most subplot, $x$-axis is time (in seconds). This shows TUCB's scalability, since TUCB's curve goes up the fastest. . . . . 52
11 Selective Net . . . . . 84
12 Two-Stage Learning System . . . . . 94
13 Upper Left: noise-free cats. Upper Right: synthetic-noise cats. Lower Left: noise-free dogs. Lower Right: synthetic-noise dogs. . . . . 96
14 SNAL vs. Selective Net in noise-free Cats vs. Dogs . . . . . 98
15 TLRO vs. Selective Net in noise-free Cats vs. Dogs . . . . . 99
16 SNAL vs. Selective Net in synthetic-noise Cats vs. Dogs . . . . . 100
17 TLRO vs. Selective Net in synthetic-noise Cats vs. Dogs . . . . . 100
18 SNAL vs. Selective Net in CIFAR-10 . . . . . 101
19 SNAL vs. Selective Net in Concrete Compressive Strength Left: Test loss comparison Right: Percentage improvement in loss against the benchmark . 102
20 Two-stage System vs. SNAL vs. Selective Net Left: Noise-free Cats vs. Dogs Right: Synthetic-noise Cats vs. Dogs . . . . . 103
21 Two-stage System vs. SNAL vs. Selective Net in CIFAR-10 . . . . . 103
22 Type-II Two-Stage Learning System . . . . . 113

# List of Algorithms

1 TreeUCB (TUCB) . . . . . 28
2 Contextual TreeUCB (CTUCB) . . . . . 31
3 UCB1 Algorithm [Auer et al., 2002] . . . . . 42
4 Simple `UniformMesh` [Kleinberg et al., 2008] . . . . . 45
5 Simple contextual `UniformMesh` [Slivkins, 2014] . . . . . 45
6 Q-learning with UCB bandit in metric space . . . . . 63
7 Two-Stage Learning System . . . . . 95
8 Two-Stage Learning System (Type-II with SNAL) . . . . . 113

# 1 Introduction

## 1.1 Machine Learning Overview

Machine learning is a branch of Artificial Intelligence. As a quote from Tom M. Mitchell, a computer scientist, and machine learning pioneer at CMU: “Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience.” Machine Learning is one of the most powerful ways to achieve AI. Neil Lawrence, a Professor from the University of Cambridge, described Machine Learning (ML) as “the principal technology underpinning the recent advances in artificial intelligence.”

Machine Learning’s development is highly correlated with AI history. Although AI arises in a variety of fields from both academia and industry nowadays, the development of AI is not smooth. The golden stage of AI came in the 1950s and 60s. The first-generation of AI researchers could design programs to achieve some elaborate goals like winning a simple chess game and proving a theorem. The way that AI programs complete these tasks is called "reasoning as a search". By searching through an environment, these programs were making a move step by step, and backtracking whenever the dead-end is reached. At the same time, researchers believed computers could communicate in natural languages like English. Daniel G. Bobrow, a researcher from MIT, built up the machine that could understand the language of math and solve simple high school algebra word problems. This program was called "STUDENT". After some success in the early stage, the first generation of AI researchers became optimistic. In the 1950s, Allen Newell and H.A. Simon, computer science professors from CMU believed: “within ten years, a digital computer will be the world’s chess champion” and “within ten years, a digital computer will discover and prove an important new mathematical theorem.” Bobrow’s Ph.D. advisor, Marvin Minsky from MIT also claimed: “In from three to eight years we will have a machine with the general intelligence of an average human being.”

However, the AI winter came in 1974. AI researchers encountered a bottleneck in their research due to several fundamental limits. These limits mainly included:

- limited computer powers
- computers’ lack of common sense and knowledge
- Moravec’s paradox: although complex tasks like proving theorems and solving geometry problems might be comparatively easy for computers, tasks simple for humans like face recognition and passing through the maze are extremely difficult.

Most of the AI tasks require large memory and fast processing speed of computers. Just like human beings, computers need a large amount of pictures and words to know what

the world looks like and talks about. Without data and memory, AI is hard to accomplish anything truly valuable in applications. For example, Ross Quillian's semantic network project in natural language could demonstrate with a vocabulary of only twenty words, since that was all the words that can fit into memory. Besides natural language processing, other major AI applications like computer vision also require enormous information to build up the fundamental system. When AI researchers' promised results failed to materialize, the national funding of AI projects also disappeared immediately. For example, the Defense Advanced Research Projects Agency (DARPA) was disappointed in the progress of the speech understanding projects at CMU and canceled the annual grant of three million dollars. At the same time, the field of connectionism (early name of neural network), was shut down for almost 10 years.

Recall that we highlight the main limits of AI development in the last paragraph. The first fundamental limit is resolved by the development of computers, while the latter two limits are improved by the accomplishment of machine learning. Due to the development of the computer system and the generation of large-scale data after the growth of the Internet, AI came back to researchers' eyes and achieved some of its oldest goals. A machine learning algorithm has a goal to maximize the performance of certain tasks. It is a data-driven approach to decision making and overlaps a great deal with statistics, applied math and operations research. In the first decades of the $21^{st}$ century, easier access to large amounts of data, known as "big data", and cheaper and faster computer system accelerates machine learning's success to many problems that were considered very hard in AI throughout the economy. By 2016, the market for machine learning-related products, hardware, and software reached more than 8 billion dollars.

We introduce some applications of machine learning in technology.

**Example 1.1.** Image search at Google: Google allows users to search for information in photos. Users can search for the subjects in the photos and similar photos online that have the same subjects or color palette. This search algorithm is based on machine learning. With sufficient data to train and learn, the machine learning algorithm can analyze color and shape patterns and pair that with any existing schema data about the photo to help the search engine understand uploaded images making them ready for future searches.

**Example 1.2.** Product Recommendations: Many of us shop online every day from dining to entertainment. After we shop a product online or even just open several pages of products, we might receive emails for related shopping recommendations. We must have also noticed that the shopping website or the app can recommend to us some items that mostly match with our taste. Machine learning refines the product recommendation based on our shopping experience, brand preferences, and similar customers' shopping cart.

**Example 1.3.** Social Media Services like Facebook.

- People Users May Know: machine learning works on a simple concept: understanding with users' experiences. To recommend the people users might know, the machine learning algorithm uses the data like the friends that users connect with, the profiles that users often visit, users' interests, and users' workplace.
- Face Recognition: if users upload photos of them with friends, then machine learning algorithm will recognize users' friends on the photos based on their poses, projections, and their friends' list.

**Example 1.4.** Commuting Predictions.

- Traffic Predictions: When users use GPS navigation services, the GPS will provide congestion in current traffic and the estimated arrival time. This is because users' current locations and velocities are being saved at a central server while using GPS. Machine learning algorithms use all the traffic data to build a map of current traffic and predict the congestion.
- Online Transportation Networks: when users ask a ride on Uber, machine learning is also the key algorithm to optimize the drivers for them and estimate the price of the ride given the current traffic.

Here at CMU, faculty has a tradition of bringing cutting-edge machine learning to the technology industry. Professor Jeff Schneider is leading Uber ATG's engineering team to use the machine learning model of defining price surge hours by predicting Uber's rider demand. Professor Ruslan Salakhutdinov is in charge of Apple's AI research team to make Siri more intelligent by deep neural networks. The head of the machine learning department, Professor Manuela Veloso is now the head of J.P. Morgan AI research and is directing a team to work on credit fraud detection and smart virtual assistant for clients.

After introducing some significant applications in industry, let us work on basic examples to elaborate on how the data is interpreted and machine learning works. The problem of searching for patterns in data is fundamental. The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of algorithms and with the use of these regularities to take actions such as classifying the data into different categories. Let us start with the classification of recognizing handwritten digits.

**Example 1.5** (MNIST)**.** We have 10 digits from $\{0, \cdots, 9\}$ [LeCun, 1999]. Each digit is corresponding to a $28 \times 28$ pixel image. We represent it as a 784 real number vector $x$ where each value on the vector stands for one-pixel value. Some examples are illustrated in Figure

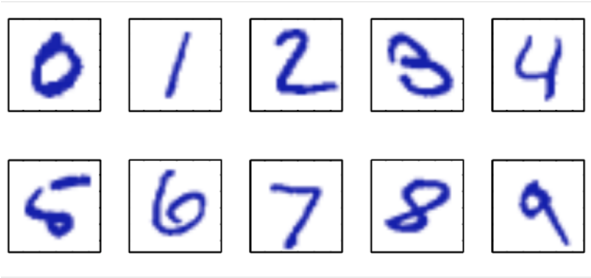


Figure 1: Examples of handwriting digits (MNIST)

1. This is the MNIST dataset. The goal of machine learning is to take a new digit image (vector $x$) as input and predict which digit it stands for from $\{0, \cdots, 9\}$ as output. Let us say we have a large set of $N$ digit images, $x_1, \cdots, x_N$, in the training phase. The corresponding categories of digits for the training set are known in advance by inspecting them individually and hand-labeling them. The machine learning algorithm can generate an adaptive model to express a function $f$, where the output vector is $y = f(x)$, encoded in the same way as the target vectors. The training set is used to tune and revise the parameters of function $f$. After training, we use $f$ to determine the identity of new digit images in a test set. The ability to classify new digit test samples into the correct place is called generalization. In practice, improving generalization power is the central goal of machine learning.

Here is another classification example of pattern recognition called CIFAR-10. We have colorful objects with multiple classes.

**Example 1.6** (CIFAR-10)**.** We have object images with 10 classes [Krizhevsky and Hinton, 2009]. These classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. We illustrate a sample in Figure 2. The goal of machine learning is to take a new object image as input and predict the category it stands for as output. Each input image ($x$) has 32×32×3 pixels, where 3 stands for three hues of light RGB. Same as above, the machine learning algorithm generates an adaptive model to express function $f$, where $f$ could map $x$ to output $y$, a 10-$d$ vector that represents the probability of the sample being into each class. The goal is to improve the generalization, the accuracy of prediction in the test set.

**Example 1.7** (SVHN)**.** In the Street View House Number [Netzer et al., 2011], each input image ($x$) is 32×32×3 pixels and stands for the house number on the streets. Many of the house number images are noisy and with blur. One example is shown in Figure 3. Function $f$ generated by machine learning algorithm could map $x$ to the output $y$, which is the specific house number. The goal of the machine learning algorithm is again to improve the generalization of this noisy image dataset.

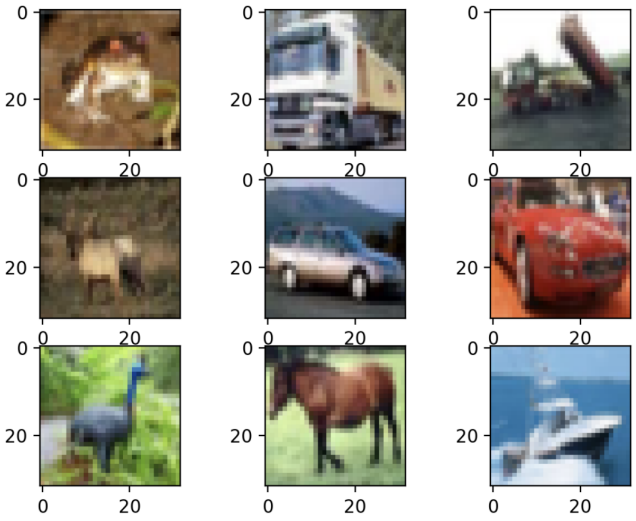


Figure 2: Examples of pattern recognition (CIFAR-10)

The above examples are the classification problem. Besides that, regression is another classical problem in machine learning with various applications. In regression, the output is a numerical value and the goal is to predict the output as accurately as possible. Let us see the following example.

**Example 1.8** (Concrete Compressive Strength Data)**.** This dataset uses materials ingredients and ages to predict the concrete compressive strength [Dheeru and Taniskidou, 2017]. Inputs are 8 numerical attributes including cement, water, fly ash and coarse aggregate, etc. and the corresponding output is concrete compressive strength. The goal of the machine learning algorithm is to construct a highly nonlinear function that can take 8 input attributes to predict the compressive strength value. The output will be 1-$d$ numerical value. Here we construct the training and test datasets by randomly splitting the whole dataset into two portions. We focus on the generalization power of the machine learning algorithm.

Machine Learning relies on two things: data and model. The high-quality prediction results of machine learning acknowledge access to massive data nowadays. Another important part is the evolution of machine learning models, especially in deep learning. Deep learning, as the most exciting and attractive subject in machine learning nowadays, contains many valuable models such as deep neural network, convolutional neural network and recurrent neural network. These models have revealed great contributions in practice. In the following sections, we introduce deep neural network and convolutional neural network, also with important concepts that are useful to later chapters. Because the practical problems usually involve large-scale data, complex models like deep neural networks typically can fit data

Figure 3: Examples of house number recognition (SVHN)

very well. In addition, complex models like neural network contain various hyperparameters. Hyperparameters are the variables that determine the network structure and how the network is trained (more details later). Optimizing the selection of hyperparameters to achieve the highest test accuracy is called hyperparameter tuning. Hyperparameter tuning is a crucial problem in deep learning that appears in every single model. In later sections, we will introduce some hyperparameters in models along the way.

## 1.2 Deep Neural Network

Human beings are very good at tasks like pattern and speech recognition. However, humans cannot memorize and process as much data as a machine. That's the reason why deep learning researchers use human brains' structure as a baseline model. A deep neural network inherits the biological structure of human brains as shown in Figure 4 [Goodfellow et al., 2016]. As we know, the perception of humans is generated through the signal transmission among neurons. In the neural network model, each node from the input, hidden, and output layers is called a neuron. Here, the input layer takes a numerical representation of data. We use Example 1.5 for illustration purpose. The input layer contains 784 neurons since the input data $x$ has dimension 784. The output layer is 10-$d$ which stands for the probability of being into 10 classes (10 digits). In between, hidden layers carry the most of the computations just like signal transmission in the human brain and can contain an arbitrary number of neurons. Each neuron carries out two-step computations. Assume there are $m$ neurons in hidden layer $L-1$ and the outputs from these neurons are $\{x_1 \cdots, x_m\}$. For neuron $j$ in next hidden layer $L$, the computation will be two steps:

- Step 1: take the linear combination of the neurons in layer $L-1$ with bias: $\sum_{i=1}^{m} w_{ij}^{(L)} x_i +$

$b_j$, where $w_{ij}^{(L)}$ is the weight that connects node $i$ at layer $L-1$ and node $j$ at layer $L$ and $\{b_j\}$ are the bias. The reason to use a linear combination is because of the easy computation of gradients and its approximation of any nonlinear function.

- Step 2: apply activation function after Step 1. Assume activation function is $g$, then the final output after two steps will be $g\left(\sum_{i=1}^{m} w_{ij}^{(L)} x_i + b_j\right)$.

The goal of activation function is to apply simplicity or nonlinearity on the output. For example, two classical choices of activation functions are rectified linear unit (relu) and sigmoid. Relu activation is

$$g(\xi) = \begin{cases} \xi & \text{if } \xi \geq 0\,; \\ 0 & \text{if } \xi < 0\,, \end{cases} \tag{1.1}$$

where $\xi = \sum_{i=1}^{m} w_{ij}^{(L)} x_i + b_j$. Relu has two great properties: non-negativity and easy for differentiation. Sigmoid activation is

$$g(\xi) = \frac{1}{1+e^{-\xi}}\,. \tag{1.2}$$

The nonlinear sigmoid function could scale the output into [0,1]. This is valuable because many classification problems require the probability of classes as output and [0,1] could be the expression of probability.

The output from two-step computations in each neuron then communicates to the next hidden layer and the same process is repeated. This forward process of signals is called the forward propagation. If we combine input, hidden, and output layers, then the neural network will represent a function $f$ that maps input $x$ to output $y$. Note that the number of layers and the number of neurons in each layer are undetermined. They are considered as hyperparameters in neural networks. As one might think, the different combinations of hyperparameters could lead to different results. How to tune the hyperparameters to get the optimal result is a challenging problem in machine learning research.

## 1.3 Convolutional Neural Network

Convolutional Neural Network (CNN) is a featured structure of deep learning that is specific for pattern/image recognition. They address the translation and scaling of the objects in the image classification problem. Recall that from Example 1.6, object images are with many pixels and 3 color channels RGB. If we use the regular neural network with fully connected architecture, then the amount of weights rapidly becomes unmanageable. Overfitting (which will be introduced later) will occur heavily if the model is too complex. Another issue is that

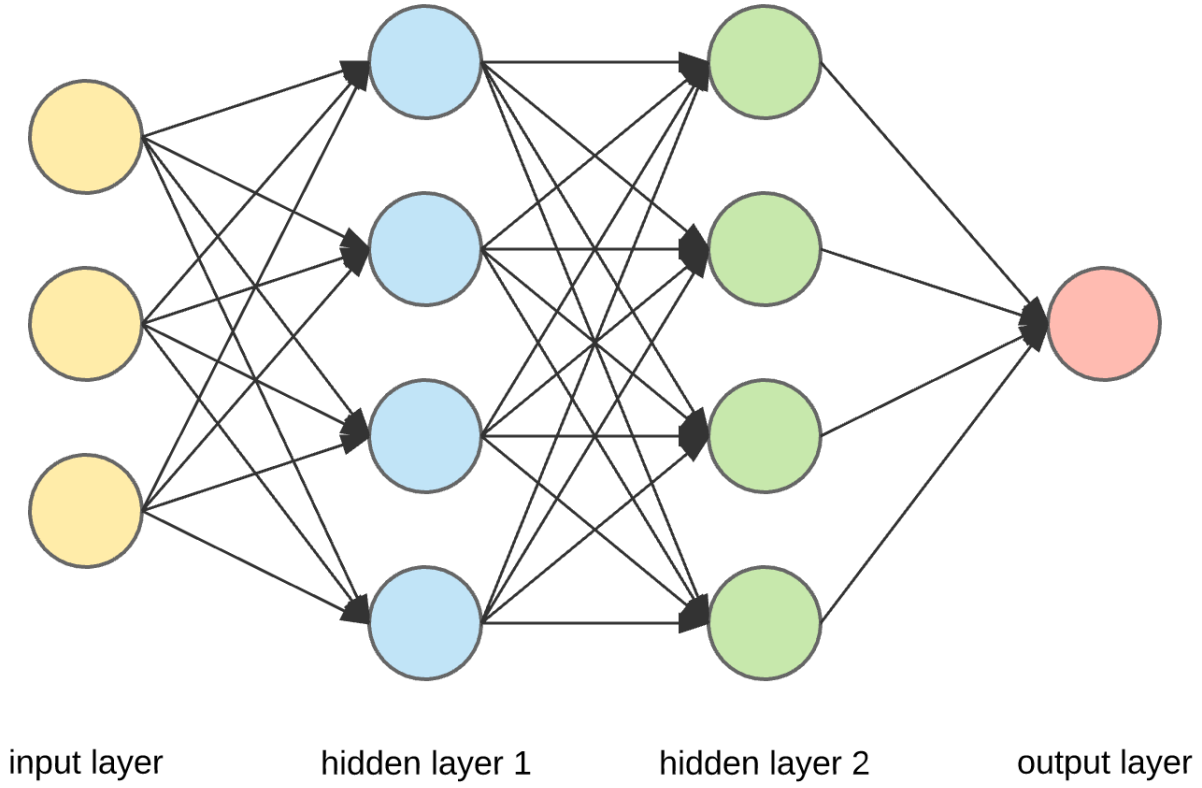



Figure 4: Neural Network

regular neural network is not translational invariant, which means it reacts very differently to an input image and its shifted version. For example, if the pattern of a dog appears in the top left in one picture and the bottom right of another picture, then the neural network will assume that a dog always appears in this section of the image. Thus, we believe that nearby pixels are more strongly related than distant ones. This phenomenon is shown in Figure 5.

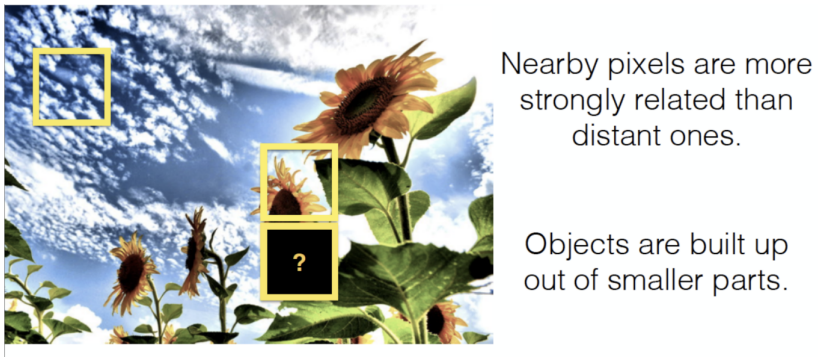



Figure 5: Higher correlation of images on nearby pixels

CNN is a method to focus on the locality of images and leverage the spatial correlation of the image features. In CNN, filter is a tool to capture the local pattern in one image. When

running the algorithm, the filter will go through the image data from the top left to bottom right. Then the convolutional layer will take the convolution operation of the filter and every scanned local part of the image to highlight the feature. Filter is the key to CNN. It can do powerful jobs such as edge detection and sharpening as highlighted in Figure 6. For example, in Example 1.5, CNN could capture features such as vertical lines and circles in various digits. After the convolution is done, CNN has a pooling layer, typically max pooling, to extract the maximum convolution values locally since the significant feature only shows up once in the local region. For example, vertical lines or circles only show up once in a local region in a digit image. This max-pooling value represents the dominating feature of the pattern.

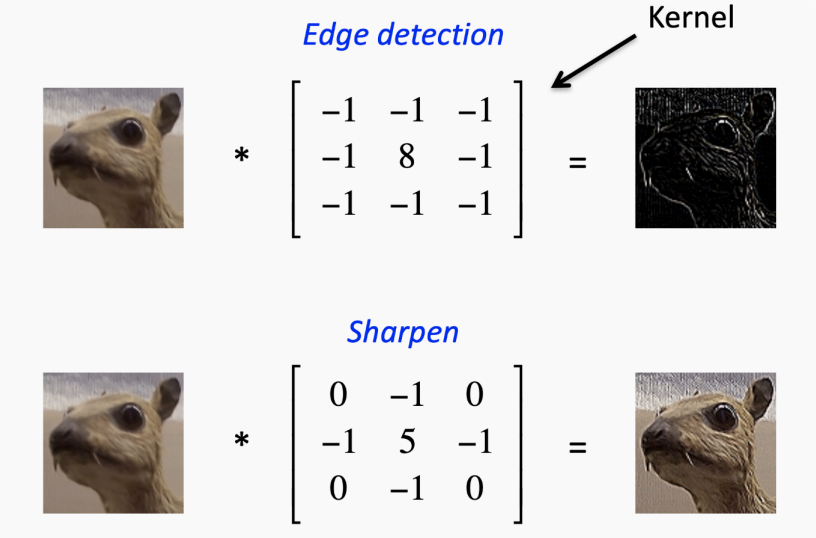



Figure 6: Filters in CNN

Readers might also be curious about how we decide the filters. At the beginning of training, we select the numbers of filters and randomly specify values in each filter. The values in filters continuously get updated as CNN is trained. In CNN, the number of filters and the dimension of each filter are also hyperparameters.

There is one more concept on CNN called padding. Padding is the way to adjust the dimension of output. Two typical paddings are zero padding and valid padding. Note that after convolutional and pooling layers, we will have the output with a smaller dimension. We might add zeros on the sides of the output matrix to remain the same dimension. This is called zero padding. On the other hand, if we accept the smaller dimension of the output without any modification, then it is valid padding. The choice of padding is necessary for CNN and also a hyperparameter.

Researchers have the trend of studying the deep architecture of CNN models to improve the accuracy in the large-scale image recognition setting. For example, models like AlexNet, VGG,

GoogLeNet, and ResNet are the most commonly used deep CNN architecture [Krizhevsky et al., 2012, Simonyan and Zisserman, 2015, Szegedy et al., 2015, He et al., 2016].

## 1.4 Back-propagation

So far, we have introduced how the neural network computes output through forward propagation. In this section, we discuss back-propagation in neural networks which is important for the inverse problem, i.e. finding the network weights given multiple input-output pairs. Determining the weights is done as an inverse problem based on a suitable optimization problem. To this end, we define a loss as the difference between the predicted output and the true output. If the loss is large, then the model is far from optimal and requires additional training time. When loss becomes small and stable, then the training should complete. How to update weights and bias of all the neurons based on some measure of loss? This answer is from back-propagation. We review two important concepts in back-propagation: loss function and optimizers.

### 1.4.1 Loss Functions

Here we introduce several common loss functions. The first one is the mean squared loss. It is used in regression problem and has the following form:

$$\mathcal{L}_{\text{mse}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2\,, \tag{1.3}$$

where $N$ stands for the size of the training dataset, $y_i$ is the true output, and $\hat{y}_i$ is the predicted output. Mean absolute loss is also very commonly seen in regression:

$$\mathcal{L}_{\text{mae}} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|\,. \tag{1.4}$$

The key difference is mean squared loss gives a relatively high weight to large errors and hence has the benefit of penalizing large errors. Mathematically, mean absolute loss has the absolute term, which is undesirable in differentiation.

One common loss function in classification is called cross-entropy loss. It is pretty often in binary class or multi-class classification problems such as Example 1.5 and 1.6. It is

$$\mathcal{L}_{\text{cross-entropy}} = -\sum_{i=1}^{N}\sum_{c=1}^{m} y_{i,c}\log p_{i,c}\,, \tag{1.5}$$

where $m$ is the number of classes, $y_i$ is $i^{th}$ sample's true class and $p_i$ is $i^{th}$ sample's predicted output (predicted probability). As in Example 1.5, if $i^{th}$ image is actually digit "3", then $y_i = \{0, 0, 1, 0, 0, 0, 0, 0, 0, 0\}$, where $y_{1,3} = 1$ and $y_{1,j} = 0$ for $j \neq 3$.

The goal of back-propagation is to adjust the weights and biases of the network to minimize the loss. This is an optimization problem. The optimizers in machine learning do the above work. Classically, we have first-order and second-order optimization methods. However, the machine learning community often cares more about the first-order method due to its better performance. The second-order method requires the computation of hessian, which is too costly in each training iteration. Besides, the second-order method is much harder for distributed computing. For example, the second-order method KFAC requires plenty of work for distributed computing, but the result is still not better than the first-order method [Osawa et al., 2019, Zhang et al., 2019]. Therefore, we mainly focus on the first-order optimizer. We briefly review two first-order optimization methods, Stochastic Gradient Descent (SGD) and ADAM, since they are used frequently in later chapters.

### 1.4.2 Back-propagation with Gradient Descent

Gradient descent is the most classical optimization method and very popular in machine learning. In gradient descent, "batch" means the total number of samples to use in one training iteration. Batch gradient descent means the whole dataset as a batch. Since all the samples are being taken, the batch gradient descent will have less noisy information to converge to minima. The total iterations will be relatively small.

Let us see how the backpropagation with gradient descent works mathematically. Assume $\mathcal{L}$ is the loss function, the total number of layers in the neural network is $T$, and the number of neurons at layer $L$ is $n_L$. In addition, recall that $w_{jk}^{(L)}$ is the weight that connects node $j$ at layer $L-1$ and node $k$ at layer $L$. Denote by $z_k^{(L)}$ the value of node $k$ at layer $L$ before activation and $a_k^{(L)}$ the value of node $k$ at layer $L$ after activation, e.g., $a_k^{(L)} = g\left(z_k^{(L)}\right)$ where $g$ is the activation function. Assume parameter set $\vec{\theta} = \left\{w_{jk}^{(L)}, a_k^{(L)}\right\}_{j,k=1,\cdots,n_L}^{L=1,\cdots,T}$ stands for all the connecting weights and activation values. The way to update $\vec{\theta}$ in the batch gradient descent is

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t - \eta \nabla_{\vec{\theta}_t} \mathcal{L}(\vec{\theta}_t)\,, \tag{1.6}$$

where $\eta$ is the learning rate and the way to update $\mathcal{L}$ using the whole population batch.

For example, the gradient with respect to the connecting weight is

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^{(L)}} = \frac{\partial \mathcal{L}}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}} = \frac{\partial \mathcal{L}}{\partial a_j^{(L)}} \cdot g'\left(z_j^{(L)}\right) \cdot a_k^{(L-1)}, \tag{1.7}$$

and the gradient with respect to the activation value is

$$\frac{\partial \mathcal{L}}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial \mathcal{L}}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial \mathcal{L}}{\partial a_j^{(L)}} \cdot g'\left(z_j^{(L)}\right) \cdot w_{jk}^{(L)}, \tag{1.8}$$

with the sum over layer $L$ above. Backpropagation is to backtrack through all of its layers using gradient to update the weights and bias of each node in the opposite direction of the loss function. It is called backpropagation since the process is in the backward order. With the completion of updating weights and bias from the ending layer to the starting layer, it is the completion of one training epoch. Then one can use the newly updated weights to compute new output and repeat the process in the next training epoch.

However, the problem arises since each training step will be extremely costly when the dataset is huge. The problem of batch gradient descent could be solved by stochastic gradient descent (SGD). In SGD, the batch size is one in each iteration. All the samples are randomly shuffled and the algorithm picks one sample each iteration for gradient computation. This will introduce heavy noise. The convergence path will be much more zigzag than batch gradient descent, but the training time of each epoch will be significantly less. In practice, we mostly prefer to use SGD since it performs well in large datasets. For iteration $t$, given only one randomly picked training sample $x^{(i)}$ and corresponding label $y^{(i)}$, SGD performs the following parameter update:

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t - \eta \nabla_{\vec{\theta}_t} \mathcal{L}\left(\vec{\theta}_t; x^{(i)}; y^{(i)}\right). \tag{1.9}$$

Note that the parameter update rule (1.9) is same as batch gradient in (1.6). The only difference is that the update of $\mathcal{L}$ here is to use one sample rather than the whole population, e.g., (1.3), (1.4), and (1.5) above have $N = 1$. We use $\mathcal{L}\left(\vec{\theta}_t; x^{(i)}; y^{(i)}\right)$ to differentiate with $\mathcal{L}\left(\vec{\theta}_t\right)$ in batch gradient descent.

The difference of convergence paths from batch gradient descent and SGD is shown in Figure 7. Batch gradient descent performs lots of redundancy in the large dataset because it recomputes the gradients for the same samples over and over before the update of the parameter. SGD removes this redundancy due to the performance of the random sample each time. SGD is faster overall in training and also called the online learning algorithm.

To remedy the slow convergence of SGD due to inherent variance, researchers have been

working on the acceleration of SGD by variance reduction technique [Johnson and Zhang, 2013] and make it work on distributed computing [Lee et al., 2017, Cutkosky and Busa-Fekete, 2018]. Interested readers can find the content from the cited literature.

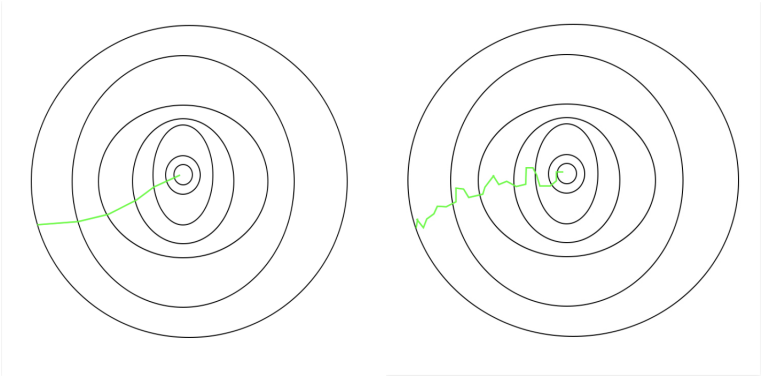


Figure 7: Path from GD vs. Path from SGD

### 1.4.3 Adam Optimizer

Batch gradient descent takes too much computational time during each step, but SGD still takes too many iteration steps, although each iteration is fast. After several intermediate revolutionary methods including momentum method, Nesterov accelerated gradient (NAG), AdaGrad, RMSProp [Duchi et al., 2011, Ruder, 2017, Sutskever et al., 2013], Kingma and Ba finally introduced Adaptive Moment Estimation (Adam) [Kingma and Ba, 2014], an efficient optimization algorithm that updates the weights in network iteratively during training. Based on Kingma and Ba, Adam has the following advantages:

- straightforward to implement
- computationally efficient
- hyperparameters have intuitive interpretation and typically require little tuning time

We know SGD has high variance oscillations that result in more training steps. Machine learning researchers invent a term called momentum. The key idea is to accelerate SGD by navigating along the relevant direction and soften the oscillations in irrelevant directions. The name of momentum is exactly from Physics. In an optimization algorithm, it is the same as throwing a ball from hill where the ball gets momentum as the velocity increases. Methods like NAG, AdaGrad, RMSProp, and Adam are all momentum-based methods. Adam mainly keeps track of two things: the exponentially decaying average of past squared gradients $v_t$ at step $t$, and the exponentially decaying average of past gradients $m_t$, where the latter term is

momentum. Adam is like the ball on the hill with friction and its goal is to reach the flat minima on the hill. Let

$$g_t = \nabla_{\vec{\theta}_t} \mathcal{L}(\vec{\theta}_t)\,,$$

where the computation of gradient is the same as the one shown in the previous back-propagation with gradient descent section. The decaying averages of past and past squared gradients $m_t$ and $v_t$ are respectively as the following:

$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t\,,$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2\,,$$

where $\beta_1$ and $\beta_2$ are hyperparameters in Adam. Mathematically, $m_t$ and $v_t$ are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively. Note that the initialization of $m_t$ and $v_t$ are the vectors of 0's, so during the initial training steps, both $m_t$ and $v_t$ are leaning towards zeros, especially when $\beta_1$ and $\beta_2$ are close to 1 (small decay rates). To resolve this issue, Kingma and Ba made a bias-corrected estimates

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}\,,$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}\,.$$

From the introduction of the corrected formula, they made the final Adam update rule:

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}\,.$$

Note that the typical default values for $\beta_1$ is 0.9 and for $\beta_2$ is 0.999. $\epsilon$ is typically as $10^{-8}$. Adam has indicated great empirical results compared to other momentum methods and is the most favorable optimization method [Ruder, 2017, Xu et al., 2016]. Recently, the convergence issue of Adam has been fixed by endowing long term memory of past gradients [Reddi et al., 2019].

## 1.5 Regularization in Deep Learning

Recall from examples that our main goal is to improve the generalization. In this section, we introduce some regularization techniques in machine learning and deep learning that can improve generalization. We start our conversation with the classical bias-variance trade-off.

### 1.5.1 Bias-Variance Trade-off

Bias-variance trade-off is one of the most important concepts in the foundation of machine learning [Bishop, 1995, Belkin et al., 2019]. Bias stands for the difference between the expected prediction of our model and the true output on a dataset. If the model is too simple and interprets the training data very little, then the model is with high bias. We call this phenomenon underfitting on dataset. At the underfitting stage, both the training and test errors are high. Variance is the variability of our model in explaining data. It shows the spread of our predicted results on a dataset. High variance typically means that the model fits the training data very well, but also over-interprets the data. As a result, it loses the generalization power on the new test data. This is called overfitting on dataset. The model with high variance performs very well on the training data, but is sensitive and makes significant errors on the test set. The difference of overfitting and underfitting is in Figure 8.

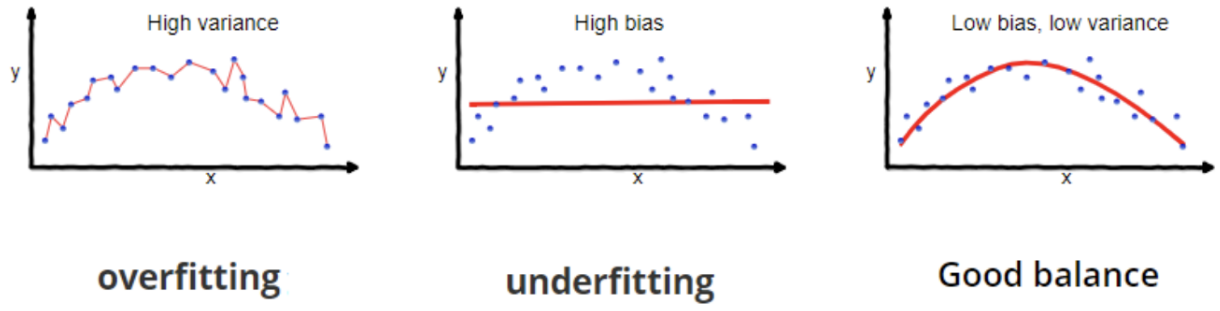



Figure 8: Bias-Variance Trade-off

We can express bias-variance trade-off in equations. Let us find a function $\hat{f}$ that approximates the true function $f$ as well as possible, by means of some learning algorithm on the training dataset. Assume we need to predict $y$ using data $x$. Given $y = f + \epsilon$, where $\mathbb{E}[\epsilon] = 0$ and $\text{var}(\epsilon) = \sigma_\epsilon^2$, the irreducible error, then the expected error is

$$\text{error}(x) = \mathbb{E}\left[y - \hat{f}(x)\right]^2 .$$

Then with further decomposition,

$$\text{error}(x) = \left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2 + \mathbb{E}\left[\left(\hat{f}(x) - \mathbb{E}[\hat{f}(x)]\right)^2\right] + \sigma_\epsilon^2 ,$$

$$\text{error}(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible error} .$$

The irreducible error above is also called Bayers error, the error that measures the noise from data and cannot be reduced by model. This concept will be revisited in chapter 4. Since

this term is irreducible and data-dependent, machine learning researchers focus on the bias and variance in the equation. For example, if our model is simple with few parameters, say a linear function, then it is with high bias and low variance. On the other hand, complex models with a large number of parameters, such as Decision Trees and Deep Neural Network [Quinlan, 1986], have low bias and high variance. Bias-variance trade-off is the subject to find the right and good balance without overfitting and underfitting on the dataset.

### 1.5.2 $L_1$ and $L_2$ Regularization

If our model contains too many parameters and hence overfits, the model is with very high variance and generalization power is low. In machine learning, regularization is an efficient way to prevent overfitting. The most standard regularization technique is $L_2$ and $L_1$ regularization. The idea is to add a penalty term concerning weights and bias to the loss function. With the control of the penalty term, the model will not learn interdependent feature weights during the training phase. $L_2$ (Gaussian) could control the scale of weights and bias. With the reasonably small scale of weights and bias, then small portions of outlier samples could not affect the whole model too much. This is a method to reduce the model variance. In addition, $L_1$ (Laplacian) conducts a feature selection in the model. $L_1$ will push some highly-correlated features to zero weights and hence reduce the model complexity. Assume parameters $\vec{\theta} = \left\{ w_{jk}^{(L)}, a_k^{(L)} \right\}_{j,k=1,\cdots,n_L}^{L=1,\cdots,T}$. Then the loss with $L_2$ regularization is:

$$\mathcal{L}_2(\vec{\theta}) = \mathcal{L}(\vec{\theta}) + \lambda ||\vec{\theta}||_2^2 \,,$$

and the loss with $L_1$ regularization is

$$\mathcal{L}_1(\vec{\theta}) = \mathcal{L}(\vec{\theta}) + \lambda ||\vec{\theta}||_1 \,.$$

$L_1$ and $L_2$ are the most common regularization techniques in classical machine learning algorithms such as support vector machine and logistic regression etc.

### 1.5.3 Dropout

In deep learning, it is more common to use dropout as the regularization technique. The concept is simple and from [Srivastava et al., 2014]. During training, dropout temporarily ignores some neurons so the network is only partially connected. The dropped-out neurons will be selected randomly during each training epoch. To be more specific, each neuron will have probability $p$ to remain in the training while $1-p$ to be dropped out for current training epoch. All the connected edges to and from the dropped-out neurons will also be temporarily

removed. This simpler network in training will reduce the variance and hence avoid overfitting. Besides, with dropout, each neuron could collaborate with different sets of neurons in each training epoch. This will enforce all the neurons' functioning. The philosophy is the same as the following. Workers change their teammates in different tasks, so each worker has no chance to lay back during tasks and hence each one is robust. The network with dropout is also more robust with all the neurons functioning. Note that the probability of being dropped, dropout rate, is also a hyperparameter.

### 1.5.4 Early Stopping

Early stopping is another useful method to avoid overfitting [Caruana et al., 2000, Prechelt, 2012]. Recent Alan Turing Award winner, Geoffrey Hinton said in his NeurIPS presentation: "Early stopping (is) beautiful free lunch." During training, we reserve a validation set for monitoring purposes. Early stopping is to monitor the change of the training error and validation error at the same time to terminate the training at a certain point. If the validation error does not improve significantly anymore, then we terminate the training process before the weights go up and the model becomes overfitted. Early stopping can help control the model complexity to the optimal point.

## 1.6 Batch Normalization

Recall that in backpropagation, the update of weights and bias will involve the product of multiple gradients. When the neural network is deep, such updates will involve a long product of gradients. As long as one of the gradients is close to zero, then the whole update will be close to zero. This is called the vanishing gradient, another challenging problem in neural network training. The vanishing gradient will slow down the training significantly. The notorious vanishing gradient problem is very common to occur without any preprocessing among layers. For example, if the change of weights and bias is significant at one layer, then the linear combination $\xi$ (as defined in section 1.2) might become extreme, and then the gradient of the activation value with respect to $\vec{\theta}$ (as defined in section 1.4), $\nabla_{\vec{\theta}} g(\vec{\theta})$ might be close to zero. This is easily observed based on the definition of relu or sigmoid in (1.1) and (1.2).

Batch normalization is about normalizing the hidden units' activation values for every mini-batch so that the distribution of the activation remains the same during training [Ioffe and Szegedy, 2015]. Technically, the final normalized version of the hidden activation is scaled and shifted. By applying batch normalization, we can choose a higher learning rate to accelerate the training and also care less about initialization. Researchers also find that

batch normalization can perform as a regularizer and reduce the need for dropout [Ioffe and Szegedy, 2015]. However, two powerful methods, batch normalization, and dropout might not be compatible. A new method called Jumpout is to combine the properties of these two regularizers [Wang et al., 2019]. Batch normalization and dropout are two of the most frequently used regularization methods in deep learning.

## 1.7 Reinforcement Learning

After the comprehensive introduction of deep learning, we briefly introduce reinforcement learning, another rich area in machine learning. Readers will find this a useful background for later chapters.

Reinforcement learning is learning how an agent could map situations (states) to suitable actions to maximize the numerical reward [Sutton and Barto, 1998, Francois-Lavet et al., 2018]. In reinforcement learning, agents could utilize past information about state-action sequences and corresponding rewards to deciding for the next move (action). Then the immediate reward will be collected and agents could repeat this trial-and-error process. This immediate reward is typically depending on the situation and action. The current action will also have an impact on the reward for future subsequent steps. Here, the optimal outputs are not given like the past learning problems. Agents should discover them by a sequence of trial and error. The most important feature distinguishing reinforcement learning from other learning methods is to use the sequence of the training information to evaluate the action rather than instruct the correct action. Similar to the bias-variance trade-off, in reinforcement learning, there is another classical trade-off problem called exploration-and-exploitation trade-off. Exploitation is to maximize the expected reward based on current knowledge of values of the actions, while exploration is to collect information about unknown options. The exploration-and-exploitation trade-off in reinforcement learning matches the learning that humans and other animals do in life.

### 1.7.1 Multi-Armed Bandit

A (non-contextual) multi-armed bandit (MAB) problem is a classical learning problem in reinforcement learning. In MAB, agents are faced repeatedly with a choice among $k$ different options, or actions. After each choice, agents receive a numerical reward chosen from a stationary probability distribution that depends on the action taken. I prefer to use the multi-slot machine in Casino as an example. Assume one goes to Casino Night that has many slot machines to play. For each slot machine one plays, one can receive a certain payoff based on some unknown probability distribution. One's goal is to maximize payoff after

thousands of trials. The exploration vs exploitation dilemma arises in this scenario with incomplete information. In this case, pulling an arm that gives the highest payoff so far is exploitation, while exploring some other arms that might potentially turn out the better payoff is called exploration. MAB addresses the primary difficulty in sequential decision-making under uncertainty [Sutton and Barto, 1998]. Balancing exploitation-and-exploration trade-off is in terms of yielding the maximum total reward in the end. Bandit framework arises in multiple practical domains including clinical trials from health care, dynamic pricing from online retailers and information retrieval [Bouneffouf and Rish, 2019].

Let us see a short conclusion of the development of bandit strategies.

- No exploration: the most naive approach where agents always choose the same action without any exploration
- Agents make exploration at random

  - $\epsilon$-greedy action selection: with probability $\epsilon$, agents take a random action, with probability $(1-\epsilon)$ pick the current best action
- Exploration smartly with preference for uncertain environment, such as

  - Upper Confidence Bound (UCB) action selection:

$$A_t = \text{argmax}_a \left[ Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right] , \tag{1.10}$$

  where $Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$ , $\{R_i\}_{i=1}^{t-1}$ stands for the reward from round 1 to $t-1$, and $N_t(a)$ is the number of times that action $a$ has been selected prior to time $t$

  - UCB action selection: Actions with lower reward estimates, or that have already been selected frequently, will be selected with decreasing frequency over time

Note that the idea of UCB action selection strategy is a tool we use in chapter 2 and 3.

The above introduction is non-contextual bandit problem. This is non-associative setting, one that does not involve learning to act in more than one situation. In comparison to non-contextual setting, in contextual bandit, agents' decisions are made with the aid of situation (contextual) information and the corresponding reward depends on the state and action pair.

### 1.7.2 Q Learning

In reinforcement learning, policy is a map from state (circumstance) to action. The goal of $Q$-learning is to learn a policy that can maximize the expected value of the total future

reward over all successive steps starting from the current step [Watkins, 1989, Sutton and Barto, 1998]. Let us see the following example in $Q$-learning.

**Example 1.9.** A self-driving car needs to cross a map with obstacles such as trees and building blocks. The situations on the map are any possible location with/without obstacles. The actions can be moving forward, backward, turning left, and right. If the self-driving car hits obstacles, then it get certain penalties. While if it does not hit anything or successfully exits, then it gets reward. In this case, policy is the strategy it uses to achieve its goal. The goal of the car is to find the optimal policy and maximize the reward.

Here we look at a finite time horizon $Q$-learning problem with $H$ the number of steps in each episode. Denote by $\mathcal{S}$ a finite state space and $\mathcal{A}$ a finite action space. Let $(x, a) \in \mathcal{S} \times \mathcal{A}$ be the current state-action pair. $\mathcal{P}$ is the state transition function where the transitional probability from the current state-action pair in $\mathcal{S} \times \mathcal{A}$ to the next state is $x' \sim \mathcal{P}(\cdot|x, a)$. Let the policy function be $\pi : \mathcal{S} \to \mathcal{A}$. The reward function at step $h \in [H]$, is $r_h : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Denote by $V_h^\pi : \mathcal{S} \to \mathbb{R}$ the value function at step $h$ under policy $\pi$. $V_h^\pi$ stands for the expected sum of remaining rewards collected from $x_h = x$ to the end of episode given by policy $\pi$. Mathematically,

$$V_h^\pi(x) := \mathbb{E}\left[\sum_{h'=h}^{H} r_{h'}(x_{h'}, \pi_{h'}(x_{h'}))|x_h = x\right],$$

which is the expectation of the future rewards. Accordingly, we can define $Q_h^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as the $Q$-value function at step $h$ under policy $\pi$. This gives the expected sum of the remaining rewards starting from $x_h = x, a_h = a$ to the end of episode. Mathematically,

$$Q_h^\pi(x, a) := r_h(x, a) + \mathbb{E}\left[\sum_{h'=h+1}^{H} r_{h'}(x_{h'}, \pi_{h'}(x_{h'}))|x_h = x, a_h = a\right].$$

There exists an optimal policy $\pi^*$ which gives the optimal value $V_h^*(x) = \sup_\pi V_h^\pi(x)$ [Azar et al., 2017]. Note that the Bellman equations for any (deterministic) policy $\pi$ is

$$\begin{cases} V_h^\pi(x) = Q_h^\pi(x, \pi(x)) \\ Q_h^\pi(x, a) = r_h(x, a) + \mathbb{E}_{x' \sim P(\cdot|x,a)} V_{h+1}(x'). \end{cases}$$

Denote by $V^*$ and $Q^*$ the optimal value and $Q$-value functions associated with the optimal

policy. Then the Bellman optimality equation is

$$\begin{cases} V_h^*(x) = \max_{a\in\mathcal{A}} Q_h^*(x,a) \\ Q_h^*(x,a) = r_h(x,a) + \mathbb{E}_{x'\sim P(\cdot|x,a)} V_{h+1}^*(x') \,. \end{cases} \tag{1.11}$$

The goal is to find the optimal $Q$-values, $Q^*$. Since there is no best action to start with, we always pick randomly. We would also not find the optimal strategy immediately without the knowledge of higher $Q$-value. The updating of $Q$-value function is an iterative process. With trial and error in numbers of episodes, agents can explore enough steps and episodes and exploit the learning. With the continuing exploration of the environment and update on the $Q$-values, $Q$-value function gives us better approximations.

It is worth noting that the above setting can easily be extended to the infinite time horizon $Q$-learning with discount factor at each step. We omit the discussion here and readers can find more details in chapter 3 and in [Watkins, 1989, Sutton and Barto, 1998].

## 1.8 Thesis Overview

Machine learning is one of the most fast-growing areas in the $21^{st}$ century. A variety of deep learning subjects and sub-fields have more than tens of thousands of publications annually. Today, in conferences and journals in the fields of learning representation, natural language processing, computer vision, and robotics, computer scientists have the trend of publishing purely empirical research work with no or very little theory supported. In scientific research and industry with production, many people are concerned about the potential risks since they require the explainability and interpretability of models due to trust and informativeness. To tackle these challenges, more mathematicians and statisticians get attracted to the theoretical foundation of machine learning and try to make machine learning more explainable and controllable. This satisfies my research interests while I always believe machine learning is an empirical science in need of support in theory and reasoning. My orientation is to develop methods with theoretical foundations, the tractability of reasoning, and practical values. However, one limitation of many theoretical works is that many researchers convert the problems into purely theoretical problems in their areas, and focus more on elaborate techniques for solutions. As the one who treats machine learning primarily as an empirical science, I add my thoughts in practice and extend the work into real-world challenging applications.

The statistician George Box has a famous aphorism: "All the models are wrong, but some are useful." In scientific research, no matter what kind of models we choose, these models are approximations and hence have their advantages/disadvantages. This is equivalent to

the classical trade-off concept in machine learning. As a machine learning researcher who cares about the theoretical foundation and interpretability of models, my works inherit the classical trade-off philosophy in machine learning. The methods and algorithms we propose in the thesis balance trade-offs including exploration-exploitation in bandits, error-reject in selective prediction, and bias-variance in generalization.

This thesis consists of three parts involving two areas in the field of machine learning: deep learning and reinforcement learning. Getting motivated by my advisor Professor Shlomo Ta'asan, I am interested in the most popular, challenging, and meaningful problems in the current machine learning challenges. For example, tuning hyperparameters is a crucial research problem in deep learning that appears in every single model. An optimal combination of hyperparameters can improve the models' generalization significantly. Although the deep neural network with high complexity can fit the large-scale data very well, it contains plenty of hyperparameters. It is challenging to find the best combination of hyperparameters for the given setting and model architecture. In chapter 2, we develop a novel stochastic Lipschitz bandit method that adaptively learns partitions of context- and arm-space. Due to this flexibility, the algorithm is able to efficiently optimize rewards and minimize regret. Our algorithm can achieve state-of-the-art performance in challenging real-world tasks such as neural network hyperparameter tuning.

The decision-making process of the robotics under uncertainty is one of the most growing learning applications. Applications like self-driving cars and drones require sophisticated Reinforcement Learning (RL) methods to perform well and intelligibly. Compared to model-based RL approaches, model-free RL algorithms, such as $Q$-learning, require less space and are more expressive since specifying value functions or policies is more flexible than specifying the model for the environment. Upper Confidence Bound (UCB) exploration policy improves the exploration bonuses in the $Q$-learning framework. In chapter 3, we develop a $Q$-learning algorithm with UCB exploration policy in the scenario of compact state-action metric space. Our algorithm can adaptively discretize the continuous state-action space, iteratively update $Q$-values, and efficiently minimize the cumulative regret (error). Our algorithm can improve data efficiency while leveraging the advantages of model-free RL methods in a general state-action metric space.

Also, image and pattern recognition problems are considered as the major application in deep learning. Although large-scale data are easily accessible nowadays, data in many applications contain various kinds of noise. Low-quality data will hinder the performance of the learning system in terms of the classification accuracy, model building time, and interpretability of the classifier. In chapter 4, we propose noise-robust learning algorithms integrated with selective prediction that can effectively estimate the intrinsic rejection rate of

the dataset. Then we generalize our method to decision thresholds learning for asymmetrical costs binary classification. Correspondingly, we provide a rigorous framework to generalize the estimation of data corruption rate. To leverage the advantage of multiple learning methods through an ensemble, we extend learning algorithms to a hierarchical two-stage system. Empirical results show that our methods can achieve state-of-the-art performance in many classification and regression problems.

# 2 Practical Lipschitz Stochastic Bandits [1]

Stochastic Lipschitz bandit algorithms are methods that govern exploration-exploitation trade-offs and have been used for a variety of important task domains, including zeroth-order optimization. While the beautiful theory has been developed for the stochastic Lipschitz bandit problem, the methods arising from these theories are not practical, and accordingly, the development of practical well-performing Lipschitz bandit algorithms has stalled in recent years. In this chapter, we present a framework for Lipschitz bandit methods that adaptively learns partitions of context- and arm-space. Due to this flexibility, the algorithm is able to efficiently optimize rewards and minimize regret, by focusing on the most relevant portions of the space. Our experiments show that (1) using adaptively-learned partitioning, our method can surpass existing stochastic Lipschitz bandit algorithms, and (2) our algorithms can achieve state-of-the-art performance in challenging real-world tasks such as neural network hyperparameter tuning.

## 2.1 Introduction

A stochastic bandit problem assumes that the expected payoff is a Lipschitz function of the arm and payoffs are noisy and are drawn from an unchanging distribution. Stochastic Lipschitz bandit algorithms, as methods that leverage exploration-exploitation trade-offs using bandit strategies, span important task domains including zeroth-order optimization. These tasks arise in important real-world scenarios. For example, in medical trials, a doctor might deliver a sequence of treatment options to achieve the best total treatment effect or to allocate the best treatment option as efficiently as possible, without conducting too many trials.

The study of stochastic bandit problems started with the discrete arm setting, where the agent is faced with a finite set of choices. Classic works on this problem include Thompson sampling [Thompson, 1933, Agrawal and Goyal, 2012], Gittins index [Gittins, 1979], $\epsilon$-greedy strategies [Sutton and Barto, 1998], and UCB methods [Lai and Robbins, 1985, Agrawal, 1995, Auer et al., 2002]. These bandit strategies have led to powerful real-life applications. For example, Deep Q-Network [Mnih et al., 2015] uses $\epsilon$-greedy for action exploration; and AlphaGO [Silver et al., 2016] uses UCT [Kocsis and Szepesvári, 2006], which is built on the UCB strategy. One recent line of work on stochastic bandit problems considers the case where the arm space is infinite. In this setting, the arms are usually assumed to be in a subset of the Euclidean space (or a more general metric space), and the expected payoff function is

[1] This chapter is based on the joint work with Tianyu Wang, Dawei Geng, and Dr. Cynthia Rudin. The author particularly acknowledges Tianyu Wang for his creativity in the problem setup. Some results and analysis are developed based on the collaboration and discussion with Tianyu Wang.

assumed to be a function of the arms. Some works along this line model the expected payoff as a linear function of the arms [Auer, 2002, Dani et al., 2008, Li et al., 2010, Abbasi-Yadkori et al., 2011, Agrawal and Goyal, 2012, Abeille and Lazaric, 2017]; some algorithms model the expected payoff as Gaussian processes over the arms [Srinivas et al., 2009, Contal et al., 2014, De Freitas et al., 2012, Vazquez and Bect, 2007]; some algorithms assume that the expected payoff is a Lipschitz function of the arms [Slivkins, 2014, Kleinberg et al., 2008, Bubeck et al., 2011, Magureanu et al., 2014]; and some assume locally Hölder payoffs on the real line [Auer et al., 2007]. When the arms are continuous and equipped with a metric, and the expected payoff is Lipschitz continuous in the arm space, we refer to the problem as a stochastic Lipschitz bandit problem. Also, when the agent's decisions are made with the aid of contextual information, we refer to the problem as a contextual stochastic Lipschitz bandit problem. In this chapter, we focus our study on the (contextual) stochastic Lipschitz bandit problem. While our analysis focuses on the case of Lipschitz expected payoffs, our empirical results do not. They demonstrate that our methods can adapt to the complicated landscape of real-world payoff functions, even on discrete domains, and leverage properties other than Lipschitzness. This means our methods have much better practical performance than analyses for Lipschitz bandit problems might suggest.

To accomplish our goals, we propose a framework that converts a general decision tree algorithm into an algorithm for stochastic Lipschitz bandit problems. In Theorem 2.3, we show that as long as the decision tree algorithm satisfies certain conditions, the converted bandit algorithm asymptotically admits zero average regrets, since a decision tree algorithm flexibly fits the observed (context-) arm reward tuples. Our methods can empirically outperform existing benchmark methods for stochastic Lipschitz bandit problems.

**Relation to prior work:** One general way of solving stochastic Lipschitz bandit problems is to finely discretize (partition) the arm space and treat the problem as a finite-arm problem. An Upper Confidence Bound (UCB) strategy can thus be used. Previous algorithms of this kind include the `UniformMesh` algorithm [Kleinberg et al., 2008], the HOO algorithm [Bubeck et al., 2011], and the (contextual) Zooming Bandit algorithm [Kleinberg et al., 2008, Slivkins, 2014]. While all these algorithms employ different analysis techniques, we show that as long as a discretization of the arm space fulfills certain requirements (outlined in Theorem 2.3), these algorithms (or a possibly modified version) can be analyzed in a unified framework.

The practical problem with previous methods is that they require either a fine discretization of the full arm space or a restrictive control of the partition formation, leading to implementations that are not flexible. By using decision trees that are grown adaptively during the run of the algorithm, our partition is essentially *learned*. This advantage enables the algorithm to (1) outperform the HOO [Bubeck et al., 2011] algorithm, the (contextual) Zooming bandit

[Kleinberg et al., 2008, Slivkins, 2014] and Contextual Combined KL-UCB (CCKL-UCB) [Magureanu et al., 2014] algorithms on benchmark tasks, and (2) be competitive on hard real-world problems that can involve difficult arm space and reward landscape. As shown in the experiments, in neural network hyperparameter tuning, our methods can outperform the state-of-the-art benchmarks that are tailored for hyperparameter selection.

In summary, our contributions are: **1)** We develop a novel stochastic Lipschitz bandit framework, *TreeUCB* and its contextual counterpart *Contextual TreeUCB*. Our framework converts a general decision tree algorithm into a stochastic Lipschitz bandit algorithm. Algorithms arising from this framework empirically outperform benchmarks methods. **2)** As a byproduct, we use our analysis framework to derive previous known regret bounds via unified arguments.

We begin by introducing the proposed framework: TreeUCB. We then analyze the framework and provide its contextual counterpart: Contextual TreeUCB.

## 2.2 The TreeUCB framework

The goal of a stochastic bandit algorithm is to locate the global maximum (minimum) of a payoff function in as few iterations as possible, balancing exploration and exploitation. The performance of the algorithm is typically measured by what is called regret. In this section, we focus our study on the following setting. A payoff function is defined over an arm space that is a compact metric space $(\mathcal{A}, d)$, the payoff function of interest is $f : \mathcal{A} \to [0, 1]$, and the actual observations are given by $y(a) = f(a) + \epsilon_a$. In our setting, the noise distribution $\epsilon_a$ could vary with $a$, as long as it is uniformly mean zero, almost surely bounded, and independent of $f$ for every $a$. Our results easily generalize to sub-Gaussian noise [Shamir, 2011]. In the analysis, we assume that the (expected) payoff function $f$ is Lipschitz, i.e., $\forall a', a'' \in \mathcal{A}$, $|f(a') - f(a'')| \leq L \cdot d(a', a'')$ for some Lipschitz constant $L$. An agent is interacting with this environment in the following fashion. At each round $t$, based on past observations $(a_1, y_1, \cdots, a_{t-1}, y_{t-1})$, the agent makes a query at point $a_t$ and observes the (noisy) payoff $y_t$, where $y_t$ is revealed only after the agent has made a decision $a_t$. For an agent executing algorithm `Alg`, the regret incurred up to time $T$ is defined to be:

$$R_T(\texttt{Alg}) = \sum_{t=1}^{T} \left( f(a^*) - f(a_t) \right) ,$$

where $a^*$ is the global maximizer of $f$.

### 2.2.1 Partition to UCB Strategy

Any TreeUCB algorithm runs by maintaining a sequence of finite partitions of the arm space. Intuitively, at each step $t$, TreeUCB treats the problem as a finite-arm bandit problem with respect to the partition bins at $t$, and chooses an arm uniformly at random within the chosen bin. The partition bins become smaller and smaller as the algorithm runs. Thus, at any time $t$, we maintain a partition $\mathcal{P}_t = \left\{P_t^{(1)}, \cdots, P_t^{(k_t)}\right\}$ of the input space. That is, $P_t^{(1)}, \cdots, P_t^{(k_t)}$ are subsets of $\mathcal{A}$, are mutually disjoint and $\cup_{i=1}^{k_t} P_t^{(i)} = \mathcal{A}$. Each element in the partition is called a *region* and by convention $\mathcal{P}_0 = \{\mathcal{A}\}$. The regions could be leaves in a tree, or chosen in some other way.

Given any $t$, if for any $P^{(i)} \in \mathcal{P}_{t+1}$, there exists $P^{(j)} \in \mathcal{P}_t$ such that $P^{(i)} \subset P^{(j)}$, we say that $\{\mathcal{P}_t\}_{t\geq 0}$ is a sequence of ***nested partitions.*** In words, at round $t$, some regions (or no regions) of the partition are split into multiple regions to form the partition at round $t+1$. We also say that the partition ***grows finer.***

### 2.2.2 The TreeUCB algorithm

Based on the partition $\mathcal{P}_t$ at time $t$, we define an auxiliary function – the *Region Selection function.*

**Definition 2.1** (Region Selection Function)**.** Given partition $\mathcal{P}_t$, a function $p_t : \mathcal{A} \to \mathcal{P}_t$ is called a Region Selection Function with respect to $\mathcal{P}_t$ if for any $a \in \mathcal{A}$, $p_t(a)$ is the region in $\mathcal{P}_t$ containing $a$.

As the name TreeUCB suggests, our algorithm is an Upper Confidence Bound (UCB) strategy. In order to define our Upper Confidence Bound, we require several definitions.

**Definition 2.2.** Let $\mathcal{P}_t$ be the partition of $\mathcal{A}$ at time $t$ ($t \geq 1$) and let $p_t$ be the Region Selection Function associated with $\mathcal{P}_t$. Let $(a_1, y_1, a_2, y_2, \cdots, a_{t'}, y_{t'})$ be the observations received up to time $t'$ ($t' \geq 1$). We define

- The *count* function $n^0_{t,t'} : \mathcal{A} \to \mathbb{R}$, such that $n^0_{t,t'}(a) = \sum_{i=1}^{t'} \mathbb{1}[a_i \in p_t(a)]$.
- The *corrected average* function $m_{t,t'} : \mathcal{A} \to \mathbb{R}$, such that

$$m_{t,t'}(a) = \begin{cases} \frac{\sum_{i=1}^{t'} y_i \mathbb{1}[a_i \in p_t(a)]}{n^0_{t,t'}(a)}, & \text{if } n^0_{t,t'}(a) > 0\,; \\ 1, \quad \text{otherwise}\,. \end{cases} \tag{2.1}$$

- The *corrected count* function, such that

$$n_{t,t'}(a) = \max\left(1, n^0_{t,t'}(a)\right)\,. \tag{2.2}$$

When $t = t'$, we shorten the notation from $m_{t,t'}$ to $m_t$, $n^0_{t,t'}$ to $n^0_t$, and $n_{t,t'}$ to $n_t$.

In words, $n^0_{t,t'}(a)$ is the number of points among $(a_1, a_2, \cdots, a_{t'})$ that are in the same region as arm $a$, with regions as elements in $\mathcal{P}_t$. We also denote by $D(\mathcal{S})$ the diameter of $\mathcal{S} \subset \mathcal{A}$, and $D(\mathcal{S}) := \sup_{a',a'' \in \mathcal{S}} d(a', a'')$.

At time $t$, based on the partition $\mathcal{P}_{t-1}$ and observations $(a_1, y_1, a_2, y_2, \cdots, a_{t-1}, y_{t-1})$, our bandit algorithm uses, for $a \in \mathcal{A}$,

$$U_t(a) = m_{t-1}(a) + C\sqrt{\frac{4 \log t}{n_{t-1}(a)}} + L \cdot D(p_{t-1}(a)) \tag{2.3}$$

for some $C$ and $L$ as the Upper Confidence Bound of arm $a$; and we play an arm with the highest $U_t$ value (with ties broken uniformly at random). Here $C$ depends on the almost sure bound on the reward, and $L$ is the Lipschitz constant of the expected payoff function, which are both problem intrinsics.

Since $U_t$ is a piece-wise constant function in the arm-space and is constant within each region, playing an arm with the highest $U_t$ with random tie-breaking is equivalent to selecting the best region (under UCB) and randomly selecting an arm within the region. After deciding which arm to play,

**Algorithm 1** TreeUCB (TUCB)

1: Parameter: $L \geq 0$. $C > 0$. Tree fitting rule $\mathcal{R}$ that satisfies 1–4 in Theorem 2.3.
▷ $C$ depends on the a.s. bound of the reward.
▷ $L$ is the Lipschitz constant of the expected payoff function.
2: **for** $t = 1, 2, \ldots, N$ **do**
3: Fit the tree $f_{t-1}$ using rule $\mathcal{R}$ on observations $(a_1, y_1, a_2, y_2, \ldots, a_{t-1}, y_{t-1})$.
4: With respect to the partition $\mathcal{P}_{t-1}$ defined by leaves of $f_{t-1}$, define $m_{t-1}$, $n_{t-1}$ as in (2.1) and (2.2). Play

$$a_t \in \arg\max_{a \in \mathcal{A}} \{U_t(a)\}, \tag{2.4}$$

where $U_t$ is defined in (2.3). Ties are broken uniformly at random.
5: Observe the reward $y_t$.

**Theorem 2.3.** *Suppose that the payoff function $f$ defined on a compact domain $\mathcal{A}$ satisfies $f(a) \in [0, 1]$ for all $a$ and is Lipschitz. Let $\mathcal{P}_t$ be the partition at time $t$ in Algorithm 1. If the tree fitting rule $\mathcal{R}$ satisfies*

1. *$\{\mathcal{P}_t\}_{t \geq 0}$ is a sequence of nested partitions (or the partition grows finer);*
2. *$|\mathcal{P}_t| = o(t^\gamma)$, as $t \to \infty$, for some $\gamma < 1$;*

3. $D(p_t(a)) = o(1)$, *as* $t \to \infty$, *for all* $a \in \mathcal{A}$, *where* $D(p_t(a)) := \sup_{a'',a' \in p_t(a)} d(a'', a')$ *is the diameter of region* $p_t(a)$;

4. *given all realized observations* $\{(a_t, y_t)\}_{t=1}^T$, *the partitions* $\mathcal{P}_1, \cdots, \mathcal{P}_T$ *are deterministic;*

*then the regret for Algorithm 1 satisfies* $\lim_{T\to\infty} \frac{R_T(\textit{TUCB})}{T} = 0$ *with probability 1.*

The proof of Theorem 2.3 is presented in later sections. The above assumptions are all mild and reasonable. For item 1, we can use incremental tree learning [Utgoff, 1989] to enforce nested partitions. For item 2, we may put a cap (that may depend on $t$) on the depth of the tree to constrain it. For item 3, we may put a cap (that may depend on $t$) on tree leaf diameters to ensure it. For item 4, any non-random tree learning rule meets these criteria, since in this case, the randomness only comes from the data (and/or number of data points observed).

**Adaptive partitioning**

TreeUCB shall be implemented using regression trees or incremental regression trees. This naturally leverages the practical advantages of regression trees. Leaves in a regression tree form a partition of the space. Also, a regression tree is designed to fit an underlying function. This leads to an adaptive partitioning where the underlying function values within each region should be relatively similar to each other. We defer the discussion on the implementation we use in our experiments to section 2.5.1. Please refer to [Breiman et al., 1984] for more details about regression tree fitting.

### 2.2.3 The Contextual TreeUCB algorithm

Next, we discuss the contextual counterpart of the TreeUCB algorithm. We present an extension of Algorithm 1 for the contextual stochastic bandit problem. The contextual stochastic bandit problem is an extension of the stochastic bandit problem. In this problem, at each time, context information is revealed, and the agent chooses an arm based on past experience as well as the contextual information. Formally, the expected payoff function $f$ is defined over the product of the context space $\mathcal{Z}$ and the arm space $\mathcal{A}$ and takes values from $[0, 1]$. Similar to the previous discussions, compactness of the product space and Lipschitzness of the payoff function is assumed. Besides, a mean zero, almost surely bounded noise that is independent of the expected reward function is added to the observed rewards.

Same as in section 2.2.2, we can define the corresponding *Region Selection function* for the contextual version (over the joint space $\mathcal{Z} \times \mathcal{A}$).

**Definition 2.4** (Region Selection Function)**.** Given partition $\mathcal{P}_t$, for a fixed $z_t \in \mathcal{Z}$, function $p_t : \mathcal{Z} \times \mathcal{A} \to \mathcal{P}_t$ is called a Region Selection Function with respect to $\mathcal{P}_t$ and $z_t$ if for any $a \in \mathcal{A}$, $p_t(z_t, a)$ is the region in $\mathcal{P}_t$ containing $(z_t, a)$.

We also require several corresponding definitions in the contextual version to define the contextual version of the UCB strategy.

**Definition 2.5.** Let $\mathcal{P}_t$ be the partition of $\mathcal{Z} \times \mathcal{A}$ at time $t$ ($t \geq 1$) and let $p_t$ be the Region Selection Function associated with $\mathcal{P}_t$ and $z_t$. Let $\{(z_i, a_i), y_i\}_{i=1}^{t'}$ be the observations received up to time $t'$ ($t' \geq 1$). We define

- The *count* function $n^0_{t,t'} : \mathcal{Z} \times \mathcal{A} \to \mathbb{R}$, such that $n^0_{t,t'}(z_t, a) = \sum_{i=1}^{t'} \mathbb{1}[(z_i, a_i) \in p_t(z_t, a)]$.

- The *corrected average* function $m_{t,t'} : \mathcal{Z} \times \mathcal{A} \to \mathbb{R}$, such that

$$m_{t,t'}(a) = \begin{cases} \frac{\sum_{i=1}^{t'} y_i \mathbb{1}[(z_i,a_i) \in p_t(z_t,a)]}{n^0_{t,t'}(z_t,a)}, & \text{if } n^0_{t,t'}(z_t, a) > 0\,; \\ 1, \quad \text{otherwise}\,. \end{cases} \tag{2.5}$$

- The *corrected count* function, such that

$$n_{t,t'}(z_t, a) = \max\left(1, n^0_{t,t'}(z_t, a)\right)\,. \tag{2.6}$$

  Similarly, when $t = t'$, we shorten the notation from $m_{t,t'}$ to $m_t$, $n^0_{t,t'}$ to $n^0_t$, and $n_{t,t'}$ to $n_t$.

Now, we can define the contextual version of the UCB strategy similar as (2.3). At time $t$, based on the partition $\mathcal{P}_{t-1}$ and observations $\{(z_i, a_i), y_i\}_{i=1}^{t-1}$, for fixed $z_t$, our bandit algorithm uses, for $a \in \mathcal{A}$,

$$U_t(z_t, a) = m_{t-1}(z_t, a) + C\sqrt{\frac{4 \log t}{n_{t-1}(z_t, a)}} + L \cdot D(p_{t-1}(z_t, a))\,, \tag{2.7}$$

for same $C$ and $L$ as before. We again play an arm with the highest $U_t$ value, which is equivalent to selecting the best region (under UCB) and randomly selecting an arm within the region.

**Algorithm 2** Contextual TreeUCB (CTUCB)

1: Parameter: $L > 0$, $C > 0$, and tree fitting rule $\mathcal{R}$.
2: **for** $t = 1, 2, \ldots, N$ **do**
3: Observe context $z_t$.
4: Fit a regression tree $f_{t-1}$ (using rule $\mathcal{R}$) on observations $\{(z_i, a_i), y_i\}_{i=1}^{t-1}$ .
5: With respect to the partition $\mathcal{P}_{t-1}$ defined by leaves of $f_{t-1}$, define $m_{t-1}$ and $n_{t-1}$ in (2.5) and (2.6) (over the joint space $\mathcal{Z} \times \mathcal{A}$). Play

$$a_t \in \arg\max_{a \in \mathcal{A}} \{U_t((z_t, a))\} ,$$

where $U_t(\cdot)$ is defined in (2.7). Ties are broken at random.
6: Observe the reward $y_t$.

At each time $t$, a contextual vector $z_t \in \mathcal{Z}$ is revealed and the agent plays an arm $a_t \in \mathcal{A}$. The performance of the agent following algorithm `Alg` is measured by the cumulative contextual regret

$$R_T^c(\texttt{Alg}) = \sum_{t=1}^{T} f(z_t, a_t^*) - f(z_t, a_t) , \tag{2.8}$$

where $f(z_t, a_t^*)$ is the maximal value of $f$ given contextual information $z_t$. A simple extension of Algorithm 1 can solve the contextual version problem. *In particular, in the contextual case, we partition the joint space $\mathcal{Z} \times \mathcal{A}$ instead of the arm space $\mathcal{A}$.* As in (2.5) and (2.6), we define the corrected count $n_t$ and the corrected average $m_t$ over the joint space $\mathcal{Z} \times \mathcal{A}$ with respect to $\mathcal{P}_t$, $z_t$, and observations in the joint space $((z_1, a_1), y_1, \cdots, (z_t, a_t), y_t)$. The guarantee of Algorithm 2 is in Theorem 2.6.

**Theorem 2.6.** *Suppose that the payoff function $f$ defined on a compact metric space $(\mathcal{Z} \times \mathcal{A}, d)$ satisfies $f(z, a) \in [0, 1]$ for all $(z, a)$ and is Lipschitz. If the tree growing rule satisfies assumptions 1-4 listed in Theorem 2.3, then $\lim_{T\to\infty} \frac{R_T^c(\texttt{CTUCB})}{T} = 0$ with probability 1.*

Theorem 2.6 is the contextual version of Theorem 2.3. In the next section, we discuss the analysis of the proof of Theorem 2.3 and 2.6 by using additional claims and lemma.

## 2.3 Main Analysis

We use $\tilde{\mathcal{O}}$ to hide poly-log terms unless otherwise noted.

### Outline

The proof of Theorem 2.3 and 2.6 are in fact the same since all the components in the proof are equivalent. As an outline, we first prove Propositions 2.7 and 2.8 to bound the single step regret. Both of the propositions are proved in the contextual version. The analysis can simply be reduced to the non-contextual version when the context is set to a fixed value. Then we prove Lemma 2.9, the point scattering inequalities. Since the point scattering inequalities hold for any sequence of (context-)arms, we have Lemma 2.9 true for both non-contextual and contextual settings. If we combine Propositions 2.7, 2.8 and Lemma 2.9 with assumptions (1) – (3) in Theorem 2.3 and 2.6, we can hence conclude Theorem 2.3 and 2.6. Thus, the regret holds for both non-contextual and contextual stochastic Lipschitz bandit problem.

### 2.3.1 Supported Propositions

Below, we state and prove Proposition 2.7, a contextual version of concentration. This naturally gives us the non-contextual version of concentration.

**Proposition 2.7.** *For any context $z$, arm $a$, and time $t$, with probability at least $1 - \frac{1}{t^4}$, we have:*

$$|m_{t-1}(z,a) - f(z,a)| \leq L \cdot D(p_{t-1}(z,a)) + C\sqrt{\frac{4\log t}{n_{t-1}(z,a)}} \tag{2.9}$$

*for a constant $C$.*

*Proof.* The proof of Proposition 2.7 uses Lipschitzness and the Azuma-Hoeffding inequality for stopped martingales [Bubeck et al., 2011]. First of all, when $t = 1$, this is trivially true by Lipschitzness. Now let us consider the case when $t \geq 2$. Let us use $A_1, A_2, \cdots, A_t$ to denote the random variables of arms selected up to time $t$, $Z_1, Z_2, \cdots, Z_t$ to denote the random context up to time $t$, and $Y_1, Y_2, \cdots, Y_t$ to denote random variables of rewards received up to time $t$. Then the random variables $\left\{\sum_{t=1}^{T}\left(f(Z_t, A_t) - Y_t\right)\right\}$ is a martingale sequence. This is easy to verify since the noise is mean zero and independent. In addition, since there is no randomness in the partition formation (given a sequence of observations), for a fixed $a$, we have the times $\mathbb{1}[(Z_i, A_i) \in p_{t-1}(z,a)]$ $(i \leq t)$ is measureable with respect to $\sigma(Z_1, A_1, Y_1, \cdots, Z_t, A_t, Y_t)$. Therefore, the sequence $\left\{\sum_{i=1}^{t}\left(f(Z_i, A_i) - Y_i\right)\mathbb{1}[(Z_i, A_i) \in p_{t-1}(z,a)]\right\}_{t=1}^{T}$ is a stopped martingale. Since stopped martingale is also a martingale, we apply the Azuma-Hoeffding

inequality (with sub-Gaussian tails) [Shamir, 2011]. For simplicity, we write

$$B_t(z,a) := C\sqrt{\frac{4\log t}{n_{t-1}(z,a)}} + L \cdot D(p_{t-1}(z,a))\,, \tag{2.10}$$

$$\mathcal{E}_t^i(z,a) := (Z_i, A_i) \in p_{t-1}(z,a)\,. \tag{2.11}$$

Combined with Lipschitz continuity, we get there is a constant $C$ (depends on the a.s. bound of the reward, as a result of Hoeffding inequality). From Azuma-Hoeffding's inequality, we have

$$\begin{aligned}
&\mathcal{P}\left\{\frac{1}{n_{t-1}(z,a)}\sum_{i=1}^{n_{t-1}(z,a)}\left|(f(Z_i,A_i)-Y_i)\,\mathbb{1}[\mathcal{E}_t^i(z,a)]\right| \geq \epsilon\right\} \\
&= \mathcal{P}\left\{\sum_{i=1}^{n_{t-1}(z,a)}\left|(f(Z_i,A_i)-Y_i)\,\mathbb{1}[\mathcal{E}_t^i(z,a)]\right| \geq \epsilon \cdot n_{t-1}(z,a)\right\} \\
&\leq 2\exp\left\{-\frac{{n_{t-1}}^2\epsilon^2}{2\cdot n_{t-1}\cdot C_t}\right\}.
\end{aligned}$$

If $\epsilon = C\sqrt{\frac{4\log t}{n_{t-1}}} = \sqrt{\frac{8C_t\log t}{n_{t-1}}}$, then $2\exp\left\{-\frac{{n_{t-1}}^2\epsilon^2}{2\cdot n_{t-1}\cdot C_t}\right\} = \frac{1}{t^4}$. This implies that

$$\mathcal{P}\left\{\frac{1}{n_{t-1}(z,a)}\sum_{i=1}^{n_{t-1}(z,a)}\left|(f(Z_i,A_i)-Y_i)\,\mathbb{1}[\mathcal{E}_t^i(z,a)]\right| \geq C\sqrt{\frac{4\log t}{n_{t-1}(z,a)}}\right\} \leq \frac{1}{t^4}\,. \tag{2.12}$$

In addition, since by the Lipschitz continuity,

$$\begin{aligned}
\left|f(z,a) - \frac{1}{n_{t-1}(z,a)}\sum_{i=1}^{t-1} f(Z_i,A_i)\mathbb{1}[\mathcal{E}_t^i(z,a)]\right| &\leq \frac{1}{n_{t-1}(z,a)}\sum_{i=1}^{n_{t-1}(z,a)}\left|f(z,a) - f(Z_i,A_i)\mathbb{1}[\mathcal{E}_t^i(z,a)]\right| \\
&\leq L\cdot D(p_{t-1}(z,a))\,.
\end{aligned} \tag{2.13}$$

Combining (2.12) and (2.13), we have

$$\begin{aligned}
&\mathcal{P}\left\{|m_{t-1}(z,a) - f(z,a)| > B_t(z,a)\right\} \\
&\leq \mathcal{P}\left\{\left|\frac{1}{n_{t-1}(z,a)}\sum_{i=1}^{n_{t-1}(z,a)}\left(f(Z_i,A_i) - Y_i\right)\mathbb{1}[\mathcal{E}_t^i(z,a)]\right|\right. \\
&\quad + \left|f(z,a) - \frac{1}{n_{t-1}(z,a)}\sum_{i=1}^{n_{t-1}(z,a)} f(Z_i,A_i)\mathbb{1}[\mathcal{E}_t^i(z,a)]\right| \\
&\quad \left.> C\sqrt{\frac{4\log t}{n_{t-1}(z,a)}} + L\cdot D(p_{t-1}(z,a))\right\} \\
&\leq \frac{1}{t^4}\,. \qquad (2.14)
\end{aligned}$$

□

Next, we prove Proposition 2.8, a contextual version of single-step regret. The proof of Proposition 2.8 uses the definition of the algorithm and Proposition 2.7. Same as Proposition 2.7, this naturally gives us the non-contextual version.

**Proposition 2.8.** *At any $t$, with probability at least $1 - \frac{1}{t^4}$, the single step contextual regret satisfies:*

$$f(z_t, a_t^*) - f(z_t, a_t) \leq 2L\cdot D(p_{t-1}(z_t,a_t)) + 2C\sqrt{\frac{4\log t}{n_{t-1}(z_t,a_t)}} \qquad (2.15)$$

*for a constant $C$. Here $a_t^*$ is the optimal arm for the context $z_t$.*

*Proof.* Since we always play the arm with highest $U_t$, then for the choice of $a_t$ corresponding to $z_t$ at $t$, we have

$$U_t(z_t,a_t) \geq U_t(z_t,a_t^*)\,, \qquad (2.16)$$

where $U_t$ is the context version of UCB strategy shown in (2.7).

Combining (2.16) and Proposition 2.7, then with probability at least $1 - \frac{1}{t^4}$, we have the

following (2.17) to (2.19) hold simultaneously,

$$\begin{aligned} & m_{t-1}(z_t, a_t) + C\sqrt{\frac{4\log t}{n_{t-1}(z_t, a_t)}} + L \cdot D(p_{t-1}(z_t, a_t)) \\ \geq\ & m_{t-1}(z_t, a_t^*) + C\sqrt{\frac{4\log t}{n_{t-1}(z_t, a_t^*)}} + L \cdot D(p_{t-1}(z_t, a_t^*)) \qquad (2.17) \\ \geq\ & f(z_t, a_t^*)\,, \qquad (2.18) \end{aligned}$$

$$\begin{aligned} f(z_t, a_t) \geq\ & m_{t-1}(z_t, a_t) - C\sqrt{\frac{4\log t}{n_{t-1}(z_t, a_t)}} \\ & - L \cdot D(p_{t-1}(z_t, a_t))\,. \qquad (2.19) \end{aligned}$$

This is equivalent to

$$\begin{aligned} -f(z_t, a_t) \leq\ & -m_{t-1}(z_t, a_t) + C\sqrt{\frac{4\log t}{n_{t-1}(z_t, a_t)}} \\ & + L \cdot D(p_{t-1}(z_t, a_t))\,. \qquad (2.20) \end{aligned}$$

This is true since we first take a one-sided version of Hoeffding-type tail bound in (2.9), and then take a union bound over the two points $(z_t, a_t)$ and $(z_t, a_t^*)$. By doing these two steps, we first halve the probability bound and then double it. Then we take the complementary event to get (2.18) and (2.19) simultaneously hold with probability at least $1 - \frac{1}{t^4}$. We then take another union bound over time $t$, as discussed in the main text. Note that throughout the proof, we do not need to take union bounds over all arms or all regions in the partition.

(2.17) above holds from the UCB algorithm definition. Otherwise we will not select $a_t$ at time $t$. Combining (2.18) and (2.20), we get

$$\begin{aligned} & f(z_t, a_t^*) - f(z_t, a_t) \\ =\ & f(z_t, a_t^*) - m_{t-1}(z_t, a_t) \\ & + m_{t-1}(z_t, a_t) - f(z_t, a_t) \\ \leq\ & 2C\sqrt{\frac{4\log t}{n_{t-1}(z_t, a_t)}} + 2L \cdot D(p_{t-1}(z_t, a_t))\,. \end{aligned}$$

□

For the non-contextual version of Proposition 2.8, we set all $z_t$ to a fixed value.

### 2.3.2 Point Scattering Inequalities

While the tree fitting rule $\mathcal{R}$ may be specified to control $D(p_{t-1}(a_t))$ and $|\mathcal{P}_t|$, the rule does not directly control $\sum_{t=1}^{T} \frac{1}{n_{t-1}(a_t)}$. In addition, as the tree (partition) grows finer, the term $n_{t-1}(a)$ is not necessarily increasing with $t$ (for an arbitrary fixed $a$). Therefore part of the difficulty is in bounding $\sum_{t=1}^{T} \frac{1}{n_{t-1}(a_t)}$. Next, we introduce the point scattering inequalities in Lemma 2.9 to bound this term. Note that although the results and proofs below are given in the non-contextual setting, we can easily extend the analysis into the contextual version. The point scattering inequalities in Lemma 2.9 in fact hold for any sequence of (context-)arms.

**Lemma 2.9** (Point Scattering Inequalities)**.** *For an arbitrary sequence of points $a_1, a_2, \cdots$ in a space $\mathcal{A}$, and any sequence of nested partitions $\mathcal{P}_1, \mathcal{P}_2, \cdots$ of the same space $\mathcal{A}$, we have, for any $T$,*

$$\sum_{t=1}^{T} \frac{1}{1 + n_{t-1}^0(a_t)} \leq |\mathcal{P}_T| \left(1 + \log \frac{T}{|\mathcal{P}_T|}\right), \tag{2.21}$$

$$\sum_{t=1}^{T} \left(\frac{1}{1 + n_{t-1}^0(a_t)}\right)^{\alpha} \leq \frac{1}{1-\alpha} |\mathcal{P}_T|^{\alpha} T^{1-\alpha}, \ 0 < \alpha < 1, \tag{2.22}$$

*where $n_{t-1}^0$ is the count as in Definition 2.2, and $|\mathcal{P}_T|$ is the cardinality of the finite partition $\mathcal{P}_T$.*

To prove (2.21) and (2.22), we use a relabelling trick. We start with the proof of (2.21).

*Proof of (2.21).* Consider the partition $\mathcal{P}_T$ at time $T$. We label the regions of the partitions by $j = 1, 2, \cdots, |\mathcal{P}_T|$. Let $t_{j,i}$ be the ordered sequence of time when the $i$-th point in the $j$-th region in $\mathcal{P}_T$ being selected in some previous time. Let $b_j$ be the number of total points in region $j$. Then we have

$$\sum_{j=1}^{|\mathcal{P}_T|} b_j = T\,.$$

If we split all the $T$ arms into $|\mathcal{P}_T|$ groups of final partitions at $T$, then

$$\sum_{t=1}^{T} \frac{1}{1 + n_{t-1}^0(a_t)} \leq \sum_{j=1}^{|\mathcal{P}_T|} \sum_{i=1}^{b_j} \frac{1}{1 + n_{t_{j,i}-1}^0(a_{t_{j,i}}^0)}\,. \tag{2.23}$$

The above is true since the left hand side $1 + n_{t-1}^0(a_t) \geq 1 + n_{t_{j,i}-1}^0(a_{t_{j,i}}^0)$ on the right hand side. The nested partition becomes finer so the final partition region at $T$ counted on the

right-hand side is a subset of the actual region counted on the left-hand side.

Since the partitions are nested, we have $1 + n^0_{t_{j,i}-1}(a_{t_{j,i}}) \geq i$ for all $i, j$. Then for $T \geq 1$,

$$\sum_{i=1}^{b_j} \frac{1}{1 + n^0_{t_{j,i}-1}(a^0_{t_{j,i}})} \leq \sum_{i=1}^{b_j} \frac{1}{i}\,.$$

Summing up to $|\mathcal{P}_T|$ on both hand sides, we have

$$\sum_{j=1}^{|\mathcal{P}_T|} \sum_{i=1}^{b_j} \frac{1}{1 + n^0_{t_{j,i}-1}(a^0_{t_{j,i}})} \leq \sum_{j=1}^{|\mathcal{P}_T|} \sum_{i=1}^{b_j} \frac{1}{i}\,. \tag{2.24}$$

Combine (2.23) and (2.24),

$$\begin{aligned}
\sum_{t=1}^{T} \frac{1}{1 + n^0_{t-1}(a_t)} &\leq \sum_{j=1}^{|\mathcal{P}_T|} \sum_{i=1}^{b_j} \frac{1}{i} && (2.25)\\
&\leq \sum_{j=1}^{|\mathcal{P}_T|} (1 + \log b_j)\\
&= |\mathcal{P}_T| + \sum_{j=1}^{|\mathcal{P}_T|} \log b_j\\
&= |\mathcal{P}_T| + \log \prod_{j=1}^{|\mathcal{P}_T|} b_j\\
&\leq |\mathcal{P}_T| + |\mathcal{P}_T| \log \frac{T}{|\mathcal{P}_T|}\,, && (2.26)
\end{aligned}$$

where (2.26) uses AM-GM inequality and $\sum_{j=1}^{|\mathcal{P}_T|} b_j = T$.

□

Before we prove the general inequality bound with power $\alpha$, we start our analysis with power $\frac{1}{2}$.

**Proposition 2.10.** *The Point Scattering Inequality for power* $\frac{1}{2}$ *is* $\sum_{t=1}^{T} \frac{1}{\sqrt{1+n^0_{t-1}(a_t)}} \leq 2\sqrt{T \cdot |\mathcal{P}_T|}$.

*Proof.* Assume at $T$, we have partition $\mathcal{P}_T$. Assume there are $b_j$ arms in the $j^{th}$ partition at $T$. Then we have $\sum_{j=1}^{|\mathcal{P}_T|} b_j = T$. Let $t_{j,i}$ be the ordered sequence of time when the $i$-th point in the $j$-th region in $\mathcal{P}_T$ being selected in some previous time. Given we split all the $T$ arms

into $|\mathcal{P}_T|$ groups of final partitions at $T$, then

$$\sum_{t=1}^{T} \frac{1}{\sqrt{1+n_{t-1}^0(a_t)}} \leq \sum_{j=1}^{|\mathcal{P}_T|} \sum_{i=1}^{b_j} \frac{1}{\sqrt{1+n_{t_{j,i}}^0(a_{t_{j,i}}^0)}}\,. \tag{2.27}$$

The above is true since for the same arm $x$, the left hand side $\sqrt{1+n_{t-1}^0(a_t)} \geq \sqrt{1+n_{t_{j,i}}^0(a_{t_{j,i}}^0)}$ on the right hand side since our nested partition becomes finer. The final partition region at $T$ counted on the right hand side is a subset of the actual region counted on the left hand side.

Since the partitions are nested, we have $1+n_{t_{j,i}-1}^0(a_{t_{j,i}}) \geq i$ for all $i, j$. Then for $T \geq 1$,

$$\sum_{i=1}^{b_j} \frac{1}{\sqrt{1+n_{t_{j,i}}^0(a_{t_{j,i}}^0)}} \leq \sum_{i=1}^{b_j} \frac{1}{\sqrt{i}}\,.$$

Summing up to $|\mathcal{P}_T|$ on both hand sides, we have

$$\sum_{j=1}^{|\mathcal{P}_T|} \sum_{i=1}^{b_j} \frac{1}{\sqrt{1+n_{t_{j,i}}^0(a_{t_{j,i}}^0)}} \leq \sum_{j=1}^{|\mathcal{P}_T|} \sum_{i=1}^{b_j} \frac{1}{\sqrt{i}}\,. \tag{2.28}$$

Since

$$\sum_{i=1}^{b_j} \frac{1}{\sqrt{i}} \leq 2\int_1^{b_j} \frac{1}{\sqrt{x}} dx \leq 2\sqrt{b_j}\,,$$

which implies

$$\sum_{j=1}^{|\mathcal{P}_T|} \sum_{i=1}^{b_j} \frac{1}{\sqrt{i}} \leq \sum_{j=1}^{|\mathcal{P}_T|} 2\sqrt{b_j}\,. \tag{2.29}$$

By the concavity of function, we have

$$\sum_{j=1}^{|\mathcal{P}_T|} 2\sqrt{b_j} = 2|\mathcal{P}_T| \sum_{j=1}^{|\mathcal{P}_T|} \frac{1}{|\mathcal{P}_T|}\sqrt{b_j} \leq 2|\mathcal{P}_T| \sqrt{\sum_{j=1}^{|\mathcal{P}_T|} \frac{1}{|\mathcal{P}_T|} b_j}\,.$$

This implies that

$$\sum_{j=1}^{|\mathcal{P}_T|} 2\sqrt{b_j} \leq 2|\mathcal{P}_T| \sqrt{\sum_{j=1}^{|\mathcal{P}_T|} \frac{1}{|\mathcal{P}_T|} b_j} = 2\sqrt{|\mathcal{P}_T|} \sqrt{\sum_{j=1}^{|\mathcal{P}_T|} b_j} = 2\sqrt{T \cdot |\mathcal{P}_T|}\,. \tag{2.30}$$

Combining (2.27), (2.28), (2.29) and (2.30) above, we hence have

$$\sum_{t=1}^{T} \frac{1}{\sqrt{1+n_{t-1}^{0}(a_t)}} \leq 2\sqrt{T \cdot |\mathcal{P}_T|}\,.$$

.

□

Now we can generalize to the proof of (2.22). For the proof of inequality (2.22), the idea is similar to that of (2.21) and the proof of Proposition (2.10).

*Proof of (2.22).* For $0 < \alpha < 1$,

$$\begin{aligned}
\sum_{t=1}^{T} \left(\frac{1}{1+n_{t-1}^{0}(a_t)}\right)^{\alpha} &= \sum_{j=1}^{|\mathcal{P}_T|} \sum_{i=1}^{b_j} \left(\frac{1}{1+n_{t_{j,i}}^{0}(a_{t_{j,i}}^{0})}\right)^{\alpha} \\
&\leq \sum_{j=1}^{|\mathcal{P}_T|} \sum_{i=1}^{b_j} \frac{1}{i^{\alpha}} \\
&\leq \sum_{j=1}^{|\mathcal{P}_T|} \frac{1}{1-\alpha} b_j^{1-\alpha} \\
&= \frac{1}{1-\alpha} \sum_{j=1}^{|\mathcal{P}_T|} b_j^{1-\alpha} \cdot 1 \\
&\leq \frac{1}{1-\alpha} \left(\sum_{j=1}^{|\mathcal{P}_T|} b_j\right)^{1-\alpha} \left(\sum_{j=1}^{|\mathcal{P}_T|} 1\right)^{\alpha} \qquad (2.31) \\
&\leq \frac{1}{1-\alpha} |\mathcal{P}_T|^{\alpha} T^{1-\alpha}\,,
\end{aligned}$$

where the inequality (2.31) is due to the Hölder's inequality and

$$\sum_{j=1}^{|\mathcal{P}_T|} b_j = T\,.$$

□

### 2.3.3 Provable Guarantee for TreeUCB Algorithm

Now we can use the above results to complete the proof of Theorem 2.3 and 2.6. As claimed, the point scattering inequalities hold for the contextual version with a trivial extension in the proof of Lemma 2.9.

*Proof of Theorem 2.6.* Recall that $R_T^c(\texttt{CTUCB}) = \sum_{t=1}^T f(z_t, a_t^*) - f(z_t, a_t)$. From (2.14), for each $t$, if we define the event

$$E_t := \left\{ (z_t, a_t) : f(z_t, a_t^*) - f(z_t, a_t) > 2L \cdot D(p_{t-1}(z_t, a_t)) + 2C\sqrt{\frac{4\log t}{n_{t-1}(z_t, a_t)}} \right\},$$

then $\mathcal{P}(E_t) \leq \frac{1}{t^4}$. This implies

$$\sum_{t=1}^{\infty} \mathcal{P}(E_t) < \infty \,.$$

From Borel-Cantelli Lemma 1,

$$\mathcal{P}\left(\lim_{t\to\infty} \sup_{s\geq t} E_s\right) = \mathcal{P}\left(E_t; \text{i.o.}\right) = 0 \,.$$

This is equivalent to

$$\mathcal{P}\left(\lim_{t\to\infty} \inf_{s\geq t} E_s^c\right) = \mathcal{P}\left(E_t^c; \text{ev}\right) = 1 \,,$$

where $E_t^c$ is the complement of $E_t$ and $(E_t^c; \text{ev})$ means $E_t^c$ eventually always happen. In words, with probability 1, $E_t$ only occurs finitely many times. Let $t'$ be the final time that $E_{t'}$ happens, then for any $\tau > t'$,

$$f(z_\tau, a_\tau^*) - f(z_\tau, a_\tau) \leq 2L \cdot D(p_{\tau-1}(z_\tau, a_\tau)) + 2C\sqrt{\frac{4\log \tau}{n_{\tau-1}(z_\tau, a_\tau)}} \,.$$

Therefore, with probability 1,

$$\lim_{T\to\infty} R_T^c(\texttt{CTUCB}) \leq \lim_{T\to\infty} 2\sum_{t=1}^{T} \left\{ L \cdot D(p_{t-1}(z_t, a_t)) + C\sqrt{\frac{4\log t}{n_{t-1}(z_t, a_t)}} \right\}. \tag{2.32}$$

From the contextual version of the novel point scattering inequality (2.21) above, we know

$$\sum_{t=1}^{T} \frac{1}{1 + n_{t-1}^0(z_t, a_t)} \leq |\mathcal{P}_T| \left(1 + \log \frac{T}{|\mathcal{P}_T|}\right). \tag{2.33}$$

By Cauchy-Schwarz inequalities, we have

$$C\sum_{t=1}^{T} \sqrt{\frac{4\log t}{n_{t-1}(z_t, a_t)}} \leq C\sqrt{T\log T}\sqrt{|\mathcal{P}_T|\left(1 + \log \frac{T}{|\mathcal{P}_T|}\right)} < o\left(T^{\frac{1+\gamma}{2}}\right) < o(T) \,,$$

where the second last inequality is from the assumption (2) in Theorem 2.3 and 2.6, where $|\mathcal{P}_t| = o(t^\gamma)$ for some $\gamma < 1$. This means

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} C \sqrt{\frac{4 \log t}{n_{t-1}(z_t, a_t)}} = 0 \,. \tag{2.34}$$

In addition, by the assumption (3) in Theorem 2.3 and 2.6, where $D(p_{t-1}(z_t, a_t)) = o(1)$,

$$\sum_{t=1}^{T} L \cdot D(p_{t-1}(z_t, a_t)) < o(T) \,.$$

This is equivalent to

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} L \cdot D(p_{t-1}(z_t, a_t)) = 0 \,. \tag{2.35}$$

Combine (2.32), (2.34), and (2.35), then with probability 1,

$$\lim_{T\to\infty} \frac{R_T^c(\texttt{CTUCB})}{T} = 0 \,. \tag{2.36}$$

We hence complete the proof.

□

The proof of Theorem 2.3 (non-contextual version) is the same as above.

## 2.4 Some Applications of Point Scattering Inequalities

In this section, we give examples of using the point scattering inequalities to derive regret bounds for some other known algorithms. For our purpose of illustrating the point scattering inequalities, the discussed algorithms are simplified. We also assume that the reward and the sub-Gaussianity are properly scaled so that the parameter before the Hoeffding-type concentration term is 1.

### 2.4.1 The UCB1 algorithm

The classic UCB1 algorithm [Auer et al., 2002] assumes a finite set of arms, each having a different reward distribution. Following our notation, at time $t$, the UCB1 algorithm (with

known horizon $T$) plays

$$a_t \in \arg\max_{a\in\mathcal{A}} \left\{ m_{t-1}(a) + \sqrt{\frac{4\log T}{n_{t-1}(a)}} \right\}. \tag{2.37}$$

Indeed, this equation can be interpreted as (2.4) under the discrete 0-1 metric: two points are distance zero if they coincide and distance 1 otherwise. Thus the Lipschitzness term vanishes in this case and Propositions 2.7 and 2.8 can be modified appropriately.

**Algorithm 3** UCB1 Algorithm [Auer et al., 2002]

1: Input: finite $K$ arms in $\mathcal{A}$.
2: **for** $t = 1, 2, \ldots, T$ **do**
3: With respect to $\mathcal{A}$, define/update $m_{t-1}$ and $n_{t-1}$. Play $a_t \in \arg\max_{a\in\mathcal{A}} \left\{ m_{t-1}(a) + \sqrt{\frac{4\log T}{n_{t-1}(a)}} \right\}$. Ties are broken at random.
4: Observe the reward $y_t$.

Assume $K$ is the number of arms in the UCB1 algorithm. We have the following theorem of the regret bound in the UCB1 algorithm. In this analysis, we apply the point scattering inequality with the partition $\mathcal{P}_t$ being the set of arms at all $t$.

**Theorem 2.11** (The Regret of UCB1 Algorithm)**.** *Algorithm 3 admits regret bound* $\mathbb{E}R_T(UCB1) = \tilde{\mathcal{O}}\left(\sqrt{K \cdot T}\right)$.

*Proof.* First, we need a concentration result like Proposition 2.7 for the UCB1 framework. Note that there are finite $K$ arms with stochastic rewards in $[0, 1]$. Since each of the $K$ arms is always itself a region in the partition, then the diameter of each region that contains each arm is 0. In other words, $L \cdot D(p_{t-1}(a))$ in Proposition 2.7 will be eliminated. Following the proof of Proposition 2.7, with Azuma-Hoeffding inequality, we have

$$|m_{t-1}(a) - f(a)| \leq \sqrt{\frac{4\log t}{n_{t-1}(a)}} \leq \sqrt{\frac{4\log T}{n_{t-1}(a)}} \tag{2.38}$$

with probability at least $1 - \frac{1}{t^4}$. From the UCB1 updating rule in (2.37), at any time $t$,

$$m_{t-1}(a_t) + \sqrt{\frac{4\log T}{n_{t-1}(a_t)}} \geq m_{t-1}(a^*) + \sqrt{\frac{4\log T}{n_{t-1}(a^*)}}. \tag{2.39}$$

Otherwise, arm $a_t$ will not be selected at time $t$. Following the proof of Proposition 2.8 and (2.38), with probability at least $1 - \frac{1}{t^4}$, we have the inequality (we use one-sided Azuma-

Hoeffding inequality and union bound),

$$f(a^*) \leq m_{t-1}(a^*) + \sqrt{\frac{4\log T}{n_{t-1}(a^*)}} \\ \leq m_{t-1}(a_t) + \sqrt{\frac{4\log T}{n_{t-1}(a_t)}} \leq f(a_t) + 2\sqrt{\frac{4\log T}{n_{t-1}(a_t)}}\,. \tag{2.40}$$

This is equivalent to, with probability at least $1 - \frac{1}{t^4}$,

$$f(a^*) - f(a_t) \leq 2\sqrt{\frac{4\log T}{n_{t-1}(a_t)}}\,, \tag{2.41}$$

as analog to Proposition 2.8. Note that

$$\sum_{t=1}^{T} 2\sqrt{\frac{4\log T}{n_{t-1}(a_t)}} \\ \leq 4\sqrt{\sum_{t=1}^{T}\log T}\cdot\sqrt{\sum_{t=1}^{T}\frac{1}{n_{t-1}(a_t)}} \tag{2.42}$$

$$\leq 4\sqrt{T\log T}\cdot\sqrt{|\mathcal{P}_T|\log\left(1+\frac{T}{|\mathcal{P}_T|}\right)} \tag{2.43}$$

$$= \tilde{\mathcal{O}}\left(\sqrt{K\cdot T}\right)\,, \tag{2.44}$$

where (2.42) uses Cauchy-Schwarz inequality, and (2.43) uses the novel point scattering inequality (2.21). (2.44) is true since the total number of arms is $K$, which means $|\mathcal{P}_T| = K$.

Let us find the regret bound of UCB1 algorithm. We split the sum of the regret,

$$R_T = \sum_{t=1}^{T}(f(a^*) - f(a_t)) = \sum_{t=1}^{\lfloor\sqrt{T}\rfloor}(f(a^*) - f(a_t)) + \sum_{\lfloor\sqrt{T}\rfloor+1}^{T}(f(a^*) - f(a_t))\,.$$

From 2.41, we know with probability at least $1 - \frac{2}{3}T^{\frac{-3}{2}}$,

$$f(a^*) - f(a_t) \leq 2\sqrt{\frac{4\log T}{n_{t-1}(a_t)}} \tag{2.45}$$

hold simultaneously for all $t = \left\lfloor\sqrt{T}\right\rfloor + 1, \cdots, T$. This is a result of taking a union bound at

$a_t$ and $a^*$, and then over $t = \left\lfloor \sqrt{T} \right\rfloor + 1, \cdots, T$. Let $E$ be the event where

$$E := \left\{ f(a^*) - f(a_t) \leq 2\sqrt{\frac{4\log T}{n_{t-1}(a_t)}}, \text{ for all } t = \left\lfloor \sqrt{T} \right\rfloor + 1, \cdots, T \right\}.$$

Denote by $E^c$ the complement of $E$. In the following, we use $R_T$ to represent $R_T(UCB1)$. By the law of total expectation, we get

$$\begin{aligned}
&\mathbb{E}[R_T] = \mathbb{E}\left[R_T|E\right]\mathcal{P}(E) + \mathbb{E}\left[R_T|E^c\right](1-\mathcal{P}(E)) \\
&\leq \mathbb{E}\left[\sum_{t=1}^{\lfloor\sqrt{T}\rfloor} f(a^*) - f(a_t) \,\middle|\, E\right] + \mathbb{E}\left[\sum_{t=\lfloor\sqrt{T}\rfloor+1}^{T} f(a^*) - f(a_t) \,\middle|\, E\right] + T \cdot \frac{2}{3} \cdot T^{\frac{-3}{2}} \\
&\leq \sqrt{T} + \sum_{t=\lfloor\sqrt{T}\rfloor+1}^{T} 2\sqrt{\frac{4\log T}{n_{t-1}(a_t)}} + \frac{2}{3}T^{\frac{-1}{2}} \qquad (2.46) \\
&\leq \sqrt{T} + \tilde{\mathcal{O}}\left(\sqrt{K \cdot T}\right) + \frac{2}{3}T^{\frac{-1}{2}} \qquad (2.47) \\
&\leq \tilde{\mathcal{O}}\left(\sqrt{K \cdot T}\right),
\end{aligned}$$

where (2.46) is from $f(a^*) - f(a_t) \leq 1$ and (2.45), while (2.47) is from (2.44).

The algorithm hence admits regret bound $\mathbb{E}R_T(UCB1) = \tilde{\mathcal{O}}\left(\sqrt{K \cdot T}\right)$. □

This result matches the gap-independent (independent of the reward gap between an arm and the optimal arm) bound derived using other traditional methods in the UCB1 algorithm [Auer et al., 2002, Bubeck and Cesa-Bianchi, 2012].

### 2.4.2 The `UniformMesh` algorithm in $[0,1]^d$

A modified version of the `UniformMesh` algorithm [Kleinberg et al., 2008] can use the point scattering inequality for regret analysis. The idea of the `UniformMesh` algorithm is to finitely uniformly partition the space, and treat the problem as a finite arm problem.

**Theorem 2.12** (The Regret of `UniformMesh` Algorithm)**.** *Algorithm 4 admits regret bound* $\mathbb{E}R_T(\textit{UniformMesh}) = \tilde{\mathcal{O}}\left(T^{\frac{d+1}{d+2}}\right)$.

### 2.4.3 The contextual `UniformMesh` in $[0,1]^d$

A modified version, the contextual `UniformMesh` algorithm [Slivkins, 2014], can also use the point scattering inequality for regret analysis. The idea of the contextual `UniformMesh`

**Algorithm 4** Simple `UniformMesh` [Kleinberg et al., 2008]

1: Input: $\mathcal{P}_0$ as the whole space.
2: **for** $t = 1, 2, \ldots, T$ **do**
3: With respect to the partition $\mathcal{P}_{t-1}$ of the space $\mathcal{A}$, define/update $m_{t-1}$ and $n_{t-1}$. Play $a_t \in \arg\max_{a\in\mathcal{A}} \left\{ m_{t-1}(a) + \sqrt{\frac{4\log t}{n_{t-1}(a)}} \right\}$. Ties are broken at random.
4: Make sure $D(p_{t-1}(a)) \leq 2\sqrt{d}t^{-\frac{1}{d+2}}$ for all $a$ at any $t$. Only when this is violated, break all hypercubes in $\mathcal{P}_{t-1}$ into $2^d$ identical (sub-)hypercubes with half its diameter. ▷ The regions in the partitions are hypercubes, and the diameter of the regions are edge length.
5: Observe the reward $y_t$.

algorithm is also to finitely uniformly partition the space, and treat the problem as a finite problem.

**Algorithm 5** Simple contextual `UniformMesh` [Slivkins, 2014]

1: Input: $\mathcal{P}_0$ as the whole space.
2: **for** $t = 1, 2, \ldots, T$ **do**
3: Observe context $z_t$.
4: With respect to the partition $\mathcal{P}_{t-1}$ of the context-arm space $\mathcal{Z}\times\mathcal{A}$, define/update $m_{t-1}$ and $n_{t-1}$. Play $a_t \in \arg\max_{a\in\mathcal{A}} \left\{ m_{t-1}(z_t, a) + \sqrt{\frac{4\log t}{n_{t-1}(z_t,a)}} \right\}$. Ties are broken at random.
5: Make sure $D(p_{t-1}(z, a)) \leq 2\sqrt{d}t^{-\frac{1}{d+2}}$ for all $(z, a)$ at any $t$. Only when this is violated, break all hypercubes in $\mathcal{P}_{t-1}$ into $2^d$ identical (sub-)hypercubes with half its diameter. ▷ The regions in the partitions are hypercubes, and the diameter of the regions are edge length.
6: Observe the reward $y_t$.

**Theorem 2.13** (The Regret of Contextual `UniformMesh` Algorithm)**.** *Algorithm 5 admits contextual regret bound* $\mathbb{E}R_T^c(\texttt{UniformMesh}) = \tilde{\mathcal{O}}\left(T^{\frac{d+1}{d+2}}\right)$.

Now we provide a proof for Theorem 2.12. The argument can be easily generalized to Theorem 2.13. To prove Theorem 2.12, we need the partition in `UniformMesh` Algorithm to conform to the following legal partition rule.

**Definition 2.14** (Legal Partition Rule)**.** For a sequence of arms $\{a_1, a_2, \ldots, a_t\} \subset [0, 1]^d$, a partition $\mathcal{P}_t = \left\{ P_t^{(1)}, P_t^{(2)}, \ldots, P_t^{(k_t)} \right\}$ of $[0, 1]^d$ is said to conform to a legal partition rule if for all $a \in [0, 1]^d$, for any $i = 1, \cdots, t$ and for any $P_i \in \mathcal{P}_i$,

$$D(P_i) \leq 2\sqrt{d} \cdot i^{-\frac{1}{d+2}}. \tag{2.48}$$

The legal partition rule above satisfies assumption (3) in Theorem 2.3. The legal partition rule always holds if we conform to the following construction.

**The Construction of Legal Partition**: note that the initial $[0,1]^d$ is a hypercube. As shown below, we guarantee all partitions in $\mathcal{P}_i$ at time $i$ are hypercubes during the construction. With the increment of $i$, the upper bound in legal partition rule (2.48) will decrease. Assume at some specific $i(i < t)$, (2.48) is violated for some hypercube $P_i \in \mathcal{P}_i$. Then our algorithm will enforce every hypercube in $\mathcal{P}_i$ to further break into $2^d$ smaller identical (sub-)hypercubes where the diameters of the new (child) hypercubes are one-half of the original (parent) hypercubes. These new hypercubes will compose up and become a new partition $\mathcal{P}_i$. This is a ***nested partition*** that satisfies assumption (1) in Theorem 2.3. After breaking, (2.48) will again hold. The new partition will stay the same until (2.48) would be violated again after adding more arms into $[0,1]^d$. In that case, our algorithm will repeat the same process and construct further ***nested partitions***.

We have the following property with the construction of legal partition.

**Lemma 2.15.** *For the* `UniformMesh` *in* $[0,1]^d$*, the cardinality of partition* $|\mathcal{P}_t|$*, based on the legal partition construction, has order* $\mathcal{O}\left(t^{\frac{d}{d+2}}\right)$.

*Proof.* Following the legal partition construction, assume at time $t$, we have total $r$ rounds of partitions where each hypercube breaks into equal $2^d$ (sub-)hypercubes in each round. These $2^d$ (sub-)hypercubes are identical with the same diameters and then become a new partition set $\mathcal{P}_t$. Thus, we multiply the original partition cardinality by $2^d$ after each round of partition. By the same reason, we have all the hypercubes with diameter $\sqrt{d}\left(\frac{1}{2}\right)^r$ after $r$ rounds of partitions.

In addition, having $r$ rounds of partitions up to $t$ means that the diameter of hypercubes after $(r-1)$ rounds, $\sqrt{d}\left(\frac{1}{2}\right)^{r-1}$, must violate the legal partition rule (2.48) before reaching $t$. It is equivalent to say,

$$\sqrt{d}\left(\frac{1}{2}\right)^{r-1} > 2\sqrt{d}\cdot\tau^{-\frac{1}{d+2}} \geq 2\sqrt{d}\cdot t^{-\frac{1}{d+2}} \tag{2.49}$$

for some $\tau \leq t$. Thus,

$$\begin{aligned} 2^{-(r-1)} &> 2t^{-\frac{1}{d+2}} \\ 2^{-r} &> t^{-\frac{1}{d+2}} \\ 2^r &< t^{\frac{1}{d+2}}\,. \end{aligned} \tag{2.50}$$

Recall that the partition cardinality is multiplied by $2^d$ after each round of partition. At $t$,

with total $r$ rounds of partitions happened, $|\mathcal{P}_t|= \left(2^d\right)^r$. From (2.50), this implies

$$|\mathcal{P}_t|= \left(2^d\right)^r < t^{\frac{d}{d+2}} . \tag{2.51}$$

It is worth noting that the coefficient $\sqrt{d}$ in legal partition cancels out the dimension dependent coefficient in the arm space $[0,1]^d$. □

The above result satisfies assumption (2) in Theorem 2.3. Now we can complete the proof of Theorem 2.12.

*Proof of Theorem 2.12.* From the point scattering inequality in (2.22) and Lemma 2.15, with $\alpha = \frac{1}{2}$,

$$\sum_{t=1}^{T} \sqrt{\frac{1}{n_{t-1}(a_t)}} \leq \frac{1}{1-\frac{1}{2}} |\mathcal{P}_T|^{\frac{1}{2}} \cdot T^{\frac{1}{2}} = \mathcal{O}\left(T^{\frac{d}{d+2}} \cdot T^{\frac{1}{2}}\right) = \mathcal{O}\left(T^{\frac{d+1}{d+2}}\right) . \tag{2.52}$$

We split the sum of the regret,

$$\sum_{t=1}^{T} \left(f(a^*) - f(a_t)\right) = \sum_{t=1}^{\lfloor\sqrt{T}\rfloor} \left(f(a^*) - f(a_t)\right) + \sum_{\lfloor\sqrt{T}\rfloor+1}^{T} \left(f(a^*) - f(a_t)\right) .$$

From the non-contextual version of Proposition 2.8, we know with probability at least $1-\frac{2}{3}T^{\frac{-3}{2}}$,

$$f(a^*) - f(a_t) \leq \left[2\sqrt{\frac{4\log t}{n_{t-1}(a_t)}} + 2L \cdot D(p_{t-1}(a_t))\right] \tag{2.53}$$

hold simultaneously for all $t = \left\lfloor\sqrt{T}\right\rfloor + 1, \cdots, T$. This is a result of taking a union bound at $a_t$ and $a^*$, and then over $t = \left\lfloor\sqrt{T}\right\rfloor + 1, \cdots, T$. Let $E$ be the event where

$$\begin{aligned} E &:= \left\{f(a^*) - f(a_t) \leq B_t, \text{ for all } t = \left\lfloor\sqrt{T}\right\rfloor + 1, \cdots, T\right\} , \\ B_t &:= \left[2\sqrt{\frac{4\log t}{n_{t-1}(a_t)}} + 2L \cdot D\left(p_{t-1}(a_t)\right)\right] . \end{aligned}$$

Denote by $E^c$ the complement of $E$. In the following, we use $R_T$ to represent $R_T(\texttt{UniformMesh})$.

By the law of total expectation, we get

$$\begin{aligned}
&\mathbb{E}[R_T] = \mathbb{E}\left[R_T|E\right]\mathcal{P}(E) + \mathbb{E}\left[R_T|E^c\right](1-\mathcal{P}(E)) \\
&\leq \mathbb{E}\left[\sum_{t=1}^{\lfloor\sqrt{T}\rfloor} f(a^*) - f(a_t) \,\middle|\, E\right] + \mathbb{E}\left[\sum_{t=\lfloor\sqrt{T}\rfloor+1}^{T} f(a^*) - f(a_t) \,\middle|\, E\right] + T\cdot\frac{2}{3}\cdot T^{\frac{-3}{2}} \\
&\leq \sqrt{T} + \sum_{t=\lfloor\sqrt{T}\rfloor+1}^{T}\left[2\sqrt{\frac{4\log t}{n_{t-1}(a_t)}} + 2L\cdot D(p_{t-1}(a_t))\right] + \frac{2}{3}T^{\frac{-1}{2}} \qquad (2.54) \\
&\leq \sqrt{T} + 4\sqrt{\log T}\sum_{t=1}^{T}\sqrt{\frac{1}{n_{t-1}(a_t)}} + 2L\sum_{t=\lfloor\sqrt{T}\rfloor+1}^{T} 2\sqrt{d}t^{-\frac{1}{d+2}} + \frac{2}{3}T^{\frac{-1}{2}} \qquad (2.55) \\
&\leq \sqrt{T} + 4\sqrt{\log T}\cdot\mathcal{O}\left(T^{\frac{d+1}{d+2}}\right) + 4L\sqrt{d}\cdot\Theta\left(T^{\frac{d+1}{d+2}}\right) + \frac{2}{3}T^{\frac{-1}{2}} \qquad (2.56) \\
&\leq \tilde{\mathcal{O}}\left(T^{\frac{d+1}{d+2}}\right), \qquad (2.57)
\end{aligned}$$

where (2.54) is from (2.53), and (2.55) is from the legal partition rule in (2.48). (2.56) above is from (2.52), which is from the point scattering inequality. We hence complete the proof. □

Note that in our simplified version, we assume the reward is properly scaled so that $C = 1$, and the term $D(p_{t-1}(a))$ takes the same value for all $a \in \mathcal{A}$ and all $t = 1, 2, \cdots, T$ for the `UniformMesh` algorithm. This recovers the form of UCB strategy in Algorithm 4. The proof of Theorem 2.13 trivially extends from the proof of Theorem 2.12.

## 2.5 Empirical Study [2]

Since the TreeUCB algorithm imposes only mild constraints on tree formation, we use greedy decision tree training to fit the reward function, instead of controlling only the region diameters and the partition cardinality. For the experiments in this section, we use the greedy decision tree training rule: at each split, we find the split that maximizes the reduction in the *Mean Absolute Error (MAE)* (1.4), and we stop growing the tree once the maximal possible reduction is below a certain threshold $\eta$. We state a more detailed description of the tree fitting below.

[2]The author acknowledges Tianyu Wang since he is the main contributor to the corresponding experiments in this joint work by accessing powerful computing resources.

### 2.5.1 Tree Fitting Rule in Experiments

One nice property of Algorithm 1 is that it only imposes loose rules on the partition formation. Therefore we can use a greedy criterion for constructing regression trees to construct the partition. Leaves in a regression tree form a partition of the space. At the same time, a regression tree is designed to fit the underlying function. This property tends to result in an adaptive partition where the underlying function values within each region are relatively close to each other. In the experiments here, we use the *Mean Absolute Error* (*MAE*) reduction criterion [Breiman et al., 1984] to adaptively construct a regression tree. More specifically, a node $\boldsymbol{N}$ containing a sequence of data samples $\{(a_1, y(a_1)), (a_2, y(a_2)), \ldots, (a_n, y(a_n))\}$ is split along a feature (can be randomly selected for scalability) into $\boldsymbol{N}_1$ and $\boldsymbol{N}_2$ (where $\boldsymbol{N}_1 \cup \boldsymbol{N}_2 = \boldsymbol{N}$ and $\boldsymbol{N}_1 \cap \boldsymbol{N}_2 = \phi$) such that the following reduction in *MAE* is maximized:

$$MAE(\boldsymbol{N}) - \left( \frac{|\boldsymbol{N}_1|}{|\boldsymbol{N}|} MAE(\boldsymbol{N}_1) + \frac{|\boldsymbol{N}_2|}{|\boldsymbol{N}|} MAE(\boldsymbol{N}_2) \right), \tag{2.58}$$

where $MAE(\boldsymbol{N}) = \frac{1}{|\boldsymbol{N}|} \sum_{a_i \in \boldsymbol{N}} |y(a_i) - \hat{y}(\boldsymbol{N})|$ and $\hat{y}(\boldsymbol{N}) = \frac{1}{|\boldsymbol{N}|} \sum_{a_i \in \boldsymbol{N}} y(a_i)$. The nodes are recursively split until the maximal possible reduction in *MAE* is smaller than $\eta$. The leaves are then used to form a partition. Each region is again associated with a corrected mean and corrected count. Using regression trees, we develop the TreeUCB algorithm (TUCB), and the Contextual TreeUCB algorithm (CTUCB), as summarized in Algorithms 1 and 2. The code is provided on request, and can also be implemented via the `scikit-learn` package [Pedregosa et al., 2011].

### 2.5.2 Synthetic Data

Here we compare the performance of our TUCB against the HOO algorithm and the Zooming Bandit Algorithm [Bubeck et al., 2011, Kleinberg et al., 2008]. We also compare the performance of CTUCB against the Contextual Zooming Bandit Algorithm and the CCKL-UCB algorithm [Slivkins, 2014, Magureanu et al., 2014]. We set the tree fitting rule $L = 1$ and $C = 1$ for TUCB (CTUCB). We use the Himmelblau function and the Goldstein function as test functions. These two functions are frequently used in the study of optimization. The test functions are two-dimensional. In the experiments for contextual algorithms: (1) we use one input dimension of the test function as the context, and this dimension is randomly sampled at each round; (2) we let the algorithm choose a point (an arm) from the other dimension.

Let us look at the following details. The (negated) Himmelblau function used for the

synthetic experiment in Figure 9 is

$$f(x_1, x_2) = -\left[(x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2\right] ,$$

where $x_1, x_2 \in [-5, 5]$.

The (negated) Goldstein function used for the synthetic experiment in Figure 9 is

$$\begin{aligned} f(x_1, x_2) = &-[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \\ &\times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] , \end{aligned}$$

where $x_1, x_2 \in [-2, 2]$.

Note that we rescale the domains of the above functions to $[-0.5, 0.5]^2$, and rescale their ranges to within $[0, 1]$ in our experiments. For the competing algorithms, the coefficients for the exploration term are all 1 (default for Zooming, Contextual Zooming, and HOO's concent). In particular for HOO, we set $v_1 = 0.1, \rho = 0.1$ (see Algorithm 1 in [Bubeck et al., 2011] for details about their $v$ and $\rho$ in HOO's setting).

As shown in Figure 9, TreeUCB and CTUCB outperform the benchmark algorithms on synthetic data. The performance gain is from the fact TreeUCB is *adaptively learning* the partition, instead of only controlling the partition with strict rules. In this experiment, we implemented an incremental tree to enforce the nested partitions.

### 2.5.3 Neural Network Tuning

One application of stochastic bandit algorithms is zeroth order optimization. Here we consider deep convolutional neural network model on three classical datasets MNIST, SVHN, and CIFAR-10 as introduced in Example 1.5-1.7 [LeCun, 1999, Netzer et al., 2011, Krizhevsky and Hinton, 2009]. We apply TUCB to tuning neural networks. In this setting, we treat the hyperparameter configurations (e.g. learning rate and network architecture as covered in chapter 1) as the arms of the bandit, and use validation accuracy as a reward. The task is to select a hyperparameter configuration and train the network to observe the validation accuracies as rewards and locate the best hyperparameter configuration rapidly. This experiment shows that TUCB can compete with the state-of-the-art methods such as Spearmint, SMAC, Tree Parzen Estimator (TPE), and Harmonica on such hard real-world tasks [Snoek et al., 2012, Hutter et al., 2011, Bergstra et al., 2011, Hazan et al., 2017]. The $x$-axis in the left most subplot of Figure 10 is time, which proves TreeUCB's scalability when implemented using `scikit-learn`. The details of the neural network model architecture and the hyperparameter configuration for each model are covered in section 2.7.

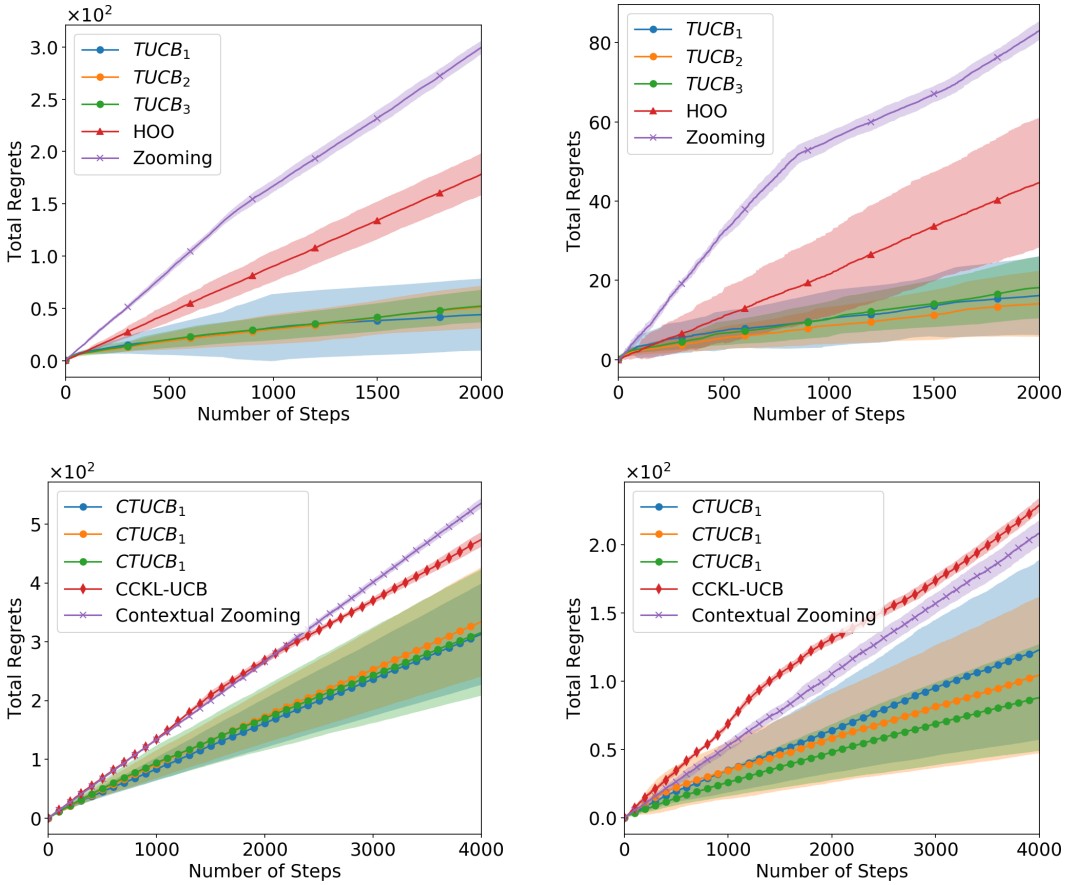
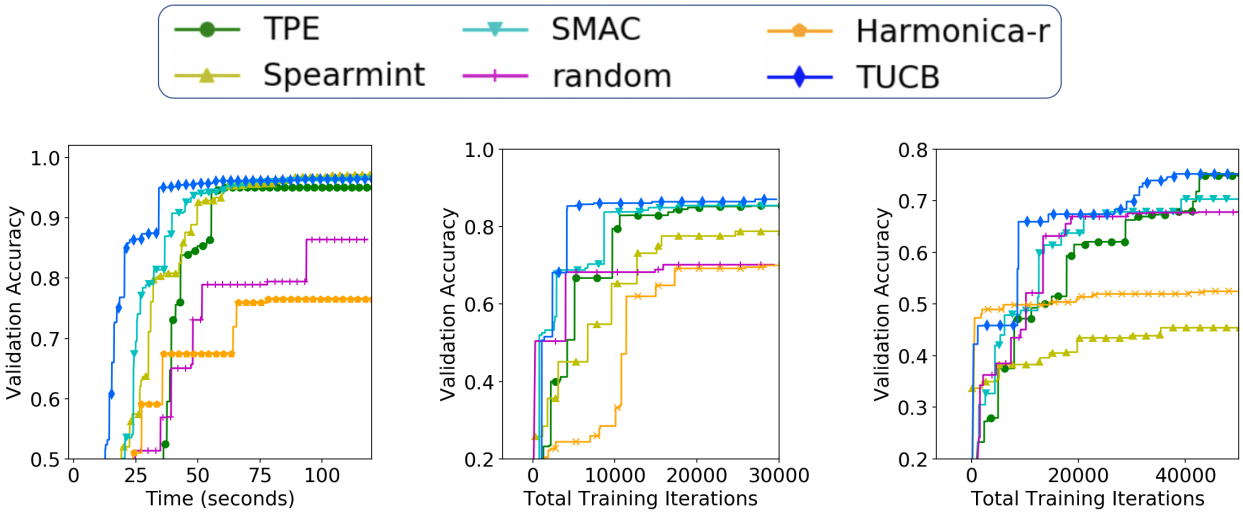



Figure 9: Upper Left: Himmelblau for the non-contextual task. Upper Right: Goldstein for the non-contextual task. Lower Left: Himmelblau for the contextual task. Lower Right: Goldstein for the contextual task. All methods use 1 as exploration parameter. (C)TUCB$_1$, (C)TUCB$_2$, (C)TUCB$_3$ corresponds to TUCB (CTUCB) with $\eta = 0.02$, $\eta = 0.01$, $\eta = 0.005$ correspondingly ($\eta$ as discussed at the beginning of this section).

## 2.6 Conclusion

We propose the TreeUCB and the Contextual TreeUCB frameworks that use decision trees (regression trees) to flexibly partition the arm space and the context-arm space as an Upper Confidence Bound strategy is played across the partition regions. This regret analysis via the point scattering inequalities can provide a unified analysis framework for multiply known algorithms such as UCB1 and `UniformMesh` Algorithms. We also provide implementations using decision trees that learn the partition. Empirical studies show that TUCB and CTUCB can empirically outperform HOO, Zooming bandit, and CCKL-UCB. Also, TUCB is competitive with state-of-the-art hyperparameter optimization methods in hard tasks like neural network




MLP for MNIST | CNN for SVHN | CNN for CIFAR-10

Figure 10: For MNIST, each plot is averaged over 10 runs. For SVHN and CIFAR-10, each plot is averaged over 5 runs. The implementation of TUCB here uses sci-kit-learn package. In the left-most subplot, $x$-axis is time (in seconds). This shows TUCB's scalability, since TUCB's curve goes up the fastest.

tuning and could save substantial computing resources. This suggests that, in addition to random search and Bayesian optimization methods, more bandit algorithms should be considered as benchmarks for difficult real-world problems such as neural network-tuning.

## 2.7 Supplementary

In this section, we present the architecture of the neural network model and the corresponding hyperparameter configuration details for the experiment in neural network tuning. Following the tree fitting rule in the experiment, we use $C = 0.1$, $L = 0.01$ and $\eta = 0.001$ for TreeUCB. Since the parameter choices are first discretized, we use the following way to compute "diameter" of a leaf region in TreeUCB: for a leaf region $l$, we count how many points in the discretized parameter space $l$ covers, and use the ratio of this count over a total number of discretized grids as the diameter for $l$. All other algorithms use their default setting.

### MLP for MNIST

The architecture and the hyperparameters for the simple Multi-Layer Perceptron (MLP) are: in the feed-forward direction, there are the *input layer*, the *fully connected hidden layer with dropout ensemble*, and then the *output layer*. The hyperparameter search space is *number of hidden neurons* (range $[10, 784]$), *learning rate* ($[0.0001, 4)$), *dropout rate* ($[0.1, 0.9)$), *batch*

*size* ($[10, 500]$), *number of iterations* ($[30, 243]$). The settings are for Figure 10a.

### AlexNet CNN for SVHN

The architecture of this CNN and the corresponding hyperparameters are summarized in Tables 1 and 2 below. The settings are for Figure 10b.

| Layer | Hyperparameters | values |
|---|---|---|
| Conv1 | conv1-kernel-size | * |
| | conv1-number-of-channels | 200 |
| | conv1-stride-size | (1,1) |
| | conv1-padding | "same" |
| MaxPooling1 | pooling1-size | (3,3) |
| | pooling1-stride | (1,1) |
| | pooling1-padding | "same" |
| Conv2 | conv2-kernel-size | * |
| | conv2-number-of-channels | 200 |
| | conv2-stride-size | (1,1) |
| | conv2-padding | "same" |
| MaxPooling2 | pooling2-size | (3,3) |
| | pooling2-stride | (2,2) |
| | pooling2-padding | "same" |
| Conv3 | conv3-kernel-size | (3,3) |
| | conv3-number-of-channels | 200 |
| | conv3-stride-size | (1,1) |
| | conv3-padding | "same" |
| AvgPooling3 | pooling3-size | (3,3) |
| | pooling3-stride | (1,1) |
| | pooling3-padding | "same" |
| Dense | batch-normalization | default |
| | number-of-hidden-units | 512 |
| | dropout-rate | 0.5 |

Table 1: CNN architecture for SVHN experiments. A value with * means that this parameter is tuned, and the batch-normalization layer uses all Tensorflow's default setting.

### AlexNet CNN for CIFAR-10

The architecture of this AlexNet-type CNN [Krizhevsky et al., 2012] and the corresponding hyperparameters are summarized in Table 3 and 4 below. The settings are for Figure 10c.

| Hyperparameters | Range |
|---|---|
| conv1-kernel-size | $\{1, 2, \cdots, 7\}$ |
| conv2-kernel-size | $\{1, 2, \cdots, 7\}$ |
| $\beta_1$ | $\{0, 0.05, \cdots, 1\}$ |
| $\beta_2$ | $\{0, 0.05, \cdots, 1\|$ |
| learning-rate | 1e-6 to 5 |
| training-iteration | $\{300, 400, \cdots, 1500\}$ |

Table 2: Hyperparameter search space for the SVHN experiments. $\beta_1$ and $\beta_2$ are parameters for the Adam optimizer [Kingma and Ba, 2014]. The learning rate is discretized in the following way: from 1e-6 to 1.08, we log-space the learning rate into 50 points, and from 1.08 to 5, we linear-space the learning rate into 49 points.

| Layer | Hyperparameters | values |
|---|---|---|
| Conv1 | conv1-kernel-size | * |
| | conv1-no.-of-channels | 200 |
| | conv1-stride-size | (1,1) |
| | conv1-padding | "same" |
| MaxPooling1 | pooling1-size | * |
| | pooling1-stride | (1,1) |
| | pooling1-padding | "same" |
| Conv2 | conv2-kernel-size | * |
| | conv2-no.-of-channels | 200 |
| | conv2-stride-size | (1,1) |
| | conv2-padding | "same" |
| MaxPooling2 | pooling2-size | * |
| | pooling2-stride | (2,2) |
| | pooling2-padding | "same" |
| Conv3 | conv3-kernel-size | * |
| | conv3-no.-of-channels | 200 |
| | conv3-stride-size | (1,1) |
| | conv3-padding | "same" |
| AvgPooling3 | pooling3-size | * |
| | pooling3-stride | (1,1) |
| | pooling3-padding | "same" |
| Dense | batch-normalization | default |
| | no.-of-hidden-units | 512 |
| | dropout-rate | 0.5 |

Table 3: CNN architecture for CIFAR-10. A value with * means that this parameter is tuned, and the batch-normalization layer uses all Tensorflow's default setting.

| Hyperparameters | Range |
|---|---|
| conv1-kernel-size | $\{1, 2, \cdots, 7\}$ |
| conv2-kernel-size | $\{1, 2, \cdots, 7\}$ |
| conv3-kernel-size | $\{1, 2, 3\}$ |
| pooling1-size | $\{1, 2, 3\}$ |
| pooling2-size | $\{1, 2, 3\}$ |
| pooling3-size | $\{1, 2, \cdots, 6\}$ |
| $\beta_1$ & $\beta_2$ | $\{0, 0.05, \cdots, 1\}$ |
| learning-rate | 1e-6 to 5 |
| learning-rate-redeuction | {1,2,3} |
| training-iteration | $\{200, 400, \cdots, 3000\}$ |

Table 4: Hyperparameter search space for the CIFAR-10 experiments. $\beta_1$ and $\beta_2$ are parameters for the Adam optimizer. The learning rate is discretized in the following way: from 1e-6 to 1.08, we log-space the learning rate into 50 points, and from 1.08 to 5, we linear-space the learning rate into 49 points. The learning-rate-reduction parameter is how many times the learning rate is going to be reduced by a factor of 10. For example, if the total training iteration is 200, the learning-rate is 1e-6, and the learning-rate-reduction is 1, then for the first 100 iteration the learning rate is 1e-6, and the for last 100 iterations the learning rate is 1e-7.

# 3 The Analysis of Performance Measure in Q Learning [3]

Compared to model-based Reinforcement Learning (RL) approaches, model-free RL algorithms, such as $Q$-learning, require less space and are more expressive since specifying value functions or policies is more flexible than specifying the model for the environment. This makes model-free algorithms more prevalent in modern deep RL. However, model-based methods can more efficiently extract the information from available data than the model-free approaches. UCB bandit can improve the exploration bonuses and hence increase the data efficiency in the $Q$-learning framework. The cumulative regret of $Q$-learning algorithm with Upper Confidence Bound (UCB) exploration policy in the episodic Markov Decision Process (MDP) has been recently explored in the underlying environment of finite time horizon and finite state-action space [Jin et al., 2018]. In this chapter, we study the regret bound of the $Q$-learning algorithm with UCB exploration in the scenario of compact state-action metric space. We present an algorithm that adaptively discretizes the continuous state-action space and iteratively updates $Q$-values. The algorithm is able to efficiently optimize rewards and minimize the cumulative regret. We provide a rigorous analysis of bounding the regret with concentration of measure analysis and combinatorial approaches. This is the first result of this kind to the best of our knowledge.

## 3.1 Preliminaries

### 3.1.1 Markov Decision Process

Markov Decision Process (MDP) is a discrete-time sequential decision-making problem. In MDP, there is a finite set of states $\mathcal{S}$ and a finite set of actions $\mathcal{A}$. At time step $t$, the agent is at some state $s \in \mathcal{S}$, and it takes some action $a \in \mathcal{A}$. By the state transition probability function $\mathcal{P}$, the agent moves to some new state $s' \in \mathcal{S}$ at time $t+1$ where $s' \sim \mathcal{P}(\cdot|s,a)$. The agent will then receive some immediate reward $r(s',a)$, the reward after transitioning to state $s'$, due to action $a$. In MDP, for the certain pair $(s,a)$, the state transition process satisfies the Markov property. That means the process is conditionally independent of all of the past states and actions. For example, if there exists only one action in each state (fixed transition probability in each state) and all the rewards are zeros, then MDP reduces to Markov Chain.

As introduced in chapter 1, policy $\pi$ is a map from state to action ($\pi : \mathcal{S} \to \mathcal{A}$). If the environment has finite time steps, then we call it a finite time horizon. Otherwise, the environment is an infinite time horizon. The core problem in MDP is to find an optimal policy that can maximize the cumulative function of random rewards over finite/infinite time

[3] A small portion of this chapter is based on the collaboration with Debsoumya Chakraborti. The author is grateful to Tianyu Wang for his comments and insights.

horizon. The typical cumulative function of random reward in the finite time horizon is a sum of rewards. For example, for specific policy $\pi$ at with finite time horizon $H$, it is

$$\mathbb{E}_{s_{t+1}\sim\mathcal{P}(\cdot|s_t,\pi(s_t))}\left[\sum_{t=0}^{H} r(s_{t+1},\pi(s_t))\right].$$

In chapter 1, we mentioned the discount factor for the infinite time horizon scenario without details. The discount factor is in favor of earlier actions. The typical cumulative function in infinite time horizon is a discounted sum of rewards. For example, for policy $\pi$, it is

$$\mathbb{E}_{s_{t+1}\sim\mathcal{P}(\cdot|s_t,\pi(s_t))}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r(s_{t+1},\pi(s_t))\right],$$

where $\gamma$ is the discount factor $0 < \gamma < 1$.

MDP contains two important functions: the reward function $r$ and state transition function $\mathcal{P}$. If we know both functions, that is, we can predict which reward will be received and which states will be in the next transition, then MDP can be solved by Dynamic Programming (DP). DP requires the full description of MDP, with known transition probabilities and reward functions.

### 3.1.2 Model-based and Model-free Reinforcement Learning

However, for most cases, reward function $r$ and state transition function $\mathcal{P}$ are not precisely known. The purpose of RL is to solve MDP when the reward and transition functions are unknown.

One approach in RL is to learn a model of how the environment works from its observation and then find a solution using that model. For example, with a new state-action pair and the observation of the corresponding reward, we can update the estimate of $r$ and $\mathcal{P}$. With the simulation of different actions in many states, we can have a learned transition function and learned reward function with high confidence. If we adequately model the environment, then we can use the learned model to find policy. This approach is called model-based RL. However, model-based RL requires too many trials to learn a particular task independent of the choice of policy iteration or value iteration. Thus, model-based RL methods are often largely inapplicable to large-scale systems. On the other hand, we do not need to learn a model but can directly find a policy. For example, $Q$-learning algorithm is one approach that learns MDP and solves it to find the optimal policy at the same time. $Q$-learning directly estimates the optimal $Q$-values for each action in each state, and the policy could be derived by choosing the action with the highest $Q$-value in the given state. Since we do not learn a

model of the environment, we call it model-free RL. The core idea of $Q$-learning is to balance the exploration and exploitation trade-off: trying random actions to explore the underlying MDP, meanwhile following the so-far optimal policy to maximize the long-term reward.

## 3.2 Introduction

Reinforcement Learning (RL), as a learning problem to maximize the numerical reward, learns how to map situations to actions in unknown underlying system dynamics [Sutton and Barto, 1998]. While model-based RL can approach high-quality and effective decisions, the stability issue about scaling up to the approximate setting with high-dimensional state and action spaces has been demonstrated [Asadi et al., 2018, Arulkumaran et al., 2017]. Besides, due to the assumption of the model, model-based RL suffers from model bias especially when the samples are few and informative prior knowledge is limited [Schaal, 1997, Schneider, 1997, Deisenroth and Rasmussen, 2011]. On the contrary, model-free RL algorithms can directly update the value and policy functions without any assumption on the model environment [Sutton and Barto, 1998]. As the most classical model-free RL algorithm, $Q$-learning was introduced by Watkins in his Ph.D. dissertation. Watkins designs the action-reply MDP algorithm, which achieves the convergence to optimal $Q$ function with finite state and action space setting [Watkins, 1989, Watkins and Dayan, 1992]. This approach is considered as a simpler way of learning optimality in a controlled Markov domain. From the classical $Q$-learning algorithm to modern DQN [Mnih et al., 2013], A3C [Mnih et al., 2016], and TRPO [Schulman et al., 2015], most state-of-the-art RL are in the model-free approach. Model-free RL algorithms require less space and are more expressive since specifying value functions or policies are more flexible compared to specifying the model of the environment. The advantages of model-free methods underly its success in deep RL applications [Jin et al., 2018].

However, model-based methods are generally more promising to efficiently extract the information from available data than the model-free approaches such as $Q$-learning [Deisenroth and Rasmussen, 2011, Schulman et al., 2015]. The requirement of data efficiency in model-free RL is hence increasing. The key to achieve good data efficiency generally lies in the trade-off between exploration and exploitation. Using bandit to explore the uncertain environment while maximizing the reward has been imported to obtain optimal data efficiency [Agrawal and Jia, 2017, Azar et al., 2017, Jin et al., 2018]. However, $Q$-learning with a common $\epsilon$-greedy exploration strategy takes exponentially many episodes to learn, which is very inefficient [Jin et al., 2018]. Upper Confidence Bound (UCB) algorithm, as a multi-armed bandit algorithm, assigns exploration bonuses in many stochastic settings [Kleinberg et al., 2008, Bubeck et al., 2011, Auer and Ortner, 2010]. UCB exploration policy is hence considered as a tool for

improving the exploration bonuses in $Q$-learning [Jin et al., 2018]. This approach can improve data efficiency while leveraging the advantages of model-free methods. However, $Q$-learning algorithm with UCB exploration is restricted to the environment of finite state and action space [Watkins, 1989, Strehl et al., 2006, Jin et al., 2018].

Due to the natural settings of RL applications, RL over underlying MDP with the environment of continuous state and action space has substantially been an intractable and challenging problem [Szepesvari and Smart, 2004, Tsitsiklis and van Roy, 1996]. For example, most robotic applications of RL require continuous state spaces defined utilizing continuous variables such as position, velocity, and torque, etc. Although $Q$-learning is commonly applied to problems with discrete states and actions, it has been extended to continuous space under different scenarios [Gaskett et al., 1999, Irpan, 2015]. This motivates us to extend the state-of-the-art $Q$-learning algorithm equipped with UCB exploration policy to continuous state-action space environment. In modern RL, the algorithm that can leverage the advantages of model-free methods and data efficiency in the continuous state-action space is demanding.

Our contributions are: **1)** Develop a state-of-the-art $Q$-learning algorithm with UCB exploration policy to compact state and action metric space and **2)** Achieve a tight bound for the cumulative regret for the underlying algorithm using the covering dimension techniques. We provide a rigorous analysis of bounding the regret with concentration of measure analysis and combinatorial approaches. For bounding the cumulative regret (performance measure), covering dimension on the regret bound has been suggested [Kleinberg et al., 2008, Bubeck et al., 2011, Slivkins, 2014]. This is the first analysis of this kind to the best of our knowledge.

## 3.3 Notations and Settings

### 3.3.1 $Q$-Learning

In this chapter, we attack the $Q$-learning data efficiency problem in the setting of the episodic MDP formalism. In this setting, an episode consists of a run of MDP dynamics for $H$ steps, where the agent aims to maximize total reward over multiple episodes. This setup follows from [Jin et al., 2018]. We consider a setting of episodic MDP $(\mathcal{S}, \mathcal{A}, H, \mathcal{P}, r)$, where state space is a compact metric space $(\mathcal{S}, d_{\mathcal{S}})$, and action space is also a compact metric space $(\mathcal{A}, d_{\mathcal{A}})$. $\mathcal{S} \times \mathcal{A}$ is a product compact metric space with metric $d_{\mathcal{S}\times\mathcal{A}}$. $H$ is the number of steps in each episode. $\mathcal{P}$ is the state transition function where the transitional probability from the current state-action pair in $\mathcal{S} \times \mathcal{A}$ to the next state is $x' \sim \mathcal{P}(\cdot|x, a)$. The reward function at step $h \in [H]$, is $r_h : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Since we discuss the finite time horizon $Q$-learning here, there is no discount factor included in the value function.

Let the policy function be $\pi : \mathcal{S} \to \mathcal{A}$. We denote by $V_h^\pi : \mathcal{S} \to \mathbb{R}$ the value function at

step $h$ under policy $\pi$. $V_h^\pi$ stands for the expected sum of remaining rewards collected from $x_h = x$ to the end of episode given by policy $\pi$. Mathematically,

$$V_h^\pi(x) := \mathbb{E}\left[\sum_{h'=h}^{H} r_{h'}(x_{h'}, \pi_{h'}(x_{h'}))|x_h = x\right].$$

Accordingly, we can define $Q_h^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as the $Q$-value function at step $h$ under policy $\pi$. This gives the expected sum of the remaining rewards starting from $x_h = x, a_h = a$ to the end of episode. Mathematically,

$$Q_h^\pi(x,a) := r_h(x,a) + \mathbb{E}\left[\sum_{h'=h+1}^{H} r_{h'}(x_{h'}, \pi_{h'}(x_{h'}))|x_h = x, a_h = a\right].$$

For simplicity, we define the expected future value

$$\mathcal{P}_h V_h(x,a) := \mathbb{E}_{x' \sim P(\cdot|x,a)} V_{h+1}(x').$$

The empirical counterpart of episode $k$, say $x_{h+1}^k$ the next observed state from $(x,a)$, is

$$\hat{\mathcal{P}}_h^k V_h(x,a) := V_{h+1}(x_{h+1}^k). \tag{3.1}$$

The above notations follow from [Jin et al., 2018]. Define the optimal policy $\pi^*$ by the optimal value $V_h^*(x) = \sup_\pi V_h^\pi(x)$. Now recall from chapter 1, Bellman equation for any (deterministic) policy $\pi$ is

$$\begin{cases} V_h^\pi(x) = Q_h^\pi(x, \pi(x)) \\ Q_h^\pi(x,a) = r_h(x,a) + \mathcal{P}_h V_h^\pi(x,a). \end{cases}$$

In addition, $V^*$ and $Q^*$ are the value and $Q$-value functions associated with the optimal policy:

$$\begin{cases} V_h^*(x) = \sup_{a \in \mathcal{A}} Q_h^*(x,a) \\ Q_h^*(x,a) = r_h(x,a) + \mathcal{P}_h V_h^*(x,a). \end{cases} \tag{3.2}$$

### 3.3.2 Lipschitz Assumption

In this work, we assume $V^*$ is Lipschitz continuous with Lipschitz constant $D$. Assume the next step from the current state-action pair is Lipschitz continuous. That is, let $x_{h+1}$ be the next step from $(x_h, a_h)$ and $y_{h+1}$ is the next step from $(y_h, b_h)$. Then $d_{\mathcal{S}}(x_{h+1}, y_{h+1}) \leq$

$M \cdot d_{\mathcal{S}\times\mathcal{A}}((x_h, a_h), (y_h, b_h))$ with probability 1. This implies that $\hat{\mathcal{P}}V^*$ is also Lipschitz continuous:

$$\begin{aligned}\left\|\hat{\mathcal{P}}_h V_h^*(x_h, a_h) - \hat{\mathcal{P}}_h V_h^*(y_h, b_h)\right\| &= \left\|V_{h+1}^*(x_{h+1}) - V_{h+1}^*(y_{h+1})\right\| \\ &\leq D \cdot d_{\mathcal{S}}(x_{h+1}, y_{h+1}) \leq D \cdot M \cdot d_{\mathcal{S}\times\mathcal{A}}((x_h, a_h), (y_h, b_h)) \\ &= L \cdot d_{\mathcal{S}\times\mathcal{A}}((x_h, a_h), (y_h, b_h)) \end{aligned} \tag{3.3}$$

for $L = D \cdot M$. The Lipschitz constant is $L$ for $\hat{\mathcal{P}}V^*$.

### 3.3.3 Performance Measure

Our main interest is to provide a practical algorithm that can bound the regret. Assume $K$ is the total number of episodes. At the beginning of each episode $k$, the adversary picks a starting state $x_1^k$. The agent will choose a corresponding greedy policy $\pi_k$ before the starting of episode $k$. Then the cumulative regret (performance measure) of $K$ episodes is

$$\texttt{Regret}(K) := \sum_{k=1}^{K} \left(V_1^*(x_1^k) - V_1^{\pi_k}(x_1^k)\right) . \tag{3.4}$$

Recall $V_h^{\pi_k}$ is the value function of $\pi_k$ in episode $k$.

### 3.3.4 Iterative Updating Rule

As mentioned in the introduction, $Q$-learning with UCB exploration strategy is data-efficient. Since $\mathcal{S} \times \mathcal{A}$ is a compact metric space, running the algorithm will partition the space into a discrete framework, e.g., balls. In the algorithm, for $(x, a) \in \mathcal{S} \times \mathcal{A}$ at episode $k$, we look at all the past state-action pairs that are in the same ball of $(x, a)$, $B(x, a)$, from episode up to $k-1$. Assume at past episode $k_i \in \{1, \cdots, k-1\}$ and step $h$, $\left(x_h^{k_i}, a_h^{k_i}\right) \in B(x, a)$ was selected. We call $\left\{\left(x_h^{k_i}, a_h^{k_i}\right)\right\}$ active points. We denote by $n_h^k(x, a)$ the total number of state-action pairs $\left\{\left(x_h^{k_i}, a_h^{k_i}\right)\right\}$ in $B(x, a)$ at step $h$ up to episode $k$. Also its observed state at next step $h+1$ is $x_{h+1}^{k_i}$ for episode $k_i$ .

We denote by $Q_h^k$, $V_h^k$, $n_h^k$ as $Q_h$, $V_h$, and $n_h$ at the beginning of episode $k$.

**Definition 3.1** (Iterative updating rule of $Q_h^k$)**.**

$$Q_h^{k+1}(x, a) = \begin{cases} (1-\alpha_t)Q_h^k(x, a) + \alpha_t[r_h(x, a) + V_{h+1}^k(x_{h+1}^k) + b_t] & \text{if } (x_h^k, a_h^k) \text{ in } B(x, a)\,; \\ Q_h^k(x, a) & \text{otherwise}\,. \end{cases} \tag{3.5}$$

Here $t$ is the counter for how many times the algorithm has visited the $B(x,a)$ at step $h$, so $t = n_h^k(x,a)$. For the choice of learning rate $\alpha_t$ in the updating rule, we choose $\alpha_t := \frac{1}{t}$.

Accordingly, the iterative updating rule for $V_h^k$ is

$$V_h^k(x) \leftarrow \min\left\{H, \sup_{a' \in \mathcal{A}} Q_h^k(x,a')\right\}, \forall x \in \mathcal{S}\,. \tag{3.6}$$

Based on the learning rate $\alpha_t$, we can define the following related quantities:

$$\alpha_t^0 = \prod_{j=1}^{t}(1-\alpha_j)\,, \qquad \alpha_t^i = \alpha_i \cdot \prod_{j=i+1}^{t}(1-\alpha_j)\,. \tag{3.7}$$

The above definition of the learning rate yields the following properties that are useful in our analysis.

**Lemma 3.2.** *The following properties hold for $\alpha_t$ and $\alpha_t^i$.*

*1.* $\sum_{t=1}^{\infty}\alpha_t = \infty$ *and* $\sum_{t=1}^{\infty}\alpha_t^2 < \infty\,,$ *2.* $\frac{1}{\sqrt{t}} \le \sum_{i=1}^{t}\frac{\alpha_t^i}{\sqrt{i}} \le \frac{2}{\sqrt{t}}\,.$

*Proof.* The first property is trivial. We will prove the second property by induction on $t$.

Let us check the right hand side. Clearly the statement is true for $t = 1$. Assume the result holds for $t-1$, i.e., $\sum_{i=1}^{t-1}\frac{\alpha_{t-1}^i}{\sqrt{i}} \le \frac{2}{\sqrt{t-1}}$ and noting that $\alpha_t^i = \alpha_t = \frac{1}{t}$. We have by using induction hypothesis,

$$\begin{aligned}
\sum_{i=1}^{t}\frac{\alpha_t^i}{\sqrt{i}} &= \frac{\alpha_t^t}{\sqrt{t}} + (1-\alpha_t)\sum_{i=1}^{t-1}\frac{\alpha_{t-1}^i}{\sqrt{i}} \\
&= \frac{\alpha_t}{\sqrt{t}} + (1-\alpha_t)\sum_{i=1}^{t-1}\frac{\alpha_{t-1}^i}{\sqrt{i}} \\
&\le \frac{\alpha_t}{\sqrt{t}} + (1-\alpha_t)\frac{2}{\sqrt{t-1}}\,.
\end{aligned}$$

It remains to prove that $\frac{\alpha_t}{\sqrt{t}} + (1-\alpha_t)\frac{2}{\sqrt{t-1}} \le \frac{2}{\sqrt{t}}$, i.e. $\frac{2(\sqrt{t}-\sqrt{t-1})}{2\sqrt{t}-\sqrt{t-1}} \le \alpha_t = \frac{1}{t}$. We have

$$\begin{aligned}
\frac{2\sqrt{t}-\sqrt{t-1}}{2(\sqrt{t}-\sqrt{t-1})} &= \frac{1}{2}\left(\sqrt{t}+\sqrt{t-1}\right)\left(2\sqrt{t}-\sqrt{t-1}\right) \\
&= \frac{1}{2}\left(t+1+\sqrt{t(t-1)}\right) \\
&\ge \frac{1}{2}\left(t+1+t-1\right) \\
&= t\,.
\end{aligned}$$

Thus, we obtain the right hand side (upper bound). Using the same induction process, the left hand side (lower bound) statement is equivalent to showing $\frac{\alpha_t}{\sqrt{t}} + (1-\alpha_t)\frac{1}{\sqrt{t-1}} \geq \frac{1}{\sqrt{t}}$, and this is trivial since $\alpha_t = \frac{1}{t}$.

□

By the updating rule for $Q$ in definition 3.1, and definition of learning rate (3.7), then

$$Q_h^k(x,a) = \alpha_t^0 \cdot H + \sum_{i=1}^{t} \alpha_t^i [r_h(x,a) + V_{h+1}^{k_i}(x_{h+1}^{k_i}) + b_i] \,. \tag{3.8}$$

## 3.4 Performance Measure Algorithms and Theorems

Next, we present the main algorithm and the main theorem. Recall that for the $Q$-learning in our setting of the episodic MDP formalism, each episode consists of a run of MDP dynamics for $H$ steps.

**Algorithm 6** Q-learning with UCB bandit in metric space

initialize ball radius $r \leftarrow 1$ and initial covering $\mathcal{B}$ of $\mathcal{S} \times \mathcal{A}$. Initialize $Q_h(x,a) \leftarrow H$ and $n_h(x,a) \leftarrow 0$ for all $(x,a)$, and $V_h(x) \leftarrow H$ for all $x$. Choose $p \in (0,1)$ where $\iota := \log\left(\frac{K}{p}\right)$.
**for** $k = 1, 2, \cdots K$ **do**
  receive $x_1$ from randomly sampling.
  **for** $h = 1, 2, \cdots, H$ **do**
    **if** $r \leq \frac{c}{8L}\sqrt{\frac{H^3\iota}{1+n_h(x_h,a_h)}}$ is violated **then**
      for the violated ball $B(x_h, a_h)$, $r \leftarrow \frac{r}{2}$ while keeping the same center.
      **while** there exists some region in $\mathcal{S} \times \mathcal{A}$ not covered by $\mathcal{B}$ **do**
        $\mathcal{B} \leftarrow \mathcal{B} \cup B_r(x,a)$
    update $n_h(x_h,a_h) \leftarrow n_h(x_h,a_h) + 1$.
    take action $a_h \leftarrow \arg\max_{a' \in \mathcal{A}} Q_h(x_h,a')$ and observe $x_{h+1}$.
    iteratively update $Q_h(x,a)$ and $V_h(x)$ if $(x_h,a_h)$ in $B(x_h,a_h)$:

$$Q_h(x_h,a_h) \leftarrow (1-\alpha_t)Q_h(x_h,a_h) + \alpha_t[r_h(x_h,a_h) + V_{h+1}(x_{h+1}) + b_t] \,.$$

$$V_h(x_h) \leftarrow \min\left\{H, \sup_{a' \in \mathcal{A}} Q_h(x_h,a')\right\} \,.$$

The main theorem is a bound for the cumulative regret in (3.4).

**Theorem 3.3.** *For any $p \in (0,1)$, if we choose bandit $b_t = c\sqrt{\frac{H^3\iota}{t}}$ where $\iota := \log\left(\frac{K}{p}\right)$ and some absolute constant $c$ (shown in the later analysis), then under certain conditions (shown in the later chapter), the Q-learning with UCB exploration algorithm (in Algorithm 6) has*

*the performance measure* $\texttt{Regret}(K) = \sum_{k=1}^{K}(V_1^*(x_1^k) - V_1^{\pi_k}(x_1^k))$ *bounded above by* $\tilde{\mathcal{O}}(K^{\frac{d+1}{d+2}})$ *with probability* $1-p$*, where* $d$ *is the covering dimension of the underlying* $\mathcal{S}\times\mathcal{A}$*.*

In the following sections, we prove some very useful techniques to bound the performance measure. We provide a novel rigorous analysis that can be valuable in the analysis of $Q$-learning problem in general.

## 3.5 Main Analysis

### 3.5.1 Concentration of Measure

First, we use the following lemmas to bound $\left(Q^k - Q^*\right)$ with high concentration.

Recall that the chosen bandit $b_t = c\sqrt{\frac{H^3\iota}{t}}$. Let $\beta_t := \sum_{i=1}^{t}\alpha_t^i b_i$. Next-step empirical observation for $(x,a)$ at step $h$ in episode $k$ is $\hat{\mathcal{P}}_h^k V_h(x,a) := V_{h+1}(x_{h+1}^k)$.

**Lemma 3.4.** *Let* $(x,a,h,k)\in\mathcal{S}\times\mathcal{A}\times[H]\times[K]$*,* $t = n_h^k(x,a)$*, and suppose* $B(x,a)$ *was selected previously at step* $h$ *of episodes* $\{k_i\}_{i=1}^t$*. We have*

$$\begin{aligned}(Q_h^k - Q_h^*)(x,a) = \alpha_t^0\left(H - Q_h^*(x,a)\right) + \sum_{i=1}^{t}\alpha_t^i\Big[\left(V_{h+1}^{k_i} - V_{h+1}^*\right)(x_{h+1}^{k_i}) \\ + \left(\hat{\mathcal{P}}_h^{k_i} - \mathcal{P}_h\right)V_h^*(x,a) + \left(\hat{\mathcal{P}}_h^{k_i}V_h^*(x_h^{k_i},a_h^{k_i}) - \hat{\mathcal{P}}_h^{k_i}V_h^*(x,a)\right) + b_i\Big]\,.\end{aligned} \tag{3.9}$$

*Proof.* From the Bellman optimality equation (3.2), we have

$$Q_h^*(x,a) = (r_h + \mathcal{P}_h V_h^*)(x,a)\,. \tag{3.10}$$

Using the fact that $\sum_{i=0}^{t}\alpha_t^i = 1$ from the definition of learning rate, we have

$$\begin{aligned}Q_h^*(x,a) &= \alpha_t^0 Q_h^*(x,a) + \sum_{i=1}^{t}\alpha_t^i\cdot Q_h^*(x,a) \\ &= \alpha_t^0 Q_h^*(x,a) + \sum_{i=1}^{t}\alpha_t^i\left[r_h(x,a) + \mathcal{P}_h V_h^*(x,a)\right]\,.\end{aligned} \tag{3.11}$$

From (3.8), this implies

$$(Q_h^k - Q_h^*)(x,a) = \alpha_t^0\left(H - Q_h^*(x,a)\right) + \sum_{i=1}^{t}\alpha_t^i\left[V_{h+1}^{k_i}\left(x_{h+1}^{k_i}\right) - \mathcal{P}_h V_h^*(x,a) + b_i\right]\,, \tag{3.12}$$

which is equivalent to

$$
\begin{aligned}
(Q_h^k - Q_h^*)(x,a) &= \alpha_t^0 \left(H - Q_h^*(x,a)\right) + \sum_{i=1}^{t} \alpha_t^i \left[V_{h+1}^{k_i}\left(x_{h+1}^{k_i}\right) - V_{h+1}^*(x_{h+1}^{k_i}) + V_{h+1}^*(x_{h+1}^{k_i})\right] \\
&\quad + \sum_{i=1}^{t} \alpha_t^i \left[-\mathcal{P}_h V_h^*(x,a) + \hat{\mathcal{P}}_h^{k_i} V_h^*(x,a) - \hat{\mathcal{P}}_h^{k_i} V_h^*(x,a) + b_i\right] \\
&= \alpha_t^0 \left(H - Q_h^*(x,a)\right) + \sum_{i=1}^{t} \alpha_t^i \left[\left(V_{h+1}^{k_i} - V_{h+1}^*\right)\left(x_{h+1}^{k_i}\right) + \left(\hat{\mathcal{P}}_h^{k_i} - \mathcal{P}_h\right) V_h^*(x,a) + b_i\right] \\
&\quad + \sum_{i=1}^{t} \alpha_t^i \left[V_{h+1}^*\left(x_{h+1}^{k_i}\right) - \hat{\mathcal{P}}_h^{k_i} V_h^*(x,a)\right] . \qquad (3.13)
\end{aligned}
$$

Note that $x_{h+1}^{k_i}$ is the next step from $\left(x_h^{k_i}, a_h^{k_i}\right)$ where $\left(x_h^{k_i}, a_h^{k_i}\right) \in B(x,a)$ by the updating rule definition 3.1. Also note that $V_{h+1}^*(x_{h+1}^{k_i}) - \hat{\mathcal{P}}_h^{k_i} V_h^*(x,a)$ satisfies

$$
\begin{aligned}
&V_{h+1}^*\left(x_{h+1}^{k_i}\right) - \hat{\mathcal{P}}_h^{k_i} V_h^*(x,a) \\
&= \left(V_{h+1}^*\left(x_{h+1}^{k_i}\right) - \hat{\mathcal{P}}_h^{k_i} V_h^*\left(x_h^{k_i}, a_h^{k_i}\right)\right) + \left(\hat{\mathcal{P}}_h^{k_i} V_h^*\left(x_h^{k_i}, a_h^{k_i}\right) - \hat{\mathcal{P}}_h^{k_i} V_h^*(x,a)\right) \\
&= \hat{\mathcal{P}}_h^{k_i} V_h^*\left(x_h^{k_i}, a_h^{k_i}\right) - \hat{\mathcal{P}}_h^{k_i} V_h^*(x,a) . \qquad (3.14)
\end{aligned}
$$

This is true since $V_{h+1}^*\left(x_{h+1}^{k_i}\right) = \hat{\mathcal{P}}_h^{k_i} V_h^*\left(x_h^{k_i}, a_h^{k_i}\right)$ by (3.1). Combining (3.13) and (3.14), we have Lemma 3.4.

□

To complete the concentration result of $Q^k - Q^*$, we need to bound each term in (3.9).

**Lemma 3.5.** $\left|\sum_{i=1}^{t} \alpha_t^i \left(\hat{\mathcal{P}}_h^{k_i} V_h^*(x,a) - \hat{\mathcal{P}}_h^{k_i} V_h^*\left(x_h^{k_i}, a_h^{k_i}\right)\right)\right| \leq \frac{\beta_t}{4}$.

*Proof.* Note that

$$
\beta_t = \sum_{i=1}^{t} \alpha_t^i b_i = \sum_{i=1}^{t} \alpha_t^i c\sqrt{\frac{H^3\iota}{i}} = c\sqrt{H^3\iota} \sum_{i=1}^{t} \frac{\alpha_t^i}{\sqrt{i}} \geq c\sqrt{\frac{H^3\iota}{t}} . \qquad (3.15)
$$

Since $\left\{\left(x_h^{k_i}, a_h^{k_i}\right)\right\}_{i=1}^{t} \in B(x,a)$, the distance between $\left(x_h^{k_i}, a_h^{k_i}\right)$ and $(x,a)$ is bounded by the radius of the ball. Intuitively, as algorithm 6 goes on, it keeps refining balls and hence the bound will be smaller. Due to the Lipschitz assumption of $\hat{\mathcal{P}}V^*$ in (3.3), $\left\|\hat{\mathcal{P}}_h^{k_i} V_h^*(x,a) - \hat{\mathcal{P}}_h^{k_i} V_h^*(x',a')\right\| \leq L \cdot d_{\mathcal{S}\times\mathcal{A}}\left((x,a),(x',a')\right)$, and follows from algorithm 6,

$$\begin{aligned}\left|\sum_{i=1}^{t}\alpha_t^i\left(\hat{\mathcal{P}}_h^{k_i}V_h^*(x,a)-\hat{\mathcal{P}}_h^{k_i}V_h^*\left(x_h^{k_i},a_h^{k_i}\right)\right)\right| &\le \sum_{i=1}^{t}\alpha_t^i\left(\frac{c}{8}\sqrt{\frac{H^3\iota}{1+n_h^k(x,a)}}\right)\\ &\le \frac{1}{8}\sum_{i=1}^{t}\alpha_t^i\cdot c\sqrt{\frac{H^3\iota}{i}}\\ &\le \frac{1}{8}\left(2c\sqrt{\frac{H^3\iota}{t}}\right) \qquad (3.16)\\ &\le \frac{\beta_t}{4}\,,\end{aligned}$$

where we have used Lemma 3.2.

□

Now we can show the concentration bound of $Q_h^k - Q_h^*$.

**Lemma 3.6.** *For any $p\in(0,1)$, with probability at least $1-p$, we have*

$$0\le\left(Q_h^k-Q_h^*\right)(x,a)\le\alpha_t^0H+\sum_{i=1}^{t}\alpha_t^i\left(V_{h+1}^{k_i}-V_{h+1}^*\right)\left(x_{h+1}^{k_i}\right)+\frac{3}{2}\beta_t\,.$$

The proof of Lemma 3.6 is similar to Lemma 4.3 in [Jin et al., 2018].

*Proof.* Based on the definition in (3.7),

$$\beta_t=\sum_{i=1}^{t}\alpha_t^ib_i\le\sum_{i=1}^{t}\alpha_ib_i=\sum_{i=1}^{t}\frac{1}{i}c\sqrt{\frac{H^3\iota}{i}}<4c\sqrt{\frac{H^3\iota}{t}}\,.$$

Recall that $t=n_h^k(x,a)$. Denote $k_i=\min\left(\{k\in[K]|k>k_{i-1}\wedge(x_h^k,a_h^k)\in B(x,a)\}\cup\{K+1\}\right)$. That is, $k_i$ is the episode in which $B(x,a)$ was selected at step $h$ for the $i^{th}$ time (or $k_i=K+1$ if it is selected for fewer than $i$ times). The random variable $k_i$ is a stopping time in this case. Let $\mathcal{F}_i$ be the $\sigma$-field generated by all the random variables until episode $k_i$ at step $h$. Then

$$\left(\mathbb{1}_{k_i\le K}\left[(\hat{\mathcal{P}}_h^{k_i}-\mathcal{P}_h)V_h^*\right](x,a)\right)_{i=1}^{t}$$

is a martingale difference sequence with filtration $\{\mathcal{F}_i\}_{i\ge0}$. By the sequence bounded difference, for any $i$,

$$\alpha_t^i\left[(\hat{\mathcal{P}}_h^{k_i}-\mathcal{P}_h)V_h^*\right](x,a)\le\alpha_t^i\cdot H\,. \qquad (3.17)$$

Apply the Azuma-Hoeffding inequality, for any $s > 0$,

$$\mathcal{P}\left(\left|\sum_{i=1}^{t}\alpha_t^i\left[(\hat{\mathcal{P}}_h^{k_i}-\mathcal{P}_h)V_h^*\right](x,a)\right|\le s\right)\ge 1-2\exp\left\{\frac{-s^2}{2H^2\sum_{i=1}^{t}(\alpha_t^i)^2}\right\}. \tag{3.18}$$

Note that there exists some absolute constant $c$ such that

$$\sqrt{2}\cdot H\sqrt{\sum_{i=1}^{t}(\alpha_t^i)^2}\sqrt{\ln\left(\frac{2H}{p}\right)}\le\frac{c}{4}\sqrt{\frac{H^3\iota}{t}}.$$

We claim that if (3.19) holds for fixed step $h$, then we can derive the result:

$$\sqrt{2}\cdot H\sqrt{\sum_{i=1}^{t}(\alpha_t^i)^2}\sqrt{\ln\left(\frac{2H}{p}\right)}\le s\le\frac{c}{4}\sqrt{\frac{H^3\iota}{t}}. \tag{3.19}$$

This is true since **1)**.

$$\beta_t=\sum_{i=1}^{t}\alpha_t^i b_i=\sum_{i=1}^{t}\alpha_t^i c\sqrt{\frac{H^3\iota}{i}}=c\sqrt{H^3\iota}\sum_{i=1}^{t}\frac{\alpha_t^i}{\sqrt{i}}\ge c\sqrt{\frac{H^3\iota}{t}}\ge 4s\,, \tag{3.20}$$

and for **2)**, since $s\ge\sqrt{2}\cdot H\sqrt{\sum_{i=1}^{t}(\alpha_t^i)^2}\sqrt{\ln\left(\frac{2H}{p}\right)}$, then

$$\begin{aligned}
&\frac{s^2}{2H^2\sum_{i=1}^{t}(\alpha_t^i)^2}\ge\ln\left(\frac{2H}{p}\right)\\
&\frac{-s^2}{2H^2\sum_{i=1}^{t}(\alpha_t^i)^2}\le\ln\left(\frac{p}{2H}\right)\\
&\exp\left\{\frac{-s^2}{2H^2\sum_{i=1}^{t}(\alpha_t^i)^2}\right\}\le\frac{p}{2H}\\
&1-2\exp\left\{\frac{-s^2}{2H^2\sum_{i=1}^{t}(\alpha_t^i)^2}\right\}\ge 1-\frac{p}{H}>1-p\,.
\end{aligned} \tag{3.21}$$

From (3.21), (3.20) and (3.18), for any fixed step $h$, we have

$$\mathcal{P}\left(\left|\sum_{i=1}^{t}\alpha_t^i\left[(\hat{\mathcal{P}}_h^{k_i}-\mathcal{P}_h)V_h^*\right](x,a)\right|\le\frac{\beta_t}{4}\right)\ge 1-p\,. \tag{3.22}$$

From (3.22), Lemma 3.4 and 3.5, we have right hand side of Lemma 3.6.

Now for the left hand side, we use the backward induction on $h = H, H-1, \cdots, 1$. We

observe the beginning case $\left(Q_H^k - Q_H^*\right)(x,a) \geq 0$. Assume $\left(Q_h^k - Q_h^*\right)(x,a) \geq 0$. Since

$$V_h^k(x) = \sup_{a' \in \mathcal{A}} Q_h^k(x,a') \geq \sup_{a' \in \mathcal{A}} Q_h^*(x,a') = V_h^*(x)\,, \tag{3.23}$$

then from (3.9) and (3.21), this implies that $\left(Q_{h-1}^k - Q_{h-1}^*\right)(x,a) \geq 0$ with probability at least $1 - \frac{p}{H}$ for one step backward. From the union bound on $h = H, H-1, \cdots, 1$, it is trivial to see left hand side.

We hence have

$$0 \leq \left(Q_h^k - Q_h^*\right)(x,a) \leq \alpha_t^0 H + \sum_{i=1}^{t} \alpha_t^i \left(V_{h+1}^{k_i} - V_{h+1}^*\right)\left(x_{h+1}^{k_i}\right) + \frac{3}{2}\beta_t \tag{3.24}$$

for all $h$ with probability at least $1 - p$.

□

### 3.5.2 Assumptions for Concentration

To establish the bound for the cumulative regret in the main theorem 3.3, we need two mild and reasonable assumptions below:

- $V_{h+1}^{k_i} - V_{h+1}^* \leq R \cdot \sqrt{\frac{H^3 \iota}{i}}$ for some large $R$,
- $\sum_{k=1}^{K} \mathbb{1}_{\{n_k = 0\}} \leq \Theta\left(K^{\frac{d+1}{d+2}}\right)$.

The first assumption is very mild and reasonable due to the iterative convergence. In the second assumption, $\sum_{k=1}^{K} \mathbb{1}_{\{n_k=0\}}$ stands for the number of empty partitioned balls by algorithm from $k = 1$ to $K$. Here the empty partitioned ball means no active state-action pair contained in. The second assumption is again mild since we will soon show the cumulative number of partitions up to $K$, $\mathbf{P}_K$, is bounded above by $\Theta\left(K^{\frac{d}{d+2}}\right)$ in Lemma 3.10. Note that $d$ here is the covering dimension in $\mathcal{S} \times \mathcal{A}$ that will also be introduced in the next section.

Recall that from (3.24),

$$\sum_{k=1}^{K} \left(Q_h^k - Q_h^*\right)(x,a) \leq \sum_{k=1}^{K} \alpha_t^0 H + \sum_{k=1}^{K} \sum_{i=1}^{t} \alpha_t^i \left(V_{h+1}^{k_i} - V_{h+1}^*\right)\left(x_{h+1}^{k_i}\right) + \frac{3}{2}\sum_{k=1}^{K} \beta_t\,. \tag{3.25}$$

Here for any $i = 1, \cdots, n_h^k$ where $n_h^k = t$, $k_i$ is the actual selection. From the first assumption, $V_{h+1}^{k_i} - V_{h+1}^* \leq \frac{R}{c} \cdot b_i$. We hence have

$$\sum_{i=1}^{t} \alpha_t^i \left(V_{h+1}^{k_i} - V_{h+1}^*\right)\left(x_{h+1}^{k_i}\right) \leq \frac{R}{c} \sum_{i=1}^{t} \alpha_t^i b_i = \frac{R}{c} \cdot \beta_t\,. \tag{3.26}$$

In addition, from (3.25), we have a term $\sum_{k=1}^{K} \alpha_t^0 \cdot H$. Recall from (3.7), $\alpha_j = \frac{1}{j}$. Since

$$\alpha_t^0 = \prod_{j=1}^{n_k}(1-\alpha_j) = \begin{cases} 1 & \text{if } n_k = 0\,; \\ 0 & \text{if } n_k > 0\,, \end{cases}$$

then

$$\begin{aligned} \sum_{k=1}^{K} \alpha_t^0 \cdot H &= H \sum_{k=1}^{K} \prod_{j=1}^{t}(1-\alpha_j) \\ &= H \sum_{k=1}^{K} \mathbb{1}_{\{n_k=0\}} \\ &\leq \Theta\left(K^{\frac{d+1}{d+2}}\right), \end{aligned} \tag{3.27}$$

where (3.27) is from the second mild assumption above. Combining (3.25), (3.26), (3.27), and Lemma 3.6, for any $p \in (0,1)$, with probability at least $1-p$, we have

$$\begin{aligned} \sum_{k=1}^{K} \left(Q_h^k - Q_h^*\right)(x,a) &\leq \sum_{k=1}^{K} \left( \alpha_t^0 \cdot H + \sum_{i=1}^{t} \alpha_t^i \left(V_{h+1}^{k_i} - V_{h+1}^*\right)\left(x_{h+1}^{k_i}\right) + \frac{3}{2}\beta_t \right) \\ &\leq \Theta\left(K^{\frac{d+1}{d+2}}\right) + \mu \sum_{k=1}^{K} \beta_t\,, \end{aligned} \tag{3.28}$$

where $\mu = \frac{R}{c} + \frac{3}{2}$. To bound $\left(Q_h^k - Q_h^*\right)$, it remains to bound $\beta_t$.

### 3.5.3 Upper Bound of Cumulative Regret

Now we return to our main discussion about the $\texttt{Regret}(K)$. The following lemma shows the relationship between $\texttt{Regret}(K)$ and $\left(Q_h^k - Q_h^*\right)$.

**Lemma 3.7.** *For any $p \in (0,1)$, with probability at least $1-p$, the cumulative regret up to $K$ has the following upper bound:*

$$\begin{aligned} \sum_{k=1}^{K} V_1^k(x_1^k) - V_1^{\pi_k}(x_1^k) \leq \Theta(K^{\frac{d+1}{d+2}}) + \sum_{j=0}^{H-1} \int_{\texttt{traj}\left(x_h^k, a_h^k\right) \in \texttt{TRAJ}(h \to h+j)} \mathcal{P}\left(\texttt{traj}\left(x_h^k, a_h^k\right)\right) \\ \cdot \left( \sum_{k=1}^{K} \left(Q_{h+j}^k - Q_{h+j}^*\right)(s, a^s) \right) d\mathcal{P}\,. \end{aligned}$$

*Here sequence of actions $\{a^s\}$ is corresponding to sequence of states $\{s\}$, which follows from the maximum action selection in (3.30) and $\texttt{traj}\left(x_h^k, a_h^k\right)$ is the trajectory starting from*

$\left(x_h^k, a_h^k\right)$. *TRAJ*$(h \to h+j)$ *is a trajectory space that contains all the possible trajectories from step* $h$ *up to step* $h+j$. $d$ *is the covering dimension of* $\mathcal{S} \times \mathcal{A}$.

*Proof.* The proof follows the idea from [Dong et al., 2019]. Note that for some action $a_h^k$,

$$V_h^k\left(x_h^k\right) - V_h^{\pi_k}\left(x_h^k\right) \leq \left(Q_h^k\left(x_h^k, a_h^k\right) - Q_h^*\left(x_h^k, a_h^k\right)\right) + \left(Q_h^*\left(x_h^k, a_h^k\right) - Q_h^{\pi_k}\left(x_h^k, a_h^k\right)\right) + \frac{k^{\frac{-1}{d+2}}}{H}. \tag{3.29}$$

The inequality holds because

$$V_h^k\left(x_h^k\right) \leq \sup_{a' \in \mathcal{A}} Q_h^k\left(x_h^k, a'\right) \leq Q_h^k\left(x_h^k, a_h^k\right) + \frac{k^{\frac{-1}{d+2}}}{H} \tag{3.30}$$

for some $a_h^k$ from (3.6). We name it maximum action selection. Also

$$\begin{aligned} Q_h^*\left(x_h^k, a_h^k\right) - Q_h^{\pi_k}\left(x_h^k, a_h^k\right) &= \left(\mathcal{P}_h V_h^*\left(x_h^k, a_h^k\right) - \mathcal{P}_h V_h^{\pi_k}\left(x_h^k, a_h^k\right)\right) \\ &= \int_{s \in \mathcal{S}} \mathcal{P}\left(s | x_h^k, a_h^k\right)\left(V_{h+1}^* - V_{h+1}^{\pi_k}\right)(s) d\mathcal{P}(s), \end{aligned} \tag{3.31}$$

where we take all the states at step $h+1$ into account, $s \sim \mathcal{P}(\bullet | x_h^k, a_h^k)$.

Note that from Lemma 3.6, for any $p$, we have $Q_h^k \geq Q_h^*$ with probability at least $1-p$ for all steps $h$. Then with probability at least $1-p$,

$$V_h^k = \sup_{a \in \mathcal{A}} Q_h^k \geq \sup_{a \in \mathcal{A}} Q_h^* = V_h^*. \tag{3.32}$$

Combining (3.29), (3.31) and (3.32), we have

$$V_h^k\left(x_h^k\right) - V_h^{\pi_k}\left(x_h^k\right) \leq \frac{k^{\frac{-1}{d+2}}}{H} + \left(Q_h^k\left(x_h^k, a_h^k\right) - Q_h^*\left(x_h^k, a_h^k\right)\right) + \int_{s \in \mathcal{S}} \mathcal{P}\left(s | x_h^k, a_h^k\right)\left(V_{h+1}^k - V_{h+1}^{\pi_k}\right)(s) d\mathcal{P}(s). \tag{3.33}$$

We observe that in (3.33), $\left(V_h^k - V_h^{\pi_k}\right)\left(x_h^k\right)$ is on the left hand side and $\left(V_{h+1}^k - V_{h+1}^{\pi_k}\right)(s)$ is in the integral on the right hand side.

Recall the space that contains all the possible trajectories from step $h$ up to step $h+j$ is $\texttt{TRAJ}(h \to h+j)$. We have the following recursive relationship:

$$
\begin{aligned}
&V_h^k(x_h^k) - V_h^{\pi_k}(x_h^k) \\
&\leq \frac{2k^{\frac{-1}{d+2}}}{H} + \left(Q_h^k(x_h^k, a_h^k) - Q_h^*(x_h^k, a_h^k)\right) + \int_{s\in\mathcal{S}} \mathcal{P}\left(s|x_h^k, a_h^k\right) \Bigg( \left(Q_{h+1}^k(s, a^s) - Q_{h+1}^*(s, a^s)\right) \\
&\qquad\qquad + \int_{s'\in\mathcal{S}} \mathcal{P}\left(s'|s, a\right) \left(V_{h+2}^k - V_{h+2}^{\pi_k}\right)(s') d\mathcal{P}(s') \Bigg) d\mathcal{P}\left(s\right) \cdots \\
&\leq k^{\frac{-1}{d+2}} + \sum_{j=0}^{H-h} \int_{\texttt{traj}\left(x_h^k, a_h^k\right)\in\texttt{TRAJ}(h\to h+j)} \left(\mathcal{P}\left(\texttt{traj}\left(x_h^k, a_h^k\right)\right) \cdot \left(Q_{h+j}^k - Q_{h+j}^*\right)(s, a^s)\right) d\mathcal{P}\,,
\end{aligned}
\tag{3.34}
$$

where sequence $\{a^s\}$ corresponding to each $\{s\}$ is the same as our selection in (3.30) and $\texttt{traj}\left(x_h^k, a_h^k\right)$ is the trajectory starting from $\left(x_h^k, a_h^k\right)$.

For simplicity, we use $\left(Q_{h+j}^k - Q_{h+j}^*\right)$ to stand for $\left(Q_{h+j}^k - Q_{h+j}^*\right)(s, a^s)$ and $\texttt{traj}$ to stand for $\texttt{traj}\left(x_h^k, a_h^k\right)$. Then

$$
\sum_{k=1}^{K} V_1^k(x_1^k) - V_1^{\pi_k}(x_1^k) \leq \sum_{k=1}^{K} k^{\frac{-1}{d+2}} + \sum_{k=1}^{K}\sum_{j=0}^{H-1} \int_{\texttt{traj}\in\texttt{TRAJ}(h\to h+j)} \mathcal{P}(\texttt{traj}) \cdot \left(Q_{1+j}^k - Q_{1+j}^*\right) d\mathcal{P}\,.
\tag{3.35}
$$

Assume $E_K = \{1, 2, \cdots, K\}$. If we define $\mu(k) = 1$ for all $k = 1, \cdots, K$, and 0 otherwise, then $\mu$ is a finite counting measure on $E_K$. (3.35) is equivalent to

$$
\sum_{k=1}^{K} V_1^k(x_1^k) - V_1^{\pi_k}(x_1^k) \leq \Theta(K^{\frac{d+1}{d+2}}) + \int_{E_K} \sum_{j=0}^{H-1} \int_{\texttt{traj}\in\texttt{TRAJ}(h\to h+j)} \mathcal{P}(\texttt{traj}) \cdot \left(Q_{1+j}^k - Q_{1+j}^*\right) d\mathcal{P} d\mu\,.
\tag{3.36}
$$

Applying the Fubini's theorem [Folland, 1999] on (3.36), we have

$$
\sum_{k=1}^{K} V_1^k(x_1^k) - V_1^{\pi_k}(x_1^k) \leq \Theta(K^{\frac{d+1}{d+2}}) + \sum_{j=0}^{H-1} \int_{\texttt{traj}\in\texttt{TRAJ}(h\to h+j)} \mathcal{P}(\texttt{traj}) \cdot \left(\sum_{k=1}^{K} \left(Q_{1+j}^k - Q_{1+j}^*\right)\right) d\mathcal{P}\,.
$$

□

## 3.6 Regret Bound in Covering Dimension

We have mentioned the covering dimension in the earlier section. The covering dimension is a useful tool when we need to reduce the upper bound of the regret in multi-armed bandit problems. We will follow the same definition as [Bubeck et al., 2011, Kleinberg et al.,

2008, Slivkins, 2014]. This notion is also the same as the commonly used Lebesgue covering dimension in Topology.

**Definition 3.8.** Consider a metric space $(\mathcal{X}, l)$. Denote by $N(r)$ the minimum number of balls with diameter $r$ to cover $\mathcal{X}$. The covering dimension of $\mathcal{X}$ is defined as $\texttt{COV}(\mathcal{X}) = \inf\{d : \exists c\ \forall r > 0\ N(r) \leq c \cdot r^{-d}\}$.

**Lemma 3.9.** *Let $\mathcal{E}$ be a bounded space that can be embedded into an $n$-dimensional Euclidean space. Then $\texttt{COV}(\mathcal{E}) \leq n$.*

*Proof.* Without loss of generality, we assume $\mathcal{E} \subseteq [0,1]^n$. Now, let $r > 0$ be any real number. Then we can find $\left(\frac{\sqrt{n}}{r}\right)^n$ many $n$-dimensional hypercubes with side-length $\frac{r}{\sqrt{n}}$ covering $[0,1]^n$. Now for every such hypercube there exists a ball with diameter $r$ such that the hypercube is contained inside the ball. Thus, we can have $c \cdot r^{-n}$ many balls with diameter $r$ cover $\mathcal{E}$, where the constant $c = (\sqrt{n})^n$ in this case. Since $\texttt{COV}(\mathcal{E})$ takes the infimum, then $\texttt{COV}(\mathcal{E}) \leq n$. □

Because of Lemma 3.9, to obtain a tighter upper bound, we use the covering dimension of $\mathcal{S} \times \mathcal{A}$ instead of the regular Euclidean dimension. In this section, we use the covering dimension to bound $\sum_{k=1}^{K} \beta_t$. Assume $\mathbf{P}_K$ stands for the number of balls (partitions) created up to episode $K$ in the algorithm 6.

**Lemma 3.10.** *We have the tight bound $\boldsymbol{P}_K \leq \Theta\left(K^{\frac{d}{d+2}}\right)$.*

*Proof.* Assume $d$ is the covering dimension of $\mathcal{S} \times \mathcal{A}$. Then the number of balls with diameter $m$ to cover $\mathcal{S} \times \mathcal{A}$ will be bounded by $c \cdot m^{-d}$ for some $c$. Note that algorithm 6 produces many balls during adaptive partition. The center of the balls (past active points) of radius $\frac{m}{2}$ will always be in different balls from the covering of $\mathcal{S} \times \mathcal{A}$ because each pair of centers have distance at least $m$ between them. For example, say $(s_i, a_i)$ and $(s_j, a_j)$ are two past active centers. Consider the covering of $\mathcal{S} \times \mathcal{A}$. Then $(s_i, a_i) \in B_i$ and $(s_j, a_j) \in B_j$ for $B_i, B_j$ in the covering and they are disjoint. Thus, the number of the balls with diameter $m$ from algorithm 6 is bounded by $c \cdot m^{-d}$, the covering number.

Note that for each ball $B$ of radius $r$ with the center inside $\mathcal{S} \times \mathcal{A}$, we can use at most $c' \cdot 2^n$ balls of radius $\frac{r}{2}$ to cover $B \cap (\mathcal{S} \times \mathcal{A})$, where $n$ is the Euclidean dimension of $\mathcal{S} \times \mathcal{A}$ and $c'$ is some constant. From the algorithm, each ball will split only if $r \leq c\sqrt{\frac{H^3\iota}{1+n_k}}$ is violated. That is, for any ball of radius $\frac{r}{2}$ split from the original ball of radius $r$ (condition violated), it must be filled by $c^2 \cdot \frac{H^3\iota}{r^2}$ past active points. Since there are $K$ points in total, then the number of balls of radius $r$ violated is at most $\frac{K}{c^2 \cdot \frac{H^3\iota}{r^2}} = \frac{1}{c^2 H^3 \iota} \cdot r^2 K$. Thus, after splitting (violation), the number of balls of radius $\frac{r}{2}$ is at most $(c' \cdot 2^n) \cdot \left(\frac{1}{c^2 H^3 \iota} \cdot r^2 K\right) = c'' \cdot r^2 K$ where $c'' = c' \cdot 2^n \cdot \frac{1}{c^2 H^3 \iota}$.

Along the algorithm, when radius $r$ becomes $\frac{r}{2}$, the number of partitioned balls is bounded by the covering number, which is $c \cdot r^{-d}$, and is also bounded by $c'' \cdot r^2 K$ for any $r > 0$ from the above analysis. Then consider $r = 1, \frac{1}{2}, \frac{1}{4}, \cdots, 2^{-i} \cdots$, we have

$$\mathbf{P}_K = \sum_{i=1}^{\infty} \min \left\{ c \left(2^{-i}\right)^{-d}, c''(2^{-i})^2 K \right\} . \tag{3.37}$$

We split this term into two case. **1).** For large radius, use $c \cdot r^{-d}$, and **2).** For very small radius, use the bound $c'' \cdot r^2 K$.

To maximize the upper bound, we find some optimal $J$ where

$$c \left(2^{-J}\right)^{-d} = c \cdot 2^{J \cdot d} = c'' \cdot 2^{-2J} \cdot K . \tag{3.38}$$

Thus,

$$\begin{aligned} 2^{J \cdot (d+2)} &= \frac{c''}{c} K = D \cdot K \\ 2^J &= \Theta(K^{\frac{1}{d+2}}) \qquad (3.39) \\ 2^{J \cdot d} &= \Theta(K^{\frac{d}{d+2}}) , \qquad (3.40) \end{aligned}$$

where $D = \frac{c''}{c}$.

From (3.39) and (3.40), this implies that

$$\begin{aligned} \mathbf{P}_K \leq \sum_{i=1}^{J} c \left(2^{-i}\right)^{-d} + \sum_{i=J+1}^{\infty} c''(2^{-i})^2 K &\leq 2c \cdot 2^{J \cdot d} + 8c'' \cdot 2^{-2(J+1)} K \\ &= \Theta \left( K^{\frac{d}{d+2}} \right) . \qquad (3.41) \end{aligned}$$

This completes the proof.

□

**Theorem 3.11.** *We have the following tight upper bound* $\sum_{k=1}^{K} \beta_t \leq \widetilde{\mathcal{O}} \left( K^{\frac{d+1}{d+2}} \right)$*, where* $d$ *is the covering dimension of* $\mathcal{S} \times \mathcal{A}$*.*

*Proof.* Recall that

$$\beta_t = \sum_{i=1}^{t} \alpha_t^i b_i \leq \sum_{i=1}^{t} \alpha_i b_i = \sum_{i=1}^{t} \frac{1}{i} c \sqrt{\frac{H^3 \iota}{i}} < 4c \sqrt{\frac{H^3 \iota}{t}} .$$

Up to episode $K$, we create $\mathbf{P}_K$ many balls in total. Assume there are $b_i$ past active state-action pairs in $i^{th}$ ball. Clearly $\sum_{i=1}^{\mathbf{P}_K} b_i = K$.

Thus,

$$\begin{aligned}
\sum_{k=1}^{K} \beta_t &\leq \sum_{k=1}^{K} 4c\sqrt{\frac{H^3\iota}{t}} \leq \sum_{i=1}^{\mathbf{P}_K}\sum_{j=1}^{b_i} 4c\sqrt{\frac{H^3\iota}{j}} \\
&\leq \sum_{i=1}^{\mathbf{P}_K} 4c\sqrt{H^3\iota}\cdot 2\sqrt{b_i} = \sum_{i=1}^{\mathbf{P}_K} 8c\sqrt{H^3\iota}\cdot\sqrt{b_i} \\
&= 8c\sqrt{H^3\iota}\cdot\mathbf{P}_K \sum_{i=1}^{\mathbf{P}_K}\left(\frac{1}{\mathbf{P}_K}\cdot\sqrt{b_i}\right) \\
&\leq 8c\sqrt{H^3\iota}\cdot\mathbf{P}_K\sqrt{\sum_{i=1}^{\mathbf{P}_K}\frac{1}{\mathbf{P}_K}\cdot b_i} \qquad (3.42) \\
&= 8c\sqrt{H^3\iota}\cdot\mathbf{P}_K\sqrt{\frac{K}{\mathbf{P}_K}} \qquad (3.43) \\
&= 8c\sqrt{H^3\iota}\cdot\sqrt{K}\cdot\sqrt{\mathbf{P}_K}\,, \qquad (3.44)
\end{aligned}$$

where (3.42) is based on the Jensen's inequality on the the concave function $\sqrt{x}$, and (3.43) is from $\sum_{i=1}^{\mathbf{P}_K} b_i = K$.

From the conclusion of (3.44), we hence have

$$\sum_{k=1}^{K}\beta_t \leq \widetilde{\mathcal{O}}\left(K^{\frac{d+1}{d+2}}\right). \qquad (3.45)$$

□

## 3.7 Conclusion of Main Theorem

### 3.7.1 Proof of Main Theorem

We can combine the above results to complete the proof of the main theorem of this chapter, Theorem 3.3.

Recall that from Lemma 3.7, the cumulative regret up to episode $K$ is given by

$$\sum_{k=1}^{K} V_1^k(x_1^k) - V_1^{\pi_k}(x_1^k) \leq \Theta(K^{\frac{d+1}{d+2}}) + \sum_{j=0}^{H-1}\int_{\texttt{traj}\in\texttt{TRAJ}(h\to h+j)} \mathcal{P}(\texttt{traj})\cdot\sum_{k=1}^{K}\left(Q_{1+j}^k - Q_{1+j}^*\right)d\mathcal{P}\,. \qquad (3.46)$$

In addition, from (3.28),

$$\sum_{k=1}^{K} \left(Q_h^k - Q_h^*\right)(x,a) \leq \Theta\left(K^{\frac{d+1}{d+2}}\right) + \mu \sum_{k=1}^{K} \beta_t \,. \tag{3.47}$$

Combined with Theorem 3.11, we have

$$\sum_{k=1}^{K} (Q_h^k - Q_h^*) \leq \widetilde{\mathcal{O}}\left(K^{\frac{d+1}{d+2}}\right) ,$$

where $d$ is the covering dimension of $\mathcal{S} \times \mathcal{A}$. Thus $\max_{h \leq H} \sum_{k=1}^{K} \left(Q_h^k - Q_h^*\right) \leq \widetilde{\mathcal{O}}\left(K^{\frac{d+1}{d+2}}\right)$. Inequality (3.46) becomes to

$$\begin{aligned}
\sum_{k\,=1}^{K} V_1^k(x_1) - V_1^{\pi_k}(x_1) &\leq \Theta(K^{\frac{d+1}{d+2}}) + \sum_{j=0}^{H-1} \int_{\texttt{traj} \in \texttt{TRAJ}(h \to h+j)} \mathcal{P}(\texttt{traj}) \\
&\qquad \cdot \left( \max_j \sum_{k=1}^{K} \left(Q_{1+j}^k - Q_{1+j}^*\right) \right) d\mathcal{P} \\
&\leq \Theta(K^{\frac{d+1}{d+2}}) + \sum_{j=0}^{H-1} \int_{\texttt{traj} \in \texttt{TRAJ}(h \to h+j)} \mathcal{P}(\texttt{traj}) \cdot \widetilde{\mathcal{O}}\left(K^{\frac{d+1}{d+2}}\right) d\mathcal{P} \\
&\leq \Theta(K^{\frac{d+1}{d+2}}) + \sum_{j=0}^{H-1} \widetilde{\mathcal{O}}\left(K^{\frac{d+1}{d+2}}\right) \int_{\texttt{traj} \in \texttt{TRAJ}(h \to h+j)} \mathcal{P}(\texttt{traj}) d\mathcal{P} \\
&= \widetilde{\mathcal{O}}(K^{\frac{d+1}{d+2}}) \,.
\end{aligned}$$

This completes Theorem 3.3, which is the main scope of this work.

### 3.7.2 Conclusion

In this chapter, we proposed a state-of-the-art $Q$-learning algorithm with UCB exploration policy in the compact state-action metric space. Our algorithm adaptively partitions the state-action space and flexibly selects the future actions to maximize $Q$-values with bandit exploration bonuses. With mild assumptions for concentration, we find a tight upper bound with respect to covering dimension for the performance measure in a general state-action metric space. Our work contains rigorous techniques including novel combinatorial and probabilistic analysis. These approaches are not limited to the analysis of performance measure in $Q$-learning but are also valuable to the general reinforcement learning analysis. This work suggests that the bandit strategy is an effective way to increase the data efficiency for a general model-free reinforcement learning problem.

## 3.8 Supplementary Discussion

Although the two assumptions for concentration in section 3.5.2 are reasonable and mild, some other approaches might also be helpful. This is also a potential direction for future research.

Here we provide some approaches that can be valuable in bounding the $Q$-value function.

### Sublinearity

**Lemma 3.12** (Sublinearity)**.** *If $Q_h^k$ converges to $Q_h^*$ in $k$ for fixed $h$, then $\sum_{k=1}^K |Q_h^k - Q_h^*| < \mathcal{O}(K)$.*

*Proof.* We prove by contradiction. Assume $a_i$ converges to $a$. Then there exists $m > 0$ such that we have the contradiction: $\sum_{i=1}^K |a - a_i| \geq mK$ for any large $K$. This is equivalent to

$$\sum_{i=1}^K |a - a_i| - mK = \sum_{i=1}^K (|a - a_i| - m) \geq 0 \,.$$

Since $a_i$ converges to $a$, there exists $N_1$ such that for any $i > N_1$, $|a - a_i| < \frac{m}{2}$, then $(|a - a_i| - m) < -\frac{m}{2}$. Thus, for $N_2$ large, $\left|\sum_{i=N_1+1}^{N_2} (|a - a_i| - m)\right| > \left|\sum_{i=1}^{N_1} (|a - a_i| - m)\right|$ and $\sum_{i=N_1+1}^{N_2} (|a - a_i| - m) < 0$.

This implies that $\sum_{i=1}^{N_2} (|a - a_i| - m) < 0$. This contradicts our assumption and completes the proof of contradiction. Thus, for $Q_h^k$ converges to $Q_h^*$ in $k$, we have

$$\sum_{k=1}^K |Q_h^k - Q_h^*| < \mathcal{O}(K) \,.$$

□

From Lemma 3.7 and the above Lemma 3.12, if one has $Q_h^k$ converges to $Q_h^*$ in $k$, then one can show performance measure in sublinear order.

### Differential Equation Approach

Here we use Grönwall's inequality. First we recall Grönwall's inequality. Given $I$ is the interval of real line in a format of $[a + \infty)$. If we have

$$u'(t) \leq \beta(t) u(t) \,,$$

then for all $t$, $u$ is bounded by the solution of the corresponding differential equation $v'(t) = \beta(t)v(t)$,

$$u(t) \leq u(a) \exp\left(\int_a^t \beta(s)ds\right) .$$

Assume $\theta_k = Q_h^k - Q_h^*$, which stands for the difference between $Q$ functions. By using Grönwall's inequality and the analysis of differential equations, we have the following lemma.

**Lemma 3.13** (Differential Equation Approach)**.** *For some $\tau$ where $0 < \tau < 1$, if*

$$\frac{d}{dk}\theta_k \leq (\tau - 1) \cdot \frac{1}{k}\theta_k \,, \tag{3.48}$$

*then $\sum_{k=1}^K |Q^k - Q^*| \leq \mathcal{O}(K^\tau)$.*

*Proof.* Notice that $e^{\left(\int_1^k \left((\tau-1)\cdot\frac{1}{s}\right)ds\right)} = k^{(\tau-1)}$. By Gronwall's inequality [Grönwall, 1919], for constant $C = \theta_1$, if

$$\frac{d}{dk}\theta_k \leq (\tau - 1) \cdot \frac{1}{k} \cdot \theta_k \,, \tag{3.49}$$

then

$$\theta_k \leq C \cdot k^{\tau-1} \,. \tag{3.50}$$

Thus, $\sum_{k=1}^K |Q^k - Q^*| = \sum_{k=1}^K \theta_k \leq \mathcal{O}(K^\tau)$. □

From Lemma 3.7 and the above Lemma 3.13, if the difference $\theta_k$ satisfies the given differential equation, then one can also show sublinear performance measure with specific convergence rate.

The above approaches are also helpful to bound the performance measure in $Q$-learning.

# 4 Learning Algorithms in Selective Prediction [4]

Data gathered from real-world applications often suffer from corruption. The low-quality data will hinder the performance of the learning system in terms of the classification accuracy, model building time, and interpretability of the classifier. Selective prediction, also known as prediction with a reject option, is to reduce the error rate by abstaining from prediction under uncertainty while keeping coverage as high as possible. Deep Neural Network (DNN) has a high capacity in fitting large-scale data. If DNNs can leverage the trade-off coverage by selective prediction, then the performance in prediction can potentially be improved. However, current DNN embedded with reject option method requires the knowledge of rejection threshold and the searching of threshold is very inefficient in large-scale applications. Besides, the abstention of prediction on partial datasets increases the model bias and hence might not be optimal. To resolve these problems, we propose sophisticated threshold learning algorithms integrated with selective prediction that can estimate the intrinsic rejection rate of the dataset. Correspondingly, we provide a rigorous framework to generalize the accurate estimation of intrinsic corruption rate. To leverage the advantage of multiple learning algorithms, we extend our learning algorithms to a hierarchical two-stage system. Our methods have the advantages of being flexible with any neural network architecture. The empirical results show that our algorithms can accomplish state-of-the-art performance in challenging real-world datasets in both classification and regression problems.

## 4.1 Introduction

Neural network models have achieved impressive results on large-scale datasets in image classification and other tasks. Although DNNs contain large numbers of parameters that can fit large data very well, some of them exhibit remarkably small generalization error. That is why DNNs proliferate in current machine vision [Zhang and Isola, 2016]. However, the success of the neural network relies on the massive collection of clean data. In real-world applications, large-scale clean data can be very difficult and expensive to obtain since the verification and annotation of data require expert knowledge in many tasks. Even after several years of success in the ImageNet database, there is still no publicly available dataset that contains an order of magnitude more clean data [Veit et al., 2017]. Alternatively, researchers turn to develop the training paradigms on large but noisy datasets since they are more easily obtained [Le et al., 2012, Chen and Gupta, 2015, Pinto et al., 2016].

The corruption on the dataset will hinder the performance of the learning in terms of the

[4] A small portion of this chapter is based on the collaboration with Ilqar Ramazanli. The author is grateful to Dr. Dangxing Chen for his comments and insights.

classification accuracy, model building time and the interpretability of the classifier [Uma, 2018]. There are various types of data corruption in real-world problems, including motion blur and heavy shadow, etc., especially for images. The presence of noise in the data may affect the intrinsic characteristics of a learning problem since corruption can introduce extra properties in the problem domain. Due to DNNs' high capacity in data fitting, the training process of fitting corrupted data will not only significantly increase the training cost, but also reduce model robustness and lead to a poor generalization [Zhang et al., 2017].

For human beings' intelligence, there is a rudimentary kind of self-awareness: the capability of knowing what you don't know. In the language of the machine, there is an equivalent term called selective prediction, also known as prediction with a reject option. This concept has existed for around 60 years [Chow, 1970, Tortorella, 2000, Geifman and El-Yaniv, 2017, El-Yaniv and Wiever, 2010]. The main motivation for selective prediction is to reduce the error rate by abstaining from prediction under uncertainty while keeping coverage as high as possible. Many future AI tasks performed by predictive models can potentially benefit from effective selective prediction. For example, if a self-driving car can figure out the situation in which it does not know how to respond, then it can alert human drivers to take over and hence the risk is controlled [Geifman and El-Yaniv, 2017].

When we work on real data, the classifiers could easily encounter samples very different from those learned in the training phase. In these cases, the cost for a wrong classification could be so high that it should suspend the decision and call for a further test, i.e. to reject the sample. Due to the uncertainty in image classification and pattern recognition, selective prediction has gained undoubtedly a consistent interest in practice [Xie et al., 2006, Santos-Pereira and Pires, 2005]. The discussion about the reject option has also been quite extensive for various hypothesis classes and learning algorithms, such as support vector machine, boosting, and nearest neighbors [Fumera and Roli, 2002, Hellman, 1970, Cortes et al., 2016]. These methods surpass traditional confidence-based methods in practice and achieve the balance of error-reject trade-off.

A classical statistical adage says that models capable of fitting too much will generalize poorly. Due to DNNs' high capacity in fitting data, if DNNs can leverage the trade-off coverage by selective prediction, then the performance in prediction can be improved. However, the reject option has rarely been discussed in the context of neural networks. To the best of our knowledge, Geifman et al. are the first people to study reject option based neural networks which could leverage the neural network's advantage in data fitting and error-reject trade-off [Geifman and El-Yaniv, 2019]. Their method requires the knowledge of the reject rate. Apparently, the reject rate is unknown in many applications and trial-and-error with different reject rates on the whole dataset is very costly for large-scale applications. This

makes their method less sophisticated and ingenious. Besides, model's abstention of prediction on a partial dataset increases the model bias and the dataset has not been fully utilized. In this chapter, we propose noise-robust learning algorithms integrated with selective prediction that can effectively estimate the intrinsic rejection rate of the dataset. We then generalize our method to decision thresholds learning for binary classification with asymmetrical costs. Correspondingly, we provide a rigorous framework to generalize the estimation of intrinsic corruption rate of the dataset. To leverage the advantage of multiple learning algorithms, we extend to a hierarchical two-stage system with an integration of threshold learning. The two-stage system can improve the test accuracy by comprehensive learning simultaneously on both the classified acceptance and rejection subsets. Empirical results show that our methods can achieve state-of-the-art performance in many classification and regression problems.

## 4.2 Preliminaries

First, we review the concept of reject option by the classical Chow's rule and Tortorella's rule. Chow's rule provides a foundation of error-reject trade-off in selective prediction. In addition, Tortorella's rule is an alternative selective prediction method for binary classification with asymmetrical costs. Our threshold learning algorithms proposed in this chapter inherit the fundamental ideas of Chow's rule and Tortorella's rule. Then we will overview Selective Net, a DNN embedded with the reject option that allows end-to-end optimization of selective models. Our threshold learning algorithms extend the Selective Net to a more general framework with effective learning on the intrinsic rejection rate of the dataset.

### 4.2.1 Chow's Rule – Optimal Rejection Rule

Chow proposed an optimal rule for pattern recognition problems based on the error-reject trade-off. Chow's rule consists in rejecting a pattern if the maximum of the a posteriori probability is less than some threshold [Chow, 1970]. Mathematically, given $n$ the total number of classes, $(\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_n)$ the a priori probability distribution of these classes, $\mathcal{F}(x|i)$ the conditional probability density function for sample (pattern) $x$ given $i^{th}$ class, and $\tau$ the rejection threshold, then the optimal rejection rule is: to **accept pattern** $x$ and predict it as in the $k^{th}$ class, we need both of the following hold:

- $\mathcal{P}_k \cdot \mathcal{F}(x|k) \geq \mathcal{P}_i \cdot \mathcal{F}(x|i)$ for all $i = 1, \cdots, n$ ,
- $\mathcal{P}_k \cdot \mathcal{F}(x|k) > (1 - \tau) \sum_{i=1}^{n} \mathcal{P}_i \cdot \mathcal{F}(x|i)$ .

On contrast, to **reject pattern** $x$,

$$\max_{1\leq i\leq n}\{\mathcal{P}_i\cdot\mathcal{F}(x|i)\}\leq(1-\tau)\sum_{i=1}^{n}\mathcal{P}_i\cdot\mathcal{F}(x|i)\,.$$

Equivalently, if we define $m(x):=\frac{\max_{1\leq i\leq n}\{\mathcal{P}_i\cdot\mathcal{F}(x|i)\}}{\mathcal{F}(x)}$ as the maximum of the a posteriori of the classes with $x$, where $\mathcal{F}(x)=\sum_{i=1}^{n}\mathcal{P}_i\cdot\mathcal{F}(x|i)$, then the optimal rule can be restated as

$$\texttt{Selection}=\begin{cases}\texttt{Accept} & \text{if } m(x)>1-\tau\,;\\ \texttt{Reject} & \text{if } m(x)\leq 1-\tau\,.\end{cases}$$

From Chow, the above rule is also a minimum-risk rule if the cost function is uniform within each class, i.e., no distinction is made among the errors, the correct recognition, and the rejection [Chow, 1970]. Given $\mathcal{C}_r,\mathcal{C}_e,\mathcal{C}_c$ the cost of rejection, error (accept but wrong), and correct (accept and correct) portions respectively, if we know the a posteriori distribution, then the optimal rejection threshold related to the cost is $\tau=\frac{\mathcal{C}_r-\mathcal{C}_c}{\mathcal{C}_e-\mathcal{C}_c}$. The optimality of Chow's rule relies on the exact knowledge of the a posteriori probabilities for each sample. However, the full knowledge about the distributions of the classes is extremely difficult to obtain in practical applications and thus the Chow's rule is rarely applicable [Tortorella, 2000, Fumera and Roli, 2002, Fumera et al., 2000]. Chow's rule is hence in an ideal scenario. It mainly provides us a great intuition of the error-reject trade-off.

### 4.2.2 Tortorella's Rule – Reject Rule in Binary Classification

In binary classification, each sample can be classified as one of the two disjoint classes, `Positive (P)` and `Negative (N)`. The costs are not symmetrical in general, because the consequences of different errors are not equivalent and depend on the nature of the particular applications. As an example, in automated cancer diagnosis, a false negative outcome can be much more costly than a false positive since false negative hides patients' potential high risk [Santos-Pereira and Pires, 2005]. Tortorella introduced an idea of selective prediction with asymmetrical cost in binary classification problems [Tortorella, 2000]. To overview it, we define `True Positive Rate (TPR)` as the fraction of actually-positive cases being correctly classified and `False Positive Rate (FPR)` as the fraction of actually-negative cases mistakenly classified as positive. Correspondingly, we can define `True Negative Rate (TNR)` and `False Negative Rate (FNR)`.

### Elementary Decision Rule

Assume $s(x)$ the prediction score for predictor $s$ on sample $x$. Then we have the elementary decision rule based on the decision threshold $\tau$:

$$\texttt{class} = \begin{cases} \texttt{positive} & \text{if } s(x) > \tau\,; \\ \texttt{negative} & \text{if } s(x) \leq \tau\,. \end{cases}$$

Denote by $f_{\texttt{P}}, f_{\texttt{N}}$ the density function of classes P and N. Then

$$\texttt{TNR}(\tau) = \int_0^{\tau} f_{\texttt{N}}(x)dx = 1 - \texttt{FPR}(\tau)\,, \qquad \texttt{FNR}(\tau) = \int_0^{\tau} f_{\texttt{P}}(x)dx = 1 - \texttt{TPR}(\tau)\,.$$

The utility function, $\mathcal{U}$, which measures the effectiveness of binary classifier, is defined based on the costs of TP, FN, FP, TN, $w_{\texttt{TP}}, w_{\texttt{FN}}, w_{\texttt{FP}}$, and $w_{\texttt{TN}}$ [Tortorella, 2000]:

$$\mathcal{U}(\tau) = \mathcal{P}(\texttt{P}) \cdot \Big(w_{\texttt{TP}} \cdot \texttt{TPR}(\tau) + w_{\texttt{FN}} \cdot \texttt{FNR}(\tau)\Big) + \mathcal{P}(\texttt{N}) \cdot \Big(w_{\texttt{TN}} \cdot \texttt{TNR}(\tau) + w_{\texttt{FP}} \cdot \texttt{FPR}(\tau)\Big)\,.$$

### Decision Rule with Reject Option

To accomplish the reject option in binary classifier, the decision rule is changed to:

$$\texttt{class} = \begin{cases} \texttt{positive} & \text{if } s(x) > \tau_2\,; \\ \texttt{negative} & \text{if } s(x) < \tau_1\,; \\ \texttt{reject} & \text{else}\,, \end{cases}$$

where $\tau_1$ and $\tau_2$ are decision thresholds ($\tau_1 \leq \tau_2$). As a consequence, the rates are modified to

$$\texttt{TNR}(\tau_1) = \int_0^{\tau_1} f_{\texttt{N}}(x)dx\,, \qquad \texttt{TPR}(\tau_2) = \int_{\tau_2}^{1} f_{\texttt{P}}(x)dx\,,$$
$$\texttt{FNR}(\tau_1) = \int_0^{\tau_1} f_{\texttt{P}}(x)dx\,, \qquad \texttt{FPR}(\tau_2) = \int_{\tau_2}^{1} f_{\texttt{N}}(x)dx\,,$$

and the reject rates (`Reject Negative (RN)` and `Reject Positive (RP)`) related to negative and positive samples are

$$\texttt{RN}(\tau_1, \tau_2) = \int_{\tau_1}^{\tau_2} f_{\texttt{N}}(x)dx = 1 - \texttt{TNR}(\tau_1) - \texttt{FPR}(\tau_2)\,,$$
$$\texttt{RP}(\tau_1, \tau_2) = \int_{\tau_1}^{\tau_2} f_{\texttt{P}}(x)dx = 1 - \texttt{TPR}(\tau_1) - \texttt{FNR}(\tau_2)\,.$$

The decision utility function with reject option then becomes to

$$\begin{aligned}\mathcal{U}(\tau_1,\tau_2) = \mathcal{P}(\mathtt{P})\Big(w_{\mathtt{TP}}\cdot \mathtt{TPR}(\tau_2) + w_{\mathtt{FN}}\cdot \mathtt{FNR}(\tau_1) + w_{\mathtt{R}}\cdot \mathtt{RP}(\tau_1,\tau_2)\Big)\\ + \mathcal{P}(\mathtt{N})\Big(w_{\mathtt{TN}}\cdot \mathtt{TNR}(\tau_2) + w_{\mathtt{FP}}\cdot \mathtt{FPR}(\tau_2) + w_{\mathtt{R}}\cdot \mathtt{RN}(\tau_1,\tau_2)\Big)\,. \end{aligned} \tag{4.1}$$

Tortorella's rule provides us a great intuition of setting up reject rule in binary classification and also a measurement of the effectiveness of the classifier.

### 4.2.3 Selective Net

Some classifiers, like neural networks and the $k$-nearest neighbor classifier, provide approximations of the a posteriori probabilities [Bishop, 1995, Ruck et al., 1990, Fumera et al., 2000]. In such a case, Chow's rule can be used despite its non-optimality. Selective Net, a DNN embedded with reject option, allows end-to-end optimization of selective models [Geifman and El-Yaniv, 2019]. This is the first work considered as applying Chow's selective prediction rule to the neural network model. Assume we have $\mathcal{D}\times\mathcal{Y}$ where $\mathcal{D}$ has $n$ total samples and $\mathcal{Y}$ is the corresponding labels (in classification) or numerical values (in regression). As shown in Figure 11, Selective Net has a selection net (in green) connecting to a main body neural net. Note that networks here can be assembled by any architecture such as a fully-connected network or CNN. Selective Net learns a pair $(f,g)$, where $f:\mathcal{D}\to\mathcal{Y}$ is a prediction function that maps data to either prediction scores for classes (in classification) or numerical values (in regression) and is learned by main body net, while $g:\mathcal{D}\to\{0,1\}$ is a selection function that serves as a binary qualifier for $f$ and is learned by selection net. Therefore, for sample $x\in\mathcal{D}$,

$$g(x) = \begin{cases} 1 & \text{if } x \text{ is accepted}\,;\\ 0 & \text{if } x \text{ is rejected}\,. \end{cases}$$

For $\{x_i,y_i\}_{i=1}^n \in \mathcal{D}\times\mathcal{Y}$ and $l:\mathcal{Y}\times\mathcal{Y}\to\mathbb{R}^+$ a given loss function, the empirical coverage is

$$\hat{\phi}(g) = \frac{1}{n}\sum_{i=1}^{n} g(x_i)\,, \tag{4.2}$$

and the empirical selective risk is

$$\hat{r}(f,g) = \frac{\frac{1}{n}\sum_{i=1}^{n} l(f(x_i),y_i)\cdot g(x_i)}{\hat{\phi}(g)}\,. \tag{4.3}$$

For a given target coverage (corruption threshold) $\tau$ and hyperparameter $\lambda$ which controls

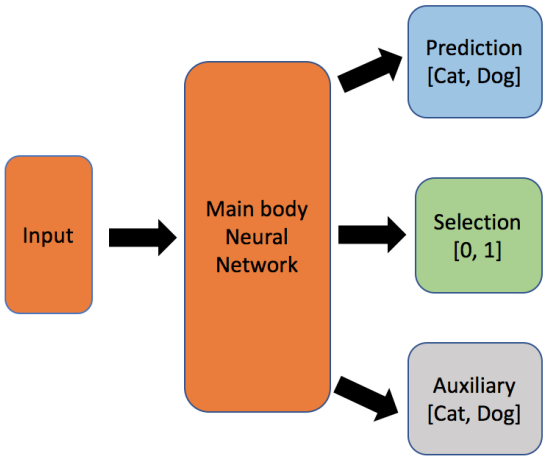



Figure 11: Selective Net

the relative importance of the constraint, we have prediction and auxiliary loss defined as:

$$\mathcal{L}_{\texttt{pred}} = \hat{r}(f,g) + \lambda \cdot \max\left(0, \tau - \hat{\phi}(g)\right)^2 , \tag{4.4}$$
$$\mathcal{L}_{\texttt{aux}} = \frac{1}{n}\sum_{i=1}^{n} l\left(f(x_i), y_i\right) ,$$

which together implies the overall training loss with $\alpha$ highlighting the weights of the prediction loss

$$\mathcal{L} = \alpha \cdot \mathcal{L}_{\texttt{pred}} + (1-\alpha) \cdot \mathcal{L}_{\texttt{aux}} .$$

From [Geifman and El-Yaniv, 2019], $\alpha > 0$ since without the consideration of $\mathcal{L}_{\texttt{aux}}$, the Selective Net will focus on a fraction $\tau$ of the training set before any valuable low-level features are constructed. This will direct Selective Net to overfit the wrong subset of the training set.

## 4.3 Threshold Learning with Selective Prediction

Next, we propose the threshold learning methods with selective prediction. Our method is based on neural network model. As introduced, classifiers like neural networks provide approximations of the a posteriori probabilities [Bishop, 1995, Ruck et al., 1990, Fumera et al., 2000]. In such a case, Chow's rule can be used despite its non-optimality.

### 4.3.1 Advantage of Deep Learning with Selective Prediction

First, we reveal some reasons to study the reject option neural network:

- Avoid over-training on the corrupted data to improve the model's noise robustness and

save the training cost

- Resolve the potential vanishing gradient problem on corrupted samples that are outliers

If a partial dataset is corrupted, the gap between corrupted data and clean data will lead to longer fitting time in the training phase. Moreover, the model is affected by the corrupted data and will decrease the noise robustness. This gives us the first reason (bullet point) above. In addition, corrupted outliers are extreme values before activation. From [Janocha and Czarnecki, 2017], we prove the vanishing gradient problem caused by outliers (second bullet point) in Lemma 4.1. Our method is capable of identifying and rejecting outliers, hence avoiding the problem.

**Lemma 4.1.** *For common sigmoid activation and standard loss function based on $l_1, l_2$ norms, the activation value has gradient asymptotically converging to 0 at both infinities. That is, the vanishing gradient problem exists for corrupted samples that are outliers.*

*Proof.* Denote by $\sigma(x) = (1+e^{-x})^{-1}$ the sigmoid activation. Without loss of generality, assume the actual output is $s$ ($s$ could be numerical value in regression or classes in classification), and the output before activation is $\mathbf{o}$. For simplification, let $(l_1 \circ \sigma)$ stand for the $l_1$ loss joint with $\sigma$. Then

$$\frac{\partial(l_1 \circ \sigma)}{\partial \mathbf{o}}(\mathbf{o}) = \frac{\partial}{\partial \mathbf{o}} \left( \left| s - (1+e^{-\mathbf{o}})^{-1} \right| \right)(\mathbf{o}) = \frac{\partial}{\partial \mathbf{o}} \left( \pm \left( s - (1+e^{-\mathbf{o}})^{-1} \right) \right)(\mathbf{o}) = \mp \frac{e^{-\mathbf{o}}}{(1+e^{-\mathbf{o}})^2} \,.$$

Since

$$\lim_{\mathbf{o} \to -\infty} -\frac{e^{-\mathbf{o}}}{(1+e^{-\mathbf{o}})^2} = \lim_{\mathbf{o} \to \infty} -\frac{e^{-\mathbf{o}}}{(1+e^{-\mathbf{o}})^2} = 0 \,,$$

then the gradient vanishes at both infinities. Similarly, let $(l_2 \circ \sigma)$ stand for the $l_2$ loss joint with $\sigma$. Then

$$\frac{\partial(l_2 \circ \sigma)}{\partial \mathbf{o}}(\mathbf{o}) = 2(s - (1+e^{-\mathbf{o}})^{-1}) \cdot \frac{-e^{-\mathbf{o}}}{(1+e^{-\mathbf{o}})^2} \,, \tag{4.5}$$

where the vanishing gradient again occurs. Thus, for corrupted samples that are outliers, we have the vanishing gradient problem. □

In fact, the vanishing gradient problem caused by outliers is not only for $l_1$ and $l_2$ losses but also appears in classical cross-entropy loss if the output is very large. This can be easily verified by taking the gradient of the $\log(1+e^{-\mathbf{o}})$ term in the cross-entropy loss.

### 4.3.2 Selective Net with Automatic Learning (SNAL)

Selective Net has shown great empirical results with the leverage of reject option [Geifman and El-Yaniv, 2019]. However, the rejection threshold is not known in most of the applications and is by the choice of users. If we use trial-and-error with multiple thresholds on the whole dataset, then it is very costly for large-scale datasets and not sophisticated. This motivates us to design a data-driven selective prediction algorithm that can dynamically learn the rejection threshold of the dataset. In addition, machine learning has a universal rule: the more data in training, the better the interpretation and representation in general. Our algorithm should also prefer high coverage of acceptance while keeping as many high-quality samples in training as possible.

Following the settings in section 4.2.3, we have $\mathcal{D} \times \mathcal{Y}$ with $n$ samples $\{x_i, y_i\}_{i=1}^n$. Note that $l(f(x_i), y_i)$ can be any loss function such as square loss, hinge loss, and log loss and recall that $(f, g)$ is the prediction function and selection function pair in a neural network model with selective prediction. Also from section 4.2.3, the selection net connecting to the main body neural network learns the selection function $g$. In our discussion, we continue to use the empirical selective risk and the penalty on rejection set in (4.4). To design a loss function with the estimation of the intrinsic rejection rate of the dataset, we consider an additional penalty on the low target coverage of the dataset. We motivate this by studying the minimization problem with respect to the coverage $\tau$:

$$\begin{aligned} &\min_{\tau} \ \left\{ \hat{r}\left(f, g\right) + w_{\texttt{pen}} \cdot \max\left(\tau - \hat{\phi}(g), 0\right)^2 \right\}, \\ &\text{s.t. } \tau \leq 1\,, \end{aligned} \tag{4.6}$$

where $\tau \leq 1$ is equivalent to $1 \geq \tau$, and $w_{\texttt{pen}}$ controls the relative importance of the constraint. By the constraint inequality in (4.6), we have the corresponding Lagrangian function:

$$\mathcal{L}_{\text{lagrange}} = \hat{r}(f, g) + w_{\texttt{pen}} \cdot \max\left(\tau - \hat{\phi}(g), 0\right)^2 + w_\tau \cdot (1 - \tau)\,, \tag{4.7}$$

where $w_\tau \geq 0$ and the goal is to minimize $\mathcal{L}_{\text{lagrange}}$ with respect to $\tau$. Following the idea of the minimization problem and Lagrangian on $\tau$, we propose an algorithm that automatically learns the threshold $\tau$, so-called Selective Net Algorithm with Automatic Learning (SNAL). Denote by $\vec{\theta}_f, \vec{\theta}_g$ the set of parameters including weights and bias in the main body neural

network and selection net respectively. The loss function for SNAL is

$$\mathcal{L}_{\texttt{hard}}(\vec{\theta}_f, \vec{\theta}_g, \tau) = \hat{r}\left(f(x;\vec{\theta}_f), g(x;\vec{\theta}_g)\right) + w_{\texttt{pen}} \cdot \max\left(\tau - \hat{\phi}\left(g(x;\vec{\theta}_g)\right), 0\right)^2 + w_\tau \cdot (1-\tau)\,, \tag{4.8}$$

where $w_\tau$ is then the penalty weight on the low coverage. This loss function is always non-negative. Theoretically, if all of the data samples are perfectly clean, then the algorithm can potentially fit the training data perfectly where the coverage threshold $\tau$ can be equal to 1.

Let us say the selection function $g$ classifies $\mathcal{D}$ into $\mathcal{D}_\mathcal{A}$ or $\mathcal{D}_\mathcal{R}$, where $\mathcal{D}_\mathcal{A}$ is the acceptance set and $\mathcal{D}_\mathcal{R}$ is the rejection set. (4.8) is a **hard margin** loss, which means each sample will be either in $\mathcal{D}_\mathcal{A}$ or $\mathcal{D}_\mathcal{R}$. Intuitively, samples in $\mathcal{D}_\mathcal{A}$ contribute to the empirical selective risk $\hat{r}$ and samples in $\mathcal{D}_\mathcal{R}$ contribute to the rejection and coverage penalty in (4.8).

In some applications, especially regression problems, it is hardly possible to recognize if the sample is corrupted. For example, consider male's height in the United States. A sample with a height of 235 cm is corrupted almost surely, while the sample with a height of 205 cm might not be sure of corruption. Applications like randomized control trials from exposure study in biostatistics and index movement prediction in finance should not have a hard margin decision boundary either. Thus, it is better to provide a likelihood/confidence score of samples being corrupted.

We require a confidence rate function for each sample as $c : \mathcal{D} \to [0, 1]$, where $c$ represents the confidence score of a sample to be accepted (in $\mathcal{D}_\mathcal{A}$). To make it work, we require a small modification of the network architecture as in Figure 11. We change the original selection net to the confidence net, where the confidence rate is now the output learned by the confidence net. Denote by $\vec{\theta}_c$ the set of parameters including weights and bias in the confidence net. Then we have a **soft margin** loss function modified from hard margin loss in (4.8):

$$\begin{aligned}\mathcal{L}_{\texttt{soft}}(\vec{\theta}_f, \vec{\theta}_c, \tau) = \frac{1}{n}\sum_{i=1}^{n} c(x_i;\vec{\theta}_c) \cdot l\left(f(x_i;\vec{\theta}_f), y_i\right) \\ + w_{\texttt{pen}} \cdot \max\left(\tau - \frac{1}{n}\sum_{i=1}^{n} c(x_i;\vec{\theta}_c), 0\right)^2 + w_\tau \cdot (1-\tau)\,.\end{aligned} \tag{4.9}$$

The original selection function $g$ with output $\{0, 1\}$ makes $\hat{\phi}(g)$ in $\mathcal{L}_{\texttt{hard}}$ discontinuous. Compared to $\mathcal{L}_{\texttt{hard}}$, confidence rate $c$ can make the soft margin loss function $\mathcal{L}_{\texttt{soft}}$ continuous and in fact differentiable. We prove it in the following Lemma 4.2. This is favorable in neural network backpropagation.

**Lemma 4.2.** *If the loss $l\left(f(x), y\right)$ is continuous for all the samples and the confidence rate*

*function c in learning is differentiable, then $\mathcal{L}_{soft}$ defined in (4.9) is differentiable.*

*Proof.* For the gradient with respect to $\vec{\theta}_f$, it is trivial. Now consider the gradients with respect to confidence rate $\vec{\theta}_c$ and $\tau$,

$$\begin{aligned}\nabla_{\vec{\theta}_c}\mathcal{L}_{\texttt{soft}} =& \frac{1}{n}\sum_{i=1}^{n} l\left(f(x_i;\vec{\theta}_f), y_i\right)\cdot\nabla_{\vec{\theta}_c}c(x_i;\vec{\theta}_c) \\ &+ w_{\texttt{pen}}\cdot\frac{-2}{n}\sum_{i=1}^{n}\nabla_{\vec{\theta}_c}c(x_i;\vec{\theta}_c)\cdot\max\left(\tau-\frac{1}{n}\sum_{i=1}^{n}c(x_i;\vec{\theta}_c),0\right),\end{aligned}$$

$$\frac{\partial\mathcal{L}_{\texttt{soft}}}{\partial\tau} = 2w_{\texttt{pen}}\cdot\max\left(\tau-\frac{1}{n}\sum_{i=1}^{n}c(x_i;\vec{\theta}_c),0\right)-w_\tau\,.$$

From the given assumption, $\nabla_{\vec{\theta}_c}c(x_i;\vec{\theta}_c)$ exists and is continuous. Thus, the soft margin loss is differentiable. □

It is worth noting that $\vec{\theta}_f$, $\vec{\theta}_g$, and $\vec{\theta}_c$ above are the parameters in neural network. Therefore, these parameters get updated in the training phase of the neural network.

### 4.3.3 Threshold Learning with Reject Option (TLRO)

As introduced in the Preliminaries, Tortorella's rule is a selective prediction method unique for a binary classification problem with asymmetrical costs. Similar to threshold $\tau$ in Selective Net, the decision thresholds of negative and positive class, $\tau_1$ and $\tau_2$ as in Preliminaries, are also unknown in applications. Here we require $\tau_1 \leq 0.5$ and $\tau_2 \geq 0.5$. We generalize our work to a learning algorithm that can dynamically search for the decision thresholds. We name the algorithm Threshold Learning with Reject Option (TLRO). TLRO also uses a neural network with selective prediction to abstain from the partial prediction, while dynamically learning the decision thresholds of the dataset. Due to the preference for high data coverage, our algorithm prefers $\tau_1$ and $\tau_2$ as close to 0.5. Note that TLRO has the asymmetrical costs for negative and positive samples with $w_{\texttt{neg}}$ and $w_{\texttt{pos}}$. With the rejection penalty $w_{\texttt{R}}$, here we

define the loss function of TLRO:

$$\begin{aligned}\mathcal{L}_{\texttt{TLRO}}(\vec{\theta}_f, \tau_1, \tau_2) = w_{\texttt{neg}} \cdot \frac{1}{n} \left( \sum_{i=1}^{n} l\left(f(x_i; \vec{\theta}_f), y_i\right) \mathbb{1}_{\{x_i | f(x_i; \vec{\theta}_f) < \tau_1\}} \right) \\ + w_{\texttt{pos}} \cdot \frac{1}{n} \left( \sum_{i=1}^{n} l\left(f(x_i; \vec{\theta}_f), y_i\right) \mathbb{1}_{\{x_i | f(x_i; \vec{\theta}_f) > \tau_2\}} \right) \\ + w_{\texttt{R}} \cdot \frac{1}{n} \left( \sum_{i=1}^{n} \mathbb{1}_{\{x_i | \tau_1 \le f(x_i; \vec{\theta}_f) \le \tau_2\}} \right) + w_\tau \cdot \left( (0.5 - \tau_1)^2 + (\tau_2 - 0.5)^2 \right) . \quad (4.10)\end{aligned}$$

The loss function contains the empirical selective risk, penalty on the rejection set under coverage, and the penalty on the decision thresholds. This is analogous to the SNAL loss function. It is worth highlighting that (4.10) is perfectly inherited from the decision utility function with reject option (4.1) from Tortorella's rule.

(4.10) is the **hard margin** loss where the indicator functions might not be continuous. To make the loss function continuous and hence avoid the potential issues in backpropagation, we design a **soft margin** loss as in SNAL. We consider the sigmoid function as an approximation of the indicator function. Assume $\sigma_\beta^\alpha(x) := \frac{e^{\alpha(x-\beta)}}{1+e^{\alpha(x-\beta)}}$, the sigmoid function with order $\alpha$ and center $\beta$. Then we can use $1 - \sigma_{\tau_1}^\alpha(x)$ to approximate indicator function $\mathbb{1}_{\{x_i | f(x_i) < \tau_1\}}$ and $\sigma_{\tau_2}^\alpha(x)$ to approximate indicator function $\mathbb{1}_{\{x_i | f(x_i) > \tau_2\}}$ given a relatively large $\alpha$. From the above approximations, $\left(1 - \sigma_{\tau_1}^\alpha(x) + \sigma_{\tau_2}^\alpha(x)\right)$ can approximate $\mathbb{1}_{\{x_i | f(x_i) < \tau_1\} \cup \{x_i | f(x_i) > \tau_2\}}$, and $\left(\sigma_{\tau_1}^\alpha(x) - \sigma_{\tau_2}^\alpha(x)\right)$ can approximate $\mathbb{1}_{\{x_i | \tau_1 \le f(x_i) \le \tau_2\}}$. By the replacement of functions, the soft margin loss of TLRO can be differentiable.

## 4.4 Concentration of Measure Results

In the last section, we design learning algorithms integrated with selective prediction that can effectively estimate the rejection rate of the dataset. Here we provide some theoretical justification of the data corruption rate. We require the corruption rate in the training set to serve as a natural and rigorous estimation of the intrinsic value for the population. This is called generalization, a key topic in machine learning. Generalization measures how accurately an algorithm can predict output values for unseen data. As an intrinsic value of an unknown dataset, the corruption rate provides us the important information about the quality and interpretability of data and should be estimated accurately from out of sample. In this section, we study the generalization of data corruption rate. We provide a rigorous framework that utilizes the concentration of measure to estimate the corruption rate in testing.

Assume the training and test data come from a universal underlying system sharing the same corruption rate. Let each sample follow from a Bernoulli distribution with corruption

probability $\gamma$. That is, the universal (population) corruption rate is $\gamma$. For a sample $s$, if $\mathcal{I}$ is an indicator function and $\mathcal{I}_s$ stands for whether sample $s$ is corrupted or not, then

$$\mathcal{I}_s = \begin{cases} 1 & \text{with probability } \gamma\,; \\ 0 & \text{with probability } 1-\gamma\,. \end{cases} \tag{4.11}$$

Let us say $\mathcal{X}$ and $\mathcal{Z}$ are the training and test sets. Denote by $n_{\mathcal{X}}$ and $n_{\mathcal{Z}}$ the size of $\mathcal{X}$ and $\mathcal{Z}$. Then for $\{\mathbf{x}_i\}_{i=1}^{n_{\mathcal{X}}} \in \mathcal{X}$, we can write the training set corruption rate $\gamma_{\mathcal{X}}$ as

$$\gamma_{\mathcal{X}} = \frac{1}{n_{\mathcal{X}}} \sum_{i=1}^{n_{\mathcal{X}}} \mathcal{I}_{\mathbf{x}_i}\,, \tag{4.12}$$

where $\mathcal{I}_{\mathbf{x}_i}$ is the corruption indicator function. Similarly, for $\{\mathbf{z}_i\}_{i=1}^{n_{\mathcal{Z}}} \in \mathcal{Z}$, the test set corruption rate $\gamma_{\mathcal{Z}}$ is

$$\gamma_{\mathcal{Z}} = \frac{1}{n_{\mathcal{Z}}} \sum_{i=1}^{n_{\mathcal{Z}}} \mathcal{I}_{\mathbf{z}_i}\,, \tag{4.13}$$

where $\mathcal{I}_{\mathbf{z}_i}$ is the corruption indicator function. The following lemma indicates that the training corruption rate is close to the population corruption rate with high probability.

**Lemma 4.3** (Concentration result between the training set and universal)**.** *Assume both $\mathcal{X}$ and $\mathcal{Z}$ are from some universal distribution where each sample is corrupted with probability $\gamma$. The following inequality is satisfied for any positive $\epsilon_1$:*

$$\mathcal{P}(\|\gamma_{\mathcal{X}} - \gamma\| \geq \epsilon_1) \leq 2e^{-2n_{\mathcal{X}}\epsilon_1^2}\,. \tag{4.14}$$

*Proof.* From the Hoeffding's inequality, for any $\epsilon_1 > 0$,

$$\mathcal{P}\left(\left|\frac{1}{n_{\mathcal{X}}}\sum_{i=1}^{n_{\mathcal{X}}} \mathcal{I}_{\mathbf{x}_i} - \mathbb{E}\left[\frac{1}{n_{\mathcal{X}}}\sum_{i=1}^{n_{\mathcal{X}}} \mathcal{I}_{\mathbf{x}_i}\right]\right| \geq \epsilon_1\right) \leq 2e^{-2n_{\mathcal{X}}\epsilon_1^2}\,, \tag{4.15}$$

where $\mathbb{E}\left[\frac{1}{n_{\mathcal{X}}}\sum_{i=1}^{n_{\mathcal{X}}} \mathcal{I}_{\mathbf{x}_i}\right]$ is equal to the universal corruption rate $\gamma$. Recall that the definition of $\gamma_{\mathcal{X}}$ is in (4.12). Thus, (4.15) is equivalent to

$$\mathcal{P}(\|\gamma_{\mathcal{X}} - \gamma\| \geq \epsilon_1) \leq 2e^{-2n_{\mathcal{X}}\epsilon_1^2}\,. \tag{4.16}$$

This gives the concentration result between the training $\gamma_{\mathcal{X}}$ and universal $\gamma$.

□

Since we are interested in the generalization of the corruption rate, we need a concentration of measure results between the training and test set.

**Lemma 4.4** (Concentration result between the training and test set)**.** *Assume both $\mathcal{X}$ and $\mathcal{Z}$ are from some universal distribution where each sample is corrupted with probability $\gamma$. Then for any positive pair $(\epsilon_1, \epsilon_2)$, we have the following inequality:*

$$\mathcal{P}(\|\gamma_{\mathcal{X}} - \gamma_{\mathcal{Z}}\| \geq \epsilon_1 + \epsilon_2) \leq 2e^{-2n_{\mathcal{X}}\epsilon_1^2} + 2e^{-2n_{\mathcal{Z}}\epsilon_2^2}\,. \tag{4.17}$$

*Proof.* Same as the proof of Lemma 4.3, from the Hoeffding's inequality on $\mathcal{Z}$ and the definition of $\gamma_{\mathcal{Z}}$ in (4.13), we have for any $\epsilon_2 \geq 0$,

$$\mathcal{P}(\|\gamma - \gamma_{\mathcal{Z}}\| \geq \epsilon_2) \leq 2e^{-2n_{\mathcal{Z}}\epsilon_2^2}\,. \tag{4.18}$$

From the union bounds on (4.16) and (4.18), then with probability more than $1 - 2e^{-2n_{\mathcal{X}}\epsilon_1^2} - 2e^{-2n_{\mathcal{Z}}\epsilon_2^2}$, both of the following inequalities are satisfied at the same time:

$$\|\gamma_{\mathcal{X}} - \gamma\| < \epsilon_1\,,$$
$$\|\gamma - \gamma_{\mathcal{Z}}\| < \epsilon_2\,.$$

Using the triangle inequality, we then have

$$\|\gamma_{\mathcal{X}} - \gamma_{\mathcal{Z}}\| \leq \|\gamma_{\mathcal{X}} - \gamma\| + \|\gamma - \gamma_{\mathcal{Z}}\| < \epsilon_1 + \epsilon_2$$

with probability more than $1 - 2e^{-2n_{\mathcal{X}}\epsilon_1^2} - 2e^{-2n_{\mathcal{Z}}\epsilon_2^2}$ as desired.

□

From the above concentration of measure results, we know the corruption rate on the training set could be a good estimator of the corruption rate of the population and test set.

## 4.5 Hierarchical Two-Stage Learning Algorithm

Our learning methods can dynamically search for the optimal threshold rate and classify the data into acceptance and rejection subsets. If we completely abstain from the prediction on the rejection set, it might not be optimal since we block partial datasets and have not fully utilized the whole dataset. In fact, we may still discover some useful patterns from the rejected

data by using additional learning algorithms. The improvement to that is the two-stage learning algorithm. The two-stage learning algorithm is a hierarchical system that classifies the whole dataset into two subsets at stage I and applies the further learning methods on either rejected subset or both of the two subsets at stage II. Since our algorithms have the capability of classifying the data into acceptance and rejection sets, it is natural to extend them into the two-stage system.

### 4.5.1 Advantage of Two-Stage System

Some good reasons motivate us to study the hierarchical two-stage system. The first one is the trade-off between accuracy and resource allocation, e.g., response time or size of system [Giusti et al., 2002]. Let us consider a scheme where a fast first classifier with rejection is used to classify patterns with high confidence. Rejected patterns are then forwarded to a more complex and slower second-level classifier for further classification. Typically, the hierarchical system as a whole returns better classification results. This hierarchical system turns out to be more flexible if constraints on the response time are imposed by the operating environment. It is hence possible to reach the best trade-off between error and response time by tuning the rejection criterion in the first classifier.

Besides, the two-stage system with forwarding rejected patterns to a second-level classifier can reach the optimal irreducible Bayes error. Recall from chapter 1, each dataset has intrinsic irreducible errors that cannot be eliminated by any learning algorithm. In other words, creating an algorithm that has the potential to achieve irreducible Bayes error is the best we can do. From the next lemma, even if neither of the two classifiers in this two-stage system is optimal, this two-stage system as a whole may still reach the optimal Bayes error.

**Lemma 4.5** (Bayes Error)**.** *Assume Stage-I classifier is $f$ and the Stage-II classifier on rejected patterns from the Stage-I classifier is $g$. Assume $\mathcal{D}_{\mathcal{A}}$ and $\mathcal{D}_{\mathcal{R}}$ are the acceptance and rejection sets classified by $f$ respectively. Given $e_f$ and $e_g$ the error rates for classifier $f$ and $g$. If $f$ only misclassifies data in $\mathcal{D}_{\mathcal{R}}$ and $g$ only misclassifies data in $\mathcal{D}_{\mathcal{A}}$, then the two-stage system error, $e_{\mathtt{TS}} := e_f + e_g$, is equal to the optimal Bayes error $e_{\mathtt{bayes}}$.*

*Proof.* Assume there are $K$ classes in the classification problem. For class $w$ and sample $x$, given $\mathcal{P}(w|x)$ the posterior probability, and define $\mathcal{P}_1(x) := \max_{j=1\cdots K} \mathcal{P}(w_j|x)$, then the optimal Bayes error is defined as

$$e_{\mathtt{bayes}} := \int_{\mathcal{D}} (1 - \mathcal{P}_1(x))\mathcal{P}(x)dx\,. \tag{4.19}$$

Denote by $\mathcal{D}_{\mathcal{A}}$ and $\mathcal{D}_{\mathcal{R}}$ the acceptance and rejection sets classified by $f$ where $\mathcal{D}_{\mathcal{A}} \cup \mathcal{D}_{\mathcal{R}} = \mathcal{D}$.

Let $\mathcal{P}_f(\cdot|\cdot)$ and $\mathcal{P}_g(\cdot|\cdot)$ be the posterior probabilities from classifiers $f$ and $g$. For $\mathcal{F}_1(x) := \max_{j=1\cdots K} \mathcal{P}_f(w_j|x)$ and $\mathcal{G}_1(x) := \max_{j=1\cdots K} \mathcal{P}_g(w_j|x)$, the error rates for $f$ and $g$ are

$$e_f := \int_{\mathcal{D}_\mathcal{A}} (1 - \mathcal{F}_1(x))\mathcal{P}(x)dx\,,$$
$$e_g := \int_{\mathcal{D}_\mathcal{R}} (1 - \mathcal{G}_1(x))\mathcal{P}(x)dx\,.$$

Observe that

$$e_f = \int_{\mathcal{D}_\mathcal{A}} (1 - \mathcal{P}_1(x))\mathcal{P}(x)dx + \int_{\mathcal{D}_\mathcal{A}} (\mathcal{P}_1(x) - \mathcal{F}_1(x))\mathcal{P}(x)dx\,. \tag{4.20}$$

Assume the discrepancy value $\Delta_f(x) = \mathcal{P}_1(x) - \mathcal{F}_1(x)$ and $\mathcal{S}_f = \{x|\Delta_f(x) \neq 0\}$. Then

$$e_f = \int_{\mathcal{D}_\mathcal{A}} (1 - \mathcal{P}_1(x))\mathcal{P}(x)dx + \int_{\mathcal{D}_\mathcal{A}\cap\mathcal{S}_f} \Delta_f(x)\mathcal{P}(x)dx\,. \tag{4.21}$$

Similarly, given $\Delta_g(x) = \mathcal{P}_1(x) - \mathcal{G}_1(x)$ and $\mathcal{S}_g = \{x|\Delta_g(x) \neq 0\}$, we have

$$e_g = \int_{\mathcal{D}_\mathcal{R}} (1 - \mathcal{P}_1(x))\mathcal{P}(x)dx + \int_{\mathcal{D}_\mathcal{R}\cap\mathcal{S}_g} \Delta_g(x)\mathcal{P}(x)dx\,. \tag{4.22}$$

Combining $e_f$ in (4.21) and $e_g$ in (4.22), we have

$$\begin{aligned} e_f + e_g &= \int_{\mathcal{D}} (1 - \mathcal{P}_1(x))\mathcal{P}(x)dx + \int_{\mathcal{D}_\mathcal{A}\cap\mathcal{S}_f} \Delta_f(x)\mathcal{P}(x)dx + \int_{\mathcal{D}_\mathcal{R}\cap\mathcal{S}_g} \Delta_g(x)\mathcal{P}(x)dx \\ &= e_{\texttt{bayes}} + \int_{\mathcal{D}_\mathcal{A}\cap\mathcal{S}_f} \Delta_f(x)\mathcal{P}(x)dx + \int_{\mathcal{D}_\mathcal{R}\cap\mathcal{S}_g} \Delta_g(x)\mathcal{P}(x)dx\,. \end{aligned} \tag{4.23}$$

Therefore, if $\mathcal{D}_\mathcal{A} \cap \mathcal{S}_f = \emptyset$ and $\mathcal{D}_\mathcal{R} \cap \mathcal{S}_g = \emptyset$, then the last two terms in (4.23) are zeros. Given the two-stage system error $e_{\texttt{TS}} := e_f + e_g$, $e_{\texttt{TS}}$ is equal to $e_{\texttt{bayes}}$. Equivalently, if all the patterns accepted by $f$ are classified correctly and the rejected patterns are classified correctly by $g$, then two-stage system error will be Bayes error. □

### 4.5.2 Algorithm

In this section, we study a two-stage learning system that classifies the whole dataset into two subsets using the neural network integrated with selective prediction at stage I and then applies additional learners on both of the two subsets at stage II. A hierarchical two-stage system is shown in Figure 12. In practice, we use the threshold learning algorithm (e.g., SNAL) at stage I that can estimate the rejection threshold of the dataset. After the threshold

is determined, we use this well-trained threshold learning model to classify datasets. Note that the additional learners at stage II are flexible and can be anything from deep CNN to regression tree algorithms. In other words, the two-stage system can be considered as an ensemble method with the leverage of multiple learning algorithms. Different learning algorithms can focus on different subsets and generate good prediction results through the ensemble. This is also called a two-stage algorithm integrated with threshold learning as in algorithm 7.

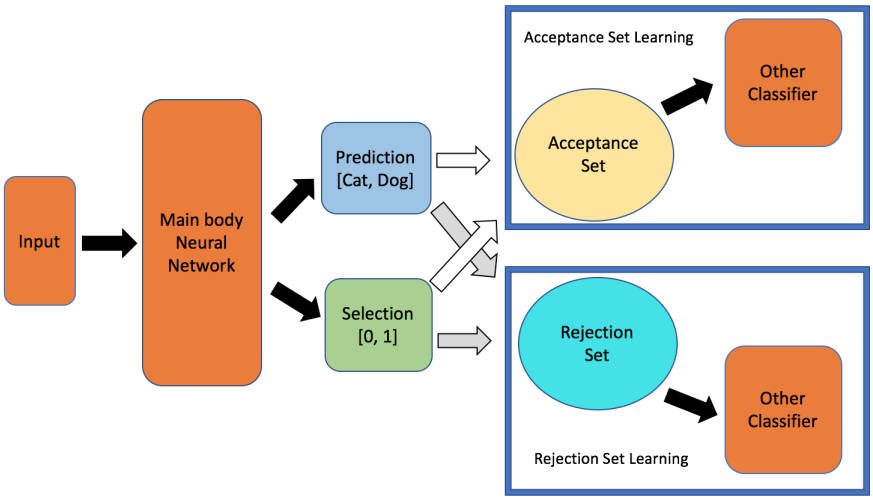



Figure 12: Two-Stage Learning System

Assume the neural network with selective prediction at stage I is with prediction function and selection function pair $(f, g)$ and it classifies the dataset into acceptance set $\mathcal{D}_{\mathcal{A}}$ and rejection set $\mathcal{D}_{\mathcal{R}}$. Also $m$ and $h$ are the predictors for additional learners on $\mathcal{D}_{\mathcal{A}}$ and $\mathcal{D}_{\mathcal{R}}$ respectively. Denote by $\tau$ the threshold discovered from the threshold learning algorithm. Then the two-stage system integrated with SNAL has the following loss functions for the main body neural net with selective prediction, and two additional learners:

$$\mathcal{L}_{\text{main}} = \hat{r}(f, g) + w_{\texttt{pen}} \cdot \max\left(\tau - \hat{\phi}(g), 0\right)^2 + w_\tau \cdot (1 - \tau)\,,$$
$$\mathcal{L}_{\text{learner-1}} = \frac{1}{|\mathcal{D}_{\mathcal{A}}|} \sum_{i=1}^{|\mathcal{D}_{\mathcal{A}}|} l\left(m(x_i), y_i\right)\,, \qquad \mathcal{L}_{\text{learner-2}} = \frac{1}{|\mathcal{D}_{\mathcal{R}}|} \sum_{i=1}^{|\mathcal{D}_{\mathcal{R}}|} l\left(h(x_i), y_i\right)\,. \tag{4.24}$$

### Algorithm

The hierarchical two-stage learning algorithm integrated with threshold learning is below.

**Algorithm 7** Two-Stage Learning System

1: **input:** training data $\mathcal{X}$ and test data $\mathcal{Z}$
2: perform threshold learning algorithm (e.g. SNAL) on $\mathcal{X}$ to find optimal setting
3: with this trained threshold learning model, we classify $\mathcal{X}$ into $\mathcal{X}_{\mathcal{A}}, \mathcal{X}_{\mathcal{R}}$
4: with this trained threshold learning model, we classify $\mathcal{Z}$ into $\mathcal{Z}_{\mathcal{A}}, \mathcal{Z}_{\mathcal{R}}$
5: predict $\mathcal{Z}_{\mathcal{A}}$ by training $\mathcal{X}_{\mathcal{A}}$ on additional learner 1
6: predict $\mathcal{Z}_{\mathcal{R}}$ by training $\mathcal{X}_{\mathcal{R}}$ on additional learner 2
7: **output:** prediction of $\mathcal{Z} = \mathcal{Z}_{\mathcal{A}} \cup \mathcal{Z}_{\mathcal{R}}$

## 4.6 Empirical Study

We test the performance of the threshold learning algorithms including SNAL, TLRO, and Two-stage systems. We first describe the experiment details such as datasets used and the baseline neural network models.

### 4.6.1 Datasets

Our datasets range from classification (binary classification and multi-class classification) to regression problems.

**Cats vs. Dogs**. Cats vs. Dogs dataset is an image binary classification dataset extracted from the ASIRRA dataset. It contains 25,000 images of cats and dogs with 12,500 in each class. We partition the dataset to 9000 cats and 9000 dogs images in training, and 7000 cats and dogs images in testing. Images have the average dimension $360 \times 400 \times 3$ pixels (RGB images) and each pixel has a value from 0 to 255. For the training acceleration purpose, we rescale each image to a smaller dimension $60 \times 60 \times 3$. An example is shown on the left-hand side of Figure 13. This dataset contains some low-quality images with motion blur and heavy shadow objects.

**Synthetic-Noise Cats vs Dogs**. Synthetic-noise Cats vs Dogs dataset is to apply random noise on a random subset of the Cats vs. Dogs datasets. To be more specific, we randomly select 50% of the total images to compose a subset and then apply uniform random noise from -80 to 80 to each image's pixel in this subset. After adding the noise, we bound each pixel from 0 to 255. The partition of the training set and test set here is the same as in the noise-free scenario. An example is represented on the right-hand side of Figure 13.

**CIFAR-10**. CIFAR-10 dataset is an image multi-class classification dataset with 10 object classes comprising a training set of 50,000 images and a test set of 10,000 images [Krizhevsky and Hinton, 2009]. An example is in Figure 2 and details are in Example 1.6. CIFAR-10 is

well-implemented in `Tensorflow` library.

**Concrete Compressive Strength**. Concrete Compressive Strength is a dataset for regression task [Dheeru and Taniskidou, 2017]. It contains 1030 instances with eight numerical features and one target numerical value, the compressive strength of concrete. The goal is to predict the target numerical value using its eight numerical features. We randomly split the dataset into 75% training set and 25% test set. The details about the dataset are in Example 1.8. In terms of data preprocessing, we normalize all the data into [0,1] using `Minmax Scaler` from `TensorFlow` including the target output.

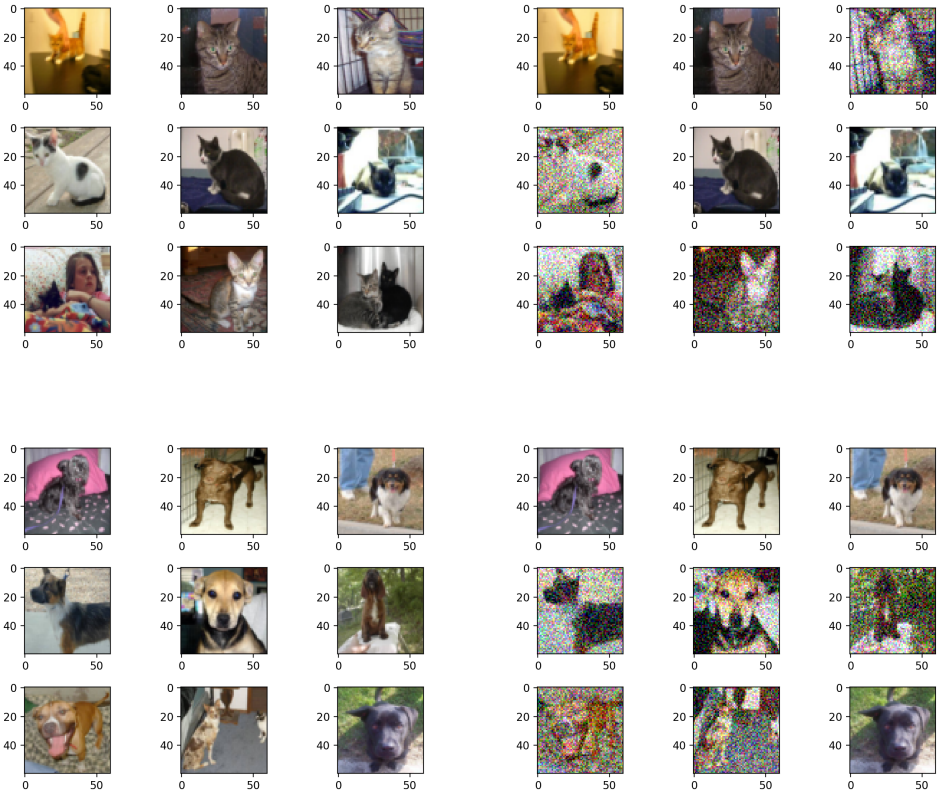


Figure 13: Upper Left: noise-free cats. Upper Right: synthetic-noise cats.
Lower Left: noise-free dogs. Lower Right: synthetic-noise dogs.

### 4.6.2 Baseline Models

Due to the popularity of machine learning in scientific research and industry with production, more people require the explainability and interpretability of models due to trust and informativeness. For example, in medical applications, since a misdiagnosis could potentially

have dramatic consequences, using the outcome from an unintelligible black-box model to perform a diagnostic have very high-risk [Caruana et al., 2015]. In credit scoring, regulators and auditors require lenders to explain why they reject credit applications and prove the decisions does not rely on discriminatory features [Horel et al., 2018]. In self-driving cars, the government and manufacturers need a clear interpretation of how cars perform under different extreme situations to assure drivers' security [Montavon et al., 2018]. Interpretation that provides a clear understanding of how a model generates its output is key for trust and assure the model can generalize well on a new dataset. The interpretability of the neural network model is from the input features' importance and significance [Horel et al., 2018]. The input features' importance in DNN requires the computations of gradients with respect to each input feature. However, deep and complex neural networks have the vanishing gradient problem. The gradient computation is sensitive and will lead to an inaccurate neural network's features' importance. Furthermore, the probability of error for an output neuron increases as the number of layers in the network increases [Stevenson et al., 1990]. Thus, a DNN with lots of layers lacks of interpretability and does not meet the high explainability requirement in people's expectation. This leads us to prefer simple neural networks in applications.

The state-of-the-art neural network models, such as ResNet or VGG Net, have the architecture of tens and hundreds of hidden layers [He et al., 2016, Simonyan and Zisserman, 2015]. In practice, they provide good results mainly due to stacking up a large number of layers. This is a mechanical process and not considered as a sophisticated and ingenious approach. Based on the reasons highlighted in the last paragraph, we consider simple neural networks as our baseline models. It is worth noting that the simple neural network also has the advantage of faster training, which is valuable when we have limited computing resources.

In the empirical study, we compare our threshold learning algorithms and two-stage system with the regular Selective Net, which is considered as the best-so-far method. Through the empirical study, we show that our learning methods outperform Selective Net in all datasets. For the baseline CNN and DNN models, we also add dropout or batch normalization layers if we state. The details of model architecture are in Supplementary. To determine the appropriate optimizers of this experiment, we first consider SGD and Adam since they are the most commonly used. Through our various experiments, we observe that Adam is capable of reaching the optimal much faster than SGD, although the average accuracy of them is almost the same when both of them reach the optimal. For example, Cats vs. Dogs dataset with Adam can reach the optimal accuracy within 55 training epochs, while the same setting with SGD takes more than 250 training epochs to the optimal. Due to the preference for the faster convergence in practice, we always use the Adam optimizer in our experiment. It is worth noting that Adam is slightly more volatile than SGD since standard SGD comes up with a

small learning rate and slow updates.

### 4.6.3 Cats vs. Dogs dataset

In the first experiment, we test our methods on the noise-free binary classification: noise-free Cats vs. Dogs dataset. We check if the threshold learning methods such as SNAL and TLRO can automatically learn the threshold and perform good results without trial-and-error on multiple thresholds in Selective Net.

#### SNAL

Due to the fast convergence of the Adam optimizer, the training can reach optimal within 55 training epochs. Let the accuracy on the $y$-axis stand for the test prediction accuracy. In Figure 14, we compare SNAL with the Selective Net with common thresholds such as 50%, 80%, 90%, and 95%. The accuracy of SNAL from epoch 20 to 55 is 1.7% higher than Selective Net with the best-performing thresholds. The details about the SNAL architecture are in the Supplementary.

It is worth noting that the test accuracy here is not too high because we select a very simple neural network architecture as claimed. Correspondingly, the model bias is high and such accuracy is expected.

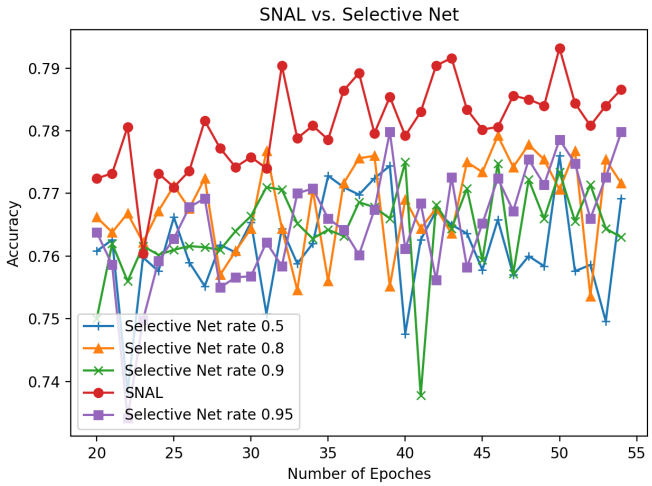



Figure 14: SNAL vs. Selective Net in noise-free Cats vs. Dogs

#### TLRO

Recall that Cats vs Dogs is a binary classification dataset. For binary classification specifically, we propose the TLRO algorithm for decision thresholds learning. We compare the performance

of TLRO with Selective Net. We choose Selective Net with thresholds 80% and 95% since they perform the best. In Figure 15, TLRO surpasses the best-performing Selective Net significantly. The improvement of TLRO in the test accuracy is more than 5.1% against the best-performing Selective Net. The details about the TLRO architecture are in the Supplementary. From this experiment, TLRO is very effective in binary classification problem.

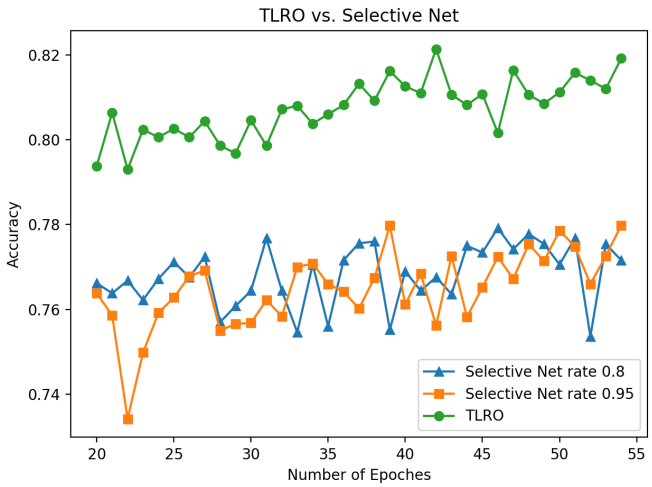



Figure 15: TLRO vs. Selective Net in noise-free Cats vs. Dogs

### 4.6.4 Synthetic-noise Cats vs. Dogs Dataset

Next, we check our methods on the binary classification with artificial noise: synthetic-noise Cats vs. Dogs dataset.

#### SNAL

We compare SNAL with the Selective Net with common thresholds like 65%, 85%, and 95%. In Figure 16, the accuracy of SNAL from epoch 20 to 55 is 1.6% higher than the best-performing Selective Net. In the meantime, the test accuracy on the synthetic-noise dataset here is lower than the accuracy in the noise-free scenario before. This naturally matches up with our expectations and our preference for high-quality datasets.

#### TLRO

Here we test TLRO's performance on this synthetic-noise dataset. In comparison to TLRO, we choose Selective Net with thresholds 85% and 95% since they perform the best in the last experiment. In Figure 17, TLRO surpasses the best-performing Selective Net significantly

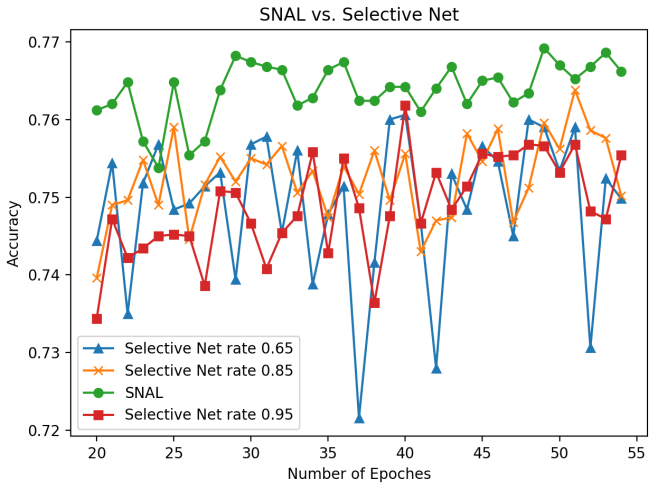



Figure 16: SNAL vs. Selective Net in synthetic-noise Cats vs. Dogs

with more than 4.1% improvement in test accuracy. This experiment shows the success of TLRO in binary classification with a heavy noise dataset scenario.

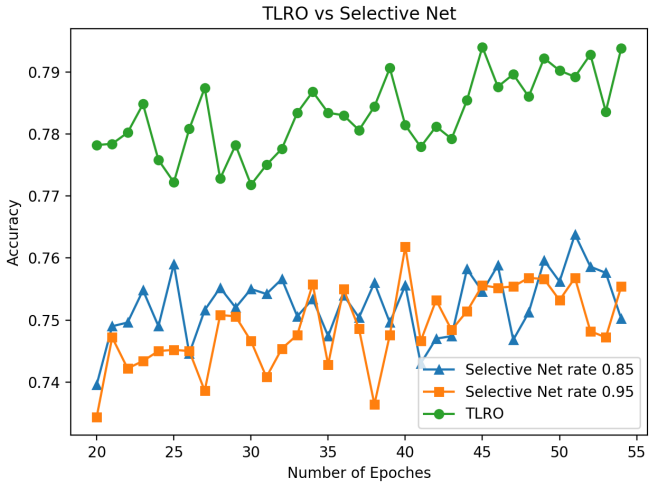



Figure 17: TLRO vs. Selective Net in synthetic-noise Cats vs. Dogs

### 4.6.5 CIFAR-10 Dataset

In this experiment, we consider the multi-class classification: CIFAR-10 dataset. Again, we compare the performance of the SNAL algorithm to Selective Net. The details about the network architecture are in the Supplementary. Using the Adam optimizer, the training can reach optimal within 35 training epochs. In Figure 18, we compare SNAL algorithm with Selective Net with thresholds 60%, 80% and 90%. SNAL again consistently surpasses all

the Selective Nets and the accuracy of SNAL from epoch 5 to 35 is 2.2% higher than the best-performing Selective Net. SNAL is a winner in multi-class classification.

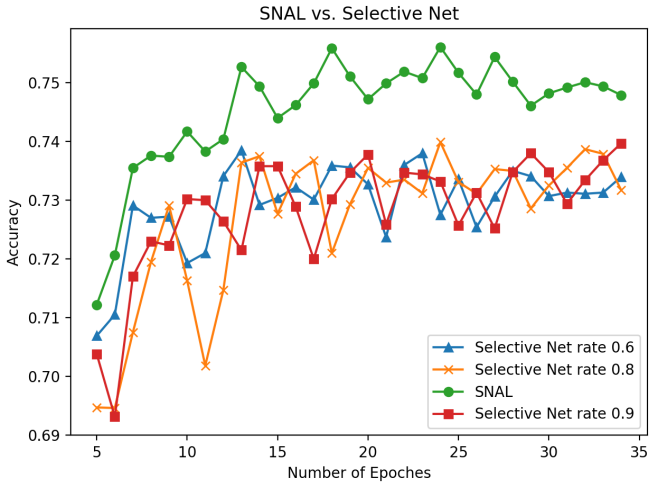



Figure 18: SNAL vs. Selective Net in CIFAR-10

### 4.6.6 Concrete Compressive Strength Dataset

The above experiments are all classification problems. In this section, we consider a regression problem: the Concrete Compressive Strength dataset. We compare SNAL algorithm with Selective Net. Our baseline model is the main body neural networks that contain one, two, or three hidden layer(s). The details about the model architectures are shown in the Supplementary.

The comparison between SNAL and Selective Net with multiple thresholds is in Figure 19a. In the plot, we separate the comparison into one-layer, two-layer, and three-layer networks. Note that the loss on the $y$-axis is the testing *Mean Absolute Error*. As a result, SNAL outperforms Selective Net among all three simple neural network architectures. The improvement might not be obvious in Figure 19a. In Figure 19b, for each network architecture, we show the percentage improvement on the testing *Mean Absolute Error*. Note that the loss improvement is compared to the benchmark with the worst performance in each case. For example, for the one-layer network, the Selective Net with a threshold of 50% gives the worst prediction among all. We consider it as a benchmark in the comparison. Compared to this benchmark, Selective Net with threshold 90% has the test loss 5.05% lower and SNAL is 9.50% lower in test loss respectively. SNAL has the highest improvement among all three network architectures. From this experiment, SNAL is a great solution for the regression problem.

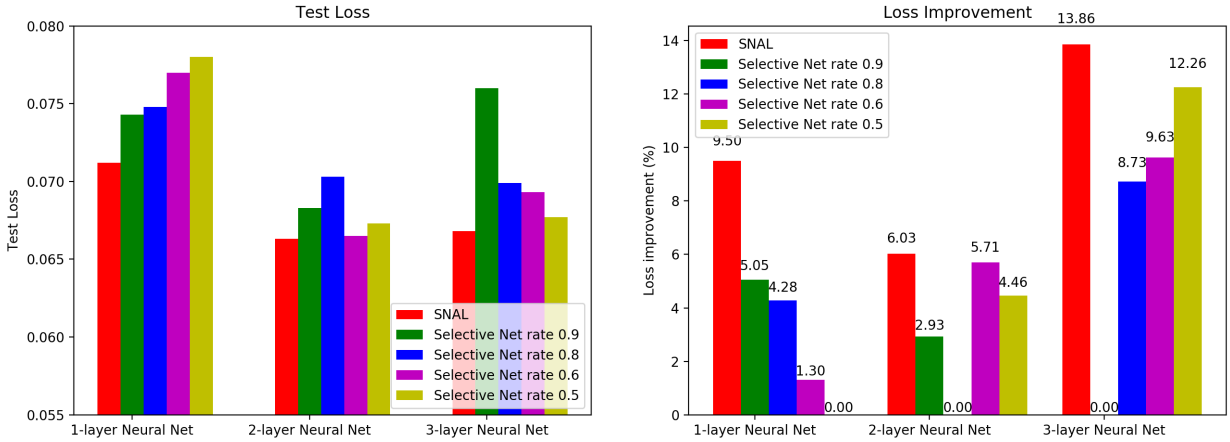



Figure 19: SNAL vs. Selective Net in Concrete Compressive Strength
Left: Test loss comparison Right: Percentage improvement in loss against the benchmark

### 4.6.7 Two-Stage System

The last part of the empirical study will be the two-stage system. In this experiment, the two-stage learning system is integrated with SNAL. We compare it with the best-performing Selective Net and SNAL from the previous experiments. The acceptance and rejection nets are both simple with one convolutional layer followed by one dense layer. The details about the architecture of the two-stage system are given in the Supplementary.

In the Cats vs. Dogs dataset, we choose Selective Net with threshold 80% since it is the best-performing. The two-stage system outperforms the best-performing Selective Net by 4.3% and SNAL by 2.6% in test accuracy. Besides, in the synthetic-noise Cats vs. Dogs dataset, we choose Selective Net with threshold 85% since it is the best-performing. Similar to before, the two-stage system surpasses the best-performing Selective Net by 3.5% and SNAL by 2.0% in test accuracy. The results are shown in Figure 20. Furthermore, in the CIFAR-10 dataset, we choose Selective Net with a threshold of 80%. The two-stage system outperforms the best-performing Selective Net by 3.5% and SNAL by 1.0% in test accuracy. This result is in Figure 21. Two-stage learning system integrated with SNAL performs significantly better than the Selective Net and SNAL from all three experiments. Naturally, the two-stage system is considered as a great solution in practice with little training time sacrifice.

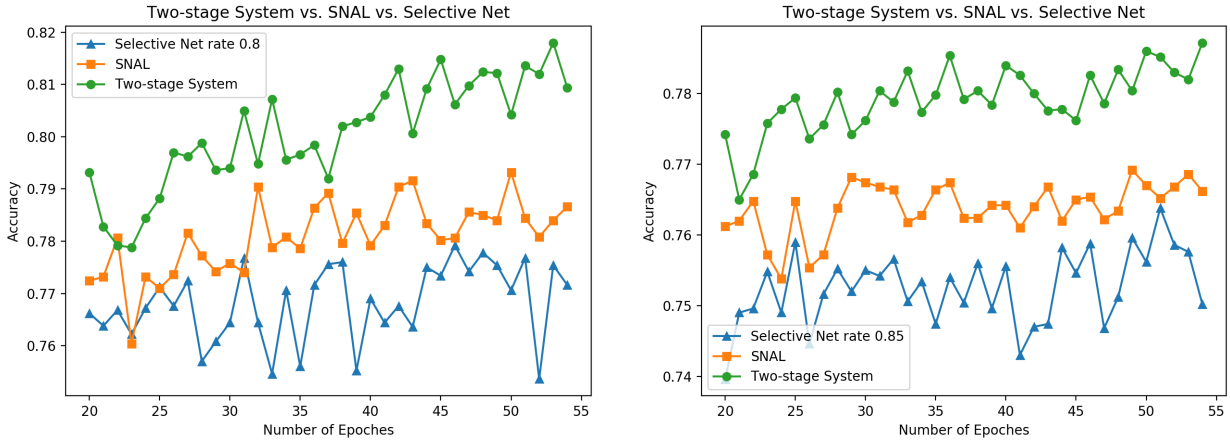



Figure 20: Two-stage System vs. SNAL vs. Selective Net
Left: Noise-free Cats vs. Dogs Right: Synthetic-noise Cats vs. Dogs

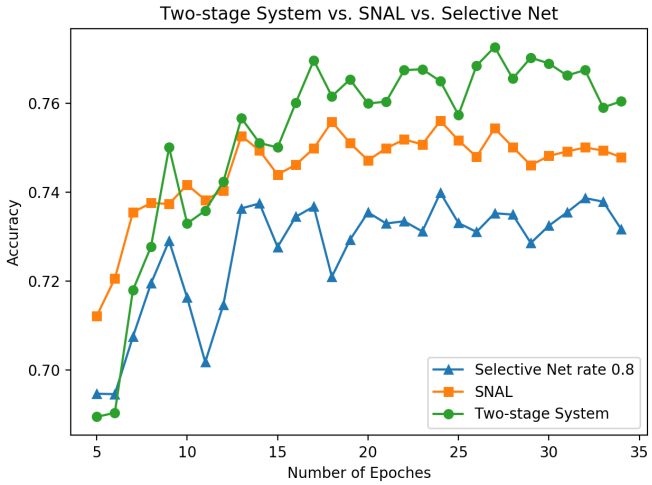



Figure 21: Two-stage System vs. SNAL vs. Selective Net in CIFAR-10

## 4.7 Conclusion

In this chapter, we propose a sophisticated threshold learning algorithm integrated with selective prediction (e.g., SNAL). The generalization to decision thresholds learning algorithm for asymmetrical costs binary classification (e.g., TLRO) is then designed. Furthermore, we extend them to the hierarchical two-stage system that can leverage the advantage of multiple learning algorithms and effectively reduce model bias. Our methods have multiple advantages of being noise-robust, intelligible, and flexible with any neural network architecture. This work provides a general framework for image and pattern recognition with corrupted data and

is supported by rigorous analysis. Our algorithms accomplish state-of-the-art performance in real-world datasets in many classification and regression problems. This work suggests that the selective prediction based learning algorithm is an effective and efficient approach in real-world practice. This area should attract the attention of researchers for further exploration.

We leave several issues for potential future research. In this chapter, we provide a rigorous framework to generalize the accurate estimation of the intrinsic corruption rate. However, the generalization of decision thresholds in TLRO has not been explored. Besides, although the soft margin TLRO loss function has been discussed, sigmoid function as an approximation of the indicator function might also incur the potential vanishing gradient problems. Here, we have not tried to optimize these cases.

## 4.8 Supplementary

### Model Architecture

In our experiments, all of the algorithms are implemented using `Keras` with `TensorFlow` backend. The code is available on request. For any request, please contact the author, `weicheny@andrew.cmu.edu`.

Here we provide the model summary of our experiments. Note that we use CNN baseline model for (both noise-free and synthetic-noise) Cats vs. Dogs and CIFAR-10 dataset, and neural network baseline model for Concrete Compressive Strength dataset. The hyperparameters in these models are with reasonable effort of tuning.

### SNAL architecture in Cats vs. Dogs

We use selection layer to classify the data into acceptance and rejection.

| Layer | Hyperparameters | Values |
|---|---|---|
| Conv1 | conv1-filter | 64 |
| | conv1-kernel-size | (3,3) |
| | conv1-kernel-initializer | Xavier Normal initializer |
| | conv1-activation | “relu” |
| MaxPooling1 | pooling1-size | (2,2) |
| | pooling1-padding | “valid” |
| Dropout1 | dropout-rate | 0.5 |
| Conv2 | conv1-filter | 64 |
| | conv2-kernel-size | (3,3) |
| | conv2-kernel-initializer | Xavier Normal initializer |
| | conv2-activation | “relu” |
| MaxPooling2 | pooling2-size | (2,2) |
| | pooling2-padding | “valid” |
| Dropout2 | dropout-rate | 0.5 |
| Selection | activation | “sigmoid” |
| | number-of-units | 1 |
| Prediction | activation | “sigmoid” |
| | number-of-units | 1 |

### TLRO architecture in Cats vs. Dogs

We use dense layer to learn upper bound (ub) and lower bound (lb) decision threshold.

| Layer | Hyperparameters | Values |
|---|---|---|
| Conv1 | conv1-filter | 64 |
| | conv1-kernel-size | (3,3) |
| | conv1-kernel-initializer | Xavier Normal initializer |
| | conv1-activation | "relu" |
| MaxPooling1 | pooling1-size | (2,2) |
| | pooling1-padding | "valid" |
| Dropout1 | dropout-rate | 0.5 |
| Conv2 | conv1-filter | 64 |
| | conv2-kernel-size | (3,3) |
| | conv2-kernel-initializer | Xavier Normal initializer |
| | conv2-activation | "relu" |
| MaxPooling2 | pooling2-size | (2,2) |
| | pooling2-padding | "valid" |
| Dropout1 | dropout-rate | 0.5 |
| Dense-ub-1 | activation | "relu" |
| | dense1-kernel-initializer | Xavier Normal initializer |
| | number-of-hidden-units | 64 |
| Dense-ub-2 | activation | "sigmoid" |
| | dense1-kernel-initializer | Xavier Normal initializer |
| | number-of-hidden-units | 1 |
| Dense-lb-1 | activation | "relu" |
| | dense1-kernel-initializer | Xavier Normal initializer |
| | number-of-hidden-units | 64 |
| Dense-lb-2 | activation | "sigmoid" |
| | dense1-kernel-initializer | Xavier Normal initializer |
| | number-of-hidden-units | 1 |
| Prediction | activation | "sigmoid" |
| | number-of-units | 1 |

**SNAL architecture in Concrete Compressive Strength**

The presented architecture is for 3 hidden layer case. For one or two hidden layer cases, we only consider Dense 1 and Dense1/Dense 2 layers.

| Layer | Hyperparameters | Values |
|---|---|---|
| Dense1 | activation | "sigmoid" |
| | dense1-kernel-initializer | Xavier Normal initializer |
| | number-of-hidden-units | 128 |
| Dense2 | activation | "sigmoid" |
| | dense2-kernel-initializer | Xavier Normal initializer |
| | number-of-hidden-units | 64 |
| Dense3 | activation | "sigmoid" |
| | dense3-kernel-initializer | Xavier Normal initializer |
| | number-of-hidden-units | 64 |
| Selection | activation | "sigmoid" |
| | number-of-units | 1 |
| Prediction | activation | "sigmoid" |
| | number-of-units | 1 |

**SNAL architecture in CIFAR-10**

| Layer | Hyperparameters | Values |
|---|---|---|
| Conv1 | conv1-filter | 32 |
| | conv1-kernel-size | (3,3) |
| | conv1-kernel-initializer | Xavier Normal initializer |
| | conv1-activation | "relu" |
| MaxPooling1 | pooling1-size | (2,2) |
| | pooling1-padding | "valid" |
| Conv2 | conv1-filter | 32 |
| | conv2-kernel-size | (3,3) |
| | conv2-kernel-initializer | Xavier Normal initializer |
| | conv2-activation | "relu" |
| MaxPooling2 | pooling2-size | (2,2) |
| | pooling2-padding | "valid" |
| Conv3 | conv1-filter | 64 |
| | conv2-kernel-size | (3,3) |
| | conv2-kernel-initializer | Xavier Normal initializer |
| | conv2-activation | "relu" |
| MaxPooling3 | pooling2-size | (2,2) |
| | pooling2-padding | "valid" |
| Dense | activation | "relu" |
| | number-of-hidden-units | 256 |
| | dropout-rate | 0.5 |
| Selection | activation | "sigmoid" |
| | number-of-units | 1 |
| Prediction | activation | "softmax" |
| | number-of-units | 10 |

### Two-stage System in Cats vs. Dogs

#### Main body neural network for classification

| Layer | Hyperparameters | Values |
|---|---|---|
| Conv1 | conv1-filter | 64 |
| | conv1-kernel-size | (3,3) |
| | conv1-kernel-initializer | Xavier Normal initializer |
| | conv1-activation | "relu" |
| MaxPooling1 | pooling1-size | (2,2) |
| | pooling1-padding | "valid" |
| Dropout1 | dropout-rate | 0.5 |
| Conv2 | conv2-filter | 64 |
| | conv2-kernel-size | (3,3) |
| | conv2-kernel-initializer | Xavier Normal initializer |
| | conv2-activation | "relu" |
| MaxPooling2 | pooling2-size | (2,2) |
| | pooling2-padding | "valid" |
| Dropout2 | dropout-rate | 0.5 |
| Selection | activation | "sigmoid" |
| | number-of-units | 1 |

#### Acceptance Net for Acceptance Set

| | | |
|---|---|---|
| Conv Acceptance | conv-acc-filter | 32 |
| | conv-acc-kernel-size | (3,3) |
| | conv-acc-kernel-initializer | Xavier Normal initializer |
| | conv-acc-activation | "relu" |
| MaxPooling Acceptance | pooling-acc-size | (2,2) |
| | pooling-acc-padding | "valid" |
| Dense Acceptance | activation | "relu" |
| | number-of-hidden-units | 64 |
| | dropout-rate | 0.5 |

| | | |
|---|---|---|
| Acc Prediction | activation | "sigmoid" |
| | number-of-units | 1 |

**Rejection Net for Rejection Set**

| | | |
|---|---|---|
| Conv Rejection | conv-rej-filter | 32 |
| | conv-rej-kernel-size | (3,3) |
| | conv-rej-kernel-initializer | Xavier Normal initializer |
| | conv-rej-activation | "relu" |
| MaxPooling Rejection | pooling-rej-size | (2,2) |
| | pooling-rej-padding | "valid" |
| Dense Rejection | activation | "relu" |
| | number-of-hidden-units | 64 |
| | dropout-rate | 0.5 |
| Rej Prediction | activation | "sigmoid" |
| | number-of-units | 1 |

**Two-stage System in CIFAR-10**

**Main body neural network for classification**

| Layer | Hyperparameters | Values |
|---|---|---|
| Conv1 | conv1-filter | 32 |
| | conv1-kernel-size | (3,3) |
| | conv1-kernel-initializer | Xavier Normal initializer |
| | conv1-activation | "relu" |
| MaxPooling1 | pooling1-size | (2,2) |
| | pooling1-padding | "valid" |
| Conv2 | conv2-filter | 32 |
| | conv2-kernel-size | (3,3) |

| | | |
|---|---|---|
| | conv2-kernel-initializer | Xavier Normal initializer |
| | conv2-activation | "relu" |
| MaxPooling2 | pooling2-size | (2,2) |
| | pooling2-padding | "valid" |
| Conv3 | conv3-filter | 64 |
| | conv3-kernel-size | (3,3) |
| | conv3-kernel-initializer | Xavier Normal initializer |
| | conv3-activation | "relu" |
| MaxPooling3 | pooling3-size | (2,2) |
| | pooling3-padding | "valid" |
| Selection | activation | "sigmoid" |
| | number-of-units | 1 |

**Acceptance Net for Acceptance Set**

| | | |
|---|---|---|
| Conv Acceptance | conv-acc-filter | 64 |
| | conv-acc-kernel-size | (3,3) |
| | conv-acc-kernel-regularizer | L2 regularizer |
| | conv-acc-activation | "relu" |
| MaxPooling Acceptance | pooling-acc-size | (2,2) |
| | pooling-acc-padding | "valid" |
| Dense Acceptance | activation | "relu" |
| | number-of-hidden-units | 64 |
| | dropout-rate | 0.5 |
| Acc Prediction | activation | "sigmoid" |
| | number-of-units | 1 |

**Rejection Net for Rejection Set**

| | | |
|---|---|---|
| Conv Rejection | conv-rej-filter | 64 |
| | conv-rej-kernel-size | (3,3) |

| | | |
|---|---|---|
| | conv-rej-kernel-regularizer | L2 regularizer |
| | conv-rej-activation | "relu" |
| MaxPooling Rejection | pooling-rej-size | (2,2) |
| | pooling-rej-padding | "valid" |
| Dense Rejection | activation | "relu" |
| | number-of-hidden-units | 64 |
| | dropout-rate | 0.5 |
| Rej Prediction | activation | "sigmoid" |
| | number-of-units | 1 |

**Type-II Two-Stage System**

In this supplementary section, we propose an additional two-stage algorithm integrated with selective prediction. We name it type-II two-stage system and it is a unified learning system shown in Figure 22. We have not fully explored this system and treat it as a potential future work.

In practice, we use the threshold learning algorithm (e.g., SNAL) as a stage-I classifier that can estimate the rejection threshold of the dataset. Then the acceptance net and rejection net at stage II receive the classified acceptance and rejection subsets from the stage-I classifier. In this case, acceptance and rejection nets are also neural networks that allow the gradient backpropagation. During each training epoch, samples that are in the acceptance set at previous epoches might be switched to rejection set due to weights update in training. In other words, samples in the acceptance and rejection sets change step-by-step. Assume $m$ and $h$ are the predictors for additional learners on $\mathcal{D}_{\mathcal{A}}$ and $\mathcal{D}_{\mathcal{R}}$. Following the same notations as before, we have the cumulative loss function of the type-II system integrated with SNAL:

$$\begin{aligned}\mathcal{L}_{\texttt{Type-II}} = \frac{1}{|\mathcal{D}_{\mathcal{A}}|}\sum_{i=1}^{|\mathcal{D}_{\mathcal{A}}|} l\left(m(x_i), y_i\right) + w_{\mathtt{R}} \cdot \frac{1}{|\mathcal{D}_{\mathcal{R}}|}\sum_{i=1}^{|\mathcal{D}_{\mathcal{R}}|} l\left(h(x_i), y_i\right) \\ + w_{\texttt{pen}} \cdot \max\left(\tau - \hat{\phi}(g), 0\right)^2 + w_\tau \cdot (1-\tau)\,, \end{aligned} \tag{4.25}$$

where $w_{\mathtt{R}}$ is the penalty weight for the rejection set. We show the type-II two-stage system in algorithm 8.

**Algorithm 8** Two-Stage Learning System (Type-II with SNAL)

1: **input:** training data $\mathcal{X}$ and test data $\mathcal{Z}$
2: **while** validation loss is not optimal, e.g., no early stopping yet **do**
3: update weights in additional network learners during gradient backpropagation
4: update weights in stage-I SNAL during gradient backpropagation
5: perform updated SNAL to re-classify $\mathcal{Z}$ into $\mathcal{Z}_\mathcal{A}$ and $\mathcal{Z}_\mathcal{R}$
6: predict $\mathcal{Z}_\mathcal{A}$ and $\mathcal{Z}_\mathcal{R}$ using additional network learners
7: **output:** prediction of $\mathcal{Z} = \mathcal{Z}_\mathcal{A} \cup \mathcal{Z}_\mathcal{R}$

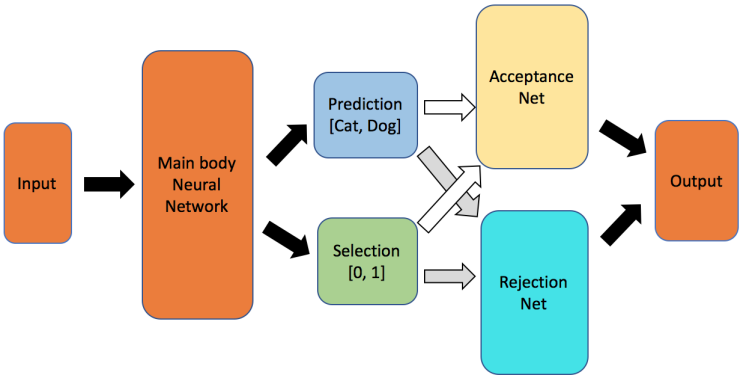



Figure 22: Type-II Two-Stage Learning System

# References


Yasin Abbasi-Yadkori, Dávid Pál, and Csaba Szepesvári. Improved algorithms for linear stochastic bandits. In *Advances in Neural Information Processing Systems*, pages 2312–2320, 2011.

Marc Abeille and Alessandro Lazaric. Linear thompson sampling revisited. *Electronic Journal of Statistics*, 11(2):5165–5197, 2017.

Rajeev Agrawal. Sample mean based index policies by $o(\log n)$ regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27(4):1054–1078, 1995.

Shipira Agrawal and Randy Jia. Optimistic posterior sampling for reinforcement learning: Worst-case regret bounds. In *Neural Information Processing Systems*, 2017.

Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on Learning Theory*, pages 39.1–39.26, 2012.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34, 2017.

Kavosh Asadi, Dipendra Misra, and Michael Litterman. Lipschitz continuity in model-based reinforcement learning. In *International Conference on Machine Learning*, 2018.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, 2002.

Peter Auer and Ronald Ortner. Ucb revisited: improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, 61:55–65, 2010.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

Peter Auer, Ronald Ortner, and Csaba Szepesvári. Improved rates for the stochastic continuum-armed bandit problem. In *International Conference on Computational Learning Theory*, pages 454–468. Springer, 2007.

Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for reinforcement learning. In *International Conference on Machine Learning*, 2017.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning practice and the classical bias-variance trade-off. *Proceedings of the National Academy of Sciences of the United States of America*, 32, 2019.

James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

Christopher M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.

Djallel Bouneffouf and Irina Rish. A survey on practical applications of multi-armed and contextual bandits. *arXiv preprint arXiv:1904.10040*, 2019.

Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and regression trees*. CRC press, 1984.

Sébastien Bubeck and Nicolo Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12(5):1655–1695, 2011.

Rich Caruana, Steve Lawrence, and Lee Giles. Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping. In *Neural Information Processing Systems*, 2000.

Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturn, and Noemie Elhadad. Intelligible models for health-care: Predicting pneumonia risk and hospital 30-day readmission. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

Xinlei Chen and Abhinav Gupta. Webly supervised learning of convolutional networks. In *International Conference on Computer Vision*, 2015.

C. K. Chow. On optimal recognition error and reject tradeoff. *IEEE Transactions on Information Theory*, 16, 1970.

Emile Contal, Vianney Perchet, and Nicolas Vayatis. Gaussian process optimization with mutual information. In *International Conference on Machine Learning*, pages 253–261, 2014.

Corinna Cortes, Giulia DeSalvo, and Mehryar Mohri. Boosting with abstention. In *Advances in Neural Information Processing Systems*, 2016.

Ashok Cutkosky and Robert Busa-Fekete. Distributed stochastic optimization via adaptive sgd. In *Neural Information Processing Systems*, 2018.

Varsha Dani, Thomas P. Hayes, and Sham M. Kakade. Stochastic linear optimization under bandit feedback. In *Conference on Learning Theory*, pages 355–366, 2008.

Nando De Freitas, Alex Smola, and Masrour Zoghi. Exponential regret bounds for gaussian process bandits with deterministic observations. In *International Conference on Machine Learning*, 2012.

Marc Peter Deisenroth and Carl Edward Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, 2011.

D. Dheeru and Karra Taniskidou. Uci machine learning repository. *URL http://archive.ics.uci.edu/ml*, 2017.

Kefan Dong, Yuanhao Wang, Xiaoyu Chen, and Liwei Wang. Q-learning with ucb exploration is sample efficient for infinite-horizon mdp. *arXiv preprint arXiv:1901.09311v1*, 2019.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

Ran El-Yaniv and Yair Wiever. On the foundations of noise-free selective classification. *Journal of Machine Learning Research*, 11, 2010.

Gerald B. Folland. *Real analysis: modern techniques and their applications*, volume 40. Wiley, 1999.

Vincent Francois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*, 2018.

Giorgio Fumera and Fabio Roli. Support vector machines with embedded reject option. *Pattern Recognition with Support Vector Machines*, 2002.

Giorgio Fumera, Fabio Roli, and Giorgio Giacinto. Reject option with multiple thresholds. *Pattern Recognition*, 33, 2000.

Chris Gaskett, David Wettergreen, and Alexander Zelinsky. Q-learning in continuous state and action spaces. *Advanced Topics in Artificial Intelligence*, 1999.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. *arXiv preprint arXiv:1705.08500*, 2017.

Yonatan Geifman and Ran El-Yaniv. Selectivenet: A deep neural network with an integrated reject option. In *International Conference on Machine Learning*, 2019.

John C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B. (Methodological)*, pages 148–177, 1979.

Nicola Giusti, Francesco Masulli, and Alessandro Sperduti. Theoretical and experimental analysis of a two-stage system for classification. *IEEE Transactions On Pattern Analysis and Machine Intelligence*, 24, 2002.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Thomas Hakon Grönwall. Note on the derivatives with respect to a parameter of the solutions of a system of differential equations. *Annals of Mathematics*, 20(4):292–296, 1919.

Elad Hazan, Adam Klivans, and Yang Yuan. Hyperparameter optimization: A spectral approach. *arXiv preprint arXiv:1706.00764*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Martin E. Hellman. The nearest neighbor classification rule with a reject option. *IEEE Transactions on Systems Science and Cybernetics*, 6(3), 1970.

Enguerrand Horel, Virgile Mison, Tao Xiong, Kay Giesecke, and Lidia Mangu. Sensitivity based neural networks explanations. *arXiv preprint arXiv:1812.01029*, 2018.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Alex Irpan. Q-learning in continuous state action spaces. Technical report, University of California, Berkeley, 2015.

Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017.

Chi Jin, Allen-Zeyuan Zhu, Xinyan Yan, Sebastian Bubeck, and Michael Jordan. Is q-learning provably efficient? In *Advances in Neural Information Processing Systems*, 2018.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Neural Information Processing Systems*, 2013.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-armed bandits in metric spaces. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pages 681–690. ACM, 2008.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, pages 282–293. Springer, 2006.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Technical Report*, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, and Andrew Y. Ng. Building high-level features using large scale unsupervised learning. In *International Conference on Machine Learning*, 2012.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1999.

Jason D. Lee, Qihang Lin, Tengyu Ma, and Tianbao Yang. Distributed stochastic variance reduced gradient methods by sampling extra data with replacement. *Journal of Machine Learning Research*, 18, 2017.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pages 661–670. ACM, 2010.

Stefan Magureanu, Richard Combes, and Alexandre Proutiere. Lipschitz bandits: Regret lower bound and optimal algorithms. In *Conference on Learning Theory*, pages 975–999, 2014.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

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 33rd International Conference on Machine Learning*, pages 1928–1937, 2016.

Gregoire Montavon, Wojciech Samek, and Klaus-Robert Muller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73, 2018.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, volume 2011, page 5, 2011.

Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Large-scale distributed second-order optimization using kronecker-factored approximate curvature for deep convolutional neural networks. *arXiv preprint arXiv:1811.12019*, 2019.

Fabien Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta. The curious robot: Learning visual representations via physical interactions. In *European Conference on Computer Vision*, 2016.

Lutz Prechelt. Early stopping-but when? *Neural Networks: Tricks of the Trade*, 2012.

J.R. Quinlan. Induction of decision trees. *Machine Learning 1*, 1986.

Sashank Reddi, Satyen Kale, and Sanjiv Kuma. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Dennis W. Ruck, Steven K. Rogers, Matthew Kabrisky, M. E. Oxley, and Bruce W. Suter. The multi-layer perceptron as an approximation to a bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 4, 1990.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2017.

Carla M. Santos-Pereira and Ana M. Pires. On optimal reject rules and roc curves. *Pattern Recognition Letters*, 26:943–952, 2005.

Stefan Schaal. Learning from demonstration. In *Neural Information Processing Systems*, 1997.

Jeff G. Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. In *Neural Information Processing Systems*, 1997.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

Ohad Shamir. A variant of azuma's inequality for martingales with subgaussian tails. *arXiv preprint arXiv:1110.2392*, 2011.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484 – 489, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Aleksandrs Slivkins. Contextual bandits with similarity information. *Journal of Machine Learning Research*, 15(1):2533–2568, 2014.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Maryhelen Stevenson, Rodney Winter, and Bernard Widrow. Sensitivity of feedforward neural networks to weight errors. *IEEE Transactions on Neural Networks*, 1, 1990.

Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael. L. Litterman. Pac model-free reinforcement learning. In *International Conference on Machine Learning*, pages 881–888, 2006.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, 2013.

Richard S. Sutton and Andrew G. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

Csaba Szepesvari and William D. Smart. Interpolation-based q-learning. In *International Conference on Machine Learning*, 2004.

William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Francesco Tortorella. An optimal reject rule for binary classifiers. *Advances in Pattern Recognition*, pages 611–620, 2000.

John N. Tsitsiklis and Benjamin van Roy. *Feature-based methods for large scale dynamic programming*, volume 22. Machine Learning, 1996.

Kaliappan Uma. Improving the classification accuracy of noisy dataset by effective data preprocessing. *International Journal of Computer Applications*, 180, 2018.

Paul E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2):161–186, 1989.

Emmanuel Vazquez and Julien Bect. Convergence properties of the expected improvement algorithm. *arXiv preprint arXiv:0712.3744*, 2007.

Andreas Veit, Neil Alldrin, Gal Chechik, Ivan Krasin, Abhinav Gupta, and Serge Belongie. Learning from noisy large-scale datasets with minimal supervision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Shengjie Wang, Tianyi Zhou, and Jeff A. Bilmes. Jumpout: Improved dropout for deep neural networks with relus. In *International Conference on Machine Learning*, 2019.

Christopher J.C.H. Watkins. *Learning from delayed rewards*. PhD thesis, Doctoral Dissertation, King's College, University of Cambridge, 1989.

Christopher J.C.H. Watkins and Peter Dayan. *Technical note: Q-learning*, volume 8. Machine Learning, 1992.

Jigang Xie, Zhengding Qiu, and Jie Wu. Bootstrap methods for reject rules of fisher lda. In *International Conference on Pattern Recognition*, pages 425–428, 2006.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2016.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

Hongyi Zhang, Yann Dauphin, and Tengyu Ma. Fixup initialization: residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.

Richard Zhang and Efros Alexei A. Isola, Phillip. Colorful image colorization. In *European Conference on Computer Vision*, 2016.