Unified Models for Dynamical Systems

Carlton Downey

MAY 2019
CMU-ML-19-107

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

Thesis Committee:

Geoffrey Gordon, Chair
Byron Boots
Arthur Gretton
Ruslan Salakhutdinov

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

This research was sponsored by the Office of Naval Research award N000141512365 and grants from Intel Corporation and PNC Financial Services.

Keywords: Spectral Algorithms, Bayes Filters, Recurrent Neural Networks, Kernel Methods, Filtering, Time Series, Tensors, Tensor Decomposition, Predictive State Representations, Method of Moments, Quantum Mechanics

To my Family

Abstract

Intuitively a dynamical system is any observable quantity which changes over time according to some fixed rule. Building models to understand, predict, and control dynamical systems has been a field of study for many years, resulting in a large and diverse array of distinct models. Each of these models offers its own unique advantages and disadvantages, and choosing the best model for a given task is a difficult problem where a variety of tradeoffs must be considered. In this work we explore the complex web of relationships between these models, and use the insights gained to derive new connections. Our goal is to unify these many diverse models into sophisticated hybrid models which offer the best of all worlds. In particular we focus on unifying two main categories of models: Bayes Filters and Recurrent Neural Networks. Bayes Filters model dynamical systems as probability distributions and offer a wealth of statistical insight. In contrast Recurrent Neural Networks are complex functions design to produce accurate predictions, but lack the statistical theory of Bayes Filters. By drawing on insights from each of these fields we develop new models which combine an axiomatic statistical theory with rich functional forms, are widely applicable and offer state of the art performance.

Acknowledgments

I would like to take a moment to express my profound gratitude to all the incredible people who were a part of this journey with me, and without whom this work would not have been possible.

First and foremost I would like to thank my adviser, Geoffrey Gordon. It is under his guidance that I first fell deeply in love with the wonder of machine learning, an affection which has only grown as time moved forwards. Geoff encouraged me to explore widely and let my interests guide me, whilst also keeping me firmly focused on the path forwards. Through the highs and lows of my PhD Geoff has been supportive at every turn. He has listened to my crazy ideas, nurtured me with his insights, and helped me grow into the person that I am today.

I would also like to thank Stephen Fienberg for helping advise me during my early years at CMU. Steve was a mentor, friend and teacher, and I regret that his passing robbed me of the opportunity to spend more time learning from him.

I would like to thank my thesis committee: Byron Boots, Arthur Gretton, and Ruslan Salakhutdinov. I am indebted to them for their time, their support, their ideas and their advice.

The work presented in this thesis is the result of many fruitful collaborations. To this end I would like to thank Ahmed Hefny, Byron Boots, Boyue Li, Krzysztof Choromanski, and Siddarth Srinivasan. I learned a great deal from working with each and every one of them, and this work is as much theirs as it is mine. I would also like to thank Avinava Dubey, Sashank Reddi, Zita Marinho, Suvrit Sra, Quan Wang, Ignacio Lopez Moreno, Li Wan, and Philip Andrew Mansfield for similar contributions to work which was not included in this thesis.

I would like to thank the CMU staff for their myriad contributions great and small to the department, and for helping ensure my time at CMU was so special. In particular I would like to thank Diane Stidle for her tireless work to help and support me through the complexities of grad school.

My time at CMU has been deeply enriched by sharing this journey with the many other incredible people there. In particular I would like to thank Shing-hon Lau, Adona Iosif, Jesse Dunietz, Avinava Dubey, Sashank Reddi, Jing Xiang, William Bishop, Eric Wong, Maruan Al-Shedivat, Manzil Zaheer, and Aaditya Ramdas for the countless interesting discussions.

Finally I would like to thank my family; Kristin, Rod, and Alex Downey. A PhD is a difficult journey, and their continuous support meant everything to me. They have been there for me day in and day out, through good times and bad, always ready to pick me back up when I was knocked down.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Building models to understand, predict, and control dynamical systems has been a field of study for many years, resulting in a large and diverse array of distinct models. Each of these models offers its own unique advantages and disadvantages, and choosing the best model for a given task is a difficult problem where a variety of tradeoffs must be considered.

Many of these approaches can be categorized as either recursive Bayes Filtering or Recurrent Neural Networks. Bayes Filters are a probabilistic approach for modelling dynamical systems. They maintain a belief state corresponding to a probability distribution over future observations, and update this distribution recursively over time using incoming measurements and Bayes Rule. In contrast Recurrent Neural Networks are arbitrary functions which take past observations as inputs and produces predictions about future observations as output.

Bayes Filters possess a strong statistical theory, affording these models a host of useful properties: we can easily include prior or expert knowledge when constructing our model; there exist efficient and consistent learning algorithms; and we can use them to gain insight into the underlying system. Unfortunately due to various issues practical implementations of these models are often limited to relatively simple functional forms which make unrealistic assumptions about the underlying distribution.

Recurrent Neural Networks offer complementary advantages; they focus on developing rich, complex models which can still be efficiently learned from data. To achieve this goal Recurrent Neural Networks abandon much of the axiomatic statistical theory which makes Bayes Filters so useful. In particular their lack of statistical theory makes them difficult to analyze, and limits their learning to simple gradient based methods.

In this thesis we explore the complex web of relationships between Bayes Filters and RNNs. We leverage the insights to develop novel hybrid models which unify the axiomatic statistical theory of Bayes Filters with the rich functional forms and practical performance of RNNs.

1.1 Main Contributions

The main contributions of this thesis are:

- We unify the various Method-of-Moments learning algorithms for Bayes Filters under a single learning framework. Specifically, we propose a framework which reduces MoM learning to solving three supervised learning problems using the idea of instrumental-variable regression. This allows us to directly apply the rich literature on supervised learning methods to incorporate many types of prior knowledge about problem structure. We show that this framework subsumes a wide variety of existing spectral algorithms. This work is split into two parts: In the first we develop these ideas in the context of uncontrolled systems. In the second we extend our earlier work to the more difficult setting of controlled systems.

- We leverage this framework to develop two new hybrid models which unify the advantages of Bayes Filters and Recurrent Neural Networks in a single model. Our new models are Predictive State Recurrent Neural Networks (PSRNNs) and Hilbert Space Embedding of Hidden Quantum Markov Models (HSE-HQMMs).

  PSRNNs use bilinear functions to combine information from multiple sources. We show that such bilinear functions arise naturally from state updates in Bayes filters, in which observations can be viewed as gating belief states. We show that PSRNNs directly generalize discrete PSRs, and can be learned effectively by combining Backpropagation Through Time (BPTT) with two-stage regression. We also show that PSRNNs can be factorized using tensor decomposition, reducing model size and suggesting interesting connections to existing multiplicative architectures such as LSTMs.

  We investigate the link between Quantum Mechanics and HSEs and show that the sum rule and Bayes rule in Quantum Mechanics are equivalent to the kernel sum rule in HSEs and a special case of Nadaraya-Watson kernel regression, respectively. We show that these operations can be kernelized, and use these insights to propose a Hilbert Space Embedding of Hidden Quantum Markov Models (HSE-HQMM) to model dynamics. We present experimental results showing that HSE-HQMMs are competitive with state-of-the-art models like LSTMs and PSRNNs on several datasets, while also providing a nonparametric method for maintaining a probability distribution over continuous-valued features.

- We show how these new ideas can be implemented at scale. We focus on solving two concrete problems: 1) how to use Tensor Sketching to reduce the cost of two-stage regression and 2) how to use Orthogonal Random Features to reduce the cost of embedding distributions in Hilbert spaces.

  We present Sketched 2SR, a novel formulation of two-stage regression which uses tensor sketches, combined with tensor power iteration, to efficiently learn PSRNNs from data. Sketched 2SR requires only $O(d)$ space and $O(d^2)$ time, compared with two-stage regression which requires $O(d^3)$ space and $O(d^4)$ time.

  We show that orthogonal random features can be used in place of conventional random features to learn smaller, faster PSRNNs. We theoretically analyze the orthogonal version of two-stage regression, and show that we obtain strictly better empirical risk. In particular, we show that OPSRNN models can achieve accuracy similar to PSRNNs with an order of magnitude fewer features needed.

1.2 Organisation

The remainder of this thesis is organised as follows:
Part I: Background

- **Chapter 2** We present an overview of dynamical systems, how they are modelled, and the algorithms used for learning said models. This chapter concludes with a discussion on the pros and cons of the various techniques, and sets the stage for later contributions.

Part II: Unifying Method of Moments Learning.

- **Chapter 3** We develop a new framework for learning uncontrolled dynamical systems by reduction to supervised learning. This framework defines a new class of models, called Predictive State Models, and a new learning algorithm, called Two Stage Regression. This chapter is based on [59].
- **Chapter 4** We extend the framework developed in chapter 3 to controlled dynamical systems. This chapter is based on [60]

Part III: Hybrid Models

- **Chapter 5** We introduce a new model in the class of predictive state models which is called a Predictive State Recurrent Neural Network (PSRNN). PSRNNs draw on insights from both Recurrent Neural Networks and Predictive State Models, and inherit advantages from both types of models. This chapter is based on [44].
- **Chapter 6** We generalize PSRNNs using ideas from quantum mechanics to obtain a new model, called Hilbert Space Embeddings of Hidden Quantum Markov Models (HSE-HQMMs), which describes a rich class of probability distributions over sequences of continuous observations. This chapter is based on [107].

Part IV: Practical Implementations

- **Chapter 7** We show how to learn large predictive state models efficiently using ideas from tensor-sketching. This chapter is not yet submitted.
- **Chapter 8** We show how to further reduce the size of predictive state models using orthogonal random features. This chapter is based on [38].

.
Part V: Conclusions and Future Work

- **Chapter 9** We present a summary of our conclusions.

Part I

Background

Chapter 2

Background

In this chapter we present an overview of the background material on which the work in this thesis both depends and builds. We begin by introducing the notion of dynamical systems, and how they can be used. We then discuss the various types of dynamical system models, with particular attention to the distinction between Bayes Filters and Recurrent Neural Networks. We then discuss the various ways these models can be learned from data, distinguishing between Generative approachs and Discriminative approaches. Finally we end with a discussion of the pros and cons of these various techniques and how we might want to extend them. Our hope is that the reader can take away from this section that 1) Dynamical systems are ubiquitous and important, 2) There are a wide variety of techniques for modelling dynamical systems, each with their own strengths and weaknesses, and 3) Many of these ideas have been developed in isolation, and have the potential to benefit from being combined.

2.1 Dynamical Systems

Intuitively a dynamical system is any observable quantity which changes over time according to some fixed rule. Dynamical systems are an incredibly broad class of objects, with examples as complex and disparate as a pendulum swinging, a car being driven, the stock market, a leaf blowing in the wind, speech, text, a person running, or even the human brain. We can also categorize dynamical systems as controlled or uncontrolled, i.e., whether or not our actions can influence the behavior of the system.

Formally we define a dynamical system in terms of time t, (system) states q, observations o, actions a, system dynamics $f()$ and emission dynamics $g()$. State q_t is a description or snapshot of the system at a given point in time t. For example the state of a pendulum would consist of its position and velocity. Observations o_t are a (probabilistic) function of the state, and correspond to our perception of the state at time t. For example an observation of a pendulum might consist of its position, or simply a picture of the pendulum at a given point in time. We note that there may be partial observability, e.g. while it is possible to obtain the position of the pendulum from a single observation (picture), it is not possible to obtain its velocity. Actions a_t correspond to interventions we take at time t which influence the behavior of the system. These will be discussed in more detail in Chapter 4; for now we will focus on uncontrolled systems.

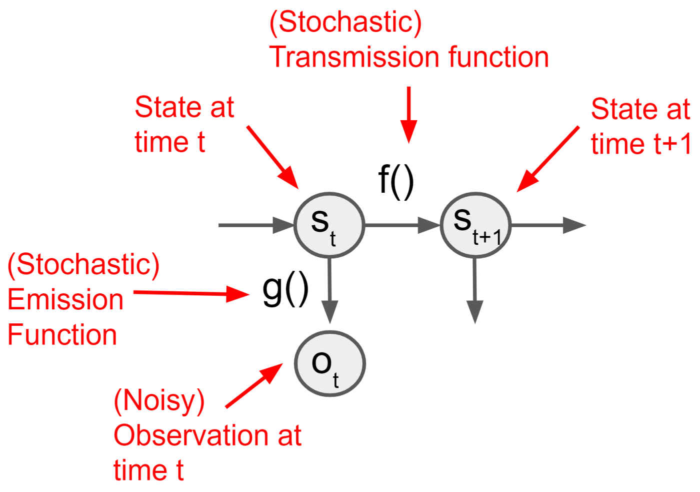


Figure 2.1: Dynamical System

The state evolves over time according to the system dynamics $f()$:

$$q_{t+1} = f(q_1, ..., q_t) = f(q_{1:t})$$

One way to think of $f()$ is as a description of the physical laws governing the system. In the case of the pendulum if we know the length of the string, the weight and shape of the pendulum, the materials used, etc., then we can use our knowledge of physics to describe the future state (position and velocity) as a function of the current state (position and velocity) and the change in time.

$f()$ may be stochastic, i.e.:

$$q_{t+1} = f(q_1, ..., q_t, z_t) = f(q_{1:t})$$

where z_t is random.

The emission dynamics $g()$ define the relationship between the q_t and o_t:

$$o_t = g(q_t)$$

We can think of $g()$ as a description of our ability to observe q_t with a set of measuring devices (e.g. camera, radar, etc). In some rare cases the system is *fully observable*, i.e. $q_t = o_t$, however for the vast majority of interesting systems the system is partially observable.

Thus far we have assumed that observations occur at fixed discrete intervals, i.e. time is an Integer. In reality time can be Integer or Real, and can be measured at fixed or varying intervals. For example a sentence forms a dynamical system with discrete time where each word corresponds to an observation, while the previous pendulum example clearly exists in the physical world with continuous time. In a similar vein observations can be categorical, Integer, Real or Complex, and can be scalar or vector valued. Different disciplines have traditionally focused on dynamical systems of a particular type: For example, Physicists study problems with continuous time and complex vector valued observations (e.g. atomic particles), whereas Linguists study problems with discrete fixed interval time, and categorical observations (e.g. text). The category of dynamical system has a major impact on the types of models developed, with different fields developing models with different strengths and weaknesses.

In practice we often treat continuous dynamical systems as if time was discrete and occurred at fixed intervals. The reason for this is that our perception of the world is often via measurement devices which record observations at fixed intervals, e.g. cameras, microphones, sonar, etc. Thus for all intents and purposes these systems can be viewed as discrete. In this thesis we will focus on dynamical systems with discrete time at fixed intervals, and real vector valued observations. Assuming discrete time is common in many domains, as even if the system is continuous, the observations are obtained via a sensor with a discrete fixed sampling rate.

To give some idea of the sheer number and variety of applications which use dynamical system model we now list a few examples:

- Aircraft Autopilot [32]
- Brain-computer interface [79]
- Object Tracking [39]
- Stock Market Forecasting [58]
- Satellite navigation [40]
- Seismology [55]
- Speech Recognition [54]
- Weather forecasting [13]
- Molecular analysis [99]
- Cryptography [9]
- Handwriting recognition [54]
- Activity recognition [45]
- Protein folding [91]

2.1.1 Models

Modelling dynamical systems is the problem of finding a mathematical description of the system. Dynamical Systems are of great interest in just about every scientific discipline, and a great deal of effort has been spent developing models for them. We model dynamical systems for a variety of purposes:

- **Understanding:** By developing a good model we can often gain insight or understanding about a system. This is a common approach in the biological sciences where we wish to better understand complex subjects such as the human brain.
- **Prediction:** We can use models to make predictions about future states/observations of the system. Having good estimates for future quantities is of obvious benefit when purchasing stocks, making policy decisions, etc. The task of computing predictions or related quantities from history is often called inference.

- **Control:** Good models are important in any controlled system, such as robotics. If we want our robot to be able to navigate the world, we first need a model for how our low level actions such as activating a motor will influence our robots state.
- **Optimize:** In disciplines such as chemical engineering we are often trying to build dynamical systems such as chemical pipelines. By obtaining good models of these systems it allows us to optimize them for the required task without physical trial and error.
- **Simulation:** Good models of dynamical systems allow us to generate fake or imaginary data via simulation. This is incredibly useful in fields such as game design as it allows studios to obtain dynamic textures such as fires, hair, etc., without expensive hand sculpting.

In this thesis we will focus on modelling dynamical systems for the purpose of prediction, where we predict future observations given a history of previous observations. We note that it is possible to use the techniques discussed for other tasks, however this is beyond the scope of this thesis.

Dynamical systems can be extremely difficult to model for a number of reasons:

- *Transitions/Observations are noisy* Many dynamical systems are stochastic, with transition/behavior which is highly random. This means that it is necessary to extract the model dynamics from the noise. If care is not taken models can end up learning spurious correlations and performing poorly.
- *We do not observe the true state* Most dynamical systems of interest are *Partially Observable*, in other words it is not possible to observe the state, only the observations. This means we must determine what state cause the given observations without ever actually seeing it. This can be difficult given stochasticity, and the fact that different states can result in the same observations.
- *We don't know what the true state looks like, or it may be too large to be useful* In many simple systems we know exactly the information required precisely and compactly describe the system (position and velocity). However in more complex systems such as text, the brain, the stock market, we are stuck making educated guesses.
- *We don't know the functional form of f() or g()* Similarly in simple systems we often know the exact relationship between one state and the next. In the case of the pendulum the change in the state depends on the laws of gravity, momentum, etc. However in more complex systems we once again can only make estimates.
- *We have to generalize to unseen observations and states* We learn dynamical systems by collecting a data set, use this data to build a model of the system, then apply the learned model to make predictions about a different data set, or future behavior. The difficulty is that this data is different to the data we used to learn the system. If we have learned a good model which accurately represents the state/dynamics of the system this will work well, however instead we often end up learning models which are fragile, and only work on previously seen data.

Due to their almost universal interest across so many domains there are a vast array of models for dynamical systems. Traditionally we model dynamical systems using *recursive* models. These models parallel the functional form of the dynamical systems they model: They define a state, and update that state recursively over time as they receive new information (in the form of observations).

Most recursive models can be categorized as either recursive Bayes Filters (BFs) or Recurrent Neural Networks (RNNs). These two classes follow fundamentally different design philosophies, resulting in two very different classes of models. Bayes Filters maintain a probability distribution over system states, and update this distribution using Bayes rule when they receive new information. Conversely RNNs maintain an arbitrary state with no relation to the system state, and update it using some arbitrary function. For many years, most dynamical systems models were Bayes Filters, however the last decade has seen the emergence of new RNNs which have quickly overtaken Bayes Filters in performance and popularity on many tasks. We will now introduce these two classes in more depth, and compare and contrast their various strengths and weaknesses.

Inference can be separated into *filtering* and *prediction*.[1] Filtering (also known as State Tracking) is the process of updating the state as we receive new information: i.e. given the current state q_t and an observation o_t we wish to obtain an updated state q_{t+1}. Prediction is the problem of predicting the value of future observations: i.e. given the current state q_t we would like to estimate the probability of future observations $o_{t+1:t+n}$. These two problems are intimately linked, as we often learn to filter by updating our model parameters based on errors in predictions.

We note that in this thesis we restrict ourselves to the class of *Markovian* movdels, where the state is a function of only the previous state:

$$q_{t+1} = f(q_t)$$

Many systems of interest admit simple Markovian models, and many that are not can be effectively approximated by Markovian models. This Markovian property is important as it allows for tractable computation and modelling, as it allows the size of the state to be fixed rather than growing unboundedly over time.

2.1.2 State

The notion of *State* is at the core of the study of dynamical systems. Unfortunately there are several different notions of state, all of which are often confusing referred to as simply “the” state of the system, with the reader left to figure out which kind of state is being discussed from context. In particular the differences between the notions of Observable State, System State, Belief State, Latent State, and RNN (arbitrary) state are often poorly defined.

An **Observable State** is any state which can be observed. In contrast a *latent state* is any state which cannot be observed. Observable states are easy to model, as their value can be directly observed from training data. In contrast latent states are often difficult to learn as their value must be obtained via inference.

We note that the notion of observability is a function of our sensors. A state may be observable with one set of sensors, but latent with a different set of sensors. Consider the example of a pendulum, where the state consists of the position and the velocity. If we use a camera as our only sensor then the position is observable, while the velocity is not. If we wish to obtain an estimate of the velocity then it must be inferred from a combination of several positional observations.

[1]There are other forms of inference in addition to filtering and prediction, such as smoothing and likelihood evaluation, but they are outside the scope of this thesis.

However if we augment our sensor suite with a radar gun then we can directly observe the velocity at a given point in time, rendering the full state observable.

A **System State** is a state of the underlying dynamical system. It represents a description of the instantaneous condition of the system at a specific point in time. The state evolves over time according to the system dynamics described previously:

$$q_{t+1} = f(q_1, ..., q_t) = f(q_{1:t})$$

The notion of a system state is useful for reasoning about dynamical systems and their models. In particular it allows us to encode prior knowledge about the system dynamics or physical laws governing the system. Unfortunately the system state view often leads to latent states which are difficult to learn.

A **Belief State** is a state which corresponds to a probability distribution or belief over future observations. Formally a belief state is a set of statistics, which, if known at time t, are sufficient to predict all future observations as accurately as if we know the full history. An overly simplistic, but useful, interpretation of the belief state is as a distribution over system states. A more precise interpretation is the statistics of a distribution over the latent state of a data generating process, conditioned on history. A third is as a bottleneck that compresses the entire observation history for the purpose of future predictions.

An **RNN** or **Arbitrary** state is an arbitrary collection of information maintained by an RNN at each time step and which is used to make predictions about future observations. It does *NOT* correspond in any simple way to a probability distribution, or have any simple relationship to the underlying system state. This is in contrast to a belief state which directly corresponds to a probability distribution.

These definitions and distinctions will become clear in the context of the various models and learning algorithms introduced throughout this chapter.

2.2 Bayes Filters

Bayes Filters (BFs) [94] are a general probabilistic approach for modelling dynamical systems. They maintain a belief state corresponding to a probability distribution over future observations, and update this distribution recursively over time using incoming measurements and Bayes Rule. Another way to think about BFs is that the entire BF is a single large probability distribution which has been factorized using conditional independence assumptions.

Formally a Bayes Filter maintains a conditional probability distribution or *belief* over system states $q_{t|t} = \Pr(s_t \mid o_{1:t})$ where $o_{1:t}$ denotes the first t observations. $q_{t|t}$ is called a *Belief State* and represents both our knowledge and our uncertainty about the true state of the system.

Filtering with BFs is straightforwards: given the current belief $q_t = q_{t|t}$ and a new observation o_t, we use Bayes rule to calculate an updated belief $q_{t+1} = q_{t+1|t+1}$ that incorporates o_{t+1}. This can be achieved using Bayes rule as follows:

$$
\begin{aligned}
q_{t+1} &= \Pr(s_{t+1} \mid o_{1:t+1}) && (2.1)\\
&= \frac{\Pr(s_{t+1}, o_{t+1} \mid o_{1:t})}{\Pr(o_{t+1} \mid o_{1:t})} && (2.2)\\
&= \frac{\sum_{s_t} \Pr(o_{t+1} \mid s_{t+1})\Pr(s_{t+1} \mid s_t)\Pr(s_t \mid o_{1:t})}{\Pr(o_{t+1} \mid o_{1:t})} && (2.3)\\
&= \frac{\sum_{s_t} \Pr(o_{t+1} \mid s_{t+1})\Pr(s_{t+1} \mid s_t)q_t}{\Pr(o_{t+1} \mid o_{1:t})} && (2.4)
\end{aligned}
$$

We note that the denominator in Equation 2.4 is rarely computed explicitly, and instead we normalize the resulting probability distribution to sum to 1. As we shall see shortly in the case of an HMM q_t is represented by a probability vector over latent system states, and the numerator of Equation 2.4 is the forwards algorithm. 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 a Kalman Filter.

Prediction with BFs involves projecting our belief into the future: given a belief $q_{t|t}$ we estimate $q_{t+k|t} = \Pr(s_{t+k} \mid o_{1:t})$ for some $k > 0$ (without incorporating any intervening observations). We achieve this via the sum rule:

$$\Pr(o_{t+1} \mid o_{1:t}) = \sum_{s_t, s_{t+1}} \Pr(o_{t+1} \mid s_{t+1})\Pr(s_{t+1} \mid s_t)\Pr(s_t \mid o_{1:t}) \qquad (2.5)$$

Due to their probabilistic grounding, BFs possess a strong statistical theory leading to efficient learning algorithms. In particular, Method-of-Moments algorithms provide consistent parameter estimates for a range of BFs [22, 59, 68, 100, 113]. Unfortunately, current versions of Method-of-Moments initializations restrict BFs to relatively simple functional forms such as linear-Gaussian (KFs) or linear-multinomial (HMMs). We will discuss this in more detail in Chapter 3.

We now discuss a few of the most common types of BFs in more detail:

2.2.1 Hidden Markov Models

Hidden Markov Models [15] (HMMs) are one of the simplest and most popular examples of Bayes Filters [14]. HMMs assume the state is categorical, i.e. discrete and finite, and that the system dynamics are linear. The observations are often assumed to be categorical as well, although the model can handle continuous observations with a simple extension.

Formally let there be a finite set of possible states $s_t \in 1, ..., n$ and a finite set of observations $o_t \in 1, ..., m$. We represent the system state (observation) as a one-hot encoded vector of length n (m). Given this formalism we define a HMM in terms of:

- T: The Transition Model, an $n \times n$ stochastic matrix (i.e. non-negative entries, where each column sums to 1)
- E: The Emission Model, an $m \times n$ stochastic matrix
- q_0: The initial belief state, an $n \times 1$ stochastic vector

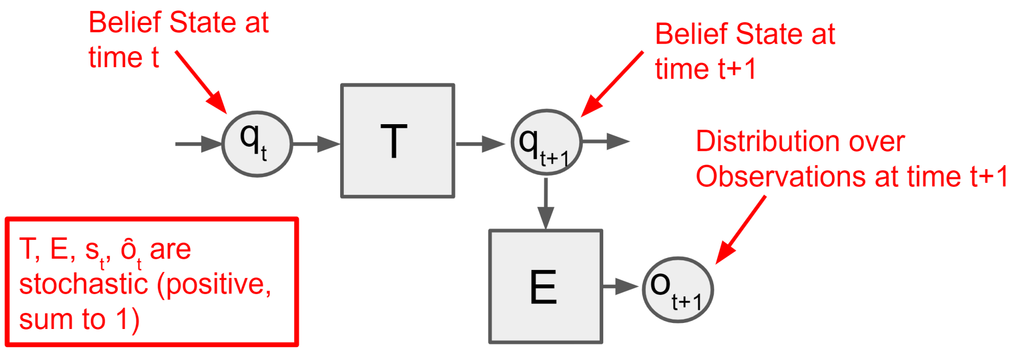

Belief State at time t
Belief State at time t+1
Distribution over Observations at time t+1
T, E, s_t, ô_t are stochastic (positive, sum to 1)
q_t
T
q_t+1
E
o_t+1

Figure 2.2: Hidden Markov Model (HMM) as a graphical model. Here nodes represent random variables, while arrows represent conditional independence assumptions.

T encodes the conditional probability of transitioning between each possible pair of system states:

$$\Pr(q_{t+1} \mid q_t) = Tq_t \quad \forall t$$

E encodes the conditional probability of observing each possible observation conditioned on each possible state:

$$\Pr(o_t \mid q_t) = Eq_t \quad \forall t$$

and q_0 is a stochastic vector which describes the initial belief state. Together T, E, and q_0 define a joint distribution over all possible sequences of observations:

$$\Pr(o_{1:T}) = \mathbf{1}^\top \left(\prod_t T \text{diag}(Eo_t) \right) q_0$$

This is represented graphically using a factor graph in figure 2.2.

HMMs are generative models in the sense that the HMM formulation leads to a simple algorithm for sampling sequences of observations distributed according to the density defined by T, E, and q_0. We sample $s \sim q_0$, then repeatedly sample $o \sim Es$ and $s \sim Ts$.

Prediction in HMMs is straightforwards: Given state q_t we apply T to obtain q_{t+1} then apply emission matrix E to obtain a distribution over observations:

$$\Pr(o_{t+1} \mid s_t) = ETq_t$$

As a Bayes Filter, we perform filtering in HMMs by invoking Bayes rule. Given state $q_t = \Pr(s_t \mid o_{1:t-1})$ and observation o_t we wish to obtain $q_{t+1} = \Pr(s_{t+1} \mid o_{1:t})$. Using equation 2.4 we have:

$$\Pr(s_{t+1} \mid o_{1:t}) = \sum_{s_t} \frac{\Pr(s_{t+1} \mid s_t)\Pr(o_t \mid s_t, o_{1:t-1})\Pr(s_t \mid o_{1:t-1})}{\Pr(o_t \mid o_{1:t-1})} \tag{2.6}$$

$$= \frac{T\text{diag}(Eo_t)q_t}{\mathbf{1}^\top T\text{diag}(Eo_t)q_t} \tag{2.7}$$

Filtering in HMMs, and Bayes Filters in general, can be visualized using a neural network diagram, as shown in figure 2.3. Please note the difference between figure 2.2 and figure 2.3: while both are graphical models, figure 2.2 is a factor graph, representing a factorized probability distribution via variables and conditional independence assumptions. Conversely figure 2.2 is a neural network diagram, showing the flow of information through a model/algorithm. Neural network diagrams can be thought of as a visualization of a complex function which takes inputs and produces outputs.

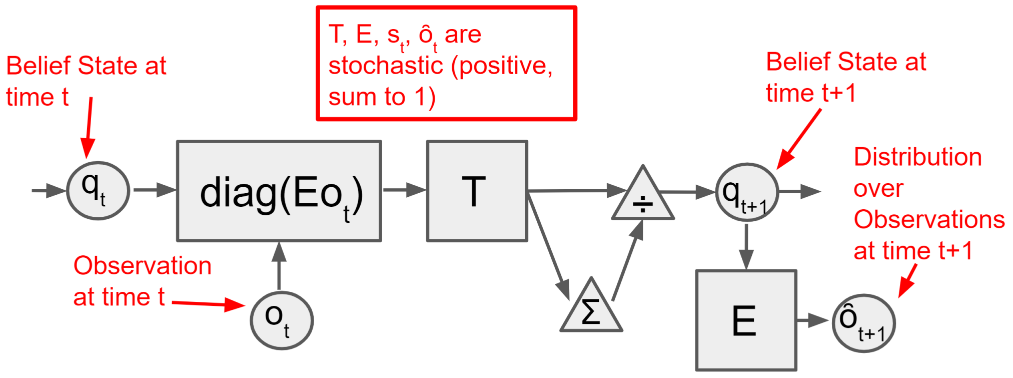


Figure 2.3: Filtering (The Forward Algorithm) In an HMM

HMMs are typical of the class of Bayes filters: We begin by defining a generative model corresponding to a factorized probability distribution over sequences of observations, then use the axioms of probability to derive the corresponding filtering procedure. HMMs serve as a good example of both the strengths and weaknesses of BFs: On the plus side they are a complete, axiomatic model which can be used for any number of tasks, are well understood, can be used to gain insights from data, and can easily be modified to incorporate domain knowledge. The downside is that we are limited to extremely simple assumptions about the model dynamics (linear) and the distribution (categorical).

Despite their limitations and restrictive modelling assumptions, HMMs are used for a wide variety of applications including many of those listed earlier.

2.2.2 Kalman Filters

Kalman Filters [70] are the continuous Gaussian analog of Hidden Markov Models. Specifically Kalman Filters (KFs) assume the state is continuous, the system dynamics are linear, and random noise is always Gaussian. This means the state is an arbitrary vector of length n, and observations are arbitrary vectors of length m. Formally we define a Kalman Filter in terms of a belief state q_t and the following quantities:

- F: The transmission model, an $n \times n$ matrix
- H: The emission model, an $n \times m$ matrix
- Q: The covariance of the transmission noise, an $n \times n$ matrix
- R: The covariance of the observation noise, an $m \times m$ matrix

- q_0: The initial belief state, an $m \times 1$ vector

Note that in the Hidden Markov model the Transmission and Emission matrices accounted for both the dynamics and the noise, whereas in the Kalman Filter we separate these quantities and explicitly account for the noise. In a Kalman Filter the state transition is:

$$q_{t+1} = Fq_t + w_t$$

where $w_t \sim \mathcal{N}(0, Q)$. The emission is:

$$o_{t+1} = Ho_t + v_t$$

where $v_t \sim \mathcal{N}(0, R)$.

We can generate samples from KFs with the same approach used for HMMs: We begin in the initial belief state q_0, then repeatedly apply the trasmission and emission equations with noise vectors sampled from the appropriate distributions to obtain a sequence of observations.

Prediction in Kalman Filters is also similar to HMMs, but with one important difference; we need to explicitly keep track of the "noise" or "uncertainty" associated with our estimated state q_t. As a reminder our belief state is a probability distribution over system states. In HMMs we kept track of this distribution explicitly as a vector of probabilities, with one number for each possible system state – clearly this is not possible in Kalman Filters, where we are working with continuous observations. Instead we make use of our assumption that the noise in our model is Gaussian, allowing us to parameterize our belief state in terms of the mean q_t and the covariance of our system state P_t. We can think of these two quantities together as our new belief state, as together they describe a distribution over system states.

Given q_t and P_t prediction is straightforwards due to the nice properties of Gaussian distributions. First we project q_t and P_t forwards in time using the transmission model:

$$q_{t+1} = Fq_t \tag{2.8}$$

$$P_{t+1} = FP_tF^\top + Q \tag{2.9}$$

We then obtain a (Gaussian) distribution over observations using the emission model, parameterized in terms of a mean o_{t+1} and covariance C_{t+1}:

$$o_{t+1} = Hq_t \tag{2.10}$$

$$C_{t+1} = HP_tH^\top + R \tag{2.11}$$

Filtering in Kalman Filters requires us to use incoming observations to refine our estimate of both the mean system state q_t and the noise covariance P_t. This is similar to filtering in HMMs – a simple application of Bayes Rule, however the algebra is significantly more complex:

$$y_t = o_t - Hq_t \tag{2.12}$$

$$S_t = R + HP_tH^\top \tag{2.13}$$

$$K_t = P_tH^\top S_k^{-1} \tag{2.14}$$

$$\tag{2.15}$$

$$q_{t+1} = q_t + K_ty_t \tag{2.16}$$

$$P_{t+1} = (I - K_tH)P(I - K_tH)^\top + K_tRK_t^\top \tag{2.17}$$

$$\tag{2.18}$$

For a complete derivation please see [70].

HMMs and Kalman Filters are by far the two most commonly used Bayes Filters, and were the workhorses of dynamical system modelling for many years. However they both possess several signifcant limitations, leading to the development of alternative models.

2.2.3 Observable Operator Models

One of the problems with many BFs, and HMMs in particular, is that they are difficult to learn due to their reliance on latent variables. When variables are observable they are easy to learn, as we can simply compare the estimated values to the observed values, and update our model parameters to correct discrepancies. We cannot do this with latent variables, so instead we are reduced to various strategies which involve making educated guesses about the value of latent variables.

Observable Operator Models (OOMs) [69] are an attempt to avoid this problem by constructing a dynamical system model using observable operators instead of latent states. To understand the intuition behind this idea, consider the joint distribution over all possible sequences of observations defined by an HMM:

$$\Pr(o_{1:T}) = \mathbf{1}^\top \left(\prod_t T \text{diag}(E o_t) \right) q_0$$

We note that the latent state does not appear in this equation, except through the initial state q_0. In fact the probability of the given sequence is entirely determined by the collection of terms $T\text{diag}(Eo_t)$. Furthermore because the observation o_t can is finite, there are at most m unique terms. The idea behind OOMs is to define one observable operator A_x for each possible value x of o. With this definition the probability of observing a sequence of observations becomes:

$$\Pr(o_{1:T}) = \mathbf{1}^\top \left(\prod_t A_{o_t} \right) q_0$$

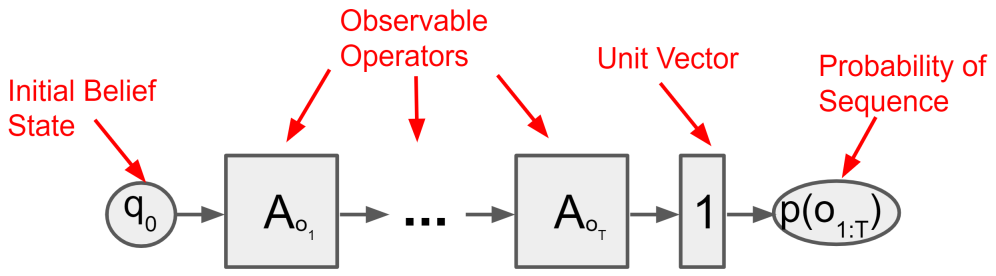


Figure 2.4: Neural Network Diagram of an OOM

Formally an OOM with linear dynamics consists of:

- $\{A_1, ..., A_m\}$: A collection of m observable operators, one for each possible observation, each of which is an $n \times n$ matrix
- q_0: The initial belief state, an $m \times 1$ stochastic vector

In order for an OOM to be valid, i.e. to represent a valid probability distribution over sequences, the following two conditions must hold:

$$\mathbf{1}^\top \sum_x A_x = \mathbf{1} \tag{2.19}$$

$$\mathbf{1}^\top A_{o_1} ... A_{o_n} q_0 >= 0 \quad \forall o_{1:n}, n \tag{2.20}$$

These two conditions can be derived directly from the axioms of probability, see .[69] for more details.

At first glance OOMs may seem like a simple algebraic reformulation of HMMs, however the key advantage is a change in perspective: Instead of viewing filtering as a sequence of states, we can instead view it as *a sequence of operators which generate the next observation from the previous one*. Crucially these operators are observable: The input and output to each operator are simply pairs of consecutive data points. This in contrast to an HMM, where we have no idea what the input and output to T is for a given training sequence.

An additional advantage of OOMs is that they are a generalization of HMMs. It is clear that every HMM can be converted into an OOM, however it turns out that there are valid OOMs for which there is no equivalent HMM. Another way of stating this result is that there are dynamical systems which can be modelled exactly by an OOM, which cannot be modelled by an HMM. One example of such an OOM is known as the "Probability Clock" and is defined as follows:

Let $O = a, b$ be the space of observations. Let T_ψ be the linear mapping which rotates vectors in $\mathbb{R}^3$ by an angle ψ around $(1, 0, 0)$ and ψ is not a rational multiple of 2π. Let $T_a = \alpha T_\psi$, where $0 < \alpha < 1$. Let T_b be an operator which projects every vector on the direction of w_0. Define the following observable operators:

$$T_a = \alpha \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & \sin(\psi) \\ 0 & -\sin(\psi) & \cos(\psi) \end{pmatrix}$$

$$T_b = w_0^\top \begin{pmatrix} 1 - \alpha \\ 1 + \alpha(\sin(\psi) - \cos(\psi)) \\ 1 - \alpha(\sin(\psi) + \cos(\psi)) \end{pmatrix}^\top$$

Note that every occurrence of b "resets" the process to a multiple of the initial vector w_0. Therefore, only the conditional probabilities $P(o_t = a \mid o_0 = a, o_1 = a, ..., o_{t-1} = a) = 1T_a^{t+1}w0/1T_a^t w_0$ The probability clock cannot be modelled by an HMM, which shows that the class of OOMs is greater than the class of HMMs. This can be proved by means of convex analysis, see Jaeger [69].

One downside of OOMs is that they can be much larger than HMMs. An HMM consists of $T = n \times n$, $E = n \times m$, and $q_0 = n \times 1$, hence it contains $n(n + m + 1)$ model parameters. Using the transformation described earlier, the equivalent OOM would consist of m operators, each of size $n \times n$, hence it contains $n^2 m + n$ model parameters.

Another downside of OOMs is that they no longer explicitly model the underlying system dynamics, and hence it is more difficult to gain insight, and or leverage expert knowledge about the system to develop useful priors.

2.2.4 Predictive State Representations

Predictive State Representations (PSRs) [102] are another Bayes Filter developed in an effort to avoid explicitly modelling latent states. Where HMMs/KFs explicitly model latent states, and OOMs model sequences of operators, PSRs take a third approach: They use a *predictive state*. The key insight behind PSRs is that we can view dynamical systems as generators of observations, and therefore the state of the system is equivalent to a probability distribution over future observations. In the language of PSRs a sequence of future observations is called a test, and our state is defined to be a probability distribution over tests conditioned on past observations. Now, clearly we can't keep track of all possible tests, as such a state would be infinite, however it turns out that it is often sufficient to keep track of a finite subset of such sequences. This subset, known as the core tests, is then used to recover the probabilities of other sequences outside the core tests. Such a state is known as a *Predictive State*. We make special note of the terms *Past* and *Future* observations, they will occur frequently throughout this thesis.

To formally define a predictive state we begin by selecting our core tests. Let $Z = z_1, .., z_n$ each be a sequence of observations. We define our predictive state at time t to be:

$$q_t = \Pr(Z \mid o_{1:t})$$

In other words q_t is a vector of probabilities, each corresponding to the probability of a particular test z_i occurring in the *future* given that the *history* $o_{1:t}$ has already occurred in the *past*. For example, suppose we are using a dynamical system to model the weather by tracking whether each day is sunny or rainy (a binary observation). Then our core tests might be "tomorrow will be sunny" and "the next 3 days will be rainy", and our state would be the probability of each of these two events occurring, given the previous weather which has already occurred. Here we are making the assumption that by tracking the probabilities of these two events, we are able to use them to make predictions about the weather on other days not covered by the core tests (e.g. the weather in 4 days time).

Formally we assume that for an arbitrary test z, which may or may not be in the core tests, that there is some function $f_z()$ which recovers the conditional probability of z given the history from the current state q_t:

$$\Pr(z \mid o_{1:t}) = f_z(q_t)$$

$f_z()$ can be non-linear, however for ease of exposition we will focus on the linear case where $f_z()$ is a vector a_z:

$$\Pr(z \mid o_{1:t}) = a_z^\top q_t$$

Given our predictive state, we now need to formalize how to perform filtering (state update) and prediction. In filtering, given our current state $q_t = p(Z \mid o_{1:t})$ and a new observation o_{t+1} we wish to obtain $q_{t+1} = p(Z \mid o_{1:t+1})$. In other words we want to obtain the conditional probabilities of the same set of tests after we augment the history with an additional observation. If we consider a single core test $z \in Z$ we obtain:

$$q_{t+1} = \Pr(z \mid o_{1:t+1}) \tag{2.21}$$
$$= \frac{\Pr(o_{t+1}z \mid o_{1:t})}{\Pr(o_{t+1} \mid o_{1:t})} \tag{2.22}$$
$$= \frac{a_{o_{t+1}z}^{\top} q_t}{a_{o+1}^{\top} q_t} \tag{2.23}$$

Where the last line follows from our assumptions on the core tests. If we collect the vectors $a_{o_{t+1}z}$ together for all core tests z we obtain the matrix $A_{o_{t+1}z}$. Using this operator we obtain our state update:

$$q_{t+1} = \frac{A_{o_{t+1}z}^{\top} q_t}{a_{o+1}^{\top} q_t}$$

We can now formally define the class of PSRs as follows:

- q_0: The initial predictive belief state, an $n \times 1$ vector of non-negative numbers in the range $[0, 1]$.
- $\{A_1, ..., A_m\}$: A collection of m expansion operators, one for each possible observation, each of which is an $n \times n$ matrix
- $\{a_1, ..., a_m\}$: A collection of m normalizing operators, one for each possible observation, each of which is an $n \times 1$ matrix

We can view each of the operators A_i as an operator which maps a marginal probability distribution to a joint distribution, and each of the operators a_i as an operator which maps a joint distribution to a conditional distribution.

The key benefit of using a Predictive State is that it is observable: Given a sequence of observations we can count the number of times each test occurs for each history, allowing us to learn the value of our predictive state from data.

As a final point we note the equivalence between the update equation for OOMs and for PSRs. As it turns out the matrices A_i correspond to observable operators in an OOM, and from an OOM one can always obtain a PSR using the observable operators. Therefore OOMs and PSRs are equivalent classes of models.

One disadvantage of the naive formulation of PSRs is that the set of core tests, and hence the size of the state, may both be extremely large. This can be partly solved by working with *Transformed PSRs* (TPSRs) [93]. The idea TPSRs is to maintain a linear transformation of the state distribution over core tests, rather than the actual distribution itself. This (often) allows us to represent the state in a much more compact form.

One facet of PSRs we have been silent on thus far is the question of how we select the set of core tests in the first place. There are a variety of heuristics used in practice, however a this is currently an open question.

2.2.5 Hilbert Space Embeddings of Bayes Filters

There are a couple of glaring weaknesses with many of the methods discussed thus far:

- They assume the system dynamics are linear.
- They assume discrete, low cardinality states/observations.

These assumptions are obviously false for a wide variety of applications, and attempting to shoehorn such systems into these models can result in poor modelling behavior.

One recently developed technique which helps address these problems is Hilbert Space Embeddings (HSE) of Distributions [103]. The idea is to embed each probability distribution into a Reproducing Kernel Hilbert Space (RKHS), and manipulate the probability distributions in that space using kernel versions of the rules of probability (e.g. kernel Bayes rule). When applied to Bayes Filters this approach allows us to embed our state (a probability distribution) into an RKHS, then update our state using Kernel Bayes Rule. The key advantages of using a HSE for modelling are that:

- Linear transformations in the kernel space can correspond to non-linear transformations in the original space. This means that we can learn a linear model which corresponds to non-linear system dynamics
- They provide us with a straightforwards way to generalize discrete models to continuous data.
- Data may be modeled non-parametrically, i.e. without assumptions about the form of the distribution and the relationship between variables.
- Intermediate density estimation is not required. In other words we only estimate the density when we wish to obtain an estimate of a particular quantity.

For those familiar with SVMs, HSE of distributions is the Bayes Filter equivalent of Kernel SVMs, and shares many of the same advantages. We now give a formally overview of HSE of distributions, and their applications to Bayes Filters.

Formal Definition

Let X denote a random variable with codomain Ω and distribution $\Pr(X)$. Given a kernel k on $\Omega \times \Omega$ an RKHS $\mathcal{H}$ is a Hilbert space of functions $f : \Omega \to \mathbb{R}$ equipped with an innter product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ and norm $|| \cdot ||_{\mathcal{H}}$ in which the element $k(x, \cdot)$ satisfies the reproducing property: $\langle f, k(x, \cdot) \rangle_{\mathcal{H}} = f(x) \quad \forall f \in \mathcal{H}, \forall x \in \Omega$. Intuitively $k(x, \cdot)$ is an implicit feature map which we denote by $\phi(x)$ from Ω to $\mathcal{H}$ so that $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$ can be viewed as a measure of similarity between x and x'.

The core concept behind HSE of distributions is the *mean map*:

$$\mu_X = \mathbb{E}_X[k(X, \cdot)] \tag{2.24}$$
$$= \mathbb{E}_X[\phi(X)] \tag{2.25}$$
$$= \int_{\Omega} \phi(x) dP(x) \tag{2.26}$$

We can think of the mean map as the expected value of the feature mapping. Under mild conditions it encodes a full description of the original distribution as an element of the RKHS. We can calculate a similar embedding for joint distributions:

$$C_{XY} = \mathbb{E}_{XY}[\phi(X) \otimes \phi(Y)] = \int_{\Omega \times \Omega} \phi(x) \otimes \phi(y) dP(x, y)$$

and for conditional distributions:

$$\begin{aligned} \mu_{Y|x} &= \mathbb{E}_{Y|x}[\phi(Y)] && (2.27) \\ &= \int_{\Omega} \phi(y) dP(y|x) && (2.28) \end{aligned}$$

We now have a way to embed each possible probability distribution into our RKHS. Furthermore, given samples from any such distribution, we can calculate an estimate of the embedding by using empirical moments. In other words an empirical estimate of an embedding is a weighted sum of samples in the feature space.

Now that we have obtained a method for embedding distributions into an RKHS we can define the axiomatic statistical rules which allow us to manipulate these distributions. Specifically we want to define kernel analogs for Sum Rule, Chain Rule, and Bayes Rule. We begin by noting that by the equivalence between a tensor and a linear map we can view the joint embedding C_{XY} as an uncentered cross-covariance operator $\mathcal{C}_{XY} : \mathcal{H} \to \mathcal{H}$ from which the cross-covariance of mean-zero functions $f, g \in \mathcal{H}$ can be computed as:

$$\begin{aligned} \mathrm{Cov}_{XY}(f(X), g(Y)) &= \mathbb{E}_{XY}[f(X)g(Y)] && (2.29) \\ &= \langle f, \mathcal{C}_{XY} g \rangle_{mathcalH} && (2.30) \\ &= \langle f \otimes g, \mathcal{C}_{XY} \rangle_{\mathcal{H} \otimes \mathcal{H}} && (2.31) \end{aligned}$$

Using this operator we can now define our three rules of probability. Kernel sum rule is defined as:

$$\begin{aligned} \mu_X &= \mathbb{E}_Y[\mathcal{C}_{X|Y} \phi(Y)] && (2.32) \\ &= \mathcal{C}_{X|Y} \mathbb{E}_Y[\phi(Y)] && (2.33) \\ &= \mathcal{C}_{X|Y} \mu_Y && (2.34) \end{aligned}$$

Kernel Chain Rule is defined as:

$$C_{XY} = C_{X|Y} C_{YY} \qquad (2.35)$$

And Kernel Bayes Rule is defined as:

$$\begin{aligned} \mu_{Y|x} &= C_{Y|X} \phi(x) && (2.36) \\ &= C_{YX} (C_{XX})^{-1} \phi(x) && (2.37) \end{aligned}$$

Together these three operators allow us to manipulate embedded probability distributions freely in the RKHS.

Kernel	$k(\Delta)$	$\rho(\omega)$
Gaussian	$e^{-\frac{\|\|\delta\|\|_2^2}{2}}$	$(2\pi)^{-\frac{D}{2}} e^{\frac{\|\|\omega\|\|_2^2}{2}}$
Laplacian	$e^{-\|\|\Delta\|\|_1}$	$\prod_d \frac{1}{\pi(1+\omega_d^2)}$
Cauchy	$\prod_d \frac{2}{1+\Delta_d^2}$	$e^{-1\|\|\Delta\|\|_1}$

Table 2.1: Some popular kernels and their Fourier transforms

Random Features

Kernel methods such as HSE of distributions have traditionally had a glaring weakness – their lack of scalability. Traditionally kernel methods are implemented via the "kernel trick", where we calculate calculate model parameters as a function of the *Gram Matrix* – a matrix of inner products between all pairs of data points. Unfortunately such an implementation scales quadratically in size and computational complexity with the amount of training data.

Fortunately, the development of Random Features [92] provides us with a solution to this problem. The idea behind random features is that given data points $x \in \mathbb{R}^n$ and kernel $k(\cdot,\cdot)$ there exists a mapping $z = f(x)$ such that $\langle f(x_1), f(x_2)\rangle = k(x_1, x_2)$. The existence of such a function is guaranteed for positive definite shift-invariant kernels by Bochner's theorem.

One popular variant is to use *Random Fourier Features*. Formally given a positive definite shift-invariant kernel $k(x, y) = k(x-y)$ we can obtain a random Fourier feature map $z(X) : \mathbb{R}^d \to \mathbb{R}^D$ such that $z(x)^T z(y) \approx k(x - y)$. To obtain z we first compute the Fourier transform p of the kernel k : $p(\omega) = \frac{1}{2\pi}\int e^{-i\omega^T\delta}k(\delta)d\Delta$. We then draw D i.i.d. samples $\omega_1, ..., \omega_D \in \mathbb{R}^d$ from p and D i.i.d samples $b_1, ..., b_D \in \mathbb{R}$ from the uniform distribution on $[0, 2\pi]$. We can now define our mapping z:

$$z(x) = \sqrt{\frac{2}{D}}[cos(\omega_1^T x + b_1), ..., cos(\omega_D^T x + b_D)]^T$$

Some popular kernels and their Fourier transforms are listed in table 2.1:

2.3 Recurrent Neural Networks

Recurrent Neural Networks are one answer to the question "What models can we build if we don't force our function to be a valid probability distribution?". We can think of Recurrent Neural Networks as a function which takes past observations as inputs and produces predictions about future observations as output. RNNs belong to the class of "Neural Networks", functions loosely inspired by connectionist models of the brain. These models build complex functions by combining large numbers of simple operators; typically alternating linear operators followed by non-linear activation functions.

Formally an RNN is defined in terms of:

- q_0: the initial state, an $n \times 1$ vector of arbitrary numbers.
- $f()$: an arbitrary transition function
- $g()$: an arbitrary emission function

$f()$ performs filtering by mapping the current state q_t and an observation o_t to the next state q_{t+1}:

$$q_{t+1} = f(q_t, o_t)$$

while $g()$ maps the current state to a prediction about future observations. This prediction could can take on many forms: It might be a vector encoding a distribution over observations, the parameters of some distribution (e.g. mean and variance for a Gaussian), the most likely observation, the top few most likely observations etc. In the discrete case where we are predicting a vector encoding a distribution over observations we would have:

$$p(o_{t+1} \mid o_{1:t}) = g(q_{t+1})$$

At the core of most RNNs is the perceptron: A simple function loosely corresponding to the connectionist model of a single neuron. A perceptron maps an input vector x to an output scalar y using a linear operator w, a scalar bias b, and a nonlinear activation function $\sigma()$:

$$y = \sigma(w^\top x + b)$$

The particular activation function used varies depending on the application, the larger model, and also what is trendy at a particular point in time, however common choices include:

- **Sigmoid:** $\sigma(y) = \dfrac{1}{1+e^{-y}}$
- **Tanh:** $\sigma(y) = \dfrac{e^y + e^{-y}}{e^y - e^{-y}}$
- **Relu:** $\sigma(y) = \begin{cases} 0 & \text{for } y < 0 \\ y & \text{for } y \geq 0 \end{cases}$
- **Step:** $f(y) = \begin{cases} 0 & \text{for } y < 0 \\ 1 & \text{for } y \geq 0 \end{cases}$
- **Gaussian:** $f(y) = e^{-y^2}$
- **Exponential Linear:** $f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$
- **Soft Plus:** $f(x) = \ln(1 + e^x)$

A wide variety of RNNs have been developed over the years, some of which are extremely complex, however the vast majority of such networks all use these same basic building blocks. We now cover three of the most common RNN architectures in more detail.

2.3.1 Elman Networks

Elman Networks [47] were among the first RNNs to be developed, and are the simplest non-linear RNN. Elman networks have the following architecture:

$$q_{t+1} = f(q_t, o_t) = \sigma_h(W_h o_t + U_h q_t + b_h) \tag{2.38}$$
$$o_{t+1} = g(q_{t_1}) = \sigma_o(U_o q_{t+1} + b_o) \tag{2.39}$$

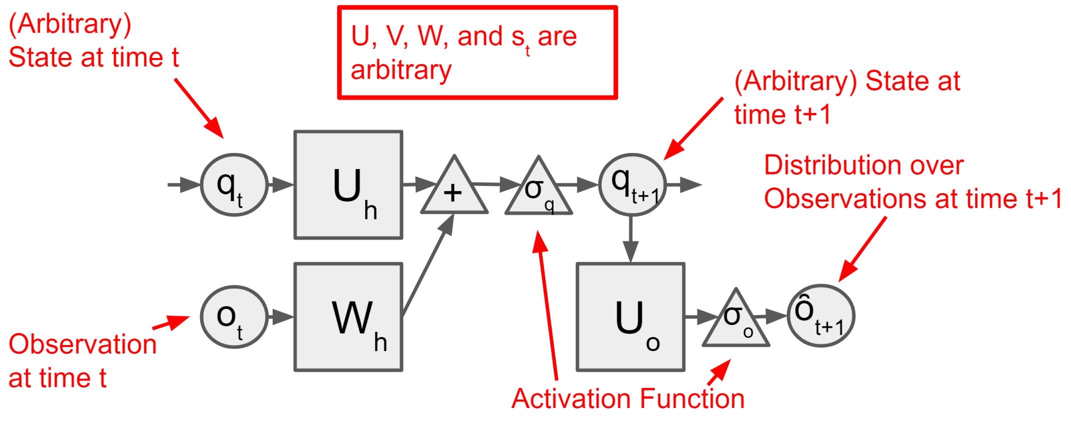

(Arbitrary) State at time t
U, V, W, and s_t are arbitrary
(Arbitrary) State at time t+1
Distribution over Observations at time t+1
q_t
U_h
+
σ_q
q_{t+1}
o_t
W_h
U_o
σ_o
ô_{t+1}
Observation at time t
Activation Function

Figure 2.5: Network Diagram for an Elman Network

which is illustrated in figure 2.5. Due to their simplicity these networks are also known as *simple recurrent networks*.

Historically Elman Networks saw limited success and were often ignored in favor of Bayes Filters, however they set the stage for more sophisticated networks such as Long-Short Term Memory Units and Gated Recurrent Units which resulted in state of the art performance in a number of fields.

2.3.2 Long-Short Term Memory Units

Long-Short Term Memory Units [64] (LSTMs) were the innovation that catapulted RNNs to the forefront of sequence modelling. LSTMs were developed in response to a major problem with Elman (and related) networks: an inability to learn long term dependencies in sequences.

Long term dependencies are one of the most important aspects of sequence modelling. It is easy to build a model which can spot short term correlations between closely positioned observations, however often the most interesting and informative connections are between observations separated by large distances in the sequences. For example, consider the problem of language modelling; we might name a character early on in a paragraph, then refer to “him” or “her” later on, using our implicit knowledge of narrative context to disambiguate the character being referenced.

Now we note that it is theoretically possible for elman networks to model long term dependencies, and in fact it can be done in practice with hand constructed networks, however the problem is that it is difficult to do so when learning these networks from data. RNNs are learned by initializing their parameters at random, then refining the parameters using Gradient Descent in the form of Back-Propagation Through Time. This will be covered in more depth in Section 2.5, however the result is that initializing an Elman network randomly results in a model which tends to quickly forget information about previous observations. This in turn results in a lack of gradients and hence an inability for gradient descent to learn long term dependencies. We note that this problem can be partly addressed in specific cases via orthogonal initialization with good scaling, however this is not a general solution.

LSTMs are an RNN architecture specifically designed to learn long term dependencies. The idea is that LSTMs have the notion of long-term memory built in as the default behavior for random initialization. This means that random initializations will result in useful gradients, allowing us to learn useful models.

An LSTM is defined as follows:

$$f_t = \sigma_f(W_f o_t + U_f q_t + b_f) \tag{2.40}$$
$$i_t = \sigma_i(W_i o_t + U_i q_t + b_i) \tag{2.41}$$
$$j_t = \sigma_o(W_j o_t + U_o q_t + b_o) \tag{2.42}$$
$$c_{t+1} = f_t \circ c_t + i_t \circ \sigma_c(W_c x_t + U_c q_t + b_c) \tag{2.43}$$
$$q_{t+1} = j_t \circ \sigma_h(c_{t+1}) \tag{2.44}$$

The intermediate quantities are given special names related to their purpose in the network:

- c_t is known as the *cell*. c_t should be viewed as a type of internal memory separate to the hidden state and responsible for remembering long-term dependencies.
- f_t is known as the *forget gate*. f_t controls the degree to which the new value of the cell c_{t+1} "forgets" the previous value of the cell c_t.
- i_t is known as the *input gate*. i_t controls the degree to which the current input o_t and current state q_t influence the new value of the cell c_{t+1}
- j_t is known as the *output gate*. j_t controls the degree to which the new value of the cell c_{t+1} is expressed in the new hidden state q_{t+1}.

Gradients don't vanish because remembering is the "default" behavior for a random initialization. LSTMs have been wildly successful, and are by far the most commonly used tool for sequence modelling in modern machine learning applications.

2.3.3 Gated Recurrent Units

LSTMs solved the problem of modelling long dependencies, but they suffer from the problem of objectionable complexity. This can lead to a variety of problems: for example if one uses an LSTM to model a dynamical system it is difficult to use the resulting model to gain any insight into the system. Furthermore if the LSTM fails to learn a good model it is often difficult to determine why.

Gated Recurrent Units [34] (GRUs) emerged as an alternative to LSTMs which preserve the ability to learn long-term dependencies, whilst offering a greatly simplified architecture. A GRU is defined as follows:

$$z_t = \sigma_g(W_z o_t + U_z q_t + b_z) \tag{2.45}$$
$$r_t = \sigma_g(W_r o_t + U_r q_t + b_r) \tag{2.46}$$
$$q_t = (1 - z_t) \circ q_t + z_t \circ \sigma_h(W_q o_t + U_q(r_t \circ q_t) + b_q) \tag{2.47}$$

Similar to LSTMs, the intermediate quantities in GRUs are also given special names:

- z_t is known as the *update gate*. z_t controls the degree to which we update or modify the current state. If z_t is zero no update is performed and the previous state is copied verbatim.
- r_t is known as the reset gate. z_t controls the degree to which we reset or forget the current state. If z_t is 1 we completely reset the state.

The key insight behind GRUs is that the current state q_t influences the next state q_{t+1} in two ways:

- The new state is a weighted combination of the current state and an update term.
- The update term itself is a function of both the input and the current state

This idea is very similar to Kalman Filters: We use the current state (and an observation) to determine an update term, then comput the new state as a mixture of the current state and the update term with mixture weights determined by our "confidence" in the update.

2.4 Generative Learning

Thus far we have spent considering time describing various dynamical system models, but with almost no mention of how we might actually go about learning such models from data. This is somewhat negligent of us, as the study of models and their learning algorithms are intimately connected. In fact much of the development of new models has been driven by a desire to improve their learnability. We already got a taste of this with LSTMs, which were developed to solve the problem of learning long-term dependencies.

Algorithms for learning dynamical system models fall into two main categories: *Generative Approaches* and *Discriminative Approaches*. Generative approaches view the model as a generator of observations and try to optimize the model parameters to make the model distribution match the observed distribution. Discriminative approaches directly learn the filter by minimizing the error resulting from its use in predicting future observations. Generative approaches can be further partitioned into Frequentist and Bayesian. In this thesis we will focus on frequentist and discriminative approaches, however for more information about the Bayesian approaches such as Variational Inference and Sampling please see Foti et al. [48], Frühwirth-Schnatter [49], Rydén et al. [95].

Frequentist Generative approaches include Maximum Likelihood and Method of moments, while discriminative approaches include Back Propagation Through Time (BPTT) and reduction to supervised learning. We now discuss these techniques in more detail.

In the following discussion we assume we have a set of training examples $x_1, ..., x_n$, each of is a sequence of target observations $o_{i,1:T}$, together with a model $\mathcal{M}$ with parameters θ which produces predictions $\tilde{o}_{1:T}$ via filtering:

$$\begin{aligned}\tilde{x}_i &= \tilde{o}_{i,1:T} \\ &= \mathcal{M}(\theta, o_{i,1:T}) \\ &= \mathcal{M}(\theta, x_i)\end{aligned}$$

We begin with a discussion of Generative Approaches to learning. Generative approaches view the model as a generator of observations and try to optimize the model parameters to make the model distribution match the observed distribution. Generative approaches are only appropriate for models which encode valid probability distributions. i.e. Bayes Filters. RNNs must be trained by discriminative approaches (see Section 2.5). Generative approaches have many attractive properties: They result in valid probability distributions, they come with theoretical gaurantees on the resulting model and the rate of convergence, etc. However they also have have several weaknesses, such as being highly specialized and limited to models of modest complexity.

2.4.1 Maximum Likelihood

The traditional way of learning a probability distribution is by optimizing the likelihood of the model given the data. The likelihood of a model is defined as the conditional probability of the training data given the model:

$$l(\theta) = \sum_{i=1}^{n} \Pr(x_i; \theta)$$

where x_i denotes the ith training trajectory. We note that in practice we often work with the log of the likelihood (*log-likelihood*) in place of the likelihood as the likelihood may be extremely small, especially for rare events or long sequences. This can result in numerical problems for the optimizer when using floating point representations. This works because the $\log$ is a monotonically increasing function.

In some **extremely** simple models this expression can be solved analytically, however in virtually all interesting cases we are forced to resort to local optimization. The two most widely used such methods are *Gradient Methods* and *Expectation Maximization*. It is worth noting that local optimizers often perform well in practice, however except in certain extremely specific situations they have no gaurantees on the final model, as they converge only to a local optimum.

2.4.2 Gradient Methods

Gradient Methods are a class of iterative optimization methods for optimizing a function. They proceed by repeatedly updating the parameters by taking a step in the direction of a function of gradient. The simplest and most widely used example is gradient descent, also known as steepest descent, which takes a step in the direction of the gradient:

$$\theta^{(j+1)} = \theta^{(j)} + \alpha^{(j)} \nabla \mathcal{M}(\theta_i, x_i)$$

Where $\alpha^{(j)}$ is known as *learning rate* or *step size* and controls the magnitude of the step taken. We note that there exist a variety of gradient methods other than steepest descent such as ADAM [71], Nesterov Acceleration [85] etc. but they are beyond the scope of this thesis.

Gradient methods are the workhorse of optimization. They have a rich history dating back centuries all the way to the time of Newton. Modern gradient methods are flexible, sophisticated, and can can be quickly and easily deployed using a variety of black box solvers. They also allow a great of flexibility in the loss function.

A key problem with using gradient based methods to optimize a generative model is constraining the resulting model to be a valid probability distribution. For example suppose we are attempting to optimize the likelihood of an HMM using gradient descent. Taking a step in the direction of the gradient may result in either the transmission matrix T or the emission matrix E no longer being stochastic matrices. Indeed it might even result in either matrix containing negative numbers! Clearly such a model is no longer a valid HMM, nor even a valid probability distribution.

Performing gradient descent whilst constraining ourselves to the space of valid models is known as constrained optimization. There are a number of techniques, the simplest of which is called *projected gradient descent* and involves projecting back to the space of valid models after each gradient step. The problem with such techniques is that the projection step may cause large changes in the behavior of the model, and in fact in practice the model resulting from a gradient step followed by a projection step may actually be worse than the original model! These problems tend to limit the practical effectiveness of gradient methods for learning generative models.

We will revisit gradient methods in Section 2.5 when we cover discriminative approaches to learning. In that setting we no longer require our models to be valid probability distributions, allowing us to unleash the full potential of these approaches.

2.4.3 Expectation Maximization

As discussed above, a major problem with using gradient methods to maximize the likelihood is that it can result in invalid models. Expectation Maximization (EM) [43] is a class of iterative methods explicitly designed to optimize the likelihood over the space of valid models. This means that when using EM to learn, for example, an HMM, each step will result in a valid HMM. Expectation Maximization achieves this by explicitly representing the value of the state during learning. This is in contrast to gradient descent which ignores the state and treats the entire model as a compound function which produces some output.

EM consists of two steps for which it is named: An *Expectation* step E and and a *Maximization* step M. In the Expectation step we fix model parameters and find the expected state sequence which lead to the observations conditioned on the current parameters. In the maximization step we fix the states and optimize the model parameters conditioned on the current state sequence. We can think of these two steps as alternately optimizing state/model parameters while fixing the other.

Formally the E and M steps of EM are defined as follows:

- **Expectation (E):** Compute the expected value of the state $b_{1:T}$ conditioned on the model parameters and training data. i.e. $\mathbb{E}_{[}\Pr(b_{1:T} \mid o_{1:T}, \theta^{(i)})]$.
- **Maximization (M):** Find θ which maximizes the expected complete log likelihood. i.e. $\max_\theta \mathbb{E}_{\Pr(b_{1:T}} \mid o_{1:T}, \theta^{(i)}[\log \Pr(o_{1:T}, b_{1:T}, \theta^{(i)})]$

We repeat these two steps until the model converges to a solution. There are many variants of the basic EM algorithm, a few examples include:

- **stochastic EM [31]:** The E step is replaced by sampling of the latent variables
- **generalized EM [82]:** The M step need only improve the value of the expected complete log likelihood, rather than maximize it.

One additional benefit of EM is that there is no step size parameter $\alpha^{(i)}$ to tune. In contrast a major disadvantage of EM is its limited flexibility, as EM lacks access the many sophisticated tools from the optimization literature.

2.4.4 Method of Moments

Method of Moments (MoM) [57] is another classic technique from statistics for learning the parameters of a probability distribution. The idea behind MoM is to express the model parameters as a function of the moments of the distribution. We can use training data to calculate the empirical moments, then substitute these values to obtain estimates of the model parameters.

Formally MoM assumes there exists some θ_0 such that $x \sim \Pr(x;\theta_0)$, and the existence of known functions $f(x)$ and $m(\theta)$ which satisfy the *moment condition*:

$$\mathbb{E}_{x \ \Pr(x;\theta_0)}[f(x)] = m(\theta) \quad \iff \theta = \theta_0 \tag{2.48}$$

Given samples $x_1, ..., x_n \ \Pr(x;\theta_0)$, method of moments computes the empirical moments:

$$\tilde{m} = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

and then solves equation 2.48 for θ i.e.:

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

We note that the above assumes that the datapoints are sampled i.i.d. from some underlying distribution. Clearly this makes little sense in the context of dynamical systems, where data consists of sequences of dependent observations. Fortunately it turns out we only require the milder condition that the empirical moments converge to the true analytical moments. Therefore it is acceptable to replace i.i.d. data with trajectories sampled from a stationary process. If the process is also ergodic, we can compute empirical moments from a single long trajectory. The only downside of this change is that the rate at which the empirical moments converge to the true moments will now depend on the mixing rate of the underlying stochastic process.

MoM is a particularly attractive learning algorithm because under certain mild assumptions it guarantees convergence to a globally optimal solution. This is in contrast to techniques such as gradient descent of the log likelihood and EM which only guarantee a locally optimal solution. Furthermore the resulting learning algorithm is typically simpler than maximum likelihood, particularly EM.

While MoM has had some success for a few specific models (HMMs, PSRs), it has seen little use in practice for two reasons: First, because there is no simple way to take an existing MoM learner and adapt it to a new model architecture. Instead for each new model we must painstakingly re-derive the relationship between the model parameters and the moments. Given how difficult this can be for even simple models, it is little wonder practitioners have shied away from it. Secondly, because MoM is less *Statistically Efficient* than alternative approaches such as EM or Maximum Likelihood. This means that significantly more data is required to obtain a model with comparable performance.

There are two main approaches to learning dynamical systems models via MoM: Subspace Identification and Tensor Decomposition. Both subspace identification and tensor decomposition are sometimes referred to as *Spectral Algorithms* as both make use of the spectrum or eigenvalues of a matrix. This is somewhat unfortunate naming, as it obscures the connection to MoM.

2.4.5 Subspace Identification

Subspace Identification [112] is a technique from the signal processing and robotics literature. Subspace Identification consists of two steps: 1) determining a belief state, and 2) Using MoM to determine the model parameters in terms of that state.

We usually obtain a belief state by factorizing the cross-covariance matrix between past and future observations C_{PF}. We can think of ith, jth entry of C_{PF} as the ability to predict future observation j using past observation i. We can therefore think of C_{PF} as encoding the part of the future which is linearly predictable from the past. By factorizing C_{PF} we obtain a basis for this space, in other words a set of vectors which spans the space of all such predictions. There are several there are many possible factorizations, but popular options include Singular Value Decomposition (SVD) and Canonical Correlation Analysis (CCA).

Once we have obtained (identified) a state space, we can then use MoM to learn the model parameters in terms of that state space. This typically results in solving a series of regression problems.

2.4.6 Tensor Decomposition Methods

Tensor Decomposition [5] refers to a family of methods which combine MoM with tensor CP-Decomposition. The key insight behind this line of work is that several important and well-studied latent variable models – including Gaussian mixture models, HMMs, and Latent Dirichlet allocation – share a certain structure in their low-order moments, and this permits certain tensor decomposition approaches to parameter estimation.

Tensors

A n-mode tensor is an n dimensional array of numbers. Tensors are the n dimensional generalization of matrices; a scalar is a 0 mode tensor, vector a 1 mode tensor, and a matrix is a 2 mode tensor. Formally a real n-mode tensor A is a member of the tensor product of Euclidean spaces $\mathbb{R}^{d_i}$, $i \in [n]$:

$$A \in \bigotimes_{i=1}^{n} \mathbb{R}^{d_i}$$

where $\otimes$ denotes the tensor product.

Tensors can also be viewed as multi-linear maps, mapping tensors in one space to tensors in another space via the Tensor Contraction operation. Given Tensors $A \in \mathbb{R}^{d_1,..,d_n}$ and $B \in \mathbb{R}^{c_1,...,c_m}$ tensor contraction over modes $x \in [1, n]$ and $y \in [1, m]$ where $d_x = c_y$ is defined as:

$$[A \times_{(x,y)} B]_{i_1,...,i_{x-1},i_{x+1}...,i_n,j_1,...,j_{y-1},j_{y+1},...,j_m} = \sum_{z=1}^{d_x} A_{i_1,...,i_{x-1},z,i_{x+1}...,i_n} B_{j_1,...,j_{y-1},z,j_{y+1},...,j_m}$$

This can be viewed as the mult-dimensional generalization of matrix multiplication. The extension to contraction over multiple modes is obvious and straightforwards.

Another commonly used notation for tensor contraction is defined as follows. The contraction of tensor A with matrices $V_1, ..., V_p$ is denoted by $A(V_1, ..., V_p)$:

$$[A(V_1, ..., V_p)]_{i_1,...,i_p} := \sum_{j_1,...,j_p \in [d_1,...,d_n]} A_{j_1,...,j_p} [V_1]_{j_1,i_1} \cdots [V_p]_{j_p,i_p}$$

Note that if A is a matrix ($p = 2$) then we have:

$$A(V_1, V_2) = V_1^\top A V_2$$

We can write matrix-vector multiplication as follows:

$$A(I, v) = Av \in \mathbb{R}^n$$

Where I is the $n \times n$ identity matrix. As a final example of this notation, observe:

$$A(e_{i_1}, ..., e_{i_p}) = A_{i_1,...,i_p}$$

where $e_1, ..., e_p$ are orthogonal basis vectors.

A tensor is *symmetric* if it is invariant under permutations of its indices. i.e. :

$$A_{i_1,...,i_p} = A_{i_\pi i,...,i_\pi p}$$

for any permutation π on $[p]$. This reduces to the normal definition of symmetry for matrices when $p = 2$.

We can also define an inner product between tensors. Given tensors $A, B \in R^{d_1,...,d_n}$ their inner products $\langle A, B\rangle$ is defined as:

$$\langle A, B\rangle := \sum_{j_1,...,j_p \in [d_1,...,d_n]} A_{j_1,...,j_p} B_{j_1,...,j_p}$$

Which induces a norm:

$$\|A\| = \sqrt{\langle A, A\rangle}$$

CP Decomposition

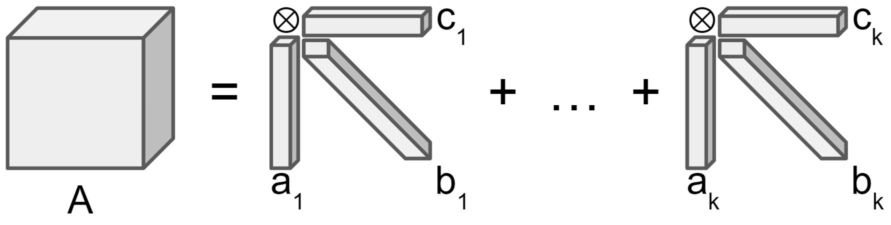


Figure 2.6: CP Decomposition

This induces a Frobenius norm $\|\mathbf{A}\|_F := \sqrt{\langle \mathbf{A}, \mathbf{A}\rangle}$. Another key operation is the tensor product, which is the generalization of the matrix product.

Candecomp/Parafac (CP) decomposition [63] can be viewed as the tensor generalization of the singular value decomposition. Given a tensor $A \in \mathbb{R}^{d_1,..,d_n}$ we can factorize the tensor as a sum of rank one tensors, each of which is a tensor product of vectors. For example factorizing a 3 mode tensor A we obtain the following decomposition:

$$A = \sum_{i=1}^{k} a_i \otimes b_i \otimes c_i$$

Where a_i, b_i and c_i are vectors for all i. We note that contrary to the case of matrices, the rank m of a tensor is presently not well understood, and in fact computing the rank of a tensor, and hence its minimal CP decomposition, is NP hard.

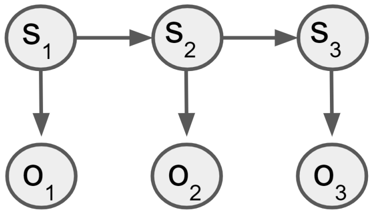


Figure 2.7: A variable HMM. Note how all three observations o_1,o_2, and o_3 depend on state s_2 and hence provide different views of s_2

Tensor Decomposition Methods for Dynamical Systems

They key observation behind tensor methods for learning dynamical system models is that there are multiple different observations, or views, each of which is correlated with the same latent states. Models that have this property are called multi-view models. For example consider the Bayes net for a 3 observation HMM shown in figure 2.7. We note that all three observations o_1,o_2, and o_3 depend on state s_2 and hence provide different views of s_2. We can think about this in terms of the conditional independence assumptions enforced in an HMM. The Current observation is independent of all other observations given the current state, the previous observation is independent of all future observations given the current state, and the next observation is independent of all past observations given the current state. Informally we can think of the state as a bottleneck which controls our ability to exchange information between past present and future. We know that learning something about the past, present, or future observations should tell us something about the other two, and that this information must be passed through the state. Therefore each of these observations tell us something about the value of the state from a different perspective.

Tensor decomposition methods tell us how to combine the information present in each of these disparate views to obtain estimates of the model parameters. A good metaphor for multi-view learning is GPS triangulation to determine position. In this setting a number of different satellites each provide a weak signal containing some information your position. In isolation none of the satellites has the ability to determine your location, but by combining the information from a large number of different satellites, each of which contains slightly different but realted information, we are able to obtain a good estimate.

Formally let o_1, o_2 and o_3 be three views of a state s_2 as shown in Figure 2.7. Assume $s_2 \in \{1, ..., m\}$. Define

$$\omega_j = \Pr(s_2 = j) \tag{2.49}$$

$$\mu_{ij} = \mathbb{E}[o_i \mid s_2 = j) \forall i \in \{1, 2, 3\}, j \in 1, ..., N \tag{2.50}$$

It follows from conditional independence and iterative expectation that:

$$\mathbb{E}[o_1 \otimes o2 \otimes o3] = \sum_{j} = 1^m \omega_j \mu_{1,j} \otimes \mu_{2,j} \otimes \mu_{3,j} \tag{2.51}$$

We would like to obtain the expected value of the belief state s_2. We note that Equation 2.51 has the same form as the CP decomposition, thus the essence of tensor decomposition methods is to estimate the empirical 3rd moment tensor from data, then decompose it into a sum of rank-1 tensors to obtain an estimate of ω_2. The model parameters can then be recovered by matching the results of tensor decomposition to their theoretical values. For more details please see [7].

We note that earlier it was mentioned that finding the CP-decomposition was NP-Hard in general, however it turns out that under certain rank conditions [75] the decomposition is unique up to a permutation of the rank-1 tensors and a re-scaling of their factors. Furthermore the scaling ambiguity can be resolved by enforcing the constraint $\sum_{j=1}^{m} \hat{w} = 1$ as well as additional constraints on the expectations $\hat{\mu}_{ij}$ depending on the model. To this end there are a number of recent tensor decomposition methods with theoretical guarantees to perform CP decomposition such as symmetric tensor power iteration [5], alternating rank-1 updates [7] and simultaneous diagonalization [74].

On a final note, we include under the umbrella of of tensor methods previous work which has the same basic approach but uses the singular value decomposition of matrices instead: e.g. [6, 68]. This work is included in the tensor decomposition framework as a special case.

2.5 Discriminative Learning

Discriminative approaches directly learn a model by minimizing the error resulting from its use in making predictions. They differ from Generative methods as they do not require the model correspond to a valid probability distribution. This allows us far greater flexibility in the types of model architectures.

All discriminative learning begins by defining a loss function. A loss function is a function $l()$ which takes the targets $x = o_{1:T}$ and predictions $\tilde{o}_{1:T}$ as inputs and produces a single real y number as output which represents the performance of the model:

$$y = l(o_{1:T}\tilde{o}_{1:T})$$

By convention the loss is large $\mathcal{M}$ makes poor predictions, and small if $\mathcal{M}$ makes good predictions. Given a training set consisting of a collection of sequences $x_1, ..., x_n$ we define the empirical loss L associated with parameters θ to be:

$$L(\theta) = \sum_{i=1}^{n} l(x_i, \tilde{x}_i)$$

The goal of discriminative learning is to minimize this expression. We often augment our loss with an additional term $R(\theta)$ called a regularizer which is a function of the model parameters, but not the model output. This term is used to control the complexity of the model by acting as loss on the complexity of the model.

The exact form of the loss function $l()$ and regularizer $R()$ depend on a wide range of factors, including the data type, model type, and desired application. In particular the output of a model might be a probability distribution (parameterized or explicit), a point prediction, a category (one hot encoded or otherwise) etc. Each of these leads to different types of loss. If we output a point prediction some common examples of loss include:

- $(x - \tilde{x})^2$: squared loss
- $\mid x - \tilde{x} \mid$: absolute loss

These loss functions measure the distance between the target observation and the predicted observation. If we output a vector of probabilities over discrete observations:

- $\sum_i x_i \log \tilde{x}_i$: Cross entropy loss
- $\max_i \mid x_i - \tilde{x}_i \mid$: Total variation loss
- $\|x - \tilde{x}\|_p$: p-Wasserstein distance

Common regularizers include:

- $\|\theta\|_2^2$: ridge regularization
- $\|\theta\|_1$: lasso regularization

There are a huge variety of other regularizers, many hand crafted for specific applications, however they are beyond the scope of this thesis.

By specifying a loss function we have now reduced the learning problem to an optimization problem.

2.5.1 Back Propagation Through Time

By far the most common way to optimize equation 2.5 is by gradient descent. Gradient descent in the context of dynamical systems is given a special name: Back Propagation Through Time (BPTT) [119]. In order to perform BPTT we first unfold the recursive computation graph to compute the belief state at each step. We then use backpropagation to compute the gradients via the following recursion:

$$\frac{\partial L}{\partial q_t} = \frac{\partial L}{\partial \tilde{x}_t}\frac{\partial \tilde{x}_t}{\partial q_t} + \begin{cases} \frac{\partial L}{\partial q_{t+1}}\frac{\partial q_{t+1}}{\partial q_t} & \text{if } t < T \\ 0 & \text{if } t = T \end{cases} \tag{2.52}$$

From the chain rule we obtain:

$$\frac{\partial L}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial L}{\partial q_t}\frac{\partial q_t}{\partial \theta} + \frac{\partial L}{\partial \tilde{x}}\frac{\partial \tilde{x}}{\partial \theta}$$

We note that BPTT for RNNs does not encounter many of the issues that plagued gradient descent of the likelihood function when optimizing Bayes Filters. Specifically BPTT of RNNs is an unconstrained optimization, in contrast to gradient descent of the likelihood function, which was a constrained optimization. This is because any set of parameters constitutes a valid RNN, wheras when working with Bayes Filters we were forced to optimize over the space of parameters corresponding to valid probability distributions.

2.6 Discussion

As the previous sections show there are a wide variety of models for dynamical systems, and an equally diverse set of methods for learning them. It is the (thankless) job of the machine learning practitioner to intelligently choose a combination of model and learning algorithm given a problem specification. Unfortunately this job is extremely difficult, even for expert practitioners, due to a number of factors. First there are the large number of moving parts; besides the model and learning algorithm we must also decide on the hyperparameters, the loss function, etc. Furthermore the model and the learning algorithm cannot be chosen independently, and instead must be chosen together. For instance if you decide to use an RNN model you must use a discriminative classifier. This problem is further exacerbated by many of these techniques being developed by different fields and often using diverse and contradictory terminology/notation in the relevant literature.

But what if we didn't have to chose? What if we could develop models which combine all these advantages in a single package? In this thesis we study the idea of developing hybrid models which unify many of the models covered in the previous section. Specifically we want a model which:

- Works for a wide range of common data types
- Is generative, i.e. corresponds to a probability distribution over observations
- Has a strong statistical theory
- Can be initialized via a globally optimal method of moments algorithm
- Can be refined using BPTT
- Performs well on a wide range of practical application
- Scales to large problems
- Has a simple functional form

In the remainder of this thesis we work towards this goal.

Part II

Unifying Method of Moments Learning

Chapter 3

Method of Moments Learning for Uncontrolled Systems

In this chapter we take our first steps towards developing a unified model for dynamical systems. One desirable property for such a unified model is the ability to perform learning via MoM. MoM is special, in that it offers something virtually other learning algorithm for dynamical systems can claim – guaranteed convergence to a globally optimal solution. The key difficulty in developing a model which supports MoM learning is that the class of such models is not well defined. We know that certain specific models admit MoM learning, but in each case the learning algorithm has been hand derived from scratch. If we want to develop a unified model which supports MoM learning we must first improve our understanding of this class of algorithms, and the model class they support.

This chapter addresses this problem by developing a novel framework which unifies the many disparate MoM algorithms. This framework consists of a model class and a learning algorithm. The model class is *Predictive State Models (PSMs)*, a Bayes filter that utilizes predictive states. The learning algorithm is two stage-regression (2SR), which reduces MoM to supervised learning. We focus for now on uncontrolled systems in order to simplify development and presentation of these ideas. We extend this work to controlled systems in Chapter 4.

We propose a framework which reduces MoM learning to solving three supervised learning problems using the idea of instrumental-variable regression. This allows us to directly apply the rich literature on supervised learning methods to incorporate many types of prior knowledge about problem structure. We show that this framework subsumes a wide variety of existing spectral algorithms. We refer to the class of the algorithms subsumed by this framework as *predictive belief methods* since they rely on representing the system state in terms of predictions of observed quantities.

The remainder of this chapter is organised as follows: In Section 3.1 we define the class of Predictive State Model, and in Section 3.2 we introduce the two-stage algorithm. In Section 3.3 we show that many previous instances of MoM learning are subsumed by our framework. In Section 3.4 we present a theoretical analysis of two-stage regression. In section 3.5 we present experimental results demonstrating how this framework can be used to design and learn novel filters. A discussion of related work can be found in Section 3.6, conclusions in Section 3.7, and proofs of key results in Appendix 3.A.

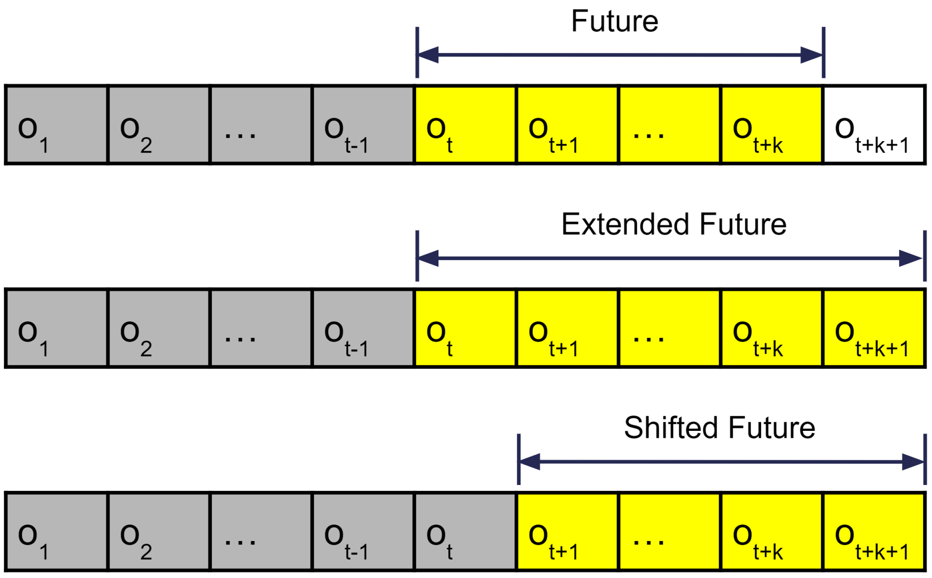


Figure 3.1: Illustration of the concepts of Future, Extended Future, and Shifted Future (for a fixed, finite horizon k at time t).

3.1 Predictive State Models

Our discussion of PSMs hinges on the concepts of Future observations, Past observations, and Present observations. Imagine that we are performing filtering on an observation sequence. Past observations are observations which have already occurred, Future observations are observations which have yet to occur, and the present observation is a new observation which we just obtained, but has not yet been used to refine our estimate of the state.

Similar to PSRs, Predictive State Models (PSMs) are based on the concept of a predictive belief state where the state corresponds to a belief over sufficient statistics of features of future observations. We now define three quantities which will be used to define PSMs: *Future features*, *Extended Future Features*, and *Shifted Future Features*. Future Features are defined as a function ψ of a window of future observations:

$$\psi_t = \psi(o_{t:t+k})$$

The extended future features are similar to the future features, but we extend the window to encompass an additional observation:

$$\xi_t = \xi(o_{t:t+k+1})$$

The shifted future is just the future features for a window of observations one time step in the future:

$$\psi_{t+1} = \psi(o_{t+1:t+k+1})$$

These ideas are illustrated in figure 3.1. We note that it's really not necessary to use a window, but we chose to do so for clarity of presentation as it leads to substatially simpler notation.

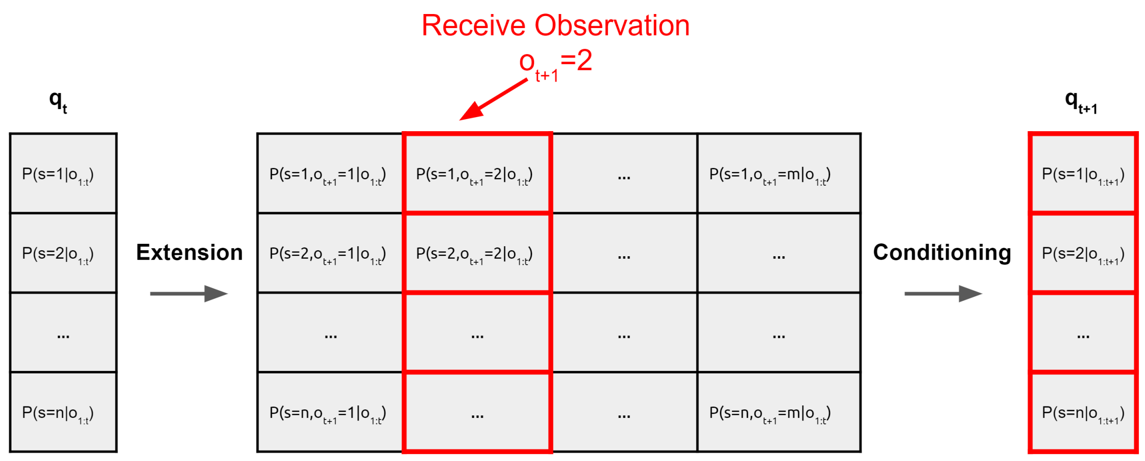

Receive Observation
$o_{t+1}=2$
q_t
q_{t+1}
Extension
Conditioning
P(s=1|$o_{1:t}$)
P(s=2|$o_{1:t}$)
P(s=n|$o_{1:t}$)
P(s=1,o_{t+1}=1|$o_{1:t}$)
P(s=1,o_{t+1}=2|$o_{1:t}$)
P(s=1,o_{t+1}=m|$o_{1:t}$)
P(s=2,o_{t+1}=1|$o_{1:t}$)
P(s=2,o_{t+1}=2|$o_{1:t}$)
P(s=n,o_{t+1}=1|$o_{1:t}$)
P(s=n,o_{t+1}=m|$o_{1:t}$)
P(s=1|$o_{1:t+1}$)
P(s=2|$o_{1:t+1}$)
P(s=n|$o_{1:t+1}$)

Figure 3.2: Illustration of the Expansion/Conditioning decomposition of the Bayes Filter update rule.

The key insight behind PSMs is that we can reformulate the Bayes Filter state update (Equation 2.4) in terms of these quantities. Consider Equation 2.2:

$$q_{t+1} = \frac{\Pr(s_{t+1}, o_{t+1} \mid o_{1:t})}{\Pr(o_{t+1} \mid o_{1:t})}$$

We see that the numerator of this expression is exactly the extended future. One way of thinking about this is we extended the current marginal distribution over states to obtain a joint distribution over states and observations. The denominator of this expression acts to condition this joint distribution to obtain a conditional distribution corresponding to the next state. Therefore we can think of the Bayes Filter state update as consisting of two steps:

- **Extension** where we obtain a joint distribution over states and observations
- **Conditioning** where we condition on the current observation to obtain the updated belief state

This decomposition is illustrated in figure 3.2. We define the state q_t at time t of a PSM as the expected value of ψ_t conditioned on past observations:

$$q_t = \mathbb{E}[\psi_t \mid o_{1:t-1}]$$

ψ is *sufficient* if $\mathbb{E}[\psi_t \mid o_{1:t-1}]$ is a sufficient state representation. Two common choices of ψ are 1) an indicator vector, or one-hot vector (for discrete models, e.g. 1-observable HMM [68]) and 2) a stack of the next n observations (e.g. linear Gaussian systems [114]). It follows immediately that the state at time $t+1$ corresponds to the shifted future:

$$q_{t+1} = \mathbb{E}[\psi_{t+1} \mid o_{1:t}]$$

Unfortunately it is not possible to learn a mapping directly from the future to the shifted future from training data for reasons which will be made clear in Section 3.2. This is where the notion of the extended future comes in handy: We use the extended future to define the notion of an extended state p_t:

$$p_t = \mathbb{E}[\xi_t \mid o_{1:t-1}]$$

We then define two mappings, one from q_t to p_t and a second from p_t to q_{t+1}. By doing so we circumvent the problems associated with learning a mapping directly from q_t to q_{t+1}. Formally a PSM is defined as follows:

Definition 1. *A dynamical system is said to conform to a predictive state model (PSM) if it satisfies the following properties for some future features ψ and extended future features ξ:*

- *For each time t there exists a predictive belief state $q_t = \mathbb{E}[\psi_t \mid o_{1:t-1}]$ which constitutes a sufficient state representation*
- *For each time t there exists an extended state $p_t = \mathbb{E}[\xi_t \mid o_{1:t-1}]$*
- *There exists a filtering function f_{filter} such that, for each time t:*

$$q_{t+1} = f_{filter}(p_t, o_t)$$

  f_{filter} is typically non-linear, but is known in advance.
- *There exists a linear map W_{system} such that, for each time t:*

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

This definition implies that the recursive state update for PSMs is:

$$q_{t+1} = f_{\text{filter}}(W_{\text{system}} q_t, o_t)$$

This filter is a Bayes Filter that follow sthe two-step update described in Section 3.2.

- 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 need only learn the expansion operator W_{system} and the initial state q_0. The linearity of W_{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.

3.2 Two-Stage Regression

The main idea behind Two-Stage Regression is that it suffices for the purpose of prediction to learn the mapping W from q_t to p_t (as well as the initial distribution of $q_{t=1}$). Unfortunately, we do not observe q_t or p_t but noisy versions thereof, namely ψ_t and ξ_t. Moreover, due to the overlap between observation windows, the noise terms on ψ_t and ξ_t are correlated. This noise correlation means that "naïve" linear regression (using samples of ψ_t and ξ_t in place of q_t and p_t) will give a biased estimate of W.

To counteract this bias, we employ instrumental regression [89, 109]. Instrumental regression uses *instrumental variables* that are correlated with the input q_t but not with the noise $\epsilon_{t:t+k}$. 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. Since past observations $o_{1:t-1}$ do not overlap with future or extended future windows, they are not correlated with the noise $\epsilon_{t:t+k+1}$. Therefore, we can use *history features* $h_t \equiv h(o_{1:t-1})$ as instrumental variables.

Note that we assume that the process noise is white (i.e. independent at each time step). With this assumption, one can think of a graphical model at time t where observed future features are a common child of future noise, past features and past noise. This v-structure implies that past observations are independent of future noise.

The key idea behind instrumental regression is that if two random variables are related by a linear map W, their conditional expectations are related by the same linear map W. By taking conditional expectation w.r.t instrumental variables, we can eliminate the noise while preserving the linear map that we are attempting to estimate.

In more detail, by taking the expectation of (3.1) over h_t we obtain an instrument-based moment condition (for all t)

$$\begin{aligned} \mathbb{E}[p_t \mid h_t] &= \mathbb{E}[Wq_t \mid h_t] \\ \mathbb{E}[\mathbb{E}[\xi_t \mid o_{1:t-1}] \mid h_t] &= W\mathbb{E}[\mathbb{E}[\psi_t \mid o_{1:t-1}] \mid h_t] \\ \mathbb{E}[\xi_t \mid h_t] &= W\mathbb{E}[\psi_t \mid h_t] \end{aligned} \tag{3.2}$$

Assuming that there are enough independent dimensions in h_t that are correlated with q_t, we maintain the rank of the moment condition when moving from (3.1) to (3.2), and we can recover W 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.

So, we first use regression models to estimate $\mathbb{E}[\psi_t \mid h_t]$ and $\mathbb{E}[\xi_t \mid h_t]$, effectively *denoising* the training examples and then use these estimates to compute W be finding the least squares solution to (3.2), which can be thought of as performing linear regression on denoised training data.

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 $\xi_t = \xi(o_{t:t+k})$. ψ_t must be a sufficient statistic for $\mathbb{P}(o_{t:t+k-1} \mid o_{1:t-1})$. ξ_t must satisfy
  - $\mathbb{E}[\psi_{t+1} \mid o_{1:t-1}] = f_{\text{predict}}(\mathbb{E}[\xi_t \mid o_{1:t-1}])$ for a known function f_{predict}.
  - $\mathbb{E}[\psi_{t+1} \mid o_{1:t}] = f_{\text{filter}}(\mathbb{E}[\xi_t \mid o_{1:t-1}], o_t)$ for a known function f_{filter}.
- **S1A (Stage 1A) Regression:** Learn a (possibly non-linear) regression model to estimate $\bar{\psi}_t \equiv \mathbb{E}[\psi_t \mid h_t]$. The training data for this model are (h_t, ψ_t) across time steps t.[1]
- **S1B Regression:** Learn a (possibly non-linear) regression model to estimate $\bar{\xi}_t \equiv \mathbb{E}[\xi_t \mid h_t]$. The training data for this model are (h_t, ξ_t) across time steps t.
- **S2 Regression:** Use the feature expectations estimated in the previous two steps to train a model to predict $\bar{\xi}_t = W\bar{\psi}_t$, where W is a linear operator. The training data for this model are estimates of $(\bar{\psi}_t, \bar{\xi}_t)$ across time steps t obtained from S1 steps.

[1]Our bounds assume 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.

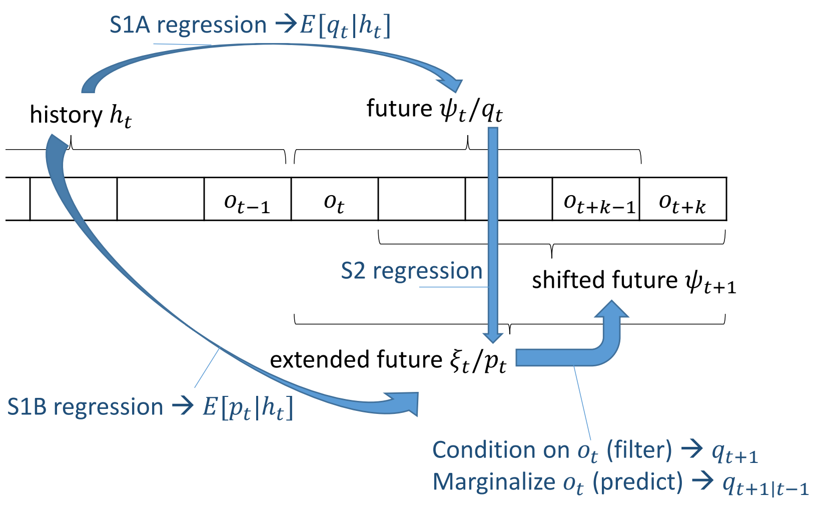

S1A regression → $E[q_t|h_t]$
history h_t
future ψ_t/q_t
o_{t-1}
o_t
o_{t+k-1}
o_{t+k}
S2 regression
shifted future ψ_{t+1}
extended future ξ_t/p_t
S1B regression → $E[p_t|h_t]$
Condition on o_t (filter) → q_{t+1}
Marginalize o_t (predict) → $q_{t+1|t-1}$

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

- **Initial State Estimation:** Estimate an initial state $q_1 = \mathbb{E}[\psi_1]$ by averaging ψ_1 across several example realizations of our time series.[2]
- **Inference:** Starting from the initial state q_1, we can maintain the predictive belief $q_t \equiv \mathbb{E}[\psi_t \mid o_{1:t-1}]$ through filtering: given q_t we compute $p_t \equiv \mathbb{E}[\xi_t \mid o_{1:t-1}] = Wq_t$. Then, given the observation o_t, we can compute $q_{t+1} = f_{\text{filter}}(p_t, o_t)$. Or, in the absence of o_t, we can predict the next state $q_{t+1|t-1} = f_{\text{predict}}(p_t)$. Finally, by definition, the predictive belief q_t is sufficient to predict $\mathbb{P}(o_{t:t+k-1} : o_{1:t-1})$.

The process of learning and inference is depicted in Figure 3.3. Modeling assumptions are reflected in the choice of the statistics ψ, ξ and h as well as the regression models in stages S1A and S1B. Table 3.1 demonstrates that we can recover existing spectral algorithms for dynamical systems learning using linear S1 regression. The two stage framework not only provides a unifying view of some of the successful dynamical systems learning algorithms but also paves the way for extending them in a theoretically justified manner, as we demonstrate in the experiments.

3.3 Connections with prior work

In this section we provide examples of mapping some of the successful dynamical system learning algorithms to our framework.

[2]This is the only step that needs multiple realizations of our time series. If only a single long realization is available, we need additional assumptions to be able to estimate an initial state; for example, if we assume stationarity, we can set the initial state to be the empirical average vector of future features, $\frac{1}{T}\sum_{t=1}^{T}\psi_t$.

Model/Algorithm	future features ψ_t	extended future features ξ_t	f_{filter}
Spectral Algorithm for HMM [65]	$U^\top e_{o_t}$ where e_{o_1} is an indicator vector and U spans the range of q_t (typically composed of top m left singular vectors of the joint probability table $P(o_2, o_1)$)	$U^\top e_{o_{t+1}} \otimes e_{o_t}$	Estimate a state normalize from S1A output states.
SSID for Kalman filters (time dependent gain)	x_t and $x_t \otimes x_t$, where $x_t = U^\top o_{t:t+k-1}$ for a matrix that U spans the range of q_t (typically composed of top m left singular vectors of the covariance matrix $\text{Cov}(o_{t:t+k-1}, o_{t-k:t-1})$)	y_t and $y_t \otimes y_t$, where y_t is formed by stacking $U^\top o_{t+1:t+k}$ and o_t.	p_t specifies a Gaussian distribution where conditioning on o_t is straightforward.
SSID for stable Kalman filters (constant gain)	$U^\top o_{t:t+k-1}$ (U obtained as above)	o_t and $U^\top o_{t+1:t+k}$	Estimate steady-state covariance by solving Riccati equation [112]. p_t together with the steady-state covariance specify a Gaussian distribution where conditioning on o_t is straightforward.
Uncontrolled HSE-PSR [28]	Evaluation functional $k_s(o_{t:t+k-1}, .)$ for a characteristic kernel k_s	$k_o(o_t, .) \otimes k_o(o_t, .)$ and $\psi_{t+1} \otimes k_o(o_t, .)$	Kernel Bayes rule [50]

Table 3.1: Examples of existing spectral algorithms reformulated as two-stage instrument regression with linear S1 regression. Here $o_{t_1:t_2}$ is a vector formed by stacking observations o_{t_1} through o_{t_2} and $\otimes$ denotes the outer product. Details and derivations can be found in the supplementary material.

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 [65]. 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:

$$\hat{\mathbb{E}}[\psi_2 | h_2] = \hat{\Sigma}_{o_2 o_1} \hat{\Sigma}_{o_1}^{-1} h_2 = \hat{P}_{2,1} \hat{P}_{1,1}^{-1} h_t \equiv \hat{W}_{2,1} h_2 \tag{3.3}$$

$$\hat{\mathbb{E}}[\xi_2 | h_2] = \hat{\Sigma}_{o_{2:3} o_1} \hat{\Sigma}_{o_1}^{-1} h_2 = \hat{P}_{2:3,1} \hat{P}_{1,1}^{-1} h_2 \equiv \hat{W}_{2:3,1} h_2, \tag{3.4}$$

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

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

for given samples of h_2, which gives

$$\begin{aligned}\hat{W} &= \hat{W}_{2:3,1}\hat{\mathbb{E}}[e_{o_1}e_{o_1}^\top]\hat{W}_{2,1}^\top\left(\hat{W}_{2,1}\hat{\mathbb{E}}[e_{o_1}e_{o_1}^\top]\hat{W}_{2,1}^\top\right)^{-1}\\ &= \left(\hat{P}_{2:3,1}\hat{P}_{1,1}^{-1}\hat{P}_{2,1}^\top\right)\left(\hat{P}_{2,1}\hat{P}_{1,1}^{-1}\hat{P}_{2,1}^\top\right)^{-1}\\ &= \hat{P}_{2:3,1}\left(\hat{P}_{2,1}\right)^{-1} \qquad (3.5)\end{aligned}$$

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 [65]:

For a given value x of o_2, define

$$B_x = u_x^\top\hat{W} = u_x^\top\hat{P}_{2:3,1}\left(\hat{P}_{2,1}^\top\right)^{-1}, \qquad (3.6)$$

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, $u_x = \delta_x \otimes_k I_{|\mathcal{O}|}$. [3].

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

with a normalization constant given by

$$\frac{1}{1^\top B_{o_t}q_t} \qquad (3.7)$$

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 [65] 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

$$\begin{aligned}\hat{W}^{rr} &= (I_{|\mathcal{O}|} \otimes_k U^\top)\hat{W}_{2:3,1}\hat{\mathbb{E}}[e_{o_1}e_{o_1}^\top]\hat{W}_{2,1}^\top U\left(U^\top\hat{W}_{2,1}\hat{\mathbb{E}}[e_{o_1}e_{o_1}^\top]\hat{W}_{2,1}^\top U\right)^{-1}\\ &= (I_{|\mathcal{O}|} \otimes_k U^\top)\hat{P}_{2:3,1}\hat{P}_{1,1}^{-1}VS\left(SV^\top\hat{P}_{1,1}^{-1}VS\right)^{-1}\\ &= (I_{|\mathcal{O}|} \otimes_k U^\top)\hat{P}_{2:3,1}\hat{P}_{1,1}^{-1}V\left(V^\top\hat{P}_{1,1}^{-1}V\right)^{-1}S^{-1} \qquad (3.8)\end{aligned}$$

[3]Following the notation used in [65], $u_x^\top\hat{P}_{2:3,1} \equiv \hat{P}_{3,x,1}$

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 (3.8) gives

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

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

$$B_x = u_x^\top \hat{W}^{rr} \to U^\top P_{3,x,1}\left(U^\top P_{2,1}\right)^+ \tag{3.9}$$

This is exactly the observation operator obtained in [65]. However, instead of using 3.8, they use 3.9 with $P_{3,x,1}$ and $P_{2,1}$ replaced by their empirical estimates.

Note that for a state $b_t = \mathbb{E}[\psi_t|o_{1:t-1}]$, $B_xb_t = P(o_t|o_{1:t-1})\mathbb{E}[\psi_{t+1}|o_{1:t}] = P(o_t|o_{1:t-1})b_{t+1}$. 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_∞ we solve the aforementioned condition for states estimated from all possible values of history features h_t. This gives,

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

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 [65].

3.3.2 Stationary Kalman Filter

A Kalman filter is given by

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

We consider the case of a *stationary* filter where $\Sigma_t \equiv \mathbb{E}[s_t s_t^\top]$ is independent of t. We choose our statistics

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

Where a window of observations is represented by stacking individual observations into a single vector. It can be shown [20, 112] that

$$\mathbb{E}[s_t|h_t] = \Sigma_{s,h}\Sigma_{h,h}^{-1}h_t$$

and it follows that

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

where Γ is the extended observation operator

$$\Gamma \equiv \begin{pmatrix} O \\ OT \\ \vdots \\ OT^F \end{pmatrix}, \Gamma_+ \equiv \begin{pmatrix} O \\ OT \\ \vdots \\ OT^{F+1} \end{pmatrix}$$

It follows that F and H must be large enough to have $\mathrm{rank}(W) = n$. Let $U \in \mathbb{R}^{mF\times n}$ 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

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

which matches the instrumental regression framework. For the steady-state case (constant Kalman gain), one can estimate Σ_ξ given the data and the parameter W by solving Riccati equation as described in [112]. $\mathbb{E}[\xi_t|o_{1:t-1}]$ and Σ_ξ then specify a joint Gaussian distribution over the next $F+1$ observations where marginalization and conditioning can be easily performed.

We can also assume a Kalman filter that is not in the steady state (i.e. the Kalman gain is not constant). In this case we need to maintain sufficient statistics for a predictive Gaussian distribution (i.e. mean and covariance). Let vec denote the vectorization operation, which stacks the columns of a matrix into a single vector. We can stack h_t and $\mathrm{vec}(h_t h_t^\top)$ to into a single vector that we refer to as 1st+2nd moments vector. We do the same for future and extended future. We can, in principle, perform linear regression on these 1st+2nd moment vectors but that requires an unnecessarily large number of parameters. Instead, we can learn an S1A regression function of the form

$$\mathbb{E}[\psi_t|h_t] = W_1 h_t \tag{3.10}$$

$$\mathbb{E}[\psi_t\psi_t^\top|h_t] = W_1 h_t h_t^\top W_1 + R \tag{3.11}$$

$$\tag{3.12}$$

Where R is simply the covariance of the residuals of the 1st moment regression (i.e. covariance of $r_t = \psi_t - \mathbb{E}[\psi_t|h_t]$). This is still a linear model in terms of 1st+2nd moment vectors and hence we can do the same for S1B and S2 regression models. This way, the extended belief vector p_t (the expectation of 1st+2nd moments of extended future) fully specifies a joint distribution over the next $F+1$ observations.

3.3.3 HSE-PSR

We define a class of non-parametric two-stage instrumental regression models. By using conditional mean embedding [105] as S1 regression model, we recover a single-action variant of HSE-PSR [28]. Let $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ denote three reproducing kernel Hilbert spaces with reproducing kernels $k_{\mathcal{X}}, k_{\mathcal{Y}}$ and $k_{\mathcal{Z}}$ respectively. Assume $\psi_t \in \mathcal{X}$ and that $\xi_t \in \mathcal{Y}$ is defined as the tuple $(o_t \otimes o_t, \psi_{t+1} \otimes o_t)$. Let $\mathbf{\Psi} \in \mathcal{X} \otimes \mathbb{R}^N$, $\mathbf{\Xi} \in \mathcal{Y} \otimes \mathbb{R}^N$ and $\mathbf{H} \in \mathcal{Z} \otimes \mathbb{R}^N$ be operators that represent training data. Specifically, ψ_s, ξ_s, h_s are the s^{th} "columns" in $\mathbf{\Psi}$ and $\mathbf{\Xi}$ and $\mathbf{H}$ 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 β_s and γ_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

$$W = \mathbf{\Xi}\mathbf{\Gamma}(\mathbf{B}^\top G_{\mathcal{X},\mathcal{X}}\mathbf{B} + \lambda I_N)^{-1}\mathbf{B}^\top \mathbf{\Psi}^* \tag{3.13}$$

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

$$\mathbf{B} = (G_{\mathcal{Z},\mathcal{Z}} + \lambda I_N)^{-1} G_{\mathcal{Z},\mathcal{Z}} \tag{3.14}$$

HSE-PSR learning is similar to this setting, with ψ_t being a conditional expectation operator of test observations given test actions. For this reason, kernel ridge regression is replaced by application of kernel Bayes rule [50].

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

$$\hat{\Sigma}_{\psi_{t+1} o_t | h_t} = \sum_{s=1}^{N} \alpha_{t,s} \psi_{s+1} \otimes o_s = \mathbf{\Psi}' \mathrm{diag}(\alpha_t) \mathbf{O}^*$$
$$\hat{\Sigma}_{o_t o_t | h_t} = \sum_{s=1}^{N} \alpha_{t,s} o_s \otimes o_s = \mathbf{O} \mathrm{diag}(\alpha_t) \mathbf{O}^*$$

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 [50] to condition on o_t which gives

$$q_{t+1} = \hat{E}[\psi_{t+1} \mid h_t] = \hat{\Sigma}_{\psi_{t+1} o_t | h_t} (\hat{\Sigma}_{o_t o_t | h_t} + \lambda I)^{-1} o_t. \tag{3.15}$$

Replacing o_t in (3.15) with its conditional expectation $\sum_{s=1}^{N} \alpha_s o_s$ corresponds to marginalizing over o_t (i.e. prediction). A stable Gram matrix formulation for (3.15) is given by [50]

$$\begin{aligned} & q_{t+1} \\ & \quad = \mathbf{\Psi}' \mathrm{diag}(\alpha_t) G_{\mathcal{O},\mathcal{O}} ((\mathrm{diag}(\alpha_t) G_{\mathcal{O},\mathcal{O}})^2 + \lambda N I)^{-1} \\ & \qquad .\mathrm{diag}(\alpha_t) \mathbf{O}^* o_t \\ & \quad = \mathbf{\Psi}' \tilde{\alpha}_{t+1}, \end{aligned} \tag{3.16}$$

which is the state update equation in HSE-PSR. 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 (3.13).

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 $\mathbb{E}[y_t \mid z_t]$. 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:

$$\hat{W}_\lambda = \left(\sum_{t=1}^{T} \hat{y}_t \otimes \hat{x}_t \right) \left(\sum_{t=1}^{T} \hat{x}_t \otimes \hat{x}_t + \lambda I_{\mathcal{X}} \right)^{-1} \tag{3.17}$$

where $\otimes$ denotes tensor 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 2 (S1 Regression Bound)**.** *For any $\delta > 0$ and $N \in \mathbb{N}^+$ the* ***S1 regressino bound*** $\eta_{\delta,N}$ *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 (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 $1 \leq t \leq N$:*

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

Proposition 4. *Let $\tilde{\eta}_{\delta,N}$ be a uniform S1 regression bound that satisfies Definition 3. Assuming that $||\bar{y}_t||_{\mathcal{Y}} < c$ and $||\bar{x}_t||_{\mathcal{X}}$, then*

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

satisfies Definition 2

A consistent learning algorithm requires that, for each fixed δ, $\lim_{N\to\infty} \eta_{\delta,N} = 0$. 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 [28]. 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 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 square (i.e. setting $\lambda = 0$) will give the same bound, but with λ in the firsrt two terms replaced by the minimum eigenvalue of $\mathcal{C}_{\bar{x}}$, and the last term dropped.

Theorem 5. *Assume that $\|\bar{x}\|_{\mathcal{X}}, \|\bar{x}\|_{\mathcal{Y}} < c < \infty$ almost surely. Assume W is a Hilbert-Schmidt operator, let $\hat{W}_\lambda$ be as defined in* (3.17)*. Then with probability at least $1-\delta$ for each $x_{\text{test}} \in \overline{\mathcal{C}_{\bar{x}}}$ 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:*

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

A variation of Theorem 5 applies if the true model is not linear. In this case the reference value W is a linear predictor of y given $\bar{x}$ that minimizes MSE.

It is important to note that Theorem 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_{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 6 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 6 is the assumption that f_{filter} is L-Lipchitz, which essentially means that the models estimated probability for o_t is bounded below. A similar condition was assumed in [68] for HMMs. To guarantee this property depends heavily on the filtering function. Therefore, Lemma 6 provides suggestive evidence rather than a guarantee that our learned dynamical system will predict well.

Lemma 6. *For a test sequence $o_{1:T}$, let $\hat{q}_t$ denote the estimated state given $o_{1:t-1}$. Let $\tilde{q}_t$ denote the projection of $\hat{q}_t$ onto $\overline{\mathcal{R}(\mathcal{C}_{\bar{x}})}$. Assume that f_{filter} is L-Lipchitz continuous on P_t and that $f_{\text{filter}}(P_t, o_t) \in \overline{\mathcal{R}(\mathcal{C}_{\bar{x}})}$ for any $P_t \in \overline{\mathcal{R}(\mathcal{C}_{\bar{y}})}$. Given the assumptions in theorem 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 ϵ_t dimishes at the same rate as $||\epsilon_t||_{\mathcal{X}}$.

3.5 Experiments

3.5.1 Learning A Knowledge Tracing Model

In this section we demonstrate that we can learn a hidden Markov model using the two stage regression framework. In addition we show that we can use non-linear S1 regression models to reduce the number of parameters we need to learn, resulting in better empirical prediction accuracy compared to linear models while still maintaining consistency.

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 [41], 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^{th} question in a sequence of questions that cover the said KC. Figure 3.4 summarizes transitions and emissions in that model. The events denoted by guessing, slipping, learning and forgetting typically have relatively low probabilities.

Data Description

We evaluate the model using "Geometry Area (1996-97)" data available from DataShop[73]. 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 iff the student entered the correct answer on the first try, without requesting any hints from the help system. 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.

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,

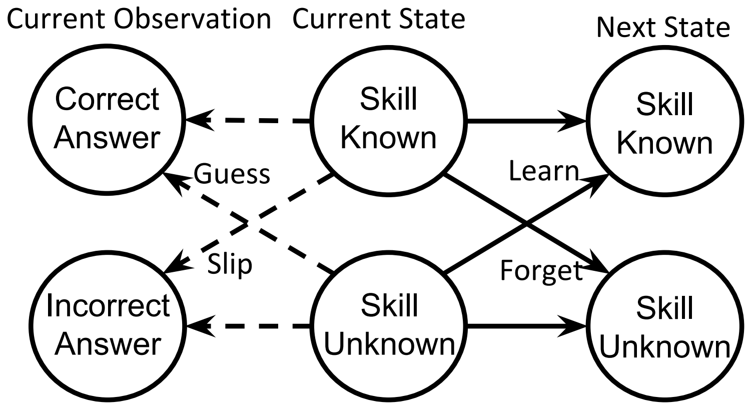


Figure 3.4: Transitions and observation emissions of the BKT model. (Each node represents a possible *value* of the state/observation). Solid arrows represent transitions while dashed arrows represent emissions.

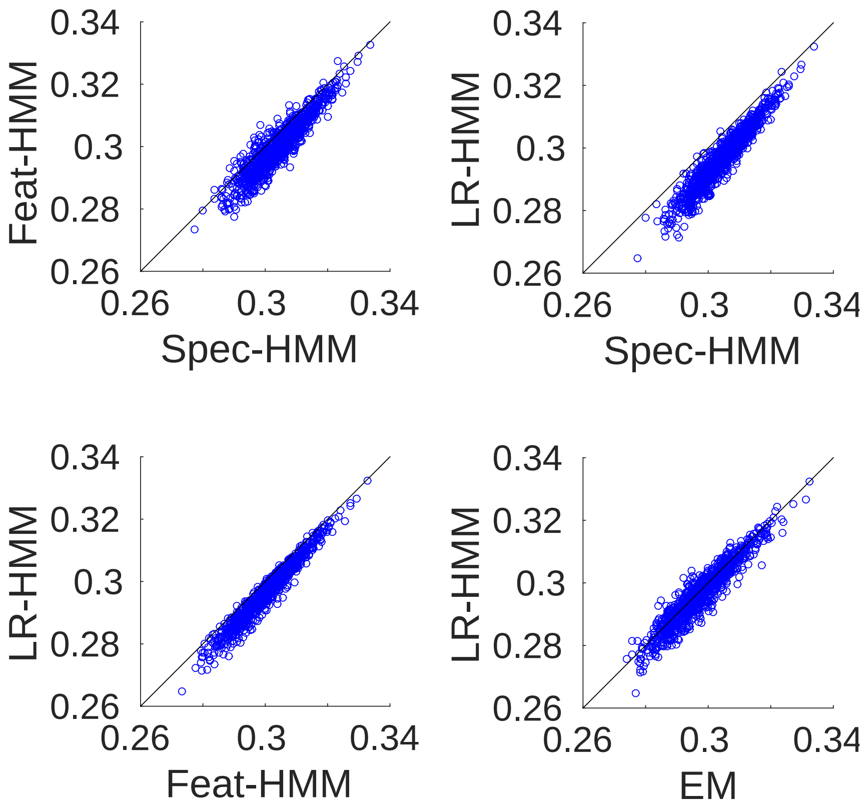
which results in the following statistics: $\psi_t = o_t$ and $\xi_t = o_t \otimes_k o_{t+1}$,

where o_t is represented by a 2 dimensional indicator vector and $\otimes_k$ denotes the Kronecker product. Given these statistics, $p_t = \mathbb{E}[\xi_t | o_{1:t-1}]$ is a joint probability table of $o_{t:t+1}$ from which conditioning on o_t (filtering) and marginalizing over o_t (prediction) are simple operations. It remains to choose the history features h_t and the S1 regression model. In the supplementary material, we show that if use $h_t = o_{t-1}$ and linear regression as S1 regression model, the resulting algorithm is equivalent to spectral HMM method of [65]. We use this model (denoted by "Spec-HMM") as a baseline.

Our second baseline (denoted by "Feat-HMM") is feature-based spectral HMM [101]. This can be thought of as using linear S1 regression with arbitrary history features. We consider a window of previous observations of length $b > 1$ and represent h_t as an indicator vector of dimension 2^b. This is to ensure the linearity of the optimal of o_t from h_t.

We compare these baselines to a model that exploits our insights on predictive belief methods ("LR-HMM"). This model represents the previous b observations with a vector of length b and uses logistic regression as S1 regression. This effectively reduces the number of parameters we need to learn from $O(2^b)$ to $O(b)$.

We evaluated the aforementioned 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. $|\delta_{o_t=1} - \hat{P}(o_t = 1 | o_{1:t-1})|$). We report the mean absolute error for each split. The results are displayed in figure 3.5. We see that, while incorporating more history information increases accuracy (model 2 vs. model 1), being able to incorporate the same information using a more compact model gives an additional gain in accuracy (model 3 vs model 2). We also compared our predictive belief method (model 3) to an HMM trained using expectation maximization (EM). We found that the predictive belief model is much faster to train than EM while being on par with it in terms of accuracy.



Model	Spec-HMM	Feat-HMM	LR-HMM	EM
Training time[4] (relative to Spec-HMM)	1	1.02	2.219	14.323

Figure 3.5: Experimental results: each graph depicts the performance of two models (measured by mean absolute error) on 1000 train/test splits. The black line represents the $x = y$ lines. More points below the line indicates that model y is better than model x. The table depicts training time of each model relative to model 1 (spectral HMM).

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. In this experiment, we show that we can use Lasso as S1 regression algorithm. This is useful when the system consists of multiple independent subsystems each of which affects a subset of observation coordinates.

To test this idea we generate a sequence of 30-dimensional observations from a Kalman filter. Observations 1 through 10 and 11 through 20 are generated from two independent subsystems of state dimension 5. Observations 21-30 are generated from white noise. Each subsystem was generated by a transition and observation matrices with random Gaussian coordinates and scaling the transition matrix to have a maximum eigenvalue of 0.95. States and observations are perturbed by Gaussian noise with diagonal covariance matrix of value equal to 0.01 and 1.0 respectively.

We estimate the state space basis using 1000 examples (assuming 1-observability) and compare

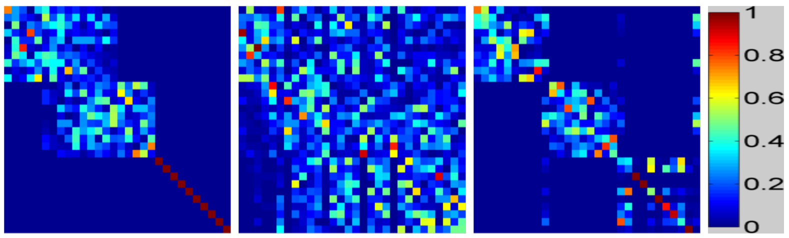


Figure 3.6: Left singular vectors of (left) true linear predictor from o_{t-1} to o_t (i.e. OTO^+), (middle) covariance matrix between o_t and o_{t-1} and (right) S1 Sparse regression weights. Each column corresponds to a singular vector (only absolute values are depicted). Singular vectors are ordered by their mean coordinate, interpreting absolute values as a probability distribution over coordinates.

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 predictive state learning algorithms for dynamical systems, which include spectral algorithms for Kalman filters [20], Hidden Markov Models [65, 101], Predictive State Representations (PSRs) [23, 25] and Weighted Automata[12]. One important advancement to these algorithms is introducing kernel variants such as [104] and [28]. These methods allow for modeling non-linear dynamics by transforming observations and predictive beliefs into higher dimensional spaces where linear ridge regression is performed. However, by allowing regression forms other than ridge regression, we can incorporate prior knowledge to learn compact or regularized models, which is not directly offered by kernel extensions, as we demonstrate in experiments.

One common aspect of predictive state learning algorithms is that they exploit the covariance structure between future and past observation sequences to obtain an unbiased observable state representation. Boots and Gordon [24] note the connection between the HSE-HMM and instrumental variables. We use this connection to build a general framework for dynamical system learning where the state-space can be identified using arbitrary supervised learning methods.

Reducing dynamical systems learning to supervised learning dates back to auto-regressive models [87], where the state of the system is assumed to be fully determined by the previous k observations. Our aim is to use supervised learning methods to learn latent state models from observation sequences. This bears similarity to Langford et al.'s sufficient posterior representation (SPR) [76], which encodes the state by the sufficient statistics of the conditional distribution of the next observation and represents system dynamics by three vector-valued functions that are estimated using supervised learning approaches. While SPR allows all of these functions to be non-linear, there are some advantages that distinguish our work. First, while SPR is limited to 1-step observable systems, our framework can seamlessly handle k-step observable systems by choosing a large enough (or even unbounded) window size. Secondly, SPR involves a rather complicated training procedure, involving multiple iterations of model refinement and model averaging, whereas our framework only requires solving three regression problems in sequence. Finally, the theoretical analysis of [76] only establishes the consistency of SPR learning assuming that all regression steps are solved perfectly. Our work, on the other hand, establishes convergence rates based on the performance of S1 regression.

This work can also be viewed as an extension to convex optimization based approach for learning weighted automata by Balle et. al.[12], which first estimates a set of Hankel matrices and then solves a least squares problem with trace norm regularization to estimate system matrices. We go one step further by allowing an arbitrary model of the dependency between past and future observations in place of Hankel matrices.

3.7 Conclusions

In this work we developed a general framework for dynamical system learning using supervised learning methods. The proposed framework is based on two-stage regression: in the first stage we use history features to train regression models that denoise future observation windows into state estimates. In the second stage we use these state estimates to train a linear model that represents system dynamics.

This framework encompasses and provides a unified view of some successful dynamical system learning algorithms. We demonstrated the proposed framework in learning a Hidden Markov Model, where we have shown that we can use non-linear regression to incorporate more history features in identifying the latent state without an exponential increase in the number of parameters.

As future work, we would like to apply this framework to more scenarios where we can leverage additional techniques such as manifold embedding and transfer learning in stage 1 regression. We would also like to extend the framework to controlled processes.

3.A Proofs

3.A.1 Proof of Main Theorem

In this section we provide a proof for theorem 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 7 (Matrix Bernstein's Inequality [67])**.** *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 $t > 0$,

$$\Pr\left[\lambda_{\max}\left[\frac{1}{N}\sum_{t=1}^{N} A_t\right] > \sqrt{\frac{2vt}{N}} + \frac{rt}{3N}\right] \leq \frac{kt}{v}(e^t - t - 1)^{-1}. \tag{3.18}$$

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:

$$W_\lambda = \mathcal{C}_{\bar{y}\bar{x}} \left(\mathcal{C}_{\bar{x}} + \lambda I\right)^{-1} \tag{3.19}$$
$$\bar{W}_\lambda = \hat{\mathcal{C}}_{\bar{y}\bar{x}} \left(\hat{\mathcal{C}}_{\bar{x}} + \lambda I\right)^{-1}, \tag{3.20}$$

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_λ 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 λ_{xi} and λ_{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 $\zeta^{\bar{x}\bar{x}}_{\delta,N}$ and $\zeta^{\bar{x}\bar{y}}_{\delta,N}$ which we use to bound the effect of covariance estimation from finite data, as stated in the following lemma:

Lemma 8 (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:*

$$\zeta_{\delta,N}^{\bar{x}\bar{y}} = \sqrt{\frac{2vt}{N}} + \frac{rt}{3N}, \tag{3.21}$$

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:

$$\zeta_{\delta,N}^{\bar{x}\bar{x}} = \sqrt{\frac{2v't'}{N}} + \frac{r't'}{3N}, \tag{3.22}$$

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 7 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 $\zeta^{\bar{y}\bar{y}}_{\delta,N}$ can be proven similarly. Now moving to $\zeta^{\bar{x}\bar{y}}_{\delta,N}$, 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 follows[110]:

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

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 $\zeta^{\bar{x}\bar{x}}_{\delta,N}$ case. $\square$

To prove theorem 5, we write

$$\begin{aligned} \|\hat{W}_\lambda x_{\text{test}} - W x_{\text{test}}\|_{\mathcal{Y}} \leq\ & \|(\hat{W}_\lambda - \bar{W}_\lambda)\bar{x}_{\text{test}}\|_{\mathcal{Y}} \\ & + \|(\bar{W}_\lambda - W_\lambda)\bar{x}_{\text{test}}\|_{\mathcal{Y}} \\ & + \|(W_\lambda - W)\bar{x}_{\text{test}}\|_{\mathcal{Y}} \end{aligned} \tag{3.23}$$

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 the boundedness of $\hat{W}_\lambda - W$.

Lemma 9 (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. The following holds with probability at least* $1-\delta$

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

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 $\phi_y \in \mathcal{Y}$

$$\begin{aligned} \langle \phi_y, \Delta_{yx}\phi_x\rangle_{\mathcal{Y}} &= \frac{1}{N}\sum_{t=1}^N \langle \phi_y, \hat{y}_t\rangle_{\mathcal{Y}}\langle \phi_x, \hat{x}_t\rangle_{\mathcal{X}} \\ &- \langle \phi_y, \hat{y}_t\rangle_{\mathcal{Y}}\langle \phi_x, \bar{x}_t\rangle_{\mathcal{X}} \\ &+ \langle \phi_y, \hat{y}_t\rangle_{\mathcal{Y}}\langle \phi_x, \bar{x}_t\rangle_{\mathcal{X}} - \langle \phi_y, \bar{y}_t\rangle_{\mathcal{Y}}\langle \phi_x, \bar{x}_t\rangle_{\mathcal{X}} \\ &= \frac{1}{N}\sum_t \langle \phi_y, \bar{y}_t + (\hat{y}_t - \bar{y}_t)\rangle_{\mathcal{Y}}\langle \phi_x, \hat{x}_t - \bar{x}_t\rangle_{\mathcal{X}} \\ &+ \langle \phi_y, \hat{y}_t - \bar{y}_t\rangle_{\mathcal{Y}}\langle \phi_x, \bar{x}_t\rangle_{\mathcal{X}} \\ &\leq 2c\eta_{\delta,N} + \eta^2_{\delta,N} \end{aligned}$$

Therefore,

$$\|\Delta_{yx}\| = \sup_{\|\phi_x\|_{\mathcal{X}}\leq 1, \|\phi_y\|_{\mathcal{Y}}\leq 1} \langle \phi_y, \Delta_{yx}\phi_x\rangle_{\mathcal{Y}} \leq 2c\eta_{\delta,N} + \eta_{\delta,N}^2,$$

and similarly

$$\|\Delta_x\| \leq 2c\eta_{\delta,N} + {\eta_{\delta,N}}^2,$$

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

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

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 10 (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 8.

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

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

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 11 (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 λ_{xi} and λ_{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 5 is simply the result of plugging the bounds in Lemma 9, 10, and 11 into (3.23) and using the union bound.

3.A.2 Proof of Lemma 6

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

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

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 $\|A\hat{P}_i\|_{\mathcal{Y}} \leq \|\hat{P}_i - \bar{P}_i\|_{\mathcal{Y}}$. □

3.B Examples of S1 Regression Bounds

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

Proposition 12. *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 $\eta_{\delta,N}$ be as defined in Definition 3. Then*

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

Proof. (sketch) This is based on results that bound parameter estimation error in linear regression with univariate response (e.g. [66]). 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. $\square$

Variants of Proposition 12 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 13. *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 [105] with regularization parameter $\lambda_0 > 0$ and that $\|\bar{x}\|_{\mathcal{X}}, \|\bar{y}\|_{\mathcal{Y}} < c < \infty$ almost surely. Let $\eta_{\delta,N}$ be as defined in Definition 3. It follows that*

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

Proof. (sketch) This bound is based on [105], 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. $\square$

Chapter 4

Method of Moments Learning for Controlled Systems

We now extend the framework introduced in Chapter 3 to controlled systems.

4.1 Introduction

Controlled dynamical systems, where an agent can influence an environment through actions and receive partial observations, emerge in numerous applications in robotics and automatic control. Modeling and learning these systems from data is of great importance in these fields.

Unfortunately, the formulation in Chapter 3 is limited to uncontrolled systems. We are now interested in controlled systems, where the user can affect the system through actions. This gives rise to a key issue: the policy that determines the actions can change at test time. For this reason the representation of the predictive state must be independent of the training policy and therefore must encode a *conditional distribution* of future observations given future actions. To adopt such a representation into a practical method that retains the benefits of the two-stage regression formulation, there are a number of challenges that need to be tackled.

First, A key assumption of two-stage regression for uncontrolled systems is that future observations provide an unbiased estimate of the predictive state, which is not true when the state is a conditional distributions. This means we need to define a new state representation which is valid for conditional distributions. Second, if we modify the state representation and introduce an action policy then the theoretical analysis from Chapter 3 is no longer valid. Third, because they are based on method of moments, two stage regression models are statistically inefficient. Having the ability to refine the model using local optimization can lead to significant gains in predictive performance.

4.2 Formulation

We define a class of models that extends predictive state models (PSMs) to controlled systems. We first introduce some notation: We denote by $\Pr[x \mid \mathbf{do}(Y = y)]$ the probability of x given that

Method	Actions	Continuous	Non-linear	Partially observable	Scalable	Consistent
Non-linear ARX	✓	✓	✓	×	✓	✓
N4SID for Kalman Filter	✓	✓	×	✓	✓	✓
Non-convex optimization (e.g. EM)	✓	✓	✓	✓	✓	×
Gram-Matrix (e.g. HSE-PSR)	✓	✓	✓	✓	×	✓
Spectral PSR/POMDP	✓	×	✓	✓	✓	✓
Reduction to Supervised Learning	×	✓	✓	✓	✓	✓
RFF-PSR	✓	✓	✓	✓	✓	✓

Table 4.1: Comparison between proposed RFF-PSR and existing system identification methods in terms of the type of systems they can model as well as their computational efficiency and statistical consistency. The table should be interpreted as follows: for each method there exists an instantiation that *simultaneously* satisfies all properties marked with ✓ but there is no instantiation that is guaranteed to satisfy the properties marked with $\times$. A method is scalable if computational and memory costs scale at most linearly with the number of training examples. For RFF-based methods, consistency is up to an approximation error that is controllable by the number of features [92].

we *intervene* by setting Y to y. This is different from $\Pr[x \mid Y = y]$ which denotes conditioning on *observing* $Y = y$; in the former case, we ignore all effects on Y by other variables. We denote by $V_{A|B;c}$ the linear operator that satisfies

$$\mathbb{E}[A|B = b, C = c] = V_{A|B;c} b \quad \forall b, c$$

In other words for each c, $V_{A|B;c}$ is a conditional expectation operator from B to A (In the discrete case, $V_{A|B;c}$ is just a conditional probability table).

When dealing with multiple variables, we will use tensor notation e.g. $V_{A,B|C,D}$ is a 4-mode tensor. We will use

$$V_{A,B|C,D} \times_C c \times_D d$$

to denote multiplying $V_{A,B|C,D}$ by c along the mode corresponding to C and by d along the mode corresponding to D. If c is a matrix then the multiplication is performed along the first dimension of c.

We will also use $\|\cdot\|_F$ to denote Frobenius norm, $a \otimes b$ to denote Kronecker product of two vectors and $A \star B$ to denote the Khatri-Rao product of two matrices (columnwise Kronecker product).

4.2.1 Model Definition

We will consider k-observable systems, where the posterior belief state given all previous observations and actions is uniquely identified by the conditional distribution $\Pr[o_{t:t+k-1} \mid \mathbf{do}(a_{t:t+k-1})]$.

we denote by ψ_t^o, ψ_t^a, ξ_t^o and ξ_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 $h_t^\infty \equiv o_{1:t-1}, a_{1:t-1}$ to denote the entire history of observations and actions at time t and 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[1].

We are now ready to define the class of systems we are interested in.

Definition 14. *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 $Q_t = V_{\psi_t^o|\mathbf{do}(\psi_t^a);h_t^\infty}$ (referred to as 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 $P_t = V_{\xi_t^o|\mathbf{do}(\xi_t^a);h_t^\infty}$ (referred to as 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_{sys} (referred to as system parameter map), such that, for each time t,*

$$P_t = W_{\text{sys}}(Q_t) \tag{4.1}$$

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

It follows that a PSCM is specified by the tuple $(Q_0, W_{\text{sys}}, f_{\text{filter}})$, where Q_0 denotes the initial belief state.

There are a number of aspects of PSCMs that warrant discussion. 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^∞. It represents the *belief* state that one should maintain after observing the history to make optimal predictions. 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

$$Q_{t+1} = f_{\text{filter}}(W_{\text{sys}}(Q_t), o_t, a_t) \tag{4.2}$$

This construction allows us to learn a linear map W_{sys} and use it to build models with non-linear state updates, including IO-HMMs citepiohmm, Kalman filters with inputs [112] and HSE-PSRs [29] [2]. As we see in Section 4.3, avoiding latent variables and having a linear W_{sys} enable the formulation of a consistent learning algorithm.

4.3 Learning A Predictive State Controlled Model

We assume that the extended features ξ_t^o and ξ_t^a are chosen such that f_{filter} is known. The parameters to learn are thus W_{sys} and Q_0. We also assume that a fixed blind (open-loop) policy is

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

[2]We discuss HSE-PSRs in more detail in Section 4.4. We defer a more detailed discussion of Kalman filters and IO-HMMs to the supplementary material.

used to collect training data and so, we can treat causal conditioning on action $\mathbf{do}(a_t)$ as ordinary conditioning on a_t. [3] 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 method of 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 state (resp. expected extended state) conditioned on finite history features h_t. 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 $\mathbb{E}[\psi_t^o \mid \psi_t^a, h_t] = \bar{Q}_t \psi_t^a$ and $\mathbb{E}[\xi_t^o \mid \xi_t^a, h_t] = \bar{P}_t \xi_t^a$; and it follows from the linearity of W_{sys} that

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

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_{sys} 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 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 below, instances of both methods exist in the literature of system identification.

4.3.1 Joint S1 Approach

Let ψ_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

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

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 [50] to estimate states in HSE-PSR [29].

4.3.2 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 ψ_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

[3] One way to deal with non-blind training policies is to assign importance weight to training examples to correct the bias resulting from non-blindness [25, 30]. 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 unknown non-blind policy to future work.

learning a function $f : h_t \mapsto \bar{Q}_t$ that best matches the training examples. i.e. we solve the problem

$$\arg\min_{f \in \mathcal{F}} \sum_{t=1}^{T} l(f(h_t)\psi_t^a, \psi_t^o) + R(f) \tag{4.3}$$

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 spectral system identification [112]. 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.

4.3.3 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 1.

Algorithm 1 Two-stage regression for predictive state controlled models

```
**Input:** $\psi^o_{n,t}$, $\psi^a_{n,t}$, $\xi^o_{n,t}$, $\xi^a_{n,t}$ 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 $\hat{W}_{\text{sys}}$ and initial state $\hat{Q}_0$
Use S1A regression to estimate $\bar{Q}_{n,t}$.
Use S1B regression to estimate $\bar{P}_{n,t}$.
Let $\hat{W}_{\text{sys}}$ be the (regularized) least squares solution to the system of equations
    $\bar{P}_{n,t} \approx W_{\text{sys}}(\bar{Q}_{n,t}) \quad \forall n, t$
**if** $N$ is sufficiently large **then**
    Let $\bar{Q}_0$ be the (regularized) least square solution to the system of equations $\psi^o_{n,1} \approx Q_0\psi^a_{n,1} \quad \forall n$
**else**
    Set $\hat{Q}_0$ to the average of $\bar{Q}_{n,t}$
**end if**
```

4.3.4 Theoretical Guarantees

It is worth noting that Algorithm 1 is still an instance of the two stage regression framework described in Chapter 3 and hence retains its theoretical guarantees: mainly that we can bound the error in estimating the dynamics matrix W_{sys} in terms of S1 regression error bounds, assuming that we collect examples from the stationary distribution of some blind policy. [4]

A blind policy provides sufficient exploration if it has a stationary distribution that (1) visits a sufficient history set such that the set of equations $\mathbb{E}[P_t|h_t] = W_{sys}(\mathbb{E}[Q_t|h_t])$ are sufficient for estimating W_{sys} ,and (2) provides training data to estimate $\mathbb{E}[Q_t|h_t]$ with increasing accuracy.

[4]The theoretical guarantee of Algorithm 1 applies to any blind policy. However, to obtain low S1 regression errors, a good exploration policy is needed

Theorem 15. *Let π be a blind data collection policy with a stationary distribution. If history, action and observation features are bounded, π provides sufficient exploration, and ridge regression is used with λ_1 and λ_2 regularization parameter for S1 and S2 regression respectively, then for all valid states Q the following is satisfied with probability at least $1-\delta$.*

$$\begin{aligned}\|(\hat{W}_{\text{sys}} - W_{\text{sys}})(Q)\| \leq O &\left(\eta_{\delta,N} \left(\frac{1}{\lambda_2} + \frac{\sqrt{1+\sqrt{\frac{\log(1/\delta)}{N}}}}{\lambda_2^{\frac{3}{2}}} \right) \right) \\ &+ O\left(\frac{\log(1/\delta)}{\sqrt{N}} \left(\frac{1}{\lambda_2} + \frac{1}{\lambda_2^{\frac{3}{2}}} \right) \right) \\ &+ O\left(\sqrt{\lambda_2}\right),\end{aligned}$$

where

$$\eta_{\delta,N} = O_p\left(\frac{1/\sqrt{N} + \lambda_1}{\lambda_1} \right)$$

We provide proofs and discussion of sufficient exploration condition in the supplementary material.

4.4 Connections with HSE-PSRs

Having a general framework for learning controlled dynamical systems, we now focus on HSE-PSR [29] as a non-parametric instance of that framework using Hilbert space embedding of distributions [105]. We describe HSE-PSR learning as a two-stage regression method.

4.4.1 HSE-PSR as a predictive state controlled model

HSE-PSR is a generalization of IO-HMM that has proven to be successful in practice [21, 29]. It is suitable for high dimensional and continuous observations and/or actions. HSE-PSR uses 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.

We can then define $\psi_t^o = \phi_O(o_{t:t+k-1})$ and similarly $\psi_t^a = \phi_A(a_{t:t+k-1})$. We will also use ϕ_t^o and ϕ_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

$$Q_t = V_{\psi_t^o|\psi_t^a;h_t^\infty} \tag{4.4}$$

$$P_t = (P_t^\xi, P_t^o) = (V_{\psi_{t+1}^o \otimes \phi_t^o|\psi_{t+1}^a \otimes \phi_t^a;h_t^\infty}, V_{\phi_t^o \otimes \phi_t^o|\phi_t^a;h_t^\infty}) \tag{4.5}$$

Therefore, 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}. In more detail, filtering in an HSE-PSR is carried out as follows

- From o_t and a_t, obtain ϕ_t^o and ϕ_t^a.
- Compute $C_{o_t o_t|h_t^\infty,a_t} = V_{\phi_t^o \otimes \phi_t^o|\phi_t^a;h_t^\infty}\phi_t^a$
- Multiply by inverse observation covariance to change "predicting ϕ_t^o" into "conditioning on ϕ_t^o":

$$V_{\psi_{t+1}^o|\psi_{t+1}^a,\phi_t^o,\phi_t^a;h_t^\infty} = V_{\psi_{t+1}^o \otimes \phi_t^o|\psi_{t+1}^a,\phi_t^a;h_t^\infty} \times_{\phi_t^o} (C_{o_t o_t|h_t^\infty,a_t} + \lambda I)^{-1} \tag{4.6}$$

- Condition on ϕ_t^o and ϕ_t^a to obtain shifted state

$$\begin{aligned} Q_{t+1} &\equiv V_{\psi_{t+1}^o|\psi_{t+1}^a;\phi_t^o,\phi_t^a,h_t^\infty} \\ &= V_{\psi_{t+1}^o|\psi_{t+1}^a,\phi_t^o,\phi_t^a;h_t^\infty} \times_{\phi_t^o} \phi_t^o \times_{\phi_t^a} \phi_t^a \end{aligned}$$

Thus, in HSE-PSR, the parameter W_{sys} is composed of two linear maps; f_o and f_ξ 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_ξ, f_o can then be carried out using kernel regression.

Learning and filtering in an HSE-PSR can be implicitly carried out in RKHS using a Gram matrix formulation. We will describe learning in terms of RKHS elements and refer the reader to [29] for details on the Gram matrix formulation.

4.4.2 S1 Regression for HSE-PSR

As discussed in section 4.3 we can use a joint or conditional approach for S1 regression. We now demonstrate how these two approaches apply to HSE-PSR.

Joint S1 Regression for HSE-PSR

This is the method used in [29]. 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 $T_{oa}(h_t) \approx C_{\psi_t^o\psi_t^a|h_t}$ and $T_a(h_t) \approx C_{\psi_t^a\psi_t^a|h_t}$. 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 $\bar{Q}_t$.

Conditional S1 Regression for HSE-PSR

It is also possible to apply the conditional S1 regression formulation in Section 4.3.2. Specifically, let $\mathcal{F}$ be the set of 3-mode tensors, with modes corresponding to ψ_t^o, ψ_t^o and h_t. We estimate a tensor T^* by optimizing

$$T^* = \arg\min_{T\in\mathcal{F}} \|(T \times_{h_t} h_t \times_{\psi_a^t} \psi_a^t) - \psi_o^t\|^2 + \lambda\|T\|_{HS}^2,$$

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 ψ_t^o and ψ_t^a with their extended counterparts ξ_t^o and ξ_t^a.

4.5 Experiments

4.5.1 Synthetic Data

We also test the proposed model on the benchmark synthetic non-linear system used by Boots et al. [29]:

$$\begin{aligned}
\dot{x}_1(t) &= x_2(t) - 0.1\cos(x_1(t))(5x_1(t) - 4x_1^3(t) + x_1^5(t)) \\
&\quad - 0.5\cos(x_1(t))a(t), \\
\dot{x}_2(t) &= -65x_1(t) + 50x_1^3(t) - 15x_1^5(t) - x_2(t) - 100a(t), \\
o(t) &= x_1(t)
\end{aligned}$$

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 $o(t)$.

4.5.2 Predicting windshield view

In this experiment we used TORCS car simulation server, which outputs 64x64 images (see Figure 4.1). 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 gets of of track. We used 40 trajectories for training, 5 for validation and 5 for testing. The prediction target is the projected image.

4.5.3 Predicting the nose position of a simulated swimmer robot

We consider the 3-link simulated swimmer robot from the open-source package RLPy [52]. 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_{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.

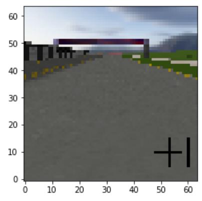


Figure 4.1: An example of windshield view output by TORCS.

4.5.4 Tested Methods and Evaluation Procedure

We tested three different initializations of RFF-PSR (with RBF kernel): random initialization, two-stage regression with joint S1, and two-stage regression with conditional S1 (Section 4.4.2). For each initialization, we tested the model before and after BPTT. For BPTT we used 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 formulation of HSE-PSR as described in [29], learned using a combination of two-stage regression and BPTT.

N4SID: We used MATLAB's implementation of subspace identification of linear dynamical systems.

Non-linear Auto Regression with Exogenous Inputs (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 a future length of 10 a history of 20. For RFF-PSR and RFF-ARX we used 10000 RFF 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 $H \in \{1, 2, \ldots, 10\}$.

4.5.5 Results and Discussion

The results are shown in Figure 4.2 [5]. There are a number of important observations.

- In general, joint S1 training closely matches or outperforms conditional S1 training, both with and without BPTT.
- BPTT significantly improves predictive performance for all initialization methods.

[5] We omit the results for randomly initialized RFF-PSR as they were significantly worse. A comparison with HSE-PSR on the swimmer dataset was not possible as it required prohibitively large memory.

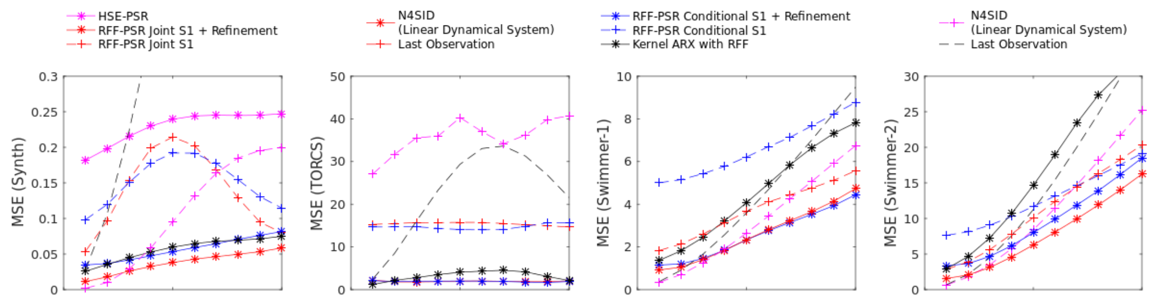


Figure 4.2: Mean square error for 10-step prediction on (from left to right) synthetic model, TORCS car simulator, swimming robot simulation with 80% blind test-policy, and swimming robot with 20% blind test policy. Baselines with very high MSE are not shown for clarity. A comparison with HSE-PSR on TORCS and swimmer datasets was not possible as it required prohibitively large memory.

- BPTT, on its own, is not sufficient to produce a good model. The two stage regression provides a good initialization of the BPTT procedure.
- Even without BPTT, RFF-PSR matches or outperforms HSE-PSR. This is remarkable since RFF-PSR requires less computation and memory.
- In the synthetic experiment, non-refined RFF-PSR is giving poor mid-range predictions. We conjecture that short-range predictions are easier as they largely depend on the previous observations, and long-range predictions are also easier because they largely depend on the sequence of actions. However, BPTT was able to produce a model that performs well at all prediction horizons.
- Compared to other methods, RFF-PSR has better performance with non-blind test policies.

4.6 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.

4.6.1 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 $O_k \times_s s_t \times_a a_{t:t+k-1} = \mathbb{E}[o_{t:t+k-1}|a_{t:t+k-1}, s_t]$

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

$$O^k \times_s e_i \times_a (e_j \otimes e_l) = (O_{ij} \otimes (O^{k-1} \times_a e_l \times_s T_{ij})) \tag{4.7}$$

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

$$\begin{aligned} Q_t &= O^k \times_s s_t \\ P_t &= O^{k+1} \times_s s_t \end{aligned}$$

Now to show that (4.1) holds, let $\tilde{O}^k$ be a reshaping of O^k into a matrix such that

$$(Q_t) = \tilde{O}^k s_t$$

It follows that

$$P_t = O^{k+1} \times_s s_t = O^{k+1} \times_s ((\tilde{O}^k)^{+}(Q_t)),$$

which is linear in Q_t.

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 ψ_t^o is an unbiased estimate of a single column of Q_t corresponding to ψ_t^a. We can use (4.3) to learn a function $f : h_t \mapsto \bar{Q}_t$ that best matches the training examples.

4.6.2 Kalman Filter with inputs

The Kalman filter is given by

$$\begin{aligned} x_t &= Ax_{t-1} + Bu_t + \epsilon_t \\ o_t &= Cx_t + \nu_t \end{aligned}$$

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 Γ_k and U_k replaced with Γ_{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^∞ (e.g. stacking of a finite window of observations and actions), it can also be shown [112] that

$$\mathbb{E}[Q_t | h_t] = \tilde{\Gamma}_k h_t,$$

for some matrix $\tilde{\Gamma}_k$.

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 [112] uses linear regression to estimate Γ and B (essentially solving (4.3)). Having a fixed B, the conditional operator is determined by Γh_t through an affine transformation. Therefore we can use $\bar{Q}_t = \Gamma h_t$.

4.7 Theoretical Analysis

Let $\mathcal{H} = \{h_i\}_{i=1}^N$ be set of histories generated from an i.i.d distribution. [6] We use $Q(h)$ to denote $\mathbb{E}[Q|h]$.

The main theorem in [59] bounds parameter estimation error in terms of S1 regression error. This implies that we need to analyze the properties of S1 regression to prove Theorem 15. We will look at multiple scenarios where in each scenario we develop sufficient exploration conditions and provide an S1 error bound for these conditions.

[6]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.

Definition 16 (Sufficient history set)**.** *Consider a PSCM that satisfies*

$$P_t = W_{\text{sys}}(Q_t)$$

*A set of histories $\mathcal{H} = \{h_i\}_{i=1}^{M}$ is called a **sufficient history set** if it is sufficient to estimate W_{sys} using $\mathbb{E}[Q|h(h)]$ and $\mathbb{E}[P|h(h)]$ for each $h \in \mathcal{H}$.*

Note that W_{sys} may not be unique, we care about estimating $W_{\text{sys}}Q$ for any valid Q. From the above definition, it follows that a data collection policy provides sufficient exploration if it allows for estimating $\mathbb{E}[Q|h(h)]$ and $\mathbb{E}[P|h(h)]$ for a sufficient history set with increasing accuracy.

4.7.1 Case 1: Discrete Observations and Actions

Consider a discrete system where $\mathcal{H}$, $\mathcal{A}$, $\mathcal{A}^+$, $\mathcal{O}$, $\mathcal{O}^+$ are the set of all possible histories, future action sequences, extended future action sequences, future observation sequences and extended future observation sequences respectively.

Theorem 17. *Assume a discrete system where the data collection policy induces an i.i.d 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 $N = M|\mathcal{H}||\mathcal{A}^+|$ with S1 error bound $\eta_{N,\delta} = \sqrt{\frac{|\mathcal{H}||\mathcal{A}^+||\mathcal{O}^+|}{2M} \log\left(\frac{2|\mathcal{H}||\mathcal{A}^+||\mathcal{O}^+|}{\delta}\right)}$*

Proof. The proof follows immediately from Heoffding's inequality which bounds the error in estimating the probability of an event by averaging.

Note that we need to estimate $|\mathcal{H}||\mathcal{A}||\mathcal{O}|$ probabilities to estimate Q and $|\mathcal{H}||\mathcal{A}^+||\mathcal{O}^+|$ probabilities to estimate P. Therefore we divide δ by $2|\mathcal{H}||\mathcal{A}^+||\mathcal{O}^+|$ to correct for multiple probability estimates. □

Remark 18. *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 $P(o_{t-1:t+1}, a_{t-1:t+1})$.*

4.7.2 Case 2: Continuous System

Definition 19 (Range and span of a policy)**.** *Let π be a data collection policy with a stationary distribution. For a random vector $X_t = f(h, O, A)$, the **range of** π **on** X is the support of the stationary distribution of X_t induced by the policy π (i.e. the set of all possible values of X_t that can be generated by the stationary distribution).*

*The **span of** π **on** X is the subspace spanned by the range of π on X.*

When referring to the policy range or span, we may omit the variable name when it is clear in the context.

Condition 20 (Action span for joint S1)**.** *Let π be data collection policy and let $\mathcal{H}$ be the range of π on histories. The action span condition for joint S1 requires that the following are satisfied:*

1. *$\mathcal{H}$ is a sufficient history set.*
2. *For any $h \in \mathcal{H}$, the conditional covariance $\Sigma_{A|h}$ is full rank.*

Condition 21 (Action span for conditional S1)**.** *Let π be data collection policy and let $\mathcal{H}$ be the range of π on histories. The action span condition for conditional S1 requires that the following are satisfied:*

1. *$\mathcal{H}$ is a sufficient history set.*
2. *For any $h \in \mathcal{H}$ and any future action feature vector ψ^a, the quantity $(\psi^h \otimes \psi^a)$ is in policy span.*

Remark 22. *Condition 20 implies Condition 21.*

Assumption 23 (Bounded features)**.** *Assume that $\|\psi^h\| < c_h$ for all $h \in \mathcal{H}$. Also, assume that $\|\psi^o\| \leq c_O$ and $\|\psi^a\| \leq c_A$ for any valid future observation sequence and action sequence respectively.*

Theorem 24. *Let π be data collection policy and let $\mathcal{H}$ be the span of π on histories. If Assumption 23 and Condition 21 are satisfied and conditional S1 regression is used with a liner model as the correct model, then π provides sufficient exploration and for all $h \in \mathcal{H}$ the following holds with probability at least $1-\delta$*

$$\|\hat{Q}(h) - Q(h)\| \leq c_h \left(\sqrt{\frac{\lambda_{\max}(\Sigma_O)}{\lambda_{\min}(\Sigma_{h\otimes A})}} \left(\frac{\sqrt{\lambda_{\min}(\Sigma_{h\otimes A})}\Delta_1 + \lambda}{\lambda_{\min}(\hat{\Sigma}_{h\otimes A}) + \lambda} \right) + \frac{\Delta_2}{\lambda_{\min}(\hat{\Sigma}_{h\otimes A}) + \lambda} \right),$$

where

$$\Delta_1 = 2c_h c_A \sqrt{\frac{\log(2d_h d_A/\delta)}{N}} + \frac{2\log(2d_h d_A/\delta)}{3N} \left(\frac{c_h^2 c_A^2}{\sqrt{\lambda_{\min}(\Sigma_{h\otimes A})}} + c_h c_A \right)$$
$$\Delta_2 = 2c_O c_h c_A \sqrt{\frac{\log((d_O + d_h d_A)/\delta)}{N}} + \frac{4c_O c_h c_A \log((d_O + d_h d_A)/\delta)}{3N}$$

In the following section we provide a proof sketch for the asymptotic form in Theorem 15 for joint S1.

Remark 25 (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.

4.8 Conclusions

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 BPTT to enhance the model after a two-stage regression initialization. We have demonstrated promising results for the proposed method in terms of predictive performance. As future work, we would like to go beyond predicting observations, by using this framework for planning, imitation learning and reinforcement learning.

4.A RFF-PSR Learning Algorithm

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. Here we use $A \star B$ to denote the Khatri-Rao product of two matrices (columnwise Kronecker product).

Algorithm 2 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:
    SVD$(X, p)$, returns the tuple $(U, U^\top X)$, where $U$ consists of top $p$ singular vectors of $X$.

    {Feature projection using PCA}
    $U^h, \Phi^h \leftarrow$ SVD$(\Phi^h, p)$;
    $U^o, \Phi^o \leftarrow$ SVD$(\Phi^o, p)$; $U^a, \Phi^a \leftarrow$ SVD$(\Phi^a, p)$;
    $U^o_\psi, \Psi^o \leftarrow$ SVD$(\Psi^o, p)$; $U^a_\psi, \Psi^a \leftarrow$ SVD$(\Psi^a, p)$;
    $U^o_\xi, \Xi^o \leftarrow$ SVD$((U^{o\top}_\psi \Psi^{o\prime}) \star \Phi^o, p)$;
    $U^a_\xi, \Xi^a \leftarrow$ SVD$(\Phi^a \star (U^{a\top}_\psi \Psi^{a\prime}), p)$;
    $U^{oo}, \Phi^{oo} \leftarrow$ 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 the one of the S1 methods in 4.4.2.
    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 $\mathbf{P}^o$.
    $U^q, \mathbf{Q} \leftarrow$ SVD$(\mathbf{Q}, p)$
    {S2 Regression}
    $\hat{W}_\xi \leftarrow \arg\min_{W \in \mathbb{R}^{p^2 \times p}} \|\mathbf{P}^\xi - W\mathbf{Q}\|^2 + \lambda_2 \|W\|_F^2$
    $\hat{W}_o \leftarrow \arg\min_{W \in \mathbb{R}^{p^2 \times p}} \|\mathbf{P}^o - W\mathbf{Q}\|^2 + \lambda_2 \|W\|_F^2$
```

4.B Proofs of theorems

Lemma 26 (Matrix Bernstein [111]). *Consider a finite sequence $\{S_k\}$ of independent, random matrices with common dimensions $d_1 \times d_2$. Assume that*

$$\mathbb{E}[S_k] = 0 \textit{ and } \|S_k\| \leq L \quad \textit{for each index } k$$

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 27 (Error in empirical cross-covariance)**.** *With probability at least $1 - \delta$*

$$\|\hat{\Sigma}_{YX} - \Sigma_{YX}\| \leq \sqrt{\frac{2\log((d_X + d_Y)/\delta)v}{N}} + \frac{2\log((d_X + d_Y)/\delta)L}{3N},$$

where

$$\begin{aligned} L &= c_y c_x + \|\Sigma_{YX}\| \leq 2c_y c_x \\ v &= \max(c_y^2 \|\Sigma_X\|, c_x^2 \|\Sigma_Y\|) + \|\Sigma_{YX}\|^2 \leq 2c_y^2 c_x^2 \end{aligned}$$

Proof. Define $S_k = y_k x_k^\top - \Sigma_{YX}$, it follows that

$$\begin{aligned} \mathbb{E}[S_k] &= 0 \\ \|S_k\| &= \|y_k x_k^\top - \Sigma_{YX}\| \leq \|y_k\|\|x_k\| + \|\Sigma_{YX}\| \leq c_y c_x + \|\Sigma_{YX}\| \\ \|\mathbb{E}[ZZ^\top]\| &= \left\| \sum_{i,j} (\mathbb{E}[y_i x_i^\top x_j y_j^\top] - \Sigma_{YX}\Sigma_{XY}) \right\| \\ &= \left\| \sum_i (\mathbb{E}[\|x_i\|^2 y_i y_i^\top] - \Sigma_{YX}\Sigma_{XY}) + \sum_{i,j \neq i} (\mathbb{E}[y_i x_i^\top]\mathbb{E}[x_j y_j^\top] - \Sigma_{YX}\Sigma_{XY}) \right\| \\ &\leq N(c_x^2 \|\Sigma_Y\| + \|\Sigma_{YX}\|^2) \\ \|\mathbb{E}[Z^\top Z]\| &\leq N(c_y^2 \|\Sigma_X\| + \|\Sigma_{YX}\|^2) \end{aligned}$$

Applying Lemma 26 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 28 (Normalized error in empirical covariance)**.** *With probability at least* $1-\delta$

$$\|\Sigma_X^{-1/2}(\hat{\Sigma}_X - \Sigma_X)\| \leq 2c\sqrt{\frac{2\log(2d/\delta)}{N}} + \frac{2\log(2d/\delta)L}{3N},$$

where

$$L = \frac{c^2}{\sqrt{\lambda_{\min}(\Sigma_X)}} + c$$

Proof. Define $S_k = \Sigma_X^{-1/2} x_k x_k^\top - \Sigma_X^{1/2}$, it follows that

$$\begin{aligned}
\mathbb{E}[S_k] &= 0 \\
\|S_k\| &\leq \|\Sigma_X^{-1/2}\|\|x_k\|^2 + \|\Sigma_X^{1/2}\| \leq \frac{c^2}{\sqrt{\lambda_{\min}(\Sigma_X)}} + c \\
\|\mathbb{E}[Z^\top Z]\| = \|\mathbb{E}[ZZ^\top]\| &= \left\|\sum_{i,j}(\Sigma_X^{-1/2}\mathbb{E}[x_i x_i^\top x_j x_j^\top]\Sigma_X^{-1/2} - \Sigma_X)\right\| \\
&= \left\|\sum_i(\mathbb{E}[\|x_i\|^2 \Sigma_X^{-1/2} x_i x_i^\top \Sigma_X^{-1/2}] - \Sigma_X) + \sum_{i,j\neq i}(\Sigma_X^{-1/2}\mathbb{E}[x_i x_i^\top]\mathbb{E}[x_j x_j^\top]\Sigma_X^{-1/2} - \Sigma_X)\right\| \\
&\leq N(c_x^2 + \|\Sigma_X\|^2) \leq 2Nc^2
\end{aligned}$$

Applying Lemma 26 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 27, we can show that

$$t \leq \frac{2\log(2d/\delta)L}{3N} + 2c\sqrt{\frac{\log(2d/\delta)}{N}}$$

□

Lemma 29. *Let* $\hat{\Sigma}_{YX} = \Sigma_{YX} + \Delta_{YX}$ *and* $\hat{\Sigma}_X = \Sigma_X + \Delta_X$ *where* $\mathbb{E}[\Delta_{YX}]$ *and* $\mathbb{E}[\Delta_{YX}]$ *are not necessarily zero and* $\hat{\Sigma}_X$ *is symmetric positive semidefinite. Define* $W = \Sigma_{YX}\Sigma_X^{-1}$ *and* $\hat{W} = \hat{\Sigma}_{YX}(\hat{\Sigma}_X + \lambda)^{-1}$. *It follows that*

$$\|\hat{W} - W\| \leq \sqrt{\frac{\lambda_{\max}(\Sigma_Y)}{\lambda_{\min}(\Sigma_X)}} \left(\frac{\sqrt{\lambda_{\min}(\Sigma_X)}\|\Sigma_X^{-1/2}\Delta_X\| + \lambda}{\lambda_{\min}(\hat{\Sigma}_X) + \lambda} \right) + \frac{\|\Delta_{YX}\|}{\lambda_{\min}(\hat{\Sigma}_X) + \lambda}$$

Proof.

$$\hat{W} - W = \Sigma_{YX}\left((\Sigma_X + \Delta_X + \lambda I)^{-1} - \Sigma_X^{-1}\right) + \Delta_{YX}(\Sigma_X + \Delta_X + \lambda I)^{-1} = T_1 + T_2$$

It follows that

$$\|T_2\| \leq \frac{\|\Delta_{YX}\|}{\lambda_{\min}(\hat{\Sigma}_X) + \lambda}$$

As for T_1, using the matrix inverse Lemma $B^{-1} - A^{-1} = B^{-1}(A - B)A^{-1}$ and the fact that $\Sigma_{YX} = \Sigma_Y^{1/2} V \Sigma_X^{1/2}$, where V is a correlation matrix satisfying $\|V\| \leq 1$ we get

$$\begin{aligned} T_1 &= -\Sigma_{YX}\Sigma_X^{-1}(\Delta_X + \lambda I)(\Sigma_X + \Delta_X + \lambda I)^{-1} \\ &= -\Sigma_Y^{1/2} V \Sigma_X^{-1/2}(\Delta_X + \lambda I)(\Sigma_X + \Delta_X + \lambda I)^{-1}, \end{aligned}$$

and hence

$$\begin{aligned} \|T_1\| &\leq \sqrt{\lambda_{\max}(\Sigma_Y)} \left(\frac{\|\Sigma_X^{-1/2}\Delta_X\| + \lambda\|\Sigma_X^{-1/2}\|}{\lambda_{\min}(\hat{\Sigma}_X) + \lambda} \right) \\ &= \sqrt{\frac{\lambda_{\max}(\Sigma_Y)}{\lambda_{\min}(\Sigma_X)}} \left(\frac{\sqrt{\lambda_{\min}(\Sigma_X)}\|\Sigma_X^{-1/2}\Delta_X\| + \lambda}{\lambda_{\min}(\hat{\Sigma}_X) + \lambda} \right) \end{aligned}$$

□

Corollary 30. *With probability* $1 - 2\delta$.

$$\|\hat{W} - W\| \leq \sqrt{\frac{\lambda_{\max}(\Sigma_Y)}{\lambda_{\min}(\Sigma_X)}} \left(\frac{\sqrt{\lambda_{\min}(\Sigma_X)}\Delta_1 + \lambda}{\lambda_{\min}(\hat{\Sigma}_X) + \lambda} \right) + \frac{\Delta_2}{\lambda_{\min}(\hat{\Sigma}_X) + \lambda},$$

where

$$\begin{aligned} \Delta_1 &= 2c_x\sqrt{\frac{\log(2d_X/\delta)}{N}} + \frac{2\log(2d_X/\delta)}{3N}\left(\frac{c_x^2}{\sqrt{\lambda_{\min}(\Sigma_X)}} + c_x \right) \\ \Delta_2 &= 2c_y c_x\sqrt{\frac{\log((d_Y + d_X)/\delta)}{N}} + \frac{4c_y c_x \log((d_Y + d_X)/\delta)}{3N} \end{aligned}$$

Proof. This corollary follows simply from applying Lemmata 27 and 28 to Corollary 29. □

Lemma 31. *Let $\hat{\Sigma}_{YX} = \Sigma_{YX} + \Delta_{YX}$ and $\hat{\Sigma}_X = \Sigma_X + \Delta_X$ where $\mathbb{E}[\Delta_{YX}]$ and $\mathbb{E}[\Delta_{YX}]$ is not necessarily zero and $\hat{\Sigma}_X$ is symmetric but not necessarily positive semidefinite. Define $W = \Sigma_{YX}\Sigma_X^{-1}$ and $\hat{W} = \hat{\Sigma}_{YX}\hat{\Sigma}_X(\hat{\Sigma}_X^2 + \lambda)^{-1}$. It follows that*

$$\|\hat{W} - W\| \leq \sqrt{\frac{\lambda_{\max}(\Sigma_Y)}{\lambda_{\min}^3(\Sigma_X)}} \frac{\|\Delta_X\|^2 + 2\lambda_{\max}(\Sigma_X)\|\Delta_X\| + \lambda}{\lambda_{\min}^2(\hat{\Sigma}_X) + \lambda} + \frac{\|\Sigma_{YX}\|\|\Delta_X\| + \|\Delta_{YX}\|\|\Sigma_X\| + \|\Delta_{YX}\|\|\Delta_X\|}{\lambda_{\min}^2(\hat{\Sigma}_X) + \lambda}$$

Proof.

$$\begin{aligned}
\hat{W} - W &= (\Sigma_{YX} + \Delta_{YX})(\Sigma_X + \Delta_X)((\Sigma_X + \Delta_X)^2 + \lambda I)^{-1} - \Sigma_{YX}\Sigma_X\Sigma_X^{-2} \\
&= \Sigma_{YX}\Sigma_X(((\Sigma_X + \Delta_X)^2 + \lambda I)^{-1} - \Sigma_X^{-2}) + (\Sigma_{YX}\Delta_X + \Delta_{YX}\Sigma_X + \Delta_{YX}\Delta_X)((\Sigma_X + \Delta_X)^2 + \lambda I)^{-1} \\
&= T_1 + T_2
\end{aligned}$$

Using the matrix inverse Lemma $B^{-1} - A^{-1} = B^{-1}(A - B)A^{-1}$ and the fact that $\Sigma_{YX} = \Sigma_Y^{1/2} V \Sigma_X^{1/2}$, where V is a correlation matrix satisfying $\|V\| \leq 1$ we get

$$\begin{aligned}
T_1 &= -\Sigma_{YX}^{1/2} V \Sigma_X^{-3/2}(\Delta_X^2 + \Sigma_X\Delta_X + \Delta_X\Sigma_X + \lambda I)((\Sigma_X + \Delta_X)^2 + \lambda I)^{-1} \\
\|T_1\| &\leq \sqrt{\frac{\lambda_{\max}(\Sigma_Y)}{\lambda_{\min}^3(\Sigma_X)}} \frac{\|\Delta_X\|^2 + 2\lambda_{\max}(\Sigma_X)\|\Delta_X\| + \lambda}{\lambda_{\min}^2(\hat{\Sigma}_X) + \lambda} \\
\|T_2\| &\leq \frac{\|\Sigma_{YX}\|\|\Delta_X\| + \|\Delta_{YX}\|\|\Sigma_X\| + \|\Delta_{YX}\|\|\Delta_X\|}{\lambda_{\min}^2(\hat{\Sigma}_X) + \lambda}
\end{aligned}$$

□

4.B.1 Proof of Theorem 24

Proof. In the linear case, we estimate a tensor T with modes corresponding to h, A and O by solving the minimization problem in Section 4.4.2. 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 ψ^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 21 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 30 to bound $\|\hat{T}_r - T_r\|$. □

4.B.2 Sketch Proof for joint S1

In order to prove Theorem 15 for joint S1, note that

$$\|\hat{\Sigma}_{A|h} - \Sigma_{A|h}\| \leq \|\hat{T}_A - T_A\|\|h\|$$
$$\|\hat{\Sigma}_{OA|h} - \Sigma_{OA|h}\| \leq \|\hat{T}_{OA} - T_{OA}\|\|h\|$$

From Lemma 29, we obtain a high probability bound on $\|\hat{T}_A - T_A\|$ and $\|\hat{T}_A - T_A\|$. Then we apply these bounds to Lemma 31 to obtain an error in $Q(h)$.

Part III

Hybrid Models

Chapter 5

Predictive State Recurrent Neural Networks

Having formalized a class of models which admit consistent method of moments learning in chapters 3 and 4 we are now ready to begin our exploration of hybrid models.

As we saw in chapter 2, RNNs and BFs offer complementary advantages and disadvantages: RNNs offer rich functional forms at the cost of statistical insight, while BFs possess a sophisticated statistical theory but are restricted to simpler functional forms in order to maintain tractable training and inference.

By drawing insights from both PSRs *and* RNNs we now develop a novel hybrid model, Predictive State Recurrent Neural Networks (PSRNNs). Like many successful RNN architectures, PSRNNs use (potentially deeply composed) bilinear transfer functions to combine information from multiple sources. We show that such bilinear functions arise naturally from state updates in Bayes filters, in which observations can be viewed as gating belief states. We show that PSRNNs directly generalize discrete PSRs, and can be learned effectively by combining Backpropagation Through Time (BPTT) with an approximately consistent method-of-moments initialization based on two-stage regression. We also show that PSRNNs can be factorized using tensor decomposition, reducing model size and suggesting interesting connections to existing multiplicative architectures such as LSTMs.

5.1 Predictive State Recurrent Neural Networks

In this section we introduce Predictive State Recurrent Neural Networks (PSRNNs), a new Hybrid model which is both a PSM and a Bayes Filter. PSRNNs allow for a principled initialization and refinement via BPTT. The key contributions which led to the development of PSRNNs are: 1) a new normalization scheme for PSRs which allows for effective refinement via BPTT; 2) the extention of the 2SR algorithm to a multilayered architecture; and 3) the optional use of a tensor decomposition to obtain a more scalable model.

5.1.1 HSE-PSRs as RNNs

As discussed in chapter 2 PSRs can be embedded in an RKHS to obtain HSE-PSRs. HSE-PSRs possess a rich functional form and can model a wide variety of dynamical systems. In Chapter 3 we showed that HSE-PSRs belong to the class of PSMs. We saw (see Equation 3.15) that the HSE-PSR state update can be expressed in terms of two operators $\hat{\Sigma}_{\phi_{t+1},o_t|h_t}$ and $\hat{\Sigma}_{o_t,o_t|h_t}$:

$$q_{t+1} = \hat{\Sigma}_{\phi_{t+1},o_t|h_t}(\hat{\Sigma}_{o_t,o_t|h_t})^{-1}o_t \tag{5.1}$$

We can reformulate this as follows. First the simpler task: predict $q_{t+1} = \mathbb{E}[f_{t+1} \mid h_{t+1})$ from o_t and h_t using kernel Bayes rule.

$$\begin{aligned} q_{t+1} &= \mathbb{E}[f_{t+1} \mid o_t h_t)] \\ &= C_{f_{t+1},o_t|h_t} C^{-1}_{o_t,o_t|h_t} o_t \end{aligned}$$

where

$$\begin{aligned} C_{f_{t+1},o_t|h_t} &= C_{(f_{t+1},o_t)h_t} C^{-1}_{h_t,h_t} h_t \\ C_{o_t,o_t|h_t} &= C_{(o_t,o_t)h_t} C^{-1}_{h_t,h_t} h_t \end{aligned}$$

We want to condition on (and update) q_t, not condition on h_t. However:

$$\begin{aligned} q_t &= \mathbb{E}(f_t \mid h_t) \\ &= C_{f_t,h_t} C^{-1}_{h_t,h_t} h_t \end{aligned}$$

and so

$$C^{-1}_{h_t,h_t} h_t = C^{\dagger}_{f_t,h_t} q_t$$

Here o_t, and q_t are each elements of an RKHS. If we represent them explicitly using Random Features (see Chapter 2) each is a finite dimensional vector.

Let $W = C_{(f_{t+1},o_t)h_t} C^{-1}_{h_t,h_t}$ and $Z = C_{(o_t,o_t)h_t} C^{-1}_{h_t,h_t}$, each is a 3-mode tensor. Therefore our update equation becomes:

$$q_{t+1} = (W \times_3 q_t)\,(Z \times_3 q_t)^{-1} \times_2 o_t. \tag{5.2}$$

We can now view (5.2) as an RNN, parameterized by tensors W and Z. In fact if we wanted to we could ignore all of the preceding theory and learn such a network entirely via BPTT.

The key difference between this network and more conventional networks, is the presence of the matrix inverse. This is not a commonly used operation in RNNs because it is poorly conditioned, making it difficult to optimize via local optimization techniques such as BPTT. Suitability for local optimization is an essential component of any useful RNN, therefore if we wish to obtain a true hybrid model, we need a model which is well conditioned and can be easily optimized via BPTT. We now show how obtain such a model by extending PSRs to obtain PSRNNs, a family of models which share a similar update and statistical theory, but do not require a matrix inverse in the state update.

5.1.2 From PSRs to PSRNNs

The basic building block of a PSRNN is a 3-mode tensor, which can be used to compute a bilinear combination of two input vectors. We note that, while bilinear operators are not a new development (e.g., they have been widely used in a variety of systems engineering and control applications for many years [78]), the current chapter shows how to chain these bilinear components together into a powerful new predictive model.

Let q_t and o_t be the state and observation at time t. Let W be a 3-mode tensor, and let q be a vector. The 1-layer state update for a PSRNN is defined as:

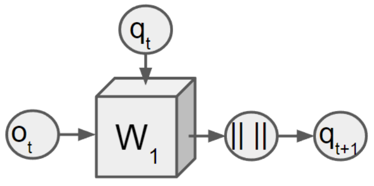
$$q_{t+1} = \frac{W \times_2 o_t \times_3 q_t + b}{\|W \times_2 o_t \times_3 q_t + b\|_2} \tag{5.3}$$

Here the 3-mode tensor of weights W and the bias vector b are the model parameters. This architecture is illustrated in Figure 5.1a. It is similar, but not identical, to the PSR update (Eq. 5.2); sec 5.1.1 gives more detail on the relationship. This model may appear simple, but crucially the tensor contraction $W \times_2 o_t \times_3 q_t$ integrates information from b_t and o_t multiplicatively, and acts as a gating mechanism, as discussed in more detail in section 5.5.

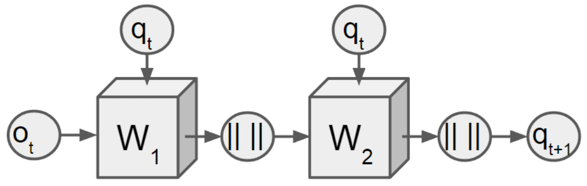
The typical approach used to increase modeling capability for BFs (including PSRs) is to use an initial fixed nonlinearity to map inputs up into a higher-dimensional space [78, 104]. PSRNNs incorporate such a step, via RFFs. However, a multilayered architecture typically offers higher representation power for a given number of parameters [17].

To obtain a multilayer PSRNN, we stack the 1-layer blocks of Eq. (5.3) by providing the output of one layer as the observation for the next layer. (The state input for each layer remains the same.) In this way we can obtain arbitrarily deep RNNs. This architecture is displayed in Figure 5.1b.

We choose to chain on the observation (as opposed to on the state) as this architecture leads to a natural extension of 2SR to multilayered models (see Sec. 5.3). In addition, this architecture is consistent with the typical approach for constructing multilayered LSTMs/GRUs [64]. Finally, this architecture is suggested by the full normalized form of an HSE PSR, where the observation is passed through two layers.



(a) Single Layer PSRNN



(b) Multilayer PSRNN

Figure 5.1: PSRNN architecture: See equation 5.3 for details. We omit bias terms to avoid clutter.

5.2 Theory

We now provide a theoretical justification for PSRNNs by showing that in the case of discrete observations and a single layer the PSRNN provides a good approximation to a consistent model.

We first show that in the discrete setting using a matrix inverse is equivalent to a sum normalization. We subsequently show that, under certain conditions, two-norm normalization has the same effect as sum-normalization.

Let q_t be the PSR state at time t, and o_t be the observation at time t (as an indicator vector). In this setting the covariance matrix $C_t = E[o_t \times o_t | o_{1:t-1}]$ will be diagonal. By assumption, the normalization term Z in PSRs is defined as a linear function from q_t to C_t, and when we learn PSRNNs by 2-stage regression we estimate this linear function consistently. Hence, for all q_t, $Z \times_3 q_t$ is a diagonal matrix, and $(Z \times_3 q_t)^{-1}$ is also a diagonal matrix. Furthermore, since o_t is an indicator vector, $(Z \times_3 q_t)^{-1} \times_2 o_t = o_t / P(o_t)$ in the limit. We also know that as a probability distribution, q_t should sum to one. This is equivalent to dividing the unnormalized update $\hat{q}_{t+1}$ by its sum. i.e.

$$\begin{aligned} q_{t+1} &= \hat{q}_{t+1} / P(o_t) \\ &= \hat{q}_{t+1} / (\mathbf{1}^\top \hat{q}_{t+1}) \end{aligned}$$

Now consider the difference between the sum normalization $\hat{q}_{t+1}/(\mathbf{1}^\top \hat{q}_{t+1})$ and the two-norm normalization $\hat{q}_{t+1} / \left\| \hat{q}_{t+1} \right\|_2$. Since q_t is a probability distribution, all elements will be positive, hence the sum norm is equivalent to the 1-norm. In both settings, normalization is equivalent to projection onto a norm ball. Now let S be the set of all valid states. Then if the diameter of S is small compared to the distance from (the convex hull of) S to the origin then the local curvature of the 2-norm ball will be negligible, and both cases will be approximately equivalent to projection onto a plane. We note we can obtain an S with this property by augmenting our state with a set of constant features.

5.3 Learning Multilayer PSRNNs

As a hybrid model PSRNNs are both valid PSMs and well conditioned RNNs. Therefore we can initialize PSRNNs using two-stage regression, followed by refinement via BPTT.

we now extend two-stage regression to multilayered PSRNNs. Suppose we have learned a 1-layer PSRNN P using two-stage regression. We can use P to perform filtering on a dataset to generate a sequence of estimated states $\hat{q}_1, ..., \hat{q}_n$. According to the architecture described in Figure 5.1b, these states are treated as observations in the second layer. Therefore we can initialize the second layer by an additional iteration of two-stage regression *using our estimated states $\hat{q}_1, ..., \hat{q}_n$ in place of observations*. This process can be repeated as many times as desired to initialize an arbitrarily deep PSRNN. If the first layer were learned perfectly, the second layer would be superfluous; however, in practice, we observe that the second layer is able to learn to improve on the first layer's performance.

Once we have obtained a PSRNN using the 2SR approach described above, we can use BPTT to refine the PSRNN. We note that we choose to use 2-norm divisive normalization because it is not practical to perform BPTT through the matrix inverse required in PSRs: the inverse operation is ill-conditioned in the neighborhood of any singular matrix. We observe that 2SR provides us with an initialization which converges to a good local optimum.

5.4 Factorized PSRNNs

In this section we show how the PSRNN model can be factorized to reduce the number of parameters prior to applying BPTT.

Let (W, b_0) be a PSRNN block. Suppose we decompose W using CP decomposition to obtain

$$W = \sum_{i=1}^{n} a_i \otimes b_i \otimes c_i$$

Let A (similarly B, C) be the matrix whose ith row is a_i (respectively b_i, c_i). Then the PSRNN state update (equation (5.3)) becomes (up to normalization):

$$\begin{aligned} q_{t+1} &= W \times_2 o_t \times_3 q_t + b && (5.4) \\ &= (A \otimes B \otimes C) \times_2 o_t \times_3 q_t + b && (5.5) \\ &= A^T (Bo_t \odot Cq_t) + b && (5.6) \end{aligned}$$

where $\odot$ is the Hadamard product. We call a PSRNN of this form a *factorized PSRNN*. This model architecture is illustrated in Figure 5.2. Using a factorized PSRNN provides us with complete control over the size of our model via the rank of the factorization. Importantly, it decouples the number of model parameters from the number of states, allowing us to set these two hyperparameters independently.

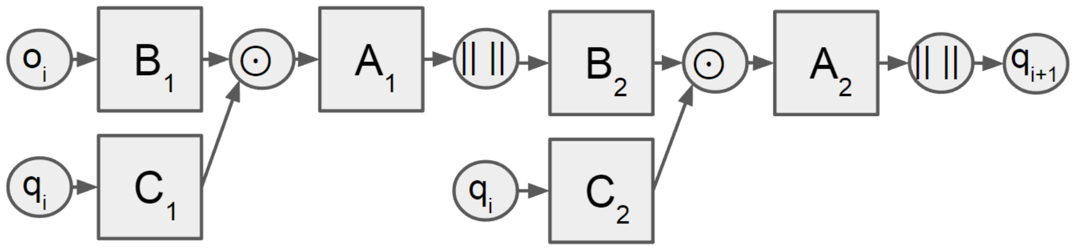


Figure 5.2: Factorized PSRNN Architecture

We determined experimentally that factorized PSRNNs are poorly conditioned when compared with PSRNNs, due to very large and very small numbers often occurring in the CP decomposition. To alleviate this issue, we need to initialize the bias b in a factorized PSRNN to be a small multiple of the mean state. This acts to stabilize the model, regularizing gradients and preventing us from moving away from the good local optimum provided by 2SR.

We note that a similar stabilization happens automatically in randomly initialized RNNs: after the first few iterations the gradient updates cause the biases to become non-zero, stabilizing the model and resulting in subsequent gradient descent updates being reasonable. Initialization of the biases is only a concern for us because we do not want the original model to move away from our carefully prepared initialization due to extreme gradients during the first few steps of gradient descent.

In summary, we can learn factorized PSRNNs by first using 2SR to initialize a PSRNN, then using CP decomposition to factorize the tensor model parameters to obtain a factorized PSRNN, then applying BPTT to the refine the factorized PSRNN.

5.5 Discussion

The value of bilinear units in RNNs was the focus of recent work by Wu et al [121]. They introduced the concept of Multiplicative Integration (MI) units — components of the form $Ax \odot By$ — and showed that replacing additive units by multiplicative ones in a range of architectures leads to significantly improved performance. As Eq. (5.6) shows, factorizing W leads precisely to an architecture with MI units.

Modern RNN architectures such as LSTMs and GRUs are known to outperform traditional RNN architectures on many problems [64]. While the success of these methods is not fully understood, much of it is attributed to the fact that these architectures possess a gating mechanism which allows them both to remember information for a long time, and also to forget it quickly. Crucially, we note that PSRNNs also allow for a gating mechanism. To see this consider a single entry in the factorized PSRNN update (omitting normalization).

$$[q_{t+1}]_i = \sum_j A_{ji}\left(\sum_k B_{jk}[o_t]_k \odot \sum_l C_{jl}[q_t]_l\right) + b \tag{5.7}$$

The current state q_t will only contribute to the new state if the function $\sum_k B_{jk}[o_t]_k$ of o_t is non-zero. Otherwise o_t will cause the model to forget this information: the bilinear component of the PSRNN architecture naturally achieves gating.

We note that similar bilinear forms occur as components of many successful models. For example, consider the (one layer) GRU update equation:

$$\begin{aligned}
z_t &= \sigma(W_z o_t + U_z q_t + c_z)\\
r_t &= \sigma(W_r o_t + U_r q_t + c_r)\\
q_{t+1} &= z_t \odot q_t + (1 - z_t) \odot \sigma(W_h o_t + U_h(r_t \odot q_t) + c_h)
\end{aligned}$$

The GRU update is a convex combination of the existing state q_t and and update term $W_h o_t + U_h(r_t \odot q_t) + c_h$. We see that the core part of this update term $U_h(r_t \odot q_t) + c_h$ bears a striking similarity to our factorized PSRNN update. The PSRNN update is simpler, though, since it omits the nonlinearity $\sigma(\cdot)$, and hence is able to combine pairs of linear updates inside and outside $\sigma(\cdot)$ into a single matrix.

Finally, we would like to highlight the fact that, as discussed in section 5.5, the bilinear form shared in some form by these models (including PSRNNs) resembles the first component of the Kernel Bayes Rule update function. This observation suggests that bilinear components are a natural structure to use when constructing RNNs, and may help explain the success of the above methods over alternative approaches. This hypothesis is supported by the fact that there are no activation functions (other than divisive normalization) present in our PSRNN architecture, yet it still manages to achieve strong performance.

5.6 Experiments

In this section we describe the datasets, models, model initializations, model hyperparameters, and evaluation metrics used in our experiments and present our results.

We use the following datasets in our experiments:

- **Penn Tree Bank (PTB)** This is a standard benchmark in the NLP community [80]. Due to hardware limitations we use a train/test split of 120780/124774 characters.
- **Swimmer** We consider the 3-link simulated swimmer robot from the open-source package OpenAI gym.[1] The observation model returns the angular position of the nose as well as the angles of the two joints. We collect 25 trajectories from a robot that is trained to swim forward (via the cross entropy with a linear policy), with a train/test split of 20/5.
- **Mocap** This is a Human Motion Capture dataset consisting of 48 skeletal tracks from three human subjects collected while they were walking. The tracks have 300 timesteps each, and are from a Vicon motion capture system. We use a train/test split of 40/8. Features consist of the 3D positions of the skeletal parts (e.g., upper back, thorax, clavicle).
- **Handwriting** This is a digit database available on the UCI repository [4, 46] created using a pressure sensitive tablet and a cordless stylus. Features are x and y tablet 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.

Models compared are LSTMs [78], GRUs [33], basic RNNs [47], KFs [70], PSRNNs, and factorized PSRNNs. All models except KFs consist of a linear encoder, a recurrent module, and a linear decoder. The encoder maps observations to a compressed representation; in the context of text data it can be viewed as a word embedding. The recurrent module maps a state and an observation to a new state and an output. The decoder maps an output to a predicted observation.[2] We initialize the LSTMs and RNNs with random weights and zero biases according to the Xavier initialization scheme [53]. We initialize the the KF using the 2SR algorithm described in [59]. We initialize PSRNNs and factorized PSRNNs as described in section 5.1.1.

In two-stage regression we use a ridge parameter of $10^{(-2)}n$ where n is the number of training examples (this is consistent with the values suggested in [27]). (Experiments show that our approach works well for a wide variety of hyperparameter values.) We use a horizon of 1 in the PTB experiments, and a horizon of 10 in all continuous experiments. We use 2000 RFFs from a Gaussian kernel, selected according to the method of [92], and with the kernel width selected as the median pairwise distance. We use 20 hidden states, and a fixed learning rate of 1 in all experiments. We use a BPTT horizon of 35 in the PTB experiments, and an infinite BPTT horizon in all other experiments. All models are single layer unless stated otherwise.

We optimize models on the PTB using Bits Per Character (BPC) and evaluate them using both BPC and one-step prediction accuracy (OSPA). We optimize and evaluate all continuous experiments using the Mean Squared Error (MSE).

5.6.1 Results

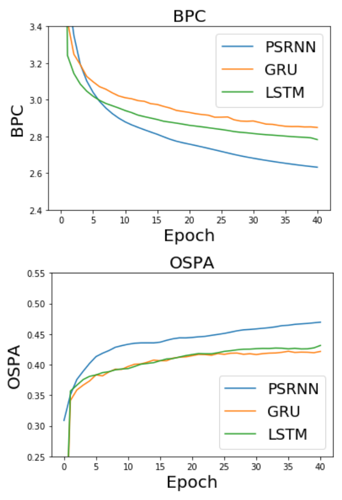
In Figure 5.3a we compare performance of LSTMs, GRUs, and Factorized PSRNNs on PTB, where all models have the same number of states and approximately the same number of parameters. To

[1] `https://gym.openai.com/`

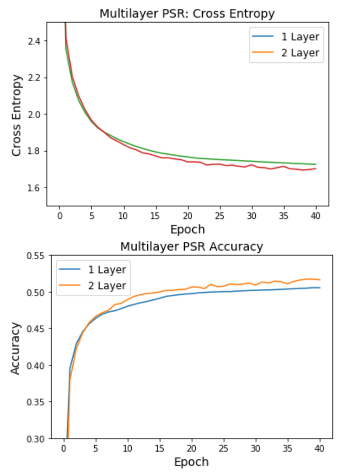
[2] This is a standard RNN architecture; e.g., a PyTorch implementation of this architecture for text prediction can be found at `https://github.com/pytorch/examples/tree/master/word_language_model`.

achieve this we use a factorized PSRNN of rank 60. We see that the factorized PSRNN significantly outperforms LSTMs and GRUs on both metrics. In Figure 5.3b we compare the performance of 1- and 2-layer PSRNNs on PTB. We see that adding an additional layer significantly improves performance. In Figure 5.3c we compare PSRNNs with factorized PSRNNs on the PTB. We see that PSRNNs outperform factorized PSRNNs regardless of rank, even when the factorized PSRNN has significantly more model parameters. (In this experiment, factorized PSRNNs of rank 7 or greater have more model parameters than a plain PSRNN.) This observation makes sense, as the PSRNN provides a simpler optimization surface: the tensor multiplication in each layer of a PSRNN is linear with respect to the model parameters, while the tensor multiplication in each layer of a Factorized PSRNN is bilinear. In addition, we see that higher-rank factorized models outperform lower-rank ones. However, it is worth noting that even models with low rank still perform well, as demonstrated by our rank 40 model still outperforming GRUs and LSTMs, despite having fewer parameters.

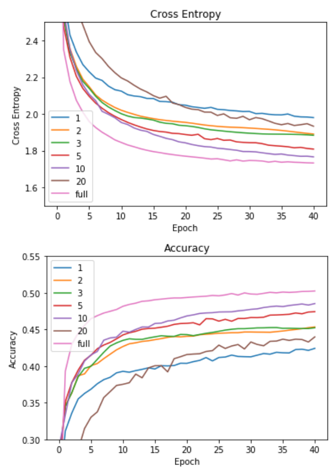
In Figure 5.4 we compare model performance on the Swimmer, Mocap, and Handwriting datasets. We see that PSRNNs significantly outperform alternative approaches on all datasets. In Figure 5.5 we attempt to gain insight into why using 2SR to initialize our models is so beneficial. We visualize the the one step model predictions before and after BPTT. We see that the behavior of the initialization has a large impact on the behavior of the refined model. For example the initial (incorrect) oscillatory behavior of the RNN in the second column is preserved even after gradient descent.



(a) BPC and OSPA on PTB. All models have the same number of states and approximately the same number of parameters.



(b) Comparison between 1- and 2-layer PSRNNs on PTB.



(c) Cross-entropy and prediction accuracy on Penn Tree Bank for PSRNNs and factorized PSRNNs of various rank.

Figure 5.3: PTB Experiments

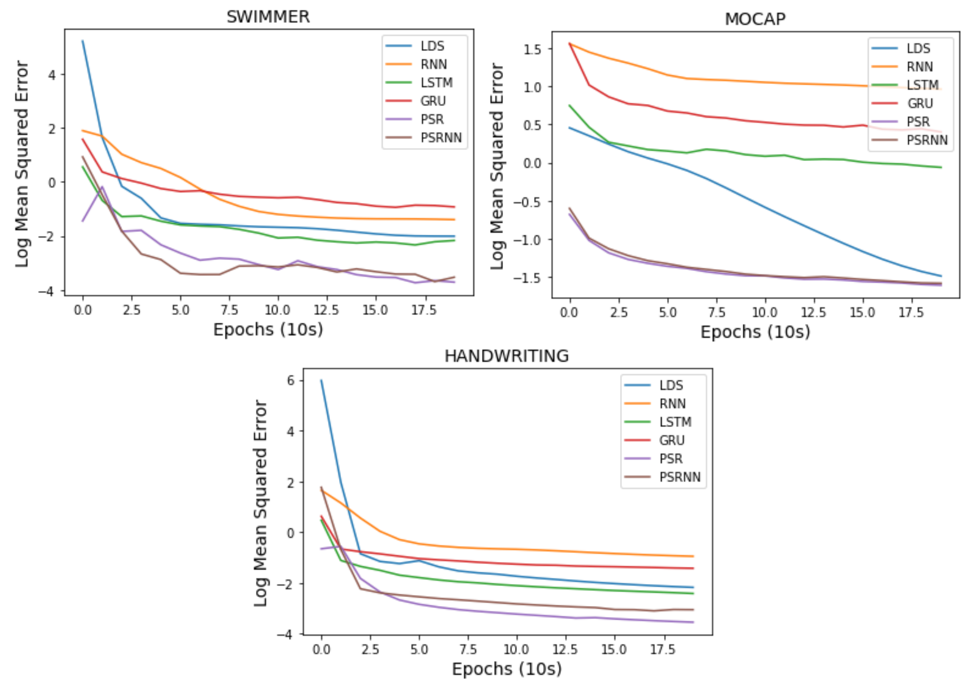


Figure 5.4: MSE vs Epoch on the Swimmer, Mocap, and Handwriting datasets

5.7 Related Work

It is well known that a principled initialization can greatly increase the effectiveness of local search heuristics. For example, Boots [19] and Zhang et al. [125] use subspace ID to initialize EM for linear dyanmical systems, and Ko and Fox [72] use N4SID [114] to initialize GP-Bayes filters.

Pasa et al. [88] propose an HMM-based pre-training algorithm for RNNs by first training an HMM, then using this HMM to generate a new, simplified dataset, and, finally, initializing the RNN weights by training the RNN on this dataset.

Belanger and Kakade [16] propose a two-stage algorithm for learning a KF on text data. Their approach consists of a spectral initialization, followed by fine tuning via EM using the ASOS method of Martens [81]. They show that this approach has clear advantages over either spectral learning or BPTT in isolation. Despite these advantages, KFs make restrictive linear-Gaussian assumptions that preclude their use on many interesting problems.

Haarnoja et al. [56] also recognize the complementary advantages of Bayes Filters and RNNs, and propose a new network architecture attempting to combine some of the advantages of both. Their approach differs substantially from ours as they propose a network consisting of a Bayes Filter concatenated with an RNN, which is then trained end-to-end via backprop. In contrast our entire network architecture has a dual interpretation as both a Bayes filter and a RNN. Because of this, our entire network can be initialized via an approximately consistent method of moments algorithm, something not possible in [56].

Finally, Kossaifi et al. [74] also apply tensor decomposition in the neural network setting. They propose a novel neural network layer, based on low rank tensor factorization, which can

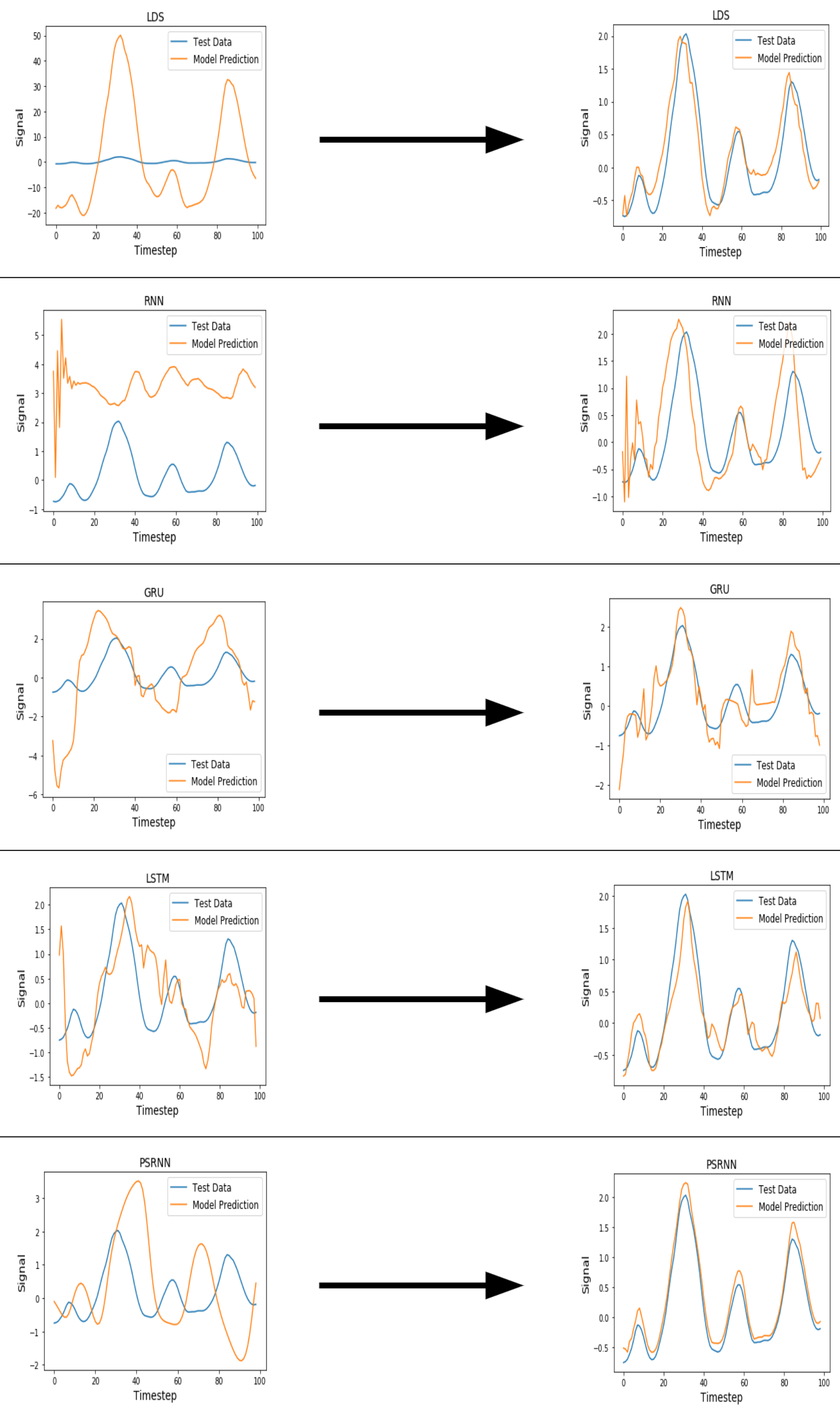


Figure 5.5: Test Data vs Model Prediction on a single feature of Swimmer. The first row shows initial performance. The second row shows performance after training. In order the columns show KF, RNN, GRU, LSTM, and PSRNN.

directly process tensor input. This is in contrast to a standard approach where the data is flattened to a vector. While they also recognize the strength of the multilinear structure implied by tensor weights, both their setting and their approach differ from ours: they focus on factorizing tensor input data, while we focus on factorizing parameter tensors which arise naturally from a kernelized interpretation of Bayes rule.

5.8 Conclusions

We present PSRNNs: a new approach for modelling time-series data that hybridizes Bayes filters with RNNs. PSRNNs have both a principled initialization procedure and a rich functional form. The basic PSRNN block consists of a 3-mode tensor, corresponding to bilinear combination of the state and observation, followed by divisive normalization. These blocks can be arranged in layers to increase the expressive power of the model. We showed that tensor CP decomposition can be used to obtain factorized PSRNNs, which allow flexibly selecting the number of states and model parameters. We showed how factorized PSRNNs can be viewed as both an instance of Kernel Bayes Rule and a gated architecture, and discussed links to existing multiplicative architectures such as LSTMs. We applied PSRNNs to 4 datasets and showed that we outperform alternative approaches in all cases.

Chapter 6

Hilbert Space Embedding of Hidden Quantum Markov Models

In this chapter we present our second hybrid model, Hilbert Space Embedding of Hidden Quantum Markov Models (HSE-HQMMs). Where PSRNNs brought together ideas from HSE-PSRs and RNNs, HSE-HQMMs include an additional element: Quantum Mechanics. Quantum Mechanics is a key formalism for modelling many dynamical systems in physics, and is often useful where alternative models fail. We provide a brief introduction to quantum mechanics, and show how it can be used to derive a new class of models. Specifically we investigate the link between Quantum Mechanics and HSEs and show that the sum rule and Bayes rule in Quantum Mechanics are equivalent to the kernel sum rule in HSEs and a special case of Nadaraya-Watson kernel regression, respectively. We show that these operations can be kernelized, and use these insights to propose a Hilbert Space Embedding of Hidden Quantum Markov Models (HSE-HQMM) to model dynamics. We present experimental results showing that HSE-HQMMs are competitive with state-of-the-art models like LSTMs and PSRNNs on several datasets, while also providing a nonparametric method for maintaining a probability distribution over continuous-valued features.

Overall, we present four contributions: (1) we show that the sum rule for QGMs is identical to the kernel sum rule for HSEs, while the Bayesian update in QGMs is equivalent to performing Nadaraya-Watson kernel regression, (2) we show how to kernelize these operations and argue that with the right choice of features, we are mapping our data to quantum systems and modeling dynamics as quantum state evolution, (3) we use these insights to propose a HSE-HQMM to model dynamics by mapping data to quantum systems and performing inference in Hilbert space, and, finally, (4) we present a learning algorithm and experimental results showing that HSE-HQMMs are competitive with other state-of-the-art methods for modeling sequences, while also providing a nonparametric method for estimating the distribution of continuous-valued features.

6.1 Quantum Mechanics

There is a great deal of missinformation and hype surrounding the field of Quantum Mechanics. Slogans such as “light is both a wave and a particle”, “the cat is neither dead nor alive until you look”, and “you can ask about the position or the momentum, but not both” have given the field a (mostly undeserved) reputation for complexity. While many of the problems in physics which are studied *using* quantum mechanics are indeed complex, quantum mechanics itself is not. Quantum Mechanics should be viewed as nothing more or less than a system for formalising and manipulating uncertainty. In this it is no different to Classical Probability theory, except in the way it choose to represent uncertainty, and the functions we choose to manipulate such representations.

Classical Probability theory, otherwise known as classical mechanics, represents uncertainty over a discrete system using the notion of a stochastic vector – a vector of real, non-negative numbers which sums to 1. We manipulate such vectors using stochastic matrices – matrices of non-negative real numbers where each column sums to 1. This system works because the space of stochastic vectors is closed under the action of stochastic matrices, in other words applying a stochastic matrix to a stochastic vector always results in another stochastic vector. This ensures we never end up with something which is not a valid distribution. An example of classical mechanics is provided in figure 6.1

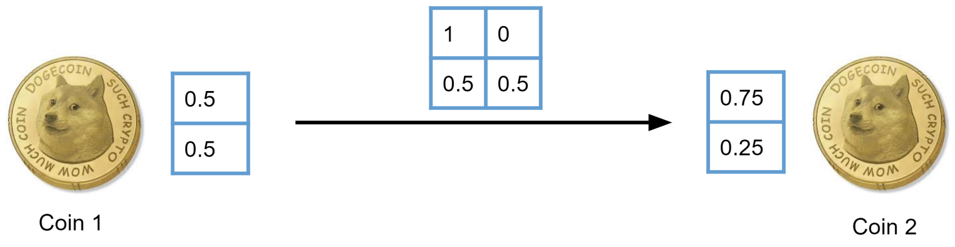


Figure 6.1: An example of classical mechanics. We transform a stochastic vector representing information about an unknown quantity (in this case a coin flip) into another stochastic vector using a stochastic matrix.

Quantum Mechanics takes an alternative approach, instead of representing uncertainty over a discrete system using stochastic vectors instead we use unitary vectors – a vector of complex numbers with norm 1. Crucially the square of the absolute value of a unitary vector is a stochastic vector, therefore every unitary vector corresponds to a valid probability distribution. In the language of quantum mechanics unitary vectors contain *probability amplitudes* whose squared values are probabilities. Instead of stochastic matrices we manipulate unitary vectors using unitary matrices – matrices whose conjugate transpose is equal to their inverse. We note that in the case of real numbers the class of unitary matrices is equal to the class of orthonormal matrices – matrices whoose columns are orthogonal and have norm 1.

Vectors and matrices in quantum mechanics are traditionally represented using *Bra-Ket* notation. In Braket notation a row vector x is denoted by $\langle x|$ called a *Bra* and a column vector is denoted by $|x\rangle$ called a *Ket*. Using this notation an inner product between x and y is denoted by $\langle x|y\rangle$ and an outer product between x and y is denoted by $|x\rangle\langle y|$. To maintain consistency with literature we also use this notation for this section.

In quantum mechanics unitary vectors are known as pure states. We we can combine quantum mechanics with classical mechanics to obtain a classical mixture of quantum pure states, known as a mixed state. We can think about this as a mixture of particles, each of which has a distribution represented as a pure state. We represent a mixed state using the notion of a density matrix.

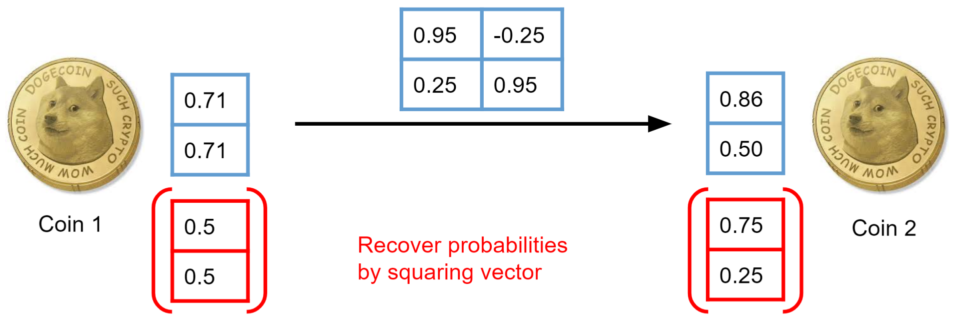


Figure 6.2: An example of quantum mechanics. We transform a unitary vector representing information about an unknown quantity (in this case a coin flip) into another unitary vector using a unitary matrix. The vectors in red are the squared absolute values of the unitary vectors.

Given a collection of pure states $x_1, ..., x_n$ and a classical probability distribution $p(x)$ over $x_i \forall i$ a density matrix D is defined as the expected outer product of the pure states:

$$D = \mathbb{E}_{x\ p}[|x\rangle\langle x|] \tag{6.1}$$

$$= \sum_i \Pr(x_i)|x_i\rangle\langle x_i| \tag{6.2}$$

If the pure states are orthogonal we can recover them from the density matrix using the eigendecomposition operation. An example of the construction of a mixed state is shown in figure 6.3

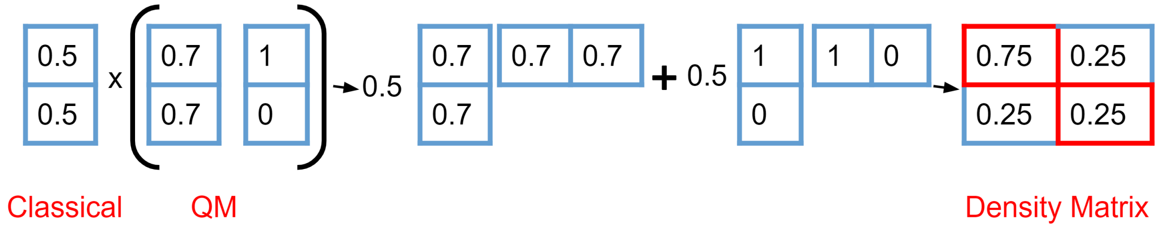


Figure 6.3: Example illustrating the construction of a mixed state (density matrix) from a distribution over two pure states

The diagonal entries of a density matrix give the probabilities of being in each system state, and off-diagonal elements represent *quantum coherences*, which have no classical interpretation. Consequently, the normalization condition is tr$(\hat{\rho}) = 1$. Uncertainty about an n-state system is represented by an $n \times n$ density matrix. The density matrix is the quantum analogue of the classical belief x.

6.2 Advantages of Quantum Mechanics

A natural question one might ask is why it's worth going through the effort to develop machine learning models which incorporate quantum mechanics at all. Why not just stick with the models based on classical mechanics that we already have? In this section we discuss the various ways in which quantum mechanical systems differ from classical ones, and the possible advantages this offers when building machine learning systems. Note that we do not claim that our models, as presented here, provably benefit from all of these properties. Rather this material instead is intended to serve as a thought-provoking discussion, and a general motivation for the study of quantum - machine learning systems. In this chapter we will take the initial steps towards building such models, with the hope that future work may realize some of these benefits.

6.2.1 Continuous Time

Many real world systems are continuous in the time domain. In contrast the vast majority of existing models assume that observations occur at fixed discrete intervals. This is often a reasonable assumption as measuring devices tend to collect samples at fixed intervals, it still falls short in several cases. For example there might be missing data points, or we might collect data at irregular intervals. Both cases are difficult to deal with when filtering algorithms are designed around fixed regular observations. Furthermore we might want to make predictions about the behavior of our system between scheduled observations. For example we might have a GPS system that reports an estimate of the location once per second, but we might want to estimate our location after half a second.

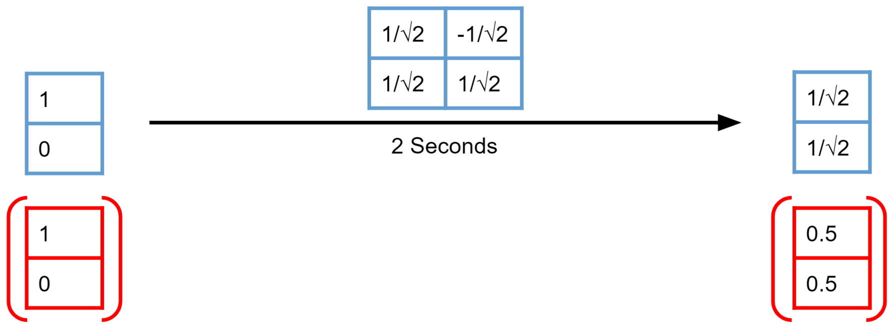
The problem is that in most Bayes filters such as HMMs, when we are given an operator which advances the system forwards one second in time, there is no obvious way to extend it to an operator which advances us forwards half a second in time (or by some other arbitrary multiple). Formally given a state q_t and an operator A which advances q_t to q_{t+1} via $q_t = Aq_t$ where t is measured in 1 second intervals there should be some way to obtain a new operator B such that B advances us forwards half a second in time. Clearly a desirable property of any such B is that applying it twice should be the same as applying A once. In other words what we really want is to be able to take the square root of the update operator A. Unfortunately in classical mechanics A is a stochastic matrix, and this square root is unlikely to be a valid stochastic matrix.

Fortunately this is not the case when working with Quantum Mechanics. This is because the square root of a unitary operator is guaranteed to exist, and furthermore is guaranteed to be unitary. The intuition behind this is that every unitary operator can be thought of as a set of rotations in the complex field. Therefore we can always compute arbitrary fractions of this operator in terms of partial rotations. Another way to see this is via the eigendecomposition. Given a unitary matrix A its eigendecomposition $A = USU^*$ is gauaranteed to exist. Therefore for arbitrary $n \in [0, 1]$ we can compute $A^{\frac{1}{n}}$ as:

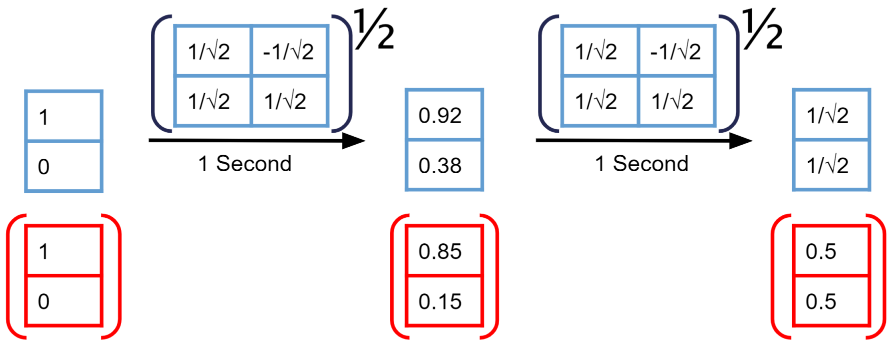
$$A^{\frac{1}{n}} = US^{\frac{1}{n}}U^* \tag{6.3}$$

since A is unitary, all of the eigenvalues of A in have absolute value of 1. Therefore the eigenvalues of $A^{\frac{1}{n}}$ also have value 1. This idea is illustrated in figure 6.4

Therefore by developing models based on quantum mechanics we hope to one day obtain models which are naturally continuous in the time domain.



(a) Operator A advances the state 2 seconds forwards in time



(b) The square root of operator A advances the state 1 second forwards in time

Figure 6.4: Example of how quantum mechanics allow for continuous time. Given an operator which advances the system state by 2 seconds we can obtain an operator which advances the system one second by taking the square root of the original operator.

6.2.2 Richer Class of Transformations

Another potential benefit of quantum mechanics is that it represents a potentially richer class of functions. Many operations which require exponential space for classical systems can be represented using linear space using quantum mechanics. This suggests that it may be possible to build much more compact models by using quantum mechanics.

6.2.3 Easier to Learn

A third potential benefit of quantum mechanics is that it may lead to models which are easier to learn (in the optimization sense) than many many systems which rely on classical mechanics. As we discussed in chapter 2 many Bayes Filters are difficult to learn because of the positivity constraint. While there exist techniques such as projected gradient descent which attempt to solve this problem they often produce poor results. In contrast there is no such constraint on quantum mechanical systems as unitary matrices may contain negative numbers.

It is important to note here that we have trade one constraint for another: In order to learn a valid quantum mechanical model we are required to constrain our optimization procedure to valid unitary operators. Nonetheless this is a different problem, and may prove easier to solve.

6.2.4 Rotational Symmetry

Finally quantum mechanical systems may offer a particular advantage when modelling systems which exhibit rotational symmetry. Many dynamical systems exhibit rotational symmetry in their features, where the feature space forms an n dimensional sphere. For example the wind direction is a feature which lies on a circle, the images from a camera lie on the surface of a 3 dimensional sphere, etc. These features can prove difficult to model using classical mechanics. To see why consider the state space.

In classical mechanics our state space consists of all valid stochastic vectors. The space of such vectors of length n is known as the *Probability simplex* of dimension n. For length 2 vectors this is a line, for length 3 vectors a triangle, etc. This means it can be difficult to model features with rotational symmetry, where if we move far enough away from our original state we will eventuall end back at the same state. In contrast the space of valid unitary vectors is a sphere of dimension n. This perfectly matches the symmetry of the underlying dynamical system. The state space for 3 dimensional systems is illustrated in figure 6.5.

(a) Classical Mechanics State Space – the probability simplex (b) Quantum Mechanics State space – the unit sphere

Figure 6.5: Illustration of the state space: Probability Simplex vs. Unit Sphere

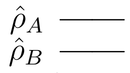
.

6.3 Quantum Circuits

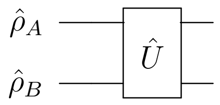
Prior to launching into a discussion of quantum mechanics in the context of dynamical system models we take a short detour to introduce the concept of quantum circuits, a useful tool used throughout the remainder of this chapter. Quantum Circuits provide a useful graphical language for illustrating models involving quantum mechanics, in much the same way that graphical models provide a useful graphical language for illustrating conditional independence assumptions in classical mechanics. For this reason quantum circuits are ubiquitous in the quantum computing literature, and we refer the reader to Nielsen and Chuang [86] for a more complete treatment. We note that we use quantum circuits for d-dimensional quantum states, while traditionally quantum circuits illustrate operations on 2-dimensional quantum states ('qubits').

Generally speaking, the boxes in a quantum circuit represent a bilinear unitary operation (or matrix) on a density matrix, and the wire entering from the left represents the input, and the wire exiting the right represents the output. A matrix $\hat{U}$ that take multiple wires as input/output are operating on a state space that is the tensor product of the state spaces of the inputs. A matrix $\hat{u}$

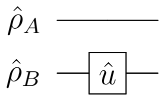
that takes one wire as input while a parallel wire passes untouched can equivalently be written as a matrix $(\mathbb{I} \otimes \hat{u})$ operating on the joint state space of the two inputs. Finally, note that while we can always tensor two smaller state spaces to obtain a larger state space which we then operate on, the reverse isn't usually possible even though we may draw the wires separately, i.e., it may not be possible to decompose the output of an operation on a larger state space into the smaller state spaces. This is analogous to how distributions of two random variables can be tensored to obtain the joint, but the joint can only be factored if the constituent variables are independent. These examples are illustrated below.



(a) $\hat{\rho}_A \otimes \hat{\rho}_B = \hat{\rho}_{AB}$ if the two states start off separable



(b) $\hat{U}(\hat{\rho}_A \otimes \hat{\rho}_B)\hat{U}^\dagger = \hat{\rho}'_{AB} \neq \hat{\rho}'_A \otimes \hat{\rho}'_b$ in general



(c) $\hat{\rho}_A \otimes \hat{u}\hat{\rho}_B\hat{u}^\dagger = (\mathbb{I} \otimes \hat{u})\rho\hat{_{AB}}(\mathbb{I} \otimes \hat{u})^\dagger$

Figure 6.6: Simple quantum circuit operations

6.4 Quantum Graphical Models

In this section we connect the core concepts of classical mechanics (Joint Distribution, Marginalization, Sum rule, chain rule, Bayes Rule) to the quantum mechanics setting. Once we have established these connections we can use them to build quantum mechanical models for dynamical systems. Our key assumption throughout this section is that that the density matrix is the quantum analogue of a classical belief state. This work is extends earlier work by Srinivasan et al. [108].

Joint Distributions The joint distribution of an n-state variable A and m-state variable B can be written as an $nm \times nm$ 'joint density matrix' $\hat{\rho}_{\text{AB}}$. When A and B are independent, $\hat{\rho}_{\text{AB}} = \hat{\rho}_A \otimes \hat{\rho}_B$. As a valid density matrix, the diagonal elements represent probabilities corresponding to the states in the Cartesian product of the basis states of the composite variables (so $\text{tr}\,(\hat{\rho}_{\text{AB}}) = 1$).

Marginalization Given a joint density matrix, we can recover the marginal 'reduced density matrix' for a subsystem of interest with the 'partial trace' operation. This operation is the quantum analogue of classical marginalization. For example, the partial trace for a two-variable joint system $\hat{\rho}_{AB}$ where we trace over the second particle to obtain the state of the first particle is:

$$\hat{\rho}_A = \text{tr}_B\,(\hat{\rho}_{AB}) = \sum_j {}_B\langle j|\hat{\rho}_{AB}|j\rangle_B \tag{6.4}$$

Finally, we discuss the quantum analogues of the sum rule and Bayes rule. Consider a prior $\pi = P(X)$ and a likelihood $P(Y|X)$ represented by the column stochastic matrix $\mathbf{A}$. We can then ask two questions: what are $P(Y)$ and $P(X|y)$?

Sum Rule The classical answer to the first question involves multiplying the likelihood with the prior and marginalizing out X, like so:

$$P(Y) = \sum_x P(Y|x)P(x) \tag{6.5}$$

$$= \mathbf{A}\pi \tag{6.6}$$

Srinivasan et al. [108] show how we can construct a quantum circuit to perform the classical sum rule (illustrated in Figure 6.7a. First, recall that all operations on quantum states must be represented by unitary matrices in order to preserve the 2-norm of the state. $\hat{\rho}_{env}$ is an environment 'particle' always prepared in the same state that will eventually encode $\hat{\rho}_Y$: it is initially a matrix with zeros everywhere except $\hat{\rho}_{1,1} = 1$. Then, if the prior π is encoded in a density matrix $\hat{\rho}_X$, and the likelihood table $\mathbf{A}$ is encoded in a higher-dimensional unitary matrix, we can replicate the classical sum rule. Letting the prior $\hat{\rho}_X$ be *any* density matrix and $\hat{U}_1$ be any unitary matrix generalizes the circuit to perform the 'quantum sum rule'. This circuit can be written as the following operation (the unitary matrix produces the joint distribution, the partial trace carries out the marginalization):

$$\hat{\rho}_Y = \text{tr}_X \left(\hat{U}_1 \left(\hat{\rho}_X \otimes \hat{\rho}_{env} \right) \hat{U}_1^\dagger \right) \tag{6.7}$$

Bayes Rule Classically, we perform Bayesian update as follows (where $\text{diag}(\mathbf{A}_{(:,y)})$ selects the row of matrix $\mathbf{A}$ corresponding to observation y and stacks it along a diagonal):

$$P(X|y) = \frac{P(y|X)P(X)}{\sum_x (y|x)P(x)} \tag{6.8}$$

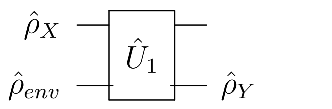
$$= \frac{\text{diag}(\mathbf{A}_{(y,:)})\pi}{\mathbb{1}^T \text{diag}(\mathbf{A}_{(y,:)})\pi} \tag{6.9}$$

The quantum circuit for Bayesian update presented by Srinivasan et al. [108] is shown in Figure 6.7b. It involves encoding the prior in $\hat{\rho}_X$ as before, and encoding the likelihood table $\mathbf{A}$ in a unitary matrix $\hat{U}_2$. Applying the unitary matrix $\hat{U}_2$ prepares the joint state $\hat{\rho}_{XY}$, and we apply a von Neumann projection operator (denoted $\hat{P}_y$) corresponding to the observation y (the D-shaped symbol in the circuit), to obtain the conditioned state $\hat{\rho}_{X|y}$ in the first particle. The projection operator selects the entries from the joint distribution $\hat{\rho}_{XY}$ that correspond to the actual observation y, and zeroes out the other entries, analogous to using an indicator vector to index into a joint probability table. This operation can be written (denominator renormalizes to recover a valid density matrix) as:

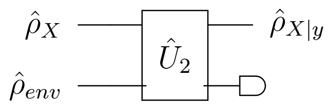
$$\hat{\rho}_{X|y} = \frac{\text{tr}_{env} \left(P_y \hat{U}_2 \left(\hat{\rho}_X \otimes \hat{\rho}_{env} \right) \hat{U}_2^\dagger P_y^\dagger \right)}{\text{tr} \left(\text{tr}_{env} \left(P_y \hat{U}_2 \left(\hat{\rho}_X \otimes \hat{\rho}_{env} \right) \hat{U}_2^\dagger P_y^\dagger \right) \right)} \tag{6.10}$$

However, there is an *alternate* quantum circuit that could implement Bayesian conditioning. Consider re-writing the classical Bayesian update as a linear update as follows:

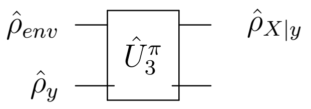
$$P(X|y) = \left(\mathbf{A} \cdot \text{diag}(\pi) \right)^T \left(\text{diag} \left(\mathbf{A}\pi \right) \right)^{-1} e_y \tag{6.11}$$



(a) Quantum circuit to compute $P(Y)$



(b) Quantum circuit to compute $P(X|y)$



(c) Alternate circuit to compute $P(X|y)$

Figure 6.7: Quantum-circuit analogues of conditioning in graphical models

where $(\mathbf{A} \cdot \text{diag}(\pi))^T$ yields the joint probability table $P(X, Y)$, $(\text{diag}(\mathbf{A}\pi))^{-1}$ is a diagonal matrix with the inverse probabilities $\frac{1}{P(Y=y)}$ on the diagonal, serving to renormalize the columns of the joint probability table $P(X, Y)$. Thus, $(\mathbf{A} \cdot \text{diag}(\pi))^T (\text{diag}(\mathbf{A}\pi))^{-1}$ produces a column-stochastic matrix, and e_y is just an indicator vector that selects the column corresponding to the observation y. Then, just as the circuit in Figure 6.7a is the quantum generalization for Equation 6.6, we can use the quantum circuit shown in 6.7c for this alternate Bayesian update. Here, $\hat{\rho}_y$ encodes the indicator vector corresponding to the observation y, and $\hat{U}_3^{\pi}$ is a unitary matrix constructed using the prior π on X. Letting $\hat{U}_3^{\pi}$ to be any unitary matrix constructed from some prior on X would give an alternative quantum Bayesian update.

These are two *different* ways of generalizing classical Bayesian rule within quantum graphical models. So which circuit should we use? One major disadvantage of the second approach is that we must construct different unitary matrices $\hat{U}_3^{\pi}$ for different priors on X (note that in figure 6.7 $\hat{\rho}_X$ corresponds to the prior). The first approach also explicitly involves measurement, which is nicely analogous to classical observation. As we will see in the next section, the two circuits are different ways of performing inference in Hilbert space, with the first approach being equivalent to Nadaraya-Watson kernel regression and the second approach being equivalent to kernel Bayes rule for Hilbert space embeddings.

6.5 Translating to the language of Hilbert Space Embeddings

In the previous section, we generalized graphical models to quantum graphical models using the quantum view of probability. And since quantum states live in complex Hilbert spaces, inference in QGMs happens in Hilbert space. Here, we re-write the operations on QGMs in the language of Hilbert space embeddings, which should be more familiar to the statistical machine learning community.

6.5.1 Hilbert Space Embeddings

Previous work [103] has shown that we can embed probability distributions over a data domain $\mathcal{X}$ in a reproducing kernel Hilbert space (RKHS) $\mathcal{F}$ – a Hilbert space of functions, with some kernel k. The feature map $\phi(x) = k(x, \cdot)$ maps data points to the RKHS, and the kernel function satisfies $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{F}}$. Additionally, the dot product in the Hilbert space satisfies the reproducing property:

$$\langle f(\cdot), k(x, \cdot)\rangle_{\mathcal{F}} = f(x) \tag{6.12}$$

$$\langle k(x, \cdot), k(x', \cdot)\rangle_{\mathcal{F}} = k(x, x') \tag{6.13}$$

Mean Embeddings

The following equations describe how a distribution of a random variable X is embedded as a *mean map* [103], and how to empirically estimate the mean map from data points $\{x_1, \ldots, x_n\}$ drawn i.i.d from $P(X)$, respectively:

$$\mu_X := \mathbb{E}_X [\phi(X)] \tag{6.14}$$

$$\hat{\mu}_X = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i) \tag{6.15}$$

Quantum Mean Maps We still take the expectation of the features of the data, except we require that the feature maps $\phi(\cdot)$ produce valid density matrices representing pure states (i.e., rank 1). Consequently, quantum mean maps have the nice property of having probabilities along the diagonal. Note that these feature maps can be complex and infinite, and in the latter case, they map to *density operators*. For notational consistency, we require the feature maps to produce rank-1 *vectorized* density matrices (by vertically concatenating the columns of the matrix), and treat the quantum mean map as a vectorized density matrix $\mu_X = \rho_X$.

Cross-Covariance Operators

Cross-covariance operators can be used to embed joint distributions; for example, the joint distribution of random variables X and Y can be represented as a cross-covariance operator (see Song et al. [106] for more details):

$$\mathcal{C}_{XY} := \mathbb{E}_{XY} [\phi(X) \otimes \phi(Y)] \tag{6.16}$$

Quantum Cross-Covariance Operators The quantum embedding of a joint distribution $P(X, Y)$ is a square $mn \times mn$ density matrix $\hat{\rho}_{XY}$ for constituent $m \times m$ embedding of a sample from $P(X)$ and $n \times n$ embedding of a sample from $P(Y)$. To obtain a quantum cross-covariance matrix $\mathcal{C}_{XY}$, we simply reshape $\hat{\rho}_{XY}$ to an $m^2 \times n^2$ matrix, which is also consistent with estimating it from data as the expectation of outer product of feature maps $\phi(\cdot)$ that produce vectorized density matrices.

6.5.2 Quantum Sum Rule as Kernel Sum Rule

We now re-write the quantum sum rule for quantum graphical models from Equation 6.7, in the language of Hilbert space embeddings. Srinivasan et al. [108] showed that Equation 6.7 can be written as $\hat{\rho}_Y = \sum_i V_i \hat{U} W \hat{\rho}_X W^\dagger \hat{U}^\dagger V_i^\dagger$, where matrices W and V tensor with an environment particle and partial trace respectively. Observe that a quadratic matrix operation can be simplified to a linear operation, i.e., $\hat{U}\hat{\rho}\hat{U}^\dagger = \text{reshape}((\hat{U}^* \otimes \hat{U})\rho)$ where ρ is the vectorized density matrix $\hat{\rho}$. Then:

$$\mu_Y = \sum_i \left(\left(V_i\hat{U}W\right)^* \otimes \left(V_i\hat{U}W\right)\right) \tag{6.17}$$

$$\mu_X = \left(\sum_i \left(V_i\hat{U}W\right)^* \otimes \left(V_i\hat{U}W\right)\right) \tag{6.18}$$

$$\mu_X = A\mu_X \tag{6.19}$$

where $A = (\sum_i (V_i\hat{U}W)^* \otimes (V_i\hat{U}W))$. We have re-written the complicated transition update as a simple linear operation, though A should have constraints to ensure the operation is valid according to quantum mechanics. Specifically each operator should be sub-unitary, and the collection of such operators should be normalized. Consider estimating A from data by solving a least squares problem: suppose we have data (Υ_X, Φ_Y) where $\Phi \in \mathbb{R}^{d_1 \times n}, \Upsilon \in \mathbb{R}^{d_2 \times n}$ are matrices of n vectorized d_1, d_2-dimensional density matrices and n is the number of data points. Solving for A gives us $A = \Phi_Y \Upsilon_X^\dagger (\Upsilon_X \Upsilon_X^\dagger)^{-1}$. But $\Phi_Y \Upsilon_X^\dagger = n \cdot \mathcal{C}_{YX}$ where $\mathcal{C}_{YX} = \frac{1}{n}\sum_i^n (\mu_{Y_i} \otimes \mu_{X_i}^\dagger)$. Then, $A = \mathcal{C}_{YX}\mathcal{C}_{XX}^{-1}$, allowing us to re-write Equation 6.17 as:

$$\mu_Y = \mathcal{C}_{YX}\mathcal{C}_{XX}^{-1}\mu_X \tag{6.20}$$

But this is exactly the kernel sum rule from Song et al. [106], with the conditional embedding operator $\mathcal{C}_{Y|X} = \mathcal{C}_{YX}\mathcal{C}_{XX}^{-1}$. Thus, when the feature maps that produce valid (vectorized) rank-1 density matrices, the quantum sum rule is identical to the kernel sum rule. One thing to note is that solving for A using least-squares needn't preserve the quantum-imposed constraints; so either the learning algorithm must force these constraints, or we project μ_Y back to a valid density matrix.

Finite Sample Kernel Estimate We can straightforwardly adopt the kernelized version of the conditional embedding operator from HSEs [106] (λ is a regularization parameter):

$$\mathcal{C}_{Y|X} = \Phi(K_{xx} + \lambda\mathbb{I})^{-1}\Upsilon^\dagger \tag{6.21}$$

where $\Phi = (\phi(y_1), \ldots, \phi(y_n))$, $\Upsilon = (\phi(x_1), \ldots, \phi(x_n))$, and $K = \Upsilon^\dagger\Upsilon$, and these feature maps produce vectorized rank-1 density matrices. The data points in Hilbert space can be written as $\mu_Y = \Phi\alpha_Y$ and $\mu_X = \Upsilon\alpha_X$ where $\alpha \in \mathbb{R}^n$ are weights for the training data points, and the kernel quantum sum rule is simply:

$$\mu_Y = \mathcal{C}_{Y|X}\mu_X \Rightarrow \Phi\alpha_Y = \Phi(K_{xx} + \lambda\mathbb{I})^{-1}\Upsilon^\dagger\Upsilon\alpha_X \Rightarrow \alpha_Y = (K_{xx} + \lambda\mathbb{I})^{-1}K_{xx}\alpha_X \tag{6.22}$$

6.5.3 Quantum Bayes Rule as Nadaraya-Watson Kernel Regression

Here, we re-write the Bayesian update for QGMs from Equation 6.10 in the language of HSEs. First, we modify the quantum circuit in 6.7b to allow for measurement of a rank-1 density matrix $\hat{\rho}_y$ in *any* basis (see Appendix 6.A for details) to obtain the circuit shown in Figure 6.8, described by the equation:

$$\hat{\rho}_{X|y} \propto \text{tr}_{env}\left((\mathbb{I} \otimes \hat{u})P(\mathbb{I} \otimes \hat{u}^\dagger)\hat{U}\left(\hat{\rho}_X \otimes \hat{\rho}_{env}\right)\hat{U}^\dagger(\mathbb{I} \otimes \hat{u}^\dagger)^\dagger P^\dagger(\mathbb{I} \otimes \hat{u})^\dagger\right) \tag{6.23}$$

where $\hat{u}$ changes the basis of the environment variable to one in which the rank-1 density matrix encoding the observation $\hat{\rho}_Y$ is diagonalized to Λ – a matrix with all zeros except $\Lambda_{1,1} = 1$. The projection operator will be $P = (\mathbb{I} \otimes \Lambda)$, which means the terms $(\mathbb{I} \otimes \hat{u})P(\mathbb{I} \otimes \hat{u}^\dagger) = (\mathbb{I} \otimes \hat{u})(I \otimes \Lambda)(\mathbb{I} \otimes \hat{u}^\dagger) = (\mathbb{I} \otimes u\Lambda u^\dagger) = (\mathbb{I} \otimes \hat{\rho}_y)$, allowing us to rewrite Equation 6.23 as:

$$\hat{\rho}_{X|y} \propto \text{tr}_{env}\left((\mathbb{I} \otimes \hat{\rho}_y)\hat{U}\left(\hat{\rho}_X \otimes \hat{\rho}_{env}\right)\hat{U}^\dagger(\mathbb{I} \otimes \hat{\rho}_y)^\dagger\right) \tag{6.24}$$

Let us break this equation into two steps:

$$\begin{aligned}\hat{\rho}_{XY} &= \hat{U}\left(\hat{\rho}_X \otimes \hat{\rho}_{env}\right)\hat{U}^\dagger = \hat{U}W\hat{\rho}_X W^\dagger \hat{U}^\dagger \\ \hat{\rho}_{X|y} &= \frac{\text{tr}_{env}\left((\mathbb{I} \otimes \hat{\rho}_y)\hat{\rho}_{XY}(\mathbb{I} \otimes \hat{\rho}_y)^\dagger\right)}{\text{tr}\left(\text{tr}_{env}\left((\mathbb{I} \otimes \hat{\rho}_y)\hat{\rho}_{XY}(\mathbb{I} \otimes \hat{\rho}_y)^\dagger\right)\right)}\end{aligned} \tag{6.25}$$

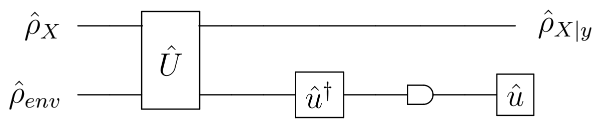


Figure 6.8: Quantum circuit to compute posterior $P(X|y)$

Now, we re-write the first expression in the language of HSEs. The quadratic matrix operation can be re-written as a linear operation by vectorizing the density matrix as we did in Section 6.5.2: $\mu_{XY} = ((\hat{U}W)^* \otimes (\hat{U}W))\mu_X$. But for $\mu_X \in \mathbb{R}^{n^2 \times 1}$, $W \in \mathbb{R}^{ns \times n}$, and $\hat{U} \in \mathbb{C}^{ns \times ns}$ this operation gives $\mu_{XY} \in \mathbb{R}^{n^2 s^2 \times 1}$, which we can reshape into an $n^2 \times s^2$ matrix $\mathcal{C}_{XY}^{\pi_X}$ (the superscript simply indicates the matrix was composed from a prior on X). We can then directly write $\mathcal{C}_{XY}^{\pi_X} = B \times_3 \mu_X$, where B is $((\hat{U}W)^* \otimes (\hat{U}W))$ reshaped into a three-mode tensor and $\times_3$ represents a tensor contraction along the third mode. But, just as we solved $A = \mathcal{C}_{YX}\mathcal{C}_{XX}^{-1}$ in Section 6.5.2 we can estimate $B = \mathcal{C}_{(XY)X}\mathcal{C}_{XX}^{-1} = \mathcal{C}_{XY|X}$ as a matrix and reshape into a 3-mode tensor, allowing us to re-write the first step in Equation 6.25 as:

$$\mathcal{C}_{XY}^{\pi_X} = \mathcal{C}_{(XY)X}\mathcal{C}_{XX}^{-1}\mu_X \tag{6.26}$$

$$= \mathcal{C}_{XY|X} \times_3 \mu_X \tag{6.27}$$

Now, to simplify the second step, observe that the numerator can be rewritten to get $\mu_{X|y} \propto \mathcal{C}_{XY}^{\pi_X}\left(\hat{\rho}_y^T \otimes \hat{\rho}_y\right)t$, where t is a vector of 1s and 0s that carries out the partial trace operation. But, for a rank 1 density matrix $\hat{\rho}_y$, this actually simplifies further:

$$\mu_{X|y} \propto \mathcal{C}_{XY}^{\pi_X}\rho_y \tag{6.28}$$

$$= \left(\mathcal{C}_{XY|X} \times_3 \mu_X\right)\rho_y \tag{6.29}$$

One way to renormalize $\mu_{X|y}$ is to compute $\left(\mathcal{C}_{XY|X} \times_3 \mu_X\right)\rho_y$ and reshape it back into a density matrix and divide by its trace. Alternatively, we can rewrite this operation using a vectorized identity matrix $\mathbb{I}$ that carries out the full trace in the denominator to renormalize as:

$$\mu_{X|y} = \frac{\left(\mathcal{C}_{XY|X} \times_3 \mu_X\right)\rho_y}{\mathbb{I}^T\left(\mathcal{C}_{XY|X} \times_3 \mu_X\right)\rho_y} \tag{6.30}$$

Finite Sample Kernel Estimate We kernelize these operations as follows (where $\phi(y) = \rho_y$):

$$\mu_{X|y} = \frac{\Upsilon \cdot \text{diag}\,(\alpha_X) \cdot \Phi^T \phi(y)}{\mathbb{I}^T \Upsilon \cdot \text{diag}\,(\alpha_X) \cdot \Phi^T \phi(y)} \tag{6.31}$$

$$= \frac{\sum_i \Upsilon_i \,(\alpha_X)_i \, k(y_i, y)}{\sum_j \,(\alpha_X)_j \, k(y_j, y)} \tag{6.32}$$

$$= \Upsilon \alpha_Y^{(X)} \tag{6.33}$$

where $(\alpha_Y^{(X)})_i = \frac{(\alpha_X)_i k(y_i,y)}{\sum_j (\alpha_X)_j k(y_j,y)}$, and $\mathbb{I}^T \Upsilon = \mathbb{1}^T$ since Υ contains vectorized density matrices, and $\mathbb{I}$ carries out the trace operation. As it happens, this method of estimating the conditional embedding $\mu_{X|y}$ is equivalent to performing Nadaraya-Watson kernel regression [84, 118] from the joint embedding to the kernel embedding. Note that this result only holds for the kernels satisfying Equation 4.22 in Wasserman [117]; importantly, the kernel function must only output positive numbers. One way to enforce this is by using a squared kernel; this is equivalent to a 2nd-order polynomial expansion of the features or computing the outer product of features. Our choice of feature map produces density matrices (as the outer product of features), so their inner product in Hilbert space is equivalent to computing the squared kernel, and this constraint is satisfied.

6.5.4 Quantum Bayes Rule as Kernel Bayes Rule

As we discussed at the end of Section 6.4, Figure 6.7c is an alternate way of generalizing Bayes rule for QGMs. But following the same approach of rewriting the quantum circuit in the language of Hilbert Space embeddings as in Section 6.5.2, we get exactly Kernel Bayes Rule [106]:

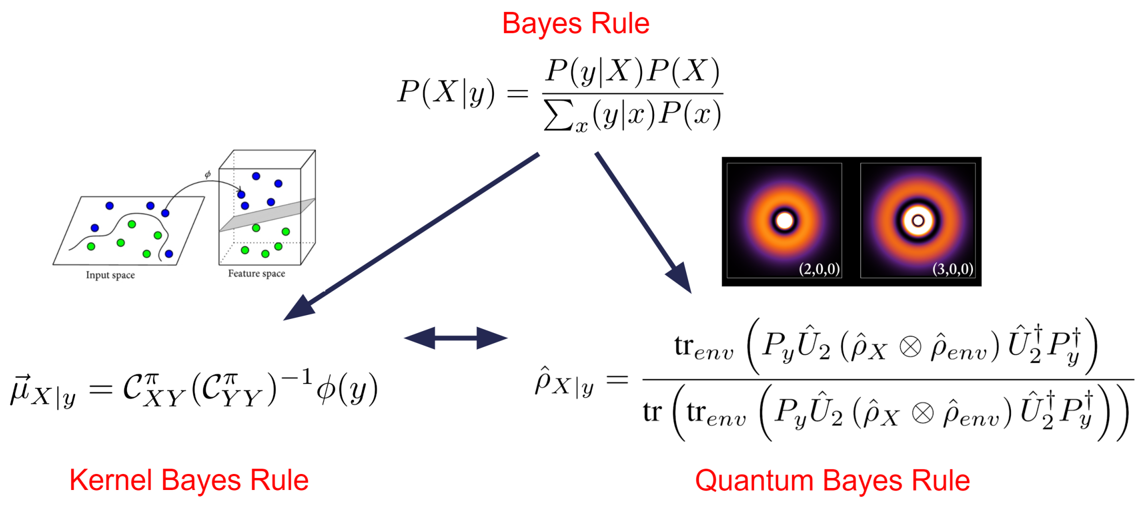
$$\mu_{X|y} = \mathcal{C}^{\pi}_{XY}(\mathcal{C}^{\pi}_{YY})^{-1}\phi(y) \tag{6.34}$$

The relationship between Bayes Rule, Quantum Bayes Rule, and Kernel Bayes Rule is illustrated in figure 6.9

What we have shown thus far As promised, we see that the two different but valid ways of generalizing Bayes rule for QGMs affects whether we condition according to Kernel Bayes Rule or Nadaraya-Watson kernel regression. However, we stress that conditioning according to Nadaraya-Watson is computationally much easier; the kernel Bayes rule given by Song et al. [106] using Gram matrices is written:

$$\hat{\mu}_{X|y} = \Upsilon D K_{yy}((DK_{yy})^2 + \lambda \mathbb{I})^{-1} D K_{:y} \tag{6.35}$$

where $D = \text{diag}((K_{xx} + \lambda \mathbb{I})^{-1} K_{xx} \alpha_X)$. Observe that this update requires squaring and inverting the Gram matrix K_{yy} – an expensive operation. By contrast, performing Bayesian update using Nadaraya-Watson as per Equation 6.31 is straightforward. This is one of the key insights of this paper; showing that Nadaraya-Watson kernel regression is an alternate, valid, but simpler way of generalizing Bayes rule to Hilbert space embeddings. We note that interpreting operations on QGMs as inference in Hilbert space is a special case; if the feature maps don't produce density



$$P(X|y) = \frac{P(y|X)P(X)}{\sum_x (y|x)P(x)}$$

$$\vec{\mu}_{X|y} = \mathcal{C}^{\pi}_{XY}(\mathcal{C}^{\pi}_{YY})^{-1}\phi(y)$$

$$\hat{\rho}_{X|y} = \frac{\text{tr}_{env}\left(P_y \hat{U}_2\left(\hat{\rho}_X \otimes \hat{\rho}_{env}\right)\hat{U}_2^{\dagger} P_y^{\dagger}\right)}{\text{tr}\left(\text{tr}_{env}\left(P_y \hat{U}_2\left(\hat{\rho}_X \otimes \hat{\rho}_{env}\right)\hat{U}_2^{\dagger} P_y^{\dagger}\right)\right)}$$

Figure 6.9: The relationship between Bayes Rule, Kernel Bayes Rule, and Quantum Bayes Rule. Quantum Bayes Rule and Kernel Bayes Rule both generalize bayes rule. Quantum Bayes Rule is obtained from Bayes by representing uncertainy using Quantum Mechanics via density matrices. Kernel Bayes Rule is obtained from Bayes Rule by embedding our probability distributions in an RKHS. Kernel Bayes Rule and Quantum Bayes rule are equivalent.

matrices, we can still perform inference in Hilbert space using the quantum/kernel sum rule, and Nadaraya-Watson/kernel Bayes rule, but lose the probabilistic interpretation of a quantum graphical model.

6.6 HSE-HQMMs

We now consider mapping data to vectorized density matrices and modeling the dynamics in Hilbert space using a specific quantum graphical model – hidden quantum Markov models (HQMMs). The quantum circuit for HQMMs is shown in Figure 6.10 [108].

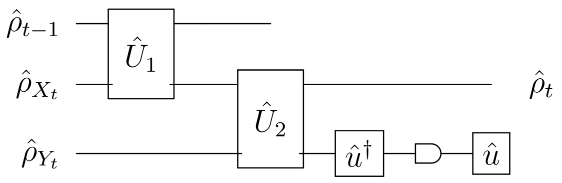


Figure 6.10: Quantum Circuit for HSE-HQMM

We use the outer product of random Fourier features (RFF) [92] (which produce a valid density matrix) to map data to a Hilbert space. $\hat{U}_1$ encodes transition dynamics, $\hat{U}_2$ encodes observation dynamics, and $\hat{\rho}_t$ is the density matrix after a transition update and conditioning on some observation. The transition and observation equations describing this circuit (with

$\hat{U}_I = \mathbb{I} \otimes \hat{u}^\dagger$) are:

$$\hat{\rho}'_t = \text{tr}_{\hat{\rho}_{t-1}} \left(\hat{U}_1 \left(\hat{\rho}_{t-1} \otimes \hat{\rho}_{X_t} \right) \hat{U}_1^\dagger \right) \tag{6.36}$$

$$\hat{\rho}_t \propto \text{tr}_{Y_t} \left(\hat{U}_I P_y \hat{U}_I^\dagger \hat{U}_2 \left(\hat{\rho}'_t \otimes \hat{\rho}_{Y_t} \right) \hat{U}_2^\dagger \hat{U}_I P_y^\dagger \hat{U}_I^\dagger \right) \tag{6.37}$$

As we saw in the previous section, we can rewrite these in the language of Hilbert Space embeddings:

$$\mu'_{x_t} = (\mathcal{C}_{x_t x_{t-1}} \mathcal{C}^{-1}_{x_{t-1} x_{t-1}}) \mu_{x_{t-1}} \tag{6.38}$$

$$\mu_t = \frac{(\mathcal{C}_{x_t y_t | x_t} \times_3 \mu'_{x_t}) \phi(y_t)}{\mathbb{I}^T (\mathcal{C}_{x_t y_t | x_t} \times_3 \mu'_{x_t}) \phi(y_t)} \tag{6.39}$$

And the kernelized version of these operations (where $\Upsilon = (\phi(x_1), \ldots, \phi(x_n))$ is (see appendix):

$$\alpha'_{x_t} = (K_{x_{t-1} x_{t-1}} + \lambda \mathbb{I})^{-1} K_{x_{t-1} x_t} \alpha_{x_{t-1}} \tag{6.40}$$

$$\alpha_{x_t} = \frac{\sum_i \Upsilon_i \left(\alpha'_{x_t} \right)_i k(y_i, y)}{\sum_j \left(\alpha'_{x_t} \right)_j k(y_j, y)} \tag{6.41}$$

It is also possible to combine the operations setting $\mathcal{C}_{x_t y_t | x_{t-1}} = \mathcal{C}_{x_t y_t | x_t} \mathcal{C}_{x_t x_{t-1}} \mathcal{C}^{-1}_{x_{t-1} x_{t-1}}$ to write our single update in Hilbert space:

$$\mu_{x_t} = \frac{(\mathcal{C}_{x_t y_t | x_{t-1}} \times_3 \mu_{x_{t-1}}) \phi(y_t)}{\mathbb{I}^T (C_{x_t y_t | x_{t-1}} \times_3 \mu_{x_{t-1}}) \phi(y_t)} \tag{6.42}$$

Making Predictions As discussed in Srinivasan et al. [108], conditioning on some discrete-valued observation y in the quantum model produces an unnormalized density matrix whose trace is the probability of observing y. However, in the case of continuous-valued observations, we can go further and treat this trace as the *unnormalized density* of the observation y_t, i.e., $f_Y(y_t) \propto \mathbb{I}^T (\mathcal{C}_{x_t y_t | x_{t-1}} \times_3 \mu'_{t-1}) \phi(y_t)$ – the equivalent operation in the language of quantum circuits is the trace of the unnormalized $\hat{\rho}_t$ shown in Figure 6.10. A benefit of building this model using the quantum formalism is that we can immediately see that this trace is bounded and lies in $[0, 1]$. It is also straightforward to see that a tighter bound for the unnormalized densities is given by the largest and smallest eigenvalues of the reduced density matrix $\hat{\rho}_{Y_t} = \text{tr}_{X_t}(\hat{\rho}_{X_t Y_t})$ where $\hat{\rho}_{X_t Y_t}$ is the joint density matrix after the application of $\hat{U}_2$.

To make a prediction, we sample from the convex hull of our training set, compute densities as described, and take the expectation to make a point prediction. This formalism is potentially powerful as it lets us maintain a whole distribution over the outputs (e.g. Figure 6.12), instead of just a point estimate for the next prediction as with LSTMs. A deeper investigation of the density estimation properties of our model would be an interesting direction for future work.

Learning HSE-HQMMs We estimate model parameters using 2-stage regression (2SR) [59], and refine them using back-propagation through time (BPTT). With this approach, the learned parameters are not guaranteed to satisfy the quantum constraints, and we handle this by projecting the state back to a valid density matrix at each time step. Details are given in Algorithm 3

```
Algorithm 3 Learning Algorithm using Two-Stage Regression for HSE-HQMMs
Input: Data as $\mathbf{y}_{1:T} = \mathbf{y}_1, ..., \mathbf{y}_T$
Output: Cross-covariance matrix $\mathcal{C}_{x_t y_t | x_{t-1}}$, can be reshaped into 3-mode tensor for prediction
1: Compute features of the past ($\mathbf{h}$), future ($\mathbf{f}$), shifted future ($\mathbf{s}$) from data (with window $k$):
       $\mathbf{h}_t \leftarrow h(\mathbf{y}_{t-k:t-1})$
       $\mathbf{f}_t \leftarrow f(\mathbf{y}_{t:t+k})$
       $\mathbf{s}_t \leftarrow f(\mathbf{y}_{t+1:t+k+1})$
2: Project data and features of past, future, shifted future into RKHS using random Fourier features of desired kernel (feature map $\phi(\cdot)$) to generate quantum systems:
       $|\mathbf{y}\rangle_t \leftarrow \phi_y(\mathbf{y}_t)$
       $|\mathbf{h}\rangle_t \leftarrow \phi_h(\mathbf{h}_t)$
       $|\mathbf{f}\rangle_t \leftarrow \phi_f(\mathbf{f}_t)$
       $|\mathbf{s}\rangle_t \leftarrow \phi_f(\mathbf{s}_t)$
   Construct density matrices in the RKHS and vectorize them:
       $\rho_t^{(y)} \leftarrow \text{vec}\left(|\mathbf{y}\rangle_t \langle \mathbf{y}|_t\right)$
       $\rho_t^{(h)} \leftarrow \text{vec}\left(|\mathbf{h}\rangle_t \langle \mathbf{h}|_t\right)$
       $\rho_t^{(f)} \leftarrow \text{vec}\left(|\mathbf{f}\rangle_t \langle \mathbf{f}|_t\right)$
       $\rho_t^{(s)} \leftarrow \text{vec}\left(|\mathbf{s}\rangle_t \langle \mathbf{s}|_t\right)$
3: Compose matrices whose columns are the vectorized density matrices, for each available time-step (accounting for window size $k$), denoted $\Phi_y$, $\Upsilon_h$, $\Upsilon_f$, and $\Upsilon_s$ respectively.
4: Obtain extended future via tensor product $\rho_t^{(s,y)} \leftarrow \rho_t^{(s)} \otimes \rho_t^{(y)}$ and collect into matrix $\Gamma_{s,y}$.
5: Perform Stage 1 regression
       $\mathcal{C}_{f|h} \leftarrow \Upsilon_f \Upsilon_h^\dagger \left(\Upsilon_h \Upsilon_h^\dagger + \lambda\right)^{-1}$
       $\mathcal{C}_{s,y|h} \leftarrow \Gamma_{s,y} \Upsilon_h^\dagger \left(\Upsilon_h \Upsilon_h^\dagger + \lambda\right)^{-1}$
6: Use operators from stage 1 regression to obtain denoised predictive quantum states:
       $\tilde{\Upsilon}_{f|h} \leftarrow \mathcal{C}_{f|h} \Upsilon_h$
       $\tilde{\Gamma}_{s,y|h} \leftarrow \mathcal{C}_{s,y|h} \Upsilon_h$
7: Perform Stage 2 regression to obtain model parameters
       $\mathcal{C}_{x_t y_t | x_{t-1}} \leftarrow \tilde{\Gamma}_{s,y|h} \tilde{\Upsilon}_{f|h}^\dagger \left(\tilde{\Upsilon}_{f|h} \tilde{\Upsilon}_{f|h}^\dagger + \lambda\right)^{-1}$
```

6.7 Connections with PSRNNs

As a reminder, the PSRNN update equation is:

$$\omega_t = \frac{W \times_3 \omega_{t-1} \times_2 \phi(y_t)}{||W \times_3 \omega_{t-1} \times_2 \phi(y_t)||_F} \tag{6.43}$$

where W is a three mode tensor corresponding to the cross-covariance between observations and the state at time t conditioned on the state at time $t-1$, and ω is a factorization of a p.s.d state matrix $\mu_t = \omega\omega^T$ (so renormalizing ω by Frobenius norm is equivalent to renormalizing μ by its trace). There is a clear connection between PSRNNs and the HSE-HQMMs; this matrix μ_t is what we vectorize to use as our state μ_t in HSE-HQMMs, and both HSE-HQMMs and PSRNNs are parameterized (in the primal space using RFFs) in terms of a three-mode tensor (W for PSRNNs and ($\mathcal{C}_{x_t y_t | x_{t-1}}$ for HSE-HQMMs). We also note that while PSRNNs modified kernel Bayes rule (from Equation 6.34) *heuristically*, we have shown that this modification can be interpreted as a generalization of Bayes rule for QGMs or Nadaraya-Watson kernel regression. One key difference between these approaches is that we directly use states in Hilbert space to estimate the probability density of observations; in other words HSE-HQMMs are a generative model. By contrast PSRNNs are a discriminative model which rely on an additional ad-hoc mapping from states to observations.

6.8 Experiments

We use the following datasets in our experiments:

- **Penn Tree Bank (PTB)** [80]. We train a character-prediction model with a train/test split of 120780/124774 characters due to hardware limitations.
- **Swimmer** Simulated swimmer robot from OpenAI gym[1]. We collect 25 trajectories from a robot that is trained to swim forward (via the cross entropy with a linear policy) with a 20/5 train/test split. There are 5 features at each time step: the angle of the robots nose, together with the 2D angles for each of it's joints.
- **Mocap** Human Motion Capture Dataset. We collect 48 skeletal tracks from three human subjects with a 40/8 train/test split. There are 22 total features at each time step: the 3D positions of the skeletal parts (e.g., upper back, thorax, clavicle).

We compare the performance of three models: HSE-HQMMs, PSRNNs, and LSTMs. We initialize PSRNNs and HSE-HQMMs using Two-Stage Regression (2SR) [44] and LSTMs using Xavier Initialization and refine all three models using Back Propagation Through Time (BPTT). We optimize and evaluate all models on Swimmer and Mocap with respect to the Mean Squared Error (MSE) using 10 step predictions as is conventional in the robotics community. This means that to evaluate the model we perform recursive filtering on the test set to produce states, then use these states to make predictions about observations 10 steps in the future. We optimize all models on PTB with respect to Perplexity (Cross Entropy) using 1 step predictions, as is conventional in the NLP community. As we can see in Figure 6.11, HSE-HQMMs outperform both PSRNNs

[1]`https://gym.openai.com/`

and LSTMs on the swimmer dataset, and achieve comparable performance to the best alternative on Mocap and PTB. Hyperparameters and other experimental details can be found in Appendix 6.C. One important point to note is that LSTMs and PSRNNs are both point-prediction models,

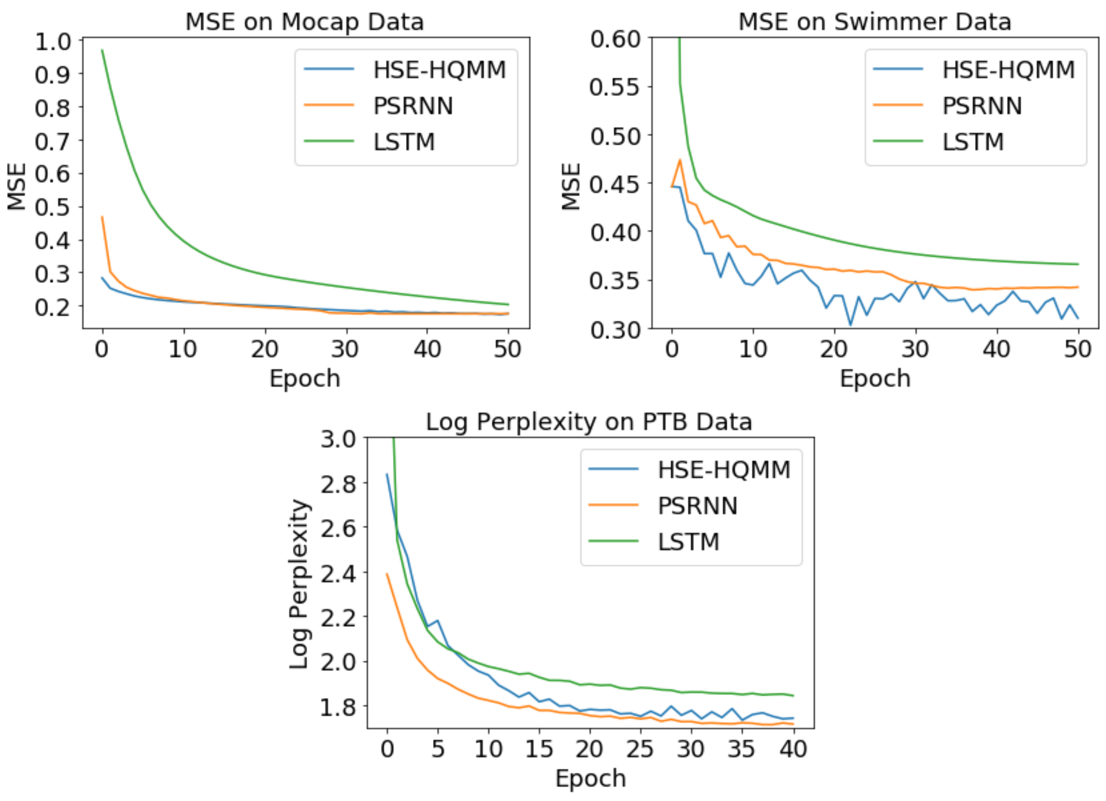


Figure 6.11: Performance of HSE-HQMM, PSRNN, and LSTM on Mocap, Swimmer, and PTB

while HSE-HQMMs are generative models which maintain a full probability distribution over all possible observations. This means that this experiment actually disadvantages HSE-HQMMs as we are asking them to perform a task which other two models are optimized to solve, without taking into account the additional benefits offer by HSE-HQMMs (i.e. the ability to predict multiple models, relative confidence of solution, etc.). This is particularly the case on PTB, where we are building a continuous probability distribution over a discrete observation space when using HSE-HQMMs. Therefore these results are extremely promising, as they show that we can build fully principled models which also offer good point predictions.

Visualizing Probability Densities As mentioned previously, HSE-HQMMs can maintain a probability density function over future observations, and we visualize this for a model trained on the Mocap dataset in Figure 6.12. We take the 22 dimensional joint density and marginalize it to produce three marginal distributions, each over a single feature. We plot the resulting marginal distributions over time using a heatmap, and superimpose the ground-truth and model predictions. We observe that BPTT (second row) improves the marginal distribution. Another interesting observation, from the the last $\sim$30 timesteps of the marginal distribution in the top-left image, is that our model is able to produce a bi-modal distribution with probability mass at both $y_i = 1.5$ and $y_i = -0.5$, without making any parametric assumptions. This kind of information is difficult to obtain using a discriminative model such as a LSTM or PSRNN.

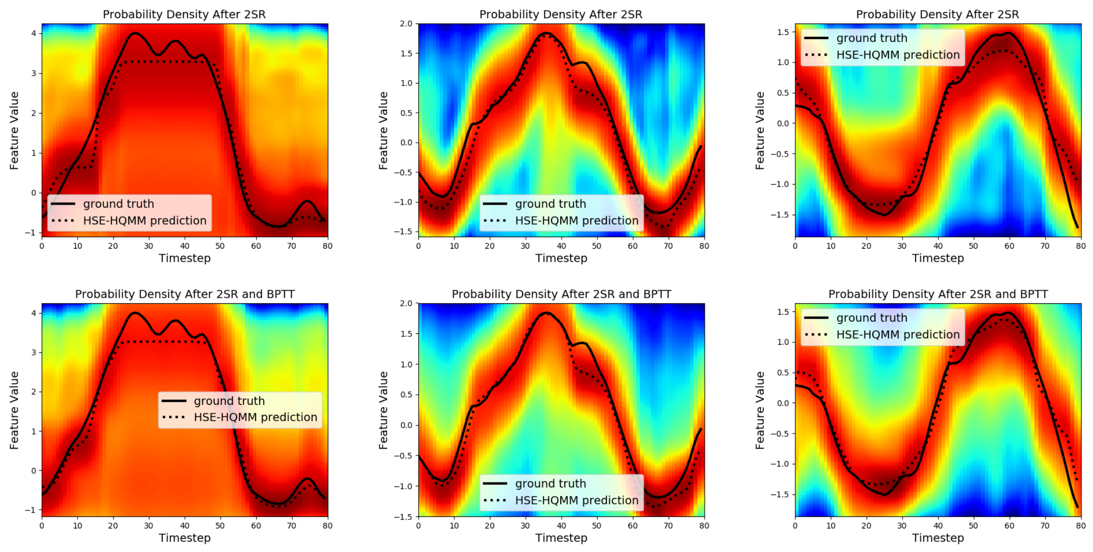


Figure 6.12: Heatmap Visualizing the Probability Densities generated by our HSE-HQMM model. Red indicates high probability, blue indicates low probability, x-axis corresponds to time, y-axis corresponds to the feature value. Each column corresponds to the predicted marginal distribution of a single feature changing with time. The first row is the probability distribution after 2SR initialization, the second row is the probability distribution after the model in row 1 has been refined via BPTT.

6.9 Related Work

Various formulations of Quantum Graphical Models (QGMs) have been proposed by researchers in physics and machine learning [77, 108, 122] as a way of generalizing probabilistic inference on graphical models by adopting quantum mechanics' formalism for reasoning about uncertainty. While Srinivasan et al. [108] focused on modeling dynamical systems with Hidden Quantum Markov Models (HQMMs) [83], they also describe the basic operations on general quantum graphical models, which generalize Bayesian reasoning within a framework consistent with quantum mechanical principles. Inference using Hilbert space embeddings (HSE) is also a generalization of Bayesian reasoning, where data is mapped to a Hilbert space in which kernel sum, chain, and Bayes rules can be used [103, 105, 106]. These methods can model dynamical systems such as HSE-HMMs [104], HSE-PSRs [26], and PSRNNs [44]. [98] present related but orthogonal work connecting kernels, Hilbert spaces, and quantum computing.

In the work most closely related to ours Srinivasan et al. [108] present a maximum-likelihood learning algorithm to estimate the parameters of a HQMM from data. However, it is very limited in its scope; the algorithm is slow and doesn't scale for large datasets. In this paper, we leverage connections to HSEs, kernel methods, and RFFs to achieve a more practical and scalable learning algorithm for these models. However, one difference to note is that the algorithm presented by Srinivasan et al. [108] guaranteed that the learned parameters would produce valid quantum operators, whereas our algorithm only approximately produces valid quantum operators; we will need to project the updated state back to the nearest quantum state to ensure that we are tracking a valid quantum system.

6.10 Conclusions

We explored the connections between QGMs and HSEs, and showed that the sum rule and Bayes rule in QGMs is equivalent to kernel sum rule and a special case of Nadaraya-Watson kernel regression. We proposed HSE-HQMMs to model dynamics, and showed experimentally that these models are competitive with LSTMs and PSRNNs on making point predictions, while also being a nonparametric method for maintaining a probability distribution over continuous-valued features. Looking forward, we note that our experiments only consider real kernels/features, so we are not utilizing the full complex Hilbert space; it would be interesting to investigate whether incorporating complex numbers improves our model. Additionally, by estimating parameters using least-squares, the parameters only approximately adhere to quantum constraints. The final model also bears strong resemblance to PSRNNs [44]. It would be interesting to investigate both what happens if we are stricter about enforcing quantum constraints, and if we give the model greater freedom to drift from the quantum constraints. Finally, the density estimation properties of the model are also an avenue for future exploration.

6.A Modifying the Quantum Circuit for Bayes rule

In Figure 6.7b, an assumption in the way the observation update is carried out is that the measurement on $\hat{\rho}_Y$ is carried out in the same basis that the unitary operator $\hat{U}$ is performed. When using a HQMM to explicitly model some number of discrete observations, this is fine, since each observation is a basis state, and we always measure in this basis. However, in the general case, we may wish to account for measuring $\hat{\rho}_Y$ in *any* basis. To do this, we get the eigendecomposition of $\hat{\rho}_Y$, so that $u\Lambda u^\dagger = \hat{\rho}_Y$ (the eigenvectors of $\hat{\rho}_Y$ will form a unitary matrix). We only need to rotate $\hat{\rho}_Y$ into the 'correct' basis before measurement; we can leave $\hat{\rho}_{X|y}$ unchanged. This new, general observation is implemented by the circuit in 6.8. The final $\hat{u}$ is not strictly necessary; since it has no effect on the first particle, which stores $\hat{\rho}_{X|y}$, which is ultimately what we're interested in. However, including it allows us to simplify the update rule.

6.B Kernelizing HSE-HQMMs

Here we derive the kernel embedding update for HSE-HQMMs as given in Equation 6.41. The observation update follows directly from the Nadaraya-Watson kernel regression given in Equation 6.31. The transition update comes from a recursive application of the quantum sum rule given in Equation 6.21. Let $\Upsilon = (\phi(x_{t-1}^{(1)}), \phi(x_{t-1}^{(2)}), \ldots, \phi(x_{t-1}^{(n)}))$ and $\Phi = (\phi(x_t^{(1)}), \phi(x_t^{(2)}), \ldots, \phi(x_t^{(n)}))$. Starting with $\hat{\mu}_{t-1} = \Upsilon\alpha_{t-1}$ using the kernel sum rule recursively twice with $\mathcal{C}_{X_t|X_{t-1}} = \Phi(K_{x_{t-1}x_{t-1}} + \lambda\mathbb{I})^{-1}\Upsilon^\dagger$, we get:

$$\begin{aligned} \hat{\mu}_{t+1} &= \mathcal{C}_{X_t|X_{t-1}}\mathcal{C}_{X_t|X_{t-1}}\hat{\mu}_{t-1} \\ \Rightarrow \Phi\alpha_{t+1} &= \Phi(K_{x_{t-1}x_{t-1}} + \lambda\mathbb{I})^{-1}\Upsilon^\dagger\Phi(K_{x_{t-1}x_{t-1}} + \lambda\mathbb{I})^{-1}\Upsilon^\dagger\Upsilon\alpha_{t-1} \\ \Rightarrow \alpha_{t+1} &= (K_{x_{t-1}x_{t-1}} + \lambda\mathbb{I})^{-1}K_{x_{t-1}x_t}(K_{x_{t-1}x_{t-1}} + \lambda\mathbb{I})^{-1}K_{x_{t-1}x_{t-1}}\alpha_{t-1} \end{aligned} \tag{6.44}$$

If we break apart this two-step update to update for a single time-step, starting from α_{t-1} we can write $\alpha'_t = (K_{x_{t-1}x_{t-1}} + \lambda\mathbb{I})^{-1}K_{x_{t-1}x_t}\alpha_{t-1}$.

6.C Experimental Details

6.C.1 State Update

Given HSE-HQMM model parameters consisting of the 3-mode tensor $\mathcal{C}_{x_t y_t | x_{t-1}}$, current state μ_{x_t} and observation in Hilbert space $\phi(y_t)$ we perform filtering to update the state as follows:

$$\mu_{x_{t+1}} = \left(\mathcal{C}_{x_t y_t | x_{t-1}} \times_3 \mu_{x_t}\right) \phi(y_t) \tag{6.45}$$

where $A \times_c b$ corresponds to tensor contraction of tensor A with vector b along mode c. In our case this means performing tensor contraction of $\mathcal{C}_{x_t y_t | x_{t-1}}$ with the current state and observation along the appropriate modes.

6.C.2 Prediction

Given HSE-HQMM model parameters consisting of the 3-mode tensor $\mathcal{C}_{x_t y_t | x_{t-1}}$ and current state μ_{x_t} we can estimate the probability density of an observation y_t (up to normalization) as follows:

$$f_Y(y_t) = \mathbb{I}^T((\mathcal{C}_{x_t y_t | x_{t-1}} \times_3 \mu_{x_t})\phi(y_t) \tag{6.46}$$

If we want to make a point prediction from our model (in order to calculate the mean squared error, etc) we can use the mean $\mathbb{E}[y_t | x_t]$. In practice we approximate this quantity using a sample $y_{1:n}$:

$$\mathbb{E}[y_t | x_t] = \frac{1}{n} \sum_{i=1}^{n} y_i f_Y(y_i) \tag{6.47}$$

6.C.3 Pure State HSE-HQMMs

In our experiments we use a HSE-HQMM consisting of a single pure state, as opposed to the full density matrix formalism. Using a single pure state is a special case of the general (mixed state) HSE-HQMM, and allows the model to be efficiently implemented. The learning algorithm modified for a pure state is shown in Algorithm 4.

6.C.4 Parameter Values

Hyperparameter settings can be found in table 6.1. Some additional details: In two-stage regression we use history (future) features consisting of the past (next) 10 observations as one-hot vectors concatenated together. We use a ridge-regression parameter of 0.05 (this is consistent with the values suggested in Boots et al. [27], Downey et al. [44]). The kernel width is set to the median pairwise (Euclidean) distance between neighboring data points. We use a fixed learning rate of 0.1 for BPTT with a BPTT horizon of 20.

Algorithm 4 Learning Algorithm using Two-Stage Regression for Pure State HSE-HQMMs

```
Input: Data as $\mathbf{y}_{1:T} = \mathbf{y}_1, ..., \mathbf{y}_T$
Output: Cross-covariance matrix $\mathcal{C}_{x_t y_t | x_{t-1}}$, can be reshaped to 3-mode tensor for prediction
1: Compute features of the past ($\mathbf{h}$), future ($\mathbf{f}$), shifted future ($\mathbf{s}$) from data (with window $k$):
       $\mathbf{h}_t = h(\mathbf{y}_{t-k:t-1}) \qquad \mathbf{f}_t = f(\mathbf{y}_{t:t+k}) \qquad \mathbf{s}_t = f(\mathbf{y}_{t+1:t+k+1})$
2: Project data and features of past, future, shifted future into RKHS using random features of desired kernel (feature map $\phi(\cdot)$) to generate quantum systems:
       $|\mathbf{y}\rangle_t \leftarrow \phi_y(\mathbf{y}_t) \quad |\mathbf{h}\rangle_t \leftarrow \phi_h(\mathbf{h}_t) \quad |\mathbf{f}\rangle_t \leftarrow \phi_f(\mathbf{f}_t) \quad |\mathbf{s}\rangle_t \leftarrow \phi_f(\mathbf{s}_t)$
3: Compose matrices whose columns are $|\mathbf{y}\rangle_t$, $|\mathbf{h}\rangle_t$, $|\mathbf{f}\rangle_t$, $|\mathbf{s}\rangle_t$ for each available time-step (accounting for window size $k$), denoted $\Phi_y$, $\Upsilon_h$, $\Upsilon_f$, and $\Upsilon_s$ respectively.
4: Obtain extended future via tensor product $|\mathbf{s}, \mathbf{y}\rangle_t \leftarrow |\mathbf{s}\rangle_t \otimes |\mathbf{y}\rangle_t$ and collect into matrix $\Gamma_{s,y}$.
5: **Perform Stage 1 regression**
       $\mathcal{C}_{f|h} \leftarrow \Upsilon_f \Upsilon_h^\dagger \left(\Upsilon_h \Upsilon_h^\dagger + \lambda\right)^{-1}$
       $\mathcal{C}_{s,y|h} \leftarrow \Gamma_{s,y} \Upsilon_h^\dagger \left(\Upsilon_h \Upsilon_h^\dagger + \lambda\right)^{-1}$
6: Use operators from stage 1 regression to obtain denoised predictive quantum states:
       $\tilde{\Upsilon}_{f|h} \leftarrow \mathcal{C}_{f|h} \Upsilon_h$
       $\tilde{\Gamma}_{s,y|h} \leftarrow \mathcal{C}_{s,y|h} \Upsilon_h$
7: **Perform Stage 2 regression to obtain model parameters**
       $\mathcal{C}_{x_t y_t | x_{t-1}} \leftarrow \tilde{\Gamma}_{s,y|h} \tilde{\Upsilon}_{f|h}^\dagger \left(\tilde{\Upsilon}_{f|h} \tilde{\Upsilon}_{f|h}^\dagger + \lambda\right)^{-1}$
```

Parameter	Value
Kernel	Gaussian
Kernel Width	Median Neighboring Pairwise Distance
Number of Random Fourier Features	1000
Prediction Horizon	10
Batch Size	20
State Size	20
Max Gradient Norm	0.25
Number of Layers	1
Ridge Regression Regularizer	0.05
Learning Rate	0.1
Learning algorithm	Stochastic Gradient Descent (SGD)
Number of Epochs	50
BPTT Horizon	20

Table 6.1: Hyperparameter values used in experiments

Part IV

Scalable Implementations

Chapter 7

Sketching for Predictive State Methods

In Chapter 5 and Chapter 6 we introduced PSRNNs and HSE-HQMMs, two models which unify Bayes Filters with Recurrent Neural Networks. We showed that these models possess many attractive theoretical properties, and also perform well on a variety of real world data sets.

Unfortunately both models as currently described possess a significant weakness: They do not scale well to large state spaces. To see why note that both PSRNNs and HSE-HQMMs are parameterized using three-mode tensors, hence they require $O(d^3)$ space, where d is the size of the state vector. In Chapter Section 5.4 we suggested a partial solution to this problem by using tensor CP decomposition to factorize the 3-mode parameter tensors into low rank components, reducing the size of the model from $O(d^3)$ to $O(Kd)$. Unfortunately this approach is only a partial solution, as such models cannot be learned directly using 2SR in its current form. At present the only way to obtain a factorized model is to first learn the full tensor, then subsequently perform a costly tensor decomposition. In other words the final model we obtain is small, but the cost of obtaining it actually increases. If factorized models are to be of practical value for solving large scale problems, we require a new formulation of 2SR which directly learns a factorized model from data *without* being forced to compute the full model as an intermediate quantity.

We propose a solution to this problem using the recently developed concept of tensor sketching. A tensor sketch maintains a “sketch” or summary of a tensor, similar to how we might compress an image. The key difference between tensor sketch and alternative compression techniques (such as random projections) is that many tensor operations can be performed directly on the sketched tensors without reconstructing the full tensor. Tensor sketches are particularly useful in the context of moment tensors (such as the tensors used in PSMs). In this setting we can compute a sketch of the moment tensor as a simple function of the individual data points. We can then perform operations such as tensor decomposition on the sketched moment tensor to recover the tensor decomposition.

We present Sketched 2SR, a new formulation of 2SR which uses tensor sketches, combined with tensor power iteration, to directly learn a factorized PSRNN from data. Sketched 2SR requires only $O(d)$ space and $O(d^2)$ time, compared with $2SR$ which requires $O(d^3)$ space and $O(d^4)$ time.

For the remainder of this section we will focus on PSRNNs, in order to have a concrete example to use for developing a sketched version of 2SR. Note that while we focus on PSRNNs, this work is generally applicable to the entire class of predictive state models.

7.1 Tensor Sketch

Tensor sketching [90] is a compression algorithm specifically designed for tensors. Tensor sketching allows us to convert large tensors into compact summaries or *sketches* which preserve certain information.

The tensor sketch is an extension of the count sketch algorithm [42] for compressing vectors. To hash a vector of dimensionality d, count sketch utilizes two pairwise independent hash functions $h : [d] \mapsto [b]$ and $\zeta : [d] \mapsto \{\pm 1\}$. A d dimensional vector x is hashed into a b dimensional vector s_x such that

$$s_x[i] = \sum_{j:h(j)=i} \zeta(j)x_j. \tag{7.1}$$

Thus, the sketch of a vector can be computed in $O(d)$ time. The above construction leads to high-variance reconstructions on its own, so we repeat it, computing multiple hash functions and using the median of the estimated inner products to increase robustness.

We can approximately reconstruct elements of a sketched vector by taking an inner product with the corresponding indicator vector:

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

We note that count sketching is a linear operation, an observation which will be important later. We also note that count sketching is a lossy compression algorithm – we cannot recover the original vector, we can only recover an approximation.

Tensor sketch extends the count sketch from vectors to arbitrary tensors as follows. To sketch a tensor $A \in \mathbb{R}^{d_1,\dots,d_n}$ into a vector of length b we select hash functions $\{h_1, \dots, h_n\}$ where $h_i : [d_i] \rightarrow \mathbf{Z}$ and $\zeta_1, \dots, \zeta_n$ where $\zeta_i : [d_i] \rightarrow \{\pm 1\}$. Using these we define two new hash functions $h()$ and $\zeta()$, where $h()$ maps locations in the tensor to locations in the sketch, and $\zeta()$ maps locations in the tensor to $\{\pm 1\}$:

$$h(i_1, \dots, i_n) = \left(\sum_j h_j(i_j) \right) \mod b \tag{7.2}$$

$$\zeta(i_1, \dots, i_n) = \prod \zeta_j(i_j) \tag{7.3}$$

Therefore the sketch of tensor $A \in \mathbb{R}^{d_1,\dots,d_n}$ is defined as:

$$s_A[j] = \sum_{i_1,\dots,i_n : h(i_1,\dots,i_n)=j} \zeta(i_1, \dots, i_n) A_{i_1,\dots,i_n}. \tag{7.4}$$

The main advantage of constructing tensor sketches in this particular way is that it allows for efficient sketching of rank-1 tensors. If tensor as is rank-1 it can be factorized into vectors $a_i, \dots, a_n$ such that:

$$A = a_i \otimes \dots \otimes a_n$$

Suppose we have access to the factors $a_1, ..., a_n$. Then we can construct the sketch of A using $a_1, ..., a_n$ by convolving the sketches of its factors:

$$s_A = s_{a_1} * ... * s_{a_n} \tag{7.5}$$
$$= \mathcal{F}^{-1}\left(\mathcal{F}(s_{a_1}) \circ ... \circ \mathcal{F}(s_{a_n})\right) \tag{7.6}$$

where $\mathcal{F}$ denotes the fourier transform. This computation can be performend in $O(b \log(b)$ time, as opposed to sketching the original tensor which requires $O(\prod d_i)$ time. Furthermore because sketching is a linear operation we can use this same technique to efficiently sketch factorized tensors, which are simply sums of rank-1 tensors.

The key advantage of tensor sketch over alternative compression algorithms is that tensor sketch preserves the inner product between tensors in expectation. In other words:

$$\mathbb{E}[\langle s_x, s_y \rangle] = \langle x, y \rangle, \quad \mathrm{Var}[\langle s_x, s_y \rangle] = O(\frac{|\langle s_x, s_y \rangle|^2}{b}). \tag{7.7}$$

For a single copy; with n copies the variance scales as:

$$\mathbb{E}[\langle s_x, s_y \rangle] = \langle x, y \rangle, \quad \mathrm{Var}[\langle s_x, s_y \rangle] = O(\frac{|\langle s_x, s_y \rangle|^2}{nb}).$$

This means that many tensor operations, including tensor contractions, are also approximately preserved:

$$A(a_1, ..., a_n) \approx \langle s_A, s_{a_1} * ... * s_{a_n}, s \rangle_{T(I,b,c)} \approx \langle s_t, s_{e_i} * s_b * s_c \rangle \tag{7.8}$$

Furthermore for a 3-mode tensor we can sketch partial contraction:

$$s_{A(I,b,c)} \approx \mathcal{F}^{-1}\left(\mathcal{F}(s_A) \circ \bar{\mathcal{F}(s_b)} \circ \bar{\mathcal{F}(s_c)}\right) \tag{7.9}$$

7.2 Sketched Two-Stage Regression

As a reminder (one layer) PSRNNs are parameterized by a single 3-mode tensor W which is used to update the state q_t:

$$q_{t+1} = W(q_t, o_t, I)/||W(q_t, o_t, I)|| \tag{7.10}$$

Suppose we are given a training dataset of tuples of the form (h_t, o_t, ψ_t) for $t = 1, .., T$. Two-stage regression for learning PSRNNs (using ordinary least squares for stage 1B) first uses ridge regression to compute a matrix W_1 such that:

$$\hat{\psi}_t = \mathbb{E}[\psi_t \mid h_t] \tag{7.11}$$
$$= W_1 h_t \tag{7.12}$$

We then use $\hat{\psi}_t$ to estimate W:

$$W = M_3((M_2 + \lambda I)^{-1}, I, I) \tag{7.13}$$

where:

$$M_3 = \sum_{t=1}^{T} \hat{\psi}_t \otimes o_t \otimes \hat{\psi}_{t+1} \tag{7.14}$$

$$M_2 = \sum_{t=1}^{T} \hat{\psi}_t \times \hat{\psi}_t \tag{7.15}$$

Equation 7.13 can be understood as reshaping M_3 into a matrix with $\dim \hat{\psi}_t$ rows and $\dim o_t \times \dim \hat{\psi}_{t+1}$ columns, then pre-multiplying it by $(M_2 + \lambda I)^{-1}$. Given T training examples, computing M_3 requires $O(Tp^3)$ where p is the dimension of features. We need $O(p^4)$ time to compute W via 7.13. We also require $O(p^3)$ space to store both W and M_3. We now show how to compute a sketch of W using only sketched quantities. We can then use that sketch to reconstruct a CP decomposition of W. This will be faster; analysis below.

From (7.14) and (7.13) we can rewrite $\mathbf{W}$ as follows

$$\mathbf{W} = \sum_t ((M_2 + \lambda I)^{-1} W_1 h_t) \otimes o_t \otimes \psi_{t+1} \tag{7.16}$$

We note that each term in Equation 7.16 is either a sum of rank-1 tensors, or a tensor contraction. This means we can efficiently compute a sketch of W by first sketching the components, then combining them using Equation 7.6. Computing a sketch of W in this way requires three passes through the training data, as shown in Algorithm 5. For simplicity, the algorithm assumes a single sketch is used but can easily be extended to $\mathfrak{B}$ sketches.

Algorithm 5 Sketching the parameter tensor $\mathbf{W}$

```
Input:
  • Training examples $\{(h_t, \phi_t^o, \psi_t)\}_{t=1}^T$
  • Hash functions ($\{h_m, \zeta_m\}_{m=1}^3$)
  • S1 regularization $\lambda_1$
  • S2 regularization $\lambda_2$
Output: Parameter tensor sketch $s_W \in R^b$.
1: // Stage 1 Regression
2: $W_1 \leftarrow (\sum_t \psi_t \otimes h_t)(\sum_t h_t \otimes h_t + \lambda_1 I)$
3: // Stage 2 Regression
4: $M_2 \leftarrow \sum_{t=1}^T (W_1 h_t) \otimes (W_1 h_t)$
5: $s_W \leftarrow \mathbf{0}$
6: for $t = 1$ to $T - 1$ do
7:     $a \leftarrow (M_2 + \lambda_2 I)^{-1} W_1 h_t, b \leftarrow o_t, c \leftarrow \psi t + 1$
8:     $s_\mathbf{W} \leftarrow s_\mathbf{W} + \mathcal{F}^{-1}(\mathcal{F}(s_a^{(1)}) \circ \mathcal{F}(s_b^{(2)}) \circ \mathcal{F}(s_c^{(3)}))$
9: end for
```

The following proposition shows the time and space complexities of Algorithm 5.

Proposition 32. *Assume that all feature vectors are of dimension p and that we use $\mathfrak{B}$ sketches of size b each. Then, for T training examples, Algorithm 5 has a time complexity of $O(p^3 + T[p^2 + \mathfrak{B}p + \mathfrak{B}b \log b])$ and a space complexity of $O(p^2 + \mathfrak{B}b)$.*

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 [$O(\mathfrak{B}b \log b)$].

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

Compared to the $O(d^4 + Td^3)$ time and $O(d^3)$ memory when learning $\mathbf{W}$ directly, Algorithm 5 can result in significant gains for large values of d.

7.2.1 Tensor Sketch as PSRNN Parameters

Having shown that it is possible to efficiently obtain a sketch of $\mathbf{W}$ from data, we now discuss how to use it to obtain a functional model. One obvious approach is to maintain a sketch of *both* the state q_t and the parameter tensor W and update the state using sketched tensor contraction. We note that it is still possible to perform BPTT when using sketched quantities as sketching is a linear (and hence differentiable) operation. However, initial experiments have shown that this approach results in poor PSRNN performance. As we demonstrate in 7.3, this can be attributed to the fact that, while tensor sketches provide a decent approximation when used inside of a tensor decomposition algorithm, the approximation quality for general tensor contraction can be very poor. We conjecture that tensor power iteration tends to correct errors because of contractive iteration. We propose an alternative approach, where we use a sketch of W to compute a factorized PSRNN which does not use sketches. As a reminder, if we factorize $W = \sum_i a_i \otimes b_i \otimes c_i$ we can write the PSRNN update equation as:

$$q_{t+1} = \frac{C^\top (Aq_t \circ Bo_t)}{||C^\top (Aq_t \circ Bo_t)||} \tag{7.17}$$

which is referred to as a factorized PSRNN. Our approach involves using the sketch of W to compute an approximate CP decomposition of the full $\mathbf{W}$. To do this we use a modification of alternating least squares to tensor-decompose W directly from its sketch.

Experimental results suggest that this approach is not subject to the limitations of the naive approach and can be used to obtain a practical model. We currently lack a rigorous theoretical explanation for this result, however we conjecture it is due to alternating least squares acting as a denoising procedure.

7.2.2 Hybrid ALS with Deflation

Wang et al. [116] proposed a CP decomposition method based on alternating least squares (Algorithm 6). The core operation in alternating least squares is the tensor product $\mathbf{T}(I, b, c)$, which can be carried out with sketches using (7.9).

Proposition 33. *For K components and L iterations, Algorithm 6 has $O(LK\mathfrak{B}(d\log\mathfrak{B} + b\log b) + L(dK^2 + K^3))$ time complexity and $O(b\mathfrak{B} + Kd + K^2)$ space complexity.*

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

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

Algorithm 6 scales linearly (instead of cubically) in the dimension d. [116] demonstrated that Algorithm 6 can recover the top few components of a 1000 dimensional tensor. However, we have observed that Algorithm 6 often has trouble when simultaneously considering a large number of components, as we demonstrate in Section 7.3.2. Also, for a large number of components (which may be required to accurately approximate $\mathbf{W}$), the fact that the cost is super-linear in K can become problematic.

For this reason we opt to use a deflation based approach. We use Algorithm 6 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 7. Recovering a single component means the intermediate quantities $A^\top A$, $B^\top B$ and $C^\top C$ in Algorithm 6 are simply scalars. Not only does this approach produce better decomposition, as we demonstrate in Section 7.3.2, but it is also better in terms of time and space complexity, as we show below.

Proposition 34. *For K components and L iterations per component, Algorithm 7 has $O(LK\mathfrak{B}(d\log b + b\log b) + KLd)$ time complexity and $O(b\mathfrak{B} + Kd)$ space complexity.*

Proof. The result is derived by substituting $K = 1$ in the time complexity of Algorithm 6 (Proposition 33) and multiplying the result by K times. The deflation step needs $O(\mathfrak{B}b\log b)$ time and thus does not change the asymptotic rate. □

Of course, it is possible to use a batched version of Algorithm 7, where in each iteration we extract M components using ALS. However, we show in Section 7.3.2 that using $M = 1$ is more effective.

7.2.3 Learning a Factorized PSRNN

Given the previous discussion, we propose the following algorithm for learning a factorized PSRNN:

1. Estimate the initial state q_1 as the average future feature vector $\frac{1}{T}\sum_{t=1}^{T}\phi_t^f$.
2. Estimate the parameter tensor sketch $s_{\mathbf{W}}$, using Algorithm 5.
3. Factorize the parameter tensor using Algorithm 7.
4. Use the factorized PSRNN to compute states $q_{1:t}$ by applying (7.10).
5. Solve a linear regression problem from q_t to o_t to estimate the prediction matrix W_{pred}.
6. Further refine the resulting PSRNN using backpropagation through time.

Algorithm 6 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 $s_{\mathbf{T}}^{(j)}$ for $j \in [\mathfrak{B}]$ where $\mathbf{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$.
  • 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 $\mathbf{W} \approx \sum_{k=1}^K \lambda_k a_k \otimes b_k \otimes c_k$.
1:  $A, B, C \leftarrow$ random $(d_1, d_2, d_3) \times K$ matrices with normalized columns.
2:  for $i = 1$ to $L$ do
3:      for $k = 1$ to $K$ do
4:          for $\beta = 1$ to $\mathfrak{B}$ do
5:              Compute $s_{b_k}^{(2,\beta)}$ and $s_{c_k}^{(3,\beta)}$
6:              $s_{a_k}^{(1,\beta)} \leftarrow s_{\mathbf{T}(I,b_k,c_k)}^{(1,\beta)}$ (using (7.9))
7:          end for
8:          // reconstruct $a_k$:
9:          for $j = 1$ to $d_1$ do
10:             $a_{k,j} \leftarrow \mathrm{med}(\mathfrak{Re}(\{s_{a_k}^{(1,\beta)}[h^{(1,\beta)}(j)]\zeta^{(1,\beta)})\}_{\beta=1}^{\mathfrak{B}})$
11:         end for
12:     end for
13:     $A \leftarrow A((C^\top C) \circ (B^\top B))^+$.
14:     $\lambda_k \leftarrow \|a_k\|$ for $k \in \{1, \dots, K\}$
15:     Normalize the columns of $A$.
16:     Update $B$ and $C$ similarly.
17: end for
```

Algorithm 7 Hybrid Decomposition with ALS and Deflation (DECOMPHYBRID)

```
    • Hash functions $(h^{(i,j)}, \zeta^{(i,j)})$ for $i \in \{1,2,3\}$ and $j \in [\mathfrak{B}]$.
    • Tensor sketch $s_{\mathbf{T}}^{(j)}$ for $j \in [\mathfrak{B}]$ where $\mathbf{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$.
    • Number of factors $K$, block size $M$.
      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
  $\mathbf{W} \approx \sum_{k=1}^K \lambda_k a_k \otimes b_k \otimes c_k$.
1: for $k = 1$ to $K$ do
2:     $\lambda_k, a_k, b_k, c_k \leftarrow$ DECOMPALS$(s_{\mathbf{T}}, K = 1)$
3:     for $j = 1$ to $\mathfrak{B}$ do
4:         $s_{\mathbf{\Delta}}^{(j)} \leftarrow \lambda_i s_{a_i}^{(1,j)} * s_{b_i}^{(2,j)} * s_{c_i}^{(3,j)}$
5:         $s_{\mathbf{T}}^{(j)} \leftarrow s_{\mathbf{T}}^{(j)} - s_{\mathbf{\Delta}}^{(j)}$
6:     end for
7: end for
```

7.3 Experiments

7.3.1 Tensor Product vs. Tensor Decomposition

As mentioned in Section 7.1, tensor sketching allows us to approximate the tensor contraction by applying (7.9) to the sketches of the tensor and the vectors. Tensor contraction appears as part of the state update in (7.10) 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 $\mathbf{T} = \sum_{i=1}^{50} \lambda_i u_i \otimes v_i \otimes w_i \in \mathbb{R}^{200\times200\times200}$, where $\lambda_i = e^{-0.5(i-1)}$ and u_i, v_i and w_i are sampled uniformly from unit sphere. We use sketching with $\mathfrak{B} = 10$ and $b = 10000$ 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 6). We then compute the angle between the true and approximated vectors.

The results are shown in Figure 7.1. 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 different vector u_j. The results of this experiment provides justification to use tensor sketching as a means to obtain a factorized PSRNN as opposed to using it to represent the PSRNN, as described in Section 7.2.1.

7.3.2 Tensor Decomposition: Alternating Least Squares vs. Deflation

In the experiment shown in Figure 7.3, we compare different methods for tensor decomposition, showing the efficacy of rank-1 updates with deflation.

7.3.3 Sketching for Factorized PSRNNs

Downey et al. [44] showed that initializing a PSRNN using 2SR can improve initial performance, the convergence rate of BPTT, and the performance of the final model. In this experiment we compare the initial performance of factorized PSRNNs initialized using sketched 2SR with that of the corresponding full (unfactorized) PSRNN initialized using conventional 2SR. Each factorized PSRNN can be thought of as an approximation of the full PSRNN, with factorized models of higher rank corresponding to better approximations. The key question we seek to answer in this experiment is to what degree the error introduced by compression via sketched factorization impacts the performance of the model as a recursive filter. We compare the initial performance (prior to BPTT) as our goal here is to examine how sketching degrades the performance of two-stage regression.

We use the **Penn Tree Bank (PTB)** dataset for our experiments [80]. Due to hardware limitations we use a train/test split of 120780/124774 characters. We use history (similarly future) features consisting of the past (next) 10 characters as one-hot vectors concatenated together. We use a ridge-regression parameter of 0.05 (consistent with the values suggested in [27, 44]). We use a Gaussian kernel with width set to the median pairwise (Euclidean) distance between neighboring

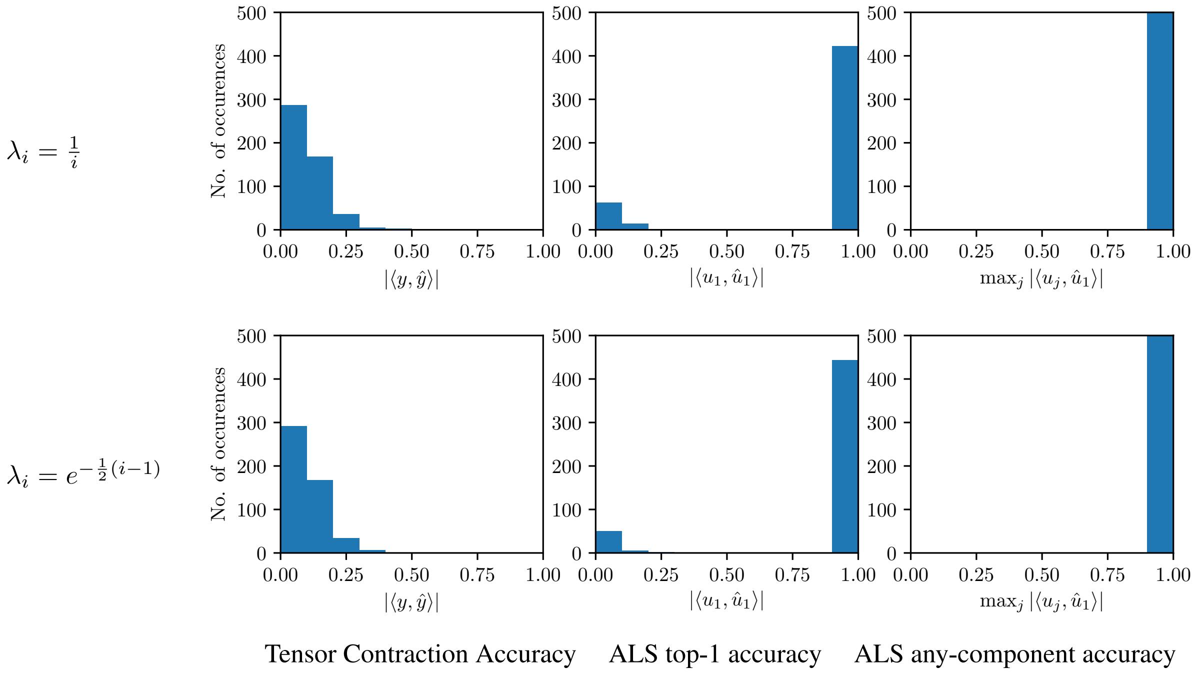


Figure 7.1: Approximation quality of general tensor contraction vs. recovering the first rank-1 component of a tensor. **(left)**: Histogram of dot product between normalized true and approximate contraction results. **(middle)**: Histogram of dot product between true and approximate first rank-1 component 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.

data points, approximated using 1000 RFFs. We use a fixed learning rate of 0.1 for BPTT with a BPTT horizon of 20. We evaluate using perplexity on the one-step prediction task.

Figure 7.3 shows the results. Each blue dot corresponds to a factorized PSRNN with some level of compression. Compression is obtained by using smaller sketches, and a factorized PSRNN with lower rank factors. A full PSRNN has n^3 parameters, while a factorized PSRNN has $3nm$ parameters where m is the number of factors. We use a state of size 50, therefore a full PSRNN has $125,000$ parameters, and a factorized PSRNN with the same number of parameters has rank m 800. Similarly a factorized PSRNN of size 0.2 has rank 160. We use sketches of size 2000, and the number of sketches varies so that the memory used is equivalent to the number of parameters in the final factorized PSRNN. We see that (unsurprisingly) as the compression rate increases, model performance decreases. However this decrease is graceful, and we see that it is possible to achieve high compression rates (e.g. 20x) while still offering significant performance benefits over a randomly initialized model. This result shows that we can use sketched 2SR to directly learn factorized PSRNNs which are orders of magnitude smaller than the corresponding full PSRNNs, yet have comparable performance.

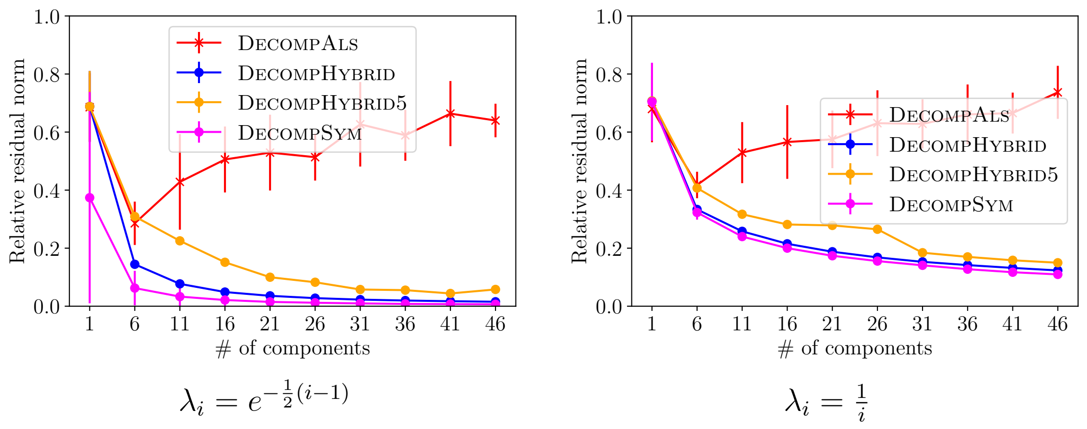


Figure 7.2: Relative residual norm for different decomposition methods using tensor sketches.

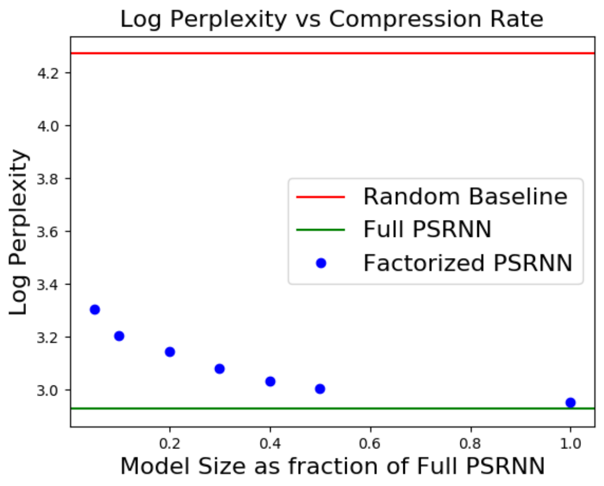


Figure 7.3: Log Perplexity on PTB for factorized PSRNN initialized using Sketched 2SR

7.4 Conclusions

We present Sketched 2SR, a novel formulation of two-stage regression (2SR) which uses tensor sketching, combined with tensor power iteration, to directly learn a factorized PSRNN from data. Sketched 2SR requires only $O(d)$ space and $O(d^2)$ time, compared with $2SR$ which requires $O(d^3)$ space and $O(d^4)$ time.

This tensor sketch approach removes the scaling limitation of PSRNNs, overcoming a major barrier preventing this class of models from being applied to real-world problems at commercial scale. We note that while we focus on PSRNNs, this work can be easily extended to a variety of other models which can be initialized using the 2SR framework.

In addition as part of this work, we also demonstrate observations about the use of tensor sketching that could be of independent interest.

Chapter 8

Orthogonal Random Features for Predictive State Models

The unified models previously developed in this thesis all rely on embedding and manipulating the underlying probability distributions in an RKHS. To obtain practical implementations of these models we use the machinery of Random Features (RFs): input features are mapped into a new space where dot products approximate the kernel well [92]. While RFs are an extremely powerful technique, they have the unfortunate downside that we often require a significant number of RFs in order to obtain good results. Furthermore, the number of required RFs grows with the dimensionality of the input, resulting in models which can be large, slow to execute, and slow to train.

One technique that has proven to be effective for reducing the required number of RFs for kernel machines is Orthogonal Random Features (ORFs) [123]. When using ORFs, the matrix of RFs is replaced by a properly scaled random orthogonal matrix, resulting in significantly decreased kernel approximation error. A particularly nice feature of ORFs is that [37, 123] prove that using ORFs results in a guaranteed improvement in pointwise kernel approximation error when compared with RFs.

Unfortunately the guarantees in Yu et al. [123] are not directly applicable to the models discussed in this thesis. To see why note that PSMs first obtain a set of model parameters via ridge regression, then use these model parameters to calculate inner products in RF space. This “downstream” application of RFs goes beyond the results proven in Yu et al. [123] and Choromanski et al. [37]. Hence it is not clear whether or not ORF can be applied to obtain an improvement in the our setting.

In this work, we show that ORFs can be used to obtain Orthogonal PSRNNs (OPSRNs)and we provide empirical and theoretical evidence that OPSRNNs are both smaller, and faster to execute and train those initialized using conventional unstructured RFs. We theoretically analyze an orthogonal version of the 2-stage regression algorithm for PSRNNs and show that orthogonal models lead to kernel algorithms with strictly better spectral properties and explain how this translates to strictly smaller upper bounds on failure probabilities regarding KRR empirical risk. We compare the performance of OPSRNNs with that of LSTMs as well as conventional PSRNNs on a number of robotics tasks, and show that OPSRRNs are consistently superior on all tasks. In particular, we show that OPSRNN models can achieve accuracy similar to PSRNNs with an order of magnitude smaller number of features needed.

The ideas developed in this chapter are more generally applicable than just PSRNNs, however we focus on PSRNNs for clarity of exposition.

8.1 Orthogonal Random Features

We explain here how to construct orthogonal random features to approximate values of kernels defined by the prominent family of radial basis functions. We then use ORFs to conduct kernel ridge regression for the OPSRNN model. A class of RBF kernels $\mathcal{K}$ (RBFs in shorthand) is a family of functions: $K_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ for $n = 1, 2, ...$ such that $K_n(\mathbf{x}, \mathbf{y}) = \phi(z)$, for $z = \|\mathbf{x} - \mathbf{y}\|_2$, where $\phi : \mathbb{R} \to \mathbb{R}$ is a fixed positive definite function (not parametrized by n). An important example is the class of Gaussian kernels, $\phi = e^{\frac{-x^2}{2\lambda^2}}$.

Every RBF kernel K is shift-invariant, thus in particular its values can be described by an integral via Bochner's Theorem [92]:

$$K(\mathbf{x}, \mathbf{y}) = \mathrm{Re} \int_{\mathbb{R}^n} \exp(i\mathbf{w}^\top(\mathbf{x} - \mathbf{y}))\mu_K(d\mathbf{w}), \tag{8.1}$$

where $\mu_K \in \mathcal{M}(\mathbb{R}^n)$ stands for some finite Borel measure. Some commonly used RBF kernels K together with the corresponding functions ϕ and probability density functions for measures μ_K are given in Table 8.1. The above formula leads straightforwardly to the standard unbiased Monte-Carlo estimator of $K(\mathbf{x}, \mathbf{y})$ given as $K(\mathbf{x}, \mathbf{y}) = \Phi^\top_{m,n}(\mathbf{x})\Phi_{m,n}(\mathbf{y})$, where a random embedding $\Phi_{m,n} : \mathbb{R}^n \to \mathbb{R}^{2m}$ is given as:

$$\Phi_{m,n}(\mathbf{x}) = \left(\frac{1}{\sqrt{m}}\cos(\mathbf{w}_i^\top \mathbf{x}), \frac{1}{\sqrt{m}}\sin(\mathbf{w}_i^\top \mathbf{x})\right)_{i=1}^m \tag{8.2}$$

Here vectors $\mathbf{w}_i \in \mathbb{R}^n$ are sampled independently from μ_K and m stands for the number of random features used. In this scenario we will often use the notation $\mathbf{w}_i^{\mathrm{iid}}$, to emphasize that different $\mathbf{w}_i$s are sampled independently from the same distribution.

Name	Positive-definite function ϕ	Probability density function
Gaussian	$\exp\left(-\frac{z^2}{2\lambda^2}\right)$	$\frac{1}{(2\pi\lambda^2)^{n/2}}\exp\left(-\frac{\|\mathbf{w}\|_2^2}{2\lambda^2}\right)$
Laplacian	$\exp\left(\frac{-z}{\lambda}\right)$	$\prod_{i=1}^{n}\frac{\lambda}{\pi(1+\lambda^2 w_i^2)}$

Figure 8.1: Common RBF kernels, the corresponding functions ϕ, and probability density functions (here: $\mathbf{w} = (w_1, ..., w_n)^\top$).

For a dataset $\mathcal{X}$ we can use random features to obtain transformed data $\Phi(\mathcal{X}) = \{\Phi(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}$. The advantage of this transformation is that inner products in the new space are equivalent to kernel evaluations in the original space, i.e. given $x, y \in \mathcal{X}$:

$$K(x, y) = \Phi(x)^\top \Phi(y)$$

Random Features are crucial to scalable implementations for kernel methods. Specifically when implementing kernels using random features the size of the model is independent of the number of training examples. This is in contrast to conventional approaches to kernel methods which leverage the gram matrix, hence the size of such models grows quadratically with the number of training examples.

Orthogonal random features are obtained by replacing the standard mechanism of constructing vectors $\mathbf{w}_i$ described above with one where the sequence $(\mathbf{w}_1, ..., \mathbf{w}_m)$ is sampled from a "related" joint distribution $\mu^{\mathrm{ort}}_{K,m,n}$ on $R^n \times ... \times R^n$ where $\mu^{\mathrm{ort}}_{K,m,n}$ satisfies the orthogonality condition: with probability 1 different vectors $\mathbf{w}_i$ are pairwise orthogonal. Since in practice we often need $m > n$, the sequence $(\mathbf{w}_i)_{i=1,...,m}$ is obtained by stacking some number of blocks, each of length $l \leq n$, where the blocks are sampled independently from $\mu^{\mathrm{ort}}_{K,l,n}$.

It remains to explain how $\mu^{\mathrm{ort}}_{K,m,n}$ is constructed. We consider two main approaches: The first, which we will refer to as continuous-orthogonal, involves using the gram-schmidt process to select random orthogonal vectors. The second, which we will refer to as discrete-orthogonal, involves constructing specific matrices with useful properties and orthogonal columns.

In the continuous-orthogonal setting the marginal distributions (distributions of individual vectors $\mathbf{w}_i$) are μ_K. Given $w_1, ..., w_i$ the distribution of w_{i+1} is the restriction of μ_K to the subspace orthogonal to $w_1, ..., w_i$. Therefore a sample $(\mathbf{w}^{\mathrm{ort}}_1, ..., \mathbf{w}^{\mathrm{ort}}_m)$ from the joint distribution might be obtained for instance by constructing a random matrix $\mathbf{G} = [(\mathbf{w}^{\mathrm{iid}}_1)^\top; ...; (\mathbf{w}^{\mathrm{iid}}_m)^\top] \in \mathbb{R}^{m \times n}$, performing Gram-Schmidt orthogonalization and then renormalizing the rows such that the length of the renormalized row is sampled from the distribution from which $\|\mathbf{w}^{\mathrm{iid}}_i\|$s are sampled. Thus the Gram-Schmidt orthogonalization is used just to define the directions of the vectors. The fact that for RBF kernels the marginal distributions are exactly μ_K, and thus the kernel estimator is still unbiased, is a direct consequence of the isotropicity of distributions from which the directions of vectors $\mathbf{w}^{\mathrm{iid}}_i$ are sampled. For this class of orthogonal estimators we prove strong theoretical guarantees showing that they lead to kernel ridge regression models superior to state-of-the-art ones based on vectors $\mathbf{w}^{\mathrm{iid}}_i$.

In the discrete-orthogonal setting we denote by $\mathbf{D}$ a random diagonal matrix with nonzero entries taken independently and uniformly at random from the two-element set $\{-1, +1\}$. We also denote by $\mathbf{H}$ a *Hadamard matrix* obtained via Kronecker-products (see: Choromanski et al. [37]). An m-vector sample from the *discrete-orthogonal* joint distribution is obtained by taking m first rows of matrix $\mathbf{G}$ defined as $\mathbf{G}_{\mathrm{HAD}} = \mathbf{HD}_1 \cdot ... \cdot \mathbf{HD}_k$ for fixed $k > 0$, independent copies $\mathbf{D}_i$ of $\mathbf{D}$ and then renormalizing each row in exactly the same way as we did it for continuous-orthogonal joint distributions. Note that $\mathbf{G}_{\mathrm{HAD}}$ is a product of orthogonal matrices, thus its rows are also orthogonal. In practice using $k = 3$ is typically sufficient to obtain a sufficiently good approximation. The key advantage of a discrete approach is that it leads to a more time- and space-efficient method for computing random feature maps (with the use of Fast Walsh-Hadamard Transform; notice that the Hadamard matrix does not even need to be stored explicitly), however this is not our focus in this work. Accuracy-wise, discrete-orthogonal distributions lead to slightly biased estimators since RFs aren't quite uniform on the unit sphere (the bias is a decreasing function of the dimensionality n). However we have observed that in practice discrete- and continuous-orthogonal implementations of PSRNNs give equally good results, with both consistently beating approaches based on unstructured random features. One intuitive explanation of this phenomenon is that, even though in that setting kernel estimators are biased, they still have much lower variance than those based on unstructured features and therefore total MSE is lower. We leave a throughout theoretical analysis of discrete-orthogonal joint distributions in the RNN context to future work.

8.2 The theory of the orthogonal kernel ridge regression

In this section we extend the theoretical guarantees of Yu et al. [123] to give a rigorous analysis of the initialization phase of OPSRNN. Specifically, we provide theoretical guarantees for kernel ridge regression with orthogonal random features, showing that they provide strictly better bounds on our spectral approximation of the ground-truth kernel matrix than unstructured random features. As a corollary, we prove that orthogonal random features lead to strictly smaller empirical risk of the model. While our theoretical results do not quantify this improvement, our experimental results demonstrate significant improvement. Our results go beyond second moment guarantees and enable us to provide the first exponentially small bounds on the probability of a failure for random orthogonal transforms.

Before we state our main results, we will introduce some basic notation and summarize previous results. Assume that labeled datapoints $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \mathbb{R}$ for $i = 1, 2, ...$, are generated as follows: $y_i = f^*(\mathbf{x}_i) + \nu_i$, where $f^* : \mathbb{R}^n \to \mathbb{R}$ is a function that the model aims to learn, and ν_i for $i = 1, 2, ...$ are independent Gaussians with zero mean and standard deviation $\sigma > 0$. The *empirical risk of the estimator* $f : \mathbb{R}^n \to \mathbb{R}$ is defined as follows:

$$\mathcal{R}(f) \equiv \mathbb{E}_{\{\nu_i\}_{i=1,\ldots,N}}[\frac{1}{N}\sum_{j=1}^{N}(f(\mathbf{x}_i) - f^*(\mathbf{x}_i))^2], \tag{8.3}$$

where N stands for a dataset size.

By $\mathbf{f}^*_{\text{vec}} \in \mathbb{R}^N$ we denote a vector whose i^{th} entry is $f^*(\mathbf{x}_i)$. Denote by f_{KRR} a kernel ridge regression estimator applying an exact kernel method (no random feature map approximation). Assume that we analyze kernel $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ with the corresponding kernel matrix $\mathbf{K}$. It is a well known result [3, 11] that the empirical risk of f_{KRR} is given by the formula:

$$\mathcal{R}(f_{\text{KRR}}) = N^{-1}\lambda^2(\mathbf{f}^*_{\text{vec}})^\top(\mathbf{K} + \lambda N\mathbf{I}_N)^{-2}\mathbf{f}^*_{\text{vec}} + N^{-1}\sigma^2\text{Tr}(\mathbf{K}^2(\mathbf{K} + \lambda N\mathbf{I}_N)^{-2}) \tag{8.4}$$

where $\lambda > 0$ stands for the regularization parameter and $\mathbf{I}_N \in \mathbb{R}^{N\times N}$ is an identity matrix. Equation 8.4 can be viewed as a bias-variance decomposition of the risk. The first term corresponds to the bias, the degree to which our modelling assumptions are incorrect. The second term corresponds to the variance, the sensitivity of our model to small fluctuations in the training data. We note that as the regularization increases the biases also increases, but the variance decreases.

Denote by $\widehat{f}_{\text{KRR}}$ an estimator based on some random feature map mechanism and by $\widehat{\mathbf{K}}$ the corresponding approximate kernel matrix.

The expression that is used in several bounds on the empirical risk for kernel ridge regression (see for instance Avron et al. [10]) is a modified version of the above formula for $\mathcal{R}(f_{\text{KRR}})$ which is easier to manipulate: $\bar{\mathcal{R}}_{\mathbf{K}}(\mathbf{f}^*_{\text{vec}}) \equiv N^{-1}\lambda^2(\mathbf{f}^*_{\text{vec}})^\top(\mathbf{K}+\lambda N\mathbf{I}_N)^{-1}\mathbf{f}^*_{\text{vec}} + N^{-1}\sigma^2 s_\lambda(\mathbf{K})$, where $s_\lambda(\mathbf{K}) \equiv \text{Tr}(\mathbf{K}(\mathbf{K}+\lambda N\mathbf{I}_N)^{-1})$. It can be easily proven that $\mathcal{R}(f_{\text{KRR}}) \le \bar{\mathcal{R}}_{\mathbf{K}}(\mathbf{f}^*_{\text{vec}})$.

To measure how similar two kernel matrices are in their spectral properties we use the notion of Δ-spectral approximation [10].

Definition 35. *For a given $0 < \Delta < 1$, matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ is a Δ-spectral approximation of a matrix $\mathbf{B} \in \mathbb{R}^{N\times N}$ if $(1-\Delta)\mathbf{B} \preceq \mathbf{A} \preceq (1+\Delta)\mathbf{B}$.*

It turns out that one can upper-bound the risk $\mathcal{R}(\widehat{f}_{\text{KRR}})$ for the estimator $\widehat{f}_{\text{KRR}}$ in terms of Δ if the matrix $\widehat{\mathbf{K}}+\lambda N\mathbf{I}_N$ is a Δ-spectral approximation of the matrix $\mathbf{K}+\lambda N\mathbf{I}_N$, as the next result [10] shows:

Theorem 36. *Suppose that $\|\mathbf{K}\|_2 \ge 1$ and that matrix $\widehat{\mathbf{K}}+\lambda N\mathbf{I}_N$ obtained with the use of random features is a Δ-spectral approximation of matrix $\mathbf{K}+\lambda N\mathbf{I}_N$. Then the empirical risk $\mathcal{R}(\widehat{f}_{\text{KRR}})$ of the estimator $\widehat{f}_{\text{KRR}}$ satisfies:*

$$\mathcal{R}(\widehat{f}_{\text{KRR}}) \le \frac{1}{1-\Delta}\bar{\mathcal{R}}_{\mathbf{K}}(\mathbf{f}^*_{\text{vec}}) + \frac{\Delta}{1+\Delta}\frac{\text{rank}(\widehat{\mathbf{K}})}{N}\sigma^2. \tag{8.5}$$

Theorem 36 bounds the risk of the approximate KRR estimator as a function of both the risk upper bound $\bar{\mathcal{R}}_{\mathbf{K}}$ and an additive term which is small if $\text{rank}(\widehat{\mathbf{K}})$ and/or Δ is small. We note that an approximation $\widehat{\mathbf{K}}$ is only useful computationally if $\text{rank}(\widehat{\mathbf{K}}) << n$ so $\widehat{\mathbf{K}}$ gives a significantly compressed approximation to the original kernel matrix. Ideally we should have $\text{rank}(\widehat{\mathbf{K}})/N \to 0$ as $N \to \infty$ so the additive term will also approach zero, and be small when N is large.

8.2.1 Superiority of the orthogonal features for kernel ridge regression

Consider the following RBF kernels, which we call *smooth RBFs*.

Definition 37 (smooth RBFs)**.** *Let a fixed $\phi : \mathbb{R} \to \mathbb{R}$ be a class of RBF kernels, where different elements of the class corresponds to different input dimensionalities. Let $\{\mu_1, \mu_2, ...\}$ ($\mu_i \in \mathcal{M}(\mathbb{R}^i)$) be an associated sequence of probabilistic measures. We say such a class is smooth if there exists a nonincreasing function $f : \mathbb{R} \to \mathbb{R}$ such that $f(x) \to 0$ as $x \to \infty$ and furthermore the k^{th} moments of random variables $X_n = \|\mathbf{w}\|$, where $\mathbf{w} \sim \mu_n$ satisfy for every $n, k \ge 0$: $\mathbb{E}[X_n^k] \le (n-1)(n+1)\cdot ... \cdot (n+2k-3)k!f^k(k)$.*

Many important classes of RBF kernels are smooth, in particular the class of Gaussian kernels. This follows immediately from the well-known fact that for Gaussian kernels the above k^{th} moments are given by: $\mathbb{E}[X_n^k] = 2^k\frac{(\frac{n}{2}+k-1)!}{(\frac{n}{2}-1)!}$ for $n > 1$.

Our main result is given below and shows that orthogonal random features lead to tighter bounds on Δ for the spectral approximation of $\mathbf{K}+\lambda N\mathbf{I}_N$. Tighter bounds on Δ, as Theorem 36 explains, lead to tighter upper bounds also on the empirical risk of the estimator. We will prove it for the setting where each structured block consists of a fixed number $l > 1$ of rows (note that many independent structured blocks are needed if $m > n$), however our experiments suggest that the results are valid also without this assumption.

Theorem 38 (spectral approximation)**.** *Consider a smooth RBF (such as the Gaussian kernel). Let $\widehat{\Delta}_{\text{iid}}$ denote the smallest positive number such that $\widehat{\mathbf{K}}_{\text{iid}} + \lambda N \mathbf{I}_N$ is a Δ-approximation of $\mathbf{K} + \lambda N \mathbf{I}_N$, where $\widehat{\mathbf{K}}_{\text{iid}}$ is an approximate kernel matrix obtained by using unstructured random features. Then for any $a > 0$,*

$$\mathbb{P}[\widehat{\Delta}_{\text{iid}} > a] \leq p^{\text{iid}}_{N,m}(\frac{a\sigma_{\min}}{N}), \tag{8.6}$$

where: $p^{\text{iid}}_{N,m}$ is given as: $p^{\text{iid}}_{N,m}(x) = N^2 e^{-Cmx^2}$ for some universal constant $C > 0$, m is the number of random features used, $\sigma_{\min}$ is the smallest singular value of $\mathbf{K} + \lambda N \mathbf{I}_N$ and N is dataset size. If instead orthogonal random features are used then for the corresponding spectral parameter $\widehat{\Delta}_{\text{ort}}$ the following holds:

$$\mathbb{P}[\widehat{\Delta}_{\text{ort}} > a] \leq p^{\text{ort}}_{N,m}(\frac{a\sigma_{\min}}{N}), \tag{8.7}$$

where function $p^{\text{ort}}_{N,m}$ satisfies: $p^{\text{ort}}_{N,m} < p^{\text{iid}}_{N,m}$ for n large enough.

We see that both constructions lead to exponentially small (in the number of random features m used) probabilities of failure, however the bounds are tighter for the orthogonal case. An exact formula on $p^{\text{ort}}_{N,m}$ can be derived from the proof that we present in the Appendix, however for space we do not give it here.

Theorem 38 combined with Theorem 36 lead to risk bounds for the kernel ridge regression model based on random unstructured and random orthogonal features. We use the notation introduced before and obtain the following:

Theorem 39. *Under the assumptions of Theorem 36 and Theorem 38, the following holds for the kernel ridge regression risk and any $c > 0$ if m-dimensional unstructured random feature maps are used to approximate a kernel: $\mathbb{P}[\mathcal{R}(\widehat{f}_{\text{KRR}}) > c] \leq p^{\text{iid}}_{N,m}(\frac{a_c\sigma_{\min}}{N})$, where a_c is given as: $a_c = 1 - \frac{\bar{\mathcal{R}}_{\mathbf{K}}(\mathbf{f}^*_{\text{vec}})}{c - \frac{m\sigma^2}{2N}}$ and the probability is taken with respect to the random choices of features. If instead random orthogonal features are used, we obtain the following bound: $\mathbb{P}[\mathcal{R}(\widehat{f}_{\text{KRR}}) > c] \leq p^{\text{ort}}_{N,m}(\frac{a_c\sigma_{\min}}{N})$.*

As before, since for large n function $p^{\text{ort}}_{N,m}$ satisfies $p^{\text{ort}}_{N,m} < p^{\text{iid}}_{N,m}$, for orthogonal random features we obtain strictly smaller upper bounds on the failure probability regarding empirical risk than for the state-of-the-art unstructured ones. In practice, as we will show in the experimental section, we see gains also in the regimes of moderate dimensionalities n.

8.3 Experiments

In section 8.2 we extended the theoretical guarantees for ORFs to the case of the initialization phase of OPSRNNs. In this section we confirm these results experimentally and show that they imply better performance of the entire model by comparing the performance of PSRNNs with that of OPSRNNs on a selection of robotics time-series datasets. Since OPSRNN models obtained via continuous-orthogonal and discrete-orthogonal joint sampling (see: Section 8.1) gave almost the same results, presented OPSRNN-curves are for the continuous-orthogonal setting.

8.3.1 Experimental Setup

We now describe the datasets and model hyperparameters used in our experiments. All models were implemented using the Tensorflow framework in Python.

We use the following datasets in our experiments:

- **Swimmer** We consider the 3-link simulated swimmer robot from the open-source package OpenAI gym.[1] The observation model returns the angular position of the nose as well as the (2D) angles of the two joints, giving a total of 5 features. We collect 25 trajectories from a robot that is trained to swim forward (via the cross entropy with a linear policy), with a train/test split of 20/5.
- **Mocap** A Human Motion Capture dataset consisting of 48 skeletal tracks from three human subjects collected while they were walking. The tracks have 300 time steps each, and are from a Vicon motion capture system. We use a train/test split of 40/8. There are 22 total features consisting of the 3D positions of the skeletal parts (e.g., upper back, thorax, clavicle).
- **Handwriting** This is a digital database available on the UCI repository [4] created using a pressure sensitive tablet and a cordless stylus. Features are x and y tablet coordinates and pressure levels of the pen at a sampling rate of 100 milliseconds giving a total of 3 features. We use 25 trajectories with a train/test split of 20/5.
- **Moving MNIST** Pairs of MNIST digits bouncing around inside of a box according to ideal physics. `http://www.cs.toronto.edu/~nitish/unsupervised_video/`. Each video is 64x64 pixels single channel (4096 features) and 20 frames long. We use 1000 randomly selected videos, split evenly between train and test.

In two-stage regression we use history (similarly future) features consisting of the past (next) 2 observations concatenated together. We use a ridge-regression parameter of 10^{-2} (this is consistent with the values suggested in Boots et al. [27], Downey et al. [44]). The kernel width is set to the median pairwise (Euclidean) distance between neighboring data points. We use a fixed learning rate of 0.1 for BPTT with a BPTT horizon of 20. We use a single layer PSRNN.

We optimize and evaluate all models with respect to the Mean Squared Error (MSE) of one step predictions (this should not be confused with the MSE of the pointwise kernel approximation which does not give the downstream guarantees we are interested in here). This means that to evaluate the model we perform recursive filtering on the test set to produce states, then use these states to make predictions about observations one time step in the future. Making recursive predictions with longer horizons is also possible, and significantly increases the difficulty of the task.

8.3.2 Results

Orthogonal RF for 2SR

In our first experiment we examine the effectiveness of Orthogonal RF with respect to learning a good PSRNN via 2SR. In figure 8.2 we compare the MSE for a PSRNN learned via Orthogonal

[1]`https://gym.openai.com/`

RF with that of one learned using Standard RF for varying numbers of random features. Note that these models were initialized using 2SR but were *not* refined using BPTT. We see that in all cases when the ratio of RF to input dimension is small Orthogonal RF significantly outperforms Standard RF. This difference decreases as the number of RF increases, with both approaches resulting in similar MSE for large RF to input ratios.

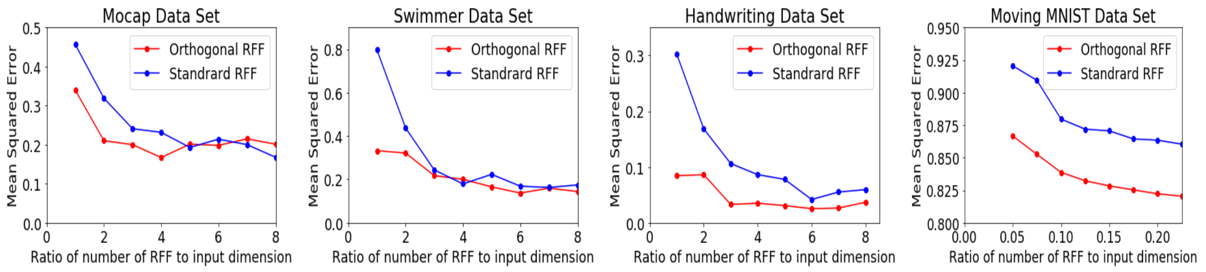


Figure 8.2: MSE for Orthogonal RF vs Standard RF after two stage regression

Orthogonal RF for BPTT

In our second experiment we examine the effectiveness of Orthogonal RF with respect to learning a good PSRNN via 2SR initialization *combined with refinement via BPTT*. In figure 8.3 we compare the MSE for a PSRNN learned via Orthogonal RF with that of one learned using Standard RF over a number of epochs of BPTT. We see that on all datasets, for both Orthogonal RF and Standard RF, MSE decreases as the number of epochs increases. However it is interesting to note that in all datasets Orthogonal RF converges to a better MSE than Standard RF.

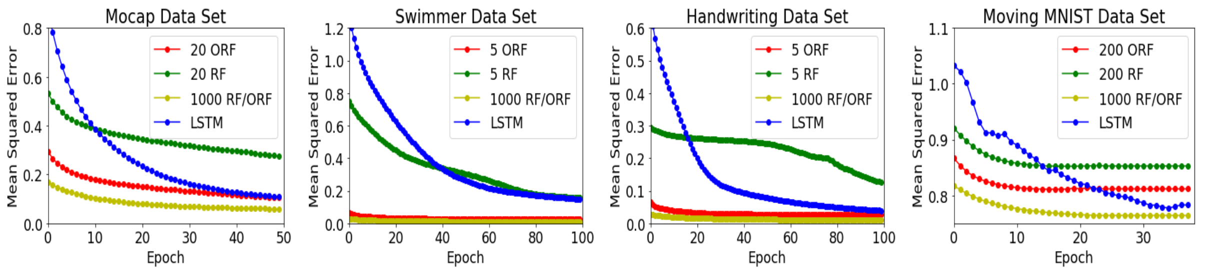


Figure 8.3: MSE for Orthogonal RF vs Standard RF after two stage regression and BPTT

8.3.3 Discussion

These results demonstrate the effectiveness of Orthogonal RF as a technique for improving the performance of downstream applications. First we have shown that Orthogonal RF can offer significant performance improvements for kernel ridge regression, specifically in the context of the 2SR algorithm for PSRNNs. Furthermore we have shown that not only does the resulting model have lower error, it is also a better initialization for the BPTT gradient descent procedure.

In other words, using a model initialization based on orthogonal RF results in BPTT converging to a superior final model.

While the focus of these experiments was to compare the performance of PSRNNs and OPSRNNs, for the sake of completeness we also include error plots for LSTMs. We see that OPSRNNs significantly outperform LSTMs on all datasets, and outperform PSRNNs of the same size when the number of RFs is small.

8.4 Related Work

Orthogonal random features were introduced in Yu et al. [123] as an alternative to the standard approach for constructing random feature maps to scale kernel methods [92]. Several other structured constructions were known before [2, 18, 35, 36, 62, 115, 124], however these were motivated by computational and space complexity gains and led to weaker accuracy guarantees. In contrast to this previous work, orthogonal random features were proposed to improve accuracy of the estimators relying on them. Such an improvement was theoretically and experimentally verified, but only for pointwise kernel approximation [37, 123] and for specific kernels (such as Gaussian for dimensionality large enough, as well as dot-product and angular kernels). It was not clear whether these pointwise gains translate to downstream guarantees for algorithms relying on kernels (for instance kernel ridge regression), even though there was some partial empirical evidence that this might be the case (in Choromanski et al. [37] orthogonal random features were experimentally tested to provide more accurate approximation of the groundtruth kernel matrix in terms of the Frobenius norm error). Even for the pointwise estimators and for the selected kernels, the guarantees were given only with the use of second moment methods (variance computation) and thus did not lead to strong concentration results with exponentially small probabilities of failure, which we obtain in this paper.

To the best of our knowledge, we are the first to apply orthogonal random features via kernel ridge regression for recurrent neural networks. There is however vast related literature on orthogonal recurrent neural networks, where the matrices are enforced to be orthogonal or initialized to be random orthogonal. Probably some of the most exploited recent directions are unitary evolution RNN architectures [8], where orthogonal and unitary matrices are used to address a key problem of RNN training – vanishing or exploding gradients. Related results are presented in
Henaff et al. [61], Saxe et al. [96] (orthogonal initialization for training acceleration), Ganguli et al. [51] and White et al. [120]. Most of these results do not provide any strict theoretical guarantees regarding the superiority of the orthogonal approach.

Even though these approaches are only loosely related to our work, there is a common denominator: orthogonality, whether applied in our context or the aforementioned ones, seems to be responsible in part for disentangling in (deep) representations [1]. Our theoretical analysis suggests that this phenomenon occurs for the orthogonal KRR that is used as a subroutine of OPSRNNs, but the general mechanism is still not completely understood from the theoretical point of view.

8.5 Conclusions

We showed how structured orthogonal constructions can be effectively integrated with recurrent neural network based architectures to provide models that consistently achieve performance superior to the baselines. They also provide significant compression, achieving similar accuracy as PSRNNs with an order of magnitude smaller number of features needed. Furthermore, we gave the first theoretical guarantees showing that orthogonal random features lead to exponentially small bounds on the failure probability regarding empirical risk of the kernel ridge regression model. The latter result is an important property for RNN based architectures for state prediction that we consider in this paper. Finally, we proved that these bounds are strictly better than for the standard non-orthogonal random feature map mechanism. Our theoretical results do not provide a quantitative bound on the improvement, however Experiments conducted on several robotics task demonstrate that this improvement leads to significant practical performance improvements.

Proofs

We will use the notation from the main body of the paper.

8.A Proof of Theorem 38

We will assume that a dataset $\mathcal{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ under consideration is taken from a ball $\mathcal{B}$ of a fixed radius r (that does not depend on data dimensionality n and dataset size N) and center $\mathbf{x}_0$. We begin with the following lemma:

Lemma 40. *Fix an RBF kernel $K : \mathbb{R}^n \times \mathbb{R}^n$. Consider a randomized kernel estimator $\widehat{K}$ with a corresponding random feature map: $\Phi_{m,n} : \mathbb{R}^n \rightarrow \mathbb{R}^{2m}$ and assume that for any fixed $i, j \in \{1, ..., N\}$ the followig holds for any $c > 0$: $\mathbb{P}[|\Phi_{m,n}(\mathbf{x}_i)^\top \Phi_{m,n}(\mathbf{x}_j) - K(\mathbf{x}_i, \mathbf{x}_j)| > c] \leq g(c)$ for some fixed function $g : \mathbb{R} \rightarrow \mathbb{R}$. Then with probability at least $1 - N^2 g(c)$, matrix $\widehat{\mathbf{K}} + \lambda \mathbf{I}_N$ is a Δ-spectral approximation of matrix $\mathbf{K} + \lambda \mathbf{I}_N$ for $\Delta = \frac{Nc}{\sigma_{\min}}$, where $\sigma_{\min}$ stands for the minimal singular value of $\mathbf{K} + \lambda \mathbf{I}_N$.*

Proof. Denote $\mathbf{K} + \lambda N \mathbf{I}_N = \mathbf{V}^\top \boldsymbol{\Sigma}^2 \mathbf{V}$, where an orthonormal matrix $\mathbf{V} \in \mathbf{R}^{N \times N}$ and a diagonal matrix $\boldsymbol{\Sigma} \in \mathbf{R}^{N \times N}$ define the eigendecomposition of $\mathbf{K} + \lambda N \mathbf{I}_N$. Following [10], we notice that in order to prove that $\widehat{\mathbf{K}} + \lambda \mathbf{I}_N$ is a Δ-spectral approximation of $\mathbf{K} + \lambda \mathbf{I}_N$, it suffices to show that:

$$\|\boldsymbol{\Sigma}^{-1}\mathbf{V}\widehat{\mathbf{K}}\mathbf{V}^\top\boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1}\mathbf{V}\mathbf{K}\mathbf{V}^\top\boldsymbol{\Sigma}^{-1}\|_2 \leq \Delta. \tag{8.8}$$

From basic properties of the spectral norm $\|\|_2$ and the Frobenius norm $\|\|_F$ we have:

$$\mathbb{P}[\|\boldsymbol{\Sigma}^{-1}\mathbf{V}\widehat{\mathbf{K}}\mathbf{V}^\top\boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1}\mathbf{V}\mathbf{K}\mathbf{V}^\top\boldsymbol{\Sigma}^{-1}\|_2 > \Delta] \leq \mathbb{P}[\|\boldsymbol{\Sigma}^{-1}\mathbf{V}\|_2 \|\widehat{\mathbf{K}} - \mathbf{K}\|_F \|\mathbf{V}^\top\boldsymbol{\Sigma}^{-1}\|_2 > \Delta] \tag{8.9}$$

The latter probability is equal to $p = \mathbb{P}[\|\widehat{\mathbf{K}} - \mathbf{K}\|_F^2 > \frac{\Delta^2}{\|\boldsymbol{\Sigma}^{-1}\mathbf{V}\|_2^2 \cdot \|\mathbf{V}^\top\boldsymbol{\Sigma}^{-1}\|_2^2}]$.

Furthermore, since $\mathbf{V}$ is an isometry matrix, we have: $\|\boldsymbol{\Sigma}^{-1}\mathbf{V}\|_2^2 \leq \frac{1}{\sigma_{\min}}$ and $\|\mathbf{V}^\top\boldsymbol{\Sigma}^{-1}\|_2^2 \leq \frac{1}{\sigma_{\min}}$.

Thus we have:

$$p \leq \mathbb{P}[\|\widehat{\mathbf{K}} - \mathbf{K}\|_F^2 > \Delta^2 \sigma_{\min}^2]. \tag{8.10}$$

Now notice that from the union bound we get:

$$p \leq N^2 \mathbb{P}[|\widehat{\mathbf{K}}(i,j) - \mathbf{K}(i,j)| > \frac{\Delta\sigma_{\min}}{N}] = N^2 \mathbb{P}[|\Phi_{m,n}(\mathbf{x}_i)^\top \Phi_{m,n}(\mathbf{x}_j) - \mathbf{K}(i,j)| > \frac{\Delta\sigma_{\min}}{N}]. \tag{8.11}$$

Therefore the probability that $\widehat{\mathbf{K}} + \lambda \mathbf{I}_N$ is a Δ-spectral approximation of $\mathbf{K} + \lambda \mathbf{I}_N$ is at least $1 - N^2 g(c)$ for $c = \frac{\Delta\sigma_{\min}}{N}$ and that completes the proof.

□

Our goal right now is to compute function g from Lemma 40 for random feature maps constructed according to two procedures: the standard one based on independent sampling and the orthogonal one, where marginal distributions corresponding to the joint distribution $(\mathbf{w}_1, ..., \mathbf{w}_m)$ are the same, but vectors $\mathbf{w}_i$ are conditioned to be orthogonal.

We start with a standard random feature map mechanism. Note first that from basic properties of the trigonometric functions we conclude that for any two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, the random feature map approximation of the RBF kernel $K(\mathbf{x}, \mathbf{y})$ which is of the form $\widehat{K}(\mathbf{x}, \mathbf{y}) = |\Phi_{m,n}(\mathbf{x})^\top \Phi_{m,n}(\mathbf{y})$ can be equivalently rewritten as: $\widehat{\mathbf{K}}(\mathbf{x}, \mathbf{y}) = \frac{1}{m}\sum_{i=1}^m \cos(\mathbf{w}_i^\top \mathbf{z})$ for $\mathbf{z} = \mathbf{x} - \mathbf{y}$. This is true for any joint distribution $(\mathbf{w}_1, ..., \mathbf{w}_m)$.

Lemma 41. *If mapping $\Phi_{m,n}$ is based on the standard mechanism of independent sampling then one can take as function g from Lemma 40 a function given by the following formula: $g(x) = e^{-Cmx^2}$ for some universal constant $C > 0$.*

Proof. Notice first that by the remark above, we get: $\mathbb{P}[|\Phi_{m,n}(\mathbf{x}_i)^\top \Phi_{m,n}(\mathbf{x}_j) - K(\mathbf{x}_i, \mathbf{x}_j)| > x] = \mathbb{P}[\sum_{i=1}^m Z_i > x]$, where $Z_i = \frac{1}{m}\cos(\mathbf{w}_i^\top \mathbf{z}) - \frac{1}{m}\phi(z)$, $z = \|\mathbf{x} - \mathbf{y}\|_2$ and ϕ is a positive definite function associated with an RBF kernel K. From the unbiasedness of the estimator we see that $\mathbb{E}[Z_i] = 0$. Also, notice that: $|Z_i| \leq \frac{2}{m}$ and different Z_is are independent. Thus, from the standard application of Chernoff inequality we get: $\mathbb{P}[\sum_{i=1}^n Z_i > x] \leq e^{-Cmx^2}$ for some universal constant $C > 0$ (that does not depend on m). That completes the proof. $\square$

By combining Lemma 40 with the formula on g for the standard unstructured case from Lemma 41, we already obtain the formula for $p^{\mathrm{iid}}_{N,m}$ from the statement of the theorem. It remains to show that: $p^{\mathrm{ort}}_{N,m} < p^{\mathrm{iid}}_{N,m}$.

Note that in the previous proof the upper bound on g is derived as a monotonic function of $\mathbb{E}[e^{t\sum_{i=1}^m Z_i}]$ for a parameter $t > 0$ (that is then being optimized),as it is the case for the standard Chernoff's argument. Now, since variables Z_i are independent, we obtained: $\mathbb{E}[e^{t\sum_{i=1}^m Z_i}] = \prod_{i=1}^m \mathbb{E}[e^{tZ_i}]$. Thus, if we can prove that for the continuous-orthogonal distribution we have: $\mathbb{E}[e^{t\sum_{i=1}^m Z_i}] < \prod_{i=1}^m \mathbb{E}[e^{tZ_i}]$, then we complete the proof of Theorem 38 (note that the marginal distributions of $\mathbf{Z}_i$ are the same for both: standard mechanism based on unstructured random features and the one based on continuous-orthogonal sampling of the m-tuple of n-dimensional vectors).

This is what we prove below.

Lemma 42. *Fix some $\mathbf{z} \in \mathbb{R}^n$ and $t > 0$. For a sample $(\mathbf{w}_1^{\mathrm{ort}}, ..., \mathbf{w}_m^{\mathrm{ort}})$ from the continuous-orthogonal distribution the following holds for n large enough:*

$$\mathbb{E}[e^{\frac{t}{n}\sum_{i=1}^m \cos((\mathbf{w}_i^{\mathrm{ort}})^\top \mathbf{z})}] < \prod_{i=1}^m \mathbb{E}[e^{\frac{t}{n}\cos((\mathbf{w}_i^{\mathrm{ort}})^\top \mathbf{z})}]. \tag{8.12}$$

Proof. Since different blocks of vectors $\mathbf{w}_i$ used to construct the orthogonal feature map are independent (the number of blocks is greater than one if $m > n$), it suffices to prove the inequality just for one block. Thus from now on we will focus just on one block and thus without loss of generality we will assume that $m \leq n$.

Note first that

$$\prod_{i=1}^m \mathbb{E}[e^{\frac{t}{n}\cos((\mathbf{w}_i^{\mathrm{ort}})^\top \mathbf{z})}] = \mathbb{E}[e^{\frac{t}{n}\sum_{i=1}^m \cos((\mathbf{w}_i^{\mathrm{iid}})^\top \mathbf{z})}], \tag{8.13}$$

where $(\mathbf{w}_1^{\mathrm{iid}}, ..., \mathbf{w}_m^{\mathrm{iid}})$ stands for the m-tuple sample constructed by the standard unstructured mechanism.

Thus we need to prove that

$$\mathbb{E}[e^{\frac{t}{n}\sum_{i=1}^{m}\cos((\mathbf{w}_i^{\mathrm{ort}})^\top \mathbf{z})}] < \mathbb{E}[e^{\frac{t}{n}\sum_{i=1}^{m}\cos((\mathbf{w}_i^{\mathrm{iid}})^\top \mathbf{z})}] \tag{8.14}$$

$\square$

Using Taylor expansion for e^x, we conclude that it sufficies to prove that:

$$\begin{aligned}&\sum_{j_1,j_2,\ldots,j_m}(\frac{t}{n})^{j_1+\ldots+j_m}\mathbb{E}[\frac{\cos((\mathbf{w}_1^{\mathrm{ort}})^\top\mathbf{z})^{j_1}\cdot\ldots\cdot\cos((\mathbf{w}_m^{\mathrm{ort}})^\top\mathbf{z})^{j_m}}{j_1!\cdot\ldots\cdot j_k!}]\\ <&\sum_{j_1,j_2,\ldots,j_m}(\frac{t}{n})^{j_1+\ldots+j_m}\mathbb{E}[\frac{\cos((\mathbf{w}_1^{\mathrm{iid}})^\top\mathbf{z})^{j_1}\cdot\ldots\cdot\cos((\mathbf{w}_m^{\mathrm{iid}})^\top\mathbf{z})^{j_m}}{j_1!\cdot\ldots\cdot j_k!}],\end{aligned} \tag{8.15}$$

i.e. that:

$$\sum_{j_1,j_2,\ldots,j_m}(\frac{t}{n})^{j_1+\ldots+j_m}\frac{1}{j_1!\cdot\ldots\cdot j_m!}\Lambda(j_1,\ldots,j_m) > 0, \tag{8.16}$$

where:

$$\Lambda(j_1,\ldots,j_k) = \mathbb{E}[\cos((\mathbf{w}_1^{\mathrm{iid}})^\top\mathbf{z})^{j_1}\cdot\ldots\cdot\cos((\mathbf{w}_m^{\mathrm{iid}})^\top\mathbf{z})^{j_m} - \cos((\mathbf{w}_1^{\mathrm{ort}})^\top\mathbf{z})^{j_1}\cdot\ldots\cdot\cos((\mathbf{w}_m^{\mathrm{ort}})^\top\mathbf{z})^{j_m}] \tag{8.17}$$

By applying the trigonometric formula:

$$\cos(\alpha)\cos(\beta) = \frac{1}{2}(\cos(\alpha+\beta)+\cos(\alpha-\beta)), \tag{8.18}$$

we get:

$$\begin{aligned}\Lambda(j_1,\ldots,j_k) = \frac{1}{2^{j_1+\ldots+j_m}}&\sum_{s_1,\ldots,s_{j_1+\ldots+j_m}\in\{-1,+1\}}\mathbb{E}[\\ \cos\big(((\mathbf{w}_1^{\mathrm{iid}}\otimes_{s_1}\mathbf{w}_1^{\mathrm{iid}}\otimes_{s_2}&\ldots\otimes_{s_{j_1-1}}\mathbf{w}_1^{\mathrm{iid}})\otimes_{s_{j_1}}\ldots)^\top\mathbf{z}\big) -\\ \cos\big(((\mathbf{w}_1^{\mathrm{ort}}\otimes_{s_1}\mathbf{w}_1^{\mathrm{ort}}\otimes_{s_2}&\ldots\otimes_{s_{j_1-1}}\mathbf{w}_1^{\mathrm{ort}})\otimes_{s_{j_1}}\ldots)^\top\mathbf{z}\big)],\end{aligned} \tag{8.19}$$

where $\otimes_1$ stands for vector-addition operator and $\otimes_{-1}$ stands for vector-subtraction operator.

Note that without loss of generality we can assume that $s_{j_1} = s_{j_1+j_2} = \ldots = +1$, since for other configurations we obtain a random variable of the same distribution. Consider a fixed configuration $(s_1, s_2, \ldots, s_{j_1+\ldots+j_m})$ and the corresponding term of the sum above that can be rewritten in the compressed way as:

$$F = \mathbb{E}[\cos(n_1\mathbf{w}_1^{\mathrm{iid}}+n_2\mathbf{w}_2^{\mathrm{iid}}+\ldots+n_m\mathbf{w}_m^{\mathrm{iid}})^\top\mathbf{z}] - \mathbb{E}[\cos(n_1\mathbf{w}_1^{\mathrm{ort}}+n_2\mathbf{w}_2^{\mathrm{ort}}+\ldots+n_m\mathbf{w}_m^{\mathrm{ort}})^\top\mathbf{z}], \tag{8.20}$$

for some $n_1, \ldots, n_m \in \mathbb{Z}$. Without loss of generality, we can assume that $n_1, \ldots, n_m \in \mathbb{N}$, since the distribution of the random variables under consideration does not change if n_i is replaced with $-n_i$. Without loss of generality we can also assume that there exists $i \in \{1, ..., m\}$ such that $n_i > 0$, since otherwise the corresponding term F is equal to 0.

Denote by $R_1, ..., R_m$ the set of independent random variables, where each is characterized by the distribution which is the distribution of vectors $\mathbf{w}_i^{\mathrm{iid}}$ (and thus also of vectors $\mathbf{w}_i^{\mathrm{ort}}$). Denote: $R = \sqrt{n_1^2 R_1^2 + ... + n_m^2 R_m^2}$. Note that $n_1 \mathbf{w}_1^{\mathrm{ort}} + n_2 \mathbf{w}_2^{\mathrm{ort}} + ... + n_m \mathbf{w}_m^{\mathrm{ort}} \sim R\mathbf{v}$, where $\mathbf{v}$ is a unit L_2-norm vector taken uniformly at random from the sphere of radius 1 and furthermore: R and $\mathbf{v}$ are chosen independently. That is implied by the isotropicity of vectors $\mathbf{w}_i^{\mathrm{ort}}$. Similarly, denote $\widehat{R} = \sqrt{R^2 + \sum_{i,j \in \{1,...,m\}} n_i n_j R_i R_j \mathbf{v}_i^\top \mathbf{v}_j}$, where $\mathbf{v}_1, ..., \mathbf{v}_m$ stand for the independent copies of $\mathbf{v}$. Note that, by the similar analysis as before, we conclude that $n_1 \mathbf{w}_1^{\mathrm{iid}} + n_2 \mathbf{w}_2^{\mathrm{iid}} + ... + n_m \mathbf{w}_m^{\mathrm{iid}} \sim \widehat{R}\mathbf{v}$ and furthermore, $\widehat{R}$ and $\mathbf{v}$ are chosen independently.

Therefore, by expanding $\cos(x)$ using Taylor expansion, we get:

$$F = \sum_{k=0}^{\infty} \frac{\|\mathbf{z}\|^{2k} (-1)^k \mathbb{E}[(\mathbf{v}^\top \widehat{\mathbf{z}})^{2k}]}{(2k)!} \mathbb{E}[\widehat{R}^{2k}] - \sum_{k=0}^{\infty} \frac{\|\mathbf{z}\|^{2k} (-1)^k \mathbb{E}[(\mathbf{v}^\top \widehat{\mathbf{z}})^{2k}]}{(2k)!} \mathbb{E}[R^{2k}], \tag{8.21}$$

where: $\widehat{\mathbf{z}} = \frac{\mathbf{z}}{\|\mathbf{z}\|}$. Denote: $A(k, n) = \mathbb{E}[(\mathbf{v}^\top \widehat{\mathbf{z}})^k]$ (note that $\mathbf{v}, \widehat{\mathbf{z}} \in \mathbb{R}^n$). It is easy to see that for odd k we have: $A(n, k) = 0$. We obtain:

$$F = \sum_{k=0}^{\infty} \frac{\|\mathbf{z}\|^{2k} (-1)^k A(2k, n)}{(2k)!} (\mathbb{E}[\widehat{R}^{2k}] - \mathbb{E}[R^{2k}]). \tag{8.22}$$

The following technical fact will be useful:

Lemma 43. *Expression $A(k, n)$ is given by the following formula:*

$$A(k, n) = \frac{1}{\int_0^\pi \sin^{n-2}(\theta) d\theta} \int_0^\pi \cos^k(\theta) \sin^{n-2}(\theta) d\theta, \tag{8.23}$$

which can be equivalently rewritten (by computing the integrals explicitly) as:

$$\begin{aligned} A(2k, n) = \frac{(n-2)(n-4) \cdot ... \cdot \delta(n=2)}{(n-3)(n-5) \cdot ... \cdot \gamma(n=2)} \cdot \frac{(2k-1)!!}{(n-1)(n+1)...(n+2k-3)} \cdot \\ \frac{(n+2k-3)(n+2k-5)... \cdot \gamma(n=2)}{(n+2k-2)(n+2k-4)... \cdot \delta(n=2)}, \end{aligned} \tag{8.24}$$

where: $\delta(n=2) = 2$ if $n = 2$ and $\delta(n=2) = 1$ otherwise and: $\gamma(n=2) = 1$ if $n = 2$ and $\gamma(n=2) = 2$ otherwise.

In particular, the following is true:

$$|A(2k, n)| \leq \frac{(2k-1)!!}{(n-1)(n+1) \cdot ... \cdot (n+2k-3)}. \tag{8.25}$$

We will use that later. Note that $\mathbf{v}_i^\top \mathbf{v}_j \sim \mathbf{v}^\top \widehat{\mathbf{z}}$. Therefore we obtain:

$$F = \sum_{k=0}^{\infty} \frac{\|\mathbf{z}\|^{2k} (-1)^k A(2k, n)}{(2k)!} \alpha_k, \tag{8.26}$$

where

$$\alpha_k = \sum_{i=1}^{k} \binom{k}{i} \mathbb{E}[(R^2)^{k-i}\lambda^i], \tag{8.27}$$

and $\lambda = \sum_{i,j \in \{1,...,m\}} n_i n_j R_i R_j \mathbf{v}_i^\top \mathbf{v}_j$.

Note that $\mathbb{E}[(R^2)^{k-1}\lambda] = 0$ since $\mathbb{E}[(\mathbf{v}_i^\top \mathbf{v}_j)] = 0$ and furthermore, directions $\mathbf{v}_1, ..., \mathbf{v}_m$ are chosen independently from lengths $R_1, ..., R_m$. Therefore we have:

$$\alpha_k = \binom{k}{2} \mathbb{E}[(R^2)^{k-2}\lambda^2] + \beta_k, \tag{8.28}$$

where:

$$\beta_k = \sum_{i=3}^{k} \binom{k}{i} \mathbb{E}[(R^2)^{k-i}\lambda^i]. \tag{8.29}$$

Now let us focus on a single term $\rho = \mathbb{E}[(R^2)^{k-l}\lambda^l]$ for some fixed $l \geq 3$.

Note that the following is true:

$$\rho \leq \mathbb{E}[(R^2)^{k-l}(\sum_{i,j \in \{1,...,m\}} n_i n_j R_i R_j)^l] \cdot \max_{i_1,j_1,...,i_l,j_l} \mathbb{E}[|\mathbf{v}_{i_1}^\top \mathbf{v}_{j_1}| \cdot ... \cdot |\mathbf{v}_{i_l}^\top \mathbf{v}_{j_l}|], \tag{8.30}$$

where the maximum is taken over $i_1, j_1, ..., i_l, j_l \in \{1, ..., m\}$ such that $i_s \neq j_s$ for $s = 1, ..., l$.

Note first that:

$$\mathbb{E}[(R^2)^{k-l}(\sum_{i,j \in \{1,...,m\}} n_i n_j R_i R_j)^l] \leq$$

$$\mathbb{E}[(R^2)^{k-l}(\sum_{i,j \in \{1,...,m\}} \frac{(n_i R_i)^2 + (n_j R_j)^2}{2})^l] \leq \mathbb{E}[(R^2)^{k-l}(m-1)^l (R^2)^l] \leq (m-1)^l \mathbb{E}[R^{2k}]. \tag{8.31}$$

Let us focus now on the expression $\max_{i_1,j_1,...,i_l,j_l} \mathbb{E}[|\mathbf{v}_{i_1}^\top \mathbf{v}_{j_1}| \cdot ... \cdot |\mathbf{v}_{i_l}^\top \mathbf{v}_{j_l}|]$.

We will prove the following upper bound on $\max_{i_1,j_1,...,i_l,j_l} \mathbb{E}[|\mathbf{v}_{i_1}^\top \mathbf{v}_{j_1}| \cdot ... \cdot |\mathbf{v}_{i_l}^\top \mathbf{v}_{j_l}|]$.

Lemma 44. *The following is true:*

$$\mathbb{E}[|\mathbf{v}_{i_1}^\top \mathbf{v}_{j_1}| \cdot ... \cdot |\mathbf{v}_{i_l}^\top \mathbf{v}_{j_l}|] \leq (\frac{\log(n)}{\sqrt{n - \sqrt{n}\log(n)}})^l + l(e^{-\frac{\log^2(n)}{4}} + e^{-\frac{\log^2(n)}{2}}). \tag{8.32}$$

Proof. Note that from the isotropicity of Gaussian vectors we can conclude that each single $|\mathbf{v}_{i_s}^\top \mathbf{v}_{j_s}|$ is distributed as: $\frac{g_1}{\sqrt{g_1^2 + ... + g_n^2}}$, where $g_1, ..., g_n$ stand for n independent copies of a random variable taken from $\mathcal{N}(0,1)$. Note that $g_1^2 + ... + g_n^2$ is taken from the χ_n^2- distribution. Using the well-known bounds for the tails of χ_n^2- distributions, we get: $\mathbb{P}[g_1^2 + ... + g_n^2 - n \leq a] \leq e^{-\frac{a^2}{4n}}$. Note also that $\mathbb{P}[|g_1| > x] \leq 2\frac{e^{-\frac{x^2}{2}}}{x\sqrt{2\pi}}$. Thus, by taking: $a = \sqrt{n}\log(n)$, $x = \log(n)$ and applying union bound, we conclude that with probability at least $1 - e^{-\frac{\log^2(n)}{4}} - e^{-\frac{\log^2(n)}{2}}$ a fixed random variable $|\mathbf{v}_{i_s}^\top \mathbf{v}_{j_s}|$ satisfies: $|\mathbf{v}_{i_s}^\top \mathbf{v}_{j_s}| \leq \frac{\log(n)}{\sqrt{n-\sqrt{n}\log(n)}}$. Thus, by the union bound we conclude that for any fixed $i_1, j_1, ..., i_l, j_l$ random variable $|\mathbf{v}_{i_1}^\top \mathbf{v}_{j_1}| \cdot ... \cdot |\mathbf{v}_{i_l}^\top \mathbf{v}_{j_l}|$ satisfies: $|\mathbf{v}_{i_1}^\top \mathbf{v}_{j_1}| \cdot ... \cdot |\mathbf{v}_{i_l}^\top \mathbf{v}_{j_l}| \leq (\frac{\log(n)}{\sqrt{n-\sqrt{n}\log(n)}})^l$ with probability at least $1 - l(e^{-\frac{\log^2(n)}{4}} + e^{-\frac{\log^2(n)}{2}})$. Since $|\mathbf{v}_{i_1}^\top \mathbf{v}_{j_1}| \cdot ... \cdot |\mathbf{v}_{i_l}^\top \mathbf{v}_{j_l}|$ is upper bounded by one, we conclude that:

$$\mathbb{E}[|\mathbf{v}_{i_1}^\top \mathbf{v}_{j_1}| \cdot ... \cdot |\mathbf{v}_{i_l}^\top \mathbf{v}_{j_l}|] \leq (\frac{\log(n)}{\sqrt{n-\sqrt{n}\log(n)}})^l + l(e^{-\frac{\log^2(n)}{4}} + e^{-\frac{\log^2(n)}{2}}). \tag{8.33}$$

□

Using Lemma 44, we can conclude that:

$$\rho \leq (m-1)^l \mathbb{E}[R^{2k}] \cdot \left((\frac{\log(n)}{\sqrt{n-\sqrt{n}\log(n)}})^l + l(e^{-\frac{\log^2(n)}{4}} + e^{-\frac{\log^2(n)}{2}}) \right) \tag{8.34}$$

Therefore we have:

$$\beta_k \leq \sum_{i=3}^{k} \binom{k}{i} (m-1)^i \mathbb{E}[R^{2k}] \cdot \left((\frac{\log(n)}{\sqrt{n-\sqrt{n}\log(n)}})^i + i(e^{-\frac{\log^2(n)}{4}} + e^{-\frac{\log^2(n)}{2}}) \right) \tag{8.35}$$

Thus we can conclude that for n large enough:

$$\beta_k \leq k(2m)^k (\frac{2\log^3(n)}{n^{\frac{3}{2}}} + 2ke^{-\frac{\log^2(n)}{4}}) \mathbb{E}[R^{2k}]. \tag{8.36}$$

Thus we get:

$$F = \sum_{k=0}^{\infty} \frac{\|\mathbf{z}\|^{2k}(-1)^k A(2k,n)}{(2k)!} \binom{k}{2} \mathbb{E}[(R^2)^{k-2}\lambda^2] + \Gamma, \tag{8.37}$$

where:

$$\begin{aligned} |\Gamma| &\leq \sum_{k=0}^{\infty} \frac{\|\mathbf{z}\|^{2k} A(2k,n)}{(2k)!} k(2m)^k (\frac{2\log^3(n)}{n^{\frac{3}{2}}} + 2ke^{-\frac{\log^2(n)}{4}}) \mathbb{E}[R^{2k}] \\ &= \frac{2\log^3(n)}{n^{\frac{3}{2}}} A + 2e^{-\frac{\log^2(n)}{4}} B, \end{aligned} \tag{8.38}$$

B satisfies: $B = Ak$ and A is given as:

$$A = \sum_{k=0}^{\infty} \frac{\|\mathbf{z}\|^{2k} \mathbb{E}[R^{2k}] A(2k,n)}{(2k)!} k(2m)^k. \tag{8.39}$$

Now note that since data is taken from the ball of radius r, we have: $\|\mathbf{z}\| \leq 2r$. Furthermore, from the smoothness of the considered class of RBF kernels, we obtain:

$$\mathbb{E}[R_{2k}] \leq \max_{i=1,...,m} n_i^{2k} m^k (n-1)(n+1) \cdot .. \cdot (n+2k-3) f^k(k) k!. \tag{8.40}$$

Denote $h = \mathrm{argmax}_{i=1,\ldots,m} n_i$. Note that $(2k-1)!! = \frac{(2k)!}{2^k k!}$. Thus, by applying the above upper bound on $A(2k,n)$, we obtain:

$$A, B \leq \sum_{k=0}^{\infty} (m^2(2r)^2 n_h^2 f(k))^k k^2 \leq \sum_{k=0}^{\infty} (4m^2(2r)^2 n_h^2 f(k))^k. \tag{8.41}$$

Now, notice that for a given n_h, m and r, the above upper bound is of the form $\sum_{k=0}^{\infty} q_k^k$, where $q_k \to 0$ as $k \to \infty$ (since $f_k \to 0$ as $k \to \infty$ for smooth RBF kernels). Then we can conclude that the contribution of the terms Γ to the expression $\tau = \frac{\Lambda(j_1,\ldots,j_m)}{j_1! \cdot \ldots \cdot j_m!}$ from the LHS of 8.16 is of the order $O(\frac{1}{n^{\frac{3}{2}}})$ (notice that every n_i satisfies: $n_i \leq j_i$). Now let us consider $F - \Gamma$. We have:

$$F - \Gamma = \sum_{k=0}^{\infty} \frac{\|\mathbf{z}\|^{2k}(-1)^k A(2k,n)}{(2k)!} \binom{k}{2} \mathbb{E}[(R^2)^{k-2}\lambda^2] \tag{8.42}$$

By the same analysis as before we conclude that

$$F - \Gamma = \rho + o_n(\frac{1}{n}), \tag{8.43}$$

where

$$\rho = \sum_{k=0}^{\infty} \frac{\|\mathbf{z}\|^{2k}(-1)^k A(2k,n)}{(2k)!} \binom{k}{2} \mathbb{E}[(\widehat{R}^2)^{k-2}\lambda^2] \tag{8.44}$$

and futhermore: $\rho = \frac{1}{n}\tilde{\rho} + o_n(\frac{1}{n})$, where

$$\tilde{\rho} = W \sum_{k=0}^{\infty} \frac{\|\mathbf{z}\|^{2k}(-1)^k A(2k,n)}{(2k)!} \binom{k}{2} \mathbb{E}[(\widehat{R}^{2k})] \tag{8.45}$$

and W is some constant (that depends on m). Thus to show Inequality 8.16 for n large enough, it suffices to show that $\tilde{\rho} > 0$ and that ρ does not depend on n (even though n is explicitly embedded in the formula on $\tilde{\rho}$ via $A(2k,n)$). This follows from the straightforward extension (for different n_is) of the fact that for $n_1 = \ldots = n_m$, $\tilde{\rho}$ can be rewritten in terms of the positive definite function ϕ describing an RBF kernel under consideration, namely:

$$\tilde{\rho} = \frac{1}{8n}((\|n_1\mathbf{z}\|^2 \frac{d^2(\phi^m(x))}{dx^2})_{|x=\|n_1\mathbf{z}\|} - (\|n_1\mathbf{z}\| \frac{d(\phi^m(x))}{dx})_{|x=\|n_1\mathbf{z}\|}). \tag{8.46}$$

That the above expression is positive is implied by the fact that every positive definite function ϕ (not parametrized by n) can be rewritten as $\phi(r) = \sigma(r^2)$, where σ is completely monotone on $[0, +\infty]$ and from the basic properties of completely monotone functions (see: [97]). That completes the proof of the theorem.

8.B Proof of Theorem 39

The theorem follows straightforwardly from Theorem 36, Theorem 38 and the fact that $\mathrm{rank}(\widehat{\mathbf{K}}) \leq m$ (since m-dimensional random feature maps are used for kernel approximation).

Part V

Conclusions

Chapter 9

Conclusions

Building models to understand, predict, and control dynamical systems has been a field of study for many years, resulting in a large and diverse array of distinct models. In this thesis we focused on the idea of developing novel hybrid models which unify these many disparate approaches. In particular we focused on two types of models: Bayes Filters and Recurrent Neural Networks. We showed that these two model classes had complementary strengths and weakness, and that combining them offered significant advantages. The work done in this thesis advances the field of dynamical system modelling in a number of ways:

We began by unifying the various various Method-of-Moments learning algorithms for Bayes Filters under a single learning framework called two-stage regression. This unification greatly improves our understanding of this class of algorithms, and the models to which they are applicable. As a consequence of this formulation it allows practitioners to directly apply the rich literature on supervised learning methods to incorporate many types of prior knowledge about problem structure.

We then leveraged this framework to develop two new hybrid models which unify the advantages of Bayes Filters and Recurrent Neural Networks in a single model. In our first model, Predictive State Recurrent Neural Networks (PSRNNs), we developed a Bayes Filter with a predictive state which can also be initialized via Method-of-Moments and further refined via Gradient Descent. In our second model, Hilbert Space Embedding of Hidden Quantum Markov Models (HSE-HQMMs), we combine machine learning models with ideas from quantum mechanics. We showed that by manipulating uncertainty using density matrices and unitary operators we can obtain a nonparametric method for maintaining a probability distribution over continuous-valued features.

Finally we showed how these new ideas can be implemented at scale by solving two implementation issues. Firstly, naive implementations of two-stage regression scale poorly due to the cubic relationship between the size of the state and the number of model parameters. We showed how to avoid this problem by using tensor sketches, combined with tensor power iteration, to efficiently learn PSRNNs from data. Secondly techniques which use Random Features to approximate kernel embeddings often require a large number of such features. We show that orthogonal random features can be used in place of conventional random features to learn smaller, faster PSRNNs.

Together these contributions show that hybrid models which unify Bayes Filters with Recurrent Neural Networks offer many exciting new possibilities. In essence they allow us to combine the best parts of statistics, optimization, and machine learning into a single model which offers statistical insight while often outperforming the best discriminative approaches on real world applications. Furthermore we believe that we have only begun to scratch the surface of what is possible with hybrid models, and we hope to encourage future researchers to further explore this area.

Bibliography

[1] Alessandro Achille and Stefano Soatto. On the emergence of invariance and disentangling in deep representations. *CoRR*, abs/1706.01350, 2017. URL `http://arxiv.org/abs/1706.01350`. 8.4

[2] N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *STOC*, 2006. 8.4

[3] A. El Alaoui and M. Mahoney. Fast randomized kernel ridge regression with statistical guarantees. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 775–783, 2015. 8.2

[4] E Alpaydin and Fevzi Alimoglu. Pen-based recognition of handwritten digits data set. *University of California, Irvine, Machine Learning Repository. Irvine: University of California*, 1998. 5.6, 8.3.1

[5] Anima Anandkumar, Rong Ge, Daniel J. Hsu, Sham M. Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *CoRR*, abs/1210.7559, 2012. 2.4.6, 2.4.6

[6] Animashree Anandkumar, Sham M Kakade, Dean P Foster, Yi-Kai Liu, and Daniel Hsu. Two svds suffice: Spectral decompositions for probabilistic topic modeling and latent dirichlet allocation. Technical report, 2012. 2.4.6

[7] Animashree Anandkumar, Rong Ge, and Majid Janzamin. Guaranteed non-orthogonal tensor decomposition via alternating rank-1 updates. *CoRR*, abs/1402.5180, 2014. URL `http://arxiv.org/abs/1402.5180`. 2.4.6

[8] Martín Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1120–1128, 2016. URL `http://jmlr.org/proceedings/papers/v48/arjovsky16.html`. 8.4

[9] M Arvandi, S Wu, and A Sadeghian. On the use of recurrent neural networks to design symmetric ciphers. *IEEE computational intelligence magazine*, 3(2):42–53, 2008. 2.1

[10] H. Avron, M. Kapralov, C. Musco, C. Musco, A. Velingker, and A. Zandieh. Random fourier features for kernel ridge regression: Approximation bounds and statistical guarantees. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 253–262, 2017. URL `http://proceedings.mlr.press/v70/avron17a.html`. 8.2, 8.2, 8.A

[11] F. Bach. Sharp analysis of low-rank kernel matrix approximations. In *COLT 2013 - The 26th Annual Conference on Learning Theory, June 12-14, 2013, Princeton University, NJ, USA*, pages 185–209, 2013. URL `http://jmlr.org/proceedings/papers/v30/Bach13.html`. 8.2

[12] Borja Balle, William Hamilton, and Joelle Pineau. Methods of moments for learning stochastic languages: Unified presentation and empirical comparison. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1386–1394. JMLR Workshop and Conference Proceedings, 2014. URL `http://jmlr.org/proceedings/papers/v32/balle14.pdf`. 3.6

[13] Thanasis G Barbounis, John B Theocharis, Minas C Alexiadis, and Petros S Dokopoulos. Long-term wind speed and power forecasting using local recurrent neural network models. *IEEE Transactions on Energy Conversion*, 21(1):273–284, 2006. 2.1

[14] Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966. 2.2.1

[15] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):pp. 164–171, 1970. ISSN 00034851. URL `http://www.jstor.org/stable/2239727`. 2.2.1

[16] David Belanger and Sham Kakade. A linear dynamical system model for text. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 833–842, Lille, France, 07–09 Jul 2015. PMLR. 5.7

[17] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009. 5.1.2

[18] M. Bojarski, A. Choromanska, K. Choromanski, F. Fagan, C. Gouy-Pailler, A. Morvan, N. Sakr, T. Sarlos, and J. Atif. Structured adaptive and random spinners for fast machine learning computations. In *AISTATS*, 2017. 8.4

[19] Byron Boots. Learning stable linear dynamical systems. *Online]. Avail.: https://www. ml. cmu. edu/research/dap-papers/dap_boots. pdf*, 2009. 5.7

[20] Byron Boots. *Spectral Approaches to Learning Predictive Representations*. PhD thesis, Carnegie Mellon University, December 2012. 3.3.2, 3.6

[21] Byron Boots and Dieter Fox. Learning dynamic policies from demonstration. *NIPS Workshop on Advances in Machine Learning for Sensorimotor Control*, 2013. 4.4.1

[22] Byron Boots and Geoffrey Gordon. An online spectral learning algorithm for partially observable nonlinear dynamical systems. In *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI)*, 2011. 2.2

[23] Byron Boots and Geoffrey Gordon. An online spectral learning algorithm for partially observable nonlinear dynamical systems. In *Proceedings of the 25th National Conference on Artificial Intelligence (AAAI-2011)*, 2011. 3.6

[24] Byron Boots and Geoffrey Gordon. Two-manifold problems with applications to nonlinear

system identification. In *Proc. 29th Intl. Conf. on Machine Learning (ICML)*, 2012. 3.6

[25] Byron Boots and Geoffrey J. Gordon. Predictive state temporal difference learning. *CoRR*, abs/1011.0041, 2010. 3.6, 3

[26] Byron Boots, Arthur Gretton, and Geoffrey J. Gordon. Hilbert space embeddings of PSRs. *NIPS Workshop on Spectral Algorithms for Latent Variable Models*, 2012. 6.9

[27] Byron Boots, Geoffrey J. Gordon, and Arthur Gretton. Hilbert space embeddings of predictive state representations. *CoRR*, abs/1309.6819, 2013. 5.6, 6.C.4, 7.3.3, 8.3.1

[28] Byron Boots, Arthur Gretton, and Geoffrey J. Gordon. Hilbert Space Embeddings of Predictive State Representations. In *Proc. 29th Intl. Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2013. **??**, 3.3.3, 3.4, 3.6

[29] Byron Boots, Arthur Gretton, and Geoffrey J. Gordon. Hilbert Space Embeddings of Predictive State Representations. In *UAI*, 2013. 4.2.1, 4.3.1, 4.4, 4.4.1, 4.4.1, 4.4.2, 4.5.1, 4.5.4

[30] Michael Bowling, Peter McCracken, Michael James, James Neufeld, and Dana Wilkinson. Learning predictive state representations using non-blind policies. In *ICML*, 2006. 3

[31] Gilles Celeux and Jean Diebolt. A stochastic approximation type em algorithm for the mixture problem. *Stochastics: An International Journal of Probability and Stochastic Processes*, 41(1-2):119–134, 1992. 2.4.3

[32] HaiYang Chao, YongCan Cao, and YangQuan Chen. Autopilots for small unmanned aerial vehicles: a survey. *International Journal of Control, Automation and Systems*, 8(1):36–44, 2010. 2.1

[33] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. 5.6

[34] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. 2.3.3

[35] A. Choromanska, K. Choromanski, M. Bojarski, T. Jebara, S. Kumar, and Y. LeCun. Binary embeddings with structured hashed projections. In *ICML*, 2016. 8.4

[36] K. Choromanski and V. Sindhwani. Recycling randomness with structure for sublinear time kernel expansions. In *ICML*, 2016. 8.4

[37] K. Choromanski, M. Rowland, and A. Weller. The unreasonable effectiveness of structured random orthogonal embeddings. In *to appear in NIPS*, volume arXiv abs/1703.00864, 2017. 8, 8.1, 8.4

[38] Krzysztof Choromanski, Carlton Downey, and Byron Boots. Initialization matters: Orthogonal predictive state recurrent neural networks. 2018. 1.2

[39] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (5):564–575, 2003. 2.1

[40] Simon Cooper and Hugh Durrant-Whyte. A kalman filter model for gps navigation of land vehicles. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, volume 1, pages 157–163. IEEE, 1994. 2.1

[41] Albert T. Corbett and John R. Anderson. Knowledge tracing: Modelling the acquisition of procedural knowledge. *User Model. User-Adapt. Interact.*, 4(4), 1995. 3.5.1

[42] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proc. VLDB Endow.*, 1(2):1530–1541, August 2008. ISSN 2150-8097. doi: 10.14778/1454159.1454225. URL `http://dx.doi.org/10.14778/1454159.1454225`. 7.1

[43] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. 2.4.3

[44] Carlton Downey, Ahmed Hefny, Boyue Li, Byron Boots, and Geoffrey J. Gordon. Predictive state recurrent neural networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2017. 1.2, 6.8, 6.9, 6.10, 6.C.4, 7.3.3, 8.3.1

[45] Thi V Duong, Hung Hai Bui, Dinh Q Phung, and Svetha Venkatesh. Activity recognition and abnormality detection with the switching hidden semi-markov model. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 838–845. IEEE, 2005. 2.1

[46] Fevzi. Alimoglu E. Alpaydin. Pen-Based Recognition of Handwritten Digits Data Set. https://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits. 5.6

[47] Jeffrey L. Elman. Finding structure in time. *COGNITIVE SCIENCE*, 14(2):179–211, 1990. 2.3.1, 5.6

[48] Nick Foti, Jason Xu, Dillon Laird, and Emily Fox. Stochastic variational inference for hidden markov models. In *Advances in neural information processing systems*, pages 3599–3607, 2014. 2.4

[49] Sylvia Frühwirth-Schnatter. Markov chain monte carlo estimation of classical and dynamic switching and mixture models. *Journal of the American Statistical Association*, 96(453): 194–209, 2001. 2.4

[50] Kenji Fukumizu, Le Song, and Arthur Gretton. Kernel bayes rule: Bayesian inference with positive definite kernels. *Journal of Machine Learning Research*, 14(1):3753–3783, 2013. **??**, 3.3.3, 3.3.3, 4.3.1

[51] Surya Ganguli, Dongsung Huh, and Haim Sompolinsky. Memory traces in dynamical systems. *105(48):18970– 18975, 2008. doi: 10.1073/pnas.0804451105.*, 2008. 8.4

[52] Alborz Geramifard, Robert H Klein, Christoph Dann, William Dabney, and Jonathan P How. RLPy: The Reinforcement Learning Library for Education and Research. http://acl.mit.edu/RLPy, April 2013. 4.5.3

[53] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial*

Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics, 2010. 5.6

[54] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013. 2.1

[55] Ligdamis Gutiérrez, Jesús Ibañez, Guillermo Cortés, Javier Ramírez, Carmen Benítez, Virginia Tenorio, and Álvarez Isaac. Volcano-seismic signal detection and classification processing using hidden markov models. application to san cristóbal volcano, nicaragua. In *2009 IEEE International Geoscience and Remote Sensing Symposium*, volume 4, pages IV–522. IEEE, 2009. 2.1

[56] Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop kf: Learning discriminative deterministic state estimators. In *Advances in Neural Information Processing Systems*, pages 4376–4384, 2016. 5.7

[57] Lars Peter Hansen. Large sample properties of generalized method of moments estimators. *Econometrica: Journal of the Econometric Society*, pages 1029–1054, 1982. 2.4.4

[58] Md Rafiul Hassan and Baikunth Nath. Stock market forecasting using hidden markov model: a new approach. In *5th International Conference on Intelligent Systems Design and Applications (ISDA'05)*, pages 192–196. IEEE, 2005. 2.1

[59] Ahmed Hefny, Carlton Downey, and Geoffrey J. Gordon. Supervised learning for dynamical system learning. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1963–1971, 2015. 1.2, 2.2, 4.7, 5.6, 6.6

[60] Ahmed Hefny, Carlton Downey, and Geoffrey Gordon. An efficient, expressive and local minima-free method for learning controlled dynamical systems. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 1.2

[61] Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2034–2042, 2016. URL `http://jmlr.org/proceedings/papers/v48/henaff16.html`. 8.4

[62] A. Hinrichs and J. Vybíral. Johnson-Lindenstrauss lemma for circulant matrices. *Random Structures & Algorithms*, 39(3):391–398, 2011. 8.4

[63] Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Studies in Applied Mathematics*, 6(1-4):164–189, 1927. 2.4.6

[64] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. 2.3.2, 5.1.2, 5.5

[65] Daniel Hsu, Sham M. Kakade, and Tong Zhang. A spectral algorithm for learning hidden markov models. 2009. **??**, 3.3.1, 3.3.1, 3.3.1, 3, 3.3.1, 3.5.1, 3.6

[66] Daniel Hsu, Sham M. Kakade, and Tong Zhang. 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, 2012. URL `http://www.jmlr.org/proceedings/papers/v23/hsu12/hsu12.pdf`. 3.B

[67] Daniel Hsu, Sham M Kakade, and Tong Zhang. Tail inequalities for sums of random matrices that depend on the intrinsic dimension. *Electronic Communications in Probability*, 17(14):1–13, 2012. 7

[68] Daniel J. Hsu, Sham M. Kakade, and Tong Zhang. A spectral algorithm for learning hidden markov models. *CoRR*, abs/0811.4413, 2008. 2.2, 2.4.6, 3.1, 3.4

[69] Herbert Jaeger. Observable operator models for discrete stochastic time series. *Neural Computation*, 12(6):1371–1398, 2000. 2.2.3, 2.2.3

[70] R. E. Kalman. A new approach to linear filtering and prediction problems. *ASME Journal of Basic Engineering*, 1960. 2.2.2, 2.2.2, 5.6

[71] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2.4.2

[72] Jonathan Ko and Dieter Fox. Learning gp-bayesfilters via gaussian process latent variable models. *Autonomous Robots*, 30(1):3–23, 2011. 5.7

[73] Kenneth R. Koedinger, R. S. J. Baker, K. Cunningham, A. Skogsholm, B. Leber, and John Stamper. A data repository for the EDM community: The PSLC DataShop. *Handbook of Educational Data Mining*, pages 43–55, 2010. 3.5.1

[74] Jean Kossaifi, Zachary C Lipton, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. Tensor regression networks. *arXiv preprint arXiv:1707.08308*, 2017. 2.4.6, 5.7

[75] Joseph B Kruskal. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear algebra and its applications*, 18 (2):95–138, 1977. 2.4.6

[76] John Langford, Ruslan Salakhutdinov, and Tong Zhang. 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, 2009. doi: 10.1145/1553374.1553451. URL `http://doi.acm.org/10.1145/1553374.1553451`. 3.6

[77] Matthew Leifer and David Poulin. Quantum graphical models and belief propagation. *Ann. Phys.*, 323:1899, 2008. 6.9

[78] Lennart Ljung. *System identification*. Wiley Online Library, 1999. 5.1.2, 5.1.2, 5.6

[79] Fabien Lotte, Marco Congedo, Anatole Lécuyer, Fabrice Lamarche, and Bruno Arnaldi. A review of classification algorithms for eeg-based brain–computer interfaces. *Journal of neural engineering*, 4(2):R1, 2007. 2.1

[80] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993. 5.6, 6.8, 7.3.3

[81] James Martens. Learning the linear dynamical system with asos. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 743–750, 2010. 5.7

[82] Geoffrey McLachlan and Thriyambakam Krishnan. *The EM algorithm and extensions*, volume 382. John Wiley & Sons, 2007. 2.4.3

[83] Alex Monras, Almut Beige, and Karoline Wiesner. Hidden quantum Markov models and non-adaptive read-out of many-body states. *arXiv preprint arXiv:1002.2337*, 2010. 6.9

[84] Elizbar A Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9 (1):141–142, 1964. 6.5.3

[85] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983. 2.4.2

[86] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002. 6.3

[87] S.M. Pandit and S.M. Wu. *Time series and system analysis, with applications*. Wiley, 1983. ISBN 9780471868866. URL `http://books.google.com/books?id=v4sQAQAAIAAJ`. 3.6

[88] Luca Pasa, Alberto Testolin, and Alessandro Sperduti. A hmm-based pre-training approach for sequential data. In *22th European Symposium on Artificial Neural Networks, ESANN 2014, Bruges, Belgium, April 23-25, 2014*, 2014. URL `http://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2014-166.pdf`. 5.7

[89] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York, NY, USA, 2000. ISBN 0-521-77362-8. 3.2

[90] Ninh Pham and Rasmus Pagh. 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, 2013. ACM. ISBN 978-1-4503-2174-7. doi: 10.1145/2487575.2487591. URL `http://doi.acm.org/10.1145/2487575.2487591`. 7.1

[91] Gianluca Pollastri, Darisz Przybylski, Burkhard Rost, and Pierre Baldi. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics*, 47(2):228–235, 2002. 2.1

[92] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008. (document), 2.2.5, 4.1, 5.6, 6.6, 8, 8.1, 8.4

[93] Matthew Rosencrantz and Geoff Gordon. Learning low dimensional predictive representations. In *ICML '04: Twenty-first international conference on Machine learning*, pages 695–702, 2004. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.5163`. 2.2.4

[94] Sam Roweis and Zoubin Ghahramani. A unifying review of linear gaussian models. *Neural Comput.*, 11(2):305–345, February 1999. ISSN 0899-7667. doi: 10.1162/089976699300016674. URL `http://dx.doi.org/10.1162/089976699300016674`. 2.2

[95] Tobias Rydén et al. Em versus markov chain monte carlo for estimation of hidden markov

models: A computational perspective. *Bayesian Analysis*, 3(4):659–688, 2008. 2.4

[96] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2013. URL `http://arxiv.org/abs/1312.6120`. 8.4

[97] I. Schoenberg. Metric Spaces and Completely Monotone Functions. *The Annals of Mathematics*, 39(4):811–841, 1938. 8.A

[98] Maria Schuld and Nathan Killoran. Quantum machine learning in feature Hilbert spaces. *arXiv preprint arXiv:1803.07128*, 2018. 6.9

[99] Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4(1):120–131, 2017. 2.1

[100] Amirreza Shaban, Mehrdad Farajtabar, Bo Xie, Le Song, and Byron Boots. Learning latent variable models by improving spectral solutions with exterior point methods. In *Proceedings of The International Conference on Uncertainty in Artificial Intelligence (UAI-2015)*, 2015. 2.2

[101] Sajid Siddiqi, Byron Boots, and Geoffrey J. Gordon. Reduced-rank hidden Markov models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-2010)*, 2010. 3.5.1, 3.6

[102] Satinder Singh, Michael R. James, and Matthew R. Rudary. 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, 2004. AUAI Press. ISBN 0-9749039-0-6. 2.2.4

[103] Alex Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. A hilbert space embedding for distributions. In *International Conference on Algorithmic Learning Theory*, pages 13–31. Springer, 2007. 2.2.5, 6.5.1, 6.5.1, 6.9

[104] L. Song, B. Boots, S. M. Siddiqi, G. J. Gordon, and A. J. Smola. Hilbert space embeddings of hidden Markov models. In *Proc. 27th Intl. Conf. on Machine Learning (ICML)*, 2010. 3.6, 5.1.2, 6.9

[105] Le Song, Jonathan Huang, Alex Smola, and Kenji Fukumizu. Hilbert space embeddings of conditional distributions with applications to dynamical systems. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 961–968. ACM, 2009. 3.3.3, 13, 3.B, 4.4, 6.9

[106] Le Song, Kenji Fukumizu, and Arthur Gretton. Kernel embeddings of conditional distributions: A unified kernel framework for nonparametric inference in graphical models. *IEEE Signal Processing Magazine*, 30(4):98–111, 2013. 6.5.1, 6.5.2, 6.5.2, 6.5.4, 6.5.4, 6.9

[107] Siddarth Srinivasan, Carlton Downey, and Byron Boots. Learning and inference in hilbert space with quantum graphical models. In *Advances in Neural Information Processing Systems*, pages 10359–10368, 2018. 1.2

[108] Siddarth Srinivasan, Geoffrey J. Gordon, and Byron Boots. Learning hidden quantum Markov models. In *Proceedings of the 21st International Conference on Artificial Intelli-*

gence and Statistics, 2018. 6.4, 6.4, 6.4, 6.5.2, 6.6, 6.6, 6.9

[109] J.H. Stock and M.W. Watson. *Introduction to Econometrics*. Addison-Wesley series in economics. Addison-Wesley, 2011. ISBN 9780138009007. URL `http://books.google.com/books?id=prLxRQAACAAJ`. 3.2

[110] Joel A. Tropp. User-friendly tools for random matrices: An introduction. NIPS Tutorial, 2012. 3.A.1

[111] Joel A. Tropp. An introduction to matrix concentration inequalities. *Found. Trends Mach. Learn.*, 8(1-2):1–230, May 2015. ISSN 1935-8237. doi: 10.1561/2200000048. URL `http://dx.doi.org/10.1561/2200000048`. 26

[112] P. van Overschee and L.R. de Moor. *Subspace identification for linear systems: theory, implementation, applications*. Kluwer Academic Publishers, 1996. 2.4.5, **??**, 3.3.2, 4.2.1, 4.3.2, 4.6.2, 4.6.2

[113] Peter Van Overschee and Bart De Moor. N4sid: numerical algorithms for state space subspace system identification. In *Proc. of the World Congress of the International Federation of Automatic Control, IFAC*, volume 7, pages 361–364, 1993. 2.2

[114] Peter Van Overschee and Bart De Moor. N4sid: Subspace algorithms for the identification of combined deterministic-stochastic systems. *Automatica*, 30(1):75–93, January 1994. ISSN 0005-1098. doi: 10.1016/0005-1098(94)90230-5. URL `http://dx.doi.org/10.1016/0005-1098(94)90230-5`. 3.1, 5.7

[115] J. Vybíral. A variant of the Johnson-Lindenstrauss lemma for circulant matrices. *Journal of Functional Analysis*, 260(4):1096–1105, 2011. 8.4

[116] Y. Wang, H.-Y. Tung, A. J. Smola, and A. Anandkumar. Fast and guaranteed tensor decomposition via sketching. In *NIPS*, 2015. 7.2.2, 7.2.2

[117] Larry Wasserman. *All of Nonparametric Statistics (Springer Texts in Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387251456. 6.5.3

[118] Geoffrey S Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372, 1964. 6.5.3

[119] Paul J Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. 2.5.1

[120] O. White, D. Lee, and H. Sompolinsky. Short-term memory in orthogonal neural networks. *Physical Review Letters, 92, 2004. ISSN 14.*, 2004. 8.4

[121] Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan Salakhutdinov. On multiplicative integration with recurrent neural networks. *CoRR*, abs/1606.06630, 2016. URL `http://arxiv.org/abs/1606.06630`. 5.5

[122] Chen-Hsiang Yeang. A probabilistic graphical model of quantum systems. In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, pages 155–162. IEEE, 2010. 6.9

[123] F. Yu, A. Suresh, K. Choromanski, D. Holtmann-Rice, and S. Kumar. Orthogonal random features. In *NIPS*, pages 1975–1983, 2016. 8, 8.2, 8.4

[124] H. Zhang and L. Cheng. New bounds for circulant Johnson-Lindenstrauss embeddings. *CoRR*, abs/1308.6339, 2013. 8.4

[125] Yuchen Zhang, Xi Chen, Denny Zhou, and Michael I Jordan. Spectral methods meet em: A provably optimal algorithm for crowdsourcing. In *Advances in neural information processing systems*, pages 1260–1268, 2014. 5.7