UNIVERSITY OF CALIFORNIA,
IRVINE

Improving Statistical Inference through Flexible Approximations

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Statistics

by

Lingge Li

Dissertation Committee:
Prof. Pierre Baldi, Chair
Prof. Babak Shahbaba
Prof. Zhaoxia Yu

2020

DEDICATION

To my parents and friends...

"...the great adventure of statistics is in gathering and using data to solve interesting and important real world problems."

– Leo Breiman

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would like to thank my advisors Professor Pierre Baldi and Professor Babak Shahbaba for showing me how to become an independent researcher. In particular, I am grateful for the intellectual freedom and genuine encouragement that Professor Babak Shahbaba has given; I appreciate the high level guidance and valuable insights from Professor Pierre Baldi. I would also like to thank the department of statistics for the generous support. Lastly, I would like to thank my co-authors, especially Lars Hertel, Dustin Pluta, Nitish Nayak, Andrew Holbrook, Professor Jianming Bian, and Professor Norbert Fortin. All of you have made my academic journey an exciting one!

I would like to acknowledge funding from NIH grant R01-MH115697 and the department of statistics. I would also like to acknowledge Springer Nature who has given me permission to incorporate Chapter 2 into my dissertation.

VITA

Lingge Li

EDUCATION

Doctor of Philosophy in Statistics **2020**
University of California, Irvine *Irvine, California*

Bachelor of Arts in Mathematics **2014**
Pomona College *Claremont, California*

RESEARCH EXPERIENCE

Graduate Research Assistant **2018 – 2020**
University of California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

Pierre Baldi, Jianming Bian, Lars Hertel, **Lingge Li**, "Improved energy reconstruction in NOvA with regression convolutional neural networks," *Physical Review D*, 2019. https://doi.org/10.1103/PhysRevD.99.012011.

Lingge Li, Andrew Holbrook, Babak Shahbaba, Pierre Baldi, "Neural network gradient hamiltonian monte carlo," *Computational Statistics*, 2019. https://doi.org/10.1007/s00180-018-00861-z.

Lingge Li, Nitish Nayak, Jianming Bian, Pierre Baldi, "Efficient neutrino oscillation parameter inference using Gaussian processes," *Physical Review D*, 2020. https://doi.org/10.1103/PhysRevD.101.012001.

CONFERENCE PRESENTATIONS

Lingge Li, Dustin Pluta, Babak Shahbaba, Norbert Fortin, Hernando Ombao, Pierre Baldi, "Modeling dynamic functional connectivity with latent factor Gaussian processes," *Conference on Neural Information Processing Systems 2019.*

Babak Shahbaba, **Lingge Li**, Forest Agostinelli, Mansi Saraf, Gabriel A. Elias, Pierre Baldi, Norbert Fortin, "Hippocampal ensembles represent sequential relationships among discrete nonspatial events," *Neuroscience Society 2018.*

ABSTRACT OF THE DISSERTATION

Improving Statistical Inference through Flexible Approximations

By

Lingge Li

Doctor of Philosophy in Statistics

University of California, Irvine, 2020

Prof. Pierre Baldi, Chair

In the statistics and machine learning communities, there exists a perceived dichotomy between statistical inference and out-of-sample prediction. Statistical inference is often done with models that are carefully specified *a priori* while out-of-sample prediction is often done with "black-box" models that have greater flexibility. The former is more concerned with model theoretical properties when data become infinite; the later focuses more on algorithms that scale up to larger data sets. To a scientist who is outside of these communities, the distinction of inference and prediction might not seem so clear. With technological advancements, scientists can now collect overwhelming amounts of data in various formats and their objective is to make sense of the data. To this end, we propose the synergy of statistical inference and prediction workhorses that are neural networks and Gaussian processes. Despite hardware improvements under Moore's law, ever bigger data and more complex models pose computational challenges for statistical inference. To address these computational challenges, we approximate functional forms of the data to effectively reduce the burden of model evaluation. In addition, we present a case study where we use flexible models to learn scientifically interesting representations of rat memories from experimental data for better understanding of the brain.

Chapter 1

Introduction

1.1 Statistical Inference on Scientific Data

Statistics, as its own discipline that is separate from mathematics, has always been motivated by and driven forward the advancement of sciences. On one hand, many statistical models were created to answer specific scientific questions. On the other hand, no scientific study could be legitimized without presenting compelling statistical evidence. Generally speaking, the loop of statistical analysis for scientific data goes as follows:

- generate hypothesis based on current scientific knowledge
- collect data from either an experiment or an observational study
- build statistical model and estimate relevant parameters
- translate statistical results to new scientific knowledge.

Scientific data in the 21st century have become more challenging to analyze as we are able to collect much more data due to new technologies and study more complex phenomena on the shoulders of previous giants. For instance, the large hadron collider for studying particle physics collects over 2TB of data every hour and that is roughly 10,000 millions lines of tabular data. This is not just a singular example; on the macro level, connected telescopes around the world let us observe an expanded universe while on the micro level, next generation sequencing allows us measure the DNAs of each single cell. At the scientific frontiers, a wide range of questions are raised from the origin and the future of our universe to the innerworkings of our brains. Some of these questions are so exploratory that we do not yet have specific hypotheses, let alone well formulated models and interpretable parameters.

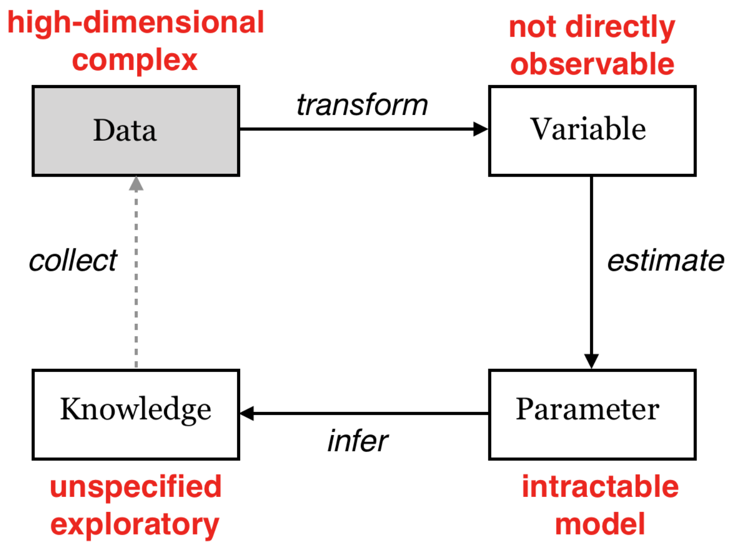


Figure 1.1: Illustration of the data analysis loop and challenges within the loop when science gets to the cutting edge.

The gigantic volume of these data poses computational challenges for statistical analysis. While theoretical statistics relies on mathematical analysis of model behaviors asymptotically, applied statistics has its backbone in computation. In the frequentist framework, mixed effect models commonly used for longitudinal data analysis would not be possible without numerical optimization and integration algorithms. On the other hand, the entire Bayesian framework would have been rendered useless if not for modern computers that can run fast Monte Carlo Markov Chain (MCMC) algorithms.

While computation enables statistical analysis, the translation between statistical results and scientific knowledge is crucial. Without proper and careful interpretation, statistical results can easily lead to unreliable and even false scientific understanding. As many well know, there is a reproducibility crisis in many fields centered at statistical significance. While statistical significance is a valid concept, it does not tell the whole story, is widely mis-used and is often irrelevant. To shed a clear light through fuzzy scientific data, we need better ways of conveying statistical results to the greater scientific audience.

The computational challenges can be attributed to two sources: big data and complex model. Assuming that we use a sequential algorithm on the same machine, the running time should roughly scale linearly with the number of data points. Whereas when the data are held constant, the model complexity can have immense impact; some models do not have an explicit likelihood function and simulating it numerically is time consuming. In this dissertation, we present two lines of work that both attempt to cut out either the large data or the complex model with approximation. The first is neural network gradient hamiltonian monte carlo. While hamiltonian monte carlo is great for sampling from high dimensional posteriors, it requires expensive gradient calculation on the data repeatedly. We propose replacing the data with an approximated gradient field that is parametrized with a neural network. The second is Gaussian process accelerated neutrino oscillation inference. Data models for neutrino osclliation experiments are not directly available and simulations are used to model how neutrino oscillation parameters affect observations. To efficiently utilize computational resources, we use a Gaussian process model to guide an iterative search for confidence intervals of neutrino oscillaiton parameters.

Latent variable models, or so-called representation learning in the machine learning community, offer great potential to extract scientific insights from high-dimensional data. They can reduce the data dimensionality to a reasonable degree while still preserving the structure in the data so that we can make sense of it. To this end, we present an application in a neuroscience experiment where we make use the hidden layer of a neural network model. The learned representation enables us to visualize how the rat hippocampus keeps memories.

Also, inspired by the same experiment, we create a general latent Gaussian process model for studying dynamic functional connectivity in the brain.

1.2 Neural networks

In theory, it is proven that neural networks can provide universal functional approximation [26]; in practice, neural networks have achieved state-of-the-art performance when carefully designed and trained. Researchers have created neural networks to classify certain images at human level, translate languages coherently, and play strategy games better than humans. The success of neural networks largely attributes to the explosion of data and computational power alongside recent developments such as deep network architecture, stochastic gradient descent, and drop-out regularization [56][13][11]. With a variety of model architectures and training techniques, neural networks can be adapted to tasks across data modalities and application domains. In addition, neural networks can learn useful low-dimensional representations of complex data. On the other hand, it often requires a non-trivial amount of human efforts to design and train a suitable neural network. Moreover, we are only starting to understand the probabilistic behaviors of neural networks better.

Neural networks originate from a computational model of a single neuron in the brain [71]. The naive model of a neuron has multiple inputs, denoted by $\vec{x}$, coming from previously connected neurons and a single output y to other connected neurons. The inputs are weighted by the strength of connections from the input neurons and we can denote the weights by $\vec{w}$. Given the total weighted input to the neuron $\vec{w} \cdot \vec{x}$, the neuron either becomes activated or stays inactive. If the neuron is activated, the output would be 1; otherwise the output would be 0. The relationship between the total weighted input and the neuron output is modeled by an activation function f and we have $y = f(\vec{w} \cdot \vec{x})$. The activation function f is often chosen to be sigmoidal. A bias term b could also be added to account for the neuron's innate tendency to be activated or not. The model is therefore given by

$$y = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}. \tag{1.1}$$

For statisticians, this formulation is very similar to a generalized linear model, where f is the link function.

Many of such neurons together form a neural network as illustrated in 1.2. The input neurons on the left do not have previous connections; they form an input layer. On the other hand, the neurons on the right form the output layer; there is only a single output in this case. The layers of neurons in the middle are called hidden layers. Denote each neuron in the first hidden layer by h_{1j} with weights $\vec{w_{1j}}$, each neuron in the second hidden layer by h_{2k} with weights $\vec{w_{2k}}$, and the output neuron by y with weights $\vec{w_o}$. Using successive relations, we can write out the function from the input layer to the output layer as follows

$$h_{1j} = f(\vec{w_{1j}} \cdot \vec{x}), h_{2k} = f(\vec{w_{2k}} \cdot \vec{h_1}), y = f(\vec{w_o} \cdot \vec{h_2}). \tag{1.2}$$

Though each neuron is modeled by a simple function f, the behavior of the entire neural network is much more complex as long as f is non-linear. We can see such an example of a two layer neural network with sigmoidal activation and random weights. Compared to the logistic regression model whose decision boundary is a plane, the neural network is capable of producing surfaces that are more flexible decision boundaries. In 2, we leverage the flexibility of neural networks to approximate the gradient field in Hamiltonian Monte Carlo.

The universal approximation theorem states that neural networks can be arbitrarily close to any function. However, it is not guaranteed that we would find a desired network in practice. To fit a neural network to data, gradient based optimization methods are most commonly

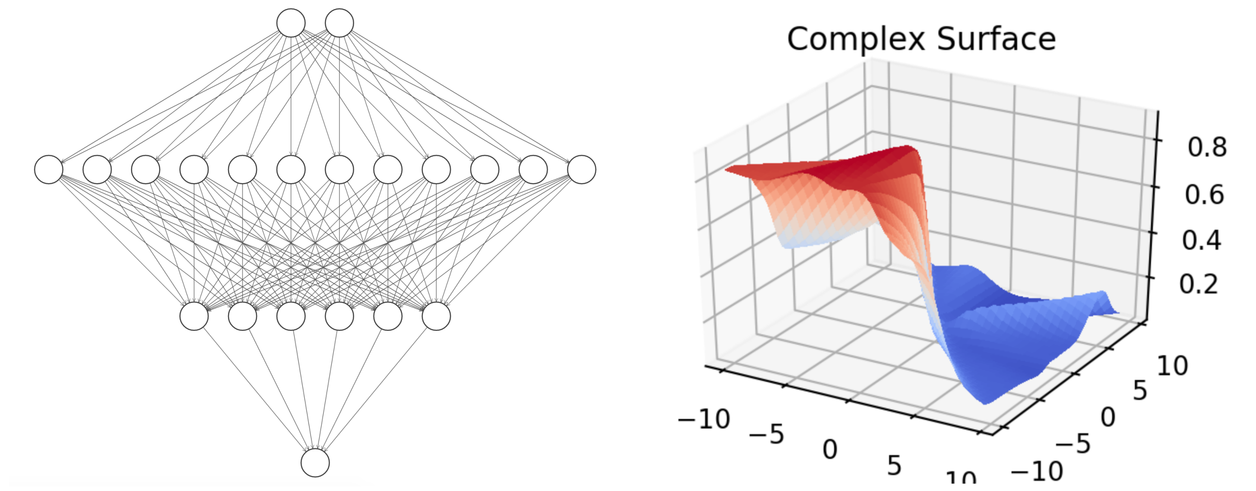


Figure 1.2: Illustration of the neural network architecture with two hidden layers (left) and example output surface of model with random weights (right). The surface can represent a more flexible approximation in $\mathbb{R}^2$ than the logistic function.

used [72]. As neural networks do not have an explicit analytical form, the gradient with respect to weights and biases is calculated for each layer and "back-propagated" with chain rule. This calculation could be tedious by hand but efficiently implemented with automatic differentiation libraries. When there is a large volume of data, performing such gradient calculations on highly optimized computing hardware can still be time consuming. Therefore, second order derivatives are rarely used to train neural networks. To make model fitting more scalable, stochastic gradient descent and its variants are usually used. In stochastic gradient descent, subsets of the training data are randomly sampled for gradient calculation in every iteration.

Given the non-convexity of neural networks, it is surprising that stochastic gradient descent should succeed at all. For a deep neural network, the landscape of loss function is extremely high dimensional and treacherous due to the number of parameters and layers of non-linearity. The noise in stochastic gradient descent actually helps the training algorithm escape saddle points and find good local minima. However, the uncertainty in the trained parameters is not directly measurable as their sampling distribution is intractable. In practical settings where data are sufficiently big, usual numerical methods such as sampling and bootstrap are not feasible for neural networks because of the high computational cost. One ad hoc method is to inject noise during prediction as well as during training with the drop-out algorithm. The drop-out algorithm randomly removes neurons or their connections from the network in every iteration and can be related to approximate Bayesian inference [35].

One important advantage of using neural networks is that one could compose new model architecture with simple building blocks to take advantage of the structure in the data. Besides feedforward neural networks, convolutional and recurrent neural networks are preferred for computer vision and natural language processing [51][40]. Different from feedforward neural networks, they leverage parameter sharing to be more data efficient and customized for specific tasks. Convolutional neural networks run the same filters across an image and can capture translation invariance within the image. Recurrent neural networks have internal states based on past inputs and can account for temporal dependence in sequential data such as text. The latest image classification models have so-called inception modules that perform convolution at various scales and the attention mechanism for language translation combines a feedforward network with a recurrent network. In 4, we create a new convolutional architecture that is able to efficiently extract local interactions between two types of neuronal data.

Another important advantage of neural networks is representation learning [15]. Each layer of the neural network can be considered as a latent variable; from the input data to the output target, these latent variables can gradually become lower dimensional and more interpretable. In the most basic case of an auto-encoder, the neural network has one hidden layer that is smaller than the input dimension and maps it back to the input data; the auto-encoder can be considered to perform non-linear principal component analysis (PCA). When we have data such as images and text that live on complicated manifolds, neural network layers could allow for modeling in the Euclidean space. The most prominent example is the word2vec model that learns a transformation from the discrete space of words to latent vectors. The latent vectors form a continuous representation of words that preserves their semantic meanings. In 4, we visualize the latent representation of different stimuli in the rat hippocampus to provide evidence for the phenomenon of sequential memory replay.

1.3 Gaussian processes

Gaussian processes can be considered as probability distributions in the functional space. By definition, a Gaussian process ($\mathcal{GP}$) is a collection of random variables, any finite number of which have a joint Gaussian distribution [69]. To understand the Gaussian process, one could start with a multivariate Gaussian with probability density function $f(\vec{x})$. Consider $\mathbb{R}^{[0,1]}$ which is the space of functions from $[0,1]$ to $\mathbb{R}$. Define a σ-algebra as

$$\{f : f(a_1, ..., a_k) \in \mathcal{B}\}$$

where $a_1, ..., a_k$ are k fixed points in $[0,1]$. Then the k-dimensional Gaussian probability measure induces a probabiliy measure on $\mathbb{R}^{[0,1]}$. If we extend the Gaussian to infinite dimensions, we would obtain a probability measure on all continuous functions from $\mathbb{R}$ to $\mathbb{R}$, allowing great flexibility for modeling. In practice, data sets are finite and modeling with $\mathcal{GP}$s boils down to working with multivariate Gaussians. Therefore, making probabilistic inference with $\mathcal{GP}$s could be straightforward with closed analytical forms, making it very attractive for certain applications where model uncertainty is important. However, when data are abundant, $\mathcal{GP}$s do not scale as easily as neural networks; they are also less easily adaptable to different data modalities.

Typically, the zero mean $\mathcal{GP}$ is used for modeling as a prior on the unknown function to approximate and the $\mathcal{GP}$ posterior updated on observed data would be used for prediction [16]. It is impossible to specify an infinite-dimensional covariance matrix explicitly. Instead, we can parametrize a zero mean $\mathcal{GP}$ with a kernel function κ that defines the pairwise covariance. Let $f \sim \mathcal{GP}(0, \kappa(\cdot, \cdot))$. Then for any pair x and x' we have

$$\begin{pmatrix} f(x) \\ f(x') \end{pmatrix} \sim \mathcal{N}(0, \begin{bmatrix} \kappa(x,x) & \kappa(x,x') \\ \kappa(x,x') & \kappa(x',x') \end{bmatrix}).$$

Given a finite set of observed data $\vec{x}_{obs}$, we can write down the multivariate Gaussian likelihood with covariance matrix $\kappa(\vec{x}_{obs}, \vec{x}_{obs})$ and maximize it through kernel parameters ω. Conveniently, at a new point x^* we can obtain the closed form predictive distribution:

$$f(x^*)|f(\vec{x}_{obs}) \sim N(\kappa(x^*, \vec{x}_{obs})(\kappa(\vec{x}_{obs}, \vec{x}_{obs}))^{-1} f(\vec{x}_{obs}),$$
$$\kappa(x^*, x^*) - \kappa(x^*, \vec{x}_{obs})(\kappa(\vec{x}_{obs}, \vec{x}_{obs}))^{-1} \kappa(\vec{x}_{obs}, x^*)).$$

Despite the closed form, in order to calculate the $\mathcal{GP}$ likelihood, one needs to evaluate the probability density function of the multivariate Gaussian repeatedly, which requires inverting the covariace matrix. Typically, inverting the covariance matrix involves all the data compared to only a small subset in stochastic gradient descent; the inversion cost is prohibitively $\mathcal{O}(n^3)$. A general strategy to speed up computation for $\mathcal{GP}$ is using inducing points [75]. Inducing points are a set of pseudo inputs based on the data points and produce a very similar posterior. This is perhaps better understood through decomposing the covariance matrix into blocks with an inducing point in each. The covariance between points in different blocks is approximated by the between block covariance and the covariance with inducing points.

Since a $\mathcal{GP}$ is uniquely characterized by the kernel, different kernels produce distinct behaviors [29]. A commonly used kernel is squared exponential $\kappa(x_1, x_2) = \exp(-\frac{(x_1-x_2)^2}{l^2})$ where l is called the length scale. Intuitively, the length scale determines the distance over which the $\mathcal{GP}$ interpolates between points. The squared exponential kernel is infinitely differentiable and functions drawn from such a $\mathcal{GP}$ would be smooth. However, this smoothness assumption might not be appropriate for some applications; a more general kernel is the Matérn kernel. The Matérn kernel has an additional parameter ν that controls the smoothness and the squared exponential kernel is a special case where $\nu \to \infty$. Other common kernels include white noise kernel for modeling observational noise and periodic kernel that could capture seasonal trends. More importantly, kernels can be composed together through addition and multiplication to approximate different structures in the data.

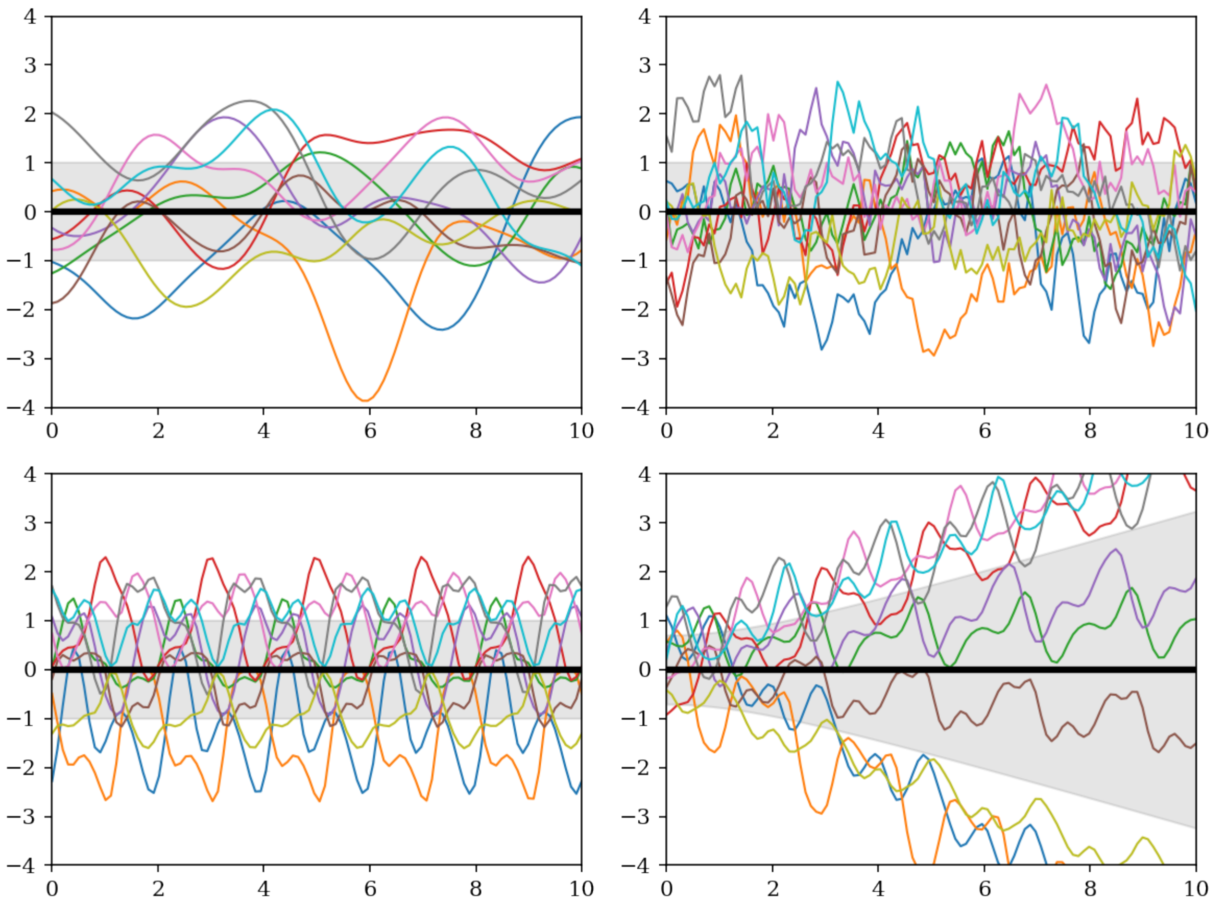


Figure 1.3: Sampled 1-dimensional paths from Gaussian processes with different kernels exhibit distinct behaviors. Note that the bottom right $\mathcal{GP}$ path has multiple features as the kernel is composed of linear and periodic functions.

Gaussian processes and neural networks both have great flexibility to approximate functions. Mathematically, Gaussian processes and neural networks can be shown to be equivalent [59].

A neural network with Gaussian priors on the hidden layer has the asymptotic properties of a Gaussian process. Gaussian processes are also used as latent variables to learn useful representations from data [49]. For many simple prediction tasks, specifying an appropriate kernel and performing the computation for Gaussian processes require more expertise whereas high-level libraries for neural networks are easily accessible and make rapid iterations possible. However, in the context of statistical inference and sequential decision making, the availability of full posterior distribution makes Gaussian processes much more attractive than neural networks, whose parameter uncertainty is hard to quantify. In 3 we use a Gaussian process to iteratively explore the bounded parameter space for neutrino oscillation inference. In 5, we propose a new dynamic functional connectivity model with latent Gaussian processes.

Chapter 2

Neural Network Gradient Hamiltonian Monte Carlo

2.1 Introduction

Hamiltonian Monte Carlo is a widely used algorithm for sampling from posterior distributions of complex Bayesian models. It can efficiently explore high-dimensional parameter spaces guided by simulated Hamiltonian flows. However, the algorithm requires repeated gradient calculations, and these computations become increasingly burdensome as data sets scale. We present a method to substantially reduce the computation burden by using a neural network to approximate the gradient. First, we prove that the proposed method still maintains convergence to the true distribution though the approximated gradient no longer comes from a Hamiltonian system. Second, we conduct experiments on synthetic examples and real data to validate the proposed method.

Hamiltonian Monte Carlo (HMC) uses local geometric information provided by the log-posterior gradient to explore the high posterior density regions of the parameter space [61]. Compared to the Metropolis-Hastings random walk algorithm, HMC has high acceptance probability and low sample auto-correlation even when the parameter space is high-dimensional. That said, the advantages of HMC come at a computational cost that limits its application to smaller data sets. The gradient calculation involves the entire data set and scales linearly with the number of observations. As HMC needs to calculate the gradient multiple times within every single step, performing HMC on millions of observations requires an enormous computational budget. Allowing HMC to scale to large data sets would help tackle the double challenge of big data and big models.

There have been two main approaches to scaling HMC to larger data sets. The first is stochastic gradient HMC, which calculates the gradient on subsets of the data. [83] implemented a stochastic gradient version of Langevin Dynamics, which may be viewed as single-step HMC. [24] introduced stochastic gradient HMC with "friction" to counterbalance the inherently noisy gradient. However, these methods may not be optimal because subsampling substantially reduces the acceptance probability of HMC [17].

The second approach relies on a surrogate function, the gradient of which is less expensive to calculate. [70][47] used Gaussian process (GP) to produce satisfactory results in lower dimensions. However, training a GP is itself computationally expensive and training points must be chosen with great care. More recently, [85] implemented neural network surrogate with random bases. It outperforms GP surrogate in their experiments but fails in parameter spaces of moderate dimensionality.

In this paper, we develop a third approach, neural network gradient HMC (NNgHMC), by using a neural network to directly approximate the gradient instead of using it as a surrogate. We also train all the neural network weights through backpropagation rather than having random weights [85]. Compared to existing methods, our proposed approach can emulate Hamiltonian flows accurately even when dimensionality increases. In Section 3, details of our method and proof of convergence are presented. Section 4 includes experiments to validate our method and comparisons with previous methods on synthetic and real data.

▢ 2.2 Background

▢ 2.2.1 Hamiltonian Monte Carlo

Let $x \sim \pi(x|q)$ denote a probabilistic model with q as its corresponding parameter. We also make q a random variable by giving the parameter a prior distribution $\pi(q)$. The integration constant of the posterior distribution

$$\pi(q|x) = \frac{\pi(x|q)\pi(q)}{\int \pi(x|q)\pi(q)\, dq} \tag{2.1}$$

is usually analytically intractable, but the distribution can be numerically simulated using MCMC. The Metropolis-Hastings algorithm constructs a Markov chain that randomly proposes a new state q' from current state q based on transition distribution $g(q'|q)$ and moves from q to q' with probability $\min\{1, \frac{\pi(q'|X)g(q|q')}{\pi(q|X)g(q'|q)}\}$. Unfortunately, in a higher dimensional space, the probability of randomly moving to q' drops dramatically. Therefore, the MH algorithm has trouble exploring the posterior efficiently in higher dimensions.

The idea of HMC is to explore a frictionless landscape induced by potential energy function U and kinetic energy function K where potential energy $U(q) = -\log \pi(x|q)\pi(q)$ is proportional to the negative log posterior. HMC introduces an auxiliary Gaussian momentum p, and $K(p)$ is the negative log density of p. Potential energy U tends to convert to kinetic energy K so q will likely move to a position with higher posterior density. More formally, the Hamiltonian system is defined by the following equations.

$$H(q,p) = U(q) + K(p) = -\Big(\log \pi(q) + \sum_{i=1}^{N} \log \pi(x_i|q)\Big) + \frac{1}{2}p^T p\,, \tag{2.2}$$

$$\frac{dq}{dt} = \frac{\partial H}{\partial p} = \frac{\partial K}{\partial p} = p \tag{2.3}$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q} = -\frac{\partial U}{\partial q} = \nabla_q\Big(\log \pi(q) + \sum_{i=1}^{N} \log \pi(x_i|q)\Big)\,. \tag{2.4}$$

In theory, convergence of HMC is guaranteed by the time reversibility of the Hamiltonian dynamics which, in turn, renders the Markov chain transitions reversible, thus ensuring detailed balance. By conservation of the Hamiltonian, HMC has acceptance probability 1 and can travel arbitrarily long trajectories along energy level contours. In practice, the Hamiltonian dynamics is simulated with the leapfrog algorithm which adds numerical errors. To ensure convergence to the posterior, a Metropolis correction step is used at the end of each trajectory.

Within each simulated trajectory, the leapfrog algorithm iterates back and forth between Equations (2.3) and (2.4), the latter of which features a summation over the log-likelihood evaluated at *each* separate data point. For large data sets, this repeated evaluation of the gradient becomes infeasible. In Section 2.3, we show how to greatly speed up HMC using neural network approximations to this gradient term, but first we introduce an important predecessor to our method, the surrogate HMC class of algorithms.

▢ 2.2.2 Surrogate HMC

Two methods for approximating the log-posterior in the HMC context have already been advanced. The first uses a Gaussian Process regression to model the log-posterior, the second uses a neural network. We refer to the latter as neural network surrogate HMC (NNsHMC). It is natural that both models would be used in such a capacity: Cybenko [26] showed that neural networks can provide universal function approximation, and Neal [59] showed that certain probabilistic neural networks converge to Gaussian processes as the number of hidden units goes to infinity. In this section, we focus on NNsHMC, since it is more closely related to our method (Section 2.3).

NNsHMC approximates the potential energy U with a neural network surrogate $\widehat{U}$ and uses $\nabla\widehat{U}$ during leapfrog steps. The surrogate neural network has one hidden layer with softplus activation:

$$\widehat{U}(q) = W_2 \ln(1 + \exp(W_1 q + b_1)) + b_1 \tag{2.5}$$

where W_1, W_2 and b_1, b_2 are weight matrices and bias vectors, respectively. Under this formulation, one can explicitly calculate the gradient

$$\nabla\widehat{U} = W_1^T diag(W_2) \frac{1}{1 + \exp\left(-(W_1 q + b_1)\right)} \tag{2.6}$$

and represent $\nabla\widehat{U}$ with another neural network, which is just the backpropagation graph of $\widehat{U}$. Therefore, we can view neural network surrogate as using a constrained network with tied weights and local connections to approximate the gradient.

For training the neural network, Zhang et al [85] uses extreme learning machine (ELM) [43]. ELM is a simple algorithm that randomly projects the input to the hidden layer and only trains the weights from the hidden layer to the output. Random projection is widely used in machine learning but backpropagation is the “default” training method for most neural networks with its optimality theoretically explained by [12]. Moreover, since the goal is to improve computational efficiency, we want to make the surrogate neural network as small as possible. From this point of view, large hidden layers often seen in ELMs are less than optimal.

2.3 Neural network gradient HMC

Algorithm 2.1 Neural network gradient HMC

```
Initialize $q^{(0)}$, leapfrog step number $L$ and step size $\epsilon$
for $t = 1, 2, ..., T$ do
    $q_0 = q^{(t-1)}$
    Sample momentum $p_0 \sim \mathcal{N}(0, I)$
    $p_0 = p_0 - \frac{\epsilon}{2}\widehat{\nabla U}(q_t)$    ▷ Leapfrog steps with approximated gradient $\widehat{\nabla U}$ instead of $\nabla U$
    for $i = 1, 2, ..., L$ do
        $q_i = q_{i-1} + \epsilon p_{i-1}$
        $p_i = p_{i-1} - \epsilon\widehat{\nabla U}(q_i)$
    end for
    $p_L = p_L - \frac{\epsilon}{2}\widehat{\nabla U}(q_L)$
    $r = \exp(H(q_L, p_L) - H(q_0, p_0))$, $u \sim Uniform(0, 1)$
    if $u < \min(1, r)$ then                ▷ Metropolis accept/reject based on $H = U + K$
        $q^{(t)} = q_L$
    else
        $q^{(t)} = q_0$
    end if
end for
```

In contrast to previous work, NNgHMC does not use a surrogate function for U but fits a neural network to approximate ∇U directly with backpropagation. Training data $(q, \nabla U(q))$ for the neural network are collected during the early period of HMC shortly after convergence. Once the approximate gradient is learned, the algorithm is exactly the same as classical HMC, but with neural network gradient $\widehat{\nabla U}$ replacing ∇U. Details are given in Algorithm 1.

One benefit of our method occurs as early as the data collection process. Since we approximate the gradient ∇U and not U, we can collect more training data faster: surrogate HMC must reach the end of a leapfrog trajectory before obtaining a single (functional evaluation) training sample; the same leapfrog trajectory renders a new (gradient evaluation) training sample for each leapfrog step, and the number of such steps in a single trajectory can range into the hundreds.

Suppose that there are N data points x_n and that the parameter space is d-dimensional. In this case, gradient calculations involve d partial derivatives

$$\frac{\partial U}{\partial q_j} = -\frac{\partial}{\partial q_j} \log \Big(\pi(q) \prod_{i=1}^{N} \pi(x_i|q)\Big) = -\frac{\partial}{\partial q_j} \log \pi(q) - \sum_{i=1}^{N} \frac{\partial}{\partial q_j} \log \pi(x_i|q)\,, \tag{2.7}$$

each of which involves a summation over the N data points. On the other hand, performing a forward pass in a shallow neural network is proportional only to the hidden layer size $s \ll N$. Once the neural network is trained on burn-in samples, posterior sampling with approximated gradient is orders of magnitude faster.

Although the neural network gradient approximation $\widehat{\nabla U}(q)$ is not the same as $\nabla U(q)$, the method nonetheless samples from the true posterior. If one were able to simulate the Hamiltonian system directly, i.e. without numerical integration, then all the benefits of HMC would be preserved in the limit, as the gradient field approximates the true gradient field to arbitrary degree. On the other hand, the NNgHMC transition kernel—characterized by the approximate gradient leapfrog integrator combined with the Metropolis accept-reject step—leaves the posterior distribution invariant. We formalize the relevant results here and defer proofs to the appendix.

An important litmus test for the validity of our method is that it should leave the Hamiltonian invariant in the limit as step-sizes and gradient approximation errors approach zero. In turn, this result will imply high acceptance probabilities when the system is simulated from numerically, and when gradient approximations are good.

Proposition 1. *When the system induced by the approximate gradient field is simulated directly, changes in the Hamiltonian $H(q,p) = U(q) + K(p)$ converge in probability to 0 as the approximate gradient converges pointwise to the true gradient. That is, for a sequence of approximate gradient fields $\{\widehat{\nabla_q^n U}\}_{n=1}^{\infty}$ converging to the true gradient field $\nabla_q U$, the change* *in Hamiltonian values satisfies*

$$\left(\frac{dH}{dt}\right)_n = o_p(1)\,. \tag{2.8}$$

Proof. Following [26], assume we are able to construct a sequence of approximate gradients $\widehat{\nabla_q^n H}$ satisfying

$$\nabla_q H = \widehat{\nabla_q^n H} + E_n(q), \quad E_n(q) \in B_{1/n}(0)\,, \tag{2.9}$$

where $B_{1/n}(0)$ is the ball around the origin of radius $1/n$. In this case, the vector field given by the approximate gradient induces a new system of equations:

$$\begin{aligned}\frac{dq_i}{dt} &= \frac{\partial H}{\partial p_i} \\ \frac{dp_i}{dt} &= -\widehat{\frac{\partial H}{\partial q_i}}\,.\end{aligned} \tag{2.10}$$

Then it follows that

$$\begin{aligned}\frac{dH}{dt} &= \sum_{i=1}^{d}\Big[\frac{dq_i}{dt}\frac{\partial H}{\partial q_i} + \frac{dp_i}{dt}\frac{\partial H}{\partial p_i}\Big] \\ &= \sum_{i=1}^{d}\Big[\frac{\partial H}{\partial p_i}\big(\widehat{\frac{\partial H}{\partial q_i}} + E_{n,i}(q)\big) - \widehat{\frac{\partial H}{\partial q_i}}\frac{\partial H}{\partial p_i}\Big] \\ &= \sum_{i=1}^{d}\frac{\partial H}{\partial p_i}E_{n,i} \\ &= p^T E_n \sim N(0, E_n^T E_n)\,.\end{aligned} \tag{2.11}$$

This last line implies $p^T E_n$ is $O_p(\sqrt{E_n^T E_n})$, and hence that $\frac{dH}{dt}$ is $o_p(1)$. □

We note that Proposition 1 is a local result, and that local deviations from the true Hamiltonian flow will accrue to larger global deviations in general. While this may seem disconcerting, NNgHMC maintains remarkably high acceptance rates in practice. To help understand why this is the case, we present local and global error analyses for the dynamics of the ordinary differential equation initial value problem

$$\frac{d}{dt}z = f(z)\,, \quad z(t_0) = z^0 \in \mathbb{R}^k\,, \tag{2.12}$$

approximated with function $\widehat{f} \approx f$. These results will then be related back to NNgHMC by specifying $z = (q, p)$ and

$$z = (q,p)^T, \quad f(q,p) = \Big(p, -\frac{\partial H}{\partial q}\Big)^T, \quad \text{and} \quad \widehat{f}(q,p) = \Big(p, -\widehat{\frac{\partial H}{\partial q}}\Big)^T. \tag{2.13}$$

The general form of the following proofs follows after Section 2.1.2 of [52].

Proposition 2. *(Local error bounds) Let $z^0 = z(0)$ be the initial value, let $z(\Delta t)$ be the value of the exact, true trajectory after traveling for time Δt, and let z^1 be the value of the computed trajectory using Euler's method applied to the approximated gradient field. Finally, assume that the exact solution is twice continuously differentiable. Then the local error $\epsilon^1 = z(\Delta t) - z^1$ has the following bounds:*

$$\|\epsilon^1\| \leq \Delta t\, \delta + O(\Delta t^2)\,, \tag{2.14}$$

where $\delta = \|f(z_0) - \widehat{f}(z_0)\|$ measures the difference between the true, exact trajectory and the approximated trajectory at point z_0.

Proof. The proof follows from the Taylor expansion of both $z(\Delta t)$ and z^1:

$$\begin{aligned} \epsilon^1 &= \big(z_0 + \Delta t\, \dot{z}(0) + \frac{1}{2}\Delta t^2 \ddot{z}(\tau)\big) - \big(z_0 + \Delta t \widehat{f}(z_0)\big) \\ &= \Delta t\big(f(z_0) - \widehat{f}(z_0)\big) + \frac{1}{2}\Delta t^2 \ddot{z}(\tau)\,, \end{aligned} \tag{2.15}$$

where $\tau \in [0, \Delta t]$. The result follows from the triangular inequality. $\square$

From the above result, it follows that the local error rate approaches the $O(\Delta t^2)$ error rate of Euler's method using the true gradient field as $\delta = \|f(z_0) - \widehat{f}(z_0)\| = \|\frac{\partial H}{\partial q}(z_0) - \frac{\widehat{\partial H}}{\partial q}(z_0)\|$ goes to 0. The same approach can be used to obtain global error bounds.

Proposition 3. *(Global error bounds) We adopt the same notation as above with the addition of the error at iteration n, $\epsilon^n = z(n\Delta t) - z^n$, where z^n is the value after n Euler updates using the approximate gradient field. Also, let $t_n = n\Delta t$. Again we assume that the exact solution is twice differentiable, and we further assume that it is Lipschitz with constant L. Then the following bounds on ϵ^n hold:*

$$\|\epsilon^n\| \leq \big(e^{n\Delta t L} - 1\big)\big(\frac{\delta}{L} + O(\Delta t)\big)\,, \quad \textit{for} \quad \delta = \max \|f(z(j\Delta t)) - \widehat{f}(z^j)\|\,, \tag{2.16}$$

and $j = 0, \ldots, n$.

Proof. The proof proceeds by recursion. Assume that we have obtained $\epsilon^n = z(t_n) - z^n$. Letting $\tau \in [t_n, t_{n+1}]$, a Taylor's expansion gives:

$$\begin{aligned} \epsilon^{n+1} &= \Big(z(t_n) + \Delta t \dot{z}(t_n) + \frac{1}{2}\Delta t^2 \ddot{z}(\tau)\Big) - \Big(z^n + \Delta t \widehat{f}(z^n)\Big) \\ &= \Big(z(t_n) + \Delta t\, f(z(t_n)) + \frac{1}{2}\Delta t^2 \ddot{z}(\tau)\Big) - \Big(z^n + \Delta t \widehat{f}(z^n)\Big) \\ &= \big(z(t_n) - z^n\big) + \Delta t\big(f(z(t_n)) - \widehat{f}(z^n)\big) + \frac{1}{2}\Delta t^2 \ddot{z}(\tau)\,. \end{aligned} \tag{2.17}$$

But $\ddot{z}$ is continuous by assumption, so we can bound $\ddot{z}$ on the closed interval $[t_n, t_{n+1}]$ by a constant M. Furthermore, the Lipschitz assumption combined with the triangle inequality give:

$$
\begin{aligned}
\|\epsilon^{n+1}\| &\leq \|\epsilon^n\| + \Delta t\Big(\|f(z(t_n)) - f(z^n)\| + \|f(z^n) - \widehat{f}(z^n)\|\Big) + \frac{\Delta t^2 M}{2} \\
&\leq (1 + \Delta t L)\|\epsilon^n\| + \Delta t\, \delta + \frac{\Delta t^2 M}{2}
\end{aligned}
\tag{2.18}
$$

Next we make use of the following recursion relationship:

$$
a_{n+1} \leq C\, a_n + D \Longrightarrow a_n \leq C^n\, a_0 + \frac{C^n - 1}{C - 1} D \tag{2.19}
$$

for $C = (1 + \Delta t L)$ and $D = \Delta t\, \delta + \Delta t^2 M/2$. Noting that $a_0 = \epsilon^0 = 0$ gives

$$
\|\epsilon^n\| \leq (e^{t_n L} - 1)\big(\frac{\delta}{L} + \frac{\Delta t M}{2L}\big)\,, \tag{2.20}
$$

and the result follows. □

The above result suggests that the usual numerical error caused by a large Lipschitz constant L can overpower the effects of gradient approximation error δ.

The preservation of volume entailed by both the theoretical Hamiltonian flow and the leapfrog integrator is important for HMC. The latter fact implies there is no need for Jacobian corrections within the accept-reject step. It turns out that the NNgHMC dynamics also preserve volume, both for direct and for leapfrog simulation.

Lemma 1. *Both for infinitesimal and finite step sizes, the NNgHMC trajectory preserves volume.*

Proof. For the finite case, the leapfrog algorithm iterates between shear transformations and so preserves volume [61]. For the case of direct simulation, we use the fact that the Hamiltonian vector field induced by the approximate gradient field has zero divergence (Liouville's

Theorem, [61]). We use the notation of Proposition 1, but drop the subscript n for the sake of readability:

$$\begin{aligned}\sum_{i=1}^{d}\Big[\frac{\partial}{\partial q_i}\frac{dq_i}{dt}+\frac{\partial}{\partial p_i}\frac{dp_i}{dt}\Big] &= \sum_{i=1}^{d}\Big[\frac{\partial}{\partial q_i}\frac{\partial H}{\partial p_i}-\frac{\partial}{\partial p_i}\widehat{\frac{\partial H}{\partial q_i}}\Big] \\ &= \sum_{i=1}^{d}\Big[\frac{\partial}{\partial q_i}\frac{\partial H}{\partial p_i}-\frac{\partial}{\partial p_i}\big(\frac{\partial H}{\partial q_i}-E_i\big)\Big] \\ &= \sum_{i=1}^{d}\frac{\partial}{\partial p_i}E_i = 0\,.\end{aligned} \tag{2.21}$$

□

Not only does the NNgHMC trajectory preserve volume, it is reversible as well. This easy fact is shown in the proof of Proposition 2.

Theorem 1. *The NNgHMC transition kernel leaves the canonical distribution* $\exp\{-H(q,p)\}$ *invariant.*

Proof. Since leapfrog integration preserves volume and since the Metropolis acceptance probability is the same as for classical HMC, all we need to show is that the leapfrog integration is reversible. This fact follows in the exact same way as for HMC, despite the use of an approximate gradient field to generate the dynamics:

$$\begin{aligned}p_i(t+\epsilon/2) &= p_i(t)-(\epsilon/2)\,\widehat{\frac{\partial U}{\partial q_i}}(q(t)) \\ q_i(t+\epsilon) &= q_i(t)+\epsilon\, p_i(t+\epsilon/2) \\ p_i(t+\epsilon) &= p_i(t+\epsilon/2)-(\epsilon/2)\,\widehat{\frac{\partial U}{\partial q_i}}(q(t+\epsilon))\,.\end{aligned} \tag{2.22}$$

These are the same equations as in [61] except with $\widehat{\frac{\partial U}{\partial q_i}}$ replacing $\frac{\partial U}{\partial q_i}$. Hence, just as in [61], the NNgHMC leapfrog equations are symmetric and thus reversible: to reverse a sequence of leapfrog dynamics, negate p, take the same number of steps, and negate p again. It follows that the NNgHMC transition kernel leaves the canonical distribution invariant and is an asymptotically exact method for sampling from the posterior distribution. □

Regardless of the accuracy of neural network gradient approximation, following the leapfrog simulated Hamiltonian proposal scheme would recover the true posterior distribution when combined with Metropolis-Hastings correction. If the gradient approximation is "bad," NNgHMC would break down to a random walk algorithm. If the gradient approximation is "close enough," NNgHMC would behave just like standard HMC, operating on energy level contours at a fraction of the computation cost. *The neural network gradient approximation can be controlled with two tuning parameters: hidden layer size h and training time t, in addition to leapfrog steps l and step size s.* The neural network architecture is fixed to have one hidden layer and the number of units has to be pre-determined. Neural network training time can be either set to some number of epochs or dependent on a stopping criterion (typically based on change in loss function between epochs). Since there is no noise (error) in the gradient, overfitting is not a concern; the hidden layer size and training time could be relatively large.

Given sufficient training data, the neural network will be able to accurately approximate the gradient field. The important question is: how much training data should be collected? To address this, we propose a training schedule that consists of a start point, an end point, and a rate for gradient data collection. For example, one may wish to run a HMC chain to draw 5000 samples in total. A training schedule could be training the neural network every 200 draws between the 400th and 1000th draws. After the neural network is trained each time, one would use the approximated gradient to sample for some iterations. If the acceptance probability is similar to that of standard HMC, one would stop the training schedule and complete the entire chain with NNgHMC. Otherwise, standard HMC would be used to sample the remaining draws. *Since training the neural network and using it to sample is much cheaper computationally compared to standard HMC, the training schedule* *would add little overhead even if the neural network gradient approximation fails.*

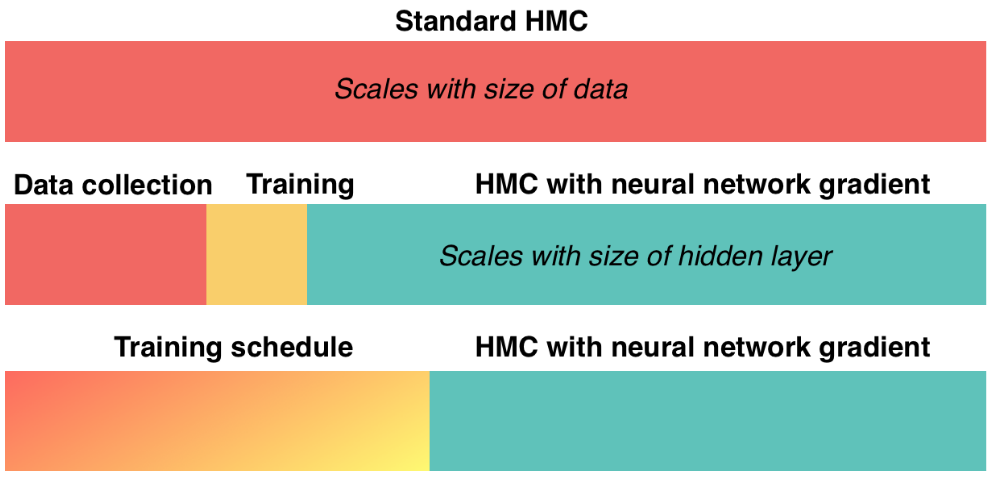

Standard HMC
Scales with size of data
Data collection
Training
HMC with neural network gradient
Scales with size of hidden layer
Training schedule
HMC with neural network gradient

Figure 2.1: After the neural network learns an accurate gradient approximation, the computation cost of sampling is substantially reduced compared to standard HMC. Therefore, the benefit of neural network gradient HMC depends on how much training data is enough for the neural network. Using a training schedule, we would stop standard HMC immediately after the neural network has learned from enough data.

2.4 Experiments

In this section, we demonstrate the merits of proposed method: accuracy and scalability through a variety of experiments. The accuracy of gradient approximation can be reflected by high acceptance probability that is similar to standard HMC using the true gradient. Compared to draws from stochastic gradient HMC, the draws using proposed method are much more similar to standard HMC draws. Scalability means better performance when both data size n and dimensionality p increase. The performance metric is effective sample size (ESS) adjusted by CPU time. ESS estimates the number of "independent" samples by factoring $\rho(k)$ correlation between samples at lag k into account:

$$ESS = \frac{n}{1 + 2\sum_{k=1}^{\infty} \rho(k)}.$$

The previous surrogate approach fails when p reaches 40 while the proposed method works well up to $p = 200$. *Lastly, speed evaluation is done on three real data sets and the proposed method consistently beats standard HMC even when the time to collect training data and train the neural network is included.* Our proposed NNgHMC method is implemented in Keras and uses the default Adam optimizer [46] during training. All experiments are performed on a 3.4 GHz Intel Quad-Core CPU and our code is available at: https://github.com/modestbayes/hamiltonian.

2.4.1 Distributions with challenging gradient fields

The banana shaped distribution in two dimensions can be sampled using the following un-normalized density

$$f(x_1, x_2) \propto \exp -\frac{(Ax_1)^2}{200} - \frac{1}{2}(Cx_2 + B(Ax_1)^2 - 100B)^2 \tag{2.23}$$

where A, C control the scale in x_1, x_2-space and B determines the curvature. For HMC, the energy function is set to be $-\log f(x_1, x_2)$ and the its gradient can be easily calculated. Using leapfrog steps $l = 5$ and step size $s = 0.1$, standard HMC is used to sample 5000 draws with acceptance probability 0.58. Gradient values collected during the first 1000 draws are then used to train a neural network with hidden layer size $h = 100$ for $t = 50$ epochs. With the same tuning parameters, NNgHMC is used to sample 5000 draws with acceptance probability 0.57. Figure 2.2 compares standard HMC and NNgHMC draws, the true and approximated gradient fields, and two long simulated leapfrog trajectories using both. The neural network learns the distorted gradient field accurately and NNgHMC completely recovers the banana shape.

Next, we illustrate the proposed method on a **multivariate Gaussian distribution with**

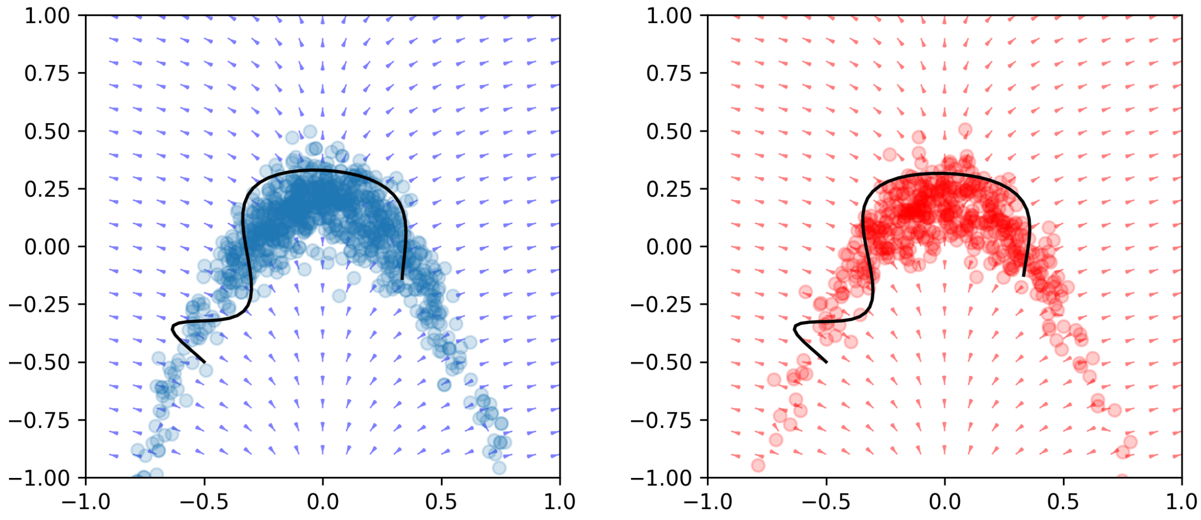


Figure 2.2: Gradient fields, samples, and leapfrog trajectories using standard HMC (blue) and NNgHMC (red) are indistinguishable.

ill-conditioned covariance. The distribution is given by $q \sim \mathcal{N}_{30}(0, \Sigma)$ where Σ is a diagonal matrix with smallest value 0.1, largest value 1000 and other values uniformly drawn between 1 and 100. As the distribution is on very disparate scales in different dimensions, HMC needs accurate gradient information to move accordingly. For HMC, the leapfrog step size s is set to be 0.5 and the number of steps l is set to be 100 so that acceptance probability is around 0.7. If the step size is too big, HMC would miss the high density region in the narrowest dimension. Without a sufficiently long trajectory, HMC would fail to explore the elongated tails in the widest dimension.

We collect sample gradients until 50 iterations after convergence to train the neural network. The neural network has $h = 100$ units in the hidden layer and is trained for $t = 10$ epochs. With the same tuning parameters as standard HMC, NNgHMC has acceptance probability around 0.5. Despite slightly lower acceptance probability, as shown in Figure 2.3, NNgHMC converges to the true posterior just as standard HMC. With more training data, the neural network will learn the gradient field more accurately and NNgHMC will have similar

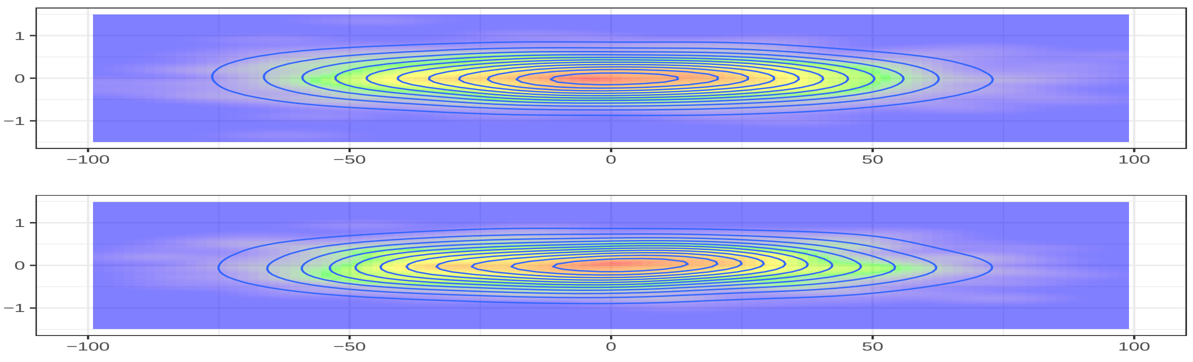


Figure 2.3: NNgHMC posterior (bottom) captures the highly elongated shape of the Gaussian distribution in the two most extreme dimensions ($\sigma_1^2 = 0.1, \sigma_{30}^2 = 1000$) as well as the HMC posterior (top). Note that the x- and y-axes are on very different scales.

acceptance probability as standard HMC.

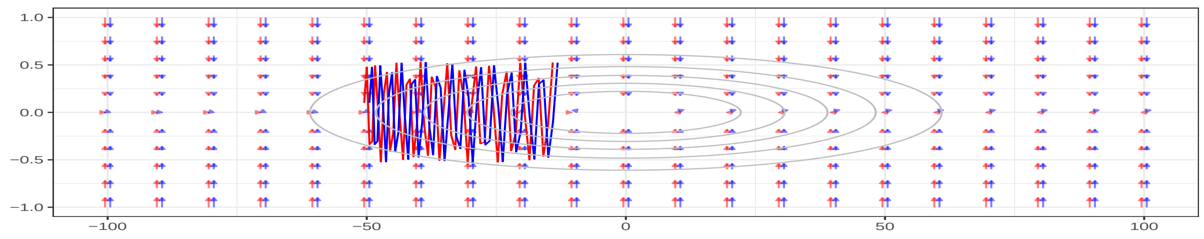


Figure 2.4: With the same initial position and momentum, the leapfrog trajectory in the same dimensions as in Figure 1 using approximated gradient (blue) faithfully resembles the one using true gradient (red) despite heavy oscillation on the energy level contour. The periodic nature of the Hamiltonian flow reflects the fact that the Hamiltonian is that of the harmonic oscillator, i.e. both potential and kinetic energies are quadratic. The vectors and trajectories are slightly jittered for plotting.

2.4.2 200-dimensional Bayesian logistic regression

Next we demonstrate the scalability of proposed method on logistic regression with simulated data. The X matrix has $50,000$ rows drawn from a 200 dimensional multivariate Gaussian distribution with mean zero and covariance I_{200}. The regression coefficients β are drawn independently from $Uniform(-1,1)$. Given X and β, the response vector is created with $Y_i \sim Bernoulli(logistic(X_i\beta))$. With $l = 20$ leapfrog steps and step size $s = 0.01$, HMC makes 1000 draws in 300 seconds with acceptance probability around 0.8. 4000 training points and gradients, which come from 200 draws after convergence, are used for neural network training.

With the same tuning parameters, NNgHMC can make 1000 draws in just 40 seconds with acceptance probability around 0.6. HMC yields 1.5 effective samples per second while NNgHMC yields 6.75 effective draws per second on average. The improvement on effective sample size and CPU time ratio is considerable and will only increase as the size of the data set increases.

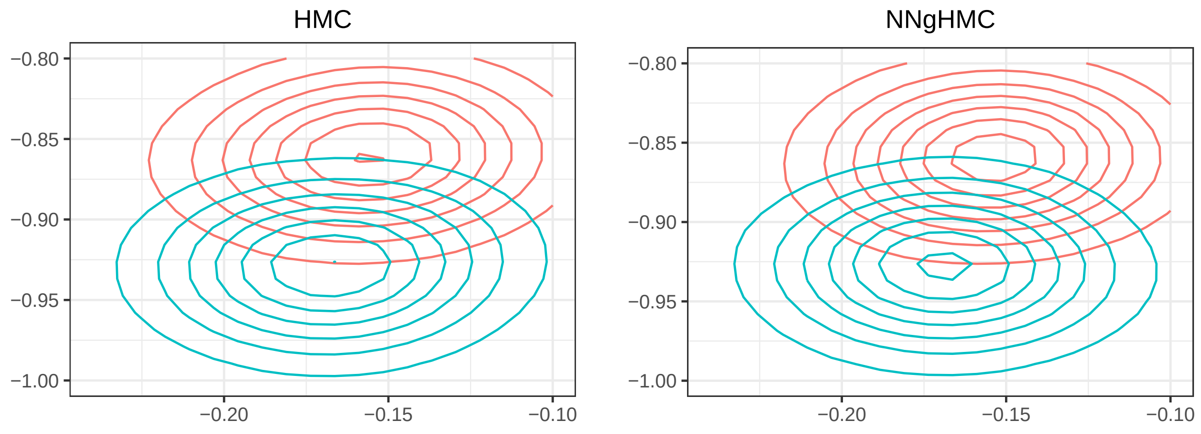


Figure 2.5: We first use HMC to collect training samples from the posterior of the 200-dimensional logistic regression model under a diffused prior with variance 10 for NNgHMC. The HMC (left) and NNgHMC (right) posteriors are colored in green. Then we use the same trained network for NNgHMC under a concentrated prior with variance 0.1. The new HMC and NNgHMC posteriors are colored in red. *Although most of the training data come from the green region, the neural network can extrapolate well to sample around the red region.*

The choice of prior plays an important role in Bayesian inference, and it is common to fit models with different priors for sensitivity analysis. The gradient of energy function ∇U is equal to the sum of the gradient of negative log-likelihood $-\nabla \log \pi(x|q)$ and the gradient of log prior $\nabla \log \pi(q)$. As the proposed method provides an accurate approximation of ∇U under prior $\pi(q)$, adding $\nabla \log \pi'(q) - \nabla \log \pi(q)$ will yield an approximation of ∇U under a new prior $\pi'(q)$. In this case, NNgHMC can sample from the new posterior much faster than HMC without additional training. Figure 2.5 compares the NNgHMC and HMC posteriors.

Remark 1. *While there are no fixed rules on the size of hidden layers, non-generative models typically have larger hidden layers than output layers. With input and output dimensions both being* 200*, a large hidden layer of size* 400 *would lead to* $160,000$ *total units, which is computationally expensive. Meanwhile, a network with a hidden layer of size* 200 *has half as many total units but is not nearly as expressive. Here we use eight disjoint hidden layers of size 50 to approximate 25 dimensional blocks of the gradient to cut down the number of total units to 90,000. Figure 2.6 compares the training losses of these three networks.*

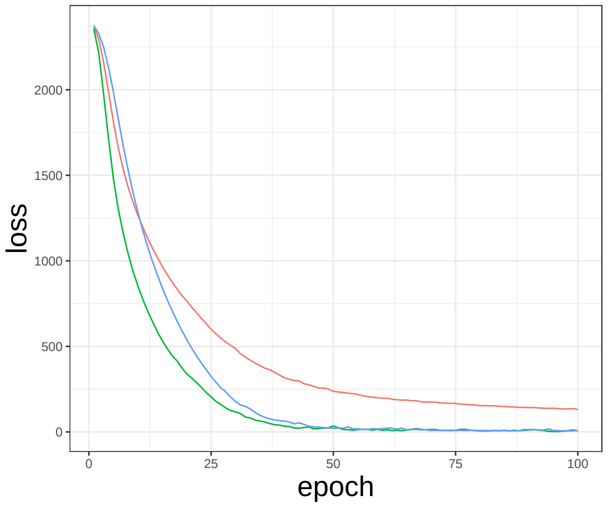


Figure 2.6: The gradient of the 200-dimensional logistic regression model is approximated by neural networks of different designs. In terms of performance measured by training L_2 loss on the true gradient, the block network (blue) matches the single large network (green) and outperforms the single small network (red) using comparable number of total units.

2.4.3 Low-dimensional models with expensive gradients

In this section, we evaluate our method using two models that involve costly gradient evaluations in spite of their typically low dimensions. First, we focus on the **generalized autoregressive conditional heteroskedasticity (GARCH)**, which is a common econometric model that models the variance at time t as a function of previous observations and variances. The general GARCH(m, r) model is given by

$$y_t \sim N(0, \sigma_t^2) \tag{2.24}$$

$$\sigma_t^2 = \alpha_0 + \sum_{j=1}^{m} \alpha_j y_{t-j}^2 + \sum_{j=1}^{r} \beta_j \sigma_{t-j}^2. \tag{2.25}$$

Conditioning on the first $\max(m, r)$ observations, the likelihood is the product of $N(0, \sigma_t^2)$ densities. The likelihood and gradient calculation for GARCH models can be slow as it has to be done iteratively and scales with the number of observations. As shown in figure 2.7, 1000 observations are generated with a $GARCH(2, 1)$ model and used as data for comparing standard HMC and NNgHMC. Truncated uninformative Gaussian priors are used because of GARCH stationarity constraints. 10000 draws are taken with standard HMC and gradient values collected between 1000 to 2000 iterations are used for training. Training a neural network with hidden layer size 50 takes 5s. With tuning parameters fixed at step size $s = 0.002$ and $l = 15$ leapfrog steps, standard HMC and NN gradient HMC both have close to 0.7 acceptance probability, but the latter is more computationally efficient (Table 2.1).

Table 2.1: Comparing standard HMC and NNgHMC using a GARCH model.

Method	AP	ESS	CPU time	Median ESS/s	Speed-up
Standard	0.72	(99, 261, 424)	436s	0.60	1
NNg	0.70	(116, 176, 303)	59s	2.98	**4.98**

AP: acceptance probability
ESS: effective sample size (min, median, max) after removing 10% burn-in

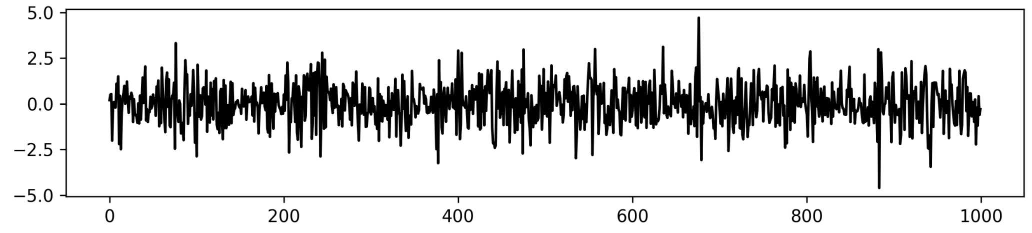


Figure 2.7: Time series data generated with a GARCH(2, 1) model.

Gaussian process is computationally expensive because the covariance matrix is $n \times n$ and inverting it requires $\mathcal{O}(n^3)$ computation. Here we consider a Gaussian process regression model with the Matérn kernel:

$$Y \sim \mathcal{N}(0, \mathcal{K}(X, X)) \tag{2.26}$$

$$\mathcal{K}(d) = \frac{2^{1-\nu}}{\Gamma(\nu)}(\sqrt{2\nu}\frac{d}{l})^{\nu} K_{\nu}(\sqrt{2\nu}\frac{d}{l}) \tag{2.27}$$

where d is the Euclidean distance between two observations x and x', length scale l and smoothness ν are the hyper-parameters. Γ denotes the gamma function and K_ν is the modified Bessel function of the second kind. Here ν is fixed at 1.5 to limit Gaussian process draws to be once differentiable functions. It is common to add white noise $\sigma^2 I$ to the covariance matrix for numerical stability. Therefore, the second free hyper-parameters besides l is σ^2. Diffused Lognormal priors are used for the hyperparameters. The 500×4 data matrix X is drawn from Multivariate Gaussian with mean zero and identity covariance. Y is obtained by mapping X with a polynomial pattern and adding noise.

10000 draws are sampled using standard HMC with leapfrog steps $l = 20$ and step size $s = 0.05$; the acceptance probability is 0.83 but it is very time consuming. Gradient collected during the first 1000 draws is then used to train a neural network with hidden layer size $h = 100$ for $t = 100$ epochs. Using the same tuning parameters, NNgHMC can sample 10000 draws in much shorter time with the same acceptance probability (Table 2.2). Figure 2.8 compares the standard HMC and NNgHMC posteriors; Figure 2.9 compares Gaussian process model posterior draws along one particular direction.

Table 2.2: Experiment results using Gaussian process regression model

Method	AP	ESS	CPU time	Median ESS/s	Speed-up
Standard	0.83	(5135, 5754, 7635)	1834s	3.14	1
NNg	0.84	(4606, 6172, 7741)	50s	123.4	**39.3**

AP: acceptance probability
ESS: effective sample size (min, median, max) after removing 10% burn-in

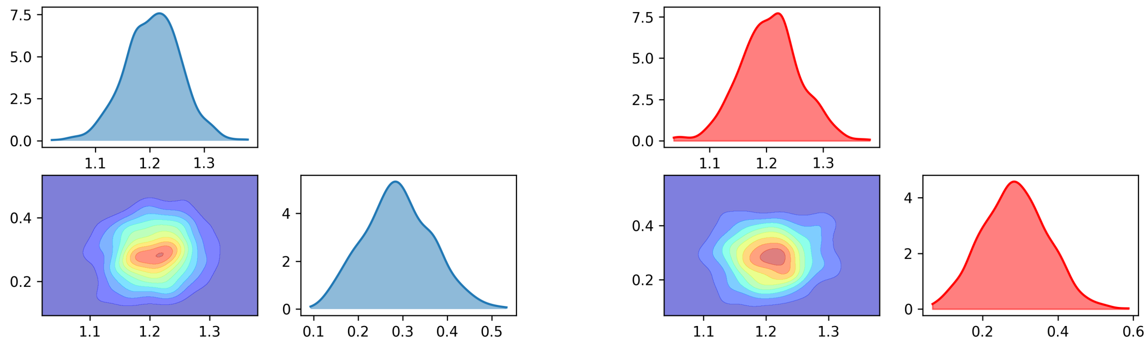


Figure 2.8: GP regression model posteriors of hyper-parameters using standard HMC (blue) and NNgHMC draws (red).

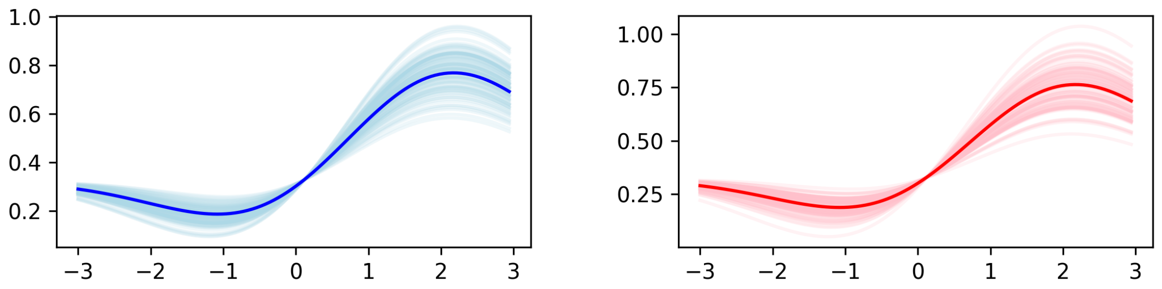


Figure 2.9: GP regression model predictions with standard HMC (blue) and NNgHMC posteriors (red).

2.4.4 Comparison with stochastic gradient HMC

Naïve stochastic gradient HMC using mini-batches of data is problematic as the noisy gradient can push the sampler away from the target region. Recent more advanced stochastic gradient method uses a friction term and is shown to sample from the true posterior asymptotically. The formulation of SGHMC is given by:

$$d\theta = M^{-1} r dt \tag{2.28}$$

$$dr = -\nabla U(\theta) dt - BM^{-1} r dt + N(0, 2Bdt) \tag{2.29}$$

where $N(0, 2Bdt)$ is the noise added to the gradient by subsampling. In practice, the friction term $BM^{-1}rdt$ is set arbitrarily.

To further improve speed, SGHMC does not perform Metropolis-Hastings correction and uses very small step sizes. The SGHMC posterior is dependent on the choice of step size; however, *a priori* one would not know the optimal step size. Here we want to show that while SGHMC provides fast approximation of the true posterior when data are abundant, the SGHMC posterior may not be suitable for inference.

In our experiment, the Cover Type data from UCI machine learning repository is used. We run standard HMC for 4000 iterations with $l = 50$ leapfrog steps and step size $s = 0.002$. We also run SGHMC for 4000 iterations with default parameters and varying step sizes from $s = 5e - 6$ to $s = 5e - 8$.

Figure 2.10 illustrates the main issue with SGHMC. For these two marginal distributions, the SGHMC posteriors have roughly the same location but completely different shapes. On the other hand, NNgHMC posteriors agree with the standard HMC posteriors almost exactly.

Another comparison with SGHMC is performed with Metropolis-Hastings correction. Here the sub-sampled data size is 5000 and the tuning parameters are $l = 10$ leapfrog steps and step size $s = 0.001$ so that the simulated trajectory is shorter for less gradient noise to compound. While SGHMC is faster still, the quality of samples is inferior compared to proposed method as indicated by lower ESS in Table 2.3 and less mixed trace plot in Figure 2.11. Overall, NNgHMC still outperforms SGHMC in terms of median EES per second.

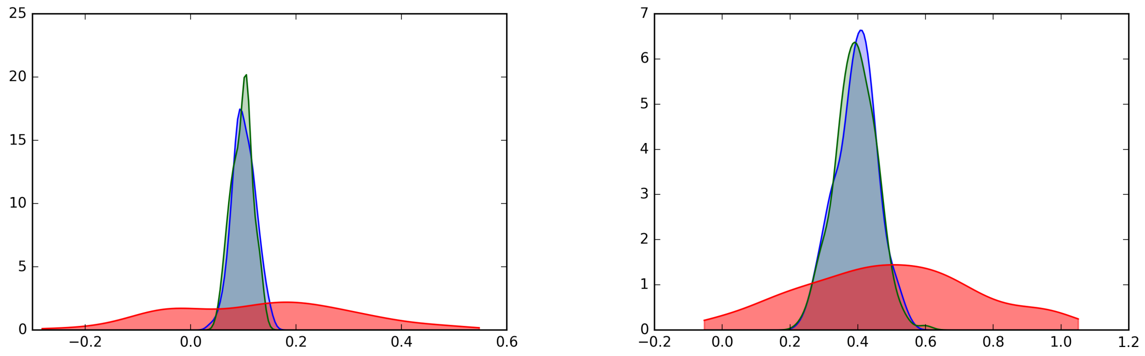


Figure 2.10: Histograms of marginal posteriors of logistic regression model coefficients with Laplace prior on Cover Type data. Blue: standard HMC; Red: stochastic gradient HMC; Green: neural network gradient HMC

Table 2.3: Experiment results on Cover Type data

Method	AP	ESS	CPU time	Median ESS/s	Speed-up
Standard	0.80	(73, 143, 10000)	3147s	0.05	1
NNg	0.67	(57, 186, 7174)	710s	0.26	**5.77**
SG	0.33	(49, 59, 246)	357s	0.17	3.64

AP: acceptance probability
ESS: effective sample size (min, median, max) after removing 10% burn-in

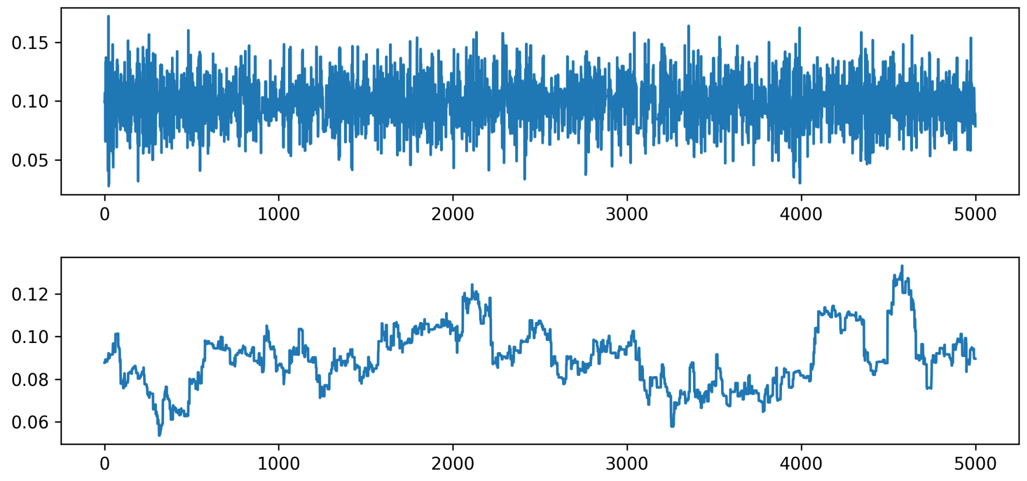


Figure 2.11: Trace plots of NNgHMC (top) and stochastic gradient HMC (below) show that the NNgHMC chain is more efficient as the approximated gradient is more accurate than sub-sampled gradient.

▩ 2.4.5 Comparison with Gaussian process surrogate

We now compare our method to the Gaussian process surrogate approach with the squared exponential kernel parametrized by $\mathcal{K}(x, x') = \exp -\frac{(x-x')^2}{2l^2}$. We also add white noise $\sigma^2 I$ to the covariance matrix. The squared exponential kernel, the default choice, is infinitely differentiable and gives rise to another Gaussian process as the derivative. Given observations X and Y, we can explicitly write down the mean of the derivative at x^*.

$$E\frac{\partial f}{\partial x^*} = \frac{\partial}{\partial x^*} Ef = \frac{\partial}{\partial x^*} \mathcal{K}(x^*, X)\mathcal{K}(X, X)^{-1}Y \tag{2.30}$$

Here we estimate both the length scale l in the squared exponential kernel and the white noise parameter σ jointly with maximum likelihood. The estimation requires inverting the observed covariance matrix, which is $\mathcal{O}(n^3)$ where n is the number of observations.

We compare with GP surrogate method on multivariate Gaussian distributions with covariance I_d where d varies from 10 to 40. For each Gaussian, we generate n training data points to train the GP surrogate and neural network. The neural network has 100 hidden units and is trained for 10 epochs. After training, both methods are used to draw 1000 samples.

Table 4 compares acceptance probability for the two methods. We can see that the neural network predicted gradient provides better approximation overall than the gradient of GP as indicated by higher acceptance probability. This advantage is more pronounced as dimensionality increases.

Table 2.4: Acceptance probability when sampling from multivariate Gaussian

Method	Dimension / Training	500	1000	2000
Gaussian process	10	0.65	0.61	0.57
	20	0.64	0.65	0.62
	40	0.31	0.32	0.32
Neural network gradient	10	0.95	0.96	0.97
	20	0.82	0.87	0.91
	40	0.61	0.75	0.87

2.4.6 Speed evaluation on real data

Similar to other surrogate methods, NNgHMC has three stages: training data collection, training, and sampling. We have demonstrated that using a neural network can provide accurate approximation of the gradient, however, the effectiveness of our method still needs to be evaluated by time. If the neural network requires too much training data, then it would not reduce computation time. Here we first run standard HMC to draw a desired number of samples (10000) and record time as benchmark. Then we collect different amounts of training data points (10%, 15% and 20% of total number) and use NNgHMC to draw remaining samples. The time to collect training data and train the neural network is included for NNgHMC. As shown in Table 5, 10% of training data is sufficient for the neural network to learn the gradient and gives the most speed-up. While adding more training data increases the quality of gradient approximation, the computation cost outweighs the benefit of higher acceptance probability.

Table 2.5: Experiment results on data sets from UCI machine learning repository

Method	AP	ESS	CPU time	Median ESS/s	Speed-up
Online News Popularity (39797×44)					
Standard	0.77	(777, 2021, 5929)	3607s	0.66	1
NNg (10%)	0.61	(605, 1416, 4865)	502s	2.82	**4.27**
NNg (15%)	0.64	(620, 1382, 5500)	678s	2.04	3.09
NNg (20%)	0.68	(700, 1731, 5397)	854s	2.03	3.08
Census Income (48842×123)					
Standard	0.84	(6306, 9390, 10000)	1796s	5.23	1
NNg (10%)	0.60	(4023, 6024, 7156)	564s	10.68	2.04
NNg (15%)	0.68	(4617, 7511, 9201)	656s	11.45	**2.19**
NNg (20%)	0.76	(5036, 7558, 8696)	740s	10.21	1.95
Dota2 Games Results (102944×116)					
Standard	0.75	(1677, 5519, 8621)	20760s	0.27	1
NNg (10%)	0.59	(1446, 4197, 6442)	2903s	1.45	**5.44**
NNg (15%)	0.70	(1901, 4865, 7600)	3911s	1.24	4.59
NNg (20%)	0.74	(2432, 5744, 8860)	4992s	1.15	4.26

AP: acceptance probability

ESS: effective sample size (min, median, max) after removing 10% burn-in

▢ 2.5 Discussion

Whereas HMC is helpful for computing large Bayesian models, its repeated gradient evaluations become overly costly for big data analysis. We have presented a method that circumvents the costly gradient evaluations, not by subsampling data batches but by learning an approximate gradient that is functionally free of the data. We find that multi-output, feedforward neural networks are ripe for this application: NNgHMC is able to handle models of comparatively large dimensionality.

The NNgHMC algorithm is an important paradigm shift away from the class of surrogate function approximate HMC algorithms, but this shift leaves many open questions. Much work is needed to extend NNgHMC to an on-line, adaptive methodology: what measures of approximation error will be useful criteria for ending the training regime of the algorithm, and are there benefits to iterating between training and sampling regimes? Are there any valid second-order extensions to the NNgHMC algorithm à la Riemannian HMC? Finally—and most interestingly—can the representational power of deep neural networks be leveraged for more accurate approximations to the Hamiltonian flow?

Chapter 3

Gaussian Process Accelerated Approach for Physical Parameter Inference

3.1 Introduction

Constructing classical confidence intervals for physical parameters with boundary conditions is challenging when dealing with small signals. The challenge is especially evident when studying neutrino oscillations because of the low event counts and multiple competing effects on the energy spectrum. The low event counts are primarily caused by the extremely low interaction cross-section of neutrinos, arising from the fact that they interact via the weak nuclear force. In order to extract meaningful statistical conclusions, one has to resort to means other than the asymptotic properties of Poisson data. The gold standard is the so-called unified approach outlined by Feldman and Cousins [30]. It builds upon the Neyman construction of classical confidence intervals by specifying an ordering principle based on likelihood ratios and is known for providing correct coverage.

The Feldman-Cousins approach is firmly grounded in statistical theory and widely used in neutrino experiments, for example Refs. [1][2][3]. However, it comes at a heavy computational cost, which in some cases such as Ref. [1] renders it infeasible for multi-dimensional confidence intervals. For the $1-\alpha$ confidence interval, the Feldman-Cousins approach includes all the values in the parameter space where the likelihood ratio test fails to reject at α level. However, it doesnt provide a prescription for how to sample that parameter space. Therefore, one is forced to sample it in its entirety in a grid-based fashion. Moreover, at each point one has to perform a large number of Monte Carlo simulations in order to calculate the p-value for the likelihood ratio test.

To accelerate the Feldman-Cousins approach, we propose approximating the function of p-values over the parameter space with Gaussian processes. Instead of performing a large number of Monte Carlo simulations, we start with just a small number of them at several parameter values to get noisy estimates of the p-values. We then train a Gaussian process model to interpolate over these estimates. Iteratively, we perform more Monte Carlo simulations to refine the Gaussian process approximation. We can control the p-value approximation error so that it does not change the likelihood ratio test decisions and the confidence interval. Meanwhile, the Monte Carlo simulations can be allocated intelligently in the parameter space to achieve substantial savings in computation.

The proposed algorithm is rooted in the framework of Bayesian optimization [57]. It was originally designed to find the extremal points of an objective function that is unknown *a priori*. In the Feldman-Cousins approach, the function of p-values over the parameter space is unknown. We adapt Bayesian optimization to locate a set of points in the parameter space that lie on the boundary of desired confidence intervals. By side-stepping points that are estimated to be either inside or outside the confidence interval with high probability, we can thus reduce the computational cost while producing the same result. We show that in the context of neutrino oscillation experiments, one can accelerate the construction of one-dimensional and two-dimensional confidence intervals by a factor of 5 and 10 respectively, without sacrificing the accuracy of the Feldman-Cousins approach.

3.2 Statistical Inference for Neutrino Oscillations

3.2.1 Neutrino Oscillations

Neutrino oscillations demonstrate that neutrinos have mass and that the neutrino mass eigenstates are different from their flavor eigenstates. In the three flavor framework, the transformation of the mass eigenstates (ν_1, ν_2, ν_3) into the flavor eigenstates (ν_e, ν_μ, ν_τ) is described by the 3×3 unitary matrix U_{PMNS} [66], which is parameterized by three mixing angles θ_{12}, θ_{23} and θ_{13}, and a CP violation phase δ_{CP}. The probability of oscillations between different neutrino flavor states of given energy E_ν over a propagation distance (baseline) L depends on the U_{PMNS} parameters and the difference of the squared masses of the eigenstates, Δm^2_{32} and Δm^2_{21}.

The mixing angles θ_{12} and θ_{13} along with the squared-mass splitting Δm^2_{12} have been measured to relatively high accuracy by several experiments, for example, Refs. [5][34][9]. One can then infer the remaining parameters, θ_{23}, δ_{CP}, and Δm^2_{32}, by measuring the probabilities $P(\nu_\mu \rightarrow \nu_\mu)$ and $P(\nu_\mu \rightarrow \nu_e)$. Of particular interest are: (1) the sign of Δm^2_{32}, positive indicating a "Normal Hierarchy" (NH) and negative indicating a "Inverted Hierarchy" (IH) of neutrino mass states; (2) whether $\delta_{CP} \neq 0, \pi$, indicating Charge-Parity (CP) violation in the lepton sector; (3) whether the mixing angle is in fact maximal, i.e $\theta_{23} = 45°$. The neutrino mass hierarchy has important implications for current and future neutrino experiments [28] involved in measuring the absolute neutrino mass and investigating the possible Majorana nature of the neutrino. Leptonic CP-violation could be important to deduce the origin of the predominance of matter in the universe.

To infer neutrino oscillation parameters θ, a typical long baseline neutrino oscillation experiment sends a beam of ν_μ neutrinos into a detector and observes a handful of oscillated ν_e neutrinos along with ν_μ neutrinos that survive over the baseline. As the oscillation probability is a function of neutrino energy, the observed neutrinos are binned by their energy. The neutrino oscillation parameters are inferred by comparing the observed neutrino energy spectra with the expected spectra for different oscillation parameters as shown in Fig. 3.1.

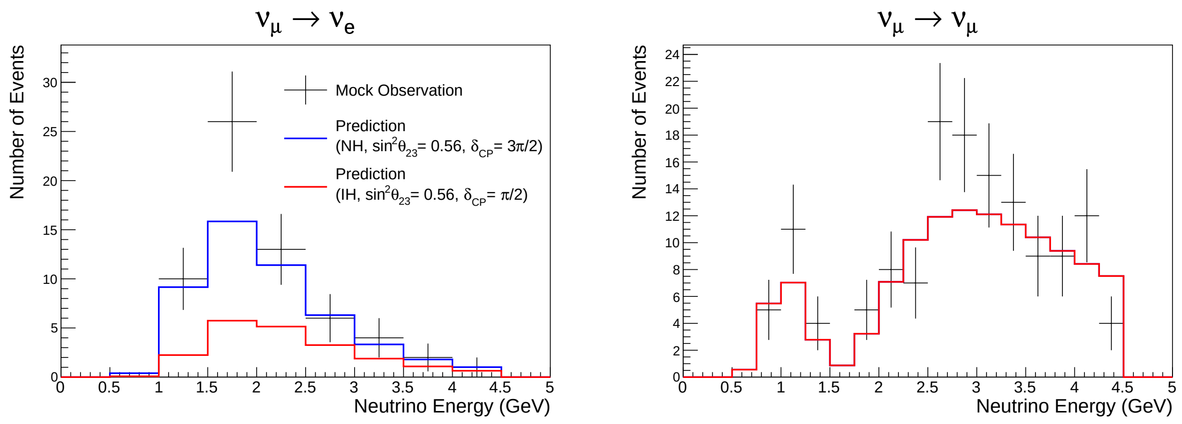


Figure 3.1: An illustration of a toy neutrino oscillation experiment setup with the $\nu_\mu \to \nu_e$ channel on the left and the $\nu_\mu \to \nu_\mu$ on the right. Expectations for different oscillation parameters are compared to mock observations in order to find maximum likelihood estimates. The likelihood of observed data is maximized using the extended likelihood function. The fit is performed in both channels simultaneously.

3.2.2 Feldman-Cousins Approach

Denote the random variable for the neutrino count in the i-th energy bin by X_i. Further, assume that each X_i follows an independent Poisson distribution with mean λ_i. For a given θ, the expectations $\vec{\lambda}$ are also influenced by systematic uncertainties in the beam configuration and the interaction model among others, which we parameterize by δ. For given oscillation and nuisance parameters (θ, δ), the expectations $\vec{\lambda}$ given (θ, δ) are obtained through simulations as they are analytically intractable. Denote the implicit mapping between $\vec{\lambda}$ and (θ, δ) by v. The extended log-likelihood of (θ, δ) is given by:

$$\log L(\theta, \delta) = \sum_{i \in I} \log Pois(x_i; v(\theta, \delta)_i) + \log Pois(\sum_{i \in I} x_i; \sum_{i \in I} v(\theta, \delta)_i) - \frac{1}{2}\delta^2$$

where $-\frac{1}{2}\delta^2$ is a penalty term for systematic error [14].

For a unified treatment of constructing classical confidence intervals for both null and non-null observations, an ordering principle based on likelihood ratios was introduced by Feldman and Cousins in 1997. The unified approach provides correct coverage even at parameter boundaries and has the highest statistical power as a result of the Neyman-Pearson lemma. In essence, a particular parameter value θ_0 is included in the $1-\alpha$ confidence interval if the likelihood ratio test fails to reject the null hypothesis $\theta = \theta_0$ at the α level. The likelihood ratio test statistic is given by

$$-2\log \frac{L(\theta_0)}{\arg\max_\theta L(\theta)}$$

and has an asymptotic χ^2 distribution by Wilks' theorem.

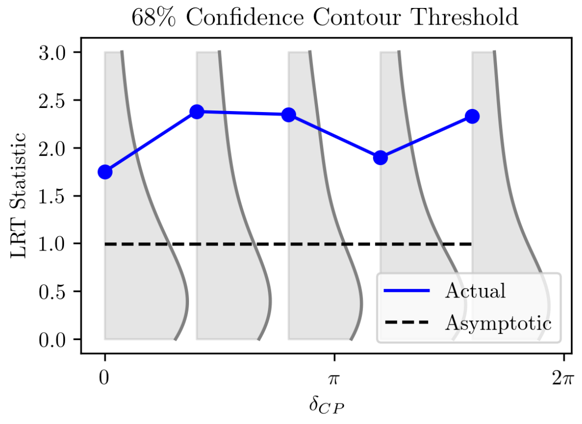


Figure 3.2: In the context of neutrino oscillations, the likelihood ratio test statistic distribution changes in the parameter space. Here the parameter is δ_{CP} and ranges from 0 to 2π. The solid blue line indicates the 68th-percentile of Monte Carlo simulated distributions while the dashed black line is the 68th-percentile of the asymptotic χ^2_1 distribution.

In the context of neutrino oscillations, the asymptotic distribution is unreliable because of the small sample size in neutrino data and physical boundaries on the oscillation parameters. The reference distribution of the likelihood ratio test statistic can vary drastically as a function of θ; Fig. 3.2 shows several distributions at different θ values and comparisons of their critical values in particular. Therefore, for any given θ, Monte Carlo experiments are used to simulate the reference distribution and calculate the p-value for the likelihood ratio test. Since the parameter space is bounded, the simulations are performed on a grid for a large number of θ values and the computational cost adds up quickly.

Another motivation for this work is sequential experimental design [?] as the proposed method allocates various amounts of Monte Carlo computation across points in the parameter space. Hence, the Gaussian process model has different levels of variance. But different from most studies in statistics literature, this work does not have an explicit parametric model. The most similar work is using a Gaussian Process surrogate in the approximate Bayesian computation framework [?] and inside an MCMC algorithm [47]. The similarity is that both of these methods and the proposed method use Gaussian process approximation in the procedure of statistical inference. However, there are two major distinctions. First, the Gaussian process surrogate in [?] approximates the likelihood function in the "likelihood-free" scenario [?] whereas we have an explicit likelihood function to evaluate. Second, these methods return an approximate posterior distribution but the Bayesian credible intervals may not achieve desired coverage probability [?]. In contrast, the proposed method only targets confidence intervals at certain levels and they are "exact."

3.3 Gaussian Process Algorithm

Different kernels can be combined to compose a $\mathcal{GP}$ as long as the new kernel covariance matrix is still positive semi-definite. With the squared exponential kernel alone, the covariance implies that the observed data has no error. To account for error in the data, a diagonal matrix $\sigma^2 I$ is usually added to model constant variance across observations. In many situations such as ours, there exists heteroskedasticity, which means that different observations have different errors. When we iteratively perform Monte Carlo simulations to calculate p-values, the errors in the estimates also vary based on the number of simulations. We can actually model the p-value error as a diagonal matrix and add it to the $\mathcal{GP}$ covariance.

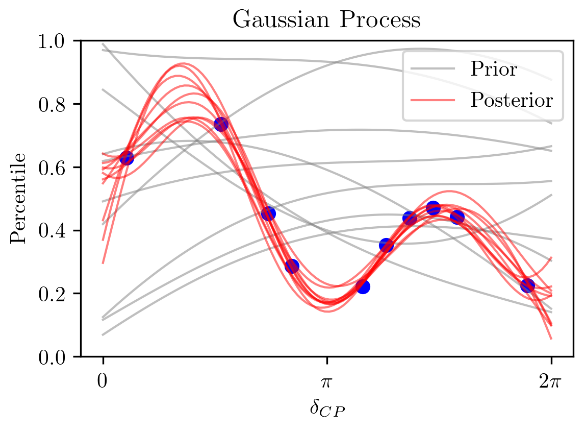


Figure 3.3: Sampled paths from Gaussian process prior and posterior with squared exponential kernel (right). The posterior paths, representing the curves drawn from the predictive distribution, are better aligned with the observed data points in solid blue.

3.3.1 Monte Carlo Error Estimation

In the Feldman-Cousins approach, a large number of Monte Carlo simulations is required in order to make the error in p-value calculation negligible. When the Monte Carlo error in p-value calculation is not negligible, we should try to quantify it. Since the p-value is the quantile of the observed likelihood ratio statistic under the reference distribution, we can use a binomial proportion confidence interval as the p-value error estimate as outlined below [38]. As shown in Fig. 3.4, the Monte Carlo error only slowly approaches zero when the number of simulations increases to 10,000.

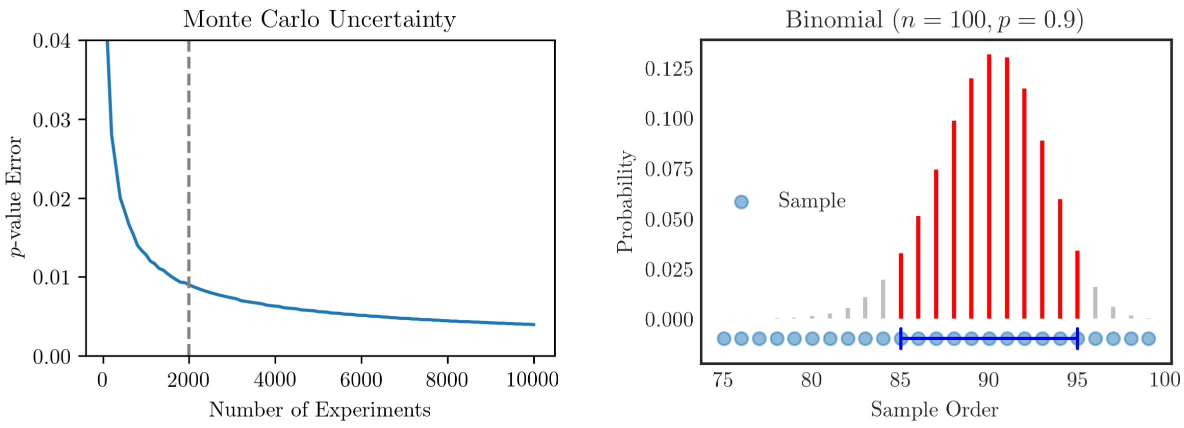


Figure 3.4: Monte Carlo error in terms of p-value as a function of the number of experiments (left). Example of non-parametric quantile interval construction using Binomial distribution (right). In a sample with 100 draws, the 85th and 95th order statistics form a 95% confidence interval for the 90th quantile of the unknown distribution.

Suppose $X_1, ..., X_n$ are independent draws from an unknown distribution F whose q^{th} quantile is denoted by $F^{-1}(q)$. Each draw X_i is either below or above $F^{-1}(q)$ with probability q. Consequently, M, the number of X_i's less than or equal to $F^{-1}(q)$, has a Binomial(n, q) distribution. We can obtain a confidence interval for $F^{-1}(q)$ with sample statistics $X_{(l)}, X_{(u)}$ (the l^{th} and u^{th} ordered draws) with $1 \leq l \leq u \leq n$ such that

$$B(u-1; n, q) - B(l-1; n, q) \geq 1 - \alpha.$$

$B(u-1; n, q) - B(l-1; n, q)$ is the probability that M is between l and $u-1$. Thus, (l, u) would form a confidence interval for M. Correspondingly, $(X_{(l)}, X_{(u)})$ would form a confidence interval for $F^{-1}(q)$. Our goal, however, is to estimate $F(x^*)$ for an arbitrary x^* given sample X, where, in our context, x^* is the observed likelihood ratio test statistic and $F(x^*)$ is the p-value. This can be done by inverting the quantile confidence interval until the confidence intervals for $F^{-1}(q_l)$ and $F^{-1}(q_u)$ no longer contain x^*. Then (q_l, q_u) would form a confidence interval for $F(x^*)$.

■ 3.3.2 Bayesian Optimization

Bayesian optimization can be used to find the extremum of a black-box function f when f is expensive to evaluate so that a grid search is too computationally intensive. Bayesian optimization is an iterative procedure; in each iteration, f is evaluated at a number of points to update an approximation of f. The approximation usually starts from a zero-mean Gaussian process prior $\mathcal{GP}(0, \kappa(\cdot, \cdot))$. After each iteration, the $\mathcal{GP}$ model yields a posterior distribution, hence Bayesian. Based on the approximation posterior, the points in the next iteration are proposed by an acquisition function a. The acquisition function a aims to balance between "exploration", reducing approximation uncertainty, and "exploitation", reaching the extremum.

Denote the $\mathcal{GP}$ kernel parameters by ω, mean function by μ, and standard deviation by σ. The typical acquisition functions include probability of improvement

$$a(x) = \frac{f(x_{best}) - \mu(x; \omega|X)}{\sigma(x; \omega|X)},$$

expected improvement, which is integrated over the $\mathcal{GP}$ posterior, and upper confidence bound

$$a(x) = \mu(x; \omega|X) - k\sigma(x; \omega|X) \text{ where } k \text{ is a constant.}$$

Take probability of improvement for example, the numerator favors points where the $\mathcal{GP}$ predictive means are greater than the current best. But if two points have the same predictive mean that is less than the current best, the numerator will be negative and the denominator will favor the point with higher variance. Compared to probability of improvement, expected improvement and upper confidence bound behave similarly as shown in Fig. 3.5 but no acquisition function is known to always outperform others [76].

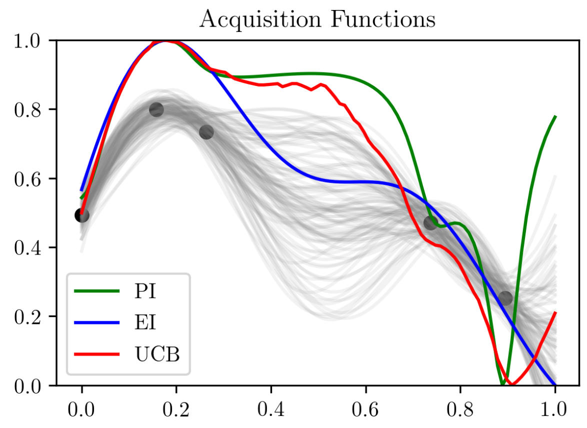


Figure 3.5: Illustration of different acquisition functions: probability of improvement (PI), expected improvement (EI), and upper confidence bound (UCB). The solid black dots represent observed data points and the transparent gray curves are drawn from the $\mathcal{GP}$ posterior.

Different from typical Bayesian optimization, where f is evaluated once at each point, it takes many Monte Carlo simulations to calculate the p-value in the Feldman-Cousins approach. It is therefore analogous to experimental design where there are repeated measurements for each design point. First proposed by George Box in [**?**], methods to place these design points iteratively are called sequential experimental design and have been studied extensively. However, the studies tend to focus on parametric models and the objective is usually very different from ours.

3.3.3 Proposed Algorithm

Bayesian optimization can be used to find the extremum of a black-box function h when h is expensive to evaluate so that a grid search is too computationally intensive. Bayesian optimization is an iterative procedure; in each iteration, h is evaluated at a number of points to update an approximation of h. The approximation usually starts from a zero-mean Gaussian process prior $\mathcal{GP}(0, \kappa(\cdot, \cdot))$. After each iteration, the $\mathcal{GP}$ model yields a posterior distribution, hence Bayesian. Based on the approximation posterior, the points in the next iteration are proposed by an acquisition function a. The acquisition function a aims to balance between "exploration", reducing approximation uncertainty, and "exploitation", reaching the extremum.

In our context, the expensive black-box function is the the function of p-values over the parameter space. Denote the grid points in the parameter space, where Monte Carlo simulations are performed, by $\vec{\theta}_o$, the simulated p-values at these points by $y(\vec{\theta}_o)$, and the independent simulation errors by $\sigma(\vec{\theta}_o)$. The $\mathcal{GP}$ predictive posterior distribution of the unobserved p-values at $\vec{\theta}_u$ conditional on obtained p-values $y(\vec{\theta}_o)$ at points $\vec{\theta}_o$ is then given by

$$\begin{aligned} f(\vec{\theta}_u)|y(\vec{\theta}_o) \sim \mathcal{N}(&K_{uo}(K_{oo} + diag(\sigma^2(\vec{\theta}_o)))^{-1}y(\vec{\theta}_o), \\ &K_{uu} - K_{uo}(K_{oo} + \sigma^2(\vec{\theta}_o)I)^{-1}K_{ou}) \end{aligned}$$

where $K_{oo}, K_{ou}, K_{uo}, K_{uu}$ denote the covariance matrices between points $\vec{\theta}_o$ and $\vec{\theta}_u$.

Different from typical Bayesian optimization, we do not simply wish to find the minimum or maximum p-value. Instead, we want to find the points where the p-value is equal to α so that they enclose the confidence interval. Moreover, we want to be able to find multiple intervals at different confidence levels. Therefore, we choose our acquisition function to be

$$a(\theta) = \sum_{\alpha_i} |\frac{f(\theta) - \alpha_i}{\sigma_{f(\theta)}}|^{-1}$$

where $f(\theta)$ is the $\mathcal{GP}$ approximated p-value (posterior mean) at θ and $\sigma_{f(\theta)}$ is the $\mathcal{GP}$ posterior standard deviation at θ.

Algorithm 3.1 $\mathcal{GP}$ iterative confidence interval construction

```
for each iteration $t = 1, 2, ...$ do
    Propose points in parameter space $\arg\max_\theta a(\theta)$
    for each point $\theta'$ do
        Simulate likelihood ratio statistic distribution
        for $k = 1, 2, ...$ do
            Perform a pseudo experiment
            Maximize the likelihood with respect to $(\theta, \delta)$
            Maximize the likelihood with constraint $\theta = \theta'$
            Calculate likelihood ratio statistic
        end for
        Calculate $p$-value based on the simulated distribution
    end for
    Train $\mathcal{GP}$ approximation $f(\theta)$ for the $p$-values
    Update confidence intervals
end for
```

Iteratively, the $\mathcal{GP}$ algorithm will seek points on the boundary of confidence intervals, for which it is unsure about. Points far from the boundary, which have p-values much greater or less than α_i, are probabilistically "ruled out." At these points, we will end up performing fewer Monte Carlo experiments or skipping them altogether. Every point on the grid would be either included or rejected with some uncertainty based on the $\mathcal{GP}$ posterior. With more iterations, the uncertainty will diminish so that the approximated confidence intervals converges to the ones produced by a full grid search. Fig. 3.6 illustrates the proposed algorithm on an 1-dimensional example.

Here we use the squared exponential kernel with the Monte Carlo errors added to the co-

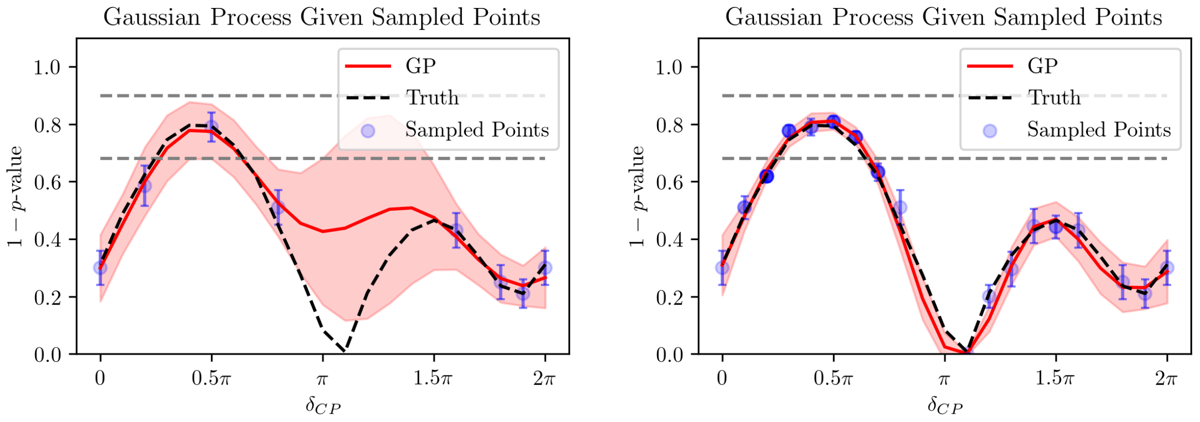


Figure 3.6: An illustration of our construction for the 68% and 90% confidence intervals for δ_{CP}, which consist of points lying underneath the dashed horizontal lines. From a few initial points with high variance, the $\mathcal{GP}$ learns a rough approximation of the true curve (left). Based on the approximation, more points are proposed around the interval boundary, shown in dark blue, and the $\mathcal{GP}$ improves itself (right). The shade of blue represents the number of simulations used to calculate the p-value and the error bars are for the p-value.

variance diagonal and a small amount of white noise as often done in a regression setting [69]. Point estimates of the $\mathcal{GP}$ kernel parameters by optimizing the log marginal likelihood

$$-\frac{1}{2}y(\vec{\theta}_o)^T(K_{oo}+diag(\sigma^2(\vec{\theta}_o)))^{-1}y(\vec{\theta}_o)-\frac{1}{2}\log|K_{oo}+diag(\sigma^2(\vec{\theta}_o))|-\frac{n}{2}\log 2\pi.$$

There are constraints on the kernel parameters that should be incorporated. For instance, the length scale l should be greater than the grid resolution and less than the grid range.

3.4 Numerical Studies

By way of illustration, we set up a toy long-baseline neutrino oscillation experiment in order to construct confidence intervals for the oscillation parameters. A flux distribution of ν_μs is modeled as a Landau function over neutrino energies, $E_\nu \in (0.5, 4.5)$ GeV with the location parameter at 2 GeV as shown in Fig 3.7. The normalisation uncertainty is taken to be 10% and is applied as a nuisance parameter. The ν_μ distribution is then oscillated into ν_es using the PMNS model for a toy baseline of 810km through the Earth. Corrections from matter interactions [74] are applied assuming a constant matter density of 2.84 g/cm^3. The setup is similar to NOvA [2], an accelerator-based long-baseline experiment at Fermilab. The oscillated ν_es are then "observed" with a toy interaction cross-section distribution, similar in shape to Ref. [32]; the cross-section increases as a function of neutrino energy from 0 GeV up to 1 GeV and decreases slowly until a maximum neutrino energy of 4.5 GeV as shown in Fig 3.7. A 10% normalisation uncertainty is applied on the cross-section as another nuisance parameter. Finally, we scale up the ν_e distribution to get an energy spectrum expectation, in energy bins of 0.5 GeV between the flux range, similar to observations from NOvA [2]. The expected spectrum is computed from scratch for each set of oscillation and nuisance parameters in the toy experiment as shown in Fig. 3.1. A similar setup is used for the $\nu_\mu \to \nu_\mu$ channel. However, in order to expedite the computation, the 2-flavor oscillation probability approximation is used. The reactor mixing angle, θ_{13} and the solar parameters, θ_{12} and Δm^2_{12} are fixed at the values given in Ref. [37]. A mock data set is obtained by applying Poisson variations on the expected spectrum at oscillation parameter values given by NOvA.

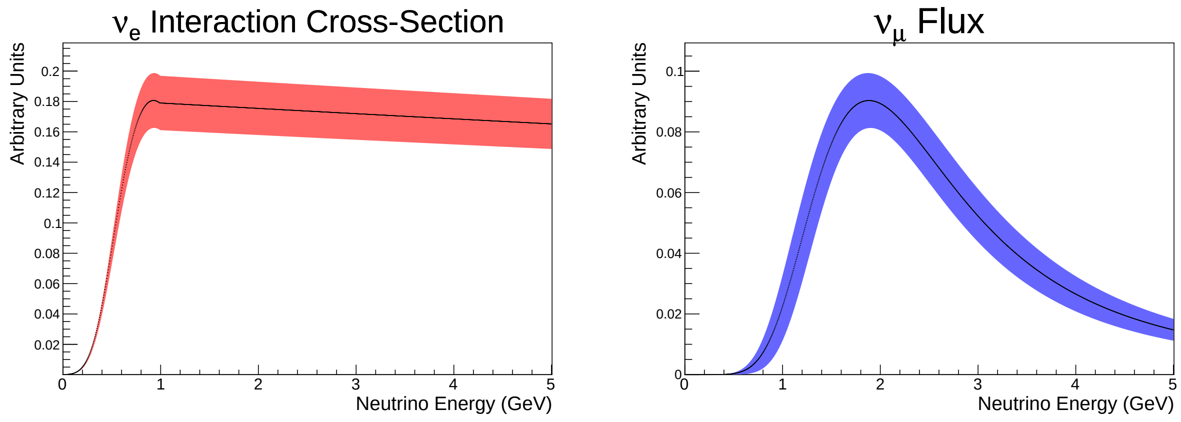


Figure 3.7: The distributions for ν_e interaction cross-section (left) and ν_μ flux (right) are shown along with a normalization systematic error of 10%.

We then use this setup to construct 1-dimensional confidence intervals for δ_{CP} and 2-

dimensional confidence intervals for $\sin^2\theta_{23}$ vs δ_{CP} by the two algorithms, a standard grid-search implementation of Feldman-Cousins and the $\mathcal{GP}$-based algorithm. $|\Delta m^2_{32}|$ is treated as a nuisance parameter while $\sin^2\theta_{23}$ is treated as another in the case of the 1-dimensional interval for δ_{CP}. The likelihood function is integrated over the nuisance parameters assuming a flat prior in the range $(2, 3)$ $\times 10^{-3}$ eV^2 for $|\Delta m^2_{32}|$ and $(0.3, 0.7)$ for $\sin^2\theta_{23}$, similar to Ref. [1]. The prior on the nuisance parameters for the systematic uncertainties in the toy model is assumed to be a standard normal distribution. The toy model and parameter fitting routine are implemented in ROOT [21] while the Gaussian process algorithm is implemented with scikit-learn [65].

3.4.1 1-dimensional Confidence Intervals

To make inference on δ_{CP}, a significance curve is usually drawn under different assumptions of mass hierarchy as shown in Fig. 3.8. The portion of the significance curve below a certain value gives us the confidence interval at that level. We can observe that the NH curves by both the standard FC and $\mathcal{GP}$ algorithms have the same intersections with 1σ horizontal line, which implies that the 1σ confidence intervals are the same. Though there are slight discrepancies, the shape of the $\mathcal{GP}$ significance curve is mostly correct.

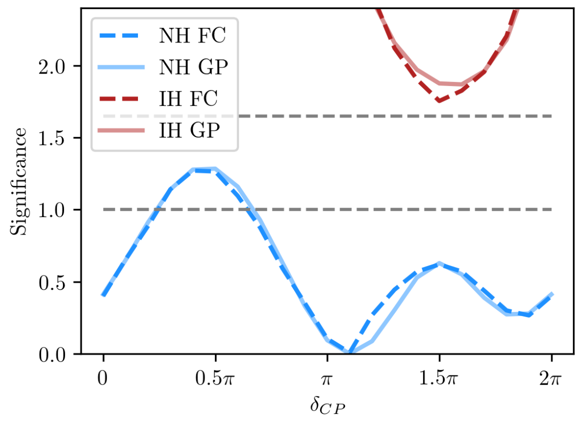


Figure 3.8: Example significance curves obtained with the standard Feldman-Cousins and Gaussian process algorithms mostly overlap, especially when the significance is close to 1σ and 1.6σ as desired. In this case, the inverted hierarchy (IH) is rejected at 1.6σ level and the normal hierarchy (NH) has the same 1σ confidence interval.

To evaluate the performance of the $\mathcal{GP}$ algorithm, we perform the same inference procedure on 200 different data sets to find the 68% and 90% confidence intervals. First, with standard FC results as ground truth, we consider the accuracy of the $\mathcal{GP}$ algorithm for classifying whether or not each grid point is included in the confidence intervals. As the $\mathcal{GP}$ algorithm is iterative, we can calculate the accuracy at the end of each iteration with fixed computation. When the computation reaches 20% of that is required by standard FC, we stop the algorithm and calculate the absolute error as the difference in confidence interval endpoints. Fig 3.9 shows that the median accuracy reaches 1 with less than 20% of computation and the error is no more than 0.1π for most data sets. As δ_{CP} ranges from 0 to 2π and there are only 20 grid points, an error of 0.1π is just one grid point. With a finer grid, we expect the performance of the $\mathcal{GP}$ algorithm to improve.

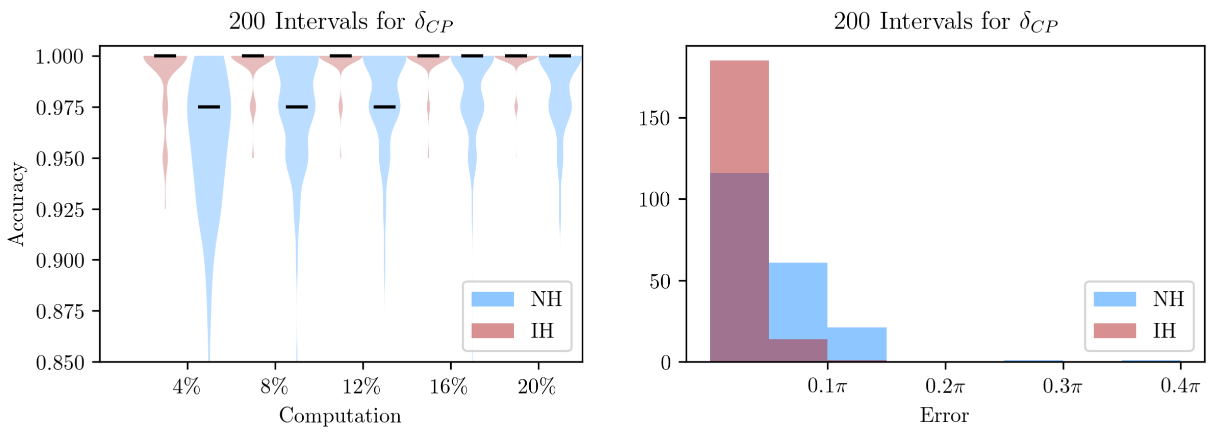


Figure 3.9: Relative accuracy of the confidence intervals in terms of correctly included grid points as a function of computation (left) and the distribution of absolute errors for both normal and inverted hierarchies (right).

3.4.2 2-dimensional Confidence Contours

To find the 2-dimensional confidence contours under hierarchy constraints, the $\mathcal{GP}$ algorithm approximates the p-value surface on the parameter grid as shown in Fig. 3.10 and specifically prioritizes points on the contour boundaries. Grid points below a certain value are included in the confidence contour at that level. To make the final smooth contours in Fig. 3.11, we use Fourier smoothing to draw the closest elliptical curves. We can observe that the FC and $\mathcal{GP}$ contours overlap in the same areas. In fact, the area difference between the contours is on the same order of magnitude with Fourier smoothing.

Similarly, we use both algorithms on 200 different data sets to find the 68% and 90% confidence contours and calculate the grid point classification accuracy after each iteration up to 10% of the standard FC computation. A concern is that contours with larger area could require more computation to achieve the same accuracy as there are more points along the boundary. We address this concern by stratifying contours by area quartile and plotting median accuracy as a function of computation. Fig. 3.12 shows that the median accuracy reaches 1 with less than 10% of computation and contour area does not have an effect. The

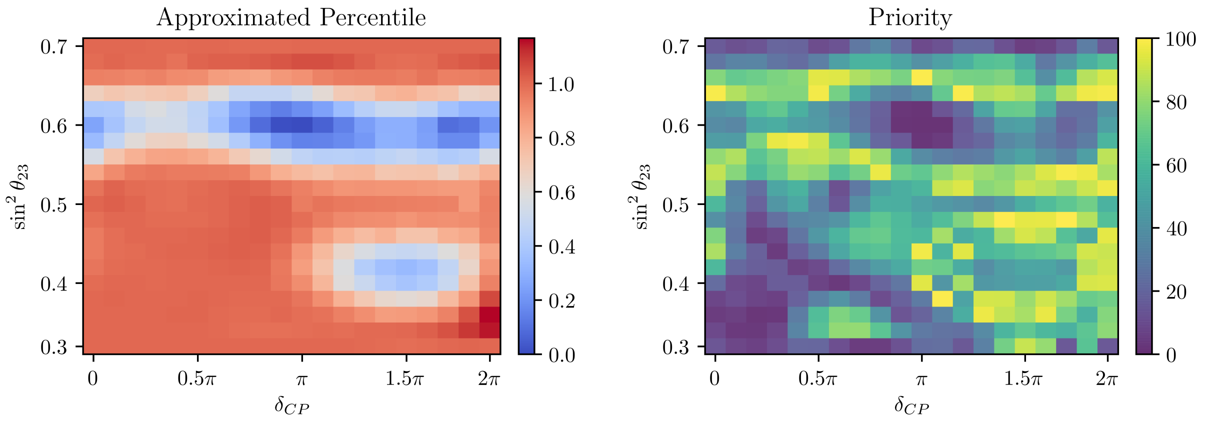


Figure 3.10: $\mathcal{GP}$ approximated percentile ($1-p$-value) on the 20×20 grid for $\sin^2\theta_{23}$ vs δ_{CP} (left) and the priority to sample points from the grid (right). Notice that the points near 68% and 90% have the highest priority.

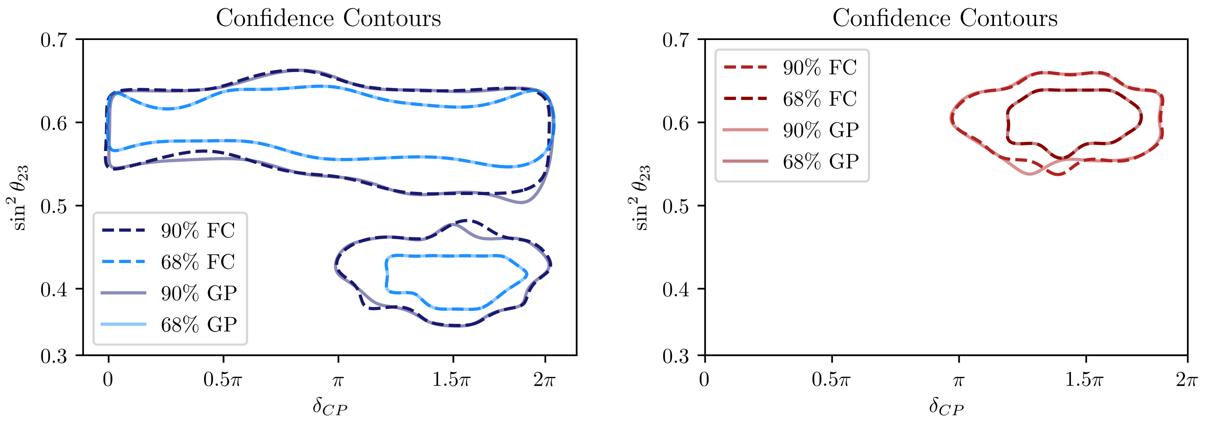
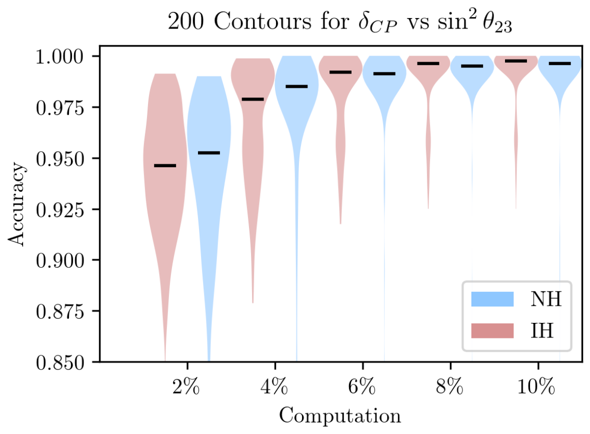


Figure 3.11: Confidence contours for the same data constrained to normal (left) and inverted hierarchies (right). The true (dashed) and approximated (transparent) contours are almost indistinguishable.

reason is that while larger contours have more points on the boundary, smaller contours are more difficult to locate precisely. Overall, it takes roughly the same computation to probe the p-value surface accurately so the $\mathcal{GP}$ algorithm should have similar performance regardless of the contour size.

Lastly, we are interested in where the computational savings come from. We keep track of the number of grid points explored by the $\mathcal{GP}$ algorithm and the number of simulations at



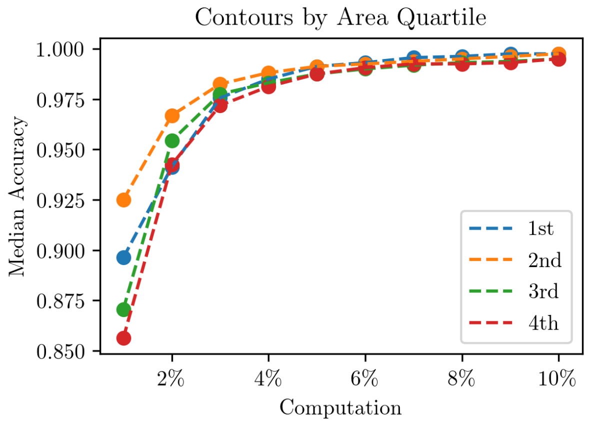
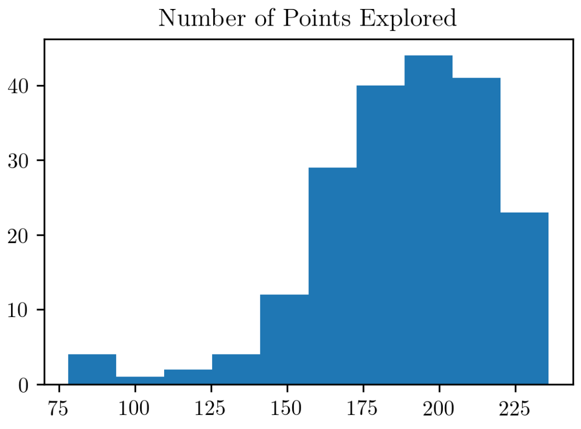


Figure 3.12: Relative accuracy of the confidence contours as a function of computation (left) and median accuracy stratified by area as a function of computation (right).

each point for the 200 data sets. Fig. 3.13 shows that the algorithm explores about half of the total grid points and on average only about 300 Monte Carlo simulations are done instead of 2000 in standard FC. We conclude that most of the computational savings come from performing fewer Monte Carlo simulations; skipping grid points nearly doubles the computational savings. As mentioned earlier, the advantage of the $\mathcal{GP}$ algorithm could be greater on a finer grid.



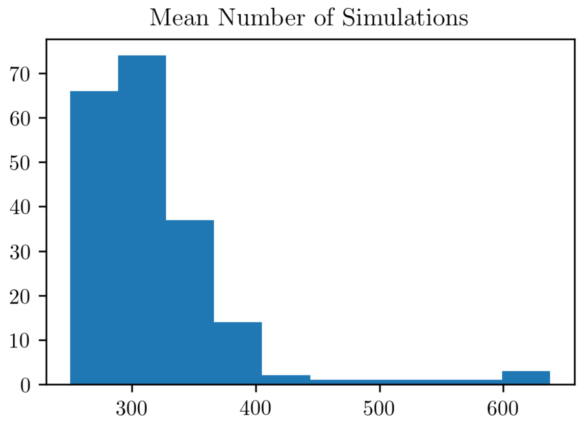


Figure 3.13: Distribution of the number of points explored on the grid (left) and distribution of the average number of Monte Carlo experiments simulated at a point (right).

▢ 3.5 Discussion

The proposed algorithm significantly accelerates the Feldman-Cousins approach wherein experiments have to devote enormous computational resources in order to estimate uncertainties in neutrino oscillation parameters [77]. This could also prove useful in estimating confidence intervals from a combined fit of neutrino oscillation results from different experiments when the respective likelihood functions are available. While we design the $\mathcal{GP}$ based construction in the neutrino oscillation context, the $\mathcal{GP}$ approximation does not have a particular parametric form. The same idea can therefore be applied to many other scenarios where the confidence interval construction for a continuous parameter over a bounded region normally proceeds via the unified approach.

Chapter 4

Sequential Memory Replay Analysis with Neural Networks

4.1 Introduction

The hippocampus is critical to the temporal organization of our experiences, including the ability to remember past event sequences and predict future ones. Although this fundamental capacity is conserved across modalities and species, its underlying neuronal mechanisms remain poorly understood. Here we recorded hippocampal ensemble activity as rats remembered a sequence of nonspatial events (five odor presentations unfolding over several seconds), using a task with established parallels in humans. We then used novel statistical methods and deep learning techniques to identify new forms of sequential organization in hippocampal activity linked with task performance. More specifically, we used a latent representation learning approach to quantify the differentiation of each type of task-critical information (stimulus, temporal order, and trial outcome information), and a neural decoding approach to quantify the decoding probability of each stimulus in the sequence. We discovered that sequential firing fields ("time cells) provided temporal information within and across events in the sequence, and that distinct types of task-critical information (stimulus identity, temporal order, and trial outcome) were also sequentially differentiated within event presentations. Finally, as previously only observed with spatial information, we report that the representations of past, present and future events were sequentially activated within individual event presentations.

◻ 4.2 Data description and modeling approach

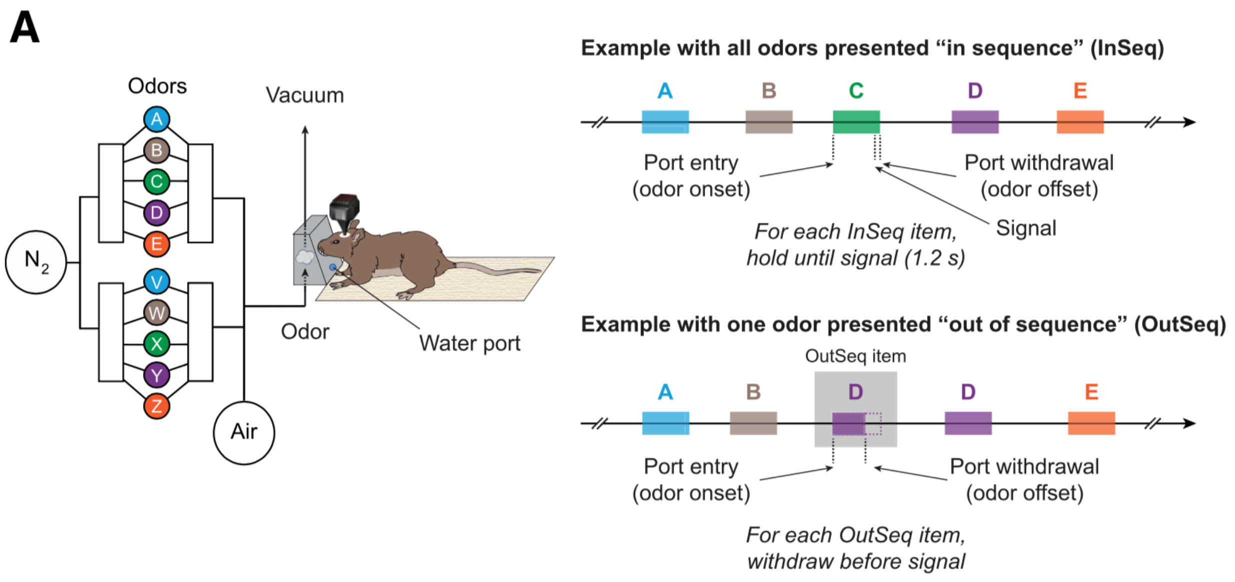


Figure 4.1: The memory task for the rat involves smelling a sequence of odors from A to E and the rat has to decide whether the odors are presented in the right order. When the odors are in the right order, the rat should keep its nose in the port. Otherwise, the rat should withdraw early.

In the experiment, rats are trained to memorize a sequence of odors from A to E as illustrated in 4.1. The presentation of one odor is considered a trial. Each rat is prompted to smell the odors in many trials and has to decide whether the odors are in the correct order at the end of each trial. There are five rats in total; for each rat, a number of tetrodes are surgically implanted in the hippocampus CA-1 region. The number of tetrodes varies across rats and the exact locations are also different. Therefore, the data are not exchangeable among the rats. From each tetrode, spiking activities of multiple neurons (spike train) and local field potentials (LFP) are collected every 1ms. For each trial, spike train and LFP data are available from -2s to 2s relative to odor presentation. In total, there are 700 trials combined from all five rats, where the rat makes the correct decision. The number of neurons ranges from 40 to 100 for different rats.

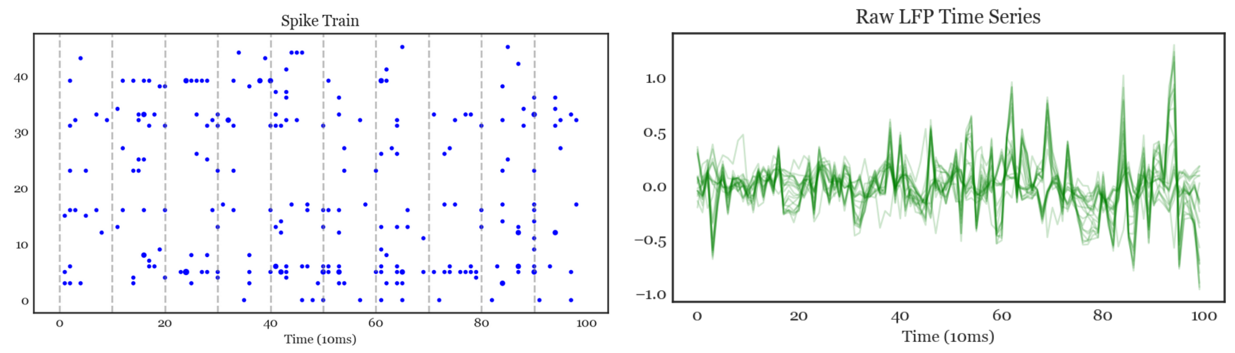


Figure 4.2: Spike train data and LFP data during the first second after odor release from one trial. The data are binned into 10ms time windows because the spike train are sparse with mostly zero counts. It is a challenge to model the sparse discrete data and the noisy continuous data simultaneously.

In previous memory related studies, spike train data are used to associate neuron firing patterns with sensory and other stimuli; this kind of analysis is called brain decoding [6]. Most commonly, the average of spike train data is calculated within small time windows as estimated neuron firing rates. These firing rates can be modeled by Poisson distributions with different means, which are conditional on external stimuli. To make predictions based on observed spike train data, the posterior distribution can be obtained with Bayes rule as follows.

$$\begin{aligned} P(\text{time}|\text{spikes, odor}) &= \frac{P(\text{spikes}|\text{time,odor}) \cdot P(\text{time}|\text{odor})}{P(\text{spikes}|\text{odor})} \\ &= C \cdot P(\text{time}|\text{odor}) \prod_{i=1}^{N} \frac{[\tau f_i(\text{time,odor})]^{n_i}}{n_i!} \exp[-\tau \sum_{i=1}^{N} f_i(\text{time,odor})] \end{aligned}$$

This Bayesian decoding model is applied on each neuron individually. Another approach is to employ a hidden markov model on the ensemble of neurons [20]. Both approaches have been successful when the decoding target is a continuous variable such as rat position that varies across time windows. However, in our experiment, even though the neuron firing rates change across time windows, the external odor stimuli, which are discrete, are the same within each trial. A different approach is needed in order to accurately decode odors and study odor related memory replay within the same trial.

Since 2s is a long time period relative to how rapidly neurons fire, it is reasonable to believe that there is a short time window when the neuron firing pattern is the most distinctive among trials of different odors. Therefore, we select a 250ms time window briefly after odor presentation and use data only in that time window to train a model to differentiate the odors. Then we would apply the trained model in the remaining time windows for decoding; the hope is to see neuron firing patterns of subsequent odors. A baseline decoding model is multinomial logistic regression with LASSO regularization on the neuron firing rates calculated within the training time window. However, we would like to extract all the information in the data by utilizing the LFP data as well. One attempt is to summarize the LFP time series with wavelet coefficients and include them in the multinomial logistic regression model [42].

The most important drawback of the baseline approach for decoding is that the time windows are fixed across trials but the neuron firing patterns actually occur at slightly varied time scales. Intuitively, the same rat may take 100ms to react to an odor in one trial but 150ms in another trial. The temporal variations in odor decoding make it difficult to aggregate results from different trials. It is well known that the hippocampus processes information at specific rhythms. Theta cycles can be determined from the LFP data and used for temporal alignment of different trials [78]. For example, the time scale of spike train data in each trial can be transformed from time in ms to the phase to LFP theta cycles. To determine theta cycles, the LFP signal is smoothed with a Butterworth filter between 4Hz and 7Hz. Then a Hilbert transformation is used to calculate the theta phase at each time point.

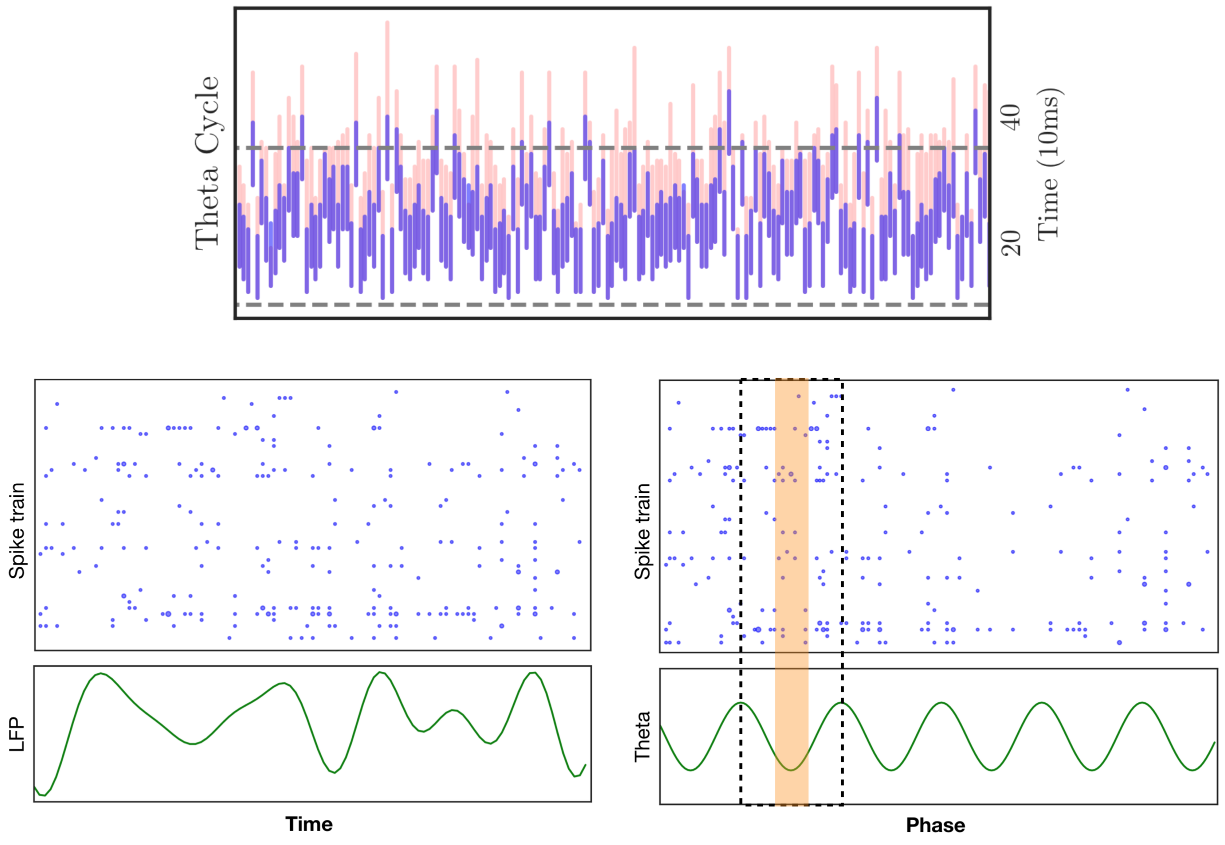


Figure 4.3: Theta cycles occur at different times across trials and last for different duration (top). One common approach is to align the horizontal scale of spike train data with the phase of theta cycles (bottom).

4.3 Tetrode convolution model

Here we introduce the convolutional neural network (CNN) decoding model. In our context, the convolution is done on a tetrode basis. As previosuly mentioned, for each tetrode in the rat hippocampus, there is a continuous LFP signal and multiple spike trains for the neurons around the tetrode. The LFP signal and spike trains are combined into one multivariate time series; filters (small matrices) are convolved on the multivaraite data over time to extract features. The filters have dimensions $(n_{neuron} + 1) \times 1$ and the extracted features have dimensions $n_{filter} \times t$. Time averaged features from different tetrodes are concatenated together and fed into a hidden layer before the output layer that has the predicted probabilities for different odors. The convolution filters and hidden layer parameters are learned during model training to minimize odor decoding error.

For each trial, the 250ms time window starting from 0.1s after odor release is used for training the CNN decoding model. For data processing, the unfiltered LFP and spike train data are first binned every 10ms then scaled to have zero mean and unit standard deviation during the training window. To prevent overfitting, the model is trained with 10-fold cross-validation where the folds are created by stratified sampling. To train the neural network, we use a variant of stochastic gradient descent algorithm and follow the early stopping rule when the validation loss plateaus. As models trained on different folds cannot be easily combined, a final model is trained on all the data to reach the cross-validated accuracy for each rat.

Compared to baseline multinomial logistic regression model with LASSO regularization, the CNN model has a higher overall classification accuracy. We think this is because convolution captures the interaction between LFP and spike data while multiple layers add more flexibility. More importantly, the CNN model creates a nonlinear projection of the LFP and spike train data within a time window. It maps the data to a hidden layer vector then makes the final prediction. The hidden layer vector can be considered as a low dimensional latent representation of the data. When we apply the decoding model on different time windows during the trial, we will obtain corresponding hidden layer vectors. We can visualize these vectors in the low dimensional latent space, where each trial is represented by a point that moves around across time windows.

4.4 Latent space analysis

We want to find recurring neuronal activities that are strongly correlated with non-spatial stimuli by visualizing latent representations in the neural network hidden layer. The idea is

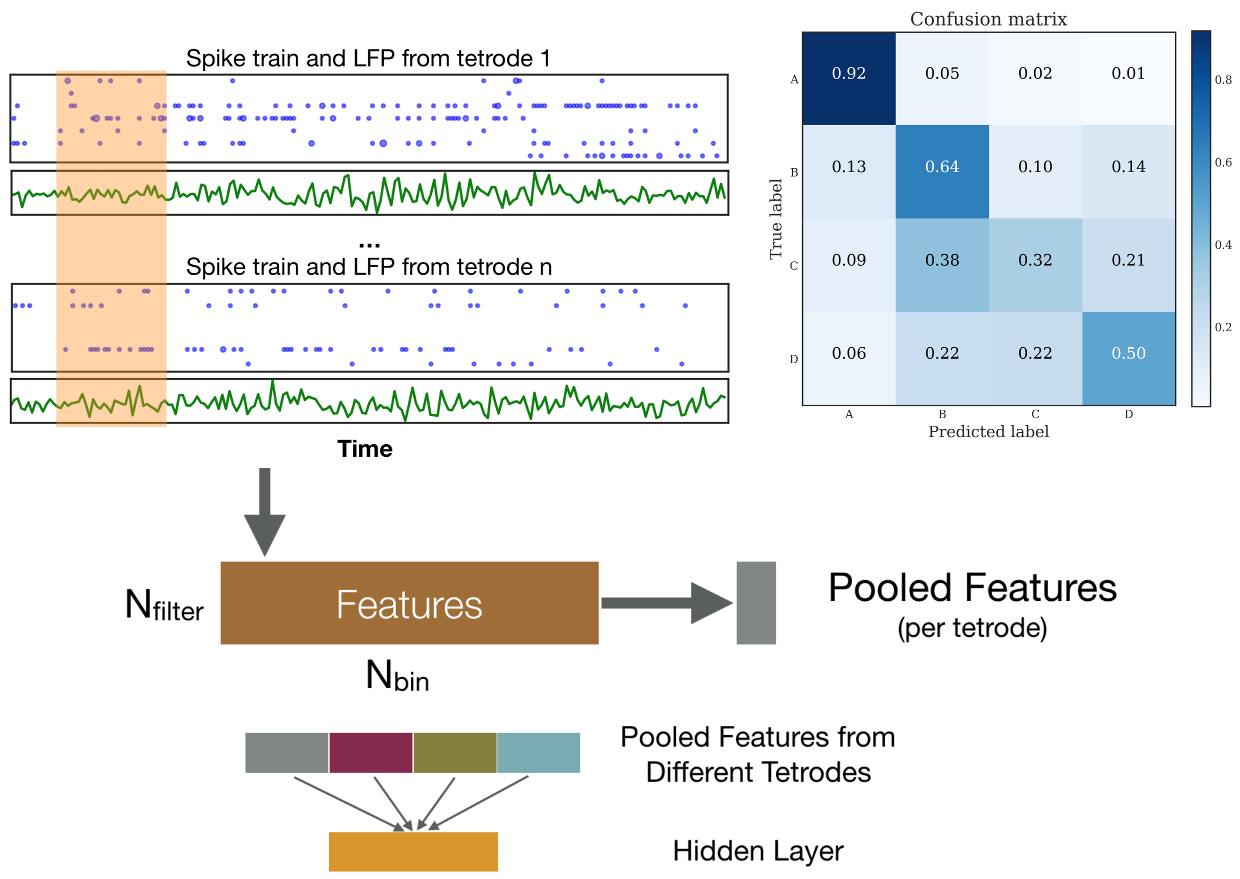


Figure 4.4: The model mapped the LFP and spike train data within the time window to a hidden layer vector and then made a prediction based on that vector. More specifically, the neural network performed convolution on each tetrode separately.

Algorithm 4.1 Neural network decoding model training

```
for each rat in the experiment do
    for each implanted tetrode do
        Organize LFP and spike train data into a multivariate time series
    end for
    Obtain $k$ cross-validation (CV) folds through stratified sampling
    for each of the $k$ training folds do
        Train a CNN decoding model until the validation loss stops decreasing
    end for
    Train the final CNN decoding model on all the data to reach CV accuracy
    for a range of time windows do
        Run the trained CNN decoding model and extract the hidden layer
    end for
end for
```

to divide the latent space into areas corresponding to different odors and movement across these areas in the correct direction would imply sequential replay. Figure shows all the odor B trials for Super Chris in several time windows from before odor release to 1.2s after. We can see that the points cluster in the area of odor B after odor release and the cluster then moves to the areas of odor C and odor D. This means that during these trials, the neuronal activities are associated with different odors in the sequence of B, C, and D.

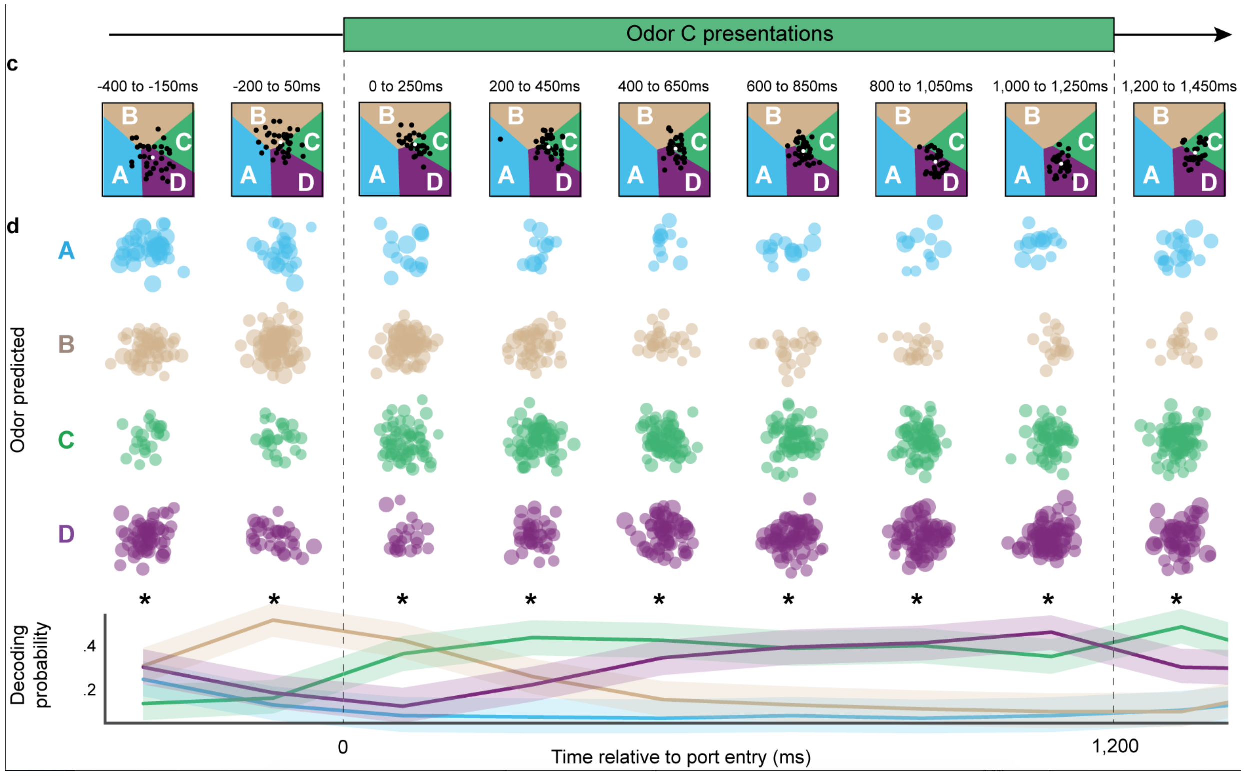


Figure 4.5: The proportions of trials in different odors and 95% multinomial confidence intervals are estimated from neural network predictions in a range of 250ms time windows from -0.4s to 1.2s relative to odor presentation. Then the estimated proportions and confidence intervals endpoints are interpolated with cubic splines for the smooth visualization across the entire time duration.

The two-dimensional space has the first two principal components of the neural network hidden layer. It can be divided by odor with CNN model decision boundaries, where different odors have equal predicted probabilities. Moving along any direction away from the boundary increases a higher predicted probability of an odor. Therefore, each area separated by decision boundaries corresponds to a specific odor. Since there is a separate model for each rat, the model hidden layers cannot be directly compared. To aggregate decoding results across rat, we simply add up the number of points in each odor cluster in the latent space for each rat during a given time window. As a result, during each time window, we obtain four aggregate odor clusters of different sizes but the total number of points remains the constant over time. Figure shows the aggregate odor clusters over time when odor B is presented for all the rats. We can see that the biggest cluster is odor B after odor release but it changes to odor C and odor D later.

4.5 Discussion

Even though the data set is not particularly large and has a lot of noise, the neural network with carefully designed tetrode-wise convolution architecture outperforms the baseline multinomial logistic regression model. Moreover, it is hard to formulate a parametric model to describe the data generation mechanism and test the scientific hypothesis in the context of this experiment. The visualization of the neural network hidden layer provides an intuitive way of understanding the novel phenomenon of sequential odor memory replay. Since the neural network computational cost is manageable here, more work could be done to assess the uncertainty in learning the latent representations.

Chapter 5

Modeling Dynamic Functional Connectivity with Latent Factor Gaussian Processes

5.1 Introduction

The celebrated discoveries of place cells, grid cells, and similar structures in the hippocampus have produced a detailed, experimentally validated theory of the formation and processing of spatial memories. However, the specific characteristics of non-spatial memories, e.g. memories of odors and sounds, are still poorly understood. Recent results from human fMRI and EEG experiments suggest that dynamic functional connectivity (DFC) is important for the encoding and retrieval of memories [27, 45, 31, 64, 79, 62], yet DFC in local field potentials (LFP) in animal models has received relatively little attention. We here propose a novel latent factor Gaussian process (LFGP) model for DFC estimation and apply it to a data set of rat hippocampus LFP during a non-spatial memory task [8]. The model produces strong statistical evidence for DFC and finds distinctive patterns of DFC associated with different experimental stimuli.

Due to the high-dimensionality of time-varying covariance and the complex nature of cognitive processes, effective analysis of DFC requires balancing model parsimony, flexibility, and robustness to noise. DFC models fall into a common framework with three key elements: dimensionality reduction, covariance estimation from time series, and identification of connectivity patterns [68]. Many neuroimaging studies use a combination of various methods, such as sliding window (SW) estimation, principal component analysis (PCA), and the hidden Markov model (HMM) (see e.g. [63, 82, 73]). In general, these methods are not fully probabilistic, which can make uncertainty quantification and inference difficult in practice.

Bayesian latent factor models provide a probabilistic approach to modeling dynamic covariance that allows for simultaneous dimensionality reduction and covariance process estimation. Examples include the latent factor stochastic volatility (LFSV) model [44] and the nonparametric covariance model [33]. In the LFSV model, an autoregressive process is imposed on the latent factors and can be overly restrictive. While the nonparametric model is considerably more flexible, the matrix process for time-varying loadings adds substantial complexity.

Aiming to bridge the gap between these factor models, we propose the latent factor Gaussian process (LFGP) model. In this approach, a latent factor structure is placed on the log-covariance process of a non-stationary multivariate time series, rather than on the observed time series itself as in other factor models. Since covariance matrices lie on the manifold of symmetric positive-definite (SPD) matrices, we utilize the Log-Euclidean metric to allow unconstrained modeling of the vectorized upper triangle of the covariance process. Dimension reduction and model parsimony is achieved by representing each covariance element as a linear combination of Gaussian process latent factors [50].

In this work, we highlight three major advantages of the LFGP model for practical DFC analysis. First, through the prior on the Gaussian process length scale, we are able to incorporate scientific knowledge to target specific frequency ranges that are of scientific interest. Second, the model posterior allows us to perform Bayesian inference for scientific hypotheses, for instance, whether the LFP time series is non-stationary, and if characteristics of DFC differ across experimental conditions. Third, the latent factors serve as a low-dimensional representation of the covariance process, which facilitates visualization of complex phenomena of scientific interest, such as the role of DFC in stimuli discrimination in the context of a non-spatial memory experiment.

5.2 Background

5.2.1 Sliding Window Covariance Estimation

Sliding window methods have been extensively researched for the estimation and analysis of DFC, particularly in human fMRI studies; applications of these methods have identified significant associations of DFC with disease status, behavioral outcomes, and cognitive differences in humans. See [68] for a recent detailed review of existing literature. For $X(t) \sim \mathcal{N}(0, K(t))$ a p-variate time series of length T with covariance process $K(t)$, the sliding window covariance estimate $\hat{K}_{SW}(t)$ with window length L can be written as the convolution $\hat{K}_{SW}(t) = (h * XX')(t) = \sum_{s=1}^{T} h(s)X(t-s)X(t-s)'\,ds$, for the rectangular kernel $h(t) = \mathbf{1}_{[0,L-1]}(t)/L$, where $\mathbf{1}$ is the indicator function. Studies of the performance of sliding window estimates recommend the use of a tapered kernel to decrease the impact of outlying measurements and to improve the spectral properties of the estimate [39, 7, 53]. In the present work we employ a Gaussian taper with scale τ defined as $h^{\tau}(t) = \frac{1}{\zeta} \exp\left\{-\frac{1}{2}\left(\frac{t-L/2}{\tau L/2}\right)^2\right\} \mathbf{1}_{[0,L-1]}(t)$, where ζ is a normalizing constant. The corresponding tapered SW estimate is $\hat{K}_{\tau}(t) = (h^{\tau} * XX')(t)$.

▨ 5.2.2 Log-Euclidean Metric

Direct modeling of the covariance process from the SW estimates is complicated by the positive definite constraint of the covariance matrices. To ensure the model estimates are positive definite, it is necessary to employ post-hoc adjustments, or to build the constraints into the model, typically by utilizing the Cholesky or spectral decompositions. The LFGP model instead uses the Log-Euclidean framework of symmetric positive definite (SPD) matrices to naturally ensure positive-definiteness of the estimated covariance process while also simplifying the model formulation and implementation.

Denote the space of $p \times p$ SPD matrices as $\mathbb{P}_p$. For $X_1, X_2 \in \mathbb{P}_p$, the *Log-Euclidean* distance is defined by $d_{LE}(X_1, X_2) = \|\mathrm{Log}(X_1) - \mathrm{Log}(X_2)\|$, where Log is the matrix logarithm, and $\|\cdot\|$ is the Frobenius norm. The metric space $(\mathbb{P}_p, d_{LE})$ is a Riemannian manifold that is isomorphic to $\mathbb{R}^q$ with the usual Euclidean norm, for $q = (p+1)p/2$.

Methods for modeling covariances in regression contexts via the matrix logarithm were first introduced in [25]. The Log-Euclidean framework for analysis of SPD matrices in neuroimaging contexts was first proposed in [10], with further applications in neuroimaging having been developed in recent years [86]. The present work is a novel application of the Log-Euclidean framework for DFC analysis.

▨ 5.2.3 Bayesian Latent Factor Models

For $x_{ij}, i = 1, \dots, n, j = 1, \dots, p$, the simple Bayesian latent factor model is $x_i = f_i \Lambda + \varepsilon_i$, with $f_i \overset{iid}{\sim} \mathcal{N}(0, I_r), \varepsilon_i \overset{iid}{\sim} \mathcal{N}(0, \Sigma)$, and Λ an $r \times p$ matrix of factor loadings [4]. Σ is commonly assumed to be a diagonal matrix, implying the latent factors capture all the correlation structure of the p features of x. The latent factor model shares some similarities with principal component analysis, but includes a stochastic error term, which leads to a different interpretation of the resulting factors [63, 82].

Variants of the linear factor model have been developed for modeling non-stationary multivariate time series [67, 58]. In general, these models represent the p-variate observed time series as a linear combination of r latent factors $f_j(t), j = 1, \ldots, r$, with $r \times q$ loading matrix Λ and errors $\varepsilon(t)$: $X(t) = f(t)\Lambda + \varepsilon(t)$. From this general modeling framework, numerous methods for capturing the non-stationary dynamics in the underlying time series have been developed, such as latent factor stochastic volatility (LFSV) [44], dynamic conditional correlation [54], and the nonparametric covariance model [33].

5.2.4 Gaussian Processes

A Gaussian process ($\mathcal{GP}$) is a continuous stochastic process for which any finite collection of points are jointly Gaussian with some specified mean and covariance. A $\mathcal{GP}$ can be understood as a distribution on functions belonging to a particular reproducing kernel Hilbert space (RKHS) determined by the covariance operator of the process [81]. Typically, a zero mean $\mathcal{GP}$ is assumed (i.e. the functional data has been centered by subtracting a consistent estimator of the mean), so that the $\mathcal{GP}$ is parameterized entirely by the kernel function κ that defines the pairwise covariance. Let $f \sim \mathcal{GP}(0, k(\cdot, \cdot))$. Then for any x and x' we have

$$\begin{pmatrix} f(x) \\ f(x') \end{pmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \kappa(x,x) & \kappa(x,x') \\ \kappa(x,x') & \kappa(x',x') \end{bmatrix}\right). \tag{5.1}$$

Further details are given in [69].

▢ 5.3 Latent Factor Gaussian Process Model

▢ 5.3.1 Formulation

We consider estimation of dynamic covariance from a sample of n independent time series with p variables and T time points. Denote the ith observed p-variate time series by $X_i(t)$, $i = 1, \cdots, n$. We assume that each $X_i(t)$ follows an independent distribution $\mathcal{D}$ with zero mean and stochastic covariance process $K_i(t)$. To model the covariance process, we first compute the Gaussian tapered sliding window covariance estimates for each $X_i(t)$, with fixed window size L and taper τ to obtain $\hat{K}_{\tau,i}$. We then apply the matrix logarithm to obtain the $q = p(p+1)/2$ length vector $Y_i(t)$ specified by $\hat{K}_{\tau,i} = \text{Log}(\vec{\mathbf{u}}(Y_i))$, where $\vec{\mathbf{u}}$ maps a matrix to its vectorized upper triangle. We refer to $Y_i(t)$ as the "log-covariance" at time t.

The resulting $Y_i(t)$ can be modeled as an unconstrained q-variate time series. The LFGP model represents $Y_i(t)$ as a linear combination of r latent factors $F_i(t)$ through an $r \times q$ loading matrix B and independent Gaussian errors ϵ_i. The loading matrix B is held constant across observations and time. Here $F_i(t)$ is modeled as a product of independent Gaussian processes. Placing priors p_1, p_2, p_3 on the loading matrix B, Gaussian noise variance σ^2, and Gaussian process hyper-parameter θ, respectively, gives a fully probabilistic latent factor model on the covariance process:

$$X_i(t) \sim \mathcal{D}(0, K_i(t)) \text{ where } K_i(t) = \exp\left(\vec{\mathbf{u}}(Y_i(t))\right) \tag{5.2}$$

$$Y_i(t) = F_i(t) \cdot B + \epsilon_i \text{ where } \epsilon_i \overset{iid}{\sim} \mathcal{N}(0, I\sigma^2) \tag{5.3}$$

$$F_i(t) \sim \mathcal{GP}(0, \kappa(t; \theta)) \tag{5.4}$$

$$B \sim p_1, \sigma^2 \sim p_2, \theta \sim p_3. \tag{5.5}$$

The LFGP model employs a latent distribution of curves $\mathcal{GP}(0, \kappa(t; \theta))$ to capture temporal dependence of the covariance process, thus inducing a Gaussian process on the log-covariance

$Y(t)$. This conveniently allows multiple observations to be modeled as different realizations of the same induced $\mathcal{GP}$ as done in [48]. The model posteriors are conditioned on different observations despite sharing the same kernel. For better identifiability, the $\mathcal{GP}$ variance scale is fixed so that the loading matrix can be unconstrained.

5.3.2 Properties

Theorem 5.1. *The log-covariance process induced by the LFGP model is weakly stationary when the GP kernel $\kappa(s,t)$ depends only on $|s-t|$.*

Proof. The covariance of the log-covariance process $Y(t)$ depends only on the static loading matrix $B = (\beta_{kj})_{1\le k\le r; 1\le j\le q}$ and the factor covariance kernels. Explicitly, for factor kernels $\kappa(s,t;\theta_k), k = 1,\dots,r$, and assuming $\varepsilon_i(t) \overset{iid}{\sim} \mathcal{N}(0,\Sigma)$, with $\Sigma = (\sigma^2_{jj'})_{j,j'\le q}$ constant across observations and time, the covariance of elements of $Y(t)$ is

$$\operatorname{Cov}(Y_{ij}(s), Y_{ij'}(t)) = \operatorname{Cov}\left(\sum_{k=1}^r F_{ik}(s)\beta_{kj} + \varepsilon_{ij'}(t), \sum_{k=1}^r F_{ik}(t)\beta_{kj'} + \varepsilon_{ij'}(t)\right) \tag{5.6}$$

$$= \sum_{k=1}^r \beta_{kj}\beta_{kj'}\kappa(s,t;\theta_k) + \sigma^2_{jj'}, \tag{5.7}$$

which is weakly stationary when $\kappa(s,t)$ depends only on $|s-t|$. □

Posterior contraction. To consider posterior contraction of the LFGP model, we make the following assumptions. The true log-covariance process $w = \vec{\mathbf{u}}(\log(K(t))$ is in the support of the product $\mathcal{GP}$ $W \sim F(t)B$, for $F(t)$ and B defined above, with known number of latent factors r. The $\mathcal{GP}$ kernel κ is α-Hölder continuous with $\alpha \ge 1/2$. $Y(t) : [0,1] \to \mathbb{R}^q$ is a smooth function in $\ell^\infty_q([0,1])$ with respect to the Euclidean norm, and the prior p_2 for σ^2 has support on a given interval $[a,b] \subset (0,\infty)$. Under the above assumptions, bounds on the posterior contraction rates then follow from previous results on posterior contraction of

Gaussian process regression for α-smooth functions given in [36, 80]. Specifically,

$$E_0\Pi_n((w,\sigma) : \|w - w_0\|_n + |\sigma - \sigma_0| > M\varepsilon_n | Y_1, \cdots, Y_n) \to 0$$

for sufficiently large M and with posterior contraction rate $\varepsilon_n = n^{-\alpha/(2\alpha+q)} \log^\delta(n)$ for some $\delta > 0$, where $E_0(\Pi_n(\cdot|Y_1, \cdots, Y_n))$ is the expectation of the posterior under the model priors.

To illustrate posterior contraction in the LFGP model, we simulate data for five signals with various sample sizes (n) and numbers of observation time points (t), with a covariance process generated by two latent factors. To measure model bias, we consider the mean squared error of posterior median of the reconstructed log-covariance series. To measure posterior uncertainty, the posterior sample variance is used. As shown in Table 5.1, both sample size n and number of observation time points t contribute to posterior contraction.

Table 5.1: Mean squared error of posterior median (posterior sample variance) $\times 10^{-2}$

	$n = 1$	$n = 10$	$n = 20$	$n = 50$
$t = 25$	12.212 (20.225)	7.845 (8.743)	7.089 (7.714)	5.869 (7.358)
$t = 50$	6.911 (7.588)	4.123 (5.836)	3.273 (3.989)	3.237 (3.709)
$t = 100$	3.728 (5.218)	1.682 (2.582)	1.672 (2.659)	1.672 (1.907)

Large prior support. The prior distribution of the log-covariance process $Y(t)$ is a linear combination of r independent $\mathcal{GP}$s each with mean 0 and kernel $\kappa(s, t; \theta_k), k = 1, \cdots, r$. That is, each log-covariance element will have prior $Y_j(t) = \sum_{k=1}^r \beta_{jk} F_k(t) \sim \mathcal{GP}(0, \sum \beta_{jk}^2 \kappa(s, t; \theta_k))$. Considering B fixed, the resulting prior for $F_i(t)B$ has support equal to the closure of the reproducing kernel Hilbert space (RKHS) with kernel $B^T \mathcal{K}(t, \cdot)B$ [69], where $\mathcal{K}$ is the covariance tensor formed by stacking $\kappa_k = \kappa(s, t; \theta_k), k = 1, \cdots, r$ [81]. Accounting for the prior p_1 of B, a function $W \in \ell_q^\infty[0, 1]$ will have nonzero prior probability $\Pi_0(W) > 0$ if W is in the closure of the RKHS with kernel $A^T \mathcal{K}(t, \cdot)A$ for some A in the support of p_1.

5.3.3 Factor Selection via the Horseshoe Prior

Similar to other factor models, the number of latent factors in the LFGP model has a crucial effect on model performance, and must be selected somehow. For Bayesian factor analysis, there is extensive literature on factor selection methods, such as Bayes factors, reversible jump sampling [55], and shrinkage priors [18]. While we can compare different models in terms of goodness-of-fit, we cannot compare their latent factors in a meaningful way due to identifiability issues. Therefore, we instead iteratively increase the number of factors and fit the new factors on the residuals resulting from the previous fit. In order to avoid overfitting with too many factors, we place a horseshoe prior on the loadings of the new factors, so that the loadings shrink to zero if the new factor is unnecessary.

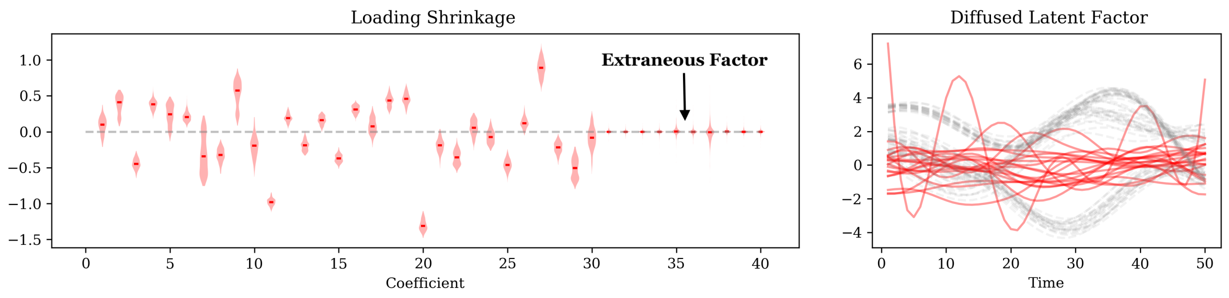


Figure 5.1: Violin plots of loading posteriors show that the loadings for the fourth factor (indices 30 to 39) shrink to zero with the horseshoe prior (left). Compared to the posteriors of the first three factors (dashed gray), the posterior of the extraneous factor (solid red) is diffused around zero as a result of zero loadings (right).

Introduced by [23], the horseshoe prior in the regression setting is given by

$$\beta|\lambda, \tau \sim N(0, \lambda^2\rho^2) \tag{5.8}$$

$$\lambda \sim Cauchy^+(0, 1) \tag{5.9}$$

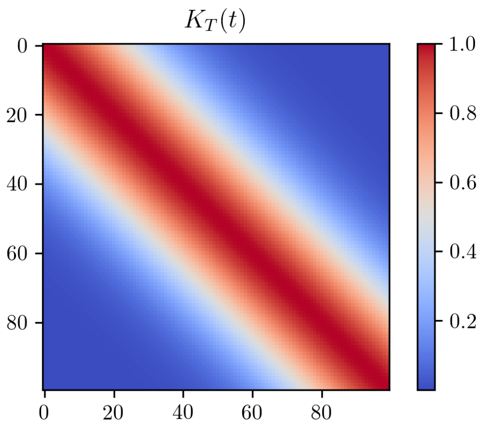
and can be considered as a scale-mixture of Gaussian distributions. A small global scale ρ encourages shrinkage, while the heavy tailed Cauchy distribution allows the loadings to escape from zero. The example shown in Figure 5.1 illustrates the shrinkage effect of the horseshoe prior when iteratively fitting an LFGP model with four factors to simulated data generated from three latent factors. For sampling from the loading posterior distribution, we use the No-U-Turn Sampler [41] as implemented in PyStan [22].

5.3.4 Scalable Computation

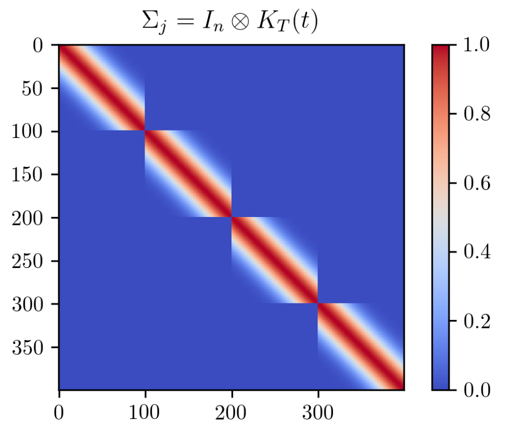
The LFGP model can be fit via Gibbs sampling, as commonly done for Bayesian latent variable models. In every iteration, we first sample $F|B, \sigma^2, \theta, Y$ from the conditional $p(F|Y)$ as F, Y are jointly multivariate Gaussian where the covariance can be written in terms of B, σ^2, θ. However, it is worth noting that this multivariate Gaussian has a large covariance matrix, which could be computationally expensive to invert. Given F, the parameters B, σ^2 and θ become conditionally independent. Using conjugate priors for Bayesian linear regression, the posterior $p(B, \sigma^2|F, Y)$ is directly available. For the $\mathcal{GP}$ parameter posterior $p(\theta|F)$, either Metropolis random walk or slice sampling [60] can be used within each Gibbs step because the parameter space is low dimensional.

For efficient $\mathcal{GP}$ posterior sampling, it is essential to exploit the structure of the covariance matrix. For each independent latent $\mathcal{GP}$ factor F_j, there are n independent sets of observations at t time points. Therefore, the $\mathcal{GP}$ covariance matrix Σ_j has dimensions $nT \times nT$. To reduce the computational burden, we notice that the covariance Σ_j can be decomposed using a Kronecker product $\Sigma_j = I_n \otimes K_{time}(t)$, where K_{time} is the $T \times T$ temporal covariance. The cost to invert Σ_j using this decomposition is $\mathcal{O}(T^3)$, which is a substantial reduction compared to the original cost $\mathcal{O}((nT)^3)$. For many choices of kernel, such as the squared-exponential or Matérn kernel, $K_{time}(t)$ has a Toeplitz structure and can be approximated through interpolation [84], further reducing the computational cost.

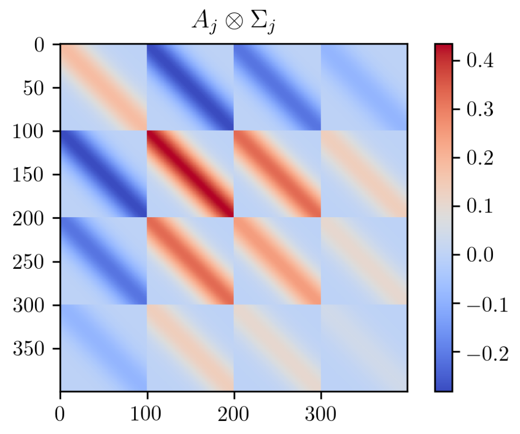
Combining the latent $\mathcal{GP}$ factors F (dimensions $n \times T \times r$) and loading matrix B (dimensions $r \times q$) induces a $\mathcal{GP}$ on Y. The dimensionality of Y is $n \times T \times q$ so the full $(nTq) \times (nTq)$



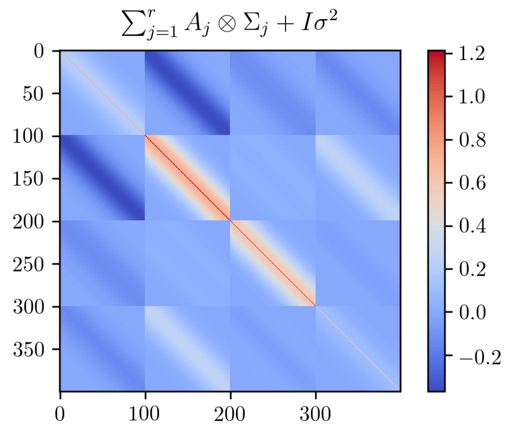
(a)



(b)



(c)



(d)

Figure 5.2: The full covariance matrix Σ_Y is composed of building blocks of smaller matrices. (a) $\mathcal{GP}$ covariance matrix at evenly-spaced time points, (b) covariance matrix of factor F_j for n sets of observations, (c) contribution to the covariance of Y from factor F_j, and (d) full covariance matrix Σ_Y.

covariance matrix Σ_Y is prohibitive to invert. As every column of Y is a weighted sum of the $\mathcal{GP}$ factors, the covariance matrix Σ_Y can be written as a sum of Kronecker products $\sum_{j=1}^{r} A_j \otimes \Sigma_j + I\sigma^2$, where Σ_j is the covariance matrix of the jth latent $\mathcal{GP}$ factor and A_j is a $q \times q$ matrix based on the factor loadings. We can regress residuals of Y on each column of F iteratively to sample from the conditional distribution $p(F|Y)$ so that the residual covariance is only $A_j \otimes \Sigma_j + I$. The inversion can be done in a computationally efficient way with the following matrix identity

$$(C \otimes D + I)^{-1} = (P \otimes Q)^T (I + \Lambda_1 \otimes \Lambda_2)^{-1} (P \otimes Q) \tag{5.10}$$

where $C = P\Lambda_1 P^T$ and $D = Q\Lambda_2 Q^T$ are the spectral decompositions. In the identity, obtaining $P, Q, \Lambda_1, \Lambda_2$ costs $\mathcal{O}(q^3)$ and $\mathcal{O}((nT)^3)$, which is a substantial reduction from the cost of direct inversion, $\mathcal{O}((nTq)^3)$; calculating $(I + \Lambda_1 \otimes \Lambda_2)^{-1}$ is straightforward since Λ_1 and Λ_2 are diagonal.

Algorithm 5.1 Sampling algorithms for LFGP model

```
Initialize parameters $B, \sigma^2, \theta$ from prior distributions, Metropolis random walk step number $L$ and step size $\epsilon$
for $t = 1, 2, ..., T$ do
    Calculate the joint distribution $p(Y, F)$ with given $B, \sigma^2, \theta$
    Sample latent factors $F_t$ from $\mathcal{MVN}(F|Y)$
    Obtain the posterior of $B, \sigma^2$ through $Y \sim F_t$ with conjugate priors
    Perform $\mathcal{GP}$ regression on latent factors $F_t$
    for $i = 1, 2, ..., L$ do
        Take a random step $\epsilon$ to update $\mathcal{GP}$ parameter $\theta$
    end for
    Keep current values of $B, \sigma^2, \theta$
end for
```

5.4 Experiments

5.4.1 Model Comparisons on Simulated Data

We here consider three benchmark models: sliding window with principal component analysis (SW-PCA), hidden Markov model, and LFSV model. SW-PCA and HMM are commonly used in DFC studies but have severe limitations. The sliding window covariance estimates are consistent but noisy, and PCA does not take the estimation error into account. HMM is a probabilistic model and can be used in conjunction with a time series model, but it is not well-suited to capturing smoothly varying dynamics in brain connectivity.

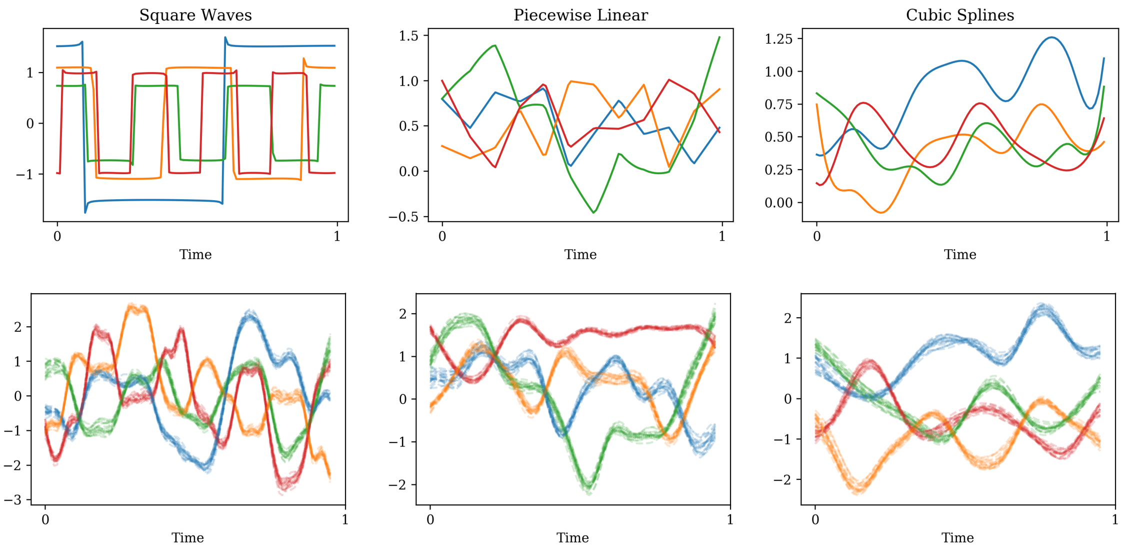


Figure 5.3: With the jagged dynamics of discrete states, the LFGP model fails to capture the “jumps” but approximates the overall trend (left). When the underlying dynamics are smooth, the LFGP model can accurately recover the shape up to some scaling constant (right).

To compare the performance of different models, we simulate time series data $X_t \sim N(0, K(t))$ with time-varying covariance $K(t)$. The covariance $K(t)$ follows deterministic dynamics that are given by $\vec{\mathbf{u}}(\log(K(t))) = U(t) \cdot A$. We consider three different scenarios of dynamics $U(t)$: square waves, piece-wise linear functions, and cubic splines. Note that both square waves and piece-wise linear functions give rise to dynamics that are not well-represented by the LFGP model when the squared-exponential kernel is used. For each scenario, we randomly generate 100 time series data sets and fit all the models. The evaluation metric is reconstruction loss of the covariance as measured by the Log-Euclidean distance. The simulation results in Table 5.2 show that the proposed LFGP model has the lowest reconstruction loss among the methods considered. Each time series has 10 variables with 1000 observations and the latent dynamics are 4-dimensional as illustrated in Figure 5.3. For the SW-PCA model, the sliding window size is 50 and the number of principal components is 4. For the HMM, the number of hidden states is increased gradually until the model does not converge, following the implementation outlined in [19]. For the LFSV model, the R package *factorstochvol* is

Table 5.2: Median reconstruction loss (standard deviation) across 100 data sets

	SW-PCA	HMM	LFSV	LFGP
Square save	0.693 (0.499)	1.003 (1.299)	4.458 (2.416)	0.380 (0.420)
Piece-wise	0.034 (0.093)	0.130 (0.124)	0.660 (0.890)	0.027 (0.088)
Smooth spline	0.037 (0.016)	0.137 (0.113)	0.532 (0.400)	0.028 (0.123)

used with default settings. All simulations are run on a 2.7 GHz Intel Core i5 Macbook Pro laptop with 8GB memory.

5.4.2 Application to Rat Hippocampus Local Field Potentials

To investigate the neural mechanisms underlying the temporal organization of memories, [8] recorded neural activity in the CA1 region of the hippocampus as rats performed a sequence memory task. The task involves the presentation of repeated sequences of 5 stimuli (odors A, B, C, D, and E) at a single port and requires animals to correctly identify each stimulus as being presented either "in sequence (e.g., ABC...) or "out of sequence (e.g., ABD...) to receive a reward. Here the model is applied to local field potential (LFP) activity recorded from the rat hippocampus, but the key reason for choosing this data set is that it provides a rare opportunity to subsequently apply the model to other forms of neural activity data collected using the same task (including spiking activity from different regions in rats [42] and whole-brain fMRI in humans).

LFP signals were recorded in the hippocampi of five rats performing the task. The local field potentials are measured by surgically implanted tetrodes and the exact tetrode locations vary across rats. Therefore, it may not make sense to compare LFP channels of different rats. This issue actually motivates the latent factor approach because we want to eventually visualize and compare the latent trajectories for all the rats. For the present analysis, we have focused on the data from a particular rat exhibiting the best memory task performance. To boost the signal-to-noise ratio, six LFP channels that recorded a majority of the attached neurons were chosen. Only trials of odors B and C were considered, to avoid potential confounders with odor A being the first odor presented, and due to substantially fewer trials for odors D and E.

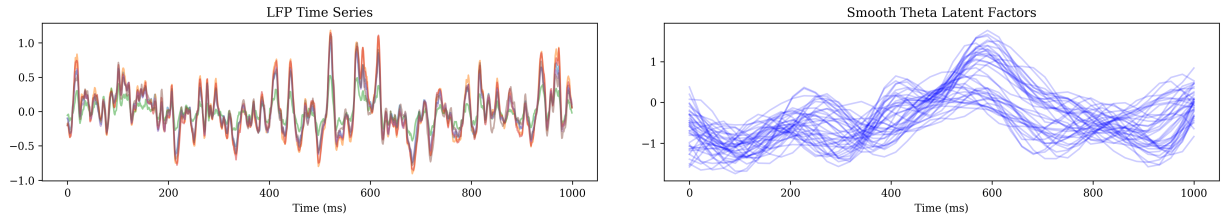


Figure 5.4: Time series of 6 LFP channels for a single trial sampled at 1000Hz include all frequency components (left). Posterior draws of latent factors for the covariance process appear to be smoothly varying near the theta frequency range (right).

During each trial, the LFP signals are sampled at 1000Hz for one second after odor release. We focus on 41 trials of odor B and 37 trials of odor C. Figure 5.4 shows the time series of these six LFP channels for a single trial. We treat all 78 trials as different realizations of the same stochastic process without distinguishing the stimuli explicitly in the model. In order to facilitate interpretation of the latent space representation, we fit two latent factors which explain about 40% of the variance in the data. The prior for $\mathcal{GP}$ length scale is a Gamma distribution concentrated around 100ms on the time scale to encourage learning frequency dynamics close to the theta range (4-12 Hz). Notably, oscillations in this frequency range have been associated with memory function but have not previously been shown to differentiate among the type of stimuli used here, thus providing an opportunity to test the sensitivity of the model. For the loadings and variances, we use the Gaussian-Inverse Gamma conjugate priors. 20,000 MCMC draws are taken with the first 5000 draws discarded as burn-in.

For each odor, we can calculate the posterior median latent factors across trials and visualize them as a trajectory in the latent space. Figure 5.5 shows that the two trajectories start in an almost overlapping area, with separation occurring around 250ms. This is corroborated by the experimental data indicating that animals begin to identify the odor 200-250ms

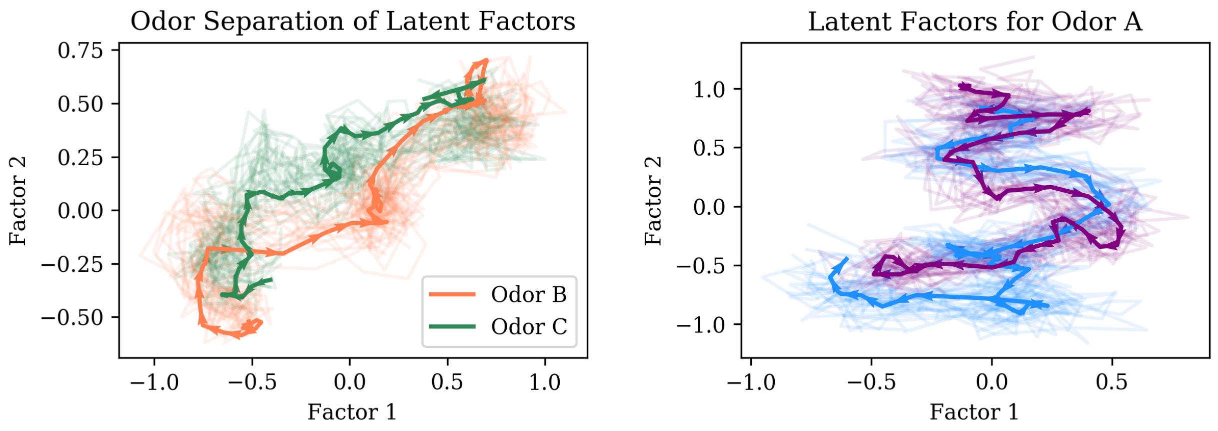


Figure 5.5: Posterior draws of median $\mathcal{GP}$ factors visualized as trajectories in latent space can be separated based on the odor, with maximum separation around 250ms (left). The latent trajectories are much more intertwined when the model is fitted to data of the same odor. (right)

after onset. We also observe that the two trajectories converge toward the end of the odor presentation. This is also consistent with the experimental data showing that, by then, animals have correctly identified the odors and are simply waiting to perform the response (thereby resulting in similar neural states). In order to quantify odor separation, we can evaluate the difference between the posterior distributions of odor median latent trajectories by using classifiers on the MCMC draws. We also fit the model to two random subsets of the 58 trials of odor A and train the same classifiers. Table 5.3) shows the classification results and the posteriors are more separated for different odors.

Table 5.3: Odor separation as measured by Latent space classification accuracy (standard deviation)

	Different odors	Same odor
Logistic regression	69.97 (0.78)	63.10 (0.91)
k-NN	87.12 (0.33)	78.41 (0.65)
SVM	74.53 (0.67)	64.75 (1.21)

As a comparison, a hidden Markov model was fit to the LFP data from the same six selected tetrodes. Figure 5.6 compares the estimated covariance with different models. Eight states were selected with an elbow method using the AIC of the HMM; we note that the minimum AIC is not achieved for less than 50 states, suggesting that the dynamics of the LFP covariance may be better described with a continuous model. Moreover, the proportion of time spent in each state for odor B and C trials given in Table 5.4 fails to capture odor separation in the LFP data.

Table 5.4: State proportions for odors B and C as estimated by HMM

Odor	State 1	State 2	State 3	State 4	State 5	State 6	State 7	State 8
B	0.123	0.089	0.146	0.153	0.109	0.159	0.160	0.061
C	0.133	0.092	0.144	0.147	0.106	0.164	0.152	0.062

Collectively, these results provide compelling evidence that this model can use LFP activity to differentiate the representation of different stimuli, as well as capture their expected dynamics within trials. Stimuli differentiation has frequently been accomplished by analyzing spiking activity, but not LFP activity alone. This approach, which may be applicable to other types of neural data including spiking activity and fMRI activity, may significantly advance our ability to understand how information is represented among brain regions.

5.5 Discussion

The proposed LFGP model is a novel application of latent factor models for directly modeling the dynamic covariance in multivariate non-stationary time series. As a fully probabilistic approach, the model naturally allows for inference regarding the presence of DFC, and for detecting differences in connectivity across experimental conditions. Moreover, the latent factor structure enables visualization and scientific interpretation of connectivity patterns. Currently, the main limitation of the model is scalability with respect to the number of observed signals. Thus, in practical applications it may be necessary to select a relevant subset of the observed signals, or apply some form of clustering of similar signals. Future

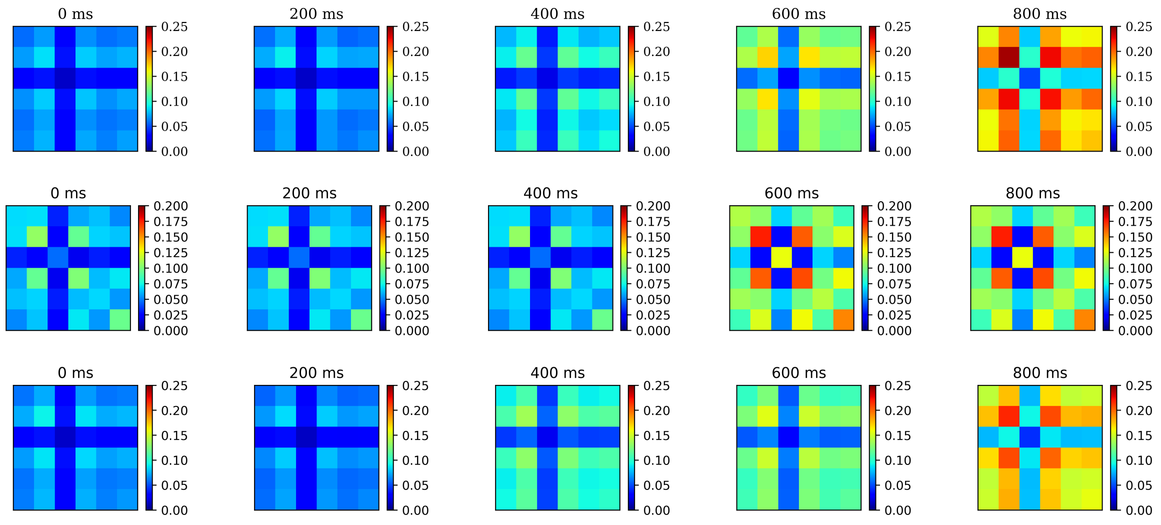


Figure 5.6: Median covariance matrices over time for odor B trials estimated with sliding window (top), HMM (middle), and LFGP model (bottom) reveal similar patterns in dynamic connectivity in the six LFP channels.

work will consider simultaneously reducing the dimension of the signals and modeling the covariance process to improve the scalability and performance of the LFGP model.

The Gaussian process regression framework is a new avenue for analysis of DFC in many neuroimaging modalities. Within this framework, it is possible to incorporate other covariates in the kernel function to naturally account for between-subject variability. In our setting, multiple trials are treated as independent observations or repeated measurements from the same rat, while in human neuroimaging studies, there are often single observations from many subjects. Pooling information across subjects in this setting could yield more efficient inference and lead to more generalizable results.

Chapter 6

Future Work

In conclusion, neural networks and Gaussian processes can provide flexible approximations to improve statistical inference on scientific data by reducing the computational burden and allowing more general model specifications. First, we have presented a neural network gradient approximation scheme to essentially remove the data burden during posterior sampling using Hamiltonian Monte carlo. An exciting future direction for this work is in the setting of distributed learning. As the volume of data further increases, it would not be reasonable to expect all the data to be on a single machine waiting to be analyzed. If the data are distributed among a cluster of machines, we could build the neural network gradient approximation for the data on each machine and aggregate the posteriors without the need to centralize the data. In addition, by keeping data on each machine, we would preserve the privacy of individuals in the data and this would be particularly important when analyzing sensitive data such as those in the healthcare system.

Second, we have demonstrated a Gaussian process guided algorithm for parameter inference with a simulation based physics model. Typically, physicists would perform a large number of simulations to reduce numerical error; our algorithm directly takes the error into account and leaves more room for error when possible to be more efficient. Numerical simulations are widely used for complex scenarios in physical sciences and engineering applications; many of these numerical simulations require substantial computational resources. Hence, there is a whole range of possible use cases for our algorithm.

Lastly, we have proposed a neural network model and a Gaussian process model to reveal interesting patterns from neuroscience experimental data. These models are not explicitly specified *a priori* and can learn latent representations of the data, which would be used for scientific interpretation. In the neural network model example, latent representations could be combined from different subjects because they share the common stimuli information. This effectively strengthens a weak pattern among the subjects into a clear trend overall. In the Gaussian process model example, the latent representation is more general and plausible compared to the usually assumed discrete latent states. This also leads to incorporation of scientific knowledge through prior distribution in the latent space. It remains a challenge for statisticians and neuroscientists to come up with informative priors together.

Bibliography

[1] K. Abe et al. Measurement of neutrino and antineutrino oscillations by the T2K experiment including a new additional sample of ν_e interactions at the far detector. *Physical Review D*, 96(9):092006, 2017.

[2] M. A. Acero et al. New constraints on oscillation parameters from ν_e appearance and ν_μ disappearance in the NOvA experiment. *Physical Review D*, 98(3):032012, 2018.

[3] P. Adamson et al. Search for sterile neutrinos in MINOS and MINOS+ using a two-detector fit. *Physical Review Letters*, 122(9):091803, 2019.

[4] O. Aguilar, G. Huerta, R. Prado, and M. West. Bayesian inference on latent structure in time series. *Bayesian Statistics*, 6(1):1–16, 1998.

[5] Q. R. Ahmad et al. Direct Evidence for Neutrino Flavor Transformation from Neutral-Current Interactions in the Sudbury Neutrino Observatory. *Physical Review Letters*, 89 (011301), 2002.

[6] J. Aljadeff, B. J. Lansdell, A. L. Fairhall, and D. Kleinfeld. Analysis of neuronal spike trains, deconstructed. *Neuron*, 91(2):221–259, 2016.

[7] E. A. Allen, E. Damaraju, S. M. Plis, E. B. Erhardt, T. Eichele, and V. D. Calhoun. Tracking whole-brain connectivity dynamics in the resting state. *Cerebral cortex*, 24(3): 663–676, 2014.

[8] T. A. Allen, D. M. Salz, S. McKenzie, and N. J. Fortin. Nonspatial sequence coding in ca1 neurons. *Journal of Neuroscience*, 36(5):1547–1563, 2016.

[9] F. P. An et al. Measurement of electron antineutrino oscillation based on 1230 days of operation of the Daya Bay experiment. *Physical Review D*, 95(072006), 2017.

[10] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache. Fast and simple calculus on tensors in the log-euclidean framework. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 115–122. Springer, 2005.

[11] P. Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49, 2012.

[12] P. Baldi and P. Sadowski. A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Networks*, 83:51–74, 2016.

[13] P. Baldi and P. J. Sadowski. Understanding dropout. In *Advances in neural information processing systems*, pages 2814–2822, 2013.

[14] R. Barlow. Extended Maximum Likelihood. *Nuclear Instruments and Methods in Physics, Volume 293, Issue 3*, 1990.

[15] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1798–1828, 2013.

[16] J. Bernardo, J. Berger, A. Dawid, A. Smith, et al. Regression and classification using gaussian process priors. *Bayesian statistics*, 6:475, 1998.

[17] M. Betancourt. The fundamental incompatibility of hamiltonian monte carlo and data subsampling. *arXiv preprint arXiv:1502.01510*, 2015.

[18] A. Bhattacharya and D. B. Dunson. Sparse bayesian infinite factor models. *Biometrika*, pages 291–306, 2011.

[19] J. A. Bilmes et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. 1998.

[20] M. Box, M. W. Jones, and N. Whiteley. A hidden markov model for decoding and the analysis of replay in spike trains. *Journal of computational neuroscience*, 41(3):339–366, 2016.

[21] R. Brun and F. Rademakers. ROOT : An object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 389(1-2):81–86, 1997.

[22] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1), 2017.

[23] C. M. Carvalho, N. G. Polson, and J. G. Scott. Handling sparsity via the horseshoe. In *Artificial Intelligence and Statistics*, pages 73–80, 2009.

[24] T. Chen, E. Fox, and C. Guestrin. Stochastic gradient hamiltonian monte carlo. In *International Conference on Machine Learning*, pages 1683–1691, 2014.

[25] T. Y. Chiu, T. Leonard, and K.-W. Tsui. The matrix-logarithmic covariance model. *Journal of the American Statistical Association*, 91(433):198–210, 1996.

[26] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[27] A. Demertzi, E. Tagliazucchi, S. Dehaene, G. Deco, P. Barttfeld, F. Raimondo, C. Martial, D. Fernández-Espejo, B. Rohaut, H. Voss, et al. Human consciousness is supported by dynamic complex patterns of brain signal coordination. *Science Advances*, 5(2): eaat7603, 2019.

[28] G. Drexlin, V. Hannen, S. Mertens, and C. Weinheimer. Current direct neutrino mass experiments. *Advances in High Energy Physics*, 2013, 2013.

[29] D. Duvenaud. *Automatic model construction with Gaussian processes.* PhD thesis, University of Cambridge, 2014.

[30] G. J. Feldman and R. D. Cousins. Unified approach to the classical statistical analysis of small signals. *Physical Review D*, 57(7):3873, 1998.

[31] M. Fiecas and H. Ombao. Modeling the evolution of dynamic brain processes during an associative learning experiment. *Journal of the American Statistical Association*, 111 (516):1440–1453, 2016.

[32] J. A. Formaggio and G. Zeller. From eV to EeV: Neutrino cross sections across energy scales. *Reviews of Modern Physics*, 84(3):1307, 2012.

[33] E. B. Fox and D. B. Dunson. Bayesian nonparametric covariance regression. *The Journal of Machine Learning Research*, 16(1):2501–2542, 2015.

[34] Y. Fukuda et al. Evidence for oscillation of atmospheric neutrinos. *Physical Review Letters*, 81(1562), 1998.

[35] Y. Gal. *Uncertainty in deep learning.* PhD thesis, PhD thesis, University of Cambridge, 2016.

[36] S. Ghosal, A. Van Der Vaart, et al. Convergence rates of posterior distributions for noniid observations. *The Annals of Statistics*, 35(1):192–223, 2007.

[37] P. D. Group. Neutrino Masses, Mixing and Oscillations. 2017.

[38] G. J. Hahn and W. Q. Meeker. *Statistical intervals: a guide for practitioners*, volume 92. John Wiley & Sons, 2011.

[39] D. A. Handwerker, V. Roopchansingh, J. Gonzalez-Castillo, and P. A. Bandettini. Periodic changes in fmri connectivity. *Neuroimage*, 63(3):1712–1719, 2012.

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

[41] M. D. Hoffman and A. Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

[42] A. Holbrook, A. Vandenberg-Rodes, N. Fortin, and B. Shahbaba. A bayesian supervised dual-dimensionality reduction model for simultaneous decoding of lfp and spike train signals. *Stat*, 6(1):53–67, 2017.

[43] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 985–990. IEEE, 2004.

[44] G. Kastner, S. Frühwirth-Schnatter, and H. F. Lopes. Efficient bayesian inference for multivariate factor stochastic volatility models. *Journal of Computational and Graphical Statistics*, 26(4):905–917, 2017.

[45] M. Khosla, K. Jamison, G. H. Ngo, A. Kuceyeski, and M. R. Sabuncu. Machine learning in resting-state fmri analysis. *arXiv preprint arXiv:1812.11477*, 2018.

[46] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[47] S. Lan, T. Bui-Thanh, M. Christie, and M. Girolami. Emulation of higher-order tensors in manifold monte carlo methods for bayesian inverse problems. *Journal of Computational Physics*, 308:81–101, 2016.

[48] S. Lan, A. Holbrook, N. J. Fortin, H. Ombao, and B. Shahbaba. Flexible bayesian dynamic modeling of covariance and correlation matrices. 2017.

[49] N. Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of machine learning research*, 6(Nov):1783–1816, 2005.

[50] N. D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *Advances in neural information processing systems*, pages 329–336, 2004.

[51] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[52] B. Leimkuhler and S. Reich. *Simulating hamiltonian dynamics*, volume 14. Cambridge university press, 2004.

[53] N. Leonardi and D. Van De Ville. On spurious and real fluctuations of dynamic functional connectivity during rest. *Neuroimage*, 104:430–436, 2015.

[54] M. A. Lindquist, Y. Xu, M. B. Nebel, and B. S. Caffo. Evaluating dynamic bivariate correlations in resting-state fmri: a comparison study and a new approach. *NeuroImage*, 101:531–546, 2014.

[55] H. F. Lopes and M. West. Bayesian model assessment in factor analysis. *Statistica Sinica*, 14(1):41–68, 2004.

[56] S. Mandt, M. D. Hoffman, and D. M. Blei. Stochastic gradient descent as approximate bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907, 2017.

[57] J. Močkus. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

[58] G. Motta and H. Ombao. Evolutionary factor analysis of replicated time series. *Biometrics*, 68(3):825–836, 2012.

[59] R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[60] R. M. Neal et al. Slice sampling. *The annals of statistics*, 31(3):705–767, 2003.

[61] R. M. Neal et al. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2:113–162, 2011.

[62] S. F. Nielsen, K. H. Madsen, R. Røge, M. N. Schmidt, and M. Mørup. Nonparametric modeling of dynamic functional connectivity in fmri data. *arXiv preprint arXiv:1601.00496*, 2016.

[63] H. Ombao, R. Von Sachs, and W. Guo. Slex analysis of multivariate nonstationary time series. *Journal of the American Statistical Association*, 100(470):519–531, 2005.

[64] H. Ombao, M. Fiecas, C.-M. Ting, and Y. F. Low. Statistical models for brain signals with properties that evolve across trials. *NeuroImage*, 180:609–618, 2018.

[65] F. Pedregosa, G. Varoquaux, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[66] B. Pontecorvo. Mesonium and antimesonium. *Zhur. Eksptl'. i Teoret. Fiz.*, 33, 1957.

[67] R. Prado and M. West. *Time series: modeling, computation, and inference*. Chapman and Hall/CRC, 2010.

[68] M. G. Preti, T. A. Bolton, and D. Van De Ville. The dynamic functional connectome: State-of-the-art and perspectives. *Neuroimage*, 160:41–54, 2017.

[69] C. E. Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[70] C. E. Rasmussen, J. Bernardo, M. Bayarri, J. Berger, A. Dawid, D. Heckerman, A. Smith, and M. West. Gaussian processes to speed up hybrid monte carlo for expensive bayesian integrals. In *Bayesian Statistics 7*, pages 651–659, 2003.

[71] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[72] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[73] S. B. Samdin, C.-M. Ting, and H. Ombao. Detecting state changes in community structure of functional brain networks using a markov-switching stochastic block model. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, pages 1483–1487. IEEE, 2019.

[74] Y. Smirnov. The MSW effect and Matter Effects in Neutrino Oscillations. *Phys.Scripta T121 (2005) 57-64*, 2004.

[75] E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264, 2006.

[76] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[77] A. Sousa, N. Buchanan, S. Calvez, P. Ding, D. Doyle, A. Himmel, B. Holzman, J. Kowalkowski, A. Norman, and T. Peterka. Implementation of Feldman-Cousins corrections and oscillation calculations in the HPC environment for the NOvA Experiment. In *Proceedings of the 23rd International Conference on Computing in High-Energy and Nuclear Physics (CHEP 2018), Sofia, Bulgaria, July 9-13, 2018*, 2019. In press.

[78] S. Terada, Y. Sakurai, H. Nakahara, and S. Fujisawa. Temporal and rate coding for discrete event sequences in the hippocampus. *Neuron*, 94(6):1248–1262, 2017.

[79] C.-M. Ting, H. Ombao, S. B. Samdin, and S.-H. Salleh. Estimating dynamic connectivity states in fmri using regime-switching factor models. *IEEE transactions on medical imaging*, 37(4):1011–1023, 2018.

[80] A. W. van der Vaart, J. H. van Zanten, et al. Rates of contraction of posterior distributions based on gaussian process priors. *The Annals of Statistics*, 36(3):1435–1463, 2008.

[81] A. W. van der Vaart, J. H. van Zanten, et al. Reproducing kernel hilbert spaces of gaussian priors. In *Pushing the limits of contemporary statistics: contributions in honor of Jayanta K. Ghosh*, pages 200–222. Institute of Mathematical Statistics, 2008.

[82] Y. Wang, C.-M. Ting, and H. Ombao. Modeling effective connectivity in high-dimensional cortical source signals. *IEEE Journal of Selected Topics in Signal Processing*, 10(7):1315–1325, 2016.

[83] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.

[84] A. Wilson and H. Nickisch. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International Conference on Machine Learning*, pages 1775–1784, 2015.

[85] C. Zhang, B. Shahbaba, and H. Zhao. Hamiltonian monte carlo acceleration using surrogate functions with random bases. *Statistics and Computing*, pages 1–18, 2015.

[86] H. Zhu, Y. Chen, J. G. Ibrahim, Y. Li, C. Hall, and W. Lin. Intrinsic regression models for positive-definite matrices with applications to diffusion tensor imaging. *Journal of the American Statistical Association*, 104(487):1203–1212, 2009.