Thesis Proposal

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

Arun Venkatraman

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

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*

September 2016

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. 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. Fundamental to our learning framework is that the prediction of observable quantities is a *lingua franca* for building AI systems. We propose three extensions that leverage this general idea and adapt it to a variety of problems. The first aims to improve the training time and performance of more sophisticated recurrent neural networks. The second extends the PSIM framework to controlled dynamical systems. The third looks to train recurrent architectures for reinforcement learning problems.

Contents

List of Figures

Chapter 1

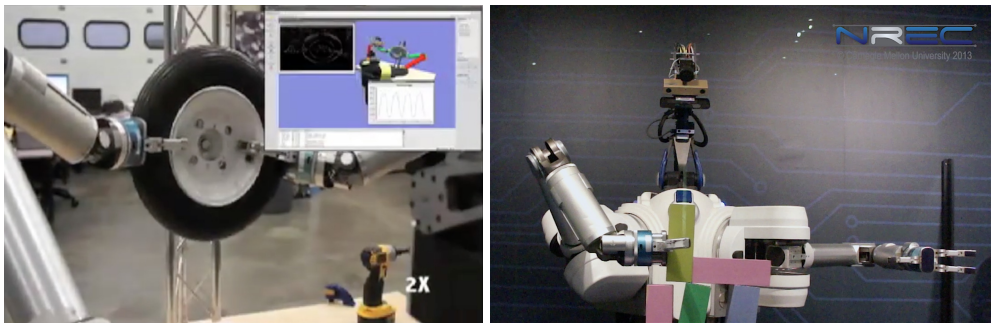
Introduction

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 the past couple decades in a variety of applications, from spam filtering (Drucker et al., 1999) to hand-writing 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.

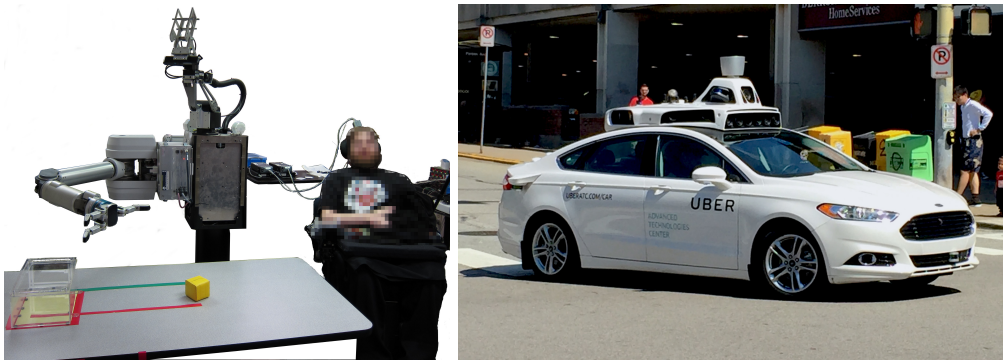
> *This thesis proposes that time series and sequential prediction, whether for forecasting, filtering, or reinforcement learning, can be effectively achieved by directly training recurrent prediction procedures rather then building generative probabilistic models.*

First, 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 wraps these traditional approaches and we show theoretically and empirically (Section 3.1) that it improves multi-step performance on model classes from RNNs (Bengio et al., 2015) to random forests and can be useful to accelerate model-based reinforcement learning (Section 3.2).

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 (Section 3.3). Conventional approaches for learning and filtering on these



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



(b) BCI Teleoperated Prosthetic (left) and Uber 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.

problems often choose parametric graphical model representations and more importantly decouple the model learning from probabilistic inference on the model. Using a DAD-style training procedure, our approach, PREDICTIVE STATE INFERENCE MACHINES (PSIMS), directly optimizes the inference (filtering) performance without local optima 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 (Section 3.4) 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.

Central to our approaches is that the prediction of observable quantities is a *lingua franca* for building AI systems. We propose three extensions that leverage

this more general idea and adapt it to a variety of problems. The first looks to improve the training time and performance of more recurrent neural networks by using statistics over future observations as additional training signal. The second extends the PSIM framework to controlled dynamical systems, opening up potential applications to imitation and reinforcement learning. The third aims to train recurrent architectures for reinforcement learning problems, exploring ways to use time-series guided learning to augment model-free methods.

Throughout this thesis, we focus our experimental evaluation on robotics applications. Robotic systems grow ever complex and the difficulty of generating physics based-models motivates us to develop data-driven methods to take advantage of volumes of sensor telemetry. 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 et al., 2015)[2]) and to state estimation (e.g. From sensor readings, tracking the state of the car and objects around it.[3]). All of these problems likely have a sequential process aspect to it that is result of time-series or sequential predictions. Though many of the experimental problem setups we use throughout our work is simpler in nature than those illustrated in Fig. 1.1, we hopefully develop foundational algorithms for learning in these large-scale problems.

Contributions

The expected contributions in this thesis support the thesis statement in three key application areas,

Forecasting, Challenge 1:
Data as Demonstrator (Section 3.1), DaD+Control (Section 3.2)

Filtering, Challenge 2:
Predictive State Inference Machines (Section 3.3), PSIM+Hints (Section 3.4), PSIM +RNN (Section 4.1)

Control and Reinforcement Learning, Challenge 3:
DaD+Control (Section 3.2), PSIM+Actions (Section 4.2), Recurrent-Q RL (Section 4.3)

[1]Video at `https://youtu.be/dSJP1BRuJ5Y`
[2]Video at `https://youtu.be/5KjLnyNxeyk`
[3]`http://uberatc.com/car`

Chapter 2

Background

In this section, we summarize and discuss the advances in the literature in developing learning algorithms for time-series and dynamical system modeling as well their applications in filtering and reinforcement learning. We develop three key challenges (Challenge 1 to Challenge 3) along the way for which this thesis develops ideas and methods to tackle in support of our goal to directly learning recurrent prediction procedures.

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 will revisit this topic later (Sections 3.3 and 3.4) and 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 the optimization of a single-step prediction error to fit a model $\hat{f} : x_t \mapsto x_{t+1}$. 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

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

squared loss or likelihood loss to find $\hat{f}$.

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.

2.1.2 Multi-Step Predictive Models

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).

The direct method.

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: $\hat{f}_{(k)} : x_t \mapsto x_{t+k}$ (Chevillon and Hendry, 2005; Langer et al., 2016). In contrast, iterative approaches (which we also refer to as recursive or recurrent), fit a model $\hat{f} : x_t \mapsto x_{t+1}$ 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.

The iterative method.

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. One class of such works attempts to handle this by propagating model uncertainty through the 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 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. In this thesis, we present an alternative training approach is does provide a theoretical performance bound for the multi-step predictive performance.

Challenge 1.
Single-step predictive models for time series problems do not produce expected results for multiple-step predictions. How can we optimize time-series models for multi-step prediction in a way that can give us both theoretical performance garauntees and good emperical results?

2.2 Applications of Dynamical System Models

In this thesis, we focus our time-series problems around dynamical system modeling and its applications. As dynamics models are commonly utilized for state-estimation (Bayesian filtering) or for controls and planning (Thrun et al., 2005), we focus our discussion on approaches developed towards these tasks.

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, 2007). 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., 2011a; Hefny et al., 2015; Venkatraman et al., 2016c). 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 representation. 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

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

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 global optimality. A couple of these approaches as well as those presented in this work try to address the below challenge.

Challenge 2.
How can we directly train predictors to target state estimation or filter performance?

2.2.3 Reinforcement Learning

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).

Model-based Control

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. 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.

Moving Between Model-Free and Model-based RL

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) have gained popularity often due to failures of planning with imperfect dynamics models. Heess et al. (2015) develop 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

[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.

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 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 hidden state. This architecture has been used in Q-learning (Lin and Mitchell, 1992, 1993; Hausknecht and Stone, 2015) or for policy optimization over future predicted states and actions (Mordatch et al., 2015). In the proposed work (Chapter 4), we start to develop ideas that bridge the gap between those developed in this thesis for system identification and filter-learning with that of directly optimizing for control performance.

Challenge 3.

How can we improve reinforcement learning by using time-series training strategies?

Chapter 3

Completed Work

In this chapter, we detail the completed works thus far. The completed works develop two primary algorithms DAD (Venkatraman et al., 2015) and PSIM (Sun et al., 2016) along with an extension to each: DAD+CONTROL (Venkatraman et al., 2016a) and PSIM+HINTS (Venkatraman et al., 2016b).

3.1 Data as Demonstrator: Improving Training of Iterative Time-Series Models for Multi-step Prediction

Challenge 1 states a significant challenge is utilizing machine learning methods for finding useful predictive time-series models for forecasting. 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,

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

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

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" 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 2.1.2. 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.

3.1.1 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,

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

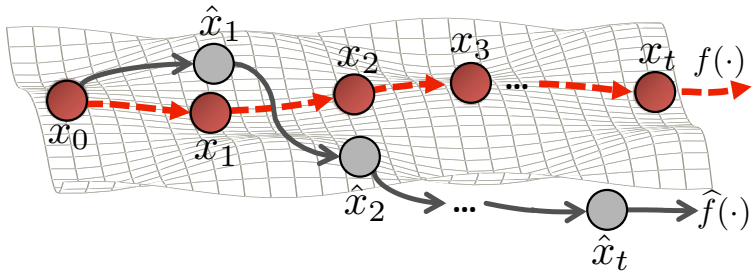
It is also worthwhile to note the Lipshitz constant conditions: For $L > 1$, we get the bound in Theorem 3.1.1. If $L = 1$, the bound reduces to $O(T\epsilon)$, which is equivalent to the result for the supervised learning setting. Finally, for $L < 1$, we get $O(L\epsilon)$. This situation specifies 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 3.1.3. 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.

3.1.2 Data as Demonstrator Algorithm

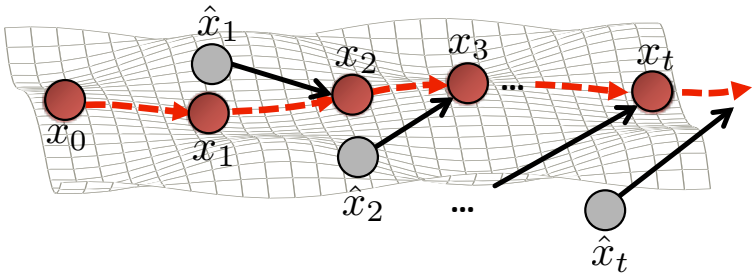
To resolve differences in the train and test (forward prediction) distributions, it would be convenient to augment our training set to meet the test distribution. This is reminiscent of work that uses the predictions from the learner to augment the dataset by searching for (Daumé III et al., 2009) or 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.

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



(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))**.

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 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 $\widehat{f}$ 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

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

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 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 3.1.2. *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 3.1.3. *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 3.1.2 and Theorem 3.1.3 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.

3.1.3 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.

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:

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

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

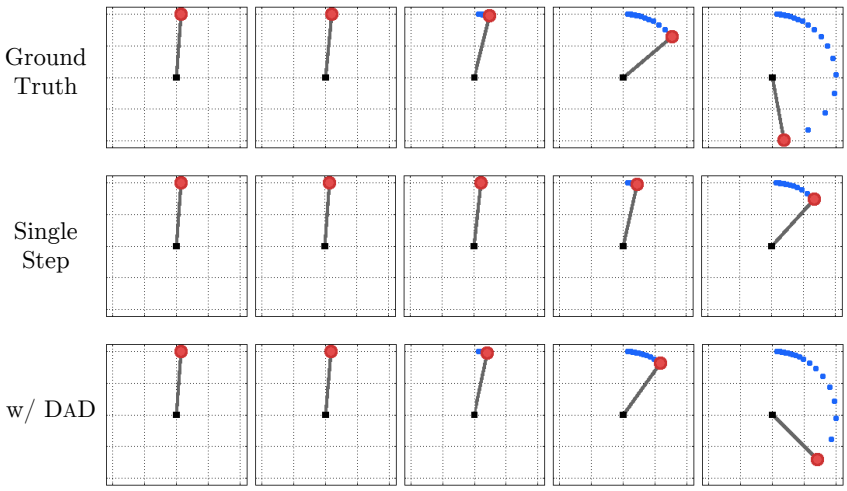


Figure 3.2: A pendulum trajectory's ground truth (top) compared with multi-step predictions displayed for time steps 1, 8, 16, 23, and 30 from the traditional minimization of single-step error (middle) and from our approach (bottom). The final output of DAD is closer to the true final state.

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 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.

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 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

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.

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.

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 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.

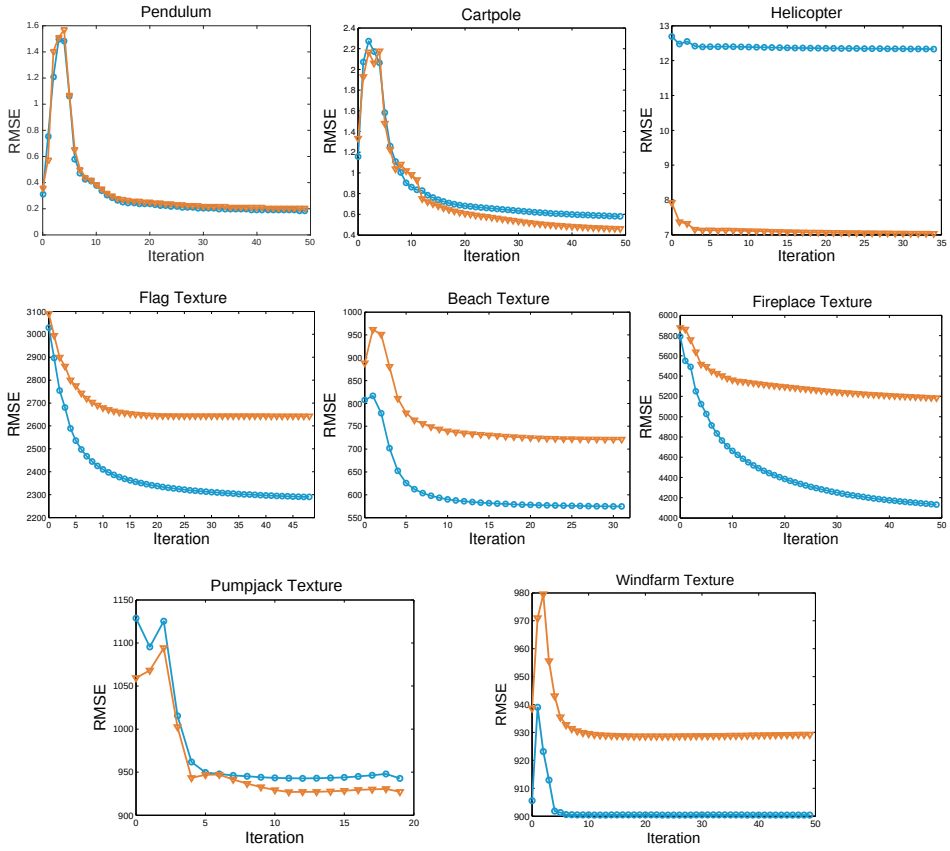


Figure 3.3: RMSE of multiple-step predictions using RFF learner for train (blue, circles) and test (orange, triangles) trajectories. Note that the shown test RMSE was not used by DAD during training and was computed for visualization only.

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 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'

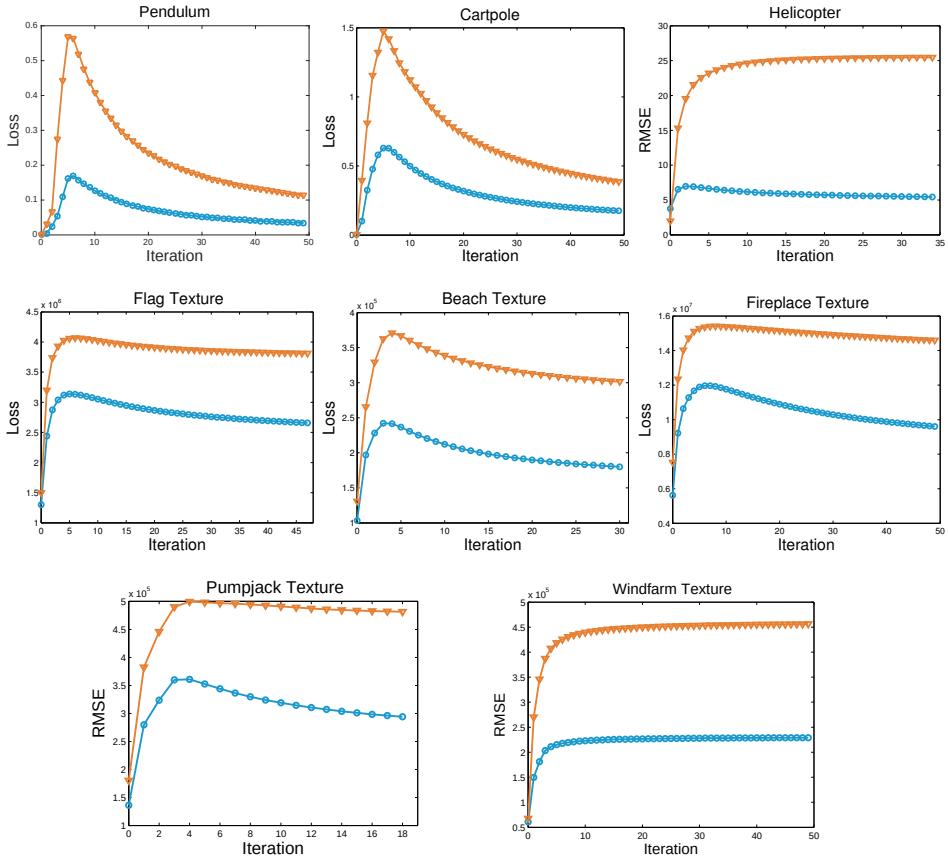


Figure 3.4: Single-step loss using RFF learner on the aggregated training (blue, circles) and test (orange, triangles) data set. Notice that the error increases and then stabilizes over iterations as distribution of states induced by learner's multiple-step prediction stabilizes. Often, the single-step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

(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

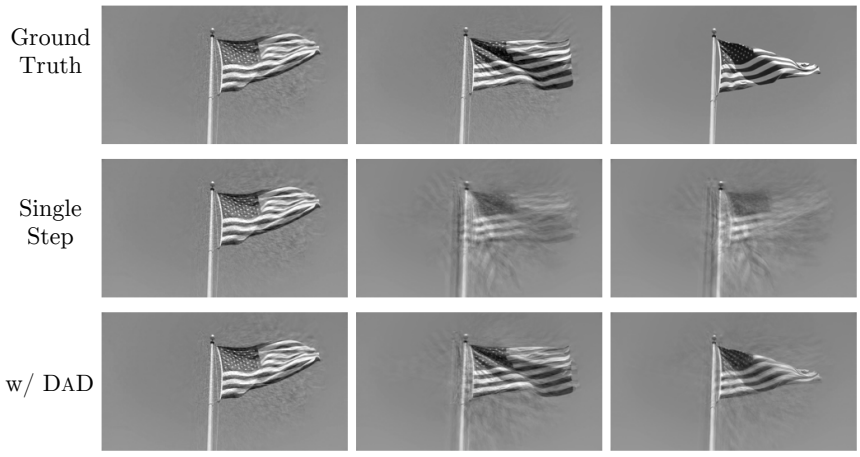
of Fig. 3.7(a), we see the system converge by halfway through the prediction horizon to the mean. DAD provides corrections that can mitigate this tendency (bottom frames in Fig. 3.7(a)). This results in a qualitatively crisper and more believable video, as shown in the close up (Fig. 3.7(b)).

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.

3.1.4 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 result futhers augments the experimental verification for generating synetetic training examples from the ground truth trajectories for improving the multi-step predictive capabilities of learned time-series models.

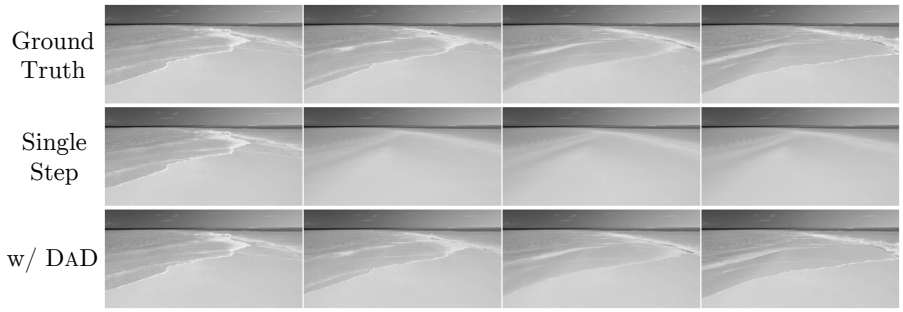
In the next section, we investigate controlled systems and the use of DAD in the context of model-based reinforcement learning.



(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.



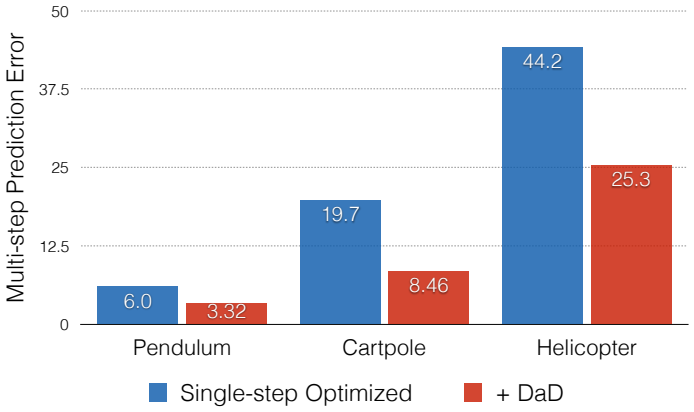
(a) Predicted Beach trajectory using Random Forest learner

Figure 3.6: Predicted trajectory of a Beach Video texture.

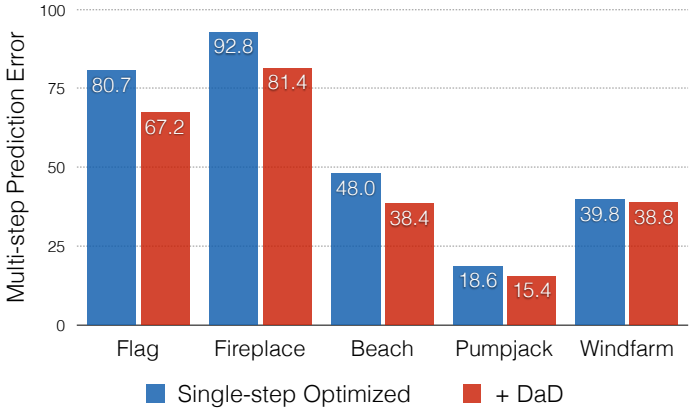
(a) Prediction of frames 14, 19, 26 for the baseline method (top) and DAD (bottom). Note the averaging effect with the single-step predictor with predictions converging on the mean from the dataset. DAD yields crisper predicted images.

(b) Comparison of final predicted frames

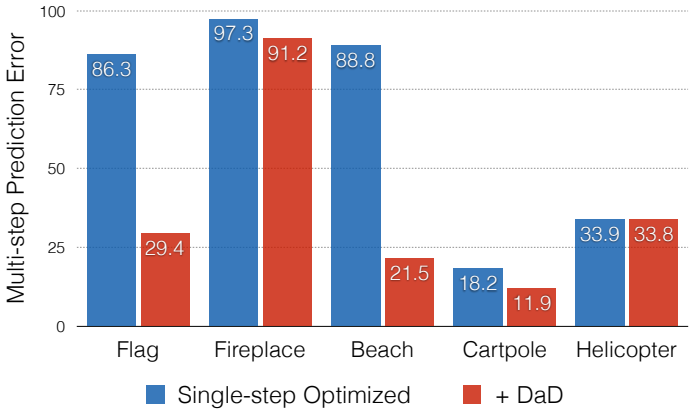
Figure 3.7: In the Fireplace video texture, our method produces a more believable evolution.



(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.

3.2 Improved Dynamics with DaD for Control

Section 2.2.3 introduced control and model-based reinforcement learning (MBRL) as problems where dynamical system modeling plays a vital role. In this section, we extend the DATA AS DEMONSTRATOR (DAD) (introduced in Section 3.1, (Venkatraman et al., 2015)) training procedure for control problems. In addressing Challenge 3, 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. Experimental results indicate that this approach can enable us to achieve good control performance with less data.

The goal of MBRL is to learn a dynamics model for the optimization of a control policy. 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.

3.2.1 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 $f(x_t, u_t) \rightarrow x_{t+1}$, where u_t are controls (actions). We additionally assume a known cost function $c : x_t, u_t \rightarrow \mathbb{R}$. Solving this MDP consists of minimizing the (expected) cumulative cost over a time horizon T, which may be infinite, by finding a control policy $\pi(x_t)$:

$$\pi = \arg\min_{\pi} \sum_{t=0}^{T-1} c(x_t, u_t) \quad \text{s.t. } u_t = \pi(x_t) \text{ and } x_{t+1} = f(x_t, u_t). \tag{3.12}$$

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. (3.12). The learning problem for the dynamics

This work was originally presented in *Improved Learning of Dynamics for Control* at ISER 2016 (Venkatraman et al., 2016a)

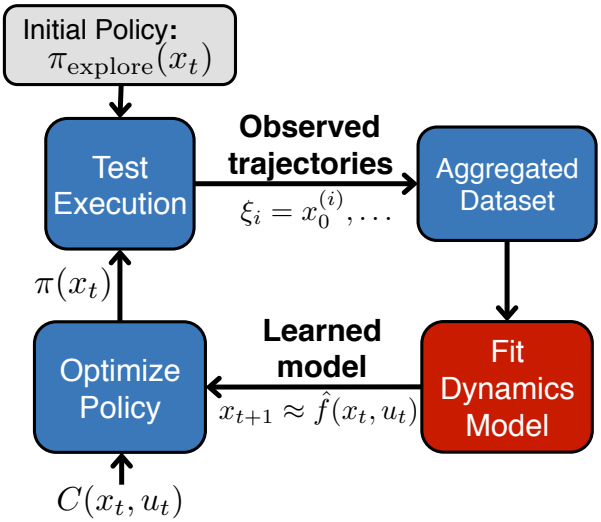


Figure 3.9: DAgger System Identification for Model Learning and Control. We show that using DAD (Section 3.1) for the dynamics model fitting (red) can improve control performance.

model is then formulated as minimizing the predictive error, similar to Eq. (3.4) but with a control input added,

$$\hat{f} = \arg\min \sum_{t=1}^{T-1} \|x_t - \hat{f}(x_{t-1}, u_{t-1})\|_2^2 \tag{3.13}$$

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 Section 3.1, the downside of only doing this optimization Eq. (3.13) 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.

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. 3.9. 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 itera-

tion, 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. 3.9 as the DAgger (Data-set Aggregation) system identification learning framework (Ross and Bagnell, 2012). A key 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.

3.2.2 DaD+Control

As discussed at length in Section 3.1, solely Eq. (3.13) 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

[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`.

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

to the control setting as shown in Fig. 3.10 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.

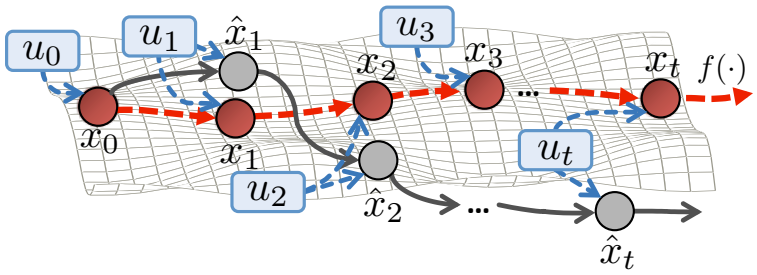
3.2.3 Experimental Evaluation

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:

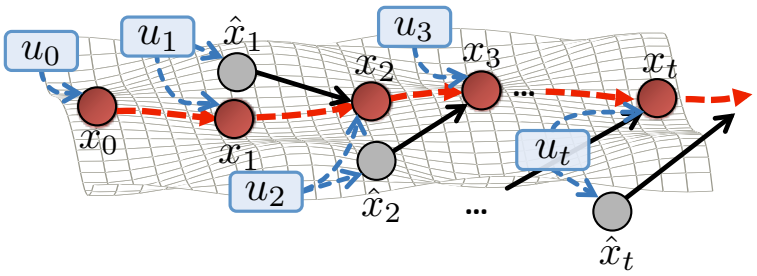
$$\Delta_t \leftarrow f(x_t, u_t), \quad \text{where } \Delta_t = x_{t+1} - x_t. \tag{3.14}$$

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 the

[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



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



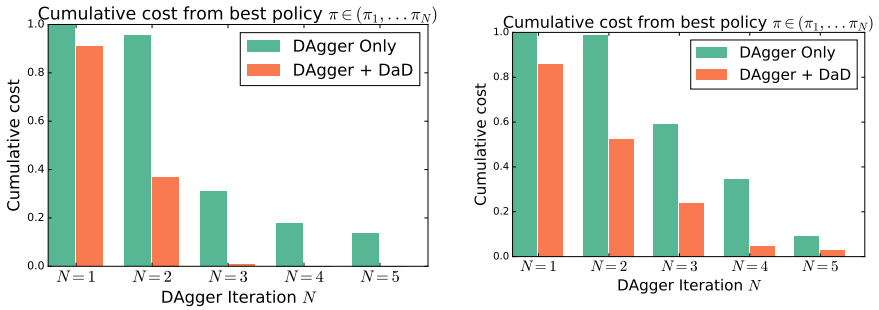
(b) Data provides a demonstration of corrections required to return back to proper prediction

Figure 3.10: Similar to setting considered in Section 3.1 (Fig. 3.1), cascading errors from model learning with controls (blue) can lead to errors in forward simulation (gray) (Fig. 3.10(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. 3.10(b)).

difficulty of optimizing Eq. (3.12) under arbitrary dynamics and cost models, for simplicity, we focus on minimizing a sum-of-quadratics cost-to-go function:

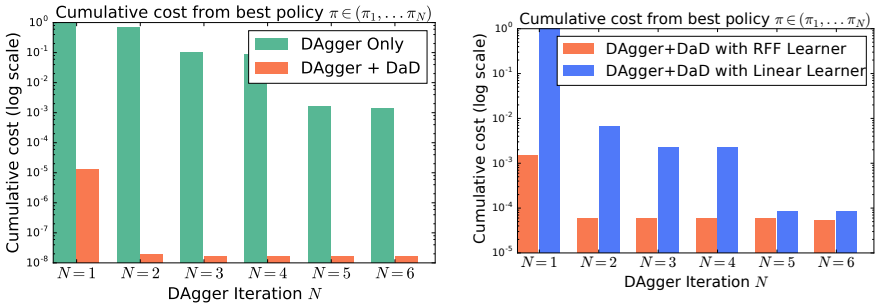
$$\sum_t c(x_t, u_t) = \sum_t x_t^T Q_t x_t + u_t^T R_t u_t. \tag{3.15}$$

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



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

Figure 3.11: Controlling a simulated cartpole for swing-up behavior.



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

Figure 3.12: Controlling a simulated helicopter to hover. Note the log-scale on cost.

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.

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. (3.14) 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

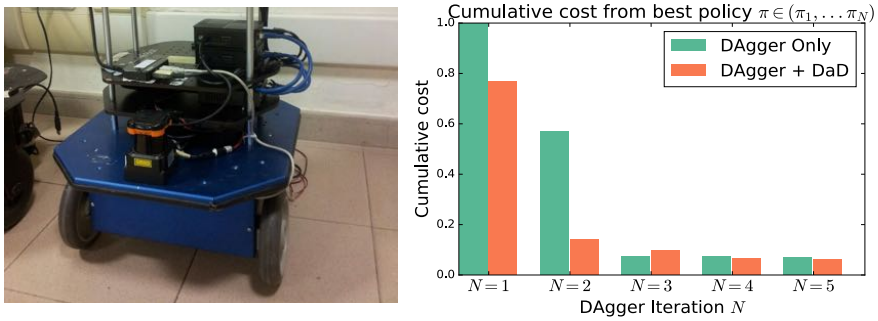


Figure 3.13: Results for controlling a Videre Erratic differential-drive mobile robot.

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. 3.11 of the evaluated trajectory costs accumulated over the problem's time horizon.

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. 3.12 show that DAD dramatically improves performance over only DAgger.

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. 3.13.

Baxter robot: We use the 'DAgger +DAD' approach to control a 7-degree-

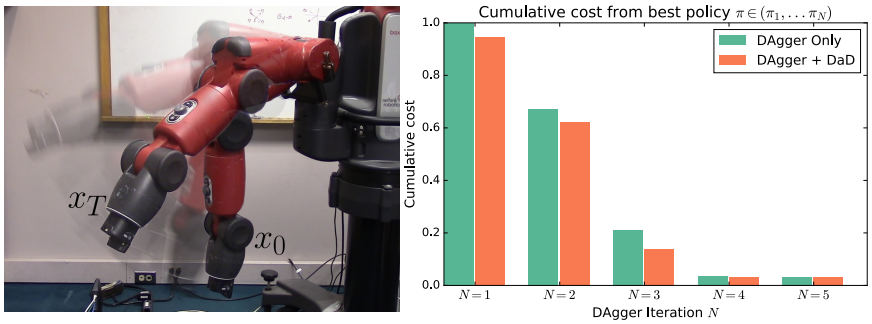


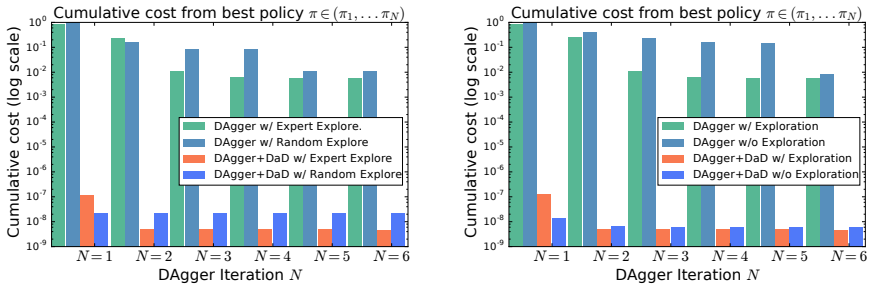
Figure 3.14: Results on controlling a Baxter robot. We learn a dynamics model and compute a control policy to move the robot manipulator from state x_0 to x_T.

of-freedom manipulator to a target joint configuration. We command the robot arm in torque control mode with *suppression* of the inbuilt gravity compensation. 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. 3.14.

3.2.4 Discussion & Conclusion

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. 3.12(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. 3.15(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 perfor-



(a) Random vs. Expert Explore Policy

(b) Explore vs. No Exploration Policy

Figure 3.15: Comparison of Exploration policies. Cost values are not normalized across plots.

mance difference (Fig. 3.15(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 3.2.3, 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), only smaller improvements over 'DAgger Only' can be achieved (consistent with the simulated noisy-actuation result in Fig. 3.11(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.

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).

3.3 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. 3.16), 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).

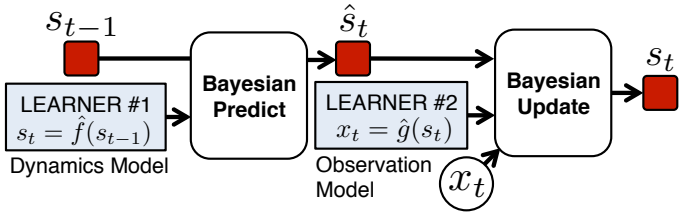


Figure 3.16: Traditional "Filter Learning". Learning is often decoupled for the transition and observation models.

In contrast to this, we propose Predictive State Inference Machines (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. 3.17). 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).

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).

This work was originally presented in *Learning to Filter with Predictive State Inference Machines* at ICML 2016 (Sun et al., 2016)

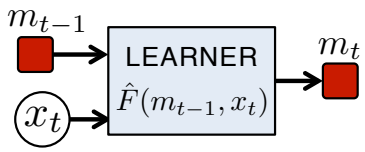


Figure 3.17: Filter Learning with PSIMs. This framework learns a joint predict-and-update function to predict the next belief state.

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., 2015) 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. 3.17). 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., 2011a; Hefny et al., 2015) 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.

3.3.1 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., 2015)[4]. 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[5] 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 of k-observable systems, where it suffices to select features from the next k observations.

Following Hefny et al. (2015), 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

$$P(s_t|h_{t-1}) \Leftrightarrow P(f_t|h_{t-1}) \Leftrightarrow \mathbb{E}[\phi(f_t)|h_{t-1}]. \tag{3.16}$$

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. 3.18 for

[4]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)

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

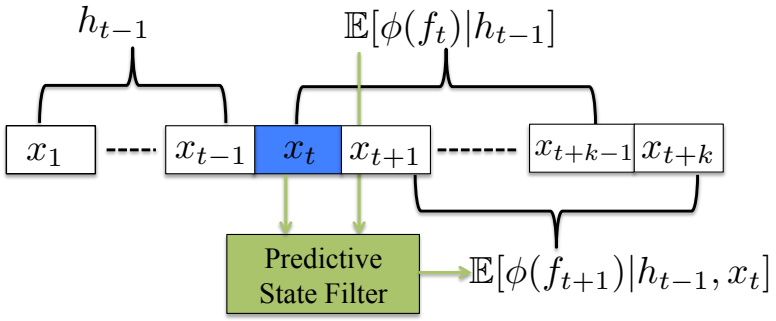


Figure 3.18: Filtering with predictive states for a k-observable system. At time step t, the filter uses the belief $\mathbb{E}[\phi(f_t)|h_{t-1}]$ and the latest observation x_t as feedback, outputs the next belief $\mathbb{E}[\phi(f_{t+1})|h_{t-1}, x_t]$.

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.

3.3.2 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. (3.16), 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. 3.18) which *deterministically* passes the predictive states forward in time conditioned on the latest observation:

$$\mathbb{E}[\phi(f_{t+1})|h_t] = F\Big(\mathbb{E}[\phi(f_t)|h_{t-1}], x_t\Big), \tag{3.17}$$

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. (3.17)). 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., 2011a), PSIM can generalize to non-linear dynamics by leveraging non-linear regression or classification models.

3.3.3 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

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

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[6]

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

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 (Section 3.1) 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]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 3 PREDICTIVE STATE INFERENCE MACHINE (PSIM) with Forward Training

1: **Input:** M independent trajectories τ_i, $1 \leq i \leq M$;
2: Set $\hat{m}_1 = \frac{1}{M}\sum_{i=1}^{M} \phi(f_1^i)$;
3: Set $\hat{m}_1^i = \hat{m}_1$ for trajectory $\tau_i, 1 \leq i \leq M$;
4: **for** t = 1 to T **do**
5: 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;
6: Train a hypothesis F_t on D_t to minimize the loss $d(F(z), f)$ over D_t;
7: For each trajectory τ_i, roll out $F_1, ..., F_t$ along the trajectory (Eq. 3.19) to compute $\hat{m}_{t+1}^i$;
8: **end for**
9: **Return:** the sequence of hypothesis $\{F_t\}_{t=1}^N$.

Forward Training

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

$$\min_{F_1 \in \mathcal{F}, ... F_T \in \mathcal{F}} \mathbb{E}_{\tau \sim \mathcal{D}_\tau} \Big[\frac{1}{T} \sum_{t=1}^{T} d(F_t(\hat{m}_t^\tau, x_t^\tau), f_{t+1}^\tau) \Big], \tag{3.18}$$

$$s.t. \;\; \hat{m}_{t+1}^\tau = F_t(\hat{m}_t^\tau, x_t^\tau), \forall t \in [1, T-1], \tag{3.19}$$

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 3.

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 3 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).

Theorem 3.3.1. *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 3 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 3.3.1 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 3 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. 3:

Theorem 3.3.2. *With infinite many training trajectories, for the sequence $\{F_t\}_t$ generated by Alg. 3, 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. 3.3.2 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 3.3.3. *With M training trajectories, for any $F_t^* \in \mathcal{F}, \forall t$, we have with probability at least $1 - \delta$:*

$$\begin{aligned}
&\mathbb{E}_{\tau\sim\mathcal{D}_\tau}\Big[\frac{1}{T}\sum_{t=1}^{T} d(F_t(\hat{m}_t^\tau, x_t^\tau), f_{t+1}^\tau)\Big] \\
&\leq \mathbb{E}_{\tau\sim\mathcal{D}_\tau}\big[\frac{1}{T}\sum_{t=1}^{T} d(F_t^*(\hat{m}_t^\tau, x_t^\tau), f_{t+1}^\tau)\big] \\
&\quad + 4\nu\bar{\mathcal{R}}(\mathcal{F}) + 2\sqrt{\frac{T\ln(T/\delta)}{2M}},
\end{aligned} \tag{3.20}$$

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 3 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.

DAgger (DaD) training

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 Section 3.1. The optimization we wish to target for finding a good stationary filter F is

$$\min_{F \in \mathcal{F}} \mathbb{E}_{\tau \sim \mathcal{D}_\tau} \frac{1}{T} \sum_{t=1}^{T} d(F(\hat{m}_t, x_t), f_{t+1}), \tag{3.21}$$

$$s.t \quad \hat{m}_{t+1} = F(\hat{m}_t, x_t), \forall t \in [1, T-1], \tag{3.22}$$

where

$$\hat{m}_1 = \arg\min_m \sum_{t=1}^{M} d(m, f_1^i) = \frac{1}{T} \sum_{i=1}^{T} \phi(f_i^i). \tag{3.23}$$

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 Section 3.1. The iterative approach we describe (Algorithm 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 4 rolls out the predictive states using its current hypothesis F_n (Eq. (3.22)) 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.

Algorithm 4 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 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. (3.22)) 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.

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 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. (3.21) can be rewritten as

$$L(F) = \mathbb{E}_\tau[\mathbb{E}_{z,f\sim\omega_{F,\tau}}[d(F(z), f)]|\tau]. \tag{3.24}$$

Define the loss function for any predictor F at iteration n of Algorithm 4 as:

$$L_n(F) = \mathbb{E}_\tau[\mathbb{E}_{z,f\sim\omega_{F_n,\tau}}[d(F(z), f)]|\tau]. \tag{3.25}$$

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

$$\hat{L}_n(F) = \frac{1}{M}\sum_{\tau=1}^{M} \mathbb{E}_{z,f\sim\omega_{F_n,\tau}}\big(d(F(z), f)\big). \tag{3.26}$$

Algorithm 4 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

$$\epsilon_N = \min_{F \in \mathcal{F}} \frac{1}{N} \sum_{n=1}^{N} L_n(F). \tag{3.27}$$

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

$$\frac{1}{N} \sum_{n=1}^{N} L_n(F_n) - \underbrace{\min_{F \in \mathcal{F}} \frac{1}{N} \sum_{n=1}^{N} L_n(F)}_{\epsilon_N} \leq \gamma_N \tag{3.28}$$

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 4. *(Ross et al., 2011a) For Algorithm 4, there exists a predictor* $\hat{F} \in \{F_n\}_{n=1}^{N}$ *such that:*

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

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 3.1.2) 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 4 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

$$\hat{\epsilon}_N = \min_{F \in \mathcal{F}} \frac{1}{N} \hat{L}_n(F) \tag{3.29}$$

$$\hat{\gamma}_N \geq \frac{1}{N} \sum_{n=1}^{N} \hat{L}_n(F_n) - \min_{F \in \mathcal{F}} \frac{1}{N} \sum_{n=1}^{N} \hat{L}_n(F) \tag{3.30}$$

we have:

Corollary 5. *(Ross et al., 2011a) For Algorithm 4, there exists a predictor* $\hat{F} \in \{F_n\}_{n=1}^{N}$ *such that with probability at least* $1 - \delta$*:*

$$\begin{aligned} L(\hat{F}) = \mathbb{E}_\tau \left[\mathbb{E}_{z, f \sim \omega_{\hat{F}, \tau}} (d(\hat{F}(z), f)) | \tau \right] &\leq \hat{\gamma}_N + \hat{\epsilon}_N \\ &+ L_{\max} \left(\sqrt{\frac{2 \ln(1/\delta)}{MN}} \right). \end{aligned} \tag{3.31}$$

3.3.4 Experimental Evaluation

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

$$\phi_1(f_t) = [x_t, ..., x_{t+k-1}] \tag{3.32}$$
$$\phi_2(f_t) = [x_t, ..., x_{t+k-1}, {x_t}^2, ..., x_{t+k-1}^2]. \tag{3.33}$$

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 3.2: Various implementations of PSIM tested on the dynamical systems.

We compare the PSIM variations in Table 3.2 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., 2015), and give some results with comparisons to a Recurrent Neural Network (**RNN**) trained with BPTT (LeCun et al., 2015).

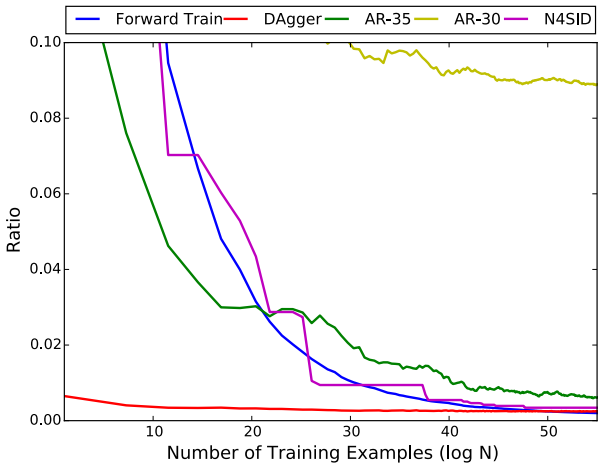


Figure 3.19: The convergence rate of different algorithms. The ratios (y-axis) are computed as $\log(\frac{e}{e_F})$ for error e from corresponding algorithms. The x-axis is computed as $\log(N)$, where N is the number of trajectories used for training.

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

$$\begin{aligned} s_{t+1} &= As_t + \epsilon_s, \quad \epsilon_s \sim \mathcal{N}(0, Q), \\ x_t &= Cs_t + \epsilon_x, \quad \epsilon_x \sim \mathcal{N}(0, R), \end{aligned} \tag{3.34}$$

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., 2015)

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. 3.19 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. 3.19. PSIM with DAgger performs much better with few training data while Forward Training eventually slightly surpasses DAgger with sufficient data. The AR-k

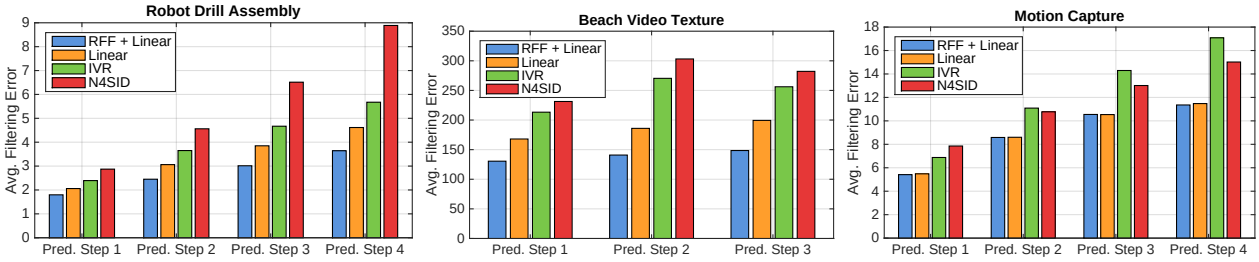


Figure 3.20: PSIM vs. Baselines. Filter error for multiple look ahead steps for the future predictions shown for a few of the datasets. We see across datasets that the performance of both IVR and N4SID are significantly worse than using PSIM. For some datasets, the nonlinearity of the Random Fourier 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.

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[7]. 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[8]. 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.

[7]Collected from the ARM-S system (Bagnell et al., 2012)
[8]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 3.1.3.

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 3.3: 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 3.3. 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. 3.20). 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 3.4. 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 3.4: 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 3.5 for feature function ϕ_2 and $k = 5$ using PSIM with linear ridge regression and DAgger training. Comparing to the results with phi_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 3.5: PSIM with linear regression using ϕ_1 vs. ϕ_2 features.

3.3.5 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 while guarantees filtering performance for general non-linear models in both the realizable and agnostic settings. The filter function learned by PSIM can be considered a dynamical system model specialized for filtering performance (Challenge 2). In the next section, 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".

3.4 Extending PSIM with Hints

The standard PSIMs (Section 3.3) 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, predicting prosthetic motion commands from neural signal decoding in biomedical applications, visual tracking where a bounding-box must be predicted in computer vision, speech-to-text in natural language processing, or localization in robotics.

In this section, we continue work for addressing 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 develops 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 section, 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.

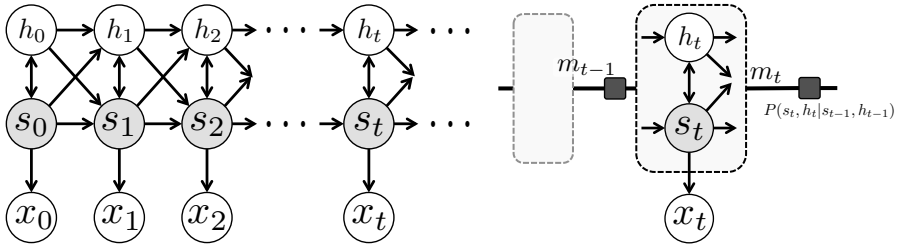
To ground the difference between *supervised-state* and *latent-state*, we consider simple pendulum dynamics (e.g. Fig. 3.2). In the supervised-state setting, we assume access at training to a sufficient-state representation of the system. For the pendulum the natural and sufficient state representation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

3.4.1 Predictive State Inference Machine with Hints

In this section, we extend PSIMs to models with partially observable states or side-information (Fig. 3.21(a)). In this 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. 3.22), it is not straightforward to extend PSIM to a model with the complicated latent structure in Fig. 3.21(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. 3.21(b), and only focus on the net message passed into the configuration

This work was originally presented in *Inference Machines for Nonparametric Filter Learning* at IJCAI 2016 (Venkatraman et al., 2016b)



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

Figure 3.21: **(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.

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. 3.21(b), m_t represents the joint belief of h_t and s_t,

$$m_t = P(h_t, s_t | p_t). \tag{3.35}$$

Note that we changed notation compared to Section 3.3. 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 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:

$$P(h_t, s_t | p_t) \Leftrightarrow P(h_t, x_{t+1:t+k_f} | p_t). \tag{3.36}$$

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:

$$\mathbf{E}\left[\phi(h_t, x_{t+1:t+k_f}) | p_t\right]. \tag{3.37}$$

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

Algorithm 5 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.

essentially becomes the RKHS embedding of $P(h_t, x_{t+1:t+k_f}|p_t)$. We call Eq. (3.37) as the predictive state[9]. 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:

$$m_t = F(m_{t-1}, x_t), \tag{3.38}$$

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

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. 3.38.

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:

$$\begin{aligned} &\min_{F\in\mathcal{F}} \frac{1}{T}\mathbf{E}_{\tau\sim\mathcal{D}}\left[\sum_{t=1}^{T} d\big(F(\hat{m}_{t-1}, x_t), (h_t^\tau, x_{t+1:t+k_f}^\tau)\big)\right], \\ &\text{s.t., } \hat{m}_t = F(\hat{m}_{t-1}, x_t), \forall t, \end{aligned} \tag{3.39}$$

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

As with PSIM, the optimization objective presented in Eq. (3.39) is non-convex in F due to the objective involving sequential prediction terms where

[9]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.

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. (3.39) directly is impractical, we utilize *Dataset Aggregation* (DAgger) (Ross et al., 2011a) training in a similar fashion to DAD Section 3.1 and PSIM Section 3.3 to find a filter function (model) F with *bounded* error. The training procedure for PSIM+HINTS is detailed in Algorithm 5. 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. 5 guarantees to output a hypothesis $\hat{F}$ such that:

$$\mathbf{E}_{\tau\sim\mathcal{D}} \frac{1}{T} \sum_{t=1}^{T} d\big(\hat{F}(\hat{m}_{t-1}^{\tau,\hat{F}}, x_t^\tau), z_t^\tau\big) \leq \epsilon \tag{3.40}$$

$$\text{where } \epsilon = \min_{F\in\mathcal{F}} \frac{1}{N} \sum_{n=1}^{N} \mathbf{E}_{(m,x,z)\sim d_{F_n}} d(F(m,x),z), \tag{3.41}$$

ϵ is the minimum batch minimization error from the entire *aggregated dataset* in hindsight after N iterations of Algorithm 5. Note that this bound applies to Eq. (3.39) for the F found by Algorithm 5.

Despite the learner's loss ϵ in Eq. (3.41) 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., 2015), 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. (3.40) and (3.41) 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.

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

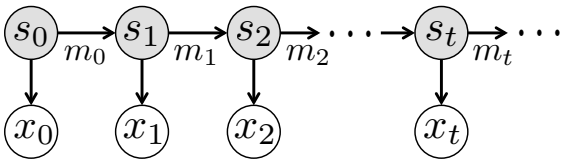


Figure 3.22: Message Passing on a Hidden Markov Model (HMM). The state of the system s_t generates an observation x_t. In the supervised-state setting, the sufficient-states s and observations x are given at training time though only observations are seen at test time. In the latent-state problem setting, we additionally do not have access to the sufficient state at training time – it may be unobserved or have an unknown representation.

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:

$$\min_{g \in \mathcal{G}} \mathbf{E}_{\tau \sim \mathcal{D}} \frac{1}{T} \sum_{t=1}^{T} \| g(\hat{m}_t^\tau) - h_t^\tau \|_2^2 \tag{3.42}$$

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 Section 3.3, 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.

3.4.2 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. 3.22. This problem setting similar 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 5, 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:

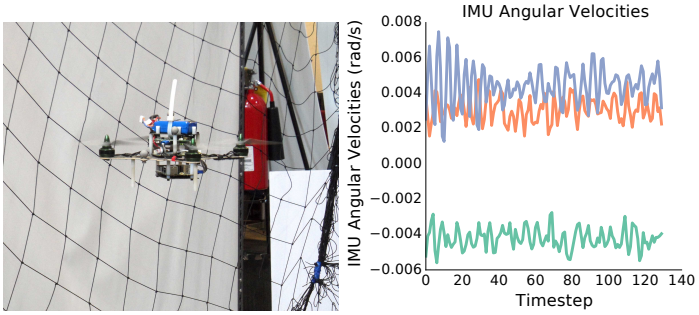
$$P(s_t|p_t) \Leftrightarrow \mathbf{E}\left[\phi(s_t)|p_t\right] = m_t = F(m_{t-1}, x_t) \tag{3.43}$$

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. (3.40) 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).

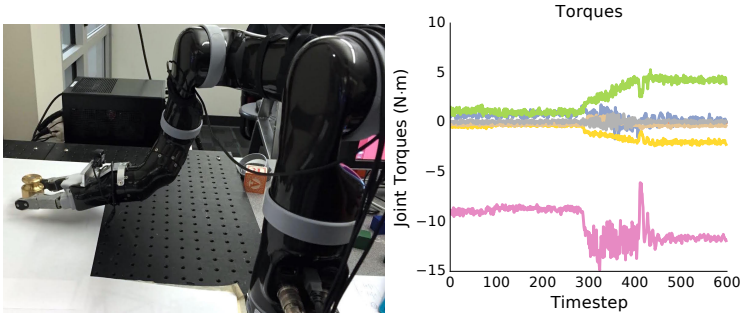
The IMF approach differs from the approach of Langford et al. (2009) in several important ways. The authors proposed a method that learns four operators: one to map the first observation to initial state, one for the belief update with an observation, one for state-to-state transition, and one for state 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 5 does not assume differentiability of the learning procedure as required for backpropagation-through-time used in (Langford et al., 2009).

3.4.3 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.



(a) Quadrotor Hovering



(b) Kinova Mico Grasping

Figure 3.23: 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).

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:

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

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

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 3.6: 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.

optimizing the objective,

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

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 $\frac{1}{T}\sum_{t=1}^{T} \|\hat{h}_t - h_t\|^2$.

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 3.6). This simulated dataset

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 3.7: Quadrotor Attitude Estimation Performance

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 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 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

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

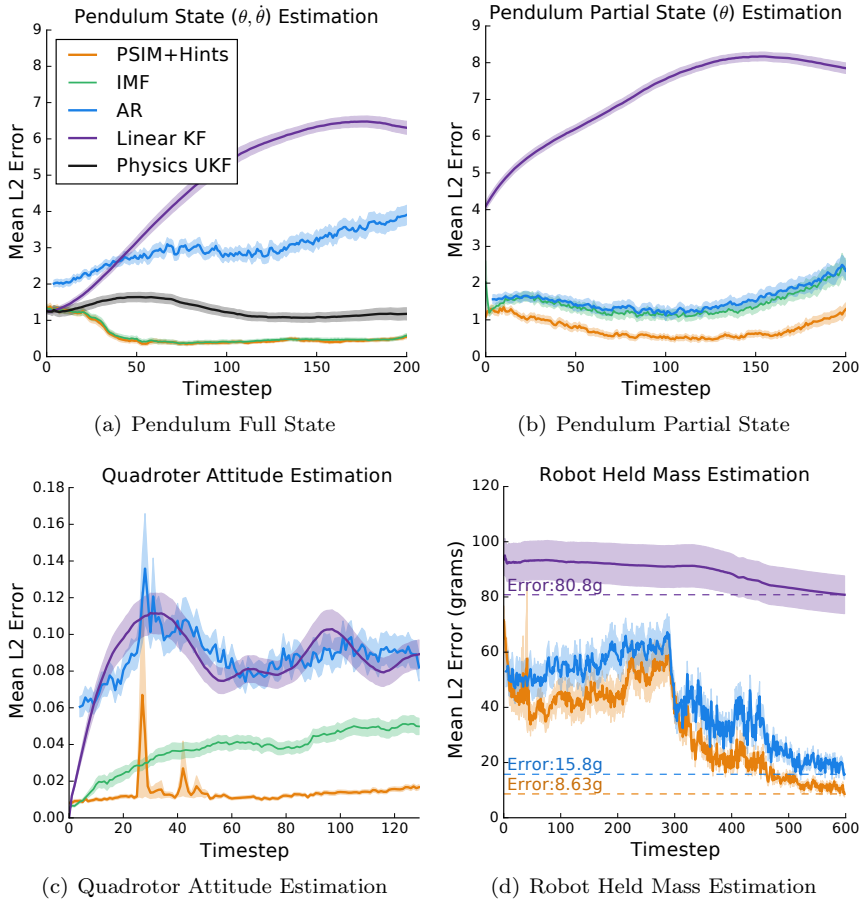
Table 3.8: Performance on mass estimation task

disturbance. The quadrotor runs a hover controller in a Vicon capture environment, shown in Fig. 3.23(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. 3.23(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.

3.4.4 Discussion & Conclusion

In all of the experiments, IMF and PSIM+HINTS outperform the baselines. Table 3.6 (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. 3.24(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 3.6) 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



(a) Pendulum Full State

(b) Pendulum Partial State

(c) Quadrotor Attitude Estimation

(d) Robot Held Mass Estimation

Figure 3.24: 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.

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 3.6's right column (or Fig. 3.24(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 3.7. In Fig. 3.24(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 3.8. 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. 3.24 show the relative stability of the inference machine filters even over large time horizons.

3.4.5 Conclusion

In this section, 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 to get the hints or state values 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.

The architectures for PSIM and PSIM+Hints are reminiscent of recurrent neural network architectures. How can we get the best of PSIM and of RNNs? How can we extend PSIM to work with controlled dynamics systems with actions? Can these be applied for reinforcement learning? We postulate and briefly investigate ideas for these future works in the next section.

Chapter 4

Proposed Work

In our prior work, we developed an optimization strategy, DAD, for multi-step prediction can be used to improve general learned time-series models. We then explored predictive state representations and developed an algorithm, PSIM, that directly optimizes for filtering performance on partially observed systems. The later work, the key insight was to provide supervision in terms of observable quantities. Here, we look to extend and adapt this idea to new problems for filtering and control.

4.1 Structured training of RNNs with PSIM

Our introduction of PSIMs in Sections 3.3 and 3.4 showed that we can achieve good performance by using Predictive State Representations (PSRs). As mentioned before, in the partially observed setting, one may instead choose a latent variable representation, such as an Hidden Markov Model (HMM) (Fig. 4.1). However, traditional methods for learning an HMM usually require assuming a fixed representation for the latent state (e.g. knowing the true state representation). This often limits performance as we don't know the true structure. Therefore, it common to utilize a more open-ended structure for problems

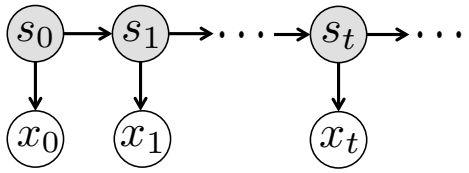


Figure 4.1: A Hidden Markov Model has some latent state s_t which generate observations x_t.

such as speech recognition (Graves and Jaitly, 2014), text generation (Sutskever et al., 2011), or generating images (van den Oord et al., 2016) – problems for

which it is difficult to parametrize and define a sufficient latent state. These approaches rely on a learning architecture known as the Recurrent Neural Network (RNN) (LeCun et al., 2015). The RNN representation shown in Fig. 4.2 is drawn to model the network as it would be used for a filtering task such as the one PSIM is used for in Section 3.3.

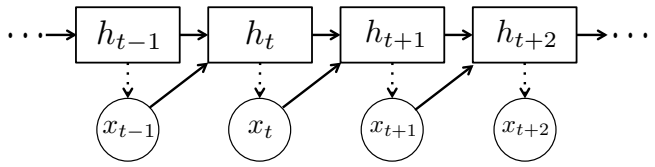


Figure 4.2: Simple recurrent neural network (RNN) structure. A hidden state h_t is propagated over time based on the previous hidden state and an observations x_t. We may assume the hidden state captures the necessary information to generate the obsevations x_t (dashed lines).

This RNN models a system which contains a hidden state h_t that we assume satisfies the Markov property (the h_t here is equivalent to the s_t in Fig. 4.1). This hidden state may also be assumed to be sufficient for generating an observation x_t (i.e. $\exists\, g : h_t \mapsto x_t$) such as a sensor reading from a robot or the next letter when parsing text. The hidden state is propagated through time by a (usually) non-linear function $f : (h_t, x_t) \mapsto h_{t+1}$.

This function can be thought of as a filter-function fulfilling the same purpose as that which we learned with PSIM. Since we do not have access to h_t at training time, we must infer this from the only supervision which is provided, the sequence of observations themselves. Training this sort of model is superficially easy. For filtering problems, we can set up a minimization of the negative log likelihood of the observed x_t (Abbeel et al., 2005a; Vega-Brown and Roy, 2013; Ondruska and Posner, 2016).

$$\min_f \left[\mathcal{L} = -\sum_t \log P(x_t | x_{1:t-1}) \right], \tag{4.1}$$

where $P(x_t|x_{1:t-1}) = P(x_t|h_t)P(h_t|x_{1:t-1})$ and $P(h_t|x_{1:t-1})$ is recursively defined by the past hidden h_{t-1} and observation. Unlike PSIM, this loss function does not in general capture an error over the a *future* belief state but is equivalent to a $k = 1$ observable assumption. The optimization $\mathcal{L}$ is difficult in that the recurrence results in non-convexity in f. This is true even if f was a linear function (as discussed in Section 3.1). Thus, the common technique to do this optimization is backpropagation-through-time (BPTT) (Werbos, 1990). As mentioned in Section 2.1.2, this procedure is prone to failures during training due to vanishing or exploding gradients (Bengio et al., 1994; Sutskever, 2013).

Some of these training difficulties may be stem from the RNN having to learn both the hidden state representation as well as the dynamics function f. This relatively unstructured learning allows the RNN to learn potentially robust

representations but can also result in very poor training signal, especially in very partially-observed systems where the loss at each timestep is unable to give very much information. The poorly conditioned gradients may also be a result of poor initialization of the recurrent network as there is no suggestion as to the initialization of f.

Towards each of these two problems, we can try to apply our lessons learned from Chapter 3. We hypothesize the below:

> **H1.** *Statistics of the future sequence of observations provide a good training signal for learning the hidden state representations for RNNs.*
>
> **H2.** *Initialization of the transition function based on the predictive statistics structure can yield more stable or faster training.*
>
> **H3.** *Using predictive state for training signal will result in a stabler training procedure and may require much shorter truncated gradients.*

In our prior work Section 3.3, we made preliminary observations that the PSIM structured performed better than a simple recurrent neural network (Table 3.3). However, a large reason to use neural networks is that they scale to a variety of differentiable function types including those with convolutions, an important model class for computer vision tasks. This is much of the motivation for using a RNN for filtering by Ondruska and Posner (2016). We note that (Chung et al., 2015) has also proposed a more structured training as inspired by Kalman filtering though this model relies of variational-autoencoders of the current observation conditioned on the history to parametrize the latent state as opposed to statistics over the futures conditioned on the history. We have good reason to believe that the useful state information comes the belief state over statistics of the futures.

Thus, we wish to see if we can improve the performance of the RNN using the predictive state representation as training signal on the hidden state. That is, towards H1, we add as a regularization to Eq. (4.1),

$$\mathcal{R} = \|h_t - \phi([x_{t+1}, x_{t+2}, \ldots])\|_2^2, \quad where\ h_t = f(h_{t-1}, x_{t-1}). \tag{4.2}$$

And we optimize,

$$\min_f \mathcal{L} + \lambda\mathcal{R} \tag{4.3}$$

where λ is our regularization tradeoff. It will be interesting to investigate if this tradeoff should be annealed over time ($\lambda \to 0$) during training. Interestingly, Ondruska and Posner (2016) noticed in their training procedure that it was better to optimize a filtering-RNN to predict a single $k - step$ ahead future observation. Though their reasoning was application specific, we note that this would be a special case of the ϕ in the loss structure introduced above. When applying PSIM-inspired RNNs, we may need to develop additional types of statistics for various applications. For example, for text generation, it could be more useful to keep a track of future counts of characters (e.g. how many '(' vs.

')' characters are to be produced in the future) instead of a just the likelihood of characters on a finite horizon.

Our second hypotheses, H2, relates to the training-structure knowledge we bring to the training procedure with Eq. (4.2). It is easier related through example. Assume a feature function ϕ that tries to capture the mean of the next k observations. That is,

$$\phi_t = \phi([x_{t+1}, \ldots, x_{t+k}]) = [x_{t+1}, x_{t+2}, \ldots, x_{t+k}]^T \tag{4.4}$$

Then, if we believe our hidden state h_t should be close to this, then our function f should also have a structure that has a 1s on the diagonal as a shift operation to remove the first entry as we move forward in time (This creates a identity-matrix, "near-unitary" structure).

$$h_{t+1} \approx \phi_{t+1} \leftarrow f(h_t, x_t) = \underbrace{\begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & \ldots \\ \vdots & & & \ddots & \\ 0 & & \ldots & 0 & 1 \\ 0 & & \ldots & & 0 \end{bmatrix}}_{W_h} \begin{bmatrix} x_{t+1} \\ x_{t+2} \\ \ldots \\ x_{t+k} \end{bmatrix} + f'(h_t, x_t) \tag{4.5}$$

We add a non-linear term $f'(h_t, x_t)$, initialized to small values, and then jointly optimize W_h and f'. The hope is that by using statistics of future observations, we can seed the transition function around this. We may alternatively choose to train a function f using the PSIM with DAgger training and then train it fully using BPTT (this is similar to having very high λ in Eq. (4.3) and then decaying it instantly to 0. Even more boldly, we may decide to train the initial model with DAgger training within PSIM and apply BPTT to fine-tune or add additional non-linearities.

Our last hypotheses, H3 proposes that the additional training signal afforded by PSIM can result in a more stable training procedure. The hidden state is no longer as free to vary as before. Additionally, we hypothesize that by doing this, we are able to lower the length of the truncated-gradient (shortening the backprop-through-time horizon) to achieve as good overall performance. The intuition here is that a truncated gradient of length 1 yields effectively DAD style training, which PSIM does. With PSIM supervision, this proposed work aims to improve the training time and performance of traditional recurrent neural networks.

4.2 Modeling actions in Predictive State Inference Machines

Neither the PSIM architecture we introduced in Section 3.3 nor the extension in Section 3.4 accommodate controlled dynamical systems with actions. There are some subtleties that present some difficulty in trivially extending this framework

to work with actions. We address these and propose possible solutions. The extension of PSRs with actions has been considered in (Bowling et al., 2006; Boots et al., 2011b). Our discussion of the problem and hypothesized solutions extend upon these works[1].

Let us first try to naively consider applying PSIM with actions. To do this, we must alter the distribution tracked by the inference machine to be

$$P(\underbrace{x_t, a_t, \ldots, x_{t+k}, a_{t+k}}_{\text{future obs. and actions}} | \underbrace{h_t}_{\text{history}}). \tag{4.6}$$

where h_t is the history (not a "hint" from Section 3.4). This will work as is within the PSIM training and would allow us to learn a filter function F that captures the joint transition distribution of the natural dynamics of the system as well as the currently running policy[2].

However, a function filter function learned to step forward Eq. (4.6) could not be used to simulate the dynamics with a *different* policy than that used to collect the data. Therefore it could *not* be used to do policy improvement or other reinforcement learning tasks. In general, the problem is that we need to be able to decouple the effect of the policy from the natural dynamics and this is a difficult problem. This is more studied for linear systems (Ljung and McKelvey, 1996) with special projection operators but these do not trivially generalize for nonlinear dynamics. In discrete observation and action spaces, Bowling et al. (2006) provides methods to address this problem. However, this does not scale for continuous observations or actions.

4.2.1 Modeling Interventions

The real difficulty is that Eq. (4.6) is not the true predictive state representation. Following (Boots et al., 2011b), the predictive representation would be

$$\begin{aligned} &g(\underbrace{x_t, \ldots, x_{t+k}}_{\text{future}} | \underbrace{h_t}_{\text{history}}, \underbrace{do(a_t, \ldots, a_{t+k})}_{\text{intervention}}) \\ &= \prod_{i=0}^{k} p(x_{t+i} | h_t, \underbrace{x_t, a_t, \ldots x_{t-(i-1)}, a_{t-(i-1)}}_{\text{interventions \& observations so far}}). \end{aligned} \tag{4.7}$$

Note this quantity is *not* a proper distribution, i.e. does not integrate to 1. Of importance is that Eq. (4.7) has a $do(\cdot)$ versus Eq. (4.6) which does not. These quantities, though they look similar are not equal. The term intervention is used as these are actions to be executed regardless of the outcome observations. We now adopt PSR notation. Let $\tau = [x_t, a_t, \ldots, a_{t+k}, x_{t+k}]$ be a *test* which consists of an observation test $\tau^o = [x_t, \ldots, x_{t+k}]$ and an *intervention* test

[1] This proposed item will be joint work with Wen Sun.

[2] If the policy is non-stationary (e.g. non-blind policies, certain blind policies) then we may need different F_t for each timestep

$\tau^a = [a_t, \ldots, a_{t+k}]$ which is tested using the $do(\cdot)$ operator. Then, we can write the predictive state as

$$g(\tau^o | h_t, do(\tau^a)) \tag{4.8}$$

Note that g is still a function of history.

If we able to properly model g, then we can successfully predict future sequence of observations given *any* sequence of future actions. However, often when running a system to collect data, there is usually a policy π in the loop and we do not get to restart the system to do an intervention to collect data. Our goal is thus, given access to policy π, can we use samples from $P(x_t, a_t, \ldots, x_{t+k}, a_{t+k} | h_t, \pi)$ to estimate g.

To do this, we note that the real difference between g and the observed sample from Eq. (4.6) is that the intervention actions are not generated from some policy but are prescribed actions to take. For $k = 2$, we can enumerate the differences in how Eq. (4.6) is computed and Eq. (4.7) is computed: This

Step-forward	**Intervention** given $\tau, do(\cdot)$	**Sample Observed** from Eq. (4.6)
First observation	$p(x_t \mid h_t)$	$p(x_t \mid h_t)$
First action	$p(a_t = \tau^a[0]) = 1$	$p(a_t \mid h_t, x_t) = \pi(a_t \mid h_t, x_t)$
Second observation	$p(x_{t+1} \mid h_t, x_t, a_t)$	$p(x_{t+1} \mid h_t, x_t, a_t)$
Second action	$p(a_{t=1} = \tau^a[1]) = 1$	$p(a_t \mid h_t, x_t, a_t, x_{t+1}) = \pi(a_{t+1} \mid h_t, x_t, x_t, a_t, x_{t+1})$

Table 4.1: Intervention vs Sampling. The *do* specifies the action deterministically, so it is drawn from a degenerate distribution with probability 1 that it is chosen.

indicates to us that the significant difference is purely from the sampling of actions from the policy and thus if we can invert this then it can be corrected for. This indicates that importance sampling will likely play a role.

Given a test $\tau = [\tau^o, \tau^a]$, we can find the probability that the policy will takes actions τ^a,

$$\pi(\tau | h) = \prod_{i=1}^{k} \pi(a_{t+i} | h, x_t, a_t, \ldots x_{t+(i-1)}, a_{t+(i-1)}) \tag{4.9}$$

This value can be computed if we have access to the policy or at least the probability for the actions in τ^a as they were chosen. Next, we can rewrite Eq. (4.7) as

$$P(x_t, a_t, \ldots, a_{t+k}, x_{t+k} | h) = P(\tau^o, \tau^a | h) = P(\tau^o | h, \tau^a) P(\tau^a | h). \tag{4.10}$$

Then,

$$\begin{aligned} g(\tau^o | h, do(\tau^a)) &= \frac{P(\tau^o, \tau^a | h)}{\pi(\tau | h)} = \frac{P(\tau^o | h, \tau^a) P(\tau^a | h)}{\pi(\tau | h)} \\ &= P(\tau^o | h, \tau^a) \frac{\prod_{i=1}^{k} P(a_{t+i} | h, a_t, \ldots, a_{t+(i-1)})}{\pi(\tau | h)} \end{aligned} \tag{4.11}$$

However, the numerator entity $P(\tau^a|h) = \int_{\tau^o} P(\tau^0, \tau^a|h)d\tau^o$ is generally difficult to evaluate as we would have to integrate out over future observations τ^o which would require knowledge of the dynamics. It is useful to note that for a blind policy, which is defined as one where the actions are conditionally independent of previous *observations* given all past actions, the above simplifies cleanly, $P(\tau^a|h) = \pi(\tau|h)$ (not a function over the observations). For blind-policies, we can then construct a conditional probability table $P(\tau^o|h, \tau^a)$. A non-blind policy is one where the blind assumption is not true; the policy is a function of the full history of actions and observations. These policies are more common in practice, e.g. proportional feedback controller. Thus, while Eq. (4.11) has developed a relation between Eq. (4.6) and the predictive state representation g, it is not yet usable for non-blind policies.

With these presented, we now try to construct the feature representation to create PSIM+Actions. Let us define a distribution G from g such that,

$$G_h(\tau) = \frac{1}{Z_h} g(\tau^o|h, do(\tau^a)), \tag{4.12}$$

where Z_h is the required normalization factor and is likely very hard to compute. Then for PSIM+Actions, we wish to track statistics of this,

$$\begin{aligned}
\mathbf{E}_{\tau\sim G_h(\tau)}\left[\phi(\tau)\right] &= \frac{1}{Z_h}\int_\tau g(\tau^o|h, do(\tau^a))\phi(\tau)d\tau & (4.13)\\
&= \frac{1}{Z_h}\int_\tau \frac{P(\tau^o, \tau^a|h)}{\pi(\tau|h)}\phi(\tau)d\tau \quad (\because Eq.\ (4.11)) & \\
&= \frac{1}{Z_h}\mathbf{E}_{\tau\sim P(\tau^o,\tau^a|h)}\left[\frac{1}{\pi(\tau|h)}\phi(\tau)\right] & (4.14)
\end{aligned}$$

The distribution $P(\tau^o, \tau^a|h)$ is sampling distribution from our policy. To get the estimator above of $\mathbf{E}_{\tau\sim G_h(\tau)}\left[\phi(\tau)\right]$, we have effectively used the importance sampling transformation with importance weights as $(\pi(\tau|h))^{-1}$ (note the similarity to Table 4.1). There is one subtle distinction between this transformation and general importance sampling – the Z_h term which is function of the history. We can at this point train a non-stationary filter function to learn PSIM+Actions, i.e. $\phi(\tau_{t+1}) = F_t(\phi(\tau_t))$. The concern is then, can we show there exists a stationary filter function? We believe there is. We can show that our representation $\mathbf{E}\left[\phi_t|h_t\right]$ can be converted to the PSR predictive state q_t. That representation can be forward propagated with a stationary function (by assumption) given a new action and observation (Bowling et al., 2006) to give the next PSR state q_{t+1}. From this, we can also construct the required stationary function to find $Z_{h_{t+1}}$ to convert q_{t+1} back to our representation to get $\mathbf{E}\left[\phi_{t+1}|h_{t+1}\right]$. Thus, the composition of these stationary functions is also stationary.

H4. *With access to the stochastic policy, importance sampling can allow us to learn* Predictive State Inference Machine *filters with actions.*

Importance sampling has been useful for off-policy learning for Temporal Difference methods (Precup et al., 2001). As standard importance sampling is known to have limited practicality due to large variance estimates, it would be useful to investigate the applicability of using weighted importance sampling (WIS) (Mahmood et al., 2014). If this proposed idea works, then it should also be applicable for doing reinforcement or imitation learning; the predictive state in PSIM+Actions may allow for more tractable reinforcement learning (e.g. Q-learning) in partially observable settings.

4.3 Recurrent Models for Model-free RL

In this last theme of proposed work, we look at how we may approach learning for time-series with a different goal in mind. Much of the Completed Work and other proposed works focus on learning to accurately predict a time series into the future either for forward prediction or for filtering. In this section we focus on how we can learn time series models directly for reinforcement learning. This is different than most prior work (including Section 3.2) which focuses on learning the dynamics model and then utilizing it for control (i.e. model-based reinforcement learning [MBRL]). Unlike MBRL, the proposed work belows focuses the the dynamics learning in a way that only targets control performance even at the expense of the time-series model quality. These proposed approaches straddle the line between model-free and model-based methods for RL, trying to pull in the best of both worlds – the sample efficiency of model-based methods with the discriminative performance of model-free.

We focus much of discussion on learning function approximators for Q, the action-value function, a method that has had recent success (Mnih et al., 2015) and provide a brief summary of it here. The key idea behind Q-learning is that the optimal true action-value function Q^* behaves according to the Bellman operator,

$$Q^*(s_t, a_t) = r_t + \gamma \max_a Q^*(s_{t+1}, a_{t+1}), \quad \textit{where } s_{t+1} = f(s_t, a_t)$$

where f the unknown system dynamics. Since we know the true action-value function obeys the Bellman operator, we define the Bellman residual on Q^*:

$$\ell_t(Q^*) = \|Q^*(s_t, a_t) - (r_t + \gamma \max_a Q^*(s_{t+1}, a_{t+1}))\|_2^2 \tag{4.15}$$

which we optimize for a given function parameterization (e.g. deep network) for Q^*. However, the Bellman residual for Q^* is difficult to optimize due to the max inside. Q-learning usually tackles this by first evaluating a target

$$y = r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})) \tag{4.16}$$

and then optimizing

$$\min_{Q^*} \left[\hat{\ell}_t(Q^*) = \|Q^*(s_t, a_t) - y\|_2^2 \right].$$

This objective is evaluated over a dataset D containing state-action-reward-next state tuples, $D = \{(s_t, a_t, r_t, s_{t+1})\}$. These are usually collected by running the 'argmax' policy $\pi = \hat{Q}^*(s_t, a_t)$ using one of the latest (there is usually some delay between the $Q*$ function being learned and being used as the policy (Mnih et al., 2015)) learned estimate of Q^*. Because of the way the problem is being optimized, samples from running multiple policies can be concatenated and used to learn Q^*.

4.3.1 Recurrent-Q SARSA

In many real world problems, it is often difficult to get access to the true underlying state s_t and we often only have access to an observation x_t from the system that is insufficient to predict the future. Latent state approaches, such as recurrent neural networks can be used to capture the dynamics. Often, the learning objective in using these latent state models is to learn the forward prediction of future observations (e.g. Eq. (4.1), Section 4.1); the added expressiveness to the learner from the hidden state is utilized to propagate the unobserved state in order to only better learn the underlying dynamics better.

However, an alternative approach is to use the expressiveness of the hidden state representation for explicitly achieving good control (Lin and Mitchell, 1992, 1993). The insight was that passing a memory-cell (hidden state) over time can allow more accurate Q values to be predicted, leading to better task performance.

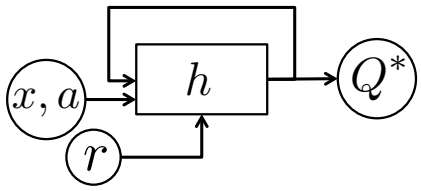


Figure 4.3: The recurrent-Q architecture from Lin and Mitchell (1992, 1993)

The recurrent architecture proposed for the Q-learning is shown in Fig. 4.3. It is not completely clear from (Lin and Mitchell, 1992) how to train the recurrent-Q network for Q-learning. Hausknecht and Stone (2015) modernized the recurrent-Q network by using Deep Learning and using LSTMs instead of simpler recurrent nodes, and the authors provide two different "boostrapping" training procedures for the recurrent network. In both, the key aspect of training is to roll forward the Q-network to produce training targets Eq. (4.16) at every timestep and use those for Q-learning. This training procedure, while following that of traditional Q-learning, is somewhat ill-fitted for the recurrent architecture, as we show below.

To observe the difficulty in doing Q-learning for the optimal Q^*, we unroll the network a few timesteps, noting that due to the recurrent hidden state, the

Q function must be time-indexed as well,

$$\ell_0 = \|Q_0^*(s_0, a_0) - (r_0 + \gamma \max_{a_1} Q_1^*(s_1, a_1))\|_2^2$$
$$\ell_1 = \|Q_1^*(s_1, a_1) - (r_1 + \gamma \max_{a_2} Q_2^*(s_2, a_2))\|_2^2$$
$$\ell_2 = \|Q_2^*(s_2, a_2) - (r_1 + \gamma \max_{a_3} Q_3^*(s_3, a_3))\|_2^2$$
$$\dots$$

Note here that the losses are all coupled as the depend on passing the Q_t function around including the hidden state, e.g. Q_2^* is dependent on Q_1^*. In regular Q-learning each datapoint $(s_t, a_t, r_t, s_{t+1}) \in D$ collected from running the real system can be used in the optimization of Q^* independently. Adding the recurrent architecture means the actions and state transitions must be temporally consistent. The action a_1 in the loss ℓ_1 must be the action taken in ℓ_0 and the a_2 in ℓ_1 must be the action taken in ℓ_2. To do this, we would need access to the underlying dynamics to generate the target state when rolling out with the $\arg\max_a Q(s, a)$ policy. However, in model-free reinforcement learning, we do not have access to a dynamics model with future state information. One could learn a dynamics model and use that to aid the Q-learning (Sutton, 1990), but this has failures from MBRL due to model bias (e.g. Gu et al. (2016) was able to use simple time-varying linear dynamics but was unable to get it work with neural networks.). Often (e.g. (Mnih et al., 2015)), we delay the Q function used by the policy from that which is currently being optimized. Thus, rolling out a recurrent Q-network in the "bootstrap" training (Hausknecht and Stone, 2015) is inconsistent since the observations and actions along the recorded trajectory in the dataset may not come from taking the arg max of the latest Q approximator, resulting in inconsistency in the propagated hidden state for the dynamics. Even without delayed updating of the policy, this would still be a problem for data aggregation across episodes. With this knowledge, we propose a more suitable training architecture for recurrent-Q networks,

H5. *SARSA is a suitable algorithm for recurrent-Q architectures.*

SARSA (Sutton and Barto, 1998) is a algorithm that iteratively optimizes for the *on-policy* Q-function Q^π and then does a policy improvement step. Q^π is optimized by minimizing the following Bellman error,

$$\ell_t(Q^\pi) = \|Q^\pi(s_t, a_t) - (r_t + \gamma Q^\pi(s_{t+1}, \pi(s_{t+1})))\|_2^2 \tag{4.17}$$

Note that Eq. (4.17) for Q^π does not have the max as we had for Q^* in Eq. (4.15). On-policy Q fitting is in many ways more attractive than Q-learning due to this. The lack of a max operator often makes the learning more stable. SARSA is so named as it uses a different dataset than Q-learning. Its dataset D is constructed from state-action-reward-next state-*next action* (i.e. SARSA) tuples, $D = \{(s_t, a_t, r_t, s_{t+1})\}$. Since we are learning Q^π, we have access to the next action that was taken under π from the training trajectories. Following Sun

et al. (2016), who found a relationship between optimizing the Bellman residual directly and the predictive error on the Q value, we aim in our proposal to tackle the problem of optimizing the Bellman residual instead of computing targets y (Eq. (4.16), albeit without the max) as is common for fitted-Q. Our proposed architecture for fitting Q^π with the Bellman residual rolled out a couple time steps is as below.

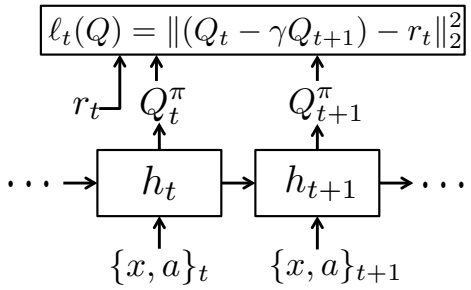


Figure 4.4: The on-policy recurrent-Q architecture proposed for SARSA.

Note that we can now allow observations x_t as a superset of just state s_t since the recurrent-Q has a hidden latent state. Finally we could augment our loss easily by considering various n-step Bellman residuals (Sutton, 1995) in our loss function to create larger gradients to backprop through time.

$$\ell_t = \lambda \sum_{n=0}^{k} \|Q_t^\pi(x_t, a_t) - (\sum_{i=0}^{n} \gamma^i r_{t+i} + \gamma^{n+1} Q_{t+n+1}^\pi(x_{t+1}, \pi(x_{t+1})))\|_2^2)$$

This is easily added to our network architecture and and can be easily optimized using backpropagation-through-time with packages such as Tensorflow. The final step in the SARSA algorithm relates to the policy improvement step. Given a learned Q^{π_i}, we can compute the next policy π_{i+1} by:

$$\pi_{i+1}(x) = \arg\max_a Q^{\pi_i}(x, a) \tag{4.18}$$

If we perfectly learn Q^π, we should have improvement by the Performance Difference Lemma (Bagnell, 2004). Finally, in order to get temporal consistency for the Q function, we would roll the Q function using π_i even though the actions chosen to run on the system is using π_{i+1}. It is this temporal consistency that differentiates this reinforcement learning approach from the prior work (Lin and Mitchell, 1992; Hausknecht and Stone, 2015). It may also be interesting to use the asynchronous training from (Mnih et al., 2016) since it works with on-policy reinforcement learning. It may be interesting to see if we can use a soft-max instead of the argmax policy and then use importance sampling in order to learn $Q^{\pi_{i+1}}$ with some of the data-set aggregation behavior suggested in (Ross and Bagnell, 2014).

4.3.2 Fitted-Q SARSA with Dynamics Objectives

We can extend the on-policy recurrent-Q discussed above to a spectrum of algorithms that move between targeting dynamics versus reducing the Bellman error. Many of these are similar to the approach described in Section 4.1. One the difficulties in learning recurrent architectures is the lack enough training signal for finding the appropriate evolution of the hidden state. The approaches we introduce below are similar to Dyna (Sutton, 1990; Gu et al., 2016) or the recurrent-model architectures (Lin and Mitchell, 1992). The main distinction is that we propose to *not* separate the learning of dynamics and the learning of the Q-function and aim to use the dynamics objective purely as guidance training signal. This is similar to Eq. (4.3),

$$\ell_t = \|Q^\pi_t(s_t, a_t) - (r_t + \gamma Q^\pi_{t+1}(s_{t+1}, \pi(s_{t+1})))\|^2_2 + \lambda \|\underbrace{f(h_t)}_{h_{t+1}} - s_{t+1}\|^2_2 \quad (4.19)$$

where we have regularized the Bellman residual with a dynamics prediction error. We would likely anneal the regularization over time ($\lambda \to 0$) to get the full expressiveness of the RNN for minimizing the Bellman residual.

> **H6.** *Predictive state statistics can help training of recurrent-Q networks on partially observed systems.*

Using observations x_t instead of state s_t we get

$$\ell_t = \|Q^\pi_t(x_t, a_t) - (r_t + \gamma Q^\pi_{t+1}(x_{t+1}, \pi(x_{t+1})))\|^2_2 + \lambda \|\underbrace{f(h_t)}_{h_{t+1}} - \underbrace{\phi([x_{t+1}, x_{t+2}, \ldots])}_{\phi_{t+1}}\|^2_2.$$

Yielding the network architecture in Fig. 4.5,

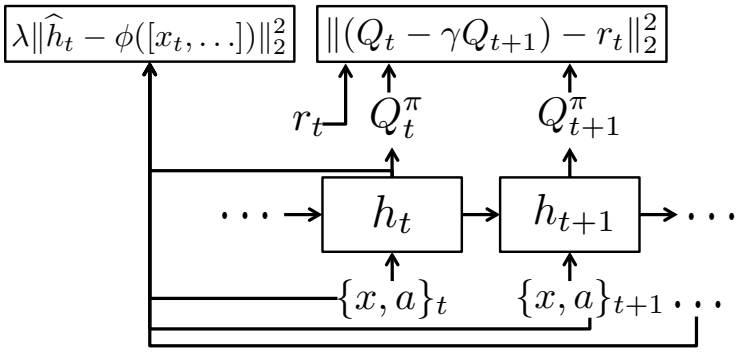


Figure 4.5: Recurrent-Q architecture augmented with predictive state dynamics objective.

Predicting future rewards (Predictive Reward Networks)

While the previous is well motivated by PSIM for dynamics and filter learning, it is not obvious that it is necessary. Is there another observable quantity that is more useful for reinforcement learning?

> **H7.** *A belief over statistics of future rewards may be sufficient hidden state for fitting Q functions.*

Intuitively, actions can be well selected for if we can predict the future sequence of rewards. Having a belief of a future sequence of rewards $r_t, \ldots, r_{t+k}$ should be enough to predict accurate Q values for the future (especially with *gamma* discounting). By using a hidden-state $h_t \in \mathbb{R}^k$ for the recurrent network, we can construct a loss of the form:

$$\ell_t = \|Q_t^\pi(x_t, a_t) - (r_t + \gamma Q_{t+1}^\pi(x_{t+1}, \pi(x_{t+1})))\|_2^2 + \underbrace{\lambda \sum_{i=0}^{k} \|h_t[i] - r_i\|_2^2}_{\text{Error on future rewards}}$$

A potentially other useful statistic is to use the hidden state to store a belief over future Q_t^π values for which we get ground truth from traces $\sum_t \gamma^t r_t$ while running the system under policy π.

In all of these setups, we are learning a new type of predictive state filter that learns a function F like PSIM which updates its belief over future Q values when we get an observation x_t from the system.

Chapter 5

Conclusion & Timeline

In this thesis proposal we showed that directly training prediction procedures for time-series and sequential prediction leads to good performance for forecasting, filtering, and reinforcement learning. The central idea to our work is that the prediction of observable quantities provides a practical method for building AI systems. We introduced several new algorithms including DAD and PSIM to address the limitations of existing machine learning methods for time-series problems. We proposed three extensions for the Proposed Work. The first proposed item looks to improve the training and performance of RNNs. The second extends PSIM to controlled dynamical systems. And the third investigates recurrent architectures for reinforcement learning problems.

5.1 Timeline

Below is an approximate timeline for completing the proposed work:

Task	**Section**	**Target Date & Deadline**
PSIM Training for RNNs	Section 4.1	May 2017, NIPS
PSIM+ACTIONS	Section 4.2	Jan. 2017, ICML
Recurrent-Q SARSA	Section 4.3	Jan.-Feb. 2017, ICML/RSS/IJCAI
Predictive State Fitted-Q	Section 4.3	Sept. 2017, AAAI
Thesis Defense	–	**Sept. 2017**

Bibliography

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.

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.

J. Andrew (Drew) Bagnell. *Learning Decisions: Robustness, Uncertainty, and Approximation*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2004.

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

Byron Boots, Sajid M Siddiqi, and Geoffrey J Gordon. Closing the learning-planning loop with predictive state representations. *The International Journal of Robotics Research*, 30(7):954–966, 2011b.

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

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.

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.

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.

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.

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. 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. 2003.

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

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. *arXiv preprint arXiv:1603.00748*, 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.

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, 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*, 2015.

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

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.

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.

J Ko, D J Klein, D Fox, and D Haehnel. GP-UKF: Unscented kalman filters with Gaussian process prediction and observation models. 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.

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.

Lucas Langer, Borja Balle, and Doina Precup. Learning multi-step predictive state representations. 2016.

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

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 nonlinear 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. 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.

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. 2007.

Lennart Ljung and Tomas McKelvey. Subspace identification from closed loop data. *Signal processing*, 52(2):209–215, 1996.

A Rupam Mahmood, Hado P van Hasselt, and Richard S Sutton. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems*, pages 3014–3022, 2014.

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.

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.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

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.

Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. Off-policy temporal-difference learning with function approximation. In *ICML*, pages 417–424, 2001.

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.

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.

Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

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.

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.

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.

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.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pages 216–224, 1990.

Richard S Sutton. Td models: Modeling the world at a mixture of time scales. In *ICML*, volume 12, pages 531–539. Citeseer, 1995.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

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

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

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*, 2016c.

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.