Bayes Filters and Recurrent Neural Networks

Carlton Downey

May 2017

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

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.

Copyright © 2017 Carlton Downey
Keywords: Spectral Algorithms, Bayes Filters, Recurrent Neural Networks, Kernel Methods, Filtering, Time Series, Tensors, CP Decomposition, Predictive State Representations
Abstract

Recurrent Neural Networks (RNNs) have seen a massive surge in popularity in recent years, particularly with the advent of modern architectures such as LSTMs. These sophisticated modern models have resulted in significant performance gains across a number of challenging tasks. Despite their success, we still struggle to provide a rigorous theoretical analysis of these models, or to truly understand the mechanism behind their success. Prior to the success of RNNs, time series data modelling was dominated by Bayes Filters in their many forms. In contrast to RNNs Bayes Filters are grounded in axiomatic probability theory, resulting in a class of models which can be easily analyzed and whose action is well understood. In this work we propose a new class of models called Predictive State Recurrent Neural Networks, which combine the axiomatic probability theory of Bayes Filters with the rich functional forms and practical success of RNNs. We show that PSRNNs can be learned effectively by combining Backpropogation Through Time (BPTT) with a method-of-moments initialization called Two-Stage Regression. Furthermore we show PSRNNs reveal interesting connections between Kernel Bayes Rule and conventional RNN architectures such as LSTMs and GRUs. Finally we show PSRNNs outperform conventional RNN architectures, including LSTMs, on a range of datasets including both text and robotics data.
Contents

1 Introduction 1
   1.1 Contributions .................................................. 2

2 Related Work 5

3 Background 7
   3.1 Dynamical Systems .............................................. 7
   3.2 Predictive State Representations .............................. 7
   3.3 Tensor Decomposition ......................................... 8
   3.4 Orthogonal Random Features ................................. 8

4 Supervised Learning for Dynamical System Learning 11
   4.1 Two Stage Regression ............................................ 12
   4.2 Experiments ..................................................... 12

5 Combining Bayes Filters with RNNs 15
   5.1 Predictive State Recurrent Neural Networks .............. 15
   5.2 Factorized PSRNNs ............................................. 16
   5.3 Learning PSRNNs .............................................. 16
   5.4 Experiments .................................................... 17
      5.4.1 Penn Tree Bank ........................................... 18
      5.4.2 Robotics Data ............................................ 19

6 Orthogonal Random Features for PSRNNs 21
   6.1 The Theory of Orthogonal Kernel Ridge Regression .... 22
      6.1.1 Superiority of the orthogonal features for kernel ridge regression 23
   6.2 Experiments .................................................... 24

7 Proposed Work 27
   7.1 Exploring connections with GRUs/LSTMs/MIs ............... 27
   7.2 Norm PSRNNs .................................................. 28
   7.3 Quantum PSRNNS .............................................. 29
   7.4 Higher Order Optimization ................................... 30
Chapter 1

Introduction

Learning to predict temporal sequences of observations is a fundamental challenge in a range of disciplines including machine learning, robotics, and natural language processing. While there are a wide variety of different approaches to modelling time series data, many of these approaches can be categorized as either recursive Bayes Filtering or Recurrent Neural Networks.

Bayes Filters (BFs) [42] focus on modeling and maintaining a belief state: 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. The belief state is generally interpreted as the statistics of a distribution over the latent state of a data generating process, conditioned on history. BFs recursively update the belief state by conditioning on new observations using Bayes rule. Examples of common BFs include sequential filtering in Hidden Markov Models (HMMs) [6] and Kalman Filters (KFs) [31].

Predictive State Representations [35] (PSRs) are a variation on Bayes filters that do not define system state explicitly, but proceed directly to a representation of state as the statistics of a distribution of features of future observations, conditioned on history. By defining the belief state in terms of observables rather than latent states, PSRs can be easier to learn than other filtering methods [11, 12, 25]. PSRs also support rich functional forms through kernel mean map embeddings [13], and a natural interpretation of model update behavior as a gating mechanism. This last property is not unique to PSRs, as it is also possible to interpret the model updates of other BFs such as HMMs in terms of gating.

Due to their probabilistic grounding, BFs and PSRs 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 including PSRs [12, 25, 43, 50]. Unfortunately, current versions of method of moments initialization restrict BFs to relatively simple functional forms such as linear-Gaussian (KFs) or linear-multinomial (HMMs).

Recurrent Neural Networks (RNNs) are an alternative to BFs that model sequential data via a parameterized internal state and update function. In contrast to BFs, RNNs are directly trained to minimize output prediction error, without adhering to any axiomatic probabilistic interpretation. Examples of popular RNN models include Long-Short Term Memory networks [27] (LSTMs), Gated Recurrent Units [14] (GRUs), and simple recurrent networks such as Elman networks [21].

RNNs have several advantages over BFs. Their flexible functional form supports large, rich models. And, RNNs can be paired with simple gradient-based training procedures that achieve
state-of-the-art performance on many tasks \cite{48}. RNNs also have drawbacks however: unlike BFs, RNNs lack an axiomatic probabilistic interpretation, and are therefore difficult to analyze. Furthermore, despite strong performance in many domains, RNNs are notoriously difficult to train; in particular it is difficult to find good initializations.

In summary, 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. The goal of this thesis is to develop novel hybrid models which inherit advantages from both Bayes Filters and RNNs.

### 1.1 Contributions

The main contributions of this thesis to this point are:

- A new framework, called Two-Stage Regression (2SR), which represents dynamical system learning in terms of supervised learning. By reducing the problem of system identification to supervised learning, it is possible to define a class of partially observable dynamical systems and a learning algorithm that together provide greater modeling capabilities compared to previous spectral algorithms while retaining the computational efficiency and consistency guarantees that are absent in local optimization algorithms such as EM. Joint work with Ahmed Hefny, Geoff Gordon \cite{25}.

- A new model, Predictive State Recurrent Neural Networks (PSRNNs), for filtering and prediction in dynamical systems. PSRNNs draw on insights from both Recurrent Neural Networks (RNNs) and Predictive State Representations (PSRs), and inherit advantages from both types of models. 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 like PSRs, in which observations can be viewed as gating belief states. We also show that PSRNNs can be learned effectively by combining Backpropagation Through Time (BPTT) with supervised learning framework. Finally, we show that PSRNNs can be factorized using tensor decomposition, reducing model size and suggesting interesting connections to existing multiplicative architectures such as LSTMs and GRUs. We apply PSRNNs to 4 datasets, and show that we outperform several popular alternative approaches to modeling dynamical systems in all cases. Joint work with Ahmed Hefny, Boyue Li, Byron Boots, and Geoff Gordon \cite{17, 19}.

- An extension of Orthogonal Random Features to PSRNNs, resulting in PSRNNs which are both smaller and faster. PSRNNs leverage the concept of Hilbert Space Embeddings of distributions \cite{45} to embed predictive states into a Reproducing Kernel Hilbert Space, then estimate, predict, and update these embedded states using Kernel Bayes Rule. Practical implementations of PSRNNs are made possible by the machinery of Random Features, where input features are mapped into a new space where dot products approximate the kernel well. Unfortunately PSRNNs often require a large number of RFs to obtain good results, resulting in large models which are slow to execute and slow to train. Orthogonal
Random Features (ORFs)\cite{53} is an improvement on RFs which has been shown to decrease the number of RFs required for pointwise kernel approximation. Unfortunately, it is not clear that ORFs can be applied to PSRNNs, as PSRNNs rely on Kernel Ridge Regression as a core component of their learning algorithm, and the theoretical guarantees of ORF do not apply in this setting. In this paper, we extend the theory of ORFs to Kernel Ridge Regression and show that ORFs can be used to obtain Orthogonal PSRNNs (OPSRNNs), which are smaller and faster than PSRNNs. In particular, we show that OPSRNN models clearly outperform LSTMs and furthermore, can achieve accuracy similar to PSRNNs with an order of magnitude smaller number of features needed. Joint work with Krzysztof Choromanski, and Byron Boots (submitted to ICLR).
Chapter 2

Related Work

It is well known that a principled initialization can greatly increase the effectiveness of local search heuristics. For example, Boots [10] and Zhang et al. [54] use subspace ID to initialize EM for linear dynamical systems, and Ko and Fox [32] use N4SID [51] to initialize GP-Bayes filters.

Pasa et al. [39] 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 [8] 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 [38]. 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.

Downey et al. [17] propose a two-stage algorithm for learning discrete PSRs, consisting of a spectral initialization followed by BPTT. While that work is similar in spirit to the current paper, it is still an attempt to optimize a BF using BPTT rather than an attempt to construct a true hybrid model. This results in several key differences: they focus on the discrete setting, and they optimize only a subset of the model parameters.

Haarnoja et al. [24] 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 [24].

Finally, Kossaifi et al. [34] also apply tensor decomposition in the neural network setting. They propose a novel neural network layer, based on low rank tensor factorization, which can 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.
Chapter 3

Background

3.1 Dynamical Systems

A dynamical system is a stochastic process (i.e., a distribution over sequences of observations) such that, at any time, the distribution of future observations is fully determined by a vector $s_t$ called the latent state. The process is specified by three distributions: the initial state distribution $P(s_1)$, the state transition distribution $P(s_{t+1} \mid s_t)$, and the observation distribution $P(o_t \mid s_t)$. For later use, we write the observation $o_t$ as a function of the state $s_t$ and random noise $\epsilon_t$.

Given a dynamical system, one of the fundamental tasks is to perform inference, where we predict future observations given a history of observations. Typically this is accomplished by maintaining a distribution or belief $b_{t \mid t-1} = P(s_t \mid o_{1:t-1})$ where $o_{1:t-1}$ denotes the first $t-1$ observations. $b_{t \mid t-1}$ represents both our knowledge and our uncertainty about the true state of the system. Two core inference tasks are filtering and prediction. In filtering, given the current belief $b_t = b_{t \mid t-1}$ and a new observation $o_t$, we calculate an updated belief $b_{t+1} = b_{t+1 \mid t}$ that incorporates $o_t$. In prediction, we project our belief into the future: given a belief $b_{t \mid t-1}$ we estimate $b_{t+k \mid t-1} = P(s_{t+k} \mid o_{1:t-1})$ for some $k > 0$ (without incorporating any intervening observations).

3.2 Predictive State Representations

Predictive state representations (PSRs) are a class of models for filtering, prediction, and simulation of discrete time dynamical systems. PSRs provide a compact representation of a dynamical system by representing state as a set of predictions of features of future observations.

Let $f_t = f(o_{t:t+k-1})$ be a vector of features of future observations and let $h_t = h(o_{1:t-1})$ be a vector of features of historical observations. Then the predictive state is $q_t = q_{t \mid t-1} = E[f_t \mid o_{1:t-1}]$. The features are selected such that $q_t$ determines the distribution of future observations $P(o_{t:t+k-1} \mid o_{1:t-1})$. Filtering is the process of mapping a predictive state $q_t$ to $q_{t+1}$ conditioned

1There 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 paper.

2For convenience we assume that the system is $k$-observable: that is, the distribution of all future observations is determined by the distribution of the next $k$ observations. (Note: not by the next $k$ observations themselves.) At the
on \( o_t \), while prediction maps a predictive state \( q_t = q_{t|t-1} \) to \( q_{t+j|t-1} = E[f_{t+j} \mid o_{1:t-1}] \) without intervening observations.

PSRs were originally developed for discrete data as a generalization of existing Bayes Filters such as HMMs \cite{35}. However, by leveraging the recent concept of Hilbert Space embeddings of distributions \cite{45}, we can embed a PSR in a Hilbert Space, and thereby handle continuous observations \cite{13}. Hilbert Space Embeddings of PSRs (HSE-PSRs) \cite{13} represent the state as one or more nonparametric conditional embedding operators in a Reproducing Kernel Hilbert Space (RKHS) \cite{3} and use Kernel Bayes Rule (KBR) \cite{45} to estimate, predict, and update the state.

For a full treatment of HSE-PSRs see \cite{13}. Let \( k_f, k_h, k_o \) be translation invariant kernels \cite{41} defined on \( f_t, h_t, \) and \( o_t \) respectively. We use Random Fourier Features \cite{41} (RFF) to define projections \( \phi_t = RFF(f_t), \eta_t = RFF(h_t), \) and \( \omega_t = RFF(o_t) \) such that \( k_f(f_i, f_j) \approx \phi_i^T \phi_j, k_h(h_i, h_j) \approx \eta_i^T \eta_j, k_o(o_i, o_j) \approx \omega_i^T \omega_j \). Using this notation, the HSE-PSR predictive state is \( q_t = E[\phi_t \mid o_{t:t-1}] \). Formally an HSE-PSR (hereafter simply referred to as a PSR) consists of an initial state \( b_1 \), a 3-mode update tensor \( W \), and a 3-mode normalization tensor \( Z \). The PSR update equation is

\[
q_{t+1} = (W \times_3 q_t) \ (Z \times_3 q_t)^{-1} \times_2 o_t. \tag{3.1}
\]

where \( \times_i \) is tensor multiplication along the \( i \)th mode of the preceding tensor. In some settings (such as with discrete data) it is possible to read off the observation probability directly from \( W \times_3 q_t \); however, in order to generalize to continuous observations with RFF features we include \( Z \) as a separate parameter.

### 3.3 Tensor Decomposition

The tensor Canonical Polyadic decomposition (CP decomposition) \cite{26} can be viewed as a generalization of the Singular Value Decomposition (SVD) to tensors. If \( T \in \mathbb{R}^{(d_1 \times \ldots \times d_k)} \) is a tensor, then a CP decomposition of \( T \) is:

\[
T = \sum_{i=1}^{m} a_1^i \otimes a_2^i \otimes \ldots \otimes a_k^i
\]

where \( a_i^j \in \mathbb{R}^{d_j} \) and \( \otimes \) is the Kronecker product. The rank of \( T \) is the minimum \( m \) such that the above equality holds. In other words, the CP decomposition represents \( T \) as a sum of rank-1 tensors.

### 3.4 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 and consequently, conduct kernel ridge regression for the OPSRNN model. A class of RBF kernels \( \mathcal{K} \) (RBFs in shorthand) is a family cost of additional notation, this restriction could easily be lifted.
of functions: $K_n : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ for $n = 1, 2, \ldots$ such that $K_n(x, y) = \phi(z)$, for $z = \|x - y\|_2$, 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.

Every RBF kernel $K$ is shift-invariant, thus in particular its values can be described by an integral via Bochner’s Theorem [40]:

$$K(x, y) = \Re \int_{\mathbb{R}^n} \exp(iw^\top(x - y))\mu_K(dw), \quad (3.2)$$

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 $\phi$ and probability density functions for measures $\mu_K$ are given in Table 3.1. The above formula leads straightforwardly to the standard unbiased Monte-Carlo estimator of $K(x, y)$ given as: $K(x, y) = \Phi_{m,n}(x)\Phi_{m,n}(y)$, where a random embedding $\Phi_{m,n} : \mathbb{R}^n \to \mathbb{R}^{2m}$ is given as:

$$\Phi_{m,n}(x) = \left(\frac{1}{\sqrt{m}} \cos(w_i^\top x), \frac{1}{\sqrt{m}} \sin(w_i^\top x)\right)_{i=1}^m, \quad (3.3)$$

vectors $w_i \in \mathbb{R}^n$ are sampled independently from $\mu_K$ and $m$ stands for the number of random features used. In this scenario we will often use the notation $w_i^{\text{iid}}$, to emphasize that different $w_i$s are sampled independently from the same distribution.

<table>
<thead>
<tr>
<th>Name</th>
<th>Positive-definite function $\phi$</th>
<th>Probability density function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian</td>
<td>$\sigma^2 \exp(-\frac{1}{2\lambda^2}z^2)$</td>
<td>$\frac{\sigma^2}{(2\pi\lambda^2)^{n/2}} \exp(-\frac{1}{2\lambda^2}|w|_2^2)$</td>
</tr>
<tr>
<td>Laplacian</td>
<td>$\exp(-z)$</td>
<td>$\prod_{i=1}^n \frac{1}{\pi(1+w_i)}$</td>
</tr>
</tbody>
</table>

**Figure 3.1:** Common RBF kernels, the corresponding functions $\phi$, and probability density functions (here: $w = (w_1, \ldots, w_n)^\top$).

For a dataset $\mathcal{X}$, random features provide an equivalent description of the original kernel via the linear kernel in the new dataset $\Phi(\mathcal{X}) = \{\Phi(x) : x \in \mathcal{X}\}$ obtained by the nonlinear transformation $\Phi$ and lead to scalable kernel algorithms if the number of random features needed to accurately approximate kernel values satisfies: $m \ll N = \|\mathcal{X}\|$.
Chapter 4

Supervised Learning for Dynamical System Learning

Figure 4.1: Learning and applying a dynamical system with instrumental regression. The predictions from S1 provide training data to S2. At test time, we filter or predict using the weights from S2.

Recently there has been substantial interest in developing efficient, globally optimal algorithms for learning dynamical systems. Traditional approaches such as EM[7] and sampling[22] are slow and suffer from local optima. This has resulted in the development of the class of so-called "spectral algorithms"[28]. Conceptually, most these algorithms rely on matrix factorization of observable quantities and method of moments estimators. They are fast, globally optimal, and can be solved efficiently by standard linear algebra software.

Despite the advantages of spectral algorithms they fall short in one important aspect compared to EM and Sampling. These latter two classes are meta-algorithms or frameworks: they offer a clear template for developing new instances and incorporating various forms of prior knowledge. In the current formulation of spectral algorithms it is not obvious how to achieve these tasks. This is primarily because these algorithms are typically described in terms of simple linear algebraic operations, instead of the higher level concepts.
4.1 Two Stage Regression

We propose a framework called Two Stage Regression (2SR) which reduces dynamical system 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. See Hefny et al. [25] for full details.

In 2SR the PSRNN model parameters $q_1$ and $W$ are learned by solving three Kernel Ridge Regression problems in two stages. Ridge regression is required in order to obtain a stable model, as it allows us to minimize the destabilizing effect of rare events while preserving statistical consistency.

In stage one we regress from past $\phi_t$ to future $\eta_t$, and from past $\phi_t$ to the outer product of shifted future $\psi := \eta_{t+1}$ with observations $\omega_t$. Let $C_\phi$ be the matrix whose $t$th column is $\phi_t$, $C_\eta$ the matrix whose $t$th column is $\eta_t$, and $C_{\psi \otimes \omega}$ be the matrix whose $t$th column is $\psi_t \otimes \omega_t$:

$$C_{\eta|\phi} = C_\eta C_\phi^T (C_\phi C_\phi^T + \lambda I)^{-1},$$
$$C_{\psi \omega|\phi} = C_{\psi \omega} C_\phi^T (C_\phi C_\phi^T + \lambda I)^{-1}.$$

In stage two we regress from $C_{\eta|\phi}$ to $C_{\psi \omega|\phi}$ to obtain the model weights $W$:

$$q_1 = C_{\eta|\phi} 1,$$
$$W = C_{\psi \omega|\phi} C_{\eta|\phi}^T (C_{\eta|\phi} C_{\eta|\phi}^T + \lambda I)^{-1},$$

where $\lambda \in \mathbb{R}$ is the ridge parameter and $I$ is the identity matrix and $1$ is a column vector of ones. Once the state update parameters have been learned via 2SR we train a kernel ridge regression model to predict $\omega_t$ from $q_t$. [1]

4.2 Experiments

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 [16], 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. The events denoted by guessing, slipping, learning and forgetting typically have relatively low probabilities.

[1] Note that we can train a regression model to predict any quantity from the state
We evaluate the model using “Geometry Area (1996-97)” data available from DataShop[33]. 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.

Under the (reasonable) assumption that the two states have distinct observation probabilities, this model is 1-observable. Hence we define the predictive state to be the expected next observation, which results in the following statistics: 

$$\psi_t = o_t$$ and $$\xi_t = o_t \otimes_k o_{t+1},$$

where $$o_t$$ is represented by a 2 dimensional indicator vector and $$\otimes_k$$ denotes the Kronecker product. Given these statistics, $$p_t = \mathbb{E}[\xi_t|o_{1:t-1}]$$ 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 [28]. We use this model (denoted by “Spec-HMM”) as a baseline.

Our second baseline (denoted by “Feat-HMM”) is feature-based spectral HMM [44]. 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 4.2. 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.
Figure 4.2: 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).

<table>
<thead>
<tr>
<th>Model</th>
<th>Spec-HMM</th>
<th>Feat-HMM</th>
<th>LR-HMM</th>
<th>EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training time $^2$ (relative to Spec-HMM)</td>
<td>1</td>
<td>1.02</td>
<td>2.219</td>
<td>14.323</td>
</tr>
</tbody>
</table>
Chapter 5

Combining Bayes Filters with RNNs

In this section we introduce Predictive State Recurrent Neural Networks (PSRNNs), a new RNN architecture inspired by PSRs. 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 extension 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 Predictive State Recurrent Neural Networks

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 [36]), the current paper 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:

\[
q_{t+1} = \frac{W \times_2 o_t \times_3 q_t + b}{\|W \times_2 o_t \times_3 q_t + b\|_2}
\]  

(5.1)

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. 3.1); sec 3.2 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.

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 [36, 47]. PSRNNs incorporate such a step, via RFFs. However, a multilayered architecture typically offers higher representation power for a given number of parameters [9].

To obtain a multilayer PSRNN, we stack the 1-layer blocks of Eq. (5.1) 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 [27]. Finally, this architecture is suggested by the full normalized form of an HSE PSR, where the observation is passed through two layers.

![Figure 5.1: PSRNN architecture: See equation 5.1 for details. We omit bias terms to avoid clutter.](image)

5.2 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 \(i\)th row is \(a_i\) (respectively \(b_i, c_i\)). Then the PSRNN state update (equation (5.1)) becomes (up to normalization):

\[
q_{t+1} = W \times_2 o_t \times_3 q_t + b
\]

\[
= (A \otimes B \otimes C) \times_2 o_t \times_3 q_t + b
\]

\[
= A^T (B o_t \otimes C q_t) + b
\]

where \(\otimes\) 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.

5.3 Learning PSRNNs

The Kernel Bayes Rule portion of the PSR update (equation 3.1) can be separated into two terms: \((W \times_3 q_t)\) and \((Z \times_3 q_t)^{-1}\). The first term corresponds to calculating the joint distribution, while
the second term corresponds to normalizing the joint to obtain the conditional distribution. In the discrete case, this is equivalent to dividing the joint distribution of \( f_{t+1} \) and \( o_t \) by the marginal of \( o_t \); see [46] for details.

If we remove the normalization term, and replace it with two-norm normalization, the PSR update becomes

\[
q_{t+1} = \frac{W \times q_t \times o_t}{\|W \times q_t \times o_t\|},
\]

which corresponds to calculating the joint distribution (up to a scale factor), and has the same functional form as our single-layer PSRNN update equation (up to bias).

It is not immediately clear that this modification is reasonable. We show in [18] that our algorithm is consistent in the discrete (realizable) setting; however, to our current knowledge we lose the consistency guarantees of the 2SR algorithm in the full continuous setting. Despite this we determined experimentally that replacing full normalization with two-norm normalization appears to have a minimal effect on model performance prior to refinement, and results in improved performance after refinement. Finally, we note that working with the (normalized) joint distribution in place of the conditional distribution is a commonly made simplification in the systems literature, and has been shown to work well in practice [49].

The adaptation of the two-stage regression algorithm [25] described above allows us to initialize 1-layer PSRNNs; we now extend this approach 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 Experiments

Models compared are LSTMs [36], GRUs [14], basic RNNs [21], KFs [31], 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.\footnote{This is a standard RNN architecture; e.g., a PyTorch implementation of this architecture for text prediction can be found at \url{https://github.com/pytorch/examples/tree/master/word_language_model}} We initialize the LSTMs and RNNs with random weights and zero biases according to the Xavier initialization scheme \cite{glorot2010understanding}. We initialize the the KF using the 2SR algorithm described in \cite{28}. We initialize PSRNNs and factorized PSRNNs as described in section 3.2.

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 \cite{13}). (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 \cite{41}, 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.

### 5.4.1 Penn Tree Bank

Our first set of experiments focus on the discrete setting using the Penn Tree Bank (PTB) dataset, a standard benchmark in the NLP community \cite{37}. Due to hardware limitations we use a train/test split of 120780/124774 characters. We optimize models using Bits Per Character (BPC) and evaluate them using both BPC and one-step prediction accuracy (OSPA).

(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](image)

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.
parameters. To 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.

5.4.2 Robotics Data

Our first set of experiments focus on the continuous setting using using a collection of robotics datasets.

- **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[^2][^20] 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.

We optimize and evaluate all continuous experiments using the Mean Squared Error (MSE).

In Figure 5.4a we compare model performance on the Swimmer, Mocap, and Handwriting datasets. We see that PSRNNs significantly outperform alternative approaches on all datasets.

In Figure 5.4b we attempt to gain insight into why using 2SR to initialize our models is so beneficial. We visualize 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.

[^1]: https://gym.openai.com/
(a) MSE vs Epoch on the Swimmer, Mocap, and Handwriting datasets

(b) 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.

Figure 5.4: Swimmer, Mocap, and Handwriting Experiments
Chapter 6

Orthogonal Random Features for PSRNNs

Practical implementations of PSRNNs are made possible by the machinery of Random Features (RFs): input features are mapped into a new space where dot products approximate the kernel well [41]. RFs are crucial to the success of PSRNNs, however PSRNNs often require a significant number of RFs in order to obtain good results. And, unfortunately, 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) [53]. 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 [15, 53] 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. [53] are not directly applicable to the PSRNN setting. PSRNNs 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. [53] and Choromanski et al. [15]. Hence it is not clear whether or not ORF can be applied to obtain an improvement in the PSRNN setting.

We show that ORFs can be used to obtain OPSRNNs: PSRNNs initialized using ORFs which are smaller, faster to execute and train than PSRNNs initialized using conventional unstructured RFs. We theoretically analyze the orthogonal version of the KRR algorithm that is used to initialize OPSRNNs. We show that orthogonal RNNs 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 OPSRNNs 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.
6.1 The Theory of Orthogonal Kernel Ridge Regression

In this section we extend the theoretical guarantees of Yu et al. [53] to give rigorous theoretical 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 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. 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 \((x_i, y_i)\), where \(x_i \in \mathbb{R}^n\), \(y_i \in \mathbb{R}\) for \(i = 1, 2, ..., \) are generated as follows: \(y_i = f^*(x_i) + \nu_i\), where \(f^* : \mathbb{R}^n \to \mathbb{R}\) is a function that the model aims to learn, and \(\nu_i\) 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:

\[
\mathcal{R}(f) \equiv \mathbb{E}_{\{\nu_i\}_{i=1,\ldots,N}} \left[ \frac{1}{N} \sum_{j=1}^{N} (f(x_i) - f^*(x_i))^2 \right],
\]

(6.1)

where \(N\) stands for a dataset size.

By \(f_{\text{vec}} \in \mathbb{R}^N\) we denote a vector whose \(j\)th entry is \(f^*(x_j)\). Denote by \(f_{\text{KRR}}\) a kernel ridge regression estimator applying 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 \(K\). It is a well known result [1, 5] that the empirical risk of \(f_{\text{KRR}}\) is given by the formula:

\[
\mathcal{R}(f_{\text{KRR}}) = N^{-1} \lambda^2 (f_{\text{vec}}^*)^\top (K + \lambda N I_N)^{-2} f_{\text{vec}}^* + N^{-1} \sigma^2 \text{Tr}(K^2(K + \lambda N I_N)^{-2}),
\]

(6.2)

where \(\lambda > 0\) stands for the regularization parameter and \(I_N \in \mathbb{R}^{N \times N}\) is an identity matrix.

Denote by \(\hat{f}_{\text{KRR}}\) an estimator based on some random feature map mechanism and by \(\tilde{K}\) 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. [4]) is the modified version of the above formula for \(\mathcal{R}(f_{\text{KRR}})\), namely: \(\mathcal{R}_K(f_{\text{vec}}^*) \equiv N^{-1} \lambda^2 (f_{\text{vec}}^*)^\top (K + \lambda N I_N)^{-1} f_{\text{vec}}^* + N^{-1} \sigma^2 s_\lambda(K)\), where \(s_\lambda(K) \equiv \text{Tr}(K(K + \lambda N I_N)^{-1})\). It can be easily proven that \(\mathcal{R}(f_{\text{KRR}}) \leq \mathcal{R}_K(f_{\text{vec}}^*)\).

To measure how similar to the exact kernel matrix (in terms of spectral properties) a kernel matrix obtained with random feature maps is, we use the notion of \(\Delta\)-spectral approximation [4].

**Definition 1.** For a given \(0 < \Delta < 1\), matrix \(A \in \mathbb{R}^{N \times N}\) is a \(\Delta\)-spectral approximation of a matrix \(B \in \mathbb{R}^{N \times N}\) if \((1 - \Delta)B \preceq A \preceq (1 + \Delta)B\).

It turns out that one can upper-bound the risk \(\mathcal{R}(\hat{f}_{\text{KRR}})\) for the estimator \(\hat{f}_{\text{KRR}}\) in terms of the \(\Delta\) parameter if matrix \(\tilde{K} + \lambda N I_N\) is a \(\Delta\)-spectral approximation of the matrix \(K + \lambda N I_N\), as the next result [4] shows:
Theorem 2. Suppose that $\|K\|_2 \geq 1$ and that matrix $\hat{K} + \lambda N I_N$ obtained with the use of random features is a $\Delta$-spectral approximation of matrix $K + \lambda N I_N$. Then the empirical risk $R(\hat{f}_{KRR})$ of the estimator $\hat{f}_{KRR}$ satisfies:

$$R(\hat{f}_{KRR}) \leq \frac{1}{1 - \Delta} R_K(f^*_\text{vec}) + \frac{\Delta}{1 + \Delta} \frac{\text{rank}(\hat{K})}{N} \sigma^2.$$ (6.3)

6.1.1 Superiority of the orthogonal features for kernel ridge regression

Consider the following RBF kernels, that we call smooth RBFs. As we show next, Gaussian kernels are smooth.

Definition 3 (smooth RBFs). We say that the class of RBF kernels defined by a fixed $\phi : \mathbb{R} \rightarrow \mathbb{R}$ (different elements of the class corresponds to different input dimensionalities) and with associated sequence of probabilistic measures $\{\mu_1, \mu_2, \ldots\}$ ($\mu_i \in \mathcal{M}(\mathbb{R}^i)$) is smooth if there exists a nonincreasing function $f : \mathbb{R} \rightarrow \mathbb{R}$ such that $f(x) \rightarrow 0$ as $x \rightarrow \infty$ and furthermore the $k^{th}$ moments of random variables $X_n = ||w||$, where $w \sim \mu_n$ satisfy for every $n, k \geq 0$:

$$E[X_n^k] \leq (n-1)(n+1) \cdot \cdots \cdot (n+2k-3)k!f(k).$$

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 the following formula:

$$E[X_n^k] = 2^k \frac{(n^2+k-1)!}{(n-1)!}$$ for $n > 1$.

Our main result is given below and shows that orthogonal random features lead to tighter bounds on $\Delta$ for the spectral approximation of $K + \lambda N I_N$. Tighter bounds on $\Delta$, as Theorem 2 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 4 (spectral approximation). Consider a smooth RBF (in particular Gaussian kernel). Let $\hat{\Delta}_{\text{iid}}$ denote the smallest positive number such that $\hat{K}_{\text{iid}} + \lambda N I_N$ is a $\Delta$-approximation of $K + \lambda N I_N$, where $\hat{K}_{\text{iid}}$ is an approximate kernel matrix obtained by using unstructured random features. Then for any $a > 0$,

$$\mathbb{P}[\hat{\Delta}_{\text{iid}} > a] \leq p_{\text{N,m}}^{\text{iid}}(\frac{a\sigma_{\text{min}}}{N}),$$ (6.4)

where: $p_{\text{N,m}}^{\text{iid}}$ is given as: $p_{\text{N,m}}^{\text{iid}}(x) = N^2 e^{-Cmx^2}$ for some universal constant $C > 0$, $m$ is the number of random features used, $\sigma_{\text{min}}$ is the smallest singular value of $K + \lambda N I_N$ and $N$ is dataset size. If instead orthogonal random features are used then for the corresponding spectral parameter $\hat{\Delta}_{\text{ort}}$ the following holds:

$$\mathbb{P}[\hat{\Delta}_{\text{ort}} > a] \leq p_{\text{N,m}}^{\text{ort}}(\frac{a\sigma_{\text{min}}}{N}),$$ (6.5)

where function $p_{\text{N,m}}^{\text{ort}}$ satisfies: $p_{\text{N,m}}^{\text{ort}} < p_{\text{N,m}}^{\text{iid}}$ 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 \( p_{N,m}^{\text{ort}} \) can be derived from the proof that we present in the Appendix, however for clarity we do not give it here.

Theorem 4 combined with Theorem 2 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 5.** Under the assumptions of Theorem 2 and Theorem 4 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: 

\[
P\left[\mathcal{R}(\hat{f}_{KRR}) > c\right] \leq p_{N,m}^{\text{iid}}(\alpha_c \sigma_{\min} N),
\]

where \( \alpha_c \) is given as:

\[
\alpha_c = 1 - \frac{\mathcal{R}_K(f^*) \sigma_c}{c - \frac{\mathcal{R}_K(f^*) \sigma_c}{N}}
\]

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:

\[
P\left[\mathcal{R}(\hat{f}_{KRR}) > c\right] \leq p_{N,m}^{\text{ort}}(\alpha_c \sigma_{\min} N).
\]

As before, since for large \( n \) function \( p_{N,m}^{\text{ort}} \) satisfies \( p_{N,m}^{\text{ort}} < p_{N,m}^{\text{iid}} \), 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 \( n \).

### 6.2 Experiments

In our first experiment we examine the effectiveness of Orthogonal RF with respect to learning a good PSRNN via 2SR. In figure 6.1 we compare the MSE for a PSRNN learned via Orthogonal 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 6.1: MSE for Orthogonal RF vs Standard RF on all three data sets.](image)

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 6.2 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 MSE than Standard RF.

![Figure 6.2: MSE for Orthogonal RF vs Standard RF on all three data sets](image)

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 data sets.
Chapter 7

Proposed Work

7.1 Exploring connections with GRUs/LSTMs/MIs

The value of bilinear units in RNNs was the focus of recent work by Wu et al [52]. 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. (5.4) 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 [27]. 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).

$$[q_{t+1}]_i = \sum_j A_{ji} \left( \sum_k B_{jk}[o_t]_k \odot \sum_l C_{jl}[q_t]_l \right) + b$$ (7.1)

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:

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

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 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.

We propose to explore this connection in detail. In particular we would like explore the continuum of models lying between PSRNNs and GRUs/LSTMs, with model features belonging to both classes. In this way we would like to determine which features of existing models are crucial to their success, and which can be removed.

### 7.2 Norm PSRNNs

One of the keys to the success of PSRNNs was replacing the matrix-inverse normalization term from HSE-PSRs with a 2-norm based normalizer, resulting in an architecture amenable to BPTT. While this resulted in a model which performs well in practice, we were not able to provide a full theoretical justification for this change. Furthermore it is not clear how to map the resulting state space back to the observation space, i.e. given a state how do we make predictions about observations; as a reasonable first attempt we have been using a linear mapping, however we would prefer something more principled.

Observable Operator Models (OOMs) [30] are a type of Bayes Filter closely related to PSRs first introduced Jaeger et al. An OOM consists of an initial state $q_1$, a collection of operators $A_1, \ldots, A_n$, and a normalizer $z$. Given a state $q_i$ and an observation $o_i$ the state update is:

$$q_{i+1} = \frac{A_o q_i}{z^TA_oq_i}$$

And observation predection is:

$$p(o) = z^TA_oq_i$$

Unfortunately OOMs suffer from the problem of negative probabilities: It is possible that an OOM learned from data may produce negative probabilities when above prediction equation is applied. Furthermore it is undecidable whether or not a given OOM will produce negative probabilities for some set of inputs.

To solve this problem Zhao and Jaeger [55] introduce Norm Observable Operator Models (NOOMs). NOOMs have the same set of model parameters as OOMs, however the NOOM update equation is:

$$q_{i+1} = \frac{A_o q_i}{||A_oq_i||}$$

And observation probability is:

$$p(o) = ||A_oq_i||$$
We note the clear similarity between the NOOM update equation, and the PSRNN update equation. This suggests that we may be able to obtain a theoretically well justified PSRNN model by making predictions based on the prediction equations for NOOMs. Specifically:

\[ p(o) = \| W \times_2 o \times_3 q_t \|_2 \]  

(7.2)

This has the additional benefit of reducing the number of parameters required by the model, as the prediction parameters are the same as the update parameters.

We propose to investigate this connection more closely and develop a variant of PSRNNs with theoretically well justified predictions based on the work of Zhao and Jaeger [55].

### 7.3 Quantum PSRNNs

Close inspection of NOOMs reveals they are clearly based on a simplified version of quantum mechanics. We propose to extend use this connection to develop a “Full Quantum” version of PSRNNs.

In the same way that classical probability theory is the manipulation of stochastic vectors using stochastic matrices, quantum mechanics can be viewed as the manipulation of norm-one vectors via unitary matrices. Specifically a quantum mechanics model consists of an initial state \( Q_1 \), and a set of operators \( A_1, ..., A_n \). The key difference here is that instead of being a vector, the state is a Positive Semi-Definite matrix with trace one. Given these quantities the state update is:

\[
q_{i+1} = \frac{A_{o_i}q_iA_{o_i}^T}{Tr(A_{o_i}q_iA_{o_i}^T)}
\]

And observation probability is:

\[
p(o) = Tr(A_{o_i}q_iA_{o_i}^T)
\]

Again we note the clear similarity between the Quantum Mechanics equations and the PSRNN equations.

It is known that quantum mechanics can often compactly represent dynamical systems which cannot be represented effectively by classical statistical models. We propose extending PSRNNs to Quantum PSRNNs using the above equation as inspiration: i.e. Let a quantum PSRNN state be a PSD matrix with trace one. Our update and prediction equations become:

\[
A_{o_i} = W \times_2 o
\]

\[
q_{i+1} = \frac{A_{o_i}q_iA_{o_i}^T}{Tr(A_{o_i}q_iA_{o_i}^T)}
\]

And observation probability is:

\[
A_{o_i} = W \times_2 o
\]

\[
p(o) = Tr(A_{o_i}q_iA_{o_i}^T)
\]
7.4 Higher Order Optimization

Gradient descent, in the guise of BPTT, is the standard procedure for optimizing RNNs, regardless of the RNN architecture. This is despite a large body of work showing that higher order approaches to optimization which consider higher order gradients can achieve significantly faster convergence in many other settings. People have attempted to apply higher order methods to RNNs in the past, however this they have not been able to achieve good results. It is believe this is due to the topology of the search space, and the inability of higher order methods to escape local optima effectively.

We believe it may be possible to achieve good performance on PSRNNs using higher order methods. PSRNNs have a consistent initialization which is guaranteed to result in an initialization close to a global optima, in contrast to classical RNN architectures which are initialized at random. This suggests that the role of BPTT in PSRNNs can be viewed as fine tuning the model, whereas BPTT in classical RNN architectures must perform both global search and fine tuning.

We propose to investigate the performance of higher order optimization techniques on PSRNNs.
Bibliography


[41] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In


