Training Strategies for Time Series:
Learning for Prediction, Filtering, and Reinforcement Learning

Arun Venkatraman

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

CMU-RI-TR-18-01

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Thesis Committee
J. Andrew Bagnell, *Co-chair*
Martial Hebert, *Co-chair*
Jeff Schneider
Byron Boots, *Georgia Institute of Technology*

October 18, 2017

Abstract

Data driven approaches to modeling time-series are important in a variety of applications from market prediction in economics to the simulation of robotic systems. However, traditional supervised machine learning techniques designed for i.i.d. data often perform poorly on these sequential problems. This thesis proposes that time series and sequential prediction, whether for forecasting, filtering, or reinforcement learning, can be effectively achieved by directly training recurrent prediction procedures rather then building generative probabilistic models.

To this end, we introduce a new training algorithm for learned time-series models, Data as Demonstrator (DaD), that theoretically and empirically improves multi-step prediction performance on model classes such as recurrent neural networks, kernel regressors, and random forests. Additionally, experimental results indicate that DaD can accelerate model-based reinforcement learning. We next show that latent-state time-series models, where a sufficient state parametrization may be unknown, can be learned effectively in a supervised way using predictive representations derived from observations alone. Our approach, Predictive State Inference Machines (PSIMs), directly optimizes – through a DaD-style training procedure – the inference performance without local optima by identifying the recurrent hidden state as a predictive belief over statistics of future observations. Finally, we experimentally demonstrate that augmenting recurrent neural network architectures with Predictive-State Decoders (Psds), derived using the same objective optimized by PSIMs, improves both the performance and convergence for recurrent networks on probabilistic filtering, imitation learning, and reinforcement learning tasks. Fundamental to our learning framework is that the prediction of observable quantities is a *lingua franca* for building AI systems.

Acknowledgements

First and foremost, I want to thank my committee for their interest in my research. I am forever indebted to my advisors, Drew Bagnell and Martial Hebert; they have offered insight and guidance on all the work developed in this thesis and taught me to conduct, write, and present research. I was fortunate in being able to collaborate extensively with my external committee member, Byron Boots. His dedication was invaluable in helping me learn, and his patience even after I sent him a last-minute draft the morning of the deadline was vital throughout my studies. I also thank my undergraduate advisor Dr. Joel Burdick and my undergraduate roommate Sandeep Chinchali at Caltech for sparking my interest in robotics and research.

Being a graduate student can be a lonely affair, but I never felt lost due to the camaraderie in the LAIRLab: Stephane Ross, Dan Munoz, Alex Grubb, Felix Duvallet, Debadeepta Dey, Kevin Waugh, Amir-massoud Farahmand, Paul Vernaza, Abdeslam Boularias, Allie Del Giorno, Jiaji Zhou, Shreyansh Daftry, and Anirudh Vemula made research feel like a team effort. I am especially thankful for the opportunities I had to collaborate on research projects with Wen Sun, Roberto Capobianco, Hanzhang Hu, Shervin Javdani, Nick Rhinehart, Lerrel Pinto, and Kris Kitani. I am thankful for the mentorship of Moslem Kazemi, Katharina Muelling, and Jean-Sebastien Valois who taught me all that I know about working on real robotics systems during my time at NREC. I am grateful to Jessica Butterbaugh and Suzanne Lyons Muth who helped make any administrivia at CMU easier.

Even with all those above, my experience would have been incomplete without my robo-grad friends that entertained my crazy research discussions and helped make my time in Pittsburgh fun: Vishnu Desaraju, Michael Koval, Karthik Lakshmanan, Venkatraman Narayanan, Humphrey Hu, Kumar Shaurya Shankar, and Nitish Thatte. Finally, a very special thank you to Jennifer Ni, who helped me through the ups and downs the most. Her support and efforts to keep me going were the necessary ingredients for getting me across the finish line.

There are no words to describe the effort, advise, and dedication that my parents Arunachalam and Seethalakshmi Venkatraman and my brother, Amruth Venkatraman, have shown. Without their support and wisdom, I am certain that none of this would have been possible.

Contents

List of Figures

List of Algorithms

Part I

Introduction and Backgound

Chapter 1

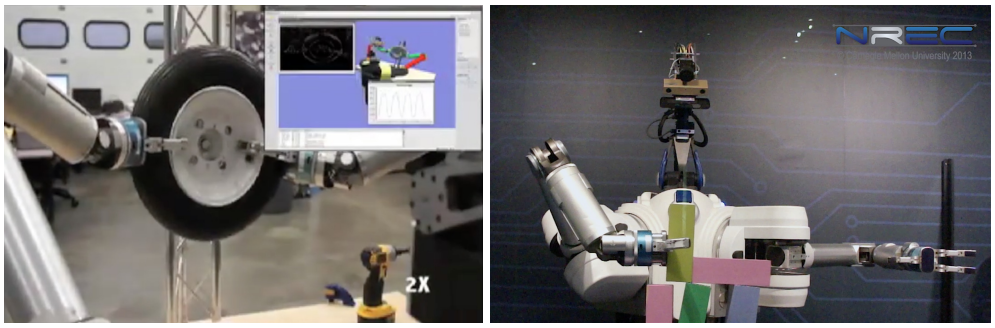
Introduction

Data driven approaches to modeling time-series are important across a variety of application domains from translating languages (Sutskever et al., 2014) in natural language processing to market prediction in economics (Marcellino et al., 2006) and to the simulation of robotic systems (Deisenroth et al., 2015). Data-driven approaches allow us to model complex systems where it often becomes difficult to robustly characterize the system *a priori* with analytic models. Indeed, machine learning and data modeling have became mainstays of technology. Much of the field of machine learning has been focused on the standard supervised learning problem (Bishop, 2006) and has been successfully used in a variety of applications, from spam filtering (Drucker et al., 1999) to handwriting recognition (Hull, 1994). However, these traditional learning techniques designed for i.i.d. data frequently perform poorly on sequential prediction problems such as those that arise when learning and using time-series models.

> *This thesis proposes that time series and sequential prediction, whether for forecasting, filtering, or reinforcement learning, can be effectively achieved by directly training recurrent prediction procedures rather then building generative probabilistic models.*

While there are many applications for time series and sequential prediction, this thesis is motivated by and often uses robotics examples for experimental evaluation, though the techniques we develop are more generally applicable. Our prior experiences with robotic systems (Fig. 1.1) showcase the breadth and diversity of the sequential and time-series problems that exist: from perception systems that must track objects in the workspace of the robot (e.g. Position of the wheel, holes, and pegs for wheel replacement (Bagnell et al., 2012) or seeing clutter in the environment (Katz et al., 2013)) to planning with dynamics (e.g. Stacking blocks to create a dynamically stable structure at the ARM-S Exhibit at the National Air and Space Museum[1]) to control (e.g. What action should the prosthetic make to help the user achieve their goal? (Muelling

[1] Video at `https://youtu.be/dSJP1BRuJ5Y`



(a) ARM-S Robot Replacing a Tire (left) and Exhibit at National Air and Space Museum (right)

(b) BCI Teleoperated Prosthetic (left) and Robotic Car (right)

Figure 1.1: Robotics problems span the space from perception to planning to control to state estimation. Each of these is often the result of a sequential process such as a visual tracking system, kinematics or dynamics planner, feedback control loop, and Bayesian filter respectively.

et al., 2015)[2]) and to state estimation (e.g. How to track the state of the car and objects around it from sensor data.[3]). All of these problems have a sequential process compent that is result of it being a time-series or having to make sequential predictions. Though many of the experimental setups we use throughout our work are simpler than those illustrated in Fig. 1.1, we hopefully develop foundational algorithms for using machine learning in these large-scale problems.

In robotics we can consider three clasees of time-series and sequential prediction problems: multi-step prediction and forecasting, filtering and state esimation, and control and reinforcement learning. To illustrate and ground these problems, we look at how they apply to robotic, autonomous self-driving vehicles. This is relevant as interest in these robotic systems has exponentially

[2]Video at `https://youtu.be/5KjLnyNxeyk`
[3]Robotic car image, Aurora Innovation, `http://aurora.tech/`

grown due to their potential[4] for large societal and environmental benefits.

The first issue, multi-step prediction and forecasting is a common time-series problem in robotics. The robot must predict what the world around it will look like in the future to reason about safety and what actions can be taken. An example of this is shown in Fig. 1.2 where an autonomous vehicle[5] uses a model $\hat{f}$ for sequential prediction – predicting the pedestrian's state in the future. However, any earlier error in the prediction can cascade and grow in future predictions.

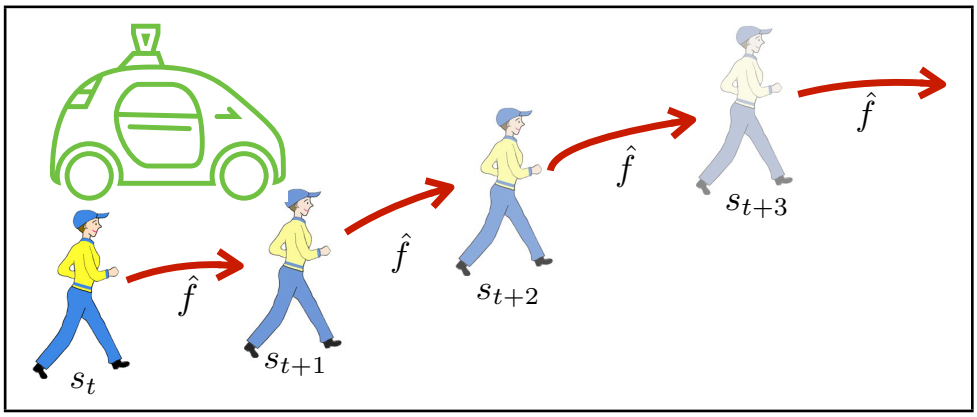


Figure 1.2: Using a model $\hat{f}$ recursively allows predict future states s_t of the pedestrian.

The second time-series problem often faced in robotics is finding a model $\hat{f}$ that achieves good performance for probabilistic filtering and state estimation. The model must be able to integrate a variety of different sensing modalities such as encoders and GPS to determine the underlying state. The challenge here is two fold. First, it can be difficult to find the analytic causative relationship between the sensor observation and the state of the system. Second, and more important, the true underlying state of the system may be directly unobservable or even unknown. An example state estimation problem is show in Fig. 1.3. The robotic vehicle uses GPS along with on-board sensors such as an IMU to estimate its position, velocity and acceleration. A predictive model $\hat{f}$ must be able to predict the state of vehicle even with noisy and lossy observations (e.g. GPS under occlusion – the orange state in Fig. 1.3).

The third sequential prediction problem confronted in robotics is selecting actions for the robot. The actions may be selected through a planning or control method in which the model $\hat{f}$ must be used to predict the future states of the robot after applying the actions. In Fig. 1.4, the model is used to predict the states of the self-driving vehicle to determine the proximity to obstacles, such as pedestrians.

[4] *Rethinking Transportation 2020-2030*, RethinkX

[5] "Google Car" icon by Guillaume Berry from thenounproject.com

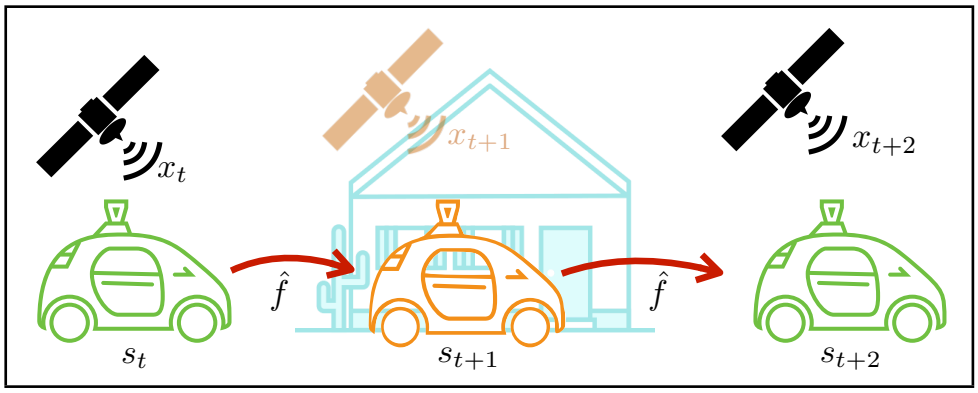


Figure 1.3: The model $\hat{f}$ is used to integrate sensor measurements x_t to predict the future state s_{t+1} (e.g. position, velocity) even under a noisy or lossy observation x_{t+1} (orange).

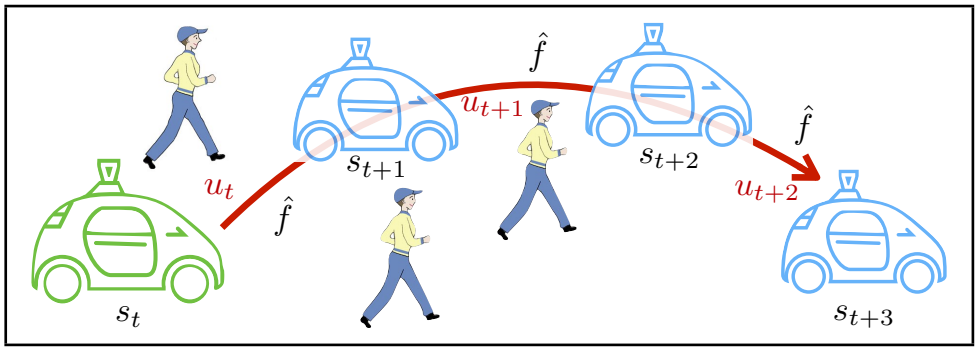


Figure 1.4: When predicting future motion u_t, the model $\hat{f}$ is used with the current state s_t to predict the future states of the vehicle.

While the three settings above were described individually, they are most often coupled. A forward predictive model $\hat{f}$ with good multi-step predictive capability is often required for good state estimation and control. A model for handling hidden state such as for probabilistic filtering is needed for control and prediction when only partial state information is known. In each of the above settings, machine learning provides a tantalizing alternative to manual specification of analytic models. However, utilizing machine learning is hampered in the above three settings, at least in part, by two key challenges which we investigate in this thesis.

Challenge 1.

Multi-step prediction cascades error.
How can we optimize time-series models for multi-step prediction that handle the model's own prediction error?

Towards Challenge 1, we introduce a new training algorithm for time-series prediction, Data as Demonstrator (DaD), based on the DAgger approach (Ross et al., 2011a) that improves the multi-step predictive capability of time-series models. Traditional methods optimize the one-step lookahead capability and iteratively apply such learned models to predict multiple steps into the future. DaD is a meta-algorithm that wraps these traditional approaches. We show theoretically and empirically (Chapter 3) that it improves multi-step performance on model classes from RNNs (Bengio et al., 2015) to random forests. The key insight is that training data can be reused as the oracle to learn from model's errors. We additionally show in Chapter 4 that DaD can be useful to accelerate model-based reinforcement learning.

Challenge 2.

Hidden state accentuates prediction error.
How can we better train recurrent predictors in partially observable systems?

We next show in Chapter 5 that latent-state time-series models, in which a sufficient state parametrization may be unknown, can be learned effectively in a supervised way using online instrumental variable regression. We show that our algorithm learns a model for the latent state system, that when used in a Kalman filter, reduces prediction error. In Chapter 6, we next show a method for directly learning a filter function for partially observable systems. More conventional approaches often choose parametric graphical model representations and more importantly decouple the model learning from probabilistic inference on the model, including the approach presented in Chapter 5. Using a DaD-style training procedure, our approach, the Predictive State Inference Machine (PSIM), directly optimizes the inference (filtering) performance by identifying the recurrent hidden state as a predictive belief over statistics of future observations. Additionally, when partial-state information is available for training, a common scenario in robotics applications, PSIM can be modified (Chapter 7) to predict this quantity and utilize it to improve performance. PSIM represents one of the first practical ways to replace hidden state with predictive belief learning. Our quantitative results indicate that our approach results in improved performance compared to spectral and backpropagation trained baselines. Finally, Chapter 8 develops a simple method using the lessons learned in PSIM to create Predictive-State Decoders (Psds), an easy-to-implement augmentation for general recurrent neural networks. We show that adding Psds improves empirical performance for probabilistic filtering, imitation learning, and reinforcement learning.

Central to our algorithms is that the prediction of observable quantities is a *lingua franca* for building AI systems. Many of the techniques developed in this thesis are meta-learning methods for time-series and sequential prediction that leverage existing supervised machine learning methods as a workhorse. The thesis' final insight is that directly training inference models for the target task has good performance, often more so than more generative graphical modelling approaches.

Chapter 2

Background

In this chapter, we setup the problems investigated in the thesis as well as summarize the relevant literature. We introduce classical single-step approaches and progress in the literature towards Challenge 1, improving multi-step prediction. We then introduce the problem of modelling partially observable systems and advances made toward Challenge 2. We finish with a short discussion on reinforcement learning as related to this thesis.

2.1 Time-series Modeling

2.1.1 Single Step Predictive Models

Determining models for time series data is important in applications ranging from market prediction to the simulation of chemical processes and robotic systems. As supervised learning algorithms have evolved and been developed in the machine learning community, they have been used towards modeling these systems (Sjöberg et al., 1995). The choice of learning algorithm is a result of trading off the desired model complexity, computational complexity of the algorithm, and its usefulness for solving a later task (e.g. control policy generation). We give in Table 2.1 a sample of the breadth of work in the machine learning community towards learning time-series models, organized by the learning algorithm. Many but not all of these assume, at least implicitly, that the system is observable. This assumption implies that learner's input (feature vector) is sufficient for predicting the future. We discuss this below in Section 2.2 and revisit it in Chapters 5 to 8 where we develop new algorithms to handle situations when there is a latent state space that could even have unknown parametrization.

Common to many of the algorithms in Table 2.1 and to many other typical statistical and machine learning approaches to time series modeling is to fit a model

$$\hat{f} : x_t \mapsto x_{t+1} \tag{2.1}$$

Machine Learning Technique	Example Literature
Linear Least Squares	(Abbeel et al., 2005b; Levine and Abbeel, 2014)
Neural Networks	(Narendra and Parthasarathy, 1990; Punjani and Abbeel, 2015)
Support Vector Regression	(Müller et al., 1997)
Gaussian process regression	(Wang et al., 2005; Ko et al., 2007; Deisenroth and Rasmussen, 2011)
Nadaraya-Watson kernel regression	(Basharat and Shah, 2009)
Gaussian mixture models	(Khansari-Zadeh and Billard, 2011; Levine et al., 2016)
Kernel PCA	(Ralaivola and D'Alche-Buc, 2004; Chan and Vasconcelos, 2007)

Table 2.1: Machine Learning Techniques applied to Dynamical System Modeling

to optimize a single-step prediction error

$$\sum_t \ell_f(\{(x_t, x_{t+1})\}_i).$$

This single-step loss corresponds to traditional supervised learning objective, using input features $x \equiv x_t$ and targets $y \equiv x_{t+1}$ to optimize a loss $\ell(\hat{f}(x), y)$ such as squared loss or likelihood loss to find $\hat{f}$. This process is shown in Fig. 2.1.

To using such models for forecasting multiple-steps into the future, the learned model $\hat{f}$ is recursively applied, feeding through the previous output $\hat{x}_t$ as its new input to predict $\hat{x}_{t+1}$. Such an estimator $\hat{f}$, however, inevitably introduces errors with each prediction, and recursive use of the model $\hat{f}$ to predict into the future accentuates the error through a feedback effect (see Theorem 3.1.1). The compounding errors alter the input distribution for future prediction steps, breaking the train-test i.i.d assumption common in supervised learning. Thus, optimizing the single-step predictive loss does not guarantee accurate multiple-step simulation accuracy.

The cascading of modeling errors is well known in the planning, control, and reinforcement learning literature. For example, failures in planning can often be attributed to inaccuracies in modeling even when using physics-based generative models (Koval et al., 2016). For controller synthesis problems, learned models are often used to optimize a policy or generate future sequence of controls. This optimization over the control problem horizon can be sensitive to long term prediction errors in the modeling (Abbeel et al., 2006; Nguyen-Tuong and Peters, 2011; Heess et al., 2015).

2.1.2 Direct Method for Multi-Step Predictive Models

Various methods to handle or mitigate the multi-step predictive error have been proposed. These approaches can usually be split into two groups, direct approaches or iterative approaches. In the direct approach, a model is fit to predict k steps in the future (Chevillon and Hendry, 2005; Langer et al., 2016):

$$\hat{f}_{(k)} : x_t \mapsto x_{t+k}$$

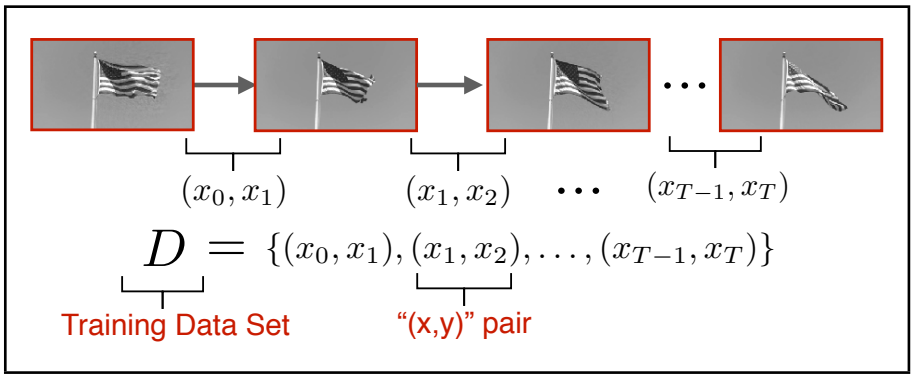


Figure 2.1: Learning a single-step predictive model consists of taking the time-series $x_0, \ldots, x_T$ (e.g., from a video texture, Section 3.3.4) and constructing a training dataset D of transitions (x_t, x_{t+1}) which can be used as the traditional features 'x' and targets 'y' for any supervised machine learning algorithm.

In contrast, iterative approaches (which we also refer to as recursive or recurrent), fit a model $\hat{f} : x_t \mapsto x_{t+1}$ (Eq. (2.1)) and repeatedly compose the model to predict

$$x_{t+k} = \overbrace{\hat{f} \circ \cdots \circ \hat{f}}^{k \text{ times}}(x_t).$$

Much of the analysis comparing direct and iterative models is is focused on the auto-regressive (AR) class of models (Weiss, 1991; Marcellino et al., 2006). The direct method's advantage is that it directly optimizes for the prediction to a fixed look-ahead length and does not suffer from bias introduced by sequential use of the iterative model for that same length. To achieve arbitrary horizons, one can fit predictors for various k and compose those functions (Langer et al., 2016). However, such models have a couple drawbacks. First, models at many scales are required for making forward predictions at arbitrary time-lengths, increasing the model complexity and thus sample complexity. The other drawback is that to achieve arbitrary time-horizon predictions still requires compositions of the functions, which leads to same feedback issues as with the iterative model. Marcellino et al. (2006) empirically found that iterative AR models perform better than direct models.

2.1.3 Iterative Method for Multi-Step Predictive Models

In this thesis, we focus on iterative (i.e. recursive or recurrent) use of time-series models as most machine learning approaches (Table 2.1) are generally used in this fashion. Within iterative models, there has been work in addressing the challenge of sequential prediction with the model. One class of such works attempts to handle this by propagating model uncertainty through the

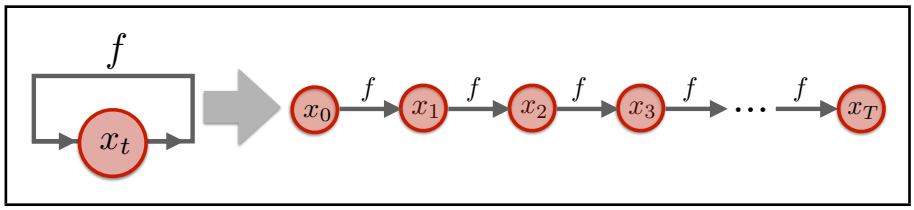


Figure 2.2: Recurrent or iterative methods use a single model f to roll-out predictions from an initial input.

predictions (Candela et al., 2003; Girard et al., 2003) over time. This uncertainty can be used in computing the expected rewards when optimizing a control policy (Deisenroth and Rasmussen, 2011). Though this approach helps to mitigate the effect of compounding model errors, these methods may not precisely capture the true cascading multi-step modeling inaccuracies. Additionally, this approach does not address how to improve the multi-step prediction itself but merely at capturing the resulting uncertainty.

An alternative is to setup a loss function that captures the multi-step predictive error when using the iterative model. Abbeel and Ng (2005a) introduce a "lagged-error" criterion in contrast with the traditional single-step maximum-likelihood objective commonly used. However, loss functions like the lagger-error criterion are often non-convex in the model parameters due to the recursive prediction and is difficult to solve. Expectation-Maximization (EM), an iterative optimization approach, can be used (Ghahramani and Roweis, 1999; Abbeel and Ng, 2005a; Coates et al., 2008) for this sort of optimization. However, EM is prone to local-minima and therefore is heavily dependent on initialization. Abbeel et al. (2005b) propose a simpler yet still non-convex objective that still only achieve local optimality.

We finish with a discussion on "backpropagation-through-time" (BPTT) (Werbos, 1990), a method that has seen resurgent interest and popularity (LeCun et al., 2015). BPTT (Fig. 2.3) extends back-propagation (i.e. gradient-descent with chain-rule) for training differentiable models in networks with recurrence relations. Unfortunately, such gradient methods are limited in the model classes they can consider, effectively ruling out broad classes of some of the most effective regression algorithms including decision trees and random forests. Moreover, such methods – even on simple linear predictive models – tend to suffer from a "gradient collapse" and ill-conditioning (Bengio et al., 1994), where the gradient decreases exponentially or "explodes" exponentially in the prediction horizon T. While various attempts to improve the stability of BPTT have been proposed such as truncated gradients (Sutskever, 2013) or gradient clipping (Pascanu et al., 2013), to our knowledge, no formal guarantees have been provided for BPTT or its variants for optimizing the multi-step predictive error. Even so, BPTT has been used to great success in fields such as natural language processing through its use for training recurrent neural networks, which we discuss below in Section 2.2.3. In this thesis, we present an alter-

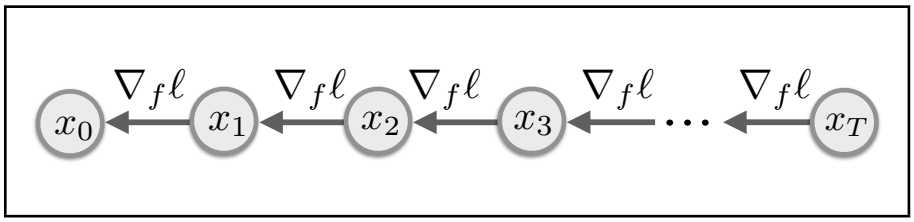


Figure 2.3: Backpropagation through time uses the chain rule to update the model at timestep t with the gradients for all the timesteps in the future. This results in an update to the model which reasons about how the local change will cascade for future timesteps of prediction.

native training procedure in Chapter 3 that provides a theoretical performance bound for the multi-step predictive performance. We apply this for both general time-series problems as well as on partially observed systems. Additionally, we present experimental findings in Chapter 8 which show that augmenting a recurrent network with secondary objective related to multi-step future prediction improves performance and convergence when using BPTT.

2.2 Modeling Partially Observable Systems

A large family of time-series and sequential problems fall within the class of partially observable systems. These target tasks are often believed to have a *latent state* sequence that characterizes the underlying sequential data-generating process. However, direct recording of the latent state is usually impossible and only *observations* of the system are accessible. In partially-observable systems, a single observation is not guaranteed to contain enough information to fully represent the system's latent state. For example, a single image of a robot is insufficient to characterize its latent velocity and acceleration. While a latent state parametrization may be known in some domains – *e.g.* a simple pendulum can be sufficiently modeled by its angle and angular velocity $(\theta, \dot{\theta})$ – data from most domains cannot be explicitly parametrized.

To model sequential prediction problems, it is common to cast this problem into the Markov Process framework. Predictive distributions in this framework satisfy the Markov property:

$$P(s_{t+1}|s_t, s_{t-1}, \ldots, s_0) = P(s_{t+1}|s_t)$$

where s_t is the latent state of the system at timestep t. Intuitively, this property tells us that the future s_{t+1} is only dependent on the current state. Note that in Markov Decision Processes (MDPs), $P(s_{t+1}|s_t)$ may depend on an action taken at s_t. With the Markov assumption, s_t and does not depend on any previous state $s_0, \ldots, s_{t-1}$. As s_t is latent, the learner only has access to observations x_t, which are produced by s_t. For example, in robotics, x_t may be joint angles from sensors or a scene observed as an image. A common graphical model

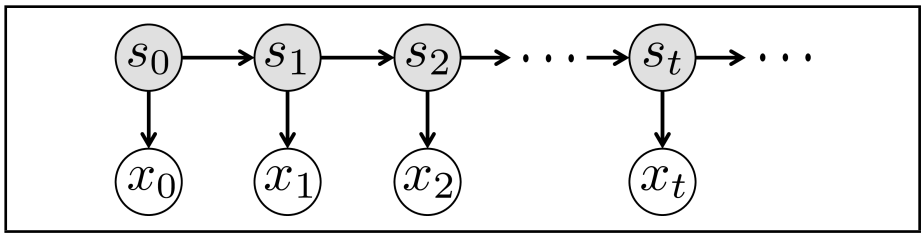


Figure 2.4: The process generating sequential data has latent state s_t which generates the next latent state s_{t+1}. s_t is often unknown but generates the observations x_t (e.g. sensor measurements) which are used to learn a model for the system.

representation is shown in Fig. 2.4. The learning goal is to find a model and possibly an *internal state* representation that allows for accurate prediction under a defined loss.

Below, in Sections 2.2.1 and 2.2.2, we discuss literature in the field of modeling time-series problems. Many of these approaches look at modeling partially observable systems for the purposes of developing Bayesian filters (Thrun et al., 2005) to estimate the underlying state given observations (e.g. sensor readings). The filter's state estimate can then be used to control the underlying system, for example through model-based reinforcement learning. In Section 2.2.3, we give an overview of a different class of approaches, recurrent neural networks, that aim to train discriminative models for a target application without directly reasoning about the underlying data-generating processes. This approach has resulted in great strides forward in fields such as natural language processing.

2.2.1 System Identification and Filtering

A body work of related to time-series modeling has been conducted in system identification, which specifically looks at the statistical modeling of dynamical systems from data. The literature in this field is expansive (Åström and Eykhoff, 1971; Sjöberg et al., 1995; Ljung, 1998). The primary use of system identification for Bayesian filtering comes in specifying and fitting (learning) the system (transition) model and the observation (sensor) model. The system model defines how the state of the system evolves and the observation model defines how a sensor generates an observation given a state.

The work in system identification can be split into those that handle identification in the *supervised*-state or *latent*-state settings. The supervised-state setting is more commonly found in the machine learning literature. It assumes ground-truth access to both the system's states and observations at training time to learn a dynamics model (Ralaivola and D'Alche-Buc, 2004; Ko et al., 2007; Deisenroth et al., 2009; Levine et al., 2016). For example. Ko et al. (2007); Deisenroth et al. (2009) use Gaussian Process to optimize two separate models for the state transition model and observation model. They are are then able

to exploit the probabilistic nature of the Gaussian process to better the predict and update steps of the Bayesian filtering. However, many real systems are not Gaussian and may be poorly represented by such an assumption.

Additionally, many times, we are unable to instrument a system to get access to the full state to collect ground truth. At other times, we may not know what form the true state of the system takes. In the latent-state formulation, the goal is to build a predictive model for predicting future observations. One way of overcoming this difficulty can be with latent-variable models like Hidden Markov Models (HMMs) (Siddiqi et al., 2010; Hsu et al., 2012; Song et al., 2010) or with more classical linear dynamical system (LDS) state space models (Van Overschee and De Moor, 2012), which represent the belief state as a distribution over the latent-state space of the model. In this thesis however, we leverage ideas from Predictive State Representations (PSRs) (Littman et al., 2001b; Singh et al., 2004; Rudary and Singh, 2005; Wingate and Singh, 2006; Boots et al., 2011; Hefny et al., 2015a; Venkatraman et al., 2016b). Unlike the previously presented latent state-space models, PSRs maintain the belief state as an equivalent belief over sufficient features of future observations. It has been observed that this structure can be more expressive than the HMM (Singh et al., 2004) and can subsume the LDS models (Rudary and Singh, 2005). These learned models can then be used for inference tasks such as Bayesian filtering or forward prediction. As a positive, this family of algorithms provides theoretical guarantees on discovering the global optimum for the model parameters under the assumptions of infinite training data and realizability. However, in the non-realizable setting — i.e. model mismatch (e.g., using learned parameters of a Linear Dynamical System (LDS) model for a non-linear dynamical system) — these algorithms lose any performance guarantees on using the learned model for filtering or other inference tasks. They can also perform arbitrarily bad (Kulesza et al., 2014) for example when the learned model rank is lower than the rank of the underlying dynamical system.

2.2.2 Direct Filter Learning

In contrast to explicitly building transition and observation models to use in Bayesian filtering, there has been effort to directly optimize the filter performance itself. Some of these methods directly try to address failures in existing filtering algorithms by using learning-based approaches to try to improve the selection of a filter's hyper-parameters (Turner and Rasmussen, 2012) given a fixed transition and observation model. Other approaches also try to improve performance of within the Kalman family of filtering algorithms by learning noise covariance models that maximize the likelihood of states or observations (Abbeel et al., 2005a; Vega-Brown and Roy, 2013; Hu and Kantor, 2015). All these works assume known dynamics and observation models as well as a known state representations for the internal state. Recent work augments the aforementioned additionally learning part of the observation model using deep learning in an end-to-end fashion (Haarnoja et al., 2016). However, these approaches have a few drawbacks. They assume the structure of the Kalman family of filters;

the systems evolve under a Gaussian distribution and often with linear transition and observation models (e.g. (Haarnoja et al., 2016)) for the belief state update. Though these can work adequately for many applications, it can be hard to model more complex systems where a sufficient state parametrization[1] may be unknown. Ondruska and Posner (2016) propose a full end-to-end model that learns a latent state representation as well as the belief state transition model and observation model. Langford et al. (2009) trains four discriminative models to learn aspects of the filtering function: an initialization function, an update function, an observation function, and a transition function. The multi-step nature of the likelihood objectives forces all these approaches to use backpropagation-through-time[2] or an EM style optimization that faces computational hardness for finding the globally optimal solution.

2.2.3 Recurrent Neural Networks

While choosing a fixed representational form for the modeled internal state can improve data-efficiency by reducing model complexity, as some of the above approaches do, it can adversely affect performance if the underlying representation is not known or well understood. For complex problems such as as speech recognition (Graves and Jaitly, 2014), text generation (Sutskever et al., 2011), or generating images (van den Oord et al., 2016), it is difficult to parametrize and define a sufficient internal state representation. These approaches rely on a learning architecture known as the Recurrent Neural Network (RNN) (LeCun et al., 2015; Sutskever, 2013), which uses an open-ended representation for its internal state.

RNNs employ internal states to summarize previous data, serving as a learner's memory. We avoid the terminology "hidden state" as it refers to the internal state in the RNN literature but refers to the latent state in the HMM, PSR, and related literature. Internal states are modified towards minimizing the target application's loss, e.g., minimizing observation loss in filtering or cumulative reward in reinforcement learning. The target application's loss is not directly defined over the internal states: they are updated via the chain rule (backpropagation) through the global loss. Although this modeling is indirect, recurrent networks nonetheless can achieve state-of-the-art results on many robotics (Duan et al., 2016; Hausknecht and Stone, 2015), vision (Ondruska and Posner, 2016; van den Oord et al., 2016), and natural language tasks (Chung et al., 2015; Graves and Jaitly, 2014; Ranzato et al., 2016) when training succeeds. These models often directly target the multi-step predictive loss

$$\min_f \mathcal{L} = \min_f \sum_t \ell_t(f(h_t, x_t))$$

by optimizing the network parameters with back-propagation-through-time (BPTT). As discussed in Section 2.1.3, BPTT often has stability issues and the techniques

[1] We define a sufficient state as one that satisfies the Markov assumption.

[2] BPTT is similar to EM in that it has a forward pass which mimics the E-step and a backwards pass that follows the gradient as the M-step.

to improve stability do so by discarding information about how future observations and predictions should backpropagate through the current internal state. A significant innovation in training internal states of recurrent models with long-term dependence was the LSTM (Hochreiter and Schmidhuber, 1997). Many variants on LSTMs exist (*e.g.* GRUs (Cho et al., 2014)), yet in the domains evaluated by Greff et al. (2016), none consistently exhibit statistically significant improvements over LSTMs. While the resurgence of RNNs has shown great success, recurrent model optimization is hampered by two main difficulties: 1) non-convexity, and 2) the loss does not directly encourage the internal state to model the latent state. A poor internal state representation can yield poor task performance, but rarely does the task objective directly measure the quality of the internal state. In this thesis, we develop a new method for improving the training of recurrent neural networks using lessons learned when developing training procedures for modeling partially observed dynamical systems.

2.3 Reinforcement Learning

Many of the time-series modeling techniques we develop in this thesis are inspired by problems in robotics where the final goal is control actuation via planning, imitation learning, or reinforcement learning. Traditional methods often require a lot of manual tuning of models. Learning based approaches then offer a potential panacea for automating the process. Learning based approaches for controlling autonomous agents fall into two primary categories: model-based (Schaal et al., 1997; Bakker et al., 2006; Hester et al., 2012) and model-free (Thrun, 1995; Matarić, 1997; Duan et al., 2007; Konidaris et al., 2011; Mnih et al., 2015). Common to both is the target objective,

$$\max_{\pi:\{u_0,\ldots,u_T\}} \sum_{t=0}^{T} r_t(s_t, u_t) \Leftrightarrow \min_{\pi:\{u_0,\ldots,u_T\}} \sum_{t=0}^{T} c_t(s_t, u_t),$$

which aims to maximize accumulated rewards or equivalently minimize accumulated costs over a finite or infinite time horizon for a sequence of states s_t and actions (controls) u_t. The actions $u_{0:t}$ may come from a parametrized closed loop policy $\pi : s \mapsto u$ that chooses an action given a state or an open loop policy $\pi : t \to u$ that returns an action for each time step of the problem. In many reinforcement learning (RL) problems, the functional form of the reward r or cost c is unknown, making the problem more difficult. In Chapter 4, we assume knowledge of $c(\cdot)$, but in Chapter 8, we do not. RL problems are also often difficult since the relationship between s_t and s_{t+1} is unknown *a priori*. In model-based approaches, this provides the starting point, developing a model to map the state transition. In model-free methods, explicit dynamics modelling is eschewed for directly learning a policy to map states (or observations) to actions.

In model-based reinforcement learning, a system transition function – a dynamics model – is used to guide the creation of a control policy. The quality of

this policy is often a function of the quality of the dynamics model. A specific control methodology, Model Predictive Control is especially reliant on the dynamics models for doing multi-step rollouts (Kelly et al., 2006; Williams et al., 2016). The ability to accurately predict state sequences in the future affects the capability to generate a good sequence of controls. Though approximate models can be used, to improve performance (Thomas and Brunskill, 2016), often times, a poor model makes the reinforcement learning problem more harder. Finally, there has been work to formalize the methods control engineers have long done in practice. Abbeel et al. (2006) formalized the iterative procedure between generating learning a dynamics model, optimizing a control policy, collecting data on the real system and starting again. This process allows for the distribution on which data is collected to match that seen by the policies. This process was further studied and developed by Ross and Bagnell (2012) to provide model-agnostic theoretical guarantees.

Though model based RL methods are thought to have a better sample complexity (e.g. having the true model can make it easier to solve difficult problems such as Go (Silver et al., 2016)), model-free methods such as Q-learning (Mnih et al., 2015) and policy gradient approaches (Kakade, 2002; Schulman et al., 2015) have gained popularity often due to failures of planning with imperfect dynamics models. Heess et al. (2015) developed a family of Stochastic Value Gradient algorithms that move along the spectrum from using a dynamics model to not. The primary learning is done by the model-free Value gradient network, but in one variation, a learned dynamics model neural network is used to improve this estimate for a single-step in time. The authors specifically note a failure for propagating the gradients longer in time through the dynamics model. A more direct 'in-between' model-free and model-based RL are those that look to optimize and estimate of the future rewards while passing around a recurrent internal state (Lin and Mitchell, 1992, 1993; Hausknecht and Stone, 2015) or for policy optimization over future predicted states and actions (Mordatch et al., 2015; Duan et al., 2016). In this thesis, we look at improving how recurrent learners are used for both model-based (Chapter 4) and model-free (Chapter 8) reinforcement learning methods.

Part II

Improving Multi-step Predictive Performance

Chapter 3

Data as Demonstrator: Training Time-Series Models for Multi-step Prediction

Most supervised machine learning techniques used for time-series problems face the challgenge of having poor predictive performance when used for forecasting multiple steps into the future (Challenge 1). These approachs often target minimizing the single-step error, defined as the error in predicting a future state or observation x_{t+1} given x_t. The prevalence of single-step modeling approaches is a result of the difficulty in directly optimizing the multiple-step prediction error. As an example, consider fitting a simple linear dynamical system model for the multi-step error over the time horizon T from an initial condition x_0,

$$A^* = \arg\min_A \sum_{t=1}^{T} \left\| x_t - A^t x_0 \right\|_2^2 \tag{3.1}$$

Even this seemingly innocuous squared-loss objective is difficult to optimize in two ways: it is non-convex in A, and though differentiable, the matrix power derivatives are non-trivial. In comparison, the single-step squared loss used in supervised learning,

$$A^* = \arg\min_A \sum_{t=0}^{T-1} \left\| x_{t+1} - A x_t \right\|_2^2 \tag{3.2}$$

is more appealing to solve as it has an easy, closed form solution. Abbeel and Ng (2005a) propose a generalization of (Eq. (3.1)) coined the "lagged error"

This work was originally presented in *Improving Multi-Step Prediction of Learned Time Series Models* at AAAI 2015 (Venkatraman et al., 2015).

criterion which penalizes deviations during forward simulation. However, as noted by the authors, this objective is also non-linear and non-convex as the second step prediction is conditioned on the output of the first. If the predictive model is differentiable, one can apply "backpropagation-through-time" (Werbos, 1990), which however has its own difficulties during optimization as discussed in Section 2.1.3. Finally, to our knowledge, no formal guarantees have been provided for any such optimization of multi-step predictive error. Perhaps as a result, many approaches in the literature focus on using single-step prediction models to take advantage of techniques developed in the statistical and machine learning communities.

In this section, we introduce a new meta-algorithm, DATA AS DEMONSTRATOR (DAD), for improving the multi-step prediction capability of a time-series learner.

- We propose a simple, easy to implement meta-algorithm, DAD, that can wrap an existing time-series learning procedure.
- Our method makes no assumption on differentiability. This allows our algorithm to be utilized on top of a larger array of supervised learning algorithms.
- Our method is data-efficient in improving a learned model. Without querying the actual system for more training data, our method is able to achieve better performance on the multi-step criterion by reusing training data to correct for prediction mistakes.
- Moreover, through a reduction to imitation learning, we demonstrate that when the learner exhibits the no-regret property, we can provide performance guarantees that relate the one-step predictive error to the multi-step error.

Finally, we demonstrate experimentally that our method improves the multi-step prediction error.

3.1 Problem Setup

We consider the problem of modeling a discrete-time time-series (system) characterized by a time-indexed state x_t generated from some stationary dynamics,

$$x_{t+1} = \mathbb{E}[f(x_t)] \tag{3.3}$$

The problem we address is to learn a model given K sample trajectories $\xi_k \in \Xi$ of $\{x_0, x_1, \ldots, x_{T_k}\}$ generated by the system (Eq. (3.3)). As motivated in the introduction, it is easiest to learn a forward prediction model by considering the prediction of single, consecutive steps in each trajectory. To do so, we create a dataset D of input-target pairs $\{(x_t, x_{t+1})\}_i$ and optimize for a learned model:

$$\widehat{f} = \arg\min_{f \in \mathcal{F}} \sum_i \ell_f(\{(x_t, x_{t+1})\}_i) \tag{3.4}$$

for some regression loss function ℓ and class of models $\mathcal{F}$. For multiple-step prediction and simulation with the learned model $\widehat{f}$, we follow the simple two-step procedure:

Recursive Time Series Prediction: (3.5)

Step 1: $\hat{x}_{t+1} = \widehat{f}(\hat{x}_t)$

Step 2: Return to **Step 1** with $\hat{x}_t \leftarrow \hat{x}_{t+1}$

In traditional supervised learning, we assume independence of each prediction. If each prediction with with $\widehat{f}$ is bounded with error ϵ, making T such predictions for a *rollout* with the model (Eq. (3.5)) leads to $T\epsilon$ error. Extending this, one may naively intuit that the multi-step predictive error of predictions made through a T-length rollout is also linear in the number of predictions as well as in the supervised setting. However, as mentioned in Section 2.1.1, the predictions $\hat{x}_t$ are not from the same distribution that the true x_t where generated from, i.e., at each execution of Step 1, prediction error from $\widehat{f}$ results in a state that could be outside of the training distribution represented in the dataset of trajectories $\Xi = \{\xi_i\}$. This multi-step error can grow to be up to exponential in T, even under nice conditions.

Theorem 3.1.1. *Let $\widehat{f}$ be learned model with bounded single-step prediction error $\|\widehat{f}(x_t) - x_{t+1}\| \leq \epsilon$. Also let $\widehat{f}$ be Lipshitz continuous with constant $L > 1$ under the metric $\|\cdot\|$. Then, $\|\widehat{f}(\hat{x}_T) - x_{T+1}\| \in O\left(\exp(T\log(L))\epsilon\right)$*

Proof. From the Lipshitz continuous property, bounded error assumption, and the triangle inequality, we can show that

$$\|\widehat{f}(\hat{x}_T) - \widehat{f}(x_T)\| \leq L\|\widehat{f}(\hat{x}_{T-1}) - \widehat{f}(x_{T-1})\| + L\epsilon.$$

Applying the same process, we eventually get

$$\|\widehat{f}(\hat{x}_T) - \widehat{f}(x_T)\| \leq \textstyle\sum_{t=1}^{T} L^t\epsilon.$$

Using the bounded error assumption along with another application of triangle inequality, we arrive at $\|\widehat{f}(\hat{x}_T) - x_{T+1}\| \leq \sum_{t=0}^{T} L^t\epsilon \in O\left(\exp(T\log(L))\epsilon\right)$. □

The bound in Theorem 3.1.1 tells us that the multi-step prediction error is bounded by an exponential in the time horizon. This bound is tight. Consider a fitted a linear model $\hat{A} > 1$ on a 1-D training trajectory ξ, with bounded error ϵ and Lipshitz constant $\hat{A}$. If we make a single-step error ϵ on the first step of forward simulation, $\hat{x}_1 = \hat{A}x_0 = x_1 + \epsilon$, we get: $\hat{x}_{T+1} = \hat{A}^T(x_1 + \epsilon)$ Then,

$$\|\hat{x}_{T+1} - x_{T+1}\| = \|\hat{A}^T(x_1 + \epsilon) - x_{T+1}\| \in \Omega(\hat{A}^T\epsilon) \tag{3.6}$$

It is also worthwhile to note the Lipshitz constant conditions: For $L > 1$, we get the bound in Theorem 3.1.1. If $L = 1$, the bound reduces to $O(T\epsilon)$, which is equivalent to the result for the supervised learning setting. Finally, for $L < 1$, we get $O(L\epsilon)$. This situation specifies a stable fitted model that decays to

zero change in state over the time horizon. For many time-series problems, this results in boring simulated behavior as the predictions converge to the mean, as exemplified with the "Fireplace" video texture in the experimental sectionSection 3.3. For many real-world problems, a model that is near the limit of stability is preferred. As a result, we may learn approximations that are unstable, yielding the exponential bound as shown above. In the following sections, we motivate and develop an algorithm which can achieve regret linear in the prediction horizon.

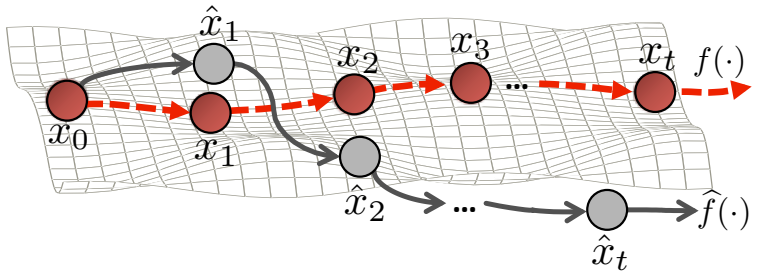
3.2 Data as Demonstrator Algorithm

To resolve differences in the train and test (forward prediction) distributions, it would be convenient to augment our training set to meet the test distribution. This is reminiscent of work that uses the predictions from the learner to augment the dataset by searching for (Daumé III et al., 2009) or querying an oracle (Ross et al., 2011a) for the optimal next prediction to make given the previous prediction. Though a natural approach, collecting new data from the true system on the induced 'test' distribution may be impossible. Practically, it could be expensive to query the true system at the incorrectly predicted state (e.g. an inverted helicopter). The more impeding concern, however, is that the predicted states may not be feasible for the generating system. Consider data collected from a pendulum on a string with radius 1. The mechanics of the system require the Euclidean coordinates to reside on the unit circle. During forward simulation with an approximate model, we may predict points off this constraint manifold from which we can never collect a true transition. This situation is illustrated in Fig. 3.1(a). Though we need to augment the dataset with predictions, the above reasons make it often difficult to get a ground truth response for the target. Below, we introduce a new algorithm for generating an "oracle" without requiring computation of the true system's optimal next prediction. Instead, we synthetically generate correction examples for the learner to use. Since the given training trajectories are time-indexed, they can provide a correction for each time step when simulating from points along the training trajectories, as depicted in Fig. 3.1(b). This idea motivates our algorithm, Data as Demonstrator (DaD), detailed in Algorithm 1.

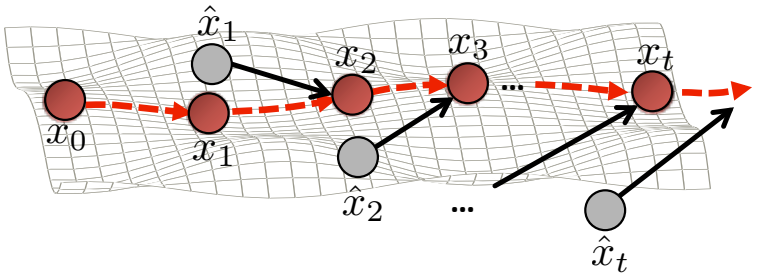
3.2.1 Reduction to imitation learning

Data as Demonstrator forward simulates a learned model, collecting data on the encountered prediction, 'test' time, distribution by simulating from points along training trajectories. The next model is trained from input-target pairs created by pointing the 'test' time prediction to the correct next time-indexed state along the trajectory. By iterating and retraining a new model on the aggregate dataset, DaD can be considered a *Follow-The-Leader* algorithm.

Since we are applying corrections from the dataset, we can also interpret DaD as a simplified scenario of interactive imitation learning. Let the expert



(a) Forward simulation of learned model (gray) introduces error at each prediction step compared to the true time-series (red)



(b) Data provides a demonstration of corrections required to return back to proper prediction

Figure 3.1: In typical time-series systems, realized states of the true system are a small subset or even a low-dimensional manifold of all possible states. Cascading prediction errors from forward simulation with a learned model will often result in predicted infeasible states **(Fig. 3.1(a))**. Our algorithm, DATA AS DEMONSTRATOR (DAD), generates synthetic examples for the learner to ensure that prediction returns back to typical states **(Fig. 3.1(b))**.

be the training data which "demonstrates" expert actions by specifying at each point in time the correct state for the next time step. The learned time-series model $\hat{f}$ acts as the state dependent action policy $\hat{\pi}$. The state dynamics simply pass on the predictions from the learned model as the input for the next state.

This reduction to the interactive imitation learning setting allows us to avail ourselves of the theoretical guarantees for the Dataset Aggregation algorithm (DAgger) introduced in (Ross et al., 2011a). Notationally, let a ground truth trajectory from the underlying system be $\xi = \{x_0, x_1, \ldots\}$, and let $\hat{\xi} = \{x_0, \hat{x}_1, \hat{x}_2, \ldots\}$ denote the trajectory induced by starting at x_0 from the true trajectory and iteratively applying the model f as described in the two-step forward prediction procedure. Let

$$P_f := P_f(\hat{\xi}, \xi) \tag{3.7}$$

denote the distribution of the time-synchronized pairs $(\hat{x}_t, x_t)$ from the predicted

Algorithm 1 DATA AS DEMONSTRATOR (DAD)

Input:
▷ Number of iterations N, set $\{\xi_k\}$ of K trajectories of time lengths $\{T_k\}$.
▷ No-regret learning procedure LEARN
▷ Corresponding PREDICT procedure for multi-step prediction that takes an initial state, model, and number of time steps.
Output: Model $\widehat{f}$

Initialize aggregate data set $D \leftarrow \{(x_t, x_{t+1})\}$ of $(T_k - 1)$ input-target pairs from each trajectory ξ_k
Train initial model $f_0 \leftarrow \text{LEARNER}(D)$
for $n = 1, \ldots, N$ **do**
  for $k = 1, \ldots, K$ **do**
    $(\widehat{x}_1, \ldots, \widehat{x}_T) \leftarrow \text{PREDICT}(\xi_k(0), f_{n-1}, T_k)$
    $D' \leftarrow \{(\widehat{x}_1, \xi_k(2)), \ldots, (\widehat{x}_{T_k-1}, \xi_k(T_k))\}$
    $D \leftarrow D \cup D'$
  end for
  $f_n \leftarrow \text{LEARN}(D)$
end for
return $\widehat{f} \leftarrow f_n$ with lowest error on validation trajectories

states and the true system's trajectory.

Let ϵ_N be the true loss of the best model in hindsight, defined as

$$\epsilon_N = \min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{x \sim P_{f_i}} [\ell_f(x)].$$

Finally, let $\widehat{f} = \arg\min_{f \in f_{1:N}} \mathbb{E}_{x \sim P_f} [\ell_f(x)]$ be the model returned by DAD that performed best on its own induced distribution of states. We are then able to achieve the following performance guarantee:

Theorem 3.2.1. *Given a bounded single-step prediction (regression) loss ℓ and associated no-regret learning procedure* LEARN*,* DAD *has found a model $\widehat{f} \in f_{1:N}$ as $N \to \infty$, such that $\mathbb{E}_{x \sim P_{\widehat{f}}} \left[\ell_{\widehat{f}}(x)\right] \leq \epsilon_N + o(1)$.*

Proof. We setup the imitation learning framework as described earlier: learned policy $\hat{\pi} = \widehat{f}$ and degenerate state dynamics that rely on the policy (learned model) to solely transition the state. By this reduction to imitation learning, the result follows from Theorem 4.1 of (Ross et al., 2011a). □

We can also relate the number of iterations of DAD to get a linear performance guarantee with respect to the prediction horizon for the regularized squared loss, a commonly used regression loss. Letting $J(f) = \sum_{t=0}^{T} \ell[(\hat{\xi}(t), \xi(t))]$ define the multiple-step prediction error, we get:

Theorem 3.2.2. *If ℓ is the regularized squared loss over a linear predictor (or kernel linear regressor) and if N is $\tilde{O}(T)$, then $\exists\ \widehat{f} \in f_{1:N}$ found by* DAD *such that $J(\widehat{f}) \leq O(T \cdot \epsilon_N)$*

Proof. The regularized squared loss on a linear prediction model (or on a kernel linear regressor) is strongly convex in the model parameters. The result then follows from our reduction and Theorem 3.2 of (Ross et al., 2011a). □

Intuitively, these results tell us that for a given time-series modeling problem, we either fail to find a model because the generation of synthetic training points creates conflicting inputs for the learner when our new data-points overlap or we guarantee good performance after a certain number of iterations. Additionally, the performance guarantees given by Theorem 3.2.1 and Theorem 3.2.2 have some subtleties and depend on a few critical assumptions. Firstly, DAD gives no guarantee on any specific learned model $f \in f_{1:N}$ from each iteration but only that there exists a generated model that performs well. In addition, since the efficacy of the learner shows up in the bound, it is necessary to have a learning procedure LEARN(D) that is capable of achieving low training error on the single-step loss. Furthermore, the bounds we have shown are valid with an infinite number of completely new trajectories at each step n, a similar requirement for related methods such as SEARN (Daumé III et al., 2009), Forward training (Ross and Bagnell, 2010), and DAgger (Ross et al., 2011a). Following the practice of the aforementioned methods, due to limited data and for data efficiency, the iterations share a single batch of training trajectories.

There are a large class of learning algorithms that meet the no-regret requirement for the internal learning procedure in DAD. A no-regret learning algorithm is one that produces a sequence of models f_n, such that the average regret

$$R_{\text{LEARNER}} = \frac{1}{N}\sum_n \ell_{f_n}(x_n) - \min_{f \in \mathcal{F}} \frac{1}{N}\sum_n \ell_f(x_n) \tag{3.8}$$

tends to 0 as $N \to \infty$. Since DAD is a *Follow-The-Leader* algorithm, any strongly convex loss (e.g. regularized squared loss) on the aggregated dataset will result in a no-regret learner. In fact, stability and asymptotic consistency are enough to ensure no-regret (Ross and Bagnell, 2011). Additionally, we can even utilize other online learning methods to achieve the same performance bounds since many are also no-regret (Cesa-Bianchi et al., 2004). The advantage of online methods is that we no longer have to store the ever-growing dataset but simply update the learned model for every new data point.

Finally, we would like to note a few additional details on using the DATA AS DEMONSTRATOR algorithm. Even though Algo. 1 starts the predictions at $\xi_k(0)$, better utilization of the dataset can be achieved by starting at other points in the training trajectories. Also, the aggregation phase may require a thresholding or filtering step. For longer horizon problems and with weaker learners, it may be difficult to achieve improvement by giving equal importance to the larger corrections at the end of the prediction horizon and to the small

errors made at the beginning. Intuitively, we hope that by the end, the learned model should not wander far away from the ground truth trajectory, and thus we ignore those points during the training iterations.

3.3 Experimental Evaluation

To demonstrate the efficacy of the DATA AS DEMONSTRATOR algorithm, we consider two markedly different, challenging time series prediction problems. Since our approach is a meta-algorithm, it requires an underlying no-regret learning procedure to optimize the single-step prediction error. Below, we use a simple but expressive learning procedure, but we wish to emphasize that it could be replaced with other domain specific learning methods.

3.3.1 Underlying single-step learning procedures

We show results using both a differentiable and non-differentiable underlying learning procedure for optimizing the single-step loss. The first, kernel regression, is a differentiable learning algorithm that allows us to embed our data points into a high, possibly infinite, dimensional space. Let $\phi(x)$ define the feature map that induces the kernel $k(x, y) = \langle \phi(\cdot), \phi(\cdot) \rangle$. We minimize the λ-regularized squared loss:

$$\min_A \frac{1}{N} \cdot \frac{1}{2} \left\| A\Phi_t - X_{t+1} \right\|_F^2 + \lambda \frac{1}{2} \left\| A \right\|_F^2 \tag{3.9}$$

for the forward prediction model:

$$\hat{x}_{t+1} = A\phi(\hat{x}_t) \tag{3.10}$$

However, for some choice of kernels, such as the Gaussian kernel, the solution to (Eq. (3.9)) cannot be evaluated as ϕ embeds the original data points into an infinite dimensional space. A standard approach would be to solve the objective in the dual. Since DAD is a dataset aggregation algorithm that augments the dataset with $O(N)$ new input-target pairs every iteration, this procedure quickly becomes intractable.

Recent work in kernel machines enables us to instead approximate ϕ through generating random Fourier features (Rahimi and Recht, 2007; Lázaro-Gredilla, 2010). For a shift-invariant kernel function k, the feature map ϕ can be approximated by constructing a mapping

$$\hat{\phi}_i(x) = \begin{bmatrix} \cos(\omega_i^T x) \\ \sin(\omega_i^T x) \end{bmatrix}, \ \omega_i \in \mathbb{R}^d \sim \mathcal{F}(k(\cdot, \cdot)) \tag{3.11}$$

where $\mathcal{F}(k(\cdot, \cdot))$ is the Fourier transform of the kernel function. This approximates the kernal function in expectation, i.e. $\mathbb{E}[\langle \phi_i, \phi_i \rangle] = k(\cdot, \cdot)$. By sampling m such ω_i and concatenating to create feature vector $\hat{\phi}(x)$, we can get a lower variance approximation of the kernel function. Using $\hat{\phi}$ in (Eq. (3.9)), allows

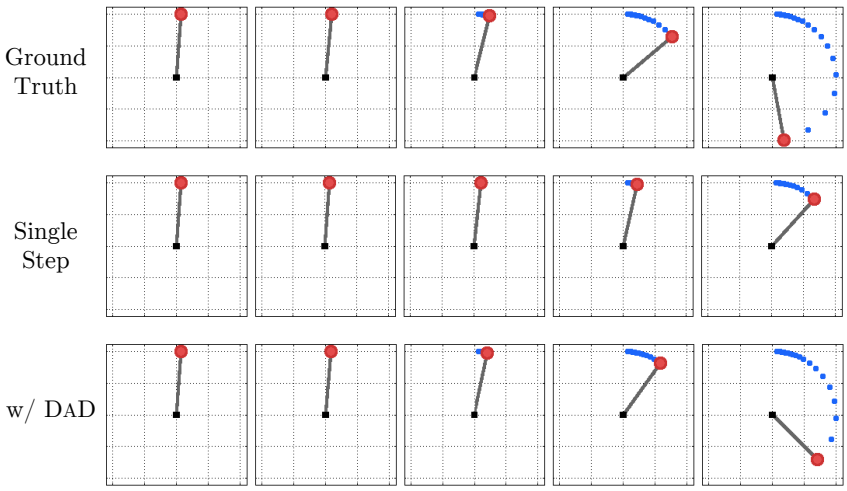


Figure 3.2: A pendulum trajectory's ground truth (top) compared with multi-step predictions displayed for time steps 1, 8, 16, 23, and 30 from the traditional minimization of single-step error (middle) and from our approach (bottom). The final output of DAD is closer to the true final state.

us to retrieve the model parameter C. In the experiments described below, we choose $m = 500$ and use the Gaussian kernel. The hyperparameters λ and kernel bandwidth σ were chosen using cross-validated grid search.

The second learning algorithm we use is a random forest regressor (Breiman, 2001). Random forests are non-differentiable learners that train a sequence of decision trees either through bagging or boosting. As they are very powerful learners, they can tend to overfit through iterations of DAD. If we let the learning rate $\eta = 1/(1+n)^{0.5}$, where n is iteration of DAD, then we regularize the random forest by setting the minimum samples per leaf as a scaled quantity on the relative size of the aggregated dataset $\max(\{1, \eta\frac{|D|}{|D_0|}\})$. We may equivalently regularize by setting the minimum weight per left to η to achieve a similar effect. We further regularize the trees by setting a max tree construction depth as well as the maximum number of trees.

3.3.2 Performance Evaluation

For each test bench, we evaluate the performance of the baseline single-step optimized predictor as well as DAD on the task of multiple-step prediction. The multiple-step prediction error is measured as the RMS error e_k between

Relative Improvement with DaD over Single-Step

Benchmark	Learner	
	Random Fourier Features	Random Forest
Flag	16.7%	66.0%
Fireplace	12.3%	6.23%
Beach	20.0%	75.8%
Pumpjack	17.0%	–
Windfarm	2.57%	–
Pendulum	44.7%	–
Cartpole	57.1%	34.7%
Helicopter	42.8%	0.33%

Table 3.1: DATA AS DEMONSTRATOR (DAD) significantly improves the performance of the traditional Single-Step method on many benchmarks. The average multi-step errors are shown in Fig. 3.8.

the prediction $\hat{\xi}_k$ and the ground truth trajectory ξ_k, computed as

$$e_k = \sqrt{\frac{1}{T_k}\sum_{t=1}^{T_k}\left\|\hat{\xi}_k(t) - \xi_k(t)\right\|_2^2}.$$

For each experiment, a random 10% of the trajectories were used as hold out data for performance reporting. In Fig. 3.8, we report the the multi-step error as the mean normalized RMSE by normalizing against the signal power for each trajectory, $p_k = \sqrt{\frac{1}{T_k}\sum_{t=1}^{T_k}\|\xi_k(t)\|_2^2}$. This normalization allows us to better ascertain the magnitude of error compared to the original trajectory. A comparison to the baseline single-step optimized method is shown in Table 3.1.

3.3.3 Dynamical Systems

We examine the performance of the proposed method on simulation test benches of physical dynamical systems of varying modeling difficulty. The pendulum and cart pole datasets are constructed from their respective uncontrolled dynamics and consist of 1000 trajectories of 30 time steps each. Both of these systems exhibit interesting limit-cycle behaviors. The final dynamical system we test on is the simulated helicopter from (Abbeel and Ng, 2005b). This dynamical system operates in a larger, 21-dimensional state space and is governed by more complicated dynamics equations. In addition, the helicopter simulation is controlled by a closed-loop LQR controller trying to hover the helicopter around the origin from randomly chosen starting points in the basin of attraction. The

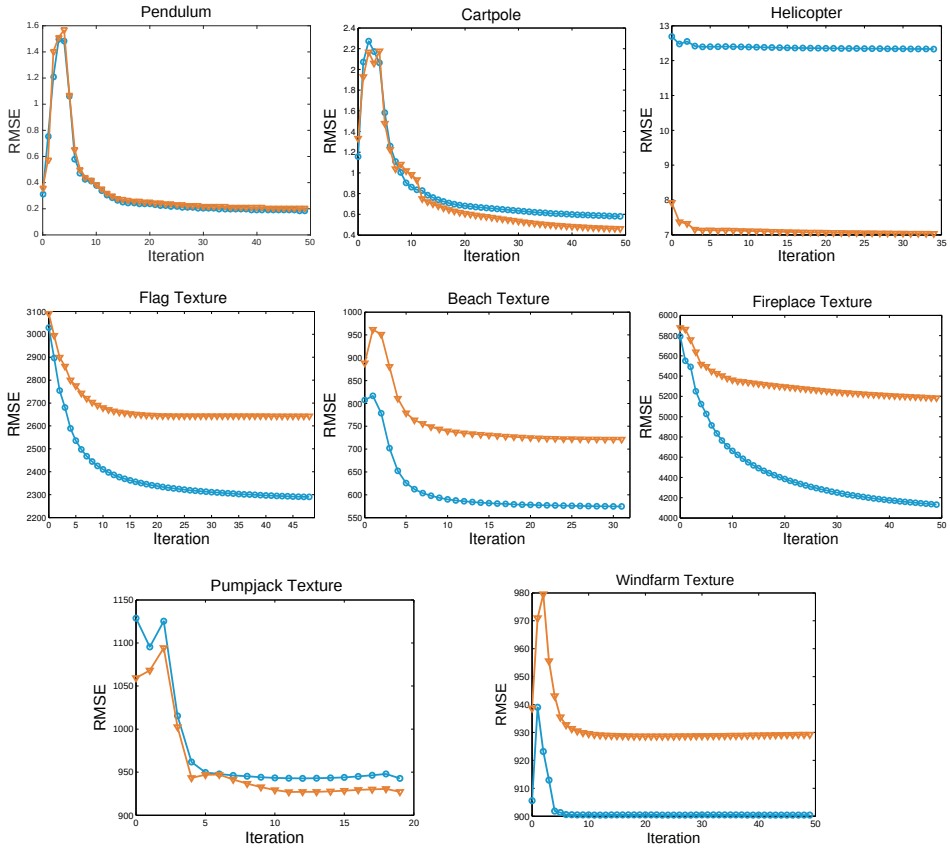


Figure 3.3: RMSE of multiple-step predictions using RFF learner for train (blue, circles) and test (orange, triangles) trajectories. Note that the shown test RMSE was not used by DAD during training and was computed for visualization only.

LQR controller chooses actions based on state and thus it poses a challenge for the learner to extract this implicit relationship from data of the system.

Qualitatively, we can see the effect of improving a single-step prediction model in Fig. 3.2 for the pendulum system. For all three of the examples, including the difficult controlled helicopter simulation, DAD is able to achieve over 40% relative improvement over the traditional baseline approach.

3.3.4 Video Textures

Dynamic video textures are image sequences generated from some underlying structure such as a flag waving, waves lapping, or a fire burning (Siddiqi et al., 2007; Chan and Vasconcelos, 2007; Basharat and Shah, 2009). Video textures are inherently complicated to simulate due to being visual observations of complex underlying dynamics. In order to test DAD, we require many training

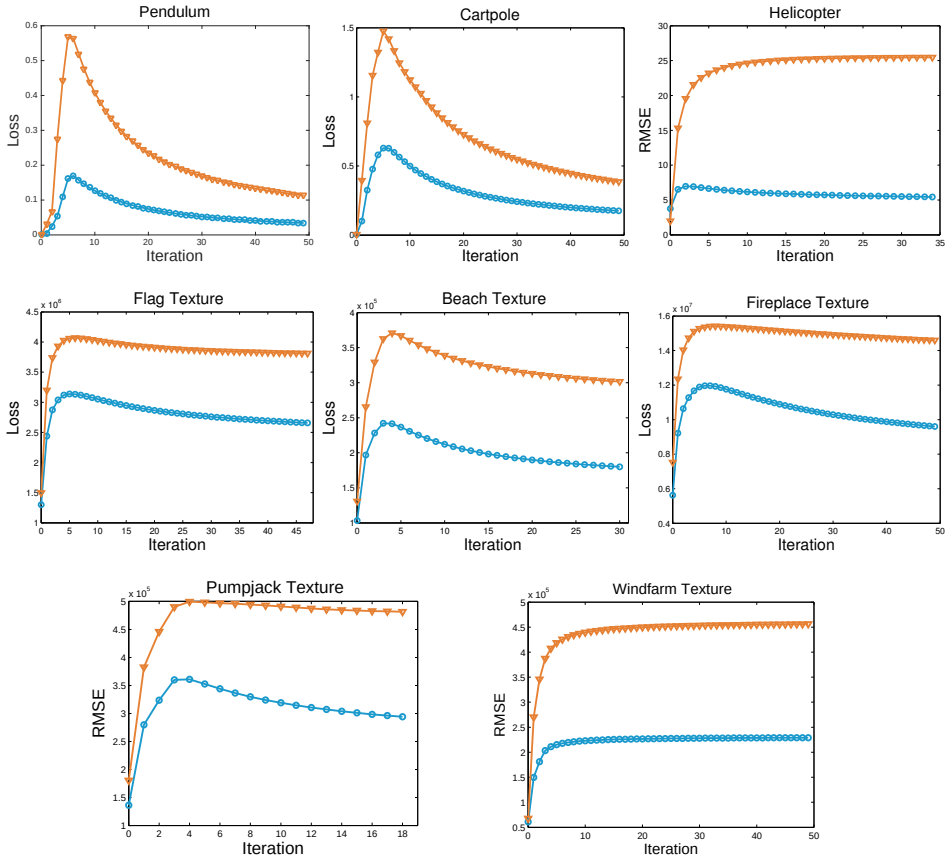


Figure 3.4: Single-step loss using RFF learner on the aggregated training (blue, circles) and test (orange, triangles) data set. Notice that the error increases and then stabilizes over iterations as distribution of states induced by learner's multiple-step prediction stabilizes. Often, the single-step error is worse than Note that the aggregated test set is never used during training and is shown for comparative purposes only.

trajectories, which is not present in standard video texture datasets. We instead use online videos that range from minutes to a half-hour at 24-30 fps. To make the learning faster and tractable, we use PCA to lower the dimensionality, sample every fifth frame, and construct trajectories with 16-26 time steps.

For many of the video texture test benches, the baseline of single-shot, single-step regression loss optimization was significantly improved upon by DAD. This is especially true for the harder and less repetitive video textures such as 'Flag' (Fig. 3.5) and 'Beach' (Fig. 3.6). These video textures are influenced by complex real-world dynamics. The simpler ones, such as 'Pumpjack' and 'Windfarm' are more cyclic, making the test and train trajectories similar. We see minor

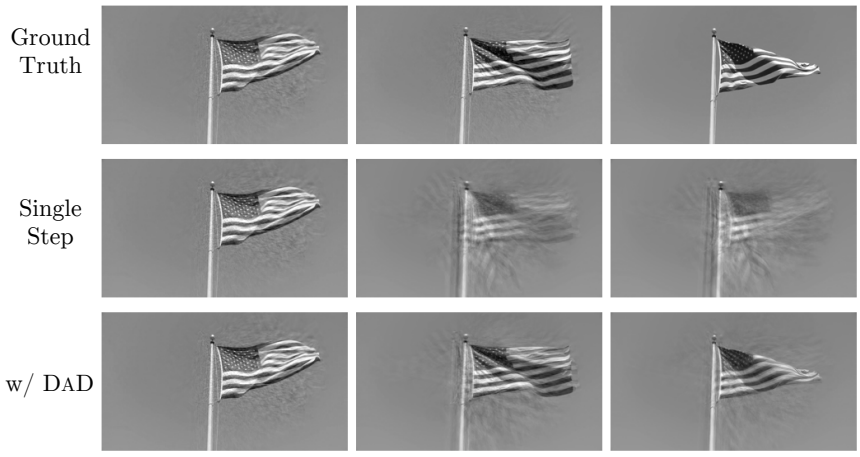
absolute improvement in both, but meaningful relative improvement for the pumpjack. This small change in absolute improvement implies that the single-step learner was able to capture the primary evolution, but even with DAD, it is unable to capture the remaining detail in the system (e.g. clouds in the background of the 'Windfarm').

A common failing of dynamical system modeling methods is the degeneration of the forward simulation to the mean value in the dataset. In the top frames of Fig. 3.7(a), we see the system converge by halfway through the prediction horizon to the mean. DAD provides corrections that can mitigate this tendency (bottom frames in Fig. 3.7(a)). This results in a qualitatively crisper and more believable video, as shown in the close up (Fig. 3.7(b)).

Finally, it is interesting to observe the multi-step error and the single-step prediction loss over iterations of DAD. As mentioned previously, the multi-step error does not necessarily decrease with each iteration (e.g. second from left in 3.3). Additionally in Fig. 3.4, we notice that the single-step regression loss on the aggregate data set initially increases as the learning problem is made harder through the addition of more data. As the multi-step prediction distribution converges, the single-step loss also stabilizes.

3.4 Conclusion

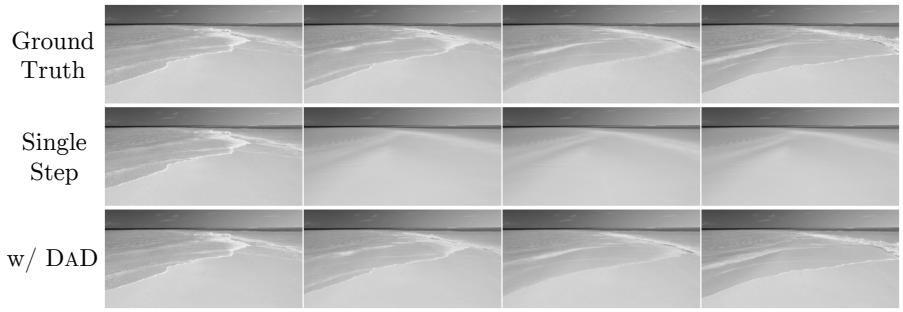
Towards Challenge 1, we presented DAD, a data-efficient, simple to implement meta-algorithm for improving the multiple-step prediction capability of a learner for modeling time-series data. Through a reduction to imitation learning, we establish strong theoretical performance guarantees. On a challenging set of experiments, we show significant performance gains over the traditional, baseline approach of minimizing the single-step prediction error. Though not included in this document, we refer to reader to preliminary results with subspace identification on a slotcar dataset and cartpole (Venkatraman et al., 2014). A very similar algorithm called *Scheduled sampling* was published after DAD for recurrent neural networks (RNNs) (Bengio et al., 2015). This work provides further experimental validation for the use of generating synthetic training examples from ground truth trajectories for the purpose of improving the multi-step predictive performance. In the next section, we investigate controlled systems and the use of DAD in the context of model-based reinforcement learning.



(a) Predicted Flag trajectory using RFF learner

(b) Predicted Flag trajectory using Random Forest learner

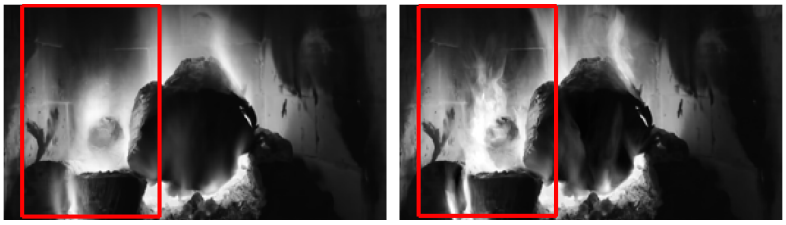
Figure 3.5: Predicted trajectories of a Flag Video texture with the RFF and Random Forest Learner. Note these are different trajectories.



(a) Predicted Beach trajectory using Random Forest learner

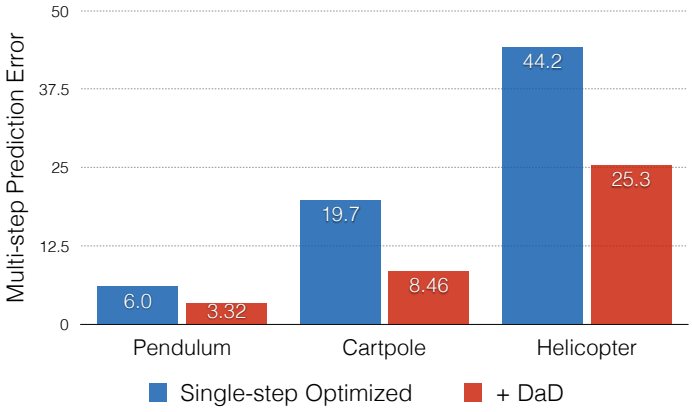
Figure 3.6: Predicted trajectory of a Beach Video texture.

(a) Prediction of frames 14, 19, 26 for the baseline method (top) and DAD (bottom). Note the averaging effect with the single-step predictor with predictions converging on the mean from the dataset. DAD yields crisper predicted images.

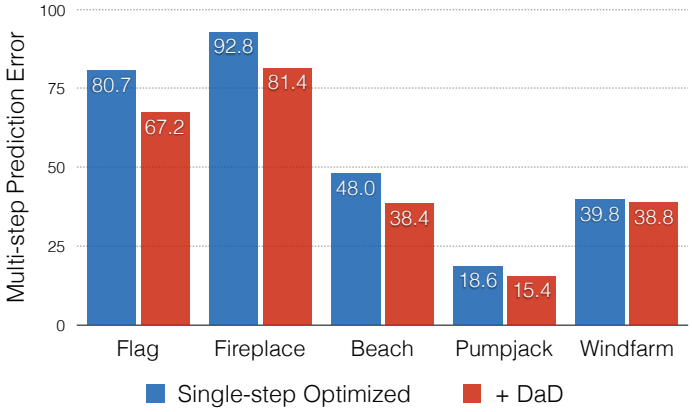


(b) Comparison of final predicted frames

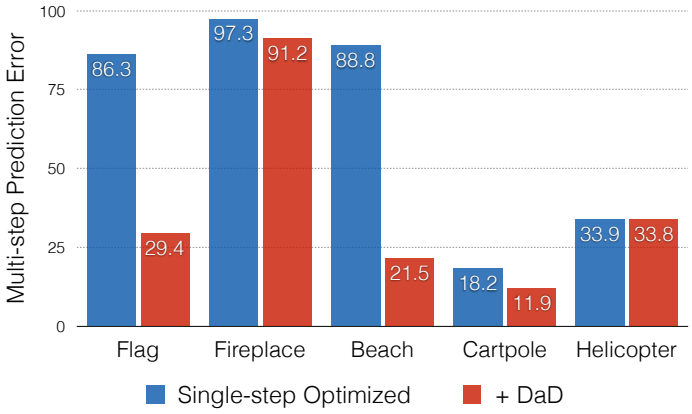
Figure 3.7: In the Fireplace video texture, our method produces a more believable evolution.



(a) Dynamical System Benchmarks using RFF learner



(b) Video Texture Benchmarks using RFF learner



(c) Selected Video Texture & Dynamical System Benchmarks using Random Forest learner

Figure 3.8: DATA AS DEMONSTRATOR (DaD) multi-step predictive performance with both a differentiable (RFF) and non-differentiable (Random Forest) learner.

Chapter 4

Model-Based Reinforcement Learning with DaD

Section 2.3 introduced control and model-based reinforcement learning (MBRL) as problems where dynamical system modeling plays a vital role. In this chapter, we extend the Data as Demonstrator (DaD) (introduced in Chapter 3) as a training procedure for control problems. In addressing Challenge 1, this improves the data efficiency of MBRL by reusing collected data for better training a dynamics model without running more the trials on the real system. Our experimental results indicate that using DaD can enable us to achieve good control performance with less data.

The goal of MBRL is to learn a dynamics model which is then used for the optimization of a control policy that minimizes future costs. Generating low cost control policies necessitates accurate dynamics models that can capture the evolution of the controlled system. However, with the increasing complexity of systems and robotic technologies, it becomes difficult to robustly characterize dynamics *a priori* with simple analytic models. Machine learning provides an avenue to to tackle this problem and to scale model-based control techniques to new systems. Prior work in using machine learning methods scale the gamut from adapting physics-based parameterizations (Abbeel et al., 2005b), augmenting physics models (Ko et al., 2007), or through non-parametric, black-box learning (Bagnell and Hneider, 2001). Typically, the accuracy of data-driven dynamics models depends on the amount of collected data. However, for many real-world robotic systems, it can be labor intensive and expensive to acquire large data-sets for training models. Hence, it is often desirable to improve model fidelity by observing fewer example trajectories on the physical system.

This work was originally presented in *Improved Learning of Dynamics for Control* at ISER 2016 (Venkatraman et al., 2016a) and contains joint work done in collaboration with Roberto Capobianco, Lerrel Pinto, and Daniele Nardi.

4.1 Preliminaries

We consider systems that operate as a Markov Decision Process (MDP). The MDP is defined by states x_t that follow an *unknown* state transition (dynamics) function $f(x_t, u_t) \to x_{t+1}$, where u_t are controls (actions). We additionally assume a known cost function $c : x_t, u_t \to \mathbb{R}$. Solving this MDP consists of minimizing the (expected) cumulative cost over a time horizon T, which may be infinite, by finding a control policy $\pi(x_t)$:

$$\pi = \arg\min_{\pi} \sum_{t=0}^{T-1} c(x_t, u_t) \quad \text{s.t. } u_t = \pi(x_t) \text{ and } x_{t+1} = f(x_t, u_t). \tag{4.1}$$

Model-based reinforcement learning (MBRL) attempts to solve the above in situations where the underlying dynamics and sometimes the cost function are unknown, adding the burden of deriving estimators of both. In this work, we assume knowledge of the cost function and focus solely on system identification in which we fit a function approximator $\hat{f}$ to be used as the dynamics constraint for the policy optimization in Eq. (4.1). The learning problem for the dynamics model is then formulated as minimizing the predictive error, similar to Eq. (3.4) but with a control input added,

$$\hat{f} = \arg\min \sum_{t=1}^{T-1} \|x_t - \hat{f}(x_{t-1}, u_{t-1})\|_2^2 \tag{4.2}$$

from a data-set of trajectories $\{(x_0, u_0) \ldots, (x_{T-1}, u_{T-1})\}$ of state-action pairs collected from the system. As shown in Chapter 3, the downside of only doing this optimization Eq. (4.2) is that errors can compound (e.g. Theorem 3.1.1). We thus propose to adapt DAD for for system identification in the controlled setting.

4.1.1 System Identification for Control

Simply collecting system trajectories, learning the dynamics, and optimizing the control policy typically results in inaccurate or unstable dynamics models and poorly performing control policies. An iterative process to achieve better performance was formalized in the MBRL literature (Abbeel et al., 2006; Deisenroth and Rasmussen, 2011), generating a procedure similar to the one outlined in Fig. 4.1. By alternating between fitting the dynamics model and collecting new data under the distribution induced by the policy, the model becomes better at capturing the dynamics over the important regions of the state-space while the control policy derived from the dynamics is either improved over that region or erroneously exploits inaccuracies in the dynamics model. Thus in each iteration, a good policy is found or data is collected from the controller's mistakes for improvement at the next iteration.

We specifically refer to the loop in Fig. 4.1 as the DAgger (Data-set Aggregation) system identification learning framework (Ross and Bagnell, 2012). A key

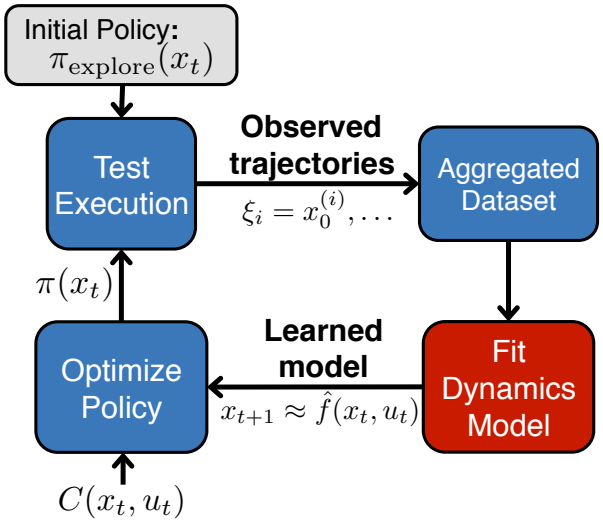


Figure 4.1: DAgger System Identification for Model Learning and Control. We show that using DAD (Chapter 3) for the dynamics model fitting (red) can improve control performance.

difference lies in the aggregation step of the procedure in order to provide model agnostic guarantees. At the beginning of the algorithm, DAgger initializes an empty training data-set and an exploration policy $\pi_{\text{explore}}(x_t)$ that generates an action (control) u_t given a state x_t. This initial policy can either consist of random controls (referred to as a random policy) or be an expert demonstration. Then, DAgger iteratively proceeds by:

1. Executing the latest policy to collect a set of new trajectories $\{\xi_i\}_{i=0}^{k-1}$ where $\xi_i = \{(x_t, u_t)...\}_i$ is a time series of state-action pairs
2. Aggregating the trajectories $\{\xi_i\}_{i=0}^{k-1}$ into the training data-set
3. Learning from the data-set a forward dynamics model $\hat{f}(x_t, u_t) \rightarrow x_{t+1}$
4. Optimizing a new control policy π that minimizes a given cost function $c(x_t, u_t)$ over the time horizon T of the control problem
5. Tracking the best policy from all those generated.

During the execution of the first DAgger loop, the state distribution induced by π can greatly differ from the initial π_{explore}; the first generated policies may perform poorly due to inaccuracies in $\hat{f}$. The iterative procedure refines the dynamics model by aggregating data from states induced by running the system with $\pi_1, \ldots, \pi_N$. In particular, Ross et al. (Ross and Bagnell, 2012) provide theoretical guarantees for this algorithm, as long as we also sample states from the exploration distribution when aggregating data. This can be simply obtained by

aggregating additionally a constant percentage of trajectories obtained from the exploration distribution. For example, this can be obtained by sampling from the original dataset or running the system with π_{explore}. This helps prevent the learner from focusing only on the induced distributions from the policies. Additionally, note that although in theory we need, at each iteration, an optimal control policy under $\hat{f}$ (Ross and Bagnell, 2012), it may be possible to use a suboptimal policy (Talvitie, 2015). Finally, the algorithm does not guarantee that the policy gets monotonically better with every iteration. Thus, we must track the performance of the executed policies and return the best one obtained so far.

4.2 DaD+Control

As discussed at length in Chapter 3, solely Eq. (4.2) can yield poor prediction capability into the future. This can be further problematic in planning or control where the control optimizer can exploit inaccuracies in the simulator (model) to result in unusual behavior that is infeasible on the real system[1]. We adapt DAD to the control setting as shown in Fig. 4.2 and Algorithm 2. In our scenario, as we try to minimize the number of times that we run the real system, we sub-sampled trajectories to be shorter than the control problem's time horizon. We believe this had added benefit as there is a tradeoff between the single-step (Fig. 3.4) and multi-step error (Fig. 3.3). This is more of an issue for DAD+CONTROL[2]. as the control synthesis process chooses actions based on every prediction. Making a poor choice of action early on can be disastrous on a highly unstable system such as a helicopter.

4.3 Experimental Evaluation

We evaluate our algorithm ('DAgger +DAD' (+CONTROL)) both on simulated dynamical systems[3] and real robotic platforms. In particular, we consider two simulated scenarios: the classic cartpole swing-up problem and the challenging helicopter hovering problem. Additionally, we show the applicability of our approach on real systems such as the Videre Erratic mobile base and the Baxter robot. In each described experiment, we learn dynamical models of the form:

$$\Delta_t \leftarrow f(x_t, u_t), \quad \text{where } \Delta_t = x_{t+1} - x_t. \tag{4.3}$$

This parametrization is similar to (Deisenroth and Rasmussen, 2011), where the previous state is used as the mean prior for predicting the next state. Due to

[1]The DARPA Virtual Robotics Challenge is a good example of human "control optimizers" exploiting a simulation model, `https://www.youtube.com/watch?v=0dC8FVuo9dc`.

[2]The code implementing DAD+CONTROL is available at `https://github.com/LAIRLAB/DaD`

[3]Simulators, except the helicopter, available at `https://github.com/LAIRLAB/control_simulators` with C++ and Python APIs

Algorithm 2 DAD+CONTROL

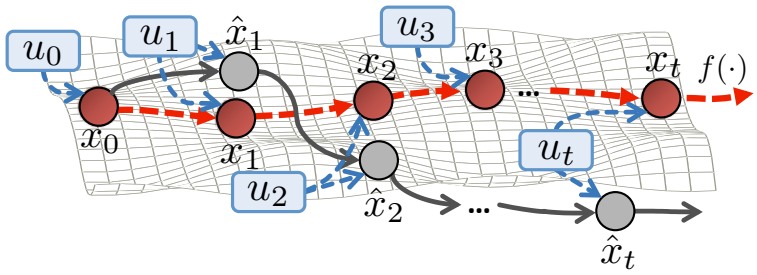
Input:
▷ Number of iterations N, set $\{\xi_k\}$ of K trajectories of time lengths $\{T_k\}$.
▷ No-regret learning procedure LEARN
Output: Model $\widehat{f}$

Initialize data-set $D \leftarrow \{([x_t, u_t], x_{t+1})\}$ of $(T_k - 1)$ input-target pairs from each trajectory ξ_k
Train initial model $f_0 \leftarrow \text{LEARNER}(D)$
for $n = 1, \ldots, N$ **do**
  for $k = 1, \ldots, K$ **do**
    Extract $x_0 \leftarrow \xi_k(0)$, $\{u_t\}_{t=0}^{T_k} \leftarrow \xi_k$
    $(\widehat{x}_1, \ldots, \widehat{x}_T) \leftarrow \text{ROLLOUT}(\hat{f}_n, x_0, \{u_t\}_{t=0}^{T_k})$
    $D' \leftarrow \{([\widehat{x}_1, u_1], x_2), \ldots, ([\widehat{x}_{T_k-1}, u_{T_k-1}], x_{T_k})\}$ where $x_t \leftarrow \xi_k(t)$
    $D \leftarrow D \cup D'$
  end for
  $f_n \leftarrow \text{LEARN}(D)$
end for
return $\widehat{f} \leftarrow f_n$ with lowest error on validation trajectories

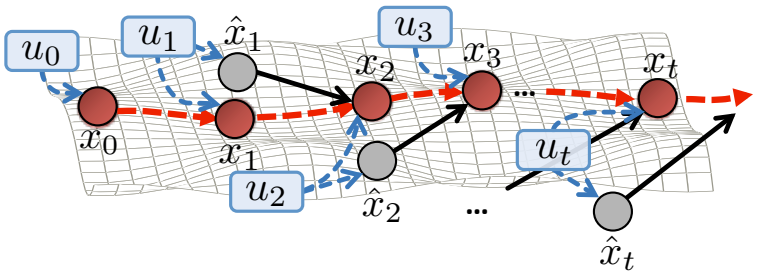
the difficulty of optimizing Eq. (4.1) under arbitrary dynamics and cost models, for simplicity, we focus on minimizing a sum-of-quadratics cost-to-go function:

$$\sum_t c(x_t, u_t) = \sum_t x_t^T Q_t x_t + u_t^T R_t u_t. \tag{4.4}$$

By using this form of cost function, along with a linearization of the learned dynamics model, we can formulate the policy synthesis problem as that of a Linear Quadratic Regulator, which allows the policy to be computed in closed-form. In each experiment, we compare 'DAD +DAgger' to 'DAgger Only'. For Cartpole, Erratic, and Baxter the data-set was initialized with a random exploratory policy, while the helicopter problem received both a random and an expert policy (generated form LQR on the true dynamics) roll-out for initialization. The simulated cartpole and helicopter experiments got additional exploratory roll-outs on every iteration of DAgger with the random and expert policies respectively. For the Baxter robot experiment, instead, we achieved exploration through an ϵ-random controller that added random perturbation to the commanded control with ϵ probability. For each method, we report the average cumulative cost at each iteration of DAgger as averaged over ran trials. Three trials were run on the Erratic while five were ran for other benchmarks. The charts that illustrate the obtained results are all normalized to the highest observed cost, since the cost functions are tuned to promote the desired control behavior rather than to have a physical interpretation.



(a) Forward simulation of learned model (gray) introduces error at each prediction step compared to the true time-series (red)

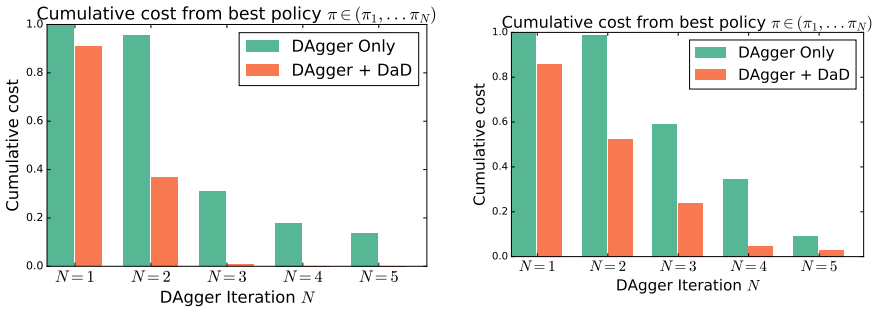


(b) Data provides a demonstration of corrections required to return back to proper prediction

Figure 4.2: Similar to setting considered in Chapter 3 (Fig. 3.1), cascading errors from model learning with controls (blue) can lead to errors in forward simulation (gray) (Fig. 4.2(a). As the control policy will not see these predicted states at test time, we do not query the policy to for new controls at these states. DAD+CONTROL reuses the sequence of controls from the trajectory in synthesizing new correction training samples (Fig. 4.2(b)).

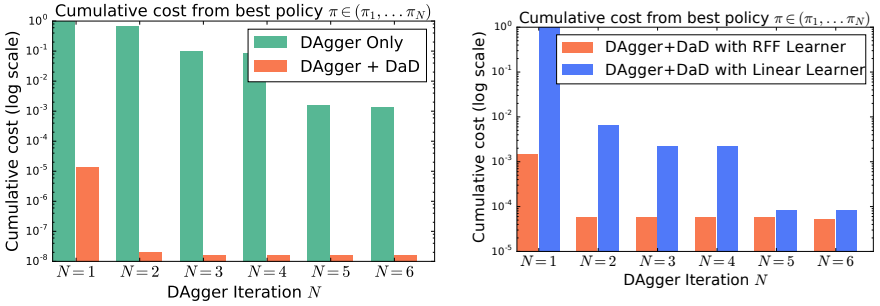
4.3.1 Simulation Experiments

Cartpole swing-up: The cartpole swing-up is a classic controls and MBRL benchmark where the goal is to swing-up a pendulum by only applying a linear force on the translatable base. We learn a linear dynamics model in the form of Eq. (4.3) using Ridge Regression (regularized linear regression). We then use an iterative Linear Quadratic Regulator (Li and Todorov, 2004) (iLQR) controller about a nominal swing-up trajectory in state-space with an initial control trajectory of zeros. The iLQR optimization procedure finds a sequence of states and controls feasible under the learned dynamics model to minimize the cost. The simulated system has system-transition noise and we compare our algorithm's performance both with and without control noise to simulate the effects of noisy actuation on a real-robot. We show results in Fig. 4.3 of the evaluated trajectory costs accumulated over the problem's time horizon.



(a) Cartpole Swing-up (b) Swing-up with Control Noise

Figure 4.3: Controlling a simulated cartpole for swing-up behavior.



(a) Helicopter Hover (b) Using Different Learning Algorithms

Figure 4.4: Controlling a simulated helicopter to hover. Note the log-scale on cost.

Helicopter simulator: Helicopter hovering is a difficult problem due to the instability of the dynamical system, especially under noise. We utilize the helicopter simulator from (Abbeel and Ng, 2005b) with additive white noise and follow a problem setup similar to (Ross and Bagnell, 2012). We make the problem more difficult by initializing the helicopter at states up to 10 meters away from the nominal hover configuration. As the dynamics are highly nonlinear, we show the advantage of using Random Fourier Features (RFF) regression (Rahimi and Recht, 2007) to learn a dynamics model in a 21-dimensional state space. We find a steady-state linear quadratic regulator (LQR) policy to map the helicopter's state to the 4-D control input. The results in Fig. 4.4 show that DAD dramatically improves performance over only DAgger.

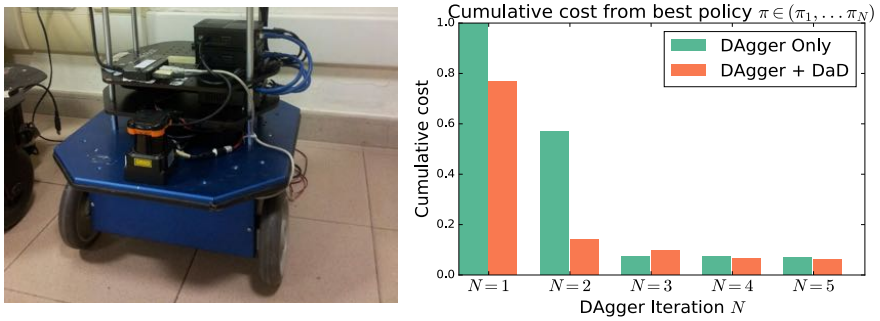


Figure 4.5: Results for controlling a Videre Erratic differential-drive mobile robot.

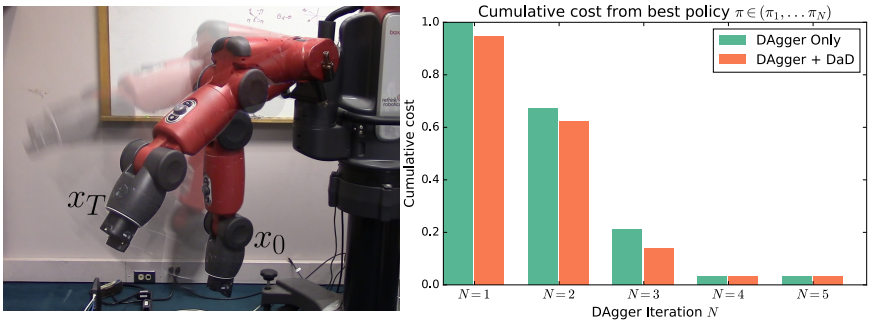


Figure 4.6: Results on controlling a Baxter robot. We learn a dynamics model and compute a control policy to move the robot manipulator from state x_0 to x_T.

4.3.2 Real-Robot Experiments

Videre Erratic: In this experiment, we control the velocity of a Videre Erratic mobile base. The goal is to drive the robot to a given position specified in the robot's reference frame. The 3-D state vector includes the robot position and orientation while the 2-D control vector is the robot velocity. The dynamics model is learned using Ridge Regression. Unlike the other experiments, we use a trajectory-control policy that finds a sequence of controls $u_1, \ldots, u_T$ to apply open-loop at run time on the robot. We compute the control sequence by simulating the learned dynamics model $\hat{f}$ with a simple proportional controller. Results are shown in Fig. 4.5.

Baxter robot: We use the 'DAgger +DAD' approach to control a 7-degree-of-freedom manipulator to a target joint configuration. We command the robot arm in torque control mode with *suppression* of the inbuilt gravity compen-

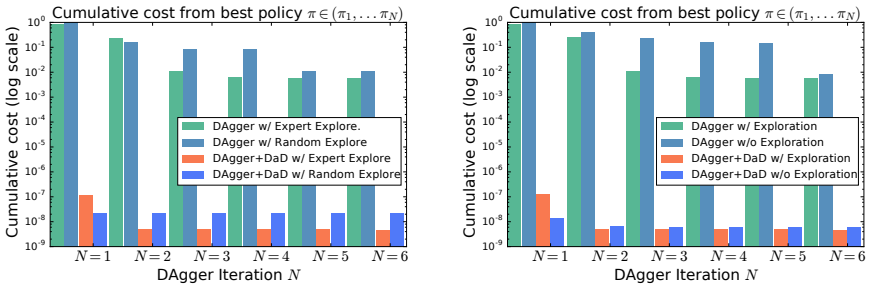
sation. The 14-dimensional state vector consists of the joint angles and their velocities. We learn the dynamics model using Ridge Regression and compute a steady-state LQR control policy, obtaining the results in Fig. 4.6.

4.4 Discussion

In our simulation experiments we compared the performance obtained by applying 'DAgger +DaD' on a cartpole with and without control noise. Results show that the improvement of our method over 'DAgger Only' decreases in presence of actuation noise. This can be explained by the fact that, over the same generated nominal controls, the state trajectories obtained during each roll-out are slightly different and represent a limitation on the efficacy of the learner over the same number of iterations – i.e. there is a higher baseline error in the dynamics model.

In the case of the helicopter, we additionally compared the results obtained by using two different learning algorithms and by applying different exploration policies. For the former, we compared the non-linear RFF (Rahimi and Recht, 2007) regression against linear regression. As shown in Fig. 4.4(b), the nonlinear learner performs much better as it better captures the heavy nonlinearity of the helicopter dynamics. The DAgger method (Ross and Bagnell, 2012) requires drawing state-transition samples at every iteration from some exploration distribution. In Fig. 4.7(a), we compare using an expert exploration policy (LQR controller using the true dynamics) versus a random-control exploration policy. With DAgger +DaD, the learned dynamics and policy yield a stable behavior for both types of exploration, with some improvement using the expert policy. The DAgger Only baseline often is unable to learn a stable policy using the random exploration policy. We believe that DAgger +DaD learns a more stable multi-step predictive dynamics model – an important aspect for the Bellman backup during policy optimization. An interesting observation is that DAgger +DaD without the exploration policy does not lead to a significant performance difference (Fig. 4.7(b)) compared to the 'DAgger Only' baselines. This comparison shows the difference between (Abbeel et al., 2005b) (no exploration) and (Ross and Bagnell, 2012) (constant fraction exploration). Note that to keep the amount of data constant in the trials without the exploration trajectories, the learners were given the difference as test trials under the current optimized policy.

The real-robot evaluations show the applicability of our method on real systems and complex platforms. In particular, the Erratic experiments show that by using DaD, we are indeed able to get a better dynamics model for forward-prediction. This model can be used for trajectory generation and optimization as described in Section 4.3.2, where the sequence of obtained controls has been directly applied to the Erratic in an open-loop as a control trajectory. While the application of 'DAgger +DaD' on the Baxter robot results in a limited performance improvement, this confirms our hypothesis that, in robotic platforms characterized by high actuation noise (e.g. Baxter's chain of noisy actuators),



(a) Random vs. Expert Explore Policy

(b) Explore vs. No Exploration Policy

Figure 4.7: Comparison of Exploration policies. Cost values are not normalized across plots.

only smaller improvements over 'DAgger Only' can be achieved (consistent with the simulated noisy-actuation result in Fig. 4.3(b)). Additionally, the considered problem on the Baxter is relatively simple with control authority at every joint. In these settings, DAgger seemingly can still efficiently capture the dynamics of the system with only a minor benefit from the additional DAD loop.

4.5 Conclusion

DAD+CONTROL can improve the data-efficiency of MBRL, requiring less data to be collected on the real system for the same level of performance. We showed a variety of experiences that showed both the strengths and weakness of the proposed algorithm. We see reduced performance under heavy control noise (e.g. Baxter) and significant gains for open-loop control synthesis (e.g. Erratic).

Part III

Learning on Partially Observable Systems

Chapter 5

Linear System Modeling with Predictive Representations via Online Instrumental Variable Regression

In this chapter, we start our investigation into the specific issues that arise when modeling dynamical systems which are only partially observed, as stated in Challenge 2. An example graphical model of a partially observed system is shown in Fig. 2.4. In this system, the observations are not independent but are conditionally independent given state s_t. That is, x_{t+1} is correlated with x_t. A function to predict x_{t+1} from the past observation x_t can therefore require all the correlated dependencies of x_t, i.e., all of the history $x_{t-1}, x_{t-2}, \ldots, x_0$. As the history can be aribitrarily long, this is computationally untenable when learning a time-series model.

In this chapter, we develop a technique for online learning predictive models on partially observed time series. We use a predictive representation derived by Boots and Gordon (2011b); Hefny et al. (2015b) as a state representation sufficient for predicting the future. Hefny et al. (2015b) showed that using instrumental variable regression (IVR) with this representation removes the bias from past obsevations to learn a linear dynamics model that can be used in a Kalman filter. Our contribution is an extension of instrumental variable regression to the online setting with streaming data. The chapter below spends most of its focus on developing online instrumental variable regression in general as

This work was originally presented in *Online Instrumental Variable Regression with Applications to Online Linear System Identification* at AAAI 2016 (Venkatraman et al., 2016b) and containts joint work done in collaboration with Wen Sun and Byron Boots.

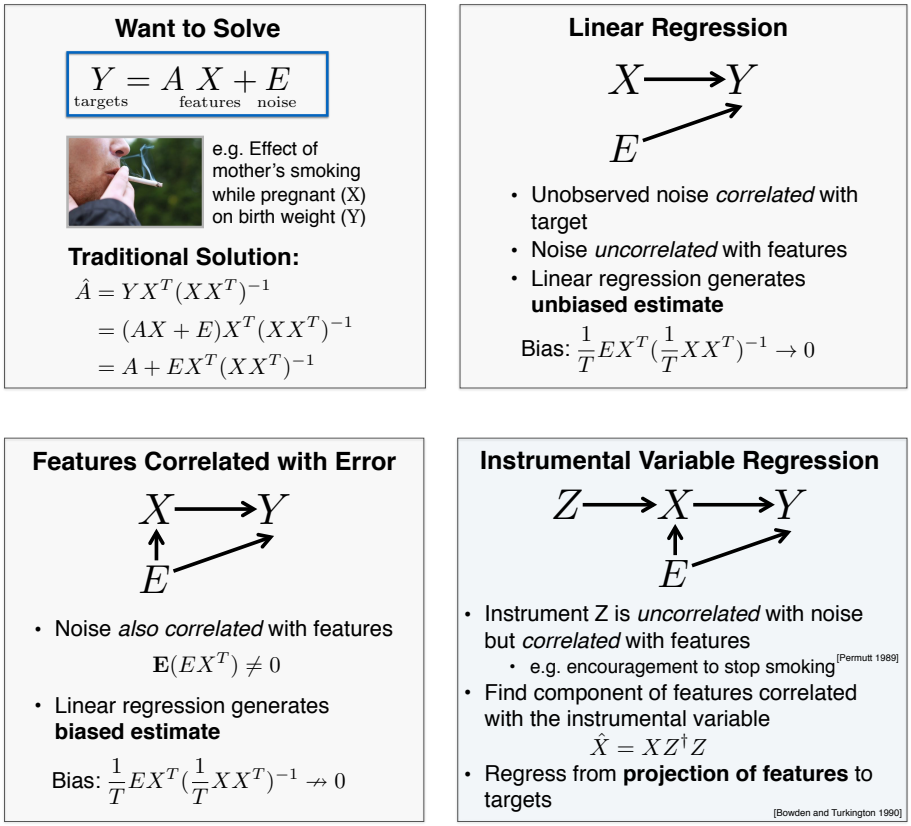


Figure 5.1: Instrumental variable regression is a classical statistical technique for linear regression when the features and targets are correlated through an unobserved variable E. The introduction of the instrument Z is used to decorrelate the effect of the unobserved variable in finding the linear predictor $\hat{A}$ from features X to targets Y.

an extension to the classical stastitics technique. From Section 5.5, the focus moves to dynamical system learning. We experimentally demonstrate the efficacy of our algorithm in combination with popular no-regret online algorithms for the task of learning predictive dynamical system models and on a prototypical econometrics instrumental variable regression problem.

5.1 Introduction

Instrumental variable regression (IVR) is a popular statistical linear regression technique to help remove bias in the prediction of targets when both the features and targets are correlated with some unknown additive noise, usually a variable omitted from the regression due to the difficulty in observing it (Bowden and

Turkington, 1990). In this setting, ordinary least squares (OLS) (i.e. linear regression) from features to targets leads to a biased estimate of the dependence between features and targets (see Fig. 5.1). For applications where the underlying unbiased dependency is required, such as in the study of causal effects for econometrics (Miguel et al., 2004), epidemiology (Greenland, 2000), or for the learning of dynamical system models (Söderström and Stoica, 2002), IVR provides a technique to remove the correlation with the unobserved variables.

We focus in this chapter on the regression application of instrumental variables where the IVR process consists of multiple linear regressions steps. Prior attention on IVR has focused on the batch learning scenario: each step of regression is performed in whole with all of the data at once. However, with the ever growing prevalence of large datasets, such an approach becomes quickly infeasible due to the scaling of the memory and computational complexity with regards to the data set size and feature dimensionality. Towards this end, we propose an online version of instrumental variable regression that replaces each of the regression steps with an online learner.

Specifically, we develop an Online Instrumental Variable Regression (OIVR) procedure that can be regarded as a reduction to no-regret online learning. Under the assumption that the set of regression and instrumental variables are i.i.d, we derive a strong no-regret bound with respect to the desired objective optimized by the batch setting (batch IVR). Our theorem allows us to take advantage of *any* no-regret online learning procedure for the multiple regression steps in IVR. We explicitly show that OIVR allows us to introduce a new family of online system identification algorithms that can exploit no-regret online learning. This reduction extends on the initial reduction given by Hefny et al. (2015b) from batch predictive state dynamical system learning to batch IVR. Finally, we investigate the experimental performance of several popular online algorithms such as Online Gradient Descent (OGD) (Zinkevich, 2003), Online Newton Step (Hazan et al., 2007) (ONS), Implicit Online Gradient Descent (iOGD) (Kulis et al., 2010), and Follow The Regularized Leader (FTRL) (Shalev-Shwartz, 2011) in the context of OIVR for both dynamical system modeling and on a simple but illustrative econometrics example.

5.2 Instrumental Variable Regression

Consider the standard linear regression scenario where we wish to find A given design matrices (datasets) $X = \begin{bmatrix} x_1 & x_2 & \dots \end{bmatrix}$ and $Y = \begin{bmatrix} y_1 & y_2 & \dots \end{bmatrix}$ representing our explanatory variables (features) $x_i \in \mathbb{R}^{n \times 1}$ and outputs (targets) $y_i \in \mathbb{R}^{m \times 1}$. This relationship is modeled by:

$$Y = AX + E \tag{5.1}$$

where $E = \begin{bmatrix} \varepsilon_1 & \varepsilon_2 & \dots \end{bmatrix}$ are independent noise (error). Solving this via least-

squares minimization, gives us:

$$\begin{aligned}\hat{A} &= YX^T(XX^T)^{-1} = (AX+E)X^T(XX^T)^{-1} \\ &= AXX^T(XX^T)^{-1} + EX^T(XX^T)^{-1} \\ &= A + EX^T(XX^T)^{-1} \end{aligned} \tag{5.2}$$

When the number of samples T goes to infinity, by law of large number, we will have: $EX^T/T \to \mathbf{E}(\varepsilon x^T)$ and $XX^T/T \to \mathbf{E}(xx^T)$ in probability. Normally, we assume that ε and x are uncorrelated, which means EX^T/T converges to zero in probability, ($\mathbf{E}(\varepsilon x^T) = 0$), which yields an unbiased estimate of A from Eq. 5.2.

$$\hat{A} = A + \frac{1}{T}EX^T\left(\frac{1}{T}XX^T\right)^{-1} \to A$$

However, if ε and x are correlated $\mathbf{E}(\varepsilon x^T) \neq 0$, we are only able to get a biased estimate of A through the least-squares optimization, since $\mathbf{E}\left[\varepsilon x^T\right]\mathbf{E}\left[xx^T\right]^{-1} \neq 0$.

On the other hand, IVR can achieve an unbiased estimate of A (Rao et al., 2008; Cameron and Trivedi, 2005). In IVR, we remove this bias by utilizing an *instrumental variable*, denoted as $Z = \begin{bmatrix} z_1 & z_2 & \dots \end{bmatrix}$ in Fig. 5.2. For a variable to be an instrumental variable, we need two conditions: (1) the instrumental variable z is correlated with x such that $\mathbf{E}(xz^T)$ is full row rank and (2) the instrumental variable is uncorrelated with ε, i.e. $\mathbf{E}(z\varepsilon^T) = 0$.

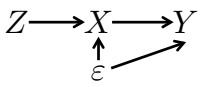


Figure 5.2: Causal diagram for IVR

Instrumental variable regression proceeds as follows. IVR first linearly regresses from Z to X to get $\hat{X} = XZ^\dagger Z$, where $Z^\dagger = Z^T(ZZ^T)^{-1}$. Then, IVR linearly regresses from the projection $\hat{X}$ to Y to get an estimate of A:

$$\begin{aligned}\hat{A}_{\text{IVR}} &= Y\hat{X}^T(\hat{X}\hat{X}^T)^{-1} \\ &= YZ^\dagger ZX^T(XZ^\dagger ZX^T)^{-1} \\ &= AXZ^\dagger ZX^T(XZ^\dagger ZX^T)^{-1} \\ &\quad + EZ^\dagger ZX^T(XZ^\dagger ZX^T)^{-1} \\ &= A + EZ^T(ZZ^T)^{-1}ZX^T(XZ^\dagger ZX^T)^{-1}\end{aligned}$$

Note that $\hat{A}_{IVR} \to A$ in probability since $EZ^T/T \to 0$ in probability under the assumption that instrumental variable z and ε are uncorrelated.

The process of instrumental variable regression can be represented through

Algorithm 3 Batch Instrumental Variable Regression

Input:
▷ Explanatory Variable Design Matrix $X \in \mathbb{R}^{d_x,n}$,
▷ Instrumental Variable Design Matrix $Z \in \mathbb{R}^{d_z,n}$,
▷ Prediction Targets Design Matrix $Y \in \mathbb{R}^{d_y,n}$
Output: $A^* \in \mathbb{R}^{d_y,d_x}$

$M^* \leftarrow \arg\min_M \|X - MZ\|_F^2$
$\widehat{X} \leftarrow M^* Z$
$A^* \leftarrow \arg\min_A \left\|Y - A\widehat{X}\right\|_F^2$

return A^*

the following two steps (also known as two-stage regression):

$$M^* \leftarrow \arg\min_M \|X - MZ\|_F^2 \tag{5.3}$$

$$A^* \leftarrow \arg\min_A \|Y - AM^*Z\|_F^2 \tag{5.4}$$

For shorthand, we will generally refer to the final regression stage, Eqn. 5.4, as the batch IVR objective .

5.3 Online Instrumental Variable Regression

The formulation of online algorithms yields a two-fold benefit – first, it allows us to use datasets that are too large to fit in memory by considering only one or a few data points at a time; second, it allows us to run our algorithm with streaming data, a vital capability in fields such as robotics where many sensors can push out volumes of data in seconds. In the following section, we formulate an online, streaming-capable adaptation of the batch Instrumental Variable Regression (IVR) algorithm. We show that our online algorithm has a strong theoretical performance guarantee with respect to the performance measure in the batch setting.

In the batch setup of IVR (Algorithm 3), we require all the datapoints *a priori* in order to find A^*. To create an online version of this algorithm, it must instead compute estimates M_t and A_t as it receives a single set of data points, x_t, z_t, y_t. To motivate our adaptation of IVR, we first consider the adaptation of OLS (i.e. linear regression) to the online setting. Given the design matrices $X = [x_0, \ldots, x_t, \ldots,]$ and $Y = [y_0, \ldots, y_t, \ldots]$, in OLS, we optimize the following batch objective over all the data points:

$$\beta^* = \arg\min_\beta \|\beta X - Y\|_F^2 = \arg\min_\beta \sum_t \ell_t(\beta) \tag{5.5}$$

where the loss function $\ell_t(\beta) = \|\beta x_t - y_t\|_2^2$ is the L2 loss for the corresponding pair of data points (x_t, y_t). To formulate an online OLS algorithm, we

may naturally try to optimize the L2 loss for an individual data point pair $\|\beta x_t - y_t\|_2^2$ at each timestep without directly considering the loss induced by other pairs. Prior work in the literature has developed algorithms that address this problem of considering losses ℓ_t and predicting a β_{t+1} while still achieving provable performance with respect to the optimization of β over the batch objective (Zinkevich, 2003; Hazan et al., 2007; Shalev-Shwartz, 2011). The performance guarantee of these algorithms is in terms of the (average) regret, which is defined as:

$$\frac{1}{T}\textsc{Regret} = \frac{1}{T}\sum_t \ell_t(\beta_t) - \frac{1}{T}\min_\beta \sum_t \ell_t(\beta) \tag{5.6}$$

We say a learning procedure is *no-regret* if $\lim_{T\to\infty} \frac{1}{T}(\textsc{Regret}) = 0 \Rightarrow \textsc{Regret} \in o(T)$.

Intuitively, the no-regret property tells us that the optimization of the the loss in this way gives us a solution that is competitive with the best result in hindsight (i.e. if we had optimized over the losses from all data points). In Ivr (Algorithm 3), lines 1 and 3 are each linear regression steps which are individually the same as Eq. (5.5). Motivated by this, we introduce Online Instrumental Variable Regression (Oivr), in which we utilize a no-regret online learner for the individual batch linear regressions in Ivr. The detailed flow of Oivr is shown in Algorithm 4 and depicted in Fig. 5.3.

Algorithm 4 Online Instrumental Variable Regression with No-Regret Learners

Input:

▷ no-regret online learning procedures $\textsc{Learn}_1$, $\textsc{Learn}_2$

▷ Streaming data sources for the explanatory variable $S_x(t) : t \to x \in \mathbb{R}^{d_x}$, the instrumental variable $S_z(t) : t \to z \in \mathbb{R}^{d_z}$, and the target variable $S_y(t) :\to y \in \mathbb{R}^{d_y}$

Output: $\bar{A}_T \in \mathbb{R}^{d_y,d_x}$

1: Initialize $M_0 \in \mathbb{R}^{d_x,d_z}$, $A_0 \in \mathbb{R}^{d_y,d_x}$
2: Initialize $\bar{M}_0 \leftarrow \mathbf{0} \in \mathbb{R}^{d_x,d_z}$, $\bar{A}_0 \leftarrow \mathbf{0} \in \mathbb{R}^{d_y,d_x}$
3: Initialize $t \leftarrow 1$
4: **while** $S_x \neq \emptyset$ and $S_z \neq \emptyset$ and $S_y \neq \emptyset$ **do**
5: $\quad (x_t, z_t, y_t) \leftarrow (S_x(t), S_z(t), S_y(t))$
6: $\quad M_t \leftarrow \textsc{Learn}_1(z_t, x_t, M_{t-1})$
7: $\quad \bar{M}_t \leftarrow ((t-1)\bar{M}_{t-1} + M_t)/t$
8: $\quad \widehat{x}_t \leftarrow \bar{M}_t z_t$
9: $\quad A_t \leftarrow \textsc{Learn}_2(\widehat{x}_t, \widehat{y}_t, A_{t-1})$
10: $\quad \bar{A}_t \leftarrow ((t-1)\bar{A}_{t-1} + A_t)/t$
11: $\quad t \leftarrow t+1$
12: **end while**
13:
14: **return** $\bar{A}_t$

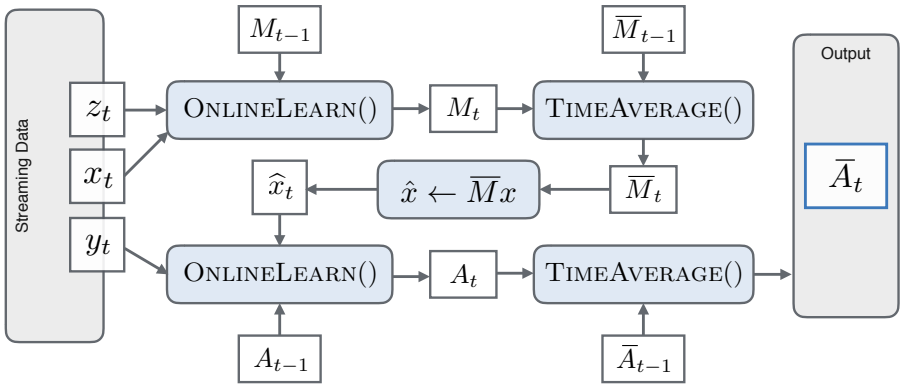


Figure 5.3: Graphical representation of Algorithm 4

From the definition of no-regret for the optimization on lines 6 and 9 in Algorithm 4, we get the following:

$$\frac{1}{T}\sum_t \|M_t z_t - x_t\|_F^2 - \frac{1}{T}\min_M \sum_t \|Mz_t - x_t\|_F^2 \leq o(T)$$
$$\frac{1}{T}\sum_t \left\|A_t \bar{M}_t z_t - y_t\right\|_F^2 - \frac{1}{T}\min_A \sum_t \left\|A\bar{M}_t z_t - y_t\right\|_F^2 \leq o(T)$$

Though these regret bounds give us a guarantee on each individual regression with respect the sequence of data points, they fail to give us the desired performance bound as we get in the single OLS scenario; these bounds do not show that this method is competitive with the optimal result from batch Ivr in hindsight (e.g., how close is A_t to A^* from Algorithm 3 with M^* instead of $\bar{M}_t$). We wish to show that this algorithm is in fact competitive with the batch instrumental variable regression algorithm (Algorithm 3). Specifically, we focus on the stochastic setting where each set of data points $(x_t, z_t, y_t) \sim P$ is i.i.d.. In this setting, we would like to show that:

$$\sum_t \mathbf{E}\left[\left\|\bar{A}_t M^* z - y\right\|_2^2\right] - \min_A \sum_t \mathbf{E}\left[\|AM^* z - y\|_2^2\right] \leq o(T)$$

where $\min_A \sum_t \mathbf{E}\left[\|AM^* z - y\|_2^2\right]$ is exactly the last objective of batch Ivr, since under the assumption that x_t, y_t, z_t are i.i.d, $\frac{1}{T}\|Y - AM^* Z\|_F^2$ (Eq. 5.4) converges to $\mathbf{E}\left[\|AM^* z - y\|_2^2\right]$ in probability. In the below section, we derive the above bound for our Oivr algorithm. Through this bound, we are able to show that even though we optimize A_t with regards to the *online* $\bar{M}_t$ at every timestep, we are finally competitive with the solution achieved by the batch procedure that uses the *batch* M^* to learn A^*. We also note that the prior technique, recursive Ivr (e.g. (Söderström and Stoica, 2002)), is similar to using FTRL (Shalev-Shwartz, 2011) with rank-1 updates. We below extend prior analysis in that we derive a regret bound for this type of update procedure.

5.4 Performance Analysis of Online IVR

In order to derive the primary theoretical contribution in this chapter, Theorem 5.4.1, we first present the lemma below. We follow with the proof for the performance guarantee of OIVR with regards to the batch IVR solution. detailed derivation.

In lines 7 and 10 of Algorithm 4, we compute an average of the sequence of predictors (matrices). This computation can be done relatively efficiently without storing all the predictors trained. The usefulness of this operation can be seen in the result of the below lemma.

Lemma 5.4.1: *Given a sequence of convex loss functions* $\{\ell_t(\beta)\}$, $1 \leq t \leq T$, *and the sequence of* $\{\beta_t\}$ *that is generated by any no-regret online algorithm, under the assumption that* ℓ_t *is i.i.d and* $\ell = \mathbf{E}(\ell_t)$, *the average* $\bar{\beta} = 1/T \sum_t \beta_t$ *of* $\{\beta_t\}$ *has the following properties:*

$$\mathbf{E}\left[\ell(\bar{\beta}) - \ell(\beta^*)\right] \leq \frac{1}{T} r(T) \to 0, \;\; T \to \infty, \tag{5.7}$$

where $r(T)$ *stands for the function of the regret bound with respect to* T[1], *which is sublinear and belongs to* $o(T)$ *for all no-regret online learning algorithms. When* ℓ *is* α *strongly convex with respect to* β *in norm* $\|\cdot\|$, *we have:*

$$\mathbf{E}\left[\|\bar{\beta} - \beta^*\|\right] \leq \frac{2}{\alpha T} r(T) \to 0, \;\; T \to \infty \tag{5.8}$$

Proof. Since we use no-regret algorithms on losses $\{\ell_t\}$, for any β^*, we have:

$$\sum_t \ell_t(\beta_t) - \sum_t \ell_t(\beta^*) \leq r(T) \in o(T), \tag{5.9}$$

where $\sum_t$ denotes $\sum_{t=1}^{T}$ and $r(T)$ is the regret rate of the online algorithm (upper bound of the regret), which is sublinear for all no-regret online algorithms. Taking the expectation of l_t on both sides of the equation (let us assume $\ell = \mathbf{E}_{\ell_t}(\ell_t), \forall t$), we have:

$$\mathbf{E}\left[\sum_t \ell(\beta_t) - \sum_t \ell(\beta^*)\right] \in o(T). \tag{5.10}$$

The expectation taken here is of β_t with respect to the stochastic losses so far. β_t (the parameter being optimized) is a random variable that depends on only the previous $t-1$ loss functions. Thus, the loss ℓ_t at step t is independent of β_t. The expectation over the losses becomes $\mathbf{E}_{\ell_1,\ldots\ell_t}[\ell_t(\beta_t)] = \mathbf{E}_{\ell_1,\ldots\ell_{t-1}}[\ell(\beta_t)]$, where the expectation of ℓ_t is ℓ (i.e. the loss is stochastic due to drawing the

[1]For instance, online gradient descent (Zinkevich, 2003) has $r_t(T) = C\sqrt{T}$ for some positive constant C.

next set of data points from an i.i.d. distribution).

Since ℓ is convex with respect to β, from Jensen's Inequality, we have:

$$\ell(\frac{1}{T}\sum_t \beta_t) \leq \frac{1}{T}\sum_t \ell(\beta_t). \tag{5.11}$$

Combining Eqn. 5.11 and Eqn. 5.10, we have:

$$\mathbf{E}\left[\ell(\bar{\beta}) - \ell(\beta^*)\right] \leq \mathbf{E}\left[\frac{1}{T}\sum_t \ell(\beta_t) - \frac{1}{T}\sum_t \ell(\beta^*)\right] \leq \mathbf{E}\left[\frac{1}{T}r(T)\right] \to 0, \text{ as } T \to \infty. \tag{5.12}$$

When ℓ is a α strongly-convex loss function with respect to β under norm $\|\cdot\|$ and $\beta^* = \arg\min_\beta \ell(\beta)$, we have that the convergence in the objective gives us convergence in the parameter:

$$\frac{\alpha}{2}\left\|\bar{\beta} - \beta^*\right\| \leq \ell(\bar{\beta}) - \ell(\beta)^* \leq \frac{1}{T}r(T). \tag{5.13}$$

Putting the expectation back and taking T to infinity, we have:

$$\mathbf{E}\left[\left\|\bar{\beta} - \beta^*\right\|\right] \leq \frac{2}{\alpha T}r(T) \to 0, \text{ as } T \to \infty. \tag{5.14}$$

□

Similar online-to-batch analysis can be found in (Cesa-Bianchi et al., 2004; Littlestone, 2014; Hazan and Kale, 2014).

With this, we now approach the main theorem for the regret bound on Online Instrumental Variable Regression. We explicitly assume that x_t, y_t and z_t are i.i.d, and $x = \mathbf{E}(x_t)$, $y = \mathbf{E}(y_t)$, $z = \mathbf{E}(z_t)$, and $\mathbf{E}(z_t z_t^T)$ is positive definite.

Theorem 5.4.1. *Assume (x_t, y_t, z_t) are i.i.d. and $\mathbf{E}(zz^T)$ is postive definite. Following any online no-regret procedure on the convex L2 losses for M_t, and A_t and computing $\bar{M}_t$, $\bar{A}_t$ as shown in Algorithm 4, we get that as $T \to \infty$:*

$$\mathbf{E}\left[\left\|\bar{A}_T M^* z - y\right\|_2^2\right] \to \mathbf{E}\left[\left\|A^* M^* z - y\right\|_2^2\right] \tag{5.15}$$

$$\text{and} \quad \bar{A}_T \to A^* \tag{5.16}$$

for the A^*, M^* from Batch IVR *(Alg. 3).*

Proof. Let $\epsilon_t = M^* - \bar{M}_t$. Then,

$$\begin{aligned}\left\|A\bar{M}_t z_t - y_t\right\|_2^2 &= \left\|A(M^* - \epsilon_t)z_t - y_t\right\|_2^2 \\ &= \left\|AM^* z_t - y_t\right\|_2^2 + \left\|A\epsilon_t z_t\right\|_2^2 - 2\left(AM^* z_t - y_t\right)^T\left(A\epsilon_t z_t\right)\end{aligned} \tag{5.17}$$

Since we run a no-regret online algorithm for A_t on loss function $\|A_t\bar{M}_t z_t - y_t\|_2^2$, we have:

$$\sum_t \|A_t\bar{M}_t z_t - y_t\|_2^2 \leq \textsc{Regret}_A + \sum_t \|A^*\bar{M}_t z_t - y_t\|_2^2$$

where $\sum_t$ denotes $\sum_{t=1}^T$.

Applying Eq. (5.17) to the right hand side, we get that for any A^* including the minimizer:

$$\sum_t \|A_t\bar{M}_t z_t - y_t\|_2^2 \leq \textsc{Regret}_A + \sum_t \|A^*M^*z_t - y_t\|_2^2 + \|A^*\epsilon_t z_t\|_2^2 - 2\left(A^*M^*z_t - y_t\right)^T\left(A^*\epsilon_t z_t\right) \tag{5.18}$$

Applying Eq. (5.17) to the left hand side, we get:

$$\begin{aligned}\sum_t \|A_tM^*z_t - y_t\|_2^2 + \|A_t\epsilon_t z_t\|_2^2 - 2\left(A_tM^*z_t - y_t\right)^T\left(A_t\epsilon_t z_t\right)& \\ \leq \textsc{Regret}_A + \sum_t \|A^*M^*z_t - y_t\|_2^2 + \|A^*\epsilon_t z_t\|_2^2 - 2\left(A^*M^*z_t - y_t\right)^T\left(A^*\epsilon_t z_t\right)&\end{aligned} \tag{5.19}$$

Rearranging terms,

$$\begin{aligned}\sum_t \|A_tM^*z_t - y_t\|_2^2 &\leq \textsc{Regret}_A + \sum_t \|A^*M^*z_t - y_t\|_2^2 + \|A^*\epsilon_t z_t\|_2^2 - 2\left(A^*M^*z_t - y_t\right)^T\left(A^*\epsilon_t z_t\right) \\ &\quad - \|A_t\epsilon_t z_t\|_2^2 + 2\left(A_tM^*z_t - y_t\right)^T\left(A_t\epsilon_t z_t\right) \qquad (5.20) \\ &\leq \textsc{Regret}_A + \sum_t \|A^*M^*z_t - y_t\|_2^2 + \|A^*\|_F^2\|\epsilon_t\|_F^2\|z_t\|_2^2 + \|A_t\|_F^2\|\epsilon_t\|_F^2\|z_t\|_2^2 \\ &\quad + 2\left|\left(A^*M^*z_t - y_t\right)^T\left(A^*\epsilon_t z_t\right)\right| + 2\left|\left(A_tM^*z_t - y_t\right)^T\left(A_t\epsilon_t z_t\right)\right| \qquad (5.21)\end{aligned}$$

The Cauchy-Swartz and the triangle inequality gives us that for any A:

$$\begin{aligned}\left|\left(AM^*z_t - y_t\right)^T\left(A\epsilon_t z_t\right)\right| &\leq \|AM^*z_t - y_t\|_2\|A\epsilon_t z_t\|_2 \\ &\leq \left(\|AM^*z_t\|_2 + \|y_t\|_2\right)\|A\epsilon_t z_t\|_2 \\ &\leq \left(\|A\|_F\|M^*\|_F\|z_t\|_2 + \|y_t\|_2\right)\|A\|_F\|\epsilon_t\|_F\|z_t\|_2 \\ &\leq \left(\|A\|_F^2\|z_t\|_2^2\|M^*\|_F + \|A\|_F\|z_t\|_2\|y_t\|_2\right)\|\epsilon_t\|_F\end{aligned} \tag{5.22}$$

Assuming that $\|z_t\|_2$, $\|y_t\|_2$, $\|M^*\|_F$, $\|A_t\|_F$, $\|A^*\|_F$ are each always upper bounded by some positive constant, define positive constants C_1 and C_2 such

that:

$$C_1 \geq \|z_t\|_2^2 \left(\|A_t\|_F^2 + \|A^*\|_F^2 \right) \tag{5.23}$$

$$C_2 \geq 2 \left(\|A_t\|_F^2 \|z_t\|_2^2 \|M^*\|_F + \|A_t\|_F \|z_t\|_2 \|y_t\|_2 + \|A^*\|_F^2 \|z_t\|_2^2 \|M^*\|_F + \|A^*\|_F \|z_t\|_2 \|y_t\|_2 \right) \tag{5.24}$$

Utilizing Eq. (5.22) with Eq. (5.21):

$$\sum_t \|A_t M^* z_t - y_t\|_2^2 \leq \textsc{Regret}_A + \sum_t \|A^* M^* z_t - y_t\|_2^2 + C_1 \|\epsilon_t\|_F^2 + C_2 \|\epsilon_t\|_F \tag{5.25}$$

Let $\mathbf{E}$ denote the expectation with regards to the whole sequence of data. Assuming that z_t, x_t, and y_t are i.i.d. we get:

$$\mathbf{E}\left[\sum_t \|A_t M^* z - y\|_2^2 \right] \leq \mathbf{E}\left[\textsc{Regret}_A \right] + \mathbf{E}\left[\sum_t \|A^* M^* z - y\|_2^2 \right] + \mathbf{E}\left[\sum_t \left(C_1 \|\epsilon_t\|_F^2 + C_2 \|\epsilon_t\|_F \right) \right] \tag{5.26}$$

Now let us consider how $\|\epsilon_t\|_F$ can be upper bounded. For ϵ_t, since we run a no-regret online algorithm on loss $\|M_t z_t - x_t\|_2^2$ assuming z_t, x_t, and y_t are i.i.d and that $\mathbf{E}[zz^T]$ is positive definite, we use Lemma 5.4.1 to get:

$$\mathbf{E}\|\epsilon_t\|_F^2 = \mathbf{E}\left\|\bar{M}_t - M^*\right\|_F^2 \leq \frac{1}{t} r_M(t) \to 0, \text{ as } t \to \infty \tag{5.27}$$

$$\Rightarrow \mathbf{E}\|\epsilon_t\|_F = \sqrt{(\mathbf{E}\|\epsilon_t\|_F)^2} \leq \sqrt{\mathbf{E}\|\epsilon_t\|_F^2} \leq \sqrt{\frac{1}{t} r_M(t)} \to 0, \text{ as } t \to \infty \tag{5.28}$$

We get the inequality in Eq. (5.28) since $\mathbf{Var}(\epsilon_t) = \mathbf{E}\left[\epsilon_t^2\right] - \mathbf{E}\left[\epsilon_t\right]^2 \geq 0$. Dividing by T in Eq. (5.26), we get:

$$\frac{1}{T}\mathbf{E}\left[\sum_t \|A_t M^* z - y\|_2^2 \right] \leq \mathbf{E}\left[\frac{\textsc{Regret}_A}{T} \right] + \mathbf{E}\left[\frac{1}{T} \sum_t \|A^* M^* z - y\|_2^2 \right] + \mathbf{E}\left[\frac{1}{T} \sum_t \left(C_1 \frac{r_M(t)}{t} + C_2 \sqrt{\frac{r_M(t)}{t}} \right) \right] \tag{5.29}$$

Note that for no-regret online algorithms, $\lim_{t\to\infty} r_M(t)/t = 0$ since $r_M(t) \in o(T)$. Then because $\mathbf{E}\left[\frac{1}{T} \sum_t \|A^* M^* z - y\|_2^2 \right] = \mathbf{E}\left[\|A^* M^* z - y\|_2^2 \right]$, applying Cesaro Mean (Hardy, 2000) and taking the limit of T, we get:

$$\frac{1}{T}\mathbf{E}\left[\sum_t \|A_t M^* z - y\|_2^2 \right] \leq \mathbf{E}\left[\|A^* M^* z - y\|_2^2 \right], \ T \to \infty \tag{5.30}$$

Thus, we have shown the algorithm is no-regret. Let $\bar{A}_T = \frac{1}{T}\sum_t A_t$. Using Jensen's inequality, we get:

$$\mathbf{E}\left[\left\|\bar{A}_T M^* z - y\right\|_2^2\right] \leq \frac{1}{T}\mathbf{E}\left[\sum_t \left\|A_t M^* z - y\right\|_2^2\right] \leq \mathbf{E}\left[\left\|A^* M^* z - y\right\|_2^2\right], \;\; T \to \infty \tag{5.31}$$

Since the above is valid for any A^*, let $A^* = \arg\min_A \mathbf{E}\left[\left\|AM^* z - y\right\|_2^2\right]$. Then by construction,

$$\mathbf{E}\left[\left\|\bar{A}_T M^* z - y\right\|_2^2\right] \geq \mathbf{E}\left[\left\|A^* M^* z - y\right\|_2^2\right], \;\; T \to \infty \tag{5.32}$$

Therefore, by Eq. (5.31) and Eq. (5.32), we have that:

$$\mathbf{E}\left[\left\|\bar{A}_T M^* z - y\right\|_2^2\right] \to \mathbf{E}\left[\left\|A^* M^* z - y\right\|_2^2\right], \;\; T \to \infty \tag{5.33}$$

With $\mathbf{E}\left[zz^T\right] \succ 0$ resulting in a strongly convex objective, we get a unique minimizer for the objective. Therefore,

$$\bar{A}_T \to A^*, \;\; T \to \infty \tag{5.34}$$

Hence, we prove the theorem. □

We also want to note that the regret rate of our algorithm depends on the no-regret online algorithms used. For instance, if we use OGD, which has no-regret rate of $O(\sqrt{T}/T)$ for $\textsc{Learn}_1$ and $\textsc{Learn}_2$, then our algorithm has a no-regret rate of $O(\sqrt{T}/T)$. The choice of learning algorithm is related to the desired trade-off between computational complexity and convergence rate. FTRL and ONS can have faster convergence, making them suitable for applications where obtaining samples is difficult: e.g., data from a physical robot. In contrast, gradient-based algorithms (e.g. iOGD, OGD) have lower computational complexity but may converge slower, making them useful for scenarios where obtaining samples is cheap, e.g., data from video games.

5.5 Dynamical Systems as Instrumental Variable Models

For a dynamical system, let us define state $s \in \mathcal{S} \in \mathbb{R}^m$ and observation $x \in \mathcal{X} \in \mathbb{R}^n$. At time step t, the system stochastically transitions from state s_t to state s_{t+1} and then receives an observation x_{t+1} corresponding to s_{t+1}. A dynamical system generates a sequence of observations x_t from latent states s_t connected in a chain (e.g., Fig. 2.4). A popular family of algorithms for representing and learning dynamical systems are predictive state representations (PSRs) (Littman et al., 2001a; Singh et al., 2004; Boots and Gordon, 2012, 2011a,b; Boots et al., 2011; Hefny et al., 2015b). It also has been shown in (Boots

and Gordon, 2011b; Hefny et al., 2015b) that we can interpret the problem of learning PSRs as linear instrumental-variable regression, which reduces the dynamical system learning problem to a regression problem.

Following (Hefny et al., 2015b), we define the predictive state Q as $Q_t = \mathbf{E}(x_{t:t+k-1}|x_{1:t-1})$ (instead of tracking the posterior distribution $\mathbf{P}(s_t|x_{1:t-1})$ on state, we track the observable representation Q_t), where $x_{t:t+k-1}$ is a k-step time window of *future* observations. We also define the *extended future* observations as $x_{t:t+k}$, which is a $(k+1)$-step time window of future observations. The predictive state representation of extended futures is defined as $P_t = \mathbf{E}(x_{t:t+k}|x_{1:t-1})$. Therefore, learning a dynamical system is equivalent to finding an operator A that maps from Q_t to P_t:

$$P_t = AQ_t \tag{5.35}$$

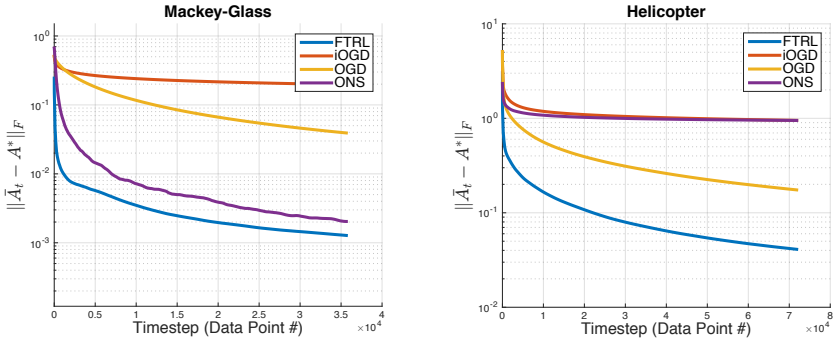
With A and the initial belief $Q_0 = \mathbf{E}(x_{0:k-1})$, we are able to perform filtering and prediction. Given the belief Q_t at step t, we use A to compute $P_t = \mathbf{E}(x_{t:t+k}|x_{1:t-1})$. To compute $\mathbf{E}(x_{t+1:t+k}|x_{1:t-1})$ (prediction), we simply drop the x_t from P_t. For filtering, given a new observation x_t, under the assumption that the extended future $x_{t:t+k}$ has constant covariance, we can compute $\mathbf{E}(x_{t+1:t+k}|x_{1:t})$ by simply performing a conditional Gaussian operation.

A naive approach to compute A is to use ordinary linear regression directly from futures $x_{t:t+k-1}$ to extended futures $x_{t:t+k}$. However, even though $x_{t:t+k-1}$ and $x_{t:t+k}$ are unbiased samples of Q_t and P_t, they are noisy observations of Q_t and P_t respectively. The noises overlap: $x_{t:t+k-1}$ and $x_{t:t+k}$ share a k-step time window (Hefny et al., 2015b). Therefore, directly regressing from Q_t to P_t gives a biased estimate of A, which can lead to poor prediction and filtering performance. Indeed, as verified in our experiments, the biased A computed by ordinary least square regression performs worse in comparison to the IVR based methods.

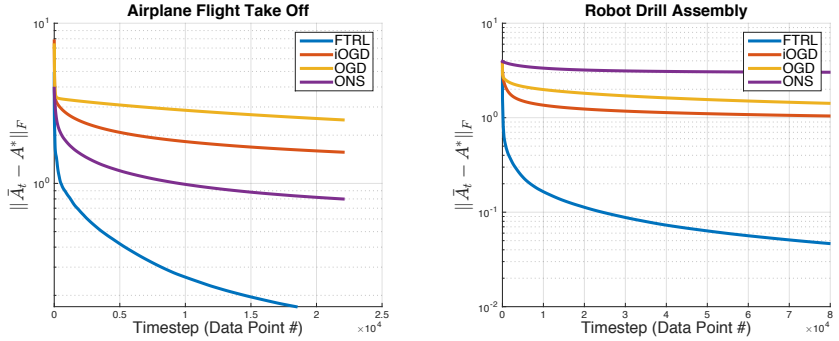
To overcome the bias, the authors in (Boots and Gordon, 2011b; Hefny et al., 2015b) introduce past observations $x_{t-k:t-1}$ as instruments. The past observations $x_{t-k:t-1}$ are not correlated with the *noise* in the future observations $x_{t:t+k-1}$ and extended future observations $x_{t:t+k}$ but are correlated with Q_t. Explicitly matching terms to those used in IVR, the noisy observation of P_t is equivalent to y, the noisy observation of Q_t is equivalent to x, and the past observations $x_{t-k:t-1}$ is the instrumental variable z. Unlike (Hefny et al., 2015b), which introduced batch IVR (Alg. 3) for system identification, we use OIVR (Alg. 4) where we receive observations online.

5.5.1 Online Learning for Dynamical Systems

Given OIVR, learning dynamical systems online becomes straightforward. To apply Alg. 4 to model dynamical systems, we maintain a k-step time window of the *future* $x_{t:t+k-1}$, a $(k+1)$-step time window of the *extended future* $x_{t:t+k}$, and a k-step time window of the *past* $x_{t-k:t-1}$. Matching terms to Alg. 4, we set $x_t = x_{t:t+k-1}$, $y_t = x_{t:t+k}$, and $z_t = x_{t-k:t-1}$. With x_t, y_t and z_t, we update M_t and A_t following lines 6 and 9. When a new observation x_{t+k+1} is received, the



(a) Mackey-Glass ($\tau = 10$) (b) Helicopter

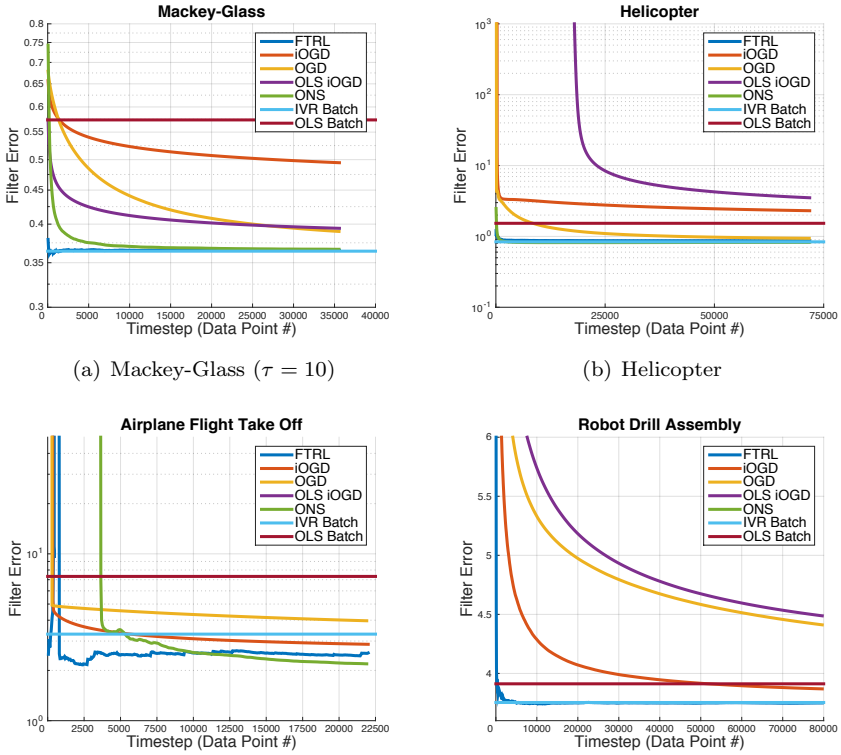


(c) Airplane Flight Take Off (d) Robot Drill Assembly

Figure 5.4: Convergence Plots for the Dynamical System Experiments. (Best viewed in color)

update of x_t and y_t and z_t to x_{t+1}, y_{t+1} and z_{t+1} is simple and can be computed efficiently (e.g., to compute $y_{t+1} = x_{t+1:t+k+1}$, we simply drop x_t from x_t and append x_{t+k+1} at the end (i.e. circular buffer)).

By maintaining these three fixed-step time window of observations instead of building a large Hankel matrix ((Hefny et al., 2015b; Boots et al., 2011)) that stores concatenations of all the observations, we significantly reduce the required space complexity. At every online update step (lines 6 and 9 in Alg. 4), the online learning procedure usually has lower computational complexity. For instance, using Online Gradient Descent (Zinkevich, 2003) requires $O((kn)^2)$ computations at each step compared to the $O((kn)^3)$ in the batch-based algorithms (usually due to matrix inversions).



(a) Mackey-Glass ($\tau = 10$)

(b) Helicopter

(c) Airplane Flight Take Off

(d) Robot Drill Assembly

Figure 5.5: Filtering Error for the Dynamical System Experiments. Note that the results for OLS iOGD in Fig. 5.5(c) and for ONS in Fig. 5.5(d) are higher than the plotted area. (Best viewed in color)

5.6 Experiments

We demonstrate the performance OIVR on a variety of dynamics benchmark and one illustrative econometrics problem. In Fig. 5.4, we show the convergence of the estimated $\bar{A}_t$ in OIVR to the A^* computed with IVR. As an additional performance metric, we report the observation prediction error with a constant covariance Kalman filter using $\bar{A}_t$ (Fig. 5.5) on a set of held out test trajectories. For computational reasons, we report the filter error after every 50 data points given to the online learner. Below we describe each our test benches.

MG-10 The Mackey-Glass (MG) time-series is a standard dynamical modeling benchmark (Ralaivola and D'Alche-Buc, 2004; Wingate and Singh, 2006) generated from the nonlinear time-delay differential equation $\dot{x}(t) = -bx(t) + \frac{ax(t-\tau)}{1+x(t-\tau)^{10}}$. This system produces chaotic behavior for larger time delays τ

(seconds).

Helicopter The simulated helicopter from (Abbeel and Ng, 2005b) computes its dynamics in a 21-dimensional state space with a 4-dimensional control input. In our experiments, a closed loop LQR controller attempts to bring the helicopter to hover at a fixed point from randomly chosen starting configurations. White noise is added in each state transition. The LQR controller chooses actions based on state and it poses a challenge for the learner to extract this implicit relationship governing the evolution of the system.

Airplane Flight Take Off We also consider the complex dynamics generated during a DA-42 airplane's take off in a flight simulator, X-plane (Research, 2015), a well known program for training pilots. Trajectories of observations, which include among others speed, height, angles, and the pilot's control inputs, were collected were collected from a human expert controlling the aircraft. Due to high correlation among the observation dimensions, we precompute a whitening projection at the beginning of online learning using a small set of observations to reduce the dimensionality of the observations by an order of magnitude.

Robot Drill Assembly Our final dynamics benchmark consists of 96 sensor telemetry traces from a robotic manipulator assembling the battery pack on a power drill. The 13 dimensional observations consist of the robot arm's 7 joint torques as well as the 3D force and torque vectors as measured at the wrist of the robotic arm. The difficulty in this real-world dataset is that the sensors, especially the force-torque sensor, are known to be noisy and are prone to hysteresis. Additionally, the fixed higher level closed-loop control policy for the drill assembly task is not explicitly given in the observations and must be implicitly learned.

We applied four different no-regret online learning algorithms on the dynamical system test benches: OGD (Online Gradient Descent), iOGD (implict Online Gradient Descent), ONS (Online Newton Step), and FTRL (Follow the Regularized Leader). Fig. 5.4 shows the convergence of these online algorithms in terms of $\|\bar{A}_t - A^*\|_F$, where $\bar{A}_t$ is the solution of OIVR at time step t and A^* is the solution from batch IVR. Though FTRL had the fastest convergence, FRTL is memory intensive and computationally expensive as it runs a batch IVR at every time step over all the data points which have to be stored. ONS, also a computationally intensive algorithm, generally achieved fast convergence on the testbenches, except in the Robot Drill Assembly benchmark due to the difficulty in tuning the parameters of ONS. In general, OGD and iOGD perform well while only requiring storage of the latest data point. Furthermore, these algorithms have lower computational complexity than ONS and FTRL at each iteration.

We also compared the filtering performance of these OIVR methods with batch IVR, batch OLS, and online OLS (via iOGD) on these datasets. The

results are shown in Fig. 5.5. First, by comparing the batch IVR and batch OLS, we observe that the biased A computed by batch OLS is consistently outperformed on the filtering error by the A computed by from batch IVR. Secondly, we also compare the performance of OIVR and online OLS where OIVR outperforms online OLS in in most cases. In Fig. 5.5(c) we notice that OIVR with FTRL, ONS, iOGD gives smaller filter error than batch IVR. This is possible since IVR does not explicitly minimize the filter error but instead minimizes the single step prediction error. The consistency result for IVR only holds if the system has truely linear dynamics. However, as our dynamics benchmarks consist of non-linear dynamics, there may exist a linear estimator of the system dynamics that can outperform IVR in terms of minimizing filter error.

College Distance We finally consider an econometrics problem, a traditional application domain for instrumental variable regression, the College Distance vignette. In this experiment, we try to predict future wages given the number of years of education as the explanatory variable (feature) and the distance to the nearest 4 year college as the instrument (Kleiber and Zeileis, 2008; Card, 1993). The claim is that the distance is correlated with the years of college but is uncorrelated with future wages except through the education level. The goal is to find the meaningful linear coefficient from education level to wages. As such, we do not compare against OLS as it does not try to find a similarly meaningful representation. We see in Table 5.1 that the online algorithm is able to converge on the solution found in the batch setting.

	Ivr	**iOGD**	**ONS**	**OGD**	**FTRL**
Computed A	0.688	0.690	0.689	0.698	0.688

Table 5.1: $\bar{A}$ found using various online no-regret algorithms versus A from batch IVR for the **College Distance** dataset.

5.7 Conclusion

We introduced a new algorithm, Online Instrumental Variable Regression, and proved strong theoretical performance bounds with regard to the traditional batch IVR setting. Through connections between IVR and dynamical system identification, we introduced a rich new family of online system identification algorithms for partially observable systems. Our results further show that with a predictive representation built from sufficient statistics of future observations, we can improve filtering performance on a variety of benchmarks, moving us toward a technique for handling Challenge 2. In Chapters 6 and 7, we derive a more robust approach to directly optimizing for filtering performance using similar ideas on predictive representations. We then expand this idea to augment general recurrent neural network in Chapter 8 for applications beyond probabilistic filtering, such as imitation and reinforcement learning.

Chapter 6

Nonparametric filter learning: Predictive State Inference Machines

Bayesian filtering plays a vital role in applications ranging from robotic state estimation to visual tracking in images to real-time natural language processing. Filtering allows the system to reason about the current state given a sequence of observations. Filtering is usually connected to the time-series domain of problems through the transition (dynamics) and observation (sensor) models. Machine learned models are primarily utilized to find these models (Fig. 6.1), e.g., linear dynamics and observation models for the Kalman Filter (Roweis and Ghahramani, 1999), the Unscented Kalman Filter for nonlinear models with Gaussian noise (Ko et al., 2007; Wan and Van Der Merwe, 2000), or particle filters for nonlinear models from other distributions (Thrun et al., 2005).

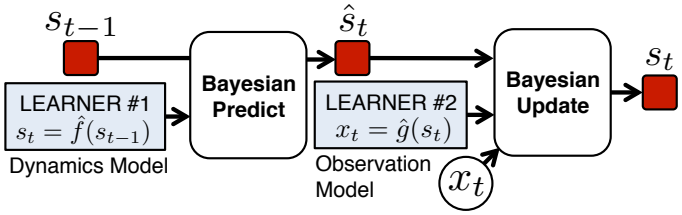


Figure 6.1: Traditional "Filter Learning". Learning is often decoupled for the transition and observation models.

In contrast to the classical separation of model learning and the probabilistic filtering method, we propose Predictive State Inference Machines

This work was originally presented in *Learning to Filter with Predictive State Inference Machines* at ICML 2016 (Sun et al., 2016) and contains joint work done in collaboration with Wen Sun and Byron Boots.

(PSIMs), an algorithm that treats the inference procedure (filtering) on a dynamical system as a composition of predictors. Our procedure takes the current predictive state and the latest observation from the dynamical system as inputs and outputs the next predictive state (Fig. 6.2). PSIM allows us to treat filtering as a general supervised learning problem handed-off to a black box learner of our choosing, where the complexity of the learner naturally controls the trade-off between computational complexity and prediction accuracy. This procedure learns an implicit dynamics model in order to directly tackles the filtering problem (Challenge 2).

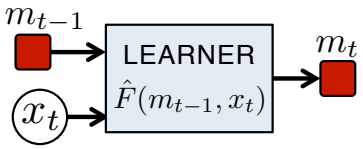


Figure 6.2: Filter Learning with PSIMs. This framework learns a joint predict-and-update function to predict the next belief state.

In traditional filtering, the process (dynamics) model describes the transition of the system from state s_t to state s_{t+1} by specifying $P(s_{t+1}|s_t)$, and the sensor (observation) model generates a distribution over observations $P(x_t|s_t)$ given state. Using these models in conjunction with a new observation x_t, the filter conditions on observations to compute the posterior $P(s_t|x_t)$. As a result, the performance of the filter, its ability to estimate the state or predict future observations, is limited by the fidelity of dynamics and observation models (Aguirre et al., 2005).

However, we often do not have knowledge of the true system's state s_t. This could be due to the difficulty of instrumenting it or we may not know how to fully parametrize the model. The classic generative approach is to assume that each observation is correlated to the value of a latent state in a graphical model. The parameters of the model are then optimized with Maximum Likelihood Estimation (MLE) based methods (e.g. (Coates et al., 2008)); however, these approaches have at least two shortcomings. First, it may be difficult to find an appropriate parametrization for the latent states. If the model is parametrized incorrectly, the learned model may exhibit poor performance on inference tasks such as Bayesian filtering or predicting multiple time steps into the future. Second, the MLE objective is non-convex and finding the globally optimal solution is often computationally infeasible. Instead, algorithms such as Expectation-Maximization (EM) are used to compute locally optimal solutions. Although the maximizer of the likelihood objective can promise good performance guarantees when it is used for inference, the locally optimal solutions returned by EM typically do not have any performance guarantees.

Spectral Learning methods are a popular alternative to MLE for learning models of dynamical systems (e.g. (Boots, 2012; Hsu et al., 2009; Hefny et al., 2015a) and provides theoretical guarantees on discovering the global optimum

for the model parameters under the assumptions of infinite training data and realizability. However, in the non-realizable setting — i.e. model mismatch (e.g., using learned parameters of a Linear Dynamical System (LDS) model for a non-linear dynamical system) — these algorithms lose any performance guarantees on using the learned model for filtering or other inference tasks.

In scenarios where our ultimate goal is to infer some quantity from observed data, a natural solution is to skip the step of learning a model, and instead directly optimize the inference procedure (Fig. 6.2). Toward this end, we generalize the *supervised message-passing Inference Machine* approach of Ross et al. (2011b); Ramakrishna et al. (2014); Lin et al. (2015). Inference machines do not parametrize the graphical model (e.g., design of potential functions) and instead directly train predictors that use incoming messages and local features to predict outgoing messages via black-box supervised learning algorithms. By combining the model and inference procedure into a single object — an *Inference Machine* — we directly optimize the end-to-end quality of inference. This unified perspective of learning and inference enables stronger theoretical guarantees on the inference procedure: the ultimate task that we care about.

One of the principal limitations of inference machines is that they require supervision. If we only have access to observations during training, then there is no obvious way to apply the inference machine framework to graphical models with latent states. We leverage ideas from *Predictive State Representations* (PSRs) (Littman et al., 2001a; Singh et al., 2004; Boots et al., 2011; Hefny et al., 2015a) to develop a training procedure without supervision. In contrast to latent variable representations of dynamical systems, which represent the belief state as a probability distribution over the unobserved state space of the model, PSRs instead maintain an *equivalent* belief over sufficient features of future observations.

We present an abbreviated introduction to Predictive State Inference Machines (PSIMs) here and refer the reader to (Sun et al., 2016) for more details.

6.1 Predictive State Representations (PSRs)

We follow a predictive state representation (PSR) framework and define state as the distribution of a k-step fixed size time window of *future* observations, $f_t = [x_t^T, ..., x_{t+k-1}^T]^T \in \mathbb{R}^{kn}$ (Hefny et al., 2015a)[1]. The key assumption in PSRs is that the state of the dynamical system at timestep t is equivalent to being able to predict everything about f_t at time-step t (e.g., the distribution of f_t) (Singh et al., 2004). We assume in our work that systems we consider are k-observable[2] for $k \in \mathbb{N}^+$: there is a bijective function that maps $P(s_t|h_{t-1})$ to $P(f_t|h_{t-1})$. For convenience of notation, we will present our results in terms

[1]This is similar to the rank k of the observability matrix $O = [CA, CA^2, \ldots, CA^k]$ for linear systems (Aström and Murray, 2010)

[2]This assumption allows us to avoid the cryptographic hardness of the general problem (Hsu et al., 2009).

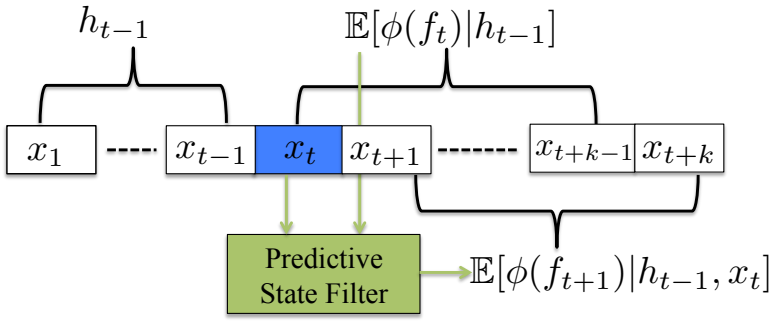


Figure 6.3: Filtering with predictive states for a k-observable system. At time step t, the filter uses the belief $\mathbb{E}[\phi(f_t)|h_{t-1}]$ and the latest observation x_t as feedback, outputs the next belief $\mathbb{E}[\phi(f_{t+1})|h_{t-1}, x_t]$.

of k-observable systems, where it suffices to select features from the next k observations.

Following Hefny et al. (2015a), we define the predictive state at time step t as $\mathbb{E}[\phi(f_t)|h_{t-1}]$ where ϕ is some feature function that is sufficient for the distribution $P(f_t|h_{t-1})$. The expectation is taken with respect to the distribution $P(f_t|h_{t-1})$: $\mathbb{E}[\phi(f_t)|h_{t-1}] = \int_{f_t} \phi(f_t)P(f_t|h_{t-1})df_t$. The conditional expectation can be understood as a function of which the input is the random variable h_{t-1}. For example, we can set $\mathbb{E}[\phi(f)|h_{t-1}] = \mathbb{E}[f, ff^T|h_{t-1}]$ if $P(f_t|h_{t-1})$ is a Gaussian distribution (e.g., linear dynamical system); or we can set $\phi(f) = [x_t \otimes ... \otimes x_{t+k-1}]$ if we are working on a discrete models (discrete latent states and discrete observations), where x_t is an indicator vector representation of the observation and $\otimes$ is the tensor product. In summary, we assume that there exists a bijective function mapping

$$P(s_t|h_{t-1}) \Leftrightarrow P(f_t|h_{t-1}) \Leftrightarrow \mathbb{E}[\phi(f_t)|h_{t-1}]. \tag{6.1}$$

Note that the mapping from $\mathbb{E}[\phi(f_t)|h_{t-1}]$ to $P(f'_t|h_{t-1})$ is not necessarily linear. To filter from the current predictive state $\mathbb{E}[\phi(f_t)|h_{t-1}]$ to the next state $\mathbb{E}[\phi(f_{t+1})|h_t]$ conditioned on the most recent observation x_t (see Fig. 6.3 for an illustration), PSRs additionally additionally assume a linear mapping to extended-state operator as well as nonlinear conditioning operator that can compute the next predictive state with the extended state and the latest observation as inputs.

6.2 Predictive State Inference Machines

Inference Machines reduce the problem of learning graphical models to solving a set of discriminative classification or regression problems, where the learned classifiers mimic message passing procedures that output marginal distributions for the nodes in the model (e.g. (Ross et al., 2011b)). However, Inference

Machines cannot be directly applied to learning latent state space models as we lack the supervision over these hidden states.

We address this limitation using predictive state representations. By using an observable representation for state, observations in the training data can be used for supervision in the inference machine. From Eq. (6.1), instead of tracking the hidden state s_t, we focus on the corresponding predictive state $\mathbb{E}[\phi(f_t)|h_{t-1}]$. The training data can then quantify how good the predictive state is by computing the likelihood of f_t. The goal is to learn an operator F (the green box in Fig. 6.3) which *deterministically* passes the predictive states forward in time conditioned on the latest observation:

$$\mathbb{E}[\phi(f_{t+1})|h_t] = F\Big(\mathbb{E}[\phi(f_t)|h_{t-1}], x_t\Big), \tag{6.2}$$

such that the likelihood of the observations $\{f_t\}_t$ being generated from the sequence of predictive states $\{\mathbb{E}[\phi(f_t)|h_{t-1}]\}_t$ is maximized. In the standard PSR framework, the predictor F can be regarded as the composition of the linear mapping (from predictive state to extended state) and the conditioning operator. It is important note the *equivalence* of predictive state representations and latent variable models. This equivalence allows us to use the inference machine framework with the "future" as supervision. Thus, if we can correctly filter with predictive states, then this is equivalent to filtering with latent states.

We do not place any parametrization assumptions on the transition and observation models (as in latent state-space models). Instead, we parametrize and restrict the class of predictors to encode the underlying dynamical system and aim to find a predictor F from the restricted class to forward propagate this predictive state (Eq. (6.2)). We call this framework for inference the Predictive State Inference Machine (PSIM).

PSIM is different from PSRs in the following respects:

1. PSIM collapses the two steps of PSRs (predict the extended state and then condition on the latest observation) into one step—as an Inference Machine—for closed-loop update of predictive states

2. PSIM directly targets the filtering task and has theoretical guarantees on the filtering performance

3. Unlike PSRs where one usually needs to utilize linear PSRs for learning purposes (Boots et al., 2011), PSIM can generalize to non-linear dynamics by leveraging non-linear regression or classification models.

6.2.1 Learning PSIMs

For notational simplicity, let τ be a trajectory which is sampled from a unknown distribution $\mathcal{D}_\tau$. We denote the predictive state as

$$m_t = \mathbb{E}[\phi(f_t)|h_{t-1}]. \tag{6.3}$$

We use $\hat{m}_t$ to denote an approximation of m_t. Given a predictive state m_t and a noisy observation f_t conditioned on the history h_{t-1}, we let the loss function[3]

$$d(m_t, f_t) = \|m_t - \phi(f_t)\|_2^2.$$

This squares loss function can be regarded as matching moments. For instance, in the stationary Kalman filter setting, we could set $m_t = \mathbb{E}[f_t|h_{t-1}]$ and $d(m_t, f_t) = \|m_t - f_t\|_2^2$ for matching the first moment. We present two different methods for optimizing the PSIM filter. The first method is based on Forward Training (Ross and Bagnell, 2010) and the second on Dataset-Aggregation (DAgger) (Ross et al., 2011a). Both methods use a similar procedure as DAD (Chapter 3) to generate synthetic training examples by treating the time-indexed training trajectories as oracles to make the predictors robust to their own induced distribution.

6.3 Forward Training PSIMs

PSIM with Forward Training aims learn a non-stationary filter. We formulate this as optimizing a good sequence of hypotheses $\{F_t\}$ such that:

$$\min_{F_1 \in \mathcal{F}, \ldots F_T \in \mathcal{F}} \mathbb{E}_{\tau \sim \mathcal{D}_\tau} \Big[\frac{1}{T} \sum_{t=1}^{T} d(F_t(\hat{m}_t^\tau, x_t^\tau), f_{t+1}^\tau) \Big], \tag{6.4}$$

$$s.t. \;\; \hat{m}_{t+1}^\tau = F_t(\hat{m}_t^\tau, x_t^\tau), \forall t \in [1, T-1], \tag{6.5}$$

where $\hat{m}_1 = \arg\min_m \sum_{t=1}^{M} d(m, f_1^i)$, which is equal to $\frac{1}{T}\sum_{i=1}^{T} \phi(f_i^i)$. This problem is not separable since we need to learn the first F_1 in order to generate the input features for learning F_2. We optimize this following the procedure described in Algorithm 5.

To better understand what this training procedure does, let us define ω_t as the joint distribution of feature variables z_t and targets f_{t+1} after rolling out $F_1, ..., F_{t-1}$ on the trajectories sampled from $\mathcal{D}_\tau$ (similar to Eq. (3.7)). Under this definition, the filter error defined above is equivalent to

$$\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}_{(z,f)\sim\omega_t} \Big[d(F_t(z), f) \Big].$$

Essentially the dataset D_t collected on Line 5 by Algorithm 5 forms a finite sample estimation of ω_t.

The forward-training method for PSIM allows us to derive a few theoretical bounds and guarantees. The theorems are reproduced below, and we refer the reader to (Sun et al., 2016) for proofs and additional details. For the first result we assume that every learning problem for F_t can be solved perfectly (i.e. risk minimizer finds the Bayes optimal) (Langford et al., 2009).

[3]Squared loss in an example Bregman divergence of which there are others that are optimized by the conditional expectation (Banerjee et al., 2005). We can design $d(m_t, f_t)$ as negative log-likelihood, as long as it can be represented as a Bregman divergence (e.g., negative log-likelihood of distributions in exponential family).

Algorithm 5 PREDICTIVE STATE INFERENCE MACHINE (PSIM) with Forward Training

Input: M independent trajectories τ_i, $1 \leq i \leq M$;
Set $\hat{m}_1 = \frac{1}{M}\sum_{i=1}^{M} \phi(f_1^i)$;
Set $\hat{m}_1^i = \hat{m}_1$ for trajectory $\tau_i, 1 \leq i \leq M$;
for t = 1 to T **do**
    For each trajectory τ_i, add the input $z_t^i = (\hat{m}_t^i, x_t^i)$ to D_t as feature variables and the corresponding f_{t+1}^i to D_t as the targets;
    Train a hypothesis F_t on D_t to minimize the loss $d(F(z), f)$ over D_t;
    For each trajectory τ_i, roll out $F_1, ..., F_t$ along the trajectory (Eq. 6.5) to compute $\hat{m}_{t+1}^i$;
end for
Return: the sequence of hypothesis $\{F_t\}_{t=1}^N$.

Theorem 6.3.1. *With infinite many training trajectories and in the realizable case, if all learning problems are solved perfectly, the sequence of predictors $F_1, F_2, ..., F_T$ from Algorithm 5 can generate exact predictive states $\mathbb{E}[\phi(f_t^\tau)|h_{t-1}^\tau]$ for any trajectory $\tau \sim \mathcal{D}_\tau$ and $1 \leq t \leq T$.*

In addition to the above assumption and the usual asymptotic in dataset size requirement, Theorem 6.3.1 assumes the problem is realizable. That is, the underlying true filters $F_1^*, ..., F_T^*$ are in the hypothesis class $\mathcal{F}$. Though this is not often the case for real-world problems and small function classes, it shows that in a well-behaved scenario, we can achieve the correct solution – similar to the guarantees given by spectral methods.

Our second result addresses this limitation. For the model-agnostic case (i.e. possibly not realizable), we show that Algorithm 5 can still achieve a reasonable upper bound. Let us define

$$\epsilon_t = \min_{F \sim \mathcal{F}} \mathbb{E}_{(z,f)\sim\omega_t}[d(F(z), f)],$$

which is the minimum batch training error under the distribution of inputs resulting from hypothesis class $\mathcal{F}$. Let us define $\epsilon_{\max} = \max_t\{\epsilon_t\}$. Under infinite many training trajectories, even in the model agnostic case, we have the following guarantees for filtering error for Alg. 5:

Theorem 6.3.2. *With infinite many training trajectories, for the sequence $\{F_t\}_t$ generated by Alg. 5, we have:*

$$\mathbb{E}_{\tau\sim\mathcal{D}_\tau}\Big[\frac{1}{T}\sum_{t=1}^{T} d(F_t(\hat{m}_t^\tau, x_t^\tau), f_{t+1}^\tau)\Big] = \frac{1}{T}\sum_t \epsilon_t \leq \epsilon_{max}.$$

Theorem. 6.3.2 shows that the filtering error is upper-bounded by the average of the minimum batch training errors from each step. If we have a rich class of hypotheses and small noise (e.g., small Bayes error), ϵ_t could be small.

The final result we derive addresses the infinite-sample assumptions from above to provide a finite sample analysis. To analyze finite sample complexity, we need to split the dataset into T disjoint sets to make sure that the samples in the dataset D_t are i.i.d. Hence we reduce forward training to T independent supervised learning problems. We have the following agnostic theoretical bound:

Theorem 6.3.3. *With M training trajectories, for any $F_t^* \in \mathcal{F}, \forall t$, we have with probability at least $1-\delta$:*

$$\begin{aligned}
&\mathbb{E}_{\tau\sim\mathcal{D}_\tau}\Big[\frac{1}{T}\sum_{t=1}^{T} d(F_t(\hat{m}_t^\tau, x_t^\tau), f_{t+1}^\tau)\Big] \\
&\leq \mathbb{E}_{\tau\sim\mathcal{D}_\tau}\big[\frac{1}{T}\sum_{t=1}^{T} d(F_t^*(\hat{m}_t^\tau, x_t^\tau), f_{t+1}^\tau)\big] \\
&\quad + 4\nu\bar{\mathcal{R}}(\mathcal{F}) + 2\sqrt{\frac{T\ln(T/\delta)}{2M}},
\end{aligned} \tag{6.6}$$

where $v = \sup_{F,z,f} 2\|F(z)-f\|_2$, $\bar{\mathcal{R}}(\mathcal{F}) = \frac{1}{T}\sum_{t=1}^{T}\mathcal{R}_t(\mathcal{F}))$ and $\mathcal{R}_t(\mathcal{F})$ is the Rademacher number of $\mathcal{F}$ under ω_t.

As one might expect, the learning problem becomes harder as T increases. Although Algorithm 5 has nice theoretical properties, it is not very data efficient. It requires many trajectories to get good performance, but in practice, it is possible that we only have small number of training trajectories. The other major limitation is that often problems have long horizons (T is big). It becomes impractical to construct as many predictors as the horizon of the problem – it would require even more data and possible be computationally and memory intractable. We introduce a second training procedure that addresses these practical concerns.

6.4 DAgger (DaD) training PSIMs

The PSIM training with DAgger trains a stationary filter F. With this filter, we can filter for an arbitrary horizon. This training method shares a lot of similarity with the multi-step training procedure introduced in Chapter 3. The optimization we wish to target for finding a good stationary filter F is

$$\min_{F\in\mathcal{F}} \mathbb{E}_{\tau\sim\mathcal{D}_\tau} \frac{1}{T}\sum_{t=1}^{T} d(F(\hat{m}_t, x_t), f_{t+1}), \tag{6.7}$$

$$s.t \quad \hat{m}_{t+1} = F(\hat{m}_t, x_t), \forall t \in [1, T-1], \tag{6.8}$$

where

$$\hat{m}_1 = \arg\min_m \sum_{t=1}^{M} d(m, f_1^i) = \frac{1}{T}\sum_{i=1}^{T}\phi(f_i^i). \tag{6.9}$$

Algorithm 6 PREDICTIVE STATE INFERENCE MACHINE (PSIM) with DAgger Training

Input: M independent trajectories τ_i, $1 \leq i \leq M$
Output: Filter function F
1: Initialize $D_0 \leftarrow \emptyset$ and initialize F_0 to be any hypothesis[4] in $\mathcal{F}$;
2: Initialize $\hat{m}_1 = \frac{1}{M} \sum_{i=1}^{M} \phi(f_1^i)$
3: **for** n = 0 to N **do**
4: Use F_n to perform belief propagation (Eq. (6.8)) on trajectory τ_i, $1 \leq i \leq M$
5: For each trajectory τ_i and each time step t, add the input $z_t^i = (m_t^{i,F_n}, x_t^i)$ encountered by F_n to D'_{n+1} as feature variables and the corresponding f_{t+1}^i to D'_{n+1} as the targets ;
6: Aggregate dataset $D_{n+1} = D_n \cup D'_{n+1}$;
7: Train a new hypothesis F_{n+1} on D_{n+1} to minimize the loss $d(F(m, x), f)$
8: **end for**
9: **Return:** the best hypothesis $\hat{F} \in \{F_n\}_n$ on validation trajectories.

Note that the above objective function is non-convex since the input feature $\hat{m}_t$ for each t depend on the prediction from the previous timestep. Optimizing this objective function via back-propagation-through-time (BPTT) can be difficult and likely leads to local optima. Instead, we optimize the above objective function using the lessons learned in Chapter 3. The iterative approach we describe (Algorithm 6, Fig. 6.4) is based on DAD, which itself based on approach called Dataset Aggregation (DAgger) (Ross et al., 2011a). Due to the non-convexity of the objective, DAgger does not promise global optimality, but PSIM with DAgger gives us a sound theoretical bound for multi-step filtering error in terms of the single-step batch error.

Given a trajectory τ and hypothesis F, we define $\hat{m}_t^{\tau,F}$ as the predictive belief generated by F on τ at time step t. We also define $z_t^{\tau,F}$ to represent the feature variables $(\hat{m}_t^{\tau,F}, x_t^\tau)$. At iteration n, Algorithm 6 rolls out the predictive states using its current hypothesis F_n (Eq. (6.8)) on all the given training trajectories (Line 4). Then it collects all the feature variables $\{(\hat{m}_t^{i,F_n}, x_t^i)\}_{t,i}$ and the corresponding target variables $\{f_{t+1}^i\}_{t,i}$ from the ground truth trajectory to form a new dataset D'_n. This is then aggregated with the to the original dataset D_{n-1}. Then a new hypothesis F_n is learned from the aggregated dataset D_n by minimizing the loss $d(F(z), f)$ over D_n. This procedure is in the in the same vein DAD extended to work for PSIM. We illustrate this procedure in Fig. 6.4

By using DAgger, we can guarantee a hypothesis that, when used during filtering, performs nearly as well as when performing regression on the aggregate dataset D_N. In practice, with a rich hypothesis class $\mathcal{F}$ and small noise (e.g., small Bayes error), small regression error is possible. We can use the results of Ross et al. (2011a) to give a couple theoretical guarantees on the filter function

[4]Better performance is often achieved if F_0 is initialized by minimizing $d(F(\phi(f_t), x), \phi(f_t)))$, the single-step loss (like in Algorithm 1).

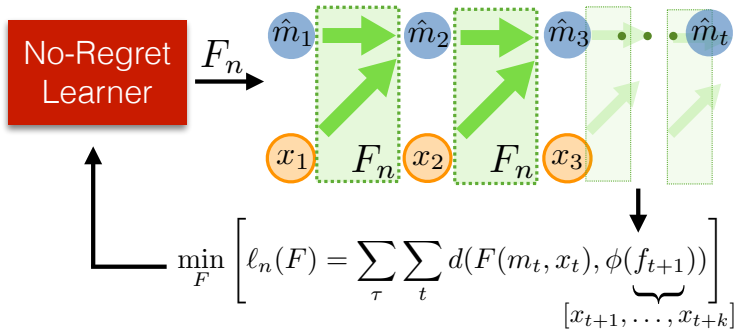


Figure 6.4: Graphical representation of PSIM with DAgger training, Algorithm 6. The model F_n from the n-th iteration is rolled forward to generated predicted message $\hat{m}$ (Eq. (6.3)). These are then trained against the ground truth trajectory from which features $\phi(f_t)$ are constructed. By aggregating data over each of the models F_n, we can learn a predictor with good multi-step filtering performance over the time horizon.

used.

Let us fix a hypothesis F and a trajectory τ, we define $\omega_{F,\tau}$ as the uniform distribution of (z, f): $\omega_{F,\tau} = \mathcal{U}\Big[(z_1^{\tau,F}, f_2^\tau), ..., (z_T^{\tau,F}, f_{T+1}^\tau)\Big]$. The filtering error Eq. (6.7) can be rewritten as

$$L(F) = \mathbb{E}_\tau[\mathbb{E}_{z,f\sim\omega_{F,\tau}}[d(F(z), f)]|\tau]. \tag{6.10}$$

Define the loss function for any predictor F at iteration n of Algorithm 6 as:

$$L_n(F) = \mathbb{E}_\tau[\mathbb{E}_{z,f\sim\omega_{F_n,\tau}}[d(F(z), f)]|\tau]. \tag{6.11}$$

At iteration n, the dataset D'_n made from M trajectories forms an empirical estimate of the loss L_n:

$$\hat{L}_n(F) = \frac{1}{M}\sum_{\tau=1}^{M} \mathbb{E}_{z,f\sim\omega_{F_n,\tau}}\big(d(F(z), f)\big). \tag{6.12}$$

Algorithm 6 can be regarded as running the Follow the Leader (FTL) (Cesa-Bianchi et al., 2004; Hazan et al., 2007) on the sequence of loss functions $\{L_n(F)\}_{n=1}^N$. When the loss function $L_n(F)$ is strongly convex with respect to F, FTL is no-regret in a sense that $\lim_{N\to\infty} \gamma_N = 0$.

Now, define the minimum average training error

$$\epsilon_N = \min_{F\in\mathcal{F}} \frac{1}{N}\sum_{n=1}^{N} L_n(F). \tag{6.13}$$

The *Regret* γ_N of an algorithm is then given as

$$\frac{1}{N}\sum_{n=1}^{N} L_n(F_n) - \underbrace{\min_{F\in\mathcal{F}} \frac{1}{N}\sum_{n=1}^{N} L_n(F)}_{\epsilon_N} \leq \gamma_N \tag{6.14}$$

For the first result, assume that $M = \infty \Rightarrow \hat{L}_n(F) = L_n(F)$. Applying Theorem 4.1 and its reduction to no-regret learning analysis from (Ross et al., 2011a) to our setting, we have the following guarantee for filtering error:

Corollary 1. *(Ross et al., 2011a) For Algorithm 6, there exists a predictor* $\hat{F} \in \{F_n\}_{n=1}^{N}$ *such that:*

$$L(\hat{F}) = \mathbb{E}_\tau\big[\mathbb{E}_{z,f\sim\omega_{\hat{F},\tau}}(d(\hat{F}(z), f))|\tau\big] \leq \gamma_N + \epsilon_N.$$

Under the assumption that L_n is strongly convex (e.g. regularized square loss), as $N \to \infty$, γ_N goes to zero. Hence the filtering error of $\hat{F}$ is upper bounded by the minimum batch training error that could be achieved by doing regression on D_N within class $\mathcal{F}$. Previous approaches such as Inference Machines (Ross et al., 2011b) and DAD (Theorem 3.2.1) have a similar upper bound by using DAgger for optimizing their multi-step objectives. In general the term ϵ_N depends on the noise of the data and the expressiveness of the hypothesis class $\mathcal{F}$.

Note that though we present Algorithm 6 using FTL to update the hypothesis, we can use other no-regret online algorithms such as *Online Gradient Descent* (Zinkevich, 2003), *Online Frank-Wolfe* (Hazan and Kale, 2012) to update F_n. Depending on the choice of algorithm, we can even relax the strong convexity assumption on L_n to convexity.

The finite sample analysis from (Ross et al., 2011b) can also be applied to PSIM. Let us define

$$\hat{\epsilon}_N = \min_{F\in\mathcal{F}} \frac{1}{N}\hat{L}_n(F) \tag{6.15}$$

$$\hat{\gamma}_N \geq \frac{1}{N}\sum_{n=1}^{N} \hat{L}_n(F_n) - \min_{F\in\mathcal{F}} \frac{1}{N}\sum_{n=1}^{N} \hat{L}_n(F) \tag{6.16}$$

we have:

Corollary 2. *(Ross et al., 2011a) For Algorithm 6, there exists a predictor* $\hat{F} \in \{F_n\}_{n=1}^{N}$ *such that with probability at least* $1-\delta$*:*

$$\begin{aligned} L(\hat{F}) = \mathbb{E}_\tau\big[\mathbb{E}_{z,f\sim\omega_{\hat{F},\tau}}(d(\hat{F}(z), f))|\tau\big] &\leq \hat{\gamma}_N + \hat{\epsilon}_N \\ &+ L_{\max}(\sqrt{\frac{2\ln(1/\delta)}{MN}}). \end{aligned} \tag{6.17}$$

6.5 Experimental Evaluation

We evaluate PSIMs on an illustrative synthetic example as well as a variety of dynamical system benchmarks. We use two feature functions:

$$\phi_1(f_t) = [x_t, ..., x_{t+k-1}] \tag{6.18}$$
$$\phi_2(f_t) = [x_t, ..., x_{t+k-1}, {x_t}^2, ..., x_{t+k-1}^2]. \tag{6.19}$$

ϕ_1 stacks the k future observations together. The resulting predicted messages $\hat{m}$ can then be regarded as a prediction of future k observations $(\hat{x}_t, .., \hat{x}_{t+k-1})$). ϕ_2 includes the diagonal-elements of the second moments as a Gaussian distribution approximating the true distribution of future observations. To measure the error on the computed predictive states are, we extract $\hat{x}_i$ from $\hat{m}_t$ and evaluate the squared error $\|\hat{x}_i - x_i\|_2^2$ between the predicted observation $\hat{x}_i$ and the corresponding true observation x_i.

For the dynamical system benchmarks, we present various PSIM results:

Method	Description
PSIM-Linear$_d$	DAgger training with linear ridge regression as the underlying learn procedure.
PSIM-RFF$_d$	DAgger training with linear ridge regression on on the original observation space plus Random Fourier Features (RFF) (Rahimi and Recht, 2007). With RFF, PSIM approximately embeds the distribution of f_t into a Reproducing Kernel Hilbert Space (RKHS).
PSIM-Linear$_b$	Back-propagation-through-time (BPTT) for linear regression. BPTT is used to optimize the multi-step error as an alternative for DAgger.
PSIM-RFF$_b$	BPTT for linear regression on RFF.
PSIM-Linear$_{d+b}$	Initialization with DAgger training followed by BPTT for linear regression.

Table 6.1: Various implementations of PSIM tested on the dynamical systems.

We compare the PSIM variations in Table 6.1 to several baselines: Autoregressive models (**AR**), Subspace State Space System Identification (**N4SID**) (Van Overschee and De Moor, 2012), PSRs implemented with **IVR** (Hefny et al., 2015a), and give some results with comparisons to a Recurrent Neural Network (**RNN**) trained with BPTT (LeCun et al., 2015).

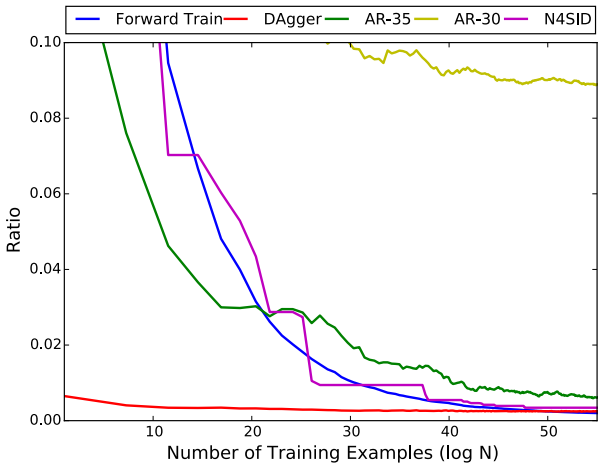


Figure 6.5: The convergence rate of different algorithms. The ratios (y-axis) are computed as $\log(\frac{e}{e_F})$ for error e from corresponding algorithms. The x-axis is computed as $\log(N)$, where N is the number of trajectories used for training.

6.5.1 Toy Example: Synthetic Linear Dynamical System

Before delving into the PSIM benchmark results, we first showcase a simple filtering example on a known linear dynamical system of the form

$$\begin{aligned} s_{t+1} &= As_t + \epsilon_s, \;\; \epsilon_s \sim \mathcal{N}(0, Q), \\ x_t &= Cs_t + \epsilon_x, \;\; \epsilon_x \sim \mathcal{N}(0, R), \end{aligned} \tag{6.20}$$

The system was designed to be exactly $k = 2$ observable. The sequences of observations are collected from the linear stationary Kalman filter of the LDS (Boots, 2012; Hefny et al., 2015a)

Since the data is collected from the stationary Kalman filter of the 2-observable LDS, we set $k = 2$ and use $\phi_1(f_t) = [x_t, x_{t+1}]$. Note that the 4-dimensional predictive state $\mathbb{E}[\phi_1(f_t)|h_t]$ will represent the exact conditional distribution of observations (x_t, x_{t+1}) and therefore is equivalent to $P(s_t|h_{t-1})$ (see the detailed case study for LDS in Appendix). With linear ridge regression, we test PSIM with forward training, PSIM with DAgger, and AR models (AR-k) with different length histories k_h. Note that the AR model uses *past* observation for its features where as PSIM uses a *belief over future* observations. For each method, we compare the average filtering error e to e_F which is computed by using the underlying linear stationary filter F of the LDS.

The convergence results are shown in Fig. 6.5 as the number of training trajectories N increases. While we tested AR models with $k_h = 5$, $k_h = 10$, and $k_h = 20$, the error with these models exceeds the axis limits of Fig. 6.5. PSIM with DAgger performs much better with few training data while Forward Training eventually slightly surpasses DAgger with sufficient data. The AR-k

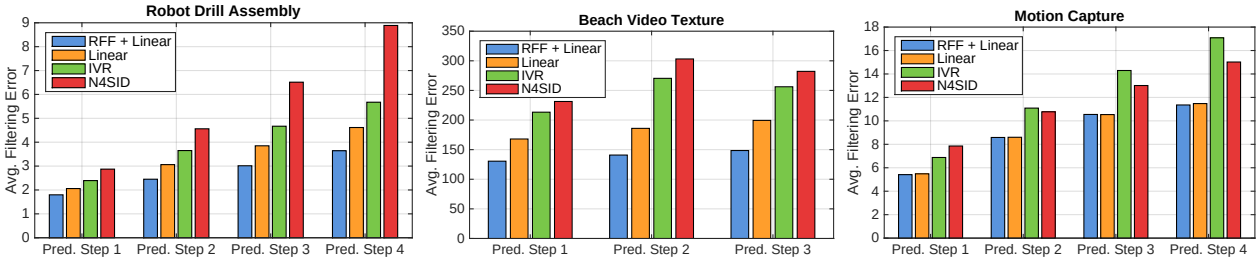


Figure 6.6: PSIM vs. Baselines. Filter error for multiple look ahead steps for the future predictions shown for a few of the datasets. We see across datasets that the performance of both IVR and N4SID are significantly worse than using PSIM. For some datasets, the nonlinearity of the Random Fourier Features helps to improve the performance.

models need long histories to perform well given data generated by latent state space models, even for this 2-observable LDS. Note AR-35 performs regression in a 70-dimensional (35×2-dimensional observations) feature space (35 past observations), while PSIM only uses 6-d features ($k = 2 \times 2 = 4$-d predictive state + 2-d current observation). This shows that *predictive state* is a more compact representation of the history – closer to "state" – and can reduce the complexity of learning problem.

6.5.2 Dynamical System Benchmarks

We consider the following three real dynamical systems:

1. *Robot Drill Assembly*: the dataset consists of 96 sensor telemetry traces, each of length 350, from a robotic manipulator assembling the battery pack on a power drill[5]. The 13 dimensional noisy observations consist of the robot arm's 7 joint torques as well as the the 3D force and torque vectors as measured from a noisy sensor located in the wrist of the robotic arm. Note the fixed higher level control policy for the drill assembly task is not given in the observations and must be learned as part of the dynamics.

2. *Human Motion Capture*: The dataset consists of 48 skeletal tracks of 300 timesteps each from a Vicon motion capture system from three human subjects performing walking actions[6]. The observations consist of the 3D positions of the various skeletal parts (e.g. upperback, thorax, clavicle, etc.); Due to very high correlation in some of the observation dimensions, we precompute a projection matrix from the eigenvectors of the first trajectory.

[5]Collected from the ARM-S system (Bagnell et al., 2012)
[6]Available at `http://mocap.cs.cmu.edu`

3. *Video Textures*: We also test two different video texture datasets, one video of a *flag* waving (e.g. Fig. 3.5) and the other one of waves on a *beach* (e.g. Fig. 3.6). More details can be found in Section 3.3.4.

Method	Robot Drill	Motion Cap.	Beach Video	Flag Video
N4SID	2.87	7.86	231.33	3.38e3
IVR	2.39	6.88	213.27	3.38e3
RNN$_b$	1.99	9.60	346.00	–
PSIM-Linear$_d$	2.15	5.75	164.23	1.28e3
PSIM-Linear$_b$	2.54	9.94	268.73	1.31e3
PSIM-Linear$_{d+b}$	*2.09*	*5.66*	*164.08*	–
PSIM-RFF$_d$	**1.80**	**5.41**	**130.53**	**1.24e3**
PSIM-RFF$_b$	2.54	9.26	202.10	–
Traj. Pwr	27.90	107.92	873.77	3.73e3

Table 6.2: Filter error (single-step ahead) for various methods and datasets. All PSIM approaches use the ϕ_1 message type. PSIM with DAgger with both RFF+Linear or Linear Regression outperforms most baselines. For a scale on the error, we also give the average trajectory power for the true observations from each dataset.

We do not test PSIM with Forward Training since some our benchmarks have a large number of time steps per trajectory. Throughout the experiments, we set $k = 5$ for all datasets except for video textures, where we set $k = 3$. For each dataset, we randomly pick a small number of trajectories as a validation set for parameter tuning (e.g., ridge, rank for N4SID and IVR, band width for RFF). For both N4SID and IVR, we only perform grid search for picking the rank while using normal least squares regression. We partition the whole dataset into ten folds, train all algorithms on 9 folds and test on 1 fold.

For the feature function ϕ_1, the average one-step filtering errors across the ten folds are shown in Table 6.2. Our approaches outperforms the three baselines (N4SID, IVR, RNN) across all tested datasets. Since the datasets are generated from complex dynamics, PSIM with RFF exhibits better performance than PSIM with Linear. This experimentally supports our theorems suggesting that with powerful regressors, PSIM could perform better. DAgger (20 iterations) trains a better linear regression for PSIM than back-propagation with random initialization (400 epochs), likely due to local optimality. PSIM Linear with DAgger +Backprop fine tunes a model initialized from DAgger training to achieve marginal performance improvement over DAgger. Our results show that PSIM with DAgger finds good models by itself. We also compare some of these approaches for multi-step look ahead (Fig. 6.6). PSIM consistently outperforms the two baselines.

To show predictive states with larger k encode more information about latent states, we additionally run PSIM with $k = 1$ using ϕ_1. The results are show in Table 6.3. Including belief over longer futures into predictive states helps capture more information and increase the performance.

Dataset	**% Improvement**
Robot Drill Assembly	5 %
Motion Capture	6 %
Beach Video	32 %
Flag Video	8 %

Table 6.3: Relative performance improvement using $k = 5$ vs. $k = 1$ for PSIM

We next investigate how the performance varies how using second moments can affect performance The results are show in Table 6.4 for feature function ϕ_2 (first and second-moment features) and $k = 5$ using PSIM with linear ridge regression and DAgger training. Comparing to the results with ϕ_1, using ϕ_2 achieves slightly better performance on all datasets, and noticeably better performance on the beach video texture.

Dataset	**PSIM +** ϕ_1	**PSIM +** ϕ_2
Robot Drill Assembly	2.15	2.05
Motion Capture	5.75	5.47
Beach Video	164.23	154.02
Flag Video	1.28e3	1.27e3

Table 6.4: PSIM with linear regression using first-moment, ϕ_1, and first+second-moment features ϕ_2.

6.6 Conclusion

We introduced Predictive State Inference Machines, a novel approach fore directly learn to filter. Leveraging ideas from PSRs, PSIM reduces the unsupervised learning of latent state space models to a supervised learning setting. Our approach gives theoretical guaranties on the filtering performance for general non-linear models in both the realizable and model-agnostic settings. The filter function learned by PSIM can be considered a dynamical system model specialized for filtering performance on partially observable systems (Challenge 2). In the next chapter, we introduce an extension that allows us to use the power of predictive states in problems with labeled partial-state or target information, which we call "hints".

Chapter 7

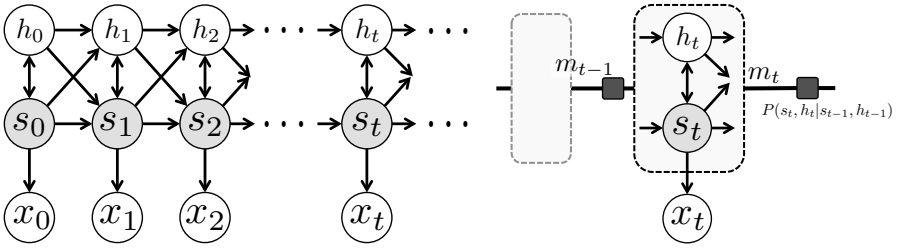
Predicting "useful" quantities: Extending PSIM with Hints

The standard PSIM (Chapter 6) address prediction over the space of future observations. Many real world applications, however, require the estimation or prediction of useful physical quantities, which traditional filtering algorithms are capable of but PSIMs were not. For example, given observation of a neural spikes from a brain implant, the filter must be able to predict prosthetic motion commands (Muelling et al., 2015).

In this chapter, we continue to address Challenge 2, extending PSIM to work with these real-world targets which we term "hints". Our extension, PSIM+Hints adds this vital capability. PSIM as introduced, developed an inference machine approach in the **latent-state** setting where the sufficient underlying state representation is at least partially *unobserved* at training time. In this chapter, we also consider the **supervised-state** setting in which we wish to learn a filter model for a system with a known state representation for which we are able to obtain ground truth at training time. We show the supervised-state is a special case of the latent-state PSIM. The supervised-state filters, or Inference Machine Filters (IMFs) can be considered a specialization of (Ross et al., 2011b) for the time-series setting. We use the feature embedding trick from PSIM to handle continuous-valued observations.

To ground the difference between *supervised-state* and *latent-state*, we consider simple pendulum dynamics (e.g. Fig. 3.2). In the supervised-state setting, we assume access at training to a sufficient-state representation of the system. For the pendulum the natural and sufficient state representation is the angle and angular velocity $(\theta, \dot{\theta})$. There exist other sufficient state representations

This work was originally presented in *Inference Machines for Nonparametric Filter Learning* at IJCAI 2016 (Venkatraman et al., 2016c) contains joint work done in collaboration with Wen Sun and Byron Boots.



(a) "Hints" Model (b) Message passing in the "Hints" Model

Figure 7.1: **(a)**: Adding hints, h_t, allow us to extend the HMM model with partially observed states (labels). The true latent-states s_t of an unknown representation generate observations x_t and labels h_t of which both are observed at training time but only the former at inference time. The state s_t and hint h_t together generate the next label h_{t+1}. **(b)**: If we do not need the messages passed between the states and hints, we can abstract them away and consider the net message output by the hint and state before the process model, drawn as the black square factor, but after the observation update.

such as the Euclidean position and velocities $(x, y, \dot{x}, \dot{y})$ which can be used to predict the future exactly. Any subset of each would *not* be a sufficient state and would not have enough information to predict the next state with certainty. If we have any of these components as observations or as the "hints", this would be the *latent-state* setting.

7.1 Predictive State Inference Machine with Hints

In this chapter, we extend PSIMs to models with partially observable states or side-information (Fig. 7.1(a)). In the semi-supervised setup, the hints h and observations x are at training time though the true sufficient-state s is either unknown or unobserved. Although PSIM is well defined on a simple chain-structured model (e.g. Fig. 7.2), it is not straightforward to extend PSIM to a model with the complicated latent structure in Fig. 7.1(a).

To handle this type of graph structure, we collapse the hints h and the latent states s into a single unit-configuration as shown in the abstracted factor graph, Fig. 7.1(b), and only focus on the net message passed into the configuration and the net message passed out from the configuration. Ideally, we would like to design an inference machine that mimics this net message passing procedure.

In Fig. 7.1(b), m_t represents the joint belief of h_t and s_t,

$$m_t = P(h_t, s_t | p_t). \tag{7.1}$$

Note that we changed notation compared to Chapter 6. h_t now refers to the hint while p_t refers to the history of observations. Directly tracking these messages is difficult due to the existence of latent state s_t. Following the approach of PSRs

Algorithm 7 PSIM+HINTS with DAgger Training

Input: M independent trajectories τ_i, $1 \leq i \leq M$;

Initialize $D \leftarrow D_0 \leftarrow \emptyset$
Initialize F_0 to be any hypothesis in $\mathcal{F}$;
Initialize $\hat{m}_1 = \frac{1}{M}\sum_{i=1}^{M} \phi(h_1^i, f_1^i = x_{1:k}^i)$
for $n = 0$ to N **do**
    Roll out F_n to perform belief propagation on trajectory τ_i, $1 \leq i \leq M$
    Create dataset D_n: $\forall \tau_i$, add predicted message with observation $z_t^i = (\hat{m}_{t-1}^{i,F_n}, x_t^i)$ encountered by F_n to D_n as feature variables (inputs) and the corresponding $\phi(h_t^i, f_t^i)$ to D_n as the learning targets
    DAgger Step: aggregate dataset, $D = D \cup D_n$
    Train a new hypothesis F_{n+1} on D to minimize the loss $d(F(m, x), \phi(h, f))$
end for
return Best hypothesis $\hat{F} \in \{F_i\}_{i=0}^{N}$ on validation trajectories.

and PSIMs, we use observable quantities to represent the belief of h_t and s_t. Since the latent state s_t affects the future observations starting from x_t and also the future partial states starting from h_t, we use sufficient features of the joint distribution of future observations and hints to encode the information of s_t in message m_{t-1}. Similar to PSIM, we assume that there exists an underlying mapping between the following two representations:

$$P(h_t, s_t|p_t) \Leftrightarrow P(h_t, x_{t+1:t+k_f}|p_t). \tag{7.2}$$

Assuming that ϕ computes the sufficient features (e.g., first and second moments for a Gaussian), we can represent $P(h_t, x_{t+1:t+k_f}|p_t)$ by the following conditional expectation:

$$\mathbf{E}\left[\phi(h_t, x_{t+1:t+k_f})|p_t\right]. \tag{7.3}$$

When ϕ is rich enough, $\mathbf{E}\left[\phi(h_t, x_{t+1:t+k_f})|p_t\right]$ is equivalent to the distribution $P(h_t, x_{t+1:t+k_f}|p_t)$. For example, if ϕ is a kernel mapping, $\mathbf{E}\left[\phi(h_t, x_{t+1:t+k_f})|p_t\right]$ essentially becomes the RKHS embedding of $P(h_t, x_{t+1:t+k_f}|p_t)$. We call Eq. (7.3) as the predictive state[1]. For notational simplicity, define

$$m_t = \mathbf{E}\left[\phi(h_t, x_{t+1:t+k_f})|p_t\right].$$

We are then capable of training an inference machine that mimics the following predictive state flow:

$$m_t = F(m_{t-1}, x_t), \tag{7.4}$$

which takes the previous predictive state m_{t-1} and current observation x_t and outputs the next predictive state m_t.

[1]The choice of only one hint at each timestep is by assumption. For some applications, we may see improved performance with a belief over more than one future hint.

Define $\tau \sim \mathcal{D}$ as a trajectory $\{x_1, h_1, ..., x_T, h_T\}$ of observations and hints sampled from an unknown distribution $\mathcal{D}$. Given a function $F \in \mathcal{F}$, let us define $m_t^{\tau,F}$ as the corresponding predictive state generated by F when rolling out using the observations in τ described by Eq. 7.4.

Similar to PSIM, we aim to find a good hypothesis F from hypothesis class $\mathcal{F}$, that can approximate the true messages well. We define the multi-step (recursive) predictive objective:

$$\min_{F \in \mathcal{F}} \frac{1}{T} \mathbf{E}_{\tau \sim \mathcal{D}} \left[\sum_{t=1}^{T} d\big(F(\hat{m}_{t-1}, x_t), (h_t^\tau, x_{t+1:t+k_f}^\tau)\big) \right],$$
$$\text{s.t., } \hat{m}_t = F(\hat{m}_{t-1}, x_t), \forall t, \tag{7.5}$$

where d is the loss function (e.g., the squared loss).

As with PSIM, the optimization objective presented in Eq. (7.5) is non-convex in F due to the objective involving sequential prediction terms where the output of F from time-step $t-1$ is used as the input in the loss for the next timestep t. As optimizing Eq. (7.5) directly is impractical, we utilize *Dataset Aggregation* (DAgger) (Ross et al., 2011a) training in a similar fashion to DAD Chapter 3 and PSIM Chapter 6 to find a filter function (model) F with *bounded* error. The training procedure for PSIM+HINTS is detailed in Algorithm 7. By rolling out the learned filter and collecting new training points (lines 6, 7), subsequent models are trained (line 9) to be robust to the induced message distribution caused by sequential prediction.

Specifically, let us define $z_t^\tau = (h_t^\tau, x_{t+1:t+k_f}^\tau)$ for any trajectory τ at time step t and define d_F as the joint distribution of $(\hat{m}_{t-1}^{\tau,F}, x_t^\tau, z_t^\tau), \forall t$ when rolling out F on trajectory τ sampled from $\mathcal{D}$. Then our objective function can be represented alternatively as $\mathbf{E}_{(m,x,z)\sim d_F} d(F(m,x), z)$. Alg. 7 guarantees to output a hypothesis $\hat{F}$ such that:

$$\mathbf{E}_{\tau \sim \mathcal{D}} \frac{1}{T} \sum_{t=1}^{T} d\big(\hat{F}(\hat{m}_{t-1}^{\tau,\hat{F}}, x_t^\tau), z_t^\tau\big) \leq \epsilon \tag{7.6}$$

$$\text{where } \epsilon = \min_{F \in \mathcal{F}} \frac{1}{N} \sum_{n=1}^{N} \mathbf{E}_{(m,x,z)\sim d_{F_n}} d(F(m,x), z), \tag{7.7}$$

ϵ is the minimum batch minimization error from the entire *aggregated dataset* in hindsight after N iterations of Algorithm 7. Note that this bound applies to Eq. (7.5) for the F found by Algorithm 7.

Despite the learner's loss ϵ in Eq. (7.7) being over the aggregated dataset, it can be driven low (e.g. with a powerful learner), making the bound useful in practice. For long-horizons, the possible exponential rollout error (Theorem 3.1.1) from optimizing only the one-step error often dominates the error over the aggregated dataset.

We conclude with a few final notes. First, even though F_0 would ideally be initialized (line 3) by optimizing for the transition between the true messages,

in practice F_0 is often initialized from the empirical estimates. Second, though the sufficient-feature assumption is common in the PSR literature (Hefny et al., 2015a), an approximate feature transform ϕ balances computational complexity with prediction accuracy: simple feature design (e.g., first moment) makes for faster training of F while Hilbert Space embeddings are harder to optimize but may improve accuracy (Boots, 2012; Boots et al., 2013). Additionally, the bound in Eqs. (7.6) and (7.7) holds for the approximate message statistics. Finally, though the learning procedure is shown in a follow-the-leader fashion on the batch data (line 9), we can use any online no-regret learning algorithm (e.g. OGD (Zinkevich, 2003)), alleviating computational and memory concerns.

7.1.1 Hint Pre-image Computation

Since the ultimate goal is to predict the hint, we need to extract a prediction of h_t from the computed predictive state $\hat{m}_t$, which is an approximation of the true conditional distribution $P(h_t, x_{t+1:t+k_f}|p_t)$. Exactly computing the MLE or mean of h_t from $\hat{m}_t$ might be difficult (e.g., sampling points from a Hilbert space embedding is not trivial (Chen et al., 2012)). Instead, we formulate the step of extracting h_t from $\hat{m}_t$ as an additional regression problem:

$$\min_{g \in \mathcal{G}} \mathbf{E}_{\tau \sim \mathcal{D}} \frac{1}{T} \sum_{t=1}^{T} \|g(\hat{m}_t^\tau) - h_t^\tau\|_2^2 \tag{7.8}$$

To find g, we roll out $\hat{F}$ on each trajectory from the training data and collect all the pairs of $\hat{m}_t^\tau, h_t^\tau$. Then we can use any powerful learning algorithm to learn this pre-image mapping g (e.g., random forests, kernel regression). Note that this is a standard supervised learning problem and not a sequential prediction problem — the output from g is not used for future predictions using g or the filter function $\hat{F}$.

There is another hypothesis as to why this is useful. In Chapter 6, we took the predicted value from the first moment portion of the message $\hat{m}_t$ to get the predicted observation. However, this prediction is the output of a filter function $\hat{F}$ optimized using DAgger style training for multi-step performance. The pre-image computation we do here is instead formulated as a standard supervised learning procedure. This step may be useful a "DAgger-cleanup" to utilize a learner's full model complexity for the singe-step performance.

7.2 The Inference Machine Filter for Supervised-State Models

In contrast to filter learning in the latent-state problem, the supervised-state setting affords us access to both the sufficient-state s and observations x during training time. We specifically look at learning a filter function on Hidden Markov Models (HMMs) as shown in Fig. 7.2. This problem setting sim-

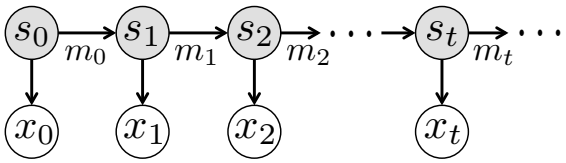


Figure 7.2: Message Passing on a Hidden Markov Model (HMM). The state of the system s_t generates an observation x_t. In the supervised-state setting, the sufficient-states s and observations x are given at training time though only observations are seen at test time. In the latent-state problem setting, we additionally do not have access to the sufficient state at training time – it may be unobserved or have an unknown representation.

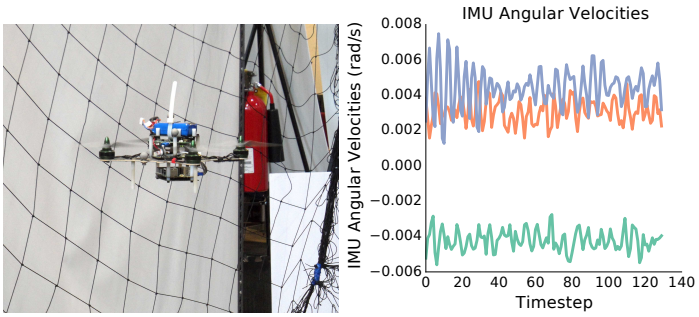
ilar to those considered in the learning-based system identification literature (e.g. (Abbeel et al., 2005a; Ko and Fox, 2009; Nishiyama et al., 2016)). However, many of these previous methods use supervised learning to learn independent dynamics and observation models and then use these learned models for inference. This approach may result in unstable dynamics models and poor performance when used for filtering and cascading prediction. As before, we directly optimize filtering performance (Challenge 2) by adapting PSIM+HINTS to learn message passing in this supervised setting. We term this simpler algorithm as the INFERENCE MACHINE FILTER (IMF).

Referencing Algorithm 7, the IMF simply sets the hints h_t in PSIM+HINTS to be the observed states s_t and sets the number of future observations $k_f = 0$; in other words, s_t is assumed to be a sufficient-state representation. The IMF learns a deterministic filter function F that combines the predict-and-update steps of a Bayesian filter to recursively compute:

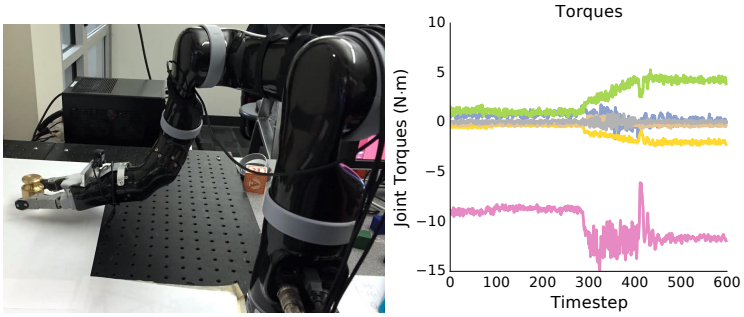
$$P(s_t|p_t) \Leftrightarrow \mathbf{E}\left[\phi(s_t)|p_t\right] = m_t = F(m_{t-1}, x_t) \tag{7.9}$$

The INFERENCE MACHINE FILTER (IMF) can be viewed as a specialization of both the theory and application of inference machines to the domain of time-series hidden Markov models. Our guarantee in Eq. (7.6) shows that the prediction error on the messages optimized by DAgger is bounded linearly in the filtering problem's time horizon. Additionally, the sufficient feature representation of PSIM+HINTS allows IMFs to represent distributions over continuous variables, compared to the discrete tabular setting of (Ross et al., 2011b).

The IMF approach differs from the approach of Langford et al. (2009) in several important ways. The authors proposed a method that learns four operators: one to map the first observation to initial state, one for the belief update with an observation, one for state-to-state transition, and one for state to observation. This results in a more complex learning procedure as well as a special initialization procedure at test time for the filter. Our algorithm learns a single filter function instead of four. It also operates like a traditional filter; it is initialized from a prior over belief state instead of mapping from the first observation to the first state. Finally, Algorithm 7 does not assume differentia-



(a) Quadrotor Hovering



(b) Kinova Mico Grasping

Figure 7.3: Many real-world systems rely on the ability to utilize observations from noisy sensors in order to update a belief over some component of state. For example, the attitude of a quadrotor from linear accelerations and angular velocities (left) or the mass of a grasped object from the robot's joint positions and applied torques (right).

bility of the learning procedure as required for backpropagation-through-time used in (Langford et al., 2009).

7.3 Experiments

We focus on robotics-inspired filtering experiments. In the supervised-state representation, we are given the state (e.g. robot's pose) and observations (e.g. IMU readings) at training time. In the latent-state setting, we only gain access to the observations and instead of observing the full state at training time, we see only a hint (e.g. only the x-position of the pose). In both scenarios, the hint or state could be collected by instrumenting the system at training time (e.g. having the robot in a motion capture arena), which we then do not have access to at test time (e.g. robot moves in an outdoor area). The latent-state setting with hints is additionally relevant in domains where it is difficult to observe the full state but easy to observe quantities that are heavily correlated with it.

7.3.1 Baselines

We compare our approach against both learning-based algorithms as well as physics based, hand-tuned filters when relevant. For the first baseline, we compare against a learned linear Kalman filter (**Linear KF**). Here, the hints h are the states X for the Kalman filter and Y are the observations. We learn the Kalman filter using the MAP estimate:

$$\begin{aligned}
A &= \arg\min_A \|AX_t - X_{t+1}\|_2^2 + \beta_1 \|A\|_F^2 \\
C &= \arg\min_C \|CX_t - Y_t\|_2^2 + \beta_2 \|C\|_F^2 \\
Q &= \mathrm{cov}(AX_t - X_{t+0}), \quad R = \mathrm{cov}(CX_t - Y_t) \\
\mu_0 &= \mathrm{mean}(X_0), \quad \Sigma_0 = \mathrm{cov}(X_0)
\end{aligned}$$

We select the regularization (Gaussian prior) terms β_1, β_2 by cross-validation.

We also compare against a model that uses a fixed-sized history k_p of observations to predict the hint at the next time step. We find this model by optimizing the objective,

$$\textsc{Ar} = \arg\min_{\textsc{Ar}} \sum_{t=k_p}^{T-1} \left\| \textsc{Ar}([y_{t-k_p}, \ldots, y_t]) - h_t \right\|_2^2,$$

where h_t is the hint we wish to predict at timestep t, $y_{t-k_p}, \ldots, y_t$ are past observations, and Ar is the learned function. This baseline is similar to Autoregressive (**AR**) Models (Wei, 1994). It is important to note that using a *past* sequence of observations is *different* than tracking a belief over the *future* observations (the predictive state) as PSIM does. The AR model does not marginalize over the whole history as a Bayesian filter would. In our experiments, we set the history (past) length $k_p = 5$. Choosing higher k_p reduces the comparability of the results as the AR model has to wait k_p timesteps before giving a prediction while the other filter algorithms predict from the first timestep. To get good performance, we chose the Ar model to be Random Fourier Features (RFF) regression (Rahimi and Recht, 2007) with hyper-parameters tuned via cross-validation.

Finally, on several of the applicable dynamics benchmarks below, we also compare against a hand-tuned filter for the problem. The overall error is reported as the mean L2 norm error $\frac{1}{T}\sum_{t=1}^{T} \|\hat{h}_t - h_t\|^2$.

7.3.2 Dynamical Systems

We describe the experimental setup below for each of the dynamical system benchmarks we use. A simulated pendulum is used to show that the inference machine is competitive with, and even outperforms, a physics-knowledgeable baseline on a sufficient-state representation (Table 7.1). This simulated dataset additionally illustrates the power of using predictive state when we only access a partial-state to use as a hint. Two real-world datasets show the applicability of our algorithms on robotic tasks. The numerical results are computed across a k-fold validation ($k = 10$) over the trajectories in each dataset. We

Algorithm	Observation Length	Full State Est. $s = h \equiv (\theta, \dot{\theta})$	Partial State Est. $s = h \equiv (\theta)$
Physics UKF	–	1.22 ± 1.2	*N/A*
AR	$k_p = 5$	2.96 ± 1.5	1.60 ± 1.5
Linear KF	–	4.67 ± 0.98	1.81 ± 1.6
IMF	–	0.577 ± 0.33	1.43 ± 1.3
PSIM+Hints	$k_f = 5$	0.554 ± 0.33	1.27 ± 1.0
	$k_f = 10$	0.549 ± 0.32	0.888 ± 0.78
	$k_f = 20$	$\mathbf{0.544 \pm 0.31}$	$\mathbf{0.758 \pm 0.68}$

Table 7.1: Mean L2 Error $\pm 1\sigma$ for Pendulum Full State $(\theta, \dot{\theta})$ and Partial State (θ) Estimation from observations of the Cartesian x position of the pendulum. Notice that when the full-state is given, the performance of PSIM and IMF are similar; increasing k_f for PSIM+Hints does not significantly improve its performance. However, when the hint defines only a partial representation ($s = h \equiv \theta$), we achieve significantly better results using PSIM+Hints.

use linear regression or Random Fourier Feature (RFF) regression to learn the message passing function F for PSIM+Hints and IMF and report the better performing result. Random forests (Breiman, 2001) or RFF regression are used to learn the pre-image map g, chosen by cross-validation over that k-fold's training trajectories.

Pendulum State Estimation In this problem, the goal is to estimate the sufficient state $s_t = h_t = (\theta, \dot{\theta})_t$ from observations x_t of the Cartesian coordinate of the mass at the end of pendulum. For PSIM+Hints and IMF we use a message that approximates the first two moments. This is accomplished by stacking the state with its element-wise squared value, with the latter approximating the covariance by its diagonal elements. IMF does this for only the state while PSIM+Hints does this for the hint (state) and the future observations. Since we know the dynamics and observation models explicitly for this dynamical system, we also compare against a baseline that can exploit this domain knowledge, the Unscented Kalman Filter (**Physics UKF**) (Wan and Van Der Merwe, 2000). The process and sensor models given to the UKF were the exact functions used to generate the training and test trajectories.

Pendulum Partial State Estimation To illustrate the utility of tracking a predictive state, consider the same simulated pendulum where we take the partial state, θ_t as the hint h_t for PSIM+Hints and use as the (insufficent) state s_t for the IMF. We use the first and approximate second moment features to generate the messages. On this benchmark, we do not compare against a UKF physics-model since the partial state is not sufficient to define a full

Algorithm	**Mean L2 Error** $\pm 1\sigma$
Complementary Filter	0.0167 ± 0.011
AR ($k_p = 5$)	0.0884 ± 0.063
Linear KF	0.0853 ± 0.066
IMF	0.037 ± 0.0305
PSIM+HINTS ($k_f = 5$)	$\mathbf{0.0136 \pm 0.017}$

Table 7.2: Quadrotor Attitude Estimation Performance

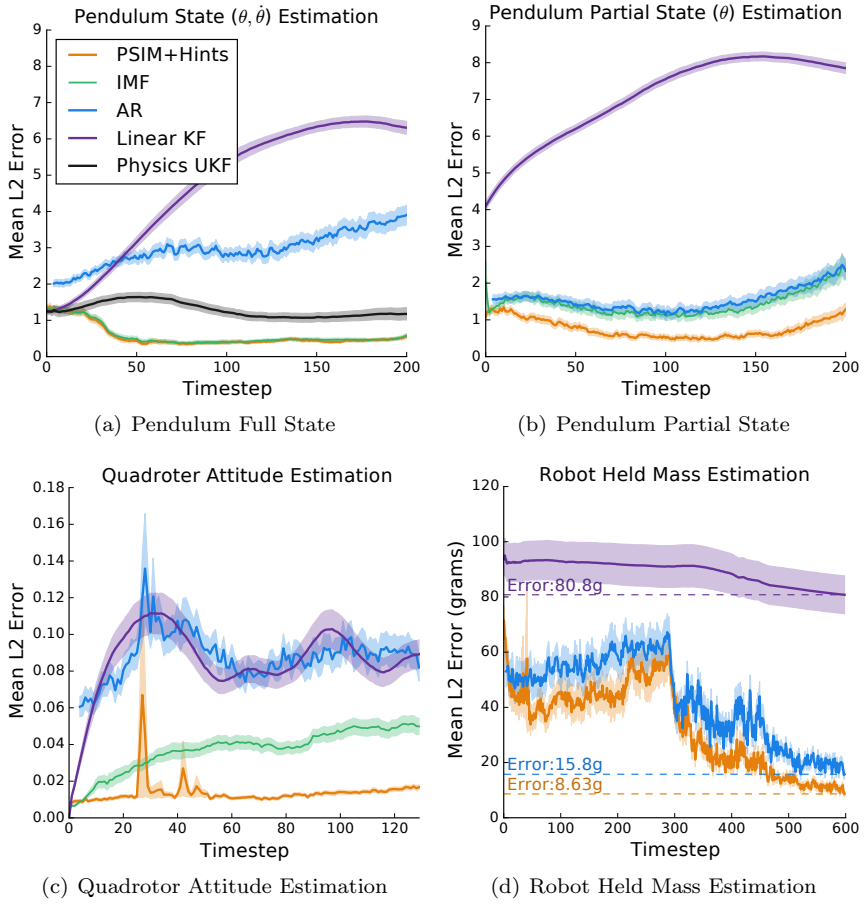
Algorithm	**Mean L2 Error** $\pm 1\sigma$
AR ($k_p = 5$)	42.82 ± 19.62
Linear KF	89.13 ± 52.22
PSIM+HINTS ($k_f = 40$)	$\mathbf{32.77 \pm 14.09}$

Table 7.3: Performance on mass estimation task

process model of the system's evolution.

Quadrotor Attitude Estimation In this real-world state-estimation problem, we look to estimate the attitude of a quadrotor in hover under external wind disturbance. The quadrotor runs a hover controller in a Vicon capture environment, shown in Fig. 7.3(a). We use the Vicon's output as the ground truth for the roll and pitch of the quadrotor and use the angular velocities and acceleration measurements from an on-board IMU as the observations. As an application specific baseline, we compare against a Complementary Filter (Hamel and Mahony, 2006) hand-tuned by domain experts. We use only first moment features for the messages in PSIM+HINTS and first and approximate second moment features for IMF messages.

Mass Estimation from Robot Manipulator This dataset tests the filter performance at a parameter estimation task where the goal is to estimate the mass carried by the robot manipulator shown in Fig. 7.3(b). Each trajectory has the robot arm lift an object with mass 45g-355g. The robot starts moving at approximately the halfway point of the recorded trajectories. We use as observations the joint angles and joint motor torques of the manipulator. Only first moment features are used for the messages for PSIM+HINTS. With this experiment, we show that that filtering helps reduce error compared to using simply a sequence of past observations (AR baseline) even on a problem of estimating a parameter held static per test trajectory.



(a) Pendulum Full State

(b) Pendulum Partial State

(c) Quadrotor Attitude Estimation

(d) Robot Held Mass Estimation

Figure 7.4: Mean L2 Error ± 1 Standard Error versus filtering time. The AR model in each was set with $k_p = 5$. See results tables for k_f parameter values for PSIM+HINTS.

7.4 Discussion

In all of the experiments, IMF and PSIM+HINTS outperform the baselines. Table 7.1 (left column) shows that the IMF and PSIM+HINTS both outperform the baseline Unscented Kalman Filter which uses knowledge of the underlying physics and noise models. We do not compare against a learned-model UKF, such as the Gaussian Process-UKF (Ko and Fox, 2009), because any learned dynamics and observation models would be less accurate than the *exact* ones in our Physics UKF baseline. For fair comparison, the UKF, Linear KF, IMF, and PSIM+HINTS all start with empirical estimates of the initial belief state (note the similar error at the beginning of Fig. 7.4(a)). We believe that the

IMF and PSIM+HINTS outperforms the Physics UKF for two reasons: (1) Inference machines do not make Gaussian assumptions on the belief updates as the UKF does, (2) The large variance for the UKF (Table 7.1) shows that it performs well on some trajectories. We qualitatively observed this variance is heavily correlated with the difference between the UKF's initial belief-state evolution and the true states. Our proposed inference machine filter methods instead directly optimize the filter's performance over all of the time-horizon are are thus more robust to the initialization of the filter .

The simulated pendulum example also demonstrates the usefulness of predictive representations. When a sufficient state is used (i.e. $(\theta, \dot{\theta})$ for the pendulum) for the filter's belief, similar performance is achieved using either IMF or PSIM+HINTS. Table 7.1's right column (or Fig. 7.4(b)) shows that when a partial-state representation is used instead (i.e. (θ)), PSIM+HINTS vastly outperforms IMF. Specifically, we require a larger predictive state representation (larger k_f) over the observation space in order to better capture the evolution of the system. This ablation-style experiment demonstrates the ability of PSIM+HINTS to produce more accurate filters.

Finally, our real-world dataset experiments provide additional experimental validation of inference machines for filtering. Both the IMF and PSIM+HINTS outperform baselines in Table 7.2. In Fig. 7.4(d), PSIM+HINTS is on average 50% more accurate at the end of the trajectory than the AR baseline; the average error over the whole trajectory is given in Table 7.3. For this problem, the largest information gain is when the robot starts moving halfway along the trajectory. We see less performance gain from using a filter compared to the AR baseline in this problem as the hint (mass) does not change over time as the state does in pendulum or quadrotor problems. The error versus time plots in Fig. 7.4 show the relative stability of the inference machine filters even over large time horizons.

7.5 Conclusion

In this chapter, we presented an extension to PSIM that allows for the learning of discriminative filter models that apply to a larger class of practical problems, making more progress on Challenge 2 of this thesis. The proposed algorithms show promise in many robotic applications where ground-truth information about state is available during training, for example by over-instrumenting during prototyping or calibration. We empirically validated our approaches on several simulated and real world tasks, illustrating the advantage of PSIMs+HINTS and IMFs over previous methods.

PSIM and PSIM+HINTS are reminiscent of recurrent neural network (RNN) architectures. They train a recurrent predictor that predicts a new target every step from a new observation as input along with the previous predicted message. The predicted message from PSIM is similar to the internal state of an RNN. We explore this connection in the next chapter and develop a procedure to improve the convergence and capability of general recurrent architectures.

Chapter 8

Predictive-State Decoders for General Recurrent Networks

In Chapter 6, we first introduced PSIM to directly learn a filter function that targets the inference output itself, predicting future observations. In Chapter 7, PSIM+Hints extended the approach to predicting other useful side-information which we coined as "hints". Common to both was to setup a supervised learning problem over a predictive representation that summarized past observations in order to predict the future. This same idea is key to a larger class of sequential prediction architectures – recurrent neural networks (RNNs). For RNNs, the *internal state* (sometimes called its memory) represents the summary of past information. In PSIM, we introduced a recurrent network with a specific structure that targeted a specific internal state representation.

However, an advantage for more complex recurrent architectures is that they can be used to find a better *internal state* representation if underlying representation is not known or well understood. In this chapter, we look at how advantages of PSIM can be used to improve learning of general recurrent architures. We develop Predictive-State Decoders (Psds) that can be used to better train neural networks to achieve lower error on their target tasks as well as faster convergence. While the original PSIM and PSIM+Hints focused on the filtering problem, Psds can be used on possibly arbitrary problems facing Challenge 2 with partial observability that already use a recurrent network.

This work was originally presented in *Predictive-State Decoders: Encoding the Future into Recurrent Networks* at NIPS 2017 (Venkatraman et al., 2017) and containts joint work done in collaboration with Nicholas Rhinehart, Wen Sun, Lerrel Pinto, Byron Boots, and Kris M. Kitani.

8.1 Motivation

Despite their wide success in a variety of domains, recurrent neural networks (RNNs) are often inhibited by the difficulty of learning an *internal state* representation. Internal state is a unifying characteristic of RNNs, as it serves as an RNN's memory. Learning these internal states is challenging because optimization is guided by the *indirect* signal of the RNN's target task, such as maximizing the cost-to-go for reinforcement learning or maximizing the likelihood of a sequence of words. These target tasks have a *latent state* sequence that characterizes the underlying sequential data-generating process. Unfortunately, most settings do not afford a parametric model of latent state that is available to the learner.

In lieu of ground truth access to latent states, recurrent neural networks (LeCun et al., 2015; Sutskever, 2013) employ internal states to summarize previous data, serving as a learner's memory. We avoid the terminology "hidden state" as it refers to the internal state in the RNN literature but refers to the latent state in the HMM, PSR, and related literature. Internal states are modified towards minimizing the target application's loss, e.g., minimizing observation loss in filtering or cumulative reward in reinforcement learning. The target application's loss is not directly defined over the internal states: they are updated via the chain rule (backpropagation) through the global loss. Although this modeling is indirect, recurrent networks nonetheless can achieve state-of-the-art results on many robotics (Duan et al., 2016; Hausknecht and Stone, 2015), vision (Ondruska and Posner, 2016; van den Oord et al., 2016), and natural language tasks (Chung et al., 2015; Graves and Jaitly, 2014; Ranzato et al., 2016) when training succeeds. However, recurrent model optimization is hampered by two main difficulties: 1) non-convexity, and 2) the loss does not directly encourage the internal state to model the latent state. A poor internal state representation can yield poor task performance, but rarely does the task objective directly measure the quality of the internal state.

In Chapters 6 and 7, we leveraged a key idea from the PSR literature: latent states can be characterized by *observations*. In partially-observable problems (*e.g.* Fig. 2.4), a single observation is not guaranteed to contain enough information to fully represent the system's latent state. However, a predictive representation using statistics of future observations was shown to provide a good internal state representation for learning probabilistic filter functions.

We leverage ideas from the both RNN and PSIM (PSR) paradigms, resulting in a marriage of two orthogonal sequential modeling approaches. When training an RNN, Predictive-State Decoders (Fig. 8.1) provide direct supervision on the internal state, aiding the training problem. The proposed method can be viewed as an instance of Multi-Task Learning (MTL) (Caruana, 1998) and self-supervision (Jaderberg et al., 2016), using the inputs to the learner to form a secondary unsupervised objective. Our contribution is a general method that improves performance of learning RNNs for sequential prediction problems. The approach is easy to implement as a regularize on traditional RNN loss functions with little overhead and can thus be incorporated into a variety of existing

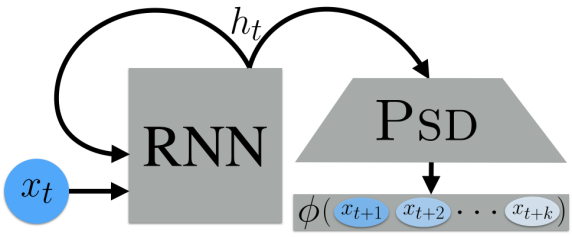


Figure 8.1: An overview of our approach for modeling the process from Fig. 2.4. We attach a decoder to the internal state of an RNN to predict statistics of future observations x_t to x_{t+k} observed at training time.

recurrent models.

In our experiments, we examine three domains where recurrent models are used to model temporal dependencies: probabilistic filtering, where we predict the future observation given past observations; Imitation Learning, where the learner attempts to mimic an expert's actions; and Reinforcement Learning, where a policy is trained to maximize cumulative reward. We observe that our method improves loss convergence rates and results in higher-quality final objectives in these domains.

Unlike traditional supervised machine learning problems, learning models for latent state problems must be accomplished without ground-truth supervision of the internal states themselves. Two distinct paradigms for latent state modeling exist. The first are discriminative approaches based on RNNs, and the second is a set of theoretically well-studied approaches based on Predictive-State Representations, such as PSIM. In the following section we provide a brief overview of RNN approaches, extending on Section 2.2.3 from Chapter 2. An overview of PSRs can be found in Section 6.1 in Chapter 6.

8.2 Recurrent Models and RNNs

Many modern approaches to learning models for partially observed systems rely on the recurrent neural network (RNN) architecture. The RNN model (Fig. 8.2). The machine learning problem is to find a model f that uses the latest observation x_t to recursively update an internal state, denoted h_t, illustrated in Fig. 8.2. **Note that h_t is distinct from s_t. h_t is the learner's internal state, and s_t is the underlying configuration of the data-generating Markov Process**. For example, the internal state in the Bayesian filtering/POMDP setup is represented as a belief state (Thrun et al., 2005), a "memory" unit in neural networks, or as a distribution over observations for PSRs. The RNN model uses the internal state to make predictions $y_t = f(h_t, x_t)$ and is trained by minimizing a series of loss functions ℓ_t over each prediction, as shown in the following

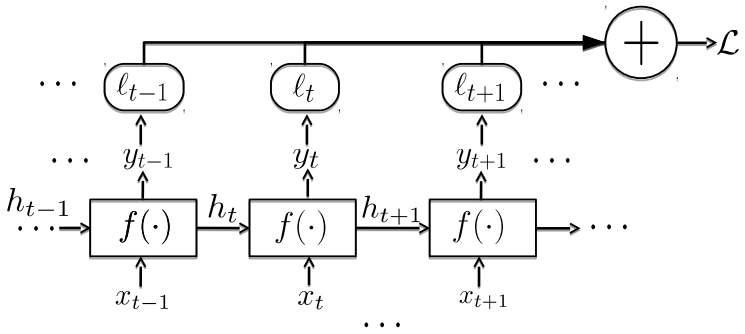


Figure 8.2: Learning recurrent models consists of learning a function f that updates the internal state h_t given the latest observation x_t. The internal state may also be used to predict targets y_t, such as control actions for imitation and reinforcement learning. These are then inputs to a loss function ℓ which accumulate as the multi-step loss $\mathcal{L}$ over all timesteps.

optimization problem:

$$\min_f \mathcal{L} = \min_f \sum_t \ell_t(f(h_t, x_t)) \tag{8.1}$$

The loss functions ℓ_t are usually application- and domain-specific. For example, in a probabilistic filtering problem, the objective may be to minimize the negative log-likelihood of the observations (Abbeel et al., 2005a; Vega-Brown and Roy, 2013) or the prediction of the next observation (Ondruska and Posner, 2016). For imitation learning, this objective function will penalize deviation of the prediction from the expert's action (Ross et al., 2011a), and for policy-gradient reinforcement learning methods, the objective includes the log-likelihood of choosing actions weighted by their observed returns. In general, the task objective optimized by the network does not directly specify a loss directly over the values of the internal state h_t. RNN models are thus often trained with backpropagation-through-time (Werbos, 1990) (BPTT). BPTT allows future losses incurred at timestep t to be back-propagated and affect the parameter updates to f. These updates to f then change the distribution of internal states computed during the next forward pass through time. The difficulty is then that small updates to f can drastically change the distribution of h_t, sometimes resulting in error exponential in the time horizon Theorem 3.1.1. Regardless, as the problem is non-convex, BPTT can only move the learner towards a local optimum while a better solution for the internal state transition model may exist elsewhere in parameter space. Below, we propose PSDs as an additional supervision signal which empirically improves the convergence basin quality for RNNs.

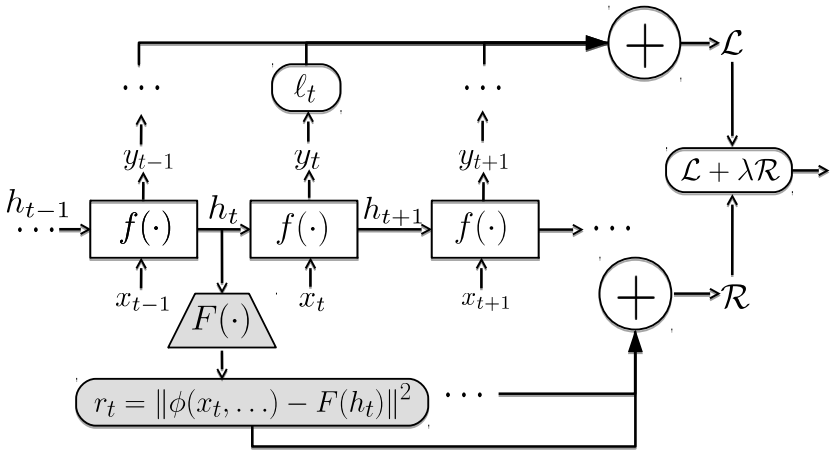


Figure 8.3: PREDICTIVE-STATE DECODER Architecture. We augment the RNN from Fig. 8.2 with an additional objective function $\mathcal{R}$ which targets decoding of the internal state through F at each time step to the *predictive-state* which is represented as statistics over the future observations.

8.3 Predictive-State Decoders Architecture

Our PREDICTIVE-STATE DECODERS architecture extends the PSIM idea to general recurrent architectures. We hypothesize that by encouraging the internal states to encode information sufficient for reconstructing the predictive state, the resulting internal states better capture the underlying dynamics and learning can be improved. The predictive-state we discuss here is equivalent to the learned message passing, Eq. (6.3), introduced in Chapter 6 for PSIM. The result is a simple-to-implement objective function which is coupled with the existing RNN loss. To represent arbitrary sizes and values of PSRs with a fixed-size internal state in the recurrent network, we attach a decoding module $F(\cdot)$ to the internal states to produce the resulting PSR estimates. Fig. 8.3 illustrates our approach.

Our PSD objective $\mathcal{R}$ is the predictive-state loss:

$$\mathcal{R} = \sum_t \|F(h_t) - \phi([x_{t+1}, x_{t+2}, \ldots])\|_2^2, \quad h_t = f(h_{t-1}, x_{t-1}), \tag{8.2}$$

where F is a decoder that maps from the internal state h_t to an *empirical sample* of the predictive-state, computed from a sequence of observed future observations available at training. The network is optimized by minimizing the weighted total loss function $\mathcal{L} + \lambda\mathcal{R}$ where λ is the weighting on the predictive-state objective $\mathcal{R}$. This penalty encourages the internal states to encode information sufficient for directly predicting sufficient future observations. Unlike more standard regularization techniques, $\mathcal{R}$ does not regularize the parameters of the network but instead regularizes the *output variables*, the internal states predicted by the network.

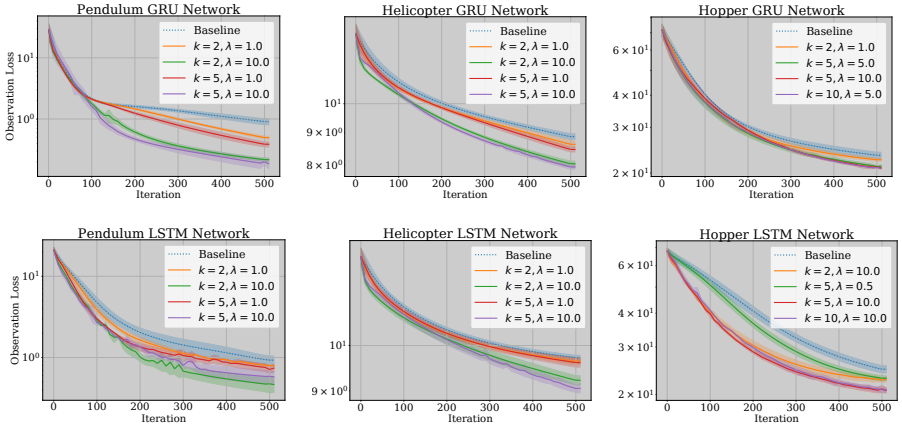
Our method may be interpreted as an instance of Multi-Task Learning (MTL) (Caruana, 1998). MTL has found use in recent deep neural networks (Agrawal et al., 2016; Jaderberg et al., 2016; Kokkinos, 2016). The idea of MTL is to employ a shared representation to perform complementary or similar tasks. When the learner exhibits good performance on one task, some of its understanding can be transferred to a related task. In our case, forcing RNNs to be able to more explicitly reason about the future they will encounter is an intuitive and general method. Endowing RNNs with a theoretically-motivated representation of the future better enables them to serve their purpose of making sequential predictions, resulting in more effective learning. This difference is pronounced in applications such as imitation and reinforcement learning (Sections 8.4.2 and 8.4.3) where the primary objective is to find a control policy to maximize accumulated future reward while receiving only observations from the system. MTL with PSDs supervises the network to predict the future and implicitly the consequences of the learned policy. Finally, our PSD objective can be considered an instance of self-supervision (Jaderberg et al., 2016) as it uses the inputs to the learner to form a secondary *unsupervised* objective.

As discussed in Section 8.2, the purpose of the internal state in recurrent network models (RNNs, LSTMs, deep, or otherwise) is to capture a quantity similar to that of state. Ideally, the learner would be able to back-propagate through the primary objective function $\mathcal{L}$ and discover the best representation of the latent state of the system towards minimizing the objective. However, as this problem highly non-convex, BPTT often yields a locally-optimal solution in a basin determined by the initialization of the parameters and the dataset. By introducing $\mathcal{R}$, the space of feasible models is reduced. We observe next how this objective leads our method to find better models.

8.4 Experiments

We present results on problems of increasing complexity for recurrent models: probabilistic filtering, Imitation Learning (IL), and Reinforcement Learning (RL). The first is easiest, as the goal is to predict the next future observation given the current observation and internal state. For imitation learning, the recurrent model is given training-time expert guidance with the goal of choosing actions to maximize the sequence of future rewards. Finally, we analyze the challenging domain of reinforcement learning, where the goal is the same as imitation learning but expert guidance is unavailable.

PREDICTIVE-STATE DECODERS require two hyperparameters: k, the number of observations to characterize the predictive state and λ, the regularization trade-off factor. In most cases, we primarily tune λ, and set k to one of $\{2, \ldots, 10\}$. For each domain, for each k, there were λ values for which the performance was worse than the baseline. However, for many sets of hyperparameters, the performance exceeded the baselines. Most notably, for many experiments, the convergence rate was significantly better using PSDs, implying that PSDs allows for more efficient data utilization for learning recurrent



(d) Pendulum (e) Helicopter (f) Hopper

Figure 8.4: Loss over predicting future observations during filtering. For both RNNs with GRU cells (*top*) and with with LSTM cells (*bottom*), adding PSDs to the RNN networks can often improve performance and convergence rate.

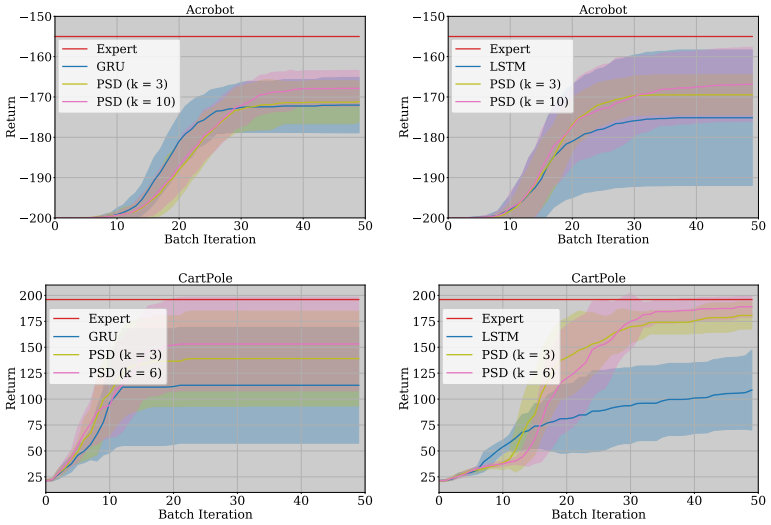
models.

PSDs also require a specification of two other parameters in the architecture: the featurization function ϕ and decoding module F. For simplicity, we use an affine function as the decoder F in Eq. (8.2). The results presented below use an identity featurization ϕ for the presented results but include a short discussion of second order featurization. We find that in each domain, we are able to improve the performance of the state-of-the-art baselines. We observe improvements with both GRU and LSTM cells across a range of k and λ. In IL with PSDs, we come significantly closer and occasionally eclipse the expert's performance, whereas the baselines never do. In our RL experiments, our method achieves statistically significant improvements over the state-of-the-art approach of (Duan et al., 2016; Schulman et al., 2015) on the 5 different settings we tested.

8.4.1 Probabilistic Filtering

In the probabilistic filtering problem, the goal is to predict the future from the current internal state. Recurrent models for filtering use a multi-step objective function that maximizes the likelihood of the future observations over the internal states and dynamics model f's parameters. Under a Gaussian assumption (e.g. like a Kalman filter (Haarnoja et al., 2016)), the equivalent objective that minimizes the negative log-likelihood is given as

$$\min_f \left[\mathcal{L} = \sum_t \|x_{t+1} - f(x_t, h_t)\|^2 \right]. \tag{8.3}$$



(c) GRU (d) LSTM

Figure 8.5: Cumulative rewards for AggreVaTeD and AggreVaTeD+Predictive-State Decoders, with both LSTM (right) and GRU (left) cell, averaged over 15 runs with different random seeds, on partially observable CartPole and Acrobot.

While traditional methods would explicitly solve for parametric internal states h_t using an EM style approach, we use BPTT to implicitly find an non-parametric internal state. We optimize the end-to-end filtering performance through the Psd joint objective $\min_{f,F} \mathcal{L} + \lambda\mathcal{R}$. Our experimental results are shown in Fig. 8.4. The experiments were run with ϕ as the identity, capturing statistics representing the first moment. We tested ϕ as second-order statistics and found while the performance improved over the baseline, it was outperformed by the first moment. In all environments, a dataset was collected using a preset control policy. In the Pendulum experiments, we predict the pendulum's angle θ. The LQR controlled Helicopter experiments (Abbeel and Ng, 2005b) use a noisy state as the observation, and the Hopper dataset was generated using the OpenAI simulation (Brockman et al., 2016) with robust policy optimization algorithm (Pinto et al., 2017) as the controller.

We test each environment with Tensorflow's built-in GRU and LSTM cells (Abadi et al., 2016). We sweep over various k and λ hyperparameters and present the average results and standard deviations from runs with different random seeds. Fig. 8.4 baselines are recurrent models equivalent to Psds with $\lambda = 0$.

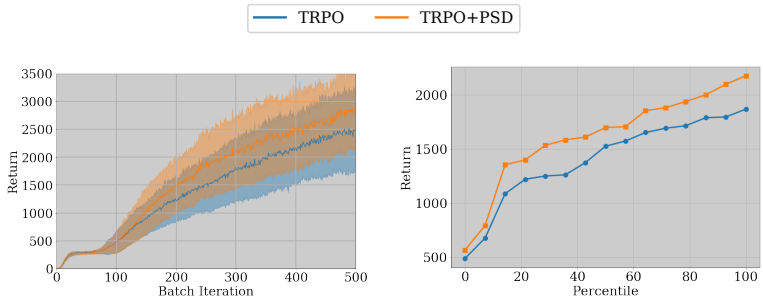


Figure 8.6: Walker Cumulative Rewards and Sorted Percentiles. $N = 15$, $5e4$ TRPO steps per iteration.

8.4.2 Imitation Learning

We experiment with the partially observable CartPole and Acrobot domains[1] from OpenAI Gym (Brockman et al., 2016). We applied the method of AggreVaTeD (Sun et al., 2017), a policy-gradient method, to train our expert models. AggreVaTeD uses access to a cost-to-go oracle in order to train a policy that is sensitive to the value of the expert's actions, providing an advantage over behavior cloning IL approaches. The experts have access to the *full* state of the robots, unlike the learned recurrent policies.

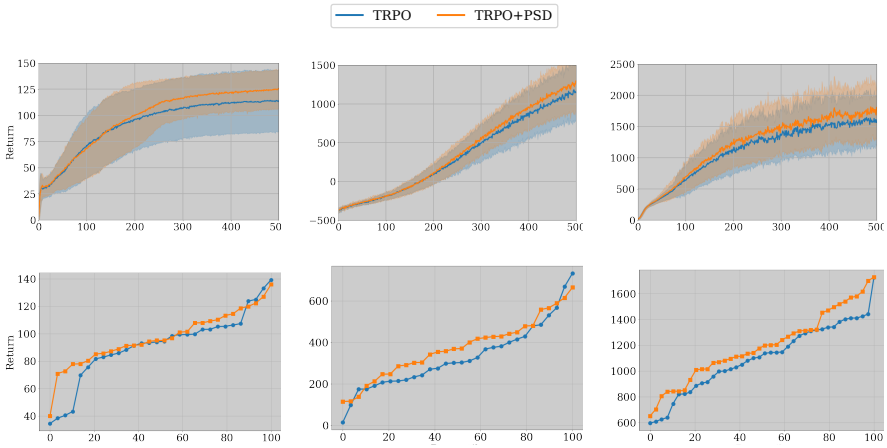
We tune the parameters of LSTM and GRU agents (e.g., learning rate, number of internal units) and afterwards only tune λ for PSDs. In Fig. 8.5, we observe that PSDs improve performance for both GRU- and LSTM-based agents and increasing the predictive-state horizon k yields better results. Notably, PSDs achieves 73% relative improvement over baseline LSTM and 42% over GRU on Cartpole. Difference random seeds were used. The cumulative reward of the current best policy is shown.

8.4.3 Reinforcement Learning

Reinforcement learning (RL) increases the problem complexity from imitation learning by removing expert guidance. The latent state of the system is heavily influenced by the RL agent itself and changes as the policy improves. We use (Duan et al., 2016)'s implementation of TRPO (Schulman et al., 2015), a Natural Policy Gradient method (Kakade, 2002). Although (Schulman et al., 2015) defines a KL-constraint on policy parameters that affect actions, our implementation of PSDs introduces parameters (those of the decoder) that are unaffected by the constraint, as the decoder does not directly govern the agent's actions.

In these experiments, results are highly stochastic due to both environment randomness and nondeterministic parallelization of `rllab` (Duan et al., 2016). We therefore repeat each experiment at least 15 times with paired random seeds.

[1]The observation function only provides positional information (including joint angles), excluding velocities.



(e) Swimmer, N=30 (f) HalfCheetah, N=30 (g) Hopper, N=40

Figure 8.7: *Top:* Per-iteration average returns for TRPO and TRPO+Predictive-State Decoders vs. batch iteration, with $5e3$ steps per iteration. *Bottom:* Sorted per-run mean average returns (across iterations). Our method generally produces better models.

We use $k = 2$ for most experiments ($k = 4$ for Hopper), the identity featurization for ϕ, and vary λ in $\{10^1, 10^0, \ldots, 10^{-6}\}$, and employ the LSTM cell and other default parameters of TRPO. We report the same metric as (Duan et al., 2016): per-TRPO batch average return. Additionally, we report per-run performance by plotting the sorted average TRPO batch returns (each item is a number representing a method's performance for a single seed).

Figs. 8.6 and 8.7 demonstrate that our method generally produces higher-quality results than the baseline. These results are further summarized by their means and stds. in Table 8.1. In Figure 8.6, 40% of our method's models are better than the best baseline model. In Figure 8.7(g), 25% of our method's models are better than the second-best (98^{th} percentile) baseline model. We compare various RNN cells in Table 8.2, and find our method can improve Basic (linear + `tanh` nonlinearity), GRU, and LSTM RNNs, and usually reduces the performance variance. We used Tensorflow (Abadi et al., 2016) and passed both the "hidden" and "cell" components of an LSTM's internal state to the decoder. We also conducted preliminary additional experiments with second order featurization ($\phi(x) = [x, \text{vec}(xx^T)]$). Corresponding to Tab. 8.2, column 1 for the inverted pendulum, second order features yielded 861 ± 41, a 4.9% improvement in the mean and a large reduction in variance.

	Swimmer	HalfCheetah	Hopper	Walker2d	Walker2d†
(Schulman et al., 2015)	91.3 ± 25.5	330 ± 158	1103 ± 264	383 ± 96	1396 ± 396
(Schulman et al., 2015)+PSDs	$\mathbf{97.0 \pm 19.4}$	$\mathbf{372 \pm 143}$	$\mathbf{1195 \pm 272}$	$\mathbf{416 \pm 88}$	$\mathbf{1611 \pm 436}$
Rel. Δ	6.30%*	13.0%*	9.06%*	8.59%*	15.4%**

Table 8.1: *Top:* Mean Average Returns $\pm$ one standard deviation, with $N = 15$ for Walker2d† and $N = 30$ otherwise. *Bottom*: Relative improvement of on the means. * indicates $p < 0.05$ and ** indicates $p < 0.005$ on Wilcoxon's signed-rank test for significance of improvement. All runs computed with $5e3$ transitions per iteration, except Walker2d†, with $5e4$.

	InvertedPendulum			Swimmer		
	Basic	GRU	LSTM	Basic	GRU	LSTM
(Schulman et al., 2015)	$\mathbf{820 \pm 139}$	673 ± 268	640 ± 265	66.0 ± 21.4	64.6 ± 55.3	56.5 ± 23.8
(Schulman et al., 2015)+PSDs	820 ± 118	$\mathbf{782 \pm 183}$	$\mathbf{784 \pm 215}$	$\mathbf{71.4 \pm 26.9}$	$\mathbf{75.1 \pm 28.8}$	$\mathbf{61.0 \pm 23.8}$
Rel. Δ	-0.08%	20.4%	22.6%	8.21%	16.1%	7.94%

Table 8.2: Variations of RNN units. Mean Average Returns $\pm$ one standard deviation, with $N = 20$. $1e3$ transitions per iteration are used. Our method can improve each recurrent unit we tested.

8.5 Conclusion

We introduced Predictive-State Decoders, a theoretically-motivated method developed from PSIMs (Chapters 6 and 7) for improving the training of RNNs. PSDs assign statistical meaning to a learner's internal state when modeling partially observable systems.. Our approach adapts the learning objective from PSIMs and applies it to more complicated recurrent models such as LSTMs and GRUs. With PSDs, we also show that the PSIM idea can be used for learning objectives beyond probabilistic filtering such as imitation and reinforcement learning. Our straightforward method improves performance across all domains with which we experimented.

Part IV

Conclusion

Chapter 9

Summary and Future

This thesis shows that by directly targeting the inference task using observable quantities to generate supervision, machine learning can be used to achieve good performance on time-series and sequential prediction problems. We first showed that observable quantities could serve as the oracle for a learner to correct its mistakes during multi-step prediction (Challenge 1). For prediction in partially observed systems with hidden state (Challenge 2), the observations themselves were used to create a sufficient state representation, allowing supervised learning algorithms to directly target inference performance.

9.1 Summary of Contributions

Data as Demonstrator (DaD)
Chapters 3 and 4 developed DaD to address the shortcomings seen in existing supervised learning methods for learning multi-step predictive models. Our meta-algorithm uses a DAgger-inspired iterative procedure to learn a model. By using the training data itself to generate synthetic training examples, DaD is able to correct for learner-induced error during multi-step sequential prediction. We then showed that DaD with model-based reinforcement learning achieves better control performance using fewer samples from the system.

Online Instrumental Variable Regression (Oivr)
Oivr was introduced in Chapter 5 as a method to model partially observable linear systems using streaming data. We note that this technique is applicable anywhere where batch instrumental variable regression (Ivr) can be used. We derived a regret analysis of Oivr and showed it was no-regret with respect to batch Ivr. The key insight into modelling partially observed system from Boots et al. (2013); Hefny et al. (2015a) showed that statistics of future observations (given the past) provide a predictive representation of the system's state. Using batch Ivr or Oivr allows for better performance in running a probabilistic filter on the dynamics model, as it works to remove correlation between the past and

future prediction of observations.

Predictive State Inference Machines (PSIMs)
OIVR enhanced filtering performance indirectly by improving the method for learning the dynamics of a model. However, it did not directly targeting the inference problem itself. Chapter 6 introduced a new inference machine procedure, PSIM, that directly optimized the end-to-end filtering performance using a similar predictive representation derived from statistics over future observations. Even when optimizing a linear model, PSIM showed significant improvement over IVR because it directly optimized for the filter's output. We presented results using multiple training methods, including Forward Training, DAgger training, and backpropagation-through-time (BPTT). PSIM was extended in Chapter 7 to simultaneously predict side-information available at training time. Our extension expanded the usefulness of PSIM to situations where there are other desired prediction targets beyond simply predicting future observations.

Predictive-State Decoders (Psds)
Finally, Chapter 8 expanded the applicability of predictive representations to include other learning architectures, such as recurrent LSTM and GRU neural networks, that cannot not be trained via DAgger or Forward Training. Our approach augments any general recurrent neural network with a PSD objective to target the prediction of statistics over future observations. We showed that this works in a variety of domains with different target objective functions. In probabilistic filtering, the objective was the same as for PSIM, however, more complex recurrent models were used. With imitation learning, the objective was a cost-sensitive loss to mimic the expert's action, and in reinforcement learning the optimization was to find a recurrent control policy to maximize expected accumulated reward. PSDs improved performance and convergence rates in these three distinct domains.

9.2 Comparison

While these algorithms target different problems, we summarize and provide a comparison in Table 9.1 of our contributions along with other algorithms discussed in this paper. We mark an algorithm as "working on observable quantities" if it does not require information about the underlying system that is not directly available in the training data. We also classify if an algorithm was "built for time-series problems" versus being a general machine learning or statistics technique. Finally, we categorize the methods by if it "directly trains an inference procedure" versus finding a model which is used within another method to do inference.

A discerning reader will note in Table 9.1 we mark DAD $\rightarrow$ MBRL Control as not "directly training the inference procedure" since the inference goal in control is to predict the action to take, and not to learn the dynamics model. DAD in this setting only targeted the multi-step predictive performance of the model.

	Observable Quantities	Built for Time-Series & Sequential Prediction	Directly Trains Inference Procedure
Single-step ML models	✓	×	×
DAD	✓	✓	✓
DAgger → MBRL Control	✓	×	×
DAD → MBRL Control	✓	✓	×
Ordinary Least Squares → Kalman Filter	✓	×	×
OIVR → Kalman Filter	✓	✓	×
Autoregressive Model	✓	×	✓
IVR → Kalman Filter	✓	×	×
N4SID → Kalman Filter	✓	×	×
PSIM via DAD (DAgger), BPTT, and Forward Training	✓	✓	✓
Kalman Filter with assumed State Parametrization	×	×	×
Unscented Kalman Filter	×	×	×
IMF	×	✓	✓
PSIM +Hints	✓	✓	✓
RNN + BPTT	✓	✓	✓
RNN + PSDs + BPTT	✓	✓+	✓

Table 9.1: Comparison of various algorithms discussed in this thesis. A ✓ specifies that an algorithm satisfies the condition and × indicates not achieving the condition.

The algorithms marked as "→ Kalman filter" are similar; they optimize for a dynamics model independent of the intended goal for use in a Kalman filter. We categorize autoregressive models as "not training a time-series model" since they are often trained for single-step performance and thus has the same issues as described in Chapter 3. Of final note, we classify RNN + PSDs as ✓+ to signify that we improved performance over the baseline which was already trained to handle sequential prediction via BPTT.

9.3 Limitations and Future Work

Neither the PSIM architecture we introduced in Chapter 6 nor the extension in Chapter 7 accommodate controlled dynamical systems with actions chosen with a non-blind policy. Non-blind policies are those that depend on observations, current or past, coming from the system. The challenge in allowing such policies when modeling partially observed systems is that it becomes difficult to decouple the effect of the control policy from the natural dynamics. In many ways this is simply the same issue as decoupling causative relationships from correlation – what part of the state transition is from control versus any inherent, unobserved dynamics? Learning a model via PSIM of a system controlled with a non-blind policy would result in a biased estimator (an example of a biased estimator for

linear dynamics was discussed in Chapter 5). This setting is more studied in the field of linear systems and can be solved with special projection operators (e.g. oblique projection), but these do not trivially generalize to nonlinear dynamics. In discrete observation and action spaces, Bowling et al. (2006) provides methods for learning unbiased PSRs. In a preliminary analysis, we found that importance sampling could be used to find an unbiased estimator for PSIMs. However, in experimental tests, we were unable to see a significant improvement in prediction or control performance. This may be due to the bias-variance trade-off in importance sampling. Using more biased estimators may decrease the variance and therefore achieve better performance. An unbiased estimator will often have high variance and thus achieve poor performance. A more elegant solution is necessary to move forward the idea of using PSIM for modeling systems were the observable quantities are collected under a non-blind policy.

We side-stepped this limitation in Chapter 8 by directly optimizing the control performance with a policy gradient approach. Even though using a PSD objective would still promote a biased estimate, we found that adding PSDs improved convergence and performance for both imitation and reinforcement learning. Both the preliminary work we have done with PSIM and actions (discussed above) as well as with PSDs suggests that good empirical performance can be achieved for control tasks without directly addressing the biased estimator problem. It would be interesting to develop more robust theory as to when this is true. One possibility is that collecting data across iterations under a changing policy decorrelates some of the bias effects. Laskey et al. (2017) performs an analysis into how the state distribution changes induced by a changing policy can affect the value of the policy. An open question remains in how such changes affects the overal bias in the learned model.

Finally, it may unnecessary for PSDs to target prediction error of future observations in the imitation and reinforcement learning settings. The strucutre of PSDs as presented was inspired by a traditional factorization of the problem: one would first define a belief state filter and then define a control policy. However, for problems where the goal is to maximize the expected accumulated reward, i.e., the expected Value of the policy, it may be sufficient to target the prediction of statistics over *future rewards*. Intuitively, actions can be easily selected if the future sequence of rewards can be accurately predicted.

This thesis developed various novel techniques addressing the two fundamental challenges, Challenge 1 and Challenge 2, as outlined in the introduction of the thesis. Yet, many fascinating problems remain in advancing machine learning towards solving those challenges. Our results, theoretical and experimental, showed significant advances over existing supervised learning methods on time-series and sequential prediction problems and serve as a stepping-stone for future advances in the field.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Pieter Abbeel and Andrew Y Ng. Learning first-order markov models for control. In *NIPS*, pages 1–8, 2005a.

Pieter Abbeel and Andrew Y Ng. Exploration and apprenticeship learning in reinforcement learning. In *ICML*, pages 1–8. ACM, 2005b.

Pieter Abbeel, Adam Coates, Michael Montemerlo, Andrew Y Ng, and Sebastian Thrun. Discriminative training of kalman filters. In *Robotics: Science and Systems (RSS)*, 2005a.

Pieter Abbeel, Varun Ganapathi, and Andrew Y Ng. Learning vehicular dynamics, with application to modeling helicopters. In *NIPS*, pages 1–8, 2005b.

Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8. ACM, 2006.

Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 5074–5082. Curran Associates, Inc., 2016.

Luis Antonio Aguirre, Bruno Otávio S Teixeira, and Leonardo Antônio B Tôrres. Using data-driven discrete-time models and the unscented kalman filter to estimate unobserved variables of nonlinear systems. *Physical Review E*, 72(2):026226, 2005.

Karl Johan Åström and Peter Eykhoff. System identification—a survey. *Automatica*, 7(2):123–162, 1971.

Karl Johan Aström and Richard M Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.

J Andrew Bagnell and Jeff G Sc Hneider. Autonomous helicopter control using reinforcement learning policy search methods. *ICRA*, 2:1615–1620, 2001.

J Andrew Bagnell, Felipe Cavalcanti, Lei Cui, Thomas Galluzzo, Martial Hebert, Moslem Kazemi, Matthew Klingensmith, Jacqueline Libby, Tian Yu Liu, Nancy Pollard, et al. An integrated system for autonomous robotics manipulation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2955–2962. IEEE, 2012.

Bram Bakker, Viktor Zhumatiy, Gabriel Gruener, and Jürgen Schmidhuber. Quasi-online reinforcement learning for robots. *ICRA*, pages 2997–3002, 2006.

Arindam Banerjee, Xin Guo, and Hui Wang. On the optimality of conditional expectation as a bregman predictor. *Information Theory, IEEE Transactions on*, 51(7): 2664–2669, 2005.

Arslan Basharat and M Shah. Time series prediction by chaotic modeling of nonlinear dynamical systems. In *IEEE International Conference on Comuter Vision*, pages 1941–1948, 2009.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2): 157–166, 1994.

Christopher M Bishop. *Pattern recognition*. Springer, 2006.

Byron Boots. *Spectral Approaches to Learning Predictive Representations*. PhD thesis, Carnegie Mellon University, December 2012.

Byron Boots and Geoffrey Gordon. An online spectral learning algorithm for partially observable nonlinear dynamical systems. In *AAAI*, 2011a.

Byron Boots and Geoffrey Gordon. Two-manifold problems with applications to nonlinear system identification. In *ICML*, 2012.

Byron Boots and Geoffrey J. Gordon. Predictive state temporal difference learning. In *NIPS*, 2011b.

Byron Boots, Sajid M Siddiqi, and Geoffrey J Gordon. Closing the learning-planning loop with predictive state representations. *IJRR*, 30(7):954–966, 2011.

Byron Boots, Arthur Gretton, and Geoffrey J. Gordon. Hilbert space embeddings of predictive state representations. In *UAI-2013*, 2013.

Roger J Bowden and Darrell A Turkington. *Instrumental variables*, volume 8. Cambridge University Press, 1990.

Michael Bowling, Peter McCracken, Michael James, James Neufeld, and Dana Wilkinson. Learning predictive state representations using non-blind policies. In *Proceedings of the 23rd international conference on Machine learning*, pages 129–136. ACM, 2006.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

A Colin Cameron and Pravin K Trivedi. *Microeconometrics: methods and applications*. Cambridge university press, 2005.

Joaquin Quinonero Candela, Agathe Girard, Jan Larsen, and Carl Edward Rasmussen. Propagation of uncertainty in bayesian kernel models-application to multiple-step ahead forecasting. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 2, pages II–701. IEEE, 2003.

David Card. Using geographic variation in college proximity to estimate the return to schooling. Technical report, National Bureau of Economic Research, 1993.

Rich Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.

Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile. On the generalization ability of on-line learning algorithms. *Information Theory, IEEE Transactions on*, 50(9): 2050–2057, 2004.

Antoni B. Chan and Nuno Vasconcelos. Classifying Video with Kernel Dynamic Textures. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6, June 2007.

Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. *arXiv preprint arXiv:1203.3472*, 2012.

Guillaume Chevillon and David F Hendry. Non-parametric direct multi-step estimation for forecasting economic processes. *International Journal of Forecasting*, 21(2): 201–218, 2005.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015.

Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Learning for control from multiple demonstrations. In *ICML*, pages 144–151, New York, NY, USA, 2008. ACM.

Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Marc Peter Deisenroth, Marco F Huber, and Uwe D Hanebeck. Analytic moment-based gaussian process filtering. In *International Conference on Machine Learning*, pages 225–232. ACM, 2009.

Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423, 2015.

Harris Drucker, Donghui Wu, and Vladimir N Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural networks*, 10(5):1048–1054, 1999.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

Yong Duan, Qiang Liu, and XinHe Xu. Application of reinforcement learning in robot soccer. *Engineering Applications of Artificial Intelligence*, 20(7):936–950, 2007.

Zoubin Ghahramani and Sam T Roweis. Learning nonlinear dynamical systems using an EM algorithm. In *NIPS*, pages 431—-437, 1999.

Agathe Girard, Carl Edward Rasmussen, Joaquin Quinonero-Candela, and Roderick Murray-Smith. Gaussian process priors with uncertain inputs – application to multiple-step ahead time series forecasting. In *NIPS*. MIT Press, 2003.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *ICML*, volume 14, pages 1764–1772, 2014.

Sander Greenland. An introduction to instrumental variables for epidemiologists. *International Journal of Epidemiology*, 29(4):722–729, 2000.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 2016.

Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop kf: Learning discriminative deterministic state estimators. *arXiv preprint arXiv:1605.07148*, 2016.

Tarek Hamel and Robert Mahony. Attitude estimation on SO(3) based on direct inertial measurements. In *ICRA*, pages 2170–2175. IEEE, 2006.

Godfrey H. Hardy. *Divergent series*. American Mathematical Society, 2000.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

Elad Hazan and Satyen Kale. Projection-free Online Learning. *29th International Conference on Machine Learning (ICML 2012)*, pages 521–528, 2012.

Elad Hazan and Satyen Kale. Beyond the regret minimization barrier: optimal algorithms for stochastic strongly-convex optimization. *JMLR*, 2014.

Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007.

Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

Ahmed Hefny, Carlton Downey, and Geoffrey J Gordon. Supervised learning for dynamical system learning. In *NIPS*, 2015a.

Ahmed Hefny, Carlton Downey, and Geoffrey J. Gordon. A new view of predictive state methods for dynamical system learning. *arXiv preprint arXiv:1505.05310*, 2015b.

Todd Hester, Michael Quinlan, and Peter Stone. Rtmba: A real-time model-based reinforcement learning architecture for robot control. *ICRA*, pages 85–90, 2012.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Daniel Hsu, Sham M. Kakade, and Tong Zhang. A spectral algorithm for learning hidden markov models. In *COLT*, 2009.

Daniel Hsu, Sham M Kakade, and Tong Zhang. A spectral algorithm for learning hidden markov models. *Journal of Computer and System Sciences*, 78(5):1460–1480, 2012.

Humphrey Hu and George Kantor. Parametric covariance prediction for heteroscedastic noise. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 3052–3057. IEEE, 2015.

Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *CoRR*, abs/1611.05397, 2016. URL `http://arxiv.org/abs/1611.05397`.

Sham Kakade. A natural policy gradient. *Advances in neural information processing systems*, 2:1531–1538, 2002.

Dov Katz, Arun Venkatraman, Moslem Kazemi, J Andrew Bagnell, and Anthony Stentz. Perceiving, learning, and exploiting object affordances for autonomous pile manipulation. In *Robotics Science and Systems*, 2013.

Alonzo Kelly, Anthony Stentz, Omead Amidi, Mike Bode, David Bradley, Antonio Diaz-Calderon, Mike Happold, Herman Herman, Robert Mandelbaum, Tom Pilarski, et al. Toward reliable off road autonomous vehicles operating in challenging environments. *The International Journal of Robotics Research*, 25(5-6):449–483, 2006.

Seyed Mohammad Khansari-Zadeh and Aude Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *Robotics, IEEE Transactions on*, 27(5):943–957, 2011.

Christian Kleiber and Achim Zeileis. *Applied econometrics with R*. Springer Science & Business Media, 2008.

J Ko, D J Klein, D Fox, and D Haehnel. GP-UKF: Unscented kalman filters with Gaussian process prediction and observation models. In *IROS*, pages 1901–1907, 2007.

Jonathan Ko and Dieter Fox. Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, 2009.

Iasonas Kokkinos. Ubernet: Training a 'universal' convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. *CoRR*, abs/1609.02132, 2016.

George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. *IJRR*, page 0278364911428653, 2011.

Michael C Koval, Nancy S Pollard, and Siddhartha S Srinivasa. Pre-and post-contact policy decomposition for planar contact manipulation under uncertainty. *The International Journal of Robotics Research*, 35(1-3):244–264, 2016.

Alex Kulesza, N Raj Rao, and Satinder Singh. Low-rank spectral learning. In *Proceedings of the 17th Conference on Artificial Intelligence and Statistics*, 2014.

Brian Kulis, Peter L Bartlett, Bartlett Eecs, and Berkeley Edu. Implicit Online Learning. *Proceedings of the 27th international conference on Machine learning (ICML)*, pages 575–582, 2010.

Lucas Langer, Borja Balle, and Doina Precup. Learning multi-step predictive state representations. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

John Langford, Ruslan Salakhutdinov, and Tong Zhang. Learning nonlinear dynamic models. In *ICML*, pages 593–600. ACM, 2009.

Michael Laskey, Jonathan Lee, Wesley Hsieh, Richard Liaw, Jeffrey Mahler, Roy Fox, and Ken Goldberg. Iterative noise injection for scalable imitation learning. *arXiv preprint arXiv:1703.09327*, 2017.

M Lázaro-Gredilla. Sparse spectrum Gaussian process regression. *The Journal of Machine Learning Research*, 11:1865–1881, 2010. URL `http://dl.acm.org/citation.cfm?id=1859914`.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.

Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for non-linear biological movement systems. In *ICINCO (1)*, pages 222–229, 2004.

Guosheng Lin, Chunhua Shen, Ian Reid, and Anton van den Hengel. Deeply learning the messages in message passing inference. In *Advances in Neural Information Processing Systems*, pages 361–369, 2015.

Long-Ji Lin and Tom M Mitchell. Memory approaches to reinforcement learning in non-markovian domains. Technical report, Carngie Mellon University, 1992.

Long-Ji Lin and Tom M Mitchell. Reinforcement learning with hidden states. In *Proceedings of the second international conference on From animals to animats 2: simulation of adaptive behavior: simulation of adaptive behavior*, pages 271–280. MIT Press, 1993.

Nick Littlestone. From on-line to batch learning. In *Proceedings of the second annual workshop on Computational learning theory*, pages 269–284, 2014.

Michael L. Littman, Richard S. Sutton, and Satinder Singh. Predictive representations of state. In *NIPS*, pages 1555–1561. MIT Press, 2001a.

Michael L Littman, Richard S Sutton, and Satinder P Singh. Predictive representations of state. In *NIPS*, volume 14, pages 1555–1561, 2001b.

Lennart Ljung. System identification. In *Signal analysis and prediction*, pages 163–173. Springer, 1998.

Massimiliano Marcellino, James H Stock, and Mark W Watson. A comparison of direct and iterated multistep ar methods for forecasting macroeconomic time series. *Journal of econometrics*, 135(1):499–526, 2006.

Maja J Matarić. Reinforcement learning in the multi-robot domain. In *Robot colonies*, pages 73–83. Springer, 1997.

Edward Miguel, Shanker Satyanath, and Ernest Sergenti. Economic shocks and civil conflict: An instrumental variables approach. *Journal of Political Economy*, 112(4): 725–753, 2004.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V Todorov. Interactive control of diverse complex characters with neural networks. In *Advances in Neural Information Processing Systems*, pages 3132–3140, 2015.

Katharina Muelling, Arun Venkatraman, Jean-Sebastien Valois, John Downey, Jeffrey Weiss, Shervin Javdani, Martial Hebert, Andrew B Schwartz, Jennifer L Collinger, and J Andrew Bagnell. Autonomy infused teleoperation with application to bci manipulation. *arXiv preprint arXiv:1503.05451*, 2015.

KR Müller, AJ Smola, and G Rätsch. Predicting time series with support vector machines. *Artificial Neural Networks — ICANN'9*, 1327:999–1004, 1997.

Kumpati S Narendra and Kannan Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, pages 4–27, 1990.

Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cognitive processing*, 12(4):319–340, 2011.

Yu Nishiyama, Amir Hossein Afsharinejad, Shunsuke Naruse, Byron Boots, and Le Song. The nonparametric kernel Bayes smoother. In *AISTATS*, 2016.

Peter Ondruska and Ingmar Posner. Deep tracking: Seeing beyond seeing using recurrent neural networks. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML*, 28:1310–1318, 2013.

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. *arXiv preprint arXiv:1703.02702*, 2017.

Ali Punjani and Pieter Abbeel. Deep learning helicopter dynamics models. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3223–3230. IEEE, 2015.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NIPS*, pages 1177–1184, 2007.

Liva Ralaivola and Florence D'Alche-Buc. Dynamical modeling with kernels for nonlinear time series prediction. *NIPS*, 2004.

Varun Ramakrishna, Daniel Munoz, Martial Hebert, J. Andrew Bagnell, and Yaser Sheikh. Pose machines: Articulated pose estimation via inference machines. In *Computer Vision–ECCV 2014*, pages 33–47. Springer, 2014.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *ICLR*, 2016.

C Radhakrishna Rao, Helge Toutenburg, Heuman C Shalabh, and M Schomaker. Linear models and generalizations. *Least Squares and Alternatives (3rd edition) Springer, Berlin Heidelberg New York*, 2008.

Laminar Research. X-plane. DVD, 2015.

Stephane Ross and Drew Bagnell. Agnostic system identification for model-based reinforcement learning. *ICML*, pages 1703–1710, 2012.

Stéphane Ross and J. Andrew Bagnell. Efficient reductions for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, pages 661–668, 2010.

Stéphane Ross and J Andrew Bagnell. Stability conditions for online learnability. *arXiv preprint arXiv:1108.3154*, 2011.

Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *AISTATS*, 2011a.

Stephane Ross, Daniel Munoz, Martial Hebert, and J Andrew Bagnell. Learning message-passing inference machines for structured prediction. In *CVPR, 2011*, pages 2737–2744. IEEE, 2011b.

Sam Roweis and Zoubin Ghahramani. A unifying review of linear gaussian models. *Neural computation*, 11(2):305–345, 1999.

Matthew Rudary and Satinder Singh. Predictive linear-gaussian models of stochastic dynamical systems. In *In 21st Conference on Uncertainty in Artificial Intelligence*, 2005.

Stefan Schaal et al. Learning from demonstration. *NIPS*, pages 1040–1046, 1997.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

Shai Shalev-Shwartz. Online Learning and Online Convex Optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2011.

Sajid Siddiqi, Byron Boots, and Geoffrey J. Gordon. A constraint generation approach to learning stable linear dynamical systems. In *NIPS 20 (NIPS-07)*, 2007.

Sajid Siddiqi, Byron Boots, and Geoffrey J. Gordon. Reduced-rank hidden Markov models. In *AISTATS*, 2010.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Satinder Singh, Michael R. James, and Matthew R. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *UAI*, 2004.

Jonas Sjöberg, Qinghua Zhang, Lennart Ljung, Albert Benveniste, Bernard Delyon, Pierre-Yves Glorennec, Håkan Hjalmarsson, and Anatoli Juditsky. Nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31(12): 1691–1724, 1995.

Torsten Söderström and Petre Stoica. Instrumental variable methods for system identification. *Circuits, Systems and Signal Processing*, 21(1):1–9, 2002.

Le Song, Byron Boots, Sajid M Siddiqi, Geoffrey J Gordon, and Alex J Smola. Hilbert space embeddings of hidden markov models. In *ICML*, pages 991–998, 2010.

Wen Sun, Arun Venkatraman, Byron Boots, and J. Andrew (Drew) Bagnell. Learning to filter with predictive state inference machines. In *International Conference on Machine Learning (ICML 2016)*, June 2016.

Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *ICML*, 2017.

Ilya Sutskever. *Training recurrent neural networks*. PhD thesis, University of Toronto, 2013.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Erik Talvitie. Agnostic system identification for monte carlo planning. In *AAAI*, pages 2986–2992, 2015.

Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148, 2016.

Sebastian Thrun. An approach to learning mobile robot navigation. *RAS*, 15(4), 1995.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

Ryan Turner and Carl Edward Rasmussen. Model based learning of sigma points in unscented kalman filtering. *Neurocomputing*, 80:47–53, 2012.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

Peter Van Overschee and BL De Moor. *Subspace identification for linear systems: Theory-Implementation-Applications*. Springer Science & Business Media, 2012.

William Vega-Brown and Nicholas Roy. Cello-em: Adaptive sensor models without ground truth. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1907–1914. IEEE, 2013.

Arun Venkatraman, Byron Boots, Martial Hebert, and J Andrew Bagnell. Data as demonstrator with applications to system identification. *NIPS Autonomously Learning Robots Workshop*, 2014.

Arun Venkatraman, Martial Hebert, and J Andrew Bagnell. Improving multi-step prediction of learned time series models. In *AAAI*, pages 3024–3030, 2015.

"Arun Venkatraman, Roberto Capobianco, Lerrel Pinto, Martial Hebert, Daniele Nardi, and J. Andrew (Drew) Bagnell". Improved learning of dynamics for control. In *International Symposium on Experimental Robotics (ISER)*, October 2016a.

Arun Venkatraman, Wen Sun, Martial Hebert, J. Andrew Bagnell, and Byron Boots. Online instrumental variable regression with applications to online linear system identification. In *AAAI*, 2016b.

Arun Venkatraman, Wen Sun, Martial Hebert, Byron Boots, and J. Andrew (Drew) Bagnell. Inference machines for nonparametric filter learning. In *25th International Joint Conference on Artificial Intelligence (IJCAI-16)*, July 2016c.

Arun Venkatraman, Nicholas Rhinehart, Wen Sun, Lerrel Pinto, Martial Hebert, Byron Boots, Kris M. Kitani, and J. Andrew Bagnell. Predictive-state decoders: Encoding the future into recurrent networks. In *Advances in Neural Information Processing Systems (NIPS)*, July 2017.

Eric A Wan and Ronell Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *AAS-SPCC*, pages 153–158. IEEE, 2000.

Jack Wang, Aaron Hertzmann, and David M Blei. Gaussian process dynamical models. In *NIPS*, pages 1441–1448, 2005.

William Wu-Shyong Wei. *Time series analysis*. Addison-Wesley publ, 1994.

Andrew A Weiss. Multi-step estimation and forecasting in dynamic models. *Journal of Econometrics*, 48(1):135–149, 1991.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

David Wingate and Satinder Singh. Kernel predictive linear gaussian models for nonlinear stochastic dynamical systems. In *International Conference on Machine Learning*, pages 1017–1024. ACM, 2006.

Martin Zinkevich. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *ICML*, pages 421–422, 2003.