Recurrent and Bayesian Kernel Learning for Small Data with Applications to Neuroimaging

by

Alex Lewandowski

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Statistical Machine Learning

Department of Mathematical and Statistical Sciences
University of Alberta

Abstract

Gaussian processes are flexible probabilistic models for regression and classification. However, their success hinges on a well-specified kernel that can capture the structure of data. For complex data, the task of hand crafting a kernel becomes daunting. In this thesis, we propose new methods for Gaussian process classification. In particular, we propose learning flexible kernels that are parameterized by recurrent neural networks. Unlike previous work in recurrent kernel learning, our recurrent kernels are learned through a scalable variational framework and applied to classification. We also investigate methods to propagate the uncertainty of neural network parameters through the Gaussian process. In doing so, we propose a novel use of batch normalization that provides estimates of uncertainty for feed-forward and convolutional neural networks. To evaluate our models, we simulate multivariate time-series data with commonly observed phenomenon in neuroimaging data and compare against two baselines: Gaussian processes and neural networks. We then compare our model to the best performing baselines on common deep learning benchmarks and real neuroimaging datasets. We find that our models provide better predictions and estimates of uncertainty when sample sizes are small and remains competitive at larger sample sizes.

Acknowledgements

First and foremost, I would like to thank my supervisor Ivor Cribben. Not only did he nurture my interest in statistics and neuroimaging, but he also consistently supported my research endeavors. I also thank my supervisor and professor Rohana Karunamuni, whose early guidance on the principles of statistical inference were invaluable. Next, I would like to express gratitude to my committee members and professors Dale Schuurmans and Russell Greiner. They instilled in me an early passion for machine learning and were strong influences on this thesis. I feel that I have learned more in the past two years than I have in my entire previous academic career. For this, I owe and appreciate the many amazing professors at the University of Alberta.

I must also express my love and thanks to my family. In Australia, Dubai, Toronto, Tokyo or Vietnam, your love and support always carried me through good times and bad. Last, but certainly not least, I would like to thank Katie. Without you, I would not be half of the person I am today.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning methodology has long been rooted in classical statistical and probabilistic models. As a result, classification models like support vector machines and Gaussian processes have dominated the literature. Gaussian processes in particular have been championed as a flexible tool for regression and classification, with modest computational costs [61]. A more recent paradigm in machine learning is deep learning: the study and application of neural networks. In recent years, deep learning has exploded due to its wide spread success in various applications, especially in computer vision and natural language processing [32]. Common to all of the success stories however, is massive amounts of data. While data may not be a problem in multimedia based domains, where the internet can provide massive data, it becomes a problem in more specialized domains like medicine. When only small amounts of medical data is available, the application of neural network becomes suspect.

The early focus on simpler models was two fold: complicated models require more data to be accurate and simpler models are easier to understand. While these two concerns hold weight today, their importance in the machine learning community has diminished. Now, complex models can make use of large amounts of raw data to make predictions in domains like natural language processing and computer vision. Still, we use these concerns as motivating research questions. First, can we learn complex models from limited data? Second, can we reason about the uncertainties of complex models? Unfortunately, neural networks alone have not answered these questions, so we search for alternatives.

Between classical probabilistic methods and deep learning there are two middle grounds: Bayesian Deep Learning (BDL), where probabilistic methods are used in service of neural networks, and Deep Bayesian Learning (DBL), where neural networks are used in service of probabilistic models. When data is limited, both BDL and DBL approaches are favoured over deep learning methods [50, 56, 28]. The classic BDL example are Bayesian neural networks, which are a valuable tool when data is limited due to the regularization they provide, referred to as the Bayesian Occam's razor [50]. Unfortunately, inference in Bayesian neural networks is still difficult and most research explores methods to approximate the inference problem [30, 38]. Hence, BDL models like Bayesian neural networks remain poorly understood compared to classical probabilistic models like Gaussian processes.

We now turn to DBL, where neural networks are used as function approximators in a well understood probabilistic model. These models are easier to use compared to (Bayesian) deep learning models since they can rely on properties of the well understood probabilistic model. For example, consider the case of classification of red versus blue in Figure 1.1. The data is originally represented in Cartesian coordinates, and a naive probabilistic classifier may struggle to learn a non-linear classifier to separate the red from the blue. However, if we could use a neural network to learn a polar coordinates representation, we would see that a linear classifier can separate red from blue. Further, given the polar representation, we could use our naive probabilistic model to provide uncertainty estimates of the classification results. While a toy example, this demonstrates the power of an efficient representation. In principle, any function can be used to obtain a new representation of the data; common examples in the statistics literature include principle components analysis and independent component analysis [9]. Neural networks are a particular choice of function approximator because they can learn efficient and hierarchical representations through gradient optimization. Hence, they can be seamlessly paired with probabilistic models that are also trained with gradient optimization.

In functional Magnetic Resonance Imaging (fMRI) data, we partition the

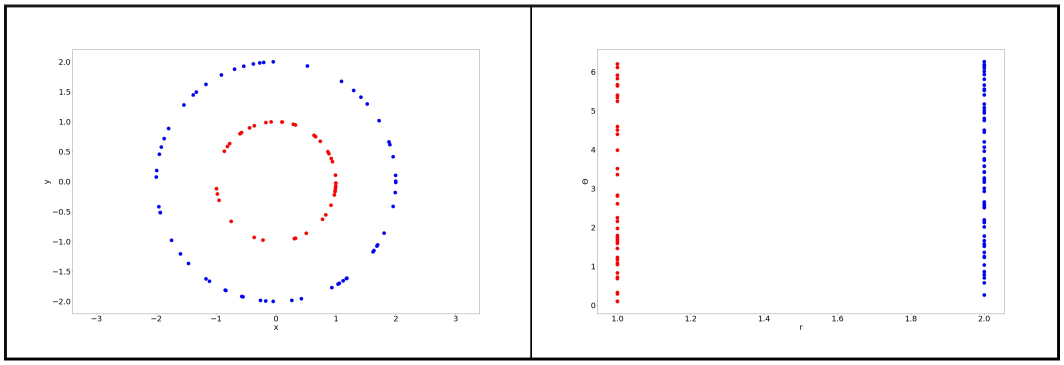

Figure 1.1: Two possible representations of the same problem. Left: Cartesian coordinates, not linearly separable. Right: Polar coordinates, linear classifier can achieve perfect accuracy.

subjects into classes that correspond to the presence versus absence of a disease. The task is then to infer the disease state of a subject given that subject's fMRI scan. The machine learning models that have been applied to classification of fMRI data [59] tend to be simplistic and forgo flexible probabilistic models like Gaussian processes [65]. Instead of a learned representation, they leverage *post-hoc* representations like covariance structure [18]. While moderately successful, this can be limiting, since there is no concrete evidence that the brain encodes all disease information in the covariance structure. Hence, there is no reason to believe that a fixed representation, like covariance structure, will generalize to other subjects or diseases. Instead, we want to *learn* a representation of the data using a neural network, while simultaneously learning a classifier.

There has been recent interest in Gaussian processes with neural network parameterized kernels; a DBL framework referred to as deep kernel learning [80, 78, 16, 14]. While the idea of "deep-ifying" and learning a kernel for representation is not new [42, 5], only recently have they been trained end-to-end. This is largely due to advances in inducing point [77] and variational [37] methods that allow for scalable inference. However, current methods fail to account for temporal dependencies and are unable to propagate the uncertainty in neural network weights. Our model builds upon previous stochastic variational deep kernel learning in two important ways. First, we propose using recurrent neural networks to learn temporal representations for Gaussian process in a classification framework. Second, while the Gaussian process in deep kernel learning can provide uncertainty estimates, the estimates fail to account for uncertainty in the neural network parameters. We extend the deep kernel learning to a BDL framework by accounting for the uncertainties in the neural network representation. We show that our BDL and DBL models perform well even when the dataset is small relative to the dimension of the data.

The remainder of this thesis is structured as follows: Chapter 2 summarizes background in machine learning, Gaussian processes, deep learning, approximate inference and neuroimaging. In particular, it aims to be a self contained introduction to the pillars of our work: (Bayesian) deep learning and its need for approximate inference. Next, Chapter 3 introduces deep kernel learning and our extension to recurrent neural networks. We then discuss batch normalization as a proxy for weight uncertainty in the context of deep neural network kernels. In Chapter 4 we perform preliminary evaluation of our model based on simulated multivariate time-series. We then extend this analysis to study some common machine learning benchmarks, as well as an application to various fMRI datasets. Lastly, Chapter 5 provides a summary of the results and a discussion on future work.

Chapter 2

Background

We will now cover the necessary machine learning background, as well as the three pillars of this work: Gaussian processes, neural networks and approximate inference. While the material on Gaussian processes is important, the central contribution of this thesis is a combination of neural networks and approximate inference. Indeed, the motivation for this work is how to best estimate a representation for a Gaussian process classifier and then reason about the representation's uncertainty. This is as opposed to quantifying the uncertainty of a neural network that learns to directly output classification labels. In some ways this can be easier: a Gaussian process is a probabilistic model, so it can quantify its own uncertainty given the intermediate representation of the neural network output. In other ways, this is more difficult: the interaction between the Gaussian process and neural network is not well understood. In addition, Gaussian process models require matrix inversions that can incur a non-trivial overhead cost. The material is organized in the order outlined above, which is also in an increasing order of importance. Lastly, we will explore how these approaches have been used in the fMRI classification literature.

2.1 Machine learning methodology

For this section, we provide an introduction to machine learning while highlighting differences from statistics. Despite many seeing machine learning as a sub-field of statistics, there is a large difference in terminology, as highlighted in the translation table in [75]. Furthermore, machine learning differs from statistics in its focus (ex. scalable algorithms, and likelihood-free approaches). Hence, while machine learning deals with statistical problems such as parameter estimation, it is often dealt with approximately, or in models that are atypical in the statistical literature. As a result, machine learning is both more general (ex. search methods, reinforcement learning) and more specific (ex. prediction models of regression and classification, without survey or sampling design). Since this thesis is mainly concerned with Bayesian approaches, we will outline the relevant statistical and machine learning tools below.

2.1.1 Terminology

Much of statistical and machine learning research is interdisciplinary, leading to a confusion in terminology and model etymology. In machine learning, models are trained as opposed to estimated, and the covariates are referred to as features. These correspondences are exact, and the difference stems from the perceived end goal of training an artificially intelligent agent on features in a computer vision setting, as opposed to estimating the effect of covariates in a model.

2.1.2 Bayesian methods

Bayesian methods treat unknown parameters as random, and hence require a distributional assumption in the form of a prior [50]. It is important to note that the true value of a parameter can still be fixed, even when the parameter is treated like a random variable. In essence, the Bayesian method uses the language of probability to quantify the parameter's uncertainty and it is a purely mechanical choice. The fact that the parameter is unknown implies that we are uncertain of their value, where the degree of uncertainty depends on the data that we observe.

In order to reason about the uncertainty of unknown parameters, we use Bayes' rule. For unknown variables $\mathbf{z}$, and observations $\mathbf{x}$, we write the posterior as follows

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{\int p(\mathbf{x},\mathbf{z})d\mathbf{z}}$$

.

In the above identity, we call $p(\mathbf{z})$ the prior. Before seeing any data, we may have reasons to believe that $\mathbf{z}$ follows some distribution due to intuition or expert advice. The $p(\mathbf{x}|\mathbf{z})$ term is the likelihood, common to frequentist statistics, which describes the behavior of our data. For example, Bernoulli likelihood can be used for classification while a Gaussian likelihood can be used for regression. Lastly, the $p(\mathbf{x})$ term is for normalization and has an specific interpretation. Unfortunately, the $p(\mathbf{x})$ term is often intractable. Hence, one may employ methods of approximate inference to avoid calculating $p(\mathbf{x})$ directly.

An important property of likelihood-prior pair is conjugacy. Simply stated, if a likelihood-prior pair are conjugate to each other, then the posterior distribution belongs to a known family, with parameters depending on the prior and likelihood. This is convenient since the parameters of the posterior distribution can be computed in a straight-forward manner, given that the likelihood and prior are conjugate to each other. When the distributions are non-conjugate, the posterior must be dealt with on a case-by-case basis. Specifically, the above integral must be evaluated or approximated for any non-conjugate pair, which is often complicated and computationally challenging.

2.1.3 Stochastic gradient descent

While machine learning models have a variety of training methods, there has been a large push for Stochastic Gradient Descent (SGD) methods. This is in part due to the success of neural networks, which are fully differentiable and most effectively optimized by SGD when the sample size is large [13]. Many variants have been introduced, but we will only detail SGD in its most basic form. If we have data $\{\mathbf{x}_i\}_{i=1}^N$ and a cost function, parameterized by θ, $J(\theta; \mathbf{x})$, then we can optimize via SGD with learning rate α as follows: SGD has been

Algorithm 1 An algorithm to calculate the minimum of the cost function $J(\cdot)$ via stochastic gradient descent. Convergence can be measured by a sufficiently small difference between the objective function at different epochs, or a sufficiently large number of iterations.

```
Input: data $\mathbf{x}$, index set $S \subset \{1, \dots, N\}$ of a mini-batch, learning rate $\alpha$
Output: $\hat{\theta}$
  function SGD($\mathbf{x}, S, \alpha$)
      init $\hat{\theta}$
      while not converged do
          $\mathbb{E}_{\mathbf{x}}[\nabla J(\theta; \mathbf{x})] \approx \frac{1}{|S|} \sum_{i \in S} \nabla J(\theta; \mathbf{x}_i)] := \hat{\nabla}$

          $\hat{\theta} \leftarrow \hat{\theta} - \alpha \hat{\nabla}$
      end while
  end function
```

the biggest breakthrough in the training of neural networks. As neural networks and the data required to train them became larger, a need for working on small partitions of the data (or mini-batches) became prominent. Stochastic gradient descent uses the mini-batches to obtain estimates of gradients in standard gradient descent. Still, the learning rates used in SGD remain an important hyper parameter, with no clear tuning process. To alleviate this issue, the majority of current SGD research revolves around momentum optimization which automatically and dynamically select the learning rates [63]. Unfortunately, this introduces the momentum parameter that, like the learning rate, needs tuning. This has led to adaptive methods of optimization [44, 82] which can automatically tune both learning rate and momentum.

2.2 Gaussian processes

A Gaussian Process $g \sim GP(f|\mu(\cdot), k(\cdot, \cdot))$ is the infinite dimensional analogue of a multivariate Gaussian distribution. Its defining property is that, for any finite set of indices $\{i_1, \dots, i_n\}$ of the data $\mathbf{x}$, we have that $\mathbf{G} =$

$\{g(\mathbf{x}_{i_1}), \ldots, g(\mathbf{x}_{i_n})\}$ is distributed as an n-dimensional multivariate normal distribution. A Gaussian Process acts as a *non-parametric* prior over the space of functions because its complexity grows with more data. That is, its covariance parameter, the (positive-definite) kernel function k, is evaluated at more points as data is accumulated. Put another way, the covariance matrix $[\mathbf{K}]_{ij} = k(\mathbf{x_i}, \mathbf{x_j})$, grows in size as we see more data. The mean function however, is usually assumed to be $\mu = 0$ unless there is reason to believe *a priori* otherwise. It is important to note that even with a prior mean function of zero, the posterior mean is not necessarily zero. Much of this background can be found in more detail in the standard GP reference [61].

To improve performance at scale, Gaussian processes can use a subsample $\mathbf{u}$ of size m called inducing points, compared to the data $\mathbf{x}$ of size n, where $m << n$ [66]. This reduces the dimension of the associated kernel matrix in some calculations from $[\mathbf{K}_{nn}]_{ij} = k(\mathbf{x_i}, \mathbf{x_j})$ which is $n \times n$ to $[\mathbf{K}_{mm}]_{ij} = k(\mathbf{u_i}, \mathbf{u_j})$ which is $m \times m$. This can be taken a step further, where we call u pseudo-inputs that have an associated distribution $q(\mathbf{u})$ with parameters that we optimize. In this case, we approximate the data by a sufficiently good coverage of smaller points, usually from a clustering algorithm [72].

2.2.1 Kernels

The kernel function is the fundamental contributor to the characterization of a Gaussian process. Formally, a kernel is a symmetric function defined on the domain of inputs, so that if $\mathbf{x} \in \mathcal{X}$ then $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and for any n scalar variables, $a_1, \ldots, a_n \in \mathbb{R}$, and any n elements of $\mathcal{X}$, $\mathbf{x}_1, \ldots, \mathbf{x}_n$ we have the positive-definite condition:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

For arbitrary kernels k_1, k_2, non-negative scalars $\alpha \geq 0$ and function f we have the following kernel properties:

- additive combinations

$$k(\mathbf{x_i}, \mathbf{x_j}) = k_1(\mathbf{x_i}, \mathbf{x_j}) + k_2(\mathbf{x_i}, \mathbf{x_j})$$

- multiplicative combinations:

$$k(\mathbf{x_i}, \mathbf{x_j}) = k_1(\mathbf{x_i}, \mathbf{x_j}) \cdot k_2(\mathbf{x_i}, \mathbf{x_j})$$

- scalar multiplication:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \alpha k_1(\mathbf{x_i}, \mathbf{x_j})$$

- function composition:

$$k(\mathbf{x_i}, \mathbf{x_j}) = k_1(f(\mathbf{x_i}), f(\mathbf{x_j}))$$

While kernels can be hand-crafted for a specific application, they can also be learned or estimated from the data. Hence, for complex tasks where the kernel structure is not clear, it may be best to construct a kernel that is flexible enough, using the properties outlined above. Some examples of common covariance kernels for Gaussian processes are:

- the linear kernel

$$k(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i}^T \mathbf{x_j}$$

- the radial basis function kernel (also known as Gaussian kernel):

$$k(\mathbf{x_i}, \mathbf{x_j}) = e^{-\frac{\|\mathbf{x_i} - \mathbf{x_j}\|^2}{2\sigma^2}}$$

- the minimum kernel, producing a Brownian motion [43]

$$k(\mathbf{x_i}, \mathbf{x_j}) = \min(i, j)$$

We stress that a kernel can drastically change the behavior of a Gaussian process [23]. For example, Gaussian processes with an RBF kernel can approximate any continuous function. Hence, the RBF kernel enforces a strict smoothness assumption. However, other kernels like the minimum kernel can force the Gaussian process to behave like Brownian motion which is almost surely nowhere differentiable [62].

2.2.2 Classification

Gaussian process classification, similar to logistic regression, must use a specific likelihood with support on the unit interval $[0, 1]$. For our purposes, we use a probit likelihood, defined by an element-wise link function $\phi(x) = P(x < Z)$ where $Z \sim N(0, 1)$. Logistic regression models also optimize this likelihood with some linear parameterized function $g_{\boldsymbol{\beta}}(\mathbf{x_i}) = \boldsymbol{\beta}\mathbf{x_i}$ such that

$$P(\boldsymbol{\beta}|\mathbf{x}, \mathbf{y}) = \prod_{n=1}^{N} \phi(g_{\boldsymbol{\beta}}(\mathbf{x_i}))^{y_i}[1 - \phi(g_{\boldsymbol{\beta}}(\mathbf{x_i})]^{1-y_i}$$

For Gaussian process classification, we draw $g \sim \mathrm{GP}(0, \mathbf{K})$ from a Gaussian process prior with kernel $\mathbf{K}$. However, the posterior of this Bayesian procedure does not have an analytic form so we instead write the joint distribution as

$$P(\mathbf{K}, \mathbf{x}, \mathbf{y}) = \mathcal{N}(g|0, \mathbf{K}) \prod_{n=1}^{N} \phi(g(\mathbf{x_i}))^{y_i}[1 - \phi(g(\mathbf{x_i})]^{1-y_i} \tag{2.1}$$

where $\mathbf{K}$ is the Kernel matrix given by the observations and specifies the covariance structure, i.e. $[\mathbf{K}]_{ij} = k(\mathbf{x_i}, \mathbf{x_j})$.

We emphasize the two main restrictions with Gaussian process classification. First, inference is difficult due to the intractable integral required to recover the posterior. This can be rectified with variational inference, which we will detail in Section 2.4. Second, selecting an appropriate kernel is a laborious task especially in fields with little established prior knowledge, such as neuroimaging. However, if we can devise a flexible enough kernel, it can discover structure instead of having it specified directly to the kernel. This presents us with two new problems: uncertainty and regularization. It is unclear how to incorporate the uncertainty of the neural network weights. Additionally, neural networks greatly increase the number of parameters in the model which could potentially lead to over-fitting. However, we will show how incorporating uncertainty in neural network weights also leads to regularization.

2.3 Deep learning

Deep learning is a collection of models and methods involving neural networks with many hidden layers. In the machine learning community, deep neural networks have provided several success stories in supervised learning tasks like classification and regression. More recently, they have been leveraged in unsupervised learning tasks like generative modeling, which aim to estimate distributions of data. In this section, we provide a primer on the following common neural network architectures: feed forward, convolutional and recurrent. In addition, some material on Bayesian deep learning methods and regularization will be covered, while a more technical discussion is deferred to Chapter 3. More details on deep learning can be found in [32].

2.3.1 Feedforward neural networks

Feedforward neural networks, are defined by a number of layers H, a number of neurons per layer $\{N_h\}_{h=0}^{H}$ where N_0 is the input dimension and N_H is the output dimension, an activation function r, and parameterized by a set of weight matrices $\{\mathbf{W_h}\}_{h=1}^{H}$ and bias vectors $\{\mathbf{b_h}\}_{h=1}^{H}$ such that $\mathbf{W_h} \in \mathbb{R}^{N_{h-1} \times N_h}$ and $\mathbf{b}_h \in \mathbb{R}^{1 \times N_h}$. The activation function r is typically taken to be the Rectified Linear Unit (ReLU), $r(\mathbf{x}) = \max(0, \mathbf{x})$. Then, a feedforward neural network f is given by the following recurrence relation:

$$\begin{aligned}
\mathbf{h_1} &= r(\mathbf{W_1 x} + \mathbf{b}_1) \\
\mathbf{h_h} &= r(\mathbf{W_h h}_{h-1} + \mathbf{b}_h) && \text{for } h \geq 2 \\
f(\mathbf{x}) &= \mathbf{h_H}
\end{aligned}$$

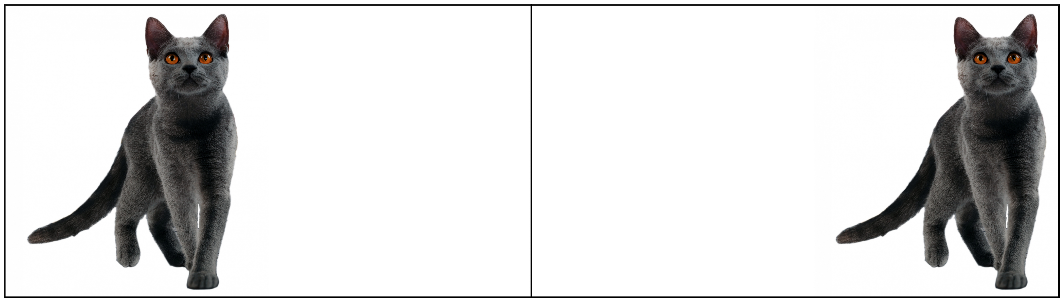

Figure 2.1: Two possible representations of the same problem that are translation invariant. Left: A cat left of center. Right: A cat right of center.

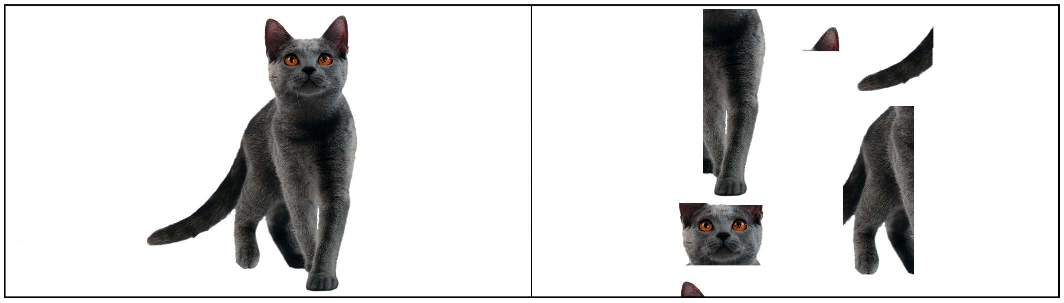

Figure 2.2: Two possible representations of the same problem that are not translation invariant. Left: A picture of a cat. Right: A deconstructed picture of a cat, with separate sections translated separately.

where r is applied element-wise to the resulting vector. Unlike convolutional and recurrent networks, feedforward models do not encode any specific structure, such as spatial or temporal dependencies.

2.3.2 Convolutional neural networks

Convolutional neural networks (CNN) are similar to feedforward neural networks, except they can encode spatial information via an additional "convolution". Intuitively, a convolution operation imposes translational invariance on the model. For example, an image classifier will be indifferent to an image that appears on the right or left.

To be concrete, we consider the input to be a gray-scale image represented as a matrix $\mathbf{I} \in \mathbb{R}^{L \times W}$ with length L and width W. Then, each layer h applies a collection of e_h matrices (referred to as filters), $\{\mathbf{F}_{h,e}\}_{e=1}^{e_h}$, to the image via the discrete convolution:

$$[\mathbf{F}_{h,e} * \mathbf{I}]_{i,j} = \sum_m \sum_n [\mathbf{I}]_{i,j} [\mathbf{F}_{h,e}]_{i-m,j-n}$$

where m and n range over the admissible index values of the filter. Notice that $\mathbf{F}_{h,e} * \mathbf{I} \in \mathbb{R}^{L \times W}$. We now define the notation that $\mathbf{F}_{h,:} * \mathbf{I} = \{\mathbf{F}_{h,e} * \mathbf{I}\}_{e=1}^{e_h} \in \mathbb{R}^{L \times W \times e_h}$. That is, we stack the results of each convolution for every available filter matrix. Then the CNN f is defined by the recursive convolutional relation:

$$\begin{aligned}
\mathbf{h_1} &= r(\mathbf{F}_{1,:} * \mathbf{I} + \mathbf{b_1}) \\
\mathbf{h_h} &= r(\mathbf{F}_{h,:} * \mathbf{h}_{h-1} + \mathbf{b_h}) && \text{for } h \geq 2 \\
f(\mathbf{I}) &= \mathbf{h}_H
\end{aligned}$$

where r is again applied element-wise. The convolutional operator and hence CNNs can be extended to multidimensional cases, such as video or fMRI.

2.3.3 Recurrent neural networks

Recurrent neural networks (RNN) are a class of neural networks that are specialized to handle sequential data. Similar to hidden Markov models, RNNs involve a hidden-state and an input at every time-step. However, the Markov assumption of fixed lag dependence can be limiting when time dependencies are unknown or unbounded. In fMRI for example, the presence of disease in an individual at a particular time step $\mathbf{y_t}$ can depend on all previous brain data, measured by Blood-level-oxygen dependent (BOLD) levels $\mathbf{x_t}$. The most naive application of a hidden Markov model would assume that the disease state only depends on the time step directly before. While a toy example, picking a fixed lag dependence in hidden Markov models is difficult. This difficulty is exacerbated by the computational difficulty of longer range dependencies and the impossibility of arbitrary length dependencies.

On the other hand, a recurrent neural network's hidden state h_t at time t depends on all the states and inputs before t. This is shown by the equations that define an RNN f for an input $\mathbf{x} = \{\mathbf{x_1}, \ldots, \mathbf{x_T}\}$ with T total time steps,

$$\begin{aligned}
\mathbf{a_t} &= \mathbf{b} + \mathbf{W}\mathbf{h_{t-1}} + \mathbf{U}\mathbf{x_t} \\
\mathbf{h_t} &= \tanh(\mathbf{a_t}) \\
\mathbf{o_t} &= \mathbf{c} + \mathbf{V}\mathbf{a_t} \\
f(\mathbf{x}) &= \mathbf{o}_T
\end{aligned}$$

weight matrices $\{\mathbf{W}, \mathbf{U}, \mathbf{V}\}$ and bias terms $\{\mathbf{b}, \mathbf{c}\}$. Notice that the hidden state $\mathbf{h_t}$ is a function of the activation $\mathbf{a_t}$, which again is a function of the previous hidden state $\mathbf{h_{t-1}}$ and input $\mathbf{x_t}$. This allows RNNs to model arbitrarily long dependencies in time. However, modeling very long range dependencies is still computationally costly and difficult to optimize. Despite these issues, they are extremely potent at modeling sequences [34].

One technical issue with recurrent neural networks is vanishing/exploding gradients, which makes capturing long term dependencies difficult. RNNs can be modified into Long-Short Term Memory (LSTM) models [40] or Gated Recurrent Unit (GRU) [19], which fix this issue by including a potential to "forget" irrelevant information and "reinforce" relevant information.

2.3.4 Bayesian neural networks

Neural networks can be made Bayesian, in the sense that we prescribe a prior distribution on the parameters (weights). While the weights themselves do not have any intrinsic meaning, the probabilistic tools provided by Bayesian reasoning allows for an Occam's razor effect via Bayesian modeling averaging [50]. Bayesian model averaging, which is the result of integrating over parameters, acts as an inherent regularization effect for the model class. If we include uncertainty in the neural network weights, we are able to provide more robust estimates with the posterior distribution over predictions. This line of study is often to referred to as Bayesian deep learning.

More concretely, we consider a feedforward neural network with prior assumptions on their weights, but without a bias term for notational convenience. For layer h of a neural network, we have that the weights $\mathbf{W_h}$ follow some prior

$p(\mathbf{W_h})$, typically taken to be isotropic Gaussian. The network is then defined recursively by an element-wise activation function r and an initial input $\mathbf{x_0}$. Then the first layer is calculated as $h_1 = r(\mathbf{W_1 x_0})$ and each subsequent layer follows $\mathbf{h_h} = r(\mathbf{W_h h_{h-1}})$ for $h = 1, \ldots, H$. Of course, Bayesian inference in this model is highly intractable due to the nonlinearities imposed by r.

Now, if we consider the model with unknown parameters $\mathbf{W}$ and observation pairs $\{\mathbf{x}, \mathbf{y}\}$ then we can write the posterior as

$$p(\mathbf{W}|\mathbf{x}, \mathbf{y}) = \frac{p(\mathbf{x}, \mathbf{y}|\mathbf{W})p(\mathbf{W})}{p(\mathbf{x}, \mathbf{y})}$$

where $p(\mathbf{W})$ is the prior distribution of all the weights, $p(\mathbf{x}, \mathbf{y})$ is the marginal of the data and $p(\mathbf{x}, \mathbf{y}|\mathbf{W})$ is the likelihood of the data. In particular, the likelihood term is difficult to calculate because each weight $\mathbf{W_h}$ depends nonlinearly on the previous weight $\mathbf{W_{h-1}}$, but must be integrated out. This makes any attempt at inference extremely difficult.

The entire problem of Bayesian deep learning is then how to reason about this uncertainty with the specific challenges that neural networks pose. There are two main approaches to this: casting ad-hoc tricks in deep learning as Bayesian inference [30, 45, 49] or developing methods that perform inference explicitly or implicitly [33, 12, 38, 46, 26].

2.3.5 Training and regularization methods

Neural networks provide a compelling and flexible modeling framework. The high degree of nonlinearity and non-convexity makes training difficult. As a result, many auxiliary methods were developed to assist in the training of neural networks. In this subsection, we will briefly outline the methods that we make use of: norm regularization of weights, dropout, and batch normalization.

2.3.5.1 Regularizing neural network weights with L_p–norm

Regularization can be imposed via an L_p norm on the weights similar to LASSO or Ridge regression [32]. This is a simple but effective method to avoid overfitting. In essence, the regularized cost function $\hat{J}(w)$ of the neural network, where the set of all weights is ω, includes a term $\|\omega\|_p$ in addition to the original cost function $J(\omega)$. Hence, the penalized cost function can be written as follows.

$$\hat{J}(\omega) = J(\omega) + \|\omega\|_p$$

There is little theoretical work on how exactly regularization effects the specific magnitude of the weights. Still, the added regularization term prevents weights from becoming too large and eases the optimization procedure.

2.3.5.2 Dropout

Regularization can also be induced by using dropout [67]: a pioneering regularization technique for deep learning. Roughly speaking, dropout in neural networks refers to the masking of individual neurons at the hidden layers (replacing neuron values with 0, with probability p). Hence, given a layer h with N_h neurons, $\mathbf{h} = \{h^1, \ldots, h^{N_h}\}$, we randomly set each h^i to 0 with probability p. This cleverly avoids overfitting by preventing any fixed subset of nodes in a neural network from overfitting to the data. The intuition for why this provides regularization is similar to the idea of ensemble modeling. That is, for each instantiation of a neural network with dropout, the size of the neural network is effectively reduced by the random zero masking. This encourages a robust parameterization because of the possibility that any nodes can be "turned off" in a neural network by being set to 0.

2.3.5.3 Batch normalization

Most recently, batch normalization was introduced as a method to enable even larger learning rates. Batch normalization [41] refers to the normalization of inputs at each layer h, $\hat{\mathbf{h}}_{\mathbf{h}} = \gamma_h \frac{\mathbf{h}_h - \mathbb{E}[\mathbf{h}_h]}{\sqrt{\mathrm{Var}[\mathbf{h}_h]}} + \beta_h$. When training is done over a mini-batch of size B, $\mathbf{x} = \mathbf{h_0} = \{\mathbf{h_0}^i\}_{i=1}^B$, then $\mathbb{E}[\mathbf{h}_h] \approx \mu_h = \frac{1}{B}\sum_{i=1}^B \mathbf{h}_h^i$ and $\sqrt{\mathrm{Var}[\mathbf{h}_h]} \approx \sigma_h = \sqrt{\frac{1}{B-1}\sum_{i=1}^B (\mathbf{h}_h^i - \mu_h)^2}$. The intuition on why this works is because it prevents hidden layers from drifting to arbitrarily large values, also known internal covariance shift. The result is an improvement in convergence speed and testing performance [41]. [70] shows that batch normalization can be interpreted as approximate variational inference on the weights of a deep neural network. However, the exact way that batch normalization improves neural network training is unclear.

2.4 Approximate inference

While Gaussian process classification is conceptually simple, the Bayesian inference problem is still intractable. In these situations, we must settle for approximate inference schemes which attempt to solve the intractable inference problem implicitly or through an auxiliary objective. There are two main approaches: sampling based methods such as Markov chain Monte Carlo (MCMC) or optimization based methods such as variational inference. Both methods have their advantages, however modern Bayesian deep learning methods has embraced variational inference despite its issues. The reason is simple: variational inference always terminates, giving a solution quickly [4].

The issue of convergence is a long standing issue with MCMC, where mixing is impossible to diagnose. That is, it is impossible to tell whether an MCMC based method has reached its steady state. Although MCMC is guaranteed to return a correct answer eventually, there is no guarantee on when and it is very difficult to tell if we achieved convergence [15]. The second issue, speed, can seen as the curse of dimensionality in MCMC. It becomes harder to traverse a high dimensional space with sampling methods, because we will be constantly rejecting proposals, which are pivotal for the asymptotic unbiased and low variance properties of MCMC.

2.4.1 Markov chain Monte Carlo

MCMC methods are sample based, meaning that they do not necessarily recover a functional form for the distribution of interest [10, 3]. Instead, a sequence of samples are taken and over time, the samples are guaranteed to come from the distribution of interest. Put formally, we take successive $\{x_i\}_{i=1}^{T}$ where $x_i \sim p_t(\cdot)$ form a Markov chain. Then, for some large T, the samples are guaranteed to come from the target distribution $p(\cdot)$. Hence, if we require S samples from a distribution, we would take the last S samples, $\{x_i\}_{i=T-S}^{T}$. Note that while you are guaranteed to converge to the target distribution *eventually* there is no guarantee on when that occurs. In fact, it could take arbitrarily long.

The simplest example of MCMC is Gibbs' sampling, which we will briefly review [17]. Consider an unknown joint distribution over D variables $p(X_1, \ldots, X_D)$, of which we are able to sample from the conditionals $p_i = p(X_i|\{X_j = x_j\}_{j\neq i})$. Then, using Gibbs' sampling, we can obtain samples from the joint as follows

Algorithm 2 An algorithm to Gibbs sample a joint distribution $p(x_1, \ldots, x_D)$ for a sample size of S

```
Input: $S, \{p_i\}_{i=1}^{D}$
Output: $\mathbf{x} \sim p(x_1, \ldots, x_D)$
  function GIBBS($S, \{p_i\}_{i=1}^{D}$)
      init $T = 0$
      while not converged do
          $T \leftarrow T + 1$
          for i = 1:D do
              $x_i^t \sim p(x_i|\{X_j = x_j\}_{j\neq 1})$
          end for
      end while
      $\mathbf{x} = \{x_1^t, \ldots, x_D^t\}_{t=T-S}^{T}$
  end function
```

While sampling is not an expensive task, the number of samples needed to explore a high dimensional space becomes exponential [60]. Hence, naive MCMC methods like Gibbs sampling are almost never used, and more advanced variants like Hamiltonian Monte Carlo can still struggle for the very high dimensional spaces in neural networks [6].

2.4.2 Variational inference

As we have seen, the Bayesian inference problem for Gaussian process classification is computationally difficult due to intractable integrals. Moreover, this problem exists in virtually any probabilistic model without conjugacy [11, 74]. When nonlinearities are introduced, as is the case with neural networks, the problem becomes only more difficult.

Variational inference (VI) is used to approximate inference for probabilistic models. The general idea is to bound a quantity of interest, say the marginal likelihood of the data, with a metric like KL-divergence. Conventionally, this will be a lower bound involving a simpler (proxy) distribution to approximate the intractable solution. Then, we optimize the parameters of the proxy distribution to improve the bound, and hence improve our approximate solution.

The general recipe outlined earlier can be made specific, detailing the many variants of VI, namely: Expectation Propagation [54], Variational Bayes [27], and the Expectation Maximization [55] algorithm. Specifically, Variational Bayes is a generalization of the Expectation Maximization algorithm, where the latter computes only point estimates of the posterior [7]. Recall that in Bayesian inference, the posterior of unknown variables $\mathbf{z}$ after seeing data $\mathbf{x}$ is written as follows.

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$$

In this case, the integral $p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z}) d\mathbf{z}$ is usually intractable, even with approximations like quadrature [11]. Without the density of the posterior, exact inference is clearly impossible. For the purpose of this review, we focus on variational Bayes with KL-divergence, in which we seek to approximate an unknown posterior $p(\mathbf{z}|\mathbf{x})$ with a simpler distribution $q_\phi(\mathbf{z})$, parameterized by ϕ:

$$KL(q_\phi(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) = \int q_\phi(\mathbf{z}) \log \frac{q_\phi(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})} d\mathbf{x}$$

Notice, the measure of closeness $KL(q||p)$ depends on $p(\mathbf{z}|\mathbf{x})$, the posterior that we do not know and are trying to approximate. However, one can write a lower bound on this quantity as follows.

$$\begin{aligned}
KL(q_\phi(\mathbf{z})||p(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}) \log \frac{q_\phi(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})} d\mathbf{x} \\
&= \int q_\phi(\mathbf{z}) \log q_\phi(\mathbf{z}) d\mathbf{x} - \int q_\phi(\mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) d\mathbf{x} \\
&= \int q_\phi(\mathbf{z}) \log q_\phi(\mathbf{z}) d\mathbf{x} - \int q_\phi(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}) d\mathbf{x} + \int q_\phi(\mathbf{z}) \log p(\mathbf{x}) d\mathbf{x} \\
&= E_{q_\phi}[\log q_\phi(\mathbf{z})] - E_{q_\phi}[\log p(\mathbf{z}, \mathbf{x})] + \log p(\mathbf{x})
\end{aligned}$$

Since $\log p(\mathbf{x})$ doesn't depend on q, the optimization depends only on the so-called "Evidence Lower BOund" or ELBO, $E_{q_\phi}[\log q_\phi(\mathbf{x})] - E_{q_\phi}[\log p(\mathbf{z}, \mathbf{x})]$. As the nomenclature suggests, the ELBO provides a lower bound on the "evidence" $p(\mathbf{x})$, since KL is a divergence and hence $KL(q_\phi||p) \geq 0$. This formulation is the basis of most variational techniques. The general recipe involves 1) an assumed distribution family that is simple, 2) a lower bound on the divergence of interest, which is constructed, 3) the inference problem becomes optimization.

This differs significantly from other approximate inference methods, including Monte Carlo sampling methods. Sampling based methods have the benefit of asymptotic convergence, but lack a measure of convergence in finite-time. On the other hand, variational methods have the benefit of modern optimization methods, including convergence diagnosis. Moreover, stochastic optimization provides an efficient way of dealing with high dimensions, while sampling in high dimensions is limited by the curse of dimensionality. For these reasons, variational inference is preferred over Monte Carlo methods despite no asymptotic guarantees. Indeed, variational methods are typically biased as the approximating family does not contain the distribution of interest.

Despite issues, variational methods are the dominant inference approach to Gaussian processes classification [37]. Specifically, variational methods are pervasive over MCMC due to scalability and ease of assessing convergence. Scalability is the largest consideration, since parameter spaces tend to be large. However, the difficulty of assessing whether a Markov chain has mixed is also very challenging. Hence, while neither MCMC nor variational methods are

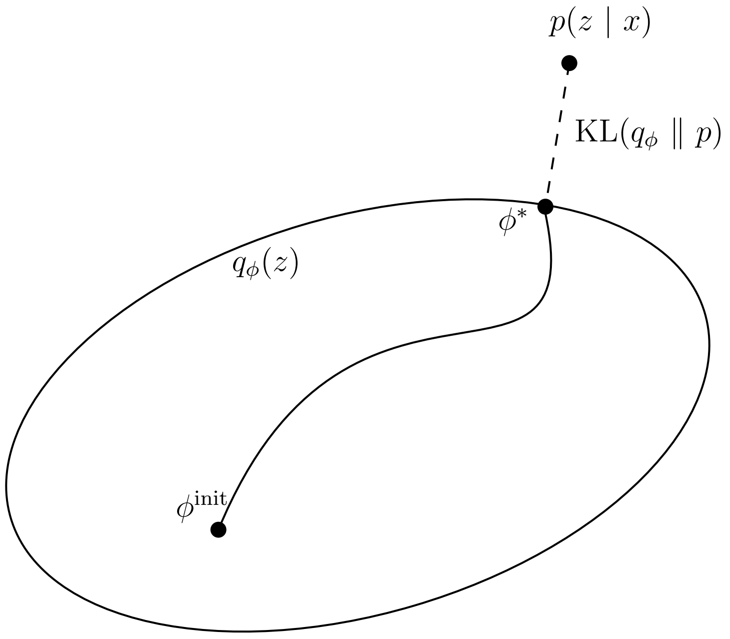

Figure 2.3: An illustration of variational inference, the family of distributions q, parameterized by ϕ, is optimized by minimizing the ELBO, which also minimizes KL divergence between the target and the posterior. Figure inspired by [11]

ideal, variational approaches address many of the challenges in machine learning. The largest issue with variational approaches is model misspecification in the variational posterior. Put simply, variational methods approximate the complicated posterior distribution with a simpler distribution and fit the simplified model by optimizing its parameters.

2.5 Challenges in neuroimaging

Perhaps the biggest issue in fMRI analysis is the dimension of the data p compared to the number of samples n. Gaussian processes are extremely effective in scenarios with large p and relatively small n. However, Gaussian processes are not widely used in the neuroimaging literature. This is possibly due to the difficulty of modeling the complex spatial and temporal dynamics of fMRI data. In contrast, Gaussian processes have been frequently applied to data that is well understood [68].

The spatiotemporal intricacies of raw fMRI data, which corresponds to a volume ($L \times W \times H$) over time T, are not well understood. This makes designing a kernel for Gaussian processes challenging. Furthermore, Gaussian process classification becomes increasingly cumbersome in multi-dimensional settings (where the data is a tensor, similar to a matrix but with more than two indices). While tensorized versions of Gaussian processes exist [84, 8, 69], they are largely inefficient since they cannot leverage sparse representations with inducing points. Alternatively, one could focus on structured Gaussian Processes with specific kernels [64]. However, this would involve a specific choice of kernel to combine information across dimensions.

2.5.1 Gaussian process classification

While Gaussian process are underutilized in fMRI, there have been attempts to apply them. In the context of regression and classification, [51] used Gaussian Processes to predict subjective pain intensity using whole-brain fMRI data. In their analysis, they leveraged the fact that Gaussian processes operate on the kernel matrix. They used a linear kernel on a “flatted” brain volume X, such that X has dimensions $n \times P$ where $P = L \times W \times H \times T$. Then, their analysis only considers the $n \times n$ matrix XX^T. However, this throws away all spatial and temporal information associated with the whole brain volume. Moreover, a linear kernel is naive even in simple cases, since it assumes the data separates linearly. Generally, Gaussian kernels or spectral mixtures therein, are preferable to model arbitrary stationary covariance structures.

Using a functional connectivity approach, the authors [18] used Gaussian Processes to classify Alzheimer patients using the empirical correlation matrix associated with fMRI data. In a similar manner, the authors [51] throw away spatial structure by only considering the covariance between predetermined ROIs. While it could be surmised that the covariance between spatially close ROIs would be high, it still presents an arbitrary simplification of an otherwise rich data structure. In addition, the kernel considered is still linear or Gaussian which may not be ideal in high dimensional spaces.

2.5.2 Classical methods

In contrast, Support Vector Machine (SVM) and Naive Bayes has seen much use in neuroimaging research [59]. For one, SVM provides a clear objective in maximizing the margin of the decision boundary. However, SVM fail to account for uncertainty in the data and model since they are not probabilistic. Furthermore, quadratic optimization is the dominant method of estimating an SVM, which becomes difficult in the presence of large data. Like Gaussian processes however, they require hand crafted representations of the data through a kernel. Naive Bayes also allows for some of the probabilistic benefits of Gaussian Processes, which enable interpretability. However, Naive Bayes tends to perform worse classification performance compared to SVM. Lastly, both SVM and Naive Bayes suffer from prescribed structure, instead of learning said structure from the data.

2.5.3 Neural network approaches

Neural networks have brought tremendous success to domains like computer vision, where data is abundant. On the other hand, little research has explored low data regimes with neural networks, such as those in health and neuroimaging. Still, there have been some attempts and discussion on using convolutional and feedforward neural networks [58, 53, 76]. Specifically, the authors [65] proposed using a convolutional neural network to classify 28 diseased patients versus 15 healthy subjects. They were able to achieve over 95% classification accuracy using images alone. Importantly, the neural network they used was pretrained on other visual data for classification and fine-tuned for the specific fMRI task. In addition, they concatenate across the spatial z and time t axes. This can make prediction difficult if the dependencies between the z and t axis are required to correctly classify diseased versus healthy patients.

Additionally, [39, 36] investigate the spatiotemporal dynamics of fMRI using recurrent neural networks. Importantly, their work focuses solely on modeling the dynamics, and not the predictive performance of their model. More recently, long short-term memory networks (a variant of recurrent networks) were used to identify Autism from resting state fMRI [24]. While this work is close in scope, the authors use 539 subjects with autism, and 579 controls. Notably, there have been no attempts at quantifying uncertainty nor using recurrent neural networks for classification.

Chapter 3

Recurrent and Bayesian Kernel Learning

Recently, the field of machine learning has found tremendous success in the use of neural networks for highly structured data. Neural networks, being efficient nonlinear function approximators, are able to learn representations of data that assist in classification and regression tasks. While deep learning (the study of neural networks) is an active area of research, there has been considerably less interest in deep learning for small data. In this chapter, we develop new methods that combine Gaussian processes with recurrent structure and uncertainty quantification to classify limited but highly structured data.

Small data problems are often ignored in the deep learning literature. After all, multimedia data like pictures and videos are easy to scrape from the internet and sufficient to benchmark neural network models. As a result, deep learning research tends to use readily available data and then explore performance at scale. In fields like neuroimaging, data is small is relative to the dimension of the data. Specifically, when the data dimension p is much larger than the amount of data available, then we say that we are in a small data regime, $n \ll p$. These issues are well explored in the classic statistical literature with regularization methods like LASSO [71]. As discussed earlier, the effect of L_p regularization on neural networks is not well understood. Bayesian non-parametric models such as Gaussian processes are also effective on small datasets [35]. Unlike neural networks however, Gaussian processes are limited by their kernel in their ability to capture complex nonlinear dependencies.

Revisiting the first question discussed in Chapter 1, “can we learn complex models from limited data”, we leverage sequential structure to improve performance for small (but sequential) data. Specifically, we propose a deep Bayesian model based on recurrent neural networks as an extension to kernel learning. Our model is able to capture temporal dependencies to effectively reduce the dimensionality of the problem. Unlike classical Gaussian process classification, our model uses recurrent neural networks, an expressive model of sequential structure.

Now, revisiting the second question, “can we reason about the uncertainties of complex models?”, we propose a new Bayesian deep learning approach that uses batch normalization to quantify the uncertainty. While the Gaussian process layer in deep kernel learning can provide uncertainty estimates at test-time, the estimates fail to account for uncertainty in the neural network weights. Further, deep kernel learning fails to propagate the uncertainty in neural network weights during training. We propose using a “bootstrapped” batch normalization procedure to regularize neural network models during training, while also providing more robust uncertainty estimates at test-time. We then explore this method through a proposed approximate probabilistic model and offer a new interpretation to batch normalization.

As our two approaches show, the questions - “can we learn complex models from limited data?” and “can we reason about the uncertainties of complex models?” - are intimately related. In Section 3.1, we encode sequential structure through recurrent neural networks in a deep Bayesian model to learn from limited data. In Section 3.2, we use our Bayesian deep learning procedure to obtain better uncertainty estimates in a deep Bayesian model. In both cases, we are leveraging Bayesian procedures. The only difference is whether it used as a form of regularization (in Section 3.1), or as a framework for uncertainty (in Section 3.2).

3.1 Going deeper with Gaussian processes

One way in which Gaussian processes can be made more complex is to compose them, in a similar fashion to the composition of nonlinear functions in neural networks. Unlike convolutional or feed-forward layers however, Gaussian processes must be integrated out.

Consider a hierarchical model of depth H in which the top most parent $g_0 \in \mathbb{R}^{D_0}$ takes $\mathbf{x}$ as an input. Then, g_0 is used as input to a Gaussian process, $g_1 \sim GP(0, \mathbf{K}_{\theta_1})$, $g_1 : \mathbb{R}^{D_0} \to \mathbb{R}^{D_1}$. This continues for each variable in the hierarchy, where each variable also follow a Gaussian process $g_h \sim GP(0, \mathbf{K}_{\theta_h})$ where $g_h : \mathbb{R}^{D_{h-1}} \to \mathbb{R}^{D_h}$. We can then define $G(\mathbf{x}) = g_H(\cdots g_1(g_0(\mathbf{x})))$. That is, the model is a composition of many Gaussian processes, or a deep Gaussian process [21]. To make inference tractable, each Gaussian processes must be integrated away. While this acts as a Bayesian Occam's razor, the process is computationally daunting.

Another way to incorporate the successes of deep learning is to parameterize kernels with neural networks. While some attempts have been made to make kernels more flexible, such as spectral kernels [79], only recently have researchers used deep neural networks in an end-to-end model [78, 80, 14, 16, 2].

Deep kernel learning [80, 78, 16, 2, 14] is the end-to-end learning of Gaussian process kernels $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ composed with deep neural networks $f_w : \mathcal{X} \to \mathbb{R}^d$ where the neural network is parameterized by the set of all its parameters, weights and biases w. As discussed in the Section 2.1.1, we can compose a kernel with a function f_w, a form of input embedding, such that

$$\hat{k}(\mathbf{x_i}, \mathbf{x_j}) = k(f_w(\mathbf{x_i}), f_w(\mathbf{x_j}))$$

It is trivial to show that, for any valid kernel function k, $\hat{k} = k \circ f_w$ is also a valid kernel. Specifically, we can consider any deep neural network architecture for f, such as feedforward or convolutional.

Then, we recall that the Gaussian process classification variational objective [37] can be written as:

$$\log p(\mathbf{y}) \geq E_{q(g)}[\log p(\mathbf{y}|g)] - KL[q(\mathbf{u})||p(\mathbf{u})]$$

where g is a Gaussian process. Recall that the inducing points $\mathbf{u}$ are essentially batches of pseudo-inputs that we can sample from a distribution that we choose, i.e. Gaussian $q(\mathbf{u}) = \mathcal{N}(\mathbf{m}, \mathbf{S})$ with mean $\mathbf{m}$ and covariance matrix $\mathbf{S}$. Then we have that,

$$q(g) = \mathcal{N}(\mathbf{Am}, \mathbf{K_{nn}} + \mathbf{A}(\mathbf{S} - \mathbf{K_{mm}})\mathbf{A^T})$$

where $\mathbf{A} = \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}$ and $\mathbf{K}_{nn}$ denotes the covariance matrix given by the kernel function evaluated at the data points, $[\mathbf{K}_{nn}]_{ij} = k(f_w(\mathbf{x_i}), f_w(\mathbf{x_j}))$. $\mathbf{K}_{nm}$ is the covariance matrix given by the kernel function evaluate at the data points with the inducing points, $[\mathbf{K}_{nm}]_{ij} = k(f_w(\mathbf{x_i}), f_w(\mathbf{u_j}))$. Lastly, $\mathbf{K}_{mm}$ is the covariance matrix given by the kernel function evaluated at the inducing points, $[\mathbf{K}_{mm}]_{ij} = k(f_w(\mathbf{u}_i), f_w(\mathbf{u}_j))$.

Hence, if we write $q(g)$ as $\int q(g|\mathbf{u})q(\mathbf{u})d\mathbf{u}$, we have an easy sampling mechanism. We first sample $\mathbf{u} \sim q(\mathbf{u})$ then sample $g \sim \int q(g|\mathbf{u})q(\mathbf{u})d\mathbf{u}$. We can then evaluate $E_{q(g)}[\log p(\mathbf{y}|g)]$ through either Monte-Carlo estimates or quadrature noting that

$$p(\mathbf{y}|g) = \prod_{n=1}^{N} (g_n)^{y_n}(1 - g_n)^{1-y_n}$$

and that

$$p(\mathbf{u}) = \mathcal{N}(0, \mathbf{K}_{mm})$$

Then, the KL divergence term is between two normal distributions, which can be calculated in closed form. Finally, because of our ability to sample, we can calculate gradients through the Gaussian process likelihood function and the neural network parameters to jointly optimize them.

3.1.1 Gaussian process classification with recurrent kernels

As discussed in Section 3.1, the idea of "deep-ifying" kernels is not new [42, 5]. Unlike previous work in recurrent kernel learning [2], recurrent classification is comparatively unexplored. To be concrete, we assume the data $\mathbf{x}$ has a dimension of d and time-series of length T, or $\mathbf{x} \in \mathbb{R}^{d \times T}$. Then a recurrent neural network, f performs the following recursive operation on the data $\mathbf{x}$,

$$\begin{aligned}
\mathbf{a_t} &= \mathbf{b} + \mathbf{W}\mathbf{h_{t-1}} + \mathbf{U}\mathbf{x}_{:,\mathbf{t}} \\
\mathbf{h_t} &= \tanh(\mathbf{a_t}) \\
\mathbf{o_t} &= \mathbf{c} + \mathbf{V}\mathbf{a_t} \\
f(\mathbf{x}) &= \mathbf{o_T}
\end{aligned}$$

That is, in a recurrent neural network, the data vector slice $\mathbf{x}_{:,t} \in \mathbb{R}^d$ is used as a sequential input at each time point. This sequential nature allows us to learn a representation for an arbitrarily complex autocorrelation structure. This is then used as input to a Gaussian process with kernel function k as follows.

$$\begin{aligned}
k_{new}(\mathbf{x}, \mathbf{x}') &= k(f(\mathbf{x}), f(\mathbf{x}')) \\
&= k(\mathbf{c} + \mathbf{V}(\mathbf{b} + \mathbf{W}\mathbf{h_{t-1}} + \mathbf{U}\mathbf{x}_{:,\mathbf{t}}), \mathbf{c} + \mathbf{V}(\mathbf{b} + \mathbf{W}\mathbf{h_{t-1}} + \mathbf{U}\mathbf{x}'_{:,\mathbf{t}}))
\end{aligned}$$

Since the Gaussian process classification problem is formulated as a variational lower bound, the entire problem is differentiable and can be optimized with stochastic gradient methods.

As an aside on Bayesian or stochastic neural networks, if f_w is stochastic and parameterized by w, then $k(f_w(\mathbf{x}), f_w(\mathbf{x}))$ is also a kernel. However, this is only true if for every instance of $f_w(\mathbf{x})$, $f_w(\mathbf{x})$ is evaluated once and fixed at a value f_w to ensure symmetry of $k(f_w, f_w)$. This becomes critical since Bayesian deep learning methods inherently inject noise into neural networks by considering the uncertainty of w, making $f_w(\mathbf{x})$ stochastic.

While the basic neural network structure is the same, recurrent connections introduce a host of problems in the context of Bayesian learning. First and foremost, since recurrent neural networks depend on the entire history, the optimization method becomes quite costly. Often, the back-propagation step must be truncated for some $t < T$. Rewriting the optimization objective in terms of the weights and biases $w = \{\mathbf{W}, \mathbf{V}, \mathbf{U}, \mathbf{c}, \mathbf{b}\}$ of a recurrent neural network, we have

$$Q(w) = \sum_{i=1}^{M} \underbrace{E_{q(g_i)}[\log p(\mathbf{y}_i|g_i)]}_{E_i} - \underbrace{\frac{1}{2}[\log|\mathbf{K}| - \log|\mathbf{S}| - M + tr(\mathbf{K^{-1}S}) + \mathbf{m^T K^{-1} m}]}_{L}$$

where $q(g) = \mathcal{N}(\mathbf{Am}, \mathbf{K_{nn}} + \mathbf{K_{nm}K_{mm}^{-1}}(\mathbf{S} - \mathbf{K_{mm}})(\mathbf{K_{nm}K_{mm}^{-1}})^{\mathbf{T}})$ and $q(g_i)$ is the marginal of $q(g)$. Then, $E_{q(g_i)}[\log p(\mathbf{y}_i|g_i)]$ is an integral of one variable that can be easily approximated through quadrature [37]. Now taking the derivative w.r.t. w of the expected likelihood term E_i, we get

$$\frac{\partial E_i}{\partial w} = \frac{\partial E_i}{\partial \mu}\frac{\partial \mu}{\partial w} + \frac{\partial E_i}{\partial \sigma}\frac{\partial \sigma}{\partial w}$$

Importantly, $\frac{\partial \mu}{\partial w}$ and $\frac{\partial \sigma}{\partial w}$ require computing derivatives of the kernel matrix evaluated at different pairs of points. Hence, this is more costly than gradient computation of a neural network that directly outputs the classification probability. However, both models are able to share weights, $\mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{c}, \mathbf{b}$, across each time step. This greatly reduces the number of parameters and reduces memory consumption. We hypothesize that this regularization in addition to the inherent regularization provided by Bayesian inference with Gaussian processes allows for good performance on small datasets.

3.2 Bayesian deep kernel learning for weight uncertainty

In the Bayesian deep learning literature, uncertainty in neural network weights is approached in one of two ways. One approach develops new training procedures that facilitate approximate inference schemes on neural network weights [33, 12, 38, 46, 26]. The other reinterprets successful methods in the deep learning literature as approximate inference [30, 45, 49].

More recently, batch normalization [41] has been reinterpreted as approximate variational inference [70]. In this work, we propose batch normalization as a way of propagating uncertainty in deep kernel learning. We show that this formulation naturally corresponds to variational inference on the neural network weights and we derive variance estimates that account for uncertainty in the neural network weights.

3.2.1 The case for and against dropout

Dropout is a popular method of regularizing neural networks by randomly setting values at each layer to 0, according to the dropout probability. This has been interpreted as a form of approximate variational inference on the neural network weights [30], enabling uncertainty estimates. Moreover, dropout has widespread use due to its intuitive implementation and computational ease. Producing Monte Carlo dropout estimates is quite easy, which allows practitioners to easily reason about the uncertainties of their neural network. However, dropout as a Bayesian approximation is not well understood. There are issues in that it does not account for additional uncertainties of influential points, and poor uncertainty estimates with fixed p [57, 31].

While the use of dropout is less common now, it is still occasionally used as a proxy for uncertainty. Still, dropout has its advantages in that it can be easily applied to recurrent neural networks [81]. This is as opposed to batch normalization, which introduces large overhead with little gain in comparison to dropout [29]. Currently, there has been no investigation in the effect of these different neural network architectures. In our informal study, dropout decreased training and testing performance even for small dropout probabilities and was catastrophic at higher dropout probabilities.

3.2.2 Batch normalized deep kernel learning

Batch normalization [41] refers to the normalization of inputs at each layer h, $\hat{\mathbf{h}}_{\mathbf{h}} = \gamma_h \frac{\mathbf{h}_h - \mathbb{E}[\mathbf{h}_h]}{\sqrt{\mathrm{Var}[\mathbf{h}_h]}} + \beta_h$. When training is done over a mini-batch of size B, $\mathbf{x} = \mathbf{h_0} = \{\mathbf{h_0}^i\}_{i=1}^B$, then $\mathbb{E}[\mathbf{h}_h] \approx \mu_h = \frac{1}{B}\sum_{i=1}^B \mathbf{h}_h^i$ and $\sqrt{\mathrm{Var}[\mathbf{h}_h]} \approx \sigma_h = \sqrt{\frac{1}{B-1}\sum_{i=1}^B (\mathbf{h}_h^i - \mu_h)^2}$. [70] shows that batch normalization can be interpreted as approximate variational inference on the weights of a deep neural network.

We consider a batch normalized neural network f_w where w is the set of all parameters, weights and biases of the neural network. Then, the Gaussian process classification variational objective [37] can still be written as:

$$\log p(\mathbf{y}) \geq E_{q(\mathbf{u},w)}[\log p(\mathbf{y}|\mathbf{u}, w)] - KL[q(\mathbf{u}, w)||p(\mathbf{u}, w)]$$

Using the inequality $\log p(\mathbf{y}|\mathbf{u}, w) \geq E_{p(f|\mathbf{u},w)} \log(\mathbf{y}|g))$, we get that

$$\log p(\mathbf{y}) \geq E_{q(g)}[\log p(\mathbf{y}|g)] - KL[q(\mathbf{u}, w)||p(\mathbf{u}, w)]$$

where we wrote $q(g) = \int p(g|\mathbf{u}, w)q(\mathbf{u}, w)d\mathbf{u}dw = \int p(g|\mathbf{u}, w)q(\mathbf{u}|w)d\mathbf{u}\, q(w)dw$. We use the fact that $q(g|w) = \int p(g|\mathbf{u}, w)q(\mathbf{u}|w)d\mathbf{u} = \mathcal{N}(\mathbf{Am}, \mathbf{K_{nn}} + \mathbf{A}(\mathbf{S} - \mathbf{K_{mm}})\mathbf{A^T})$ where $\mathbf{A} = \mathbf{K_{nm}}\mathbf{K_{mm}^{-1}}$ and $\mathbf{K}_{nn}$ denotes the covariance matrix given by the kernel function evaluated at the data points, $[\mathbf{K}_{nn}]_{ij} = k(f_w(\mathbf{x}_i), f_w(\mathbf{x}_j))$. $\mathbf{K}_{nm}$ is the covariance matrix given by the kernel function evaluate at the data points with the inducing points, $[\mathbf{K}_{nm}]_{ij} = k(f_w(\mathbf{x}_i), f_w(\mathbf{u}_j))$. Lastly, $\mathbf{K}_{mm}$ is the covariance matrix given by the kernel function evaluated at the inducing points, $[\mathbf{K}_{mm}]_{ij} = k(f_w(\mathbf{u}_i), f_w(\mathbf{u}_j))$. Then we are left with $\int q(g|w)q(w)dw$. This is intractable in general, due to the nonlinearities used in deep neural networks. However, under the stochastic sampling of mini-batches, we are implicitly sampling from the weights w. Hence, batch normalization facilitates an easy way of sampling $\int q(g|w)q(w)dw$, by sampling mini-batches and evaluating the kernel function given a mini-batch.

Turning our attention to the KL term, notice that $KL[q(\mathbf{u}, w)||p(\mathbf{u}, w)] = E_{q(w)}[KL[q(\mathbf{u}|w)||p(\mathbf{u}|w)]] + KL[q(w)||p(w)]$. The first term is easy to evaluate, since it is implicitly evaluated in the sampling of mini-batches. The second term, $KL[q(w)||p(w)]$, poses an issue since we do not explicitly specify a variational family $q(w)$ over w. If we had this term, we would have a correspondence between variational inference of the neural network weights, and batch normalized training. In Section 3.2.2.1, we derive a candidate distribution for $q(w)$ and show how this can be used to calculate the KL divergence.

The predictive distribution with weight uncertainty closely follows that of [37]. If we want to predict a test point x^*, then we need the posterior of the

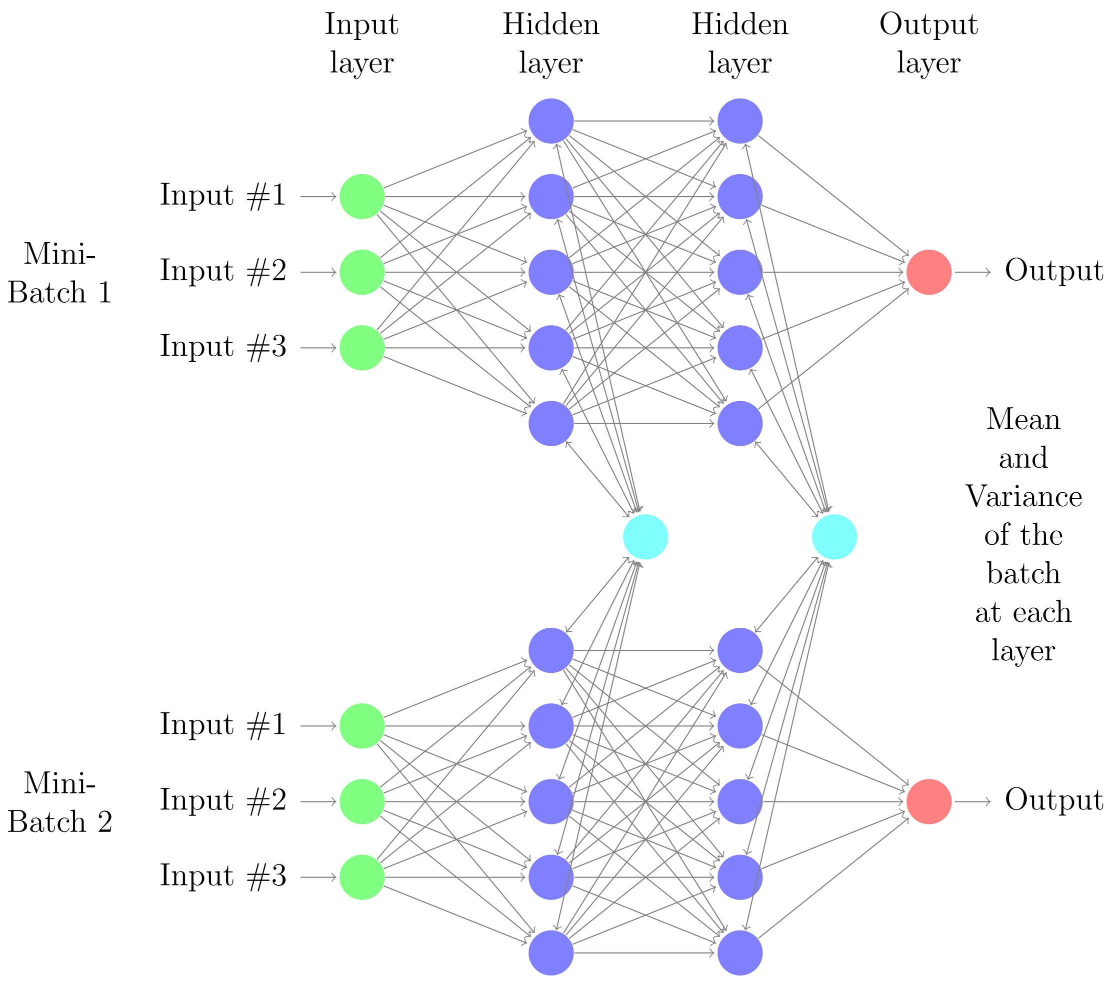

Figure 3.1: By sampling a mini-batch, a neural network is implicitly sampled from a distribution for that mini-batch. The effective value of the neural network weights depends on the stochastic mini-batch statistics calculated at each layer of the neural network.

latent Gaussian process value f^*.

$$\begin{aligned} p(f^*|\mathbf{y}) &= \int p(f^*|f, \mathbf{u}, w) p(f, \mathbf{u}, w|\mathbf{y}) df d\mathbf{u} dw \\ &\approx \int p(f^*|f, \mathbf{u}, w) p(f|\mathbf{u}, w) q(\mathbf{u}|w) q(w) df d\mathbf{u} dw \\ &= \mathbb{E}_{q(w)} \left[\int p(f^*|\mathbf{u}, w) q(\mathbf{u}|w) d\mathbf{u} \right] \end{aligned}$$

The mean and variance of $f|w$, denoted by μ_w^2 and σ_w, is tractable with respect to the distribution $p(f^*|w) = \int p(f^*|\mathbf{u}, w) q(\mathbf{u}|w) d\mathbf{u}$ for given w. Then by sampling M mini-batches and denoting the effectively sampled weights as $\hat{w}_i$, we get that

$$\begin{aligned} Var(f^*) &= \mathbb{E}_{p(f^*|\mathbf{y})}[(f^*)^2] - (\mathbb{E}_{p(f^*|\mathbf{y})}[f^*])^2 \\ &= \frac{1}{M} \sum_{i=1}^{M} \left[\mathbb{E}_{p(f^*|\hat{w}_i)}[(f^*)^2) \right] - \left[\frac{1}{M} \sum_{i=1}^{M} \mathbb{E}_{p(f^*|\hat{w}_i)}[f^*] \right]^2 \\ &= \frac{1}{M} \sum_{i=1}^{M} \left[\sigma_{\hat{w}_i}^2 + \mu_{\hat{w}_i}^2 \right] - \left[\frac{1}{M} \sum_{i=1}^{M} \mu_{\hat{w}_i} \right]^2 \end{aligned}$$

3.2.2.1 Batch normalization as a variational approximation

Now, we return to the problem of defining a suitable distribution over the weights $q(w)$. This is defined implicitly by the batch normalization procedure. In order to calculate the KL divergence $KL(q(w)||p(w))$ however, we derive plausible distributions for $q(w)$. Recall the equation for batch normalization, where at each layer h we define the batch norm layer as follows

$$\hat{\mathbf{h}}_h = BN(\mathbf{h}_{h-1}; \gamma_h, \beta_h) = \gamma_h \frac{g(\mathbf{h}_{h-1}) - \mathbb{E}[g(\mathbf{h}_{h-1})]}{\sqrt{\mathrm{Var}[g(\mathbf{h}_{h-1})]}} + \beta_h$$

When training is done over a mini-batch of size B, $\mathbf{x} = \mathbf{h}_0 = \{\mathbf{h}_0^i\}_{i=1}^B$, then $\mathbb{E}[\mathbf{h}_h] \approx \mu_h = \frac{1}{B} \sum_{i=1}^B \mathbf{h}_h^i$ and $\sqrt{\mathrm{Var}[\mathbf{h}_h]} \approx \sigma_h = \sqrt{\frac{1}{B-1} \sum_{i=1}^B (\mathbf{h}_h^i - \mu_h)^2}$.

Specifically, the batch norm layer can be inserted before or after nonlinear function r. In the case that it is done before the activation, $g(\mathbf{h}_{h-1}) = \mathbf{W}\mathbf{h}_{h-1}$. The output for that layer is $\mathbf{h}_h = r(\hat{\mathbf{h}}_h)$, where r is the element-wise nonlinearity. If done after the activation layer, then $\gamma_h = 1$ and $\beta_h = 0$ can be fixed and $g(\mathbf{h}_{h-1}) = r(\mathbf{W}\mathbf{h}_{h-1})$. Moreover, $\mathbf{h}_h = \hat{\mathbf{h}}_h$. We focus on the case where batch norm is used before activation, as is done in [41].

Hence, we have that

$$\hat{\mathbf{h}}_h = BN(\mathbf{h}_{h-1}; \gamma_h, \beta_h) = \gamma_h \frac{\mathbf{W}\mathbf{h}_{h-1} - \mathbb{E}[\mathbf{W}\mathbf{h}_{h-1}]}{\sqrt{\text{Var}[\mathbf{W}\mathbf{h}_{h-1}]}} + \beta_h$$

Consider the case where mini-batches are sampled stochastically. Hence, the parameters depending on the data, μ_h and σ_h become random variables. Now observe the following decomposition:

$$\hat{\mathbf{h}}_h = BN(\mathbf{h}_{h-1}; \gamma_h, \beta_h) = \gamma_h \frac{\mathbf{W}\mathbf{h}_{h-1} - \mu_h}{\sigma_h} + \beta_h$$

Where the bias parameter is implicit in the weight matrix and augmenting the data input by prepending a vector of 1s, i.e. $\mathbf{W} := [\mathbf{b}, \mathbf{W}]$ and $\mathbf{h} := [\mathbf{1}, \mathbf{h}]$. Then, appealing to the central limit theorem for sampling distributions, we can pick priors, $\mu_h \sim \mathcal{N}(\mu_h^*, \mathbf{I})$ for some constant μ_h^*, and $\sigma^2 \sim \chi^2(M-1)$, and then we have that:

$$\sqrt{M-1}\frac{\mathbf{W}\mathbf{h}_{h-1} - \mu_h}{\sigma_h} \sim T(M-1)$$

where T is a student-T distribution. Then, including the trainable parameters for the batch normalization layer we have that:

$$\gamma_h \frac{\mathbf{W}\mathbf{h}_{h-1} - \mu_h}{\sigma_h} + \beta_h \sim \frac{\gamma_h}{\sqrt{M-1}} T(M-1) + \beta_h$$

which is a location-scale student-T distribution. Independently, this can be rewritten as a decomposition over the multiplicative and additive random variables.

$$\begin{aligned} BN(\mathbf{h}_{h-1}; \gamma_h, \beta_h) &= \gamma_h \frac{\mathbf{W}\mathbf{h}_{h-1} - \mu_h}{\sigma_h} + \beta_h \\ &= \left[\gamma_h \frac{\mathbf{W}\mathbf{h}_{h-1}}{\sigma_h}\right] + \left[\beta_h - \gamma_h \frac{\mu_h}{\sigma_h}\right] \end{aligned}$$

We can then rewrite the above expression in a tidier form

$$BN(\mathbf{h}_{h-1}; \gamma_h, \beta_h) = \hat{\mathbf{W}}\mathbf{h}_{h-1} + \hat{\mathbf{b}}$$

where:

$$\hat{\mathbf{W}}_h = \gamma_h \frac{\mathbf{W}}{\sigma_h}$$

and

$$\hat{\mathbf{b}}_h = \beta_h - \gamma_h \frac{\mu_h}{\sigma_h}$$

Notably, since μ_h and σ_h are random, the corresponding weights ($\hat{\mathbf{W}}_h$) and biases ($\hat{\mathbf{b}}_h$) are also random. Hence, we can define a prior over $\hat{\mathbf{W}}_h$ and $\hat{\mathbf{b}}_h$ by defining priors over μ^l and σ^l. Again, we appeal to the central limit theorem for sampling distributions and use the same priors: $\mu_h \sim \mathcal{N}(\mu_h^*, \mathbf{I})$ for some constant μ_h^* and $\sigma^2 \sim \chi^2(M-1)$,

Then we have that $\hat{\mathbf{W}}_h \sim$ scale-inv-$\chi^2(M-1, \frac{1}{\gamma_h \mathbf{W}})$ and $\hat{\mathbf{b_h}} \sim \beta_h - \gamma_h \text{nct}(M-1, \mu_h^*)$ where nct is the non-central student-T distribution. With these distributions, we can calculate the KL-divergence using Monte Carlo sampling with rejection. However, it is important to note that the parameters are not independent of each other. Still, this analysis shows how batch normalized deep kernel learning provides an approximation to Bayesian learning of the neural network parameters.

Chapter 4

Applications

In this chapter, we evaluate the two proposed models outlined in Chapter 3. First, we perform a simulation study to benchmark the performance of standalone Gaussian process, neural network against our hybrid models. The simulation study attempts to model time-series phenomena that occur in neuroimaging data. We specifically focus on how the covariance matrix of the data generating process differs between our simulated "disease" patients and healthy controls. This first step will provide insight on the performance of our models, which we further investigate in Section 4.2.2 and 4.2.3.

Next we evaluate the uncertainty provided by the batch normalized kernels for convolutional neural networks, as we outlined in Section 3.2. Specifically, we compare the performance of deep kernel learning with and without neural network kernels, and with and without batch normalization. Our experiments are conducted on the MNIST (Modified National Institute of Standards and Technology) [47] data set, which are 28×28 gray-scale images of digits. This dataset is regarded to be a standard benchmark in the machine learning community and will provide the baseline for non-temporal models.

Lastly, we apply the models discussed to real fMRI datasets and attempt to classify dyslexia patients from healthy controls. That is, looking at resting-state fMRI data alone, we attempt to diagnose each subject disease status. Some challenges in neuroimaging are outlined before we describe the two datasets that we investigate. In addition, we make use of a null dataset in which all the patients are the healthy controls. This additional dataset helps identify failure modes of our model, such as over-confidently extrapolating from bad data.

For all of our real data analyses, we use a 5-fold cross validation process to calculate the context specific loss (such as likelihood), as well as the error and the accuracy in general. We aim to correctly classify diseased patients from controls. Empirical performance is measured by calculating accuracy and mean squared error (MSE) on a hold-out set $\{y_i^{\text{test}}\}_{i=1}^{n_{test}}$ of size n_{test} after training on a set of size n, and producing a set of estimates $\{y_i^{\text{estimate}}\}_{i=1}^{n_{test}}$.

$$MSE = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} (y_i^{\text{test}} - y_i^{\text{estimate}})^2$$

$$ACC = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \mathbb{1}_{y_i^{test} = y_i^{\text{test}}}$$

where $\mathbb{1}_{x=y} = 1$ if and only if $x = y$ and 0 otherwise. For simulated data, the process is not cross-validated and instead repeated 5 times, with 5 uniquely generated data sets.

All of our implementations use Tensorflow [1] for neural network components and GPflow [52] for Gaussian process components. Tensorflow is a common deep learning library that implements automatic differentiation and optimization methods for various neural network architectures. While GPflow is a Gaussian process framework built on top of Tensorflow. All of our optimizations were done via the stochastic gradient descent optimizer ADAM [44] with an informal search over learning rate, starting with the default of $\alpha = 0.001$.

4.1 Data descriptions

4.1.1 Simulation data

The goal of this simulation is to see how the tuple of data dimension and sample size, or (p, n), affects the results of the classification models. We hypothesize that Gaussian processes work well out of the box while deep learning methods, after fine-tuning, will outperform them as p increases relative to n. Finally, we hypothesize that our model is best suited in the case where n is much smaller than p, $n << p$. We explore all combinations of $n = \{8, 16, 32.64\}$ and $p = \{10, 50, 100, 250, 500\}$.

We consider the baseline GP models: Gaussian process on estimated covariance (Covariance-GP) and Gaussian process on the entire dataset (Reshape-GP). We use an RBF kernel, after finding that standard kernel choices do not have a large impact on the result.

For the baseline deep neural network models, we use both convolutional (CNN) and recurrent neural networks (RNN). The convolutional model has 4 layers, with the following number of filters per layer: (64,32,16,2). Other parameters relevant to Tensorflow are set to the default, including a kernel size of 3. The recurrent neural network follows has two hidden layers, each with 64 units. We also use GRU cells, which are a slightly modified but more efficient and stable variant of LSTM. We also note that a dropout value of 0.5 was used in both neural network models to regularize them as discussed in Section 2.3.5.2.

Lastly, we consider the following hybrid models: Gaussian process on estimated covariance with a convolutional kernel (Deep-Covariance GP) as above and Gaussian process on data with a recurrent kernel (Deep-GP). The deep kernel is parameterized by the same parameters in both cases, to avoid any confounding effects. Additionally, we use an RBF kernel similar to the standalone GP models describe earlier in this section.

We simulate a p-dimensional vector autoregressive, or VAR(1), process for n controls and n diseased subjects, for a balanced study of a total $2n$ subjects. We then explore the performance of the models outlined earlier under six different scenarios. First, we explore the null case where there is no difference in the two data generating processes. Second, we generate two different covariance matrices for the controls and diseased patients. Third, we use the same covariance matrix but set more values to zero for diseased subjects, leading to higher sparsity levels. Fourth, we use exactly the same covariance matrix but use different AR coefficients for diseased and control subjects. Fifth, we introduce a spike process in the form of a binomial distribution with different activation levels (probability q) between diseased and control subjects.

Specifically, a p-dimensional VAR(1) process is defined as

$$\mathbf{x}_t = a\mathbf{x}_{t-1} + \epsilon_t$$

where for all t, $\mathbf{x}_t \in \mathbb{R}^p$ and ϵ_t is drawn from a zero-mean Gaussian distribution $\epsilon_t \sim \mathcal{N}(0, \mathbf{K})$ with some $p \times p$ covariance matrix $\mathbf{K}$. Notably, the noise process ϵ_t is independent across time but specifies the covariance structure between vector components $\mathbf{x_t} = \{\mathbf{x}_t^i\}_{i=1}^p$ at each time point.

Every model was trained for 1000 iterations, where the models using deep neural networks took longer to converge for small p. For larger p, the Gaussian processes were also slow to converge. The experiments were repeated twice and the results were averaged together. We also note that the models do not make a class decision. Instead, they report probabilities of disease for a better sense of performance. This is more informative in evaluating the models, because it shows when models "know what they don't know". For example, if a model true for the diseased patient with probability of 0.51, the class decision is correct but we should be skeptical of the model.

4.1.1.1 No difference between two groups

We first consider a null simulation, where each group is simulated from the same p-dimensional $VAR(1)$ process, with a coefficient of $a = 0.95$. The randomly sampled covariance matrix $\mathbf{K} = \mathbf{A^T A}$, where $\mathbf{A}_{i,j} \sim \mathcal{N}(0, 1)$.

4.1.1.2 Different covariance matrices

Next we consider a simulation, where each group has a different covariance matrix for their p-dimensional $VAR(1)$ process, with a coefficient of $a = 0.95$. That is, we randomly sample two covariance matrices $\mathbf{K}_d = \mathbf{D^T D}$ and $\mathbf{K}_c = \mathbf{C^T C}$ where $\mathbf{C}_{i,j} \sim \mathcal{N}(0, 1)$ and $\mathbf{D}_{i,j} \sim \mathcal{N}(0, 1)$ independently.

4.1.1.3 Same covariance, but different sparsity levels

In this simulation, we use the same base covariance but randomly set some entries to zero to vary sparsity levels. Hence, each group is simulated from a different p-dimensional $VAR(1)$ process, with a coefficient of $a = 0.95$. The covariance matrices $\mathbf{K}_d = \mathbf{D}^{\mathbf{T}}\mathbf{D}$ and $\mathbf{K}_c = \mathbf{C}^{\mathbf{T}}\mathbf{C}$ are randomly sampled where $\mathbf{A}_{i,j} \sim \mathcal{N}(0,1)$. Then, we obtain the covariance matrices $\mathbf{C} = \mathbf{A}_{i,j}c$ and $\mathbf{D} = \mathbf{A}_{i,j}d$ where $c \sim \text{Bernoulli}(p_c)$ and $d \sim \text{Bernoulli}(p_d)$. Specifically, we set $p_c = 0.05$ and $p_d = 0.15$.

4.1.1.4 Different AR coefficients

Next, we consider a null simulation, where each group is simulated from the same p-dimensional $VAR(1)$ process. The randomly sampled covariance matrix $\mathbf{K} = \mathbf{A}^{\mathbf{T}}\mathbf{A}$, where $\mathbf{A}_{i,j} \sim \mathcal{N}(0,1)$, however their AR coefficients differ. That is, each group follows a process defined by

$$\mathbf{x}_t = a\mathbf{x}_{t-1} + \epsilon_t$$

where $a = 0.95$ for healthy controls and $a = 0.85$ for the diseased group.

4.1.1.5 Randomly occurring Bernoulli spikes

Lastly, we look at the case where each group is simulated from the same p-dimensional $VAR(1)$ process, but with different rates for an additive Bernoulli term. The randomly sampled covariance matrix $\mathbf{K} = \mathbf{A}^{\mathbf{T}}\mathbf{A}$ are still distributed as $\mathbf{A}_{i,j} \sim \mathcal{N}(0,1)$. Instead, we have the following VAR(1) process:

$$\mathbf{x}_t = a\mathbf{x}_{t-1} + \epsilon_t + B$$

where $B \sim \text{Bernoulli}(q)$, $q = 0.6$ for healthy controls and $q = 0.4$ for the diseased group.

4.1.2 Machine learning data

We now turn our attention to a common machine learning benchmark. While neural networks have a natural advantage over traditional models, these benchmarks are pivotal to evaluate large sample size performance. Indeed, we would like to produce a model that performs adequately in both large and small data scenarios, or at the very least a model that knows what it does not know.

4.1.2.1 MNIST data

We consider a binary classification task on the MNIST dataset between even and odd digits, and benchmark deep kernel learning, with (DKL-BN) and without batch normalization (DKL), against standalone deep neural networks with (DNN-BN) and without (DNN) batch normalization. The neural network architecture is the same for all models. However, deep kernel learning models have a final Gaussian process layer, while deep neural networks have an extra feed forward layer. Specifically, our convolutional model has 4 layers, with the following number of filters per layer: (64,32,16,2). Other parameters relevant to Tensorflow are set to the default, including a kernel size of 3.

4.1.3 fMRI data

Neuroimaging technology measures some proxy of neuronal activity, typically in a noninvasive fashion. Functional Magnetic Resonance Imaging (fMRI) is one type of technology that infers the Blood Oxygenated Level Dependent (BOLD) signal. As a result, fMRI is able to identify areas of the brain that have elevated activity, since brain activity requires a heightened level of oxygenation. Importantly, the BOLD signal is measured over time, where the patient (the one undergoing the fMRI scan) may be asked to rest or to perform a task during this time [48].

In the earlier simulations, we noted how imperative recurrent and temporal relations were in the study of time-series. Hence, we only use recurrent kernels in the investigate of fMRI data. Unfortunately, the data from fMRI experiments is even more higher dimensional than our simulations. Still,we hope that the performance in low sample sizes in our simulations will translate to real world data. In this situation, neural networks can overfit the data. However, recurrent models share most of their parameters across time and induce temporal regularization, which should help prevent overfitting. Nonetheless, we leverage our ability to use dropout in this model and set our dropout rate at 0.5.

One substantial issue with the use of recurrent neural networks is their computational cost. Specifically, when used in the kernel of a Gaussian process, their computational cost is becomes even larger. This is due to the inherent recurrent connections. Although there is a large amount of parameter sharing, taking the gradient becomes expensive for time-series with large T. While there exists some mitigating approaches, these are not straight forward when the RNN is used as input to a Gaussian process. However, we hope that the performance gains for small n is worth the increased computational cost, especially in fMRI where data is relatively scarce.

An additional metric that we use is the Receiving Operating Characteristic (ROC). That is, given predictions from a model, the ROC measures how the true and false positive rates relate to one another when the decision threshold is varied. As noted earlier, the accuracy during training was measured using a threshold of 0.5. Here, we will investigate a range of automatically chosen thresholds. We do not average the ROC curves, instead we plot the ROC curve for every fold in our cross validated results. Since uncertainty is of interest, this shows the degree of variation between cross-validation folds.

As mentioned at the beginning of this chapter, all of our real data analyses are cross-validated with 5 folds.

4.1.3.1 Null dataset

In this experiment, we attempt to classify patients based on preprocessed fMRI data that has been mapped to an atlas. The data set [20] involves 45 patients, each of which were scanned for 285 time points with 31 regions of interest (ROIs). This corresponds to the null dataset of an Alzheimer's study, where none of the patients have been diagnosed with the disease. Still, we split the data in partitions of 20 which we label diseased and 25 which we label controls. That is, 20 individuals that are actually healthy controls are labeled as "diseased" patients. Hence, we are testing to see if our model will over-confidently extrapolate small differences between the various healthy controls.

Our data is in the form of a multi-dimensional array, or *tensor* $X \in$

$\mathbb{R}^{45\times31\times285}$ and a binary response vector of disease labels $y \in \{0,1\}^{45}$, with 1 being presence of the disease.

4.1.3.2 Dyslexia dataset

This dataset corresponds to an fMRI study on dyslexia [25] containing 9 controls and 14 patients exhibiting dyslexia participated. The total number of patients is then $n = 23$, with 11 ROIs and 200 time points. Hence, we have a data *tensor* $X \in \mathbb{R}^{23\times11\times200}$ and a binary response vector of disease labels $y \in \{0,1\}^{23}$, with 1 being presence of the disease.

4.2 Results

4.2.1 Simulation study

4.2.1.1 No difference between two groups

When there are no differences between two groups and the labels are random, we want the MSE value to be at 0.25 with low variance. This is difficult for more powerful models that extrapolate from small noisy differences between the artificially created groups. From the results shown in Figure 4.1, we see that the GP model indeed stays at an MSE value of 0.25. The other model, Covariance-GP hovers between 0.25 and 0.3, which is a respectable level for the null experiment. Both deep models perform poorly. The CNN model in particular performs very poorly, as to be expected as convolutional models are not well suited to time-series problems. We see that an increase in p for fixed $n = 8$ results in a decrease in performance for the RNN, which is also expected since deep models require data to train properly and perform best when $n >> p$. Finally, we see that the deep GPs performs better as p increases. While the Deep-Covariance-GP begins quite strongly but it begins to perform worse as p increases.

Now referring to Figure 4.2 we see a similar trend that the GP-reshape model consistently predicts disease with a probability of 0.5 even after 1000 iterations. The CNN continues to perform poorly but this time improves

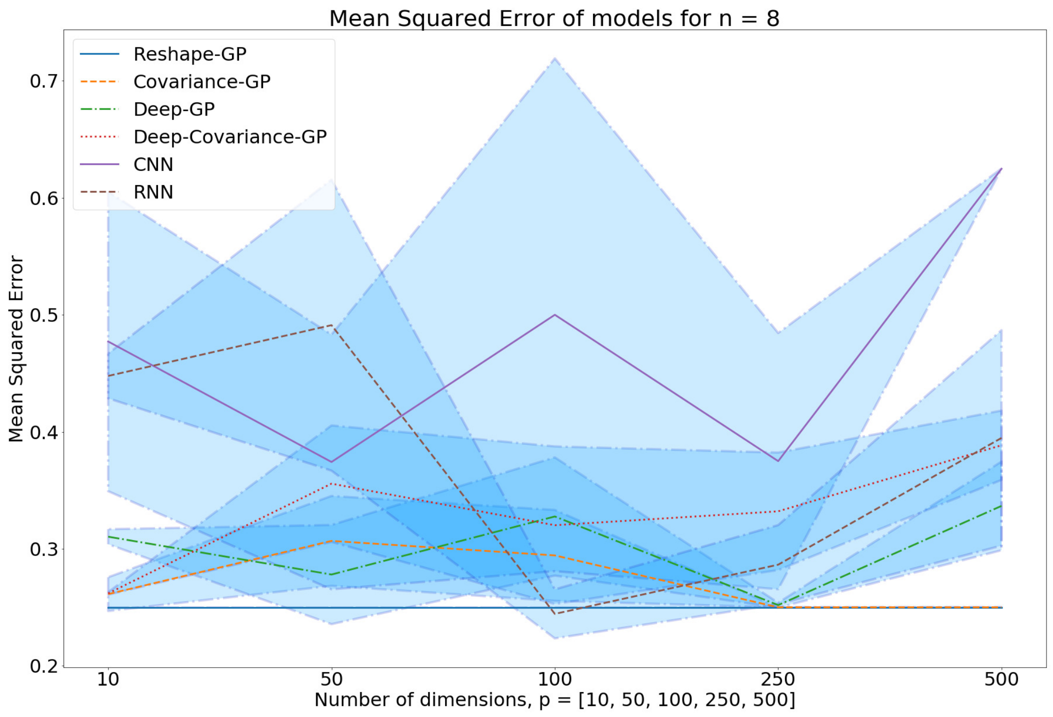

Figure 4.1: Mean Squared Error in the null case for Gaussian process, hybrid and deep models with $n = 8$ and $p = 10, 50, 100, 250, 500$

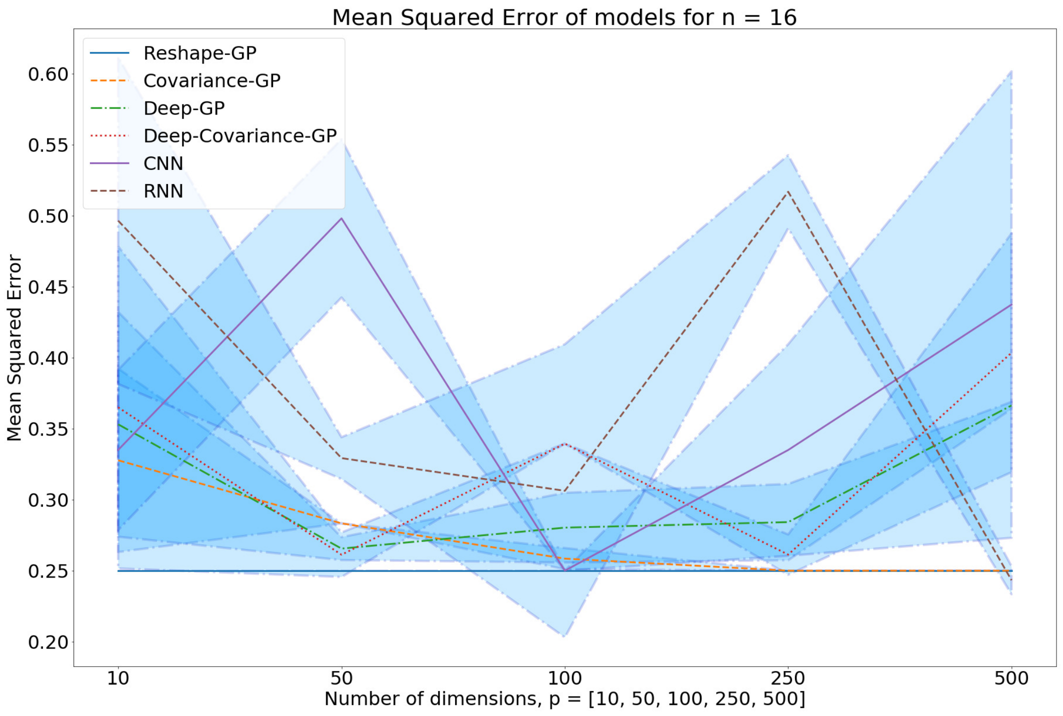

Figure 4.2: Mean Squared Error in the null case for Gaussian process, hybrid and deep models with $n = 16$ and $p = 10, 50, 100, 250, 500$

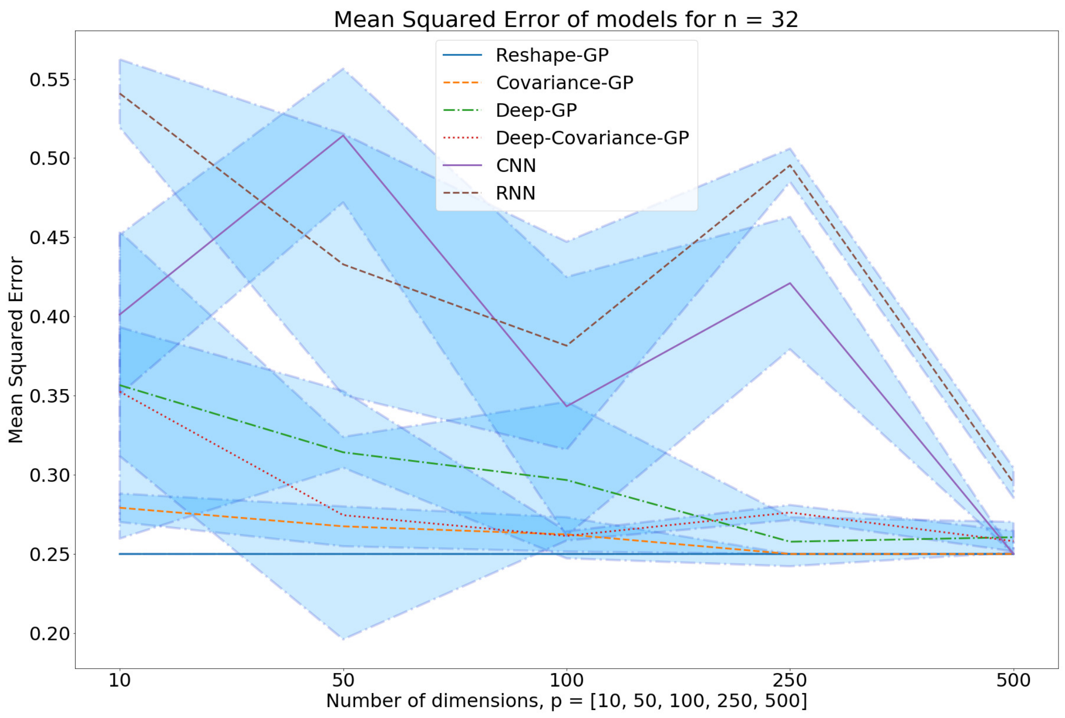

Figure 4.3: Mean Squared Error in the null case for Gaussian process, hybrid and deep models with $n = 32$ and $p = 10, 50, 100, 250, 500$

as p increases, still staying behind the traditional GP models. The Deep-Covariance-GP again stays very competitive and improves as p increases. This time, RNN also improves as p increases and narrowly comes out on top. Interestingly, the Deep Kernel GP sees a dip in performance at $p = 50$ but quickly returns to form, still being outperformed by the Deep-Covariance-GP.

Lastly, we refer to Figure 4.3, where we again see that the traditional GP models are not able to predict disease with any certainty, leaving it to a probability of 0.5 in all cases. This time, the CNN performs quite well, especially as p increases. The deep GP is also able to make significant improvements in accuracy as p increases. However, it is still beat by the deep kernel GP on estimated covariance. The RNN again comes out on top, and is able to improve as p increases. This trend continues, even for Figure 4.4, except for

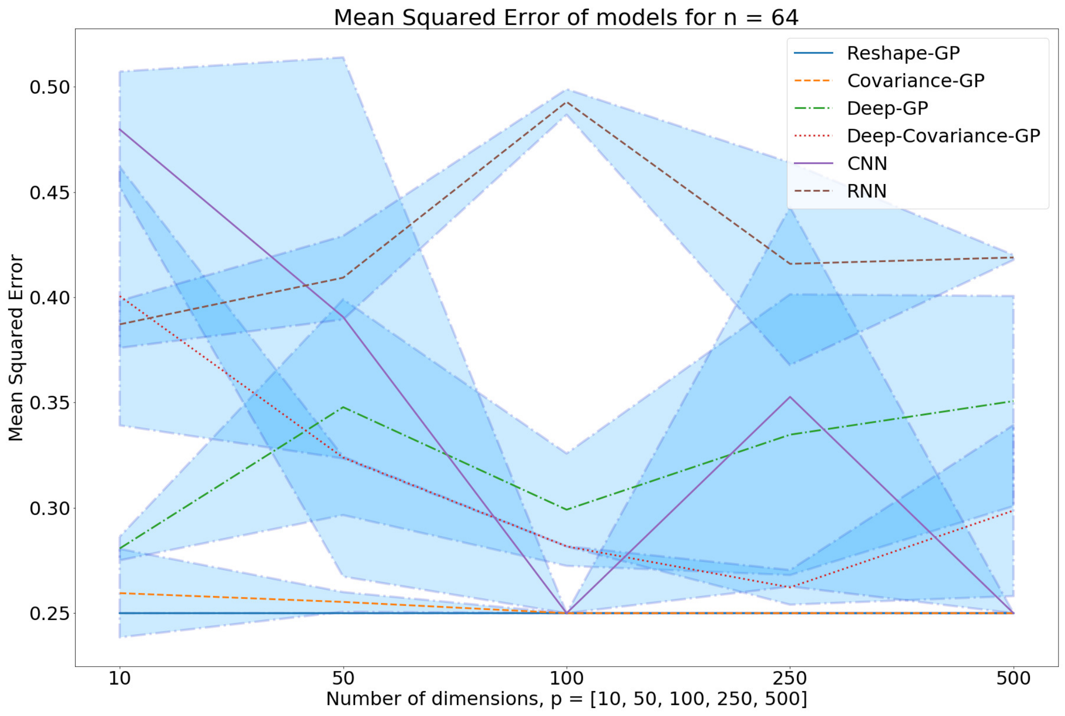

Figure 4.4: Mean Squared Error in the null case for Gaussian process, hybrid and deep models with $n = 64$ and $p = 10, 50, 100, 250, 500$

RNN which begins to perform very poorly.

4.2.1.2 Different covariance matrices

Referring to Figure 4.5 we see that both traditional GP models have a difficult time, and examining the model we see that the model consistently predicts disease with a probability of 0.5. The CNN performs very poorly, as to be expected because the strong inductive bias of convolution is not well suited to the time domain of this dataset. We see that an increase in p for fixed $n = 8$ results in a decrease in performance for the RNN, which is expected since traditional deep models perform best when $n >> p$. Finally, we see that the Deep kernel GP performs better as p increases. While the Deep Kernel GP on estimated covariances begins quite strongly and stays consistent as p increases.

Now referring to Figure 4.6 we see a similar trend that traditional GP models consistently predict disease with a probability of 0.5 even after 1000 iterations. The CNN continues to perform poorly but this time improves as p increases, still staying behind the traditional GP models. The deep kernel Gaussian process on estimated covariances again stays very competitive and improves as p increases. This time, RNN also improves as p increases and narrowly comes out on top when $p = 500$. Interestingly, the Deep Kernel GP sees a dip in performance at $p = 50$ but quickly returns to form, still being outperformed by the deep kernel GP on estimated covariances.

Lastly, we refer to Figure 4.7 we again see that GP models stay at the null MSE, predicting disease with a probability of 0.5 in all cases. The CNN is again the worst performer, but is able to see gains as p increases. The deep kernel GP is also able to make significant improvements in accuracy as p increases. However, it is still beat by the deep kernel GP on estimated covariance. The RNN again comes out on top, and is able to improve as p increases. This continues, even for Figure 4.8.

We note that this experimental set-up heavily favors the covariance estimation used in covariance-GP and deep-covariance-GP model. Simply put, the only difference in the data is the covariance structure which the estimates

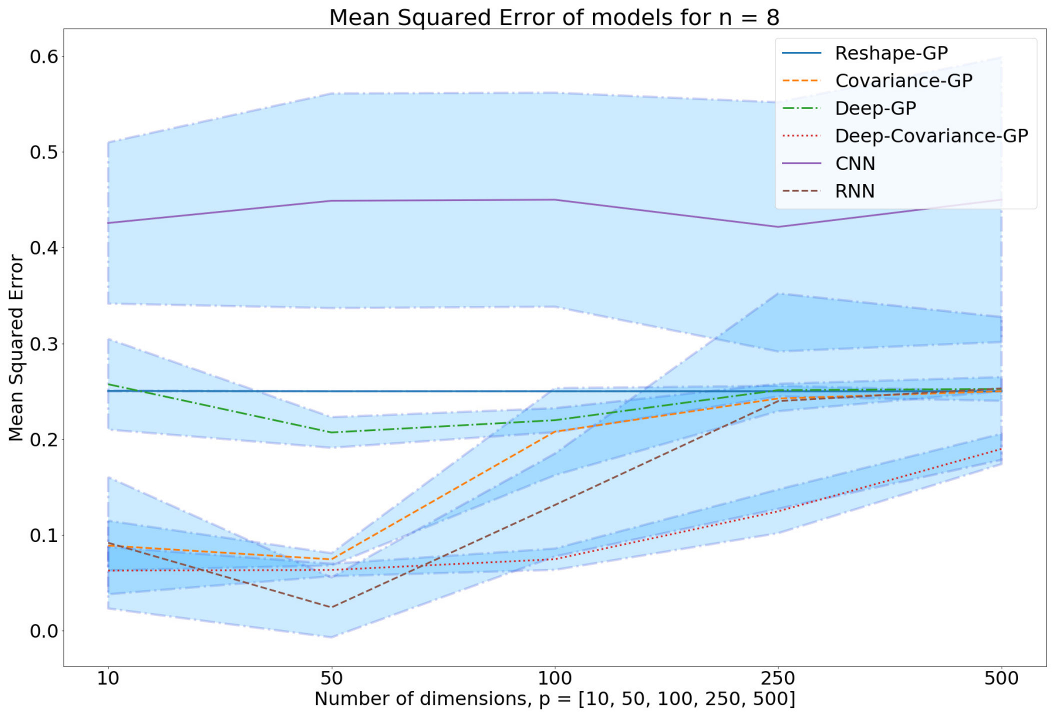

Figure 4.5: Mean Squared Error in the case of differing covariance matrices for Gaussian process, hybrid and deep models with $n = 8$ and $p = 10, 50, 100, 250, 500$

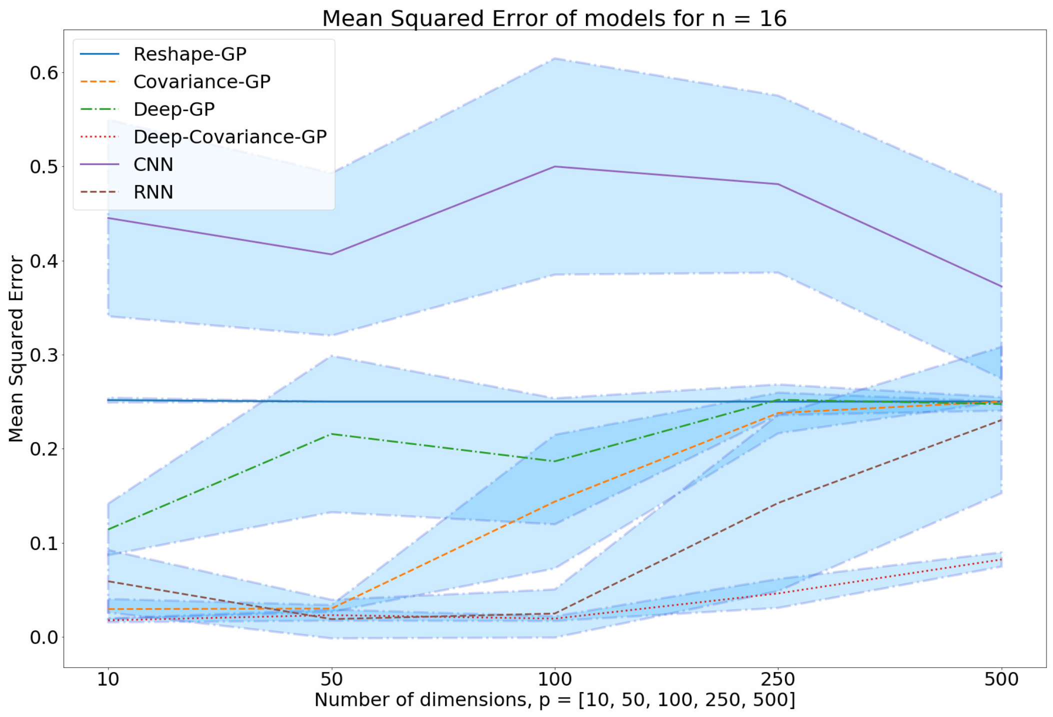

Figure 4.6: Mean Squared Error in the case of differing covariance matrices for Gaussian process, hybrid and deep models with $n = 16$ and $p = 10, 50, 100, 250, 500$

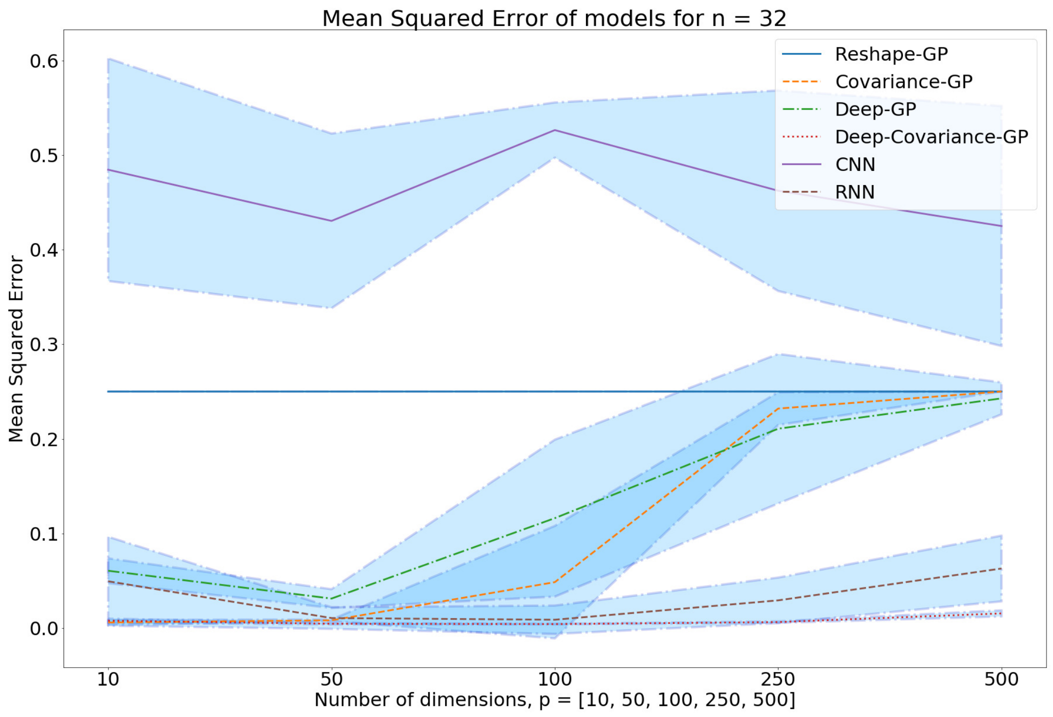

Figure 4.7: Mean Squared Error in the case of differing covariance matrices for Gaussian process, hybrid and deep models with $n = 32$ and $p = 10, 50, 100, 250, 500$

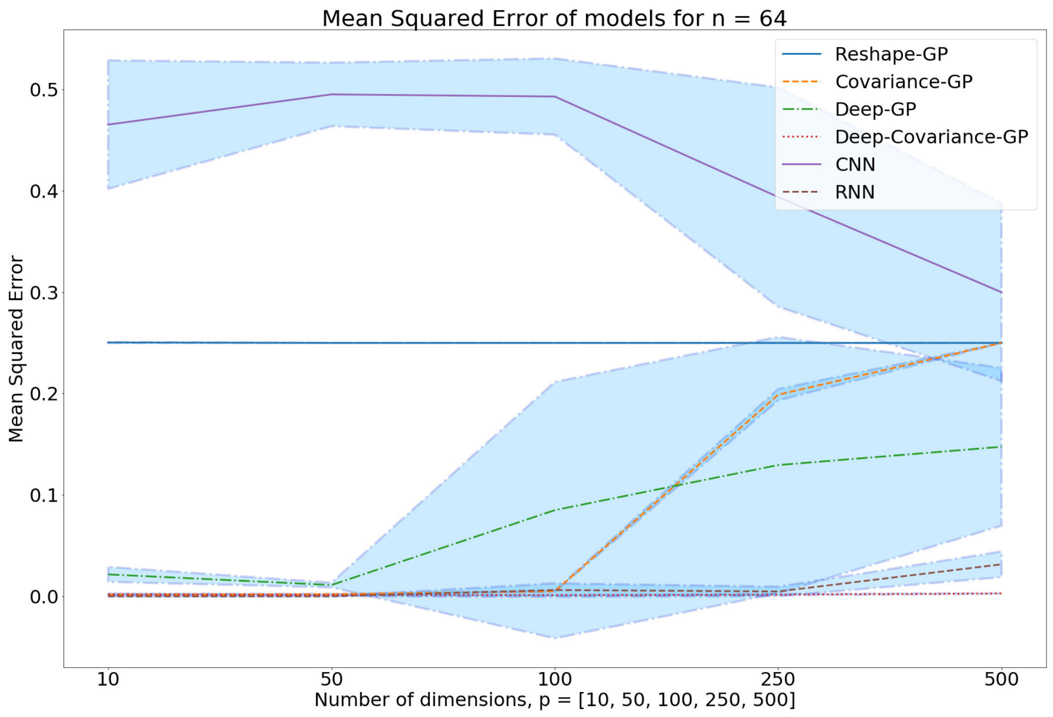

Figure 4.8: Mean Squared Error in the case of differing covariance matrices for Gaussian process, hybrid and deep models with $n = 32$ and $p = 10, 50, 100, 250, 500$

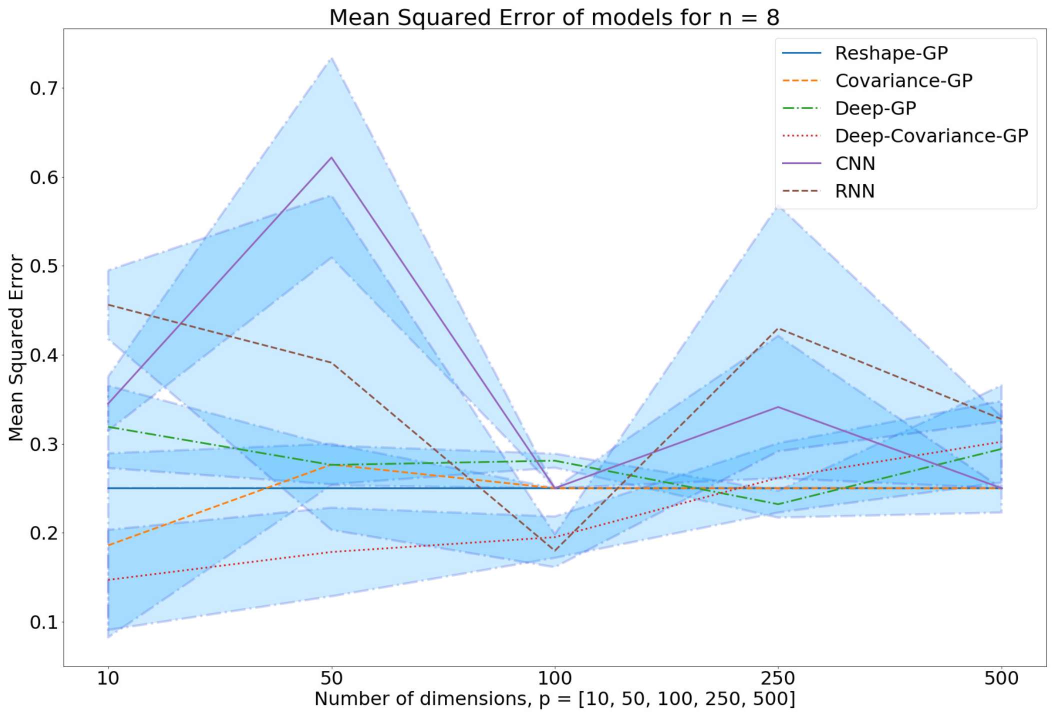

Figure 4.9: Mean Squared Error in the case of differing sparsity levels for Gaussian process, hybrid and deep models with $n = 8$ and $p = 10, 50, 100, 250, 500$

will reflect. Hence, covariance-GP and deep-covariance-GP models will have an easy time separating the space even with basic kernels.

4.2.1.3 Same covariance, but different sparsity levels

Referring to Figure 4.9 we see that all models tend towards the null result for larger dimensions. The most consistent performer in this case is the deep covariance GP, but its performance is worse than the null result as p increases.

Referring to Figures 4.10 and 4.11, we see that the null prediction beats most of the other models. As n increases, the beginning performance improves but it gradually performs worse until being worse than the null prediction.

This trend is finally overturned in Figure 4.12 where the deep covariance GP is able to achieve respectable results for all values of p tested.

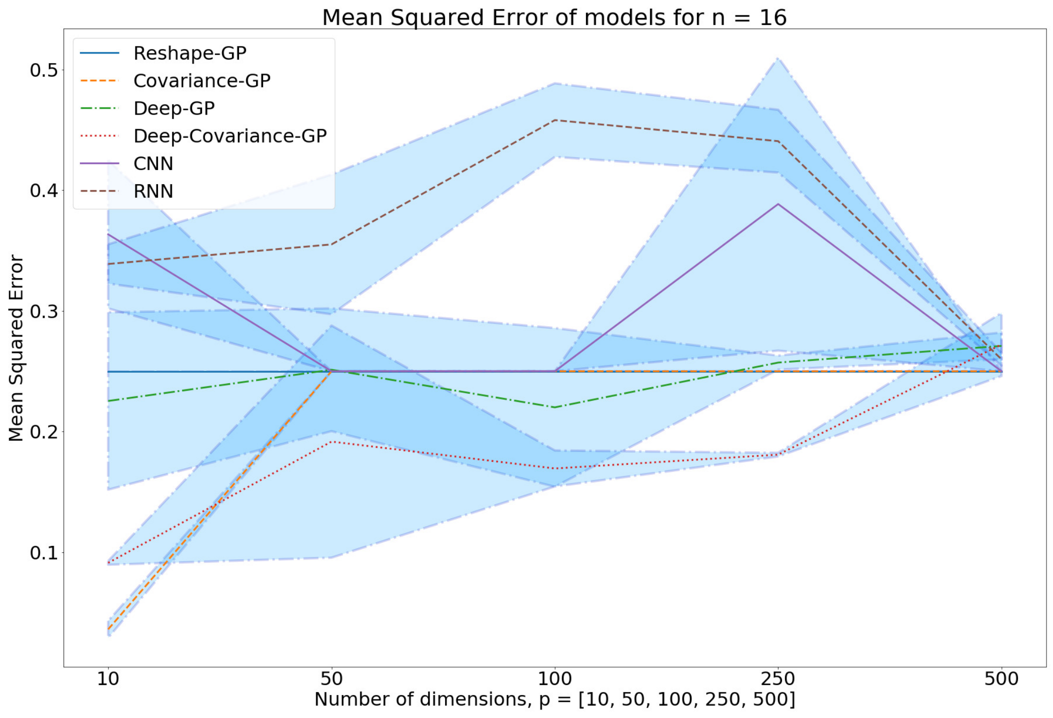

Figure 4.10: Mean Squared Error in the case of differing sparsity levels for Gaussian process, hybrid and deep models with $n = 16$ and $p = 10, 50, 100, 250, 500$

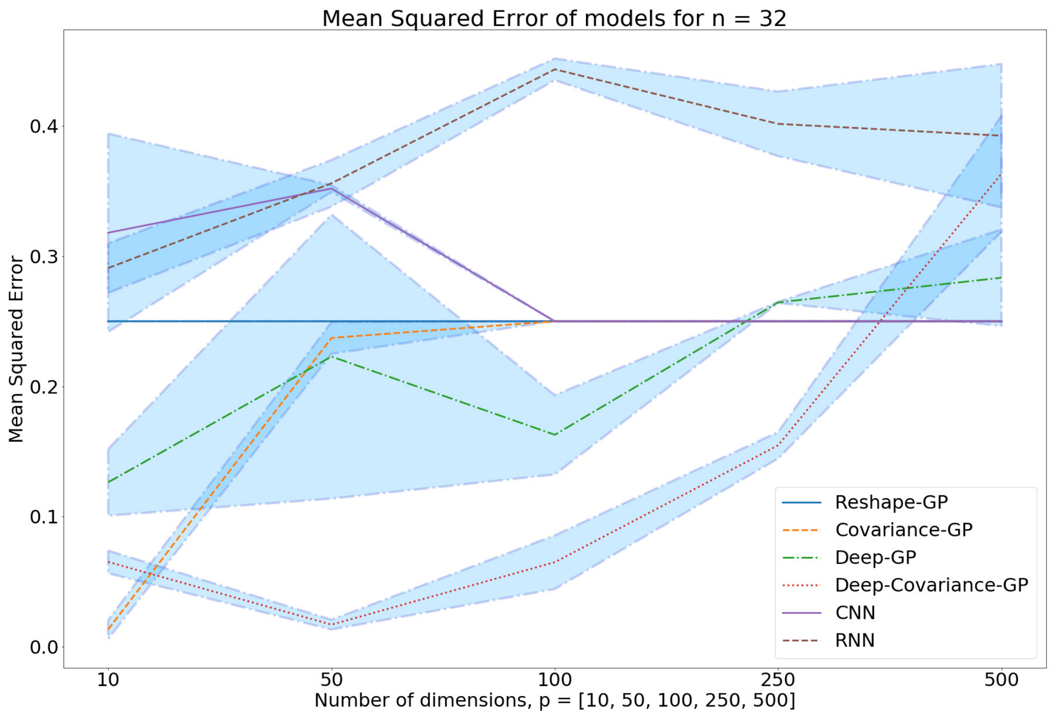

Figure 4.11: Mean Squared Error in the case of differing sparsity levels for Gaussian process, hybrid and deep models with $n = 32$ and $p = 10, 50, 100, 250, 500$

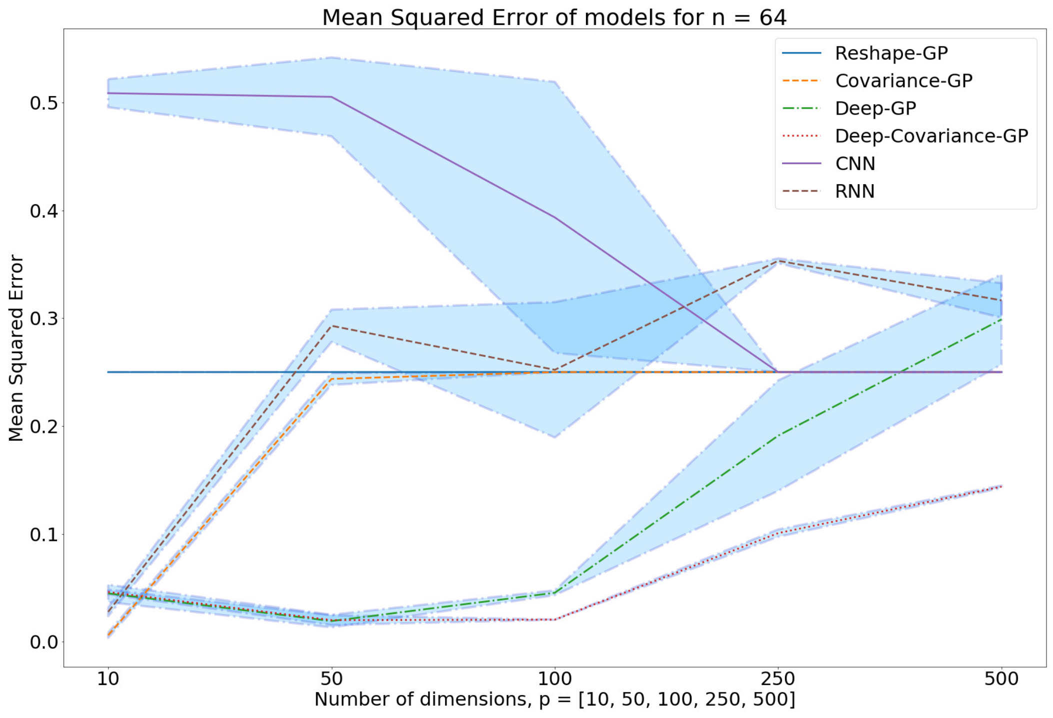

Figure 4.12: Mean Squared Error in the case of differing sparsity levels for Gaussian process, hybrid and deep models with $n = 64$ and $p = 10, 50, 100, 250, 500$

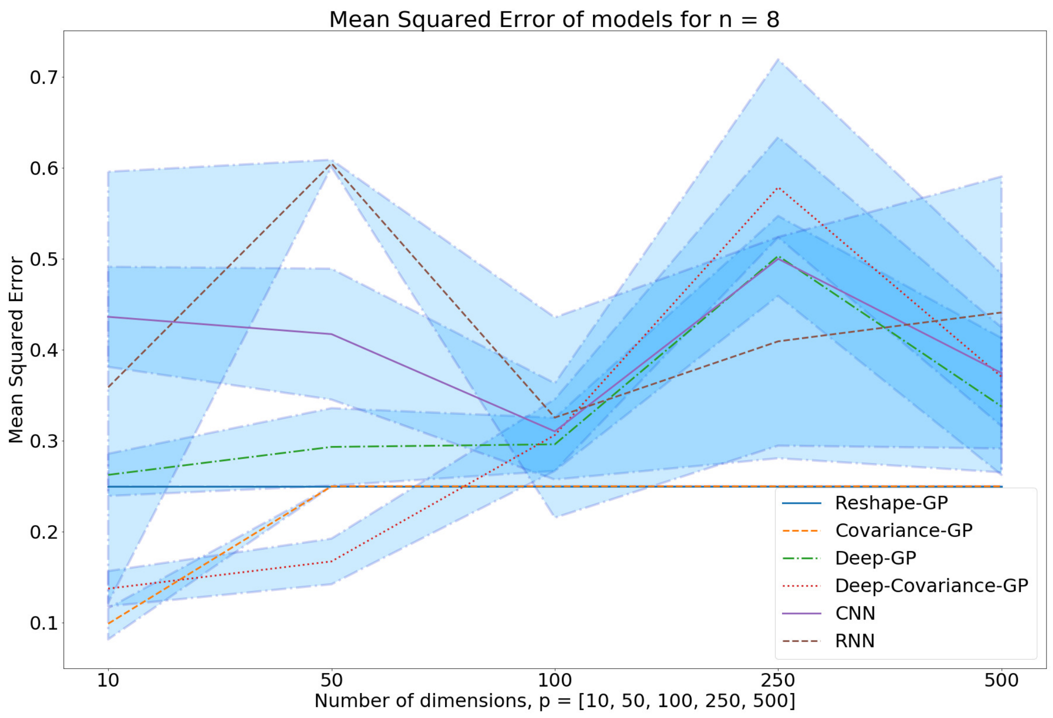

Figure 4.13: Mean Squared Error in the case of differing AR coefficients for Gaussian process, hybrid and deep models with $n = 8$ and $p = 10, 50, 100, 250, 500$

This experiment is most relevant to the neuroimaging literature and deserves a deeper investigation. We plan on expanding this further in the future.

4.2.1.4 Different AR coefficients

Referring to Figure 4.13 we see that both traditional GP models are unable to properly classify either controls or diseased patients. However, the other models actually perform much worse, and worsen as p increases.

Referring to Figures 4.14 to 4.16, we see that for small dimension sizes, the performance of most models is below the null baseline. However, their performance greatly worsens as p increases. This may be the case of the optimization requiring many more iterations for larger values of p, or the fact that the model cannot capture the temporal dependence of the AR coefficient.

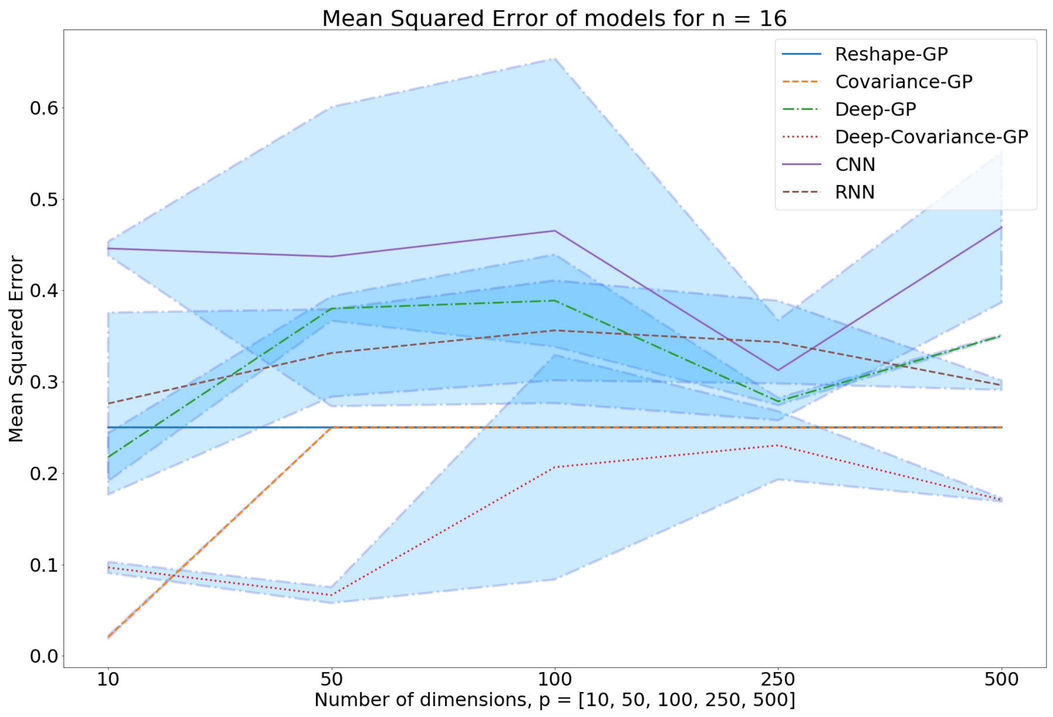

Figure 4.14: Mean Squared Error in the case of differing AR coefficients for Gaussian process, hybrid and deep models with $n = 16$ and $p = 10, 50, 100, 250, 500$

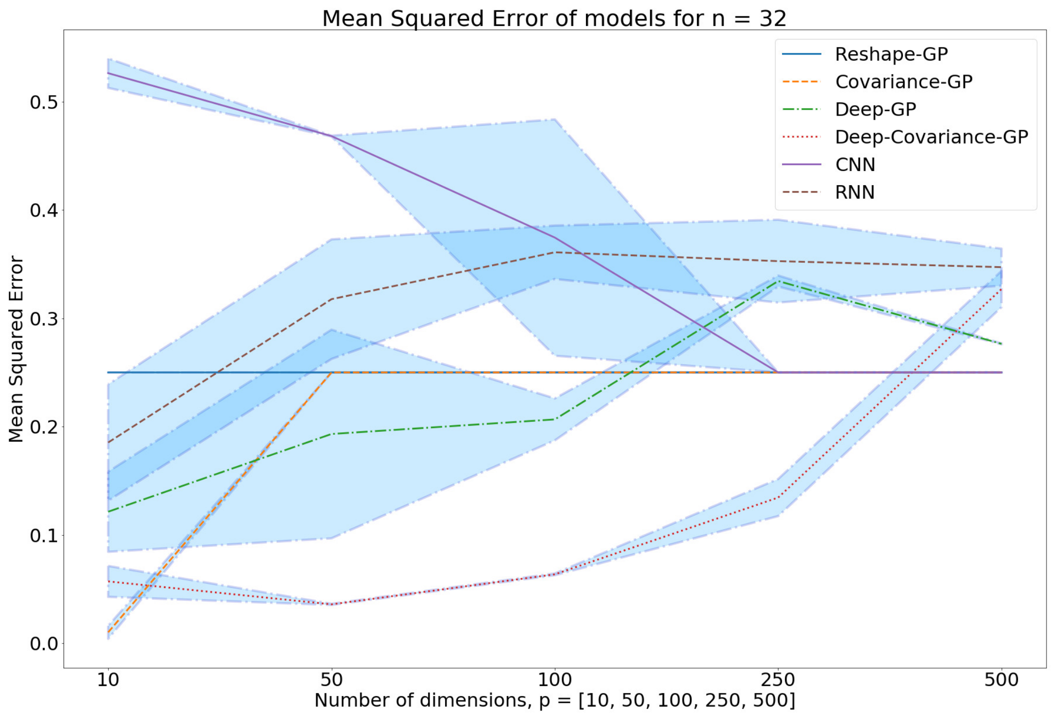

Figure 4.15: Mean Squared Error in the case of differing AR coefficients for Gaussian process, hybrid and deep models with $n = 32$ and $p = 10, 50, 100, 250, 500$

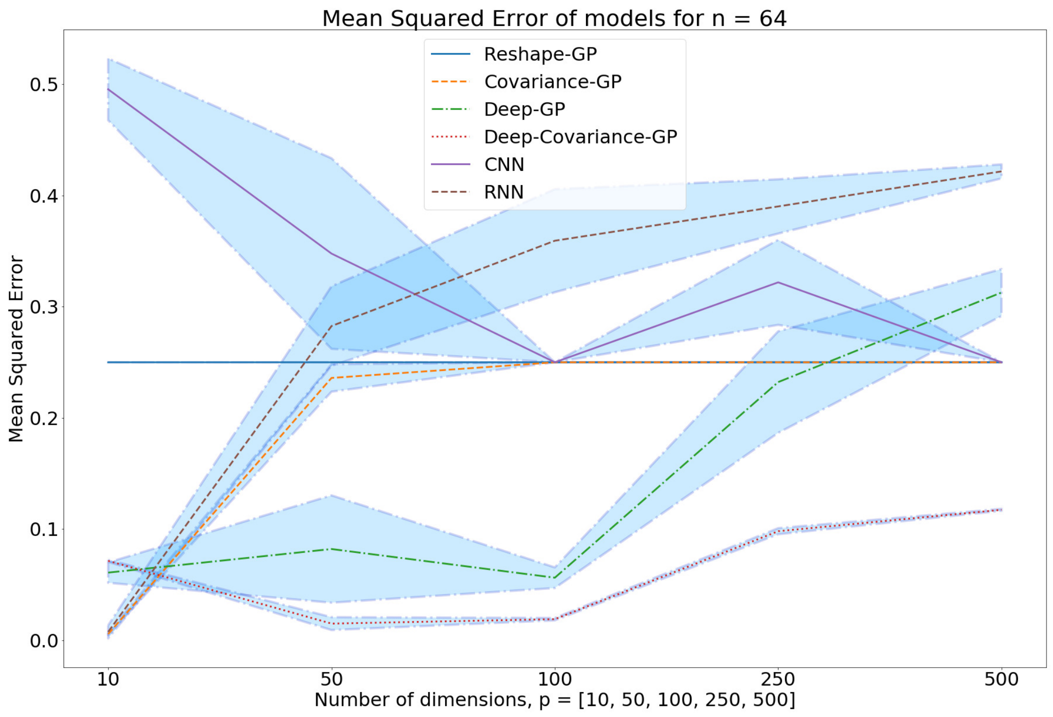

Figure 4.16: Mean Squared Error in the case of differing AR coefficients for Gaussian process, hybrid and deep models with $n = 64$ and $p = 10, 50, 100, 250, 500$

4.2.1.5 Randomly occurring Bernoulli spikes

Now, referring to Figure 4.17 we see that the trend continues with traditional GP models having a difficult time, with the reshape GP model not being able to predict anything. As p increases, we see that the covariance GP also fails to predict anything. Interestingly, the CNN begins to perform quite poorly but improves as p increases. This is in contrast to RNN which begins to perform quite well but worsens as p increases. Finally, we see that both Deep kernel GP performs better as p increases. While the Deep-Covariance-GP performs better overall, both models improve as p increases.

Referring to Figures 4.18 to 4.20, we see that as n increases, the covariance GP performs better but worsens as p increases. In addition, the beginning performance of the CNN worsens, but it still manages to improve as p increases. The RNN on the other hand, improves greatly as n increases but is still beat out by the deep kernel models, with both deep covariance GP and deep GP edging each other out.

4.2.2 Machine learning benchmark

Figure 4.21 shows that DKL-BN is the best performer when $n = 10$ and again when $n = 50000$. In Figure 4.21, we see that the size of the mini-batch improves performance, partially due to the increased capacity of the inducing variables as well as the reduction in covariate shift by batch normalization. However, diminishing returns begin to occur when the mini-batch size is large relative to the sample size.

Lastly, we look at a measure of uncertainty, which we define as the ambiguity of f, $\max_{\mathbf{x}} f(\mathbf{x})(1 - (f(\mathbf{x})))$. Then, a model's most ambiguous point is defined as $\operatorname{argmax}_{\mathbf{x}} f(\mathbf{x})(1 - (f(\mathbf{x})))$. In Figure 4.23 we take samples of most ambiguous points. We see that DKL-BN has difficulty telling whether Figure 4.23 (left) is a 5 or a 6, and hence have difficulty determining if it is odd or even. However, its prediction is 0.656 with a variance of 0.225, which is a low confidence prediction of the correct classification: odd. Similarly for Figure 4.23 (right), DKL cannot tell whether the digit is a 1 or a 2. Again, DKL provides a prediction is 0.494 with a variance of 0.499, which is a very

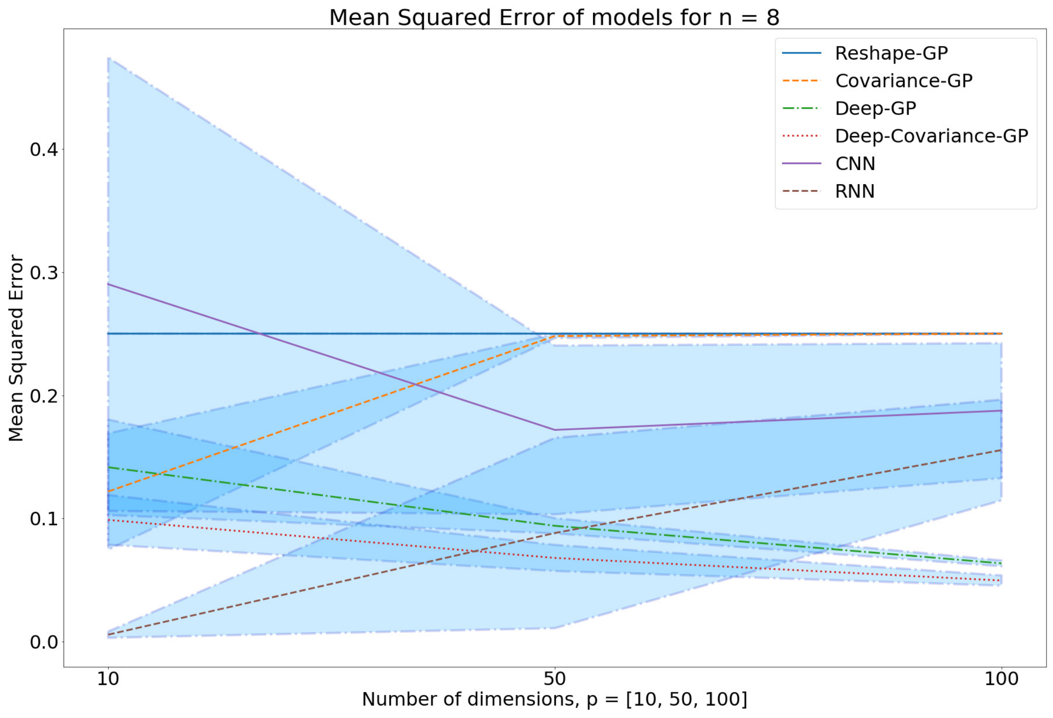

Figure 4.17: Mean Squared Error in the case of simulated spikes for Gaussian process, hybrid and deep models with $n = 8$ and $p = 10, 50, 100$

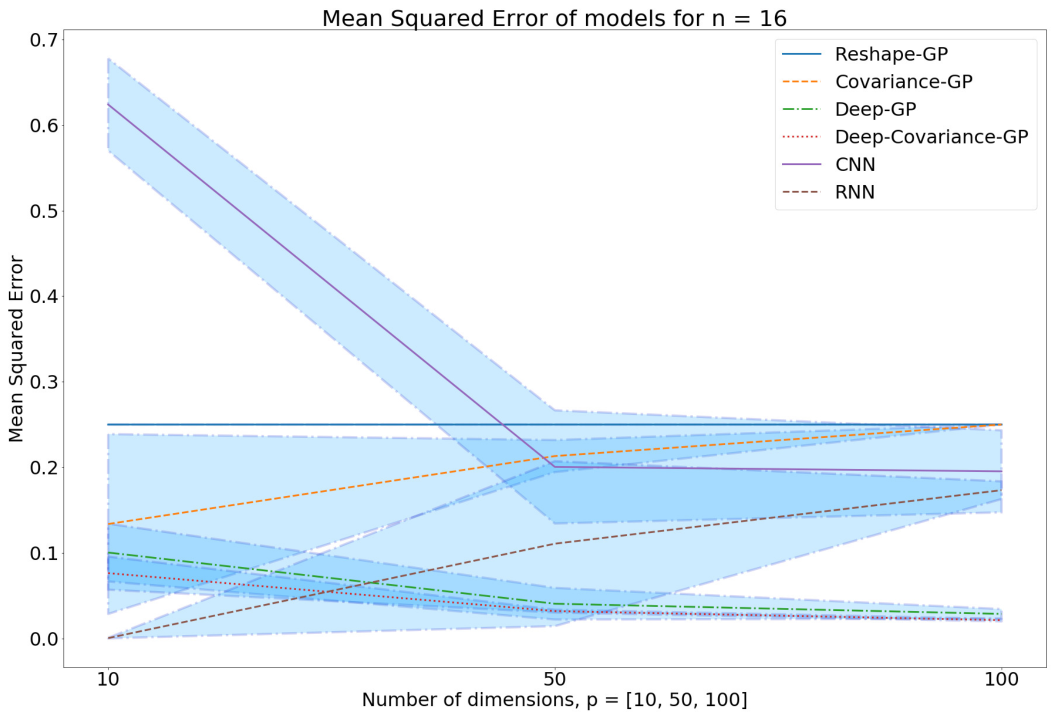

Figure 4.18: Mean Squared Error in the case of simulated spikes for Gaussian process, hybrid and deep models with $n = 16$ and $p = 10, 50, 100$

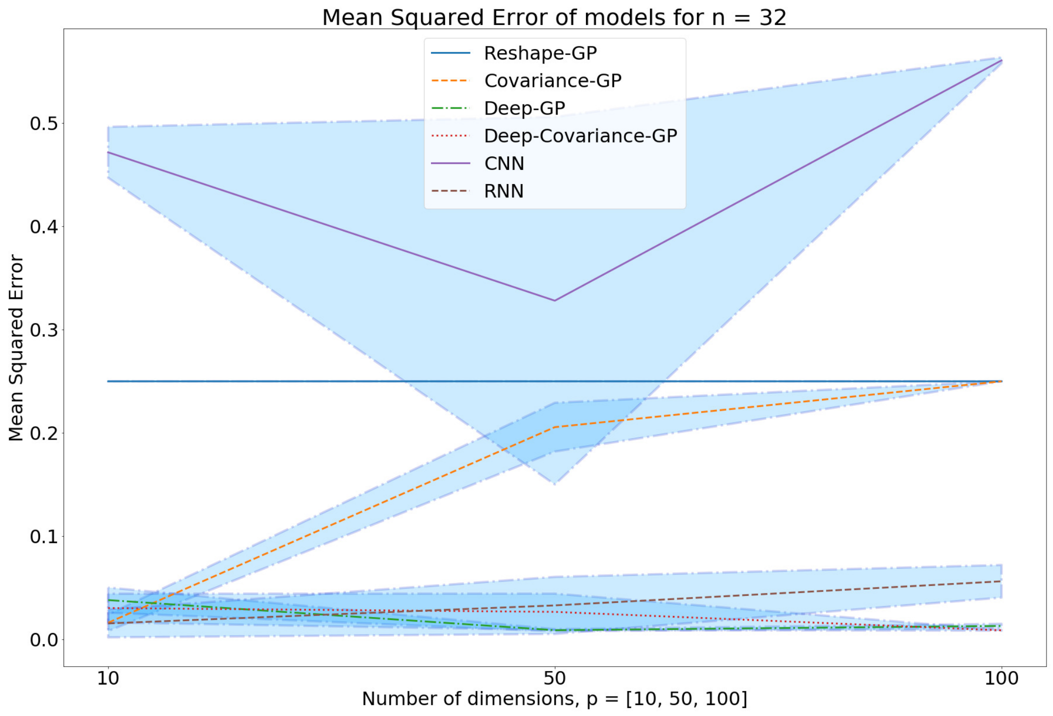

Figure 4.19: Mean Squared Error in the case of simulated spikes for Gaussian process, hybrid and deep models with $n = 32$ and $p = 10, 50, 100$

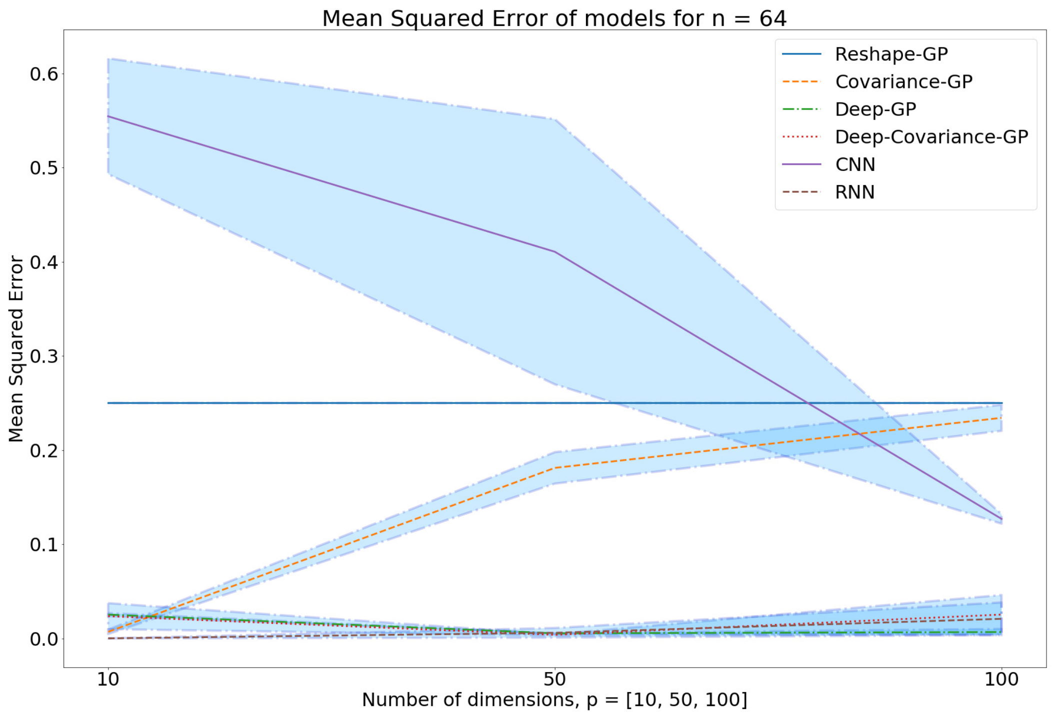

Figure 4.20: Mean Squared Error in the case of simulated spikes for Gaussian process, hybrid and deep models with $n = 64$ and $p = 10, 50, 100$

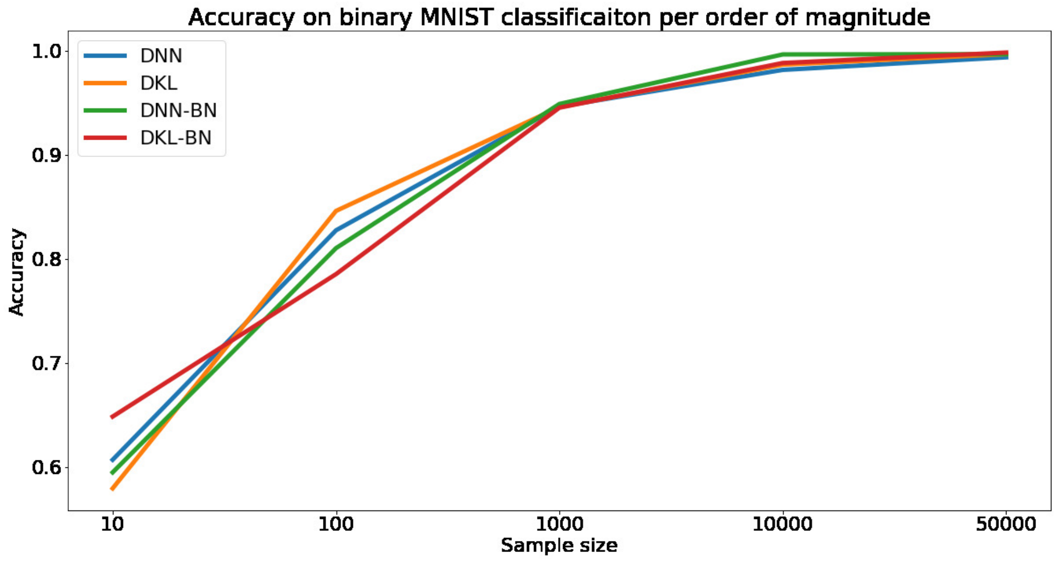

Figure 4.21: Accuracy for the Deep Kernel Learning (DKL) and Deep Neural Network (DNN) models, with and without Batch Normalization (BN), as sample size increases

low confidence prediction of the incorrect classification: even. In Table 4.1 we see that DKL-BN is able to improve greatly over DKL case. That is, if we take the most ambiguous point for one model and use the other model to evaluate it, we are able to compare the two models. We see that DKL-BN's most ambiguous point, evaluated with DKL, provides a higher confidence estimate of the correct result. On the other hand, DKL-BN evaluated at DKL's most ambiguous point is able to provide the correct classification at a much higher confidence. On average, DKL is only able to provide marginal improvement over DKL-BN. However, DKL-BN is able to substantially improve predictive accuracy and uncertainty estimates over DKL. That is, the on–average worst-case performance is better with DKL-BN than it is on DKL.

4.2.3 fMRI experiments

4.2.3.1 Null dataset

We report the training testing loss, training and test errors, training and testing accuracies as well as ROC curves below. Unsurprisingly, we find that in

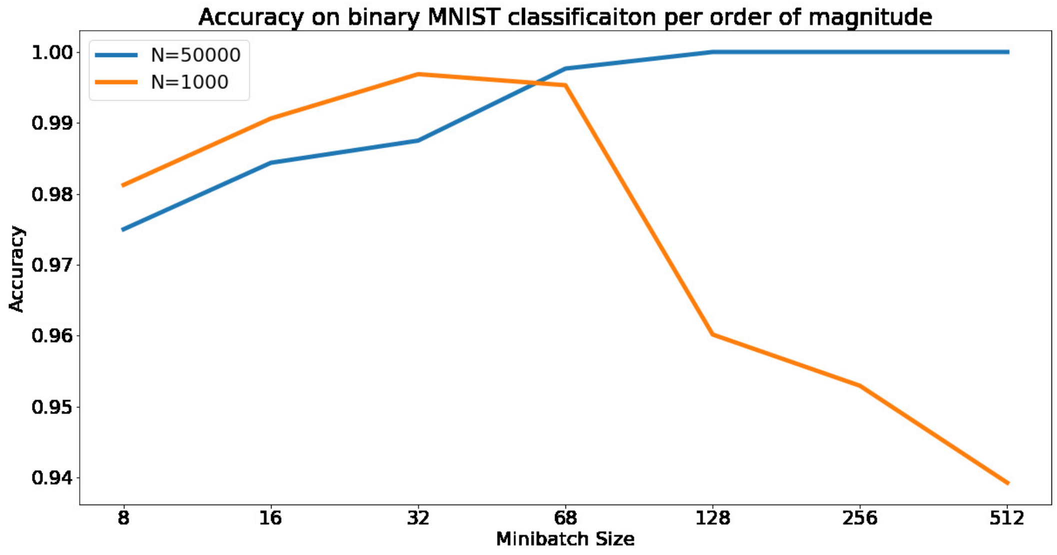

Figure 4.22: Accuracy for DKL-BN for n = 1000 and n = 50000 as mini-batch size increase

	DKL-BN		DKL	
	Mean	Var	Mean	Var
Fig 2 (left)	0.656	0.225	0.732	0.195
Fig 2 (right)	0.917	0.075	0.494	0.499

Table 4.1: Mean prediction and variance estimated by both models under the points with maximal ambiguity. Note: a mean and variance of 1.0 and 0 respectively indicates that the MNIST digit is certainly odd

Figures 4.27 and 4.28the model is able to attain a low training error relatively quickly. Furthermore, Figures 4.24 and 4.25 shows that our model has indeed converged. The low error rate on the training set is due to the model capacity [83] of the recurrent neural network that learns the representation of the data for the Gaussian process classifier. We see another story with the testing error over time, which does not improve over the optimization process. Interestingly, the log-likelihood of the testing data worsens initially but then reaches a steady state. This may have been a result of a poor local minima that was escaped by the stochastic gradient step. Lastly, we note that in Figures 4.29 and 4.30 the accuracy fluctuates similarly to the error rate described earlier, with little improvement at the end.

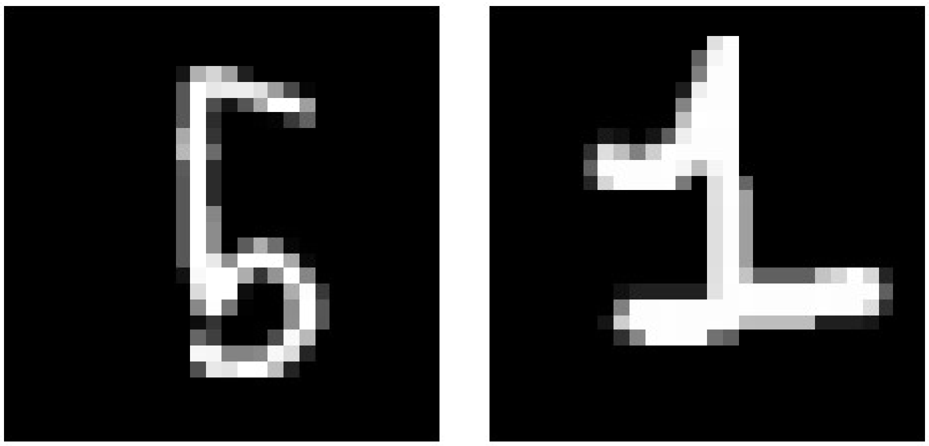

Figure 4.23: Samples of MNIST digits that maximize ambiguity for DKL-BN (left) and DKL (right)

For the recurrent neural network model, the story is similar. Namely, the optimization is stable and reaches a local minimum as evidenced by the smooth plateau in Figure 4.26. However, the smoothness also translates to the training plots of MSE in Figures 4.27 and 4.28 and accuracy in Figures 4.29 and 4.30. That is, during the training process the recurrent neural network model does not recognize that its input is randomly permuted data. This is only evidenced by the testing plot in Figure 4.28 and Figure 4.30.

In short, we find that our model does not find non-existent differences in randomly labelled data. Moreover, the optimization procedure does reach a stable local minima. We now investigate the classification results on the trained model through the ROC. That is, given predictions from a model, how do the true and false positive rates relate to one another when the decision threshold is varied. As noted earlier, the accuracy was based on a classifier that used a threshold of 0.5. Here, we will investigate a range of thresholds for each cross-validation step (without averaging).

Referring to Figure 4.31, we see that any choice of threshold may perform well on one split of the data but perform worse on another. This demonstrates that the model is not able to find a good way to classify this randomly permuted data. This is especially good, since the model does not seem to favor positive or negative classifications, since there is an even split down the median

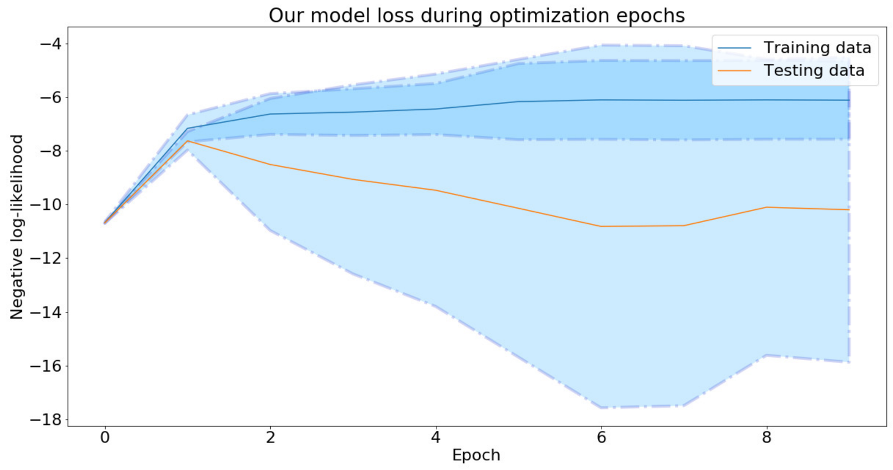

Figure 4.24: Negative loglikelihood for our model trained on the null dataset with 5 fold cross validation

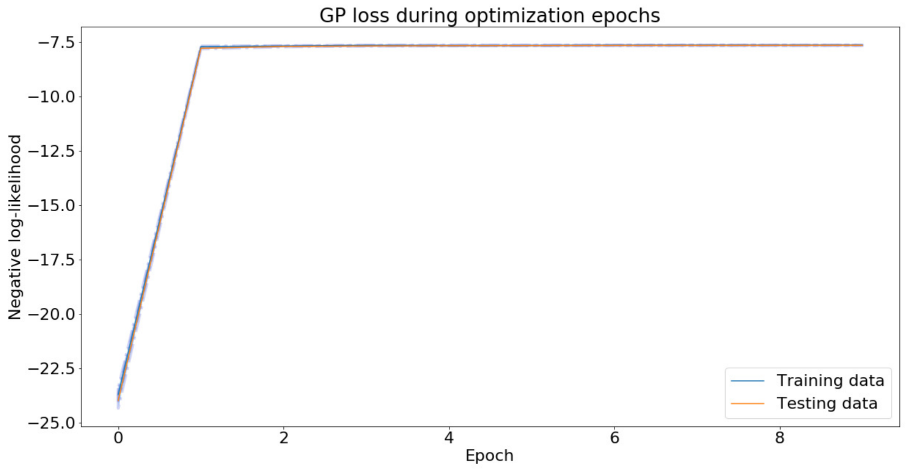

Figure 4.25: Negative loglikelihood for a Gaussian process trained and tested on the null dataset with 5 fold cross validation

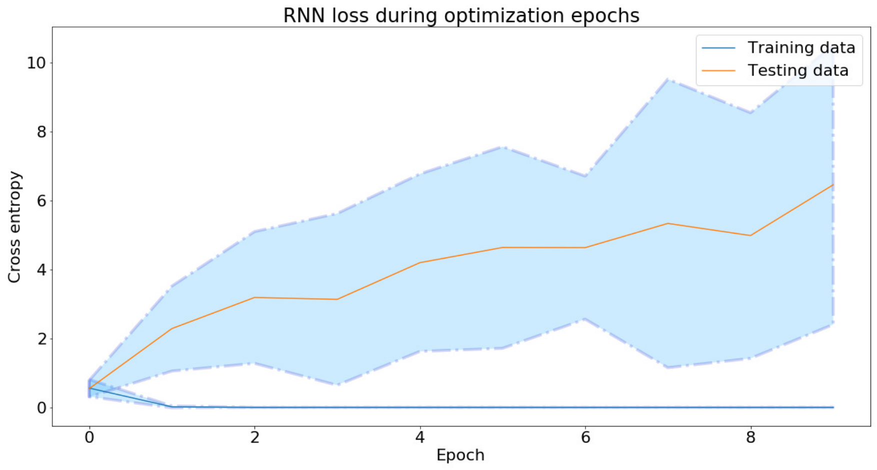

Figure 4.26: Cross entropy for a recurrent neural network trained and tested on the null dataset with 5 fold cross validation

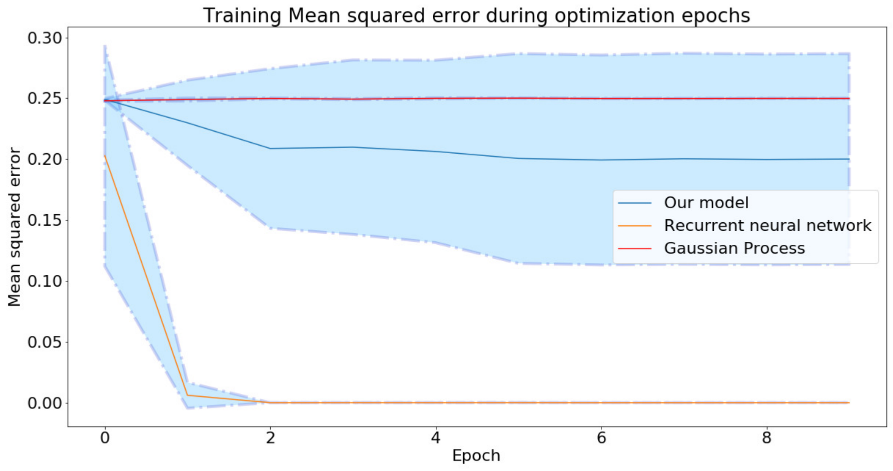

Figure 4.27: Training mean squared error for our model, a Gaussian process and a recurrent neural network trained on the null dataset

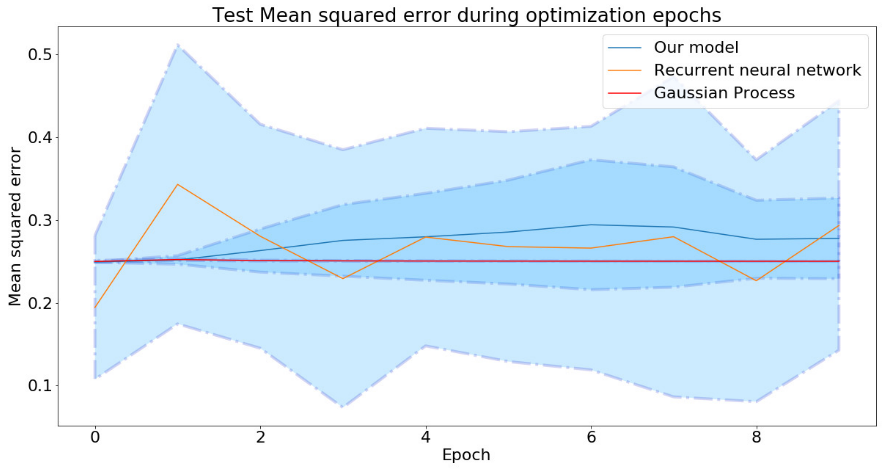

Figure 4.28: Testing mean squared error for our model, a Gaussian process and a recurrent neural network tested on the null dataset with 5 fold cross validation

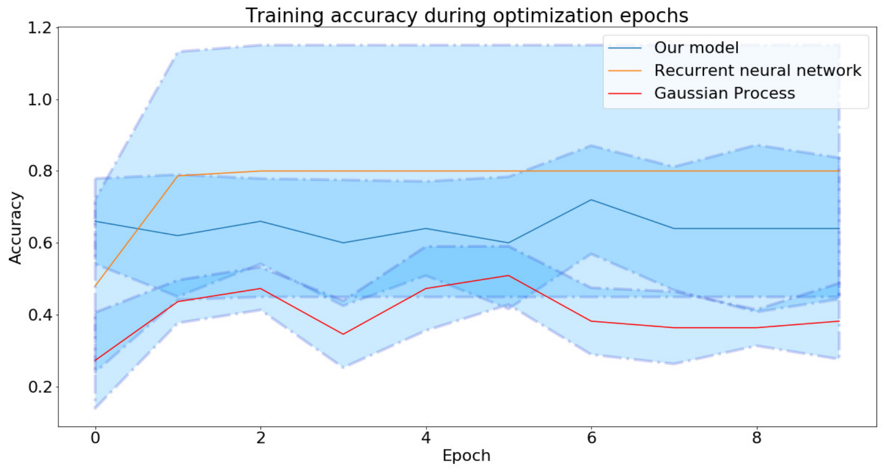

Figure 4.29: Training accuracy for our model, a Gaussian process and a recurrent neural network on the null dataset

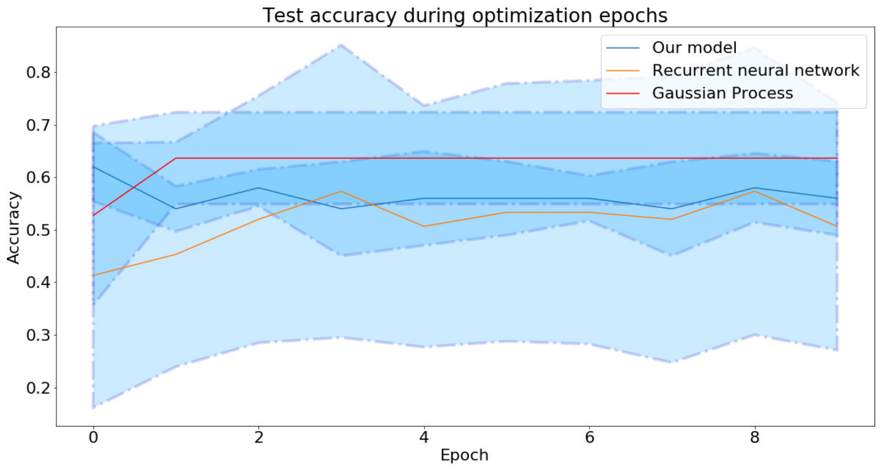

Figure 4.30: Testing accuracy for our model, a Gaussian process and a recurrent neural network on the null dataset, with 5 fold cross validation

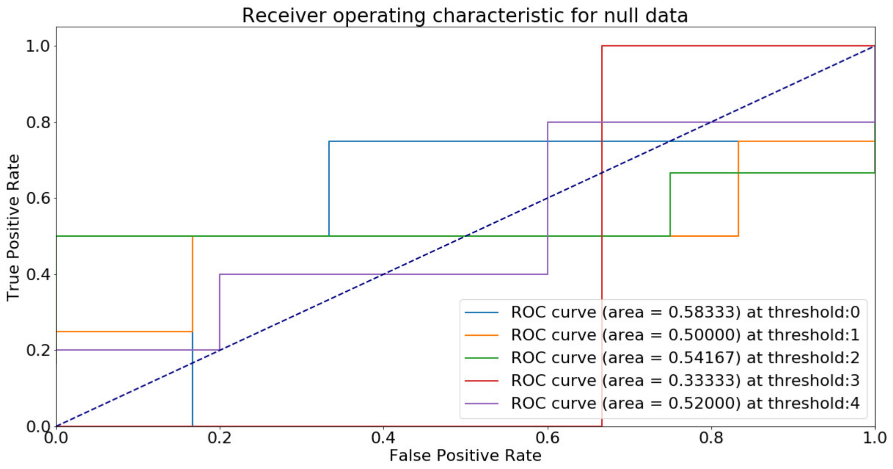

Figure 4.31: ROC curve for our model trained on the null dataset with 5 fold cross validation

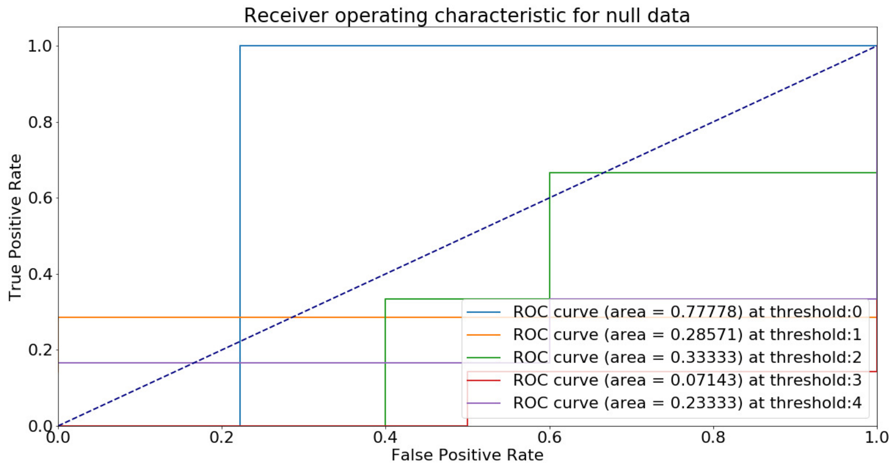

Figure 4.32: ROC curve for a Gaussian process trained on the null dataset with 5 fold cross validation

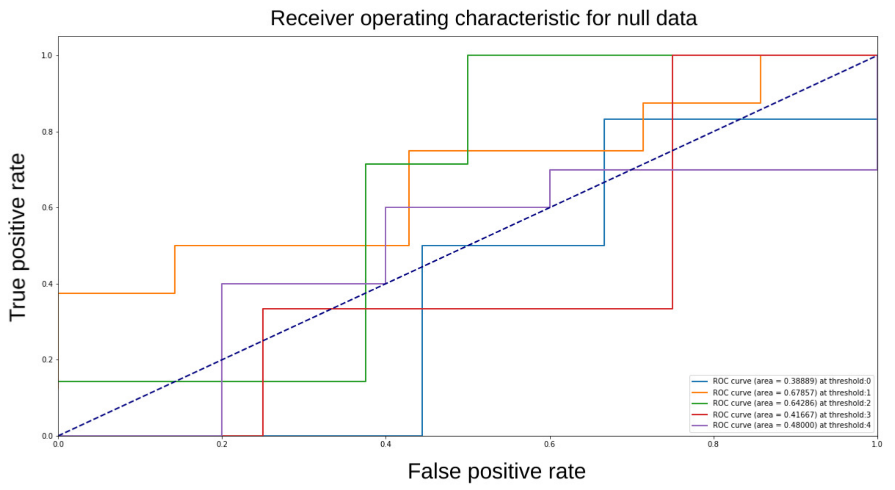

Figure 4.33: ROC curve for a recurrent neural network trained on the null dataset with 5 fold cross validation

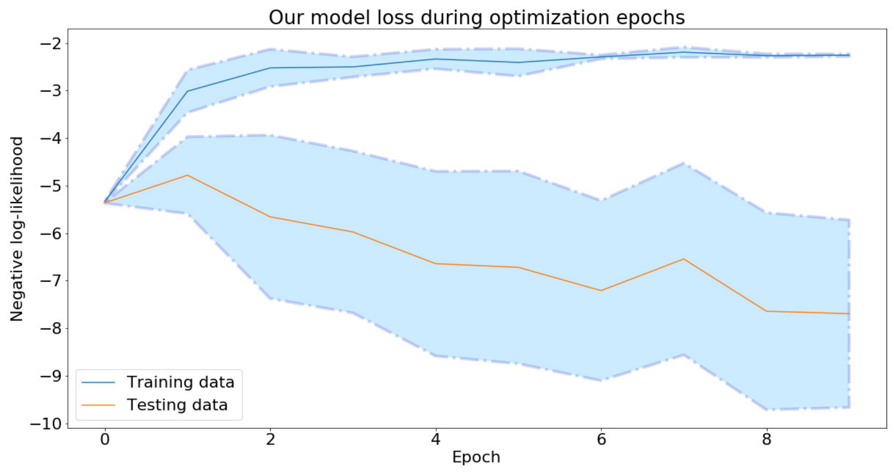

Figure 4.34: Negative loglikelihood for our model trained and tested on the dyslexia dataset with 5 fold cross validation

line. Hence our model is able to know what it doesn't know. Surprisingly, the same result holds true for the recurrent neural network model in Figure 4.33 and the Gaussian process model in Figure 4.32.

4.2.3.2 Dyslexia dataset

We again report the estimated likelihood, training and test errors, as well as the classification results for the {true, false}×{positives, negatives}. Unlike the Null data set, Figure 4.37 shows that the model is able to attain a low training error quickly. The test curve given Figures 4.34 to 4.36 is also stable, affirming that the model has indeed converged. We can see in Figures 4.37 and 4.38 that the test error is also relatively stable. The test accuracy of our model, as seen in Figures 4.39 and 4.40, continues to increase as the training accuracy also increase with each epoch.

Turning our attention to the recurrent models, we first notice how quickly the model converges. In Figure 4.34, Figure 4.37 and Figure 4.39 we see that there is a sharp increase in training performance at the very first epoch. However, this performance increase does not continue as training goes on indicating that the model has overfit the relatively small sample size. Indeed, the test

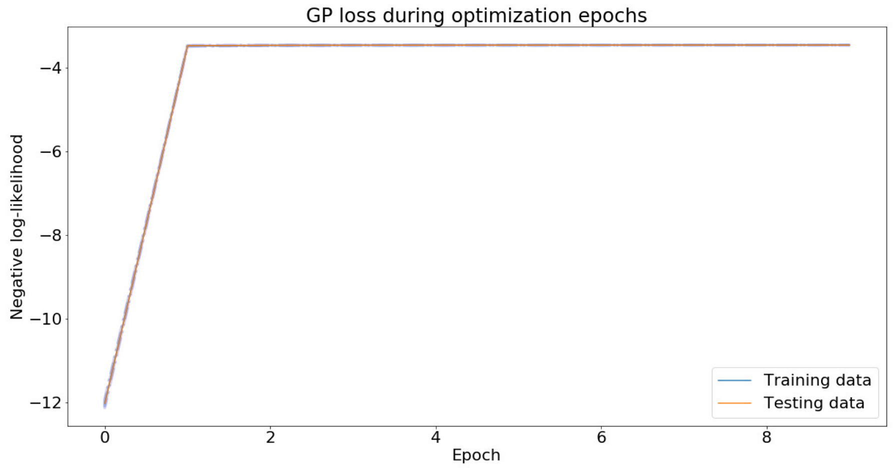

Figure 4.35: Negative loglikelihood for a Gaussian process trained and tested on the dyslexia dataset with 5 fold cross validation

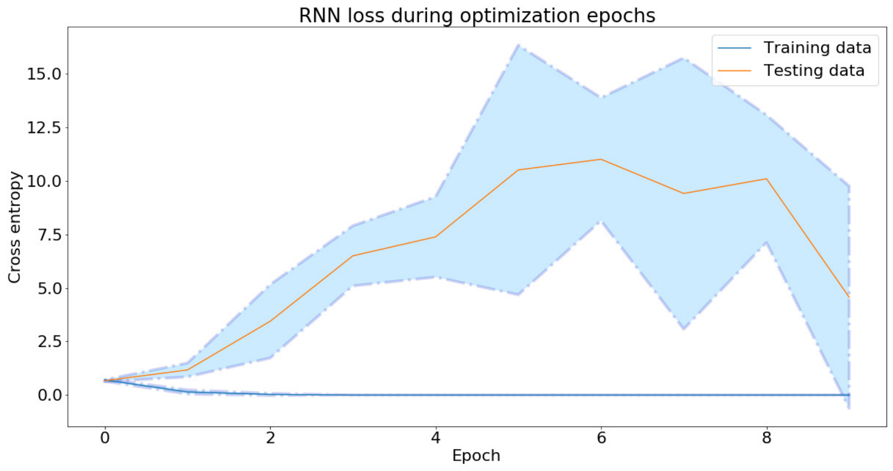

Figure 4.36: Cross entropy for a recurrent neural network trained and tested on the dyslexia dataset with 5 fold cross validation

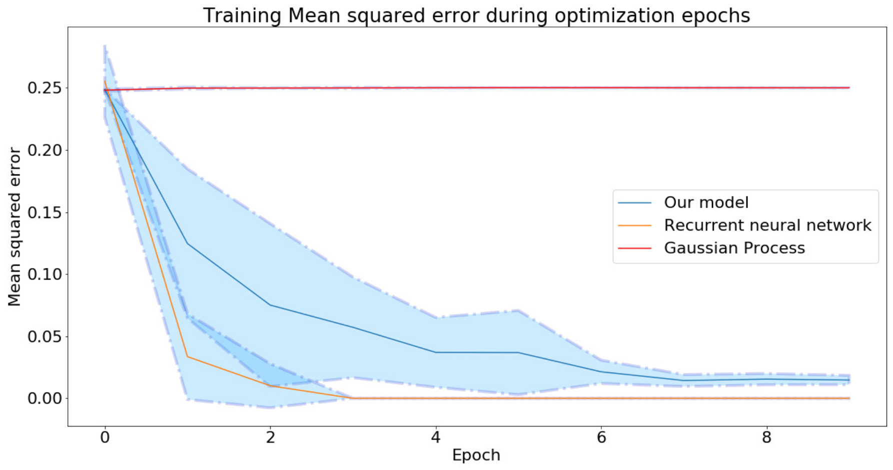

Figure 4.37: Training mean squared error for our model, a Gaussian process and a recurrent neural network trained on the dyslexia dataset with 5 fold cross validation

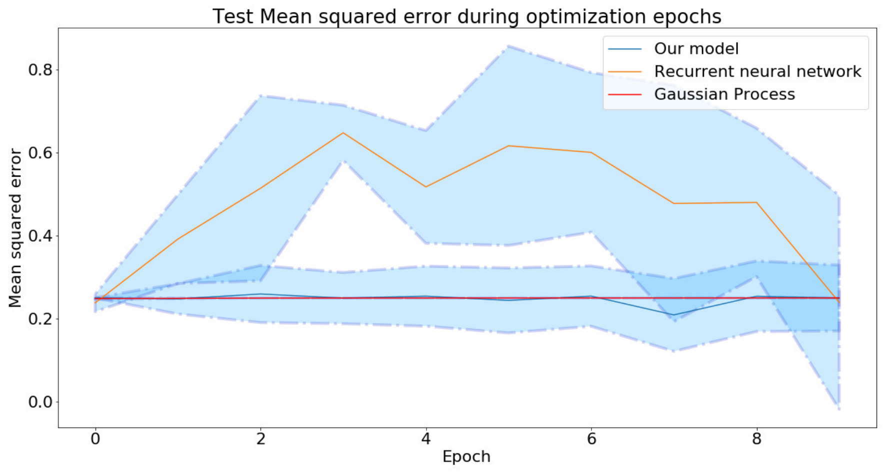

Figure 4.38: Testing mean squared error for our model, a Gaussian process and a recurrent neural network trained on the dyslexia dataset with 5 fold cross validation

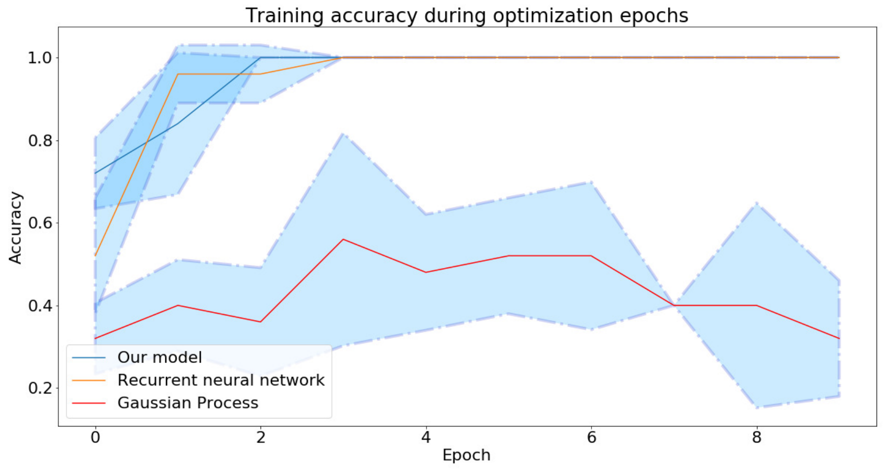

Figure 4.39: Training accuracy for our model, a Gaussian process and a recurrent neural network on the dyslexia dataset, with 5 fold cross validation

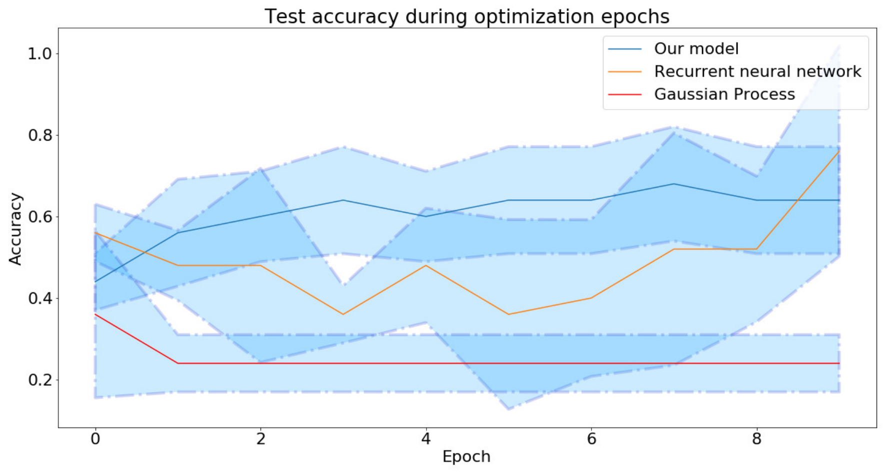

Figure 4.40: Testing accuracy for our model, a Gaussian process and a recurrent neural network on the dyslexia dataset, with 5 fold cross validation

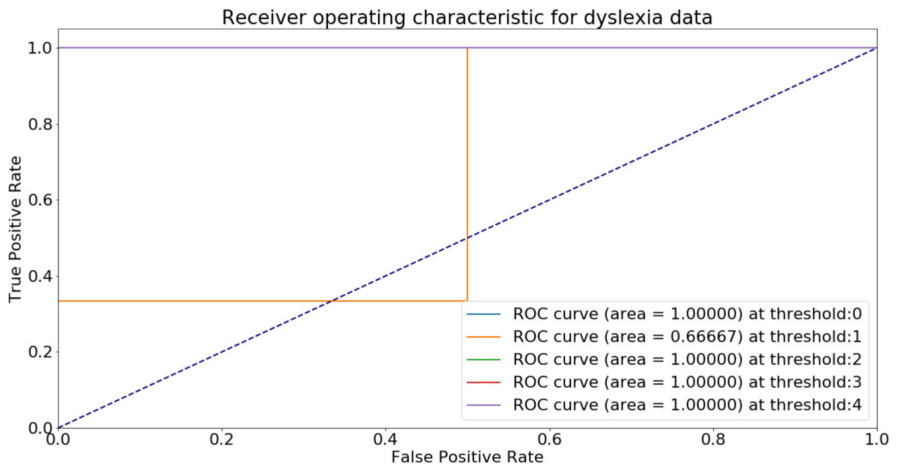

Figure 4.41: ROC curve for our model trained on the dyslexia dataset with 5 fold cross validation

performance in Figure 4.38 and Figure 4.40 wavers as the epochs continue unlike our model which smoothly increases over time.

Summarizing our results so far, we find that the recurrent model overfits very easily. This is despite the fact that both models are equally "deep" since they share the same neural network configuration. We investigate our results further by looking at the ROC curve for each cross validated fold, without any averaging.

Referring to Figure 4.41, we see that our model is able to classify the data nearly perfectly. That is, the data is pushed to relatively close 0 or 1 and correctly classified. Normally, this result would be suspect. However, since the test size is quite small its not out of the ordinary to achieve perfect classification on a specific fold. Indeed, we see that for one fold the model performs best at a specific threshold.

This result is in contrast to the recurrent model in Figure 4.43. As before, the model is not able to find a good way to classify this randomly permuted data. This is especially bad, since there is an even split down the median line. That is, the recurrent model overfits the data despite having the same initial configuration as our Gaussian process model.

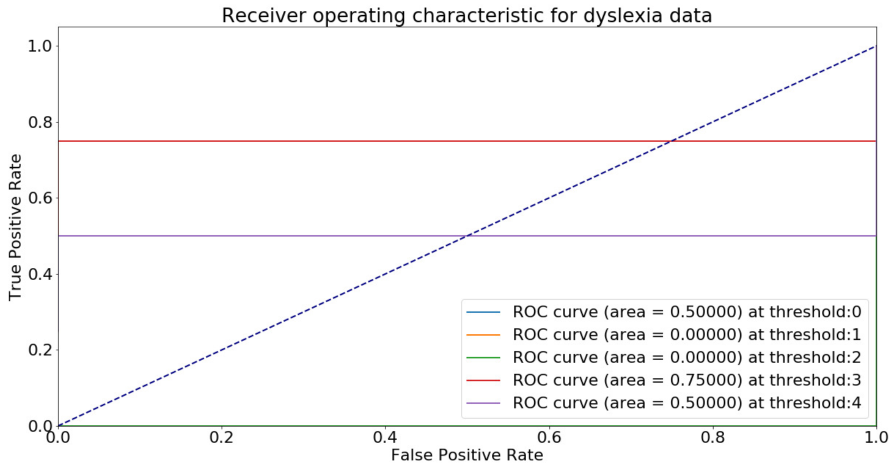

Figure 4.42: ROC curve for a Gaussian process trained on the dyslexia dataset with 5 fold cross validation

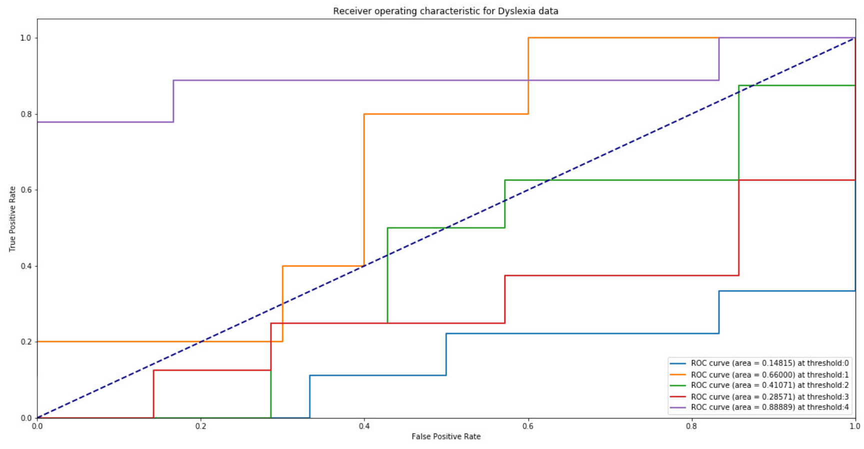

Figure 4.43: ROC curve for a recurrent neural network trained on the dyslexia dataset with 5 fold cross validation

Chapter 5

Discussion

5.1 Summary

In this thesis, we proposed two extensions to stochastic variational deep kernel learning for Gaussian processes. The primary motivation for this work was to learn an effective intermediate representation for Gaussian process classification. Our first extension broadened the family of neural networks under consideration, while our second extension used Bayesian methods to quantify the uncertainty of the intermediate representation. As a result, the models proposed in this thesis are well equipped to handle data sets with small n and large p, such as those in neuroimaging.

In our first contribution, we developed kernels parameterized by a recurrent neural network which are simultaneously trained with Gaussian processes using stochastic variational inference. We investigated not only the inference problem of Gaussian process classification with recurrent kernels, but also the computational bottleneck inherent to our model. Returning to the first question described in Chapter 1, "can we learn complex models from limited data?", we showed that the Bayesian regularization of Gaussian processes allows us to apply recurrent neural networks even when sample sizes are low.

Second, we investigated the use of batch normalization, instead of dropout, to provide uncertainty estimates in neural network parameterized kernels. This novel approach to uncertainty quantification implicitly approximates a distribution of neural networks. While batch normalization defines an implicit method of uncertainty quantification, it performed well empirically. Returning to the second question described in Chapter 2, "can we reason about the uncertainties of complex models?", we showed that a Bayesian interpration of batch normalization allows us to recover uncertainty estimates, even when other approaches like dropout fail.

Empirically, our simulation studies indicated that our recurrent model is able to maintain better classification accuracy than competing models as the dimension p increased while keeping n fixed. For real data, we found similar results on the MNIST dataset where the performance of our convolutional model was much better for small n but this gain diminished as n increased. Lastly, in the neuroimaging dataset we demonstrated that our recurrent model is robust to the null case and does not extrapolate from non-existent differences. Moreover, our recurrent model was able to greatly out perform a recurrent neural network of the same configuration when classifying dyslexia patients versus healthy controls.

In conclusion, we found that combining probabilistic models with neural networks leads to very strong empirical performance in domains where data is small relative to its dimension. Moreover, both Bayesian deep and deep Bayesian approaches should be applied and studied more generally, as uncertainty quantification is always beneficial when computational resources allow for it. To this end, we hope work in this field culminates in a cohesive theory at the interface of neural networks and Bayesian inference.

5.2 Future work

In the future, we would like to outfit a deep kernel with both convolutional and recurrent neural networks to capture the spatiotemporal regularities in neuroimaging data. Specifically, we could use a convolutional neural network on raw fMRI data, which is then fed into a recurrent neural network to learn temporal structure. Lastly, the recurrent neural network would provide the encoding for the Gaussian process to perform classification. As a result, we would be able to study the effectiveness of atlases as a representation for classification in fMRI studies by comparing them to the atlases generated by the convolutional-recurrent kernel on raw fMRI. While this could combine the strengths of both of our methods, it would also be very computationally challenging.

Another approach could compose multiple Gaussian process with neural network kernels. This would merge the existing works on deep kernel learning [80] and deep Gaussian processes [21]. The tractability of this remains unclear, since deep Gaussian processes are complex even for simple kernels. Doubly stochastic inference approaches could be useful in this regard, as they capture the uncertainties inherent in Monte Carlo sampling from a distribution which stochastically depends on another variable [73].

Lastly, we would like to explore alternatives to KL-based variational inference. While mini-batch training with stochastic gradients has been limited to KL-based variational inference, recent work has explored an upper bound minimization approach. This new approach uses a different divergence measure [22], but achieves better error rates and more accurate uncertainty estimates. In addition, they claim that their proposed upper bound objective is more stable for Gaussian process classification. Hence, it is worth exploring whether our Bayesian deep kernel learning model would benefit from the approximate inference scheme in [22].

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Software available from tensorflow.org.

[2] Maruan Al-Shedivat, Andrew Gordon Wilson, Yunus Saatchi, Zhiting Hu, and Eric P Xing. Learning Scalable Deep Kernels with Recurrent Structure. *arXiv preprint arXiv:1610.08936*, 2016.

[3] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An Introduction to MCMC for Machine Learning. *Machine Learning*, 50(1):5–43, January 2003.

[4] Elaine Angelino, Matthew James Johnson, and Ryan P. Adams. Patterns of Scalable Bayesian Inference. February 2016.

[5] Özlem Aslan, Xinhua Zhang, and Dale Schuurmans. Convex deep learning via normalized kernels. In *Advances in Neural Information Processing Systems*, pages 3275–3283, 2014.

[6] Anoop Korattikara Balan, Vivek Rathod, Kevin P Murphy, and Max Welling. Bayesian dark knowledge. In *Advances in Neural Information Processing Systems*, pages 3438–3446, 2015.

[7] Matthew James Beal. *Variational Algorithms for Approximate Bayesian Inference.* University of London London, 2003.

[8] Mikhail Belyaev, Evgeny Burnaev, and Yermek Kapushev. Gaussian Process Regression for Structured Data Sets. 2015.

[9] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. *arXiv:1206.5538 [cs]*, June 2012.

[10] Bernd A. Berg. Introduction to Markov Chain Monte Carlo Simulations and their Statistical Analysis. *arXiv:cond-mat/0410490*, October 2004.

[11] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, (just-accepted), 2017.

[12] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[13] Léon Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186. Physica-Verlag HD, 2010.

[14] John Bradshaw, Alexander G de G Matthews, and Zoubin Ghahramani. Adversarial Examples, Uncertainty, and Transfer Testing Robustness in Gaussian Process Hybrid Deep Networks. *arXiv preprint arXiv:1707.02476*, 2017.

[15] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

[16] Roberto Calandra, Jan Peters, Carl Edward Rasmussen, and Marc Peter Deisenroth. Manifold Gaussian processes for regression. In *Neural Networks (IJCNN), 2016 International Joint Conference On*, pages 3338–3345. IEEE, 2016.

[17] George Casella and Edward I. George. Explaining the Gibbs Sampler. *The American Statistician*, 46(3):167–174, August 1992.

[18] Edward Challis, Peter Hurley, Laura Serra, Marco Bozzali, Seb Oliver, and Mara Cercignani. Gaussian process classification of Alzheimer's disease and mild cognitive impairment from resting-state fMRI. *NeuroImage*, 112:232–243, 2015.

[19] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[20] Ivor Cribben and Yi Yu. Estimating whole-brain dynamics by using spectral clustering. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 66(3):607–627, April 2017.

[21] Andreas C. Damianou and Neil D. Lawrence. Deep Gaussian Processes. *arXiv:1211.0358 [cs, math, stat]*, November 2012.

[22] Adji B. Dieng, Dustin Tran, Rajesh Ranganath, John Paisley, and David M. Blei. Variational Inference via \$\chi\$-Upper Bound Minimization. *arXiv:1611.00328 [cs, stat]*, November 2016.

[23] David Duvenaud. *Automatic Model Construction with Gaussian Processes*. PhD thesis, 2014.

[24] Nicha C. Dvornek, Pamela Ventola, Kevin A. Pelphrey, and James S. Duncan. Identifying Autism from Resting-State fMRI Using Long Short-Term Memory Networks. *Machine learning in medical imaging. MLMI (Workshop)*, 10541:362–370, September 2017.

[25] Mark Fiecas, Ivor Cribben, Reyhaneh Bahktiari, and Jacqueline Cummine. A variance components model for statistical inference on functional connectivity networks. *NeuroImage*, 149:256–266, April 2017.

[26] Meire Fortunato, Charles Blundell, and Oriol Vinyals. Bayesian Recurrent Neural Networks. *arXiv preprint arXiv:1704.02798*, 2017.

[27] Charles W. Fox and Stephen J. Roberts. A tutorial on variational Bayesian inference. *Artificial Intelligence Review*, 38(2):85–95, August 2012.

[28] Yarin Gal. Uncertainty in Deep Learning. page 166.

[29] Yarin Gal and Zoubin Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. *arXiv:1512.05287 [stat]*, December 2015.

[30] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059, 2016.

[31] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete Dropout. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3581–3590. Curran Associates, Inc., 2017.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[33] Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

[34] Alex Graves and others. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385. Springer.

[35] Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. Style-based Inverse Kinematics. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 522–531, New York, NY, USA, 2004. ACM.

[36] Umut Güçlü and Marcel A. J. van Gerven. Modeling the Dynamics of Human Brain Activity with Recurrent Neural Networks. *Frontiers in Computational Neuroscience*, 11, February 2017.

[37] James Hensman, Alexander G de G Matthews, and Zoubin Ghahramani. Scalable variational Gaussian process classification. 2015.

[38] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

[39] R. Devon Hjelm, Sergey M. Plis, and Vince Calhoun. Recurrent Neural Networks for Spatiotemporal Dynamics of Intrinsic Networks from fMRI Data. *arXiv:1611.00864 [cs, q-bio]*, November 2016.

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

[41] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*, abs/1502.03167, 2015.

[42] Cijo Jose, Prasoon Goyal, Parv Aggrwal, and Manik Varma. Local deep kernel learning for efficient non-linear svm prediction. In *International Conference on Machine Learning*, pages 486–494, 2013.

[43] Ioannis Karatzas and Steven Shreve. *Brownian Motion and Stochastic Calculus*. Springer Science & Business Media, December 2012.

[44] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[45] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

[46] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

[47] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[48] Martin A. Lindquist. The Statistical Analysis of fMRI Data. *Statistical Science*, 23(4):439–464, November 2008.

[49] Christos Louizos and Max Welling. Multiplicative Normalizing Flows for Variational Bayesian Neural Networks. *arXiv preprint arXiv:1703.01961*, 2017.

[50] David J. C. MacKay. *Bayesian Methods for Adaptive Models*. Phd, California Institute of Technology, 1992.

[51] Andre Marquand, Matthew Howard, Michael Brammer, Carlton Chu, Steven Coen, and Janaina Mourão-Miranda. Quantitative prediction of subjective pain intensity from whole-brain fMRI data using Gaussian processes. *Neuroimage*, 49(3):2178–2189, 2010.

[52] Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, April 2017.

[53] Regina Meszlényi, Krisztian Buza, and Zoltán Vidnyánszky. Resting state fMRI functional connectivity-based classification using a convolutional neural network architecture. July 2017.

[54] Thomas P. Minka. Expectation Propagation for approximate Bayesian inference. *arXiv:1301.2294 [cs]*, January 2013.

[55] T. K. Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60, November 1996.

[56] Radford M. Neal. *Bayesian Learning for Neural Networks.* Lecture Notes in Statistics. Springer-Verlag, New York, 1996.

[57] Ian Osband, John Aslanides, and Albin Cassirer. Randomized Prior Functions for Deep Reinforcement Learning. *arXiv:1806.03335 [cs, stat]*, June 2018.

[58] Pinkal Patel, Priya Aggarwal, and Anubha Gupta. Classification of Schizophrenia Versus Normal Subjects Using Deep Learning. In *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing*, ICVGIP '16, pages 28:1–28:6, New York, NY, USA, 2016. ACM.

[59] Francisco Pereira, Tom Mitchell, and Matthew Botvinick. Machine learning classifiers and fMRI: A tutorial overview. *Neuroimage*, 45(1):S199–S209, 2009.

[60] Bala Rajaratnam and Doug Sparks. Mcmc-based inference in the era of big data: A fundamental analysis of the convergence complexity of high-dimensional chains. *arXiv preprint arXiv:1508.00947*, 2015.

[61] Carl Edward Rasmussen. Gaussian processes in machine learning.

[62] Daniel Revuz and Marc Yor. *Continuous martingales and Brownian motion*, volume 293. Springer Science & Business Media, 2013.

[63] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[64] Yunus Saatci. *Scalable Inference for Structured Gaussian Process Models.* PhD thesis, University of Cambridge, 2012.

[65] Saman Sarraf and Ghassem Tofighi. Classification of Alzheimer's Disease using fMRI Data and Deep Learning Convolutional Neural Networks. *arXiv:1603.08631 [cs]*, March 2016.

[66] Edward Snelson and Zoubin Ghahramani. Local and global sparse Gaussian process approximations. In *Artificial Intelligence and Statistics*, pages 524–531, March 2007.

[67] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[68] Michael L Stein. *Interpolation of spatial data: some theory for kriging.* Springer Science & Business Media, 2012.

[69] Lingbing Tang, Pin Peng, Qibin Zhao, Lihua Gui, and Luochang Qing. Tensor-Based Gaussian Processes Regression Using a Probabilistic Kernel with Information Divergence. *International Journal of Simulation–Systems, Science & Technology*, 16(5), 2015.

[70] Mattias Teye, Hossein Azizpour, and Kevin Smith. Bayesian Uncertainty Estimation for Batch Normalized Deep Networks. *arXiv:1802.06455 [stat]*, February 2018.

[71] Robert Tibshirani. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[72] Michalis Titsias. Variational Learning of Inducing Variables in Sparse Gaussian Processes. In *Artificial Intelligence and Statistics*, pages 567–574, April 2009.

[73] Michalis Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational Bayes for non-conjugate inference. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1971–1979, 2014.

[74] Martin J Wainwright, Michael I Jordan, and others. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

[75] Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference.* Springer Texts in Statistics. Springer-Verlag, New York, 2004.

[76] Dong Wen, Zhenhao Wei, Yanhong Zhou, Guolin Li, Xu Zhang, and Wei Han. Deep Learning Methods to Process fMRI Data and Their Application in the Diagnosis of Cognitive Impairment: A Brief Overview and Our Opinion. *Frontiers in Neuroinformatics*, 12, April 2018.

[77] Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *International Conference on Machine Learning*, pages 1775–1784, 2015.

[78] Andrew G Wilson, Zhiting Hu, Ruslan R Salakhutdinov, and Eric P Xing. Stochastic Variational Deep Kernel Learning. In *Advances in Neural Information Processing Systems*, pages 2586–2594, 2016.

[79] Andrew Gordon Wilson and Ryan Prescott Adams. Gaussian Process Kernels for Pattern Discovery and Extrapolation. *arXiv:1302.4245 [cs, stat]*, February 2013.

[80] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378, 2016.

[81] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. *arXiv:1409.2329 [cs]*, September 2014.

[82] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv:1212.5701 [cs]*, December 2012.

[83] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv:1611.03530 [cs]*, November 2016.

[84] Qibin Zhao, Liqing Zhang, and Andrzej Cichocki. A Tensor-Variate Gaussian Process for Classification of Multidimensional Structured Data. 2013.