# Efficient Methods for Prediction and Control in Partially Observable Environments

Ahmed Hefny

April 2018
CMU-ML-18-101

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Thesis Committee:**
Geoffrey Gordon, Chair
Martial Hebert
Eric Xing
Byron Boots (Georgia Institute of Technology)

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



This research was sponsored by the National Science Foundation award IIS1450543, the US Army Research Office award W911NF0810301, the Air Force Research Laboratory award FA87501720152, and the Defense Advanced Research Projects Agency awards FA872105C0003 and FA870215D0002.

*For my Family*

# Abstract

State estimation and tracking (also known as filtering) is an integral part of any system performing inference in a partially observable environment, whether it is a robot that is gauging an environment through noisy sensors or a natural language processing system that is trying to model a sequence of characters without full knowledge of the syntactic or semantic state of the text.

In this work, we develop a framework for constructing state estimators. The framework consists of a model class, referred to as predictive state models, and a learning algorithm, referred to as two-stage regression. Our framework is based on two key concepts: (1) predictive state: where our belief about the latent state of the environment is represented as a prediction of future observation features and (2) instrumental regression: where features of previous observations are used to remove sampling noise from future observation statistics, allowing for unbiased estimation of system dynamics.

These two concepts allow us to develop efficient and tractable learning methods that reduce the unsupervised problem of learning an environment model to a supervised regression problem: first, a regressor is used to remove noise from future observation statistics. Then another regressor uses the denoised observation features to estimate the dynamics of the environment.

We show that our proposed framework enjoys a number of theoretical and practical advantages over existing methods, and we demonstrate its efficacy in a prediction setting, where the task is to predict future observations, as well as a control setting, where the task is to optimize a control policy via reinforcement learning.

# Acknowledgments

*"He is not thankful to God who is not thankful to people"* – Prophet Muhammad
It goes without saying that this work would have not seen the light without the help of many people I am greatly indebted to.

First, I would like to thank my advisor, Geoffrey Gordon, for guiding me along this journey. Geoff always made time for me, gave me a large amount of freedom while encouraging me to pursue interesting questions and carry out deep investigations, helped my with his insights and cared much about my personal development.

I would also like to thank Byron Boots, who alongside Geoff laid the foundations of this work, for his useful insights, detailed advice and hands-on collaboration. I am also grateful to Martial Hebert and Eric Xing for their time and advice that helped in shaping the thesis. In addition to being my thesis committee, I benefited greatly from Geoff and Eric as the course instructors of Convex Optimization, Machine Learning and Graphical Models which had significant contributions to my knowledge and way of thinking. I am also grateful to Manuel Blum, Christos Faloutsos, Jing Lei, Roy Maxion, Gary Miller, Bhishka Raj, Ruslan Salakhutdinov, Alex Smola, Suvrit Sra, Ryan Tibshirani and Larry Wasserman for the wide array of beneficial courses.

The work in this thesis is a result of joint collaboration with Byron Boots, Carlton Downey, Boyue Li, Zita Marinho, Wen Sun and Sidd Srinivasa. I also had the opportunity to work with Avinava Dubey, Anika Gupta, Robert Kass, Sanjeev Khanna, Lin Ma, Gustavo Mezerhane, Dianna Needell, Andrew Pavlo, Aaditya Ramdas, Sashank Reddi, Matthew Smith, Alexander Smola, Suvrit Sra, Katia Sycara, Dana Van Aken, Sinead Williamson and Eric Xing on projects that I am proud of but they are not addressed in this dissertation. I am thankful to my office mates: Maruan Al-Shedivat, Avinava Dubey, Yifei Ma, Aaditya Ramdas, Sashank Reddi and Xun Zheng and all my colleagues for the countless interesting discussions. I would also like to thank the CMU staff for working hard to maintain a smooth experience from day one. Specially thanks to Diane Stidle for everything she did for the MLD department and for me.

I was fortunate to have a group of dear friends who enriched my stay in Pittsburgh. I am especially thankful to Waleed Ammar, Mohammed Darwish, Mohammed AbdelMoula, Mazen Soliman, Islam Shehab, Moataz Mohsen, Mohammed Mahmoud, Ahmed AbdelGawwad, Ashraf Gamal, Khaled ElGaliand, Mahmoud Bishr, Emad El-Sayed, Ezz Naser and Zia Hydari.

I am also indebted to my mentors, Amir Atiya, Kareem Darwish, Ossama Emam and Hany Hassan, who sparked my interest in the field and led me to start this journey.

Finally, I have been gifted with a kind and supporting family. Who knows how I would have ended up without them? I am really thankful to my father Said Hefny, my mother Ahlam Hashem, my sister Nehal and my brothers Mohammed and Tarek for everything they did. I would like to thank my darling wife Nouran for her patience and support throughout all these years. In the end, I want to thank my little gems, Yusuf and Youmna for all the joy they added to my life.

# Contents

# List of Figures

# List of Tables

# Symbols and Notation

| | |
|---|---|
| **Sets and Sequences** | |
| $[N]$ | The set of positive integers up to and including $N$. |
| $\emptyset$ | Empty sequence. |
| $\mathcal{O}, \mathcal{A}$ | The set of possible observations (a.k.a the alphabet) and the set of all possible actions. |
| $\mathcal{O}^t$ | The set of possible observation subsequences of length $t$. |
| $\mathcal{O}^*$ | The set of possible observation subsequences of any length. |
| $x_{1:t}$ | A sequence of random variables (or values) $x_1, x_2, \ldots, x_t$. Unless otherwise specified, the sequence is represented as a matrix where $x_i$ is the $i^{th}$ column. |
| **Linear Algebra and Tensors** | |
| $\mathbf{1}_m$ | An all-ones vector of length $m$, we might drop the length if it is clear from the context. |
| $e_i^d$ | An indicator vector in $\mathbb{R}^d$ where the $i^{th}$ coordinate is 1 and other corrdinates are 0. We may drop the dimension fi it is clear from context. |
| $A_{i.}$ | The $i^{th}$ row of a matrix $A$. |
| $A_{(i:j).}$ | Rows $i$ through $j$ of a matrix $A$. |
| $A^+$ | MoorePenrose pseudoinverse of $A$. |
| $\text{vec}(A)$ | Reshaping a matrix $A$ into a vector using column-major ordering. |
| $\otimes$ | Outer product (also known as tensor product). |
| $\otimes_k$ | Kronecker product. |
| $\circ$ | Hadamard (element-wise) product. |
| $\star$ | Khatri-Rao product (column-wise kronecker product). For two matrices $X$ and $Y$, $Z = X \star Y$ is a matrix such that $Z_{:,i} = X_{:,i} \otimes_k Y_{:,i}$. |
| $\|.\|_F$ | Frobenius norm. |
| $T(A, B, C)$ | Multinear multiplication of a 3-mode tensor $T \in \mathbb{R}^{m_a \times m_b \times m_c}$ with three matrices $A \in \mathbb{R}^{n_a \times m_a}$, $B \in \mathbb{R}^{n_b \times m_b}$ and $C \in \mathbb{R}^{n_c \times m_c}$. The result is a tesnor $Z \in \mathbb{R}^{n_a \times n_b \times n_c}$ such that $Z_{i_a,i_b,i_c} = \sum_{j_a,j_b,j_c} T_{j_a,j_b,j_c} A_{i_a,j_a} B_{i_b,j_b} C_{i_c,j_c}$. Intuitively, $T(A, I, I)$ is the result of slicing $T$ along the first mode and applying $A$ to each slice. We assume that the resulting modes of dimensionality 1 are "squeezed"– that is, for a vector $v$, we treat $T(v, v, v)$ as a scalar (rather than a $1 \times 1 \times 1$ tensor), $T(I, v, v)$ as a vector and $T(I, I, v)$ as a matrix. |

| Probability | |
|---|---|
| $1(z)$ | Indicator function of an event $z$: $1(z) = 1$ if $z$ is satisfied, otherwise 0. |
| $\Pr(x \mid y, \mathbf{do}(z); \theta)$ | Probability of $x$ given that the event $y$ is observed and the event $z$ is forced by intervention, as determined by parameters $\theta$. Depending on the context, "probability of $x$" can refer to the probability of an event $x$, a probability density function evaluated at the value $x$ or the entire proability distribution of a random variable $x$. |
| $\mathcal{C}_X$ | Uncentered covariance of a random variable $X$. When operating in a kernel feature space, we may drop the feature map $\phi$ when it is clear from the context (i.e. we use $\mathcal{C}_X$ to refer to the covariance operator $\mathcal{C}_{\phi(X)}$). This applies to all covariance and conditional expectation operators defined below. |
| $\mathcal{C}_{X,Y}$ | Uncentered cross-covariance of a two random variables $X$ and $Y$. |
| $\mathcal{C}_{X,Y\|z}$ | Uncentered cross-covariance of a two random variables $X$ and $Y$ given an event $z$. |
| $\Sigma_X$ | Centered covariance of a random variable $X$. |
| $\Sigma_{X,Y}$ | Centered cross-covariance of a two random variables $X$ and $Y$. |
| $\Sigma_{X,Y\|z}$ | Centered cross-covariance of a two random variables $X$ and $Y$ given an event $z$. |
| $X \sim f(Y)$ | A random variable $X$ is sampled from a distribution whose parameters are determined by $f(Y)$. |
| $\mathcal{W}_{X\|Y}$ | Conditional expectation operator: $\mathbb{E}[X \mid Y = y] = \mathcal{W}_{X\|Y} y$. |
| $\mathcal{W}_{X\|Y;z}$ | Conditional expectation operator given an event $z$: $\mathbb{E}[X \mid Y = y, z] = \mathcal{W}_{X\|Y;z} y$. An example is a conditional probability table whose elements are determined by $z$. |
| $\mathrm{KL}(p \mid\mid q)$ | KL-divergence between two distributions $p$ and $q$. |
| **Predictive State Models** | |
| $h_t^\infty$ | The entire history of observations and actions before time step $t$ in a controlled system: $h_t^\infty \equiv o_{1:t-1}, a_{1:t-1}$. |
| $h$, $h_t$ | History feature function and the value of the function at time $t$ (i.e. $h(h_t^\infty)$), respectively. |
| $\psi^O$, $\psi_t^O$ | Future observations feature function and the value of the function at time $t$ (i.e. $\psi^O(o_{t:\infty})$), respectively. For uncontrolled systems, we may drop the superscript. |
| $\psi^A$, $\psi_t^A$ | Future actions feature function and the value of the function at time $t$ (i.e. $\psi^A(a_{t:\infty})$), respectively. |
| $\xi^O, \xi^A, \xi_t^O, \xi_t^A$ | Extended future observation feature function and extended future actions feature function and their values at time $t$, respectively. For uncontrolled systems, we may drop the superscript. |

| Tensor Sketching | |
|---|---|
| $\mathfrak{h}, \zeta$ | Bucketing and sign hash functions. |
| $s_x^{(i)}$ | Sketch of a vector $x$ using hash functions $h_i$ and $\zeta_i$. |
| $*$ | Circular convolution. |
| $\overline{x}$ | Complex conjugate of $x$ (unless otherwise specified). |
| $\mathcal{F}$ | Fourier transform. |
| $\mathfrak{b}$ | Size of the sketch. |
| $\mathfrak{B}$ | Number of sketches. |
| **Reinforcement Learning** | |
| $\gamma$ | Discount factor |
| $R_t$ | Reward to go: $\sum_{t' \geq 0} \gamma^{t'} r_{t+t'}$ |
| $b$ | Reward baseline |

# Part I

# Introduction and Background

# Chapter 1

# Introduction

## 1.1 Problem Statement

Sequential data is ubiquitous. In natural language processing, we deal with sequences of letters that reflects the syntactic and semantic state of the speaker/writer. In robotics, we deal with sequences of noisy sensor observations that reflect the physical state of the robot. In computational neuroscience we deal with sequences of neural spikes that reflect the neurophysiological state of the brain. Therefore, modeling and reasoning about sequential data is of extreme importance in machine learning. A powerful modeling tool for such data is a dynamical system, where the data generator (i.e. the system or the environment) has a state that evolves over time according to some dynamics. The observed sequential data are generated by applying a typically stochastic function to the system's state at each time step. The dynamical system can be uncontrolled or controlled. In an uncontrolled system we can only interface to the system by receiving observation sequences. In a controlled system, we can effect evolution of the state through exogenous inputs, which we refer to as *actions*.

System identification refers to the process of learning a state representation and dynamics of a dynamical system solely from observation (and action) sequences. Accurate system identification is important for tracking the state of the system, predicting future observations and planning control actions. In this work we care mainly about the problem of state tracking or *filtering*—that is, maintaining a belief about the state of the system given the history of observations (and actions). This belief can then be used as an input to a predictor or a controller. The most commonly used construction for filtering is the *recursive filter*, which recursively updates the belief at time step $t$ given the belief at time step $t-1$ as well as the observation (and action) at time $t$. Thus, the main goal of the thesis is to develop a framework for constructing a recursive filter given observation sequences generated from a dynamical system. To formulate a framework, we specify a class of dynamical systems and an algorithm or a meta-algorithm for identifying their parameters and constructing the corresponding recursive filter. For example, two existing frameworks are (1) hidden Markov models (HMM) together with the expectation maximization (EM) algorithm and (2) linear Markovian models together with autoregressive least squares.

### 1.1.1 Desired Qualities of System Identification

There are a number of qualities that we would like our framework to attain:

**Partially observable vs fully observable state**
: Auto-regressive models assume that the state of the system is fully determined by a finite history of observations and actions. In other words, it is sufficient to maintain a finite history to make optimal predictions of future observations. We are interested in partially observable models, where optimal predictions depend on the entire history of observations (and actions). These include hidden Markov models and linear Gaussian state space models.

**Ability to represent controlled and uncontrolled systems**
: We would like to have a framework that can be adapted to both uncontrolled and controlled systems. Learning a state representation and dynamics of a controlled system is an essential tool for planning, imitation learning and reinforcement learning in partially observable environments.

**Computational efficiency and scalability**
: We would like the framework to be scalable to large datasets and realistic high-dimensional data. A minimum requirement for scalability is that the processing time and memory requirements for training should scale at most linearly with the size of the training data, and the inference time should be independent of that size.

**Theoretical guarantees**
: Some desirable theoretical properties include consistency, finite sample error guarantees for parameter estimation and for predictions, and agnostic error bounds.

**Modeling flexibility**
: System identification frameworks differ by the customization options they permit, such as the choice of features, objective function and regularization, the ability to handle non-linear dynamics, the ability to handle discrete, continuous or mixed systems, and the ability to incorporate other learning techniques.

The main goal of this thesis is push the Pareto frontier of system identification methods with respect to the aforementioned qualities, by developing models that can model systems that are partially observable, continuous, non-linear and controlled while providing an efficient learning algorithm with consistency guarantees.

## 1.2 Proposed Framework

To achieve the thesis goal, we propose a framework for learning latent-state dynamical systems that is based on *predictive state models* as a model class and *two-stage instrumental regression* as a learning algorithm. The proposed framework relies on two main principles.

- **State as a prediction of future observations:** Unlike hidden Markov models and linear Gaussian state space models, where the latent state is explicitly represented by additional latent variables, predictive state models represent the state as a prediction of sufficient future observation statistics. This way, the system identification problem is specified in terms of

quantities that can be estimated from observed data. This principle is the basis of predictive state representations (Singh et al., 2004) and observable operator models (Jaeger, 2000).

- **History as a noise removal tool:** With the predictive state representation, observation statistics are unbiased estimates of the state, which suggests that we can use supervised regression to learn system dynamics. However, to get an unbiased estimate of the dynamics, one needs first to remove the sampling noise from regression inputs.

  Instrumental regression exploits the fact that the sampling noise at a particular time is independent of the history of previous observations. Based on this fact, we employ a two-stage regression approach, where we use a regression model to remove sampling noise using history features, then we use another regression model to learn system dynamics.

We show that this framework encompasses many of the existing "spectral learning" algorithms for system identification while granting additional flexibility in choosing regression models. We also establish asymptotic and finite sample guarantees for recovering system dynamics.

We then extend this framework to controlled systems by defining the class of *predictive state controlled models* and show that it allows us to develop controlled system identification schemes that enjoy many of the aforementioned qualities. We demonstrate the efficacy of predictive state controlled models in the context of prediction and reinforcement learning.

## 1.3 Summary of Contributions

This thesis makes the following contributions:

- We propose the class of *predictive state models* together with the *two-stage regression* algorithm as a framework for learning uncontrolled dynamical systems.
- We show that the framework of predictive state models encompasses a wide range of spectral learning algorithms for dynamical systems and admits the development of novel and useful variations.
- We provide theoretical guarantees on the recovery of the parameters of a predictive state model using the two-stage regression algorithm.
- By combining predictive state models with techniques from kernel methods and recurrent neural networks, we propose a novel recurrent network architecture for modeling time series. The model, which refer to as *predictive state recurrent neural networks*, benefits from two-stage regression as an initialization procedure but can still be improved using typical neural network training algorithms.
- We demonstrate that we can use tensor sketching techniques to facilitate the training of predictive state recurrent networks with large state sizes while keeping the memory requirements reasonable.
- We extend our framework to controlled systems. We propose *predictive state controlled models*, to which we adapt the two-stage regression algorithm to produce a framework for learning controlled systems. As an instance of this framework, we propose *predictive state*

*controlled models with random Fourier features*, a model of controlled dynamical systems that satisfies the properties mentioned in the thesis goal.

- We empirically demonstrate the efficacy of predictive state controlled models in both prediction and reinforcement learning.

## 1.4 Thesis Organization

The thesis is organized as follows

- Chapter 2 presents an overview over dynamical system representations and learning algorithms, providing the necessary terminology to describe our work and its relation to the literature.
- Chapter 3 describes predictive state models, out proposed framework for learning uncontrolled dynamical systems by reduction to supervised learning. This chapter is based on (Hefny et al., 2015).
- Chapter 4 describes a non-parametric practical model for representing continuous dynamical systems with non-linear dynamics. This model combines the formulation of Chapter 3 with ideas from kernel methods, random projections and recurrent networks. This chapter is based on (Downey et al., 2017).
- Chapter 5 demonstrates training of large predictive state models with constrained memory budget using tensor sketching. (To be submitted).
- Chapter 6 extends the formulation of Chapter 3 to controlled dynamical systems, where exogenous inputs or actions can affect the system state and observations. This chapter is based on (Hefny et al., 2018a).
- Chapter 7 demonstrates the use of the framework proposed in Chapter 6 in a reinforcement learning setting, where we combine a state estimation model with a policy represented by a feed-forward network, and train the whole structure end-to-end using policy gradient methods. This chapter is based on (Hefny et al., 2018b).
- Chapter 8 presents conclusions and future work.

## 1.5 How to Read This Document

The dissertation is written with the assumption of the natural sequential reading order. However, for readers with a specific purpose there can be more efficient reading plans.

Chapter 2 is not just meant to provide the necessary background for the rest of the dissertation. Rather, it is written with the **readers who are generally interested in a high-level overview of dynamical system models and learning algorithms** in mind. The chapter gives more focus on high-level concepts and on connections between different types of models and learning algorithms. We may not cover specific models such as auto-regressive models and Gaussian processes. However, the categorization and general discussion presented in Chapter 2 is still applicable to these

models.

In the following reading plans, which assume specific goals other than an overview of dynamical system models, Chapter 2 can be consulted on a need-to-know basis. The main concepts we expect a typical reader to check are those of *belief state*, *predictive state* and *system observability*.

For **readers interested in our recommended approach for prediction in uncontrolled environments**, we recommend reading Sections 3.1 and 3.2 followed by Chapter 4.

For **readers interested in our recommended approach for prediction in controlled environments**, we recommend reading Section 4.1 for a background on some core mathematical tools that we use. With this background, Chapter 6 should provide a sufficient description of the model and the learning algorithm. However, Chapter 3 is needed for understanding the theoretical foundation and guarantees.

Finally, **readers interested in our recommended approach for reinforcement learning** can start with Chapter 7 for a description and evaluation of the overall approach. The details of the recursive filter and the two-stage regression initialization algorithm can be obtained from Chapter 6.

## 1.6 Disclaimer

We developed our proposed models for this work with a focus on improving the criteria in Section 1.1.1. We acknowledge that there are other criteria that we do not address.

One such criterion is wall clock time. While our proposed models satisfy the required computational constraints, the best performing models rely on operations that are relatively costly such as matrix inversion and tensor-vector products. We demonstrate that our proposed methods are superior to other methods in the literature in terms of predictive performance. However, we acknowledge that there are applications where every microsecond matters and one is willing to sacrifice some predictive performance for the sake of speed. This tradeoff is abundant in the field.

Another criterion is the ability to encode prior knowledge about the system dynamics. Our best performing models are designed for the situation where we have few assumptions about the system. The proposed frameworks supports injecting prior knowledge to some degree (as we show in Chapter 3). However, we do not provide direct support for prior knowledge in the form of "the system follows this stochastic difference equation" or the "recursive filter update takes this parametric form."

For applications where one of the aforementioned criteria is critical, the models proposed in this dissertation may be suboptimal.

# Chapter 2

# Background: Dynamical Systems and Recursive Filters

In this chapter we present an overview of various dynamical system representations, how to obtain recursive filters from them and how to optimize their parameters. In doing so we present some concepts that will be crucial for later chapters such as Bayes filters, predictive states and method of moments. We will also provide a categorization of recursive filter models and algorithms that allows us to situate our contribution compared to the literature. For simplicity, this chapter focuses on uncontrolled systems. Controlled systems, where the system can be affected by actions, are discussed in Chapter 6.

This chapter is composed of three main parts. First, we identify different formulations of dynamical systems (Section 2.1). Second, we discuss how these formulations are related to recursive filters, our main subject of interest (Sections 2.2 and 2.3). Third, we overview different methods of unsupervised learning of recursive filters (Sections 2.4 and 2.5). Based on these components, we conclude the chapter with a categorization of recursive filter learning methods and specify where our proposed work fits compared to the literature.

## 2.1 Three Views of Dynamical Systems

We first go through different formulations of dynamical systems. An uncontrolled stochastic dynamical system describes a probability distribution over sequences of observations $\Pr(o_{1:t})$.[1]

### 2.1.1 System State View

The traditional way of describing a dynamical system is through a notion of a stochastic state that evolves through Markovian dynamics,

$$x_t \sim f(x_{t-1}), \tag{2.1}$$

[1]We write $\Pr(o_{1:t})$ to denote the probability of observing a sequence starting with $o_{1:t}$. In that sense the probability of the empty sequence $\Pr(\emptyset)$ is 1.

where $x_t$ determines the observation $o_t$ but can also include additional latent information. It is common to factorize (2.1) into a state evolution part and an observation emission part.

$$s_t \sim f(s_{t-1})$$
$$o_t \sim g(s_t),$$

where $s_t$ is the *system state*, $f$ is a *state evolution function* and $g$ is an *observation function*.

The system state view is beneficial if we have prior insights on the underlying process that generates the data. For example, knowing the physical laws that govern the underlying process, it is typically easier to deduce the state evolution equations of a linear dynamical system (LDS) than to deduce the belief update equations. Another common case is the use of hidden Markov models to encode the insight (or assumption) that the system moves between a set of discrete states with their interpretation known beforehand (as in speech recognition (Rabiner, 1990)) or inferred from the model parameters. The use of system state view can also greatly simplify learning the dynamical system model if we have access to the system state at training time.

### 2.1.2 Belief State View

This view defines the probability $\Pr(o_1, o_2, \ldots, o_t)$ by defining the conditional probability

$$\Pr(o_t \mid o_{1:t-1}) = \Pr(o_t \mid q_t), \tag{2.2}$$

where $q_t$ is a *belief state* that is a deterministic function of all previous observations $o_{1:t-1}$. The belief state $q_t$ serves as a bottleneck that compresses the entire observation history for the purpose of future predictions (see Figure 2.1).

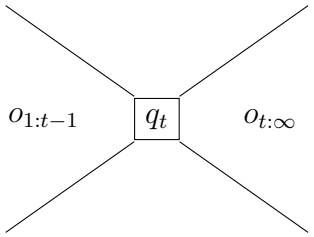



Figure 2.1: Belief state as a bottleneck: A perfect belief state preserves sufficient information about the history to make any future prediction (i.e. $\Pr(o_{t:\infty} \mid o_{1:t-1}) = \Pr(o_{t:\infty} \mid q_t)$.

Typically, the dynamical system is described as a recursive update function of the belief state,

$$q_{t+1} = f(q_t, o_t)$$
$$o_t \sim g(q_t), \tag{2.3}$$

where $f$ is a *belief state update* function and $g$ is an observation function.

This update can be thought of as an alternative factorization of the data generation process in (2.1), but it can also be thought of as a method to track the state given existing observations. In other words, it defines both a data generation process and an inference algorithm.

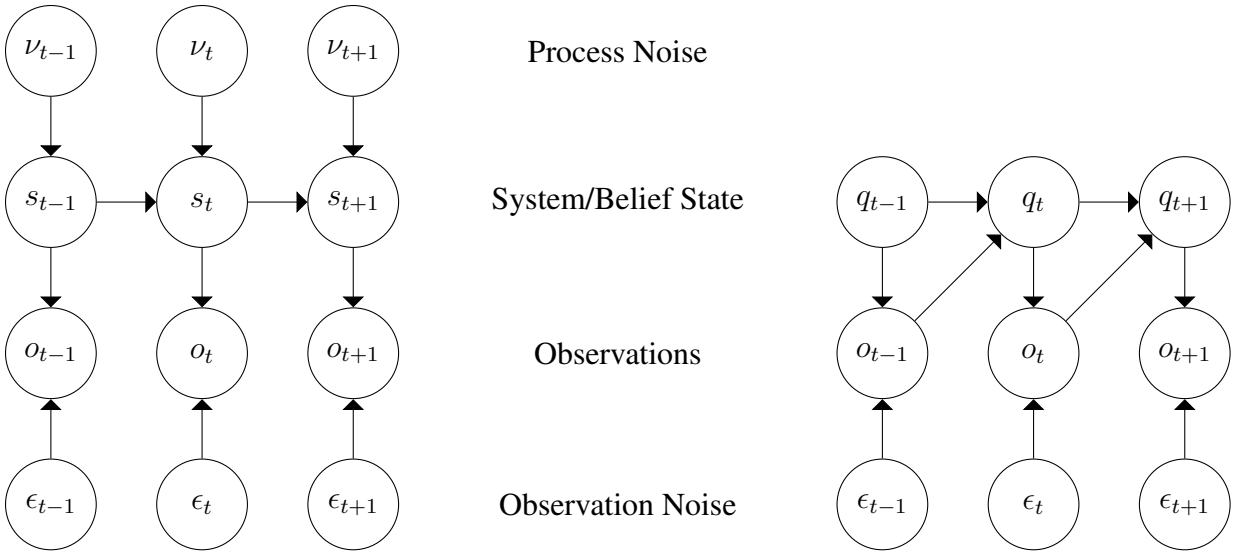



Figure 2.2: Graphical models for system state and belief state representations of dynamical systems. Each variable is fully determined by its parents. For ease of exposition, we assume $\epsilon_t$ and $\nu_t$ to be sampled independently although this is not necessary.

Figure 2.2 depicts the graphical models for system and belief state views. The main difference between the system state $s_t$ and the belief state $q_t$ is that the system state is a function of the history of observations and process noise, whereas the belief state depends only on the history of observations.

The belief state view can be derived from other views as we demonstrate in Section 2.3, or it can be formulated from scratch. An example of the latter case is recurrent neural networks (e.g. Elman networks (Elman, 1990), long short-term memory networks (Hochreiter and Schmidhuber, 1997) and gated recurrent units (Cho et al., 2014)), where the functions $f$ and $g$ are composed of learnable affine transformations and fixed non-linearities. Other examples are sufficient posterior representations (Langford et al., 2009) and predictive state inference machines (Sun et al., 2016), which we discuss in Section 2.5.2. These two approaches construct $f$ and $g$ from standard regression models.

#### 2.1.2.1 Sufficient State Representation and System Observability

In order for (2.3) to be true for all $t \geq 1$, the belief state $q_t$ must constitute a *sufficient state representation* such that

$$\Pr(o_{t:\infty} \mid o_{1:t-1}) = \Pr(o_{t:\infty} \mid q_t),$$

In other words, given a sufficient state representation $q_t$, additional information on the history of observations does not improve *any* predictions of the future.

The sufficient state representation condition can be expressed as a representation that maintains

mutual information between history and future

$$I(o_{t:\infty}; q_t) = I(o_{t:\infty}; o_{1:t-1})$$

The information-theoretic perspective is useful: since $q_t$ is a deterministic function of history, the information processing inequality tells us that the RHS is an upper-bound of the LHS. Therefore, finding a sufficient state representation problem can be posed as a maximization of $I(o_{t:\infty}; q_t)$ (Wingate and Singh, 2007).

An example for a sufficient state representation is the probability vector $b_t \equiv \Pr(s_{t-1} \mid o_{1:t-1})$, where $s_t$ is the system state of a hidden Markov model. This probability vector can be recursively updated using the forward algorithm.

An important notion related to sufficient state representation is system observability. A dynamical system is $k$-observable if the posterior distribution of the future $k$ observations $\Pr(o_{t:t+k-1} \mid o_{1:t-1})$ constitutes a sufficient state representation. It is worth noting that a $k$-observable system is different from a $k^{th}$ order Markov chain. The latter means that the actual previous $k$ observations $o_{t-k:t-1}$ constitute a sufficient state representation.

Consider, for example, the pendulum shown in Figure 2.3, where observations are angular positions. Since the system state of the pendulum is fully determined by angular position and velocity, the system is 2-observable (even in the presence of e.g. Gaussian noise). The system is a 2nd order Markov chain only in the noiseless case.

An example of a 1-observable system in an HMM where the observation and transition matrices have full column rank. In this case, a probability distribution over the next observation $\Pr(o_t \mid o_{1:t-1})$ can be uniquely mapped back to a probability distribution over the latent state $\Pr(s_{t-1} \mid o_{1:t-1})$. Thus, the probability vector $\Pr(o_t \mid o_{1:t-1})$ is a sufficient state representation.

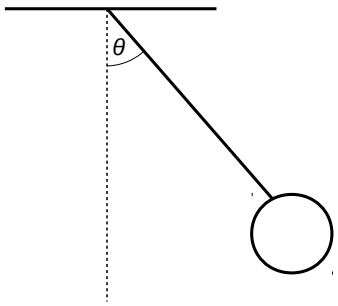



Figure 2.3: A swinging pendulum. Given (possibly noisy) readings of $\theta$, the system is not 1-observable since a position snapshot does not encode direction.

#### 2.1.2.2 Predictive Representation

We have mentioned that for a 1-observable HMM, two equivalent belief state representations are the probability vector $b_t = \Pr(s_{t-1} \mid o_{1:t-1})$ and $q_t = \Pr(o_t \mid o_{1:t-1})$. We refer to them respectively as a latent belief state and a predictive belief state. In general, we differentiate between predictive

and latent states as follows: for a latent belief state, there exists an observation function $g$ that has to be learned such that

$$o_t \sim g(b_t).$$

For a predictive state, on the other hand, there exists a *prespecified* feature function $\psi$ of future observations such that

$$\mathbb{E}[\psi(o_{t:\infty}) \mid o_{1:t-1}] = q_t.$$

In order for $q_t$ to be a sufficient state representation, the future features $\psi_t \equiv \psi(o_{t:\infty})$ must constitute sufficient statistics of the posterior distribution $\Pr(o_{t:\infty} \mid o_{1:t-1})$. The main advantage of a predictive state representation is that at any time $t$, the observable future features $\psi_t$ constitute an unbiased estimate of $q_t$. This direct connection between observable quantities and belief states is absent in the latent representation, and it drives a large class of consistent learning algorithms, as we demonstrate in Section 2.4.3.2 and Chapter 3.

It is important to acknowledge that the boundary between predictive and latent belief states can become fuzzy. In many practical cases, the feature function $\psi$ is chosen in a data-dependent fashion. For example, it could be expressed in a basis obtained by singular value decomposition (SVD) of some matrix derived from the data, or it can rely on kernel-based representations where the kernel bandwidth is selected in an adaptive fashion. However, we typically observe in these cases that the learning algorithm can be clearly divided into a feature learning step (also known as state discovery or subspace identification, depending on the context) followed by a dynamics (or update function) learning step. On the other hand, a typical learning algorithm for a latent state model jointly learns the state representation and the dynamics.

### 2.1.3 Likelihood Evaluation View

Another formulation to define a dynamical system is to directly define the likelihood of a sequence of observations. The notion of a belief state can arise as a by-product, as we will see in Section 2.3.2. We are particularly interested in likelihoods of the form

$$\Pr(o_{1:t}) = \sigma(A_{o_t} \dots A_{o_2} A_{o_1} q_1), \tag{2.4}$$

where $A_o$ is a linear operator determined by observation $o$ and $\sigma$ is a linear functional.[2] The functional form in the RHS is referred to as a *sequential system* (Carlyle and Paz, 1971) or a *weighted automaton* (Schtzenberger, 1961). For a sequence of observations $x = o_{1:t}$, we will use $A_x$ to denote the product $A_{o_t} \dots A_{o_2} A_{o_1}$. In order for a weighted automaton to define a valid probability distribution, it must satisfy the probability axioms:

- $\sigma(q_1) = 1$ (unity)
- $\sigma(\sum_{o \in \mathcal{O}} A_o q) = \sigma(q)$ (summation)

[2]There are likelihood models where $\sigma$ is not linear (e.g., to enforce non-negative probabilities (Zhao and Jaeger, 2010)). These models are out of scope of this chapter.

- $\sigma(A_x q_1) \geq 0 \quad$ for any sequence $x$ (non-negativity)

In this case it is also referred to as a linear *observable operator model* (OOM) (Jaeger, 2000).[3] The following theorem shows that linear OOMs have sufficient expressive power to represent any dynamical system.

**Theorem 2.1.** *For any probability distribution over sequences of observations, there exists a vector $q_1$ (possibly infinite dimensional), a set of linear operators $\{A_o \mid o \in \mathcal{O}\}$ and a linear functional $\sigma$ such that*

$$\Pr(o_1, \ldots, o_t) = \sigma(A_{o_T} \ldots A_{o_2} A_{o_1} q_1),$$

*Proof.* We will prove the theorem by construction. Let $\mathcal{H}$ be the vector space of all functions $f : \mathcal{O}^* \mapsto \mathbb{R}$. Define $q_1 \in \mathcal{F}$ to be a function s.t. $q_1(x) = \Pr(x)$ for each $x \in \mathcal{O}^*$. Define $A_o$ to be the operator that satisfies

$$(A_o f)(x) = f(ox), \quad \forall f \in \mathcal{H}, x \in \mathcal{O}^*$$

Note that if $f = c_1 f_1 + c_2 f_2$ for $c_1, c_2 \in \mathbb{R}$ then

$$(A_o f)(x) = c_1 f_1(ox) + c_2 f_2(ox) = c_1 (A_o f_1)(x) + c_2 (A_o f_2)(x)$$

and hence $A_o$ is a linear operator. Define

$$\sigma(f) \equiv f(\emptyset),$$

which can also shown to be a linear functional. It follows by definition that

$$\sigma(A_{o_{1:t}} q_1) = \Pr(o_{1:t})$$

□

Note, however, that an OOM can be infinite dimensional. A class of interest is the class of dynamical systems that can be modeled by a $k$-dimensional OOM (i.e., where $q_1$ and $\sigma$ are $k$-dimensional vectors and $A_o$ are $k \times k$ matrices).[4] To specify this class we need first to discuss the notion of a system matrix. For simplicity, our discussion will assume the set of observations $\mathcal{O}$ is finite. However, the resulting models can be extended to continuous observations using Hilbert space embedding of distributions (Song et al., 2010; Boots and Gordon, 2012) or continuous linear algebra (Kandasamy et al., 2016). Given a finite set of observations, we can specify an ordering of all possible observation sequences. With that ordering we can construct an infinite dimensional system matrix $\mathcal{D}$ such that each row and each column corresponds to a sequence of observations. Each element in the matrix is the probability of the concatenation of the corresponding row and observation sequences, as shown in Figure 2.4.[5]

A dynamical system is said to be of rank $k$ if the corresponding system matrix is of rank $k$. The following proposition connects the system rank and the dimensionality of the corresponding OOM representation.

[3] A model that is closely related to OOM is the *stochastic weighted automaton* (Balle et al., 2014), where the LHS of (2.4) is the probability of generating the sequence $o_{1:t}$ *and then terminating*. This is different from OOM where $\Pr(o_{1:t})$ is the probability of observing any sequence that starts with $o_{1:t}$. Assuming that the system will not generate

| | $\emptyset$ | a | b | aa | ab | ... |
|---|---|---|---|---|---|---|
| $\emptyset$ | 1.0 | $\Pr(a)$ | $\Pr(b)$ | $\Pr(aa)$ | $\Pr(ab)$ | ... |
| a | $\Pr(a)$ | $\Pr(aa)$ | $\Pr(ab)$ | $\Pr(aaa)$ | $\Pr(aab)$ | ... |
| b | $\Pr(b)$ | $\Pr(ba)$ | $\Pr(bb)$ | $\Pr(baa)$ | $\Pr(bab)$ | ... |
| aa | $\Pr(aa)$ | $\Pr(aaa)$ | $\Pr(aab)$ | $\Pr(aaaa)$ | $\Pr(aaab)$ | ... |
| ab | $\Pr(ab)$ | $\Pr(aba)$ | $\Pr(abb)$ | $\Pr(abaa)$ | $\Pr(abab)$ | ... |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ |

Figure 2.4: System dynamics matrix for a system with two observations $a$ and $b$.

**Proposition 2.2** (Thon and Jaeger (2015))**.** *For any system with rank $k$, there exists an equivalent $k$-dimensional OOM.*

That $k$-dimensional OOM is not unique. For any $k$-dimensional OOM defined by $(q_1, A, \sigma)$ and invertible $k \times k$ matrix $S$, an equivalent OOM can be defined by $(\tilde{q}_1, \tilde{A}, \tilde{\sigma})$ such that

$$\begin{aligned} \tilde{\sigma}^\top &= \sigma^\top S \\ \tilde{A}_o &= S^{-1} A_o S \\ \tilde{q}_1 &= S^{-1} q_1 \end{aligned} \tag{2.5}$$

Clearly, replacing $q_1$, $A$ and $\sigma$ in (2.4) with their transformed counterparts in (2.5) does not change the likelihood function. This means a $k$-dimensional OOM is only defined up to a similarity transformation $S$. There are a number of ways to resolve this ambiguity. One way is to require the OOM to be *interpretable*, in case of which, we can interpret the coordinates as probabilities of future *characeristic events*.

**Definition 2.3** (Interpretable OOM)**.** *Let $\mathcal{O}^l$ be the set of all observation sequences of length $l$. Let $\{O_1, \ldots, O_k\}$ be a disjoint partitioning of $\mathcal{O}^l$. A $k$-dimensional OOM $(q_1, A, \sigma)$ is said to be* ***interpretable*** *w.r.t $\{O_1, \ldots, O_k\}$ if for any observation sequence $o_{1:t}$,*

$$\Pr(o_{1:t} O_i) = (A_{o_{1:t}} q_1)_i, \tag{2.6}$$

*where $\Pr(o_{1:t} O_i)$ is the probability of observing $o_{1:t}$ followed by any sequence $x \in O_i$. The events $\{O_1, \ldots, O_k\}$ are called* ***characteristic events****.*

Note that an interpretable OOM must have $\sigma = \mathbf{1}$, the all ones vector. Another way to resolve ambiguity is to use a subspace identification method (see Section 2.4.2.1) to find a "good" coordinate basis based on training data.

an infinite sequence of observations, OOMs and stochastic weighted automata are equivalent model classes. However, the conditions on the parameters to define a valid probability distribution become different.

[4] The traditional formulation of OOM (Jaeger, 2000) requires $\sigma$ to be an all-ones vector. We follow (Thon and Jaeger, 2015) in relaxing this condition. This proves useful when discussing subspace identification methods in Section 2.4.3.2.

[5] The transpose of the system dynamics matrix is commonly referred to as *the Hankel matrix* of the system. Some references, especially in the literature of predictive state representations (Singh et al., 2004), define the system dynamics matrix in terms of conditional distributions— that is, element $(i, j)$ is the probability of observing the $j^{th}$ subsequence after observing the $i^{th}$ subsequence. Adopting this definition does not affect our discussion by much.

## 2.2 Recursive filters

Given a dynamical system where $s_t$ is the underlying system state, there are a number of fundamental inference tasks that one is typically interested in:

**Filtering:** The purpose of filtering is to a maintain and update a belief state $q_t \equiv \Pr(s_t \mid o_{1:t-1})$.[6] The belief state stores all information about the history that is needed to make future predictions. It represents both our knowledge and our uncertainty about the true state of the system at time $t$ given previous observations.

**Prediction:** In the prediction task, we aim to predict observation $o_{t+\tau}$ given $o_{1:t-1}$ for some $\tau \geq 0$.

**Smoothing:** In the smoothing task, we compute a belief state at time $t$ given previous as well as future observations— that is, we compute a representation of $\Pr(s_t \mid o_{1:t+\tau})$.

**Sampling:** We might be interested in generating observation sequences based on the dynamical system model.

**Likelihood Evaluation:** We might also be interested in evaluation of the likelihood of a sequence of observations $\Pr(o_{1:t})$.

In this work, we focus primarily on filtering. An accurate filter can provide input to other downstream tasks. For example, given a filter for maintaining the belief state $q_t$ we can use standard regression to learn a model to predict $o_{t+\tau}$ given $q_t$. We can also learn a probabilistic model to compute $\Pr(o_t \mid q_t)$, thus allowing us to perform sampling and likelihood evaluation.

Maintaining $q_t$ is usually accomplished through a *recursive filter* that takes the form

$$\begin{aligned} \mathbb{E}[o_t | o_{1:t-1}] &= g(q_t) \\ q_{t+1} &= f(x_t, o_t), \end{aligned} \tag{2.7}$$

where $f$ is a *filtering function* or *update function* and $g$ is an observation function. It is clear that a recursive filter emerges naturally from the belief state formulation in (2.3). However, we might want to construct a recursive filter based on other formulations of dynamical systems. Therefore, in the following section, we show how to construct a filter from a dynamical system that is not necessarily specified in the belief state view.

## 2.3 Constructing Filters from Dynamical System Models

Without additional assumptions on the representation (for example, requiring the state to be discrete, finite dimensional or probabilistic; or requiring a parametric form for state updates), the three views explained in Section 2.1 have the same expressive power. That means we can obtain a belief state representation (and hence a filter) from other representations of dynamical systems.[7]

[6] We might not have an explicit representation of $s_t$. The previous equation should be understood as: knowing $q_t$ is equivalent to knowing $\Pr(s_t \mid o_{1:t-1})$ for the purpose of making predictions.

[7] There is an implicit assumption that we can evaluate and marginalize probability distributions used to specify the system. This may not always be true. Some probability distributions (e.g. generative adversarial networks (Goodfellow et al., 2014) are easy to sample from but difficult to evaluate.)

In this section, we demonstrate how to obtain a filter from a system state model, which gives rise to an important construction called the Bayes filter. We then demonstrate how to convert an observable operator model to the belief state representation, resulting in a special class of Bayes filter called (transformed) predictive state representations (Singh et al., 2004; Rosencrantz et al., 2004).

### 2.3.1 Bayes Filter

We now demonstrate how to construct a belief state filter from a system state model. For notational simplicity, we assume the system state to be discrete (e.g. HMM) but the same concept applies to continuous states.

Let the belief state $q_t$ represent the probability distribution $\Pr(s_{t-1} \mid o_{1:t-1})$. We can compute the probability of the next observation $o_t$ as

$$\Pr(o_t \mid o_{1:t-1}) = \sum_{s_{t-1}, s_t} \Pr(o \mid s_t)\Pr(s_t \mid s_{t-1})\Pr(s_{t-1} \mid o_{1:t-1}). \tag{2.8}$$

To update $q_{t+1}$ we need to compute $\Pr(s_t \mid o_{1:t})$. This can be done using Bayes rule as follows.

$$\Pr(s_t \mid o_{1:t}) = \frac{\Pr(s_t, o_t \mid o_{1:t-1})}{\Pr(o_t \mid o_{1:t-1})} = \frac{\sum_{s_{t-1}} \Pr(o_t \mid s_t)\Pr(s_t \mid s_{t-1})\Pr(s_{t-1} \mid o_{1:t-1})}{\Pr(o_t \mid o_{1:t-1})} \tag{2.9}$$

The RHS of equations (2.8) and (2.9) depends only on $q_t$ and system parameters. The denominator in (2.9) does not depend on $s_t$ and typically does not need to be explicitly computed. Instead, we normalize the obtained probability distribution to sum to 1.

In the case of an HMM, $q_t$ is represented by a probability vector of latent system states, and the numerator of Equation (2.9) is the forward algorithm update. For a linear dynamical system with Gaussian noise, $q_t$ consists of the mean and covariance of the system state. The corresponding Bayes filter is known as Kalman filter (Kalman, 1960).

A Bayes filter has an interesting update pattern that we can generalize. In general, we start with a random variable $x_t$ such that the probability distribution $\Pr(x_t \mid o_{1:t-1})$ is a sufficient state representation (in the example above, $x_t \equiv s_{t-1}$). We refer to the variable $x_t$ as the *future*.

We can view the state update as follows:

- From the belief state we can infer the joint distribution $\Pr(\bar{o}_t, x_{t+1} \mid o_{1:t-1})$, where $\bar{o}_t$ is the still unknown observation at time $t$. We refer to the quantity $x_{t+1}$ as the *shifted future* and the quantity $(\bar{o}_t, x_{t+1})$ as the *extended future* and we refer to the joint distribution $\Pr(\bar{o}_t, x_{t+1} \mid o_{1:t-1})$ as the *extended belief state*.
- Given the actual observation $o_t$, we compute the conditional distribution

  $$\Pr(x_{t+1} \mid o_{1:t}) \propto \Pr(o_t, x_t \mid o_{1:t-1})$$

  Thus, a Bayes filter update can conceptually be divided into a state extension step followed by a conditioning step as demonstrated by Figure 2.5. This division will be crucial to the machinery we develop in Chapter 3.

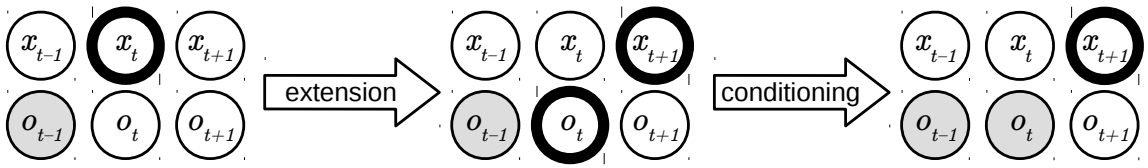



Figure 2.5: Visualization of Bayes filter update. The belief state captures the distribution of variables with thick border. Observations revealed so far are shaded. Left: We start with a belief state $q_t$ that captures the distribution of the future $\Pr(x_t \mid o_{1:t-1})$. Middle: The extended belief state captures the distribution of the extended future $\Pr(o_t, x_{t+1} \mid o_{1:t-1})$. Right: After observing $o_t$, the conditioning step computes the distribution of the shifted future $\Pr(x_{t+1} \mid o_{1:t})$.

### 2.3.2 Predictive State Representation

We first derive a belief state representation of an OOM. Then, after defining predictive state representations, we demonstrate the connection between the two models.

Define $q_{t+1}$ to be a vector such that

$$\sigma(A_{o_{t+1:t+\tau}} q_{t+1}) = \Pr(o_{t+1:t+\tau} \mid o_{1:t}) = \frac{\Pr(o_{1:t+\tau})}{\Pr(o_{1:t})} = \frac{\sigma(A_{o_{1:t+\tau}} q_1)}{\sigma(A_{o_{1:t}} q_1)}.$$

For a linear $\sigma$ this gives

$$q_{t+1} = \frac{A_{o_{1:t}} q_1}{\sigma(A_{o_{1:t}} q_1)} = \frac{A_{o_t} q_t}{\sigma(A_{o_t} q_t)}, \tag{2.10}$$

which results in a recursive update rule. To see that (2.10) constitutes a Bayes filter, define a 3-mode tensor $B$ such that[8]

$$A_o = B(I, o, I).$$

Then, we can rewrite $A_{o_t} q_t$ as $B(I, o_t, q_t) = B(I, I, q_t) o_t$. We can interpret $P_t \equiv B(I, I, q_t)$ as an extension step. For example, if the OOM is interpretable, $P_t$ is a table that stores $\Pr(o_t, x_{t+1} \mid o_{1:t-1})$ for an observation $o_t$ and a shifted future characteristic event $x_{t+1}$. Given $P_t$, the conditioning step becomes

$$q_{t+1} = \frac{P_t o_t}{\sigma(P_t o_t)}.$$

A predictive state representation (PSR) (Singh et al., 2004) is a belief state model where the belief state $q_t$ is a vector of probabilities of future events or tests. PSRs were originally conceived for controlled dynamical systems. However, in the uncontrolled case, a PSR is basically the Bayes filter corresponding to an OOM.

[8]Please refer to the notation chapter for the definition of the multilinear product $B(.,.,.)$.

In the uncontrolled case, each element of the PSR state vector $q_t$ is the probability of a particular sequence of future observations or *test* $\Pr(o_{t:t+\tau} \mid o_{1:t-1})$. The set of tests that make $q_t$ a sufficient state representation are called *core tests*. Inferring these core tests is called the *discovery problem*. Given a sufficient set of core tests, there exists a function $f_{o_{t:t+\tau}}$ for each sequence of observations $o_{t:t+\tau}$ such that

$$\Pr(o_{t:t+\tau} \mid o_{1:t-1}) = f_{o_{t:t+\tau}}(q_t).$$

As in OOM, we will focus on linear PSRs, which have the property that

$$f_{o_{t:t+\tau}}(q_t) = m_{o_{t:t+\tau}}^\top q_t$$

for a vector $m_{o_{t:t+\tau}}$. A PSR is a the canonical example of using a predictive state: define $\psi(o_{t:\infty})$ to be a binary vector, where each element indicates whether $o_{t:\infty}$ matches a particular core test. Then by definition, $q_t = \mathbb{E}[\psi(o_{t:\infty}) \mid o_{1:t-1}]$. We now show that a linear PSR with $k$ core tests can be converted to a $k$-dimensional OOM. By definition, there exists a vector $m_\emptyset$ such that ${m_\emptyset}^\top q = 1$. Now for each observation $o$, we define a matrix $M_o$ such that

$$m_{xo}^\top = m_x^\top M_o, \quad \text{for any sequence } x.$$

A transformed PSR (Rosencrantz et al., 2004) is a PSR whose state representation and parameters are subjected to linear transformations that preserve the model in the same way an OOM can be changed by a similarity transformation.

Note that, given the evaluation vectors $m$ and a minimal state dimension, $M_o$ exists and is unique: the evaluation vectors corresponding to core tests give exactly the equations required to uniquely identify $M_o$. Additional evaluation vectors will only give linear combinations of the equations resulting from core tests and so there is no inconsistency. It is clear that the matrices $M_o$ correspond to observable operators in an OOM. The likelihood of a sequence $o_{1:t}$ is given by

$$\Pr(o_{1:t}) = m_\emptyset^\top M_{o_t} \dots M_{o_1} q_1$$

And similar to (2.10), the PSR state update equation is given by

$$q_{t+1} = \frac{M_{o_t} q_t}{m_\emptyset^\top M_{o_t} q_t}$$

The converse is true: a $k$-dimensional OOM is essentially a $k$-dimensional transformed PSR, but it can also be converted to a PSR with $k$ core tests: from an OOM, one can obtain the evaluation vectors $m_{o_{1:t}} = \sigma^\top A_t \dots A_1$. These vectors are $k$-dimensional and therefore we can find a set of $k$ vectors that span them. The tests corresponding to these vectors are core tests. Therefore, $k$-dimensional OOMs and $k$-dimensional PSRs are equivalent model classes.

## 2.4 Generative Learning of Recursive Filters

We now consider the problem of learning a recursive filter from training data. More specifically, we focus on the *unsupervised learning* scenario where the training data consists only of a set of

observation trajectories. In the following sections we give an overview over classical learning approaches as well as some recent proposals that are related to our work. We focus on frequentist approaches, where filter parameters are assumed to be fixed but unknown. It should be noted, however, that a wide range of Bayesian methods have been used for learning dynamical systems, where the parameters are assumed to be latent variables with prior distributions. These methods include sampling (Fruhwirth-Schnatter, 2001; Rydn, 2008) and variational inference (Foti et al., 2014).

We can classify learning approaches into generative and discriminative methods. In the generative approach we identify the underlying dynamical system as a probability distribution whose parameters are learned by maximizing the data likelihood or through method of moments. The learned dynamical system gives rise to a filter as described in Section 2.3. In the discriminative approach we directly learn the filter by minimizing the error resulting from its use in predicting future observations. We describe generative methods in this section and describe discriminative methods in Section 2.5.

### 2.4.1 Maximum Likelihood

Given a set of $M$ training trajectories, the maximum likelihood method attempts to find the system parameters $\theta \in \Theta$ that maximizes the log-likelihood of the training data

$$l(\theta) = \sum_{i=1}^{M} \log \Pr(o_{i,1:T_i}; \theta), \tag{2.11}$$

where $o_{i,1:T_i}$ denotes the $i^{th}$ training trajectory. Usually, this maximization problem has no analytical solution. Therefore, we resort to local optimization methods. Under certain method-specific conditions, these methods converge to a limit point in the parameter space. Typically, however, this point is not guaranteed to be globally optimal. Two prominent local optimization methods are gradient descent and expectation-maximization.

#### 2.4.1.1 Gradient Descent

Gradient descent is a generic local optimization method. Given a function $f : \Theta \mapsto \mathbb{R}$, gradient descent methods iteratively update the parameters by taking steps along the gradient

$$\theta^{(i+1)} = \theta^{(i)} + \eta^{(i)} \nabla f(\theta)|_{\theta=\theta^{(i)}}, \tag{2.12}$$

where $\eta$ is a step size. There are different variations of gradient descent that vary according to method of computing the gradient and the parameter update equation. A widely used variation is *stochastic gradient descent*, where we replace the gradient with an unbiased estimate thereof that is typically obtained by considering a subset of the training examples in each iteration.

#### 2.4.1.2 Expectation-Maximization

The expectation-maximization (EM) algorithm (Dempster et al., 1977) is another local method to optimize the likelihood of a latent variable. Let $x$ denote the observed variables and let $z$ denote

the latent variables (e.g. system states). Instead of maximizing the log-likelihood function

$$\begin{aligned} l(\theta) &= \log \Pr(o_{1:t}; \theta) \\ &= \log \sum_{z} [\Pr(o_{1:t}, z; \theta)], \end{aligned} \tag{2.13}$$

the EM takes iterations maximizing the following quantity known as the expected complete log-likelihood

$$Q(\theta \mid \theta^{(i)}) = \mathbb{E}_{z|o_{1:t};\theta^{(i)}}[\log \Pr(o_{1:t}, z; \theta)], \tag{2.14}$$

where the expectation is taken w.r.t to the posterior $\Pr(z \mid o_{1:t}; \theta^{(i)})$ induced by the current estimate of the parameters $\theta^{(i)}$. In each iteration, two steps are executed:

- **Expectation:** where we compute the expected sufficient statistics of the posterior $\Pr(z \mid o_{1:t}, \theta^{(i)})$.
- **Maximization:** where we compute $\theta$ that maximizes (2.14).

It has been shown that an increase in (2.14) results in at least the same amount of increase in the log-likelihood. Other variants of EM include stochastic EM (Celeux and Diebolt, 1985), where the E-step is replaced by sampling of the latent variables, and generalized EM, where the M-step only needs to improve (2.14) (typically via one or more gradient ascent steps).

One advantage EM has over gradient descent is that there is no step size to tune. However, gradient descent gives additional flexibility in modifying the objective function, changing constraints and/or using a wide variety of tools from optimization literature out of the box.

### 2.4.2 Method of Moments and Spectral Algorithms

Method of moments is an alternative method for estimating the parameters of a probability distribution from i.i.d samples. For a probability distribution $\Pr(x; \theta_0)$, method of moments assumes the existence of known functions $f(x)$ and $m(\theta)$ that satisfy the *moment condition*:

$$\mathbb{E}_{x\sim\Pr(x;\theta_0)}[f(x)] = m(\theta) \quad \text{iff } \theta = \theta_0.$$

Given samples $x_1, x_2, \ldots, x_N \sim \Pr(x; \theta_0)$, method of moments computes the empirical moment

$$\hat{m} = \frac{1}{N} \sum_{i=1}^{N} g(x_i)$$

and then solves for $\theta$ that satisfies

$$m(\theta) = \hat{m}$$

Under mild assumptions, method of moments is consistent: it converges to the true parameter vector $\theta_0$ in the limit of infinite data. It is usually simpler than maximum likelihood estimation

although less statistically efficient. Therefore, it can be used to initialize an iterative procedure for likelihood maximization.

The i.i.d condition can be relaxed, as long as the empirical moment converges to the true expectation. If we replace i.i.d sampling by a stationary process, we can compute the empirical moment from trajectories generated by the process. If the process is also ergodic, we can compute empirical moments from a single long trajectory. The estimated moments will converge but at a slower rate depending on the mixing time of the process.

One special class of method of moments is *spectral methods*. These methods utilize matrix and/or tensor factorization in estimating the parameters. In the context of learning dynamical systems, we can identify two subclasses of spectral algorithms in the literature, which we refer to as *subspace identification* and *tensor decomposition*. We describe them in the following subsections.

#### 2.4.2.1 Subspace Identification

A subspace identification method first attempts to find a low dimensional state representation by using observed data to compute a subspace that contains the belief states. That subspace is usually obtained by factorizing the covariance of history and future observations using singular value decomposition, canonical correlation analysis or another factorization technique. This is based on the insight that noise on future observations is uncorrelated with the history and therefore this factorization removes spurious dimensions that result from noise.

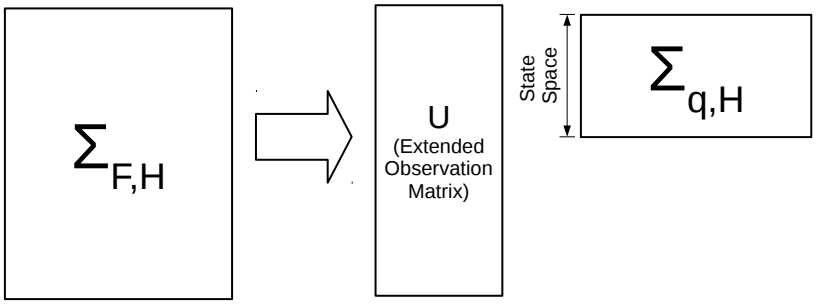



Figure 2.6: Factorizing the future/past covariance matrix results in (1) a representation of the future in a low dimensional *state-space* that neglects directions uncorrelated with the past and (2) an *extended observation matrix* that reconstructs expected future observations from the low dimensional representation.

After obtaining the state representation, the algorithm learns the system dynamics/update equation. This is done by moment matching which typically results (implicitly or explicitly) in solving regression problems. Subspace identification algorithms follow a predictive state paradigm; they seek a belief state representation in terms of projected future observations. The matrix factorization or *spectral* component of the algorithm is actually used for obtaining this projection, which serves as the feature function $\psi$. It is worth noting that a projection can be obtained through other methods such as random Gaussian matrices, although matrix factorization methods are more statistically efficient (Boots, 2012). It is also worth noting that there are regularization options other

than the low-rank assumption assumed by subspace identification methods. For example, Sun et al. (2016) have shown that replacing the low-rank assumption with $L_2$ regularization through the methodology we devlop in Chapter 3 can outperform subspace identification for linear dynamical systems.

#### 2.4.2.2 Tensor Decomposition

A *multi-view model* is a latent variable probabilistic model where there are multiple observed variables that are conditionally independent given the latent variable. Figure 2.7 shows examples of multi-view models.

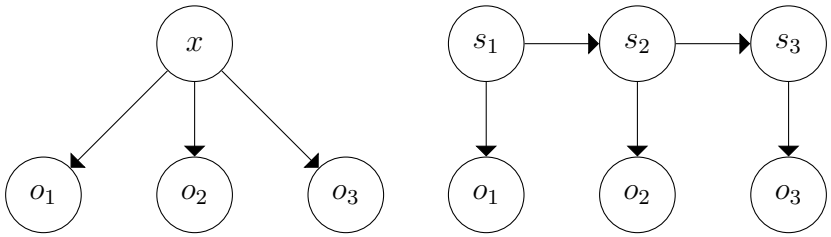



Figure 2.7: (a) A multiview model: observables $o_1$, $o_2$ and $o_3$ are three independent views of the latent variable $x$. (b) HMM as a multiview model: $o_1$, $o_2$ and $o_3$ are three independent views of $s_2$.

We are interested in multi-view models that are identifiable from observed variables— that is, the marginal distribution of the observed variables uniquely determines the parameters of the model (up to a permutation of the latent variable coordinates). Tensor decomposition methods constitute a relatively recent family of spectral methods for learning identifiable multi-view models with discrete latent variables. The main concept is to exploit the structure in low-order moments, typically second and third-order. The pair-wise and triple-wise interaction statsitics between multiple views of the same latent variable contain implicit information about that variable that can be uncovered using decomposition methods. While this idea was used by earlier methods that used singular value decomposition (Anandkumar et al., 2012; Hsu and Kakade, 2013), the tensor decomposition provides a more explicit and unifying view.

Let $o_1, o_2$ and $o_3$ be three views of a latent variable $x$ as shown in Figure 2.7. Assume $x$ can take the values $1, \ldots, m$. Define

$$\begin{aligned} \omega_j &\equiv \Pr(x = j) \\ \mu_{i,j} &\equiv \mathbb{E}[o_i \mid x = j] \quad \forall i \in \{1, 2, 3\}, j \in \{1, \ldots, N\}. \end{aligned} \tag{2.15}$$

It follows from conditional independence and iterative expectation that

$$\mathbb{E}[o_1 \otimes o_2 \otimes o_3] = \sum_{j=1}^{m} \omega_j \mu_{1,j} \otimes \mu_{2,j} \otimes \mu_{3,j} \tag{2.16}$$

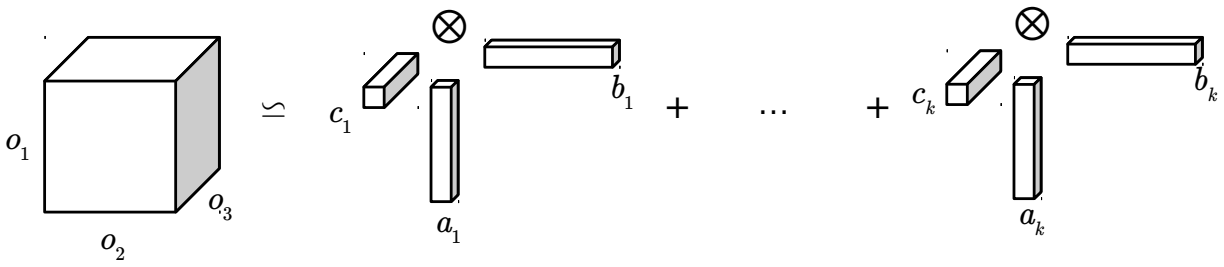



Figure 2.8: Visualization of tensor PARAFAC decomposition.

In other words, the third moment tensor is a sum of $m$ rank 1 tensors. The essence of the tensor decomposition method is to estimate the empirical third moment tensor from data and then decompose it into a sum of rank one tensors (known as CP or PARAFAC decomposition (Harshman, 1970)— see Figure 2.8).

$$T = \sum_{j=1}^{m} a_j \otimes b_j \otimes c_j = \sum_{j=1}^{m} \hat{w}_j \hat{\mu}_{1,j} \otimes \hat{\mu}_{2,j} \otimes \hat{\mu}_{3,j} \tag{2.17}$$

Under certain rank conditions (Kruskal, 1977), the decomposition is unique up to a permutation of the rank-1 tensors and a rescaling of their factors. The scaling ambiguity can be resolved by enforcing $\sum_{j=1}^{m} \hat{w}_j = 1$ as well as additional constraints on the expectations $\hat{\mu}_{i,j}$ depending on the model. The model parameters can then be recovered by matching the results of tensor decomposition to their theoretical values. There are a number of recent tensor decomposition methods with theoretical guarantees to perform the PARAFAC decomposition such as symmetric tensor power iteration (Anandkumar et al., 2014a), alternating rank 1-updates (Anandkumar et al., 2014b) and simultaneous diagonalization (Kuleshov et al., 2015).

### 2.4.3 Example I: Learning Hidden Markov Models

We now provide a concrete comparison of generative learning approaches by describing examples from the literature for learning the parameters of a hidden Markov model.

A hidden Markov model (HMM) (Rabiner and Juang, 1986), is a dynamical system with discrete system states and discrete observations. An HMM is specified by a probability vector $\pi$, a stoachstic transition matrix $T$ and a stochastic observation matrix $O$ such that

$$\begin{aligned} \Pr(s_1 = i) &= \pi_i \\ \Pr(s_{t+1} = i \mid s_t = j) &= T_{ij} \\ \Pr(o_t = i \mid s_t = j) &= O_{ij}, \end{aligned}$$

Knowing these parameters, a recursive filter can be constructed using the forward algorithm.

#### 2.4.3.1 Maximum Likelihood

The standard algorithm for unsupervised learning of an HMM used to be Baum-Welch algorithm (Baum et al., 1970), although gradient descent methods have also been proposed (Levinson

et al., 1982). Baum-Welch algorithm algorithm is an instance of expectation-maximization. The sufficient posterior statics are the number of times observation $i$ was emitted by state $j$, and the number of times state $i$ transitioned into state $j$. The expectations of these counts can be estimated using the forward-backward algorithm. The M-step computes transition and observation probabilities using these estimated counts.

#### 2.4.3.2 Spectral Algorithms

We now consider spectral algorithms for learning an HMM. For simplicity, we assume the HMM to be 1-observable. The spectral algorithms we describe make use of the triple-wise statistics of the first three observations $o_1, o_2, o_3$. Specifically, they utilize the following moments

$$\begin{aligned} P_1 &\equiv \mathbb{E}[o_1] && \text{(Probability vector of 1 observation)} \\ P_{1,2} &\equiv \mathbb{E}[o_1 \otimes o_2] && \text{(Joint probability table (matrix) of two observations)} \\ P_{1,2,3} &\equiv \mathbb{E}[o_1 \otimes o_2 \otimes o_3] && \text{(Joint probability table (tensor) of three observations)} \end{aligned}$$

In practice, we estimate these moments by considering all triplets $o_{t-1}, o_t, o_{t+1}$ under the stationarity assumption.

**Hidden Markov Models as Observable Operator Models**
To compute the probability of a sequence of observations we use the forward algorithm, which can be expressed in matrix form as follows

$$\Pr(o_{1:t}) = \mathbf{1}^\top T \mathrm{diag}(O_{o_t \cdot}) \dots T \mathrm{diag}(O_{o_1 \cdot}) \pi, \tag{2.18}$$

where $O_{o\cdot}$ is the row in $O$ corresponding to observation $o$. Comparing (2.18) to (2.4) reveals that a $k$-state HMM is a $k$-dimensional OOM[9] with

$$\begin{aligned} q_1 &= \pi, \\ A_o &= T \mathrm{diag}(O_{o\cdot}) \\ \sigma^\top &= \mathbf{1}^\top \end{aligned}$$

If we relax the requirement that $\pi$, $T$ and $O$ are stochastic, the functional form of (2.18) is called a *factorized weighted automaton* (Balle et al., 2014).

**Subspace Identification for Hidden Markov Models**
We will consider the spectral HMM learning algorithm proposed in (Hsu et al., 2009). The algorithm assumes that $\pi > 0$ element-wise and that $T$ and $O$ have full column-rank. The rank assumption entails 1-observability. It can be relaxed by replacing $o_{t-1}$ (which can be interpreted as past) and $o_{t+1}$ (which can be interpreted as shifted future) with features over observation windows (Siddiqi et al., 2010). In a nutshell, the algorithm represents the HMM as an OOM , applies

[9]The converse, however, is not true. See (Jaeger, 2000) for an example of an OOM that cannot be modeled by an HMM.

a specific similarity transformation to OOM parameters that makes it easier to perform moment matching to recover the transformed OOM parameters from moment matrices and finally recovers the parameters from empirical moments.

As discussed in Section 2.1.3, a $k$-dimensional OOM can have an invertible matrix $S$ applied to its parameters as shown in (2.5). Let $U$ be a matrix such that $S = U^\top O$ is invertible and let $b_1, B, b_\infty$ denote the parameters of the transformed OOM such that

$$\begin{aligned} b_1 &= (U^\top O)\pi \\ B_o &= (U^\top O)A_o(U^\top O)^{-1} \\ b_\infty &= 1^\top (U^\top O)^{-1} \end{aligned}$$

The choice of the transformation $S = U^\top O$ can be interpreted as follows: we replace the initial latent belief state $\pi$ with a predictive belief state $b_1 = \mathbb{E}[\psi(o_1)]$ where $\psi(o) = U^\top o$.

The subspace identification step chooses $U$ such that $U^\top O$ is invertible. Hsu et al. (2009) show that a natural choice of $U$ is the top $k$ singular vectors of the matrix $P_{2,1}$. Once $U$ is obtained, we can perform moment matching. The switch from a latent to a predictive representation makes the moment matching step tractable. Hsu et al. (2009) show that

$$\begin{aligned} b_1 &= U^\top P_1 \\ b_\infty &= (P_{2,1}^\top U)^+ P_1 \\ B_o &= (U^\top P_{3,o,1})(U^\top P_{2,1})^+, \end{aligned} \tag{2.19}$$

where the matrix $P_{3,o,1}$ is the slice of $P_{3,2,1}$ corresponding to $o_2 = o$. This gives the following algorithm:

- **Moment Estimation:** Compute empirical estimates $\hat{P}_1, \hat{P}_{2,1}, \hat{P}_{3,2,1}$.
- **Subspace Identification:** Compute the state space basis $U = \textsc{Svd}(\hat{P}_{2,1})$.
- **Moment Matching:** Estimate the transformed parameters by plugging the empirical moments into (2.19).

**Tensor Decomposition for Hidden Markov Model**

As shown in Figure 2.7, the observations $o_1, o_2$ and $o_3$ are three independent views of $s_2$. It can also be shown that (Anandkumar et al., 2014a)

$$\begin{aligned} \omega \equiv \mathbb{E}[s_2] &= T\pi \\ \mu_{1,j} \equiv \mathbb{E}[o_1 \mid s_2 = j] &= O\mathrm{diag}(\pi)T^\top \mathrm{diag}(\omega)^{-1} e_j \\ \mu_{2,j} \equiv \mathbb{E}[o_2 \mid s_2 = j] &= Oe_j \\ \mu_{3,j} \equiv \mathbb{E}[o_3 \mid s_2 = j] &= OTe_j \end{aligned} \tag{2.20}$$

Thus, given the PARAFAC decomposition of the third moment tensor, we can directly estimate $O$ and $\omega$. We can then estimate $T$ from $\mu_3$ and hence estimate $\pi = T^{-1}\pi$. The solution is unique if $\pi_1 > \pi_2 > \cdots > \pi_m > 0$ and $T$ and $O$ are both full column-rank.

#### 2.4.3.3 Discussion

Optimal estimation of hidden Markov models from data is NP-hard under cryptographic assumptions (Terwijn, 2002). Maximum likelihood approaches are not guaranteed to converge to a global optimum and therefore are not consistent. Spectral methods circumvent this problem by employing two key theoretical concepts. First, they assume additional conditions on the underlying HMM that ensure a minimum degree of separability between states and rule out the pathological cases that result in the hardness of learning. Second, they replace HMMs with a larger model class that simplifies learning. The subspace identification method actually learns a weighted automaton. The tensor decomposition method learns a factorized automaton and therefore it is one step closer to learning an HMM. Given an infinite amount of samples generated from an HMM, both methods are guaranteed to converge to a model equivalent to that HMM. However, in the finite sample setting, neither method is guaranteed to produce a valid HMM. In fact, neither method is guaranteed to produce a valid probability distribution. This is the source of the so-called *negative probability problem*, where a weighted automaton can predict negative probabilities of observation sequences. The problem is compounded by the fact that verifying whether a set of weighted automaton parameters satisfy the non-negativity constraints is an undecidable problem (Wiewiora, 2008). Another issue is that spectral techniques, while consistent, are not statistically efficient and therefore require more data to produce acceptable models.

One common approach to harness the computational efficiency of spectral methods while avoiding these issues is to compute valid HMM parameters that are close to the factorized automaton produced by the spectral method, and then use these parameters to initialize a local optimization procedure such as EM (Falakmasir et al., 2013; Balle et al., 2014; Zhang et al., 2016). A simple, but ad-hoc, approach is to invert negative entries in the obtained parameters and then normalize the columns to sum to one. Shaban et al. (2015) suggest a more principled approach, by posing the problem of finding the closest HMM to a given factored automaton as an exterior point optimization problem.

### 2.4.4 Example II: Learning Kalman Filters

The Kalman filter (Kalman, 1960) is a method for tracking the state of a linear dynamical system with Gaussian noise. Such a system has the following dynamics

$$\begin{aligned} s_{t+1} &= As_t + \nu_t, \\ o_t &= Cs_t + \epsilon_t, \\ \begin{bmatrix} \nu_t \\ \epsilon_t \end{bmatrix} &\sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} Q & S \\ S^\top & R \end{bmatrix}\right). \end{aligned} \tag{2.21}$$

The Kalman filter is essentially the Bayes filter resulting from expressing (2.21) in the belief state form, although it is sometimes used to refer directly to (2.21). Under the Gaussian noise assumption, the belief state consists of the mean and covariance of the predictive distribution $\Pr(s_t \mid o_{1:t-1})$. If all eigenvalues of $A$ are strictly inside the unit circle, the system is said to be *stable* and it can be shown in this case that the posterior covariance converges to a constant

*steady-state* value (Anderson and Moore, 1979). Thus, a *steady-state Kalman filter* needs to keep track only of the predictive mean $\mathbb{E}[s_t \mid o_{1:t-1}]$.

#### 2.4.4.1 Expectation-Maximization

Ghaharmani and Hinton (1996) proposed an expectation-maximization algorithm for learning linear dynamical systems with Gaussian noise. The expected sufficient statistics needed for the E-step are the first and second moments of the latent states $s_t$, which can be computed using a Kalman smoothing-type algorithm. Given these moments, the optimal parameters for the M-step can be computed analytically.

Unlike an HMM, there is no restriction on the matrices in (2.21) and therefore we can apply any invertible transformation to the latent state and define an equivalent system, as long as we update the parameter matrices accordingly. One method to resolve some of the ambiguity is to fix the state noise covariance $Q$ to identity (Belanger and Kakade, 2015).

#### 2.4.4.2 Subspace Identification

We will focus on the identification of steady-state Kalman filters, where the predictive covariance of the state $\Sigma_{s_t|o_{1:t-1}}$ is constant and hence we only need to keep track of the mean $q_t \equiv \mathbb{E}[s_t \mid o_{1:t-1}]$. [10]

The update equation then becomes

$$q_{t+1} = Aq_t + K(Cq_t - o_t), \tag{2.22}$$

where $K \equiv \Sigma_{s_{t+1},o_t|o_{1:t-1}} \Sigma^{-1}_{o_t|o_{1:t-1}}$ is called the *Kalman gain*.

There are a variety of system identification methods for Kalman filters. We refer the reader to (Overschee and Moor, 1993; van Overschee and de Moor, 1996) for a review. In our discussion, we will adopt some steps from (Boots, 2012) as they make it easier to draw parallels to subspace identification of hidden Markov models.

Let $d_o$ be the dimensionality of the observations. We define $\tau_f$ and $\tau_h$ to be the lengths of future and past observations windows respectively. We also define the following quantities:

$$h_t \equiv \mathrm{vec}(o_{t-\tau_h:t-1}) \in \mathbb{R}^{d_o\tau_h}$$
$$F_t \equiv \mathrm{vec}(o_{t:t+\tau_f-1}) \in \mathbb{R}^{d_o\tau_f}$$

It can be shown (Boots, 2012; van Overschee and de Moor, 1996) that

$$\mathbb{E}[s_t|h_t] = \mathcal{C}_{s_t,h_t}\mathcal{C}^{-1}_{h_t} h_t \tag{2.23}$$

and it follows that

$$\mathbb{E}[F_t|h_t] = \Gamma\mathcal{C}_{s_t,h_t}\mathcal{C}^{-1}_{h_t} h_t, \tag{2.24}$$

[10] It is important to distinguish between the predictive covariance $\Sigma_{s_t|o_{1:t-1}}$ and the marginal covariance $\Sigma_{s_t}$. While both have constant values in the steady state, these constants are not equal.

where $\Gamma$ is the *observability matrix*, which is defined as

$$\Gamma \equiv \begin{cases} C \\ CA \\ \vdots \\ CA^{\tau_f - 1} \end{cases} \tag{2.25}$$

Note that if $\Gamma$ has full column rank, then the system is $\tau_f$-observable. Let $U$ be a $d_o\tau_f \times m$ matrix of orthonormal columns such that $\text{range}(U) = \text{range}(\Gamma)$. It follows that $U^\top\Gamma$ is invertible and hence we can redfine the system in (2.21) as

$$\begin{aligned} \tilde{s}_{t+1} &= \tilde{A}\tilde{s}_t + \tilde{\nu}_t, \\ o_t &= \tilde{C}\tilde{s}_t + \epsilon_t, \\ \begin{bmatrix} \tilde{\nu}_t \\ \epsilon_t \end{bmatrix} &\sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \tilde{Q} & \tilde{S} \\ \tilde{S}^\top & R \end{bmatrix}\right), \end{aligned} \tag{2.26}$$

, where

$$\begin{aligned} \tilde{s}_t &= (U^\top\Gamma)s_t \\ \tilde{C} &= C(U^\top\Gamma)^{-1} \\ \tilde{A} &= (U^\top\Gamma)A(U^\top\Gamma)^{-1} \\ \tilde{Q} &= (U^\top\Gamma)Q(U^\top\Gamma)^\top \\ \tilde{S} &= (U^\top\Gamma)S \end{aligned}$$

Similar to the HMM case, it can be shown that a good choice of $U$ is the top $m$ singular vectors of the future-past covariance $\mathcal{C}_{F_t,h_t}$. Knowing $U$, we can recover the parameters $\tilde{A}$ and $\tilde{C}$.

**Proposition 2.4** (Adapted from (Boots, 2012))**.** *Let $\Gamma$ denote the observability matrix. Let $U$ be a $\tau_f d \times m$ matrix of orthonormal columns such that* $\text{range}(U) = \text{range}(\Gamma)$*. Assume that the matrices $\Gamma$, $A$ and $C$ are full column rank; it follows that*

$$\begin{aligned} \tilde{C} &= U_{(1:m).} \\ \tilde{A} &= (U^\top\Sigma_{F_{t+1}h_t}\Sigma_{h_t})(U^\top\Sigma_{F_th_t}\Sigma_{h_t})^+ \end{aligned}$$

*Proof.*

$$\begin{aligned} U_{(1:m).} &= U_{(1:m).}(U^\top\Gamma)(U^\top\Gamma)^{-1} \\ &= \Gamma_{1:m}(U^\top\Gamma)^{-1} = C(U^\top\Gamma)^{-1} = \tilde{C} \\ U^\top\mathcal{C}_{F_{t+1},h_t}(U^\top\mathcal{C}_{F_t,h_t}\mathcal{C}_{h_t})^+ &= U^\top\Gamma A\Sigma_{s_th_t}(U^\top\Gamma\Sigma_{s_th_t})^+ \\ &= U^\top\Gamma A\mathcal{C}_{s_t,h_t}\mathcal{C}^+_{s_t,h_t}(U^\top\Gamma)^{-1} \\ &= (U^\top\Gamma)A(U^\top\Gamma)^{-1} = \tilde{A} \end{aligned}$$

□

It remains to compute the Kalman gain, which is a bit more subtle. First, we define $P$ to denote the difference between stationary and predictive covariance[11]

$$P_t \equiv \Sigma_{\tilde{s}_t} - \Sigma_{\tilde{s}_t|o_{1:t-1}} \tag{2.27}$$

It can be shown that (van Overschee and de Moor, 1996)

$$K_t = (\Sigma_{\tilde{s}_{t+1},o_t} - \tilde{A}P_t\tilde{C}^\top)(\Sigma_{o_t} - \tilde{C}P_t\tilde{C}^\top)^{-1} \tag{2.28}$$

$$P_{t+1} = \tilde{A}P_t\tilde{A}^\top + (\Sigma_{\tilde{s}_{t+1},o_t} - \tilde{A}P_t\tilde{C}^\top)(\Sigma_{o_t} - \tilde{C}P_t\tilde{C}^\top)^{-1}(\Sigma_{\tilde{s}_{t+1},o_t} - \tilde{A}P_t\tilde{C}^\top)^\top \tag{2.29}$$

To compute the Kalman Gain, first we estimate stationary covariances from data: Estimating $\Sigma_{o_t}$ is obvious. On the other hand we have

$$\Sigma_{\tilde{s}_{t+1}o_t} = U^\top \Sigma_{F_{t+1},o_t},$$

since future observation noise is uncorrelated with $o_t$. Then we set $P_t = P_{t+1} = P_\infty$ and solve (2.29). This equation in an *Algebraic Riccati Equation*. In principle, it could be solved by treating it as a fixed point iteration starting from $P_1 = \mathbf{0}$. However, there are well-known methods to solve it more efficiently (Laub, 1978; van Overschee and de Moor, 1996).

#### 2.4.4.3 Discussion

There are remarkable similarities between learning hidden Markov models and Kalman filters. Both underlying systems are described by linear transition and observation matrices. Both admit an EM algorithm where the corresponding smoothing method is used in the E-step. And finally, both have subspace identification algorithms that follow similar recipes: (1) Use SVD of future-past covariance to find a good state-space basis. (2) After transforming the system to the new basis, recover parameters from moments that involve past, future, observation and shifted future.

On the other hand, both have their complications: An HMM requires transition and observation matrices to be stochastic. Additional post-processing is needed to recover a valid HMM. A Kalman filter does not have such requirement but, on the other hand, it needs to learn additional parameters related to the noise model.

## 2.5 Discriminative Learning of Recursive Filters

Discriminative training offers an alternative approach to learn recursive filters that directly optimizes their predictive performance. Given a recursive filter as defined in (2.7), we define a prediction target $\psi_t = \psi(o_{t:\infty})$ for some feature function $\psi$ and we train the filter by minimizing the loss

[11] van Overschee and de Moor (1996) refers to $P$ as the *forward state covariance matrix*– that is, the covariance of the Kalman filter *belief state* over all possible histories. With this definition, (2.27) is simply an application of the law of iterated variance. This should not be confused with the predictive covariance $\Sigma_{s_t|o_{1:t-1}}$, which denotes the covariance of the *system state* conditioned on a specific observation history. Unfortunately, there are other references that use $P$ to denote the predictive covariance.

function

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} l(\psi_t, \tilde{g}(q_t)) + R(\theta), \tag{2.30}$$

where $q_t$ is the result of applying the filter up to time $t$, $\tilde{g}$ is a horizon prediction function, $\theta$ is a parameter vector that parametrizes the filter as well as the horizon prediction function, $l$ is a suitable loss function (e.g. square loss) and $R$ is a regularization function (e.g. square $L_2$ norm).

Discriminative learning has its advantages if filtering is our main concern: it directly optimizes our task of interest whereas generative learning optimizes a different objective. Discrminative learning also gives additional flexibility in designing the filter and the training objective function, since we do not care about their probabilistic interpretation. Below we describe different methods to optimize (2.30).

### 2.5.1 Gradient Descent

We can minimize (2.30) using gradient descent and other numerical optimization techniques. Computation of the gradient can be performed using backpropagation through time (BPTT) (Werbos, 1990). BPTT first unfolds the recursive computation graph and computes the belief state $q_t$ at each step. Then, it computes the gradient by going backwards through the unfolded graph, applying the chain rule according to the following recursion.

$$\frac{\partial \mathcal{L}}{\partial q_t} = \frac{\partial \mathcal{L}}{\partial \hat{\psi}_t} \frac{\partial \hat{\psi}_t}{\partial q_t} + \begin{cases} \frac{\partial \mathcal{L}}{\partial q_{t+1}} \frac{\partial q_{t+1}}{\partial q_t}, & \text{if } t < T \\ 0, & \text{if } t = T \end{cases}, \tag{2.31}$$

where

$$\hat{\psi}_t = \tilde{g}(q_t)$$

Using the chain rule, it follows that

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial q_t} \frac{\partial q_t}{\partial \theta} + \frac{\partial \mathcal{L}}{\partial \hat{\psi}_t} \frac{\partial \hat{\psi}_t}{\partial \theta}, \tag{2.32}$$

Figure 2.9 visualizes gradient computation via BPTT. Backpropagation through time is the *de facto* standard method for training recurrent neural networks.

### 2.5.2 Reduction to Supervised Learning

A relatively recent approach to learn recursive filters by discriminative training is to construct the filter from classification/regression components (e.g. linear regression, feedforward networks or regression trees). The training procedure is an iterative procedure where in each iteration, the

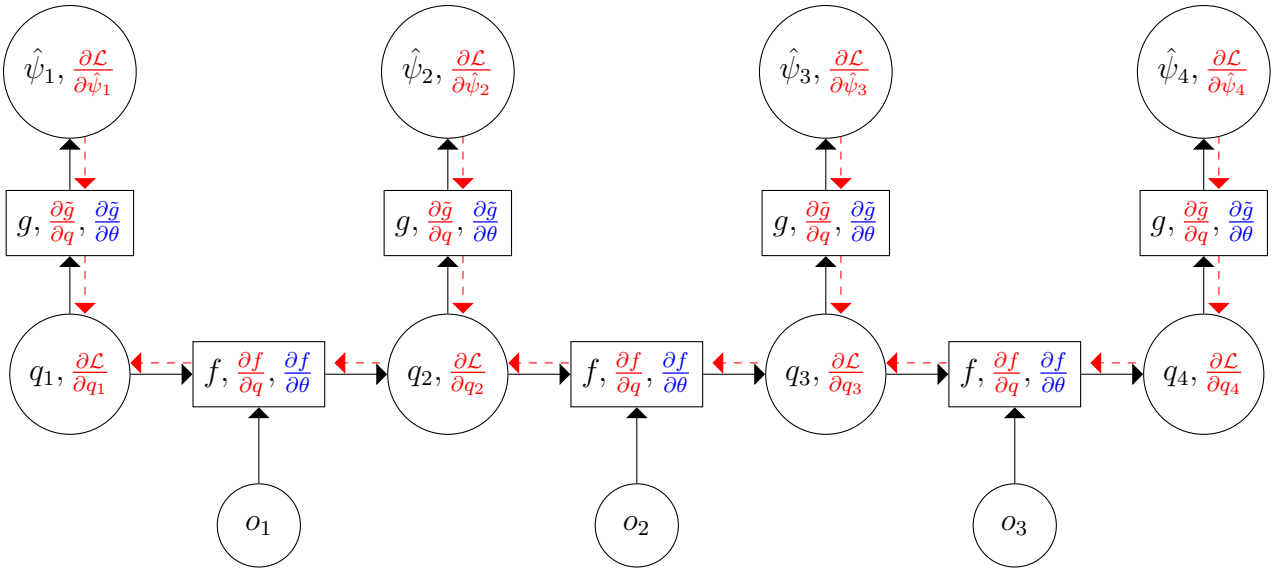



Figure 2.9: Visualization of Backpropagation through time: Circle nodes indicate variables while square nodes indicate functions in an unrolled networks. In the forward pass, the inputs are used to compute belief states $q_t$ and output estimates $\hat{\psi}_t$ where black arrows indicate flow of information. In the backward pass, we start from the gradient w.r.t to the output estimates and red dotted arrows indicate the flow of information. Each belief state node accumulates incoming gradients and sends the total gradient backward. Each function node multiplies the incoming gradient by the Jacobian w.r.t belief state and passes the result backwards. It also multiplies the incoming gradient by the Jacobian w.r.t model parameters. The results from the latter operation are accumulated to compute the total gradient of the loss w.r.t model parameters.

components are updated by solving a set of supervised learning tasks. This approach has been explored by Langford et al. (2009), who proposed the *sufficient posterior representation* model. The model consists of a state initialization component $A$, a state update component $B$ and a prediction component $C$.

The learning algorithm iteratively solves supervised regression tasks (shown in Figure 2.10) such that $C(A(o_1)) \approx \psi_2$ and $C(B(q_t, o_t)) \approx \psi_{t+1}$.

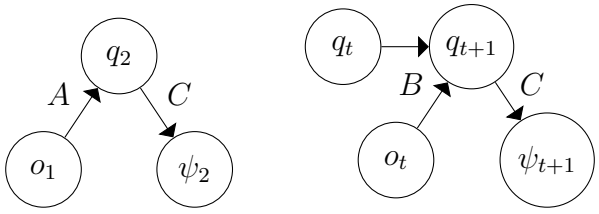



Figure 2.10: Regression tasks for learning a sufficient posterior representation model.

The existence of the prediction component $C$ complicates the training of sufficient posterior representation models, resulting in non-standard supervised learning subproblems and limiting the

flexibility of using arbitrary off-the-shelf supervised learning models and training algorithms. Sun et al. (2016), solve this problem by resorting to predictive states. Their proposed model, named the *predictive state inference machine*, defines the belief state to be the expectation of future features (i.e. $q_t = \mathbb{E}[\psi_t \mid o_{1:t-1}]$). With this restriction, one only needs to learn the state update component $B$. This can be accomplished using data aggregation (DAGGER) (Ross et al., 2011), which is depicted in Algorithm 1.

**Algorithm 1** Training a predictive state inference machine using data aggregation (DAGGER)

**Input:** A set of $M$ trajectories $\{\tau_1, \ldots, \tau_M\}$, number of iterations $N$, hypothesis class $\mathcal{B}$.
**Output:** Initial state estimate $q_1$, a state update function $B$.

1: Compute initial state $q_1 := \frac{1}{M} \sum_{i=1}^{M} \psi_{i,t}$.
2: Choose an initial state update function $B^{(0)} \in \mathcal{B}$.
3: **for** $n = 0$ to $N - 1$ **do**
4: Compute the belief state $q_{i,t}^{(n)}$ for $1 \leq i \leq M$ and $1 \leq t \leq T_i$ using $B^{(n)}$.
5: For each trajectory $\tau_i$ and each time step $t$, construct the training example with $z_{i,t} = (q_{i,t}, o_{i,t})$ as input and $\psi_{i,t+1}$ as output and add it to dataset $D_{n+1}$.
6: Train the state update component $B^{(n+1)}$ on $D^{(n+1)}$ to minimize the loss $l(B(q_t, o_t), \psi_{t+1})$.
7: **end for**
8: Set $B$ by selecting the element in $\{B^{(0)}, \ldots, B^{(N)}\}$ that achieves the lowest validation error.

In each iteration, DAGGER solves a standard supervised learning task, where the goal is to learn an update component $B^{(n)}$ that predicts shifted future statistics $\psi_{t+1}$ given the current observation $o_t$ and the belief state $q_t^{(j)}$ obtained in all previous iterations $1 < j < n$. This problem can be solved using off-the-shelf supervised learning methods, as long as the loss function is a Bregman divergence (e.g. square loss or cross-entropy). By including training data that consists of the estimated state in the input and the true future as an output, the filter effectively learns how to recover from its mistakes.

It is beneficial to contrast PSIMs with subspace identification methods, since both follow a predictive state paradigm and both offer some theoretical guarantees. In addition to the general differences between generative and discriminative learning mentioned in the beginning of Section 2.5, PSIMs offer agnostic generalization guarantees: they do not assume the correctness of the filtering model. Instead, they bound the expected error encountered at test time in terms of the error encountered at training time. That guarantee, however, can depend heavily on the performance of the initial hypothesis $B^{(0)}$. A poor choice of $B^{(0)}$ may require a large number of iterations to compensate for, or can even result in a poor asymptotic model. In addition, the PSIM filtering guarantee assumes that the minimization problem in Algorithm 1 is solved exactly, which may not be applicable to complicated hypothesis spaces where that minimization is non-convex.

Subspace identification methods, on the other hand, are not iterative and do not have initialization issues. They need a fixed number of passes over the data (usually 1) to recover the parameters. These methods are guaranteed to recover the correct parameters in the limit of infinite data in the realizable setting (i.e. when the data matches the model assumption). In general, how-

ever, when there is a model mismatch (e.g. when the assumed system rank is less than the actual value (Kulesza et al., 2014)), all bets are off.

## 2.6 Conclusion

We presented an overview of a range of dynamical system models in the literature and we have shown that we can construct recursive filters from dynamical systems specified in different forms. Based on this overview, we can identify two design choices to be made when constructing a recursive filter: one is related to the state representation and the other is related to the learning algorithm.

**Latent vs. Predictive State:**
We can use a latent belief state $q_t$ that represents a belief over the hidden state of the system $s_t$ and learn an observation function $g$ which maps the belief state to a prediction of future observations. Or, we can use a predictive belief state that is defined to be the expectation of sufficient future statistics conditioned on all previous observations. The latter choice implies a specification of the observation function $g$ and thus can simplify learning.

**Generative vs. Discriminative Filters:**
We can use a generative learning approach where we first derive the update equation $f$ in terms of the parameters of the corresponding dynamical system and then learn these parameters using maximum likelihood or method of moments. For example, in an HMM, the dynamical system parameters are the initial belief state $\pi$, the observation matrix $O$ and the transition matrix $T$, and the update function $f$ is derived from these parameters using the forwarding algorithm. Or, we can use a discriminative learning approach where we directly minimize the prediction error, which usually gives us extra flexibility in specifying the state update function $f$ without worrying much about the underlying generative process.

These two design choices result in four categories of methods to construct recursive filters for dynamical systems. We summarize these categories in Table 2.1. The framework we develop in Chapter 3 lies in the category of predictive state generative filters. At first glance, this choice seems counter-intuitive since this category seems to be more restricted in terms of both observation and state update functions. However, we will demonstrate that this category of models enjoys advantages that allow us to develop tractable and consistent learning algorithms. For example we will show that, through linear regression, we can learn a wide class of models that would require non-linear regression if formulated as discriminative filters. We will also show that we can develop a training procedure that does not have issues with the initialization of states or parameters. In addition, we will show in Chapter 4 that we can still use discriminative training approaches to further enhance the predictive performance of our model.

| | Predictive State | Latent State |
|---|---|---|
| Generative Filter | Observable Operator Models<br>Predictive State Representations<br>Predictive State Models<br><br>Algorithms:<br>Method of Moments: Two-stage regression<br>Maximum Likelihood | Hidden Markov Models<br>Kalman Filters<br><br>Algorithms:<br>Method of Moments: Tensor Factorization<br>Expectation Maximization (EM) |
| Discriminative Filter | Predictive State Inference Machines<br><br>Algorithms:<br>Data Aggregation (DAGGER) | Sigmoid Recurrent Neural Networks (RNNs)<br><br>Algorithms:<br>Backpropagation through time (BPTT) |

Table 2.1: Four categories of methods to construct recursive filters. Our proposed framework is in blue.

# Part II

# Learning Uncontrolled Systems

# Chapter 3

# Predictive State Models: Generative Learning of Recursive Filters Using Two-Stage Regression

In this chapter we propose a framework for generative learning of recursive filters in uncontrolled dynamical systems. The framework consists of a model class and a learning algorithm. The model class is *predictive state models*, a Bayes filter that utilizes predictive states. The learning algorithm is *two stage-regression* (2SR), a method-of-moments-based approach to learn predictive state models by reduction to supervised learning. Unlike the supervised learning approaches described in Chapter 2, which were iterative in nature, our learning approach consists of solving a fixed small set of supervised learning problems while providing the theoretical guarantees expected from method of moments.

The chapter is organized as follows: In Section 3.1 we define predictive state models. In Section 3.2 we describe the two-stage regression algorithm. In Section 3.3, we describe how existing subspace identification methods described in Chapter 2 can be thought of as special instances of the proposed framework. In Section 3.4 we provide a theoretical analysis of two-stage regression. Finally, in Section 3.5, we experimentally demonstrate that efficacy of using the proposed framework to create novel recursive filters.

## 3.1 Model Class: Predictive State Models

Our formulation relies on the notion of the predictive state, where the state is represented as a prediction of sufficient future observation statistics. We denote *future features* by $\psi_t = \psi(o_{t:\infty})$. The future features are sufficient if $\mathbb{E}[\psi_t|o_{1:t-1}]$ is a sufficient state representation (See Section 2.1.2.1). Two classical choices of $\psi$ are an indicator vector representing $o_t$ for 1-observable HMM (Hsu et al., 2009) and a stack of the next $\tau_f$ observations for $\tau_f-$observable steady-state Linear Gaussian systems (van Overschee and de Moor, 1996). Another quantity we need is the *extended future features* $\xi_t$, which is defined as the sufficient statistics of the distribution $\Pr(o_t, \psi_{t+1} \mid o_{1:t-1})$. The importance of $\xi_t$ is that, given an estimate of $\mathbb{E}[\xi_t \mid o_{1:t-1}]$, we can condition on $o_t$ to obtain

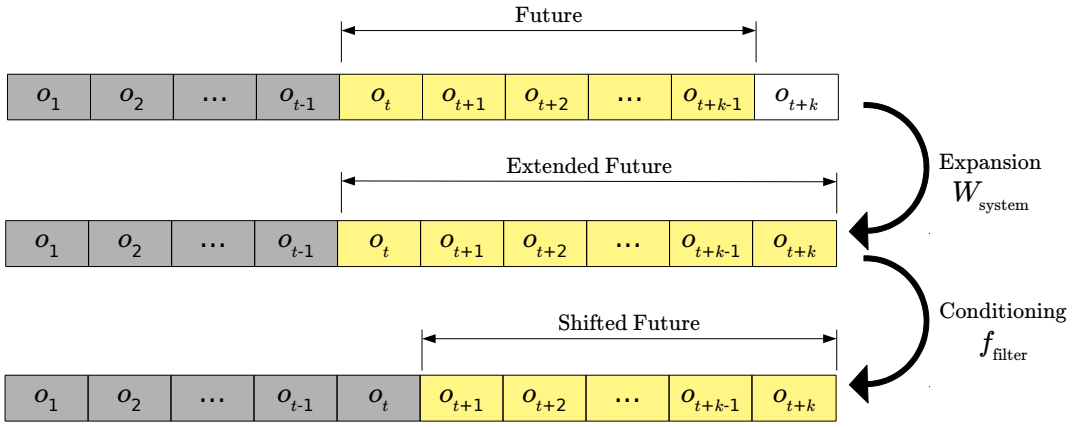



Figure 3.1: Bayes filter update for predictive state models.

$\mathbb{E}[\psi_{t+1} \mid o_{1:t}]$. We are now ready to define our class of models.

**Definition 3.1.** *A dynamical system is said to conform to a* ***predictive state model (PSM)*** *if it satisfies the following properties for some future features $\psi$ and extended future features $\xi$:*

- *For each time $t$, there exists a* predictive state $q_t \equiv \mathbb{E}[\psi_t \mid o_{1:t-1}]$ *which constitutes a sufficient state representation.*
- *For each time $t$, there exists an* extended state $p_t \equiv \mathbb{E}[\psi_t \mid o_{1:t-1}]$.
- *There exists a filtering function $f_{\text{filter}}$ such that, for each time $t$, $q_{t+1} = f_{\text{filter}}(p_t, o_t)$. $f_{\text{filter}}$ is typically non-linear but known in advance.*
- *There exists a linear map $W_{\text{system}}$ such that, for each time $t$,*

$$p_t = W_{\text{system}} q_t \tag{3.1}$$

The predictive state model is a belief state model that entails the following recursive filtering equations:

$$\begin{aligned} \mathbb{E}[\psi_t | o_{1:t-1}] &= q_t \\ q_{t+1} &= f_{\text{filter}}(W_{\text{system}} q_t, o_t) \end{aligned} \tag{3.2}$$

That filter is a Bayes filter that follows the two-step update described in Section 2.3.1 (see Figure 3.1):

- **State Expansion:** $p_t = W_{\text{system}} q_t$
- **Conditioning:** $q_{t+1} = f_{\text{filter}}(p_t, o_t)$

In other words, a predictive state model is a Bayes filter with a predictive state representation and a linear state expansion. Note that we only need to learn the expansion operator $W_{\text{system}}$ and the initial state $q_1$. The linearity of $W_{\text{system}}$ and the prespecified relation between $q_t$ and future observations are what we exploit to develop a tractable learning algorithm, which we describe in the next section.

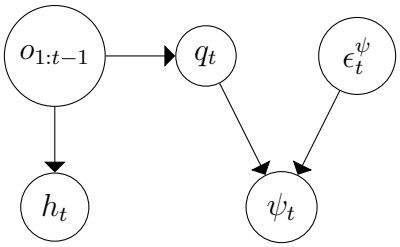



Figure 3.2: Graphical model depicting the (deterministic) dependencies between previous observations $o_{1:t-1}$, belief state $q_t$, noise $\epsilon_t^{\psi}$ and observed features $\psi_t$. Note that previous observations, and hence history features $h_t$, are correlated with the belief state but not with the noise.

## 3.2 Learning Algorithm: Two-Stage Regression

Our goal is to estimate the operator $W_{\text{system}}$. Note that, by definition, the observed statistics $\psi_t$ and $\xi_t$ are unbiased estimates of $q_t$ and $p_t$ that are contaminated with sampling noise $\epsilon_t^{\psi}$ and $\epsilon_t^{\xi}$. Since $W_{\text{system}}$ satisfies $p_t = W_{\text{system}} q_t$ and since $q_t$ and $p_t$ are not observed, a naïve choice to estimate $W_{\text{system}}$ is to use linear regression, replacing $q_t$ and $p_t$ with their unbiased estimates $\psi_t$ and $\xi_t$. Unfortunately, due to temporal overlap between $q_t$ and $p_t$, the noise terms on $\psi_t$ and $\xi_t$ are correlated. So, naïve linear regression will give a biased estimate of $W$.

To counteract this bias, we employ instrumental regression (Pearl, 2000; Stock and Watson, 2011). Instrumental regression uses *instrumental variables* that are correlated with the input $q_t$ but not with the noise ($\epsilon_t^{\psi}$) . This property provides a criterion to denoise the inputs and outputs of the original regression problem: we remove that part of the input/output that is not correlated with the instrumental variables. In our case, since past observations $o_{1:t-1}$ do not temporally overlap with future or extended future features, they are not correlated with the noise , as can be seen in Figure 3.2. Therefore, we can use *history features* $h_t = h(o_{1:t-1})$ as instrumental variables. We exploit history features together with the linearity of $W_{\text{system}}$ to form an instrument-based moment condition that connects the expectation of future and extended future statistics conditioned on history features.

In more detail, by taking the expectation of (3.1) given $h_t$, we obtain the following moment condition: for all $t$,

$$\begin{aligned} \mathbb{E}[p_t \mid h_t] &= \mathbb{E}[W_{\text{system}} q_t \mid h_t] \\ \mathbb{E}[\mathbb{E}[\xi_t \mid o_{1:t-1}] \mid h_t] &= W_{\text{system}} \mathbb{E}[\mathbb{E}[\psi_t \mid o_{1:t-1}] \mid h_t] \\ \mathbb{E}[\xi_t \mid h_t] &= W_{\text{system}} \mathbb{E}[\psi_t \mid h_t] \end{aligned} \tag{3.3}$$

Define $\bar{\psi}_t \equiv \mathbb{E}[\psi_t \mid h_t]$ and $\bar{\xi}_t \equiv \mathbb{E}[\xi_t \mid h_t]$. Assume that history features are rich enough to satisfy $\text{rank}(\mathcal{C}_{\bar{\psi}}) = \text{rank}(\mathcal{C}_{q_t})$. Then, we maintain the rank of the moment condition when moving from (3.1) to (3.3), and we can recover $W_{\text{system}}$ by least squares regression if we can compute $\mathbb{E}[\psi_t \mid h_t]$ and $\mathbb{E}[\xi_t \mid h_t]$ for sufficiently many examples $t$.

Fortunately, conditional expectations such as $\mathbb{E}[\psi_t \mid h_t]$ are exactly what supervised learning algorithms are designed to compute. More specifically, let $\hat{\psi}_t$ be a function of $h_t$. The expected

loss

$$\mathbb{E}_{\psi_t,h_t} l(\psi_t, \hat{\psi}_t) \tag{3.4}$$

is minimized if $l$ is a Bregman divergence[1] function (e.g square loss) and $\hat{\psi}_t = \mathbb{E}[\psi_t \mid h_t]$ (Banerjee et al., 2005). So, we arrive at our learning framework: we first use supervised learning to estimate $\mathbb{E}[\psi_t \mid h_t]$ and $\mathbb{E}[\xi_t \mid h_t]$, effectively *denoising* the training examples[2], and then use these estimates to compute $W_{\text{system}}$ using linear regression to solve (3.3).

In summary, learning and inference of a dynamical system through instrumental regression can be described as follows:

- **Model Specification:** Pick features of history $h_t = h(o_{1:t-1})$, future $\psi_t = \psi(o_{t:t+k-1})$ and extended future $\xi_t = \xi(o_{t:t+k})$. $\psi_t$ must be a sufficient statistic for $\mathbb{P}(o_{t:t+k-1} \mid o_{1:t-1})$. $\xi_t$ must satisfy

  $$\mathbb{E}[\psi_{t+1} \mid o_{1:t}] = f_{\text{filter}}(\mathbb{E}[\xi_t \mid o_{1:t-1}], o_t),$$

  for a known function $f_{\text{filter}}$. The choice of $\psi$ and $\xi$ may involve estimating quantities from data.
- **S1A (Stage 1A) Regression:** Learn a (possibly non-linear) regression model to estimate $\bar{q}_t = \bar{\psi}_t = \mathbb{E}[\psi_t \mid h_t]$. The training data for this model are $(h_t, \psi_t)$ across time steps [3].
- **S1B Regression:** Learn a (possibly non-linear) regression model to estimate $\bar{p}_t = \bar{\xi}_t = \mathbb{E}[\xi_t \mid h_t]$. The training data for this model are $(h_t, \xi_t)$ across time steps $t$.
- **S2 Regression:** Use the feature expectations estimated in S1A and S1B as inputs to a linear regression procedure to estimate a linear operator $W_{\text{system}}$ such that $\bar{p}_t \approx W_{\text{system}} \bar{q}_t$. The training data for this model are estimates of $(\bar{q}_t, \bar{p}_t)$ obtained from S1A and S1B across time steps $t$.
- **Initial State Estimation:** Estimate an initial state $q_1 = \mathbb{E}[\psi_1]$ by averaging $\psi_1$ across several example realizations of our time series.[4]
- **Inference:** Starting from the initial state $q_1$, we can maintain the predictive state $q_t = \mathbb{E}[\psi_t \mid o_{1:t-1}]$ using the filtering equation (3.2). In other words, we use the predictive state $q_t$ to compute $p_t = \mathbb{E}[\xi_t \mid o_{1:t-1}] = W_{\text{system}} q_t$. Then, given the observation $o_t$, we can compute

[1] A Bregman divergence function is defined as $D(x, y) = D^f(x) - D^f(y) - \langle \nabla f(y), x - y \rangle$ for a function $f$. Two common loss functions that are Bregman divergence functions are sqaure loss (corresponding to $f(x) = \|x\|^2$) and KL-divergence (corresponding $f(x) = \sum_i x_i \log x_i$).

[2] Equation (3.3) suggests that denoising $\xi_t$ is a redundant step, since there are no noise terms. Indeed, denoising $\psi_t$ is sufficient to obtain a consistent estimate of $W$. It can also be shown that if linear regression through ordinary least squares is used to estimate conditional expectations over $h_t$ then denoising $\xi_t$ has no effect. However, our preliminary experiments on small sample sizes with non-linear denoising functions applied to both $\psi_t$ and $\xi_t$ results in a better predictive performance and our theoretical analysis shows the consistency of this procedure. Denoising extended future will also be useful when dealing with controlled systems in Chapter 6.

[3] Our analysis in Section 3.4 assumes that the training time steps $t$ are sufficiently spaced for the underlying process to mix, but in practice, the error will only get smaller if we consider all time steps $t$.

[4] Assuming ergodicity, we can set the initial state to be the empirical average vector of future features in a single long sequence, $\frac{1}{T} \sum_{t=1}^{T} \psi_t$.

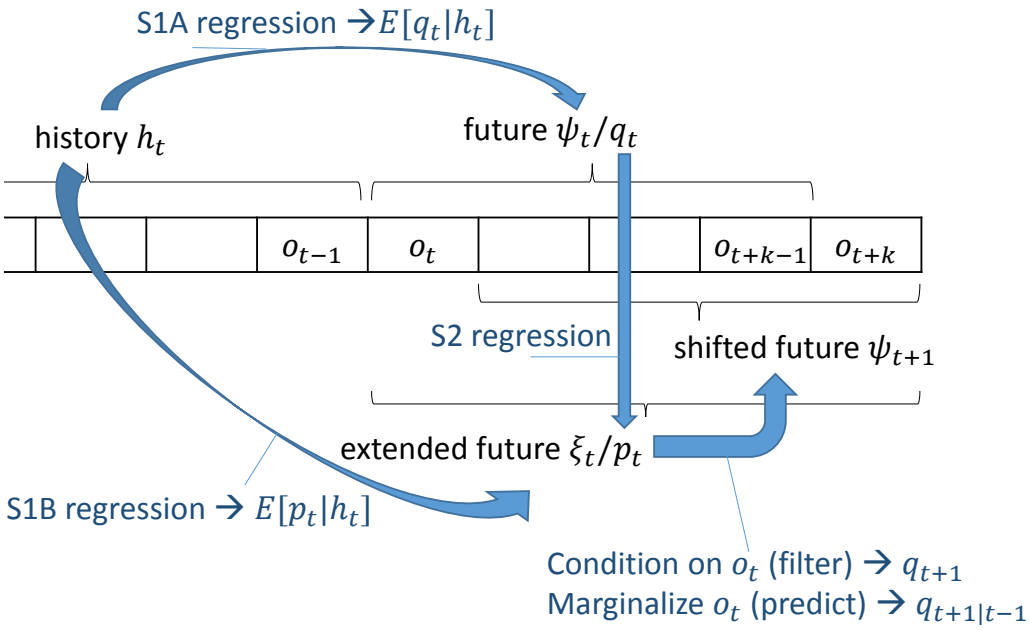



Figure 3.3: Learning and applying a dynamical system using instrumental regression. S1 regression is trained to provide data to train S2 regression. At test time, starting from an initial belief state $q_0$, we alternate between S2 regression and filtering/prediction

$q_{t+1} = f_{\text{filter}}(p_t, o_t)$. Or, in the absence of $o_t$, we can predict the next state $\hat{q}_{t+1} = f_{\text{predict}}(p_t)$, where $f_{\text{predict}}$ is the function corresponding to marginalization over $o_t$.

The process of learning and inference is depicted in Figure 3.3. Modeling assumptions are reflected in the choice of the statistics $\psi$, $\xi$ and $h$ as well as the regression models in stages S1A and S1B. We show in Section 3.3 that we can recover existing subspace identification algorithms for dynamical system learning using linear S1 regression. In addition to providing a unifying view of some successful learning algorithms, the new framework also paves the way for extending these algorithms in a theoretically justified manner, as we demonstrate in Section 3.5.

## 3.3 Subspace Identification Revisited

In this section we revisit subspace identification algorithms for hidden Markov models and Kalman filters that we described in Chapter 2. We reformulate these algorithms as instances of our proposed framework where we use linear regression for stage 1. For each algorithm, we describe the choice of features, the regression method and the filtering function.

### 3.3.1 HMM

In this section we show that we can use instrumental regression framework to reproduce the spectral learning algorithm for learning HMM (Hsu et al., 2009). We consider 1-observable models but the argument applies to $k$-observable models. In this case we use $\psi_t = e_{o_t}$ and $\xi_t = e_{o_{t:t+1}} = e_{o_t} \otimes_k e_{o_{t+1}}$, where $\otimes_k$ denotes the Kronecker product. Let $P_{i,j} \equiv \mathbb{E}[e_{o_i} \otimes e_{o_j}]$ be the joint probability table of observations $i$ and $j$ and let $\hat{P}_{i,j}$ be its estimate from the data. We start with the (very restrictive) case where $P_{1,2}$ is invertible. Given samples of $h_2 = e_{o_1}$, $\psi_2 = e_{o_2}$ and $\xi_2 = e_{o_{2:3}}$, in S1 regression we apply linear regression to learn two matrices $\hat{W}_{2,1}$ and $\hat{W}_{2:3,1}$ such that:

$$\hat{\mathbb{E}}[\psi_2|h_2] = \hat{\mathcal{C}}_{o_2,o_1}\hat{\mathcal{C}}_{o_1}^{-1}h_2 = \hat{P}_{2,1}\hat{P}_{1,1}^{-1}h_t \equiv \hat{W}_{2,1}h_2 \tag{3.5}$$

$$\hat{\mathbb{E}}[\xi_2|h_2] = \hat{\mathcal{C}}_{o_{2:3},o_1}\hat{\mathcal{C}}_{o_1}^{-1}h_2 = \hat{P}_{2:3,1}\hat{P}_{1,1}^{-1}h_2 \equiv \hat{W}_{2:3,1}h_2, \tag{3.6}$$

where $P_{2:3,1} \equiv \mathbb{E}[e_{o_{2:3}} \otimes e_{o_1}]$

In S2 regression, we learn the matrix $\hat{W}$ that gives the least squares solution to the system of equations

$$\hat{\mathbb{E}}[\xi_2|h_2] \equiv \hat{W}_{2:3,1}e_{o_1} = \hat{W}(\hat{W}_{2,1}e_{o_1}) \equiv \hat{W}\hat{\mathbb{E}}[\psi_2|h_2] \quad \text{, for given samples of } h_2$$

which gives

$$\begin{aligned}\hat{W} &= \hat{W}_{2:3,1}\hat{\mathbb{E}}[e_{o_1}e_{o_1}^\top]\hat{W}_{2,1}^\top \left(\hat{W}_{2,1}\hat{\mathbb{E}}[e_{o_1}e_{o_1}^\top]\hat{W}_{2,1}^\top\right)^{-1} \\ &= \left(\hat{P}_{2:3,1}\hat{P}_{1,1}^{-1}\hat{P}_{2,1}^\top\right)\left(\hat{P}_{2,1}\hat{P}_{1,1}^{-1}\hat{P}_{2,1}^\top\right)^{-1} \\ &= \hat{P}_{2:3,1}\left(\hat{P}_{2,1}\right)^{-1}\end{aligned} \tag{3.7}$$

Having learned the matrix $\hat{W}$, we can estimate

$$\hat{p}_t \equiv \hat{W}q_t$$

starting from a state $q_t$. Since $p_t$ specifies a joint distribution over $e_{o_{t+1}}$ and $e_{o_t}$ we can easily condition on (or marginalize $o_t$) to obtain $q_{t+1}$. We will show that this is equivalent to learning and applying observable operators as in (Hsu et al., 2009):

For a given value $x$ of $o_2$, define

$$B_x = u_x^\top \hat{W} = u_x^\top \hat{P}_{2:3,1}\left(\hat{P}_{2,1}^\top\right)^{-1}, \tag{3.8}$$

where $u_x$ is an $|\mathcal{O}| \times |\mathcal{O}|^2$ matrix which selects a block of rows in $\hat{P}_{2:3,1}$ corresponding to $o_2 = x$. Specifically, $u_x = \delta_x \otimes_k I_{|\mathcal{O}|}$. [5].

[5]Following the notation used in (Hsu et al., 2009), $u_x^\top \hat{P}_{2:3,1} \equiv \hat{P}_{3,x,1}$

$$
\begin{aligned}
q_{t+1} &= \hat{\mathbb{E}}[e_{o_{t+1}}|o_{1:t}] \propto u_{o_t}^\top \hat{\mathbb{E}}[e_{o_{t:t+1}}|o_{1:t-1}] \\
&= u_{o_t}^\top \hat{\mathbb{E}}[\xi_t|o_{1:t-1}] = u_{o_t}^\top \hat{W} \mathbb{E}[\psi_t|o_{1:t-1}] = B_{o_t} q_t
\end{aligned}
$$

with a normalization constant given by

$$
\frac{1}{1^\top B_{o_t} q_t} \tag{3.9}
$$

Now we move to a more realistic setting, where we have $\mathrm{rank}(P_{2,1}) = m < |\mathcal{O}|$. Therefore we project the predictive state using a matrix $U$ that preserves the dynamics, by requiring that $U^\top O$ (i.e. $U$ is an independent set of columns spanning the range of the HMM observation matrix $O$).

It can be shown (Hsu et al., 2009) that $\mathcal{R}(O) = \mathcal{R}(P_{2,1}) = \mathcal{R}(P_{2,1}P_{1,1}^{-1})$. Therefore, we can use the leading $m$ left singular vectors of $\hat{W}_{2,1}$ , which corresponds to replacing the linear regression in S1A with a reduced rank regression. However, for the sake of our discussion we will use the singular vectors of $P_{2,1}$. In more detail, let $[U, S, V]$ be the rank-$m$ SVD decomposition of $P_{2,1}$. We use $\psi_t = U^\top e_{o_t}$ and $\xi_t = e_{o_t} \otimes_k U^\top e_{o_{t+1}}$. S1 weights are then given by $\hat{W}_{2,1}^{rr} = U^\top \hat{W}_{2,1}$ and $\hat{W}_{2:3,1}^{rr} = (I_{|\mathcal{O}|} \otimes_k U^\top)\hat{W}_{2:3,1}$ and S2 weights are given by

$$
\begin{aligned}
\hat{W}^{rr} &= (I_{|\mathcal{O}|} \otimes_k U^\top)\hat{W}_{2:3,1}\hat{\mathbb{E}}[e_{o_1}e_{o_1}^\top]\hat{W}_{2,1}^\top U \left(U^\top \hat{W}_{2,1}\hat{\mathbb{E}}[e_{o_1}e_{o_1}^\top]\hat{W}_{2,1}^\top U\right)^{-1} \\
&= (I_{|\mathcal{O}|} \otimes_k U^\top)\hat{P}_{2:3,1}\hat{P}_{1,1}^{-1} V S \left(S V^\top \hat{P}_{1,1}^{-1} V S\right)^{-1} \\
&= (I_{|\mathcal{O}|} \otimes_k U^\top)\hat{P}_{2:3,1}\hat{P}_{1,1}^{-1} V \left(V^\top \hat{P}_{1,1}^{-1} V\right)^{-1} S^{-1}
\end{aligned} \tag{3.10}
$$

In the limit of infinite data, $V$ spans $\mathrm{range}(O) = \mathrm{rowspace}(P_{2:3,1})$ and hence $P_{2:3,1} = P_{2:3,1}VV^\top$. Substituting in (3.10) gives

$$
W^{rr} = (I_{|\mathcal{O}|} \otimes_k U^\top)P_{2:3,1} V S^{-1} = (I_{|\mathcal{O}|} \otimes_k U^\top)P_{2:3,1}\left(U^\top P_{2,1}\right)^+
$$

Similar to the full-rank case we define, for each observation $x$ an $m \times |\mathcal{O}|^2$ selector matrix $u_x = \delta_x \otimes_k I_m$ and an observation operator

$$
B_x = u_x^\top \hat{W}^{rr} \to U^\top P_{3,x,1}\left(U^\top P_{2,1}\right)^+ \tag{3.11}
$$

This is exactly the observation operator obtained in (Hsu et al., 2009). However, instead of using 3.10, they use 3.11 with $P_{3,x,1}$ and $P_{2,1}$ replaced by their empirical estimates.

Note that for a state $b_t = \mathbb{E}[\psi_t|o_{1:t-1}]$, $B_x b_t = P(o_t|o_{1:t-1})\mathbb{E}[\psi_{t+1}|o_{1:t}] = P(o_t|o_{1:t-1})b_{t+1}$. To get $b_{t+1}$, the normalization constant becomes $\frac{1}{P(o_t|o_{1:t-1})} = \frac{1}{b_\infty^\top B_x b_t}$, where $b_\infty^\top b = 1$ for any valid predictive state $b$. To estimate $b_\infty$ we solve the aforementioned condition for states estimated from all possible values of history features $h_t$. This gives,

$$
b_\infty^\top \hat{W}_{2,1}^{rr} I_{|\mathcal{O}|} = b_\infty^\top U^\top \hat{P}_{2,1}\hat{P}_{1,1}^{-1} I_{|\mathcal{O}|} = 1_{|\mathcal{O}|}^\top,
$$

where the columns of $I_{|\mathcal{O}|}$ represent all possible values of $h_t$. This in turn gives

$$\begin{aligned} b_\infty^\top &= 1_{|\mathcal{O}|}^\top \hat{P}_{1,1}(U^\top \hat{P}_{2,1})^+ \\ &= \hat{P}_1^\top (U^\top \hat{P}_{2,1})^+, \end{aligned}$$

the same estimator proposed in (Hsu et al., 2009).

### 3.3.2 Steady-State Kalman Filter

Recall that a Kalman filter is given by

$$\begin{aligned} s_t &= As_{t-1} + \nu_t \\ o_t &= Cs_t + \epsilon_t \\ \nu_t &\sim \mathcal{N}(0, \Sigma_s) \\ \epsilon_t &\sim \mathcal{N}(0, \Sigma_o) \end{aligned}$$

We consider the case of a *stationary* filter where $\mathcal{C}_{s_t}$ is independent of $t$. We choose our statistics

$$\begin{aligned} h_t &= o_{t-\tau_h:t-1} \\ \psi_t &= o_{t:t+\tau_f-1} \\ \xi_t &= o_{t:t+\tau_f}, \end{aligned}$$

Where a window of observations is represented by stacking individual observations into a single vector. It can be shown (Boots, 2012; van Overschee and de Moor, 1996) that

$$\mathbb{E}[s_t|h_t] = \mathcal{C}_{s,h}\mathcal{C}_h^{-1} h_t$$

and it follows that

$$\begin{aligned} \mathbb{E}[\psi_t|h_t] &= \Gamma \mathcal{C}_{s,h}\mathcal{C}_h^{-1} h_t = W_1 h_t \\ \mathbb{E}[\xi_t|h_t] &= \Gamma_+ \mathcal{C}_{s,h}\mathcal{C}_h^{-1} h_t = W_2 h_t \end{aligned}$$

where $\Gamma$ is the extended observation operator

$$\Gamma \equiv \begin{pmatrix} C \\ CA \\ \vdots \\ CA^{\tau_f} \end{pmatrix}, \Gamma_+ \equiv \begin{pmatrix} C \\ CA \\ \vdots \\ CA^{\tau_f+1} \end{pmatrix}$$

It follows that $\tau_f$ and $\tau_h$ must be large enough to have $\text{rank}(W) = n$. Let $U \in \mathbb{R}^{m\tau_f \times n}$ be the matrix of left singular values of $W_1$ corresponding to non-zero singular values. Then $U^\top \Gamma$ is invertible and we can write

$$\begin{aligned}
\mathbb{E}[\psi_t|h_t] &= UU^\top \Gamma \mathcal{C}_{s,h}\mathcal{C}_h^{-1} h_t = W_1 h_t \\
\mathbb{E}[\xi_t|h_t] &= \Gamma_+ \mathcal{C}_{s,h}\mathcal{C}_h^{-1} h_t = W_2 h_t \\
\mathbb{E}[\xi_t|h_t] &= \Gamma_+ (U^\top \Gamma)^{-1} U^\top \left(UU^\top \Gamma \mathcal{C}_{s,h}\mathcal{C}_h^{-1} h_t\right) \\
&= W\mathbb{E}[\psi_t|h_t]
\end{aligned}$$

which matches the instrumental regression framework. For conditioning, one needs to compute the stead-state covariance $\Sigma_\xi$, which can be done by solving a Riccati equation as described in Chapter 2. $\mathbb{E}[\xi_t|o_{1:t-1}]$ and $\Sigma_\xi$ then specify a joint Gaussian distribution over the next $\tau_f + 1$ observations where marginalization and conditioning can be easily performed.

## 3.4 Theoretical Analysis

In this section we present error bounds for two-stage regression. These bounds hold regardless of the particular S1 regression method used. Assuming that the S1 predictions converge to the true conditional expectations, the bounds imply that our overall method is consistent.

Let $(x_t, y_t, z_t) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$ be i.i.d. triplets of input, output, and instrumental variables. Let $\bar{x}_t$ and $\bar{y}_t$ denote $\mathbb{E}[x_t \mid z_t]$ and $\mathbb{E}[y_t \mid z_t]$. And, let $\hat{x}_t$ and $\hat{y}_t$ denote $\hat{\mathbb{E}}[x_t \mid z_t]$ and $\hat{\mathbb{E}}[y_t \mid z_t]$ as estimated by the S1A and S1B regression steps. We assume that $\bar{x}_t, \hat{x}_t \in \mathcal{X}$ and $\bar{y}_t, \hat{y}_t \in \mathcal{Y}$.

We want to analyze the convergence of the output of S2 regression—that is, of the weights $W$ given by ridge regression between S1A outputs and S1B outputs:

$$\hat{W}_\lambda = \left(\sum_{t=1}^{N} \hat{y}_t \otimes \hat{x}_t\right)\left(\sum_{t=1}^{T} \hat{x}_t \otimes \hat{x}_t + \lambda I_{\mathcal{X}}\right)^{-1} \tag{3.12}$$

Here $\otimes$ denotes tensor (outer) product, and $\lambda > 0$ is a regularization parameter that ensures the invertibility of the estimated covariance.

Before we state our main theorem we need to quantify the quality of S1 regression in a way that is independent of the S1 functional form. To do so, we place a bound on the S1 error

**Definition 3.2** (S1 Regression Bound)**.** *For any $\delta > 0$ and $N \in \mathbb{N}^+$, the **S1 regression bound** $\eta_{\delta,N} > 0$ is a number such that, with probability at least $(1 - \delta/2)$ the following holds:*

$$\frac{1}{N}\sum_{t=1}^{N} \|\bar{y}_t\|_{\mathcal{Y}} \|\hat{x}_t - \bar{x}_t\|_{\mathcal{X}} + \|\bar{x}_t\|_{\mathcal{X}} \|\hat{y}_t - \bar{y}_t\|_{\mathcal{Y}} + \|\hat{x}_t - \bar{x}_t\|_{\mathcal{X}} \|\hat{y}_t - \bar{y}_t\|_{\mathcal{Y}} \leq \eta_{\delta,N}$$

The S1 regression bound depends on the joint performance of two regression models. Below we show one possible method to construct such a bound

**Definition 3.3** (Uniform S1 Regression Bound)**.** *For any $\delta > 0$ and $N \in \mathbb{N}^+$, the **Uniform S1 regression bound** $\tilde{\eta}_{\delta,N} > 0$ is a number such that, with probability at least $(1-\delta/2)$, the following holds for all $1 \leq t \leq N$:*

$$\|\hat{x}_t - \bar{x}_t\|_{\mathcal{X}} < \tilde{\eta}_{\delta,N}$$
$$\|\hat{y}_t - \bar{y}_t\|_{\mathcal{Y}} < \tilde{\eta}_{\delta,N}$$

**Proposition 3.4.** *Let $\tilde{\eta}_{\delta,N}$ be a uniform S1 regression bound that satisfies Definition 3.3. Assuming that $\|\bar{y}_t\|_{\mathcal{Y}} < c$ and $\|\bar{x}_t\|_{\mathcal{X}}$, then*

$$\eta_{\delta,N} \equiv 2c\tilde{\eta}_{\delta,N} + \tilde{\eta}_{\delta,N}^2 = O(\tilde{\eta}_{\delta,N})$$

*satisfies Definition 3.2.*

A consistent learning algorithm requires that, for each fixed $\delta$, $\lim_{N\to\infty} \eta_{\delta,N} = 0$. Thus, the uniform regression bound might seem to be a strong assumption. However, we show examples where it is realizable in the following subsection.

In many applications, $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ will be finite dimensional real vector spaces: $\mathbb{R}^{d_x}$, $\mathbb{R}^{d_y}$ and $\mathbb{R}^{d_z}$. However, for generality we state our results in terms of arbitrary reproducing kernel Hilbert spaces. In this case S2 uses kernel ridge regression, leading to methods such as HSE-PSRs (Boots et al., 2013), which we discuss in Chapter 4. For this purpose, let $\mathcal{C}_{\bar{x}}$ and $\mathcal{C}_{\bar{y}}$ denote the (uncentered) covariance operators of $\bar{x}$ and $\bar{y}$ respectively: $\mathcal{C}_{\bar{x}} = \mathbb{E}[\bar{x} \otimes \bar{x}]$, $\mathcal{C}_{\bar{y}} = \mathbb{E}[\bar{y} \otimes \bar{y}]$. And, let $\overline{\mathcal{R}(\mathcal{C}_{\bar{x}})}$ denote the closure of the range of $\mathcal{C}_{\bar{x}}$.

With the above assumptions, Theorem 3.5 gives a generic error bound on S2 regression in terms of S1 regression error. If $\mathcal{X}$ and $\mathcal{Y}$ are finite dimensional and $\mathcal{C}_{\bar{x}}$ has full rank, then using ordinary least squares (i.e., setting $\lambda = 0$) will give the same bound, but with $\lambda$ in the first two terms replaced by the minimum eigenvalue of $\mathcal{C}_{\bar{x}}$, and the last term dropped.

**Theorem 3.5.** *Assume that $\|\bar{x}\|_{\mathcal{X}}, \|\bar{x}\|_{\mathcal{Y}} < c < \infty$ almost surely. Assume $W$ is a Hilbert-Schmidt operator, and let $\hat{W}_\lambda$ be as defined in* (3.12)*. Then, with probability at least $1-\delta$, for each $x_{\text{test}} \in \overline{\mathcal{R}(\mathcal{C}_{\bar{x}})}$ s.t. $\|x_{\text{test}}\|_{\mathcal{X}} \leq 1$, the error $\|\hat{W}_\lambda x_{\text{test}} - W x_{\text{test}}\|_{\mathcal{Y}}$ is bounded by*

$$\underbrace{O\left(\eta_{\delta,N}\left(\frac{1}{\lambda} + \frac{\sqrt{1+\sqrt{\frac{\log(1/\delta)}{N}}}}{\lambda^{\frac{3}{2}}}\right)\right)}_{\text{error in S1 regression}} + \underbrace{O\left(\frac{\log(1/\delta)}{\sqrt{N}}\left(\frac{1}{\lambda} + \frac{1}{\lambda^{\frac{3}{2}}}\right)\right)}_{\text{error from finite samples}} + \underbrace{O\left(\sqrt{\lambda}\right)}_{\text{error from regularization}}$$

**Remark 3.6.** *A variation of theorem 3.5 applies if the true model is not linear. In this case the reference value $W$ is linear predictor of $y$ given $\bar{x}$ that minimizes mean-square-error.*

It is important to note that Theorem 3.5 assumes $x_{\text{test}} \in \overline{\mathcal{R}(\mathcal{C}_{\bar{x}})}$. For dynamical systems, all valid states satisfy this property. However, with finite data, estimation errors may cause the estimated state $\hat{q}_t$ (i.e., $x_{\text{test}}$) to have a non-zero component in $\mathcal{R}^\perp(\mathcal{C}_{\bar{x}})$, the orthogonal complement of

$\overline{\mathcal{R}(\mathcal{C}_{\bar{x}})}$. Lemma 3.7 bounds the effect of such errors: it states that, in a stable system, this component gets smaller as S1 regression performs better. The main limitation of Lemma 3.7 is the assumption that $f_{\text{filter}}$ is $L$-Lipchitz, which essentially means that the model's estimated probability for $o_t$ is bounded below. A similar condition was assumed in (Hsu et al., 2009) for hidden Markov models. To gaurantee this property depends heavily on the filtering function. Therefore, Lemma 3.7 provides suggestive evidence rather than a guarantee that our learned dynamical system will predict well.

**Lemma 3.7.** *For observations $o_{1:T}$, let $\hat{q}_t$ be the estimated state given $o_{1:t-1}$. Let $\tilde{q}_t$ be the projection of $\hat{q}_t$ onto $\overline{\mathcal{R}(\mathcal{C}_{\bar{x}})}$. Assume $f_{\text{filter}}$ is $L$-Lipchitz on $p_t$ when evaluated at $o_t$, and $f_{\text{filter}}(p_t, o_t) \in \overline{\mathcal{R}(\mathcal{C}_{\bar{x}})}$ for any $p_t \in \overline{\mathcal{R}(\mathcal{C}_{\bar{y}})}$. Given the assumptions of theorem 3.5 and assuming that $\|\hat{q}_t\|_{\mathcal{X}} \leq R$ for all $1 \leq t \leq T$, the following holds for all $1 \leq t \leq T$ with probability at least $1 - \delta/2$.*

$$\|\epsilon_t\|_{\mathcal{X}} = \|\hat{q}_t - \tilde{q}_t\|_{\mathcal{X}} = O\left(\frac{\eta_{\delta,N}}{\sqrt{\lambda}}\right)$$

Since $\hat{W}_\lambda$ is bounded, the prediction error due to $\epsilon_t$ diminishes at the same rate as $\|\epsilon_t\|_{\mathcal{X}}$.

### 3.4.1 Examples of Uniform S1 Regression Bounds

The following propositions provide concrete examples of uniform S1 regression bounds $\tilde{\eta}_{\delta,N}$ for practical regression models.

**Proposition 3.8.** *Assume $\mathcal{X} \equiv \mathbb{R}^{d_x}, \mathbb{R}^{d_y}, \mathbb{R}^{d_z}$ for some $d_x, d_y, d_z < \infty$ and that $\bar{x}$ and $\bar{y}$ are linear vector functions of $z$ where the parameters are estimated using ordinary least squares. Assume that $\|\bar{x}\|_{\mathcal{X}}, \|\bar{y}\|_{\mathcal{Y}} < c < \infty$ almost surely. Let $\tilde{\eta}_{\delta,N}$ be defined as*

$$\tilde{\eta}_{\delta,N} = O\left(\sqrt{\frac{d_z}{N}} \log((d_x + d_y)/\delta)\right).$$

*Then, $\tilde{\eta}_{\delta,N}$ satifies Definition 3.3.*

*Proof.* (sketch) This is based on results that bound parameter estimation error in linear regression with univariate response (e.g. (Hsu et al., 2012a)). Note that if $\bar{x}_{ti} = U_i^\top z_t$ for some $U_i \in \mathcal{Z}$, then a bound on the error norm $\|\hat{U}_i - U_i\|$ implies a uniform bound of the same rate on $\hat{x}_i - \bar{x}$. The probability of exceeding the bound is scaled by $1/(d_x + d_y)$ to correct for multiple regressions. □

Variants of Proposition 3.8 can also be developed using bounds on non-linear regression models (e.g., generalized linear models).

The next proposition addresses a scenario where $\mathcal{X}$ and $\mathcal{Y}$ are infinite dimensional.

**Proposition 3.9.** *Assume that $x$ and $y$ are kernel evaluation functionals, $\bar{x}$ and $\bar{y}$ are linear vector functions of $z$ where the linear operator is estimated using conditional mean embedding (Song et al., 2009) with regularization parameter $\lambda_0 > 0$ and that $\|\bar{x}\|_{\mathcal{X}}, \|\bar{y}\|_{\mathcal{Y}} < c < \infty$ almost surely.*

*Let $\tilde{\eta}_{\delta,N}$ be defined as follows that*

$$\eta_{\delta,N} = O\left(\sqrt{\lambda_0} + \sqrt{\frac{\log(N/\delta)}{\lambda_0 N}}\right).$$

*Then, $\tilde{\eta}_{\delta,N}$ satifies Definition 3.3.*

*Proof.* (sketch) This bound is based on (Song et al., 2009), which gives a bound on the error in estimating the conditional mean embedding. The error probability is adjusted by $\delta/4N$ to accommodate the requirement that the bound holds for all training data. □

## 3.5 Experiments and Results

We now demonstrate examples of tweaking the S1 regression to gain advantage. In the first experiment we show that nonlinear regression can be used to reduce the number of parameters needed in S1, thereby improving statistical performance for learning an HMM. In the second experiment we show that we can encode prior knowledge as regularization.

### 3.5.1 Learning A Knowledge Tracing Model

In this experiment we attempt to model and predict the performance of students learning from an interactive computer-based tutor. We use the Bayesian knowledge tracing (BKT) model (Corbett and Anderson, 1995), which is essentially a 2-state HMM: the state $s_t$ represents whether a student has learned a knowledge component (KC), and the observation $o_t$ represents the success/failure of solving the $t^{\text{th}}$ question in a sequence of questions that cover this KC. With high probability, the student remains in the same knowledge state (learned or unlearned) and with smaller probability, the student may transition from unlearned to learned (learning) or learned to unlearned (forgetting). Most likely, the observation $o_t$ is assumed to match the state $s_t$. However, there is a possibility of *guessing* the right answer without having learned the tested knowledge component. There is also a possibility of *slipping* and giving an incorrect answer despite having learned the tested knowledge component. The possible transitions and observations are summarized in figure 3.4.

#### 3.5.1.1 Data Description

We evaluate the model using the "Geometry Area (1996-97)" data available from DataShop (Koedinger et al., 2010). This data was generated by students learning introductory geometry, and contains attempts by 59 students in 12 knowledge components. As is typical for BKT, we consider a student's attempt at a question to be correct if and only if the student entered the correct answer on the first try, without requesting any hints from the help system (since later attempts may be influenced by feedback from the tutor). Each training sequence consists of a sequence of first attempts for a student/KC pair. We discard sequences of length less than 5, resulting in a total of 325 sequences.

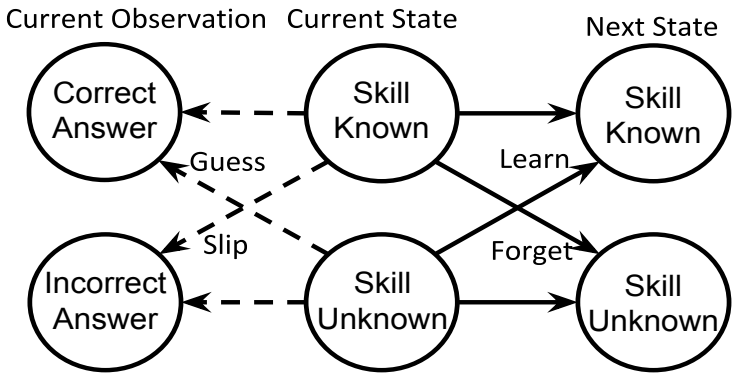



Figure 3.4: Transitions and observations in BKT. Each node represents a possible *value* of the state or observation. Solid arrows represent transitions while dashed arrows represent observations.

#### 3.5.1.2 Models and Evaluation

Under the (reasonable) assumption that the two states have distinct observation probabilities, this model is 1-observable. Hence we define the predictive state to be the expected next observation, which results in the following statistics: $\psi_t = o_t$ and $\xi_t = o_t \otimes_k o_{t+1}$, where $o_t$ is represented by a 2 dimensional indicator vector and $\otimes_k$ denotes the Kronecker product. Given these statistics, the extended state $p_t = \mathbb{E}[\xi_t \mid o_{1:t-1}]$ is a joint probability table of $o_{t:t+1}$ from which conditioning on $o_t$ ($f_{\text{filter}}$) is a simple operation.

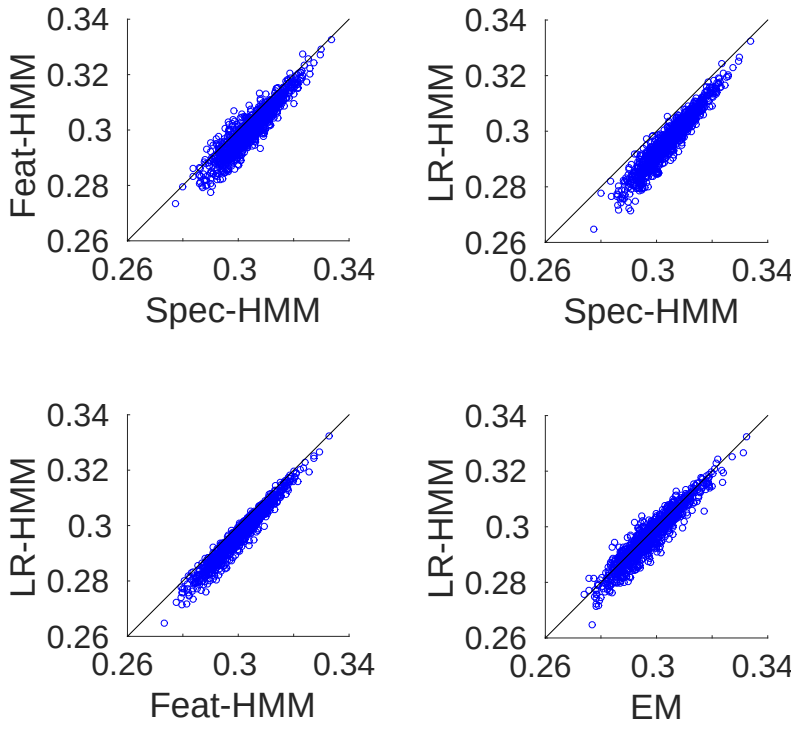
We compare three models that differ by history features and S1 regression method:

- **Spec-HMM:**
  This baseline uses $h_t = o_{t-1}$ and linear S1 regression, making it equivalent to the spectral HMM method of (Hsu et al., 2009).
- **Feat-HMM:**
  This baseline represents $h_t$ by an indicator vector of the joint assignment of the previous $\tau_h$ observations (we set $\tau_h$ to 4) and uses linear S1 regression. This is essentially a feature-based spectral HMM (Siddiqi et al., 2010). It thus incorporates more history information compared to Spec-HMM at the expense of increasing the number of S1 parameters by $O(2^{\tau_h})$.
- **LR-HMM:**
  This model represents $h_t$ by a binary vector of length $\tau_h$ encoding the previous $\tau_h$ observations and uses logistic regression as the S1 model. Thus, it uses the same history information as Feat-HMM but reduces the number of parameters to $O(\tau_h)$ at the expense of inductive bias.

We evaluated the above models using 1000 random splits of the 325 sequences into 200 training and 125 testing. For each testing observation $o_t$ we compute the absolute error between actual and expected value (i.e. $|\delta_{o_t=1} - \hat{P}(o_t = 1 \mid o_{1:t-1})|$). We report the mean absolute error for each split. The results are displayed in Figure 3.5.[6] We see that, while incorporating more history information increases accuracy (Feat-HMM vs. Spec-HMM), being able to incorporate the same information

[6]The differences have similar sign but smaller magnitude if we use RMSE instead of MAE.

using a more compact model gives an additional gain in accuracy (LR-HMM vs. Feat-HMM). We also compared the LR-HMM method to an HMM trained using expectation maximization (EM). We found that the LR-HMM model is much faster to train than EM while being on par with it in terms of prediction error.[7]




| Model | Spec-HMM | Feat-HMM | LR-HMM | EM |
|---|---|---|---|---|
| Training time (relative to Spec-HMM) | 1 | 1.02 | 2.219 | 14.323 |

Figure 3.5: Experimental results: each graph compares the performance of two models (measured by mean absolute error) on 1000 train/test splits. The black line is $x = y$. Points below this line indicate that model $y$ is better than model $x$. The table shows training time.

### 3.5.2 Modeling Independent Subsystems Using Lasso Regression

Spectral algorithms for Kalman filters typically use the left singular vectors of the covariance between history and future features as a basis for the state space. However, this basis hides any sparsity that might be present in our original basis. In this experiment, we show that we can instead use lasso (without dimensionality reduction) as our S1 regression algorithm to discover sparsity. This is useful, for example, when the system consists of multiple independent subsystems, each of which affects a subset of the observation coordinates.

[7]We used MATLAB's built-in logistic regression and EM functions.

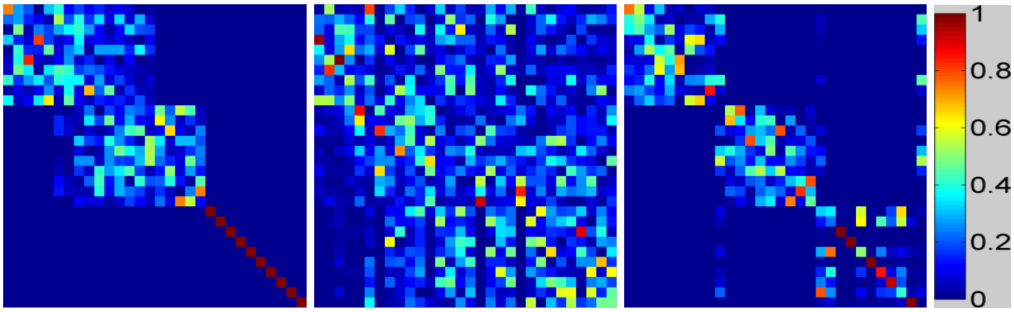



Figure 3.6: Left singular vectors of (left) true linear predictor from $o_{t-1}$ to $o_t$ (i.e. $OTO^+$), (middle) covariance matrix between $o_t$ and $o_{t-1}$ and (right) S1 Sparse regression weights. Each column corresponds to a singular vector (only absolute values are depicted). Singular vectors are ordered by their mean coordinate, which is computed as $\frac{\sum_{i=1}^{d} i|u_i|}{\sum_{i=1}^{d} |u_i|}$.

To test this idea we generate a sequence of 30-dimensional observations from a Kalman filter. Observation dimensions 1 through 10 and 11 through 20 are generated from two independent subsystems of state dimension 5. Dimensions 21-30 are generated from white noise. Each subsystem's transition and observation matrices have random Gaussian coordinates, with the transition matrix scaled to have a maximum eigenvalue of 0.95. States and observations are perturbed by Gaussian noise with covariance of $0.01I$ and $1.0I$ respectively.

We estimate the state space basis using 1000 examples (assuming 1-observability) and compare the singular vectors of the past to future regression matrix to those obtained from the Lasso regression matrix. The result is shown in figure 3.6. Clearly, using Lasso as stage 1 regression results in a basis that better matches the structure of the underlying system.

## 3.6 Related Work

This work extends method of moments learning algorithms for dynamical systems, more specifically subspace identification methods. These include spectral algorithms for Kalman filters (Boots, 2012; van Overschee and de Moor, 1996), Hidden Markov Models Hsu et al. (2009); Siddiqi et al. (2010), Predictive State Representations (PSRs) (Boots et al., 2011; Boots and Gordon, 2011) and Weighted Automata (Balle et al., 2014). One common aspect of subspace identification algorithms is that they use method of moments and exploit the covariance structure between future and past observation sequences to obtain an unbiased observable state representation. Boots and Gordon (2012) note the connection between this covariance and (linear) instrumental regression in the context of the HSE-HMM (Song et al., 2010).

On the other hand, there are learning algorithms that employ explicit reduction to supervised learning in a discriminative manner. This approach dates back to auto-regressive models Pandit and Wu (1983), where the state of the system is assumed to be fully determined by the previous $k$ observations. However, discriminative trianing approaches for systems with latent states have been proposed such as sufficient posterior representation (SPR) (Langford et al., 2009) and predictive

state inference machines (Sun et al., 2016), which we discussed in Chapter 2.

Our proposed framework of predictive state models has both characteristics mentioned above: it is a method of moments approach and it utilizes an explicit reduction to supervised learning. It differs from other method of moments approaches in that it allows for extra flexibility in specifying regression models, as opposed being constrained to linear least squares. At the same time, it differs from discriminative methods in that training consists of solving a fixed set of supervised learning problems, instead of being an iterative algorithm.

Also, the framework we present falls under the category of non-parametric instrumental regression. Prior work in that category include Hall and Horowitz (2005) and Darolles et al. (2011). They both rely on leveraging instrumental variables to estimate an operator in the space of square integrable functions, which is then used as an ingredient to estimate the regression function. For example, Darolles et al. (2011) relies on estimating the conditional expectation operator $T$ such that

$$g(z) = [Tf](z) = \mathbb{E}[f(x) \mid z],$$

for $f : \mathcal{X} \mapsto \mathbb{R}$, $z \in \mathcal{Z}$ and $x \in \mathcal{Z}$. The estimated function then is the one satisfying

$$\hat{T}f = \hat{r}$$

where $\hat{r}(z)$ estimates $\mathbb{E}[y \mid z]$.

The framework we present provides three advantages over these approaches: first, it allows the response variable to be infinite dimensional whereas these approaches assume a single dimensional response variable (so they are limited to independent finite response); second, it allows for a fully discriminative formulation whereas these approaches require the estimation of the joint density of input and instrument variables; and third, it provides flexibility on modeling the relation between instrumental variables and input variables and between instrumental variables and response variables.

Finally, it is worth mentioning that this framework was the basis of follow up work that exploited concepts from supervised learning literature. Venkatraman et al. (2016) combined predictive state models online regression to achieve an online Bayes filter learner. Xia (2016) used predictive state models with group sparsity to model dynamical systems of musical sequences.

## 3.7 Conclusion

In this chapter we described a general framework for unsupervised learning of recursive filters by reduction to supervised learning. The framework constructs a Bayes filter with a learnable linear extension step and a fixed conditioning step, and it relies on two key principles: first, we use a predictive state to represent both the belief state and the extended state as predictions of observed features. This means that observed features are unbiased but noisy estimates of these predictive states. Second, we use past features as instruments in an instrumental regression, which enables us to denoise state estimates and generate training examples to estimate system dynamics.

We have shown that this framework encompasses and provides a unified view of some previous successful dynamical system learning algorithms. We have also demonstrated that it can be used to

extend existing algorithms to incorporate nonlinearity and regularizers, resulting in better recursive filters.

# 3.A Appendix: Proofs

## 3.A.1 Proof of Main Theorem

In this section we provide a proof for theorem 3.5. We provide finite sample analysis of the effects of S1 regression, covariance estimation and regularization. The asymptotic statement becomes a natural consequence.

We will make use of matrix Bernstein's inequality stated below:

**Lemma 3.10** (Matrix Bernstein's Inequality Hsu et al. (2012b))**.** *Let $A$ be a random square symmetric matrix, and $r > 0$, $v > 0$ and $k > 0$ be such that, almost surely,*

$$\mathbb{E}[A] = 0, \quad \lambda_{\max}[A] \leq r,$$
$$\lambda_{\max}[\mathbb{E}[A^2]] \leq v, \quad \mathrm{tr}(\mathbb{E}[A^2]) \leq k.$$

*If $A_1, A_2, \ldots, A_N$ are independent copies of $A$, then for any $t > 0$,*

$$\Pr\left[\lambda_{\max}\left[\frac{1}{N}\sum_{t=1}^{N} A_t\right] > \sqrt{\frac{2vt}{N}} + \frac{rt}{3N}\right] \leq \frac{kt}{v}(e^t - t - 1)^{-1}. \tag{3.13}$$

*If $t \geq 2.6$, then $t(e^t - t - 1)^{-1} \leq e^{-t/2}$.*

Recall that, assuming $x_{test} \in \mathcal{R}(\mathcal{C}_{\bar{x}})$, we have three sources of error: first, the error in S1 regression causes the input to S2 regression procedure $(\hat{x}_t, \hat{y}_t)$ to be a perturbed version of the true $(\bar{x}_t, \bar{y}_t)$; second, the covariance operators are estimated from a finite sample of size $N$; and third, there is the effect of regularization. In the proof, we characterize the effect of each source of error. To do so, we define the following intermediate quantities:

$$W_\lambda = \mathcal{C}_{\bar{y},\bar{x}} \left(\mathcal{C}_{\bar{x}} + \lambda I\right)^{-1} \tag{3.14}$$

$$\bar{W}_\lambda = \hat{\mathcal{C}}_{\bar{y},\bar{x}} \left(\hat{\mathcal{C}}_{\bar{x}} + \lambda I\right)^{-1}, \tag{3.15}$$

where

$$\hat{\mathcal{C}}_{\bar{y},\bar{x}} \equiv \frac{1}{N}\sum_{t=1}^{N} \bar{y}_t \otimes \bar{x}_t$$

and $\hat{\mathcal{C}}_{\bar{x}}$ is defined similarly. Basically, $W_\lambda$ captures only the effect of regularization and $\bar{W}_\lambda$ captures in addition the effect of finite sample estimate of the covariance. $\bar{W}_\lambda$ is the result of S2 regression if $\bar{x}$ and $\bar{y}$ were perfectly recovered by S1 regression. It is important to note that $\hat{\mathcal{C}}_{\bar{x},\bar{y}}$ and $\hat{\mathcal{C}}_{\bar{x}}$ are *not* observable quantities since they depend on the true expectations $\bar{x}$ and $\bar{y}$. We will use $\lambda_{xi}$ and $\lambda_{yi}$ to denote the $i^{th}$ eigenvalue of $\mathcal{C}_{\bar{x}}$ and $\mathcal{C}_{\bar{y}}$ respectively in descending order and we will use $\|.\|$ to denote the operator norm.

Before we prove the main theorem, we define the quantities $\zeta_{\delta,N}^{\bar{x}\bar{x}}$ and $\zeta_{\delta,N}^{\bar{x}\bar{y}}$ which we use to bound the effect of covariance estimation from finite data, as stated in the following lemma:

**Lemma 3.11** (Covariance error bound)**.** *Let $N$ be a positive integer and $\delta \in (0,1)$ and assume that $\|\bar{x}\|, \|\bar{y}\| < c < \infty$ almost surely. Let $\zeta_{\delta,N}^{\bar{x}\bar{y}}$ be defined as:*

$$\zeta_{\delta,N}^{\bar{x}\bar{y}} = \sqrt{\frac{2vt}{N}} + \frac{rt}{3N}, \tag{3.16}$$

*where*

$$\begin{aligned} t &= \max(2.6, 2\log(4k/\delta v)) \\ r &= c^2 + \|\mathcal{C}_{\bar{y},\bar{x}}\| \\ v &= c^2 \max(\lambda_{y1}, \lambda_{x1}) + \|\mathcal{C}_{\bar{x},\bar{y}}\|^2 \\ k &= c^2(\mathrm{tr}(\mathcal{C}_{\bar{x}}) + \mathrm{tr}(\mathcal{C}_{\bar{y}})) \end{aligned}$$

*In addition, let $\zeta_{\delta,N}^{\bar{x}\bar{x}}$ be defined as:*

$$\zeta_{\delta,N}^{\bar{x}\bar{x}} = \sqrt{\frac{2v't'}{N}} + \frac{r't'}{3N}, \tag{3.17}$$

*where*

$$\begin{aligned} t' &= \max(2.6, 2\log(4k'/\delta v')) \\ r' &= c^2 + \lambda_{x1} \\ v' &= c^2\lambda_{x1} + \lambda_{x1}^2 \\ k' &= c^2\mathrm{tr}(\mathcal{C}_{\bar{x}}) \end{aligned}$$

*and define $\zeta_{\delta,N}^{\bar{y}\bar{y}}$ similarly for $\mathcal{C}_{\bar{y}}$.*

*It follows that, with probability at least $1 - \delta/2$,*

$$\begin{aligned} \|\hat{\mathcal{C}}_{\bar{y},\bar{x}} - \mathcal{C}_{\bar{y},\bar{x}}\| &< \zeta_{\delta,N}^{\bar{x}\bar{y}} \\ \|\hat{\mathcal{C}}_{\bar{x}} - \mathcal{C}_{\bar{x}}\| &< \zeta_{\delta,N}^{\bar{x}\bar{x}} \\ \|\hat{\mathcal{C}}_{\bar{y}} - \mathcal{C}_{\bar{y}}\| &< \zeta_{\delta,N}^{\bar{y}\bar{y}} \end{aligned}$$

*Proof.* We show that each statement holds with probability at least $1-\delta/6$. The claim then follows directly from the union bound. We start with $\zeta_{\delta,N}^{\bar{x}\bar{x}}$. By setting $A_t = \bar{x}_t \otimes \bar{x}_t - \mathcal{C}_{\bar{x}}$ then we would like to obtain a high probability bound on $\|\frac{1}{N}\sum_{t=1}^{N} A_t\|$. Lemma 3.10 shows that, in order to satisfy the bound with probability at least $1 - \delta/6$, it suffices to set $t$ to $\max(2.6, 2k\log(6/\delta v))$. So, it remains to find suitable values for $r, v$ and $k$:

$$\begin{aligned} \lambda_{\max}[A] &\leq \|\bar{x}\|^2 + \|\mathcal{C}_{\bar{x}}\| \leq c^2 + \lambda_{x1} = r' \\ \lambda_{\max}[\mathbb{E}[A^2]] &= \lambda_{\max}[\mathbb{E}[\|\bar{x}\|^2(\bar{x}\otimes\bar{x}) - (\bar{x}\otimes\bar{x})\mathcal{C}_{\bar{x}} - \mathcal{C}_{\bar{x}}(\bar{x}\otimes\bar{x}) + \mathcal{C}_{\bar{x}}^2] \\ &= \lambda_{\max}[\mathbb{E}[\|\bar{x}\|^2(\bar{x}\otimes\bar{x}) - \mathcal{C}_{\bar{x}}^2]] \leq c^2\lambda_{x1} + \lambda_{x1}^2 = v' \\ \mathrm{tr}[\mathbb{E}[A^2]] &= \mathrm{tr}[\mathbb{E}[\|\bar{x}\|^2(\bar{x}\otimes\bar{x}) - \mathcal{C}_{\bar{x}}^2]] \leq \mathrm{tr}[\mathbb{E}[\|\bar{x}\|^2(\bar{x}\otimes\bar{x})]] \leq c^2\mathrm{tr}(\mathcal{C}_{\bar{x}}) = k' \end{aligned}$$

The case of $\zeta_{\delta,N}^{\bar{y}\bar{y}}$ can be proven similarly. Now moving to $\zeta_{\delta,N}^{\bar{x}\bar{y}}$, we have $B_t = \bar{y}_t \otimes \bar{x}_t - \mathcal{C}_{\bar{y},\bar{x}}$. Since $B_t$ is not square, we use the Hermitian dilation $\mathscr{H}(B)$ defined as followsTropp (2012):

$$A = \mathscr{H}(B) = \begin{bmatrix} 0 & B \\ B^* & 0 \end{bmatrix}$$

Note that

$$\lambda_{\max}[A] = \|B\|, \quad A^2 = \begin{bmatrix} BB^* & 0 \\ 0 & B^*B \end{bmatrix}$$

therefore suffices to bound $\|\frac{1}{N}\sum_{t=1}^{N} A_t\|$ using an argument similar to that used in $\zeta_{\delta,N}^{\bar{x}\bar{x}}$ case. □

To prove theorem 3.5, we write

$$\begin{aligned} \|\hat{W}_\lambda x_{\text{test}} - W x_{\text{test}}\|_{\mathcal{Y}} \leq & \|(\hat{W}_\lambda - \bar{W}_\lambda)\bar{x}_{\text{test}}\|_{\mathcal{Y}} \\ & + \|(\bar{W}_\lambda - W_\lambda)\bar{x}_{\text{test}}\|_{\mathcal{Y}} \\ & + \|(W_\lambda - W)\bar{x}_{\text{test}}\|_{\mathcal{Y}} \end{aligned} \tag{3.18}$$

We will now present bounds on each term. We consider the case where $\bar{x}_{\text{test}} \in \mathcal{R}(\mathcal{C}_{\bar{x}})$. Extension to $\overline{\mathcal{R}(\mathcal{C}_{\bar{x}})}$ is a result of the assumed boundedness of $W$, which implies that $\hat{W}_\lambda - W$ is bounded and hence continuous.

**Lemma 3.12** (Error due to S1 Regression)**.** *Assume that $\|\bar{x}\|, \|\bar{y}\| < c < \infty$ almost surely, and let $\eta_{\delta,N}$ be as defined in Definition 3.2. The following holds with probability at least $1-\delta$*

$$\begin{aligned} \|\hat{W}_\lambda - \bar{W}_\lambda\| &\leq \sqrt{\lambda_{y1} + \zeta_{\delta,N}^{\bar{y}\bar{y}}}\frac{\eta_{\delta,N}}{\lambda^{\frac{3}{2}}} + \frac{\eta_{\delta,N}}{\lambda} \\ &= O\left(\eta_{\delta,N}\left(\frac{1}{\lambda} + \frac{\sqrt{1 + \frac{\log(1/\delta)}{\sqrt{N}}}}{\lambda^{\frac{3}{2}}}\right)\right). \end{aligned}$$

*The asymptotic statement assumes $\eta_{\delta,N} \to 0$ as $N \to \infty$.*

*Proof.* Write $\hat{\mathcal{C}}_{\hat{x}} = \hat{\mathcal{C}}_{\bar{x}} + \Delta_x$ and $\hat{\mathcal{C}}_{\hat{y},\hat{x}} = \hat{\mathcal{C}}_{\bar{y},\bar{x}} + \Delta_{yx}$. We know that, with probability at least $1-\delta/2$, the following is satisfied for all unit vectors $\phi_x \in \mathcal{X}$ and $\phi_y \in \mathcal{Y}$

$$\begin{aligned} \langle \phi_y, \Delta_{yx}\phi_x \rangle_{\mathcal{Y}} &= \frac{1}{N}\sum_{t=1}^{N} \langle \phi_y, \hat{y}_t \rangle_{\mathcal{Y}} \langle \phi_x, \hat{x}_t \rangle_{\mathcal{X}} \\ &- \langle \phi_y, \hat{y}_t \rangle_{\mathcal{Y}} \langle \phi_x, \bar{x}_t \rangle_{\mathcal{X}} \\ &+ \langle \phi_y, \hat{y}_t \rangle_{\mathcal{Y}} \langle \phi_x, \bar{x}_t \rangle_{\mathcal{X}} - \langle \phi_y, \bar{y}_t \rangle_{\mathcal{Y}} \langle \phi_x, \bar{x}_t \rangle_{\mathcal{X}} \\ &= \frac{1}{N}\sum_{t} \langle \phi_y, \bar{y}_t + (\hat{y}_t - \bar{y}_t) \rangle_{\mathcal{Y}} \langle \phi_x, \hat{x}_t - \bar{x}_t \rangle_{\mathcal{X}} \\ &+ \langle \phi_y, \hat{y}_t - \bar{y}_t \rangle_{\mathcal{Y}} \langle \phi_x, \bar{x}_t \rangle_{\mathcal{X}} \\ &\leq \frac{1}{N}\sum_{t=1}^{N} \|\bar{y}_t\|_{\mathcal{Y}} \|\hat{x}_t - \bar{x}_t\|_{\mathcal{X}} + \|\bar{x}_t\|_{\mathcal{X}} \|\hat{y}_t - \bar{y}_t\|_{\mathcal{Y}} + \|\hat{x}_t - \bar{x}_t\|_{\mathcal{X}} \|\hat{y}_t - \bar{y}_t\|_{\mathcal{Y}} \\ &\leq \eta_{\delta,N} \end{aligned}$$

Therefore,

$$\|\Delta_{yx}\| = \sup_{\|\phi_x\|_{\mathcal{X}}\leq 1, \|\phi_y\|_{\mathcal{Y}}\leq 1} \langle \phi_y, \Delta_{yx}\phi_x\rangle_{\mathcal{Y}} \leq \eta_{\delta,N},$$

and similarly

$$\|\Delta_x\| \leq \eta_{\delta,N},$$

with probability $1-\delta/2$. We can write

$$\begin{aligned}\hat{W}_\lambda - \bar{W}_\lambda = \hat{\mathcal{C}}_{\bar{y},\bar{x}} \left( (\hat{\mathcal{C}}_{\bar{x}} + \Delta_x + \lambda I)^{-1} - (\hat{\mathcal{C}}_{\bar{x}} + \lambda I)^{-1} \right) \\ + \Delta_{yx}(\hat{\mathcal{C}}_{\bar{x}} + \Delta_x + \lambda I)^{-1}\end{aligned}$$

Using the fact that $B^{-1} - A^{-1} = B^{-1}(A-B)A^{-1}$ for invertible operators $A$ and $B$ we get

$$\begin{aligned}\hat{W}_\lambda - \bar{W}_\lambda = -\hat{\mathcal{C}}_{\bar{y},\bar{x}}(\hat{\mathcal{C}}_{\bar{x}} + \lambda I)^{-1}\Delta_x(\hat{\mathcal{C}}_{\bar{x}} + \Delta_x + \lambda I)^{-1} \\ + \Delta_{yx}(\hat{\mathcal{C}}_{\bar{x}} + \Delta_x + \lambda I)^{-1}\end{aligned}$$

we then use the decomposition $\hat{\mathcal{C}}_{\bar{y},\bar{x}} = \hat{\mathcal{C}}_{\bar{y}}^{\frac{1}{2}} V \hat{\mathcal{C}}_{\bar{x}}^{\frac{1}{2}}$, where $V$ is a correlation operator satisfying $\|V\| \leq 1$. This gives

$$\begin{aligned}&\hat{W}_\lambda - \bar{W}_\lambda = \\ &- \hat{\mathcal{C}}_{\bar{y}}^{\frac{1}{2}} V \hat{\mathcal{C}}_{\bar{x}}^{\frac{1}{2}} (\hat{\mathcal{C}}_{\bar{x}} + \lambda I)^{-\frac{1}{2}} (\hat{\mathcal{C}}_{\bar{x}} + \lambda I)^{-\frac{1}{2}} \Delta_x (\hat{\mathcal{C}}_{\bar{x}} + \Delta_x + \lambda I)^{-1} \\ &+ \Delta_{yx}(\hat{\mathcal{C}}_{\bar{x}} + \Delta_x + \lambda I)^{-1}\end{aligned}$$

Noting that $\|\hat{\mathcal{C}}_{\bar{x}}^{\frac{1}{2}} (\hat{\mathcal{C}}_{\bar{x}} + \lambda I)^{-\frac{1}{2}}\| \leq 1$, the rest of the proof follows from triangular inequality and the fact that $\|AB\| \leq \|A\|\|B\|$ □

**Lemma 3.13** (Error due to Covariance)**.** *Assuming that $\|\bar{x}\|_{\mathcal{X}}, \|\bar{y}\|_{\mathcal{Y}} < c < \infty$ almost surely, the following holds with probability at least $1 - \frac{\delta}{2}$*

$$\|\bar{W}_\lambda - W_\lambda\| \leq \sqrt{\lambda_{y1}} \zeta_{\delta,N}^{\bar{x}\bar{x}} \lambda^{-\frac{3}{2}} + \frac{\zeta_{\delta,N}^{\bar{x}\bar{y}}}{\lambda}$$

*, where $\zeta_{\delta,N}^{\bar{x}\bar{x}}$ and $\zeta_{\delta,N}^{\bar{x}\bar{y}}$ are as defined in Lemma 3.11.*

*Proof.* Write $\hat{\mathcal{C}}_{\bar{x}} = \mathcal{C}_{\bar{x}} + \Delta_x$ and $\hat{\mathcal{C}}_{\bar{y},\bar{x}} = \mathcal{C}_{\bar{y},\bar{x}} + \Delta_{yx}$. Then we get

$$\bar{W}_\lambda - W_\lambda = \mathcal{C}_{\bar{y},\bar{x}} \left( (\mathcal{C}_{\bar{x}} + \Delta_x + \lambda I)^{-1} - (\mathcal{C}_{\bar{x}} + \lambda I)^{-1} \right) + \Delta_{yx}(\mathcal{C}_{\bar{x}} + \Delta_x + \lambda I)^{-1}$$

Using the fact that $B^{-1} - A^{-1} = B^{-1}(A-B)A^{-1}$ for invertible operators $A$ and $B$ we get

$$\bar{W}_\lambda - W_\lambda = -\mathcal{C}_{\bar{y},\bar{x}}(\mathcal{C}_{\bar{x}} + \lambda I)^{-1}\Delta_x(\mathcal{C}_{\bar{x}} + \Delta_x + \lambda I)^{-1} + \Delta_{yx}(\mathcal{C}_{\bar{x}} + \Delta_x + \lambda I)^{-1}$$

we then use the decomposition $\mathcal{C}_{\bar{y},\bar{x}} = \mathcal{C}_{\bar{y}}^{\frac{1}{2}} V \mathcal{C}_{\bar{x}}^{\frac{1}{2}}$, where $V$ is a correlation operator satisfying $\|V\| \leq 1$. This gives

$$\begin{aligned}\bar{W}_\lambda - W_\lambda = \\ &- \mathcal{C}_{\bar{y}}^{\frac{1}{2}} V \mathcal{C}_{\bar{x}}^{\frac{1}{2}} (\mathcal{C}_{\bar{x}} + \lambda I)^{-\frac{1}{2}} (\mathcal{C}_{\bar{x}} + \lambda I)^{-\frac{1}{2}} \\ &. \Delta_x (\mathcal{C}_{\bar{x}} + \Delta_x + \lambda I)^{-1} \\ &+ \Delta_{yx} (\mathcal{C}_{\bar{x}} + \Delta_x + \lambda I)^{-1}\end{aligned}$$

Noting that $\|\mathcal{C}_{\bar{x}}^{\frac{1}{2}} (\mathcal{C}_{\bar{x}} + \lambda I)^{-\frac{1}{2}}\| \leq 1$, the rest of the proof follows from triangular inequality and the fact that $\|AB\| \leq \|A\|\|B\|$ □

**Lemma 3.14** (Error due to Regularization on inputs within $\mathcal{R}(\mathcal{C}_{\bar{x}})$)**.** *For any $x \in \mathcal{R}(\mathcal{C}_{\bar{x}})$ s.t. $\|x\|_{\mathcal{X}} \leq 1$ and $\|\mathcal{C}_{\bar{x}}^{-\frac{1}{2}} x\|_{\mathcal{X}} \leq C$. The following holds*

$$\|(W_\lambda - W)x\|_{\mathcal{Y}} \leq \frac{1}{2}\sqrt{\lambda}\|W\|_{HS} C$$

*Proof.* Since $x \in \mathcal{R}(\mathcal{C}_{\bar{x}}) \subseteq \mathcal{R}(\mathcal{C}_{\bar{x}}^{\frac{1}{2}})$, we can write $x = \mathcal{C}_{\bar{x}}^{\frac{1}{2}} v$ for some $v \in \mathcal{X}$ s.t. $\|v\|_{\mathcal{X}} \leq C$. Then

$$(W_\lambda - W)x = \mathcal{C}_{\bar{y},\bar{x}} ((\mathcal{C}_{\bar{x}} + \lambda I)^{-1} - \mathcal{C}_{\bar{x}}^{-1}) \mathcal{C}_{\bar{x}}^{\frac{1}{2}} v$$

Let $D = \mathcal{C}_{\bar{y},\bar{x}} ((\mathcal{C}_{\bar{x}} + \lambda I)^{-1} - \mathcal{C}_{\bar{x}}^{-1}) \mathcal{C}_{\bar{x}}^{\frac{1}{2}}$. We will bound the Hilbert-Schmidt norm of $D$. Let $\psi_{xi} \in \mathcal{X}$, $\psi_{yi} \in \mathcal{Y}$ denote the eigenvector corresponding to $\lambda_{xi}$ and $\lambda_{yi}$ respectively. Define $s_{ij} = |\langle \psi_{yj}, \mathcal{C}_{\bar{x},\bar{y}} \psi_{xi} \rangle_{\mathcal{Y}}|$. Then we have

$$\begin{aligned}|\langle \psi_{yj}, D\psi_{xi} \rangle_{\mathcal{Y}}| &= \left| \langle \psi_{yj}, \mathcal{C}_{\bar{y},\bar{x}} \frac{\lambda}{(\lambda_{xi} + \lambda)\sqrt{\lambda_{xi}}} \psi_{xi} \rangle_{\mathcal{Y}} \right| \\ &= \frac{\lambda s_{ij}}{(\lambda_{xi} + \lambda)\sqrt{\lambda_{xi}}} = \frac{s_{ij}}{\sqrt{\lambda_{xi}}} \frac{1}{\frac{1}{\lambda/\lambda_{xi}} + 1} \\ &\leq \frac{s_{ij}}{\sqrt{\lambda_{xi}}} . \frac{1}{2} \sqrt{\frac{\lambda}{\lambda_{xi}}} = \frac{1}{2}\sqrt{\lambda} \frac{s_{ij}}{\lambda_{xi}} \\ &= \frac{1}{2}\sqrt{\lambda} |\langle \psi_{yj}, W\psi_{xi} \rangle_{\mathcal{Y}}|,\end{aligned}$$

where the inequality follows from the arithmetic-geometric-harmonic mean inequality. This gives the following bound

$$\|D\|_{HS}^2 = \sum_{i,j} \langle \psi_{yj}, D\psi_{xi} \rangle_{\mathcal{Y}}^2 \leq \frac{1}{2}\sqrt{\lambda}\|W\|_{HS}^2$$

and hence

$$\begin{aligned}\|(W_\lambda - W)x\|_{\mathcal{Y}} &\leq \|D\|\|v\|_{\mathcal{X}} \leq \|D\|_{HS}\|v\|_{\mathcal{X}} \\ &\leq \frac{1}{2}\sqrt{\lambda}\|W\|_{HS}C\end{aligned}$$

□

Note that the additional assumption that $\|\mathcal{C}_{\bar{x}}^{-\frac{1}{2}}x\|_{\mathcal{X}} \leq C$ is not required to obtain an asymptotic $O(\sqrt{\lambda})$ rate for a given $x$. This assumption, however, allows us to uniformly bound the constant. Theorem 3.5 is simply the result of plugging the bounds in Lemmata 3.12, 3.13, and 3.14 into (3.18) and using the union bound.

### 3.A.2 Proof of Lemma 3.7

for $t = 1$: Let $\mathcal{I}$ be an index set over training instances such that

$$\hat{q}_1^{\text{test}} = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\hat{q}_i$$

Then

$$\|\hat{q}_1^{\text{test}} - \tilde{q}_1^{\text{test}}\|_{\mathcal{X}} \leq \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\|\hat{q}_i - \tilde{q}_i\|_{\mathcal{X}} \leq \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}\|\hat{q}_i - q_i\|_{\mathcal{X}} \leq \eta_{\delta,N}$$

for $t > 1$: Let $A$ denote a projection operator on $\mathcal{R}^{\perp}(\mathcal{C}_{\bar{y}})$

$$\begin{aligned}\|\hat{q}_{t+1}^{\text{test}} - \tilde{q}_{t+1}^{\text{test}}\|_{\mathcal{X}} &\leq L\|\hat{p}_t^{\text{test}} - \tilde{p}_t^{\text{test}}\|_{\mathcal{Y}} \leq L\|A\hat{W}_\lambda\hat{q}_t^{\text{test}}\|_{\mathcal{Y}} \\ &\leq L\left\|\frac{1}{N}\left(\sum_{i=1}^{N}A\hat{p}_i\otimes\hat{q}_i\right)\left(\frac{1}{N}\sum_{i=1}^{N}\hat{q}_i\otimes\hat{q}_i + \lambda I\right)^{-1}\right\|\left\|\hat{q}_t^{\text{test}}\right\|_{\mathcal{X}} \\ &\leq L\left\|\frac{1}{N}\sum_{i=1}^{N}A\hat{p}_i\otimes A\hat{p}_i\right\|^{\frac{1}{2}}\frac{1}{\sqrt{\lambda}}\|\hat{q}_t^{\text{test}}\|_{\mathcal{X}} \leq L\frac{\eta_{\delta,N}}{\sqrt{\lambda}}\|\hat{q}_t^{\text{test}}\|_{\mathcal{X}},\end{aligned}$$

where the second to last inequality follows from the decomposition similar to $\Sigma_{YX} = \Sigma_Y^{\frac{1}{2}}V\Sigma_X^{\frac{1}{2}}$, and the last inequality follows from the fact that $\|A\hat{p}_i\|_{\mathcal{Y}} \leq \|\hat{p}_i - \bar{p}_i\|_{\mathcal{Y}}$. □

# Chapter 4

# A Practical Non-parametric Predicive State Model for Continuous Systems

In this chapter, we use the framework we proposed in Chapter 3 to develop a practical non-parametric recursive filter that is applicable to continuous systems. The model we propose is based on a recent continuous extension of predictive state representation proposed by Boots et al. (2013) called Hilbert space embedding of predictive state representations (HSE-PSR). After overviewing the necessary mathematical background in Section 4.1, we describe this model as a predictive state model trained by the two-stage regression method in Section 4.2. We then propose a practical approximation of HSE-PSRs using random Fourier features in Section 4.1.5. In Section 4.3.2 we describe how to apply discriminative training techniques to improve over the performance of two-stage regression. The result is a special recurrent network architecture that supports a method-of-moment-based initialization before applying backpropagation through time. We demonstrate the efficacy of the proposed model through a range of experiments in Section 4.4

## 4.1 Hilbert Space Embedding of Distributions

Hilbert space embedding of distributions (Smola et al., 2007) is a non-parametric representation of probability distributions using elements in the reproducing kernel Hilbert space (RKHS) of a suitable kernel. We first provide a linear algebraic treatment of discrete distributions, which we then generalize to Hilbert space embedding.

### 4.1.1 Motivating Example: Discrete Distributions

Let $\mathcal{F}_N$ denote the set of all real-valued functions on a finite domain of cardinality $N$ (i.e. $\mathcal{F}_N \equiv \{f : [N] \mapsto \mathbb{R}\}$). We can represent each function $f \in \mathcal{F}$ as a weight vector $w_f \in \mathbb{R}^N$. Define the *delta kernel* $k_\delta : [N] \times [N] \mapsto \mathbb{R}$ such that $k_\delta(i, j) = 1(i = j)$. Also, define the delta *feature map* $\phi_\delta^N : [N] \mapsto \mathbb{R}^N$ such that $\phi_\delta^N(i) = e_i$. Note that, for any $i, j \in [N]$ and $f \in \mathcal{F}_N$, the following

holds

$$f(i) = \langle \phi_\delta^N(i), w_f \rangle$$
$$k_\delta(i,j) = \langle \phi_\delta^N(i), \phi_\delta^N(j) \rangle$$

It follows from the above that $\phi_\delta^N(i)$ is the weight vector corresponding to the *kernel evaluation functional* $k_\delta(i,.)$.

To see how the functions defined above are related to probability distributions, let $X \in [N], Y \in [M]$ be two random variables. It can be seen that the *mean map* $\mu_X \equiv \mathbb{E}[\phi_\delta^N(X)]$ uniquely identifies the distribution $\Pr(X)$. In fact, $\mu_X$ is but the probability vector of $X$. It can also be shown that $\mu_X$ is the unique element in $\mathbb{R}^N$ that satisfies

$$\mathbb{E}[f(X)] = \langle f, \mu_X \rangle, \forall f \in \mathcal{F}_N \tag{4.1}$$

Another relevant quantity is the covariance $\mathcal{C}_{X,Y} \equiv \mathbb{E}[\phi_\delta^N(X) \otimes \phi_\delta^M(Y)]$, which is the joint probability table of $X$ and $Y$ and also the unique $N \times M$ matrix that satisfies

$$E[f(X)g(Y)] = \langle f, \mathcal{C}_{X,Y} g \rangle \tag{4.2}$$

Finally, we are also interested in the conditional distribution $\Pr(X \mid Y)$. This is represented by a conditional probability table which is a matrix $\mathcal{W}_{X|Y}$ such that

$$\mathbb{E}[f(X) \mid Y = y] = \mathcal{W}_{X|Y} \phi_\delta^M(y), \forall f \in \mathcal{F}_N, y \in [M] \tag{4.3}$$

It can be shown that

$$\mathcal{W}_{X|Y} = \mathcal{C}_{Y,X} \mathcal{C}_X^{-1}. \tag{4.4}$$

We are now ready to generalize the aforementioned concepts to continuous distributions.

### 4.1.2 Kernels, RKHSs and Feature Maps

Let $\mathcal{I}_\mathcal{X}$ be a domain of interest (e.g. the set $[N]$ in the discrete example or the set of all observations). Let $\mathcal{X}$ be a Hilbert space of real functions on $\mathcal{I}_\mathcal{X}$. $\mathcal{X}$ is called a *reproducing kernel Hilbert space (RKHS)* of a kernel $k_\mathcal{X} : \mathcal{I}_\mathcal{X} \times \mathcal{I}_\mathcal{X} \mapsto \mathbb{R}$ if $k_\mathcal{X}$ satisfies the *reproducing property*

$$\begin{aligned} f(x) =& \langle f, k_\mathcal{X}(x,.) \rangle_\mathcal{X} \quad \forall x \in \mathcal{I}_\mathcal{X}, f \in \mathcal{X} \\ \text{and hence } k_\mathcal{X}(x_1,x_2) =& \langle k_\mathcal{X}(x_1,.), k_\mathcal{X}(x_2,.) \rangle_\mathcal{X} \quad \forall x_1, x_2 \in \mathcal{I}_\mathcal{X}. \end{aligned} \tag{4.5}$$

A kernel is said to be *universal*, if the corresponding RKHS is dense in the set of all bounded continuous functions on $\mathcal{I}_\mathcal{X}$— that is, any such function can be approximated to aribtrary precision by an element in the RKHS. The simplest example of a universal kernel is the delta kernel on a discrete domain. Another commonly used universal kernel is the Gaussian RBF kernel on a compact domain. The Gaussian RBF kernel is defined as

$$k_{\text{RBF}}(x_1,x_2) = \exp\left(-\frac{1}{2\sigma^2}\|x_1 - x_2\|^2\right), \tag{4.6}$$

where $\sigma$ is a hyper-parameter known as the *kernel bandwidth*. An alternative view to the RKHS is to think of a (possibly infinite dimensional) *feature space* $\mathcal{X}'$ such that any function $f \in \mathcal{X}$ has a corresponding "weight vector" $w_f \in \mathcal{X}'$ and there exists a *feature map* $\phi^{\mathcal{X}} : \mathcal{I}_{\mathcal{X}} \mapsto \mathcal{X}'$ such that

$$\begin{aligned} f(x) =& \langle w_f, \phi^{\mathcal{X}}(x) \rangle_{\mathcal{X}'} \quad \forall x \in \mathcal{I}_{\mathcal{X}}, f \in \mathcal{X} \\ \text{and hence } k_{\mathcal{X}}(x_1, x_2) =& \langle \phi^{\mathcal{X}}(x_1), \phi^{\mathcal{X}}(x_2) \rangle_{\mathcal{X}'} \quad \forall x_1, x_2 \in \mathcal{I}_{\mathcal{X}}. \end{aligned} \tag{4.7}$$

When defining mean maps and covariance operators, we will use the two notions interchangeably.

### 4.1.3 Mean Maps and Covariance Operators

Let $X \in \mathcal{I}_{\mathcal{X}}$ and $Y \in \mathcal{I}_{\mathcal{Y}}$ be two random variables. The kernel mean map of $X$ is defined as

$$\begin{aligned} \mu_X &= \mathbb{E}[k(X, .)] \\ \mu_X &= \mathbb{E}[\phi^{\mathcal{X}}(X)] \quad \text{(feature map representation)} \end{aligned} \tag{4.8}$$

The mean map $\mu_X$ exists in $\mathcal{X}$ if $\mathbb{E}[k_{\mathcal{X}}(X, X)] < \infty$ and, by the reproducing property, it satisfies

$$\mathbb{E}[f(X)] = \langle f, \mu_X \rangle_{\mathcal{X}} \quad \forall f \in \mathcal{X}. \tag{4.9}$$

A key result in (Smola et al., 2007) is that for a universal kernel, the mapping between probability distributions and mean maps is injective—that is, a mean map uniquely identifies the corresponding distribution. Given a set of i.i.d samples $\{x_1, \ldots, x_N\}$, the mean map can be estimated by replacing the expectations in (4.8) with the empirical average. Smola et al. (2007) show that, under mild assumptions on $\mathcal{X}$ and the distribution $\Pr(X)$, the estimation error $\|\hat{\mu}_X - \mu_X\|_{\mathcal{X}}$ goes to 0 with the rate $O(N^{-\frac{1}{2}})$ with high probability.

While the mean map represents the probability distribution $\Pr(X)$, the joint probability distribution $\Pr(X, Y)$ can be represented by the (uncentered) *covariance operator* $\mathcal{C}_{X,Y}$ (Song et al., 2009; Fukumizu et al., 2013), which is defined to be a linear operator from $\mathcal{Y}$ to $\mathcal{X}$ that satisfies

$$\mathbb{E}[f(X)g(Y)] = \langle f, \mathcal{C}_{X,Y} g \rangle_{\mathcal{X}} \quad \forall f \in \mathcal{X}, g \in \mathcal{Y}. \tag{4.10}$$

This operator can be shown to have the feature map form

$$\mathcal{C}_{X,Y} = \mathbb{E}[\phi^{\mathcal{X}}(X) \otimes \phi^{\mathcal{Y}}(Y)]. \tag{4.11}$$

Equation (4.11) shows that the covariance operator is equivalent to the mean map of the pair $(X, Y)$ on the space induced by the product kernel

$$k_{\mathcal{X}\mathcal{Y}}(x_1, y_1, x_2, y_2) = k_{\mathcal{X}}(x_1, x_2) k_{\mathcal{Y}}(y_1, y_2) = \langle \phi^{\mathcal{X}}(x_1), \phi^{\mathcal{X}}(x_2) \rangle_{\mathcal{X}} \langle \phi^{\mathcal{Y}}(y_1), \phi^{\mathcal{Y}}(y_2) \rangle_{\mathcal{Y}}.$$

Similar to the mean map, the covariance operator can be estimated from finite samples by empirical avergaing. The equivalence to mean maps implies that the empirical estimate converges to the true value under mild conditions.

### 4.1.4 Conditional Operators and Kernel Bayes Rule

It now remains to construct the equivalent of a conditional probability table. We would like to construct a map $\mathcal{W}_{X|Y}$ such that

$$\mathbb{E}[f(X) \mid Y = y] = \langle f, \mathcal{W}_{X|Y} k_{\mathcal{Y}}(y, .) \rangle_{\mathcal{X}} \quad \forall y \in \mathcal{I}_{\mathcal{Y}}, f \in \mathcal{X}, \tag{4.12}$$

or equivalently

$$\mu_{X|Y=y} \equiv \mathbb{E}[\phi^{\mathcal{X}}(X) \mid Y = y] = \mathcal{W}_{X|Y} \phi^{\mathcal{Y}}(y) \quad \forall y \in \mathcal{I}_{\mathcal{Y}} \tag{4.13}$$

Song et al. (2009) have shown that $\mathcal{W}_{X|Y}$ that satisfies (4.12) also satisfies

$$\mathcal{W}_{X|Y} \mathcal{C}_Y \equiv \mathcal{C}_{X,Y}. \tag{4.14}$$

In practice, $\mathcal{C}_Y$ is typically not invertible and we use a regularized version

$$\mathcal{W}^{\lambda}_{X|Y} \equiv \mathcal{C}_{X,Y}(\mathcal{C}_Y + \lambda I)^{-1}, \tag{4.15}$$

where $\lambda > 0$ is a regularization parameter. We estimate $\mathcal{W}^{\lambda}_{X|Y}$ by plugging the emprical estimates of $\mathcal{C}_{X,Y}$ and $\mathcal{C}_Y$ into (4.15). Setting $\lambda \to 0$ as the numer of data points goes to infinity results in a consistent estimator of $\mathcal{W}_{X|Y}$. Alternatively, we can interpret (4.15) as the solution to the vector-valued regression problem given in (4.13) (Grünewälder et al., 2012).[1]

We now define the kernel equivalent of Bayes rule. Specifically, for two random variables $X$ and $Y$ and an event $z$, we define the equivalent statement of $\Pr(X \mid Y = y, z) = \Pr(X, Y = y \mid z)/\Pr(Y = y \mid z)$. In the context of Bayes filtering, the event $z$ is the history $o_{1:t-1}$ and the random variables $Y$ and $X$ are the observation $o_t$ and the shifted future $x_{t+1}$ respectively. Let $\mathcal{C}_{X,Y|z}$ denote the uncentered covariance of $X$ and $Y$ conditioned on the event $z$. It follows from (4.13) that

$$\mu_{X|Y=y,z} \approx \mathcal{W}^{\lambda}_{X|Y;z} \phi^{\mathcal{Y}}(y) \equiv \mathcal{C}_{X,Y|z}(\mathcal{C}_{Y|z} + \lambda I)^{-1} \phi^{\mathcal{Y}}(y). \quad \forall y \in \mathcal{I}_{\mathcal{Y}} \tag{4.16}$$

Following (Boots et al., 2013), we refer to (4.16) as the *kernel Bayes rule*(KBR).[2] Note that $\mathcal{W}_{X|Y;z}$ is a linear operator from $\mathcal{Y}$ to $\mathcal{X}$ whose value is determined by $z$. An example is a conditional probability table of $X$ and $Y$ where the event $z$ determines the values in the table. We summarize in Table 4.1 the correspondence between Hilbert space embedding quantities and their realization in the discrete case.

[1] More specifically, a space of vector valued functions $f : \mathcal{X} \mapsto \mathcal{Y}$ may be reproduced by an operator valued kernel $k : \mathcal{X} \times \mathcal{X} \mapsto \mathcal{L}_{\mathcal{Y} \to \mathcal{Y}}$, where $\mathcal{L}_{\mathcal{Y} \to \mathcal{Y}}$ is the set of linear operators from $\mathcal{Y}$ to $\mathcal{Y}$ (Micchelli and Pontil, 2005). The result in (4.15) is the solution to the regularized regression problem from $X \in \mathcal{X}$ to $Y \in \mathcal{Y}$ when we choose the kernel $k(x_1, x_2) = k_{\mathcal{X}}(x_1, x_2)I$, where $k_{\mathcal{X}}$ is a scalar valued kernel and $I$ is the identity map.

[2] The original form of kernel Bayes rule (Fukumizu et al., 2013) has a prior distribution on $X$ in place of the event $z$. In the Bayes filtering scenario, a history event $z$ implies a prior on the shifted future $X$.

| Kernel $k_{\mathcal{X}}$ | Delta kernel $k_\delta(x_1, x_2) = 1(x_1 = x_2)$ |
|---|---|
| Feature map $\phi^{\mathcal{X}}$ | Indicator $\phi_\delta(i) = e_i$ |
| Mean map $\mu_X$ | Probability vector |
| Covariance operator $\mathcal{C}_{XY}$ | Joint probability table |
| Conditional operator $\mathcal{W}_{X\|Y}$ | Conditional probability table |
| KBR $\mu_{X\|Y=y,z} = \mathcal{C}_{X,Y\|z}(\mathcal{C}_{Y\|z} + \lambda I)^{-1}\phi^{\mathcal{Y}}(y)$ | $\Pr(X \mid Y = y, z) = \frac{\Pr(X, Y=y\|z)}{\Pr(Y=y\|z)}$ |

Table 4.1: Correspondance between HSE embeddings in the general case (left) and the finite domain case with the delta kernel (right).

### 4.1.5 Finite Dimensional Approximation of Kenrel Features via Random Fourier Features

For a kernel such as the Gaussian RBF kernel, there is no explicit finite dimensional feature map. Instead, we implicitly operate in the infinite dimensional feature space through the *kernel trick*: we reformulate the learning and inference algorithms so that they do not require computing the feature space image $\phi^{\mathcal{X}}(x)$ of a point $x$ in the input domain but only require computing inner products in the form $\langle \phi^{\mathcal{X}}(x_1), \phi^{\mathcal{X}}(x_2) \rangle_{\mathcal{X}}$, which can be replaced by kernel evaluation $k_{\mathcal{X}}(x_1, x_2)$. A quantity in the feature space (e.g. a weight vector for linear regression or a mean map of a distribution) is typically represented as a weighted combination of the feature space images of training examples and we only store the combination weights as well as the (required) training data in the raw form.

Unfortunately, the use of the kernel trick does not scale to large training sets. In addition to the aforementioned storage requirements, most kernel-based method make use of the *Gram matrix*, which is the matrix of kernel evaluations of all pairs of training set examples. This matrix needs to be stored and inverted for training and some models, including HSE-PSRs require operations that are at least as expensive for inference. For this reason, it is of extreme benefit to devise an approximate feature map $\hat{\phi}^{\mathcal{X}}$ of fixed finite dimensionality such that $\hat{\phi}^{\mathcal{X}}(x_1)^\top \hat{\phi}^{\mathcal{X}}(x_2) \approx k_{\mathcal{X}}(x_1, x_2)$. Given this representation, we can avoid computational and storage costs that scale polynomially with the number of examples. Two prominent approximation techniques are the Nyström method (Williams and Seeger, 2000) and random Fourier features (Rahimi and Recht, 2008), the latter of which we describe below.

Random Fourier features (RFF) is a method for obtaining an approximate feature map for a positive definite translation invariant kernel $k$, such as the Gaussian kernel. Such a kernel can be expressed as $k(x_1, x_2) = h(x_1 - x_2)$ for some positive definite function $h$. RFFs are based on Bochner's theorem (Rudin, 2017), which states that a function $h$ on $\mathbb{R}^d$ is positive definite if and only if it is the Fourier transform of a positive measure. In other words, a positive function $h$ can be expressed as

$$\begin{aligned} h(x_1 - x_2) &= \int_{\mathbb{R}^d} p(\omega) e^{jw^\top (x_1 - x_2)} d\omega \\ &= \int_{\mathbb{R}^d} p(\omega) \phi_\omega(x_1) \overline{\phi_\omega(x_2)} d\omega, \end{aligned} \tag{4.17}$$

where

$$p(\omega) > 0 \quad \forall \omega \in \mathbb{R}^d,$$
$$\phi_\omega(x) \equiv e^{jw^\top x},$$

and $\overline{\phi_\omega(x)}$ is the complex conjugate of $\phi_\omega(x)$. With the appropriate scaling, we can interpret the Fourier transform $p(\omega)$ as a probability density function on $\omega$ and hence we can have an unbiased approximation of (4.17) as

$$h(x_1 - x_2) \approx \sum_{i=1}^{D} \left(\sqrt{\frac{C}{D}}\phi_{\omega_i}(x_1)\right) \overline{\left(\sqrt{\frac{C}{D}}\phi_{\omega_i}(x_2)\right)} \equiv \langle \hat{\phi}(x_1), \hat{\phi}(x_2) \rangle, \tag{4.18}$$

where $C \equiv 1/\int_{\mathbb{R}}^{d} p(\omega)d\omega$ is a normalization constant and $\omega_1, \ldots, \omega_D$ are samples from $p(\omega)$. For the RBF kernel with bandwidth $\sigma$, sampling from the normalized $p(\omega)$ is equivalent to sampling from $\mathcal{N}(\mathbf{0}, \sigma^{-2}I)$. Using the fact that the kernel function is real-valued, Rahimi and Recht (2008) suggest replacing the complex-valued feature map with the following

$$\hat{\phi}(x) \equiv \sqrt{\frac{2C}{D}}[\cos(\omega_1^\top x + b_1), \ldots \cos(\omega_D^\top x + b_D)]^\top, \tag{4.19}$$

where $b_i$ is sampled uniformly from $[0, 2\pi]$. Having an explicit feature map, we can approximate linear operations in the RKHS using their finite dimensional counterparts on the approximate feature map. Specifically, linear operators reduce to finite dimensional matrices on the RFF basis.

Rahimi and Recht (2008) have shown that the uniform approximation error of kernel evaluation using RFFs decreases with the rate $O_p(1/\sqrt{D})$. In the context of classification or regression, we would like to set $D$ in a way that maintains the typical $O(1/\sqrt{N})$ generalization error, where $N$ is the number of training examples. Rahimi and Recht (2009) have shown that, for a general loss function, this can be achieved with $D = O(N)$. This is a rather disappointing result since it implies that by using RFF, one must get worse generalization error or incur the same cost of using Gram matrices. More recently, however, Rudi and Rosasco (2017) have shown that a much smaller rate $D = O(\sqrt{N}\log(N))$ can be used in the kernel regression setting. In practice, it is typical to use a number of features in the order of thousands.

## 4.2 Hilbert Space Embedding of Predictive State Representation

A Hilbert space embedding of a predictive state representation (HSE-PSR) (Boots et al., 2013) is a predictive state model that uses Hilbert space embedding as a state representation and kernel Bayes rule as a conditioning method. We describe an uncontrolled version of the model described in (Boots et al., 2013) as a predictive state model and describe its learning algorithm as a two-stage regression algorithm. We defer the discussion of the controlled version to Chapter 6. This section

will focus on the RKHS formulation. The finite dimensional approximation will be described in Section 4.3.1.

Recall that a $\tau_f$-observable predictive state model (Chapter 3) is specified by

- A future feature function $\psi$ that is a sufficient statistic of $\Pr(o_{t:t+\tau_f-1} \mid o_{1:t-1})$.
- An extended future feature function $\xi$ that is a sufficient statistic of $\Pr(o_{t:t+\tau_f} \mid o_{1:t-1})$.
- A filtering function $f_{\text{filter}}$ such that $\mathbb{E}[\psi_{t+1} \mid o_{1:t}] = f_{\text{filter}}(\mathbb{E}[\xi_t \mid o_{1:t-1}], o_t)$.

An HSE-PSR makes use of three kernels $k_h$, $k_o$, $k_O$ that are defined over sequences of history observations, individual observations and sequences of future observations respectively. The future function $\psi$ to set to be the feature map $\phi^O$ of the future kernel $k_O$. This makes the belief state $q_t = \mathbb{E}[\psi_t \mid o_{1:t-1}]$ essentially the Hilbert space embedding of the distribution in $\Pr(o_{t:t+\tau_f-1} \mid o_{1:t-1})$ into the RKHS induced by the future kernel $k_O$.

The extended future features are defined as the tuple $\xi_t \equiv (\psi_{t+1} \otimes \phi^o_t, \phi^o_t \otimes \phi^o_t)$, where $\phi^o_t$ is the feature map of the observation kernel $k_o$. That means the extended belief state is a tuple of two covariance operators $p_t \equiv (\mathcal{C}_{\psi_{t+1},\phi^o_t|o_{1:t-1}}, \mathcal{C}_{\phi^o_t|o_{1:t-1}})$. With this choice of the extended state, we can use kernel Bayes rule defined in (4.16) to perform state update:

$$q_{t+1} \equiv \mathbb{E}[\psi^O_{t+1} \mid o_{1:t}] = \mathcal{C}_{\psi_{t+1},\phi^o_t|o_{1:t-1}}(\mathcal{C}_{\phi^o_t|o_{1:t-1}} + \lambda I)^{-1}\phi^o(o_t). \tag{4.20}$$

An HSE-PSR is thus parametrized by two maps, $W_{\text{ext}}$ and $W_{\text{oo}}$ that are linear in $q_t$ and predict $\mathcal{C}_{\psi_{t+1},\phi^o_t|o_{1:t-1}}$ and $\mathcal{C}_{\phi^o_t|o_{1:t-1}}$ respectively given $q_t$. Note that, with the finite dimensional features, $W_{\text{ext}}$ is a 3-mode tensor, with modes corresponding to future, observations and shifted future while $W_{\text{oo}}$ is a 3-mode tensor with one mode corresponding to future while the other two modes corresponding to the immediate observation. Figure 4.1 visualizes HSE-PSR parameters and state update.

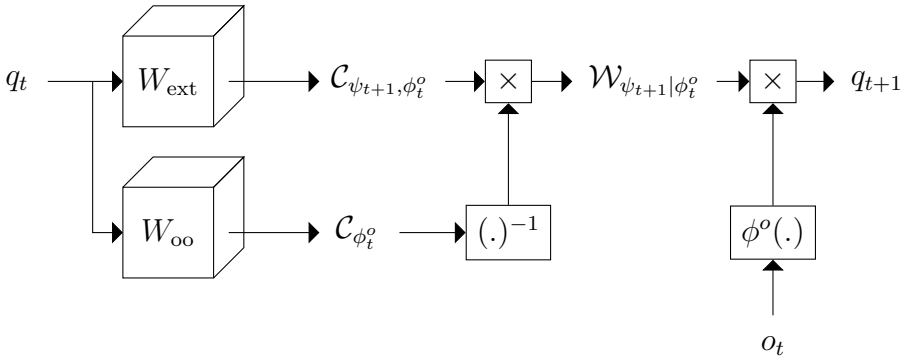



Figure 4.1: State update in an uncontrolled HSE-PSR. The diagram is for conceptual illustration. In practice, it is more efficient to first multiply the inverse observation covariance by the observation feature vector and then premultiply the result by $\mathcal{C}_{\psi_{t+1},\phi^o_t}$.

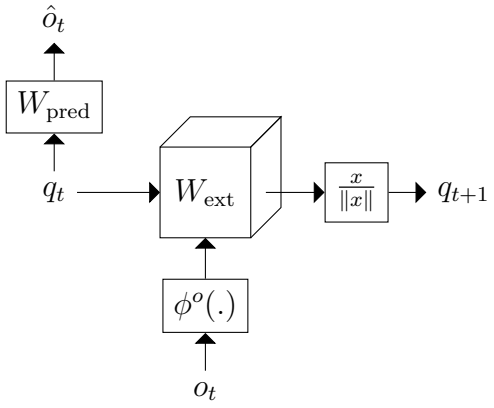



Figure 4.2: State update and prediction for PSRNN.

### 4.2.1 Learning Algorithm

The main parameters to learn in an HSE-PSR are $W_{\text{ext}}$ and $W_{\text{oo}}$. We can learn these parameters using two-stage regression. First, we use history features to construct examples $\bar{q}_t = \mathbb{E}[\psi_t \mid h_t]$ (stage 1A regression) and $\bar{\mathcal{C}}_t = \mathbb{E}[\psi_{t+1} \otimes \phi^o_t \mid h_t]$ (stage 1B regression). Given these examples we can use ridge regression to estimate $W_{\text{ext}}$. The same procedure can be used to estimate $W_{\text{oo}}$, with stage 1B regression producing estimates of $\mathbb{E}[\phi^o_t \otimes \phi^o_t \mid h_t]$. A natural choice for stage 1 regression is kernel ridge regression, which is effectively ridge regression with the feature map of the history kernel $\phi^h$ used as the history feature function.

### 4.2.2 Prediction

We described how to perform belief state updates in an HSE-PSR. Now we want to use these belief states to make predictions. Suppose we want to make a prediction of $\mathbb{E}[o_t \mid o_{1:t-1}]$. We can use supervised regression to learn a prediction function $g$. To learn this function, our training input/output pairs will be $(\hat{q}_t, o_t)$, where $\hat{q}_t$ is the estimated belief state at time $t$ after running the HSE-PSR as a recursive filter.

What is the suitable class of functions for $g$? Let $\mathcal{H}^O$ be the RKHS of the future kernel $k^O$. If we use a universal kernel such as the RBF kernel over a compact domain, then we know that, for any desired approximation precision, there exists a weight vector $w_y \in \mathcal{H}^O$ for any $y$ that is a scalar function of $o_{t:t+\tau_f-1}$ such that $y_t \approx \langle w_y, \psi_t \rangle$. Let $d$ be the dimensionality of $o_t$. We can train $d$ weight vectors each corresponding to a function that extracts a coordinate of $o_t$ from the future window $o_{t:t+\tau_f-1}$. Equivalently, we use linear regression to learn an operator $W_{\text{pred}} : \mathcal{H}^O \mapsto \mathbb{R}^d$ such that

$$o_t = W_{\text{pred}} \psi^O_t.$$

It follows from the definition of $q_t$ that

$$\mathbb{E}[o_t \mid o_{1:t-1}] = W_{\text{pred}} q_t.$$

## 4.3 Predictive State Recurrent Neural Networks

As mentioned in Section 4.1.5, we can use the kernel trick to develop an exact implementation of the HSE-PSR even for kernels with infinite dimensional feature maps such as the RBF kernel. For completeness, we provide such an implementation in the appendix. That implementation, however, requires $O(N^3)$ time and $O(N^2)$ space for both training and inference, which makes it impractical except for small datasets.

Moreover, the learning algorithm does not directly optimize for minimum prediction error and, being method of moments-based, it is not statistically efficient. We propose a recursive filter that is inspired by HSE-PSRs but overcomes these two difficulties. The proposed model, predictive state recurrent neural networks (PSRNNs) introduces three key modifications to HSE-PSRs that we describe below.

### 4.3.1 Kernel Approximation

We use random Fourier features as described in Section 4.1.5. With random Fourier features, feature maps are vectors, covariance operators are matrices and weight parameters are 3-mode tensors, which allows us to apply the HSE-PSR update rule 4.20 explicitly.

Random Fourier features, however, do not take the actual distribution of the data into account. Therefore, we can significantly reduce their dimensionality by projection. We project history, observation and future features on the top $p$ left singular vectors of the covariances $\mathcal{C}_{\phi_t^h,\phi_t^O}$, $\mathcal{C}_{\phi_t^o,\phi_t^h}$ or $\mathcal{C}_{\phi_t^O,\phi_t^h}$. This is based on the intuition from subspace identification that we care about the part of the future that is predictable from history.

### 4.3.2 Local Refinement By Discriminative Training

A common practice is to use the output of a moment-based algorithm to initialize a non-convex optimization algorithm such as EM (Falakmasir et al., 2013; Belanger and Kakade, 2015) or gradient descent Jiang et al. (2016). Since EM is not directly applicable to HSE-PSRs, we propose a gradient descent approach. We can observe that filtering in an HSE-PSR defines a recurrent structure given by

$$\begin{aligned} q_{t+1} &= f_{\text{filter}}(W_{\text{system}} q_t, o_t) \\ \mathbb{E}[o_t|q_t] &= W_{\text{pred}} q_t, \end{aligned}$$

With a differentiable $f_{\text{filter}}$ we can interpret an HSE-PSR as a recurrent neural network and apply discriminative training—that is, we minimize the error in predicting observations. Starting, from an initial value of $W_{\text{system}}$ and $W_{\text{pred}}$ that is obtained through method of moments, we can use backpropagation through time as a local optimization procedure to minimize the predictive loss

$$\sum_t l(o_t, W_{\text{pred}} \hat{q}_t), \tag{4.21}$$

where $l$ is a Bregman divergence loss (e.g. square loss) and $\hat{q}_t$ is the estimated belief state at time $t$.

### 4.3.3 Approximate Conditioning

We replace the state update in (4.20) with the following approximation (see Figure 4.2):

$$q_{t+1} = \frac{W_{\text{ext}} \times_q q_t \times_o \phi_t^o + b}{\|W_{\text{ext}} \times_q q_t \times_o \phi_t^o + b\|}, \tag{4.22}$$

where $W_{\text{ext}} \times_q q_t \times_o \phi_t^o$ is the vector $y$ given by

$$y[i] = \sum_{j,k} W_{\text{ext}}[j,k,i] \cdot q_t[j] \cdot \phi^o[k]$$

This approximation gets rid of the parameter $W_{\text{oo}}$. It also replaces the matrix inverse operation with the less costly and typically more stable $L_2$ normalization. This approximation is inspired by the state update in the discrete case using the delta kernel, which is similar to (4.22) but uses $L_1$ norm instead of $L_2$ norm. With RFF features, the $L_2$ makes sense as an approximation to the RKHS norm. It is also results in a differentiable operation, unlike $L_1$ norm. Also, normalizing the state of a Bayes filter using the $L_2$ norm bears similarity to norm observable operator models (Zhao and Jaeger, 2010). A very similar normalization scheme has also been investigated in (Ba et al., 2016), where it was suggested to speed up training of recurrent networks.

The learning algorithm of PSRNNs is similar to that of HSE-PSRs with additional steps for feature learning and local refinement. In summary, it goes as follows:

- Compute projection matrices for past and future features from the SVD of the relevant covariance matrices (Section 4.3.1).
- Learn the parameter $W_{\text{system}} \equiv W_{\text{ext}}$ using two-stage regression (Section 4.2.1) and estimate the initial state $q_1$ as the mean value of future features $\psi_t$.
- Apply the update equation (4.22) to compute $\hat{q}_t$ for all $t$.
- Using input/output examples $(\hat{q}_t, o_t)$, learn a prediction matrix $W_{\text{pred}}$.
- Use backpropagation through time through the computational graph in Figure 4.2 to update the parameters $W_{\text{ext}}$ and $W_{\text{pred}}$ to minimize prediction error.

## 4.4 Experiments

In this section we report results on experiments that compare PSRNNs to various baselines in predicting future observations.

### 4.4.1 Character-level Language Modeling

In this experiment, we demonstrate the use PSRNNs to model sequences of characters. For this purpose, we use a subset of the Penn Tree Bank dataset (Marcus et al., 1993), which is a standard benchmark in the NLP community. The observations are individual characters, which can take one out of 49 possible values. We use a train/test split of 120k characters each.

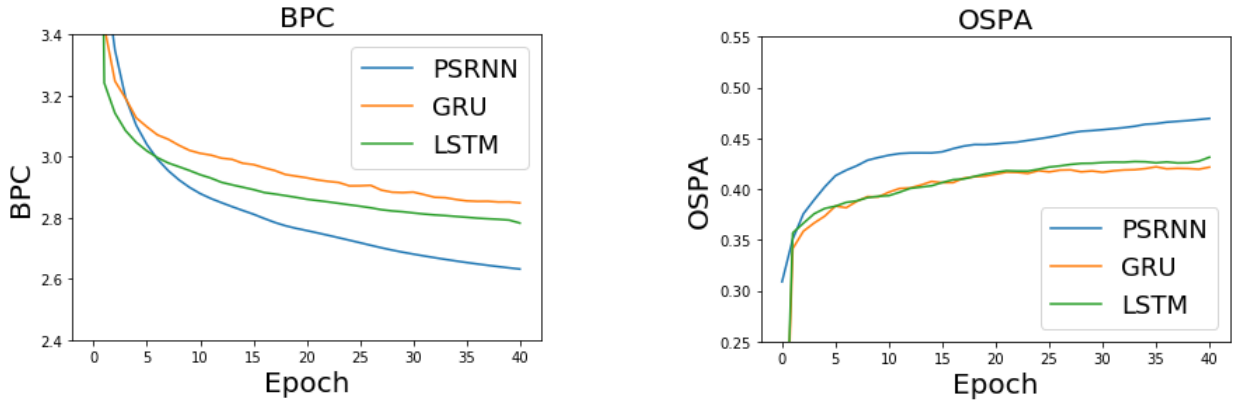



Figure 4.3: Bits per character (left) and one-step prediction accuracy (right) on Penn Tree Bank dataset.

We compare PSRNNs to LSTMs (Hochreiter and Schmidhuber, 1997) and GRUs (Cho et al., 2014). We use two-stage regression to initialize the PSRNN, with the regression method being ridge regression. We set the ridge parameter $\lambda = 10^{-2}N$, where $N$ is the total number of examples, and we set the projected feature dimension $p$ to 20. Since $o_t$ is a discrete character, we replace the linear regression step to compute $W_{\text{pred}}$ with multinomial logistic regression.

We train all models using backpropagation through time (BPTT) to minimize bits-per-character (BPC). For each training epoch, we report BPC and one-step prediction accuracy (OSPA) on the test set. The results are shown in Figure 4.3. The results show that PSRNNs outperform the commonly used baselines in both performance metrics.

### 4.4.2 Continuous Systems

We also experimented with the following continuous dynamical systems.

- **Swimmer:** We consider a 3-link simulated swimmer robot from the open-source package RLPy (Geramifard et al., 2013). The robot is controlled using a mixture of a pre-trained policy that is optimized for fast forward movement and a uniformly random policy where the random policy is executed 20% of the time. The observations consist of the angles of the joints and the position of the nose (in body coordinates). The measurements are contaminated with Gaussian noise whose standard deviation is 5% of the true signal standard deviation. We generate 25 trajectories of length 100 each and we split them into 20 training and 5 test trajectories.
- **Handwriting:** This is a digit handwriting database available on UCI repository (Alpaydin and Alimoglu, 1998). The data is created using a pressure sensitive tablet and a cordless stylus. Observations consist of $x$ and $y$ coordinates and pressure levels of the pen at a sampling rate of 100 milliseconds. We use 25 trajectories with a train/test split of 20/5.

We compare PSRNNs to basic RNNs (a.k.a Ellman networks) (Elman, 1990), LSTMs (Hochreiter and Schmidhuber, 1997), GRUs (Cho et al., 2014) and Kalman filters (Kalman, 1960). All

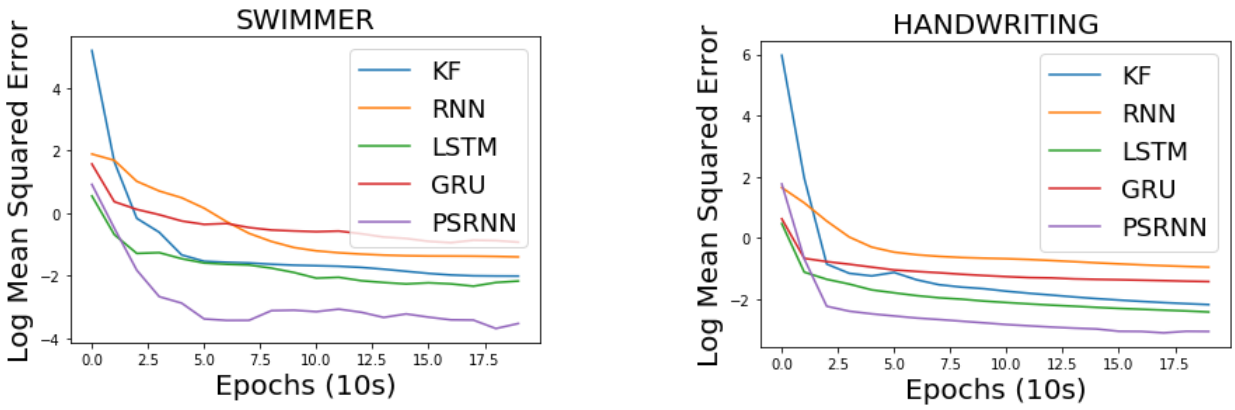



Figure 4.4: Log mean squared error on swimmer (left) and handwriting (right) datasets.

these models can be viewed as recurrent networks and can be optimized using backpropagation through time, where the optimization criterion is the mean square error (MSE). We used two-stage regression to initialize the Kalman filter and used the Xavier initialization scheme (Glorot and Bengio, 2010) to initialize the RNNs, LSTMs and GRUs.

Figure 4.4 depicts the log MSE after each epoch, we again see a clear advantage of PSRNNs over the other baselines.

To gain insight on the importance of good initialization, figure 4.5 shows examples of one-step predicted swimmer trajectories before and after BPTT. We see that the behavior of the initial model can have a large impact on the final model. For example, BPTT alone is not able to eradicate the initial oscillatory behavior of the RNN in the beginning of the trajectory.

## 4.5 Conclusion

In this chapter we introduced predictive state recurrent neural networks (PSRNNs), a practical non-parametric predictive sate model for modeling non-linear continuous systems. PSRNNs builds upon previous literature in using approximate kernel features as future statistics and augments that with discriminative training after two stage regression. The result is a novel recurrent network architecture that has a theoretically motivated initialization algorithm. We have demonstrated the superiority of this architecture to traditional recurrent architectures for modeling sequential data.

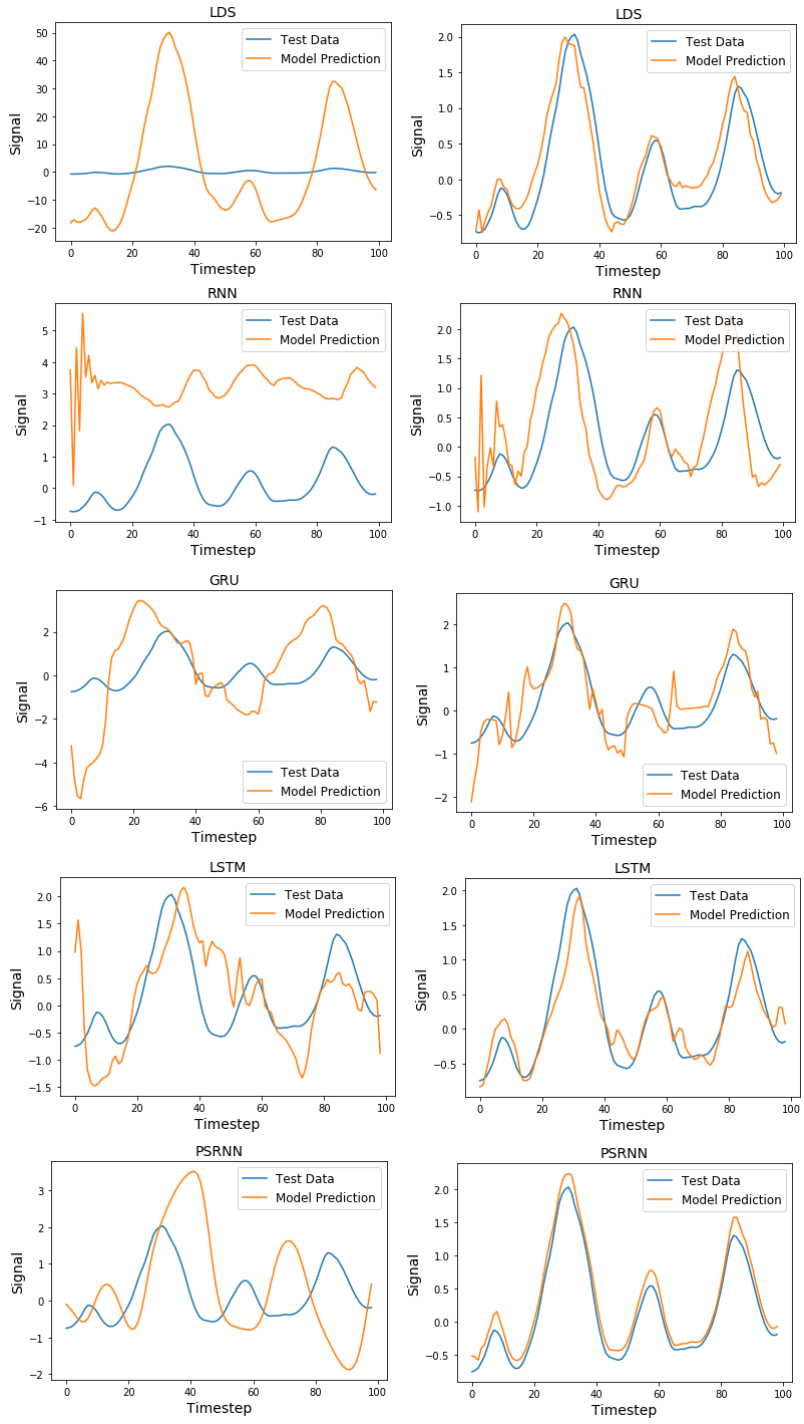



Figure 4.5: Test Data vs Model Prediction on a single feature of Swimmer. The left column shows initial performance. The right column shows performance after training. The order of the rows is KF, RNN, GRU, LSTM, and PSRNN.

# 4.A Appendix: Two-stage Regression of HSE-PSRs with Gram Matrices

We define a class of non-parametric two-stage instrumental regression models. By using conditional mean embedding (Song et al., 2009) as S1 regression model, we recover an uncontrolled variant of HSE-PSRs (Boots et al., 2013). Let $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{S}$ denote reproducing kernel Hilbert spaces with reproducing kernels $k_{\mathcal{X}}, k_{\mathcal{Y}}, k_{\mathcal{Z}}$ and $k_{\mathcal{S}}$ respectively. Assume $\psi_t \in \mathcal{X}$ and that $\xi_t \in \mathcal{Y}$ is defined as the tuple $(\phi_t^o \otimes \phi_t^o, \psi_{t+1} \otimes \phi_t^o)$. Let $\mathbf{\Psi} \in \mathcal{X} \otimes \mathbb{R}^N$, $\mathbf{\Xi} \in \mathcal{Y} \otimes \mathbb{R}^N$, $\mathbf{H} \in \mathcal{Z} \otimes \mathbb{R}^N$, $\mathbf{O} \in \mathcal{S} \otimes \mathbb{R}^N$ be operators that represent training data. Specifically, $\psi_s$, $\xi_s$, $h_s$, $\phi_s^o$ are the $s^{th}$ "columns" in $\mathbf{\Psi}$, $\mathbf{\Xi}$, $\mathbf{H}$ and $\mathbf{O}$ respectively. It is possible to implement S1 using a non-parametric regression method that takes the form of a linear smoother. In such case the training data for S2 regression take the form

$$\hat{\mathbb{E}}[\psi_t \mid h_t] = \sum_{s=1}^{N} \beta_{s|h_t} \psi_s$$
$$\hat{\mathbb{E}}[\xi_t \mid h_t] = \sum_{s=1}^{N} \gamma_{s|h_t} \xi_s,$$

where $\beta_s$ and $\gamma_s$ depend on $h_t$. This produces the following training operators for S2 regression:

$$\tilde{\mathbf{\Psi}} = \mathbf{\Psi}\mathbf{B}$$
$$\tilde{\mathbf{\Xi}} = \mathbf{\Xi}\mathbf{\Gamma},$$

where $\mathbf{B}_{st} = \beta_{s|h_t}$ and $\mathbf{\Gamma}_{st} = \gamma_{s|h_t}$. With this data, S2 regression uses a Gram matrix formulation to estimate the operator

$$W = \mathbf{\Xi}\mathbf{\Gamma}(\mathbf{B}^\top G_{\mathcal{X},\mathcal{X}} \mathbf{B} + \lambda I_N)^{-1} \mathbf{B}^\top \mathbf{\Psi}^* \tag{4.23}$$

Note that we can use an arbitrary method to estimate $\mathbf{B}$. Using conditional mean maps, the weight matrix $\mathbf{B}$ is computed using kernel ridge regression

$$\mathbf{B} = (G_{\mathcal{Z},\mathcal{Z}} + \lambda I_N)^{-1} G_{\mathcal{Z},\mathcal{Z}} \tag{4.24}$$

For each $t$, S1 regression will produce a denoised prediction $\hat{E}[\xi_t \mid h_t]$ as a linear combination of training feature maps

$$\hat{E}[\xi_t \mid h_t] = \mathbf{\Xi}\alpha_t = \sum_{s=1}^{N} \alpha_{t,s} \xi_s$$

This corresponds to the covariance operators

$$\hat{\mathcal{C}}_{\psi_{t+1},\phi_t^o|h_t} = \sum_{s=1}^{N} \alpha_{t,s}\psi_{s+1} \otimes \phi_s^o = \mathbf{\Psi}'\mathrm{diag}(\alpha_t)\mathbf{O}^*$$
$$\hat{\mathcal{C}}_{\phi_t^o|h_t} = \sum_{s=1}^{N} \alpha_{t,s}\phi_s^o \otimes \phi_s^o = \mathbf{O}\mathrm{diag}(\alpha_t)\mathbf{O}^*$$

Where, $\mathbf{\Psi}'$ is the shifted future training operator satisfying $\mathbf{\Psi}'e_t = \psi_{t+1}$ Given these two covariance operators, we can use kernel Bayes rule Fukumizu et al. (2013) to condition on $o_t$ which gives

$$q_{t+1} = \hat{E}[\psi_{t+1} \mid h_t] = \hat{\mathcal{C}}_{\psi_{t+1},\phi_t^o|h_t}(\hat{\mathcal{C}}_{\phi_t^o|h_t} + \lambda I)^{-1}\phi_t^o. \tag{4.25}$$

Replacing $\phi_t^o$ in (4.25) with its conditional expectation $\sum_{s=1}^{N} \alpha_s\phi_s^o$ corresponds to marginalizing over $o_t$ (i.e. prediction). A stable Gram matrix formulation for (4.25) is given by Fukumizu et al. (2013)

$$\begin{aligned} & q_{t+1} \\ & \quad = \mathbf{\Psi}'\mathrm{diag}(\alpha_t)G_{\mathcal{O},\mathcal{O}}((\mathrm{diag}(\alpha_t)G_{\mathcal{O},\mathcal{O}})^2 + \lambda NI)^{-1} \\ & \quad\quad .\mathrm{diag}(\alpha_t)\mathbf{O}^*o_{t+1} \\ & \quad = \mathbf{\Psi}'\tilde{\alpha}_{t+1}, \end{aligned} \tag{4.26}$$

which is the state update equation in HSE-PSR as suggested by Boots et al. (2013). Given $\tilde{\alpha}_{t+1}$ we perform S2 regression to estimate

$$\hat{P}_{t+1} = \hat{\mathbb{E}}[\xi_{t+1} \mid o_{1:t+1}] = \mathbf{\Xi}\alpha_{t+1} = W\mathbf{\Psi}'\tilde{\alpha}_{t+1},$$

where $W$ is defined in (4.23).

# Chapter 5

# Tensor Sketching for Predictive State Models with Large States

In Chapter 4 we introduced predictive state recurrent neural networks (PSRNNs) as a practical non-parametric predictive state model. Despite their many attractive properties, a major weakness of PSRNNs is that they do not scale well with large scale spaces. Training a PSRNN involves learning a 3-mode parameter tensor which requires $O(p^4)$ time and produces a model of size $O(p^3)$, where $p$ is the state size (the size of the projected RFF features). Inference involves a multilinear product that requires $O(p^3)$ time.

In this chapter we investigate the use of tensor sketching to facilitate the training of PSRNNs with large state sizes. Tensor sketching is a method to compress a tensor into a vector with a much less number of elements. It enables the approximate computation of dot products and multilinear products without the need to construct the full tensor. This makes it a potential candidate for approximating PSRNNs with large state spaces.

First, we demonstrate an interesting observation: that using tensor sketching for multilinear tensor-vector product results in poor approximation quality. Still, when used as a subroutine within tensor power iteration as suggested by Wang et al. (2015), it accurately manages to recover top rank-1 components. This observation is not predicted by the theoretical analysis in (Wang et al., 2015; Wang and Anandkumar, 2016) and we believe it to be of independent interest.

Second, we compare different approaches for non-orthogonal decomposition of asymmetric tensors via sketching and show that a deflation approach that recovers rank-1 components one by one outperforms the alternating least squares suggested in (Wang et al., 2015) and is able to compute a factorization of a low rank tensor with good accuracy.

These observations imply that tensor sketch is not by itself a good alternative representation of the PSRNN parameter tensor. However, they also suggest that we can use tensor sketching to obtain a low rank approximation of the parameter tensor, thus achieving our goal of reducing computational and space requirements.

Thus, our third contribution in this chapter is demonstrating the use of sketching to obtain a low rank representation of a PSRNN. We show that the performance of the model degrades gracefully with the amount of compression used.

The chapter is organized as follows, In section 5.1, we give the necessary background on tensor

operations and tensor sketching. In Section 5.2 we describe a two stage regression algorithm for PSRNN using sketching and describe how to use it to obtain a lightweight PSRNN model. In Section 5.3 we report the results of multiple experiments that verify the aforementioned contributions.

## 5.1 Tensors and Tensor Sketch

We introduce tensor inner products and tensor-vector products (a.k.a tensor contractions), the main operations we are interested in. Then, we describe how to approximate them with low cost using tensor sketch.

### 5.1.1 Tensor Inner Product and Tensor Contraction

A 3-mode tensor $A \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \times \mathbb{R}^{d_3}$ has $d_1 \times d_2 \times d_3$ entries. The inner (dot) product between two tensors $A$ and $B$ id defined as

$$\langle A, B \rangle \equiv \sum_{i=1}^{d_1} \sum_{j=1}^{d_2} \sum_{k=1}^{d_3} A_{i,j,k} B_{i,j,k}. \tag{5.1}$$

The inner product induces a Frobenius norm $\|A\|_F := \sqrt{\langle A, A \rangle}$. In our usage of tensors, we are interested in the tensor-vector product (also known as tensor contraction as it decreases the number of modes of the tensor). Using the notation in (Anandkumar et al., 2014a; Wang et al., 2015), we are interested in the following operations

$$T(I, b, c) \equiv \sum_{i=1}^{d_1} \sum_{j=1}^{d_2} \sum_{k=1}^{d_3} e_i^{d_1} b_j c_k \in \mathbb{R}^{d_1}, \tag{5.2}$$

$$T(a, b, c) \equiv \sum_{i=1}^{d_1} \sum_{j=1}^{d_2} \sum_{k=1}^{d_3} a_i b_j c_k \in \mathbb{R}, \tag{5.3}$$

where $e_i^{d_1}$ is an indicator vector in $\mathbb{R}^d$ where the $i^{th}$ coordinate is 1 and other coordinates are 0. The operations $T(a, I, c) \in \mathbb{R}^{d_2}$ and $T(I, b, c) \in \mathbb{R}^{d_3}$ are defined similarly. The contraction in (5.2) arises in the PSRNN state update. Tensor decomposition methods typically make use of (5.2) and (5.3). Tensor contraction operations can be expressed in terms of inner products as follows

$$T(I, b, c) \equiv \sum_{i=1}^{d_1} \langle T, e_i^{d_1} \otimes b \otimes c \rangle e_i^{d_1},$$
$$T(a, b, c) \equiv \langle T, a \otimes b \otimes c \rangle.$$

Thus, a method that approximates inner products can be used to approximate these operations.

### 5.1.2 Tensor Sketch

Before describing tensor sketch we describe count sketch for vectors (Cormode and Hadjieleftheriou, 2008). To hash a vector of dimensionality $d$, count sketch utilizes two independent hash functions $\mathfrak{h} : [d] \mapsto [\mathfrak{b}]$ and $\zeta : [d] \mapsto \{\pm 1\}$. A $d$ dimensional vector $x$ is hashed into a $\mathfrak{b}$ dimensional vector $s_x$ such that

$$s_x[i] = \sum_{j:\mathfrak{h}(j)=i} \zeta(j) x_j. \tag{5.4}$$

In words, the value of a sketch element is the sum of all colliding elements of the original vector multiplied by random signs. Thus, the sketch of a vector can be computed in $O(d)$ time. A key property of the count sketch that it approximately preserves inner products. More specifically, given two vectors $x$ and $y$ we have

$$\begin{aligned} \mathbb{E}[\langle s_x, s_y \rangle] &= \langle x, y \rangle \\ \mathrm{Var}[\langle s_x, s_y \rangle] &= O(|\langle x, y \rangle|^2 / \mathfrak{b}). \end{aligned} \tag{5.5}$$

In practice, we use multiple pair-wise independent hash functions and use the median of the estimated inner product to increase robustness. Preserving inner products also implies that we can approximately reconstruct elements of a sketched vector by taking an inner product with the corresponding indicator vector.

$$x_i \approx \langle s_x, s_{e_i} \rangle = \zeta(i) s_x[\mathfrak{h}(i)].$$

Tensor sketch (Pham and Pagh, 2013) is a generalization of count sketch. To hash a tensor into a vector of length $b$, we could use hash functions $\mathfrak{h} : [d_1] \times [d_2] \times [d_3] \mapsto [\mathfrak{b}]$ and $\zeta : [d_1] \times [d_2] \times [d_3] \mapsto \{\pm 1\}$. The main idea behind tensor sketch is that, given pairwise independent hash functions $\mathfrak{h}_m : [d_m] \mapsto [b]$ and $\zeta_m : [d_m] \mapsto \{\pm 1\}$ for $m \in \{1, 2, 3\}$, new independent hash functions $\mathfrak{h} : [d_1] \times [d_2] \times [d_3] \mapsto [\mathfrak{b}]$ and $\zeta : [d_1] \times [d_2] \times [d_3] \mapsto \{\pm 1\}$ can be constructed for tensors as follows

$$\begin{aligned} \mathfrak{h}(i, j, k) &= (\mathfrak{h}_1(i) + \mathfrak{h}_2(j) + \mathfrak{h}_3(k)) \mod \mathfrak{b} \\ \zeta(i, j, k) &= \zeta_1(i) \zeta_2(j) \zeta_3(k). \end{aligned}$$

The main utility of this construction is that is allows for efficient computation of sketched of rank-1 tensors. Let $T = a \otimes b \otimes c$ be a rank-1 tensor. It can be shown that that

$$\begin{aligned} s_T &= s_a^{(1)} * s_b^{(2)} * s_c^{(3)} \\ &= \mathcal{F}^{-1}(\mathcal{F}(s_a^{(1)}) \circ \mathcal{F}(s_b^{(2)}) \circ \mathcal{F}(s_c^{(3)})), \end{aligned} \tag{5.6}$$

where $s_x^{(i)}$ denotes the sketch of $x$ using the hash pair $(\mathfrak{h}_i, \zeta_i)$ and $\mathcal{F}$ denotes discrete Fourier transform[1]. This computation can be performed in $O(\mathfrak{b} \log \mathfrak{b})$ time. Since sketching is a linear

[1]There are multiple notions of Fourier transform that use different normalization schemes. To obtain the right results, one should use the non-normalized version which divides the sum by $\mathfrak{b}$, the length of the vector.

operation, we can use (5.6) to efficiently compute the sketch of a sum of rank-1 components given the corresponding vectors. This is beneficial for sketching empirical covariance tensors, which are our object of interest.

With the ability to sketch vectors and tensors, we can approximate tensor contractions

$$T(a,b,c) \approx \langle s_T, s_a^{(1)} * s_b^{(2)} * s_c^{(3)} \rangle \tag{5.7}$$

$$s_{T(I,b,c)}^{(1)}[i] \approx \langle s_T, s_{e_i}^{(1)} * s_b^{(2)} * s_c^{(3)} \rangle \tag{5.8}$$

Wang et al. (2015) showed that (5.8) implies the following

$$s_{T(I,b,c)}^{(1)} \approx \mathcal{F}^{-1}(\mathcal{F}(s_T) \circ \overline{\mathcal{F}(s_b^{(2)})} \circ \overline{\mathcal{F}(s_c^{(3)})}), \tag{5.9}$$

where the approximation is in the sense of (5.5) element-wise.

## 5.2 Tensor Sketching for PSRNNs

We first review a variant of PSRNNs that does not use an S1B regression step. Recall that this is equivalent to using ordinary least squares for S1B. The reason for ignoring S1B is that it allows us to formulate the training as the construction of a sum of rank-1 tensors.

We assume the existence of training data in the form of tuples $(h_t, \phi_t^o, \psi_t)$ of history, observation and future feature vectors for each time step $t \in [T]$. Assume these feature vectors to be of dimension $p$ each. The two-stage regression algorithm first uses ridge regression to compute a matrix $W_1$ such that

$$\hat{\psi}_t \equiv \hat{\mathbb{E}}[\psi_t \mid h_t] = W_1 h_t.$$

The algorithm then computes the parameter tensor $W_{\text{ext}}$ such that

$$\mathbb{E}[\phi_t^o \otimes \psi_{t+1} \mid \hat{\psi}_t] = W_{\text{ext}}(\hat{\psi}_t, I, I).$$

Using ridge regression, an estimate of $W_{\text{ext}}$ is given by

$$W_{\text{ext}} = M_3((M_2 + \lambda I)^{-1}, I, I), \tag{5.10}$$

where

$$M_3 \equiv \sum_{t=1}^{T} \hat{\psi}_t \otimes \phi_t^o \otimes \psi_{t+1} \tag{5.11}$$

$$M_2 \equiv \sum_{t=1}^{T} \hat{\psi}_t \otimes \hat{\psi}_t \tag{5.12}$$

The multiplication in (5.10) can be understood as reshaping $M_3$ into a matrix with $\dim(\hat{\psi}_t)$ rows and $\dim(\phi_t^o) \times \dim(\psi_{t+1})$ columns, pre-multiplying it by $(M_2 + \lambda I)^{-1}$, and reshaping the result

back as a tensor. Given $T$ training examples, computing $M_3$ via (5.11) requires $O(Tp^3)$ time, where $p$ is the dimensionality of the features. We also need $O(p^4)$ time to compute $W$ via (5.10). For inference, recall that the state update for a PSRNN is given by the following normalized tensor contraction

$$q_{t+1} = \frac{W_{\text{ext}}(q_t, \phi_t^o, I)}{\|W_{\text{ext}}(q_t, \phi_t^o, I)\|}. \tag{5.13}$$

Performing a single state update thus requires $O(p^3)$ computation to compute $W_{\text{ext}}(q_t, \phi_t^o, I)$. In addition to computation, we need $O(p^3)$ storage to store $M_3$ and $W_{\text{ext}}$. These requirements can be impractical when $p$ is large. Our goal is to be able to perform PSRNN training and inference without constructing the parameter tensor $W_{\text{ext}}$ or any quantity of similar size. Tensor sketching seems to be a strong candidate for this purpose. We now show how directly obtain the sketch of the parameter tensor $W_{\text{ext}}$ from training data. In sections 5.2.1 and 5.2.2 we discuss how to utilize this sketch.

From (5.11) and (5.10) we can rewrite $W_{\text{ext}}$ as follows

$$W_{\text{ext}} = \sum_t ((M_2 + \lambda I)^{-1} W_1 h_t) \otimes \phi_t^o \otimes \psi_{t+1}, \tag{5.14}$$

This means we can compute the sketch of $W_{\text{ext}}$ (which we denote by $s_W$) using three passes through the training data, as shown in Algorithm 2. For simplicity, the algorithm assumes a single sketch is sued but can easily be extended to $\mathfrak{B}$ sketches.

**Algorithm 2** Sketching the parameter tensor $W_{\text{ext}}$

**Input:**Training examples $\{(h_t, \phi_t^o, \psi_t)\}_{t=1}^T$, hash functions $(\{\mathfrak{h}_m, \zeta_m\}_{m=1}^3)$, S1 regularization $\lambda_1$, S2 regularization $\lambda_2$.

**Output:**Parameter tensor sketch $s_W \in R^{\mathfrak{b}}$.

// Stage 1 Regression
$W_1 \leftarrow (\sum_t \psi_t \otimes h_t)(\sum_t h_t \otimes h_t + \lambda_1 I)$
// Stage 2 Regression
$M_2 \leftarrow \sum_{t=1}^T (W_1 h_t) \otimes (W_1 h_t)$
$s_W \leftarrow \mathbf{0}$
**for** $t = 1$ to $T - 1$ **do**
    $a \leftarrow (M_2 + \lambda_2 I)^{-1} W_1 h_t, b \leftarrow \phi_t^o, c \leftarrow \psi_{t+1}$
    $s_W \leftarrow s_W + \mathcal{F}^{-1}(\mathcal{F}(s_a^{(1)}) \circ \mathcal{F}(s_b^{(2)}) \circ \mathcal{F}(s_c^{(3)}))$
**end for**

The following proposition shows the time and space complexities of Algorithm 2.

**Proposition 5.1.** *Assume that all feature vectors are of dimension $p$ and that we use $\mathfrak{B}$ sketches of size $\mathfrak{b}$ each. Then, for $T$ training examples, Algorithm 2 has a time complexity of $O(p^3 + T[p^2 + \mathfrak{B}p + \mathfrak{B}\mathfrak{b}\log\mathfrak{b}])$ and a space complexity of $O(p^2 + \mathfrak{B}\mathfrak{b})$.*

*Proof.* Computing $M_2$ requires $O(Tp^2)$ time. Computing $W_1$ as well as $(M_2 + \lambda_2 I)^{-1} W_1$ requires $O(p^3)$. Finally , to compute $s_W$ we repeat the following for each example $t$: Matrix-vector

product to compute $a$ [$O(p^2)$], computing the sketches [$O(\mathfrak{B}p)$], and performing convolution [$O(\mathfrak{B}\mathfrak{b}\log\mathfrak{b})$].

The space complexity is the sum of the sizes of $M_2$ and similar matrices [$O(p^2)$] and the sketches [$O(\mathfrak{B}\mathfrak{b})$]. □

Compared to the $O(p^4 + Tp^3)$ time and $O(p^3)$ memory when learning $W_{\text{ext}}$ directly, Algorithm 2 can result in significant gains for large values of $p$. We now discuss the use of the sketched PSRNN parameter tensor.

### 5.2.1 Tensor Sketch as a PSRNN Parameter

With the capability of efficiently computing $s_W$, there are multiple ways of utilizing it in inference. An obvious approach is to maintain the parameter tensor in its sketched form and replace the tensor contraction step in (5.13) by its approximation using (5.9) to approximate the tensor. This approach requires sketching the observation features at each time step which is a linear (and hence differentiable) operation.

However, our initial experiments have shown that this approach results in very poor results. As we demonstrate in Section 5.3.1, this can be attributed to the fact that, while tensor sketches provides a decent approximation when used for tensor decomposition, the approximation quality for general tensor contraction can be very poor. We instead propose factored PSRNNs, which use of sketching as a factorization technique, as we describe in the following subsection.

### 5.2.2 Factored PSRNNs

We propose using the tensor sketch to compute an approximate CP decomposition of $W_{\text{ext}}$. Let

$$W_{\text{ext}} \approx \sum_{i}^{m} a_i \otimes b_i \otimes c_i. \tag{5.15}$$

Let $A$, $B$ and $C$ be the matrix whose columns are $a_i$, $b_i$ and $c_i$ respectively. We can write $W(q_t, \phi_t^o, I)$ as

$$q_{t+1} = \frac{C^\top(Aq_t \circ B\phi_t^o)}{\|C^\top(Aq_t \circ B\phi_t^o)\|}. \tag{5.16}$$

We will refer to this representation as a *factored* PSRNN. In the following sections, we show how to use tensor sketching to obtain the decomposition in (5.15).

### 5.2.3 Hybrid ALS with Deflation

Wang et al. (2015) proposed a CP decomposition method for asymmetric tensors based on alternating least squares (Algorithm 3). The core operation in alternating least squares is the tensor product $T(I, b, c)$, which can be carried out with sketches using (5.9).

**Algorithm 3** Fast ALS using sketches (DECOMPALS)

**Input:**

- Hash functions $(h^{(i,j)}, \zeta^{(i,j)})$ for $i \in \{1,2,3\}$ and $j \in [\mathfrak{B}]$.
- Tensor sketch $s_T^{(j)}$ for $j \in [\mathfrak{B}]$ where $T \in \mathbb{R}^{d_1 \times d_2 \times d_3}$.
- Number of factors $K$.

**Output:**Factor weights $\{\lambda_k\}_{k=1}^K$ and unit vectors $\{(a_k \in \mathbb{R}^{d_1}, b_k \in \mathbb{R}^{d_2}, c_k \in \mathbb{R}^{d_3})\}_{k=1}^K$ such that $W \approx \sum_{k=1}^K \lambda_k a_k \otimes b_k \otimes c_k$.

$A, B, C \leftarrow$ random $(d_1, d_2, d_3) \times K$ matrices with normalized columns.
**for** $i = 1$ to $L$ **do**
  **for** $k = 1$ to $K$ **do**
    **for** $\beta = 1$ to $\mathfrak{B}$ **do**
      Compute $s_{b_k}^{(2,\beta)}$ and $s_{c_k}^{(3,\beta)}$
      $s_{a_k}^{(1,\beta)} \leftarrow s_{T(I,b_k,c_k)}^{(1,\beta)}$ (using (5.9))
    **end for**
    // reconstruct $a_k$:
    **for** $j = 1$ to $d_1$ **do**
      $a_{k,j} \leftarrow \text{median}(\mathfrak{Re}(\{s_{a_k}^{(1,\beta)}[h^{(1,\beta)}(j)]\zeta^{(1,\beta)}\}_{\beta=1}^{\mathfrak{B}})$
    **end for**
  **end for**
  $A \leftarrow A((C^\top C) \circ (B^\top B))^+$.
  $\lambda_k \leftarrow \|a_k\|$ for $k \in \{1, \ldots, K\}$
  Normalize the columns of $A$.
  Update $B$ and $C$ similarly.
**end for**

**Proposition 5.2.** *For $K$ components and $L$ iterations, Algorithm 3 has $O(LK\mathfrak{B}(p \log \mathfrak{B} + \mathfrak{b} \log \mathfrak{b}) + L(pK^2 + K^3))$ time complexity and $O(\mathfrak{b}\mathfrak{B} + Kp + K^2)$ space complexity.*

*Proof.* For each iteration and each component, we need to (1) sketch the factors $[O(\mathfrak{B}p)]$ (2) perform contraction $[O(\mathfrak{B}\mathfrak{b} \log \mathfrak{b})]$ and (3) perform reconstruction $[O(p\mathfrak{B} \log \mathfrak{B})]$. For each iteration we also need to compute $C^\top C \circ B^\top B^+$ $[O[(pK^2 + K^3)]]$ and update $A$ $[(pK^2)]$. Normalization and updating of $B$ and $C$ does not affect the asymptotic rate.

The space complexity arises from the need to store the sketches $[O(\mathfrak{B}\mathfrak{b})]$, rank-1 components $[O(Kp)]$ and matrices $A^\top A, B^\top B, C^\top C$ $[O(K^2)]$. □

By using sketching, Algorithm 3 scales linearly (instead of cubically) in the dimension $p$. Wang et al. (2015) demonstrated that Algorithm 3 is capable of recovering the top few components of a 1000 dimensional tensor.

However, we have observed that Algorithm 3 often has trouble when simultaneously considering a large number of components, as we demonstrate in Section 5.3.2. For this reason we opt to use a deflation based approach. We use Algorithm 3 to recover one component. Then we *deflate* the input tensor by subtracting that component and reiterate. Note that deflation can be performed

on sketched tensors. The process is detailed in Algorithm 4. Recovering a single component means the intermediate quantities $A^\top A$, $B^\top B$ and $C^\top C$ in Algorithm 3 are simply scalars. Not only does this approach produce better decomposition, as we demonstrate in Section 5.3.2 but it is also better in terms of time and space complexity as a function of $K$, as we show below.

**Proposition 5.3.** *For $K$ components and $L$ iterations per component, Algorithm 4 has $O(LK\mathfrak{B}(p \log \mathfrak{b} + \mathfrak{b} \log \mathfrak{b}) + KLp)$ time complexity and $O(\mathfrak{b}\mathfrak{B} + Kd)$ space complexity.*

*Proof.* The result is derived by substituting $K = 1$ in the time complexity of Algorithm 3 (Proposition 5.2) and multiplying the result $K$ times. Note that the deflation step needs $O(\mathfrak{B}\mathfrak{b} \log \mathfrak{b})$ time and thus does not change the asymptotic rate. □

Of course, it is possible to use a batched version of Algorithm 4, where in each iteration we extract $M$ components using ALS. However, we show in Section 5.3.2 that using $M = 1$ is more effective.

**Algorithm 4** Hybrid Decomposition with ALS and Deflation (DECOMPHYBRID)

**Input:**

- Hash functions $(\mathfrak{h}^{(i,j)}, \zeta^{(i,j)})$ for $i \in \{1, 2, 3\}$ and $j \in [\mathfrak{B}]$.
- Tensor sketch $s_T^{(j)}$ for $j \in [\mathfrak{B}]$ where $T \in \mathbb{R}^{d_1 \times d_2 \times d_3}$.
- Number of factors $K$, block size $M$.

**Output:**Factor weights $\{\lambda_k\}_{k=1}^K$ and unit vectors $\{(a_k \in \mathbb{R}^{d_1}, b_k \in \mathbb{R}^{d_2}, c_k \in \mathbb{R}^{d_3})\}_{k=1}^K$ such that $W \approx \sum_{k=1}^K \lambda_k a_k \otimes b_k \otimes c_k$.

**for** $k = 1$ to $K$ **do**
  $\lambda_k, a_k, b_k, c_k \leftarrow$ DECOMPALS$(s_T, K = 1)$
  **for** $j = 1$ to $\mathfrak{B}$ **do**
    $s_\Delta^{(j)} \leftarrow \lambda_i s_{a_i}^{(1,j)} * s_{b_i}^{(2,j)} * s_{c_i}^{(3,j)}$
    $s_T^{(j)} \leftarrow s_T^{(j)} - s_\Delta^{(j)}$
  **end for**
**end for**

### 5.2.4 Two-stage Regression for Factored PSRNN

Given the previous discussion, we propose the following two-stage regression algorithm for learning a factorized PSRNN:

- Estimate the initial state $q_1$ as the average future feature vector $\frac{1}{T}\sum_{t=1}^T \psi_t$.
- Estimate the parameter tensor sketch $s_W$, using Algorithm 2.
- Factorize the parameter tensor using Algorithm 4.
- Use the factorized PSRNN to compute states $q_{1:t}$ by applying (5.16).
- Solve a linear regression problem from $q_t$ to $o_t$ to estimate the prediction matrix $W_{\text{pred}}$.

## 5.3 Experiments

We now report the results of three experiments that are aimed to answer the following questions: (1) What is the approximation quality of tensor contraction in general and tensor decomposition in particular using sketching? (2) How does tensor decomposition with sketching using alternating least squares compare to deflation-based methods? and (3) What is the effect of using sketch two-stage regression on model performance?

### 5.3.1 Tensor Product vs. Tensor Decomposition

As mentioned in Section 5.1.2, tensor sketching allows us to approximate the tensor contraction by applying (5.9) to the sketches of the tensor and the vectors. Tensor contraction appears as a part of the state update in (5.13) as well as a core operation in tensor CP decomposition.

In this experiment, we compare using tensor sketching to approximate contraction within CP decomposition vs a general application scenario. To do so we conduct a number of trials, in each trial we generate a tensor $T = \sum_{i=1}^{50} \lambda_i u_i \otimes v_i \otimes w_i \in \mathbb{R}^{200\times200\times200}$, where $u_i, v_i$ and $w_i$ are sampled uniformly from unit sphere and $\lambda_i > 0$. We experimented with two settings of $\lambda$: an exponential decay $\lambda_i = \exp(-0.5(i-1))$ and a reciprocal decay $\lambda_i = i$. We used sketching with $\mathfrak{B} = 10$ and $\mathfrak{b} = 10000$ to approximate two operations: (1) generic tensor contraction, where we approximate $y = \frac{T(I,b,c)}{\|T(I,b,c)\|}$ for two random vectors $b$ and $c$ drawn from the unit sphere, and (2) recovering $u_1$ through the ALS method (Algorithm 3). We then computed the angle between the true and approximated vectors. We report the histogram of the cosine of the angle across trials.

The results are shown in Figure 5.1. The figure shows clearly that tensor contraction via sketching is robust when used within CP decomposition, even when its approximation quality is poor in general. We examined the cases where ALS failed to recover $u_1$ and we found that this "failure" is actually due to recovering a vector $u_j$ from a different rank-1 component. The results of this experiment provides justification to the use of tensor sketching as a means to obtain a factorized PSRNN as opposed to using it to represent the PSRNN, as described in Section 5.2.1.

### 5.3.2 Tensor Decomposition: Alternating Least Squares vs. Deflation

In this experiment we compare different methods for tensor decomposition. The methods we compare are ALS (Algorithm 3), Hybrid ALS with deflation (Algorithm 4) and a variant of Hybrid ALS that computes 5 factors instead of 1 in each iteration.

For each algorithm we conducted 30 trials. For each trial we generated a $200 \times 200 \times 200$ tensor that consists of 50 rank-1 components sampled using the same methodology in the previous experiment. We used the algorithm to recover different number of components and for each number we report the root relative mean square error which is computed as

$$\sqrt{\frac{\|\widehat{T} - T\|_F^2}{\|T\|^2}},$$

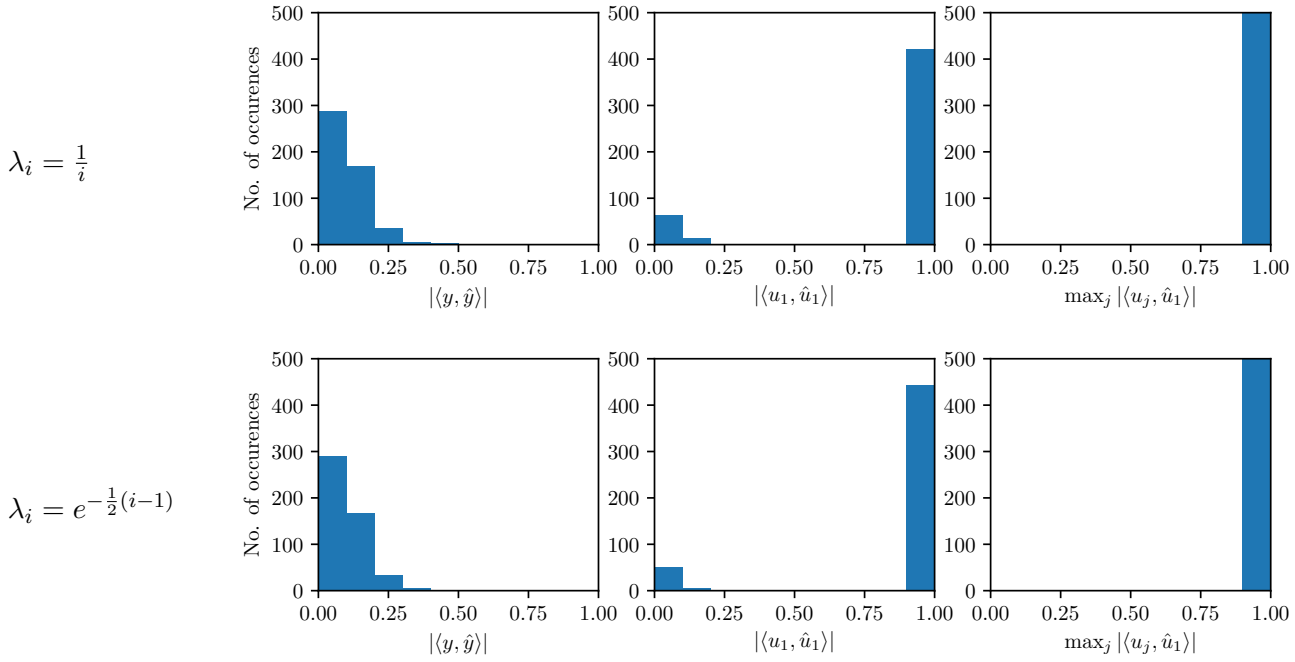



Figure 5.1: Approximation quality of general tensor contraction vs. recovering the first rank-1 component of a tensor. **(left)**: Histogram of dot product between normalized true and approximate contraction results. **(middle)**: Histogram of dot product between true and approximate first rank-1 component vector. **(right)**: Histogram of maximum dot product between approximate first rank-1 component vector and all true rank-1 components, showing that failures in (middle) are due to recovering a different rank-1 component.

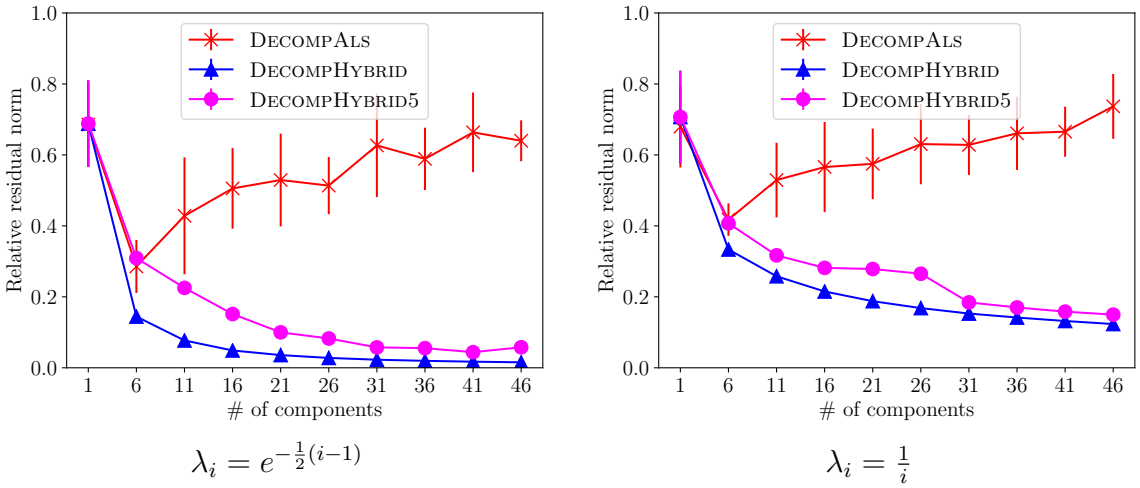



Figure 5.2: Relative residual norm for different decomposition methods using tensor sketches.

where $T$ is the true tensor and $\widehat{T}$ is the reconstructed tensor. Figure 5.2 shows the mean and standard deviation across trials. The figure shows that ALS is reliable only when the first few components are needed. Otherwise, a deflation approach is necessary. The results also show that recovering components one-by-one is better than in batches.

### 5.3.3 Factored PSRNNs with Sketching

In this experiment we investigate the use of tensor sketching to obtain a factored PSRNN without the need to incur the cost of constructing the parameter tensor.

We used the Penn Treebank Dataset (See Chapter 4) to train a PSRNN with a state of size 200 via two-stage regression. We compared that model to a factored model consisting of 60 factors that is trained using sketching as described in Section 5.2.4. The comparison criterion is the ability to predict the next character in the text. We set the number of sketches $\mathfrak{B}$ to 20 and vary the sketch size $\mathfrak{b}$.

Figure 5.3 shows the bits-per-character (BPC) and accuracy for the full model and the factorized model with different sketch sizes. We see that, with a sketch of size 10000, we can obtain a model that achieves almost the same performance as the full model. Note however, that the sketched representation of this model needs the same amount of memory as the full model. What is more important is that the performance degrades gracefully as we decrease the size of the sketch (we plot the performance of a random model as a reference for large degradation). For example, with 500 sketches, we achieve 1% compression ratio[2] at the expense of a 6 percentage point reduction in accuracy.

[2]The compression ratio assumes a reasonable implementation that, instead of maintaining the parameter tensor, maintains the 3 factor matrices $A$, $B$ and $C$ as well as the sketch of the parameter tensor and two other sketches of intermediate quantities.

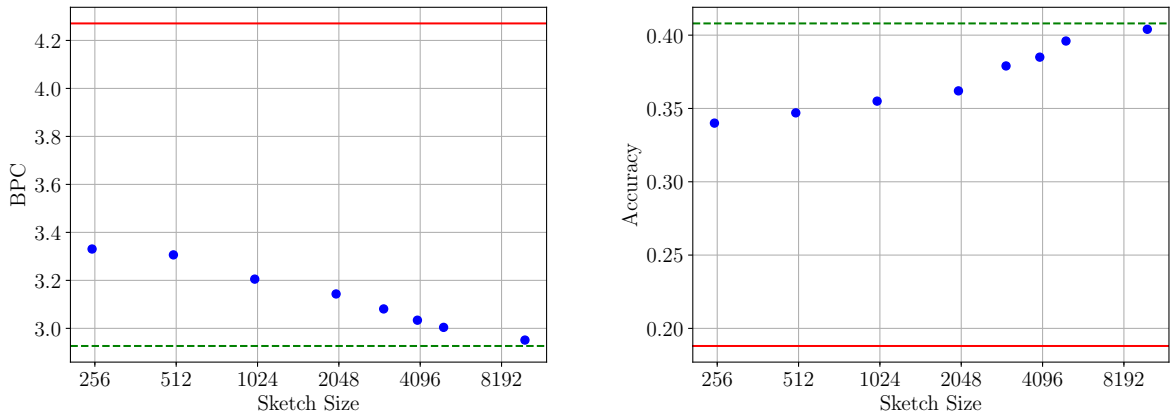



Figure 5.3: Bits-per-character and one step prediction accuracy for a factored PSRNN with 60 factors and a state of size 200 trained using 20 sketches of different sizes. The green dotted line shows the performance of the full (non-factored) model. The red solid line shows the performance of a random model as a reference value for large degradation in performance.

## 5.4 Conclusion

We investigated the use of sketching to enable the training of PSRNNs with large state sizes with low computational and memory costs. We have demonstrated the poor approximation quality attained by tensor sketching for general tensor contractions, suggesting that it is not applicable as-is for approximate inference in PSRNNs. Instead, we proposed the use of sketching to obtained factored PSRNNs with low-rank parameter tensors. We have demonstrated that deflation-based decomposition outperforms alternating least squares and that the resulting scheme results in an alternative two-stage regression approach that can save significant memory while causing graceful degradation of performance.

One of the motivations that evoked our interest in sketching was its potential use to compactly represent tensors of more than 3 modes, which can emerge in controlled systems (see Chapters 6 and 8). However, the results in this chapter suggest that further work is needed to harness the power of sketching outside the context of CP decomposition.

# Part III

# Learning Controlled Systems

# Chapter 6

# Predictive State Controlled Models

In this Chapter we extend our formulation of predictive state models to controlled dynamical systems, where a controller (a.k.a an agent) can affect the system (a.k.a the environment) through *actions*. We focus in this chapter on learning to predict. In more details, we consider the setting where the learning algorithm is given a set of observation/action trajectories produced by an external controller interacting with the system. The task is to construct a recursive filter that is capable of predicting future observations conditioned on future actions, even when these future actions are not generated by the same controller used for training.

The Chapter is organized as follows: In Section 6.1 we briefly revisit important concepts and terminology of Chapter 2 in the context of controlled dynamical systems. In Sections 6.2 and 6.3 we define the class of predictive state controlled models (PSCMs) and describe the associated two-stage regression learning algorithm. In Section 6.4 we construct a practical PSCM using techniques similar to those used in Chapter 4. We refer to the proposed model as a predictive state representation with random Fourier features (RFF-PSR). In Section 6.5 we experimentally demonstrate the efficacy of the proposed model. In Sections 6.6 and 6.7 we move back to a more general discussion of PSCMs, showing how they relate to other controlled systems in the literature and providing a theoretical analysis of the learning algorithm.

## 6.1 Recursive Filters for Controlled Dynamical Systems

Before our discussion of controlled dynamical systems we first address the concept of causal conditioning, or conditioning on *intervention*, which emerges due to the presence of actions. Then, we describe controlled dynamical systems utilizing this concept.

### 6.1.1 Causal Conditioning and The do Notation

Consider the graphical model in Figure 6.1 (left), which depicts a latent state dynamical system.[1] The latent system state $s_{t+1}$ depends on $s_t$ but also on an action $a_t$. A reactive policy chooses the

[1] Throughout this thesis, we follow the convention that $a_t$ is the action that precedes $o_t$ and that the interaction between the controller and the system starts with controller executing an action.

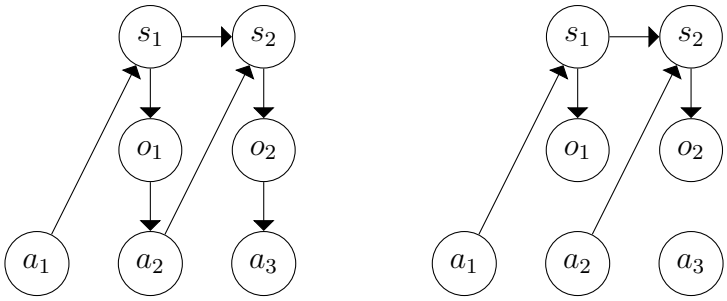



Figure 6.1: **left**: Graphical model of a controlled dynamical system with a reactive policy. **right**: Reduced model for causal conditioning on the actions.

action $a_t$ depending on observation $o_t$. Assume for simplicity that all variables are binary. One might like to characterize the initial behavior of the system at the first two timesteps using the conditional probability $\Pr(o_{1:2} \mid a_{1:2})$. However, this distribution depends on $\Pr(a_t \mid o_t)$ which is a property of the policy not the underlying system. For example, if the policy is simply that $a_{t+1} = o_t$ then we can conclude that $\Pr(o_1 \mid a_{1:2}) = 1(o_1 = a_2)$, which is a non-causal statement and certainly does not hold for a different policy.

To make a policy-independent statement, we disregard the influence of observations on actions (effectively using the model in Figure 6.1 (right)). Thus, we do not condition on *observing* $a_{1:2}$ as generated by some policy but rather on *intervening* by forcing the values of $a_{1:2}$ regardless of other variables. This is denoted by $\Pr(o_{1:2} \mid \mathbf{do}(a_{1:2}))$. From the reduced graph, we can see that

$$\Pr(o_{1:2} \mid \mathbf{do}(a_{1:2})) = \sum_{s_1,s_2} \Pr(s_1 \mid a_t)\Pr(o_1 \mid s_1)\Pr(s_2 \mid s_1, a_2)\Pr(o_2 \mid s_2), \tag{6.1}$$

where all factors depend only on the system dynamics. It is worth noting that $\Pr(o_{1:2} \mid a_{1:2})$ and $\Pr(o_{1:2} \mid \mathbf{do}(a_{1:2}))$ are identical if the policy is *blind* (a.k.a *open-loop*)—that is, the action $a_t$ is independent of previous observations (although it can depend on previous actions). It is also worth noting that $\Pr(o_{t:t+k} \mid o_{1:t-1}, \mathbf{do}(a_{1:t+k}))$ is identical to $\Pr(o_{t:t+k} \mid o_{1:t-1}, a_{1:t-1}, \mathbf{do}(a_{t:t+k}))$ unless the controller has additional access to the system beyond the observations. This means that policy blindness matters only when considering future actions. For brevity, we will denote by $h_t^\infty$ the entire history of observations and actions before time $t$ and write the aforementioned distribution as $\Pr(o_{t:t+k} \mid \mathbf{do}(a_{t:t+k}), h_t^\infty)$.

### 6.1.2 Controlled Dynamical Systems

A controlled dynamical system with observation set $\mathcal{O}$ and action set $\mathcal{A}$ models the probability distribution $\Pr(o_{1:t} \mid \mathbf{do}(a_{1:t}))$ for any $o_{1:t} \in \mathcal{O}^t$ and $a_{1:t} \in \mathcal{A}^t$ where $t \geq 1$. Similar to uncontrolled systems, the three different views of dynamical systems discussed in Chapter 2 (likelihood evaluation, system state and belief state) are applicable to controlled systems. Likelihood evaluation view exists in the form of input-output OOM (IO-OOM) (Jaeger, 1998), where an observable operator exists for each observation/action pair.

Figure 6.1 (right) is an example of a system state model. We can also allow the action $a_t$ to have a direct effect on the observation $o_t$, resulting in the following generative process

$$s_t \sim f(s_{t-1}, a_t)$$
$$o_t \sim g(s_t, a_t)$$

Examples of system state models include input-output hidden Markov models (IO-HMMs) (Bengio and Frasconi, 1995) and linear dynamical systems with Gaussian noise (van Overschee and de Moor, 1996). Note that we can remove the direct dependency of $o_t$ on $a_t$ by augmenting the state to include the action.

A belief state model, which can be used to simulate the system or as a recursive filter, applies the update equation

$$q_{t+1} = f(q_t, o_t, a_t)$$
$$o_t \sim g(q_t, a_t)$$

Thus, the belief state $q_t$ is a deterministic function of the history of observations and actions $h_t^\infty$. It constitutes a sufficient state representation if it sufficiently summarizes the history such that

$$\Pr(o_{t:\infty} \mid h_t^\infty, \mathbf{do}(a_{t:\infty})) = \Pr(o_{t:\infty} \mid q_t, \mathbf{do}(a_{t:\infty})).$$

The probability distribution $\Pr(s_t \mid h_t^\infty)$ where $s_t$ is the system state is a sufficient state representation. A system is $k$-observable if the function

$$q(o_{t:t+k-1}, a_{t:t+k-1}) \equiv \Pr(o_{t:t+k-1} \mid \mathbf{do}(a_{t:t+k-1}), h_t^\infty)$$

is a sufficient state representation.

### 6.1.3 Predictive States for Controlled Systems

For uncontrolled systems, we presented the notion of a predictive belief state $q_t = \mathbb{E}[\psi^O \mid o_{1:t-1}]$, where $\psi^O$ denotes future observation features. This representation allowed for the development of a consistent learning algorithm based on two stage regression (Chapter 3). To enable a similar methodology for controlled systems, we need a notion of a predictive state for controlled systems that encodes a conditional distribution of future observations given $\mathbf{do}$(future actions). For simplicity, we focus our discussion on $k$-observable systems. Given future action features $\psi^A$ and future observation features $\psi^O$, we define the predictive state at time $t$ to be a linear operator $Q_t$ such that $\mathbb{E}[\psi_t^O \mid \mathbf{do}(a_{t:t+k-1}), h_t^\infty] = Q_t \psi_t^A$. $\psi_t^A$ is chosen such that adding more features does not change $Q_t \psi_t^A$. $\psi_t^O$ is chosen such that it is a sufficient statistic for the probability distribution $\Pr(o_{t:t+k-1} \mid \mathbf{do}(a_{t:t+k-1}), h_t^\infty)$ for any values of $h_t^\infty$ and $a_{t:t+k-1}$. One choice of $\psi^O$ that imposes minimal assumptions is the feature function of a universal kernel. The simplest example is when we use the delta kernel on observation and action sequences of length $k$. In this case, $Q_t$ is simply a conditional probability table.

It is worth noting that there are other possible representations of the predictive state. Predictive state representations (PSRs) proposed by (Singh et al., 2004) represent the belief state as a vector

of probabilities. Each entry indicates the success probability of a *test*: the probability of observing a particular observation sequence given that we intervene with a particular action sequence. The representation we are using is equivalent to a PSR where we have a test for each sequence of observations and actions of length $k$. Since we are mainly interested in continuous systems, we do not need to actually enumerate all tests. We only need to choose features that result in an expressive model, which we will demonstrate in Section 6.4.

Having established our notion of predictive states, we are ready to define the model.

## 6.2 Model Definition

Similar to the predictive state model defined in Chapter 3, we denote by $\psi_t^O$, $\psi_t^A$, $\xi_t^O$ and $\xi_t^A$ sufficient features of future observations $o_{t:t+k-1}$, future actions $a_{t:t+k-1}$, extended future observations $o_{t:t+k}$ and extended future actions $a_{t:t+k}$ at time $t$ respectively.

We also use $h_t \equiv h(o_{1:t-1}, a_{1:t-1})$ to denote finite features of previous observations and actions before time $t$.[2]

**Definition 6.1.** *A dynamical system is said to conform to a **predictive state controlled model (PSCM)** if it satisfies the following properties:*

- *For each time $t$, there exists a linear operator $Q_t \equiv \mathcal{W}_{\psi_t^O|\mathbf{do}(\psi_t^A);h_t^\infty}$ (referred to as a predictive state) such that $\mathbb{E}[\psi_t^O \mid \mathbf{do}(a_{t:t+k-1}), h_t^\infty] = Q_t \psi_t^A$*
- *For each time $t$, there exists a linear operator $P_t \equiv \mathcal{W}_{\xi_t^O|\mathbf{do}(\xi_t^A);h_t^\infty}$ (referred to as an extended state) such that $\mathbb{E}[\xi_t^O \mid \mathbf{do}(a_{t:t+k}), h_t^\infty] = P_t \xi_t^A$*
- *There exists a linear map $W_{\text{system}}$ (referred to as the system parameter map [3] ), such that, for each time $t$,*

$$P_t = W_{\text{system}}(Q_t) \tag{6.2}$$

- *There exists a filtering function $f_{\text{filter}}$ such that, for each time $t$, $Q_{t+1} = f_{\text{filter}}(P_t, o_t, a_t)$. $f_{\text{filter}}$ is typically non-linear but known in advance.*

It follows that a PSCM is specified by the tuple $(Q_1, W_{\text{system}}, f_{\text{filter}})$, where $Q_1$ denotes the initial belief state.

There are a number of aspects of PSCMs that warrant re-emphasizing. First, unlike latent state models, the state $Q_t$ is represented by a conditional distribution of observed quantities. Second, $Q_t$ is a deterministic function of the history $h_t^\infty$. It represents the *belief* state that one should maintain after observing the history. Third, a PSCM specifies a recursive filter where given an action $a_t$ and an observation $o_t$, the state update equation is given by

$$Q_{t+1} = f_{\text{filter}}(W_{\text{system}}(Q_t), o_t, a_t) \tag{6.3}$$

[2]Often but not always, $h_t$ is a computed from fixed-size window of previous observations and actions ending at $t-1$.

[3]We note that the expression $W_{\text{system}}(Q_t)$ should *not* be understood as a matrix-matrix product (or a composition of linear operators in general). Rather, we think of $W_{\text{system}}$ as a matrix that acts on $\text{vec}(Q_t)$ to produce $\text{vec}(P_t)$.

This construction allows us to have a linear map $W_{\text{system}}$ and still use it to build models with non-linear state updates, including IO-HMMs (Bengio and Frasconi, 1995), Kalman filters with inputs (van Overschee and de Moor, 1996) and HSE-PSRs (Boots et al., 2013). As we will see in Section 6.3, avoiding latent variables and having a linear $W_{\text{system}}$ enable the formulation of a consistent learning algorithm.

## 6.3 Learning A Predictive State Controlled Model

We assume that the extended features $\xi_t^O$ and $\xi_t^A$ are chosen such that $f_{\text{filter}}$ is known. The parameters to learn are thus $W_{\text{system}}$ and $Q_0$. We also assume that a fixed blind policy is used to collect training data, and so we can treat causal conditioning on action $\mathbf{do}(a_t)$ as ordinary conditioning on $a_t$.[4] It is possible, however, that a different (possibly non-blind) policy is used at test time.

To learn model parameters, we will adapt the two-stage regression that we developed in Chapter 3. Let $\bar{Q}_t \equiv \mathbb{E}[Q_t \mid h_t]$ (resp. $\bar{P}_t \equiv \mathbb{E}[P_t \mid h_t]$) be the expected belief state (resp. expected extended state) conditioned on finite history features $h_t$ given our data collection policy. For brevity, we might refer to $\bar{Q}_t$ simply as the (predictive) state when the distinction from $Q_t$ is clear. It follows from linearity of expectation that

$$\mathbb{E}[\psi_t^O \mid \psi_t^A, h_t] = \bar{Q}_t \psi_t^A,$$
$$\mathbb{E}[\xi_t^O \mid \xi_t^A, h_t] = \bar{P}_t \xi_t^A,$$

and it follows from the linearity of $W_{\text{system}}$ that

$$\bar{P}_t = W_{\text{system}}(\bar{Q}_t)$$

So, we train regression models (referred to S1 regression models) to estimate $\bar{Q}_t$ and $\bar{P}_t$ from $h_t$. Then, we train another (S2) regression model to estimate $W_{\text{system}}$ from $\bar{Q}_t$ and $\bar{P}_t$. Being conditional distributions, estimating $\bar{Q}_t$ and $\bar{P}_t$ from $h_t$ is more subtle compared to uncontrolled systems, since we cannot use observation features as unbiased estimates of the state. We describe two methods to construct an S1 regression model to estimate $\bar{Q}_t$. The same methods apply to $\bar{P}_t$. As we show in Section 6.6, instances of both methods exist in the literature of system identification.

### 6.3.1 Joint S1 Approach

Let $\psi_t^{oa}$ denote a sufficient statistic of the joint observation/action distribution $\Pr(\psi_t^O, \psi_t^A \mid h_t)$. This distribution is fixed for each value of $h_t$ since we assume a fixed model and policy. We use an S1 regression model to learn the map $f : h_t \mapsto \mathbb{E}[\psi_t^{ao} \mid h]$ by solving the optimization problem

$$\arg\min_{f \in \mathcal{F}} \sum_{t=1}^{T} l(f(h_t), \psi_t^{oa}) + R(f)$$

[4]One way to deal with non-blind training policies is to assign importance weights to training examples to correct the bias resulting from non-blindness (Bowling et al., 2006; Boots et al., 2011). This, however, requires knowledge of the data collection policy and can result in a high variance of the estimated parameters. We defer the case of an unknown non-blind policy to future work.

for some suitable Bregman divergence loss $l$ (e.g., square loss) and regularization $R$.

Once we learn $f$, we can estimate $\bar{Q}_t$ by first estimating the joint distribution $\Pr(\psi_t^O, \psi_t^A \mid h_t)$ and then deriving the conditional operator $\bar{Q}_t$. By the continuous mapping theorem, a consistent estimator of $f$ results in a consistent estimator of $\bar{Q}_t$. An example of applying this method is using kernel Bayes rule (Fukumizu et al., 2013) to estimate states in HSE-PSR (Boots et al., 2013).

### 6.3.2 Conditional S1 Approach

In this method, instead of estimating the joint distribution represented by $\mathbb{E}[\psi_t^{OA} \mid h_t]$, we directly estimate the conditional distribution $\bar{Q}_t$. We exploit the fact that each training example $\psi_t^O$ is an unbiased estimate of $\bar{Q}_t\psi_t^A = \mathbb{E}[\psi_t^O \mid \psi_t^A, h_t]$. We can formulate the S1 regression problem as learning a function $f : h_t \mapsto \bar{Q}_t$ that best matches the training examples, i.e., we solve the problem

$$\arg\min_{f\in\mathcal{F}} \sum_{t=1}^{T} l(f(h_t)\psi_t^A, \psi_t^O) + R(f) \tag{6.4}$$

for some suitable Bregman divergence loss $l$ (e.g., square loss) and regularization $R$. An example of applying this method is the oblique projection method used in identification of linear dynamical systems (van Overschee and de Moor, 1996), which we describe in Section 6.6.2.

It is worth emphasizing that both the joint and conditional S1 approaches assume the state to be a *conditional* distribution. They only differ in the way to estimate that distribution.

### 6.3.3 S2 Regression and Learning Algorithm

Given S1 regression models to estimate $\bar{Q}_t$ and $\bar{P}_t$, learning a controlled dynamical system proceeds as shown in Algorithm 5.

**Algorithm 5** Two-stage regression for predictive state controlled models

**Input:**$h_{n,t}$,$\psi_{n,t}^O$, $\psi_{n,t}^A$, $\xi_{n,t}^O$, $\xi_{n,t}^A$ for $1 \le n \le N$, $1 \le t \le T_n$ ($N$ is the number of trajectories, $T_n$ is the length of $n^{th}$ trajectory).

**Output:**Dynamics matrix $\hat{W}_{\text{system}}$ and initial state $\hat{Q}_1$.

1: Use S1A regression to estimate $\bar{Q}_{n,t}$.
2: Use S1B regression to estimate $\bar{P}_{n,t}$.
3: Let $\hat{W}_{\text{system}}$ be the (regularized) least squares solution to the system of equations

$$\bar{P}_{n,t} \approx W_{\text{system}}(\bar{Q}_{n,t}) \quad \forall n, t$$

4: Set $\hat{Q}_1$ to the average of $\bar{Q}_{n,t}$

## 6.4 Predictive State Controlled Models With Random Fourier Features (RFF-PSR)

Having a general framework for learning controlled dynamical systems, we now focus on HSE-PSRs (Boots et al., 2013) as a non-parametric instance of that framework using Hilbert space embedding of distributions. We first describe learning of HSE-PSRs as a two-stage regression method. Then we demonstrate how to obtain a finite dimensional approximation using random Fourier features (RFF) (Rahimi and Recht, 2008). We refer the reader to Chapter 4 for a background on Hilbert space embedding and random Fourier features.

### 6.4.1 The HSE-PSR a predictive state controlled model

HSE-PSRs are a generalization of IO-HMMs that has proven to be successful in practice (Boots et al., 2013; Boots and Fox, 2013). They are suitable for high dimensional and continuous observations and/or actions. HSE-PSRs use kernel feature maps as sufficient statistics of observations and actions. We define four kernels $k_O, k_A, k_o, k_a$ over future observation features, future action features, individual observations and individual actions respectively; and we will use $\phi^x$ to denote the feature map of $k_x$.

We can then define $\psi_t^O = \phi^O(o_{t:t+k-1})$ and similarly $\psi_t^A = \phi_t^O(a_{t:t+k-1})$. We will also use $\phi_t^o$ and $\phi_t^a$ as shorthands for $\phi_o(o_t)$ and $\phi_a(a_t)$. The extended future is then defined as $\xi_t^o = \psi_t^O \otimes \phi_t^o$ and $\xi_t^a = \psi_t^A \otimes \phi_t^a$

Under the assumption of a blind learning policy, the operators $Q_t$ and $P_t$ are defined to be

$$\begin{aligned} Q_t &= \mathcal{W}_{\psi_t^O|\psi_t^A} h_t^\infty \\ P_t &= (P_t^\xi, P_t^o) = (\mathcal{W}_{\psi_{t+1}^O \otimes \phi_t^o | \psi_{t+1}^A \otimes \phi_t^a} h_t^\infty, \mathcal{W}_{\phi_t^o \otimes \phi_t^o | \phi_t^a} h_t^\infty) \end{aligned} \tag{6.5}$$

Note that we can think of the operator $P_t^\xi$ as a 4-mode tensor, with modes corresponding to $\psi_{t+1}^O$, $\phi_t^o$, $\psi_{t+1}^A$ and $\phi_t^a$. Similarly, the operator $P_t^o$ is a 3-mode tensor, with modes corresponding to $\phi_t^o$, $\phi_t^o$ and $\phi_t^a$. Based on (6.5), $Q_t$ specifies the state of the system as a conditional distribution of future observations given future actions while $P_t$ is a tuple of two operators that allow us to condition on the pair $(a_t, o_t)$ to obtain $Q_{t+1}$ using kernel Bayes rule (Fukumizu et al., 2013). In more detail, filtering in an HSE-PSR is carried out as follows

- From $o_t$ and $a_t$, obtain $\phi_t^o$ and $\phi_t^a$.
- Compute $C_{o_t o_t | h_t^\infty, a_t} = \mathcal{W}_{\phi_t^o \otimes \phi_t^o | \phi_t^a; h_t^\infty} \times_{\phi_t^a} \phi_t^a$
- Multiply by inverse observation covariance to change "predicting $\phi_t^o$" into "conditioning on $\phi_t^o$":

$$\begin{aligned} &\mathcal{W}_{\psi_{t+1}^O | \psi_{t+1}^A, \phi_t^o, \phi_t^a; h_t^\infty} \\ &\quad = \mathcal{W}_{\psi_{t+1}^O \otimes \phi_t^o | \psi_{t+1}^A, \phi_t^a; h_t^\infty} \times_{\phi_t^o} (C_{o_t o_t | h_t^\infty, a_t} + \lambda I)^{-1} \end{aligned}$$

- Condition on $\phi_t^o$ and $\phi_t^a$ to obtain shifted state

$$\begin{aligned} Q_{t+1} &\equiv \mathcal{W}_{\psi_{t+1}^O|\psi_{t+1}^A;\phi_t^o,\phi_t^a,h_t^\infty} \\ &= \mathcal{W}_{\psi_{t+1}^O|\psi_{t+1}^A,\phi_t^o,\phi_t^a;h_t^\infty} \times_{\phi_t^o} \phi_t^o \times_{\phi_t^a} \phi_t^a \end{aligned}$$

Therefore, in HSE-PSR, we need the parameter $W_{\text{system}}$ to be composed of two linear maps; $f_o$ and $f_\xi$ such that $P_t^\xi = f_\xi(Q_t)$ and $P_t^o = f_o(Q_t)$. In the following section we show how to estimate $\bar{Q}_t$ and $\bar{P}_t$ from data. Estimation of $f_\xi$, $f_o$ can then be carried out using kernel regression.

Learning and filtering in an HSE-PSR can be implicitly carried out in the RKHS using a Gram matrix formulation. We will describe learning in terms of the RKHS elements and refer the reader to (Boots et al., 2013) for details on the Gram matrix formulation. In Section 6.4.3, we show how random Fourier features provide a scalable approximation to operating in the RKHS.

### 6.4.2 S1 Regression for HSE-PSRs

As discussed in section 6.3 we can use a joint or conditional approach for S1 regression. We now demonstrate how these two approaches apply to HSE-PSRs.

#### 6.4.2.1 Joint S1 Regression for HSE-PSRs

This is the method used in (Boots et al., 2013). In this approach we exploit the fact that

$$\bar{Q}_t = W_{\psi_t^O|\psi_t^A;h_t} = C_{\psi_t^O\psi_t^A|h_t}(C_{\psi_t^A\psi_t^A|h_t} + \lambda I)^{-1}$$

So, we learn two linear maps $T_{oa}$ and $T_a$ such that

$$\begin{aligned} T_{oa}(h_t) &\approx C_{\psi_t^O\psi_t^A|h_t}, \\ T_a(h_t) &\approx C_{\psi_t^A\psi_t^A|h_t}. \end{aligned}$$

The training examples for $T_{oa}$ and $T_a$ consist of pairs $(h_t, \psi_t^O \otimes \psi_t^A)$ and $(h_t, \psi_t^A \otimes \psi_t^A)$ respectively. Once we learn this map, we can estimate $C_{\psi_t^O\psi_t^A|h_t}$ and $C_{\psi_t^A\psi_t^A|h_t}$ and consequently estimate $\bar{Q}_t$.

#### 6.4.2.2 Conditional S1 Regression for HSE-PSRs

It is also possible to apply the conditional S1 regression formulation in Section 6.3.2. Specifically, let $\mathcal{F}$ be the set of 3-mode tensors, with modes corresponding to $\psi_t^O$, $\psi_t^A$ and $h_t$. We estimate a tensor $T^*$ by optimizing

$$T^* = \arg\min_{T\in\mathcal{F}} \|(T \times_{h_t} h_t \times_{\psi_t^A} \psi_t^A) - \psi_t^O\|^2 + \lambda\|T\|_{HS}^2,$$

where $\|.\|_{HS}^2$ is the Hilbert-Schmidt norm, which translates to Frobenius norm in finite-dimensional Euclidan spaces. We can then use

$$\bar{Q}_t = T^* \times_{h_t} h_t$$

For both regression approaches, the same procedure can be used to estimate the extended state $\bar{P}_t$ by replacing features $\psi_t^O$ and $\psi_t^A$ with their extended counterparts $\xi_t^O$ and $\xi_t^A$. In our experiments, we test both S1 regression approaches.

### 6.4.3 From HSE-PSRs to RFF-PSRs

We now describe our main proposal for modeling controlled dynamical systems. Predictive state representations with random Fourier features (RFF-PSRs) build upon HSE-PSRs and improve them by (1) using finite dimensional approximations to kernel feature maps and (2) using discriminative training through backpropagation through time to further refine the model obtained by two-stage regression initialization. We describe these improvements in the following subsections.

#### 6.4.3.1 Approximating HSE-PSRs with Random Fourier Features

A Gram matrix formulation of HSE-PSRs has computational and memory requirements that grow rapidly with the number of training examples. To alleviate this problem, we resort to kernel approximation—that is, we replace RKHS vectors such as $\psi_t^o$ and $\psi_t^a$ with finite dimensional vectors that approximately preserve inner products. We use random Fourier features (RFF) (Rahimi and Recht, 2008) as an approximation but it is possible to use other approximation methods. Unfortunately RFF approximation can typically require $D$ to be prohibitively large. Therefore, we apply principal component analysis (PCA) to the feature maps to reduce their dimension to $p \ll D$. We apply PCA again to quantities that require $p^2$ space such as extended features $\xi_t^O$, $\xi_t^A$ and states $\bar{Q}_t$, reducing them to $p$ dimensions. We map them back to $p^2$ dimensions when needed (e.g., for filtering). The two-stage regression algorithm for RFF-PSRs is depicted in Algorithm 6. For ease of exposition, we assume that RFF features are computed prior to PCA. In our implementation, we compute the RFF features on the fly while performing PCA to reduce the required memory footprint. We also employ randomized SVD (Halko et al., 2011) for fast computation of PCA, resulting in an algorithm that scales linearly with $N$ and $D$. The dependency on $p$ is determined by the implementation of regression steps. A reasonable implementation (e.g. using conjugate gradient with a fixed number of iterations) would result in overall time complexity of $O(p^3N + pDN)$ and space complexity of $O(p^2N + Dp + p^3)$.

#### 6.4.3.2 Local Refinement by Discriminative Training

Similar to the PSRNN model proposed in Chapter 4, we can interpret the RFF-PSR as the following recurrent network

$$\begin{aligned} q_{t+1} &= f_{\text{filter}}(W_{\text{system}} q_t, o_t, a_t), \\ \mathbb{E}[o_t | q_t, \mathbf{do}(a_t)] &= W_{\text{pred}}(q_t \otimes \phi^a(a_t)), \end{aligned}$$

where $W_{\text{pred}}$ is a $d_o \times p^2$ prediction matrix and $d_o$ is the observation dimension. The argument for linearity of $W_{\text{pred}}$ is similar to that we used for PSRNNs: we assume the future RKHS is rich enough to contain (a good approximation of) the functions that extract individual coordinates from future observation windows. Therefore, we can improve our estimates of $W_{\text{system}}$ and $W_{\text{pred}}$ using backpropagation through time. We can also train and optimize a $k$-step predictor $W_{\text{pred}}^k$ such that

$$\mathbb{E}[o_{t:t+k-1} | q_t, \mathbf{do}(a_{t:t+k-1})] = W_{\text{pred}}(q_t \otimes \phi^A(a_{t:t+k-1}))$$

[4]MATLAB source code is available at: `https://github.com/ahefnycmu/rffpsr`

Note that it is possible to use examples from a non-blind policy for the discriminative training of $W_{\text{pred}}$ without introducing bias. However, that is not true for the multistep predictor $W^k_{\text{pred}}$, since examples from a non-blind policy can cause it to exploit future actions in predicting previous observations. One possible solution to optimize a multistep predictor with examples from a non-blind policy is to learn a set of predictors $\tilde{W}^l$ for $(1 \leq l \leq k)$ such that

$$\mathbb{E}[o_{t+l-1}|q_t, \mathbf{do}(a_{t:t+l-1})] = \tilde{W}^l(q_t \otimes \phi^{A^l}(a_{t:t+l-1})).$$

In Chapter 7, we utilize examples from non-blind policies to refine an RFF-PSR. In that scenario, a one-step predictor was sufficient to obtain an effective model.

**Algorithm 6** Learning Predictive State Representation with Random Fourier Features (LEARN-RFF-PSR)

**Input:**Matrices $\Phi^h, \Phi^o, \Phi^a$ of history, observation and action features (each column corresponds to a time step). Matrices $\Psi^o, \Psi^a, \Psi^{o\prime}, \Psi^{a\prime}$ of test observations, test actions, shifted test observations and shifted test actions.

**Output:**S2 regression weights $\hat{W}_\xi$ and $\hat{W}_o$.

**Subroutines:** $\text{SVD}(X, p)$, returns the tuple $(U, U^\top X)$, where $U$ consists of top $p$ singular vectors of $X$.

// Feature projection using PCA
$U^h, \Phi^h \leftarrow \text{SVD}(\Phi^h, p)$;
$U^o, \Phi^o \leftarrow \text{SVD}(\Phi^o, p)$; $U^a, \Phi^a \leftarrow \text{SVD}(\Phi^a, p)$;
$U^o_\psi, \Psi^o \leftarrow \text{SVD}(\Psi^o, p)$; $U^a_\psi, \Psi^a \leftarrow \text{SVD}(\Psi^a, p)$;
$U^o_\xi, \Xi^o \leftarrow \text{SVD}((U^{o\top}_\psi \Psi^{o\prime}) \star \Phi^o, p)$;
$U^a_\xi, \Xi^a \leftarrow \text{SVD}(\Phi^a \star (U^{a\top}_\psi \Psi^{a\prime}), p)$;
$U^{oo}, \Phi^{oo} \leftarrow \text{SVD}(\Phi^o \star \Phi^o, p)$

// S1 Regression and State Projection
Estimate $\bar{Q}_t$, $\bar{P}^\xi_t$, $\bar{P}^o_t$ for each time $t$ using one of the S1 methods in Section 6.4.2.
Reshape $\bar{Q}_t$, $\bar{P}_t$ as column vectors for each $t$ and then stack the resulting vectors in matrices $\mathbf{Q}$, $\mathbf{P}^\xi$ and $\mathbf{P}^o$.
$U^q, \mathbf{Q} \leftarrow \text{SVD}(\mathbf{Q}, p)$
// S2 Regression
$\hat{W}_\xi \leftarrow \arg\min_{W \in \mathbb{R}^{p^2 \times p}} \|\mathbf{P}^\xi - W\mathbf{Q}\|^2 + \lambda_2 \|W\|^2_F$
$\hat{W}_o \leftarrow \arg\min_{W \in \mathbb{R}^{p^2 \times p}} \|\mathbf{P}^o - W\mathbf{Q}\|^2 + \lambda_2 \|W\|^2_F$

## 6.5 Experiments

We demonstrate the efficacy of RFF-PSR in a number of continuous controlled dynamical systems.

### 6.5.1 Synthetic Data

We use the benchmark synthetic non-linear system used by Boots et al. (2013) :

$$\begin{aligned}
\dot{x}_1(t) &= x_2(t) - 0.1\cos(x_1(t))(5x_1(t) - 4x_1^3(t) + x_1^5(t)) \\
&\quad - 0.5\cos(x_1(t))a(t) \\
\dot{x}_2(t) &= -65x_1(t) + 50x_1^3(t) - 15x_1^5(t) - x_2(t) - 100a(t) \\
o(t) &= x_1(t)
\end{aligned}$$

The input $a$ is generated as zero-order hold white noise, uniformly distributed between $-0.5$ and 0.5. We collected 20 trajectories of 100 observations and actions at 20Hz and we split them into 10 training, 5 validation and 5 test trajectories. The prediction target for this experiment is $o(t)$.

### 6.5.2 Simulated windshield view

In this experiment we used the TORCS car simulation server, which outputs 64x64 images (see Figure 6.2). The observations are produced by converting the images to greyscale and projecting them to 200 dimensions via PCA. The car is controlled by a built-in controller that controls acceleration while the external actions control steering. We collected 50 trajectories by applying a sine wave with random starting phase to the steering control and letting the simulator run until the car goes off the track. We used 40 trajectories for training, 5 for validation and 5 for testing. The prediction target is the projected image.

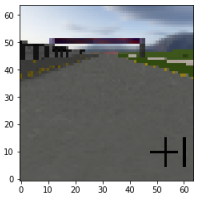



Figure 6.2: An example of windshield view output by TORCS.

### 6.5.3 Simulated swimmer robot

We consider the 3-link simulated swimmer robot from the open-source package RLPy (Geramifard et al., 2013). The 2-d action consists of torques applied on the two joints of the links. The observation model returns the angles of the joints and the position of the nose (in body coordinates). The measurements are contaminated with Gaussian noise whose standard deviation is 5% of the true signal standard deviation. To collect the data, we use an open-loop policy that selects actions uniformly at random. We collected 25 trajectories of length 100 each and use 24 for training and 1 for validation. We generate test trajectories using a mixed policy: with probability $p_{\text{blind}}$, we sample a uniformly random action, while with probability $1 - p_{\text{blind}}$, we sample an action from a pre-specified deterministic policy that seeks a goal point. We generate two sets of 10 test trajectories each, one with $p_{\text{blind}} = 0.8$ and another with $p_{\text{blind}} = 0.2$. The prediction target is the position of the nose.


Figure 6.3: Data collection process for the cell phone dataset and two sample images.

### 6.5.4 Cell phone Camera and Sensors

In this experiment we model data from a Google Pixel 2 cell phone. The phone was held in upright position and rotated around a fixed point using a random blind policy. Observations are $144 \times 176$ images captured by the main camera, converted to grayscale, and projected to 1000 dimensions using PCA. Actions are 3-dimensional vectors containing the angular velocities around $x$, $y$ and $z$ axis in rad/s. These were measured by applying a low-pass filter to the gyroscope sensor of the phone. Data were sampled at the rate of 10Hz. Figure 6.3 visualizes the data collection process and shows some images.

We collected 5234 data points that we divided into 4500 training and 734 testing. The 4500 training points were divided into 9 trajectories of length 500 each. Two trajectories were used as a validation set.

### 6.5.5 Tested Methods and Evaluation Procedure

We tested three different initialization schemes of RFF-PSR (with Gaussian RBF kernel): random initialization, two-stage regression with joint S1, and two-stage regression with conditional S1. For each initialization scheme, we tested the model before and after refinement. For refinement we used BPTT with a decreasing step size: the step size is reduced by half if validation error increases. Early stopping occurs if the step size becomes too small ($10^{-5}$) or the relative change in validation is insignificant ($10^{-3}$). We also test the following baselines.

**HSE-PSR**: We implemented the Gram matrix HSE-PSR as described in (Boots et al., 2013).

**N4SID**: We used MATLAB's implementation of subspace identification of linear dynamical systems (van Overschee and de Moor, 1996).

**Non-linear Auto Regression (RFF-ARX)**: We implemented a version of auto regression where the predictor variable is the RFF representation of future actions together with a finite history of previous observations and actions, and the target variable is future observations.

Models were trained with future length of 10 and history length of 20. For RFF-PSR and RFF-ARX we used 10000 random features and applied PCA to project features onto 20 dimensions. Kernel bandwidths were set to the median of the distance between training points (median trick). For evaluation, we perform filtering on the data and estimate the prediction target of the experiment at test time $t$ given the history $o_{1:t-H}, a_{1:t}$, where $H$ is the prediction horizon. We report the mean

square error across all times $t$ for each value of $H \in \{1, 2, \ldots, 10\}$.

### 6.5.6 Results and Discussion

The results for the first four domains are shown in Figure 6.4. There are a number of important observations.

- In general, joint S1 training closely matches or outperforms conditional S1 training, with and without refinement.
- Local refinement significantly improves predictive performance for all initialization methods.
- Local refinement, on its own, is not sufficient to produce a good model. The two stage regression provides a good initialization of the refinement procedure.
- Even without refinement, RFF-PSR outperforms HSE-PSR. This could be attributed to the dimensionality reduction step, which adds appropriate inductive bias.
- Compared to other methods, RFF-PSR has better performance with non-blind test policies.

Figure 6.5 shows prediction results for the cell phone dataset. We still see that RFF-PSR with refinement outperforms other models for long-term predictions. To gain additional insight on the effect of refinement, Figure 6.6 (right) depicts a slice of the validation error surface along the direction between the initialization obtained by two-stage regression and the final point obtained by refinement. We note that while two-stage regression is not sufficient to optimize the prediction error, it has crucial value in starting the refinement from a good basin of attraction that would be otherwise difficult to reach.

We also demonstrate "closing the loop" behavior in the cell phone dataset. In this context, closing the loop refers to the ability of the model to determine from observations that the system has returned to its initial state. We test the behavior by running the RFF-PSR model on a test sequence that consists of a single complete revolution around the fixed point. Figure 6.6 (left) shows the first three dimensions of the belief state as it progresses through time. The figure shows that the initial and final belief states are very close to each other.

## 6.6 Other Examples of Predictive State Controlled Models

Here we discuss IO-HMM and Kalman filter with inputs, showing that they are instances of PSCMs. We do this for each model by defining the predictive state, showing that it satisfies the condition $P_t = WQ_t$ and describing an S1 regression method.

### 6.6.1 IO-HMM

Let $T$ be the transition tensor such that $T \times_s s_t \times_a a_t = \mathbb{E}[s_{t+1}|a_t, s_t]$ and $O$ be the observation tensor such that $O \times_s s_t \times_a a_t = \mathbb{E}[o_t|a_t, s_t]$.

Define $O^k$ to be the extended observation tensor where $O_k \times_s s_t \times_a a_{t:t+k-1} = \mathbb{E}[o_{t:t+k-1}|a_{t:t+k-1}, s_t]$

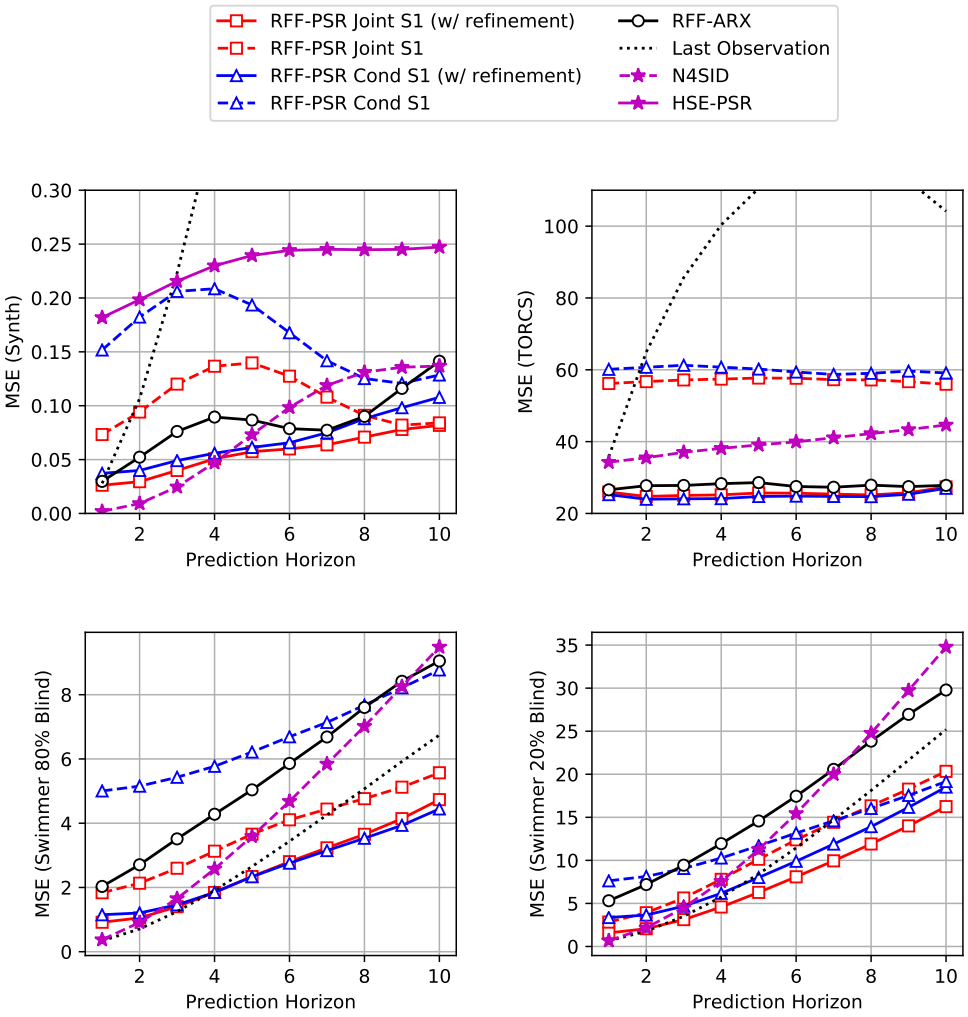



Figure 6.4: Mean square error for 10-step prediction on synthetic model, TORCS car simulator, swimming robot simulation with 80% blind test-policy, and swimming robot with 20% blind test policy. Randomly initialized RFF-PSRs obtained significantly worse MSE and are not shown for clarity. A comparison with HSE-PSR on TORCS and swimmer datasets was not possible as it required prohibitively large memory.

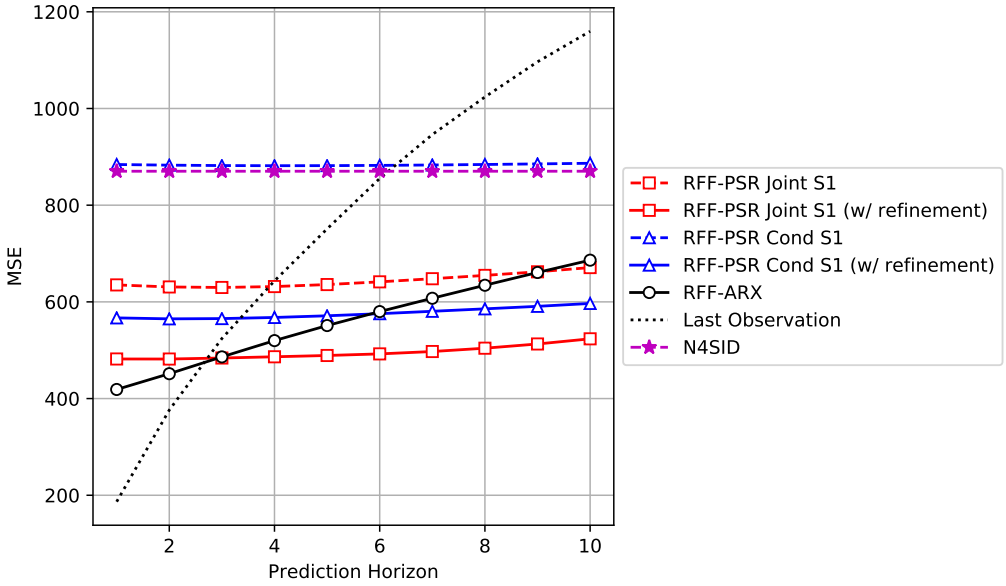



Figure 6.5: Mean square error for different prediction horizons for the cell phone dataset.

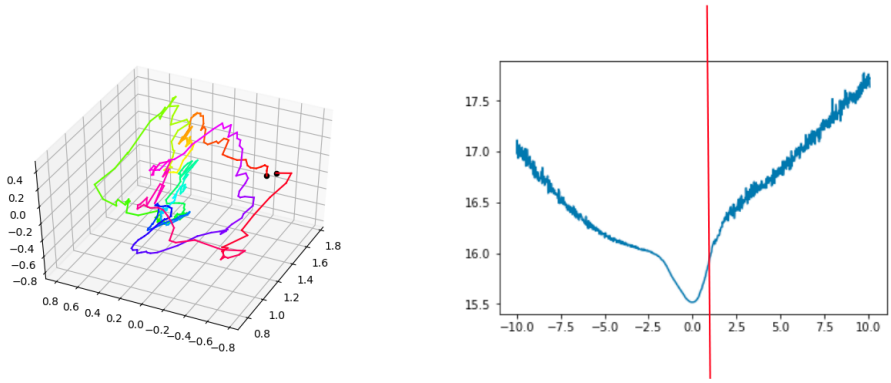



Figure 6.6: **left:** Visualization of the first three coordinates of the projected belief state for a trajectory corresponding to a full revolution of the cell phone. Black dots indicate start and end points. **right:** Log mean square validation error for the cell phone experiment along a slice in the parameter space determined by the direction from the two-stage regression initialization (indicated by the red vertical line) to the final parameters obtained by refinement (indicated by x-axis value 0).

As a shortcut, we will denote by $T_{ij}$ the product $T \times_s e_i \times_a e_j$.

For $k = 1$, we have $O^1 = O$.

For $k > 1$ we can think of $a_{t:t+k-1}$ as the outer product $a_t \otimes a_{t+1:t+k}$. So we can define $O^k$ such that

$$O^k \times_s e_i \times_a (e_j \otimes e_l) = \text{vec}(O_{ij} \otimes (O^{k-1} \times_a e_l \times_s T_{ij})) \tag{6.6}$$

In words, starting from state $e_i$ and applying an action $e_j$ followed by a sequence of $k-1$ actions denoted by indicator $e_l$. The expected indicator of the next $k$ observations is the outer product of expected observation $o_t$ (given by $O_{ij}$) with the expected indicator of observations $o_{t+1:t+k-1}$ as predicted by $O^{k-1}$. Note that the two expectations being multiplied are conditionally independent given the state $e_i$ and the action sequence.

Given the tensor $O^k$ the predictive states $Q_t$ and $P_t$ are defined to be

$$Q_t = O^k \times_s s_t$$
$$P_t = O^{k+1} \times_s s_t$$

Now to show that (6.2) holds, let $\tilde{O}^k$ be a reshaping of $O^k$ into a matrix such that

$$\text{vec}(Q_t) = \tilde{O}^k s_t$$

It follows that

$$P_t = O^{k+1} \times_s s_t = O^{k+1} \times_s ((\tilde{O}^k)^+ \text{vec}(Q_t)),$$

which is linear in $Q_t$.

#### 6.6.1.1 S1 Regression

Let $s_t = s(h_t^\infty)$ be the belief state at time $t$. Note that $s_t$ is a deterministic function of the entire history.

Under a fixed policy assumption, an indicator vector of the joint observation and action assignment is an unbiased estimate of the joint probability table $\mathbb{P}[\psi_t^A, \xi_t^A \mid h_t^\infty]$. An S1 regression model can be used to learn the mapping $h_t \mapsto \mathbb{P}[\psi_t^A, \xi_t^A \mid h_t]$. It is then easy to estimate the conditional probability table $\bar{Q}_t$ from the joint probability table $\mathbb{P}[\psi_t^A, \xi_t^A \mid h_t]$.

We can also use the conditional S1 approach. By exploiting the fact that $\psi_t^O$ is an unbiased estimate of a single column of $Q_t$ corresponding to $\psi_t^A$. We can use (6.4) to learn a function $f : h_t \mapsto \bar{Q}_t$ that best matches the training examples.

### 6.6.2 Kalman Filter with inputs

The Kalman filter is given by

$$x_t = Ax_{t-1} + Bu_t + \epsilon_t$$
$$o_t = Cx_t + \nu_t$$

Given a belief state $s_t \equiv \mathbb{E}[x_{t-1}|h_t^\infty]$ we can write the predictive state as

$$\mathbb{E}[o_{t:t+k-1} \mid s_t, a_{t:t+k-1}] = \Gamma_k s_t + U_k a_{t:t+k-1},$$

where

$$\Gamma_k = \begin{pmatrix} CA \\ CA^2 \\ \vdots \\ CA^k \end{pmatrix}$$

$$U_k = \begin{pmatrix} B & \mathbf{0} & \dots & & & \mathbf{0} \\ AB & B & \mathbf{0} & \dots & & \mathbf{0} \\ A^2B & AB & B & \mathbf{0} & \dots & \mathbf{0} \\ & & \vdots & & & \\ A^{k-1}B & & \dots & & AB & B \end{pmatrix}$$

The extended predictive state have similar form with $\Gamma_k$ and $U_k$ replaced with $\Gamma_{k+1}$ and $U_{k+1}$. Since $U$ is fixed, keeping track of the state amounts to keeping track of $Q_t \equiv \Gamma_k s_t$. It follows that

$$P_t = \Gamma_{k+1} s_t = \Gamma_{k+1} \Gamma_k^+ Q_t = W Q_t$$

If $h_t$ is a linear projection of $h_t^\infty$ (e.g. stacking of a finite window of observations and actions), it can also be shown van Overschee and de Moor (1996) that

$$\mathbb{E}[Q_t|h_t] = \tilde{\Gamma}_k h_t,$$

for some matrix $\tilde{\Gamma}_k$.

#### 6.6.2.1 S1 Regression

Let $\mathcal{F}$ be the set of functions that take the form

$$f(h)\psi_t^A = \Gamma h_t + B\psi_t^A$$

The oblique projection method van Overschee and de Moor (1996) uses linear regression to estimate $\Gamma$ and $B$ (essentially solving (6.4)). Having a fixed $B$, the conditional operator is determined by $\Gamma h_t$ through an affine transformation. Therefore we can use $\bar{Q}_t = \Gamma h_t$.

## 6.7 Theoretical Analysis of Predictive State Controlled Models

It is worth noting that Algorithm 5 is still an instance of the two stage regression framework described in Chapter 3. Does that mean it retains the theoretical guarantee in Theorem 3.5? In

other words, assuming that we collect i.i.d examples using a blind policy[5] , can we bound the error in estimating the dynamics matrix $W_{\text{system}}$ in terms of S1 regression error bounds?

The answer is highly dependent on the policy. Assume a policy always takes a fixed action. With such a policy we cannot accurately estimate $Q_t$ even with infinite data. A policy needs to provide sufficient exploration for data collection. Otherwise, we will not be able to estimate $W_{\text{system}}$ even using perfect S1 regression models. It turns out that sufficient exploration is more than trying all actions. To describe it more formally, we introduce the notion of a sufficient history set.

**Definition 6.2** (Sufficient History Set)**.** *Consider a PSCM that satisfies*

$$P_t = W_{\text{system}}(Q_t)$$

*Let $\mathcal{H} = \{h_i\}_{i=1}^{M}$ be a set of possible assignments to the history features. The set $\mathcal{H}$ is called a **sufficient history set** if it is sufficient to estimate $W_{\text{system}}$ using $\mathbb{E}[Q_t|h_t = h]$ and $\mathbb{E}[P_t|h_t = h]$ for each $h \in \mathcal{H}$.* [6]

A blind data collection policy provides sufficient exploration if it allows for estimating $\mathbb{E}[Q|h_t = h]$ and $\mathbb{E}[P|h_t = h]$ for a sufficient history set with error that vanishes as the number of training examples goes to infinity.

We discuss two scenarios: a discrete system where S1 uses counting to estimate probability tables, and a continuous system where S1 uses ridge regression. In each scenario, we describe conditions for sufficient exploration and derive an S1 error bound. As long as the exploration condition is satisfied, we can plug-in the S1 error bound in Theorem 3.5 to obtain an error bound on $W_{\text{system}}$.

### 6.7.1 Case 1: Discrete Observations and Actions

We consider a discrete system where history features are indicator vectors over sequences of observations and actions of length $\tau_h$, and future observations (resp. action) features are indictor vectors over observations (resp. actions) of length $\tau_f$.

We indicate by $\mathcal{H}$, the set of all possible history features.

**Theorem 6.3.** *Assume a discrete system where the data collection policy induces an stationary distribution over histories. If the policy generates each possible extended future action sequence starting from each possible history $M$ times, then it generates an S2 training dataset of size $N = M|\mathcal{H}||\mathcal{A}^{\tau_f+1}|$ with uniform S1 error bound $\tilde{\eta}_{\delta,N} = \sqrt{\frac{|\mathcal{A}^{\tau_f+1}||\mathcal{O}^{\tau_f+1}|}{2M} \log\left(\frac{8|\mathcal{H}||\mathcal{A}^{\tau_f+1}||\mathcal{O}^{\tau_f+1}|}{\delta}\right)}$*

**Remark 6.4.** *Assume the system to be 1-observable, where the history and future are of length 1. Then a consistent estimate of $Q$ and $P$ can be obtained by a consistent estimate of the joint probability table $P(o_{t-1:t+1}, a_{t-1:t+1})$.*

[5] The i.i.d property is achieved if we can restart the system or if the data collection policy induces an ergodic process with a stationary distribution. In the latter case, we assume the examples are sufficiently spaced in time to that allow the process to mix. However, in practice, we use all examples as this makes the error only smaller.

[6] Strictly speaking we aim to accurately estimate $W_{\text{system}}Q$ for any valid $Q$. $W_{\text{system}}$ may not be unique.

### 6.7.2 Case 2: Continuous System

We will limit our discussion to the finite dimensional case where the S1 regression model is ridge regression, since this is the model we use to contruct the RFF-PSR in Section 6.4. First, we discuss sufficient exploration conditions for joint and conditional S1 regression.

**Definition 6.5** (Range and span of a policy)**.** *Let $\pi$ be a data collection policy with a stationary distribution. For a random vector $X_t = f(h_t^\infty, o_{t:\infty}, a_{t:\infty})$, the **range of** $\pi$ **on** $X$ is the support of the stationary distribution of $X_t$ induced by the policy $\pi$ (i.e. the set of all possible values of $X_t$ that can be generated by the stationary distribution).*

*The **span of** $\pi$ **on** $X$ (denoted by $\text{span}_\pi(X)$) is the subspace spanned by the range of $\pi$ on $X$.*

When referring to the policy range or span, we may omit the variable name when it is clear in the context.

**Condition 6.6** (Action span for joint S1)**.** *Let $\pi$ be data collection policy and let $\mathcal{H}$ be the range of $\pi$ on history features. The action span condition for joint S1 is defined as the requirement to satisfy the following:*

1. *$\mathcal{H}$ is a sufficient history set.*
2. *For any $h \in \mathcal{H}$, the conditional covariance $\mathcal{C}_{\psi^A|h}$ is full rank.*

**Condition 6.7** (Action span for conditional S1)**.** *Let $\pi$ be data collection policy and let $\mathcal{H}$ be the range of $\pi$ on history features. The action span condition for conditional S1 is defined as the requirement to satisfy the following:*

1. *$\mathcal{H}$ is a sufficient history set.*
2. *For any $h \in \mathcal{H}$ and any future action feature vector $\psi^A$, the quantity $(h \otimes \psi^A)$ is in the policy span.*

**Remark 6.8.** *Condition 6.6 implies Condition 6.7.*

**Assumption 6.9** (Bounded features)**.** *Let $\mathcal{H}$ be the range of the policy on history features. We assume that $\|h\| < c_h$ for all $h \in \mathcal{H}$. Also, we assume that $\|\psi^O\| \leq c_O$ and $\|\psi^A\| \leq c_A$ for any valid future observation sequence and action sequence respectively.*

We now state our theorem for the continuous case

**Theorem 6.10.** *Let $\pi$ be a blind data collection policy with a stationary distribution. Assume that we use conditional S1 regression with ridge regression as the regression method, and that Assumption 6.9 and Condition 6.7 hold. Then $\pi$ provides sufficient exploration and there exists problem dependent constants $c_1, c_2 > 0$ such that S1 regression prdocues a uniform error bound*

$$\tilde{\eta}_{\delta,N} = O\left(\frac{\frac{\log(1/\delta)}{\sqrt{N}} + \lambda_1}{c_1 + \lambda_1}\right),$$

*as long as $N > c_2 \log(1/\delta)$, where $\lambda_1$ is the ridge regularization parameter.*

A similar theorem theorem holds for joint S1. We show a proof sketch in the appendix.

**Remark 6.11** (Conditioning)**.** *It is known that linear regression converges faster if the problem is well-conditioned. In the two stage regression we need the good conditioning of both stages—that is,*

- *The set of training histories result in a problem $\bar{P}_t = W\bar{Q}_t$ that is well conditioned (S2 conditioning).*
- *The S1 regression problem is well conditioned.*

The second requirement ensures that we converge fast to good estimates of $\bar{Q}_t$ and $\bar{P}_t$. Designing exploration policies that result in well conditioned two-stage regression problems is an interesting direction for future work.

## 6.8 Conclusion

We proposed a framework to learn controlled dynamical systems using two-stage regression. We then applied this framework to develop a scalable method for controlled non-linear system identification: using RFF approximation of HSE-PSR together with a refinement procedure to enhance the model after a two-stage regression initialization. We have demonstrated promising results for the proposed method in terms of predictive performance.

# 6.A Appendix: Proofs

## 6.A.1 Proof of Theorem 6.3

Let's consider the estimation of the extended state $P_t$. For each history $h$, we use $M$ examples to estimate a conditional probability table of size $|\mathcal{A}^{\tau_f+1}||\mathcal{O}^{\tau_f+1}|$ for $P_t$. Hoeffding's inequality tells us that, with probability at least $1-\tilde{\delta}$, the absolute error in a single cell of the table less than $\sqrt{\frac{1}{2M}\log(2/\tilde{\delta})}$. Since there are $|\mathcal{H}||\mathcal{A}^{\tau_f+1}||\mathcal{O}^{\tau_f+1}|$ cells to estimate in all tables, we use $\tilde{\delta} = \delta/4|\mathcal{H}||\mathcal{A}^{\tau_f+1}||\mathcal{O}^{\tau_f+1}|$ so that the error bound applies uniformly to all tables with probability at least $1-\delta/4$ using the union bound. Given the uniform bound, the Frobenious norm of the error in any $P_t$ is less than

$$\sqrt{\frac{|\mathcal{A}^{\tau_f+1}||\mathcal{O}^{\tau_f+1}|}{2M}\log\left(\frac{8|\mathcal{H}||\mathcal{A}^{\tau_f+1}||\mathcal{O}^{\tau_f+1}|}{\delta}\right)}$$

The same principle applies to $Q_t$. Therefore, the bound applies jointly to all $P_t$ and $Q_t$ with probability at least $1-\delta/2$.

## 6.A.2 Proof of Theroem 6.10

To prove theorem 6.10, we prove the following Lemma, of which Theorem 6.10 is an asymptotic statement. We use $Q(h)$ to denote $\mathbb{E}[Q \mid h]$.

**Lemma 6.12.** *Let $\pi$ be a data collection policy and let $\mathcal{H}$ be the range of $\pi$ on histories. If Assumption 6.9 and Condition 6.7 are satisfied and conditional S1 regression is used with a liner model as the correct model, then $\pi$ provides sufficient exploration and, for all $h \in \mathcal{H}$ and any $\delta \in (0,1)$ such that $N > \frac{c^2\log(2d_h d_A/\delta)}{\lambda_{\min}(\mathcal{C}_{h\otimes\psi^A})}$, the following holds with probability at least $1-\delta$*

$$\|\hat{Q}(h) - Q(h)\| \leq$$
$$c_h\left(\sqrt{\frac{\lambda_{\max}(\mathcal{C}_{\psi^O})}{\lambda_{\min}(\mathcal{C}_{h\otimes\psi^A})}}\left(\frac{\sqrt{\lambda_{\min}(\mathcal{C}_{h\otimes\psi^A})}\Delta_1 + \lambda}{\lambda_{\min}(\mathcal{C}_{h\otimes\psi^A})(1-\Delta_3)+\lambda}\right) + \frac{\Delta_2}{\lambda_{\min}(\mathcal{C}_{h\otimes\psi^A})(1-\Delta_3)+\lambda}\right),$$

*where*

$$\Delta_1 = 2c_h c_A\sqrt{\frac{\log(2d_h d_A/\delta)}{N}} + \frac{2\log(2d_h d_A/\delta)}{3N}\left(\frac{c_h^2 c_A^2}{\sqrt{\lambda_{\min}(\mathcal{C}_{h\otimes\psi^A})}} + c_h c_A\right)$$
$$\Delta_2 = 2c_O c_h c_A\sqrt{\frac{\log((d_O+d_h d_A)/\delta)}{N}} + \frac{4c_O c_h c_A\log((d_O+d_h d_A)/\delta)}{3N}$$
$$\Delta_3 = \frac{c_h^2 c_A^2\log(2d_h d_A/\delta)}{\lambda_{\min}(\mathcal{C}_{h\otimes\psi^A})N}$$

The proof strategy is as follows: First, we use matrix concentration bounds to analyze the effect of using estimated covariance matrices. Then, we analyze the effect of error in covariance matrix on regression weights. By combining the results of both analyses, we prove the desired theorems.

#### 6.A.2.1 Error in Covariance Matrix Estimation

**Lemma 6.13** (Matrix Chernoff Inequality (Tropp, 2015))**.** *Consider a finite sequence $\{S_k\}$ of independent, random, Hermitian matrices with common dimension $d$. Assume that*

$$0 \leq \lambda_{\min}(S_k) \quad \textit{and} \quad \lambda_{\max}(S_k) \leq L \quad \textit{for each index } k.$$

*Introduce the random matrix*

$$Z = \sum_k S_k$$

*Define*

$$\mu_{\min} \equiv \lambda_{\min}(\mathbb{E}[Z])$$

*Then, for any $\epsilon \in [0, 1)$*

$$\Pr(\lambda_{\min}(Z) \leq (1-\epsilon)\mu_{\min}) \leq d \left[ \frac{e^{-\epsilon}}{(1-\epsilon)^{1-\epsilon}} \right]^{\mu_{\min}/L} \leq 2de^{-\epsilon\mu_{\min}/L}$$

**Corollary 6.14** (Minimum eigenvalue of empirical covariance)**.** *Let $X$ be a random variable of dimensionality $d$ such that $\|X\| < c$. Let $\{x_k\}_{k=1}^N$ be $N$ i.i.d samples of the distribution of $X$.*

*Define*

$$\mathcal{C}_X \equiv \mathbb{E}[XX^\top] \quad \textit{and} \quad \hat{\mathcal{C}}_X = \frac{1}{N}\sum_{k=1}^N x_k x_k^\top$$

*For any $\delta \in (0, 1)$ such that $N > \frac{c^2 \log(2d/\delta)}{\lambda_{\min}(\mathcal{C}_X)}$ the following holds with probability at least $1 - \delta$*

$$\lambda_{\min}(\hat{\mathcal{C}}_X) \geq \left(1 - \frac{c^2 \log(2d/\delta)}{\lambda_{\min}(\mathcal{C}_X)N}\right) \lambda_{\min}(\mathcal{C}_X)$$

*Proof.* Define $S_k = \frac{1}{N} x_k x_k^\top$. It follows that $\lambda_{\max}(S_k) \leq L = c^2/N$ and $\mu_{\min} = \lambda_{\min}(\mathcal{C}_X)$. Define

$$\delta \equiv 2de^{-\epsilon N \lambda_{\min}(\mathcal{C}_X)/c^2},$$

which implies that

$$\epsilon = \frac{c^2 \log(2d/\delta)}{\lambda_{\min}(\mathcal{C}_X)N}$$

It follows from Lemma 6.13 that $\Pr(\lambda_{\min}(\hat{\mathcal{C}}_X) \leq (1-\epsilon)\mu_{\min}) \leq \delta$

□

**Lemma 6.15** (Matrix Bernstein Inequality (Tropp, 2015)). *Consider a finite sequence $\{S_k\}$ of independent, random matrices with common dimensions $d_1 \times d_2$. Assume that*

$$\mathbb{E}[S_k] = 0 \text{ and } \|S_k\| \leq L \quad \text{for each index } k$$

*Introduce the random matrix*

$$Z = \sum_k S_k$$

*Let $v(Z)$ be the matrix variance statistic of the sum:*

$$v(Z) = \max\{\|\mathbb{E}(ZZ^\top), \mathbb{E}(Z^\top Z)\|\}$$

*Then*

$$\Pr(\|Z\| \geq t) \leq (d_1 + d_2) \exp\left(\frac{-t^2/2}{v(Z) + Lt/3}\right)$$

**Corollary 6.16** (Error in empirical cross-covariance). *With probability at least $1 - \delta$*

$$\|\hat{\mathcal{C}}_{Y,X} - \mathcal{C}_{Y,X}\| \leq \sqrt{\frac{2\log((d_X + d_Y)/\delta)v}{N}} + \frac{2\log((d_X + d_Y)/\delta)L}{3N},$$

*where*

$$\begin{aligned} L &= c_y c_x + \|\mathcal{C}_{Y,X}\| \leq 2c_y c_x \\ v &= \max(c_y^2\|\mathcal{C}_X\|, c_x^2\|\mathcal{C}_Y\|) + \|\mathcal{C}_{Y,X}\|^2 \leq 2c_y^2 c_x^2 \end{aligned}$$

*Proof.* Define $S_k = y_k x_k^\top - \mathcal{C}_{Y,X}$, it follows that

$$\begin{aligned} \mathbb{E}[S_k] &= 0 \\ \|S_k\| &= \|y_k x_k^\top - \mathcal{C}_{Y,X}\| \leq \|y_k\|\|x_k\| + \|\mathcal{C}_{Y,X}\| \leq c_y c_x + \|\mathcal{C}_{Y,X}\| \\ \|\mathbb{E}[ZZ^\top]\| &= \left\|\sum_{i,j}(\mathbb{E}[y_i x_i^\top x_j y_j^\top] - \mathcal{C}_{Y,X}\mathcal{C}_{X,Y})\right\| \\ &= \left\|\sum_i(\mathbb{E}[\|x_i\|^2 y_i y_i^\top] - \mathcal{C}_{Y,X}\mathcal{C}_{X,Y}) + \sum_{i,j\neq i}(\mathbb{E}[y_i x_i^\top]\mathbb{E}[x_j y_j^\top] - \mathcal{C}_{Y,X}\mathcal{C}_{X,Y})\right\| \\ &\leq N(c_x^2\|\mathcal{C}_Y\| + \|\mathcal{C}_{Y,X}\|^2) \\ \|\mathbb{E}[Z^\top Z]\| &\leq N(c_y^2\|\mathcal{C}_X\| + \|\mathcal{C}_{Y,X}\|^2) \end{aligned}$$

Applying Lemma 6.15 we get

$$\delta = \Pr(\|Z\| \geq Nt) \leq (d_X + d_Y) \exp\left(\frac{-Nt^2/2}{v + Lt/3}\right)$$

and hence

$$t^2 - \frac{2\log((d_X + d_Y)/\delta)Lt}{3N} - \frac{2\log((d_X + d_Y)/\delta)v}{N} \leq 0$$

This quadratic inequality implies

$$t \leq \frac{\log((d_X + d_Y)/\delta)L}{3N} + \sqrt{\frac{\log^2((d_X + d_Y)/\delta)L^2}{9N^2} + \frac{2\log((d_X + d_Y)/\delta)v}{N}}$$

Using the fact that $\sqrt{a^2 + b^2} \leq |a| + |b|$ we get

$$t \leq \frac{2\log((d_X + d_Y)/\delta)L}{3N} + \sqrt{\frac{2\log((d_X + d_Y)/\delta)v}{N}}$$

□

**Corollary 6.17** (Normalized error in empirical covariance)**.** *With probability at least* $1 - \delta$

$$\|\mathcal{C}_X^{-1/2}(\hat{\mathcal{C}}_X - \mathcal{C}_X)\| \leq 2c\sqrt{\frac{2\log(2d/\delta)}{N}} + \frac{2\log(2d/\delta)L}{3N},$$

*where*

$$L = \frac{c^2}{\sqrt{\lambda_{\min}(\mathcal{C}_X)}} + c$$

*Proof.* Define $S_k = \mathcal{C}_X^{-1/2} x_k x_k^\top - \mathcal{C}_X^{1/2}$, it follows that

$$\begin{aligned}
\mathbb{E}[S_k] &= 0 \\
\|S_k\| &\leq \|\mathcal{C}_X^{-1/2}\|\|x_k\|^2 + \|\mathcal{C}_X^{1/2}\| \leq \frac{c^2}{\sqrt{\lambda_{\min}(\mathcal{C}_X)}} + c \\
\|\mathbb{E}[Z^\top Z]\| = \|\mathbb{E}[ZZ^\top]\| &= \left\|\sum_{i,j}(\mathcal{C}_X^{-1/2}\mathbb{E}[x_i x_i^\top x_j x_j^\top]\mathcal{C}_X^{-1/2} - \mathcal{C}_X)\right\| \\
&= \left\|\sum_i(\mathbb{E}[\|x_i\|^2\mathcal{C}_X^{-1/2}x_i x_i^\top \mathcal{C}_X^{-1/2}] - \mathcal{C}_X) + \sum_{i,j\neq i}(\mathcal{C}_X^{-1/2}\mathbb{E}[x_i x_i^\top]\mathbb{E}[x_j x_j^\top]\mathcal{C}_X^{-1/2} - \mathcal{C}_X)\right\| \\
&\leq N(c_x^2 + \|\mathcal{C}_X\|^2) \leq 2Nc^2
\end{aligned}$$

Applying Lemma 6.15 we get

$$\delta = \Pr(\|Z\| \geq Nt) \leq 2d \exp\left(\frac{-Nt^2/2}{2c^2 + Lt/3}\right)$$

and similar to the proof of Corollary 6.16, we can show that

$$t \leq \frac{2\log(2d/\delta)L}{3N} + 2c\sqrt{\frac{\log(2d/\delta)}{N}}$$

□

**Lemma 6.18.** *Let $\hat{\mathcal{C}}_{YX} = \mathcal{C}_{YX} + \Delta_{YX}$ and $\hat{\mathcal{C}}_X = \mathcal{C}_X + \Delta_X$ where $\mathbb{E}[\Delta_{YX}]$ and $\mathbb{E}[\Delta_{YX}]$ are not necessarily zero and $\hat{\mathcal{C}}_X$ is symmetric positive semidefinite. Define $W = \mathcal{C}_{YX}\mathcal{C}_X^{-1}$ and $\hat{W} = \hat{\mathcal{C}}_{YX}(\hat{\mathcal{C}}_X + \lambda)^{-1}$. It follows that*

$$\|\hat{W} - W\| \leq \sqrt{\frac{\lambda_{\max}(\mathcal{C}_Y)}{\lambda_{\min}(\mathcal{C}_X)}}\left(\frac{\sqrt{\lambda_{\min}(\mathcal{C}_X)}\|\mathcal{C}_X^{-1/2}\Delta_X\| + \lambda}{\lambda_{\min}(\hat{\mathcal{C}}_X) + \lambda}\right) + \frac{\|\Delta_{YX}\|}{\lambda_{\min}(\hat{\mathcal{C}}_X) + \lambda}$$

*Proof.*

$$\hat{W} - W = \mathcal{C}_{YX}\left((\mathcal{C}_X + \Delta_X + \lambda I)^{-1} - \mathcal{C}_X^{-1}\right) + \Delta_{YX}(\mathcal{C}_X + \Delta_X + \lambda I)^{-1} = T_1 + T_2$$

It follows that

$$\|T_2\| \leq \frac{\|\Delta_{YX}\|}{\lambda_{\min}(\hat{\mathcal{C}}_X) + \lambda}$$

As for $T_1$, using the matrix inverse Lemma $B^{-1} - A^{-1} = B^{-1}(A - B)A^{-1}$ and the fact that $\mathcal{C}_{YX} = \mathcal{C}_Y^{1/2} V \mathcal{C}_X^{1/2}$, where $V$ is a correlation matrix satisfying $\|V\| \leq 1$ we get

$$\begin{aligned} T_1 &= -\mathcal{C}_{YX}\mathcal{C}_X^{-1}(\Delta_X + \lambda I)(\mathcal{C}_X + \Delta_X + \lambda I)^{-1} \\ &= -\mathcal{C}_Y^{1/2} V \mathcal{C}_X^{-1/2}(\Delta_X + \lambda I)(\mathcal{C}_X + \Delta_X + \lambda I)^{-1}, \end{aligned}$$

and hence

$$\begin{aligned} \|T_1\| &\leq \sqrt{\lambda_{\max}(\mathcal{C}_Y)}\left(\frac{\|\mathcal{C}_X^{-1/2}\Delta_X\| + \lambda\|\mathcal{C}_X^{-1/2}\|}{\lambda_{\min}(\hat{\mathcal{C}}_X) + \lambda}\right) \\ &= \sqrt{\frac{\lambda_{\max}(\mathcal{C}_Y)}{\lambda_{\min}(\mathcal{C}_X)}}\left(\frac{\sqrt{\lambda_{\min}(\mathcal{C}_X)}\|\mathcal{C}_X^{-1/2}\Delta_X\| + \lambda}{\lambda_{\min}(\hat{\mathcal{C}}_X) + \lambda}\right) \end{aligned}$$

□

**Corollary 6.19.** *Let $x_k{}_{k=1}^N$ and $y_k{}_{k=1}^N$ be i.i.d samples from two random variables $X$ and $Y$ with dimensions $d_X$ and $d_Y$ and (uncentered) covariances $\mathcal{C}_X$ and $\mathcal{C}_Y$ respectively. Assume $\|X\| \leq c_x$ and $\|Y\| \leq c_y$. Let $\hat{\mathcal{C}}_{YX} = \frac{1}{N}\sum_{k=1}^N y_k x_k^\top$ and $\hat{\mathcal{C}}_X = \frac{1}{N}\sum_{k=1}^N x_k x_k^\top$. Define $W = \mathcal{C}_{YX}\mathcal{C}_X^{-1}$ and $\hat{W} = \hat{\mathcal{C}}_{YX}(\hat{\mathcal{C}}_X + \lambda)^{-1}$.*

### 6.A.2.2 Error in Regression Weights

*For any $\delta \in (0,1)$ such that $N > \frac{c_x^2 \log(2d_X/\delta)}{\lambda_{\min}(\mathcal{C}_X)}$ the following holds with probability at least $1-3\delta$*

$$\|\hat{W} - W\| \leq \sqrt{\frac{\lambda_{\max}(\mathcal{C}_Y)}{\lambda_{\min}(\mathcal{C}_X)}} \left( \frac{\sqrt{\lambda_{\min}(\mathcal{C}_X)}\Delta_1 + \lambda}{\lambda_{\min}(\mathcal{C}_X)(1-\Delta_3)+\lambda} \right) + \frac{\Delta_2}{\lambda_{\min}(\mathcal{C}_X)(1-\Delta_3)+\lambda},$$

*where*

$$\Delta_1 = 2c_x\sqrt{\frac{\log(2d_X/\delta)}{N}} + \frac{2\log(2d_X/\delta)}{3N}\left(\frac{c_x^2}{\sqrt{\lambda_{\min}(\mathcal{C}_X)}} + c_x\right)$$
$$\Delta_2 = 2c_y c_x\sqrt{\frac{\log((d_Y+d_X)/\delta)}{N}} + \frac{4c_y c_x \log((d_Y+d_X)/\delta)}{3N}$$
$$\Delta_3 = \frac{c_x^2\log(2d_X/\delta)}{\lambda_{\min}(\mathcal{C}_X)N}$$

*Proof.* This corollary follows simply from applying Corollaries 6.14, 6.16 and 6.17 to Lemma 6.18. The $1-3\delta$ bound follows from union bound; since we have three probabilitic bounds each of which holds with probability $1-\delta$. □

**Lemma 6.20.** *Let $\hat{\mathcal{C}}_{Y,X} = \mathcal{C}_{Y,X} + \Delta_{YX}$ and $\hat{\mathcal{C}}_X = \mathcal{C}_X + \Delta_X$ where $\mathbb{E}[\Delta_{YX}]$ and $\mathbb{E}[\Delta_{YX}]$ is not necessarily zero and $\hat{\mathcal{C}}_X$ is symmetric but not necessarily positive semidefinite. Define $W = \mathcal{C}_{Y,X}\mathcal{C}_X^{-1}$ and $\hat{W} = \hat{\mathcal{C}}_{Y,X}\hat{\mathcal{C}}_X(\hat{\mathcal{C}}_X^2+\lambda)^{-1}$. It follows that*

$$\|\hat{W} - W\| \leq \sqrt{\frac{\lambda_{\max}(\mathcal{C}_Y)}{\lambda_{\min}^3(\mathcal{C}_X)}}\frac{\|\Delta_X\|^2 + 2\lambda_{\max}(\mathcal{C}_X)\|\Delta_X\| + \lambda}{\lambda_{\min}^2(\hat{\mathcal{C}}_X)+\lambda} + \frac{\|\mathcal{C}_{Y,X}\|\|\Delta_X\| + \|\Delta_{YX}\|\|\mathcal{C}_X\| + \|\Delta_{YX}\|\|\Delta_X\|}{\lambda_{\min}^2(\hat{\mathcal{C}}_X)+\lambda}$$

*Proof.*

$$\begin{aligned}
\hat{W} - W &= (\mathcal{C}_{Y,X}+\Delta_{YX})(\mathcal{C}_X+\Delta_X)((\mathcal{C}_X+\Delta_X)^2+\lambda I)^{-1} - \mathcal{C}_{Y,X}\mathcal{C}_X\mathcal{C}_X^{-2} \\
&= \mathcal{C}_{Y,X}\mathcal{C}_X(((\mathcal{C}_X+\Delta_X)^2+\lambda I)^{-1} - \mathcal{C}_X^{-2}) + (\mathcal{C}_{Y,X}\Delta_X + \Delta_{YX}\mathcal{C}_X + \Delta_{YX}\Delta_X)((\mathcal{C}_X+\Delta_X)^2+\lambda I)^{-1} \\
&= T_1 + T_2
\end{aligned}$$

Using the matrix inverse Lemma $B^{-1} - A^{-1} = B^{-1}(A-B)A^{-1}$ and the fact that $\mathcal{C}_{Y,X} = \mathcal{C}_Y^{1/2}V\mathcal{C}_X^{1/2}$, where $V$ is a correlation matrix satisfying $\|V\| \leq 1$ we get

$$T_1 = -\mathcal{C}_{Y,X}^{1/2}V\mathcal{C}_X^{-3/2}(\Delta_X^2 + \mathcal{C}_X\Delta_X + \Delta_X\mathcal{C}_X + \lambda I)((\mathcal{C}_X+\Delta_X)^2+\lambda I)^{-1}$$
$$\|T_1\| \leq \sqrt{\frac{\lambda_{\max}(\mathcal{C}_Y)}{\lambda_{\min}^3(\mathcal{C}_X)}}\frac{\|\Delta_X\|^2 + 2\lambda_{\max}(\mathcal{C}_X)\|\Delta_X\| + \lambda}{\lambda_{\min}^2(\hat{\mathcal{C}}_X)+\lambda}$$
$$\|T_2\| \leq \frac{\|\mathcal{C}_{Y,X}\|\|\Delta_X\| + \|\Delta_{YX}\|\|\mathcal{C}_X\| + \|\Delta_{YX}\|\|\Delta_X\|}{\lambda_{\min}^2(\hat{\mathcal{C}}_X)+\lambda}$$

□

### 6.A.2.3 Proof of Lemma 6.12

*Proof.* In the linear case, we estimate a tensor $T$ with modes corresponding to $h$, $\psi^A$ and $\psi^O$ by solving the minimization problem in Section 6.4.2.2. Equivalently, we estimate a matrix $T_r$ of size $d_O \times d_h d_A$ where an input $h \otimes \psi^a$ is mapped to an output $\mathbb{E}[\psi^O \mid h, \psi^A]$. Note that

$$Q(h)\psi^A = T \times_h h \times_A \psi^A = T_r(h \otimes \psi^A)$$

For any history $h \in \mathcal{H}$ and future action feature vector $\psi^A$ we have

$$\begin{aligned}\|\hat{Q}(h) - Q(h)\| &= \text{argmax}_{\psi^A} \frac{\|(\hat{Q}(h) - Q(h))\psi^A\|}{\|\psi^A\|} \\ &= \text{argmax}_{\psi^A} \frac{\|(\hat{T}_r - T_r)(h \otimes \psi^A)\|}{\|\psi^A\|} \leq \|\hat{T}_r - T_r\|\|\psi^h\|\end{aligned}$$

Note that Condition 6.7 implies that $h \otimes \psi^A$ will eventually be in the span of training examples. This rules out the case where the inequality is satisfied only because $(h \otimes \psi^A)$ is incorrectly in the null space of $\hat{T}_r$ and $T_r$.

The theorem is proven by applying Corollary 6.A.2.2 to bound $\|\hat{T}_r - T_r\|$. □

## 6.A.3 Sketch Proof for Joint S1

We show a proof sketch of the asymptotic statement of Theorem 6.12 applied to joint S1 regression.

Let $T_A$ be a tensor such that $\mathcal{C}_{\psi^A|h} = T_A \times_h h$. Note that

$$\begin{aligned}\|\hat{\mathcal{C}}_{\psi^A|h} - \mathcal{C}_{\psi^A|h}\| &\leq \|\hat{T}_A - T_A\|\|h\| \\ \|\hat{\mathcal{C}}_{\psi^O,\psi^A|h} - \mathcal{C}_{\psi^O,\psi^A|h}\| &\leq \|\hat{T}_{OA} - T_{OA}\|\|h\|\end{aligned}$$

From Lemma 6.18, we obtain a high probability bound on $\|\hat{T}_A - T_A\|$ and $\|\hat{T}_{OA} - T_{OA}\|$. Then we apply these bounds to Lemma 6.20 to obtain an error in $Q(h)$.

# Chapter 7

# Reinforcement Learning with Predictive State Controlled Models

In this chapter we use the predictive state controlled models developed in Chapter 6 as a state estimator to provide input for a downstream task, namely reinforcement learning. Our scenario consists of an agent that interacts with a partially observable environment and receives both observations and rewards. The goal is to find a policy that maximizes the cumulative discounted rewards over time.

We propose a class of policies for partially observable environments that we call *recurrent predictive state policy networks* (RPSP networks). An RPSP network consists of a predictive state control model that serves as a recursive filter, and a reactive policy that acts based on the belief state of the filter. While the predictive state model can be thought of as an environment model that can be initialized using two stage regression, the entire policy network can be trained end-to-end using policy gradient methods. In a sense, RPSP networks are in the middle ground between model based and model free reinforcement learning.

The Chapter is organized as follows: In Section 7.1 we give a background on reinforcement learning with a focus on policy gradient methods. In Section 7.2 we describe the proposed RPSP policy class and we describe the learning algorithm in Section 7.3. In Section 7.4 we present empirical evaluation of RPSP networks on a number of reinforcement learning tasks.

## 7.1 Background: Reinforcement Learning and Policy Gradients

We consider an agent that is interacting with the environment in an episodic manner, where each episode consists of $T$ time steps. In each time step, the agent executes an action $a_t \in \mathcal{A}$ and receives an observation $o_t \in \mathcal{O}$ together with a reward $r_t \in \mathbb{R}$. Given a policy parameter vector $\theta$, the agent is acting based on a policy $\pi_\theta$ such that

$$\pi_\theta(a_t \mid h_t^\infty) \equiv \Pr(a_t \mid h_t^\infty; \theta)$$

The goal of reinforcement learning is to find the optimal policy $\pi^*$ that maximizes the expected sum of discounted rewards

$$J(\pi) \equiv \frac{1}{T}\mathbb{E}[\gamma^{t-1} r_t \mid \pi], \tag{7.1}$$

where $\gamma \in [0, 1]$ is a *discount factor* .

Two major approaches to optimize the policy are the value function-based methods and policy gradient methods. We describe them briefly in the following subsection. Note that we describe value functions and policies as functions of the entire history $h_t^\infty$. In practice, we either assume a fully observable environment, where it suffices to use $o_t$, or we use the belief state $q_t$ in the partially observable setting.

### 7.1.1 Value Function-based Methods

In value function-based methods we aim to learn a function that measures the "goodness" of a state as the expected sum of discounted reward when acting optimally starting from this state. A commonly used form of the value function is the $\mathcal{Q}$ function, which is defined as the value of taking a particular action $a_t$ then following a policy $\pi$.

$$\mathcal{Q}_\pi(a, h_t^\infty) \equiv \mathbb{E}[\sum_{t' \geq 0} \gamma^{t'} r_{t+t'} \mid a_t = a, \pi_{t:T} = \pi] = \mathbb{E}[R_t \mid a_t = a, \pi_{t:T} = \pi],$$

where

$$R_t \equiv \sum_{t' \geq 0} \gamma^{t'} r_{t+t'}, \tag{7.2}$$

is the *reward-to-go* starting from time $t$. Assuming that we can compute $\mathcal{Q}_{\pi^*}$ for an optimal policy $\pi^*$, the optimal policy then becomes simply a policy that, given a history $h_t^\infty$, takes the action $a_t$ that maximizes $\mathcal{Q}_{\pi^*}(a_t, h_t^\infty)$. The optimal $\mathcal{Q}$ function can be computed using methods such as value iteration or temporal difference (Sutton and Barto, 1998). In a discrete setting with a small number of observations and actions, $\mathcal{Q}_\pi$ can be represented as a table. However, with high-dimensional and/or continuous systems, we typically resort to function approximation. Approximating value functions using neural networks is the essence of deep (recurrent) $\mathcal{Q}$-networks, which have been a major success in reinforcement learning (Mnih et al., 2013; Hausknecht and Stone, 2015).

### 7.1.2 Direct Policy Optimization Methods

One issue with value function-based methods is that they require solving the problem

$$\arg\max_{a_t \in \mathcal{A}} \mathcal{Q}_\pi(a_t, h_t^\infty),$$

which can be difficult for complicated value functions and very large (or infinite) action sets. A better alternative in these settings is direct policy optimization methods, which directly optimize

the policy parameters $\theta$ to maximize the objective function (7.1). While some of these methods use gradient-free approaches such as the cross-entropy method (Rubinstein, 1999; Szita and Lrincz, 2006), most of these methods apply a form of gradient descent. The key component in the latter type is the ability to compute a *policy gradient*—that is, a gradient of (7.1) w.r.t to the policy parameters $\theta$.

#### 7.1.2.1 Vanilla Policy Gradient

The simplest policy gradient method is REINFORCE (Williams, 1992), which is also known as vanilla policy gradient. Let $\tau$ denote a trajectory generated by the policy and $r(\tau) = \sum_t \gamma^{t-1} r_t$ denote the reward for that trajectory.

It follows from standard calculus that

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \int \nabla_\theta \Pr(\tau \mid \pi_\theta) r(\tau) d\tau \\
&= \int \Pr(\tau \mid \theta) \nabla_\theta \log \Pr(\tau \mid \pi_\theta) r(\tau) d\tau \\
&= \mathbb{E}\left[ \sum_t r(\tau) \nabla_\theta \log \Pr(a_t \mid h_t^\infty, \pi_\theta) \right]
\end{aligned} \tag{7.3}$$

An unbiased estimate of (7.3) can be obtained through the empirical average of a set of trajectories generated by $\pi_\theta$. Note that we only need to compute $\nabla_\theta \log \Pr(a_t \mid h_t^\infty, \pi_\theta)$, which is possible given the parametric form of the policy. One problem with plain REINFORCE is high variance. A variance reducing version of REINFORCE, which we refer to as variance reduced policy gradient (VRPG), computes a stochastic gradient as follows

$$\nabla_\theta J(\pi_\theta) \approx \sum_\tau \sum_t (R_t - b(h_t^\infty)) \nabla_\theta \log \Pr(a_t \mid h_t^\infty, \pi_\theta) \tag{7.4}$$

where $R_t$ is the reward to go as defined in (7.2) and $b$ is a *baseline* that evaluates the state $h_t^\infty$. It can be shown that (7.4) is equal to (7.3) in expectation regardless of the baseline function. However, a good choice of the baseline can significantly reduce the variance. A common practical choice of the baseline is an estimate of the state value function (Sutton et al., 2000; Duan et al., 2016)

$$V_{\pi_\theta}(h_t^\infty) = \mathbb{E}[R_t \mid h_t^\infty, \pi_\theta].$$

We show an example of estimating $V_\pi$ in Section 7.3.1.

#### 7.1.2.2 Trust Region Policy Optimization (TRPO)

Trust regeion policy optimization (Schulman et al., 2015) improves upon vanilla policy gradient in two aspects, which were shown to result on empirical improvement (Duan et al., 2016). First, it is uses a natural gradient update. Second, it enforces a constraint that limits the change in the policy

after each update. Specifically, each TRPO update is an approximate solution to the following constrained optimization problem.

$$\theta^{(n+1)} = \arg\min_\theta \mathbb{E}_\tau \left[ \frac{\pi_\theta(a_t \mid h_t^\infty)}{\pi_{\theta^{(n)}}(a_t \mid h_t^\infty)} (R_t - b(h_t^\infty)) \right]$$
$$\text{s.t. } \mathbb{E}_\tau \left[ \mathrm{KL}(\pi_{\theta^{(n)}}(. \mid h_t^\infty) \mid\mid \pi_\theta(. \mid h_t^\infty)) \right] \leq \delta_{\mathrm{KL}}, \tag{7.5}$$

where $\delta_{\mathrm{KL}}$ is a hyper-parameter that controls the amount of change allowed in each step. Like REINFORCE, TRPO works by generating a set of trajectories using the current policy $\pi_\theta^{(n)}$ and then replacing the expectations in (7.5) with empirical average. Solving (7.5) gives the updated policy parameters. Schulman et al. (2015) proposed an approximate algorithm to solve (7.5), which is depicted in Algorithm 7.

**Algorithm 7** Parameter update using TRPO (TRPOSTEP)

**Input:**Current parameters $\theta^{(n-1)}$, trajectories $\mathcal{D} = \{\tau^i\}_{i=1}^M$, TRPO step size $\delta$.
**Output:**Updated parameters $\theta^{(n)}$

1: Compute descent direction
$v := H^{-1} g$, (using conjugate gradient) where
$H = \nabla_\theta^2 \sum_{i=1}^{M} \mathrm{KL}(\pi_{\theta^{(n-1)}}(a_{i,t} \mid q_{i,t}) \mid\mid \pi_\theta(a_{i,t} \mid q_{i,t}),$
$g = \nabla_{\boldsymbol{\theta}_{\mathrm{re}}} \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T_i} \frac{\pi_\theta(a_{i,t} \mid q_{i,t})}{\pi_{\theta^{(n-1)}}(a_{i,t} \mid q_{i,t})} (R_t - b_t).$
2: Determine a step size $\eta$ through a line search on $v$ to maximize the objective in (7.5) while maintaining the constraint.
3: $\theta^{(n)} \leftarrow \theta^{(n-1)} + \eta v$

## 7.2 Recurrent Predictive State Policy (RPSP) Networks

We now introduce our proposed class of policies, Recurrent Predictive State Policies (RPSPs). An RPSP network consists of two components: (1) a recursive filter, implemented by an RFF-PSR (Chapter 6), which tracks the state of the environment and matinains a belief state $q_t$ that can be used to predict future observation $o_t$ and (2) a reactive policy that maps the belief state $q_t$ to a Gaussian distribution over actions $\mathcal{N}(\mu_t, \Sigma)$, where

$$\mu_t = \varphi(q_t; \theta_\mu)$$
$$\Sigma = \mathrm{diag}(\exp(\theta_\sigma))^2, \tag{7.6}$$

where $\varphi$ is a feedforward network parametrized by $\theta_\mu$ and $\theta_\sigma$ is a time independent vector that stores the log standard deviation of each coordinate of $a_t$. Figure 7.1 depicts the structure of RPSP networks. An RPSP is thus a stochastic recurrent policy with the recurrent part corresponding to an RFF-PSRand consequently we have two sets of parameters to learn:

- **RFF-PSR parameters $\theta_{\text{PSR}}$:** These include the initial state $q_1$, the system linear map $W_{\text{system}}$ and prediction operator $W_{\text{pred}}$.
- **Reactive policy parameters $\theta_{\text{re}}$:** These include the parameters of the mean function $\theta_\mu$ and the log standard deviation vector $\theta_\sigma$.

We describe the learning process in the following section.

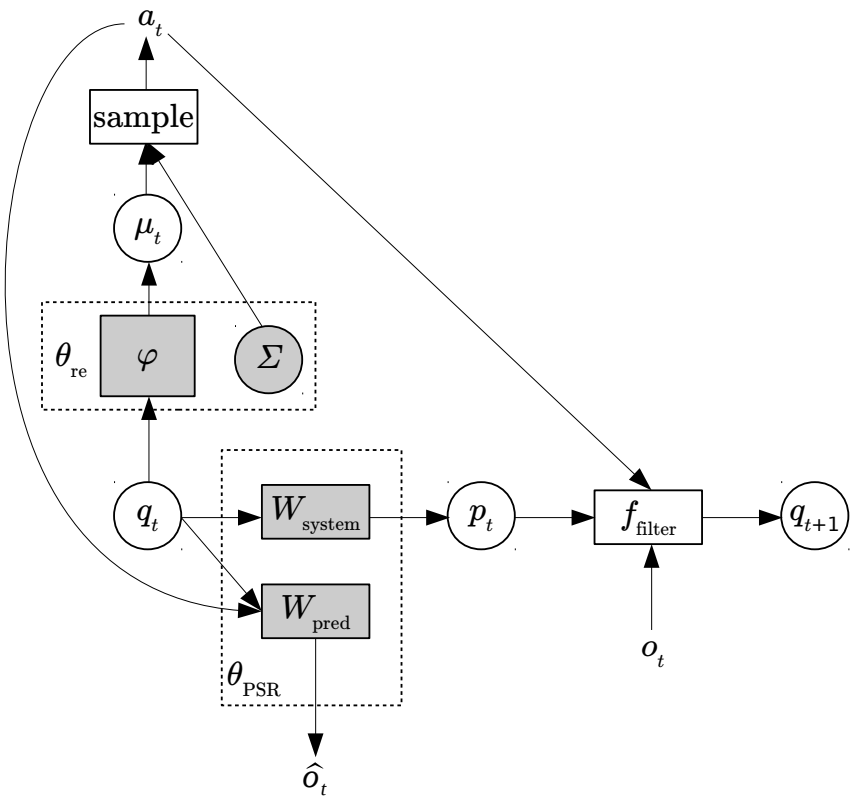



Figure 7.1: RPSP network: The predictive state is updated by a linear extension $W_{\text{system}}$ followed by a non-linear conditioning $f_{\text{filter}}$. A linear predictor $W_{\text{pred}}$ is used to predict observations, which is used to regularize training loss (see Section 7.3). A feed-forward reactive policy maps the predictive states $q_t$ to a Gaussians distribution over actions. Shaded nodes indicate learnable parameters.

## 7.3 Learning RPSP Networks

The learning process of an RPSP network consists of two phases: In the first phase, we execute a blind exploration policy to collect trajectories that are used to initialize the RFF-PSR parameters via the two-stage regression method (Algorithm 6). Optionally, we can obtain a better initialization by following the two-stage regression by local refinement via backpropagation through time to minimize observation prediction error, as described in Chapter 6.

The second phase starts with the initial RFF-PSR and a random reactive policy and uses a gradient-based technique to update both RFF-PSR parameters $\theta_{\text{PSR}}$ and the reactive policy parameters $\theta_{\text{re}}$. We seek to minimize the following loss function

$$\begin{aligned}\mathcal{L}(\theta) &= \alpha_1 l_1(\theta) + \alpha_2 l_2(\theta) \\ &= -\alpha_1 J(\pi_\theta) + \alpha_2 \sum_{t=1}^{T} \mathbb{E}_{\tau\sim\pi_\theta}\left[\|W_{\text{pred}}(q_t \otimes \phi^a(a_t)) - o_t\|^2\right] \end{aligned} \tag{7.7}$$

The loss function encourages high accumulated rewards, but it also encourages maintaining a good predictive model of the environment. The hyperparameters $\alpha_1, \alpha_2 > 0$ determine the relative importance of each criterion. Compared to reward signal, observations are typically richer. Therefore, having a secondary prediction task can be an effective regularizer (Venkatraman et al., 2017). The overall learning process is depicted in Algorithm 8.

**Algorithm 8** RPSP Network Learning

**Input:**Hyper-parameters $\eta$, exploration policy $\pi_{exp}$, Batch size $M$.
**Output:**$\theta = (\theta_{\text{PSR}}, \theta_{\text{re}})$.

Sample initial trajectories: $\{(o_t^i, a_t^i)_t\}_{i=1}^M$ from $\pi_{exp}$.
Initialize PSCM parameters $\theta_{\text{PSR}}^{(0)} = \{q_1, W_{\text{system}}, W_{\text{pred}}\}$ via 2-stage regression (Algorithm 6).
Initialize reactive policy parameters $\theta_{\text{re}}^{(0)}$ randomly.
**for** $n = 1 \dots N_{max}$ iterations **do**
    // Generate trajectories $\tau_1, \dots, \tau_m$ using $\pi_{\theta^{(n-1)}}$
    **for** $i = 1, \dots, M$: **do**
        Reset environment, Set $q_1$ to initial state.
        **for** $t = 1 \dots T$: **do**
            Execute $a_{i,t} \sim \pi_{re}^{(n-1)}(q_{i,t})$.
            Get observation $o_{i,t}$ and reward $r_{i,t}$.
            Filter $q_{i,t+1} = f_{\text{filter}}(W_{\text{system}} q_{i,t}, a_{i,t}, o_{i,t})$.
        **end for**
    **end for**
    $\mathcal{D} := \{(o_{i,t}, a_{i,t}, r_{i,t}, q_{i,t})\}_{1\le t\le T, 1\le i\le M}$
    $\theta^{(n)} :=$ UPDATE$(\theta^{(n)}, \mathcal{D}, \eta)$
**end for**

Algorithm 8 makes use of an UPDATE subroutine, which updates policy parameters based on collected trajectories. Since an RPSP network is a special type of a recurrent network policy, it is possible to adapt policy gradient methods to the joint loss in (7.7). In the following subsections, we propose two update variants.

### 7.3.1 Variance Reduced Policy Gradient (VRPG)

In this variant we use stochastic gradient descent on the loss function (7.7) by combining a stochastic gradient of the prediction loss with the variance reduced policy gradient (VRPG) of the reward

loss (7.4). Both gradients can be computed using backpropagation through time (BPTT). Recall that we need a baseline to compute the VRPG. Under the assumption that the belief state representation $q_t$ uses rich kernel-based features, we assume that

$$V_\pi(h_t^\infty) \approx w^\top q_t,$$

where $w$ is a weight vector that we re-estimate by regression using the latest batch of trajectories. The modified VRPG is depicted in Algorithm 9. The algorithm computes the stochastic gradient and then applies a gradient descent step (GRADIENTSTEP), which can utilize any variant of stochastic gradient descent. In our experiments, we use ADAM (Kingma and Ba, 2014).

**Algorithm 9** Parameter update using VRPG (VRPGUPDATE)

**Input:**Current parameters $\theta^{(n-1)}$, trajectories $\mathcal{D} = \{\tau^i\}_{i=1}^M$, step size $\eta$.
**Output:**Updated parameters $\theta^{(n)}$

1: Estimate baseline
$w := \arg\min_w \| \sum_{i=1}^{M} \sum_{t=1}^{T} R_{i,t} - w^\top q_{i,t} \|^2$
2: Compute the VRPG loss gradient w.r.t. $\theta$ using (7.4)
$\widehat{\nabla}_\theta l_1 := \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} \nabla_\theta \log \pi_{\mathrm{re}}^{(n-1)}(a_{i,t} \mid q_{i,t})(R_{i,t} - w^\top q_{i,t})$
3: Compute the prediction loss gradient
$\widehat{\nabla}_\theta l_2 := \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} \nabla_\theta \left\| W_{\mathrm{pred}}(q_{i,t} \otimes a_{i,t}) - o_{i,t} \right\|^2$
4: Compute joint loss gradient
$\widehat{\nabla}_{\boldsymbol{\theta}} \mathcal{L} := \alpha_1 \widehat{\nabla}_\theta l_1 + \alpha_2 \widehat{\nabla}_\theta l_2.$
5: Update policy parameters
$\theta^{(n)} := \textsc{GradientStep}(\theta^{(n-1)}, \widehat{\nabla}_\theta \mathcal{L}, \eta)$

### 7.3.2 Alternating VRPG/TRPO (ALTOPT)

While it is possible to extend the TRPO method described in Section 7.1.2.2 to the joint loss (7.7), we observed that TRPO tends to be computationally intensive with recurrent architectures. Instead, we resort to the following alternating optimization scheme: In each iteration we use a TRPO update (Algorithm 7) to update the reactive policy parameters $\theta_{\mathrm{re}}$, which involve only a feedforward network. Then, we use a VRPG update as described in Section 7.3.1 to update the RFF-PSR parameters $\theta_{\mathrm{PSR}}$.

### 7.3.3 Variance Normalization

Recall that the VRPG update makes use of hyperparameters $\alpha_1$ and $\alpha_2$ that determine the relative importance of reward loss vs. prediction loss. However, it is not intuitive to interpret the effect of specific values of $\alpha_1$ and $\alpha_2$, especially if the gradient magnitudes of their respective losses are not

comparable. For this reason we propose the following reparameterization of the relative weights: We use $\alpha_1 = \tilde{\alpha}_1$ and $\alpha_2 = c\tilde{\alpha}_2$, where $c$ is a user-given value, and $\tilde{\alpha}_1$ and $\tilde{\alpha}_2$ are dynamically adjusted to maintain the property that the gradient of each loss weighted by $\tilde{\alpha}$ has unit (uncentered) variance. In doing so we maintain the variance of the gradient of each loss through exponential averaging and use it to adjust the weights, as shown in (7.8)

$$
\begin{aligned}
v_k^{(n)} &= (1-\beta)v_k^{(n-1)} + \beta\|\nabla_\theta^{(n)} l_k\|^2, \\
\tilde{\alpha}_k^{(n)} &= \frac{1}{\sqrt{v_k^{(n)}}},
\end{aligned}
\tag{7.8}
$$

where $\beta \in [0, 1]$ is a hyperparamter that controls how fast we update variance estimates. We observed that it is best to update variance estimates slowly compared to parameters.

## 7.4 Experiments

In this section we report empirical evaluation of the proposed RPSP networks on reinforcement learning tasks in continuous environments.

### 7.4.1 Environments

For evaluation, we use a collection of reinforcement learning tasks based on OpenAI Gym environments that use the Mujoco simulator.[1] Specifically, we perform experiments on Swimmer, Walker, Hopper and CartPole environments, which are shown in Figure 7.2. We ensure all models and baselines use the same OpenAI Gym base environment settings. In each environment, actions correspond to torques applied to joints in a link-joint model and observations correspond to positions of the joints.[2] For Swimmer, Walker and Hopper, the reward is determined by the forward velocity minus a penalty that depends on the amount of torque exerted. For Cart-Pole, a fixed reward of 1 is given at each time step until the episode terminates. The episode terminates when the tilting of the robot exceeds a certain threshold (except for Swimmer) or the episodes times out, which happens after a certain number of timesteps (see Table 7.1).

### 7.4.2 Proposed Models

We evaluate RPSP networks using an RFF-PSR recursive filter and a feedforward reactive policy. For the RFF-PSR we use 1000 random Fourier features on observation and action sequences followed by randomized PCA projection to $d$ dimensions as described in Chapter 6.

[1] `https://gym.openai.com/envs#mujoco`

[2] By default, Mujoco also reveals the velocities of the joints. We discard this information to make the environment partially observable.

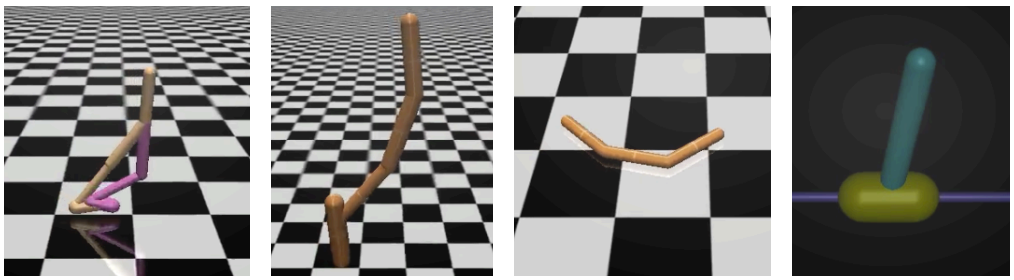


Figure 7.2: OpenAI Gym Mujoco environments. From left to right: Walker, Hopper, Swimmer, CartPole

For initialization, we use a blind policy that samples actions from a uniform distribution to collect initialization trajectories. These trajectories are used to initialize the RFF-PSR component using two-stage regression followed by 5 iterations of backpropagation through time.

The reactive policy contains one hidden layer of ReLU units and linear output units. We experiment with both a basic policy that takes the belief state $q_t$ as input and also an extended policy that takes $q_t$ together with $o_{t-2:t-1}$.

The reactive policy is randomly initialized. After initialization, we optimize the entire network using VRPG (**RPSP-VRPG**) or alternating optimization (**RPSP-Alt**). We use variance normalization with the hyperparameter $\beta$ set to 0.1.

### 7.4.3 Competing Models

We compare our models to finite memory models (**FM**). The finite memory models use a feed-forward policy on a window of past observations as inputs. We tried three variants, FM1, FM2 and FM5 with window size of 1, 2 and 5 respectively. We also compared to recurrent policies implemented by gated recurrent units (**GRU**). We experimented with $16, 32, 64$ and $128$-dimensional hidden states. We optimized network parameters using the *RLLab*[3] implementation of TRPO with two different learning rates ($\eta = 10^{-2}, 10^{-3}$).

In each model, we use a linear baseline for variance reduction where the state of the model (i.e. past observation window for FM, latent state for GRU and belief state (which may include previous observations) for RPSP) is used as the predictor variable. We also follow the common practice of using a discount factor $\gamma < 1$ even though our main criterion is the total (non-discounted) reward (Duan et al., 2016; Venkatraman et al., 2017). We observed that the use of a discount factor stabilizes the training and improves the total reward. We set $\gamma = 0.99$.

### 7.4.4 Evaluation Procedure

We run each algorithm for a fixed number of iterations based on the environment. Each iteration of the optimization algorithm uses a batch of trajectories. Since most test environments can termi-

[3] `https://github.com/openai/rllab`

| Hyper-parameter | Swimmer | Walker | Hopper | Cart-Pole |
|---|---|---|---|---|
| Experience per iteration | 5000 | 10000 | 10000 | 2000 |
| Maximum trajectory length | 500 | 1000 | 1000 | 200 |
| Step size | $10^{-2}$ | $10^{-2}$ | $10^{-2}$ | $10^{-2}$ |
| Append observation to state? | ✓ | ✓ | × | × |
| RFF-PSR state size | 10 | 30 | 30 | 20 |
| Reactive policy hidden layer size | 16 | 16 | 16 | 16 |
| GRU state size | 32 | 32 | 128 | 32 |
| Finite memory length | 2 | 5 | 1 | 2 |

Table 7.1: Best hyper-parameter settings for each environment

nate at different times, we follow Duan et al. (2016) and fix the total number of time steps (a.k.a experience) in the batch, rather than the number of trajectories.

We are interested in the average return after each iteration

$$\bar{R}_i = \frac{1}{M_i} \sum_{m=1}^{M_i} \sum_{t=1}^{T_m} r_{m,t},$$

where $M_i$ is the number of trajectories in the $i^{th}$ iteration and $T_m$ is the length of the $m^{th}$ trajectory. We also interested in the mean average return across iterations

$$C = \sum_i \bar{R}_i,$$

which we will refer to as *area under curve* since it can be thought of as the area under the average return vs. iteration curve. The area under curve favors methods that converge fast to a policy that achieves high average return. We compute the evaluation metrics for 10 trials each using a different random seed.

### 7.4.5 Results

Figure 7.3 shows the mean and standard error of the average return in each iteration across 10 trials. For ease of exposition, we report the best setting of each method in terms of area under curve (see Table 7.1). We can see that in terms of the final average return, one of our models (RPSP-VRPG or RPSP-Alt) closely matches or outperforms other competing methods.

Table 7.2 displays the cumulative average return for the compared methods. RPSP-VRPG and RPSP-Alt are the top performers in Swimmer and Hopper environments, respectively. In Walker environment, RPSP-Alt is close to FM. It is only in the Cart-Pole environment that both RPSP models are at clear disadvantage.

### 7.4.6 Ablation Study (Analyzing Contributions)

RPSP networks employ four key ingredients:

| Model | Swimmer | Walker | Hopper | CartPole |
|---|---|---|---|---|
| RPSP-VRPG | $\mathbf{88.58 \pm 5.56}$ | $564.48 \pm 40.37$ | $597.36 \pm 20.14$ | $117.98 \pm 10.79$ |
| RPSP-Alt | $68.89 \pm 2.19$ | $673.55 \pm 25.05$ | $\mathbf{598.00 \pm 10.00}$ | $106.20 \pm 4.82$ |
| GRU | $43.93 \pm 3.97$ | $414.09 \pm 25.90$ | $558.93 \pm 17.99$ | $\mathbf{146.22 \pm 11.99}$ |
| FM | $83.29 \pm 7.00$ | $\mathbf{675.79 \pm 23.77}$ | $536.62 \pm 9.33$ | $63.47 \pm 3.24$ |

Table 7.2: Mean average return (area under curve) for proposed and competing models in four different Mujoco enviornments. Table shows mean and standard error accross 10 trials.

1. Using a predictive state controlled model (RFF-PSR) as a recurrent state tracker.
2. Ability to initialize the recurrent component using two-stage regression.
3. Joint end-to-end training of a recursive filter and a reactive policy.
4. A regularized objective that penalizes errors in predicting future observations.

How important is each ingredient? To answer this we conduct two sets of experiments. In the first experiment, we compare RPSP networks to GRUs that are trained on the joint loss (7.7) (**GRU-Reg**). These are analogous to predictive state decoders proposed by Venkatraman et al. (2017). These models use the same loss function as RPSP networks but differ in the type of the recurrent component used.

In the second set of experiments, we compare RPSP networks to models that are stripped of one of the ingredients. Specifically, we compare to

- A fixed filter RPSP network (**RPSP-Fix**): An RPSP network that is initialized using two-stage regression. However, only the reactive policy is trained after initialization. Thus, it lacks the end-to-end joint training.
- A randomly initialized RPSP network (**RPSP-Rnd**): An RPSP network that lacks two-stage regression initialization.
- A non-regularized RPSP network (**RPSP-NoReg**): An RPSP network that is trained to optimize the non-regularized objective (7.1) and hence lacks predictive regularization.

The results of these experiments are shown in Figures 7.4 and 7.5 respectively. The results show that removing any of the four key ingredients of RPSP networks results in a policy that is inferior in most environments in terms of final return and/or area under curve.

## 7.5 Related Work

We presented a new class of recurrent policies based on predictive state controlled models for partially observable environments. There have been previous attempts to combine predictive state models and/or method of moments with policy learning. Boots et al. (2011) proposed a method for planning in partially observable environments. The method first learns a PSR from a set of trajectories collected using an explorative blind policy. The predictive states estimated by the PSR are then considered as states in a fully observable Markov Decision Process. A value function is learned on these states using least squares temporal difference (Boots and Gordon, 2010) or point-

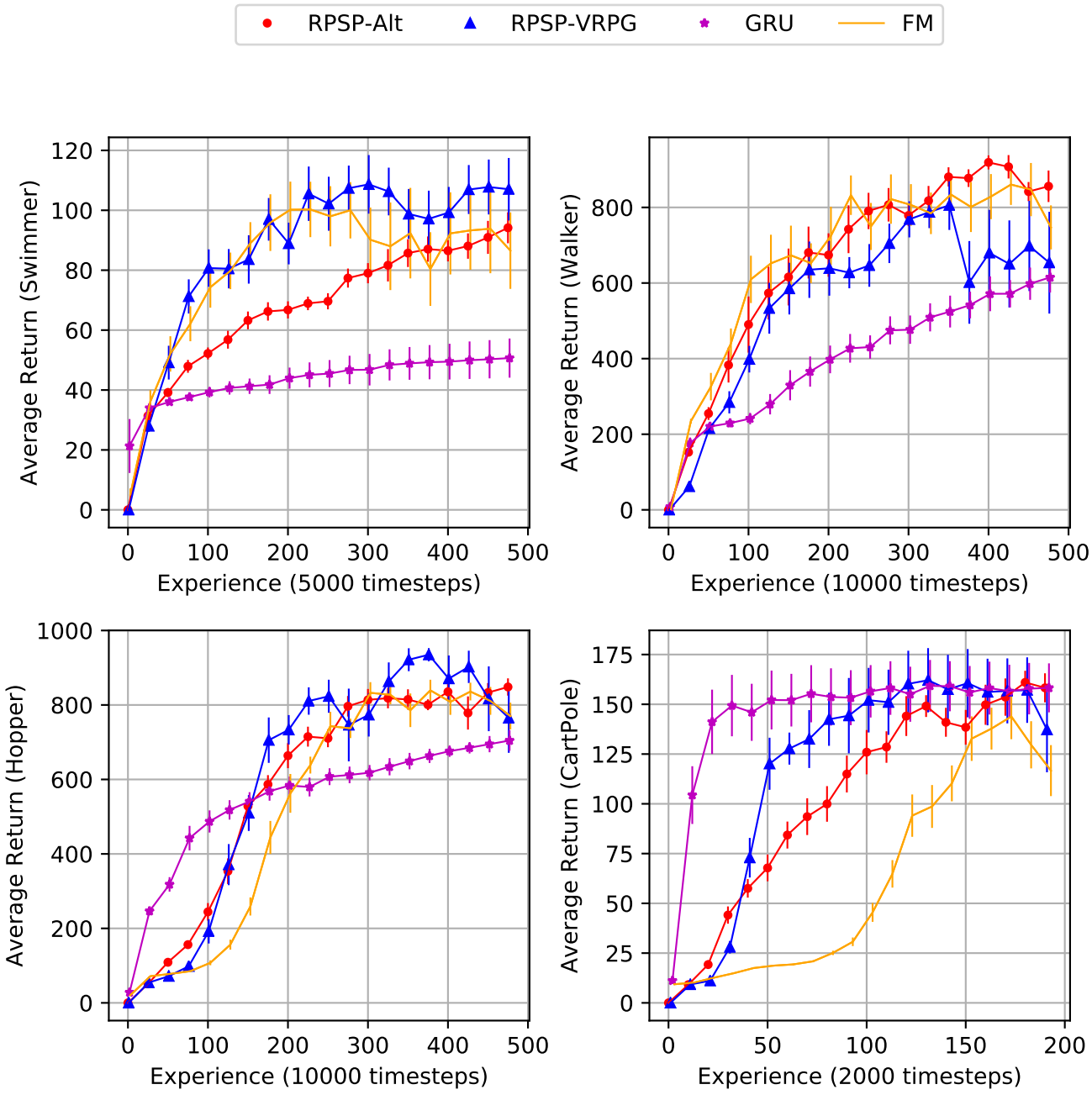



Figure 7.3: Mean and standard error of empirical average return over 10 trials. Each point indicates the total reward in a trajectory averaged over all trajectories in the batch. RPSP graphs are shifted to the right to reflect the use of extra trajectories for initialization.

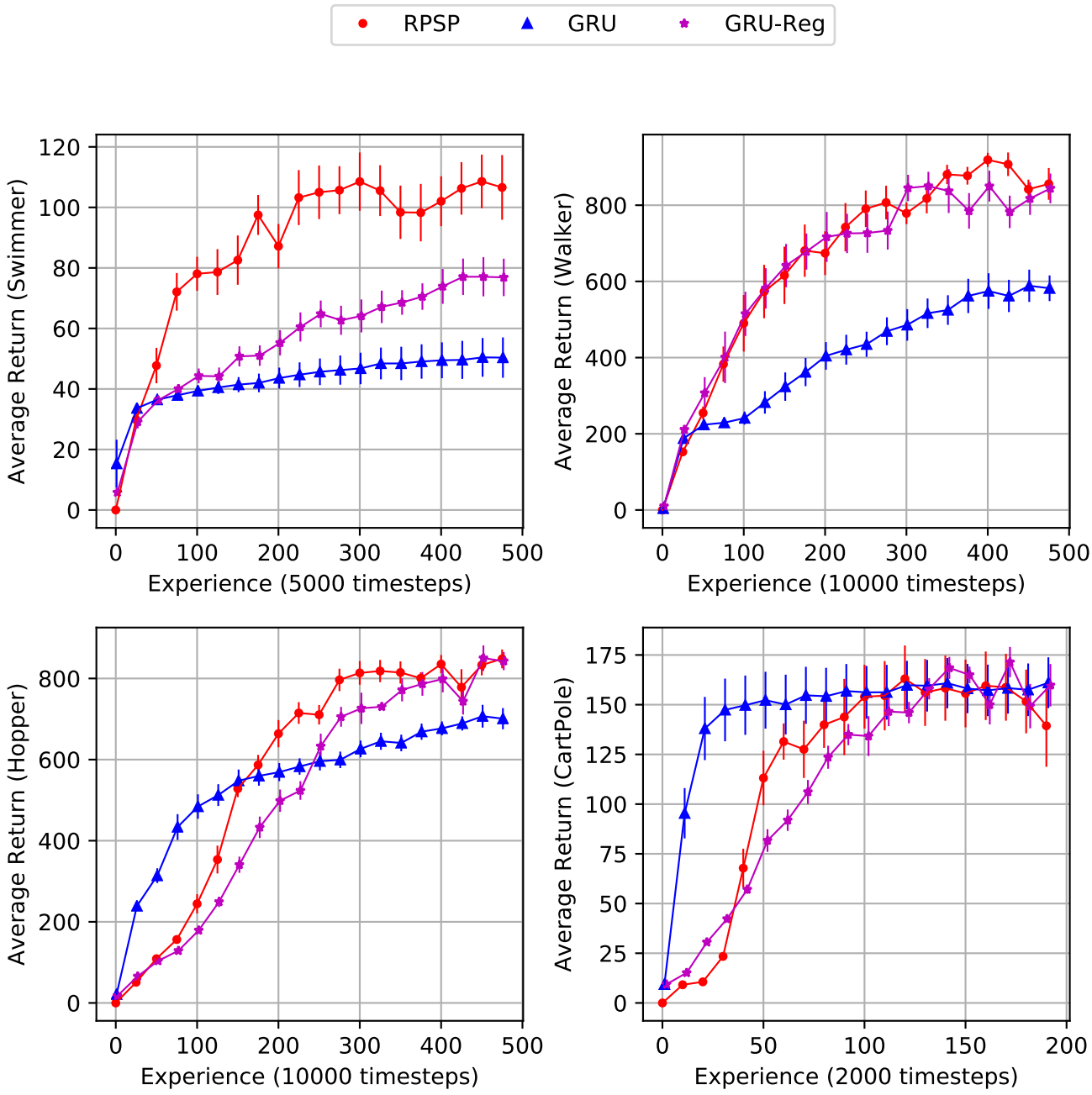



Figure 7.4: Mean and standard error of eimpircal average return over 10 trials. This experiment tests replacing the RFF-PSRcomponent with a GRU.

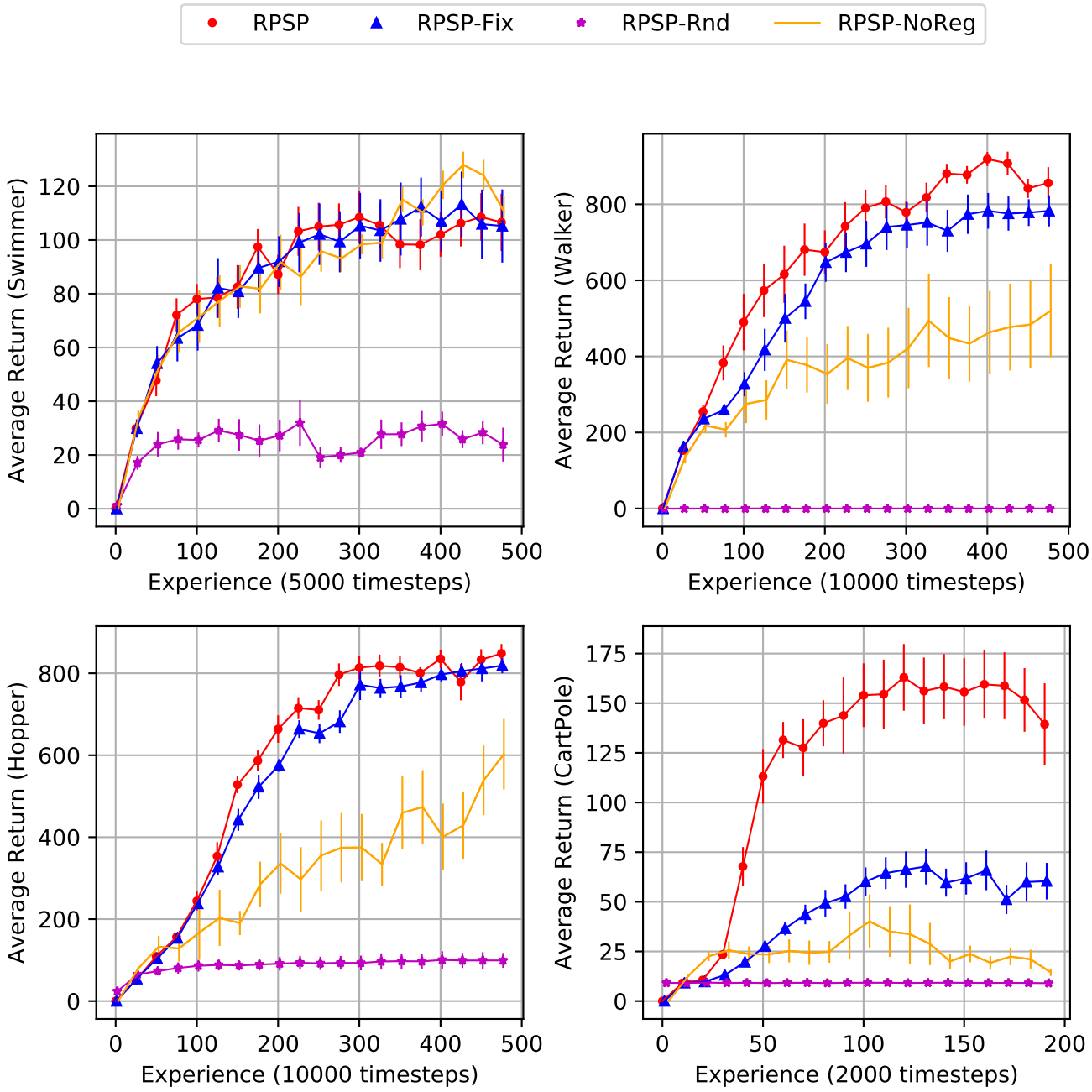



Figure 7.5: Mean and standard error of eimpircal average return over 10 trials. This experiment tests different variations of RPSP networks.

based value iteration (PBVI) (Boots et al., 2011). The main disadvantage of these approaches is that they assume a one-time initialization of the PSR and do not propose a mechanism to update the model based on subsequent experience.

Hamilton et al. (2014) proposed an iterative method to simultaneously learn a PSR and use the predictive states to fit a $\mathcal{Q}$-function. Azizzadenesheli et al. (2016) proposed a tensor decomposition method to estimate the parameters of a discrete partially observable Markov decision process (POMDP) and used concentration inequalities to choose actions that maximize an upper confidence bound of the reward. This method does not employ a PSR, however it uses a consistent moment-based method to estimate model parameters. One common limitation in the aforementioned methods is that they are restricted to discrete actions (some even assume discrete observations). Also, we have demonstrated in Chapters 4 and 6 that PSRs can benefit greatly from local optimization after a moment-based initialization.

Venkatraman et al. (2017) proposed predictive state decoders, where an LSTM or a GRU network is trained on a mixed objective function in order to obtain high cumulative rewards while accurately predicting future observations. While this method has shown improvement over using standard training objective functions, it does not solve the initialization issue of the recurrent network.

Our proposed RPSP networks alleviate the limitations of previous approaches: They support continuous observations and actions, use a recurrent state tracker with consistent initialization, and support end-to-end training after the initialization.

## 7.6 Conclusion

In this chapter, we employed predictive state controlled models in a control task. By chaining the belief state of the PSCM model to a reactive policy, we obtain RPSP networks, a class of recurrent policies that can be trained via reinforcement learning while maintaining the advantage of method-of-moment based initialization.

RPSP networks have four main characteristics: the use of PSCM as a filter, two-stage regression initialization, end-to-end training and regularization by prediction error. We proposed policy gradient-based methods to train RPSP networks and demonstrated their superiority to other methods. We also demonstrated that all the four charateristics of RPSP networks are crucial to achieve the best performance.

# Part IV

# Conclusion

# Chapter 8

# Conclusions

## 8.1 Summary of Contributions

In this thesis we studied the problem of state tracking in partially observable environments through recursive filter. The foundation of our work is a framework— that is, a model class and a learning algorithm for constructing recursive filters from data in an unsupervised manner. The framework, predictive state models, is based on representing the belief state of the filter in terms of observed future statistics, thus eliminating the need to learn an observation model. The learning algorithm, two-stage regression, uses history features to remove noise from observed future statistics, turning them into denoised (expected) states from which we can learn the dynamics of the system. Two-stage regression reduced the unsupervised learning problem of learning the system dynamics to a set of supervised learning problems. However, it allows additional flexibility in the choice of the supervised learning methods to be used compared to the pre-existing "subspace identification" algorithms, which were restricted to linear regression. We have shown that this flexibility allows us to design novel recursive filters that outperform subspace identification counterparts.

We then extended the proposed framework to controlled systems. A key change in controlled systems is that the predictive state encodes a conditional distribution of future observations given future actions. We showed how to adapt two-stage regression to this subtlety. A concrete model resulting form this effort is the predictive state controlled model with random Fourier features (RFF-PSR), which has the flexibility to model non-linear dynamical systems while having an efficient and local optima-free initialization algorithm. We demonstrate the efficacy of this model in two settings: a prediction setting where the goal is to predict future observations given future actions, and a reinforcement learning setting where the goal is to optimize the policy that uses predictive state as inputs to maximize cumulative reward.

In stating the thesis goal we mentioned a number of qualities that we care about when constructing recursive filters. In Table 8.1, we show where our proposed framework stands in terms of these qualities.

| Quality | Our Contribution |
|---|---|
| Controlled Systems | - Predictive state controlled models can represent controlled systems. A two-stage regression algorithm permits learning from blind policies.<br>- Local refinement can work with non-blind policies.<br>- Applied predictive state controlled models within reinforcement learning. |
| Scalability | - Through the use of kernel approximations and randomized SVD, we proposed methods that scale linearly with the number of examples. We also demonstrated the use of sketches to obtain low-rank approximations for models with large state sizes. |
| Modeling Flexibility | - We demonstrated the ability to model highly non-linear continuous systems with high-dimensional observations.<br>- We demonstrated the benefit of the ability to choose regression models and regularization schemes.<br>- We made use of other paradigms such as kernel methods, recurrent networks and sketching to increase the expressive power and efficacy of our models. |
| Theoretical Guarantees | - Asymptotic and finite sample guarantees for identifying system parameters.<br>- Guarantees on remaining close to the correct state subspace.<br>- Guarantees for recovering the parameters of a controlled dynamical system. |

Table 8.1: Thesis contributions and how they map to desired qualities of recursive filters mentioned in Chapter 1.

## 8.2 Future Directions

In this section we list a number of potential directions for follow-up work.

### 8.2.1 Two-stage Regression with Non-blind Policies

We demonstrated in Chapter 7 that we can deal with data collected from a non-blind policy to optimize an RFF-PSR policy. However, can we use data collected from a non-blind policy to initialize the parameters of an RFF-PSR? In other words, can we modify the two-stage regression approach to be compatible with non-blind policies? Note that with correct estimates of conditional states (i.e. correct S1 regression output), the fact that the policy is non-blind becomes of little concern. Therefore, modifying S1 regression appears to be one potential strategy to deal with this issue.

The problem of non-blind policies has different variants depending on our knowledge of the

policy. We might have zero knowledge about the policy other than the fact that it is non-blind. We might have knowledge of the probability of each action executed in the trajectory, which may permit the use of importance sampling approaches (Bowling et al., 2006). We might have full access to the policy which allows us to compute the joint observation/action probability as a function of the policy and the state and exploit that in stage 1 regression.

One of the useful state representations to deal with non-blind policies is the factored state representation (Bowling et al., 2006), where we represent the belief state as a set of distributions $P(o_{t+i} \mid a_{t:t+i}, o_{t:t+i-1}; h_t^\infty)$. Having tensors with modes corresponding to individual observation or action can greatly simply state updates by simplyfing the process of conditioning on individual observations and actions. This warrants further inverstigation of how to harness the power of tensor sketching or other compression schemes.

It is worth mentioning that the current framework can be used in the following scenario: Assume we have a roll-in policy $\pi_{\text{in}}$ and roll-out policy $\pi_{\text{out}}$. We collect training by rolling-in using $\pi_{\text{in}}$ to generate a history $h_t^\infty$ and then rolling-out using $\pi_{\text{out}}$ to generate an extended future. For learning to be consistent in a realizable setting (i.e. no model mismatch), it is only required that $\pi_{\text{out}}$ has no access to future observations (it can access history observations). Such an approach, however, is sample inefficient as it cannot use the roll-in samples for two-stage regression.

### 8.2.2 Model Uncertainty

Being able to quantify prediction uncertainty is extremely valuable. For example predicting a confidence interval of the reward can be used to aid exploration (Azizzadenesheli et al., 2016). There are two sources of uncertainty, inherent noise in the data and uncertainty of model parameters due to estimation from finite data. In its current form, a predictive state model can deal with first kind; we can train a predictor to map the belief state $q_t$ to the desired output statistics (e.g., second moment). To provide a full characterization of uncertainty, we need efficient methods that take model uncertainty into account.

### 8.2.3 From Parameter Recovery to Filtering Guarantees

We presented in Chapter 3 theoretical guarantees about parameter recovery. However, it is not clear how these guarantees translate into guarantees on filtering performance (i.e. error in the predictive state). We know that in the realizable setting, consistency in parameter estimation results in consistency in filtering. We also know that we cannot make a general statement about the filtering performance in the non-realizable setting since there are already examples where the best linear predictor can produce arbitrarily bad predictions (Kulesza et al., 2014). It is worth investigating what we can claim between these two extremes.

### 8.2.4 Online Learning

There is a large body of literature on online learning of models that use predictive state in both the prediction setting (Venkatraman et al., 2016; Boots and Gordon, 2011) and reinforcement learning setting (Ong et al., 2011). These methods, however, effectively use online method of moments to

learn a predictive state model with ordinary least squares regression model. We have observed that for some feature representations, regularization via ridge regression can helpful for model stability. Also, we have shown that local refinement after method of moments can significantly improve the model. How to update a predictive state (controlled) model in an efficient and provably consistent way that maintains the advantages of regularization and local refinement is an interesting open problem.

# Bibliography


Alpaydin, E. and Alimoglu, F. (1998). Pen-based recognition of handwritten digits data set. *University of California, Irvine, Machine Learning Repository. Irvine: University of California*. 4.4.2

Anandkumar, A., Foster, D. P., Hsu, D. J., Kakade, S. M., and Liu, Y. (2012). Two svds suffice: Spectral decompositions for probabilistic topic modeling and latent dirichlet allocation. *CoRR*, abs/1204.6703. 2.4.2.2

Anandkumar, A., Ge, R., Hsu, D., Kakade, S. M., and Telgarsky, M. (2014a). Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research*, 15(1):2773–2832. 2.4.2.2, 2.4.3.2, 5.1.1

Anandkumar, A., Ge, R., and Janzamin, M. (2014b). Guaranteed non-orthogonal tensor decomposition via alternating rank-1 updates. *CoRR*, abs/1402.5180. 2.4.2.2

Anderson, B. and Moore, J. (1979). *Optimal Filtering*. Prentice-Hall, Englewood Cliffs, NJ. 2.4.4

Azizzadenesheli, K., Lazaric, A., and Anandkumar, A. (2016). Reinforcement learning of pomdp's using spectral methods. *CoRR*, abs/1602.07764. 7.5, 8.2.2

Ba, L. J., Kiros, R., and Hinton, G. E. (2016). Layer normalization. *CoRR*, abs/1607.06450. 4.3.3

Balle, B., Hamilton, W. L., and Pineau, J. (2014). Methods of moments for learning stochastic languages: Unified presentation and empirical comparison. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1386–1394. 3, 2.4.3.2, 2.4.3.3, 3.6

Banerjee, A., Guo, X., and Wang, H. (2005). On the optimality of conditional expectation as a bregman predictor. *IEEE Trans. Inf. Theor.*, 51(7):2664–2669. 3.2

Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Ann. Math. Statist.*, 41(1):164–171. 2.4.3.1

Belanger, D. and Kakade, S. (2015). A linear dynamical system model for text. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 833–842, Lille, France. PMLR. 2.4.4.1, 4.3.2

Bengio, Y. and Frasconi, P. (1995). An input output HMM architecture. In *NIPS*. 6.1.2, 6.2

Boots, B. (2012). *Spectral Approaches to Learning Predictive Representations*. PhD thesis,

Carnegie Mellon University. 2.4.2.1, 2.4.4.2, 2.4, 3.3.2, 3.6

Boots, B. and Fox, D. (2013). Learning dynamic policies from demonstration. *NIPS Workshop on Advances in Machine Learning for Sensorimotor Control*. 6.4.1

Boots, B. and Gordon, G. (2011). An online spectral learning algorithm for partially observable nonlinear dynamical systems. In *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI-2011)*. 3.6, 8.2.4

Boots, B. and Gordon, G. (2012). Two-manifold problems with applications to nonlinear system identification. In *Proc. 29th Intl. Conf. on Machine Learning (ICML)*. 2.1.3, 3.6

Boots, B. and Gordon, G. J. (2010). Predictive state temporal difference learning. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010.*, pages 271–279. 7.5

Boots, B., Gretton, A., and Gordon, G. J. (2013). Hilbert Space Embeddings of Predictive State Representations. In *Proc. 29th Intl. Conf. on Uncertainty in Artificial Intelligence (UAI)*. 3.4, 4, 4.1.4, 4.2, 4.A, 4.A, 6.2, 6.3.1, 6.4, 6.4.1, 6.4.1, 6.4.2.1, 6.5.1, 6.5.5

Boots, B., Siddiqi, S., and Gordon, G. (2011). Closing the learning planning loop with predictive state representations. In *I. J. Robotic Research*, volume 30, pages 954–956. 3.6, 4, 7.5

Bowling, M., McCracken, P., James, M., Neufeld, J., and Wilkinson, D. (2006). Learning predictive state representations using non-blind policies. In *ICML*. 4, 8.2.1

Carlyle, J. and Paz, A. (1971). Realizations by stochastic finite automata. *Journal of Computer and System Sciences*, 5(1):26 – 40. 2.1.3

Celeux, G. and Diebolt, J. (1985). The SEM algorithm: A probabilistic teacher algorithm derived from the EM algorithm for the mixture problem. *Computational Statistics Quarterly*, 2:73–82. 2.4.1.2

Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics. 2.1.2, 4.4.1, 4.4.2

Corbett, A. T. and Anderson, J. R. (1995). Knowledge tracing: Modelling the acquisition of procedural knowledge. *User Model. User-Adapt. Interact.*, 4(4):253–278. 3.5.1

Cormode, G. and Hadjieleftheriou, M. (2008). Finding frequent items in data streams. *Proc. VLDB Endow.*, 1(2):1530–1541. 5.1.2

Darolles, S., Fan, Y., Florens, J. P., and Renault, E. (2011). Nonparametric instrumental regression. *Econometrica*, 79(5):1541–1565. 3.6

Dempster, A. P., Laird, M. N., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 39:1–22. 2.4.1.2

Downey, C., Hefny, A., Boots, B., Gordon, G. J., and Li, B. (2017). Predictive state recurrent neural

networks. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 6055–6066. Curran Associates, Inc. 1.4

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1329–1338. 7.1.2.1, 7.1.2.2, 7.4.3, 7.4.4

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211. 2.1.2, 4.4.2

Falakmasir, M. H., Pardos, Z. A., Gordon, G. J., and Brusilovsky, P. (2013). A spectral learning approach to knowledge tracing. In *6th International Conference on Educational Data Mining (EDM 2013)*., pages 28–35. International Educational Data Mining Society. 2.4.3.3, 4.3.2

Foti, N., Xu, J., Laird, D., and Fox, E. (2014). Stochastic variational inference for hidden markov models. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3599–3607. Curran Associates, Inc. 2.4

Fruhwirth-Schnatter, S. (2001). Markov chain monte carlo estimation of classical and dynamic switching and mixture models. *Journal of the American Statistical Association*, 96:194–209. 2.4

Fukumizu, K., Song, L., and Gretton, A. (2013). Kernel bayes' rule: Bayesian inference with positive definite kernels. *Journal of Machine Learning Research*, 14(1):3753–3783. 4.1.3, 2, 4.A, 4.A, 6.3.1, 6.4.1

Geramifard, A., Klein, R. H., Dann, C., Dabney, W., and How, J. P. (2013). RLPy: The Reinforcement Learning Library for Education and Research. http://acl.mit.edu/RLPy. 4.4.2, 6.5.3

Ghaharmani, Z. and Hinton, G. E. (1996). Parameter estimation for linear dynamical systems. Technical Report CRG-TR-96-2, Department of Computer Science, University of Toronto. 2.4.4.1

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 249–256. 4.4.2

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc. 7

Grünewälder, S., Lever, G., Baldassarre, L., Patterson, S., Gretton, A., and Pontil, M. (2012). Conditional mean embeddings as regressors. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1823–1830, New York, NY, USA. Omnipress. 4.1.4

Halko, N., Martinsson, P. G., and Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–

288. 6.4.3.1

Hall, P. and Horowitz, J. L. (2005). Nonparametric methods for inference in the presence of instrumental variables. *The Annals of Statistics*, 33(6):2904–2929. 3.6

Hamilton, W., Fard, M. M., and Pineau, J. (2014). Efficient learning and planning with compressed predictive states. *J. Mach. Learn. Res.*, 15(1):3395–3439. 7.5

Harshman, R. A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16(1):84. 2.4.2.2

Hausknecht, M. J. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527. 7.1.1

Hefny, A., Downey, C., and Gordon, G. J. (2015). Supervised learning for dynamical system learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 1963–1971, Cambridge, MA, USA. MIT Press. 1.4

Hefny, A., Downey, C., and Gordon, G. J. (2018a). An efficient, expressive and local minima-free method for learning controlled dynamical systems. In *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*. 1.4

Hefny, A., Marinho, Z., Sun, W., Srinivasa, S., and Gordon, G. (2018b). Recurrent predictive state policy networks. *CoRR*, abs/1803.01489 [Submitted to ICML 2018]. 1.4

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780. 2.1.2, 4.4.1, 4.4.2

Hsu, D. and Kakade, S. M. (2013). Learning mixtures of spherical gaussians: Moment methods and spectral decompositions. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 11–20, New York, NY, USA. ACM. 2.4.2.2

Hsu, D., Kakade, S. M., and Zhang, T. (2009). A spectral algorithm for learning hidden markov models. In *COLT*. 2.4.3.2, 3.1, 3.3.1, 3.3.1, 5, 3.3.1, 3.3.1, 3.4, 3.5.1.2, 3.6

Hsu, D., Kakade, S. M., and Zhang, T. (2012a). Random design analysis of ridge regression. In *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*, pages 9.1–9.24. 3.4.1

Hsu, D., Kakade, S. M., and Zhang, T. (2012b). Tail inequalities for sums of random matrices that depend on the intrinsic dimension. *Electronic Communications in Probability*, 17(14):1–13. 3.10

Jaeger, H. (1998). Discrete-time, discrete-valued observable operator models: a tutorial. Technical Report GMD 42, German National Research Center for Information Technology (GMD). 6.1.2

Jaeger, H. (2000). Observable Operator Models for Discrete Stochastic Time Series. *Neural Computation*, 12(6):1371–1398. 1.2, 2.1.3, 4, 9

Jiang, N., Kulesza, A., and Singh, S. P. (2016). Improving predictive state representations via gradient descent. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 1709–1715. 4.3.2

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45. 2.3.1, 2.4.4, 4.4.2

Kandasamy, K., Al-Shedivat, M., and Xing, E. P. (2016). Learning hmms with nonparametric emissions via spectral decompositions of continuous matrices. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 2865–2873. Curran Associates, Inc. 2.1.3

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980. 7.3.1

Koedinger, K. R., Baker, R. S. J., Cunningham, K., Skogsholm, A., Leber, B., and Stamper, J. (2010). A data repository for the EDM community: The PSLC DataShop. *Handbook of Educational Data Mining*, pages 43–55. 3.5.1.1

Kruskal, J. (1977). Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear Algebra Appl.*, 18(2):95–138. 2.4.2.2

Kuleshov, V., Chaganty, A. T., and Liang, P. (2015). Tensor factorization via matrix factorization. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*. 2.4.2.2

Kulesza, A., Rao, N. R., and Singh, S. (2014). Low-Rank Spectral Learning. In Kaski, S. and Corander, J., editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 522–530, Reykjavik, Iceland. PMLR. 2.5.2, 8.2.3

Langford, J., Salakhutdinov, R., and Zhang, T. (2009). Learning nonlinear dynamic models. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, pages 593–600. 2.1.2, 2.5.2, 3.6

Laub, A. J. (1978). A schur method for solving algebraic riccati equations. Technical Report LIDS-R; 859, Massachusetts Institute of Technology. Laboratory for Information and Decision Systems. 2.4.4.2

Levinson, S., Rabiner, L., and Sondhi, M. (1982). An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *The Bell System Technical Journal*, 62:1035–1074. 2.4.3.1

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330. 4.4.1

Micchelli, C. A. and Pontil, M. (2005). On learning vector-valued functions. *Neural Computation*, 17(1):177–204. 1

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013. 7.1.1

Ong, S. C. W., Grinberg, Y., and Pineau, J. (2011). Goal-directed online learning of predictive models. In *Recent Advances in Reinforcement Learning - 9th European Workshop, EWRL 2011, Athens, Greece, September 9-11, 2011, Revised Selected Papers*, pages 18–29. 8.2.4

Overschee, P. V. and Moor, B. D. (1993). Subspace algorithms for the stochastic identification problem, . *Automatica*, 29(3):649–660. 2.4.4.2

Pandit, S. and Wu, S. (1983). *Time series and system analysis, with applications*. Wiley. 3.6

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York, NY, USA. 3.2

Pham, N. and Pagh, R. (2013). Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 239–247, New York, NY, USA. ACM. 5.1.2

Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4 –16. 2.4.3

Rabiner, L. R. (1990). Readings in speech recognition. chapter A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 2.1.1

Rahimi, A. and Recht, B. (2008). Random features for large-scale kernel machines. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *Advances in Neural Information Processing Systems 20*, pages 1177–1184. Curran Associates, Inc. 4.1.5, 4.1.5, 4.1.5, 6.4, 6.4.3.1

Rahimi, A. and Recht, B. (2009). Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21*, pages 1313–1320. Curran Associates, Inc. 4.1.5

Rosencrantz, M., Gordon, G., and Thrun, S. (2004). Learning low dimensional predictive representations. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 88–, New York, NY, USA. ACM. 2.3, 2.3.2

Ross, S., Gordon, G. J., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, pages 627–635. 2.5.2

Rubinstein, R. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology And Computing In Applied Probability*, 1(2):127–190. 7.1.2

Rudi, A. and Rosasco, L. (2017). Generalization properties of learning with random features. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 3215–3225. Curran Associates, Inc. 4.1.5

Rudin, W. (2017). *Fourier Analysis on Groups*. Dover Books on Mathematics. Dover Publications. 4.1.5

Rydn, T. (2008). Em versus markov chain monte carlo for estimation of hidden markov models: a computational perspective. *Bayesian Anal.*, 3(4):659–688. 2.4

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In Blei, D. and Bach, F., editors, *Proceedings of the 32nd International Conference on*

*Machine Learning (ICML-15)*, pages 1889–1897. JMLR Workshop and Conference Proceedings. 7.1.2.2, 7.1.2.2

Schtzenberger, M. (1961). On the definition of a family of automata. *Information and Control*, 4(2):245 – 270. 2.1.3

Shaban, A., Farajtabar, M., Xie, B., Song, L., and Boots, B. (2015). Learning latent variable models by improving spectral solutions with exterior point methods. In *Proceedings of The International Conference on Uncertainty in Artificial Intelligence (UAI)*. 2.4.3.3

Siddiqi, S., Boots, B., and Gordon, G. J. (2010). Reduced-rank hidden Markov models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-2010)*. 2.4.3.2, 3.5.1.2, 3.6

Singh, S., James, M. R., and Rudary, M. R. (2004). Predictive state representations: A new theory for modeling dynamical systems. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, pages 512–519, Arlington, Virginia, United States. AUAI Press. 1.2, 5, 2.3, 2.3.2, 6.1.3

Smola, A., Gretton, A., Song, L., and Schlkopf, B. (2007). A hilbert space embedding for distributions. In *In Algorithmic Learning Theory: 18th International Conference*, page 1331. Springer-Verlag. 4.1, 4.1.3

Song, L., Boots, B., Siddiqi, S. M., Gordon, G. J., and Smola, A. J. (2010). Hilbert space embeddings of hidden Markov models. In *Proc. 27th Intl. Conf. on Machine Learning (ICML)*. 2.1.3, 3.6

Song, L., Huang, J., Smola, A. J., and Fukumizu, K. (2009). Hilbert space embeddings of conditional distributions with applications to dynamical systems. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, pages 961–968. 3.9, 3.4.1, 4.1.3, 4.1.4, 4.A

Stock, J. and Watson, M. (2011). *Introduction to Econometrics*. Addison-Wesley series in economics. Addison-Wesley. 3.2

Sun, W., Venkatraman, A., Boots, B., and Bagnell, J. A. (2016). Learning to filter with predictive state inference machines. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1197–1205. JMLR.org. 2.1.2, 2.4.2.1, 2.5.2, 3.6

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition. 7.1.1

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., and Müller, K., editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press. 7.1.2.1

Szita, I. and Lrincz, A. (2006). Learning tetris using the noisy cross-entropy method. *Neural Computation*, 18(12):2936–2941. 7.1.2

Terwijn, S. A. (2002). On the learnability of hidden markov models. In Adriaans, P., Fernau,

H., and van Zaanen, M., editors, *Grammatical Inference: Algorithms and Applications*, pages 261–268, Berlin, Heidelberg. Springer Berlin Heidelberg. 2.4.3.3

Thon, M. and Jaeger, H. (2015). Links Between Multiplicity Automata, Observable Operator Models and Predictive State Representations a Unified Learning Framework. *Journal of Machine Learning Research*, 16:103–147. 2.2, 4

Tropp, J. A. (2012). User-friendly tools for random matrices: An introduction. NIPS Tutorial. 3.A.1

Tropp, J. A. (2015). An introduction to matrix concentration inequalities. *Found. Trends Mach. Learn.*, 8(1-2):1–230. 6.13, 6.15

van Overschee, P. and de Moor, L. (1996). *Subspace identification for linear systems: theory, implementation, applications*. Kluwer Academic Publishers. 2.4.4.2, 2.4.4.2, 2.4.4.2, 11, 3.1, 3.3.2, 3.6, 6.1.2, 6.2, 6.3.2, 6.5.5, 6.6.2, 6.6.2.1

Venkatraman, A., Rhinehart, N., Sun, W., Pinto, L., Boots, B., Kitani, K., and Bagnell., J. A. (2017). Predictive state decoders: Encoding the future into recurrent networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. 7.3, 7.4.3, 7.4.6, 7.5

Venkatraman, A., Sun, W., Hebert, M., Bagnell, J. A. D., and Boots, B. (2016). Online instrumental variable regression with applications to online linear system identification. In *Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*. 3.6, 8.2.4

Wang, Y. and Anandkumar, A. (2016). Online and differentially-private tensor decomposition. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3531–3539. 5

Wang, Y., Tung, H.-Y., Smola, A. J., and Anandkumar, A. (2015). Fast and guaranteed tensor decomposition via sketching. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 991–999. Curran Associates, Inc. 5, 5.1.1, 5.1.2, 5.2.3, 5.2.3

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*. 2.5.1

Wiewiora, E. W. (2008). *Modeling probability distributions with predictive state representations*. PhD thesis, University of California, San Diego. 2.4.3.3

Williams, C. K. I. and Seeger, M. W. (2000). Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 682–688. 4.1.5

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256. 7.1.2.1

Wingate, D. and Singh, S. (2007). On discovery and learning of models with predictive representations of state for agents with continuous actions and observations. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, pages 187:1–187:8, New York, NY, USA. ACM. 2.1.2.1

Xia, G. G. (2016). *Expressive Collaborative Music Performance via Machine Learning*. PhD thesis, Carnegie Mellon University. 3.6

Zhang, Y., Chen, X., Zhou, D., and Jordan, M. I. (2016). Spectral methods meet em: A provably optimal algorithm for crowdsourcing. *J. Mach. Learn. Res.*, 17(1):3537–3580. 2.4.3.3

Zhao, M. and Jaeger, H. (2010). Norm-observable operator models. *Neural Computation*, 22(7):1927–1959. 2, 4.3.3