Statistical and Computational Strategies for quantifying uncertainty in deep probabilistic models with applications

by

Jurijs Nazarovs

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

Doctor of Philosophy

(Statistics)

at the

UNIVERSITY OF WISCONSIN–MADISON

2023

Date of final oral examination: 04/06/2023

The dissertation is approved by the following members of the Oral Committee:

Keith Levin, Assistant Professor, Statistics
Sameer Deshpande, Assistant Professor, Statistics
Vivak Patel, Assistant Professor, Statistics
Michael A. Newton, Professor, Statistics
Vikas Singh (Adviser), Professor, Statistics

Acknowledgments

Getting PhD is not easy, but as we know, there is no shortcut to becoming the Hokage. While it is true that a PhD is primarily an individual effort, it is still takes a village to obtain one.

I would like to thank my advisor, Vikas, for guiding me in developing my research ideas to publications, financial support, as well as for helping me to understand my life goals. I am grateful to Ronak for our numerous discussions and for his significant contribution to my first paper. The night before submission and the moment with caffeine in the soda shall stay with me forever. I would also like to thank Eric, Xingjian, Rudra, Sourav, Songwong, and Akshay for their contributions to my research, and for the enjoyable conversations that we shared.

To my internship mentors, thank you for the valuable industry experience that you provided me with during those fun summers. This was an essential aspect of my PhD journey.

A special shout-out goes to my stunt buddies for a great time. While they were not involved in my academic research life, they were a big part of me getting to this moment and defending my PhD. As the saying goes, "wheelies healing feelies."

I would like to thank my dearest friends Dimon and Vanyok, for their support throughout the PhD. Our chats helped me going! Thanks to Andrew, Lyosha, Mitya, Maxim, Vlad, Sasha, and Kryl for keeping in touch with me throughout these years. I really appreciate it!

And the last, I am deeply grateful to my parents Alla and Nikolay (I know you are proud of me, guys) and my wife Bowen. Bowen, thank you for being with me during the toughest and the saddest moments of my PhD, for the lively discussions on my research, and for making me happy! You were my biggest source of motivation and happiness throughout my PhD, and who knows how it would turn without you. Thank you, Bowen :)

Contents

Chapter 1

Introduction

The development of AI in recent years has proven to be a powerful tool in transforming the way we live and work. From autonomous vehicles to virtual assistants and beyond, AI has proven its ability to meaningfully improve our daily lives. Despite the seemingly magical nature of AI, it is the result of the tireless work and dedication of experts in fields such as computer science, mathematics, and statistics. The field of AI continues to evolve, driven by advancements in machine learning algorithms and the growing access to vast amount of data.

Contemporary AI models overwhelmingly rely on Deep Neural Network models. Originally, Deep Neural Networks were motivated by the idea of modelling the neuron architecture of the brain (McCulloch and Pitts, 1943), but have come long way since then. In recent years, Deep Learning has achieved impressive results on a variety of tasks, ranging from computer vision (Wang and Gupta, 2016; Ramachandran et al., 2019) to natural language understanding (represented either as a written text or in audio formats) (Devlin et al., 2018; Sarzynska-Wawer et al., 2021). A confluence of ideas from disparate disciplines has enabled DL research to move beyond academic research, to applications in daily life. For instance, DL has been utilized in search recommendation systems to provide more personalized and relevant results for users. Computer vision applications of DL has been applied in various industries such as in healthcare for tumor detection, in retail to analyze product images and categorize them based on features such as color, size, and shape, and in transportation/logistic to solve problems of traffic management and accelerating in development autonomous driving. The integration of DL into these diverse fields has led to improved efficiency, accuracy, and overall decision-making.

One of the key reasons behind rapid progress in DL especially since 2012, is a large fusion of concepts from different areas of science and simultaneous development in the form of mature software tools. The DL literature has vastly benefited from numerical

optimization, as well as deployment of software engineering principles and advances in computer architecture. In addition, many more mature concepts from other fields greatly improved the predictive power, generalization and practical usefulness of NN . For example, non-linear optimization solvers such as momentum and stochastic gradient descent (SGD) have been crucial for development of neural networks and deep learning, as they allow for efficient and effective optimization of the network parameters. Methods from signal processing, such as convolutions and its variants have been widely adopted in deep learning, particularly in computer vision. Convolution neural networks (CNNs) use convolution operators as a building block to extract meaningful features from images and other data, which are used to make predictions, such as image classification, object detection, segmentation, and image generation. Another more recent example of adaptation of the stable idea is Differential Equations (DE). (Chen et al., 2018b) demonstrated the practicality of generalizing common DL models with DE. The utilization of Fourier transformation in DL can offer a significant speed up in training, *e.g.*, (Pratt et al., 2017; Gashler and Ashmore, 2014; Xu et al., 2019). And of course, the incorporation of the statistical theory and corresponding methods in DL is getting more popular, while further development is coming. For example, Bayesian deep learning (Farquhar et al., 2020; Ghosh and Doshi-Velez, 2017), which combines Bayesian statistics with DL, allows for the quantification of uncertainty in predictions made by DL models. Another example is the use of Gaussian processes in DL (Hyun et al., 2016), which can be used to model the function represented by the DL model and make predictions with confidence intervals.

Statistics, including probability theory, is a far more mature as a field than DL. For example, the well-known least squares method was published by (Legendre, 1806), while initial results of the statistical development goes back to the 18-th century. Methods from statistics are ubiquitous in applications to social science (Aron and Aron, 1999), physics (Barlow, 1993), astronomy (Trumpler and Weaver, 1953), healthcare (Scott and Mazhindu, 2014), clinical trials (Frison and Pocock, 1992) and other important directions of science. The main pillar of statistics include regression/classification of the data, model's parameters and density estimation, and hypothesis testing. While classical methods of statistics for regression and classification tasks are very well understood from the theoretical perspective, in reality, applying it to high dimensional data, like images, leads to a number of numerical issues and unsatisfactory performance. One remedy is to perform different types of manual and automatic feature extraction and selection. On the other hand, methods of DL and Machine Learning (ML) are based on deriving automatic feature representation, and provide significantly more power to perform regression/classification in computer vision and other high dimensional scenarios. Given this drastic difference in power between ML

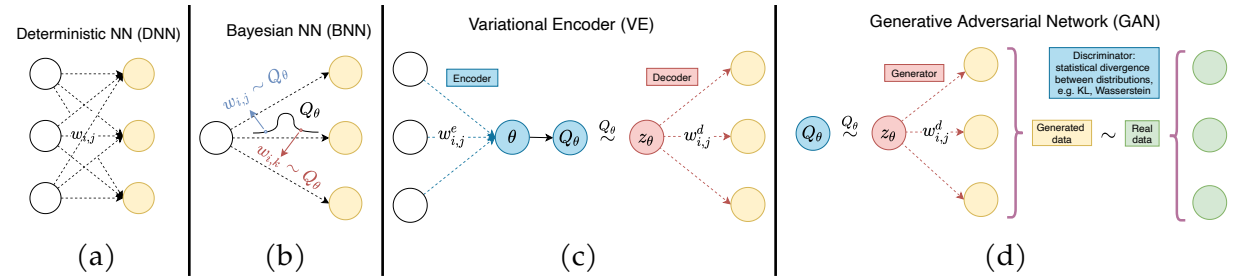


Figure 1.1: Different statistical concepts incorporated in Deep Learning Models.

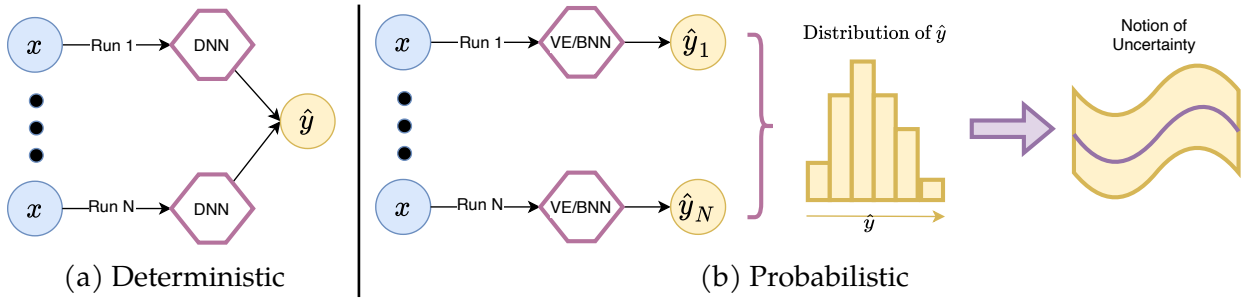


Figure 1.2: Difference between Deterministic and Probabilistic models in deriving the notion of uncertainty.

and statistical methods, what additional role can statistics play? There are vital concepts from statistics, which can be immensely beneficial to DL – not only to inform mathematically correct formulations but also for various practical benefits. For example, different ways to account for the uncertainty of the model/data (Chatfield, 1995; Meyer et al., 1986; Kruschke and Liddell, 2018) or sampling from underlying distribution (Escobar and West, 1995; Lu and Wong, 2008). Ideas from Bayesian statistics allow us to consider distribution of estimates of model's parameters, rather than point estimates, and incorporate prior information. Statistical techniques, like Gumbel-softmax approximation are useful for working with discrete variable (Bai et al., 2020). In addition, Stein gradient estimator is used to directly estimates the score function of the implicitly defined distribution (Li and Turner, 2017). However, marrying statistics with DL requires not only adaptation of concepts, but significant development on computational side.

To understand the different ways in which statistical theory can be integrated into computational and implementation sides of Deep Learning, we will consider several foundational probabilistic models commonly utilized in Deep Learning, Machine Learning, and Computer Vision. By evaluating the advantages they offer compared to a conventional deterministic Neural Network, we can better appreciate the role of statistical theory in Deep Learning. Before we go into probabilistic modeling, we start our discussion with typical deterministic network.

Example 1.1. *The Deterministic Neural Network shown in Figure 1.1a, is a typical model of DL, where all weights of the NN, $w_{i,j}$, are considered to be learned parameters. During training, the optimization scheme minimizes the deterministic loss,* i.e., *the one which does not include any notion of distribution* (e.g., *expected value*)*, like the cross-entropy or the ℓ_2 loss. Notice that a trained model has fixed parameters, and thus running the same trained model on the same input at test time, generates the same output, see Figure 1.2a. While such a behavior is acceptable if we are only concerned about the prediction generated by the model as is the case in many applications, it is less useful, however, if we want to evaluate the uncertainty of the prediction. The notion of uncertainty can be useful in downstream tasks for calibration and general understanding how wrong a model is.*

Next, we cover several probabilistic models of DL, where statistical ideas may bring benefits.

Example 1.2. *We start with the Bayesian Neural Networks (BNN), presented in Figure 1.1b. In contrast to deterministic networks, the main idea behind BNNs is to consider all weights $w_{i,j}$ as being samples from a random distribution. In this case, learnable parameters of the model are not weights $w_{i,j}$ themself, but a parameter θ, which defines the distribution of weights. To infer the value of the parameters θ, the probabilistic loss has to be optimized. Let us denote the observed data as (x, y), where x is an input to the network, and y is a corresponding response. One possible loss function is the likelihood $p(y|x)$. However, since all weights of a BNN are considered to be random variables, we can rewrite it as $p(y|x) = \int_w p(y, W|x)dW = \int_W p(y|W, x)p(W|x)dW$.*

There are at least two benefits of modelling Neural Networks in this way: (1) it allows us to get the uncertainty of the prediction by generating a set of $y|x, w_i$ and computing the posterior mean/variance and credible intervals, (Figure 1.2b) and (2) posterior predictive mean $y|x$ can be considered as an ensemble of an infinite number of deterministic NN (by varying the samples of weights w^i), which might be helpful in regimes of limited data as a remedy for over-fitting.

Example 1.3. *The next example we can consider is another core probabilistic model of Deep Learning, Variational Encoder (VE), presented in Figure 1.1c. While this is also a probabilistic model like BNN with a similar loss, the concept of VE is different from BNN. The core idea is to use deterministic encoder (with fixed weights $w_{i,j}$ to encode data into parameters θ of distribution Q_θ and then sample latent representations of data, z_θ from Q_θ. Then we apply a deterministic decoder to get a desirable output. Note that if an output of decoder corresponds to an input, VE is called a Variational Auto Encoder (VAE).*

Usually the concept of VE is used in a setup of VAE, which allows generating new samples from the input space, by sampling z_θ and decoding it to the domain of input.

Example 1.4. *Consider the Generative Adversarial Network (GAN), shown in Figure 1.1d. The idea is similar to VE that a deterministic models are used on top of the underlying distribution Q_θ. However, the implementation is conceptually different. In GAN, we initially sample z from Q_θ and then 1) use deterministic generator to simulate data similar to real data, 2) use deterministic discriminator to classify whether generated data is similar to real data or not. With the specific implementation of a discriminator, GAN optimization corresponds to minimization of statistical notions of similarities, like Wasserstein distance (Arjovsky et al., 2017), between two distributions: generated and real data.*

Example 1.5. *Deep Neural Networks as Gaussian Processes. The exact equivalence between infinitely wide deep fully connected NN and GPs was established and derived in (Lee et al., 2017). Such a correspondence enables exact Bayesian inference for infinite width NN on regression tasks, by estimating the corresponding GP. It is done by constructing the covariance matrix of the GP. This line of work continued in (Novak et al., 2018), where the authors established similar relation between Bayesian Deep Convolutional NN and GPs.*

Example 1.6. *Another interesting idea of incorporating techniques from statistics relates to introduction of Neural Tangent Kernel (NTK), (Jacot et al., 2018). NTK is a kernel that characterizes the evolution of a deep artificial neural networks during their training via gradient descent. Assuming the infinite-width of Neural Networks, NTK converges to an explicit limiting kernel and it stays constant during training. While the assumption of infinite-width of NN is strong, NTK allows analyzing NN using the tools of kernel methods. This results in simple and concise statements about the predictions, training dynamics, generalization, and loss surfaces of neural networks. For instance, it ensures that ANNs with a wide enough layers will converge to a global minimum when trained to minimize an empirical loss.*

Example 1.7. *Score Matching Models. In previous examples we mainly talk about different ways to incorporate probability theory in an architecture of NN. Alternatively, statistics can also help in defining a training procedure. The traditional approach to train a neural network is based on maximizing an explicit or implicit model log-likelihood or a lower bound of its log-likelihood (ELBO). In contrast, the game theory based approach to achieve a training objective is used successfully in GANs. In addition, new models based on score matching techniques are rising in popularity (Song and Ermon, 2019; Song et al., 2020). Traditionally in statistics, the score is defined as the gradient of logarithmic probability density with respect to the data distribution parameter. In these models, the score network is trained to approximate a real score of the probability. Later, the score network is used to generate data.*

While we discussed only a few cases of combining statistical theory with DL, clearly, these are core concepts and can be applied in different settings: from non-temporal/static vision problems, where the main goal is to perform classification, generation or segmentation, to temporal processes.

Building upon the previous discussion on probabilistic models in DL, we continue our exploration a little further and demonstrate the potential benefits that can be derived from this integration. We also aim to provide insights into how different statistical concepts can be combined to further enhance the performance of ANNs.

Example 1.8. *Computational aspects of Bayesian NN. BNNs are a prominent tool to model uncertainty on the prediction and mitigate overfitting (with small datasets). Similar to Bayesian statistics, BNNs provide the freedom to select an appropriate prior and approximate posterior on weights, which fit our data/goal the best. Unfortunately, not all choices of distributions lead to convenient computational implementation (say, in a closed form), and often training BNNs relies on sampling techniques (e.g., Monte Carlo or MC), which in combination with numerical optimization can lead to substantial burden on GPU memory and pose computational challenges.*

Example 1.9. *Temporal modeling with Differential Equations (DE). Differential Equations is a well-known modelling choice for describing continuous process. However, the application of classical DE to model temporal process in high-dimensional vision data is limited. To address this, new model, Neural ODE, was proposed as the fusion of DE and DL, which achieves the desired modelling properties, depending on whether we focus on predicting the next step (e.g., video frame), interpolating steps between observed samples, or generate a completely new trajectory. The main limitation of DE lies in a restriction that given initial point of DE and transition function, DE solution results in a deterministic trajectory. However, in reality this is not always the case and randomness of trajectories for the fixed initial point should be accounted for. To overcome this issue, we can rely on statistical concept of Mixed Effects modelling. It allows to generate the distribution of trajectories, given a fixed initial point. This directly leads to benefits of statistical modelling, such as sampling, uncertainty estimation and other statistical inference tools.*

Example 1.10. *Multimodaling is an emerging line of research, which utilizes different sources of data to make a prediction. For example, in Visual Question Answering (VQA) systems, several components such as image and text are utilized to make a correct answer for the question per image. By incorporating uncertainty in the model might improve robustness of VQA to language variations and image manipulations. Another example is trajectory generation based on natural language expressions for agents, e.g. robots, which can benefit from posterior modeling of trajectories using stochastic methods of statistics.*

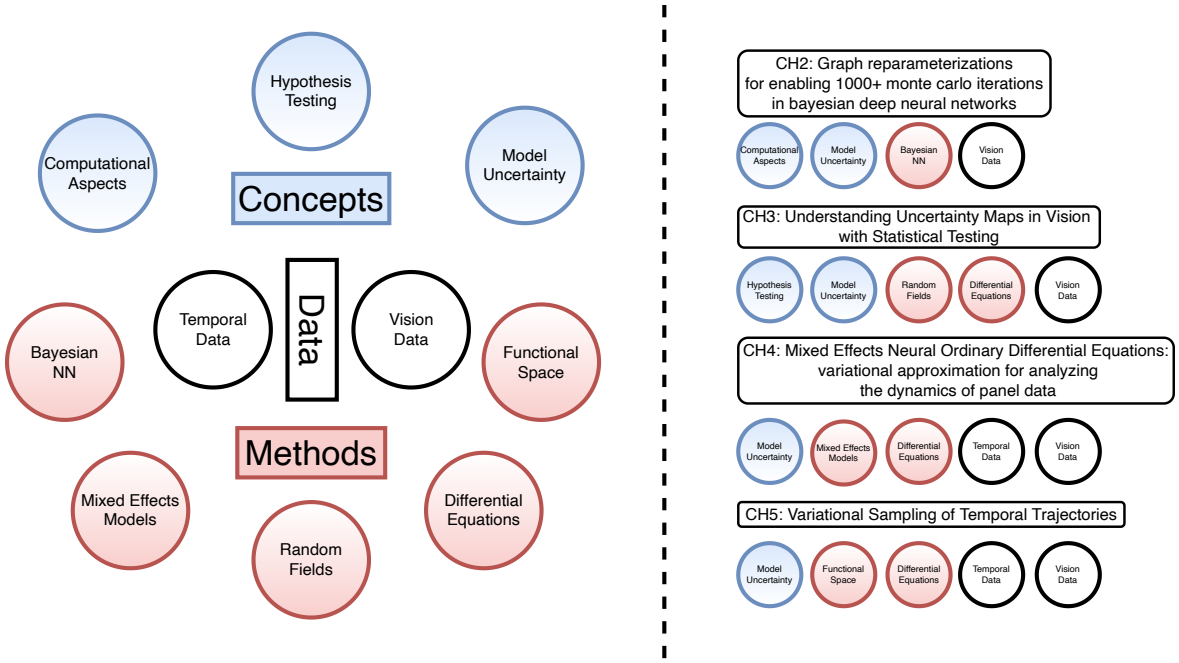


Figure 1.3: Work presented in this thesis mainly addresses computational aspects of probabilistic models, hypothesis testing in DL settings, and DL model uncertainty. This work primarily focuses on a vision data in temporal settings, and build on concepts from Bayesian statistics, Mixed Effects models, Differential Equations and Functional Space.

Contribution and scope of the Thesis

In this thesis we will explore how some ideas from statistics can help deep learning to improve their prediction by incorporating uncertainty aware models, and most importantly how it can be done in efficient ways for application in computer vision problems. The overall scope of the thesis and underlying core ideas are represented in the Figure 1.3. All problems we study will be linked with statistical theory, including but not limited to Bayesian ideas, Mixed Effects models, Stochastic Processes/Differential Equations, Random Fields and more. Furthermore, some of the work utilizes Differential Equations, both in an explicit manner to handle modeling of trajectories of temporal data and in an implicit way as a bridge to link results between statistics and DL. Next we concisely summarize problems studied in this thesis.

Reparameterization of computation graphs in Bayesian Neural Networks as a remedy for memory explosion

We first study computational aspects of the corresponding chapter primarily addresses computational aspects of BNN and presents a new solution to the memory needs in large

BNN approximation. This approach enables the modeling of uncertainty, with applications in computer vision data.

Our motivation stems from the fact that a broad range of literature on Bayesian Neural Networks support an idea of exploring different kind of priors and approximate posteriors for weights of NN. However, in a commonly used training setup, called Variational inference, it is necessary to compute a Kullback–Leibler divergence between selected distributions. Unfortunately, obtaining closed form solution for KL divergence is not trivial for any set of distributions and often Monte Carlo approximation is required. The problem is that, with increasing number of Monte Carlo (MC) samples, there is a drastic increase in memory requirement for the model, which might results in a significant memory footprint, either GPU or RAM, and failure in a model training.

Our proposed solution is based on the observation that training methods of DL relies on gradient computations at each iterate, and standard implementations make use of automatic differentiation based on computation graphs (Griewank, 2012). *Computation graphs* are directed acyclic graphs, where nodes are the inputs/outputs and edges are the operations (we will describe this in more detail in the Chapter 2). If there is a single input to an operation that requires a gradient, its output will also require a gradient. The main reason for memory explosion is an increase in the size of computation graph, which is used for optimization of the MC procedure. We realize that for many distributions it is possible to reparameterize the computation graph in a way, that its size is independent of number of MC samples. Namely, we propose the definitions of the 'size' of computation graph and show with theoretical justifications for which distributions such reparameterization can be done. Experimentally we show that our technique allows not only saving in memory, which makes training of BNN with MC procedure possible, independently of number of MC samples, but significant acceleration in training time and improvement in accuracy.

Understanding Uncertainty Maps in Vision with Statistical Testing

Next we revisit two statistical concepts: hypothesis testing and model uncertainty, and utilize theory and methods from Random Fields and Differential equations with a focus on improving the analysis of certain types of computer vision datasets.

Our motivation stems from the fact that quantitative descriptions of confidence intervals and uncertainties of the predictions of a statistical model are needed in a broad spectrum of applications in vision and machine learning. Mechanisms that enable this for deep neural network (DNN) models are slowly becoming available, and occasionally, even being integrated within production systems. The literature, however, is sparse in terms

of how to perform statistical tests with the uncertainties produced by these algorithms (for overparameterized models). For two models with a similar accuracy profile, is the former model's uncertainty behavior better in a statistically significant sense compared to the second model? For high resolution images, performing hypothesis tests to generate meaningful actionable information (say, at a user specified significance level $\alpha = 0.05$) is difficult but needed in both mission critical settings and elsewhere. In this paper, specifically for uncertainties defined on images, we show how revisiting known results from Random Field theory (RFT) when paired with DNN tools (to get around some computational hurdles) leads to efficient frameworks that can provide hypothesis test capabilities for uncertainty maps from models used in many vision tasks. This capability is not otherwise currently available. We show via many different experiments the viability of the proposed framework.

Mixed Effects Neural Ordinary Differential Equations as a variational approximation for analyzing the dynamics of panel data

Next we study how model uncertainty can also benefit DL models operating on temporal (or panel) datasets in vision and medical imaging.

Panel data involving longitudinal measurements of the same set of participants taken over multiple time points is common in studies to understand childhood development and disease modeling. Deep hybrid models that marry the predictive power of neural networks with physical simulators such as differential equations, are starting to drive advances in such applications. The task of modeling not just the observations but the hidden dynamics that are captured by the measurements poses interesting statistical/computational questions. We propose a probabilistic model called ME-NODE to incorporate (fixed + random) mixed effects for analyzing such panel data. We show that our model can be derived using smooth approximations of SDEs provided by the Wong-Zakai theorem. We then derive Evidence Based Lower Bounds for ME-NODE, and develop (efficient) training algorithms using MC based sampling methods and numerical ODE solvers. We demonstrate ME-NODE's utility on tasks spanning the spectrum from simulations and toy data to real longitudinal 3D imaging data from an Alzheimer's disease (AD) study, and study its performance in terms of accuracy of reconstruction for interpolation, uncertainty estimates and personalized prediction.

Variational Sampling of Temporal Trajectories

Finally, we study modeling uncertainty in DL models utilizing techniques from functional analysis and differential equations. As with previous chapters, the primary focus are datasets in vision and medical imaging that are temporal/longitudinal in nature.

Recall that a deterministic temporal process can be determined by its trajectory, an element in the product space of (a) initial condition $z_0 \in \mathcal{Z}$ and (b) transition function $f : (\mathcal{Z}, \mathcal{T}) \to \mathcal{Z}$ often influenced by the control of the underlying dynamical system. Existing methods often model the transition function as a differential equation or as a recurrent neural network. Despite their effectiveness in predicting future measurements, few results have successfully established a method for sampling and statistical inference of trajectories using neural networks, partially due to constraints in the parameterization. In this chapter, we introduce a mechanism to learn the distribution of trajectories by parameterizing the transition function f explicitly as an element in a function space. Our framework allows efficient synthesis of novel trajectories, while also directly providing a convenient tool for inference, i.e., uncertainty estimation, likelihood evaluations and out of distribution detection for abnormal trajectories. These capabilities can have significant implications for downstream tasks, e.g., simulation and evaluation for reinforcement learning.

Outline

In chapters 2-5 we describe the three problems we study and the proposed solution in detail. In Chapter 2 we demonstrate the family of distributions which can benefit from computation graph reparameterization to save memory consumption of the model with other computational perks. Later, in Chapter 3, we show how combination of Random Field statistical theory with newly introduced tool Warping Neural ODE, lead to a great framework for evaluating the significant regions of uncertainty, generated by probabilistic AI models, like Bayesian Neural Networks. Then, in Chapter 4, we present the new framework based on combination of statistical concepts, mixed effect, and an DL tool, Neural ODE, to provide an ability for analyzing the dynamics of panel data in vision, including medical experiments. Finally, in Chapter 5, we present the novel way of sampling trajectories, by considering the parameterization of functional space of Neural Networks, which are used to model trajectories of Neural ODE. We conclude our work and discuss the future directions in Chapter 6.

Chapter 2

Background

This chapter provides a concise overview of the central ideas related to deep learning, temporal modelling, and statistical concepts. These concepts play a critical role in supporting our discussions regarding the algorithms in the later chapters. We begin by contrasting two fundamental problems in machine learning: classification and regression.

2.1 Classification versus Regression

Classification and regression are two fundamental problems in fields of statistics, machine learning and deep learning. The main difference between classification and regression tasks is the type of output they seek to predict.

The main goal of classification problems is predicting categorical or discrete output variables, such as class labels in image classification problems or object detection, or sentiment labels in natural language processing. Given a set of predictive features, $\mathbf{X}$, a classification model aims to learn a mapping function, $f_\theta(\mathbf{X})$, from the predictive features to a specific class label, $\mathbf{y}$. Here, $\mathbf{y}$ is an element from a discrete set of labels and θ is a set of parameters, which defines f_θ, and the term "link function" is frequently used to refer to f_θ. Mathematically, such a relation can be represented as:

$$f_\theta(\mathbf{X}) = \mathbf{y}$$

Regression problems, on the other hand, seek to predict continuous output variables, such as stock prices, value of the pixel in the image or values of multiple sensors. In other words, given a set of predictive features, $\mathbf{X}$, a regression model aims to learn a mapping function, $f_\theta(\mathbf{X})$, from the predictive features to a continuous value, $\mathbf{y}$.

The objective is to learn a set of parameters θ that minimizes the error or loss between

the predicted and true target values. For example, in classification problems, the commonly used losses are binary cross-entropy for binary classification and cross-entropy for multi-class classification. In regression problems, the most commonly used are the ℓ_2 losses and ℓ_1. These losses provide a quantitative measure of the deviation between the predicted and true target values, and the model is trained to minimize this deviation. In the next section we are going to introduce deep learning and corresponding approaches to solve both classification and regression problems.

2.2 Deep Learning

Throughout all chapters, deep learning serves as the cornerstone of our novel frameworks. Consider a general description of a classification or regression model:

$$f_\theta(\mathbf{X}) = \mathbf{y},$$

where f_θ is our model, $\mathbf{X}$ is an input and $\mathbf{y}$ is modelled output. The primary difference between deep learning and shallow learning is the description of the model f_θ, which is usually referred in DL as architecture of the model. Model f_θ can be a complex, hierarchical composition of other functions, which can be thought of as layers (or building blocks) in a neural network. The greater the number of compositions or layers, the more capable the model is in learning complex relationships within the data. Deep learning models are composed of multiple layers, which consist of various mathematical operations and functions that are combined and composed to form the building blocks of the architecture. This is the main reason why DL models are capable of learning complex relationships, while shallow learning models have a few layers (and sometime even only a single layer) and are best suited for simpler problems. Classical models of statistics and machine learning such as linear regression, regression trees and support vector machines often can be thought of as shallow learning and represent models as a single-layer models.

In this section, we aim to provide an overview of widely used key concepts and techniques of Deep Learning, including two main building blocks of architecture of DL: Fully Connected Networks (FCN) and Convolutional Neural Networks (CNN). CNN are particularly useful for processing visual data, and have proven to be a powerful tool in computer vision tasks such as image classification, object detection, and semantic segmentation.

In addition to these vital computational blocks of DL, we will also discuss the importance of probabilistic models in Deep Learning and various methods for handling temporal information. By the end of this section, readers should have a solid foundation in the key

concepts and techniques that are widely used in Deep Learning research and applications and have a better understanding of this work.

Fully Connected and Convolution Neural Networks

Fully Connected Networks. Fully Connected Network (FCN), also known as Multi-Layer Perceptrons (MLP), is one of the most popular neural network architecture. It consists of multiple dense layers of neurons and was originally introduced in 1958 by the psychologist Frank Rosenblatt, (Rosenblatt, 1958). We present an architecture of general FC on Figure 2.1. The main feature of a FCN is that each neuron in a layer is connected to every neuron in the previous and next layers, hence providing a complete connection between the input and output layers. The mathematical formulation of a FCN can be expressed using the notation of $f_\theta = f_\theta^1 \cdot ... \cdot f_\theta^L$ as a composition of functions or layers, where f_θ^l is the output of l-th layer of the network. The l-th layer is described as

$$f_\theta^l = g^l(\mathbf{W}^l f_\theta^{l-1} + \mathbf{b}^l),$$

where f_θ^{l-1} is an output of $l-1$-th layer (where $f^0 = \mathbf{X}$ is the input vector), $\mathbf{W}^l$ is a weight matrix of l-th layer of size $n_{out} \times n_{in}$, $\mathbf{b}^l$ is a bias vector of l-th layer of size n_{out}, and g^l is the activation function of a choice, e.g., ReLU, sigmoid, and other. The weight matrix $\mathbf{W}^l$ defines the linear transformation of the input features, and the bias vector $\mathbf{b}^l$ shifts the output of the linear transformation. The activation function g^l introduces non-linearity into the model, allowing it to learn complex relationships in the data.

Inspired originally by the idea of simulating the way brain neurons work, FCN has become a main computational block of current neural networks which are used in many complex applications, including image classification and speech recognition. The success of FCN can be attributed to its capability to capture non-linear connections between inputs and outputs. FCNs are particularly useful when the input data is not easily separable into distinct categories, as they can learn complex representations of the data and provide continuous predictions.

Despite the fact that the FC layer is a powerful building block in modern deep learning models, it is not well-suited for computer vision settings that involve imaging data, due to the localized nature of images. Instead, convolution operations are commonly used to address this issue. In the following discussion, we will explore Convolutional Neural Networks (CNNs), which have been specifically designed to work with imaging data.

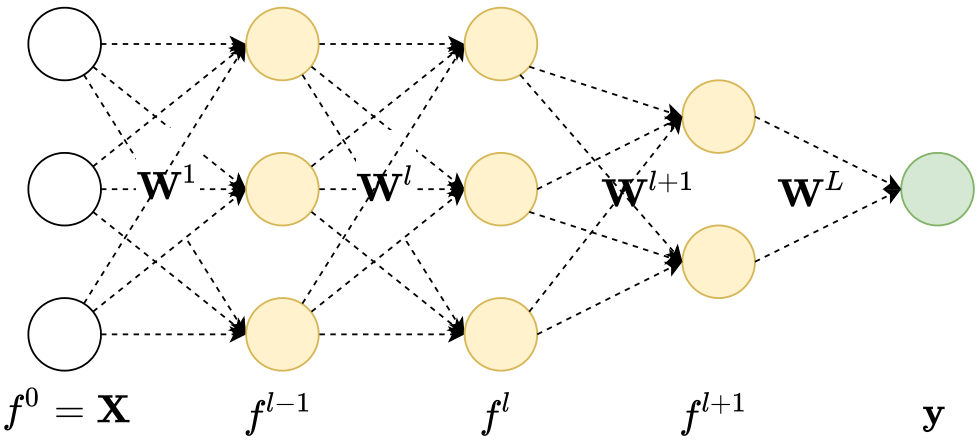


Figure 2.1: The diagram represents a general architecture of a Fully Connected Neural Network (FCN). Each neuron in a layer is connected to every neuron in the previous and next layers and provides a complete connection between the input and output layers.

Convolution Neural Networks. Convolutional Neural Network (CNN) is a class of neural networks, which are designed to capture the spatial relation within the image and are very popular in applications in computer vision tasks, such as image classification, segmentation and generation. The main component of CNN is a convolutional layers and represented on Figure 2.2, which uses convolution kernels to filter and extract features from raw input images. The mathematical formulation of a CNN is similar to FCN that $f_\theta = f_\theta^1 \cdot ... \cdot f_\theta^L$ is a composition of functions or layers, where f_θ^l is the output of l-th layer of the network. The difference lies in the definition of a single convolutional layer, which is described as

$$f_\theta^l = g^l(\mathbf{W}^l * f_\theta^{l-1} + \mathbf{b}^l),$$

where f_θ^{l-1} is an output of $l-1$-th layer (where $f^0 = \mathbf{X}$ is the input image with c_{in} channels), $\mathbf{W}^l$ is a set of convolutions kernels (filters) of size $k \times k \times c_{in} \times c_{out}$, where k is the filter size, c_{in} is the number of input channels, and c_{out} is the number of output channels, and $*$ denotes the convolution operation. Similar to FC, $\mathbf{b}^l$ is a bias and g^l is the activation function of a choice.

In addition to convolution layers, it is common to use a pooling operation, e.g., choosing the highest value from the resulting convolution, which decreases the dimensionality of feature maps and reduces over-fitting. CNNs yield excellent performance on many computer vision benchmarks and are widely used in industry for various applications, including image classification, object detection, semantic segmentation, video analysis and others. Next, we will describe another important Deep learning model: encoder-decoder architecture.

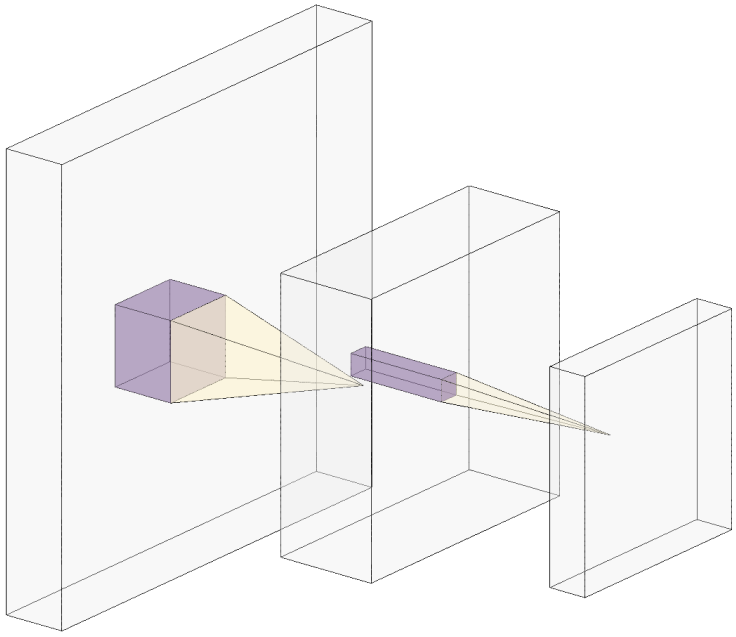


Figure 2.2: Figure represents a general architecture of a Convolutional Neural Network (CNN). The main component of CNN is a convolutional layer, which uses convolution kernels to extract local features from image data.

Encoder-decoder architecture.

The encoder-decoder architecture has been heavily utilized in different variations of DL models. Popular examples of encoder-decoder architectures are auto-encoder, used for dimension reduction and image denoising (Vincent et al., 2008) and variational auto-encoder, primarily used for image generation (Kingma and Welling, 2013). The architecture consists of two main components and represented on Figure 2.3: the encoder and the decoder. The encoder takes the input data and transforms it into a compact representation, known as a latent representation or bottleneck, which is designed to capture features of the data. The encoder concept can be thought of as a cascade of PCA-like projections with non-linear transformations (Dai et al., 2018) that reduce the dimensionality of the input data and preserve the most relevant information. The decoder then takes this compact representation and uses it to generate the final output, typically by using upsampling operations to increase the spatial resolution of the representation. The main feature which differentiates an auto-encoder from an encoder-decoder is the output of decoder. In an auto-encoder, the decoder is designed to output the same value, as the input, while in general encoder-decoder architecture that might not be a case.

The encoder-decoder architecture has proven to be very effective in various applications due to its ability to capture essential features of the data through its latent representation.

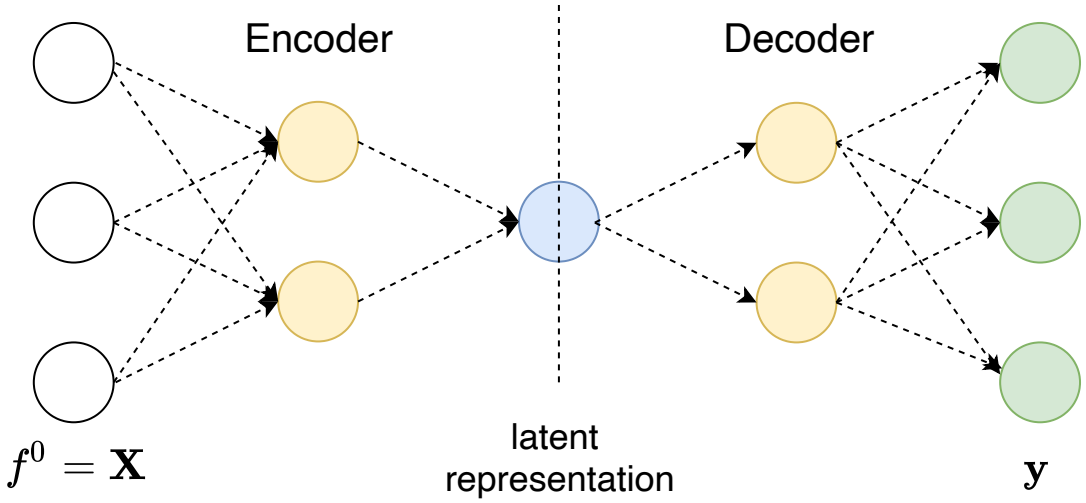


Figure 2.3: Figure represents an Encoder-Decoder architecture. The main idea of encoder is to transform an input data it into a compact representation, known as a latent representation or bottleneck, which is designed to capture features of the data. The decoder then generates the final output $\mathbf{y}$ based on encoded compact representation. In case of Auto-Encoder we have $\mathbf{y} = \mathbf{X}$.

Furthermore, the decoder's ability to generate high-resolution outputs using the compact representation from the encoder makes the encoder-decoder architecture well-suited for vision tasks, such as increasing the resolution of images (Mao et al., 2016).

Note that encoder-decoder is a methodology, rather than a specific model, and lately, many variations of the encoder-decoder architecture have been proposed. Ideas proposed in the literature include the use of attention mechanisms to enhance the information flow between the encoder and decoder (Vaswani et al., 2017), and the integration of additional components such as skip connections (Cao et al., 2023), which allow the model to retain fine-grained information from the input data. These modifications have further improved the performance of encoder-decoder models in various applications, demonstrating their continued versatility and effectiveness. The concept of encoder-decoder, with a focus on auto-encoders, is heavily used in the chapters 4, 5, 6, and expanded to accommodate temporal data.

Deterministic versus probabilistic DL models

Although we discussed probabilistic models earlier (in chapter 1), it is crucial to distinguish between deterministic and probabilistic models, as the latter play a significant role in this thesis. Deterministic and probabilistic DL models are two broad categories of machine learning models that differ in their approach to modeling the relationship between inputs

and outputs.

Deterministic models, as the name suggests, do not contain stochasticity in the model and produce a fixed output for a given input. Namely, if we run the same network N times on the same input, we get exactly the same output for each run (see Figure 1.2a). Usually deterministic model provides a higher accuracy, compared to its probabilistic version, given the same amount of data and architecture (Wenzel et al., 2020). However, deterministic models tend to overfit on a smaller dataset (Lu et al., 2019) and are not able to provide any notion on uncertainty of the prediction. On the other hand, probabilistic models are designed to tackle these issues and prevent over-fitting on a smaller data accounting for model's prediction uncertainty. Instead of producing a fixed output for the same input, probabilistic models produce a probability distribution over the possible outputs, Figure 1.2b. These models are trained to predict the probability distribution over the output, rather than a point estimate, allowing them to capture the uncertainty and variability inherent in the data. Because probabilistic models allow us to sample from the distribution of outcomes, they are commonly used as generative models. Examples include image (Kingma et al., 2015), music (Briot et al., 2017), and text generation (Radford et al., 2018).

There are different types of probabilistic models, but several popular ones include Bayesian Neural Networks, Variational Encoder (VE) or Variational Auto-Encoder (VAE) and Generative Adversarial Networks, which we previously introduced in chapter 1. One of the common assumptions of probabilistic models in DL is mean field assumption of independence of random variables. For example, if we talk about VE, then mean field assumption of independence can be stated as

$$q\left(z_1, \cdots, z_m\right) = \prod_{j=1}^{m} q\left(z_j\right),$$

where z_i is variational latent representation. If we refer to BNN, then

$$p(W) = \prod_{d=1}^{D} p^d\left(W^d\right),$$

where W is a set of weights of NN. The concept of probabilistic models is central to this dissertation and is used in every chapter but in distinct ways. For example, in chapter 3 we focus on Bayesian Neural Networks, while in other chapters we introduce new methods of probabilistic modelling, motivated by concept of mixture models from statistics and stochastic differential equations.

2.3 Temporal modeling

Temporal data, which tracks changes in events and processes over time, is common in a broad range of applications. For instance, in healthcare, temporal data can describe disease progression and aid healthcare providers in making better-informed treatment decisions and predicting patient outcomes. In transportation, temporal modelling helps analyze traffic patterns, predict travel times, and optimize routes to improve efficiency and reduce congestion.

Modelling temporal processes has been an area of extensive research in statistics for a long time, and has often been approached from the perspective of autoregressive models (Lai et al., 2018), functional (Yosida, 2012) and longitudinal analysis (Fitzmaurice et al., 2012). Autoregressive models examine the dependence between observations at different time points, allowing for the prediction of future observations based on past data. Similarly, longitudinal data analysis studies changes in a response over time but accounting for variation within and between groups of individuals. Functional analysis is a little different in that it focuses on modeling the underlying functional relationship between a response and time. These approaches offer unique insights into the nature of temporal processes and continue to be actively developed in the field of statistics, machine and deep learning.

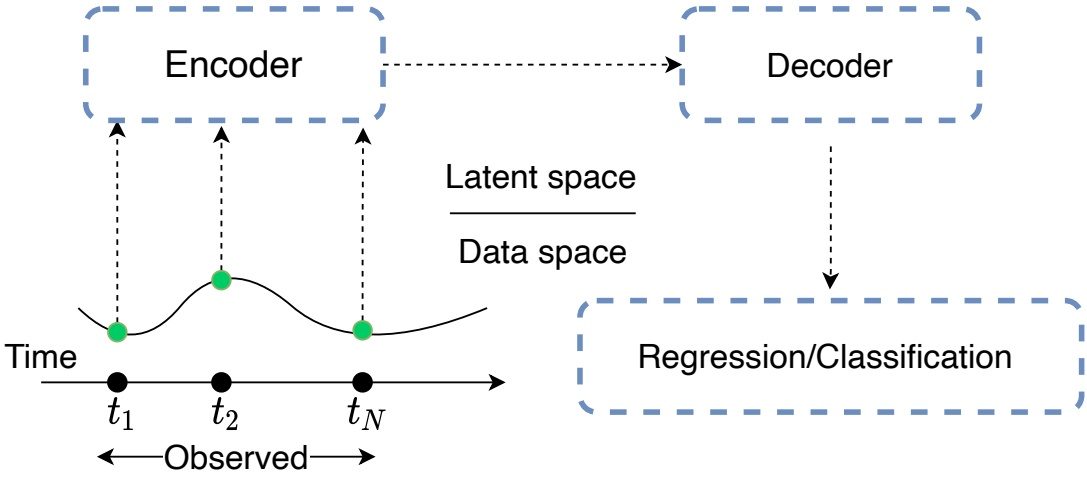


Figure 2.4: When work with temporal data, the model can be considered as a combination of two modules: an Encoder, which processes the observed data, and a decoder, which conduct either classification or regression task, based on output from the Encoder.

Conceptually, to account for temporal information, the modelling of temporal data can be described by two main components, presented in Figure 2.4: (**a**) an Encoder module (Figure 2.4, Left) which processes the observed data and maps it to the latent space (a latent representation of the data), and (**b**) a Decoder which is designed to perform classification

or regression (interpolation or extrapolation), depending on the problem (Figure 2.4, Right).

Given a summary of the temporal information, there are wide variety of tools from statistics, machine learning and deep neural networks evaluate whether to use a classification or regression setup. For example, one can use a simple logistic regression (Menard, 2002), Bayesian Networks (Heckerman, 1997), k-mean clustering (Hartigan and Wong, 1979), regression trees (De'ath and Fabricius, 2000), fully connected networks (Ruck et al., 1990) and many other models. However, the important question is how to summarize the temporal information? Given a temporal data x^i for data sample i, two possible simple solutions include: **(a)** using the function of choice, aggregate p-dimensional vectors x_t^i through all time steps $t \in T$ as a p-dimensional vector $\tilde{x}^i$, e.g. using maximum or a sum, or **(b)** append time step t as a new variable, increasing the dimension of x_t^i from p to $p+1$.

Examples of other classical statistical methods which are designed for analysis of temporal information are ARMA (Brockwell, 2001), Gaussian Process (Williams and Rasmussen, 1995), and Vector Autoregressive Models (Lütkepohl, 1985) for multivariate time-series. However, despite the theoretical rationale behind these methods, they may not always offer the capacity needed to model the complicated temporal processes in the data, especially when we must also deal with spatial information in temporal imaging data of computer vision problems. In contrast, over-parameterized models from deep learning can provide rich representations of temporal processes. Examples include deep recurrent models range from the traditional Recurrent Neural Network (RNN) (Cho et al., 2014) and the Long Short Term Memory (LSTM) model (Huang et al., 2015), as well as more advanced attention-transformer models (Vaswani et al., 2017). More recent work includes type of continuous models, such as the Neural Ordinary Differential Equations (Neural ODEs) (Chen et al., 2018b).

Neural ODE

This dissertation heavily relies on the Neural ODE, which forms a crucial component of our research framework presented in chapters 4, 5, and 6. Similar to Ordinary Differential Equations (ODE), the Neural ODE (Chen et al., 2018b) is designed to model the behavior of a dynamical system as a continuous flow in time, rather than discrete events as seen in RNN, LSTM and attention, Figure 2.5 (Left). The main idea behind Neural ODEs is to represent the transition function $f(x, t)$ of an ODE $\dot{x}_t = f(x, t)$ as a neural network with learned weights/parameters. This approach provide several benefits, for example: Neural ODE is suitable for modeling of complex, non-linear temporal processes that are difficult to

capture using traditional temporal models of statistics, and it naturally deals with irregular time steps, which are depicted in Figure 2.5 (Right). Irregularity of temporal data means that the gap between time steps $\Delta t_j = t_{j+1} - t_j$ does not need to be the the same for different choices of j. While the RNN, LSTM and attention based models ignore the information about the time gap in their basic setup, and engineering techniques are necessary to address it, the continuous nature of the Neural ODE handles it in its core formulation (Rubanova et al., 2019).

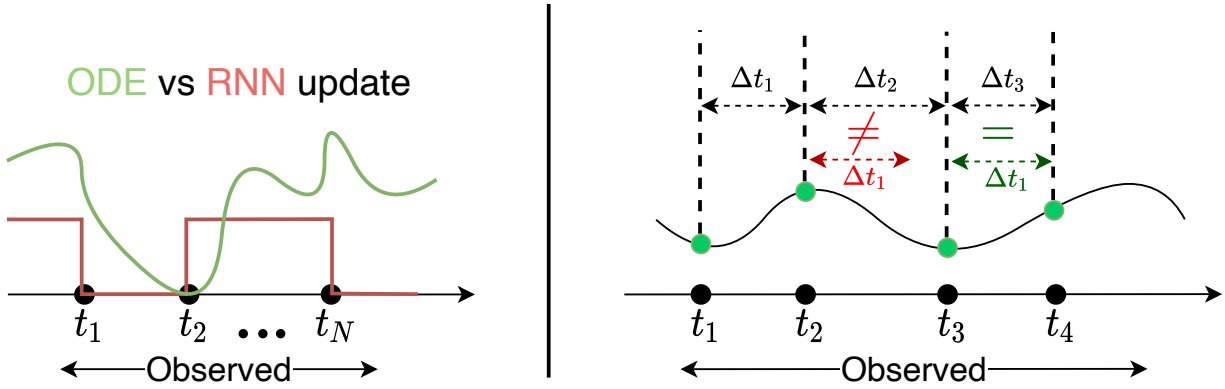


Figure 2.5: **Left:** One main difference between the RNN (red) and the Neural ODE (green) is due to the representation of time in the temporal process. Between observed time steps, the hidden state of the RNN is either constant or not defined, while for the Neural ODE model, the state develops in a continuous fashion corresponding to the underlying ODE. **Right:** This figure depicts the representation of irregular time steps in temporal analysis. $\Delta t_j = t_{j+1} - t_j$, and Δt_1 is not necessarily equal to Δt_2.

2.4 Statistical Concepts

In this section, we review several concepts we will use throughout the dissertation, starting with hypothesis tests.

Hypothesis test is a statistical procedure, which consists of four main parts: **(1)** Null hypothesis H_0 and an Alternative hypothesis H_A, **(2)** test statistics F, **(3)** critical value α, which controls the *probability of Type-1 error*, i.e., $\mathbb{P}(\text{reject } H_0 | H_0 \text{ is true}) \leqslant \alpha$ and **(4)** a threshold value $u := u(\alpha)$, which defines the rejection region. While test statistics, hypotheses and critical value are design choices, the threshold u has to be derived such that the **p-value** $\mathbb{P}(F \geqslant u | H_0) = \alpha$. Via hypothesis tests, we can assess whether there is evidence to reject the null H_0 at a certain level of confidence α. Usually, H_0 states that there is no difference (say, from zero or between two groups), while H_A states that there is a

difference. The decision is based on checking whether the observed test statistics F^{obs}, falls in the rejection region, defined by the threshold u.

Family wise error rate (FWER). Recall that the rejection region is selected based on α, which controls $\mathbb{P}$ (Type-1 error) of a single test, i.e., $\mathbb{P}$ (reject $H_0|H_0$) $\leqslant \alpha$. However, assume that we conduct $N = 100$ tests, e.g., the same test for different pixels, with $\alpha = 0.05$. Then the $\mathbb{P}$ (reject at least one $H_0|H_0$ is true) $= 1 - (1 - \alpha)^N = 0.994$, and on average 5 tests would be rejected purely by chance. For this reason, in multiple comparison testing, we do not want to control $\mathbb{P}$ (Type-1 error), but *FWER:* $\mathbb{P}$ (reject at least one $H_0|H_0$). Note that the FWER for 1 test equals to $\mathbb{P}$ (Type-1 error).

Random Fields

The formulation described later in chapter 4 will be based entirely on a mature concept in statistics known as random fields. Random fields are a class of stochastic processes that describe a set of random variables defined on a spatial or spatio-temporal domain. In other words, they model a collection of random variables, each associated with a particular point in space or time, and describe how the variables are correlated with each other. Random fields have wide-ranging applications in various fields, including astronomy and brain imaging analysis. A large body of work is dedicated to studying random fields and their application to brain image analysis, for example: (Adler et al., 2007).

There are different type of random fields with their own properties. Some of the examples are Chi-squared, F, T and most well-known, Gaussian random fields. Gaussian random fields are widely used because of their mathematically tractable distributions and availability of closed form solutions for different applications, including hypothesis testing to control FWER, given dependencies among data. To understand better the idea behind Gaussian Random Field, we provide a technical definition below.

Gaussian Random Field (GRF). GRF is a family of functions $Z : S \to \mathbb{R}$, where for all finite $k \geqslant 1$ and $\{s_1, \ldots, s_k\} \subset S$, the collection of random variables $\{Z(s_1), \ldots, Z(s_k)\}$ has a multivariate Gaussian distribution. GRFs are parameterized by a mean function $\mu(s) = \mathbb{E}\{f(s)\}$ and covariance function $C(s,t) = \mathbb{E}\{(f(s) - \mu(s))(f(t) - \mu(t))\}$. A *UGRF* is a special Gaussian RF with mean zero, variance 1, and $Var\left(\dot{Z}(s)\right) = I$. An *Isotropic GRF* is a special case where the covariance function $C(s,t)$ depends only on the Euclidean distances $\|s - t\|_2$.

Gaussian related RFs (Adler et al., 2007) or GRRF is another broad class of random fields $F = f(Z)$, obtained as *functions* of GRF. For example, a Chi-squared RF with d degrees of freedom $\chi^2_d(t) = \sum_{j=1}^{d} Z_j^2(t)$ is a GRRF.

Chapter 3

Graph Reparameterizations for Enabling 1000+ Monte Carlo Iterations in Bayesian Deep Neural Networks

Motivated by the need to provide measures of uncertainty in the deployment of deep neural networks in mission critical and medical applications, there has been a strong recent interest in deep Bayesian learning. While deep Bayesian learning provides many methods to estimate posterior distributions, Variational Inference (VI) is a convenient choice for many problem settings (Blundell et al., 2015). Many libraries such as Tensorflow Probability (Dillon et al., 2017) are also now available that offer a rich set of features. In this chapter we study the excessive memory consumption problem, which occurs during Monte Carlo approximation of a KL term in the Variational Inference setup, common in Bayesian Neural Networks.

We start this paragraph with notation. Denote the observed data as (x, y), where x is an input to the network, and y is a corresponding response (in autoencoder settings we may have $y = x$). When using VI in Bayesian Neural Networks (BNNs), one considers all weights $W = (W^1, \ldots, W^D)$ as a random vector and approximates the true unknown posterior distribution $P(W|y, x)$ with an *approximate posterior* distribution Q_θ of *our choice*, which depends on learned parameters θ. Let $W_\theta = (W_\theta^1, \ldots, W_\theta^D)$ denote a random vector with a distribution Q_θ and pdf q_θ. VI seeks to find θ such that Q_θ is as close as possible to the real (unknown) posterior $P(W|y, x)$, accomplished by minimizing the KL divergence between Q_θ and $P(W|y, x)$. Given a prior pdf of weights p, along with a likelihood term $p(y|W, x)$, and a common *mean field* (as covered in Chapter 2) assumption of independence for W^d and W_θ^d, for $d \in 1, \ldots, D$, i.e. $p(W) = \prod_{d=1}^{D} p^d(W^d)$ and $q_\theta(W_\theta) = \prod_{d=1}^{D} q_\theta^d(W_\theta^d)$,

$$\theta^* = \arg\min_{\theta} \mathrm{KL}\left(q_\theta || p\right) - \mathbb{E}_{q_\theta}\left[\ln p(y|W, x)\right] \tag{3.1}$$

$$\mathrm{KL}\left(q_\theta || p\right) = \sum_{d=1}^{D} \mathbb{E}_{q_\theta^d}\left[\ln q_\theta^d(w)\right] - \mathbb{E}_{q_\theta^d}\left[\ln p^d(w)\right] \tag{3.2}$$

A key consideration in VI is the choice of prior p and the approximate posterior q_θ. This choice does not drastically change the computation of the likelihood term $p(y|W, x)$ which is influenced more by the problem and the complexity of the network instead of W (e.g., it is Gaussian for regression problems). But it strongly impacts the computation of KL term. For example, a common choice for p, and q_θ is Gaussian, which allows calculating (3.2) in a closed form. However, there is emerging evidence (Farquhar et al., 2020; Fortuin et al., 2020) that the Gaussian assumption may not work well on medium/large scale Bayesian NNs. (Farquhar et al., 2020) attributes this to the probability mass in high-dimensional Gaussian distributions concentrating in a narrow "soap-bubble" far from the mean. Choosing a correct distribution is an open problem (Ghosh and Doshi-Velez, 2017; Farquhar et al., 2020; McGregor et al., 2019; Krishnan et al., 2019), and unfortunately, more complex distributions frequently lack closed form solutions for (3.2).

Numerical approximations. When the integrals for these expectations cannot be solved in closed form, an approximation is used (Ranganath et al., 2014; Paisley et al., 2012; Miller et al., 2017). One strategy is Monte Carlo (MC) sampling, which gives an unbiased estimator with variance $O(\frac{1}{M})$ where M is number of samples. For a general function $g(\cdot)$ (in Section 3.3 we show how $g(\cdot)$ is related to the minimization loss of VI to train BNN):

$$\mathbb{E}_{q_\theta}\left[g(w)\right] = \int g(w) q_\theta(w) dw \approx \frac{1}{M} \sum_{i=1}^{M} g(w_i),$$
$$\text{where } w_i \sim Q_\theta. \tag{3.3}$$

Expected value terms in (3.2) can be estimated by applying the scheme in (3.3) and in fact, even if a closed form expression can be computed, with enough samples an MC approximation may perform similarly (Blundell et al., 2015). Unfortunately, MC procedures are costly, and may need many samples (i.e., iterations) for a good estimation as the model size grows: (Miller et al., 2017) shows this relationship for small networks, and demonstrates that using fewer samples leads to large variances in the approximation. In general, for deep BNNs, computation of both KL and expectation of log-likelihood requires numerical approximation with MC sampling, but for now, we will only focus on the KL

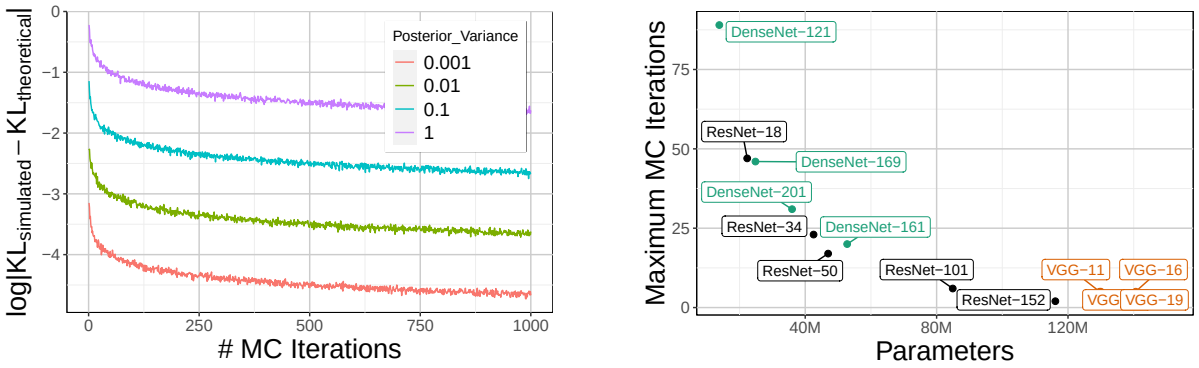


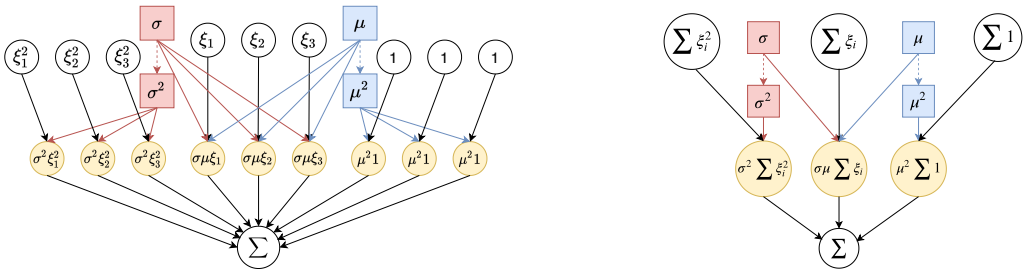
Figure 3.1: (**left**) Approximation error (log) of simulated KL for the single-parameter Bayesian Neural Network at different variance values of approximate posterior distribution, (**right**) Maximum number of feasible MC iterations required for training Bayesian versions of different neural networks on a single GPU .

term.

How does M **affect the** KL **approximation necessary for large scale VI?** Consider a standard Gaussian distribution for the approximate posterior q_θ and prior p for the weights of an arbitrary BNN, and also consider an MC approximation of the KL term in (3.2). In this case, we have a closed form solution for KL, which allows checking the approximation quality: the gap between the MC approximation $\widehat{\mathrm{KL}}$ and the closed form KL.

(**a**) Figure 3.1 (left) shows this gap for different variances of the approximate posterior for a BNN. While decreasing the variance of the posterior distribution indeed reduces the variance of an estimator, with such a small variance on weights, the model is essentially deterministic. Clearly by increasing M, we decrease the error. However, in current DNNs, increasing the number of MC iterations not only slows down computation, but severely limits GPU memory. (**b**) Figure 3.1 (right) presents the maximum number of iterations possible on a single GPU (Nvidia 2080 TI) with a direct implementation of MC approximation for Bayesian versions of popular DNN architectures: ResNet, DenseNet and VGG. Extrapolating Figure 3.1, we see clearly that Bayes versions of these networks will result in large variances. This raises the question: is there a way to increase the number of MC iterations for deep networks without sacrificing performance, memory, or time?

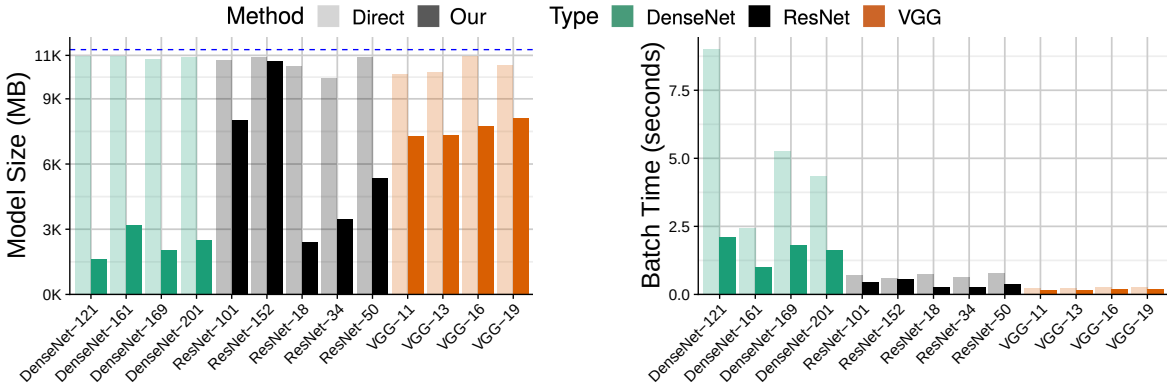
Contributions. This work makes three contributions. (**a**) We propose a new framework to construct an MC estimator for the KL term, by reparameterizing a corresponding computation graph to preserve its size regardless of number of MC samples, Figure 3.2. This significantly decreases GPU memory needs and improves runtime, Figure 3.3. Memory savings allow us to run up to 1000× more MC iterations on a single GPU, resulting in smaller variances of the MC estimators, improving both training convergence and final accuracy,



(a) AutoGrad implementation, $d_P = 9$ (b) $d_P = 3$

Figure 3.2: Two computation graphs of the same MC expression $\sum_{i=1}^{3}(\mu + \sigma\xi_i)^2$, with different parameterizations. Filled squares represent elements of the vector function $n(\theta)$, clear circles represent functions of auxiliary variables $t(\xi)$, yellow circles represent the Hadamard product $n(\theta) \circ t(\xi)$. Clearly, parameterization affects the size of the graph, and there exists parameterization (b) where the size is independent of the number of MC iterations M. **Note:** We slightly modify the computational graph presentation for space and clarity. Actual computation graphs from PyTorch convey the same message.

especially on subsets of data where the model is not confident. We show feasibility for popular architectures including ResNets (He et al., 2016b), DenseNets (Huang et al., 2017), VGG (Simonyan and Zisserman, 2014) and U-Net (Ronneberger et al., 2015) – strategies for successfully training Bayesian versions of many of these (deep) networks remain limited (Dusenberry et al., 2020). **(b)** From the user perspective, we provide a simple interface for implementing and estimating BNNs (Figure. 3.4). **(c)** On the technical side, we obtain a scheme under which we can determine whether our *reparameterization* can be applied. The result covers a broad class of distributions used in VI as an approximate posterior and



(a) (b)

Figure 3.3: For maximum possible number of MC iterations for a given model via the direct MC method, we show: **(a)** Model size (dashed blue line indicates GPU capacity, 11GB), **(b)** Training time. For some networks, our method occupies less than 25% of memory and is $5\times$ faster.

```
model = AlexNet(n_classes=10, n_channels=3,
                approx_post="Radial",
                kl_method="repar",
                n_mc_iter=1000)
```

Figure 3.4: Proposed MC reparameterization presented as an API. Only a minimal change in an existing programming interface is required to incorporate our method. See the appendix for details.

prior, Table 3.1. Inspired by the Pitman–Koopman–Darmois theorem (Koopman, 1936), we show that our method is effective when an exponential family is used as a prior on weights in deep BNNs estimated via VI, and the approximate posterior is modeled as location-scale or certain other distributions, expanding the range of distributions that can be used.

3.1 Related work

In addition to VI, the literature provides a broad range of ways to estimate posterior predictive distributions. Ensemble methods (Lakshminarayanan et al., 2017; Pearce et al., 2018; Newton et al., 2018) can be applied to common networks with minimal modifications; however, they require many forward passes, often similar in terms of space/time to a standard *gradient accumulation* schemes (we provide a PyTorch code snippet in Figure 3.5). Figure 3.6 provides experimental results showing that gradient accumulation is much slower. Other methods like Deterministic Variational Inference (Wu et al., 2019) and Probabilistic Backpropagation (Hernández-Lobato and Adams, 2015), improve over naïve MC implementations of VI, but often approximate the posterior of a neural network with a Gaussian distribution. However, (Farquhar et al., 2020) shows that a choice of Gaussian distribution for BNN leads to results being highly sensitivity to hyperparameters choices, in addition to other problems occurring during training. For this reason, a non-Gaussian distribution can be used as an approximate posterior in the traditional VI setup, but its lack of a closed form solution for the KL term ends up needing MC approximation. This is where our proposal offers value. Also, note some other issues that emerge in Deterministic Variational Inference and Probabilistic Backpropagation: (a) the methods need non-trivial modification of the network to perform a moment matching and (b) replacing the Gaussian assumption with another distribution requires new analytical solutions of closed forms. This is more complicated than a MC approximation.

Our proposal is distinct from other works that also target MC estimation in neural networks. For example, one may seek to derive new estimators with an explicit goal of

Sampling: $W(\theta, \xi)$	Approximate Posterior p.d.f. q_θ		Prior p.d.f. $p(w)$		
Scaling property family: $W(\theta, \xi) = \theta\xi$ and related – Corollary 3.4	Exponential(θ) Rayleigh(θ) Erlang(k, θ) Error(a, θ, c) Inverse-Gamma(k, θ)	Standard Wald(θ) Weibull(k, θ) Gamma(k, θ) Log-Gamma(k, θ)	Exponential Dirichlet Inverse-Gamma Log-normal Inverse-Gaussian	Standard Wald Chi-squared Gamma Error Normal	Rayleigh Pareto Erlang Weibull
Location-Scale: $W(\theta, \xi) = \mu + \sigma\xi$, $\theta = (\mu, \sigma)$	Normal(μ, σ) Logistic(μ, σ) Radial(μ, σ)	Laplace(μ, σ) Horseshoe(μ, σ)	Logistic	Exponential Laplace	Normal
			Normal variations, e.g., Horseshoe, Radial		
Corollary 3.7	Log-Normal(μ, σ)		Dirichlet	Pareto	

Table 3.1: Summary list of approximate posterior distributions q_θ and priors $p(w)$, which allows to define a parameterization tuple P for MC estimation, such that d_P is independent of M. For every cell in "Sampling: $W(\theta, \xi)$" we can select any combination of q_θ and $p(w)$. Reference: Radial (Farquhar et al., 2020), Horseshoe (Ghosh and Doshi-Velez, 2017).

variance reduction (e.g., (Miller et al., 2017)). Here, we do *not* obtain a new estimator replacing the MC procedure with a smaller variance procedure. Instead, we study a scheme that makes the computation graph mostly independent of the number of samples, and is applicable to ideas such as those in (Miller et al., 2017) as well.

3.2 Computation Graphs for MC iterations

Despite the ability to approximate the expectation in principle, the minimization in (3.1) via (3.3) is difficult for common architectures, and relies on gradient computations at each iterate. Standard implementations make use of automatic differentiation based on computation graphs (Griewank, 2012).

Computation graphs are directed acyclic graphs, where nodes are the inputs/outputs and edges are the operations. If there is a single input to an operation that requires a gradient, its output will also require a gradient. As noted in PyTorch manual (PyTorch,

```
optimizer.zero_grad()
for _ in range(n_mc_iter):
    output = model(inputs)
    loss = computeLoss(output, targets)
    loss.backward()
optimizer.step()
```

Figure 3.5: PyTorch implementation of "gradient accumulation" technique, a standard method to collect gradient from several different forward passes. Memory consumption is equivalent to 1 forward pass, but time complexity is proportional to number of forward passes.

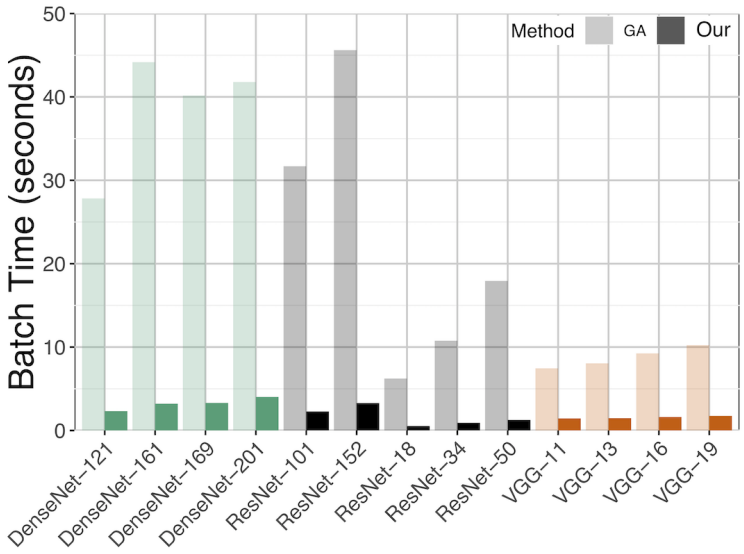


Figure 3.6: MC sampling is significantly slower in existing neural network libraries incorporating Gradient Accumulation (GA). In contrast, our proposed MC reparameterization reduces the compute time up to 14× for some networks.

2023), a backward computation is never performed for subgraphs where no nodes require gradients. This allows us to replace such a subgraph with one output node and to define the size of the computation graph as the minimal number of nodes necessary to perform backpropagation: the number of nodes which require gradients. Modern neural networks lead to graphs where the number of nodes range from a few hundred to millions. To define the size of a graph, accounting for the probabilistic nature of the MC approximation, we propose the following construction.

Definition 3.1. *Consider w as sampled based on a parameter θ and an ancillary random variable ξ, i.e., $w = W(\theta, \xi)$. If there exist functions G, n, and t such that a function $F(w_1, \ldots, w_n)$ can be expressed as $G(n(\theta) \circ t(\xi_1, \ldots, \xi_n))$, then we say $P := (G, n, t)$ is a **parameterization tuple** for the function F, where $\circ$ is the Hadamard product. Let d_P be the dimension of $n \circ t$, corresponding to the number of nodes requiring gradients with respect to θ.*

To demonstrate the application of Definition 3.1, as an example, consider the computation graph for the MC approximation of the function $g(w) = w^2$ in (3.3) and given one weight $W_\theta \sim N(\mu, \sigma^2)$. Applying the reparameterization trick: $W_\theta = \mu + \sigma\xi$, $\xi \sim N(0, 1)$, the Python form is,

```
for i in range(M):
    # sample 1 observation from N(0, 1)
    sample = sampler_normal.sample()
    w = mu * 1 + sigma * sample
```

```
    loss += w^2 / M
```

The computation graph, a function of both the parameters $\theta = (\mu, \sigma)$ and of the auxiliary samples ξ_1, ξ_2, and ξ_3, generated by PyTorch/AutoGrad for $\mathbf{M = 3}$ **iterations** of this loop is shown in Figure 3.2a. According to Def. 3.1, $d_P = 9$ and

$$\begin{aligned} n(\theta) &= (\mu^2, 2\sigma\mu, \sigma^2, \mu^2, 2\sigma\mu, \sigma^2, \mu^2, 2\sigma\mu, \sigma^2), \\ t(\xi) &= (1, \xi_1, \xi_1^2, 1, \xi_2, \xi_2^2, 1, \xi_3, \xi_3^2), \\ G(n(\theta) \circ t(\xi)) &= n_1(\theta)t_1(\xi) + \cdots + n_9(\theta)t_9(\xi) \end{aligned}$$

Naïvely, the graph size grows linearly $O(M)$ with the number of MC iterations, as in the direct implementation (Figure 3.2a). For Bayesian VI in DNNs, this is a problem. We need to perform MC approximations of KL terms at every layer. Also, (Miller et al., 2017) shows that iterating over a large number of samples M might be important for convergence. This constrains model sizes given limited hardware resources. One might suspect that a "for" loop is a poor way to evaluate this expectation and instead the expression should be vectorized. Indeed, creating a vector of size M and summing it will clearly help runtime. But the loop does *not* change the computation graph; all trainable parameters maintain the same corresponding connections to samples, and rapidly exhaust memory.

But graphs for the same function can be constructed differently (see Figure 3.2b). For the right parameterization tuple P, we can achieve $d_P = 3$. This leads us to,

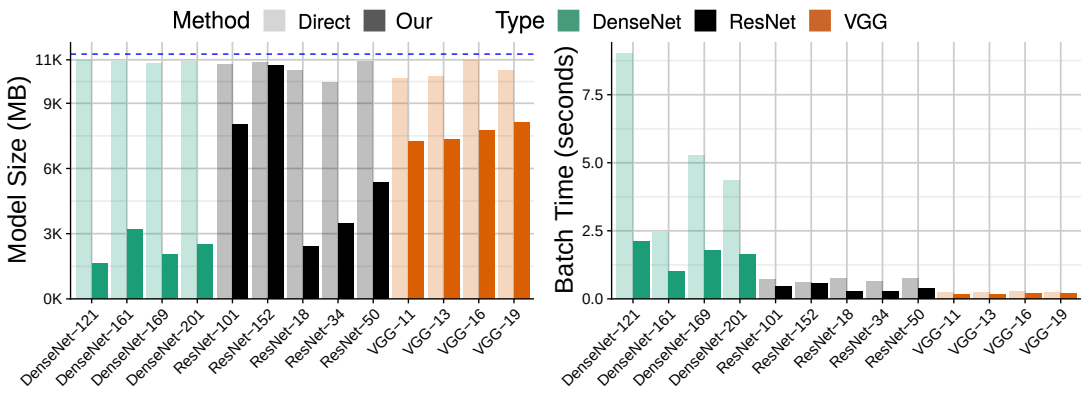
Remark 3.2. *For computation graph of MC approximation* $\sum_{i=1}^{M} g(w_i)$ *and specific* g*, there exists a parameterization tuple* $P = (G, n, t)$*, such that* d_P *is independent of* M*.*

For which class of distributions Q_θ and functions $g(\cdot)$ can we always construct reparameterizations of the MC estimation (3.3), maintaining the size of the computation graph d_P as **independent** of number of iterations M? We explore this in the next section.

3.3 MC reparameterization enables feasible training

Our approach is partly inspired by a vast literature on known distributional families and their use within VI. For example, in VI, commonly one chooses distributions that fall within *exponential families* (e.g., Gaussian, Laplace, Horseshoe). With this assumption on the prior, we can express

$$p(w; \zeta) = h(w)\exp(\eta(\zeta)'T(w) - A(\zeta)) \tag{3.4}$$



(a) (b)

Figure 3.7: For maximum possible number of MC iterations for a given model via the direct MC method, we show: **(a)** Model size (dashed blue line indicates GPU capacity, 11GB), **(b)** Training time. For some networks, our method occupies less than 25% of memory and 5 times faster.

where ζ is a parameter defining w. The sufficient statistics $T(w)$ and natural parameters $\eta(\zeta)$ completely define a specific distribution.

Relevance of PKD theorem. While the foregoing discussion links our approach to well-known statistical concepts, it does not directly yield our proposed scheme. To see this, recall that the Pitman-Koopman-Darmois (PKD) theorem states that for exponential families in (3.4), there exist sufficient statistics such that the number of scalar components does not increase as the sample size increases. However, in approximating (3.2) with MC, we need to compute not only terms containing the sufficient statistics $T(w)$ *but also* $\frac{1}{M}\sum_{i=1}^{M}\log h(w_i)$. Regardless, even though the PKD result cannot be applied directly in our case, it still suggests considering members of the exponential family as candidates for Q_θ. We derive technical results for the forms of $W(\theta,\xi)$ and $g(\cdot)$, where the graph size is not affected by MC sampling.

To approximate KL in (3.2), we need to compute MC estimation (3.3) for $g(w) = \log q_\theta(w)$ (or $\log p(w)$). Assume that the factorization form (3.4) of distributions $q_\theta^d(w)$ (and similarly $p^d(w)$) and recall that the weights of NN are parameterized as $w \sim W(\theta,\xi)$. Then, $\mathbb{E}_\theta \log q_\theta(w)$ is approximated as:

$$\frac{1}{M}\sum_{i=1}\{\log h(w(\theta,\xi_i)) + \eta(\theta)'T(w(\theta,\xi_i))\} - A(\theta) \tag{3.5}$$

To keep the graph size agnostic of M, we must handle the initial two terms in (3.5). Checking distributions from Tab. 3.1, our work reduces to functions of the form w^k and $\log(w)$.

The following text presents valuable theorems, and their proofs can be located in Appendix A. Denote S as the dimension of θ, i.e., number of parameters defining the

distribution Q_θ. For example, for the Exponential(λ): $S = 1$ and $\theta = (\lambda)$; for Gaussian(μ, σ): $S = 2$ and $\theta = (\mu, \sigma)$. Denote k to be a positive integer.

Theorem 3.3. *If $W(\theta, \xi) = \eta(\theta)T(\xi)$ ($S = 1$), then there exists a parameterization tuple P with $d_P = 1$ for the following functions $g(w)$: w^k, $\log(w)$, and $\frac{1}{w^k}$.*

Corollary 3.4. *If $W(\theta, \xi)' = f(W(\theta, \xi))$ and $W(\theta, \xi) = \eta(\theta)T(\eta)$, then Theorem 3.3 applies to $W(\theta, \xi)'$ and $g(W(\theta, \xi)')$ if $g(w'(w))$ is: w^k, $\log(w)$, and $\frac{1}{w^k}$.*

Theorem 3.5. *If $W(\theta, \xi) = \sum_{s=1}^{S} \eta_s(\theta)T_s(\xi)$, and $g(w) = w^k$, then there exists a parameterization tuple P with*

$$d_P = \binom{k + S - 1}{S - 1}. \tag{3.6}$$

Remark 3.6. *As long as $d_P < M$, it is possible to create a computation graph of a smaller size by reparameterization, compared to a direct implementation of the MC approximation. Note that for a small M it is still possible for a parameterization tuple to generate a graph larger than a naïve implementation. For example, consider $\sum_{i=1}^{M}(\mu + \sigma\xi)^2$. When $M = 1$, the naïve construction would have $d_P = 2$, ($n = (\mu, \sigma), t = (1, \xi)$, while a "nicer" tuple may have $d_P = 3$ independent of M ($n = (\mu^2, 2\mu\sigma, \sigma^2), t = (1, \xi, \xi^2)$).*

Corollary 3.7. *If $W(\theta, \xi)' = f(W(\theta, \xi))$ and $W(\theta, \xi) = \sum_{s=1}^{S} \eta_s(\theta)T_s(\eta)$, where $S \geqslant 2$, then Theorem 3.5 applies to $W(\theta, \xi)'$ and $g(W(\theta, \xi)')$ if $g(w'(w)) = w^k$.*

Relevance of results: (1) Thm. 3.3 can be applied when $W(\theta, \xi)$ represents a distribution with scaling property: any positive real constant times a random variable having this distribution comes from the same distributional family. (2) Thm. 3.5 can be applied, when $W(\theta, \xi)$ is a member of the location-scale family. (3) Corollaries 3.4 and 3.7 are useful when random variables can be presented as a transformation of other distributions, e.g. LogNormal(μ, σ^2) can be generated as $\exp(N(\mu, \sigma^2))$. Table 3.1 summarizes the choice of q_θ and p for Bayesian VI, which lead to the computation graph size d_P being independent of M in MC estimation.

Although Theorem 3.5 does not suggest that there are no nice parameterization tuples for the case where $g(w) = \log w, 1/w^k$, empirically we did not find tuples that allow for d_P to be independent of M. But it is interesting to consider an approximation which does allow for this independence.

Taylor Approximated Monte Carlo

Our results extend to the generic polynomial case where $g(w) = p_K(w)$, an arbitrary polynomial of degree K:

Corollary 3.8. *If* $W = \sum_{s=1}^{S} \eta_s(\theta) T_s(\xi)$, *and* $g(w) = p_K(w)$, *then there exists parameterization tuple* P, *such that for any* M *iterations*

$$d_P = \binom{K+S}{S} - 1. \tag{3.7}$$

So, can we find a parameterization tuple for any $g(w)$ that we can approximate via a polynomial Taylor expansion?

Theorem 3.9. *Let* $W = \sum_{s=1}^{S} \eta_s(\theta) T_s(X)$, $S \geqslant 2$. *If an approximation of* $g(w)$ *uses* K *Taylor terms, then Corollary 3.8 applies.*

Practical implications. If one is limited to running a maximum number of MC iterations M_{max}, such an approximation of $g(w)$ allows a tradeoff between accuracy of running just M_{max} iterations for the real $g(w)$ versus approximating $g(w)$ with $K(M_{max})$ terms and running $M \gg M_{max}$ iterations instead, since d_p is independent of M. This strategy may not work for approximating non-polynomial functions, and is a "fall-back" that could be used for arbitrary distributions.

Example 3.10. *Let* $W = \mu + \sigma\xi \implies S = 2$ *and* $g(w) = \log w$, *then*

$$\sum_{i=1}^{M} g(w_i) = \sum_{i=1}^{M} \log(w_i) \approx \sum_{i=1}^{M} \sum_{k=0}^{K} \frac{1}{k!} (\mu + \sigma\xi_i - 1)^k$$

where we take the Taylor expansion of $\log(w)$ *around* $w = 1$. *This is clearly a polynomial function of order* K, *and applying Corollary 3.8, we have* $d_P = \frac{1}{2}(K+1)(K+2) - 1$ *interactions. For example, if one is able to run just 9 direct MC iterations, it is possible to approximate* $g(w)$ *with* $K = 3$ *terms, allowing any number of MC iterations* M.

Applying reparameterization in Bayesian NN

Recall that training a Bayesian NN via VI requires the approximation of both the KL term and expected value of log-likelihood in (3.2). While it is clear how MC reparameterization can be applied to approximate the KL term, what can we say about the likelihood term? In general, this term cannot be handled by the ideas described so far although some practical strategies are possible.

Usually, estimating the expectation of the likelihood term is based on (Kingma and Welling, 2013; Kingma et al., 2015), where for every data item b in the minibatch (of size B), one MC sample is selected, which results in B different samples – in fact, (Kingma and Welling, 2013) suggests that the number of samples per data item can be set to one if the minibatch size is "large enough" which we will discuss more shortly. If a large B is feasible, then our scheme might not contribute substantially in estimating the likelihood term. However, if B is small, then our scheme can provide some empirical benefits, described next.

Let (x, y) be the observed data and (x_b, y_b) be the observed b-th data point. Let w correspond to the weights of NN with L layers. We can use $w(l, \cdot)$ to index the weights of layer l. Note that we can draw a unique sample of w for each data point b which we denote as $w(l, b)$. When M samples are drawn for b, these will be indexed by $w_i(l, b)$ for $i = 1, \cdots, M$. Notice that $w_1(l, b)$ is the same as $w(l, b)$. In the forward pass, u_b^l is the output for the b-th data point and u_b^L is the output of the last layer for data point x_b.

Observation 3.11 (Likelihood form in BNN)**.** *Consider the following form for regression and classification tasks,*
Regression: *Consider* $y \sim N(u^L, \widehat{\sigma})$*, where* $\widehat{\sigma}$ *is fixed. Then,*

$$\begin{aligned}\log p\left(y_b \mid w, x_b\right) &= \log\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(y_b - u_b^L\right)^2\right)\right) \\ &= \log\frac{1}{\sqrt{2\pi}} - \frac{1}{2}y_b^2 - y_b u_b^L + \frac{1}{2}(u_b^L)^2.\end{aligned}$$

Classification: *Consider a binary classification problem. Then,* $y \sim Bern\,(p)$*, where* $p = \frac{1}{1+\exp(-u^L)}$*. Thus,*

$$\begin{aligned}\log p\left(y_b \mid w, x_b\right) &= \log\left(p^{y_b}(1-p)^{1-y_b}\right) \\ &= -\log\left(1 + e^{-u_b^L}\right) - (1 - y_b)u_b^L \\ &= -\log(2) + \frac{u_b^L}{2} - \frac{\left(u_b^L\right)^2}{4} + O\left(\left(u_b^L\right)^3\right) \\ &\quad - (1 - y_b)u_b^L.\end{aligned}$$

Based on the above description, let us assume that the final layer output u^L corresponds to a convolution or a fully connected layer with no activation function. Then, the log-

likelihood term in a regression and classification setup can be expressed as

$$\log p(y_b|w, x_b) = \text{polynomial}(u_b^{L-1} w(L)).$$

SGVB Estimator. Following (Kingma and Welling, 2013), the $\mathbb{E}_{q_\theta}[\log p(y|w, x)]$ term for the minibatch (of size B) can be written as

$$S_1 := \frac{1}{B} \sum_{b=1}^{B} \mathbb{E}_{q_\theta}[\log p(y_b|w, x_b)].$$

To approximate the expectation, we use 1 sample $w(\cdot, b)$ for each data point (x_b, y_b), which results in $S_1 = \frac{1}{B} \sum_{b=1}^{B} \log p(y_b|w(\cdot, b), x_b)$. Substituting in $\text{polynomial}(u_b^{L-1} w(L, b))$ into $\log p(y_b|w(\cdot, b), x_b)$ leads to the following form for variance $V(S_1)$,

$$\frac{1}{B} \left(V(w(L, b)) \mathbb{E}\left[\left(u_b^{L-1}\right)^2 \right] + V(u_b^{L-1}) \mathbb{E}^2[w(L, b)] \right), \tag{3.8}$$

plus higher order terms which decreases as B grows. By efficiently evaluating the KL term, we can utilize the memory savings to increase the batch size B and thus, to decrease the variance of S_1.

MC Reparameterization estimator of likelihood. The above strategy is practically sufficient. However, if B is limited by hardware, we can use the memory savings for more MC samples (higher M) for improving the estimate of the log likelihood term. This reduces the variance of first term in (3.8) by a factor of M, but the scheme described is restricted to the last layer.

3.4 Experiments: Bayesian DenseNet, U-Net, and other networks

We perform experiments on Bayesian forms of several architectures and show that training is feasible. While we expect some drop in overall accuracy compared to a deterministic version of the network, these experiments shed light on the benefits/ limitations of increasing MC iterations. Since model uncertainty is important in scientific applications, we also study the feasibility of training such models for classifying high-resolution brain images from a public dataset.

Setup. For deterministic comparisons, we run several variations of PreActResNet (He et al., 2016b) and Densenet (Huang et al., 2017) (9 in total) on CIFAR10. For brain images,

we use a simple modification of 3D U-Net (Ronneberger et al., 2015). Since our method is most relevant when a closed form for KL is unavailable, we select the approximate posterior to be a Radial distribution, where samples can be generated as: $\mu + \sigma * \frac{\xi}{||\xi||} * |r|$, where $\xi \sim MVN(0, I)$, $r \sim N(0, 1)$ and the prior of our weights is a Normal distribution. This satisfies the conditions of Theorem 3.5, allowing us to find a parameterization tuple that does not grow with respect to M: we can run 1000+ MC iterations with almost no additional GPU memory cost compared to 1 MC iteration. Another reason for choosing the Radial distribution as our approximate posterior q_θ is because Gaussian-approximate posteriors do not perform well in high-dimensional settings (Farquhar et al., 2020). Empirically, we find this to be the case as well; we were not able to train any models with a standard Gaussian assumption without any ad-hoc fixes such as pretraining, burn-in, or KL-reweighting (common in many implementations).

Parameter settings/hardware. All experiments used Nvidia 2080 TIs. The code was implemented in PyTorch, using the Adam optimizer (Kingma and Ba, 2014) for all models, with training data augmented via standard transformations: normalization, random re-cropping, and random flipping. All models were run for 100 epochs.

Time and Space Considerations

We first examine whether our MC-reparameterization leads to meaningful benefits in model size or runtime. We should expect a competitive advantage in model size as the number of MC iterations grows, which may come at the cost of significantly increased runtime. To allow ease of comparison, we fix the batch size for all models to be 32. We determine the maximum number of MC iterations able to run on a single GPU for a given model via the classical direct method. For DenseNet-121, we are able to run 89 MC iterations, while for VGG-16 we are only able to run 5 MC iterations.

Figure 3.7 shows a comparison of computational performance between our method and the direct approach. **(a)** With **our** construction, we significantly **reduce model size** on the GPU (Figure 3.7a). For smaller models like DenseNet, for the same number of MC iterations our method uses less than 25% of GPU memory, which allows for a significant **increase in batch size**. Since the size of the computation graph in our construction is independent of M, for the memory used in Figure 3.7a we are able to run for $M = 1000$ or more. **(b)** The significant reduction of model size on the GPU results in a reduction of training time per batch, up to 5× (Figure 3.7b); the generated computation graph has fewer parameters (nodes on the path) during backpropagation.

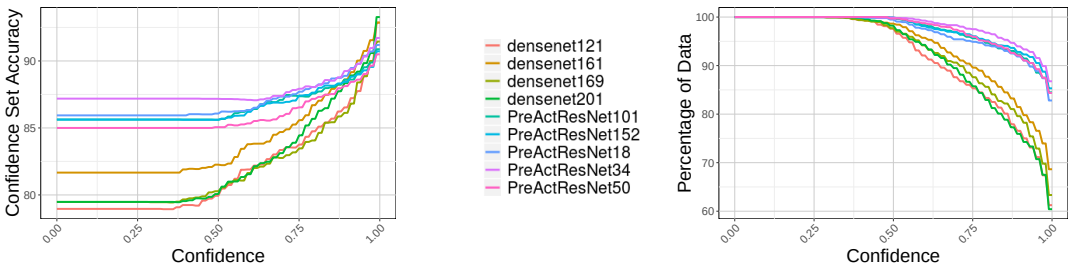


Figure 3.8: Confidence Set Accuracy and Confidence Sets on CIFAR-10 for a variety of ResNet and DenseNet models with 100 MC iterations (not previously possible). Both ResNet and Densenet achieve accuracy of more than 90% with 100% confidence, but ResNet is 100% confident on almost 90% of the data.

Prediction confidence/accuracy and how many MC iterations?

For our next set of experiments, we run a Bayesian version of PreActResNet and DenseNet with 100 MC iterations, which is feasible.

(**a**) We evaluate the accuracy concurrently with the confidence of the prediction, offered directly by the model. We expect that the model has a higher accuracy for those samples where it highly confident. This is indeed the case – Figure 3.8 shows the accuracy for varying levels of confidence over the entire validation set for a number of models. At high confidence levels, all models perform well, competing strongly with state of the art results. Additionally, we observe the proportion of data for which the model is confident is large (Figure 3.8 right). We can see that Bayesian model is at least 75% confident on 85%–95% of data.

(**b**) One issue in Bayesian networks is evaluating the expected drop in accuracy (compared to its deterministic versions), a behavior common in both shallow and deep models (Wenzel et al., 2020). Figure 3.8 (left) reassures us that the drop in performance for a number of widely used architectures is not that significant even when the model is not confident.

(**c**) To understand the effect of increasing the number of MC iterations, we run replications of experiment on ResNet-50 for 3 different number of MC iterations, Figure 3.9(left): 1 iteration (black), 17 iterations – maximum possible on GPU with the traditional method – (blue), and 100 iterations (red) possible to run due to our method. In all cases, as the threshold increases, model confidence increases and as expected, the accuracy does as well. However, we see that training with 100 MC iterations, consistently provides higher accuracy for the entire range of confidences. In contrast, with 1 MC iteration, accuracy has higher variance for the non-confident set.

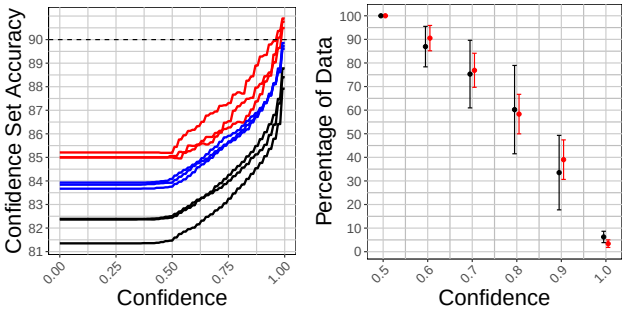


Figure 3.9: **(left)** Replicated Confidence Set Accuracy on CIFAR-10 for Resnet-50 with different number of MC iterations: 1 (black), 17 (blue, maximum allowed on GPU with standard method) and 100 (red). With $M = 100$ the accuracy is higher for any confidence. **(right)** Distributions of Confidence Set Size for a number of replications, with 1 MC iteration (black) and 100 MC iterations (red). With 100 MC iterations variance is smaller.

Neuroimaging: Predictive Uncertainty in Brain Imaging Analysis

While we demonstrated advantages of our reparameterization in traditional image classification settings and benchmarks – mostly as a proof of feasibility – a real need for BNNs is in scientific/biomedical domains: where high confidence and accurate predictions may inform diagnosis/treatment. To evaluate applicability, we focus on a learning task with brain imaging data.

Data. Data used in our experiments were obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI). As such, the investigators within the ADNI contributed to the design and implementation of ADNI and/or provided data but did not participate in analysis or writing of this report. A complete listing of ADNI investigators can be found in (Mueller et al., 2005). The primary goal of ADNI has been to test whether serial magnetic resonance imaging (MRI), positron emission tomography (PET), other biological markers, and clinical and neuropsychological assessment can be combined to measure the progression of mild cognitive impairment (MCI) and early Alzheimer's disease (AD). Classifying healthy and diseased individuals via their MR images, similar to ADNI, is common in the literature However, over-fitting when using deep models remains an issue for two reasons: small dataset size and a large feature space. Here, we look at a specific setting where we have 388 individuals with pre-processed MR images of size $105 \times 127 \times 105$. *Preprocessing.* All MR images were registered to MNI space using SPM12 with default settings.

Network. We use a slightly modified version of the encoder from an off-the-shelf 3D U-Net architecture (Ronneberger et al., 2015), demonstrated in Figure 3.10, to learn a classifier for cognitively normal (CN) and Alzheimer's Disease (AD) subjects. We note

```
Conv3D_Block(1, 16)
MaxPool3d((3,3,3))
Conv3D_Block(16, 32, stride=1)
MaxPool3d((2,2,2))
Conv3D_Block(32, 64, stride=1)
MaxPool3d((2,2,2))
Conv3D_Block(64, 128, stride=3),
MaxPool3d((2,2,2))
Conv3D_Block(128, 256, stride=3)
Linear(256, 2)
```

Figure 3.10: Structure of the model we used for ADNI classification.

that while this architecture is not competitive with those which achieve state-of-the-art classification accuracy on ADNI, our aim here is to demonstrate feasibility of training deep Bayesian models in this setting and evaluate the value of accurate confidence estimation.

We train the model on 300 individuals, and validate on the remaining 88. Additional experimental details can be found in the appendix. Since the input to the network is a mini-batch of high dimensional images, when we take into account the memory already needed by a deterministic model, we already reach the limits of the GPU memory. While we cannot perform more than 1 MC iteration with the standard method, we can successfully perform more than 100 with our scheme. We evaluate the consistency of performance with several runs of training when we are allowed to use 1 versus 100 MC iterations. (**a**) Table 3.2 shows the average validation accuracy for the choice of MC iterations and their difference. We see that for every confidence threshold, training with 100 MC iterations provides higher accuracy on average. This is especially noticeable on a high confident set, where the difference approaches 26.7%. (**b**) In addition to accuracy, it is important to understand how consistent the estimation is. Figure 3.9 (right) demonstrates the distribution of the size of confident set. While on average, the size of the "confident set" of the two models is similar, the variance is significantly smaller when we use a larger number of MC iterations, consistent with our hypothesis in Chapter 3. In cases where this confidence needs to be measured as accurately as possible, one obtains benefits over a single MC iteration.

3.5 Summary

While a broad variety of neural network architectures are used in vision and medical imaging, successfully training them in a Bayesian setting poses challenges. Part of the reason has to do with distributional assumptions. Moving to a broader class of distributions

	Confidence					
	0.5	0.6	0.7	0.8	0.9	1
$m = 1$	63.07 ± 1.47	62.14 ± 1.59	63.01 ± 0.64	64.13 ± 4.15	59.59 ± 4.40	60.71 ± 15.15
$m = 100$	64.39 ± 4.59	66.23 ± 4.19	66.05 ± 1.00	67.77 ± 4.00	66.82 ± 1.24	87.50 ± 17.68
Δ	1.33	4.09	3.04	3.64	7.23	26.79

Table 3.2: Average validation accuracy per model confidence for 2 values of MC iterations. $\Delta = A_{100} - A_1$, where A_i is validation accuracy for i MC iterations. With 100 MC iterations we got on average much better results, especially when prediction is highly confident.

involves MC estimations but direct implementations pose serious demands on memory and run-time. In this chapter, we identify that different computation graphs can be constructed for different parameterizations of the target function. Specifically when one is attempting a Monte Carlo approximation, these graphs can grow linearly with the number of MC iterations needed, which is undesirable. By directly characterizing the parameterizations that lead to different graphs, we analyze situations where it is possible for graphs to be constructed independent of this sampling rate (number of MC iterations). Evaluating our parameterization empirically, we find that it is feasible to run a large number of MC iterations for large networks in vision, with a nominal drop in accuracy (compared to deterministic versions). This chapter was published in the Conference on Uncertainty in Artificial Intelligence (UAI) as (Nazarovs et al., 2021b), and our API is a core component of a novel ransomware detection system in Microsoft with corresponding paper (Nazarovs et al., 2022b) and a patent. The code is available at https://github.com/vsingh-group/mcrepar.

Chapter 4

Understanding Uncertainty Maps in Vision with Statistical Testing

In the previous chapter, we describe our method to decrease memory consumption of BNN, which allows us to train Bayesian DNNs, especially various architectures common in vision. As we stated earlier, one reason to use probabilistic methods is to derive a notion of uncertainty. However, it is not always clear how to classify whether the defined uncertainty is significant. In this chapter, we propose a connection to Random Field theory, which allows us to derive significantly uncertainty regions.

With the adoption of deep neural network models in production systems for vision tasks, there is a growing consensus that we must be aware of what our model *does not* know. This is relevant not only for systems used for autonomous driving or medical imaging but also in less critical situations where such a model informs decision making in general and/or is responsible for generating triggers for user intervention. For example, inaccurate but overconfident predictions can lead to undesirable outcomes in assembly line manufacturing and logistics. This need has led to interest in the design of mechanisms for model calibration as well as for estimating uncertainties from deep neural network (DNN) models used in vision for tasks including but not limited to prediction (Lv et al., 2014; Poplin et al., 2018), segmentation (Akkus et al., 2017; Minaee et al., 2021; Qi et al., 2017), depth estimation (Fu et al., 2018; Godard et al., 2017) and visual odometry (Almalioglu et al., 2019; Konda and Memisevic, 2015).

Uncertainties can be roughly categorized into *aleatoric* (statistical) and *epistemic* (systematic). Aleatoric uncertainty can help capture inherent and irreducible data noise, which cannot be reduced even if more data were collected. It can be represented by heteroscedastic models (Holgersson and Shukur, 2004; McCulloch, 1985), since they assume that the observation noise (uncertainty) can vary with the input. Epistemic uncertainty accounts

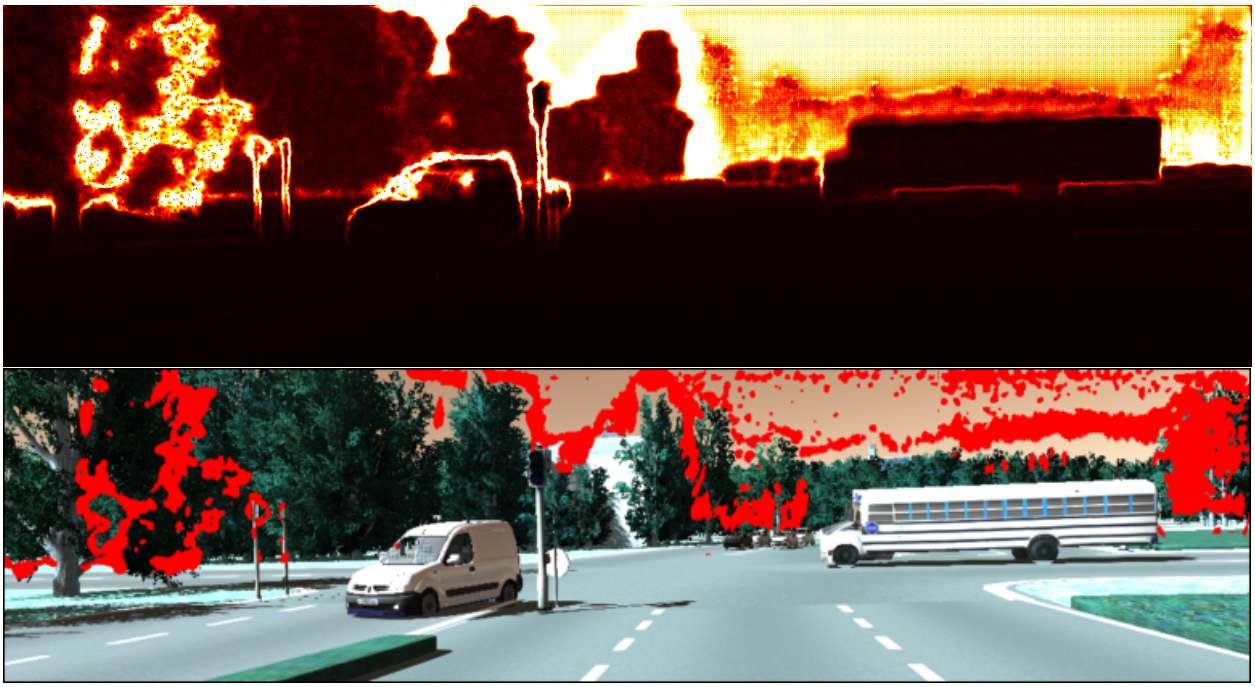


Figure 4.1: *Top* is a raw uncertainty of the depth estimation process. *Bottom* is **significant** region selected by our method, with theoretical guarantees to restrict family-wise error rate. This region can be used for calibration by practitioners.

for uncertainty in model parameters, and can be improved by observing more data. Capturing epistemic uncertainty in a DNN can involve putting a prior on the latent space (e.g., Variational Auto Encoder (VAE) (Pu et al., 2016)) or model parameters (e.g., Bayesian Neural Networks (BNN) (Blundell et al., 2015; Kononenko, 1989; Nazarovs et al., 2021b)), and adopting any available scheme to estimate the posterior probability. Several strategies exist which use hybrid approaches to capture either aleatoric or epistemic (or both) by combining heteroscedastic NNs and BNNs, e.g., (Kendall and Gal, 2017).

Example scenarios. While capturing different types of uncertainties is useful, in practical scientific/industrial settings, uncertainty estimates are merely a "means to an end". We must understand what actions the estimates enable, regardless of whether it is aleatoric or epistemic.

Scenario 1. Uncertainty estimates enable calibration, e.g., by a practitioner evaluating medical images. If a specialist can see that model is uncertain in some specific regions, he/she can evaluate whether to acquire more data if the regions where the model is uncertain are anatomically important. In other cases, such information can guide whether to request a biopsy. However, to decide, we need a statistically sound scheme to generate "significant" uncertain regions. Otherwise, interpreting the raw uncertainty is entirely subjective. Similar applications appear in depth estimation for autonomous vehicles (Hub-

schneider et al., 2019), Figure 4.1.

Scenario 2. Uncertainty can be used to compare confidences of models. Say a user is satisfied by the accuracy profiles of two models Model_A and Model_B but the second one has a higher latency. An upgrade to Model_B is only justified if one is 99% confident that it reduces uncertainty in a statistically significant sense on a held-out test dataset. This needs a "go/no-go" answer. Similarly, consider two systems for tumor volume dynamics using segmentation, which will drive treatment options (e.g., RECIST criteria (Eisenhauer et al., 2009)). Both systems offer similar accuracy and are FDA approved, but one is more expensive. The investment may be justified if the reduction in uncertainty is significant at a 99.9% level. Alternatively, consider a model on a small form factor device. The choice is between low-precision and high precision operations, the latter will need a larger battery. If both models satisfy client accuracy needs, is the reduction in uncertainty of predictions statistically significant?

Despite the growing body of work on uncertainty, frameworks that enable *actionable* information are limited. The goal of this the proposal described in this chapter is to close this gap.

Classical techniques from statistics. The problems above can be tackled with classical statistical testing. Here, we can set this up as pixel-wise statistical tests (although not strictly necessary; we will discuss alternative forms shortly). Scenario 1 will be a one sample test, while Scenario 2 will be a two sample test: we ask whether the uncertainty at a pixel is different across the two models.

Bottleneck. Deriving a scientifically valid conclusion for the image based on pixel-wise statistical tests will require conducting many tests, equal to the number of pixels. For example, an image of size 28×28 leads to 784 tests. For a common 0.05 critical value (probability of Type-1 error), we expect to select 40 ($\approx 784 \times 0.05$) pixels as significant, purely by chance (number of false positives). This issue escalates for higher resolution images, say 3D medical images. To avoid inflating the number of false positives, a multiple testing correction (e.g., Bonferroni, Benjamini-Hochberg) (Streiner and Norman, 2011) is used. However, for high-resolution images common in vision, this tends to over-correct making *none* of the tests significant (Adler et al., 2007; Vo et al., 2021), making the analysis less meaningful.

Many testing setups conservatively assume that the pixels are independent. The classical strategy to *avoid* this restrictive assumption leverages Random Field Theory (RFT), as studied in seminal papers by Adler and Worsley (Adler et al., 2017, 2007; Worsley et al., 1992). However, many theoretical results based on RFT remain restricted to the Gaussian Random Fields (GRF) and some specific generalizations. It is not obvious to what extent

these assumptions are viable for uncertainty maps obtained from deep neural networks popular in vision.

Contributions. We show how existing DNN tools when instantiated with suitable results from Random Field theory provide a mechanism to perform hypothesis tests on uncertainty maps, generated by different probabilistic DNN models common in vision. Specifically, we develop a probabilistic framework, based on Neural ODE and Wasserstein distance, which enables learning a diffeomorphism between uncertainty maps and GRFs. We refer to it as *Warping Neural ODE*. Roughly, this allows performing hypothesis tests on the resultant GRFs and mapping results back to the domain of uncertainty maps.

4.1 Statistical Review

Previously, in Chapter 2, we review several concepts we will use throughout this chapter. The main concepts are: hypothesis test, test statistics, family wise error rate (FWER) and random fields. Recall that via hypothesis tests, we can assess whether there is an evidence to reject the null hypothesis H_0 at a certain level of confidence α. Usually, H_0 states that there is no difference (say, from zero or between two groups), while H_A states that there is a difference. The decision is based on checking whether the observed test statistics F^{obs}, falls in the rejection region, defined by the threshold u. Threshold u is derived to control FWER.

4.2 Tests on uncertainty maps in vision

We start with an input image $\mathbf{x}$ and address Scenario 1 from Chapter 4. Leaving the specific task (depth estimation, segmentation) aside for the moment, we assume that a trained probabilistic model M provides an *uncertainty map* on the input $\mathbf{x}$, denoted as $M_{\mathbf{x}}$ where $M_{\mathbf{x}}(s)$ is the uncertainty for pixel $s \in S$. Our model will operate on these uncertainty maps/images (and not on $\mathbf{x}$). To infer which pixels of the uncertainty map (if any) are significant (rather significantly different from 0), we must conduct a hypothesis test.

A standard approach is a test for each pixel s, with $H_0 : M_{\mathbf{x}}(s) = 0$ and $H_A : M_{\mathbf{x}}(s) \neq 0$. The test statistics $F(s)$, for example, can be the student statistics

$$F(s) = \frac{(\overline{M_{\mathbf{x}}(s)})}{\sigma(\overline{M_{\mathbf{x}}(s)})},$$

where σ is an estimate of standard deviation and $\overline{(\cdot)}$ represents the sample mean. *For*

Scenario 2 in Chapter 4, to check if there is a difference between uncertainties of models A *and* B*, we replace* $M_{\mathbf{x}}(s)$ *by uncertainty* $A_{\mathbf{x}}(s) - B_{\mathbf{x}}(s)$.

In the next subsection, we describe how to address the multiple comparison issue. More specifically, we want a procedure that derives a threshold u of the rejection region $\{F(s) \geqslant u\}$ such that it **(a)** controls **FWER** and **(b)** accounts for spatial correlation of the image.

Random Field theory to the rescue

We consider the pixel-wise (or voxel-wise for 3D volumes) uncertainty map $M_{\mathbf{x}}$ as an RF over S with covariance C. Note that the pixel-wise uncertainties may not be independent from each other. We would like to statistically evaluate whether F is different from 0 or not. We will eventually use this strategy to find pixels with significant uncertainty (Scenario 1) and check the difference between uncertainties from two models (Scenario 2). This leads to the hypothesis setup, denoted as $\mathbb{H}_F$:

$$\begin{bmatrix} H_0 : \forall s \in S, M_{\mathbf{x}}(s) = 0 \\ H_A : \exists s \in S, M_{\mathbf{x}}(s) \neq 0 \end{bmatrix} \tag{4.1}$$

To perform the hypothesis test, it is necessary to establish a test statistic, which ideally describes the nature of the data and is a good indicator of whether to reject the null hypothesis. For RFs, a common test statistic for $\mathbb{H}_F$ is

$$F_{max} = \max_{s \in S} M_{\mathbf{x}}(s),$$

(Worsley et al., 1992). Finally, for the test $\mathbb{H}_F$, we need to find the threshold u_F, such that $\mathbb{P}(F_{max} \geqslant u_F | H_0) = \alpha$. Then, if the observed statistics[1] $F_{max}^{obs} > u_F$, we may reject H_0 in favor of H_A. However, computing $\mathbb{P}(F_{max} \geqslant u_F | H_0)$ is often nontrivial.

Typically, to obtain $\mathbb{P}(F_{max} \geqslant u_F | H_0)$, we need to know the theoretical distribution of the test statistics, denoted as $\mathbf{P}_{F_{max}}$, which may not have a closed form in general. Nevertheless, RF theory provides a way to estimate $\mathbb{P}(F_{max} \geqslant u_F | H_0)$ *indirectly*, namely through the *Euler Characteristic Heuristic* (ECH) (Taylor et al., 2005), an important result in RF theory. Given a u, we define an excursion set $A_u = \{s \in S : F(s) \geqslant u\}$. ECH shows that, for sufficiently large values u, $\mathbb{P}(F_{max} \geqslant u_F | H_0) \approx \mathbb{E}\{\phi(A_{u_F})\}$, where $\phi(A_u)$ is the Euler Characteristic (EC), a well studied quantity in topology that describes the shape of a topological space. Note that hereafter, EEC stands for $\mathbb{E}\{\phi(A_{u_F})\}$. Figure 4.2 demonstrates several examples of excursion sets and corresponding Euler Characteristics for different thresholds u.

[1] meaning the statistics F_{max} for an observed uncertainty map $M_{\mathbf{x}}^{obs}(s)$

Figure 4.2: Examples of excursion sets A_u for different levels of thresholds u (from left to right): $u = -2$, $u = 0$, $u = 1.8$.

How to compute $\mathbb{E}\{\phi(A_{u_F})\}$? A standard approach to computing $\mathbb{E}\{\phi(A_{u_F})\}$ is to use Monte Carlo (MC) approximation given Empirical ECs $\widehat{\phi}(A^{obs}_{u_F})$ over observed excursion sets $A^{obs}_{u_F}$. However, in addition to the MC approximation error, (Adler et al., 2017) shows that Empirical ECs at very high levels of u_F are generally too noisy to directly estimate the threshold u_F, such that $\mathbb{P}(F_{max} \geqslant u_F|H_0) = 0.05$. In practical setups, it might lead to incorrect hypothesis tests. An alternative approach is to derive the theoretical closed form for $\mathbb{E}\{\phi(A_{u_F})\}$, based on Theorem 4.1 below.

Theorem 4.1 (GKF: Gaussian Kinematic Formula(Taylor et al., 2006))**.** *If* F *is GRRF* (*isotropic or non-isotropic*), *EEC is given as,*

$$\mathbb{P}(F_{max} \geqslant u|H_0) \approx \mathbb{E}\{\phi(A_u)\} = \sum_{d=0}^{D} L_d(S, \Lambda(S))\rho_d(u), \tag{4.2}$$

where D *is the dimension of domain* S, $\rho_d(u_F)$ *is the Euclidean density* (*ED*), $L_d(S, \Lambda(S))$ *is the Lipschitz-Killing curvatures* (*LKC*) (*Zähle, 2011*), *and* $\Lambda(s) = Var(\dot{Z}(s))$ *is the variance of the spatial derivative of the underlying UGRF* $Z(s)$.

In simpler terms, Theorem 4.1 provides a general form of computing P-value $\mathbb{P}(F_{max} \geqslant u|H_0)$ for a threshold u, given that a test statistics F_{max} is Gaussian Related RF.

The problem in using (4.2): Even though (4.2) applies to a wide range of RFs, it is limited to the availability of the corresponding $L_d(S)$ and $\rho_d(u_F)$. For most RFs, these are unknown. **(a)** While curvatures $L_d(S)$ are available for isotropic RFs, when dealing with *non-isotropic* RFs, it is often extremely difficult to evaluate $L_d(S)$ (Adler et al., 2017). **(b)** Concurrently, the closed form solution for $\rho_d(u_F)$ is available only for a few distributions (Cao and Worsley, 2001).

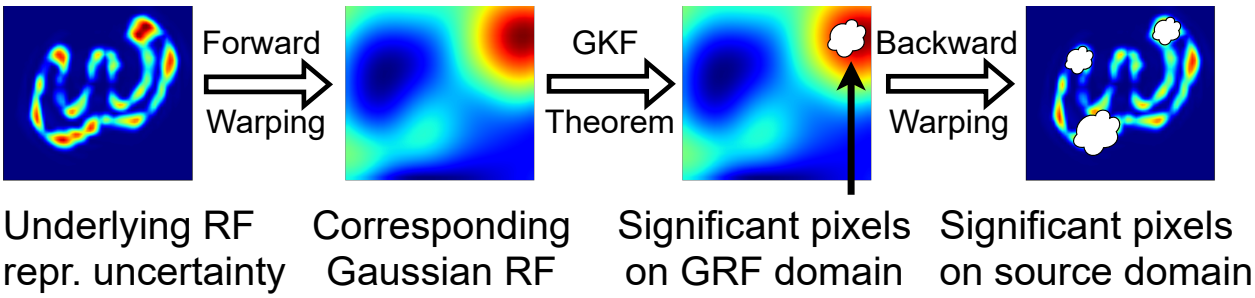


Figure 4.3: To understand the significant region of the uncertainty map we learn the diffeomorphism warping from general non-isotropic RF to the isotropic GRF. Then, given resulted GRF, we apply Theorem 4.1 to determine the significant region, and warp it back to the source domain.

Observe that the statistics $F(s)$, representing pixel-wise uncertainty, can in fact be considered as GRRF in several situations. This is because it is a function of Gaussian latent space in VAEs or the weights in BNNs. If so, we can apply Theorem 4.1. However, (**a**) the assumption of a RF F being isotropic on a domain S is unrealistic, which makes closed form solutions $L_d(S)$ defined for isotropic RF inapplicable. (**b**) The exact distribution of F is unknown, and thus ρ_d are also unknown. Then, is there a way to apply Theorem 4.1 on the observed uncertainty maps, generated by a DNN?

Let us warp to GRFs!

For the development of our proposal, we first (informally) state the following simple result, which describes how the warping of the domain (coordinate system) can help, proofs are in the Appendix B.

Theorem 4.2. *The domain S of the GRRF F can be warped via a one-to-one smooth transformation Γ to a domain S' without fundamentally changing the problem, namely:*

$$\mathbb{P}\left(\max_{s'\in S'} F(s') \geqslant t\right) = \mathbb{P}\left(\max_{s\in S} F(s) \geqslant t\right).$$

Theorem 4.3. *Consider the GRRF $F(S)$ on the domain S with Euler densities $\{\rho_d^F(u)\}$, and the GRF $Z(S^Z)$ on the domain S^Z with Euler densities $\{\rho_d^Z(u)\}$. Assume that both Euler densities $\{\rho_d^F(u)\}$ and $\{\rho_d^Z(u)\}$ are defined on the same domain $u \in U$ and*

$$\max_d\{\rho_d^F(u)/\rho_d^Z(u)\} \leqslant 1.$$

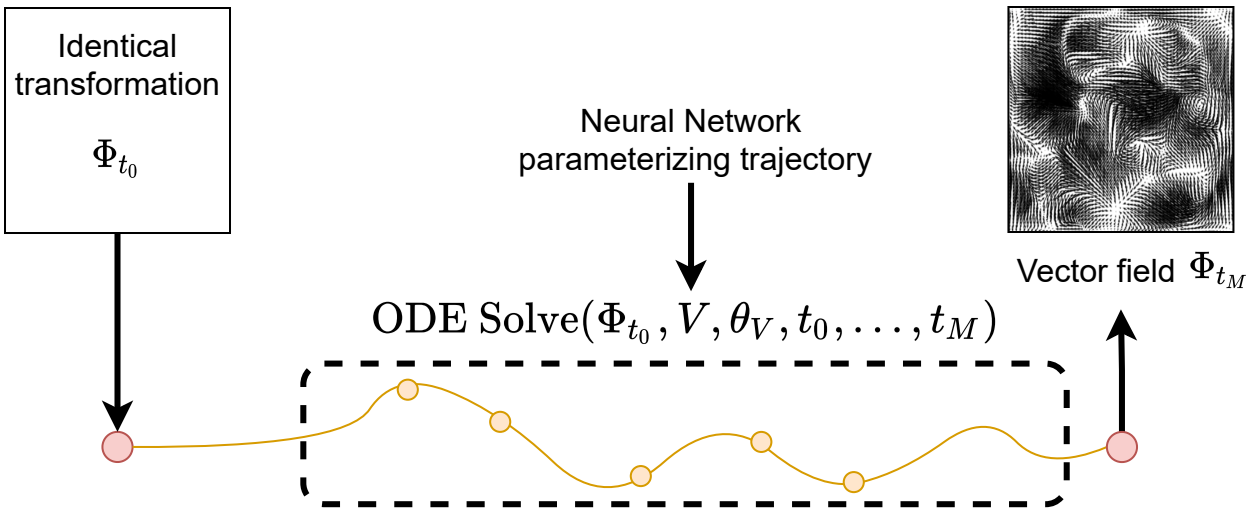


Figure 4.4: Neural Warping ODE: we model diffeomorphism Φ as a solution of ODE (4.3), where the RHS is modelled by NN. The resulted transformation Φ_{t_M} is applied to the coordinate system of input image to generate warped domain.

Then, by finding a one-to-one transformation Γ*, such that*

$$S = \Gamma S^Z \text{ and } S^Z = \Gamma^{-1} S,$$

and selecting a threshold u^**, such that*

$$\mathbb{P}\left(\max_{s \in S^Z} Z(s) \geqslant u^*\right) = 0.05,$$

guarantees that

$$\mathbb{P}\left(\max_{s \in S} F(s) \geqslant u^*\right) \leqslant 0.05.$$

Remark 4.4. *For the* **isotropic** *GRF* $Z(s)$*, all components of Theorem* (4.1)*,* L_d *and* ρ_d *are known in a closed form and thus, the corresponding threshold* u *can be computed:* $\mathbb{P}\left(F_{max} \geqslant u | H_0\right) \leqslant 0.05$.

Based on Theorem 4.3, we could warp the uncertainty map to the isotropic GRF (Figure 4.3, 1st arrow). Then based on Remark 4.4, we apply Theorem 4.1 and derive the significant region (Figure 4.3, 2nd arrow), and warp the significant region back (Figure 4.3, 3rd arrow). This approach is like (Taylor and Worsley, 2007), which warps a domain of non-isotropic GRF to achieve local isotropy. But in contrast to (Taylor and Worsley, 2007), we seek to find a warping of non-isotropic GRRF to isotropic GRF. To achieve this, we

should satisfy **two properties**: (**a**) the learned warping has to be a diffeomorphism, (**b**) the warped version of GRRF should be an isotropic GRF. *That is*, given a general (isotropic or non-isotropic) GRRF $F(S)$ in the source domain S and the isotropic GRF $Z(S^Z)$ on the GRF domain S^Z, we must find a transformation (warping) $\Phi(S)$, such that $F(\Phi(S)) \sim Z(S^Z)$, i.e., equal in distribution. Here, we can use recent developments in machine learning.

Learning diffeomorphisms

Learning the warp $\Phi(S)$ as a diffeomorphism guarantees invertibility of the transformations, which conserves topological features (Rousseau et al., 2020). For us, this means that we can recover significant regions from the Gaussian domain S^Z, but back in the source domain S, Figure 4.3 (3rd arrow). A specific class of diffeomorphisms, which define a subgroup structure in the underlying Lie group(Iserles et al., 2000), can be parameterized by an ordinary differential equation (ODE)(Rousseau et al., 2020; Ashburner, 2007):

$$\frac{d\Phi_t}{dt} = V(\Phi_t), \tag{4.3}$$

where Φ_t is the diffeomorphism at time t, and V the stationary velocity vector field. *Forward warping:* by starting from the initial point (identity transformation) Φ_0, we are able to integrate (4.3) in time $(t : 0 \rightarrow 1)$ to obtain Φ_1, such that $F(\Phi_1(S)) \sim Z(S^Z)$. *Backward warping:* in general with learning warping transformations, integrating backward in time $(t : 1 \rightarrow 0)$ does not result in a reverse warping (Ashburner, 2007). However, (4.3) defines a member of a Lie group, which provides a definition of the exponential operator. So, the correct way to define a backward warping Φ_{-1} is by integrating (4.3) over time $(t : 0 \rightarrow -1)$. To account for the richness of transformations, we parameterize the velocity V as a neural network, which gives a **Warping Neural ODE**, see Figure 4.4.

Mechanisms for generating a GRF

Given the warping $\Phi(S)$, we need to make sure that $F(\Phi(S))$ is an isotropic GRF. While various divergences can be used, e.g., Jensen-Shannon (Fuglede and Topsoe, 2004) or KL (Hershey and Olsen, 2007), we simply minimize the Wasserstein (EM) distance (Rüschendorf, 1985) between distribution of warped images $F(\Phi(S))$ and GRF:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|], \tag{4.4}$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. To achieve this, we minimize an efficient approximation of the Wasserstein

distance similar to (Arjovsky et al., 2017; Gulrajani et al., 2017). However, in contrast to GAN, in our setup the generator (Neural ODE component) does not generate images based on random samples, but is only used to create a warping Φ with no randomness.

Summary of the procedure (with final loss)

Algorithm 1 Learning diffeomorphism $\Phi : F \to Z$

Input: General RF $F = \{F_i\}_{i=1}^{N_F}$ and GRF $Z = \{Z_i\}_{i=1}^{N_Z}$
Output: Diffeomorphism $\Phi(S)$

Require: parameterized by Neural Networks: V in (4.3), critic $\mathcal{D}$ (to minimize Wasserstein Distance), $n_{\mathcal{D}}$ number of critic's updates

while V has not converged **do**
  Set Φ_0 as identical transformation (vector field).
  Using Neural ODE(V, Φ_0) find a solution Φ_1.
  Given Φ_1, warp F to $\hat{F}$
  Run **MinWasDist**$(\hat{F}, Z)$ to minimize the Wasserstein distance
end while
procedure MinWasDist$(\hat{F}, Z)$
  for $i = 0, \ldots, n_D$ **do**
    update D by minimizing *Critic's loss*:
    $-\mathcal{D}(Z) + \mathcal{D}(\hat{F}) + \lambda GP(\mathcal{D})$
    ▷ where $GP(\mathcal{D})$ is gradient penalty for Critic $\mathcal{D}$ (Gulrajani et al., 2017)
  end for
  update V by minimizing *ODE loss*:
  $-\mathcal{D}(\hat{F}) + JD + OG$ ▷ JD, OG defined below
end procedure

Remark 4.5. *While theoretically, it is guaranteed that there is a unique solution to the system (4.3) given Φ_0, see (Öhrnell, 2020) (pp. 8), to accelerate convergence, we add constraints (penalties) to the ODE loss in Algorithm 1, JD and OG respectively. Namely, we require (a) the Jacobian Determinant of each Φ_t to be non-negative (Kuang, 2019), to avoid collapsing several pixels into one, and (b) prevent generating warping Φ_t, with vectors going outside the grid (image frame).*

$$JD = \sum_t \sum_s \Big(|JD(\Phi_t(s))| - JD(\Phi_t(s)) \Big)$$
$$OG = \sum_t \sum_s \Big((grid(s) + \Phi_t(s) - F_{size}) + (grid(s) + \Phi_t(s)) \Big)$$

Note that computation of OG term is motivated by our implementation of the warping Φ_t as a vector field, common in vision (Jaderberg et al., 2015; Ashburner, 2007), and considering the 'grid' as a mesh coordinate system from 0 to size of image F_{size}. Then, pixels

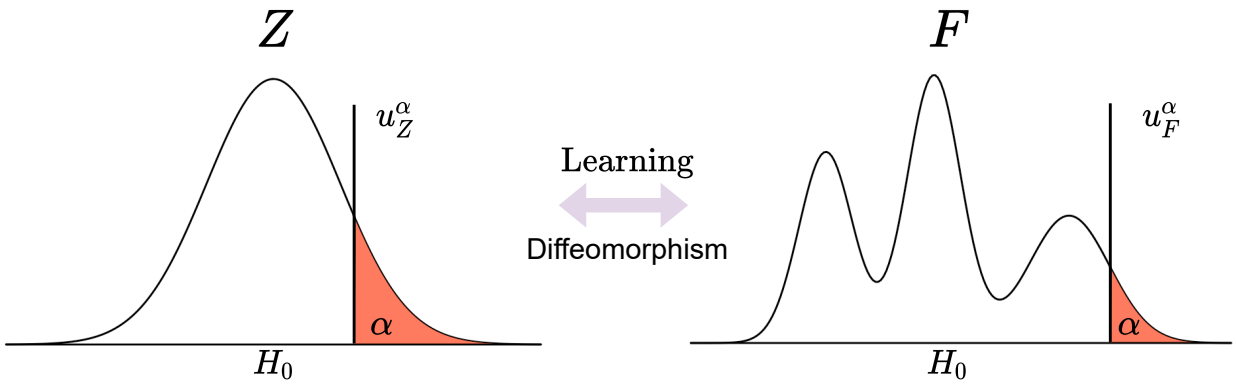


Figure 4.5: *Top:* Distribution of theoretical statistics $\mathbf{P}_{Z_{max}}$ and $\mathbf{P}_{F_{max}}$ under corresponding null hypotheses H_0 and thresholds u_Z and u_F, such that $\mathbb{P}(Z_{max} \geqslant u_Z | H_0) = \alpha$ and $\mathbb{P}(F_{max} \geqslant u_F | H_0) = \alpha$.

in the 'grid(s)' will be sampled from (or move to) location grid(s) + Φ_t(s). The OG term prevents learning vector fields Φ_t, which map to outside of the grid. Algorithm 2 describes the second part of the framework to select significant pixels on the source domain, given a learned warping Φ_t.

Algorithm 2 Selecting significant region $\mathcal{M}_F$

Input: RF $F = \{F_i\}_{i=1}^{N_F}$, learned diffeomorphism Φ_t
Output: Significant region $\mathcal{M}_F$

1: Apply forward warping Φ_1 to F to generate $\hat{F}$ from the GRF.
2: Select significant pixels $\mathcal{M}_{\hat{F}}$ of $\hat{F}$ according to Theorem 4.1.
3: Apply reverse warping Φ_{-1} to $\mathcal{M}_{\hat{F}}$ to generate $\mathcal{M}_F$ on domain of F.

Applications

So far, we have discussed how to get a significant region for a general RF F, which corresponds to the rejection region of GRF, see Figure 4.5. Our discussion was for a general case, without specifying how to obtain the RF F. Depending on how the RF F is obtained, this idea can be used for the two scenarios in Chapter 4: **(1)** to understand for which parts of the generated image, a model is the most uncertain and **(2)** to compare uncertainty between two models with different architectures. While (1) is important in scientific/healthcare settings, where we want to check whether we can trust a model in the region of interest, (2) helps evaluate whether users need to invest more in deploying a new model to decrease the uncertainty of their predictions.

Uncertainty within a model. To understand which part of the output image is the most uncertain, we generate F, given N outputs of the model, and using the variance per pixel.

Thus, we have F, with $F(s)$ showing uncertainty per pixel. Since our goal is to decide which pixels are the most uncertain, we apply the Algorithm (1) on the F and Z, generated under H_A. That is, all generated RF Z_i have some uncertain pixels, and on Figure 4.5 we map only α regions between F and Z. Then, we find $\mathcal{M}_F$ as in Algorithm 2.

Deriving 'significant regions' of uncertainty may appear similar to strong/weak class activations map methods (Zhou et al., 2016). However, our method is complimentary – it can be used downstream if the heat maps *also* include pixel-wise *confidence intervals* and satisfy GRRF assumptions.

Uncertainty between the models. Given two sets of images, we compute test statistics F^{obs} as mentioned before. Then, we use a bootstrap technique to construct a set of statistics $\{F_i\}_{i=1}^N$ (see Appendix B). We follow Algorithm 1 to map the distribution F to Z completely, i.e., not just the α like in Scenario 1. Finally, we obtain significant region $\mathcal{M}_{F^{obs}}$ on F^{obs}. If $\mathcal{M}_{F^{obs}}$ contains at least a 1, then we reject null hypothesis that there are no improvements in uncertainty of the model in favor of H_A. While we provide a way to test the uncertainty *between* models, here, we will restrict our presentation to the uncertainty within a model.

Limitations: In the current form, our method cannot be used directly to prioritize deep uncertainty quantification approaches, e.g., BNNs(Nazarovs et al., 2021b), deep ensembles(Lakshminarayanan et al., 2017), and so on. Similar to hypothesis tests, we do not obtain a model ranking. But if we know, say the model/software cost, then the cheaper model is better *if* the H_0 is *not* rejected.

4.3 Experiments

We seek to demonstrate the ability of our model to provide estimates of statistical significance for the uncertainty generated by different probabilistic models, e.g., Variational Autoencoders (Kingma and Welling, 2013), Neural Networks with MC Dropout (Gal and Ghahramani, 2016), and Bayesian Neural Networks (Nazarovs et al., 2021b; Kendall and Gal, 2017; Gustafsson et al., 2020). In our experiments, we use a broad range of common vision datasets, e.g. CelebA (Liu et al., 2015), AFHQ (Choi et al., 2020), KITTY (Geiger et al., 2013), MS-COCO (Lin et al., 2014), and MR image data from the Alzheimer's Disease Neuroimaging Initiative (ADNI) (three time-points, representing disease progression).

We construct significant regions according to $\mathbb{P}(F_{max} \geqslant u_F | H_0) = 0.05$, a commonly used threshold (Storey and Tibshirani, 2003). For the baseline, we consider a typically used 5% quantile (Kendall and Gal, 2017). The Appendix B provides description of the DNN architectures as well as a simple experiment for different RFs with ground truth for significant regions.

Proof of concept: We start the experimental section by introducing Warping Neural ODE as a generative model, we train our model to warp samples from a distribution in the shape of handwritten digits '3' to digits '4'. Since we can evaluate the solution of ODE in (4.3) at arbitrary time t, in Figure 4.6, we visualize the evolution of Φ_t. It is evident that our model can indeed learn a smooth diffeomorphism.

VAE: Given the generation mechanism of VAE (Razavi et al., 2019; Zhang et al., 2019), the estimation of the epistemic uncertainty, i.e., uncertainty of the model, is direct. For each input image x, we run the inference M times generating M samples $x_1, \ldots, x_M$, on which we compute pixel-wise and channel-wise variance. Since the latent space of a VAE follows a Gaussian distribution, the resultant uncertainty F satisfies our assumptions (regarding GRRF), and so, we can apply our method directly to understand the significant regions.

For these experiments we consider different variations of VAE models (based on ResNet-18, ResNet-34 and ResNet-50 (He et al., 2016a)) and four different datasets: CelebA (Liu et al., 2015), AFHQ (Choi et al., 2020): closeups of 3 types of animals: Cat, Dog and Wild.

(**a**) *CelebA.* In Figure 4.7, we show the uncertainty map and compare significant pixels derived from our method with the usual 5% quantile. *First*, we see that our approach picks up regions of clustered uncertainty, which indicates that our model is aware of spatial correlation in uncertainties. But, a standard 5% quantile picks up boundary points and stray/disperse points. It is especially obvious on the third row, for the most expressive network ResNet-50, where the generated uncertainty of the entire region is small. Our method picks up only the maximum and sensible regions (like teeth), where you would expect the most variation happens generating human faces (open-close mouth). While the 5% quantile picks up the boundary of the generated object – not a very meaningful region for calibration, since it does not capture important face features. *Second*, using the more expressive model (from top to bottom), our method picks up less significant uncertainty regions. In contrast, the 5% quantile picks up about the same number of pixels regardless of model confidence.

Observations: While we expect to have smaller significantly uncertain regions with an increase in complexity of the model, we do not expect models to be the most uncertain in the same exact regions. Moreover, we expect that with an increase in the complexity

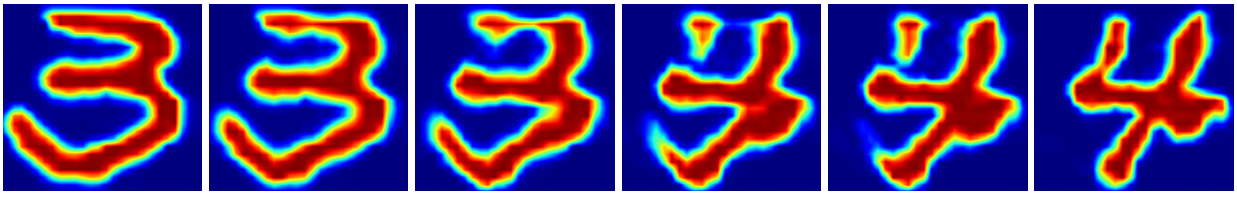


Figure 4.6: Continuous warping of '3' to '4' using our Warping NODE

of the model, regions of significant uncertainty will be removed first. This can be seen by comparing uncertainty maps and significant pixels of our method in Figure 4.7 across models. This behavior is harder to observe using the 5% quantile. We find that the process of elimination we observe is similar to 'backward' feature selection method in statistics (Derksen and Keselman, 1992), when we remove significant features based on p-values after each pass.

Computation/storage complexity: For CelebA, our model (14M parameters) occupies about 1934MiB. Runtime is 0.3s with a batch size of 1. On a standard system with four 2080TIs, 1 epoch of 10000 images needs 120s and full training (for warping) takes 7 hours. At test time, the hypothesis test (on PyTorch) is negligible ($\leqslant$ 1 ms).

(b) *AFHQ.* Since we showed the benefits of our method compared to the 5% quantile for selecting the significant uncertainty regions generated by VAE, for the AFHQ dataset, we only provide results of our method for a single network ResNet-18 in Figure 4.8. The goal is to demonstrate that significant uncertain regions are coherent and interconnected, as opposed to being isolated and sporadic points (like random points on edges of the frame). By doing so, we can obtain a more realistic portrayal of the limitations and uncertainties that exist in real-world systems. We see that the most uncertain regions are areas around eyes and ears. This result is not surprising as these regions tend to have the most variable feature representations in animals.

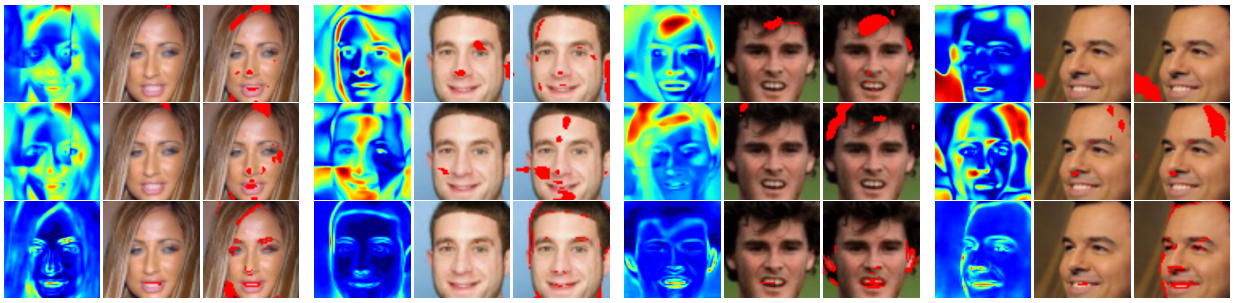


Figure 4.7: Rows: 1st – ResNet-18, 2nd – ResNet-34, 3d – ResNet50. Columns (by 3): 1) uncertainty map, generated by the VAE model with the ResNet base corresponding to the row, 2) significant uncertainty, derived by **our** method, 3) top 5% of uncertainty, typically used as significant uncertainty in vision.

MC dropout: In (Gal and Ghahramani, 2016), the authors showed that when applying dropout on every layer, the dropout objective minimizes the Kullback–Leibler divergence between an approximate distribution and the posterior of a deep Gaussian process. Thus, uncertainty obtained from MC dropout satisfies our assumption of GRRF. Given a trained deterministic network, we can inject MC dropout layers to estimate uncertainty. We evaluate

Figure 4.8: Rows: cats, dogs, wild. Columns: significant uncertainty, derived by **our** method for different samples.

our method on uncertainty derived from MC dropout applied to two large scale datasets on different tasks: depth estimation and segmentation.

(**a**) *Depth estimation on Virtual KITTI dataset:* The virtual KITTI dataset (Gaidon et al., 2016) is a photo-realistic synthetic video dataset, which consists of high resolution scenes and is usually used for vision tasks such as object detection, multi-object tracking, scene-level and instance-level semantic segmentation, and depth estimation. We evaluate the ability of our model to pickup significantly uncertain pixels in high-resolution uncertainty maps. We follow the experiment setup of (Gustafsson et al., 2020) to evaluate the depth of objects in images. Using MC-dropout, we generate uncertainty maps of size 320×1216 (Gal and Ghahramani, 2016) and evaluate our model on three different types of uncertainties: epistemic, aleatoric and predictive (sum of both: epistemic and aleatoric). The results are shown in Figure 4.9. Clearly, for all types of uncertainty (rows), our method (middle column) picks up regions of clustered uncertainty making the significance mask more smooth, indicating that our model is aware of spatial correlation in images, compared to the usual 5% quantile (right column). This is quite noticeable for epistemic uncertainty – uncertainty of the model (middle row). Further, Figure 4.10 shows the predictive uncertainty of zoomed-in regions, for objects with strong edges, like a light pole or a tree. Our method does not pick up the edges as significant, making significant regions more meaningful and avoiding noise.

(**b**) *Segmentation on MS-COCO:* Common Objects in Context (COCO) (Lin et al., 2014) is

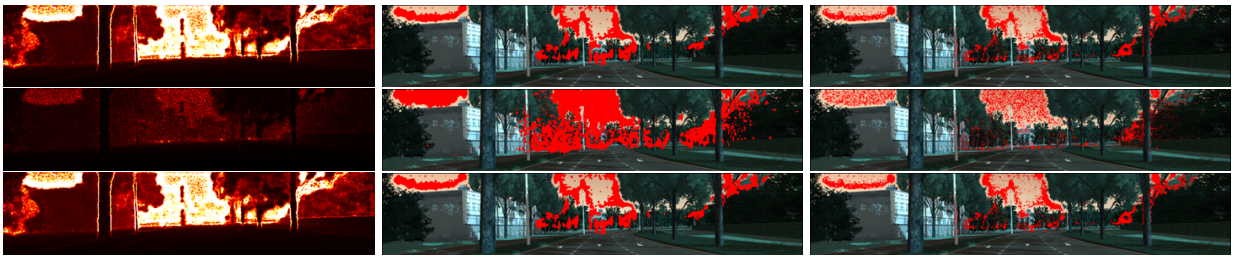


Figure 4.9: For each of the tree types of uncertainties along the rows (aleastoric, epistemic, and predictive) we demonstrate (left) the uncertainty of the depth estimation, (center) significant uncertainty region derived by our method, and (right) typically used as significant 5% quantile.

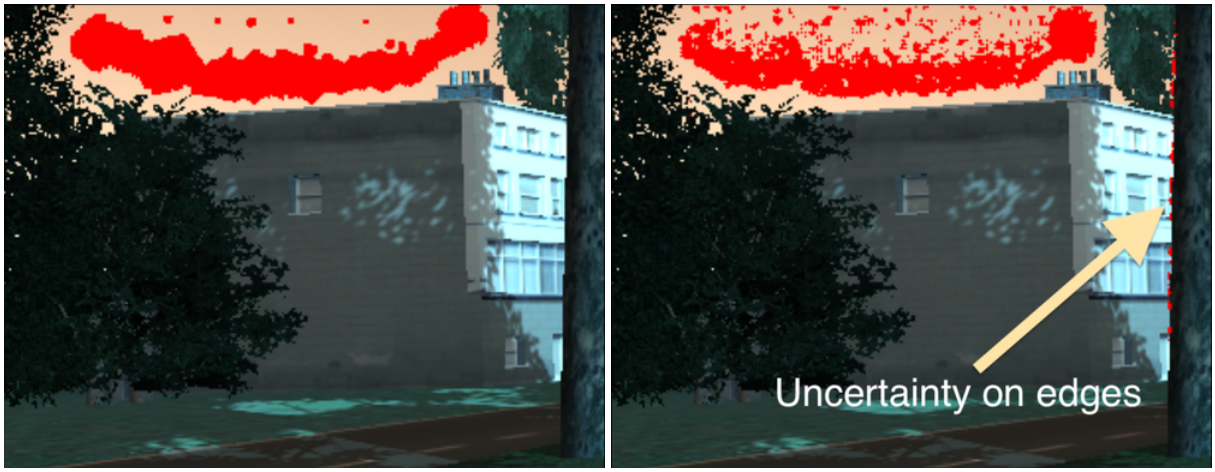


Figure 4.10: Zoomed region of predictive uncertainty, to show that in contrast to 5% qunatile (right), our method (left) does **not** pick up seldom (and meaningless) points on the edge of the tree.

a large-scale vision dataset that provides a rich set of visual descriptors and is widely used for baseline evaluations of semantic segmentation algorithms (He et al., 2017; Chen et al., 2018a). To evaluate the effectiveness of our method, we generate segmentation uncertainty by applying MC Dropout to each layer of the DeepLab V3 (Chen et al., 2017) with the pre-trained checkpoint in PyTorch (Paszke et al., 2019). We measure uncertainties by summing over the individual variances of the model's predictions in softmax-normalized scores for each segmentation class and pixel-wise. To evaluate the effect of probability of dropout p on the generated uncertainty, we consider 3 variations: $p = 0.01$, $p = 0.03$ and $p = 0.04$. We noticed that using dropout with probability $p \geqslant 0.05$ leads to a very high uncertainty and no meaningful segmentations. Based on results in Figure 4.11 for all 3 values of dropout probability p, there are 2 interesting observations. **(1)** As noted previously, the 5% quantile shows that significant uncertainty on one image is located in a lot of different classes and difficult to interpret. In contrast, our method is consistent when providing significant

uncertainty within the class. **(2)** While the 5% quantile significant region changes through different values of dropout probability p (across the rows in Figure 4.11), the significant region based on our method is consistent across p.

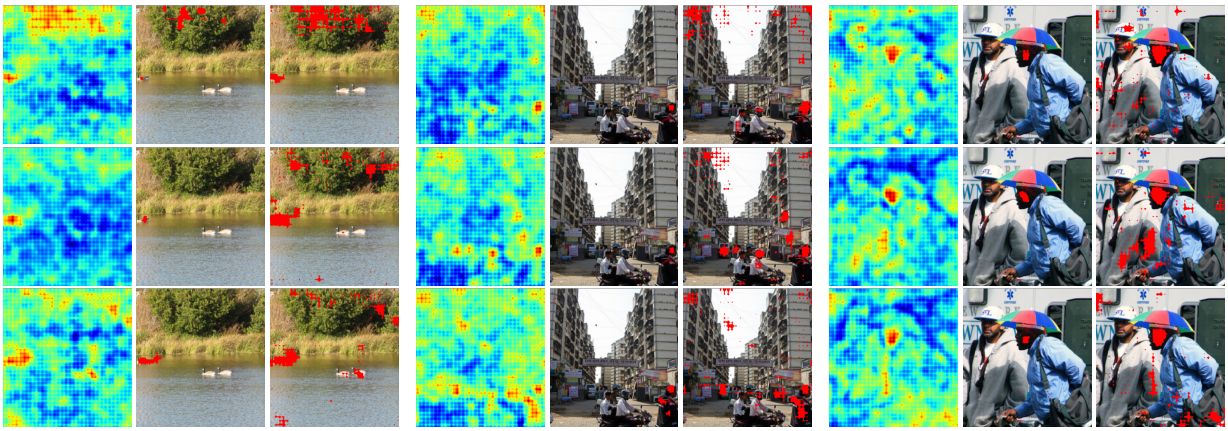


Figure 4.11: For each value of dropout probability p along the rows (0.01, 0.03, 0.04) we demonstrate by triplets: (left) the uncertainty of the pixel-wise segmentation, (center) significant uncertainty region derived by our method, and (right) typically used as significant 5% quantile.

Bayesian Neural Networks: The final method we use is Bayesian Neural Networks (BNN) (Nazarovs et al., 2021b; MacKay, 1995; Gal and Ghahramani, 2015). We apply Temporal BNN to longitudinal (3 time point) brain imaging data obtained from Alzheimer's Disease Neuroimaging Initiative (ADNI), see Appendix B. This data set is also analysed in Chapter 5. The goal is to predict the brain image at the third time point, given that the first two steps are observed. To generate the uncertainty map, we collect predictions from 100 feed-forward runs of a trained BNN and compute voxel-wise standard deviations. Then we take a 2D slice as a final uncertainty map. We note that the application of a 5% quantile as a threshold for significance does *not* yield very meaningful results, completely covering two big regions of the brain: corpus callosum and caudate nucleus, and some stray pixels all around. In contrast, our method highlights small clustered regions. Since the data contains two clinically disparate groups, we should expect that samples from different groups have different significantly uncertain regions, generated by the predictive model. The 5% quantile threshold shows the same significant pixels, independent of diseased or control subjects (group difference testing). In contrast, our method nicely differentiates between CN (healthy) and AD (subjects with Alzheimer's disease) groups. In summary, we find that identifying statistically significant pixels shows that longitudinal progression in AD is quite different from CN, captured using our method (see Appendix B).

4.4 Summary

This paper provides a strategy for using existing deep neural network tools in conjunction with known results in Random Field Theory (RFT) to perform hypothesis tests on uncertainty maps from DNN models. Such a capability allows moving from subjective interpretation of uncertainties or the evaluation of deciles/quantiles to answering precisely stated hypotheses in a rigorous way. We believe that this capability is essential but currently missing and can further enable the use of DNN models from vision in mission-critical applications and for informing business/policy decisions. This chapter was published in the Conference on Computer Vision and Pattern Recognition (CVPR) as (Nazarovs et al., 2022a). The code is available at https://github.com/vsingh-group/uncertainty_with_rf.

Chapter 5

Mixed Effects Neural ODE: A Variational Approximation for Analyzing the Dynamics of Panel Data

Temporal data is common in many applications including but not limited to vision (video, sequence of medical images), physical models, audio, text and so on. To analyze temporal data it is necessary to account for the temporal nature of the data in the model. In this chapter we present a new framework Mixed effects Neural ODE, which allows to model temporal data in a continuous way, accounting for correlation within and between groups of observations.

Of particular interest are observational studies in the social and health sciences which often involve acquiring repeated measurements **over time** for participants/subjects. If most participants stay enrolled, we can consider each row in the *panel* to correspond to longitudinal measurements or records of an individual, at regularly or irregularly sampled time points. Modeling development or growth trends while accounting for *variability within and across individuals* leads to the need for statistical models for analysis of such "panel data" (Kreindler and Lumsden, 2006; Katsev and L'Heureux, 2003).

The modeling of temporal processes can be set up as a regression task, where a function (with unknown parameters) that is plausible for the domain is estimated using the observed longitudinal data samples. Apart from splines and tools from functional data analysis, a common alternative is to use differential equations (Chen and Wu, 2008; Liang and Yu, 2013; Fang et al., 2011), which provides expressive power and many computational tools developed over decades. However, differential equations do not directly account for variability within and across subjects – hallmark features of panel data. To capture some of these characteristics, the widely used Auto-Regressive Models (ARMA) literature

incorporates white noise type functions in differential equation models, leading to various forms of stochastic differential equation (SDE) (Brockwell, 2001; MaCurdy, 1982; Hedeker and Gibbons, 2006),

$$z_t = f_\mu(z,t)dt + L_\Sigma(z,t) \circ d\beta(t) \tag{5.1}$$

where $z_t \in \mathbf{R}$, f, L denote the drift and noise sensitivity functions with unknown parameters μ and Σ respectively.

Suppose we are given a set of (partial) measurements $\{z_{t_k}\}$ at certain K time points $\{t_k\}_{k=1}^K$, and an efficient numerical scheme to simulate the SDE in (5.1). Then, the unknown parameters μ and Σ can be found by simply maximizing the likelihood function $p(z_t|\mu;\Sigma)$. In principle, it is straightforward to extend the model in (5.1) to high dimensional z_t. However, there are two main technical challenges in this setting with stand-alone likelihood based methods: **(i)** such an approach requires a large number of longitudinal measurements which is often infeasible, especially in the applications that motivate our work (Marinescu et al., 2018); **(ii)** numerical schemes to simulate nonlinear SDEs in high dimensions are quite involved. Indeed, these issues become pronounced when we assume that the observed data is not z but x which are actually measurements governed by a process that reflects the dynamics z.

The literature provides a principled way, called **Bayesian filtering**, to tackle the problem described above, see (Särkkä and Solin, 2014, Chapter 7). Let us consider that the object or measurement x is evolving as non-linear function D from a *latent* measure of dynamics/progression z_t (Pierson et al., 2019; Hyun et al., 2016; Whitaker et al., 2017): $x_t = D(z_t) + \varepsilon_t$. So, it is natural to think of the observable x of an unknown dynamics z – which we can call the "latent" representation. The goal in Bayesian filtering, which aligns nicely with our task, is to compute, $p(z|x_{t_1}, ..., x_{t_K})$. Interestingly, under some assumptions, closed form solutions for the posterior distributions $p(z|x_{t_1}, ..., x_{t_K})$ are available, see in (Särkkä and Solin, 2019, Chapter 10). However, these assumptions are hard to verify in general, and direct utility of these approaches for modern applications is not obvious.

Main ideas/contributions. The most important takeaway from the description above is not the mechanics of *how* Bayesian filtering is carried out in practice, rather, *what* it seeks to estimate. If we focus on the key object of interest – the conditional distribution – we realize that recent works in machine learning do provide a recipe that exploits the universal approximation properties of neural networks to represent fairly complex conditional distributions. In this case, the parameters are simply trained using off-the-shelf procedures, and DE numerical solvers are required for (5.1). While both ODE and SDE solvers are available, SDE is typically less efficient (Li et al., 2020; Liu et al., 2019). Notice that when $L_\Sigma \equiv 0$, that is, the observable x follows an ODE and corresponding solvers can be applied,

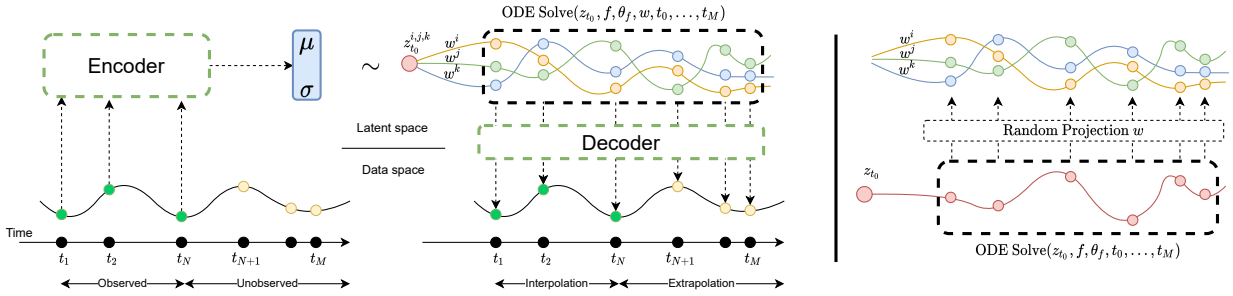


Figure 5.1: Structure of the model. First, encoder is applied to temporal data of an subject to generate initial point of the trajectory in latent space; Second, ME ODE solver is used to generate trajectory from the specified initial point; Last, decoder is used to map latent space ODE stages into observed values. On the right side we show how ME ODE can be viewed as random projection of trajectory of standard ODE, where trajectory is defined by random effect **w**.

the approach that would instantiate this idea has already been successfully tried in (Yildiz et al., 2019). However, given the Bayesian filtering motivation above, is there a way to utilize ODE solvers, while preserving stochastic nature of (5.1)?

Our development begins by rewriting the noise term in (5.1) with a series of basis functions with standard normal coefficients, which comes from representation of Brownian motion through Karhuen-Loeve expansion (Loeve, 1963) and asymptotic approximation of Stratonovich SDE with Wong and Zakai theorem (Wong and Zakai, 1965). We show that this modification enables incorporating random effects in our predictions – which appropriately models the variability of the data, the requirement for successful analysis of panel data. More importantly, using our approach, we show that for a special class of latent SDE based models, which we will refer as Mixed Effects Neural ODE (ME-NODE), the parameters of the underlying neural network can be trained efficiently *without* backpropagating through any SDE solvers. To achieve this, we derive the Evidence Lower Bound loss – where widely available libraries for numerical ODE solvers – are sufficient and directly applicable. The framework is presented on Figure 5.1. We show applications to brain imaging where our formulation can provide personalized prediction together with uncertainty, a feature of Bayesian methods.

5.1 Notation and review

In this section we present the notations and some basic concepts we use in the paper.

Notation. For a time-varying vector $\mathbf{z} = (z_0, \ldots, z_n)$, (with n time points) we denote a vector without the j-th component as a $\mathbf{z}_{-j} = (z_0, \ldots, z_{j-1}, \quad z_{j+1}, \ldots z_n)$. Often each time point z_t is vector valued, and in the rest of the paper we denote it by $\mathbf{z}_t \in \mathbf{R}^p$, where p is the number of variables. Thus, to denote a time-varying vector where each time point represents a p dimensional vector we use $\mathbf{z} = (\mathbf{z}_0, \ldots, \mathbf{z}_n)$, where each $\{\mathbf{z}_t\}_{t=1}^n \subset \mathbf{R}^p$. We denote the indicator function as $\mathbb{1}_{\mathbf{x}}\{\mathbf{y}\}$: it is 1 if $\mathbf{y} = \mathbf{x}$, and 0 otherwise. We refer to a general form of ODE as $\dot{\mathbf{z}}_t = h(\mathbf{z}_t)$, where $h(\mathbf{z}_t)$ defines a "trajectory" and depends on the current value of the process at time t. Without loss of generality, in this section we assume that the DE is defined on $\mathbf{R}$.

Smooth Approximations of SDE. Consider the standard form of the Stratonovich SDE given by (5.1). According to (Särkkä and Solin, 2019, Chapter 7) and (Loeve, 1963), we can represent the differential of standard Brownian motion using Karhunen-Loeve expansion as:

$$d\beta(t) = \sum_{n=1}^{\infty} b_n \varphi_n(t), \tag{5.2}$$

where $\{\varphi_n\}$ are a suitable set of basis functions. For example, if $t \in [0, T]$, then $\{\varphi_n\}$ can be the Fourier cosine basis as

$$\varphi_n(t) = \left(\frac{2}{T}\right)^{1/2} \cos\left(\frac{(2n-1)\pi}{2T} t\right).$$

Given Karhunen-Loeve expansion of differetial of standard Brownian motion in (5.2), Wong and Zakai theorem (Wong and Zakai, 1965; Hairer and Pardoux, 2015; Särkkä and Solin, 2019, Chapter 9) shows that the solution to (5.1) can be approximated asymptotically ($N \to \infty$) by the solution of the following equation:

$$\dot{z}_t = f(z, t) + L(z, t) \sum_{n=1}^{N} b_n \varphi_n(t), \tag{5.3}$$

where $b_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1)$ and $\{\varphi_n\}$ are a suitable set of basis functions. Based on mild simplifying assumptions, we get

$$\dot{z}_t = f(z, t) + g(z, t) b \tag{5.4}$$

is an approximation of Stratonovich's SDE in Equation (5.1), where $b \sim \mathcal{N}(0, 1)$.

Proof. Assume that $\exists g(z,t) < \infty$ and $\forall n$, $\exists\ \sigma_n(z,t) < \infty$ such that

$$\varphi_n(t)L(z,t) = \frac{\sigma_n(z,t)}{\sqrt{\sum_{n=1}^{N}\sigma_n^2(z,t)}}g(z,t). \quad \text{Then,} \tag{5.5}$$

$$\sum_{n=1}^{N}\varphi_n(t)L(x,t) = g(z,t)\frac{1}{\sqrt{\sum_{n}^{N}\sigma_n^2}}\sum_{n=1}^{N}\xi_n(z,t),$$

where $\xi_n(z,t) = b_n\sigma_n(z,t) \sim \mathcal{N}(0,\sigma_n(z,t))$. If for some $\delta > 0$,

$$\frac{1}{\left(\sqrt{\sum_{n=1}^{N}\sigma_n^2}\right)^{2+\delta}}\sum_{n=1}^{N}\mathbb{E}\left(|\xi_n-\mu_n|^{2+\delta}\right) \overset{N\to\infty}{\to} 0, \tag{5.6}$$

then according to Lyapunov's Central Limit Theorem $\frac{1}{\sqrt{\sum_{n}^{N}\sigma_n^2}}\sum_{n=1}^{N}\xi_n(z,t) \to \mathcal{N}(0,1)$. Then, $\dot{z}_t = f(z,t) + g(z,t)b$ is an approximation of Stratonovich's SDE (5.1), where $b \sim \mathcal{N}(0,1)$. □

This means that under standard second moment conditions on the stochastic part, the solution of (5.4) can be seen as an approximation of the solution to Stratonovich's SDE in Equation (5.1). The benefits of this simplified form are two-fold: **(a)** since z is a random variable, we can incorporate uncertainty within the ODE similar to a SDE, **(b)** for a given z, the trajectory in (5.4) can be modeled using an ODE and the associated computational benefits become available. Concurrently, (Hodgkinson et al., 2020) showed that it is possible to generalize this result to a more general class of SDEs using tools from the theory of rough paths. We must note (discussed briefly later), that for specific choice of f and g, the RHS of Equation (5.4) is well known in statistics and machine learning as a **Mixed Effects model** (Hyun et al., 2016). Therefore, we will refer to our model as a mixed effects model and use the relevant terminology from this literature whenever possible. We hope that this will make the presentation more accessible and clarify that our scheme is *not a general purpose replacement to a* deep neural network based SDE solver.

Mixed effects model. Assuming individuals/groups are denoted by i, a nonlinear mixed effects (ME) model (Demidenko, 2013) can be written as:

$$\boldsymbol{\phi}^i = \nu\left(X^i\boldsymbol{\beta} + U^i\mathbf{b}^i\right) + \boldsymbol{\epsilon}^i, \tag{5.7}$$

where $X^i \in \mathbf{R}^{n \times m}$ is a matrix of covariates where n and m are the number of observations and variables respectively. Here, ν is a non-linear (vector-valued) function, $\boldsymbol{\beta} \in \mathbf{R}^m$ is a vector of *fixed* effects, $\mathbf{b}^i \sim \mathcal{N}(\mathbf{0}, \Sigma_b)$ is a vector of *random* effects, U^i is a design matrix (modeling choice) for random effects, $\boldsymbol{\phi}^i \in \mathbf{R}^n$ is the response variable and $\boldsymbol{\epsilon}^i \sim \mathcal{N}(\mathbf{0}, \Sigma_{\epsilon^i})$ represents a noise term.

Task. Our goal is to learn a latent representation of a time-varying physical process/dynamics z, and distribution $p(z|x_1, \ldots, x_{t_k})$. Because we focus on learning a latent representation z of the observable process x, we focus on variational approximation techniques.

Learning latent representations with a VAE. Variational auto-encoders (VAE) (Kingma and Welling, 2013) enable learning a probability distribution on a latent space. Then, we can draw samples in the latent space – and the decoder can generate samples in the space of observations. In practice, the parameters of the latent distribution are learned by maximizing the *evidence lower bound* (ELBO) of the intractable likelihood:

$$\log p(\mathbf{x}) \geqslant -\mathrm{KL}(q(\mathbf{z}) \| p(\mathbf{z})) + \mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x} \mid \mathbf{z})] \tag{5.8}$$

where $\mathbf{z}$ is a sample in the latent space from the approximate posterior distribution $q(\mathbf{z})$, with a prior $p(\mathbf{z})$, and $\mathbf{x}$ is a reconstruction of a sample (e.g., an image) with the likelihood $p(\mathbf{x} \mid \mathbf{z})$. A common choice for q is $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\mu}$ and Σ are trainable parameters (Kingma and Welling, 2013).

5.2 Mixed Effects Neural ODE

Given a latent representation $\boldsymbol{z}$ of a time-varying process with $\boldsymbol{z} = (\mathbf{z}_0, \ldots, \mathbf{z}_n)$ and $\mathbf{z}_t \in \mathbf{R}^p$, we now model the latent representation $\boldsymbol{z}$ as a mixed effects neural ODE. We will assume each of the p variables to be independent, so we will seek to learn p mixed effects models. Without any loss of generality, below we assume $z \in \mathbf{R}$ to denote the latent representation of the time-varying process at a time point t.

Modeling random effect $\mathbf{b}^i$ in a network Γ. Mixed effects in the context of ODE is a well studied topic in longitudinal data analysis literature (Wang et al., 2014; Liang and Yu, 2013). Formally, for subject i, given z^i, $\mathbf{b}^i$, and (population level) fixed effects $\boldsymbol{\beta}$, we assume that there exist a smooth function h such that,

$$\dot{z}^i = h(z^i, \boldsymbol{\beta}, \mathbf{b}^i). \tag{5.9}$$

Due to the universal approximation properties of neural networks (Zhou, 2020), such models are a sensible choice to express the nonlinear function h. Recall the non-linear mixed effects model from (5.7), and let us model

$$h(z^i, \boldsymbol{\beta}, \mathbf{b}^i) = \nu\left(\eta(z^i)\boldsymbol{\beta} + U^i\mathbf{b}^i\right).$$

Here, $\eta : \mathbf{R}^n \to \mathbf{R}^{n\times m}$ is a non-linear function, with n and m being the number of observations and variables respectively and $\boldsymbol{\beta} \in \mathbf{R}^m$, $U^i \in \mathbf{R}^{n\times m}$. With a choice of $U^i = \eta(z^i)$, we can model $h(z^i, \boldsymbol{\beta}, \mathbf{b}^i) = \Gamma\left(z^i\right)\left(\boldsymbol{\beta} + \mathbf{b}^i\right)$, where Γ is a neural network with $\Gamma\left(z^i\right) \in \mathbf{R}^{n\times m}$. Now, we can derive the expressions for representing mixed effects in ODE, parameterized by a neural network as

$$h(z_t^i, \boldsymbol{\beta}, \mathbf{b}^i) = \Gamma\left(z_t^i\right)\mathbf{w}^i, \tag{5.10}$$

where $\mathbf{w}^i \sim \mathcal{N}(\boldsymbol{\beta}, \Sigma_b)$ is a mixed effect for subject i. This can be thought of as a projection from $\mathbf{R}^m$ to $\mathbf{R}$ along the direction given by $\mathbf{w}^i \in \mathbf{R}^m$.

Remark 5.1. *Observe the difference between standard SDE in* (5.1) *– where the noise is added at each step* t *– and our formulation, where* $\mathbf{w}^i$ *(or* $\mathbf{b}^i$*) is sampled once for subject* i *and completely defines the trajectory through* $h(z^i, \boldsymbol{\beta}, \mathbf{b}^i)$ *for all steps of time* t*. This is* crucial *from the computational perspective: with this strategy, we can simply apply existing ODE solvers whereas backpropagating through a blackbox SDE requires specialized solutions, which are typically slower* (*Li et al., 2020; Liu et al., 2019*).

Initializing ODE $h(z^i, \boldsymbol{\beta}, \mathbf{b}^i)$ with an encoder E. Often in real-world analysis tasks involving panel data, the initial point of the process z_0 is not observed. While it can be learned as a parameter (Huang et al., 2008), it is desirable to also provide uncertainty pertaining to the learned z_0. For this reason, we learn the distribution $q(z_0) = \mathcal{N}(\mu, \sigma)$, by training an encoder E to map observed data $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ (at all n time points) to parameters of $q(z_0)$, μ and σ:

$$(\mu, \sigma) = E(\mathbf{x}) \tag{5.11}$$

We use $q(z_0)$ to sample initial points of ODE z_0.

Mapping z to x via decoder D. Given the latent representation z and the non-linear function D, which can recover the output $\mathbf{x}_t^i$, for a subject i at time point t, we can model the output of the dynamic process (e.g., in our application, a brain image) $\mathbf{x}_t^i$ as a non-linear

transformation of the latent measure of progression z_t^i:

$$\mathbf{x}_t^i = D(z_t^i) + \boldsymbol{\epsilon}_t, \tag{5.12}$$

where $\boldsymbol{\epsilon}_t$ is measurement error at each time point. This idea has been variously used in the literature (Pierson et al., 2019; Hyun et al., 2016; Whitaker et al., 2017).

The final model. Combining Equations (5.11), and (5.12) we obtain our Mixed Effects Neural ODE model in (5.13) and illustrated in Figure 5.1.

$$\begin{bmatrix} z_0^i \sim \mathcal{N}(\mu, \sigma), \text{ where } \mu, \sigma = E(\mathbf{x}^i) \\ \mathbf{w}^i = \boldsymbol{\beta} + b^i \sim \mathcal{N}(\boldsymbol{\beta}, \Sigma_b) \\ \dot{z}_t^i = \Gamma\left(z_t^i\right) \mathbf{w}^i \\ \mathbf{x}_t^i = D(z_t^i) + \boldsymbol{\epsilon}_t \end{bmatrix} \tag{5.13}$$

Synopsis. Here, for subject i, we use the encoder E to map the observed data $\mathbf{x}^i$ to parameters of the distribution of ODE initialization z_0^i. Then, we parameterize the derivative of ODE $\dot{z}_t$ with a neural network $\Gamma(z_t^i)$ and mixed effects $\mathbf{w}^i$, and use D to map solution of ODE to the original space.

Structure of the latent space. When the latent space z can be embedded in a low dimensional space, it is natural to ask whether simulating the ODE $\dot{z}$ in (5.13) can be accomplished efficiently. The following result shows a link between our model in (5.13) and random projection ideas (Vempala, 2005). Moreover, in contrast to Neural ODE (Chen et al., 2018b), random projections allow using high dimensional representations, $\Gamma(z_t^i)$, and mapping it back to $\mathbf{R}$ using random projections. This provides expressive power but also approximately preserves the distance (using JL lemma).

Lemma 5.2 (Random projection)**.** *With a certain choice of* $h(z_t)$ *in the ODE formulation and given a mixed effect* $\mathbf{w}$ *with a choice of approximate posterior* $q(\mathbf{w})$ *as a Normal distribution, the solution to ME Neural ODE (Equation (5.13)) is a random projection of a solution to Neural ODE (Chen et al., 2018b), Figure 5.1 (right part).*

Proof. Let us use the following notations,

(a) $dz_t = f(z_t)\,dt$ defines trajectory of Neural ODE setup;

(b) $d\widetilde{z}_t = \Gamma(\widetilde{z}_t) \cdot w dt$ defines trajectory of our ME setup;

(c) $\widetilde{z}_t = z_t \cdot w$: random projection of z_t. Then we have,

$$d\widetilde{z}_t = w \cdot dz_t = w \cdot f(z_t)\, dt = f\left(\frac{\widetilde{z}_t}{w}\right) \cdot w dt$$
$$= \Gamma(\widetilde{z}_t) \cdot w db$$

If $\Gamma(\widetilde{z}_t) = f\left(\frac{\widetilde{z}_t}{w}\right)$, then $\widetilde{z}_t$ is random projection of z_t and $\widetilde{z}_t = z_t \cdot w$. □

While have a model, efficient training is still unresolved. Next, we show how for (5.13), we can derive ELBO-like bounds using Approximate Bayesian Computation (ABC) (Wilkinson, 2013; Fearnhead and Prangle, 2010).

Connection with approximation of Stratonovich's SDE. Observe that if we select f and g in (5.4) as $f(z,t) = \Gamma(z,t)\beta$ and $g(z,t) = \Gamma(z,t)\Sigma_b^{1/2}$, the approximation of Stratonovich's SDE becomes ME-ODE defined in (5.10), i.e., we set $\dot{z}_t = \Gamma(z,t)\mathbf{w}$, where $\mathbf{w} = \beta + b\Sigma_b^{1/2} \sim \mathcal{N}(\beta, \Sigma_b)$.

Note that our ODE/SDE based derivation of the expression in (5.10) coincides with the mixed effects form proposed in (Xiong et al., 2019) for single panel (time) data.

Remark 5.3. *Recall that in* (5.9)*, we assumed that* h *is smooth. In theory, the smoothness assumption is justified due to Wong-Zakai approximation, see* (5.4)*. In practice, this can be achieved by choosing a sufficiently fine discretization.*

Model training: ME-NODE ELBO

The objective of the training scheme we describe now is to learn **(a)** distribution of z_0, **(b)** fixed effect β, **(c)** variance of random effect Σ_b . To reduce clutter, in this section, we drop index i (which specifies a subject).

At a high level, our approach is to infer the random effects **b** by learning it as a parameter. For our purposes, learning **b** corresponds to ensuring that **b** satisfies the following key requirement (accounting for a small reconstruction error): **b** needs to be random by design for statistical reasons such as uncertainty quantification. Using our model in (5.13), it is easy to see that this requirement is satisfied because **b** is sampled from $\mathcal{N}(\mathbf{0}, \Sigma_b)$. A common strategy to satisfy the requirement is to use a VAE (Chen et al., 2016). It is known that in such probabilistic models computing the marginal likelihood $p(x)$ is usually intractable. Let $p(z, \mathbf{w})$ be the prior joint distribution, $q(z, \mathbf{w})$ as approximate joint posterior, and $p(x \mid z, \mathbf{w})$

as likelihood of reconstruction. Using concepts from Chapter 5.1, we can derive a lower bound for the $p(x)$ of our ME-NODE model as:

$$\log p(x) = \log \int p(x|z,\mathbf{w}) p(z,\mathbf{w}) \frac{q(z,\mathbf{w})}{q(z,\mathbf{w})} d(z,\mathbf{w}) \tag{5.14}$$

$$= \log \mathbb{E}_{q(z,\mathbf{w})} \left(p(x \mid z,\mathbf{w}) \cdot \frac{p(z,\mathbf{w})}{q(z,\mathbf{w})} \right)$$

$$\geqslant \mathbb{E}_{q(z,\mathbf{w})} \log p\,(x|z,\mathbf{w}) - \mathrm{KL}(q(z,\mathbf{w}) \| p(z,\mathbf{w})), \tag{5.15}$$

where (5.14) follows from the marginalization property and then we use Jensen's inequality. Next, we define $q(z,\mathbf{w})$ which is needed to compute ELBO. Note that in the following description, $\xi|\psi$ refers to the random variable ξ conditioned on a value of ψ, regardless of what the value is, i.e., it can be $\xi|\psi = 0$ or $\xi|\psi = 1$.

Defining $q(z,\mathbf{w})$. Assuming that z_0 and $\mathbf{w}$ are independent random variables, we get

$$q(z,\mathbf{w}) = q\,(z_0, z_{-0}, \mathbf{w}) = q\,(z_{-0} \mid z_0, \mathbf{w})\; q(z_0) q(\mathbf{w}).$$

Recall that $z_{-0} = (z_1, \ldots, z_n)$ is a vector of ODE solutions at time step t, except $t = 0$. At every step t, z_t is a random variable, which is a function of z_0 and $\mathbf{w}$. However, with a fixed initial point z_0 and mixed effect $\mathbf{w}$, the progression follows a *defined* trajectory (i.e., there is no randomness). It means that $z_t|z_0,\mathbf{w}$ is deterministic and hence the distribution $q(z_t|z_0,\mathbf{w})$ is degenerate (Danielsson, 1994), which results in $q(z_{-0}|z_0,\mathbf{w}) = \mathbb{1}_{z_{-0}^{\text{obs}}}\{z_{-0}|z_0,\mathbf{w}\}$. Thus,

$$q(z,\mathbf{w}) = \mathbb{1}_{z_{-0}^{\text{obs}}}\{z_{-0}|z_0,\mathbf{w}\} q\,(z_0)\; q(\mathbf{w}) \tag{5.16}$$

Note. While the derivation from (5.14) to (5.15) is well defined for point mass distributions stated in (5.16), the use of 'KL', although consistent with (Bai et al., 2020) (pp 3, (4)–(6)) is not ideal (because $\log(0)$ and thus KL is undefined). We will avoid using KL notation in the loss in (5.17).

MC approximation of $\mathbb{E}_{q(z,\mathbf{w})} g(z,\mathbf{w})$. The key in computing (5.15) is to estimate $\mathbb{E}_{q(z,\mathbf{w})} g(z,\mathbf{w})$ for a given function $g(z,\mathbf{w})$. Based on parameterization of $q(z,\mathbf{w})$ in (5.16),

$$\mathbb{E}_q g(z,\mathbf{w}) = \int_{z_0,\mathbf{w}} g(z,\mathbf{w}) \cdot \mathbb{1}_{z_{-0}^{\text{obs}}}\{z_{-0}|z_0,\mathbf{w}\} q\,(z_0)\; q(\mathbf{w}) dz_0 d\mathbf{w}.$$

While the integration may be intractable, it can be estimated by Monte Carlo (MC)

techniques. Sampling $(z_0^m, \mathbf{w}^m)$ from q, we compute $\frac{1}{M}\sum_{m=1}^{M} g^*(\mathbf{z}^m, \mathbf{w}^m)$, where

$$g^*(\mathbf{z}^m, \mathbf{w}^m) = g(\mathbf{z}^m, \mathbf{w}^m) \cdot \mathbb{1}_{\mathbf{z}_{-0}^{obs}}\{\mathbf{z}_{-0}|z_0^m, \mathbf{w}^m\}.$$

This type of sampling is called likelihood-free rejection sampling (Del Moral et al., 2012): we reject all samples (z_0^m and $\mathbf{w}^m$), which do not generate observed $\mathbf{z}_{-0}^{obs}$.

The final loss. Given this MC approximation (with M samples), with the approximate posterior $q(\mathbf{z}, \mathbf{w})$ defined in (5.16) and with a similarly defined prior $p(\mathbf{z}, \mathbf{w}) = \mathbb{1}_{\mathbf{z}_{-0}^{obs}}\{\mathbf{z}_{-0}|z_0, \mathbf{w}\}p(z_0)\,p(\mathbf{w})$, the final loss is

$$\frac{1}{|S|}\sum_{s\in S}\left(\log p\,(x|\mathbf{z}^s, \mathbf{w}^s) - \log\frac{q(z_0^s)q(\mathbf{w}^s)}{p(z_0^s)p(\mathbf{w}^s)}\right), \tag{5.17}$$

where S is a set: $\{\forall s \in S : \mathbb{1}_{\mathbf{z}_{-0}^{obs}}\{\mathbf{z}_{-0}|z_0^s, \mathbf{w}^s\} = 1\}$. The proof is provided in Appendix C.1.

Remark 5.4. *While a MC approximation in (5.17) is an unbiased estimator of the Lower Bound in (5.15), its variance is* $O\left(\frac{1}{|S|}\right)$. *This leads to efficiency issues in that it may require a large* M *until we get* z_0 *and* $\mathbf{w}$ *to generate* $\mathbf{z}_{-0}$ *exactly along the observed trajectory to populate the set* S. *But we can address this problem using ABC methods (Wilkinson, 2013; Fearnhead and Prangle, 2010).*

Efficient sampling: approximating $\mathbb{1}_{\mathbf{z}_{-0}^{obs}}\{\mathbf{z}_{-0}|z_0, \mathbf{w}\}$. ABC recommends finding samples of z_0 and $\mathbf{w}$ to generate trajectories $\mathbf{z}_{-0}$ which are *approximately* equal to the observed one, rather than exactly equal. The idea in (Marin et al., 2012) proposes using $\mathbb{1}_y\{z\}$ as $\mathbb{1}_{A_{\epsilon,y}}\{z\}$, where $A_{\epsilon,y} = \{z \mid d\{z, y\} \leqslant \epsilon\}$ is an ϵ-neighborhood of y, and d is a distance function. For a direct application of these methods on $\mathbb{1}_{\mathbf{z}_{-0}^{obs}}\{\mathbf{z}_{-0}|z_0, \mathbf{w}\}$, we must have access to $\mathbf{z}_{-0}^{obs}$ in the latent space, which is unavailable unless the encoder E and the decoder D are identity functions. Nonetheless, we can approximate $\mathbb{1}_{\mathbf{z}_{-0}^{obs}}\{\mathbf{z}_{-0}|z_0, \mathbf{w}\}$, by comparing if the decoded $\mathbf{z}_{-0}|z_0, \mathbf{w}$ indeed corresponds to $\mathbf{x}_{-0}^{obs}$, i.e., we need to compute $\mathbb{1}_{\mathbf{x}_{-0}^{obs}}\{D(\mathbf{z}_{-0}|z_0, \mathbf{w})\}$. We simply use the mean squared error (MSE) as the distance for this comparison.

By decreasing ε, we can improve the quality of the samples for MC estimation, but at higher compute cost. However, because we learn the distribution of z_0, during the first steps of training, our model provides poor reconstructions. For this reason, setting ε to a small value at the beginning of the training is inefficient. Therefore, we make ε adaptive through the training, by choosing the sample, closest to our observed trajectory, i.e., sample with smallest distance d, and ε is a function of the initial point z_0.

Choice of $q(\mathbf{w})$. To optimize the ELBO, it is necessary to define the approximate posterior $q(\mathbf{w})$ and prior $p(\mathbf{w})$. While we assumed in (5.10) and Lemma 5.2 that the true

distribution of $\mathbf{w}$ is Normal, $q(\mathbf{w})$ and $p(\mathbf{w})$ remain design choices for the user. For example, if we believe that the correlation structure of the data is sparse, then we have the following choices: Horseshoe (Carvalho et al., 2009), spike-and-slab with Laplacian spike (Deng et al., 2019) or Dirac spike (Bai et al., 2020). However, for our experiments, we found that modeling $q(\mathbf{w})$ and $p(\mathbf{w})$ as Normal is sufficient.

Calibration. One feature of our model is that learned distribution of mixed effects $\mathbf{w}$ can be used for personalized prediction during extrapolation (Wang et al., 2014; Ditlevsen and De Gaetano, 2005; Bouriaud et al., 2019). First, we train our model to learn the parameters of distribution of z_0, fixed effects $\boldsymbol{\beta}$, and variance of random effect Σ_b. Then at test time, we make use of the observed temporal data for a previously unseen test subject $\mathbf{x}^{\text{obs}}$. Given the learned distribution of mixed effects $\mathbf{w}$, we want to find a sample $w \sim \mathbf{w}$, which minimizes error w.r.t. $\mathbf{x}^{\text{obs}}$. This selection provides the most appropriate mixed effect w corresponding to $\mathbf{x}^{\text{obs}}$. We call this process calibration: a solution to $\text{argmin}_{w\sim\mathbf{w}}\, \text{MSE}\left(\mathbf{x}^{\text{obs}}, \hat{\mathbf{x}}(w)\right)$, where $\hat{\mathbf{x}}(w)$ is a prediction from our model. Note that this is slightly different from the average (used in probabilistic models like VAE).

Method summary. We provide a step-by-step summary,

Train and test phases

In the **training** phase, the observed data for subject i, x_t^i for $t \in [0, T]$ is assumed to be provided. Then, we

1. Use a suitable encoder E to map $\mathbf{x}^i = \{x_t^i\}$ to the latent representation of initial point z_0 of underlying ODE, as $z_0^i \sim \mathcal{N}(\mu, \sigma)$, where $\mu, \sigma = E(\mathbf{x}^i)$.
2. Given a suitable decoder D, we fit the ME-NODE model, by minimizing the loss in (5.17), thereby learning the appropriate distribution of mixed effects $\mathbf{w}^i$.

The output from this phase latent representation z_t^i described by ME-NODE model and the corresponding distribution of mixed effects $\mathbf{w}^i$.

In the **test** phase, the observed data for subject i, x_t^i for $t \in [0, T]$ is assumed to be provided. Then, we

1. Select a personalized mixed effect $w^i \sim \mathbf{w}^i$, according to the calibration scheme.
2. Then, we use the selected mixed effect sample w^i, to generate personalized prediction for the subject i for either interpolation or extrapolation.

The output from this phase is the prediction for subject i, $\hat{x}_t^i$ for $t \in [0, \ldots, T^*]$, where T^* can be extrapolated time, i.e., $T^* \geqslant T_i$, and/or a denser interpolation in $[0, T]$.

Estimated parameters	n_{z_0}, n_w				True values
	1, 1	1, 10	10, 1	10, 10	
$\hat{\mu}$	1.252	1.258	1.316	1.313	1.3
$\hat{\sigma}$	0.003	0.002	0.005	0.016	0.01
$\hat{\beta}$	0.311	0.315	0.311	0.319	0.3
$\hat{\sigma_b}$	0.036	0.051	0.054	0.060	0.01
MSE (all)	0.0017	0.0011	0.0006	0.0005	

Table 5.1: The first four rows show the estimated parameters for specific choices of n_{z_0} and n_w. Here we use the following convention: $(n_{z_0}, n_w) = (i, j)$ denotes we draw i and ij number of samples from z_0 and w respectively. The last row presents the MSE values for the estimated parameters.

5.3 Experiments

We evaluate our model on five temporal datasets: **(1)** simulations, **(2)** MuJoCo hopper, **(3)** rotating MNIST, and **(4)** two different Neuroimaging datasets, representing disease progression in the brain.

Goals. We will evaluate: **(a)** ability to learn mixed effects, given different types of correlations in the data **(b)** the effect of mixed effects dimension m on extrapolation power and confidence of the model, and **(c)** the ability to preserve statistical group differences in the data in latent representations.

The baselines are given separately for each experiment. We provide description of hardware and neural networks architectures, including architectures of encoders/decoders in Appendix C.2.

Synthetic dataset

We start with a synthetic setup where all parameters are known. Using a ODE solver and conditioning on z_0 and w, we generate a solution of the mixed effect ODE

$$\begin{cases} z_0^i \sim \mathcal{N}(\mu = 1.3, \sigma = 0.01) \\ w^i \sim \mathcal{N}(\beta = 0.3, \sigma_b = 0.01) \\ \dot{z}_t^i = z_t^i w^i \end{cases} \tag{5.18}$$

We set the encoder E and decoder D in (5.13) to the identity transformation. Given 1000 (80 : 20 split for train/test) numerical solutions of the ODE in (5.18), we uniformly sampled 20 time points from $[0, 3]$, and use the first 10 time steps for interpolation and the last 10 for extrapolation.

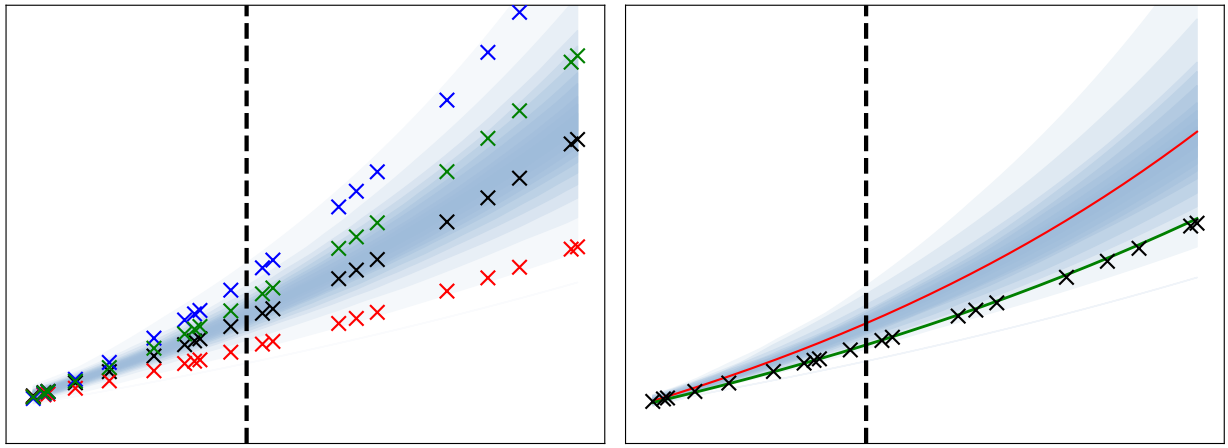


Figure 5.2: Real samples are denoted by “x" while blue lines are predictions. For each subfigure, the LHS and RHS of the dotted line contains interpolation and extrapolation results respectively. *Left:* the learned distribution of trajectories (from all test subjects) satisfies dynamic process described by the known system (5.1). *Right:* using the interpolation for calibration, we infer mixed effect w^i for i^{th} trajectory and generate a personalized prediction (green line). The prediction using posterior mean (commonly used in BNN) is shown in red.

Parameters. As the optimization of ELBO in (5.17) requires samples, we evaluate the performance of our model by varying the number of samples for z_0 and w (denoted by n_{z_0} and n_w respectively). The results in Table 5.1 suggest that MSE goes down with an increase of n_{z_0} or n_w.

In Figure 5.2 (*left panel*), we show samples (trajectories) drawn from the learned model (blue lines) with the real trajectories (‘x’ marker). Notice that the sampled trajectories from the learned model almost cover the “range” of real trajectories and the results appear meaningful.

Mixed effects. To evaluate generation of a personalized prediction for a subject i, by learning mixed effect w^i, recall that we split our data in two parts: interpolation (observed) and extrapolation (unknown). We use observed samples (interpolation part) to calibrate the mixed effect w^i and pass w^i to the selected trajectory during extrapolation. Figure 5.2 (*right panel*), shows that personalized prediction (green line) follows the observed data nicely across the entire time interval. In comparison, the taking the posterior mean of all trajectories (standard way of making prediction in BNN) generates trajectory close to observed data for interpolation, and fails to extrapolate as well as our proposed model.

Runtime. For 1000 samples, runtime for an epoch of our method is 2.1 seconds, while Neural SDE takes about 27.13 seconds for the same memory utilization (~965MB).

MuJoCo Hopper

Here, we evaluate the performance of our model for simple Newtonian physics. Similar to NODE (Rubanova et al., 2019), we created a physical simulation using MuJoCo Hopper. While in (Rubanova et al., 2019), the generated samples were i.i.d, we explicitly introduce correlation between the samples. The process of MuJoCo Hopper is defined by the initial position and velocity. In order to generate correlated samples, we choose the initial velocity from the pre-specified set containing $1, 4$ or 8 vectors. The entries of the velocity vectors are uniformly sampled from $[-2, 2]$. We evaluate our model on interpolation (10 steps) and extrapolation (10 steps) and compare results with NODE in Table 5.2. As our model implicitly learns correlation structure of the data by learning the distribution of mixed effect $q(\mathbf{w})$, we see an improvement in both interpolation and extrapolation. In addition, Figure 5.3 presents representative extrapolated samples using our proposed model.

		velocities		
	model	1	4	8
Interpolation	NODE	7.4	5.4	5.5
	This work	5.7	4.6	4.6
Extrapolation	NODE	166.1	82.1	80.3
	This work	164.1	81.2	80.0

Table 5.2: MSE (in scale of 10^{-3}) on MuJoCo Hopper data set, generated for three settings: 1, 4 and 8 initial velocities. We compare these two models using identical neural networks with the same number of levels and hyperparameters, however, in our model the dimension of mixed effect is $m = 50$.

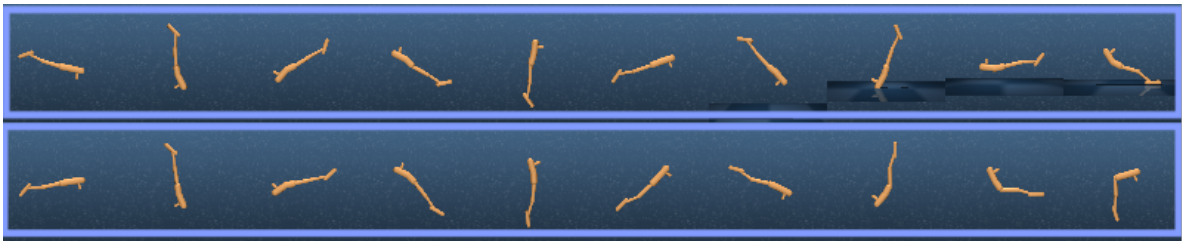


Figure 5.3: Visualization of 10 steps of extrapolation after observing 10 previous steps, with dimension of mixed effect $m = 50$. *Top* ground truth, *bottom* our prediction.

Rotating MNIST

We now evaluate a slightly more complicated rotating MNIST dataset. Here, we consider different types of correlations in the data and check: **(i)** relation between mixed effect

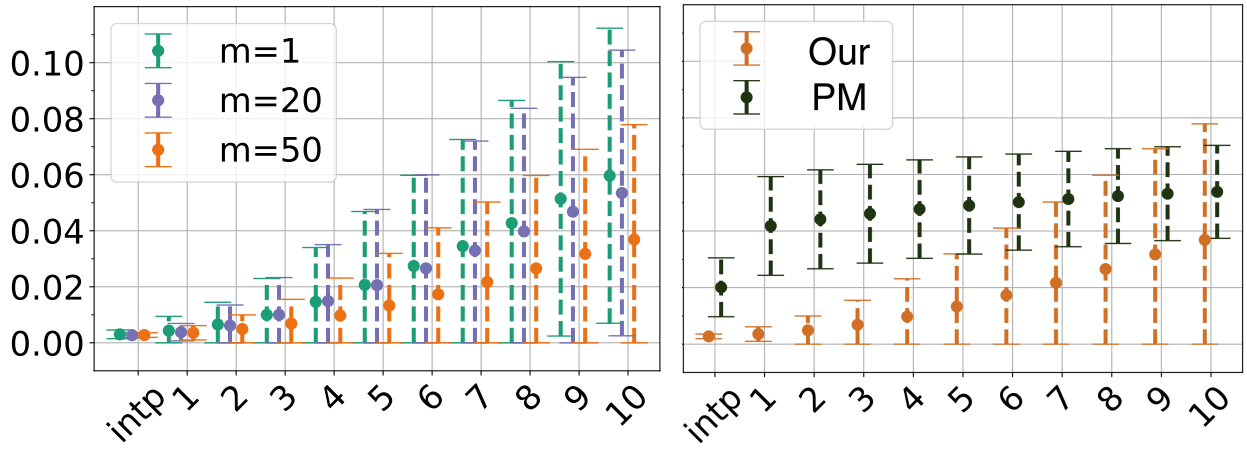


Figure 5.4: Distribution $(\mu \pm \sigma)$ of MSE: *left:* varying with m: $1, 20, 50$, *right:* 'Our' is our model with calibration, 'PM' is a posterior mean. First point on x-axis ('intp') indicates average MSE for all time steps: (1-10) of interpolation, and 1-10 indicate steps of extrapolation. Results for data with 8 possible angles shown.

dimension m (which we can think of as a dimension of random projection) and performance of the model, **(ii)** the performance of personalized prediction for extrapolation in comparison with posterior mean.

Data description. Similar to the setup in ODE2VAE (Yildiz et al., 2019), we construct a dataset by rotating the images of different handwritten digits, in order to learn a digit specific mixed effect model. In ODE2VAE, digits were rotated by 22.5°. We used a slightly different scheme: for a sampled digit we randomly choose an angle from the set of $1, 4$, or 8 angles from the range $[-\pi/4, \pi/4]$ and apply it at all time steps. For example, if we choose the set containing 4 angles, then a sampled digit is rotated using one of the 4 angles, selected randomly. In order to simulate a practical scenario, we spread out the initial points, by randomly rotating a digit by angles from $-\pi/2$ to $\pi/2$. We generate 10K samples of different rotating digits for 20 time steps and split it in two equal sets: interpolation and extrapolation.

Effect of mixed effect (random projection) dimension m. Recall from Lemma 5.2, we showed that mixed effects $\mathbf{w}$ in (5.13) can be considered as a random projection. So, we can expect that with an increase in m, MSE should decrease as it leads to a richer latent representation of a trajectory. We see that this is indeed true as shown in Figure 5.4 (*left panel*). Here, we demonstrate the MSE of reconstruction for three values of m: $1, 20$, and 50. We observe that for each time step, MSE decreases monotonically with an increase of m.

In addition, Figure 5.5 demonstrates visual comparison of one of the sample for different value of mixed effect dimension m. We find that while for any m interpolation frames

(frames at steps 1, 5, 10) look similar to real data, the further we move in extrapolation (frames at steps 11-20), the more noticeable the differences are. For example, for $m = 1$ some frames look blurry and in the last steps of extrapolation, the rotation is wrong. Increasing the dimension of random projection to $m = 20$ improves image quality, but does not fix rotation. Increasing dimension further to $m = 50$, not only improves quality of digits, but also leads to a better prediction of rotation.

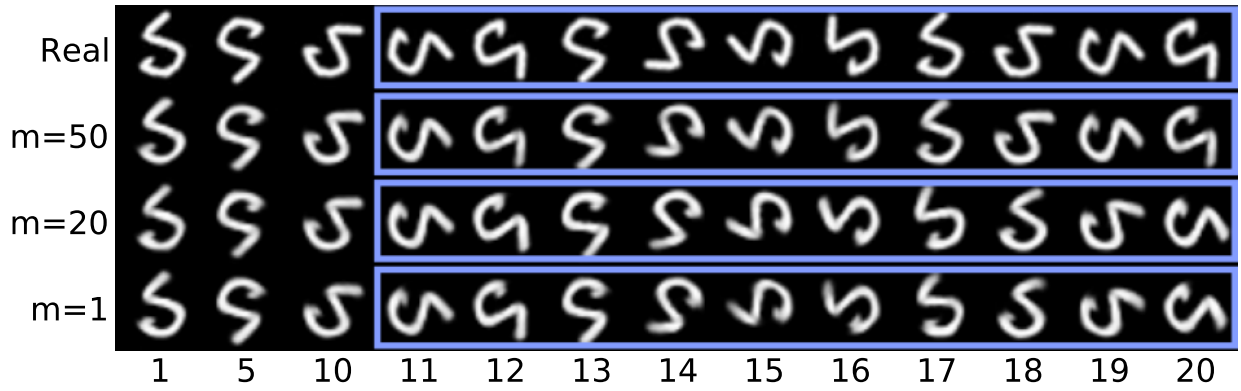


Figure 5.5: Visualization for one of the samples. From bottom to top we increase the dimension m of random projection (displayed on y axis), with real images on top row. Because of space limitation we show only 3 time steps of interpolation $(1, 5, 10)$ and all steps of extrapolation $(11 - 20)$ – blue frame.

Calibration for personalized prediction. Here we choose $m = 50$ and compare our model for interpolation with ODE2VAE using 3 different rotation settings: $1, 4$, and 8 available angles. The calibration results in Table 5.3 show that our model significantly outperforms the baseline ODE2VAE; however, making the correlation structure of the data more complicated (increasing number of possible rotation angles), does lead to a larger MSE. This is expected: with an increase in complexity of correlation in data, the learning task (and thereby, prediction) becomes harder.

		angles		
	model	1	4	8
Interpolation	ODE2VAE	0.0648	0.0644	0.0640
	Ours-50	0.0006	0.0014	0.0027

Table 5.3: MSE of two models, given different complexity of the data. Low interpolation error of our model indicates the properly learned mixed effect distribution $q(\mathbf{w})$.

Recall that in order to generate a personalized prediction for subject i we have to sample mixed effect $\mathbf{w}^i$ resulting in trajectory closed to the observed. Thus, if the model fails to

learn the distribution $q(\mathbf{w})$ accurately, sampling such $\mathbf{w}^i$ is less likely, and will result in a larger interpolation error.

For extrapolation, we compare with a posterior mean in Figure 5.4. We observe that for each extrapolation step, we obtain, on average, much smaller MSE and smaller variance during the initial steps of extrapolation.

In Figure 5.6 we provide visualization of 2 samples with the same digit style, but 2 different angles of rotation through interpolation and extrapolation, to compare the ability of models (our with calibration versus posterior mean (PM)) to capture different angles: slow (Figure 5.6, top) and fast (Figure 5.6, bottom). We see that with slow rotation interpolation on all 10 steps for PM is a little worse than our, but still sensible, while extrapolation is not good anymore. Same time, for fast rotated data even for interpolation PM provides worse results, and very bad for extrapolation. While our method is good for both.

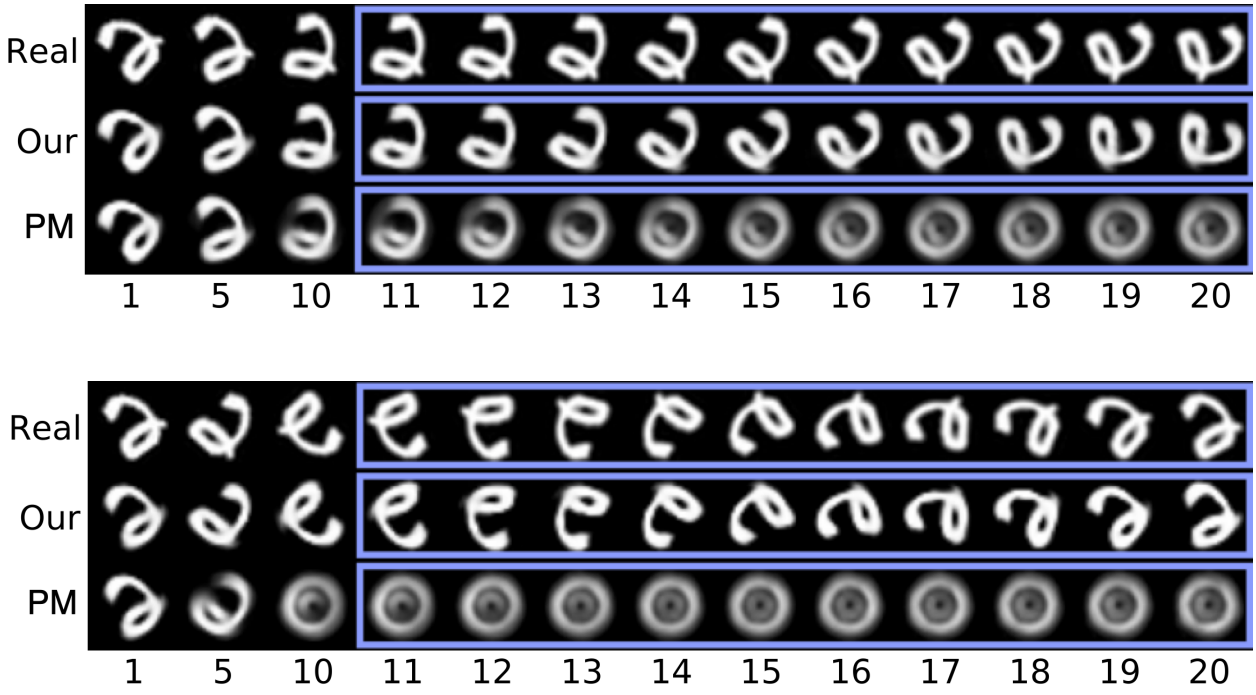


Figure 5.6: Notation: 'Our' is our model with calibration, 'PM' is a posterior mean. Visualization for two samples from data set with 8 possible angles (slow rotation–top, fast rotation – bottom). Because of space limitation we show only 3 time steps of interpolation(1, 5, 10) and all steps of extrapolation (11-20) – blue frame. We show the calibration effect of our model on extrapolation, compare to PM.

Extrapolation steps. Earlier, the number of steps for extrapolation were smaller than the number of observed steps, used for calibration. In Figure 5.7, we show results of

interpolation and extrapolation, varying the number of observed time steps used for calibration. Expectedly, decreasing the number of steps to be small for calibration yields a smaller number of steps where the extrapolation is meaningful.

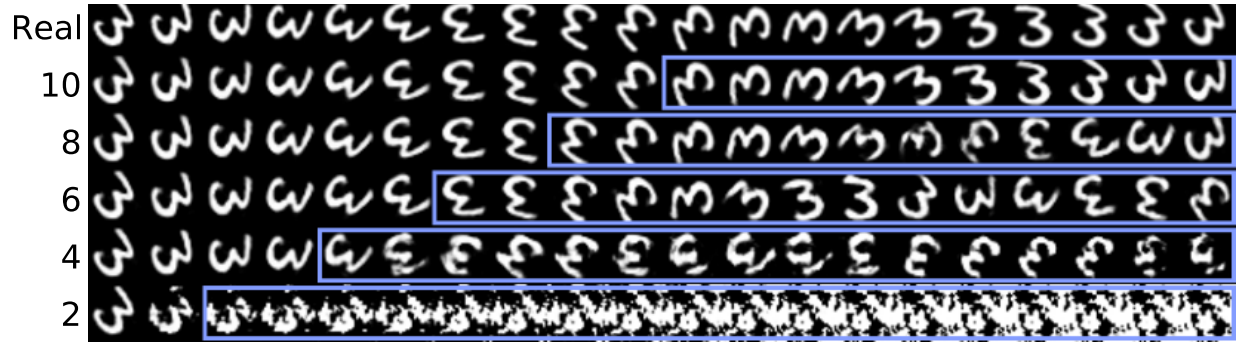


Figure 5.7: Visualization of extrapolated results (blue frame), given n time steps for calibration, where n is shown on y axis. The top row indicate real data.

Longitudinal Neuroimaging data

In this section, we conduct experiments on two longitudinal brain imaging datasets obtained from Alzheimer's Disease Neuroimaging Initiative (ADNI) (`adni.loni.usc.edu`), both of which describe AD progression through time, but are derived from two different imaging modalities.

(A) TADPOLE. TADPOLE dataset includes data for 276 participants with 3 time points. It represents Florbetapir (AV45) Positron Emission Tomography (PET) scans, which measure the level of amyloid-beta pathology in the brain (Marinescu et al., 2018). Scans were registered to a template (MNI152) to derive the 82 gray matter regions. Thus, each sample, at time t is a 82 dimensional vector, i.e., $\mathbf{x}^t \in \mathbf{R}^{82}$.

Given a few time points (3 time points), we evaluate generation of a personalized prediction for a subject in an interpolation setting. We compare our personalized prediction with ground truth and prediction using a posterior mean approach. The predictions of both our model and posterior mean are based on samples from the same distribution $q(\mathbf{z}, \mathbf{w})$. However, Figure 5.8 shows that the calibration of our model (Figure 5.8, *second row*) provides better prediction than posterior mean (Figure 5.8, *third row*). Even though the learned distribution of mixed effects $q(\mathbf{w})$ is capable of providing the correct trajectory (calibrated prediction), the direct application of the model without personalized calibration (posterior mean) leads to high subject-wise uncertainty.

(B) ADNI. Our second dataset from ADNI contains processed MRIs (3D brain scans) of size $105 \times 127 \times 105$ per subject at 3 time steps. The subjects are divided into two

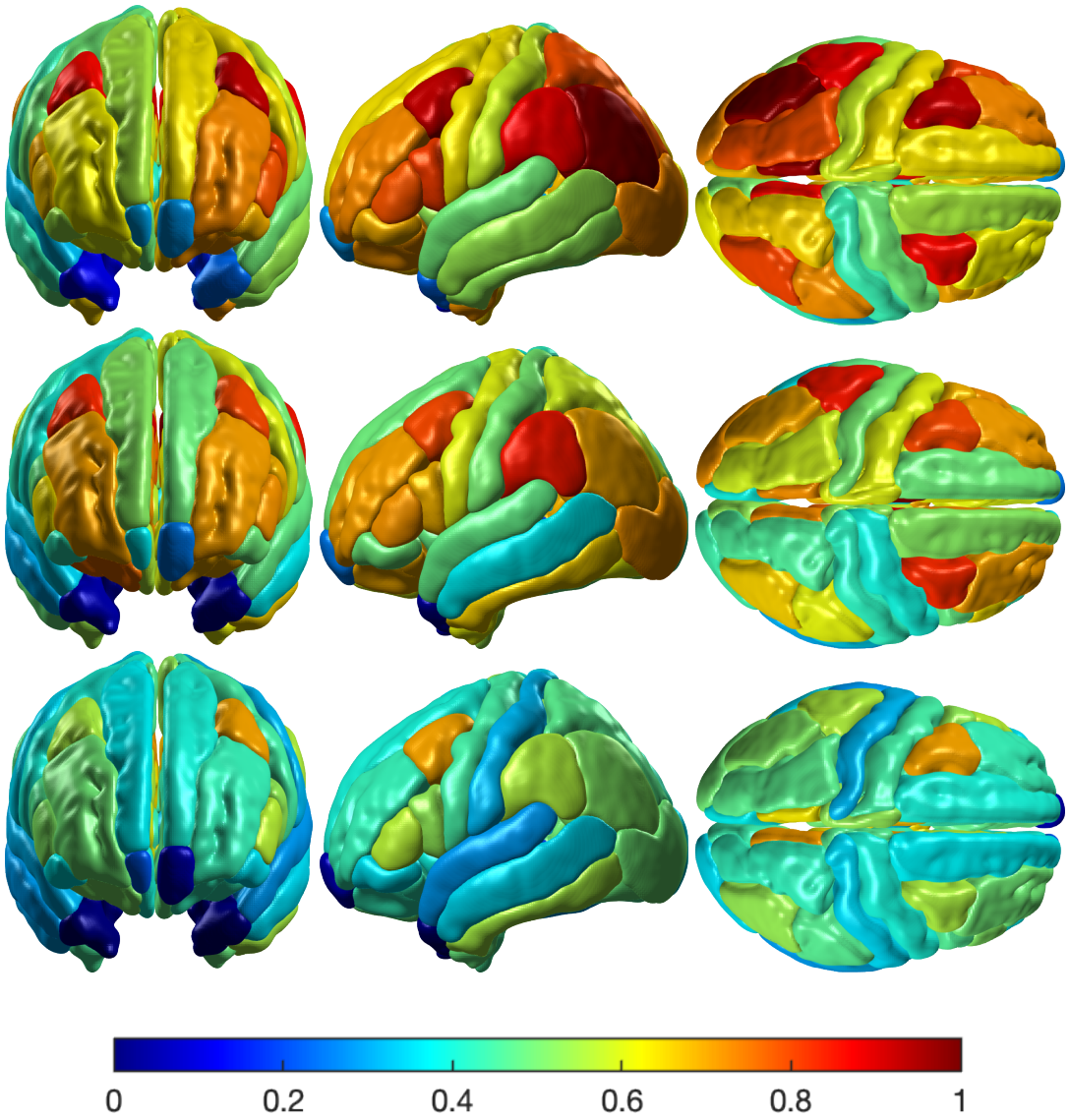


Figure 5.8: *Top - bottom row:* ground truth, our predicted, posterior mean predicted. We present the state of the brain at time step t = 3. Red and blue indicate high and low AV45 respectively. Compared to the baseline, our model is able to generate a sample for the subject with predicted values of AV45 closer to the ground truth.

groups: diagnosed with Alzheimer's disease (abnormal: 377 subjects) and healthy controls (normal: 152 subjects).

Interpolation. Given high resolution 3D images, we would like to evaluate whether our model is able to learn the distribution of mixed effects and perform calibration for personalized prediction. Similar to TADPOLE, we conduct an interpolation experiment and provide representative samples of brain images in Figure 5.9. We find by inspecting the axial/sagital/coronal views that our model yields meaningful brain images.In Figures 5.10 and 5.11, we provide a sample of our model, comparing with posterior mean. To evaluate

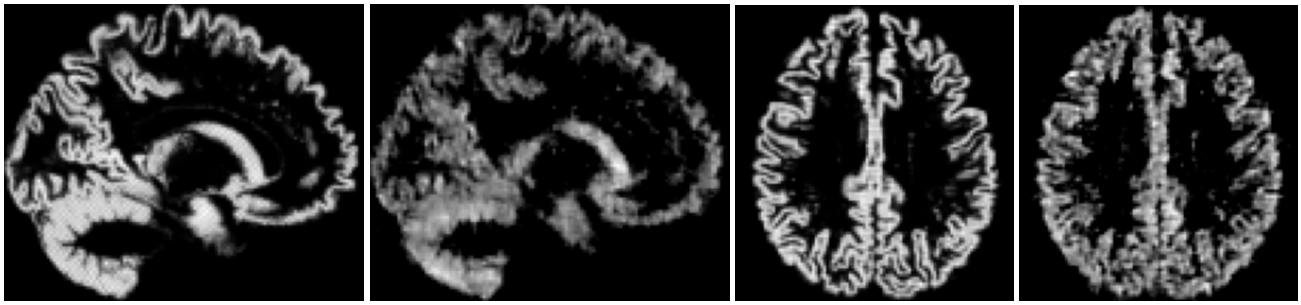


Figure 5.9: *Left-Right* Ground truth (1, 3) and prediction (2, 4) of two slices (side and top) of 3D brain at time point 3.

the result visually, we provide a difference between real and prediction in a 3rd column for both figures. We see that our model gives much better results in prediction, than posterior mean.

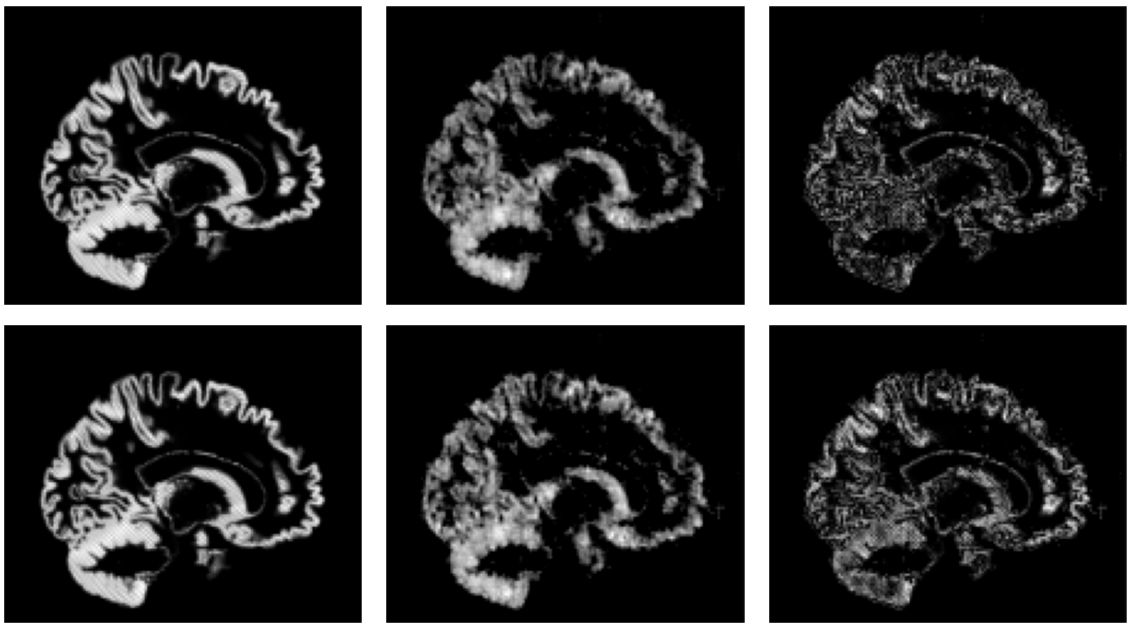


Figure 5.10: **Top:** Our method, **Bottom:** Posterior mean. **From left:** Truth, prediction, difference between truth and prediction. According to difference (3 column), our method performs better than Posterior mean.

Extrapolation. To evaluate the extrapolation capability, given limited number of time steps (only 3), we perform a statistical test. Recall that our method explicitly models the mixed effect term inside the trajectory to learn the data hierarchy. If our method works as intended, there should be a statistical difference between latent space of trajectories for normal and diseased/abnormal groups. Ideally, this difference should be preserved for several more extrapolation steps. To check this, we train our model on 3 time points.

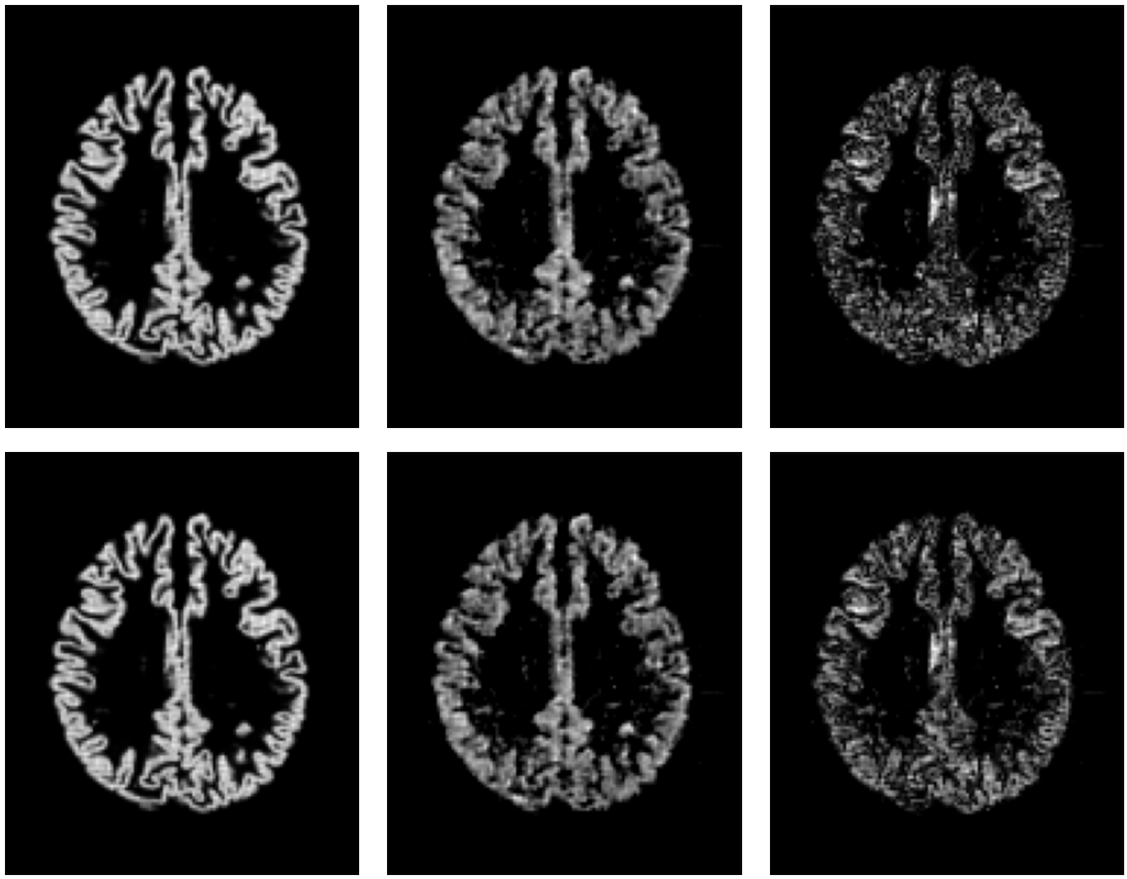


Figure 5.11: **Top:** Our method, **Bottom:** Posterior mean. **From left:** Truth, prediction, difference between truth and prediction. According to difference (3 column), our method performs better than Posterior mean.

During testing, we use 3 observed time points for calibration and extrapolate for 5 more time steps: we get latent trajectories defined for 8 (3 interpolation and 5 extrapolation) time points. Finally, the resultant trajectories are used to evaluate differences between normal and abnormal groups via a permutation test. The resultant distribution of distances and p-values for interpolation and extrapolation is in Figure 5.12. As expected, for interpolation and some steps of extrapolation (up to step 7) differences between trajectories is significant (with p-value $\leqslant 0.1$), and becomes less significant with more extrapolation steps.

5.4 Summary

We proposed a novel ME-NODE model that enables us to incorporate both fixed and random effects for analyzing the dynamics of panel data. Our evaluations on several different tasks show that the ME-NODE loss function can be trained using existing ODE

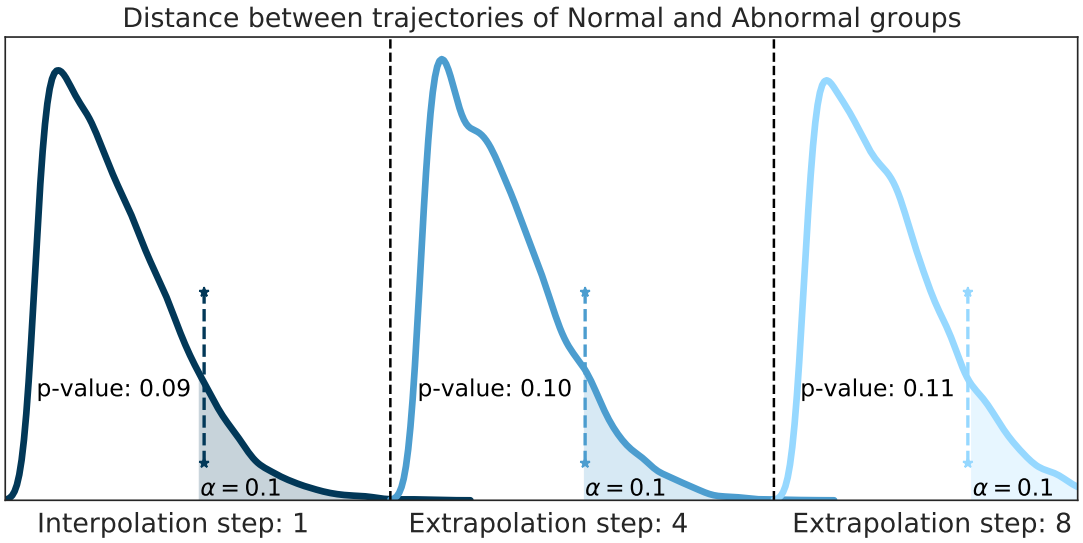


Figure 5.12: Distribution of distances resulted from permutation test with original distances (dashed lines), and corresponding p-values. The difference between trajectories is significant up to the 7^{th} extrapolated step and degradation from 8^{th}.

solvers in a stable and efficient manner. We see various benefits from incorporating mixed effects, (**1**) model explicitly learns the correlation structure of the data during the training, which improves prediction accuracy in setups where samples can be grouped by some criteria; (**2**) in contrast to generative models where only initial point is sampled from the distributions, by fixing initial point z_0 we can still provide uncertainty of the predictions, because from one initial point we can sample different trajectories; (**3**) since our model learns random effects for individual i, it allows personalized prediction, given a short history of data, which is useful in biomedical or scientific applications with a limited number of time points per individual along trajectory. One limitation of our approach is that there is not an explicit noise handling mechanism for test time calibration and prediction. This is problematic in large scale high dimensional settings. For example, say that the encoding distribution $\mathcal{N}(\mu, \Sigma)$ produces a small fraction of noisy trajectories. Even in small noise settings, filtering them for robust personalized prediction requires solving complex optimization problem (Bakshi and Kothari, 2021) and so handling noise is especially an open problem in real time, edge deployments. This chapter was published in yhe Conference on Uncertainty in Artificial Intelligence (UAI) as (Nazarovs et al., 2021a). The code is available at `https://github.com/vsingh-group/panel_me_ode`.

Additional applications

In Chapter 5, we conducted multiple experiments on synthetic and scientific data sets to perform interpolation and extrapolation of temporal trajectories. To extend the application of Mixed Effects Neural ODE, we collaborated with researchers from John Hopkins University, to understand the relationship of APOE genotypes and level of cognitive reserve to cognitive trajectories among cognitively normal individuals (Pettigrew et al., 2023). In our analysis we consider a longitudinal mixed effect model, similar to Mixed Effects Neural ODE, to learn more about data dependencies and how appropriate an application of mixed effects model. Results suggest that APOE-ε4 and non-APOE-ε4 AD polygenic risk are independently associated with global cognitive. Executive function declines among individuals APOE-ε4 is associated with declines in episodic memory. Importantly, higher levels of cognitive reserve (CR) may mitigate APOE-ε4-related declines in some cognitive domains. You can find more about our founding regarding APOE-4 and relation to global cognitive and episodic memory in the corresponding paper (Pettigrew et al., 2023).

Chapter 6

Variational Sampling of Temporal Trajectories

As we have seen in Chapter 5, learning to predict and infer from temporal/sequential data is an important topic in machine learning, with numerous application in many settings, including semantic video processing (Nilsson and Sminchisescu, 2018) in vision and text generation/completion (Lewis et al., 2019; Radford et al., 2018) in natural language processing. The literature has continued to evolve, from handling evenly-sampled discrete observations (e.g., LSTM (Hochreiter and Schmidhuber, 1997), GRU (Cho et al., 2014)) to irregularly-sampled or even continuous observations (e.g., Neural ODE (Chen et al., 2018b) / SDE (Rubanova et al., 2019; Tzen and Raginsky, 2019)). These capabilities have been extensively used, from learning complex non-linear tasks such as language modeling (Devlin et al., 2018) to forecasting of physical dynamics (Lusch et al., 2018; Brunton et al., 2020; Rudy et al., 2017).

Despite the mature state of various methods currently available for prediction tasks involving temporal sequences as we seen in Chapter 5, fewer options exist that allow sampling from a distribution of trajectories, similar to VAE for standard images. Existing methods, such as Neural ODEs, facilitate the ability to sample trajectories by varying the initial conditions of the underlying differential equation. However, these methods are limited in their ability to model real-world systems where the same initial conditions can result in vastly different outcomes, (e.g., common in reinforcement learning). In particular, brain development of identical twins can differ despite starting from exactly the same state (Strike et al., 2022). Another example is that two individuals with similar PiB/amyloid load may have different trajectory in terms of pathology accumulation. To address this limitation, Bayesian versions of Neural ODEs, similar to stochastic differential equations, can provide a suitable model by accounting for uncertainty in trajectories (see Chapter 5). However,

these methods are not designed to handle a wide variety of different trajectories with the same initial point and primarily not used as methods to sample dynamical processes. How can we address the problem of sampling a wide variety of different trajectories with the same initial condition? In other words, given an NODE setup in (6.1), where f is NN, can we learn a mechanism, which allows us to sample f as an element from a functional space of NN?

At a high level, one inspiration of our work is a recent result dealing with explicit embeddings of neural network functions (Dupont et al., 2022). We can treat the transition function f of the trajectory as an entity on its own and embed it explicitly in a Euclidean space. There, we can conveniently learn the distribution of embeddings which in-turn help us learn the distribution of the trajectories. Then, we can directly estimate the likelihoods and even sample new trajectories, with little to no overhead. Ideally, the embedding space will also have far fewer dimensions compared to the dimension of the data, which can make the distribution easier to learn. While Dupont et al. (2022) have explored learning the distribution of data-generating neural network functions through explicit embeddings termed *functa*s, our work extends such an idea to infer temporal trajectories by jointly constructing an appropriate functional embedding space for trajectories and learning a variational statistical model on top.

Contributions. We study a simple mechanism to embed the trajectory of temporal processes and show that one can directly perform variational inference of trajectories in this embedding space. The end result is a probabilistic model that allows efficient sampling and likelihood estimation of novel trajectories, which is immediately useful for simulation and for applications that requires statistical inference, such as outlier detection. We show via experiments that the proposed framework can achieve competitive performance in prediction tasks compared with conventional temporal models, while benefiting from the additional capabilities described above.

6.1 Notation and review

Notation: We denote a time-varying vector with T time steps as $z = (z_0, \ldots, z_{T-1})$, where each time step z_t is a vector valued sample, specifically $z_t \in \mathbf{R}^p$, where p is the number of covariates/channels.

(Neural) Ordinary Differential Equations: Recall that in Chapter 5 we introduced Neural ODE, the backbone of our model, Mixed Effects Neural ODE. Here we refresh the definition. We consider the ODE as $\dot{z}_t = f(z_t, t)$ to model transitions, where $f(z_t, t)$ represents a change in state z_t at time step t, and depends on a value of a state of the process

at time t. Given the initial time t_0 and target time t, ODEs compute the corresponding state z_t by performing the following operations:

$$z(t_0) = z_0, \quad z(t) = z(t_0) + \int_{t_0}^{t} f(z_t, t)dt \tag{6.1}$$

Parameterized neural networks have been shown to model $f(z_t, t)$ for complex nonlinear dynamics as in (Chen et al., 2018b; Kidger et al., 2020).

Learning latent representations with a VAE: One of our goals is to learn a latent representation of a temporal trajectory, and so we focus on variational approximation techniques. Recall that Variational auto-encoders (VAE) (Kingma and Welling, 2013) learn a probability distribution on a latent space. We can draw samples in the latent space and the decoder can generate samples in the space of observations. In practice, the parameters of the latent distribution are learned by maximizing the *evidence lower bound* (ELBO) of the intractable likelihood:

$$\log p(\mathbf{x}) \geqslant -\mathrm{KL}(q(\mathbf{z})\|p(\mathbf{z})) + \mathbb{E}_{q(\mathbf{z})}[\log p(\mathbf{x} \mid \mathbf{z})] \tag{6.2}$$

where $\mathbf{z}$ is a sample in the latent space from the approximate posterior distribution $q(\mathbf{z})$, with a prior $p(\mathbf{z})$, and $\mathbf{x}$ is a reconstruction of a sample (e.g., an image or temporal sample) with the likelihood $p(\mathbf{x} \mid \mathbf{z})$. One choice for q is $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\mu}$ and Σ are trainable parameters (Kingma and Welling, 2013). Note, $\mathbf{z} \in \mathcal{Z}$ denotes a latent space sample and $\mathbf{x} \in \mathcal{X}$ denote an observed space sample.

6.2 Variational Sampling of Temporal Trajectories

We start with deterministic temporal processes which can be defined by the corresponding trajectory on the underlying space $\mathcal{Z}$. A trajectory can be identified by an element in the product space of (a) initial condition $z_0 \in \mathcal{Z}$ (often given) and (b) transition function $f : \mathcal{Z} \times [0, T] \to \mathcal{Z}$ often influenced by the control of the underlying dynamical system. Thus, to learn the distribution of the temporal process, it is necessary to learn distributions over both the initial condition z_0 and the transition function f.

One way to model the transition function between states z_t of the temporal process is by using the ODE: $\dot{z}_t = f(z_t, t)$. We know that one can parameterize $f(z_t, t)$ by a neural network (NN) $f_\theta(z_t, t)$ to leverage their rich function approximation capacity (Chen et al., 2018b). Thus, we can (possibly) use a NN to fully describe a trajectory. *However, if we want to sample from this space of trajectories using the above NN parameterization, we need to learn a*

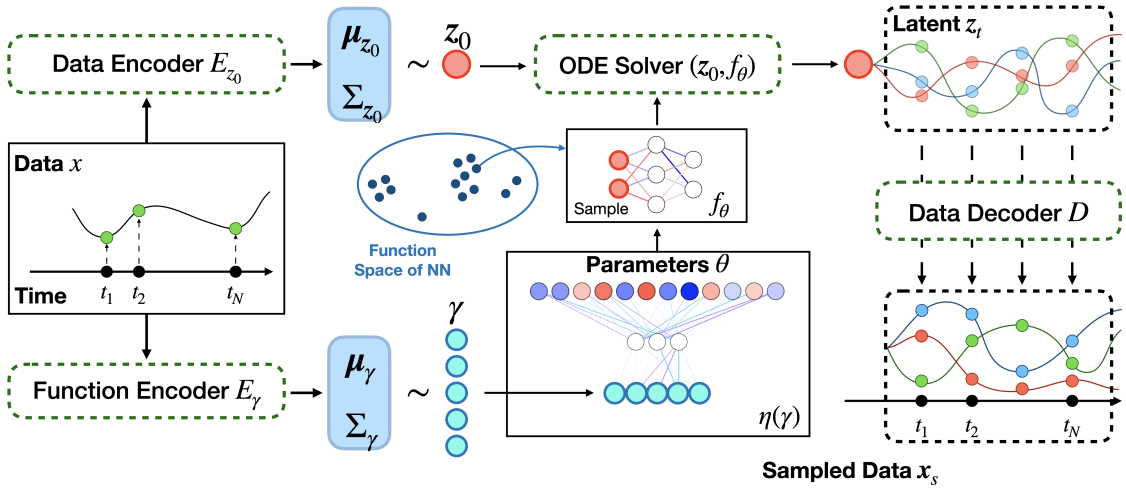


Figure 6.1: Structure of the model. First, Data Encoder is applied to temporal data sample to generate the initial point of the trajectory in latent space z_0; At the same time, the Function Encoder (embedding module) is applied to generate parameterization θ of the sample from function space, f_θ. The sample from function or embedding space, f_θ, is used to describe together with a differential equation solver, to solve function DE and generate z_t. Lastly, decoder is used to map latent space ODE stages into observed values.

latent representation of a NN, analogous to INR (*Dupont et al., 2022*).

Functional Neural ODE

Goal: Given a dataset of N temporal observations with length T,

$$\{\mathbf{x}^{(i)}\}_{i\in[N]} = \{\mathbf{x}_0^{(i)}, \ldots, \mathbf{x}_{T-1}^{(i)}\}_{i\in[N]},$$

we aim to learn a set of parameters $(\phi_E, \phi_D, \phi_\gamma)$ corresponding to data encoder/decoder (E_{z_0}, D) and function encoder E_γ (as shown in Figure 6.2) such that $p(\mathbf{x}^{(i)})$ is maximized. Formally, we can solve the following problem

$$\max \frac{1}{N}\sum_{i=1}^{N} p(\mathbf{x}^{(i)}) \equiv \max_{\phi_D} \frac{1}{N}\sum_{i=1}^{N} p_{\phi_D}(\mathbf{x}^{(i)}|z_0, f_\theta)p(z_0, \theta) \tag{6.3}$$

where z_0 is the initial condition, and f_θ is a NN with weights θ inferred from the data. Similar to VAE, since we do not know the true posterior distribution $p(z_0, \theta)$, we replace it with a variational distribution $p(z_0, \theta) \approx q_{\phi_E}(z_0|\mathbf{x}^{(i)})q_{\phi_\gamma}(\theta|\mathbf{x}^{(i)})$. The resulting objective is

$$\max_{\phi_E, \phi D, \phi\gamma} \frac{1}{N}\sum_{i=1}^{N} p_{\phi_D}(\mathbf{x}^{(i)}|z_0, f_\theta)q_{\phi_E}(z_0|\mathbf{x}^{(i)})q_{\phi_\gamma}(\theta|\mathbf{x}^{(i)}) \tag{6.4}$$

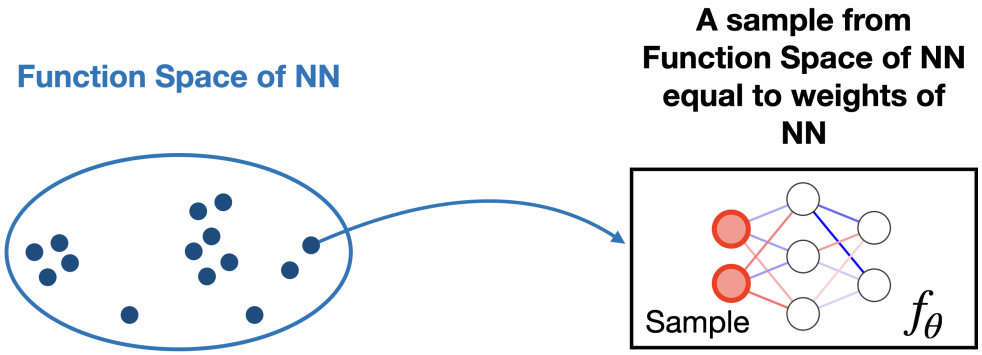


Figure 6.2: Function/embedding space, where each element is a DNN.

Sampling the transition function f_θ: In contrast to typical Neural ODEs (where $f_\cdot$ is a neural network with parameters θ), we do not tie $f_\cdot$ to a specific parameter $\theta \in \Theta$, where Θ is the space of parameters for f_θ. Instead, we model θ as a random variable that conditionally depends on $\mathbf{x}_0$. We motivate our design choice by an idea that each observed process is defined in a latent representation by its own transition function f_θ, and not only initial value z_0. Since θ consists of all parameters in f_θ and therefore is high dimensional, we choose an additional low-dimensional embedding $\gamma \in \Gamma$ to map to Θ with mapping function η such that $\theta = \eta(\gamma)$.

Now, analogous to VAE, we model $q_{\phi_\gamma}(\gamma|\mathbf{x})$ as a Gaussian distribution and learn its parameters, i.e., $(\boldsymbol{\mu}_\gamma, \Sigma_\gamma)$, by training an encoder E_γ. The encoder, E_γ essentially learns to map an observed data $\mathbf{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_{T-1})$ (at all T time points) to $(\boldsymbol{\mu}_\gamma, \Sigma_\gamma)$:

$$(\boldsymbol{\mu}_\gamma, \Sigma_\gamma) = E_\gamma(\mathbf{x}) \tag{6.5}$$

Remark: We use continous activation functions in f_θ which ensures that any instantiation of the NN is a continuous function. Peano existence theorem guarantees the existence of at least one solution to the above ODE locally.

We have now collected the components to (a) sample from $q(\gamma|\mathbf{x})$ and (b) map the embedding γ to the NN parameter θ by learning η. This gives us a sampler to sample f_θ to generate a solution of the corresponding Neural ODE.

Estimate initial condition z_0 with a Data Encoder E_{z_0}: Similar to the process of learning the latent representation of the transition functions, we learn the distribution of initial values z_0 as $q_{\phi_E}(z_0|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_{z_0}, \Sigma_{z_0})$, by training an encoder E_{z_0} to map observed data $\mathbf{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_{T-1})$ (at all T time points) to the latent space of $\mathcal{Z}$ whose samples are denoted by z_t via parameters $(\boldsymbol{\mu}_{z_0}, \Sigma_{z_0})$:

$$(\boldsymbol{\mu}_{z_0}, \Sigma_{z_0}) = E_{z_0}(\mathbf{x}) \tag{6.6}$$

Now, with the initial point, z_0 and transition function $f(z_t, t)$ in hand, the remaining step is to map z_t to the corresponding $\mathbf{x}_t$ in the original data space.

Mapping z to x via decoder D: Given the initial value z_0 and transition function $f(z_t, t)$, we can use existing ODE solvers to compute z_t. Once we obtain z_t using a non-linear function D, we recover the corresponding output $\mathbf{x}_t$, at time t. We model the output of the dynamic process $\mathbf{x}_t$ as a non-linear transformation of the latent measure of progression z_t as:

$$\mathbf{x}_t = D(z_t) + \boldsymbol{\epsilon}_t, \tag{6.7}$$

where $\boldsymbol{\epsilon}_t$ is measurement error at each time point. As we mentioned in Chapter 5, the idea of representing observed data with encoder/decoder and latent space has been variously used in the literature (Pierson et al., 2019; Hyun et al., 2016; Whitaker et al., 2017).

The final model: Combining (6.5), (6.6), and (6.7) we get our Functional ODE model in (6.8), see Figure 6.1.

$$\begin{cases} z_0 \sim \mathcal{N}(\boldsymbol{\mu}_{z_0}, \Sigma_{z_0}), \text{ where } \boldsymbol{\mu}_{z_0}, \Sigma_{z_0} = E_{z_0}(\mathbf{x}) \\ \gamma \sim \mathcal{N}(\boldsymbol{\mu}_{\gamma}, \Sigma_{\gamma}), \text{ where } \boldsymbol{\mu}_{\gamma}, \Sigma_{\gamma} = E_{\gamma}(\mathbf{x}) \\ \theta = \eta(\gamma) \mapsto f_\theta \\ \dot{z}_t = f_\theta(z_t, t) \\ \mathbf{x}_t = D(z_t) + \boldsymbol{\epsilon}_t \end{cases} \tag{6.8}$$

Summary. In contrast to Neural ODE and its variants, for a data $\mathbf{x}$, we not only use the encoder E_{z_0} to map the observed data $\mathbf{x}$ to parameters of the distribution of ODE initialization z_0, but we also utilize $\mathbf{x}$ and E_γ to sample from function (or embedding) space of NN, characterizing trajectories. Given samples z_0 and γ (and thus corresponding θ of $f_\theta(z_t, t)$), the z_t are computed as a solution to corresponding ODE, and D is used to map z_t to $\mathbf{x}_t$ in the original space. Figure 6.1 shows an overview. We now show how we can derive ELBO-like likelihood bounds used to train the model in (6.8).

ELBO for Functional ODE

The training scheme seeks to learn (**a**) a distribution of z_0, (**b**) a distribution of γ, which is used to get the corresponding θ by learning (**c**) a mapping function η, represented by NN.

At a high level, our approach is to infer the subspace of the function space, samples from which describe the underlying trajectories, f_θ, of the observed temporal process $\{\mathbf{x}_t\}$. We want to learn two components: an underlying simpler distribution for γ, i.e., $q_{\phi_\gamma}(\gamma|\mathbf{x})$, and mapping η, which maps γ to parameters θ, in-turn defining a sample, f_θ, from the function/embedding space of NNs.

A common strategy to learn the underlying distributions, $q_{\phi_\gamma}(\gamma|\mathbf{x})$ and $q_{\phi_E}(z_0|\mathbf{x})$, is to use a VAE (Chen et al., 2016). It is known that in such probabilistic models, one wants to optimize the likelihood of data $\mathbf{x}$, but computing this likelihood $p(\mathbf{x})$ is usually intractable. Thus, we use a lower bound as described next.

Let $p(z_0, \gamma)$ be the joint prior distribution, $q(z_0, \gamma)$ be an approximate joint posterior. Let $p(\mathbf{x} \mid z_0, \gamma)$ be the likelihood of reconstruction, where $\mathbf{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_{T-1})$ is a function of the latent representation $\{z_t\}$. Note that we have two sources of randomness: initial value of the latent trajectory z_0, and a latent representation, γ, of a sample from function space of NNs. Using the marginalization property of probabilities, we can write the log likelihood of the data as $\log p(\mathbf{x}) = \log \int p(\mathbf{x}, z_0, \gamma) d(z_0, \gamma)$, here $d(.,.)$ is the measure for integral. Using concepts from §6.1, we can derive a lower bound for the $p(\mathbf{x})$ of our Function NODE model as:

$$\begin{aligned} \log p(\mathbf{x}) &= \log \int p(\mathbf{x}|z_0, \gamma) p(z_0, \gamma) \frac{q(z_0, \gamma)}{q(z_0, \gamma)} d(z_0, \gamma) && (6.9) \\ &= \log \mathbb{E}_{q(z_0,\gamma)} \left(p(\mathbf{x} \mid z_0, \gamma) \cdot \frac{p(z_0, \gamma)}{q(z_0, \gamma)} \right) \\ &\geqslant \mathbb{E}_{q(z_0,\gamma)} \log p\left(\mathbf{x}|z_0, \gamma\right) - \mathrm{KL}\left(q(z_0, \gamma) \| p(z_0, \gamma)\right), && (6.10) \end{aligned}$$

where (6.9) follows from the definition of joint distribution and using Jensen's inequality.

Next, we define $q(z_0, \gamma)$ and $p(z_0, \gamma)$, which is necessary to optimize the above lower bound (ELBO).

Choice of $q(z_0, \gamma)$: Recall that we identify the trajectory as a product of initial value (distribution $q_{\phi_E}(z_0|\mathbf{x})$) and transition function (distribution $q_{\phi_\gamma}(\gamma|\mathbf{x})$). Thus, we want to find disentangled distribution of z_0 and γ. This leads to a form where $q(z_0, \gamma) = q_{\phi_E}(z_0|\mathbf{x}) q_{\phi_\gamma}(\gamma|\mathbf{x})$. Different choices are available to define $q_{\phi_E}(z_0|\mathbf{x})$ and $q_{\phi_\gamma}(\gamma|\mathbf{x})$ and priors $p(z_0)$ and $p(\gamma)$, e.g., Horseshoe (Carvalho et al., 2009), spike-and-slab with Laplacian

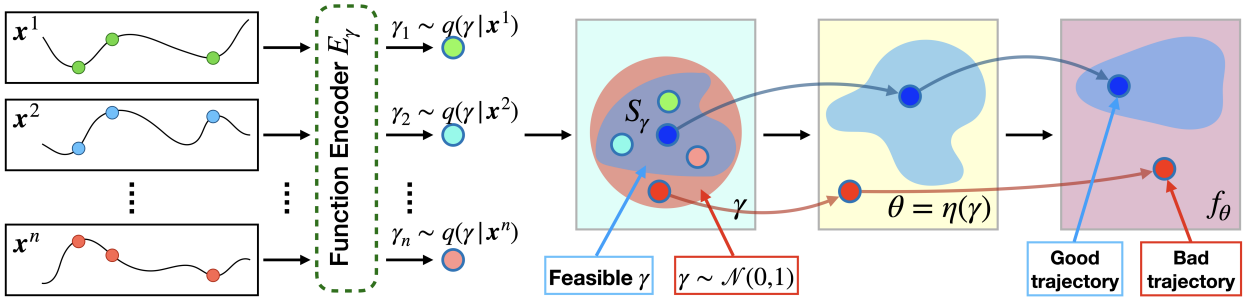


Figure 6.3: The direct sampling of γ from $N(0,1)$ results in a bad reconstruction of trajectories. However, given the samples from the learned approximate posterior distribution $q(\gamma|x^i)$, we can fit a better distribution on top of these sample, refereed as S_γ, to generate proper trajectories. The fitting is happened after we train our model.

spike (Deng et al., 2019) or Dirac spike (Bai et al., 2020); For our experiments, modeling $q_{\phi_E}(z_0|x)$, $q_{\phi_\gamma}(\gamma|x)$ and $p(z_0)$, $p(\gamma)$ as a Gaussian distribution offered good performance (reconstruction and sample diversity). Other distributions will nonetheless be useful for other real-world time-series datasets.

Training: Now, with all the pieces in hand, we summarize the training of the functional NODE (FNODE) as follows.

Given the observation of temporal process, x:

1. Use encoder $E_{z_0} : x = \{x_t\} \mapsto q_0$ and further sample initial point, z_0 from q_0.
2. Use encoder $E_\gamma : x = \{x_t\} \to q_\gamma$, and sample latent representation of transition function, γ from q_γ.
3. Use mapping η to map γ to define weights of the NN f_θ and hence characterize $f(z_t, t)$.
4. Use decoder D, fit the FNODE model, by minimizing the loss in (6.10)

The output from this phase is the latent representation of both components (a) initial value z_0 and (b) transition function $f(z_t, t)$, which allows us to sample temporal processes as in common VAE settings.

Note, if the trajectories are the same for all training samples, then η might collapse to a degenerate distribution due to the singularity. To avoid collapse, it is recommended to use KL weight annealing (from 0 to 1), (Huang et al., 2018).

Correcting posterior inaccuracies

While the variational setup is effective in deriving and optimizing for the maximum likelihood, prior results (Kingma et al., 2016) have observed that the Gaussian assumption

on the prior $p(z_0, \gamma)$ can often be too restrictive. This results in discrepancies between the prior $p(z_0, \gamma)$ and marginalized approximate posterior $q(z_0, \gamma)$.

To remedy this situation, using more flexible and sometimes hierarchical priors for VAE training (Kingma et al., 2016; Vahdat and Kautz, 2020) is useful. However, this leads to additional complexity for optimization since the computation of KL divergence $\mathrm{KL}\left(q\left(z_0, \gamma\right) | p\left(z_0, \gamma\right)\right)$ can be non-trivial for non-Gaussian priors. Instead, inspired by Ghosh et al. (2020), we empirically found that fitting a Gaussian Mixture Model (GMM) on the learned embeddings z_0, γ in an ex-post manner accounts for the multimodal nature of the data and leads to drastic improvement in sample quality. Since the GMM is learned independently after VAE training on the low-dimensional embeddings, the additional computational complexity is negligible. We refer to this GMM model as S_γ in the following discussions.

Implementation: For each sample x^j from the training data, we encode parameters $(\mu_\gamma^j, \Sigma_\gamma^j)$ and draw n_γ independent samples of $\gamma^j \sim N(\mu_\gamma^j, \Sigma_\gamma^j)$. Thus, given N training samples, we obtain $N \times n_\gamma$ samples of γ, using which a GMM is learned. The motivation for fitting GMM on samples of $N(\mu_\gamma^j, \Sigma_\gamma^j)$ instead of simply the means μ_γ^j is to also account for the variances Σ_γ^j – the target distribution to approximate is $q(\gamma)$ rather than $q(\mu_\gamma)$. Details on fitting GMM and selecting number of components is in the Appendix D.

Statistical inference

Given a learned S_γ we can perform different kinds of statistical inference.

Sample generation: Given an initial value z_0, obtained by encoding some $\mathbf{x}^i$ with E_{z_0}:

1. To sample new trajectories, we can directly sample from S_γ.
2. To sample trajectories described by an example $\mathbf{x}^j$, we can derive γ_j corresponding to $\mathbf{x}^j$ through the encoder E_γ, and use it to generate a new trajectory from the given z_0, but emulating the behavior of $\mathbf{x}^j$.
3. Relevant to example $\mathbf{x}^j$, we can also sample $\gamma\prime \in S_\gamma$, s.t. $|\gamma\prime - \gamma_j| \leqslant \delta$. Then, varying a value δ, we can generate new trajectories closer or farther from example $\mathbf{x}^j$.

Uncertainty estimation: At test time for the given observed temporal process $\mathbf{x}^j$, we are able to sample trajectories from the learned approximate posterior distribution $q_{\phi_\gamma}(\gamma|\mathbf{x}^j)$ and compute 95% credible intervals (similar to confidence intervals but from a Bayesian

perspective), posterior mean and variance. These summary statistics can express the uncertainty regarding temporal trajectories.

Out-of-Distribution (OOD) detection: OOD detection on trajectories is one important application of our method. In such a setting, outliers are defined by trajectories that has low-probability in the data-generating distribution, and thus unlikely to appear in the dataset.

For conventional temporal models, performing OOD detection is particularly challenging due to the need of integrating the conditional probabilities $p(x_t|x_{[0,t)})$ over the entire time span $t \in [0, T]$. However, since our method treats trajectories simply as embedded functional instances on which a statistical model is learned directly, there is no need to integrate over the trajectory in order to perform likelihood estimation or statistical testing, which greatly simplifies the formulation.

Advantages: Assuming that the initial condition is fixed for NODE, there is no variation in the trajectory – so, no distribution to setup a statistical test. But in our case, statistical inference is still possible via sampling. Compared to BNN, via sampling γ, we use a valid likelihood test based on GMM fitted to sampled γ's. Doing this with BNNs (assuming training succeeds) would either need comparisons of actual trajectories or a test on the posteriors of the entire network's weights.

Limitations: Several challenges or limitations in our formulation should be noted. First, since we parameterize f_θ through the computation of both γ and $\eta(\gamma)$, it results in a larger computation graph for backpropagation than a standard NN with parameters as weights. This limitation requires a more careful architectural choice for f_θ. Further, although using a Gaussian Mixture Model S_γ significantly improves the performance of the data decoder to reconstruct unseen trajectories, yet we encounter cases where the performance is not comparable to those from a VAE/GAN for a single frame.

6.3 Related work

While our formulation is different from an Neural ODE (where only the initial condition of the ODE varies), there are also distinct differences from Bayesian versions of NODE (Yildiz et al., 2019), where f is parameterized by a Bayesian Neural Network. In general, Bayesian Neural Networks (BNN) (Goan and Fookes, 2020) are not considered generative models, like VAE or GAN (Goodfellow et al., 2014). While VAE-based models, as in our

work, seek to learn a latent representation of the data (temporal process), the main goal in BNNs is to account for uncertainty in the data, learning a posterior distribution of a *network*, given all the observed data. In contrast to BNN, our work learns an encoding of the temporal process $\mathbf{x}^i$ in its own latent representation $q_{\phi_\gamma}(\gamma|\mathbf{x}^i)$ from which different trajectories are sampled. Conceptually, for each temporal process $\mathbf{x}^i$, the sampling from latent representations γ_i with distribution $q_{\phi_\gamma}(\gamma|\mathbf{x}^i)$ can be mapped to a BNN form of NODE. However, since the learned latent representation can be considered as a mixture of models, our approach can be thought as a mixture of BNN ODEs. In addition, the latent representation of the trajectory γ allows us to easily perform a likelihood-based test for Out-of-Distribution detection, which is not directly available for Bayesian Network.

One of the question can be whether the proposed functional neural ODE is a special case of the (Chen et al., 2018b; Rubanova et al., 2019)? That idea might appear if we view γ and z_0 as parameters of ODE system such that the ODE function f takes both γ and z_0 as $f(z_0, \gamma, t)$. However, this is misleading. We consider our Functional NODE as a more general version of NODE, since we sample the entire neural network f_θ, which is otherwise fixed in NODE. For instance, each NODE, in our formulation, is simply a sample from a space of FNODE. By considering γ to be a parameter of f, rather then parameter, which selects f, it does not guarantee that γ accurately describes the transition function f. Consequently, statistical inference on γ may not even be sensible if our goal is to perform statistical inference on trajectories, like OOD or clustering. On the other hand, in our formulation, trajectories (neural networks f) are explicit samples from the space of NNs, and this casting is statistically meaningful. Therefore, we can conduct testing and other inference on samples of interest, which in this case, are samples representing trajectories. Please, note that we did run NODE for our experiments to report MSE.

To be more rigorous, consider the formulation described in (Kidger et al., 2020). Specifically, note the Theorem in Section 3.3 which notes: "Any equation of the form $z_t = z_0 + \int_{t_0}^{t} h_\theta(z_s, X_s)ds$ may be represented exactly by a Neural CDE of the form as $z_t = z_0 + \int_{t_0}^{t} f_\theta(z_s)dX_s$. However the converse statement is not true."

This should fully resolve the doubt. In our case, the different γ's correspond to different X's i.e., they are functions controlling the evolution of the trajectory. More precisely, γ_1 is the parameterization for a control path (function) $X1$, γ_2 for $X2$ and so on. Note that these functions are evaluated along the domain of definition as usual, meaning that $X1(s) \equiv X1_s$ is the value of the function $X1$ at the point s. Therefore, as argued in the NCDE paper, our proposed formulation generalizes NODE in a similar way that NCDE does. Further, the distinction between NCDE and functional neural ODE is also clear since the latter deals with modeling a distribution over the control paths (functions) rather than simply one

path as in the NCDE case.

6.4 Experiments

We evaluate our model on various temporal datasets, of varying difficulty in the underlying trajectories, representing physical processes, vision data, and human activity. Namely: (**1**) a synthetic temporal dataset, (**2**) 3 different datasets from MuJoCo, (**3**) rotating/moving MNIST, and (**4**) a longer (100 time steps) real-life data, describing human body pose, NTU-RGBD.

Goals. We evaluate: (**a**) ability to learn latent representation of the samples of function space, i.e., representation of NN, (**b**) ability to generate new trajectories, including the trajectories based on examples (**c**) the ability to perform an Out-of-Distribution analysis for the trajectory and other kinds of statistical inference. We defer details of hardware, neural network architectures, including encoders/decoder and hyperparameters to the Appendix D.1.

Baselines. We compare the performance of our method (FODE) with several probabilistic temporal models, namely: (**a**) VAE-RNN (Chung et al., 2015), where the full trajectory is encoded in the latent representation; (**b**) Neural ODE (NODE) (Rubanova et al., 2019), where only initial point of ODE is considered to be stochastic; (**c**) Bayesian Neural ODE (BNN-NODE)(Yildiz et al., 2019), where the stochasticity is modeled through initial value z_0 and representing transition function f (which is a NN), using Bayesian version of Neural ODE; (**d**) Mixed-Effect Neural ODE (Nazarovs et al., 2021a), where both components, initial value z_0 and transition function f (which is a NN), through mixed-effects.

We start the experiment section showcasing a comprehensive analysis of a synthetic panel data set, which conceptually represents observations of multiple individuals over time. Our aim is to showcase the effectiveness of our framework in learning meaningful representations of the transition function f of temporal trajectories.

Later in the section, we provide an in-depth exploration of various components of our framework, with a specific focus on the independent elaboration of variational sampling of trajectories and statistical inference techniques. This analysis is performed on more complex computer vision data sets, demonstrating the versatility of our framework.

Complete analysis of a synthetic panel data

To demonstrate the ability of our framework to learn meaningful representation of the transition function f of temporal trajectories, we consider a synthetic panel data, which is

represented in the form:

$$\begin{cases} x_t^{(i)} = A^{(i)} \cdot \sin(\psi_t^{(i)}) + \varepsilon^{(i)} \\ \frac{d\psi_t^{(i)}}{dt} = 2\pi \\ \varepsilon^{(i)} \sim N(0, 10^{-3}) \end{cases} \tag{6.11}$$

In this representation, $x_t^{(i)}$ conceptually might represent the i patient's observation at time t, for example, it can describe amyloid accumulation in the region of the brain (Betthauser et al., 2022). The derivative of $x_t^{(i)}$ with respect to t represents the rate of change of $x_t^{(i)}$ with respect to time. The amplitude of changes $A^{(i)}$ is defined per group i, e.g. patient, and can vary among groups. While $x_t^{(i)}$ can be a multi-dimensional vector, withot loss of generality we consider $x_t^{(i)}$ to be a one dimensional. Namely, for each group i at time step t, $x_t^{(i)}$ is 1 dimensional vector.

For this set of experiments, we generate $10,000$ examples with 10 unique values of $A^{(i)} \sim \mathrm{Unif}(0, 10)$. In addition to make data more complicated in separation, and check that our framework is capable to capture the difference in transition functions $f^{(i)}$, rather than underlying $z_0^{(i)}$, we use the same initial condition $x_0^{(i)} = 0, \forall i$. We generate data for $t \in [0, 1.5]$, and randomly sample 10 time points for each individual data entry as observed data.

Learning a meaningful representation of transition function. Studying the form of equation (6.11), it is clear that samples of $x_t^{(i)}$ are separated corresponding to different values of $A^{(i)}$. To illustrate the ability of our model to learn meaningful representation of transition functions $f^{(i)}$, we plot T-SNE representation of learned by our framework $\gamma^{(i)}$, Figure 6.4. While there is some overlap between classes of derived $gammas$ corresponding to different values of amplitude A, there is a clear separation as one would expect by observing equation (6.11). A good separation between encoded values γ of temporal trajectories allows us to perform statistical inference, including sampling, which we are going to talk about next.

Variational sampling of temporal trajectories. Recall in Section 6.2, we described how we can correct posterior inaccuracies of learned approximate posterior distribution of $\gamma^{(i)}$, which allows us to perform variational sampling of high quality. Namely, for each sample x^j from the training data, we encode parameters $(\mu_\gamma^j, \Sigma_\gamma^j)$ and draw n_γ independent samples of $\gamma^j \sim N(\mu_\gamma^j, \Sigma_\gamma^j)$. Thus, given N training samples, we obtain $N \times n_\gamma$ samples of γ, using which a GMM is learned. Details on fitting GMM and selecting number of components is in the Appendix D.

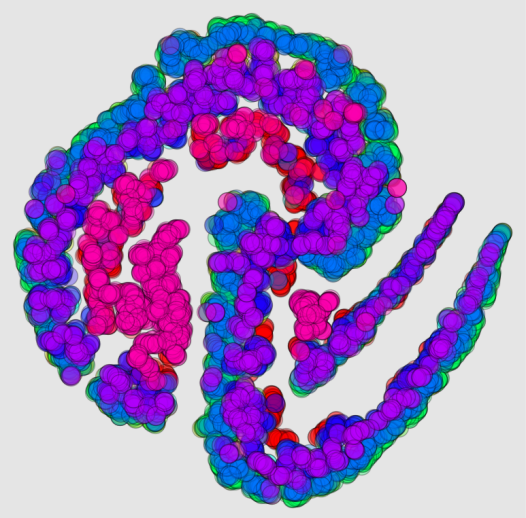


Figure 6.4: T-SNE representation of learned by our framework $\gamma^{(i)}$, which correspond to embedding of transition function $f^{(i)}$ of corresponding data trajectories. Each color correspond to a different value of amplitude $A^{(i)}$.

Given a learned GMM, we are able to sample temporal trajectories corresponding to data $x^{(i)}$, which is presented in Figure 6.5. In addition to observed values used for training the model (crosses on the image), we represent the 95% credible interval (red region) and a posterior mean (red dashed line), which are derived through sampling $\gamma^{(i)}$ from learned GMM and passing it through decoder, to obtain new values $\hat{x}^{(i)}$. As we can see that all observed values lie inside the credible interval, which indicate a simple check of our model's capability to generate variational samples of temporal trajectories. Next, we are going to discuss more of a statistical inference, and demonstrate the usability of our framework to detect trajectories outside the distribution.

Out-of-Distribution (OOD) detection. To evaluate the ability of our model to detect OOD trajectories, we introduce another set of trajectories, defined as:

$$\begin{cases} x_t^{B,(i)} = \sin(\psi_t^{(i)}) + \varepsilon^{(i)} \\ \frac{\psi_t^{(i)}}{dt} = 2\pi \cdot B^{(i)} \\ \varepsilon^{(i)} \sim N(0, 10^{-3}) \end{cases} \tag{6.12}$$

Similar to the previous setup, $x_t^{B,(i)}$ conceptually might represent the i patient's observation at time t, but belongs to the group with an underlying biological risk factor, which effects the development of the temporal process $x_t^{B,(i)}$. in this case the frequency coefficient $B^{(i)}$ is defined per group i, e.g. patient, and can vary among groups.

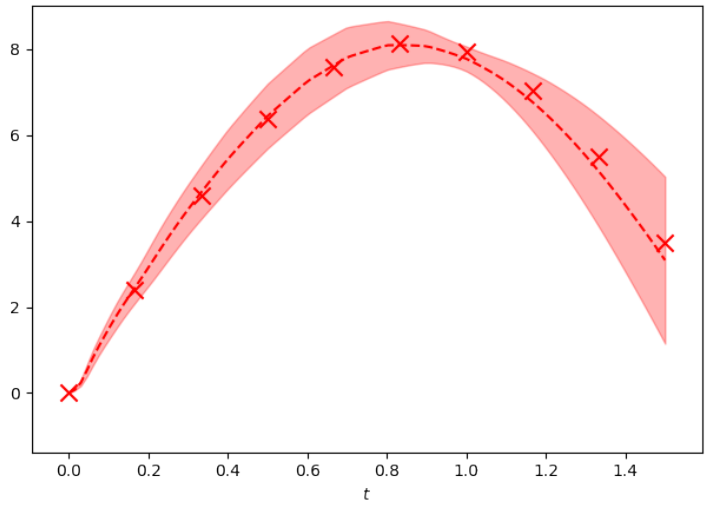


Figure 6.5: We demonstrate the variational sampling of temporal trajectories, given the 10 samples of observed data (x) used to train the model to derive representation of transition function. Red region is 95% credible interval and dashed line corresponds to the posterior mean.

We generate $10,000$ examples with 10 unique values of $B^{(i)} \sim \mathrm{Unif}(0, 10)$, with the same initial condition $x_0^{B,(i)} = 0, \forall i$. We generate data for $t \in [0, 1.5]$, and randomly sample 10 time points for each individual data entry as observed data.

Recall in Section 6.2, we describe the ability of our model to conduct OOD. Given a set of observed data $x^{(i)}$, we train our model to learn the approximate posterior distribution (and corresponding GMM) of $\gamma^{(i)}$ – the latent representation of transition functions corresponding to $x^{(i)}$. Then, given a trained model, we embed newly observed trajectories $x^{B,(i)}$ to the $\gamma^{B(i)}$ and conduct the likelihood outlier test to understand whether $\gamma^{B(i)}$ belongs to the learned GMM – distribution of $\gamma^{(i)}$. In Figure 6.6 we present two findings: on the left side we have a T-SNE representation of embedding of trajectories and on the right, the boxplot of negative log-likelihood for corresponding samples. As we can see, the negative log-likelihood of data for the set B is much higher than for a training set, which clearly indicates the outlier nature of set B.

In the following subsection we provide an in-depth exploration of variational sampling of trajectories and statistical inference techniques, which is performed on more complex computer vision data sets, demonstrating the versatility of our framework.

Variational Sampling of trajectories

We start this section by showing the ability of our model to sample trajectories for other datasets. Mainly we are interested in validating two aspects: (a) for the same initial

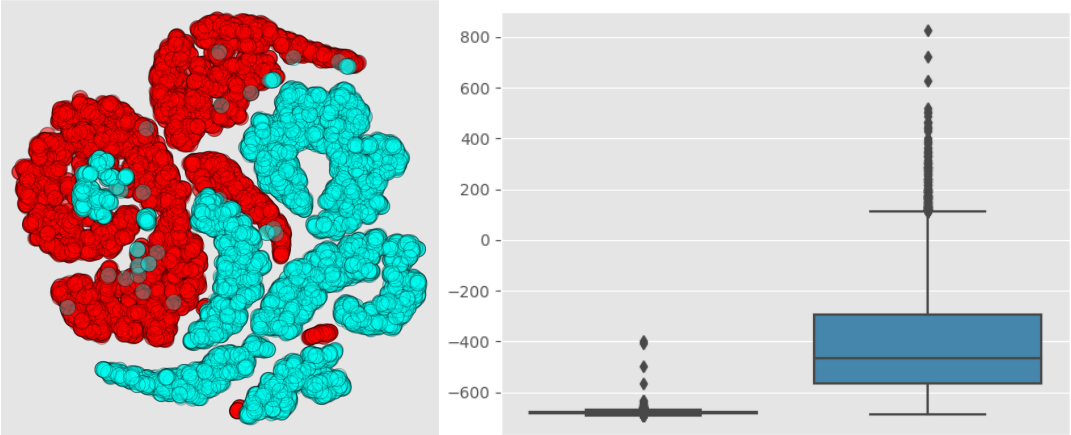


Figure 6.6: Outlier test: red is set participating in the training, blue is newly observed set B. Noticed the meaningfull separation in T-SNE representation of embedding between two distributions. in addition, we provide boxplots of negative log-likelihood for corresponding samples. As we can see, the negative log-likelihood of data for the set B is much higher, then for a training set, which clearly indicates the outlier nature of set B.

condition, can our model generate wide variety of trajectories? (b) can our model "transfer" trajectory of an example $\mathbf{x}^j$ to another temporal process $\mathbf{x}^i$?

Figure 6.7: We present visualization of generated trajectories of three different data sets from MuJoCo. From left to right: Walker, Hopper, and 3 poles cartpole. **Top row:** Trajectories are generated by fixing initial condition z_0 and sampling $\gamma \sim S_\gamma$, **Bottom row:** We transfer trajectory by sampling γ from the exemplar, and applying it to z_0. First row in each block represent original data from which we derive z_0, second row is an exemplar used to transfer trajectory, and last is sampled/transferred.

MuJoCo: Data from MuJoCo (Todorov et al., 2012) provides accurate simulation of articulated structures interacting with their environment to represents simple Newtonian physics. We vary the difficulty of the temporal process, by considering different objects and tasks: humanoid walker, humanoid hopper, and 3 poles cartpole which is a moving along the x-axis cart with rotation of 3 joints, attached to the base. *Sample trajectories:* For each experiment, we randomly generated initial condition and applied 10 different

Model	Data									
	Rotating MNIST					Moving MNIST				
	Interpolation	Extrapolation				Interpolation	Extrapolation			
		10%	20%	50%	100%		10%	20%	50%	100%
VAE-RNN	0.0206	0.0865	0.0850	0.0840	0.0831	0.0120	0.0565	0.0587	0.0578	0.0579
NODE	0.0169	0.0903	0.0904	0.0887	0.0863	0.0161	0.0534	0.0566	0.0553	0.0554
BNN-NODE	0.0193	0.0870	0.0870	0.0856	0.0834	0.0208	0.0518	0.0555	0.0537	0.0534
ME-ODE	0.0152	0.0877	0.0875	0.0870	0.0859	0.0193	0.0522	0.0554	0.0552	0.0553
FODE (Ours)	0.0108	0.0929	0.0932	0.0925	0.0911	0.0100	0.0554	0.0591	0.0590	0.0594

Table 6.1: Interpolation and Extrapolation MSE, where extrapolation is computed in respect of % of observed steps. Since all methods are probabilistic models, to evaluate MSE we compute the average among generated samples. While interpolation error of our method is lower compared to baselines, the extrapolation error is slightly higher. This happens because our method provides a wide variety of trajectories, which leads to a higher variance.

forces, then we observed 20 time steps. To make visualization easier, we plot every other step. In Figure 6.7(top), we present trajectories, generated from the same initial value z_0, but varying samples of the γ (latent representation of the transition function). For all three datasets, we are able to generate diverse and in-distribution trajectories. *Transfer trajectory to other temporal process:* We present a few examples of transferring trajectories in Figure 6.7(bottom).

In Figure 6.7, notice that comparing the temporal process in the first row (from which we derive z_0) and the example in the second row (from which we derive γ), there is a significant difference in the speed of walker. In the second row walker falls down slowly and needs almost the entire row. While the original fall (first row) occurs at a faster pace and from a higher height. As expected, after a successful transfer, the resulting trajectory of our walker (third row) does not fall down completely, because the pace is slower, derived from second row, while the initial position is at a similar height as in the first row. Similar observations can be made for hopper (2nd set): different speeds of rotation and unbending; and for cartpole: different directions of x-axis movement (3rd set).

Rotating/Moving MNIST: While MuJoCo dataset provides a wide variety of trajectories, it pertains to the same object. In this experiment, we want to check whether the sampling of trajectories based on an example, works not only within the same class of digits (similar to MuJoCo), but across different styles of Rotating and Moving MNIST. We model Rotating MNIST by 3 random components: digit (from 0 to 9, any style), random initial rota-

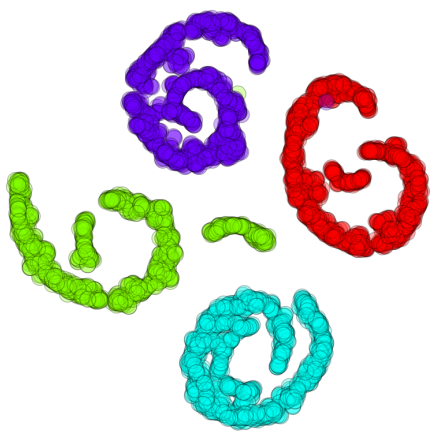


Figure 6.8: TSNE representation of γ. Colors correspond to the angle of rotation.

tion, and 10 different angles of rotation varying from $-45°$ to $45°$. For training, we used data observed for 10 time steps. For Moving MNIST we generate 4 random components: digit (from 0 to 9, any style), random initial position and velocity, and 5 different speeds of movement varying from 5 to 50. While the main goal of our method is a variational sampling of temporal trajectories, and not predicting the future steps, it is important to check that the proposed model (FODE) provides sensible results with respect to interpolation and extrapolation, compare to other baselines. For this experiment we generate 20 time-steps for both datasets. First 10 time-steps are used as observed to fit the model and learn distribution of z_0 and γ. The other steps, from 11 to 20, are used to evaluate the ability of our model to extrapolate. In Table 6.1, we provide Mean Squared Error (MSE) for both interpolation and extrapolation. Notice that our method (FODE) has a smaller interpolation error, compare to other baselines; however, the extrapolation error is higher. One reason is that we sample both initial value z_0 and representation of transfer function γ, we have a wider range of trajectories. We provide plots in the Appendix D.3.

Figure 6.9 shows the ability of our model to transfer the trajectory of an exemplar (second row) on the data from which we derived the initial value z_0 (first row). The resultant transferred trajectory is shown in the third row. As can be seen, our model can transfer trajectories across objects of different classes. This implicitly follows from the choice of our model of disentangling trajectory in 2 spaces: space of initial values z_0 and parameterization of the transfer function γ. Notice how smoothly the speed of rotation is transferred to the original data generating a new trajectory. In Figure 6.8, we show a TSNE visualization of samples from γ, embedding of transition functions that parameterize trajectories.

Computational and memory efficiency: For batch size 8 (rotation MNIST), our **(1)** memory need is $3\times$ more than VAE-RNN/NODE/ME-ODE, and $2\times$ more than BNN-NODE. However, for **(2)** wall-clock time, we need $2\times$ compared to VAE-RNN/NODE, but only $0.75\times$ of ME-ODE/BNN-NODE. Given that our model allows us to sample trajectories and perform statistical inference (which is not available for other methods) we believe that increase in memory and not drastic improvement in wall-clock tim is a sensible trade-off.

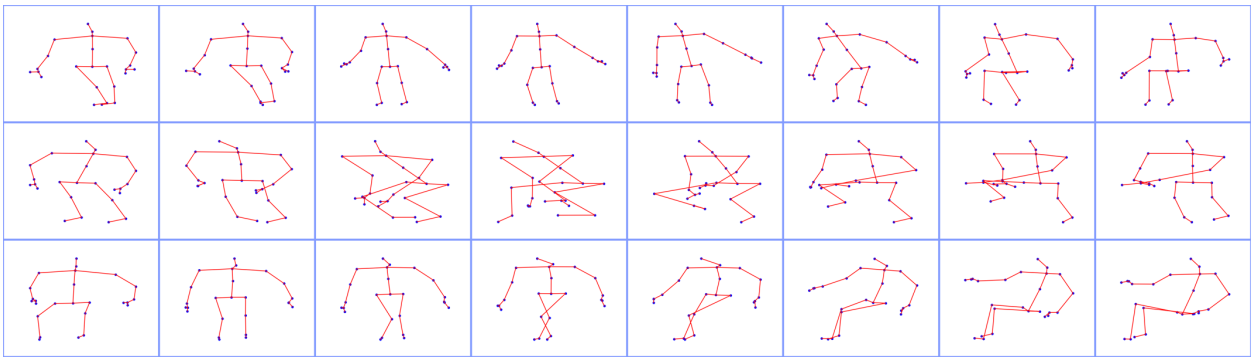
NTU: NTU is a real-life large-scale dataset for human actions recognition (Shahroudy et al., 2016). It consists of 126 states with varying number of time-steps. Each state is described by 26 joints in 3D coordinates, which are captured using Kinect v2.

We select data from 10 states, spanning 100+ time steps. To effectively deal with the

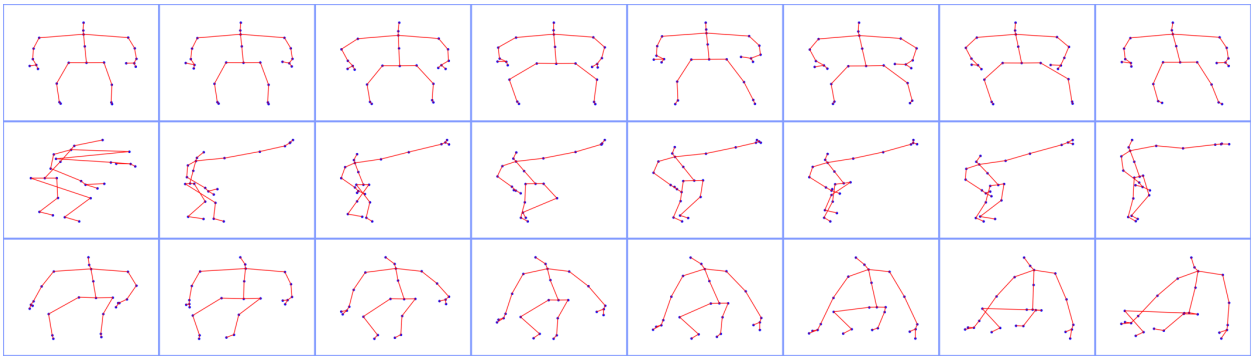
Figure 6.9: Per image: first row is a process used to derive z_0, second row is an exemplar, projection of which we would like to transfer, and third row is a result of transfer. Notice how the speed of second row transferred to the first row, given an initial condition.

longer time series data in NTU dataset, we use signature transforms as used in (Morrill et al., 2021). We compute signature transform on the segments of 10 steps with signature depth of 5 (please refer to (Morrill et al., 2021) for more details). We show that despite using the compact summary measures instead of the original observed data, we are still able to perform satisfactory generation of new trajectories in the observed space.

In Figure 6.10, we show a 2D projection of human activity, generated by our model. Figure 6.10b shows an interesting trajectory, where model learns from the exemplar to extend the hand, and incorporates this together with the sitting action.



(a) Initial trajectory and exemplar are actions of sitting, however, on the second row sitting is faster, which results in a fast sitting in the third row.



(b) Initial trajectory is a sitting action, while exemplar is hand waving. This results in a novel motion, that in the third row, the person sits but with extended hands.

Figure 6.10: Represent two different samples of transferring trajectory. First row is a process used to derive z_0, second row is an exemplar, projection of which we would like to transfer, and third row is a result of transfer. Notice how with the exemplar we can 'teach' another object to do similar things.

Statistical inference

In the previous section, we showed that our model can not only generate different trajectories, but can also transfer it from a given exemplar/sample.

As described earlier, learning latent representations of the transition function γ allows statistical inference. Here, we show that given a learned distribution of γ, we are able to perform an Out-of-Distribution analysis, to detect trajectories unfamiliar (or anomalous) for our learned model.

Moving MNIST: We create a dataset A by sampling different MNIST digits, random initial position in the frame and direction and traveling at a given speed. We train our model to learn the embedding of the transition function, γ. Then, we

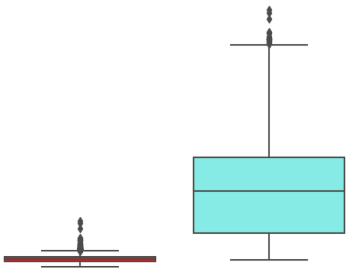


Figure 6.11: Outlier test for Moving MNIST: red is set A, blue is B. We pro-

generate additional data traveling at a different velocity (call it B), resulting in a different set of trajectories. We conduct OOD analysis, following the description in Sec. 6.2. In Figure 6.11 boxplots of negative log-likelilhood show the significant difference between groups. Clearly, implicit representation of trajectories using our method is beneficial for OOD.

NTU: We carry out a similar experiment on the NTU dataset. However, here, we increase the number of unknown states in the set B. The observed set A, used to train the model, contains information on only about 6 randomly selected classes of human activity. For the testing set B, we use the trajectories from 53 unobserved classes and perform the OOD testing as before. Because of a large number of states, instead of plotting 59 boxplots representing negative log-likelihoods, we provide a table with the ratio of points per class, which is defined as an outlier for set A, Table 6.2. We can see that almost 100% of samples from a set B, not in the training set A, are identified as outliers.

	Set A							Set B
Classes of Human Actions	6	8	9	23	27	36	43	all
Proportion of outliers	0.06	0.07	0.06	0.03	0.05	0.02	0.06	> 0.97

Table 6.2: Proportion of data points detected as outliers per class in: training set A (in-distribution) and testing set B (out-of-distribution).

6.5 Summary

We studied mechanisms enabling generation of new trajectories of temporal processes based on variational sampling. One key contribution is a procedure to sample from the function/embedding space of neural networks corresponding to a particular architecture choice. Experimental evaluations on several real and synthetic datasets show that the above proposed sampling procedure is effective in sampling new trajectories and in other statistical inference tasks such as OOD detection, topics that have remained underexplored in the generative modeling literature for temporal datasets. The code is available at https://github.com/vsingh-group/functional_node.

Chapter 7

Conclusions

This thesis explores how statistical concepts can inspire the creation of new deep learning probabilistic frameworks, thereby expanding their capabilities. We introduce novel frameworks that are based on a range of probabilistic models and demonstrate their effectiveness through various applications in computer vision and machine learning. Specifically,

1. In Chapter 3, we discuss the importance of bayesifying deep models for real-world applications and identify the limitations of current approaches based on Variational Inference (VI). Specifically, we note that deviating from the common assumption of a Gaussian distribution necessitates a numerical approximation of the KL divergence, which is often a Monte Carlo estimate. However, this approach can lead to scalability issues as the dimensions of the input data and model increase, and the structure of the computation graph can grow linearly with the number of MC samples required. To address this issue, we propose a framework to describe the size of the computation graph and identify large classes of probability distributions where computation graphs can be made independent of the number of Monte Carlo samples. Our empirical results show that this parameterization is feasible for running a large number (1000+) of Monte Carlo iterations for large networks in computer vision. This approach significantly decreases GPU memory requirements and improves runtime. Our proposed framework has real-world applications in medical imaging and serves as a core component of a novel ransomware detection system in Microsoft, which has led to a patent application and results described in (Nazarovs et al., 2022b).

2. Moving forward, in Chapter 4, we discuss the need for quantitative descriptions of uncertainties in predictions made by deep neural network (DNN) models, especially in computer vision applications. Although mechanisms for producing uncertainties, such as the bayesification of DNNs discussed in Chapter 3, are becoming available,

there is still a lack of literature on how to determine which uncertainty regions are significant. To address this issue, we propose a new framework that combines Random Field theory (RFT) with DNN tools, which enables us to perform hypothesis tests on uncertainty maps derived from DNN models. Specifically, we develop a probabilistic framework based on Neural ODE that allows us to learn a diffeomorphism between uncertainty maps and Gaussian Random Fields (GRFs). We refer to this model as the *Warping Neural ODE*. This approach enables us to perform hypothesis tests on the resulting GRFs and map the results back to the domain of uncertainty maps. As a result, we can identify the significant regions of uncertainty maps while controlling the family-wise error rate of the corresponding test and accounting for the dependency of pixels in the uncertainty mask, which is common in computer vision applications. Overall, our framework has the potential to significantly enhance the applicability and reliability of DNN models in many computer vision applications, including those that are mission-critical and require informed business/policy decisions.

3. In Chapter 5, we focused on analyzing panel data, which involves longitudinal measurements taken over multiple time points across several subjects. We proposed a novel probabilistic model called ME-NODE, which is designed to incorporate both fixed and random effects and is derived using smooth approximations of SDEs. In other words, ME-NODE incorporates random effects in predictions, similar to Stochastic Differential Equations (SDE), but the parameters of the underlying neural network can be trained efficiently without back-propagating through any SDE solvers. As an addition to the model, we also developed efficient training algorithms for ME-NODE and evaluated its performance on a range of tasks, including simulations and real longitudinal 3D imaging data from an Alzheimer's disease study. The results showed that ME-NODE can improve prediction accuracy, provide uncertainty estimates, and enable personalized prediction. To extend the application of our ideas from this chapter, we collaborated with researchers from John Hopkins University. The main goal of the follow-up project was to understand the relationship between APOE genotypes and the level of cognitive reserve to cognitive trajectories among cognitively normal individuals. In our analysis, we considered a longitudinal mixed effect model, similar to ME-NODE, led to a result describing the analysis of longitudinal data from several Alzheimer's disease (AD) studies (Pettigrew et al., 2023).

4. In Chapter 6, we proposed a new framework, Functional Neural ODE, a probabilistic model for learning and sampling the distribution of trajectories in temporal processes. Our framework explicitly parameterizes the transition function of differ-

ential equations as an element in a function space, allowing for efficient synthesis of novel trajectories and statistical inference tasks such as uncertainty estimation, likelihood evaluations, and out-of-distribution detection. The proposed approach has significant implications for downstream tasks, such as simulation and evaluation for reinforcement learning. Experimental evaluations on various real and synthetic datasets demonstrate the effectiveness of the method in generating new trajectories and performing statistical inference tasks that have remained underexplored in the generative modeling literature for temporal datasets. Overall, this chapter provides a valuable contribution to the field of generative modeling of temporal processes.

7.1 Future work

Next, we will delve into potential areas of research that we plan to pursue in the future.

Other interesting applications of our methods

Warping NODE. Chapter 4 introduces a continuous warping model that has the ability to learn an accurate one-to-one warping between two distributions of data. This model holds potential for future applications such as the analysis of security cameras or autonomous vehicles, which use fisheye-based cameras (Kumar et al., 2023). One of the challenges faced in these applications is the difficulty in performing segmentation due to the lack of data with fisheye properties (Blott et al., 2018). However, the Warping NODE approach can be used to learn a "fisheye warping" and thereby transfer a segmentation mask from standard images to fisheye images. This technique could potentially improve the segmentation performance of fisheye images in security and autonomous vehicle applications.

Functional NODE algorithm. In Chapter 6, we performed multiple experiments on synthetic data commonly used in Reinforcement Learning and scientifically meaningful data sets associated with biomedical datasets. However, our method can be applied to more diverse data representing temporal processes.

Multimodality

In various real-world applications, data is often available from multiple sources, or modalities. In the context of autonomous driving systems, a combination of data from various sensors such as cameras, LIDAR (Light Detection and Ranging) to measure depth, and proximity sensors to measure the speed of objects can be utilized to improve the accuracy

and reliability of the system (Cui et al., 2019; Royo and Ballesta-Garcia, 2019; Kocić et al., 2018). Similarly, in healthcare, data from multiple sources can be harnessed to enhance patient diagnosis and treatment. For instance, magnetic resonance imaging (MRI) is widely used for detecting and monitoring various diseases (Kiser et al., 2019). Blood pressure and temperature measurements are commonly utilized in clinical settings to diagnose and manage cardiovascular diseases and infections (Erden et al., 2016; Franckena et al., 2010). Additionally, electrocardiogram (ECG) and electroencephalogram (EEG) signals are frequently analyzed for the diagnosis of heart rhythm disorders and neurological conditions such as epilepsy, respectively (van Velzen et al., 2022; Altuğlu et al., 2020). Therefore, leveraging data from multiple modalities can offer valuable insights and improve the accuracy of various real-world applications such as autonomous driving and healthcare.

The integration of information from multiple modalities is commonly known as multimodal analysis (Blasch et al., 2019). Correspondingly, neural networks that can process multiple modalities are referred to as multimodal neural networks (Joshi et al., 2021; Ma et al., 2015; Mao et al., 2014). By leveraging complementary information from each modality, these models can potentially achieve better performance than unimodal models that only process one type of data (Baltrušaitis et al., 2018). This is because each modality provides a unique perspective on the data, and by combining them, the model can make use of more information and gain a more complete understanding of the underlying patterns.

Overall, the increasing availability of data from multiple sources in many real-world applications makes multimodal data analysis a highly relevant direction of research, and poses many interesting questions, where models such as those described in Chapters 3, Chapter 4, and Chapter 5 can serve as meaningful starting points.

Uncertainty/Robustness. Multimodal machine learning has emerged as an important research area in recent years, with a focus on leveraging multiple sources of information, or modalities, to improve prediction performance. One important question in this context is how different modalities, such as video, audio, and text, affect the uncertainty of predictions (Trick et al., 2019). This has implications in mission-critical applications, like analysis of security cameras. A key challenge is to determine whether certain modalities can be disregarded if they do not contribute to the prediction, or if removing modalities will lead to unstable model performance. One approach to studying the impact of different modalities on prediction uncertainty is to explore the use of mixed effects models covered in Chapter 5, to account for the distribution of trajectories and how it changes with the incorporation of different modalities.

Data synthesis to improve robustness of a model. To improve the robustness of neural networks, a promising method is to use synthetic data for training, which involves creating diverse and realistic examples. For instance, for image classification tasks, one can generate multiple versions of an image with different attributes like backgrounds, rotations, and zoom levels(Carlson et al., 2018). Multi-modal techniques have made it possible to generate images based on textual prompts, enabling more variety and novelty in the training set (Rombach et al., 2022). Similarly, in natural language processing tasks, one can generate different versions of text data (Iyyer et al., 2018) with varying levels of noise, spelling errors, or paraphrasing to improve the network's performance (Kant et al., 2021). Many of the ideas discussed in this thesis are centered around the concept of data generation and can be useful for synthesizing data. For instance, Chapter 4 presents method which ca be used for generating new images conditioning on textual descriptions, while Chapter 5 and Chapter 6 might be useful to synthesize new temporal processes, such as those found in biomedical data.

Video analysis

Video analysis represents a rich and active area of research in deep learning, and the integration of multimodality and probabilistic temporal models is key to making further advancements in this field (Luo et al., 2020). Probabilistic temporal models are particularly relevant to video analysis, as they can effectively model the dynamics of objects and events over time in video sequences. By integrating these models into deep learning architectures, we can achieve predicting future frames in videos, detecting anomalous events, and generating natural language descriptions of video content to conduct a search (Yu et al., 2021).

Appendix A

Graph Reparameterizations for Enabling 1000+ Monte Carlo Iterations in Bayesian Deep Neural Networks

In this document we provide more details about experiments, introduce our interactive application to analyze the quality of KL approximation, and give examples of computation graphs of KL terms for different distributions, in comparison between direct implementation and our parameterization technique. Proofs of the results in the main paper can also be found towards the end of the document.

A.1 Experiments Details

A working version of the code is attached in the directory "main_code". In our experiments, we follow the re-weighting scheme for mini-batches proposed by (Graves, 2011) as $\beta = \frac{1}{B}$, where B is number of mini-batches. For all experiments with VGG, we decrease the number of nodes by half in the last dense layers to fit the Bayesian model on a single GPU. We choose an exponential family with 2 parameters, which results in doubling the number of parameters compared to the original networks.

A.2 Making your own Bayesian network, using our API

Figure A.1 provides an example of how to implement your own Bayesian neural network with our API.

```
import bayes_layers as bl
class AlexNet(nn.Module):
    def __init__(self, num_classes, in_channels,
                 **bayes_args):
        super(AlexNet, self).__init__()
        self.conv1 = bl.Conv2d(in_channels, 64,
                               kernel_size=11, stride=4,
                               padding=5,
                               **bayes_args)
        self.classifier = bl.Linear(1*1*128,
                                    num_classes,
                                    **bayes_args)
        ...

    def forward(self, x):
        kl = 0
        for layer in self.layers:
            tmp = layer(x)
            if isinstance(tmp, tuple):
                x, kl_ = tmp
                kl += kl_
            else:
                x = tmp

        x = x.view(x.size(0), -1)
        logits, _kl = self.classifier.forward(x)
        kl += _kl
```

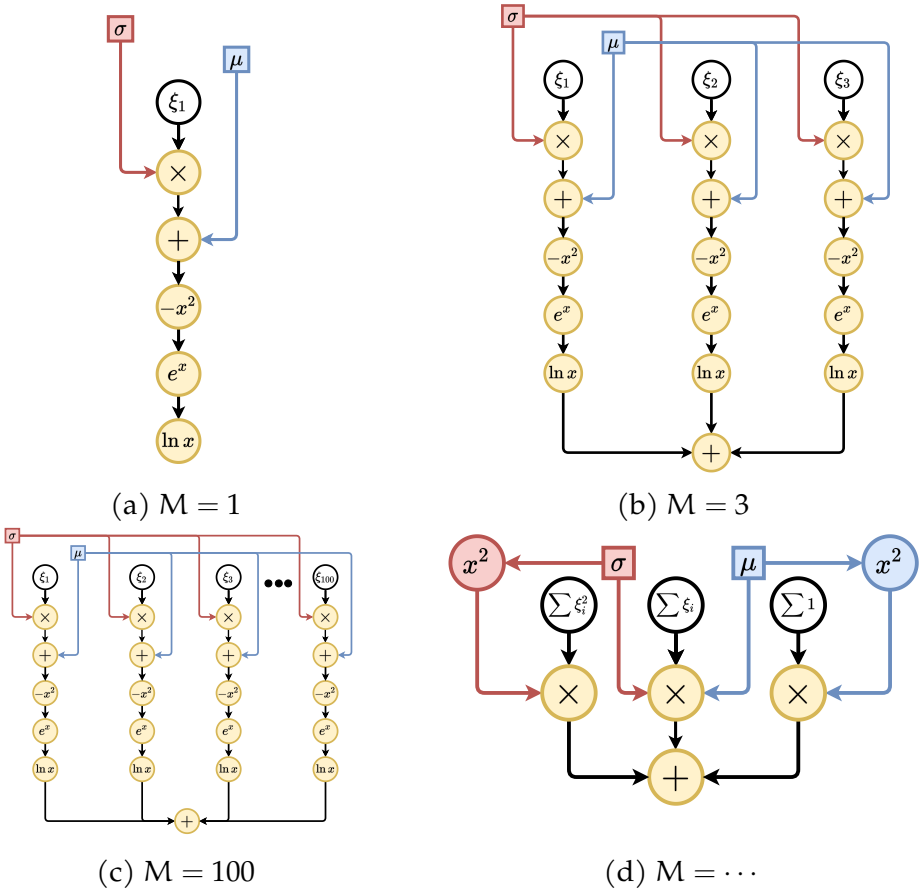
Figure A.1: An example of how to implement your own version of the bayesian neural network, using our API. We need to import bayesian layers module, which provides new functional for convolution1d, convolution2d, convolution3d and fully connected layers. In addition we need to redefine forward function, as shown.

A.3 Computation graphs

In this section we demonstrate computation graphs corresponding to the MC estimation of one of the expectation terms in KL (sometimes it is called KL cross-entropy): $\mathrm{E}_{Q_\theta} \log p(w)$, where Q_θ is the approximate posterior distribution with pdf q_θ, and $p(w)$ is the prior distribution on w. We compare the size of computation graphs for different numbers of MC iterations for a direct implementation and our reparameterization method.

Approximate posterior: Radial(μ, σ^2); Prior: Gaussian(0, 1)

For the following setup there is no closed form solution for KL term, and approximation with MC sampling is required. Samples from approximate posterior Q_θ can be generated as $\mu + \sigma\xi$, where $\xi = \frac{w}{|w|}r$, $w \sim \text{MVN}(0, I)$, $r \sim N(0, 1)$. Assumption about the prior gives us the following term to estimate $E_{Q_\theta} \log(\exp(-w^2))$. Figure A.2 shows computation graphs which correspond to different number of MC iterations. We can see that with the direct implementation, the size is proportional to the number of MC iterations, while our approach constructs a graph whose size is independent of the number of MC iterations.



(a) M = 1 (b) M = 3

(c) M = 100 (d) M = ···

Figure A.2: Computation graphs corresponding to MC approximation of KL term. (a)-(c) direct implementation, (d) - our method for any number of MC iterations

A.4 Interactive application to evaluate MC approximation of KL terms

Example

To demonstrate the relationship between MC estimation quality of the KL term and number of MC iterations, we provide an interactive Shiny application in this Appendix. If one assumes that approximate posterior and prior are Gaussian distributed, in this setting, we can calculate the "ground truth" KL. The main purpose here is to evaluate MC approximation by calculating the sample variance. The main parameters which will influence the quality of the MC approximation are: number of MC iterations, choice of variance of the approximate posterior distribution, and the size of the model (i.e. number of parameters in Neural Network). All these parameters can be set in our application. In addition, we provide an option to plot results in log-scale (for a better comparison). Additionally, graphs can be zoomed in, by highlighting a selected zone on the plot and double clicking (to zoom out, double click again).

Sample runs of our application (if the reader cannot run the tool) appear in Figure A.3. We fix variance equal to 10^{-4}, number of MC iterations up to 10^3, number of simulations per MC equals to 10^2 (to smooth the variance estimation). Then we compare the variance of 4 different models with number of parameters: $10^2, 10^4, 10^6$ and 10^8, and plot the results on the bottom figure with a log-scale. We see that despite the small variance, with growing size of the model it is necessary to increase number of MC iterations to decrease the variance of the MC estimator for KL.

Installation

Files are located in the directory "interactive_app". There are two ways to prepare our application for execution, both of them are handled by the integer parameter "method":

```
install_shiny_mc_repar.sh method
```

"method" can be one of 2 values: 1 or 2

1. If you have R installed, then the following packages are required to be installed and their installation is handled automatically:

```
c("shiny", "RColorBrewer",
  "dplyr", "ggplot2", "latex2exp")
```

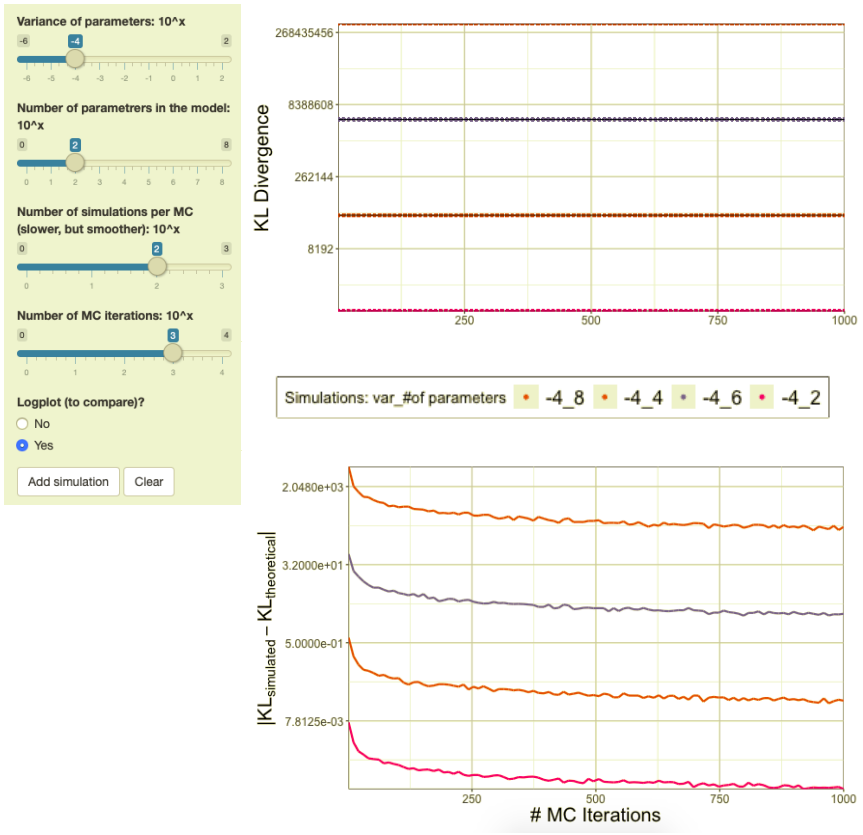


Figure A.3: Demonstration of interactive Shiny application to evaluate the performance of MC estimation.

2. If you would like to avoid installing R, but you have Docker installed, the script creates a Docker image with all necessary dependencies. It will take about 1.8GB of space and can be checked by running

```
docker images
```

Execution

After installation is successful, to run the application, execute the following script with a new “method" parameter:

```
run_shiny_app.sh method
```

“method" can be one of 2 values: 1 and 2

1. (R route) It will start app automatically in a browser.

2. (Docker route) In this case script starts shiny application and provides a local address, which you can access through the browser (this is address on your local machine, not external: http://localhost:3838)

Additional information about version of packages, which were used to run application

```
> sessionInfo()
R version 3.4.2 (2017-09-28)
Platform: x86_64-apple-darwin15.6.0
Running under: macOS  10.14.6

attached packages:
latex2exp_0.4.0
ggplot2_2.2.1
dplyr_0.7.4
RColorBrewer_1.1-2
shiny_1.0.5
```

A.5 Theory: Proofs and Clarifications

Proofs of Main Results

Theorem A.1. *If* $W(\theta, \xi) = \eta(\theta)T(\xi)$ $(S = 1)$*, then there exists a parametrization tuple* P *with* $d_P = 1$ *for following functions* $g(w)$*:* w^k, $\log(w)$, *and* $\frac{1}{w^k}$.

Proof. First, we are going to show that for $g(w)$: w^k, $\log(w)$, and $\frac{1}{w^k}$ there exists a parametrization tuple P of a specific form, and then we show that for these tuples $d_P = 1$.
Case (1). $\mathbf{g(w) = w^k}$

$$\begin{aligned}\sum_{i=1}^{M} g(w_i) = \sum_{i=1}^{M} w_i^k &= \sum_{i=1}^{M} (\eta(\theta)T(\xi_i))^k \\ &= \eta^k(\theta) \sum_{i=1}^{M} T^k(\xi_i) = \eta^k(\theta)T^k(\underset{\sim}{\xi}) \\ &= n(\theta)t(\xi)\end{aligned}$$

Case (2). $\mathbf{g(w) = \log(w)}$

$$\begin{aligned}\sum_{i=1}^{M} g(w_i) &= \sum_{i=1}^{M} \log(w_i) = \sum_{i=1}^{M} \log(\eta(\theta)T(\xi_i)) \\ &= M\log(\eta(\theta)) + \sum_{i=1}^{M} \log(T(\xi_i)) \\ &= M\log(\eta(\theta)) + \log(T(\underset{\sim}{\xi})) \\ &= n(\theta)t(\xi)\end{aligned}$$

Case (3). $\mathbf{g(w) = \frac{1}{w^k}}$

$$\begin{aligned}\sum_{i=1}^{M} g(w_i) &= \sum_{i=1}^{M} \frac{1}{w_i^k} = \sum_{i=1}^{M} \frac{1}{(\eta(\theta)T(\xi_i))^k} \\ &= \frac{1}{\eta^k(\theta)} \sum_{i=1}^{M} \frac{1}{T^k(\xi_i)} \\ &= n(\theta)t(\xi)\end{aligned}$$

We see that for all $g(w)$ from the list, we identify a parametrization tuple $P = (G(n,t), n(\theta), t(\xi))$, such that $G(n,t) = nt$ and $n(\theta)$, $t(\theta)$ depends on choice of $g(w)$. Clearly, for all these parametrization tuples P, $d_P = 1$. □

Theorem A.2. *If* $W(\theta,\xi) = \sum_{s=1}^{S} \eta_s(\theta)T_s(\xi)$, *and* $g(w) = w^k$, *then there exists a parametrization tuple* P *with* $d_P = \binom{k+S-1}{S-1}$.

Proof. Consider an MC expression $\frac{1}{M}\sum_{i=1}^{M} g(w_i)$. Given assumptions on $g(w) = w^k$ and $W = \sum_{s=1}^{S} \eta_s(\theta)T_s(\xi)$, we get:

$$\sum_{i=1}^{M} g(w_i) = \sum_{i=1}^{M} w_i^k = \sum_{i=1}^{M} \left(\sum_{s=1}^{S} \eta_s(\theta)T_s(\xi_i) \right)^k$$

We observe that $(\sum_{s=1}^{S} \eta_s(\theta)T_s(\xi_i))^k$ is a polynomial of order k with S indeterminates $T_s(\xi_i)$, and coefficients $\eta_s(\theta)$ independent of ξ_i. Let us denote the polynomial as $p_k^S(T(\xi_i);\eta(\theta))$. Since coefficients are independent of ξ_i for all i, then

$$\sum_{i=1}^{M} p_k^S(T(\xi_i);\eta(\theta)) = p_k^S(T(\underset{\sim}{\xi});\eta(\theta)),$$

where $\underset{\sim}{\xi} = (\xi_1, \ldots, \xi_M)$. Which results in the following parametrization tuple $P = (G, n(\theta), t(\xi))$, such that $G(n,t) = \sum_{i=1}^{d_P} n_i t_i$, and n_i, t_i are coefficients and indeterminates of the polyno-

mial $p_k^S(T(\underset{\sim}{\xi}); \eta(\theta))$.
The expansion of a polynomial of order k with S indeterminates has $\binom{k+S-1}{S-1}$ coefficients, exactly the number of interactions d_P. □

Corollary A.3. *If* $W(\theta, \xi) = \sum_{s=1}^{S} \eta_s(\theta) T_s(\xi)$, *and* $g(w) = p_K(w)$, *then there exists parametrization tuple* P, *such that for any* M *iterations*

$$d_P = \binom{K+S}{S} - 1. \tag{A.1}$$

Note. We consider a polynomial $p_K(w) = \sum_{k=0}^{K} a_k w^k$, such that coefficients a_k do not depend on optimized parameter θ. Since the first element a_0 of the polynomial $p_k(w)$ is not important for the analysis of d_P, we can ignore it and compute d_P as presented in Eq. 3.7. However, if there is a need to consider a_0, then one only needs to add $+1$ to the Eq. 3.7.

Lemma A.4. *Consider a polynomial of order* k *with* $S+1$ *indeterminates*

$$\left(\sum_{s=1}^{S} \eta_s(\theta) T_s(\xi) - a \right)^k,$$

where a *is a constant. If there* $\exists j : T_j(\xi) = c = const$, *then*

$$\left(\sum_{s=1}^{S} \eta_s(\theta) T_s(\xi) - a \right)^k = \left(\sum_{s=1}^{S} \eta_s^*(\theta) T_s(\xi) \right)^k$$

where $\eta^* = (\eta_1^*, \ldots, \eta_S^*) : \forall s \neq j,\ \eta_s^*(\theta) = \eta_s(\theta)$ *and* $\eta_j^*(\theta) = \eta_j(\theta) - a/c$.

Proof. The proof is direct by substituting $\eta_j^*(\theta)$. □

Theorem A.5. *Let* $W(\theta, \xi) = \sum_{s=1}^{S} \eta_s(\theta) T_s(X)$, $S \geqslant 2$. *If an approximation of* $g(w)$ *is made with* K *Taylor terms, then Corollary 3.8 applies.*

Proof. Consider Taylor approximation of $g(w)$, with proper a and K terms:

$$g(w) = \sum_{k=0}^{K} \frac{g^{(k)}(a)}{k!} (w-a)^k, \text{ where } K \text{ can be } \infty.$$

Then,

$$\begin{aligned}\sum_{i=1}^{M} g(w_i) &= \sum_{i=1}^{M} g(w_i) \\ &= \sum_{i=1}^{M}\sum_{k=0}^{K} c_k (w_i - a)^k \\ &= \sum_{k=0}^{K}\sum_{i=1}^{M} c_k (w_i - a)^k \\ &= \sum_{k=0}^{K}\sum_{i=1}^{M} c_k \left(\sum_{s=1}^{S} \eta_s(\theta) T_s(\xi_i) - a\right)^k \end{aligned} \tag{A.2}$$

From this point there are 2 ways to apply Corollary 3.8:

1. Polynomial of order K with $S+1$ indeterminates

Eq. (A.2) can be considered as a polynomial of order K with $S+1$ indeterminates. Then according to Corollary 3.8, for corresponding parametrization tuple P, $d_P = \frac{(K+1)\binom{K+S+1}{S}}{S+1} - 1$. However, since a in Eq. A.2 is constant, then $p_K(w-a)$ has K terms depending just on a and can be disregarded in the computation graph, since there is no interaction with parameters. This results in

$$d_P = \frac{(K+1)\binom{K+S+1}{S}}{S+1} - (K+1)$$

2. Polynomial of order K with S indeterminates

In some cases Eq. (A.2) can be considered as polynomial of order K with S new indeterminates, following Lemma A.4. Then according to Corollary 3.8, in addition to a node responsible for reparameterization of $\eta^*(\theta)$, we get

$$d_P = \frac{(K+1)\binom{K+S}{S-1}}{S} + 1$$

which reduces to the form in Eq. (3.7). □

Appendix B

Understanding Uncertainty Maps in Vision with Statistical Testing

B.1 Theoretical justifications

Lemma B.1. *Let* $Z(s)$ *be a UGRF, and* $\Lambda(s) = \mathrm{Var}(\dot{Z}(s))$. *The following properties are satisfied,* (*Taylor et al.*, *2006*)*:*

1. *If* Γ *is spatially constant non-singular matrix, then* $L_d(S, \Gamma'\Lambda\Gamma) = L_d(\Gamma^{-1}S)$.
2. *If* $\Gamma = \gamma I_{D\times D}$, *then* $L_d(\Gamma^{-1}S, \Lambda) = \frac{1}{\gamma^d} L_d(S, \Lambda)$.
3. *If* $S' = \Gamma S$, *then* $\Lambda(S') = \Gamma' Var(\dot{Z}(S))\Gamma$.

Theorem B.2. *The domain* S *of the GRRF* F *can be warped via a one-to-one smooth transformation* Γ *to a domain* S' *without fundamentally changing the problem, namely:*

$$\mathbb{P}\left(\max_{s'\in S'} F(s') \geqslant t\right) = \mathbb{P}\left(\max_{s\in S} F(s) \geqslant t\right)$$

.

Proof. Since Γ is a one-to-one mapping, let $S' = \Gamma^{-1}S$ and $S = \Gamma S'$. Since F is a Gaussian Related RF, we consider Z as an underlying UGRF, and denote Λ' and Λ as variance of spatial derivatives of Z on S' and S, respectively. We notate Euler density of F as ρ. To show that with warping we do not change the problem, we need to show the following:

$$\mathbb{P}\left(\max_{s'\in S'} F(s') \geqslant u\right) = \mathbb{P}\left(\max_{s\in S} F(s) \geqslant u\right).$$

We start with Application of GKF Theorem (from the main paper):

$$
\begin{aligned}
&\mathbb{P}\left(\max_{s'\in S'} F(s') \geqslant u\right) \\
\approx &\sum_d L_d\left(S', \Lambda'\right) \rho_d(u) \\
= &\sum_d L_d\left(\Gamma^{-1} S, \Lambda'\right) \rho_d(u) \\
= &\sum_d L_d\left(S, \Gamma' \Lambda' \Gamma\right) \rho_d(u) \\
= &\sum_d L_d\left(S, \Gamma' \mathrm{Var}\left(\dot{Z}(S')\right)' \Gamma\right) \rho_d(u) \\
= &\sum_d L_d\left(S, \mathrm{Var}\left(\dot{Z}(\Gamma S')\right)\right) \rho_d(u) \\
= &\sum_d L_d\left(S, \mathrm{Var}\left(\dot{Z}(S)\right)\right) \rho_d(u) \\
= &\sum_d L_d\left(S, \Lambda\right) \rho_d(u) \\
= &\sum_d \mathbb{P}\left(\max_{s\in S} F(s) \geqslant u\right) \qquad \square
\end{aligned}
$$

Remark B.3. *For u_Z, s.t. $\mathbb{P}\left(Z_{\max} \geqslant u_Z | H_0\right) = 0.05$, $\rho_d^Z(u_Z) > 0$ for $d = 0, \ldots, 3$. Note, it is actually true for $u << u_Z$, but we only need u_Z case. We focus on $d = 0, \ldots, 3$, because it is the most common case in practical application, and corresponds to the dimension of the Search region/Domain S. Usually in computer vision we only focus on 2d and 3d images and do not consider domains with dimension > 3.*

Proof. It is easy to see that from the form of $\rho_d^Z(u)$.

$$
\begin{aligned}
\rho_0^Z(u) &= \int_u^\infty \frac{1}{(2\pi)^{\frac{1}{2}}} e^{-u^2/2} du \\
\rho_1^Z(u) &= \frac{\lambda^{\frac{1}{2}}}{2\pi} e^{-u^2/2} \\
\rho_2^Z(u) &= \frac{\lambda}{(2\pi)^{\frac{3}{2}}} e^{-u^2/2} u \\
\rho_3^Z(u) &= \frac{\lambda^{\frac{3}{2}}}{(2\pi)^2} e^{-u^2/2} \left(u^2 - 1\right)
\end{aligned}
$$

The EC densities $\rho_d^Z(u)$ depend on a single parameter λ, the *roughness* of the random field. We can see that for $u > 1$ and $d = 0, \ldots, 3$, $\rho_d^Z(u) > 0$. And in a case of 2d GRF, common in vision, $u > 0$ is enough to get $\rho_d^Z(u) > 0$. $\square$

Theorem B.4. *Consider Gaussian Related RF $F(S)$ on domain S with Euler densities $\{\rho_d^F(u)\}$,*

and underlying UGRF $Z(S^Z)$ *on domain* S^Z *with Euler densities* $\{\rho_d^Z(u)\}$*. Assume, that both Euler densities* $\{\rho_d^F(u)\}$ *and* $\{\rho_d^Z(u)\}$ *are defined on the same domain* $u \in U$ *and* $\max\{\rho_d^F(u)/\rho_d^Z(u)\} \leqslant 1$*. Then, by finding a one-to-one transformation* Γ*, such that* $S = \Gamma S^Z$ *and* $S^Z = \Gamma^{-1}S$*, and selecting threshold* u^**, such that* $\mathbb{P}\left(\max_{s\in S^Z} Z(s) \geqslant u^*\right) = 0.05$*, guarantees that* $\mathbb{P}\left(\max_{s\in S} F(s) \geqslant u^*\right) \leqslant 0.05$.

Proof. Let $\rho_d^F(u) = c_d(u)\rho_d^Z(u)$. Given the Remark B.3, consider two cases with respect to sign of $\rho_d^F(u)$:

1. If $\rho_d^F(u) < 0$, then $c_d(u) < 0$, then $\rho_d^F(u) < \max\{c_d(u)\}\rho_d^Z(u)$.
2. If $\rho_d^F(u) \geqslant 0$, then $c_d(u) \geqslant 0$, then $\rho_d^F(u) \leqslant \max\{c_d(u)\}\rho_d^Z(u)$.

That is, for any sign of $\rho_d^F(u)$, we have $\forall d$, $\rho_d^F(u) \leqslant c\rho_d^Z(u)$, where $c = \max\{c_d(u)\}$. And there $\exists j$, such that $\rho_j^F(u) = c\rho_j^Z(u)$. Assuming that $\max\{\rho_d^F(u)/\rho_d^Z(u)\} \leqslant 1$, we get $c \leqslant 1$.

We start with an application of GKF Theorem (from the main paper):

$$\begin{aligned}\mathbb{P}\left(\max_{s\in S} F(s) \geqslant u\right) &\approx \sum_d L_d(S, \Lambda^F)\rho_d^F(u) \\ &\leqslant c\sum_d L_d(S, \Lambda^F)\rho_d^Z(u) \\ &\leqslant L_d(S, \Lambda^F)\rho_d^Z(u)\end{aligned}$$

Applying Theorem 4.2, if we find a warping Γ, s.t. $S = \Gamma^{-1}S^Z$ and $S^Z = \Gamma S$, then

$$\begin{aligned}&= L_d(S^Z, \Lambda^Z)\rho_d^Z(u) \\ &= \mathbb{P}\left(\max_{s\in S^Z} F(s) \geqslant u\right)\end{aligned}$$

□

B.2 Bootstrap

In the method section of the main paper ('Uncertainty between the models'), we briefly attributed to an application of bootstrap technique to approximate the distribution of test statistics $\{F_i\}_{i=1}^N$ under the null H_0 and conduct a hypothesis test, by checking if observed statistics falls into the rejection region, defined by corresponding u_F.

The bootstrap procedure for approximation of distribution of test statistics under H_0 is presented in Algorithm 3 and illustrated in Figure B.1.

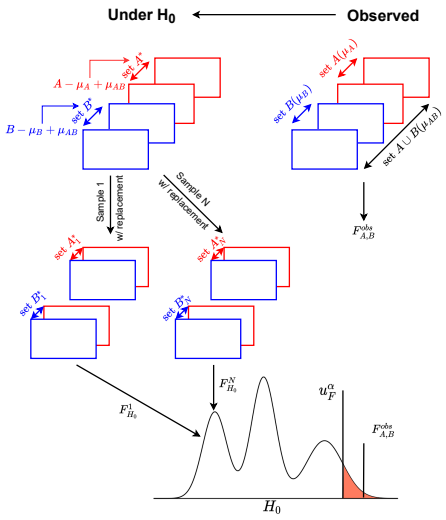


Figure B.1: Illustration of application bootstrap technique to generate distribution of statistics F under H_0.

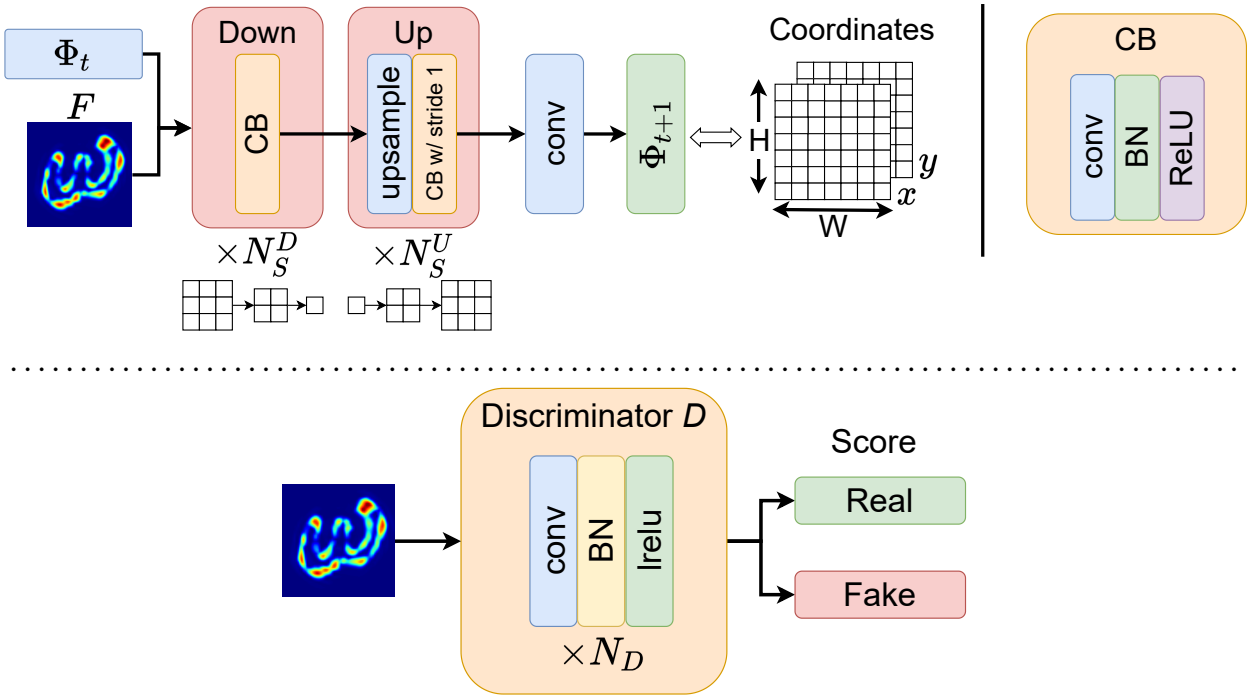


Figure B.2: *Top:* Warping Neural ODE part, which is used to generate the warping Φ_{t+1} at time point $t+1$. *Bottom:* Discriminator D, main goal of which is to differentiate between warped RF $\widehat{F}$ and observed GRF Z. N stands for number of convolution blocks.

Algorithm 3 Bootstrap algorithm for approximation of test statistics distribution under H_0

Input: Uncertainty masks for models A and B, $A_x(s)$ and $B_x(s)$ correspondingly, where s represents pixel
Output: Distribution of test statistics F under H_0

Require: Test statistics $F(x, y)$ between masks x and y

1: Compute observed statistics $F^{obs}_{A,B} := F(A_x, B_x)$
2: Generate 2 data sets of uncertainty masks under H_0:
$A^* = A - \mu_{group} + \mu_{overall}$ and
$B^* = A - \mu_{group} + \mu_{overall}$

Note: Algorithm 3 requires definition of test statistics $F(x, y)$. For example, it can be a common student statistics

$$F_{A,B}(s) = \frac{(\overline{A_x(s) - B_x(s)})}{\sigma(\overline{A_x(s) - B_x(s)})},$$

where σ is an estimate of standard deviation and $\bar{x}$ represents the sample mean.

The approximation of the distribution of test statistics $\{F_i\}_{i=1}^N$ under the null hypothesis H_0 allows us to find a threshold u_F, such that $\mathbb{P}(F_{max} \geqslant u_F | H_0) = \alpha$. Thus, if observed test statistics $F_{A,B}^{obs}$ is higher than this threshold u_F (or in our case significant mask $\widehat{\mathcal{M}}_F$ contains significant pixels), then we reject H_0 in favour of H_A.

B.3 Description of the Warping NODE Networks

Warping NODE

Learning the warping $\Phi(S)$ defined in equation B.1.

$$\frac{d\Phi_t}{dt} = V(\Phi_t), \tag{B.1}$$

As we mentioned, to model warping Φ_t we follow a common in vision technique (Jaderberg et al., 2015; Ashburner, 2007). The main idea is to model warping as target coordinates, from which we sample (or move to). That is, for input image with dimension $h \times w$, Φ_t is of dimension $h \times w \times 2$, where 2 appears because the dimension of image is 2. Since we model warping Φ_t as NODE, the modelled derivative $V(\Phi_t)$ have to be of the same size $h \times w \times 2$. Since it is important to generate different warping according to the input x, we model $V(\Phi_t)$ conditioning on image x, as $V(\Phi_T, x)$. We do this using U-Net, demonstrated in Figure B.2 (Top) with number of layers/channels depending on a problem.

Discriminator

Discriminator is used to minimize Wasserstrein distance, and presented in Figure B.2 (Bottom).

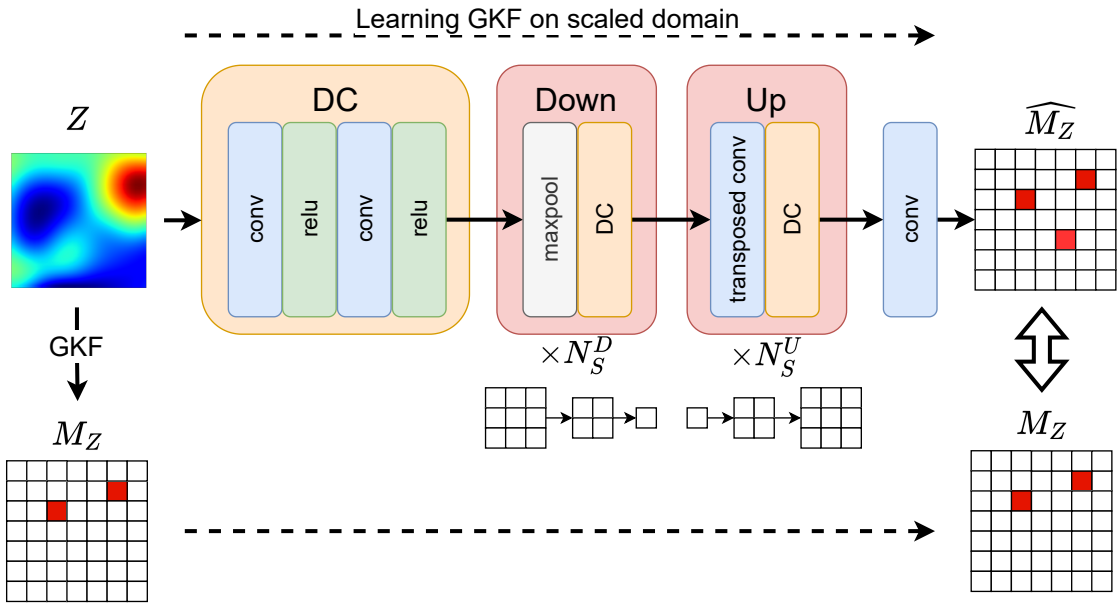


Figure B.3: Segmentation network, the main goal of which is to learn the GKF theorem on a scaled domain and derive the significance mask $\widehat{\mathcal{M}}_Z$ and later $\widehat{\mathcal{M}}_F$. The implementation is based on U-net structure. DC block corresponds to double convolution block, where first convolution changes number of channels and second preserves it.

Notes on assumptions of the Theorem 4.3 and GKF

One of the assumption of the Theorem 4.3 is that $\rho^F(s)$ and $\rho^Z(s)$ are defined for the same domain $s \in S$. In other words, since we are focusing on the warping, rather than Generative models, we are not allowed to generate pixels with completely new color values, but only warp (make wider, change order) observed pixels. For this reason, we scale both observed RF F and GRF Z to the same scale, e.g. $[0, 1]$. Since the scale is changed, we cannot directly use significant thresholds u_Z, obtained from GKF. We have to properly scale it, corresponding to the scaling used for F and Z. Instead, we use a "smart" scaling. To derive the threshold, corresponding to the scaled u_Z, we train (in a supervised manner) a segmentation network on (scaled GRF Z; significant mask M_Z, derived before scaling based on u_Z), represented on Figure B.3. Then resulted network is applied to the warped image $\widehat{F}$ to derive $\mathcal{M}_F$.

Notes on generating GRF

The main idea of our method is based on mapping an uncertainty map to the isotropic GRF, for which we can derive significant region based on GKF theorem. To generate an isotropic GRF, it is necessary to provide 1) dimensions of a RF, which corresponds to the

dimension of the input uncertainty map, 2) covariance function: $C(t) = \exp\left(-\beta\|t\|^2\right)$ with the proper scale β. Note that scale β directly affects the threshold u_Z for which elements (in our case pixels) of GRF is considered to be significant, through the computation of LKC (or EEC). We consider scale β as a hyper parameter, which impacts how easy it is to find a warping from uncertainty map to GRF, and mainly depends on an uncertainty map size. However, given the scale it is important to recompute a threshold u_Z to generate a significant mask $\mathcal{M}_Z$ for GRF.

B.4 Experiments

Synthetic

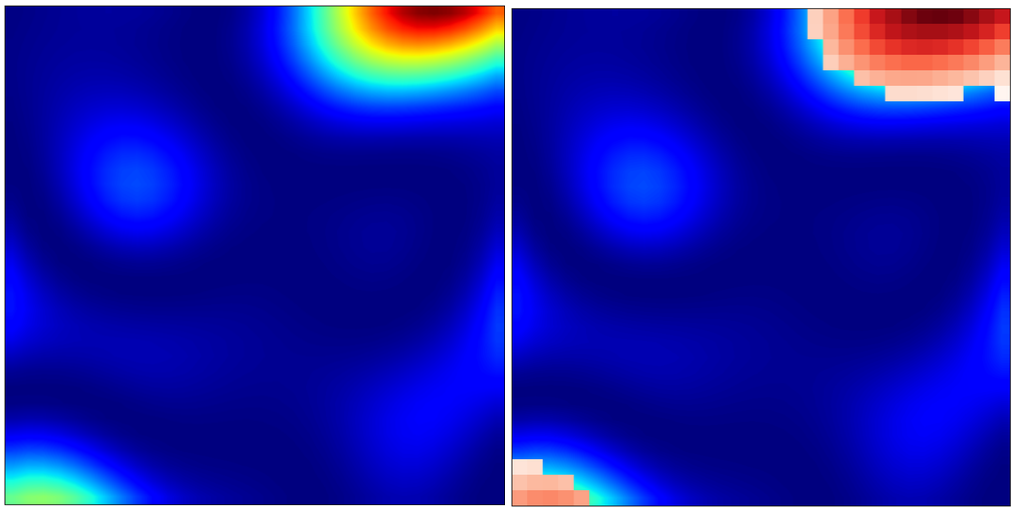


Figure B.4: (Left) GRRF, χ^2 with 5 degrees of freedom and (Right) overlapped significance mask $\widehat{\mathcal{M}}_F$ derived by our model.

We show our model learns correct significant region, preserving power and limiting type-1 error rate by a significance level. We generate 1000 samples of Gaussian RF Z and 3 sets of the 1000 samples of Chi-squared RF F, with the following degrees of freedom: 1, 3, and 5. All sets were generated to satisfy the conditional that $\mathbb{P}(Z_{max} \geqslant u_Z|H_0) = 0.05$ and $\mathbb{P}(F_{max} \geqslant u_F|H_0) = 0.05$. Since there is a closed form solution to compute $\mathbb{E}\{\phi(A_u)\}$ for F, we can obtain a ground-truth significance mask $\mathcal{M}_F$. Therefore, we can evaluate the performance of our model by comparing generated $\mathcal{M}_F$ with a real (ground truth) $\mathcal{M}_F$. Table B.1 provides achieved power and type-1 error on the learned test. We see that for all cases power of test is higher than 80%, while preserving type-1 error less than 0.05.

	χ^2_1	χ^2_3	χ^2_5		χ^2_1	χ^2_3	χ^2_5
Power	0.83	0.83	0.87	Type-1	0.02	0.02	0.02

Table B.1: The achieved power and type-1 error of our generalized test, performed on three different GRRF, i.e. χ^2 with varying degrees of freedom.

Longitudinal Brain Image data, ADNI.

Alzheimer's Disease Neuroimaging Initiative (ADNI) (adni.loni.usc.edu). It contains gray matter probability masks from the T-1 weighted MR images of size $105 \times 127 \times 105$ per subject at 3 time steps. The subjects are divided into two groups: diagnosed with Alzheimer's disease (AD: 377 subjects) and healthy controls (CON: 152 subjects). Results mentioned in the main paper are displayed in Figure B.5.

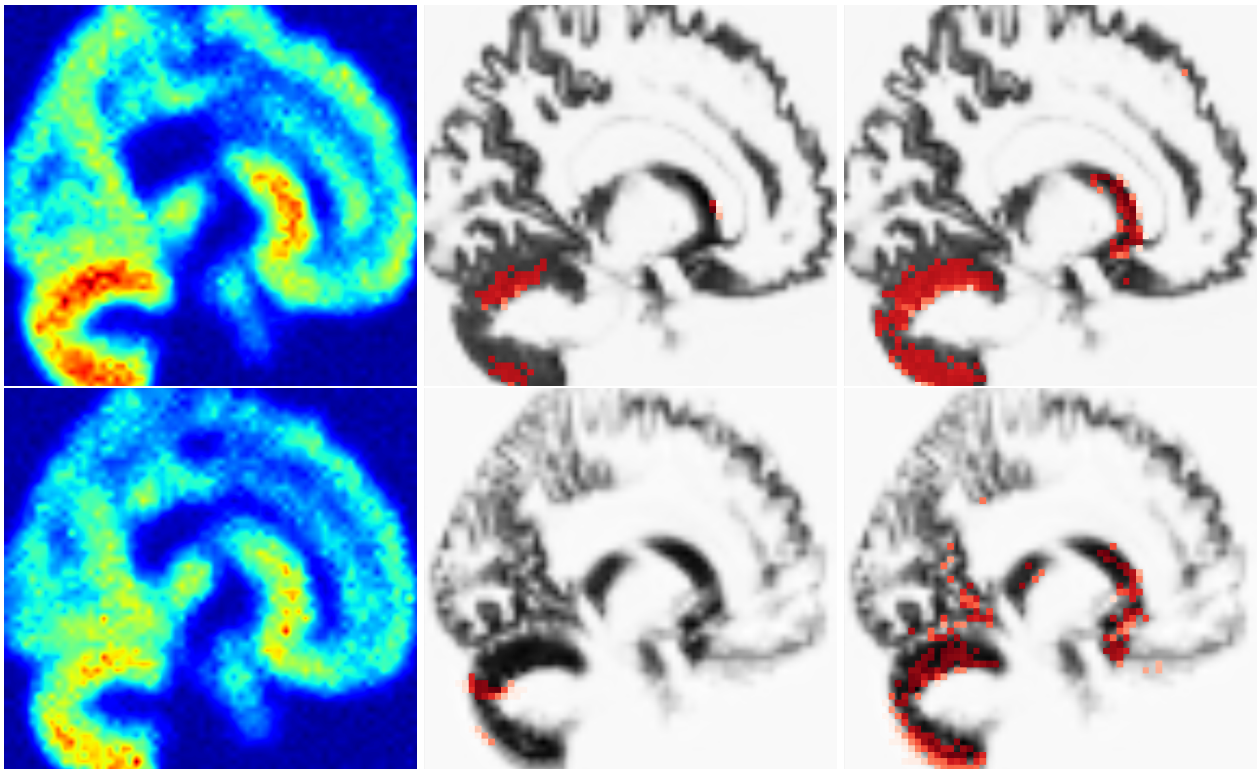


Figure B.5: Top: AD group, Bottom: Con group. Each triplet consists of (uncertainty mask, significant region based on our method, significant region based on 5% quantile).

Appendix C

Mixed Effects Neural ODE: A Variational Approximation for Analyzing the Dynamics of Panel Data

C.1 Proofs

Derivation for final loss

$$
\begin{aligned}
\log p(x) \geqslant & \mathbb{E}_{q(z,w)} \log p\left(x|z,w\right) - KL(q(z,w)||p(z,w)) \\
\approx & \frac{1}{M}\sum_{m=1}^{M} \log p\left(x|z,w\right) \cdot \mathbb{1}_{z_{-0}^{obs}}\{z_{-0}|z_0^m, w^m\} + \\
& \frac{1}{M}\sum_{m=1}^{M} \log \frac{q(z,w)}{p(z,w)} \cdot \mathbb{1}_{z_{-0}^{obs}}\{z_{-0}|z_0^m, w^m\} \\
= & \frac{1}{M}\sum_{m=1}^{M} \left(\log p\left(x|z,w\right) + \log \frac{q(z,w)}{p(z,w)}\right) \cdot \\
& \qquad \mathbb{1}_{z_{-0}^{obs}}\{z_{-0}|z_0^m, w^m\} \\
= & \frac{1}{|S|}\sum_{s\in S} \left(\log p\left(x|z^s,w^s\right) + \log \frac{q(z_0^s)q(w^s)}{p(z^s,w^s)}\right),
\end{aligned}
$$

where S is a set, such that $\{\forall s \in S : \mathbb{1}_{z_{-0}^{obs}}\{z_{-0}|z_0^s, w^s\} = 1\}$ and $|S|$ is its size.

Encoder structure used in ROTATING MNIST

```
encoder = nn.Sequential(
        nn.Conv2d(input_dim, 12, ks,
                  stride=1, padding=1),
        nn.ReLU(),
        nn.Conv2d(12, 24, ks,
                  stride=2, padding=1),
        nn.ReLU(),
        nn.Conv2d(24, output_dim, ks,
                  stride=2, padding=1),
        nn.Flatten(2),
        nn.Linear(49, 1),
        nn.Flatten(1)
        )
```

Decoder structure used in ROTATING MNIST

```
extend_to_2d = nn.Linear(input_dim,
                         49 * input_dim)
decoder = nn.Sequential(
        nn.ConvTranspose2d(input_dim,
                           24,
                           ks,
                           stride=2,
                           padding=1,
                           output_padding=1),
        nn.ConvTranspose2d(24,
                           12,
                           ks,
                           stride=2,
                           padding=1,
                           output_padding=1),
        nn.ConvTranspose2d(12, output_dim, ks,
                           stride=1, padding=1),
        nn.Sigmoid(),
        )
```

Figure C.1: Description of Encoder and Decoder used in experiment with 2d data structure: Rotating MNIST.

C.2 Experiments

Hardware specifications and architecture of networks

For Rotating MNIST (2d data) encoder/decoder is described in Figure C.1 and for ADNI (3d data) encoder is described in Figure C.2 and in Figure C.3 and decoder in Figure C.4.

Encoder structure used in ADNI

```
nn.Conv3d(64,
          128,
          kernel_size=3,
          stride=1,
          padding=1,
          bias=False),
nn.ReLU(),
nn.MaxPool3d(kernel_size=2, return_indices=True),
nn.Conv3d(128,
          256,
          kernel_size=3,
          stride=1,
          padding=1,
          bias=False),
nn.ReLU(),
nn.MaxPool3d(kernel_size=2, return_indices=True),
nn.Conv3d(256,
          output_dim,
          kernel_size=3,
          stride=1,
          padding=1,
          bias=False),
nn.ReLU(),
nn.Flatten(2),
nn.Linear(8, 1),
nn.Flatten(1)
)
```

Figure C.2: Encoder for ADNI, part 1

Encoder structure used in ADNI

```
        nn.Conv3d(64,
                  128,
                  kernel_size=3,
                  stride=1,
                  padding=1,
                  bias=False),
        nn.ReLU(),
        nn.MaxPool3d(kernel_size=2, return_indices=True),
        nn.Conv3d(128,
                  256,
                  kernel_size=3,
                  stride=1,
                  padding=1,
                  bias=False),
        nn.ReLU(),
        nn.MaxPool3d(kernel_size=2, return_indices=True),
        nn.Conv3d(256,
                  output_dim,
                  kernel_size=3,
                  stride=1,
                  padding=1,
                  bias=False),
        nn.ReLU(),
        nn.Flatten(2),
        nn.Linear(8, 1),
        nn.Flatten(1)
        )
```

Figure C.3: Encoder for ADNI, part 2.

Decoder structure used in ADNI

```
extend_to_3d = nn.Linear(input_dim, 2 * 2 * 2 * input_dim)
decoder = nn.Sequential(
        nn.ConvTranspose3d(in_channels=input_dim,
                           out_channels=256,
                           kernel_size=3,
                           padding=1),
        nn.ReLU(),
        nn.MaxUnpool3d(kernel_size=2),
        nn.ConvTranspose3d(in_channels=256,
                           out_channels=128,
                           kernel_size=3,
                           padding=1),
        nn.ReLU(),
        nn.MaxUnpool3d(kernel_size=2),
        nn.ConvTranspose3d(in_channels=128,
                           out_channels=64,
                           kernel_size=3,
                           padding=1),
        nn.ReLU(),
        nn.MaxUnpool3d(kernel_size=2),
        nn.ConvTranspose3d(in_channels=64,
                           out_channels=32,
                           kernel_size=3,
                           padding=2),
        nn.ReLU(),
        nn.MaxUnpool3d(kernel_size=2),
        nn.ConvTranspose3d(in_channels=32,
                           out_channels=16,
                           kernel_size=3,
                           padding=1),
        nn.ReLU(),
        nn.MaxUnpool3d(kernel_size=2),
        nn.ConvTranspose3d(in_channels=16,
                           out_channels=8,
                           kernel_size=3,
                           padding=2),
        nn.ReLU(),
        nn.MaxUnpool3d(kernel_size=2),
        nn.ConvTranspose3d(in_channels=8,
                           out_channels=output_dim,
                           kernel_size=3,
                           padding=1),
        nn.ReLU()
       )
```

Figure C.4: Decoder for ADNI

Appendix D

Variational Sampling of Temporal Trajectories

D.1 Hardware specifications and architecture of networks

All experiments were executed on NVIDIA - 2080ti, and detailed code will be provided in the github repository later (currently it is attached).

Neural Differential Equations: In any of the implementation we used Runge-Kutta 4 method with ode step size 0.1. To model the equation $\dot{z}_t = f(z_t, t)$, we consider latent representation z_t to be in 1 dimension with p channels. Recall that we learn initial value z_0 from the data directly, and thus we are required to apply encoder.

Encoder E_{z_0}: For the z_0 encoder we use a known and established ode-rnn encoder (Rubanova et al., 2019). For the datasets, like Moving/Rotation MNIST and NTU in addition to ode-rnn encoder we apply a pre-encoder. The main purpose of pre-encoder is to map input in 1 dimensional data with k channels. Which is later used by E_{z_0} to map into p channels.

For **Moving/Rotation MNIST** we use ResNet-18 pre-encoder, with following number of channels (8, 16, 32, 64) for an Encoder.

For **NTU**, the used data consist of 100 time steps with 26 different sensors, and each sesor is with 3d coordinates. As we mentioned in the main paper, we apply the signature transforms. Given the sample with 100 time steps, we split it in chunks of 10, and apply signature transform of depth 5 for each chunk. This results in 10 chunks total with 363 channels (formula: $3 + 3^2 + \ldots + 3^5$). As a result of signature transformation we obtain 10 time steps for 86 sensors and 363 channels each. We then apply several FC layers to change

number of channels, then we collapse 26 sensors with resulted channels and convert it to k channels, as mentioned above.

```
nn.Sequential(
    nn.Linear(363, 128), nn.ReLU(),
    nn.Linear(128, 256), nn.ReLU(),
    nn.Flatten(2),
    nn.Linear(last_dim * 256, 128),
    nn.ReLU(),
    nn.Linear(128, output_dim))
```

Encoder E_γ: This is an important part of our modelling, which given temporal observations with T time steps, $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)$, encodes the information about trajectory into latent representation γ. For this we tried a lot of different encoders, including: (**a**) fully connected, (**b**) neural-ode, (**c**) dilated CNN-1d, (**d**) LSTM, (**e**) RNN, and (**f**) Time Series Transformer. To our surprise the fully connected network performed the best in respect of learning trajectory and training time. The second place was Time Series Transformer.

Projection $\eta : \gamma \implies \theta$: For the projection network η we use a fully connected network, with $\mathrm{Tanh}()$ activation function between layers, and after the last layer. Because η eventually returns us weights of a neural network θ, which cannot be extremely big, it was necessary for us not only apply $\mathrm{Tanh}()$ as a last layer, but also introduce another single-value parameter λ_θ, which is used to scale $\theta = \lambda_\theta \tanh(F(\gamma))$. We noticed a significant boost in performance once we introduced a tanh on the last layer and a learned scalar λ_θ, compare to weights being completely unbounded.

NN parameterization of transition function f: The previously described weights θ, are the weights of NN, which parameterize f in $\dot{z}_t = f(z_t, t)$. For all experiments we used f as a fully connected network with 3 layers with 100 hidden units each layer and tanh activation function between layers. While the size of this network might look small, empirically it has enough power to model the changes in latent space over time.

Computation aspects of optimization: (This paragraph represents an implementation in the PyTorch, however, it is similar for other popular choices like Tensorflow.) Since we model weight of the NN f, rather than consider them as parameters, we cannot use a standard implementation of layers in PyTorch, like 'nn.Linear()'. Instead, we create a class of functional networks, where each layer is presented by an operation, like nn.functional.linear.

The class is initialized by copying the structure of the provided network, but replacing layers with nn.functional correspondence. The forward function of the class accepts input x and weights θ. Then θ is sliced to fit the size of the current layer and layers is applied to x. As we mentioned in the Limitations of the main paper, this results in a larger than usual computation graph, and requires a more careful architectural choice for f.

Decoder D_x: For the Moving/Rotation MNIST, similar to Encoder E_{z_0}, we use ResNet-18 decoder, but with $(32, 16, 8, 8)$ channels. For other data sets we mainly use fully connected neural networks.

D.2 Selecting GMM

One of the components of our work is to utilize Gaussian Mixture Model (GMM) on top of samples γ from the learned posterior distributions $q_{\phi_\gamma}(\gamma|\mathbf{x}^i)$. The architecture of GMM is based on 2 key factors: number of components and covariance type. To select the best GMM model, which is later used for sampling from posterior distribution, we range number of components from 10 to 200, with a step 10, and consider 4 different types of covariance matrix provided in python machine learning package “sklearn": “spherical", “tied", “diag", “full". Among all these models we select the one with lowest BIC score as the best.

D.3 Experiments

Rotating MNIST

In this section we provide additional samples of trajectories. To show the quality of our reconstruction, regardless of MSE values from the table in main paper, in Figure D.1 we present the ablation studies of several samples for different baselines.

In addition to comparison to quality of reconstruction, we show the variation of samples derived by sampling different representations of the transition function γ, but preserving the initial value of the latent representation z_0.

MuJoCo

Below we provide several samples from our model for different sets of MuJoCo, namely Hopper, Walker and 3-poles cart. Notice how trajectories are different, despite the same initial values.

Figure D.1: Reconstruction of the same samples, from top to bottom by block: VAE-RNN, NODE, BNN-NODE, FODE (our). Clearly, our model provides the most clear reconstruction.

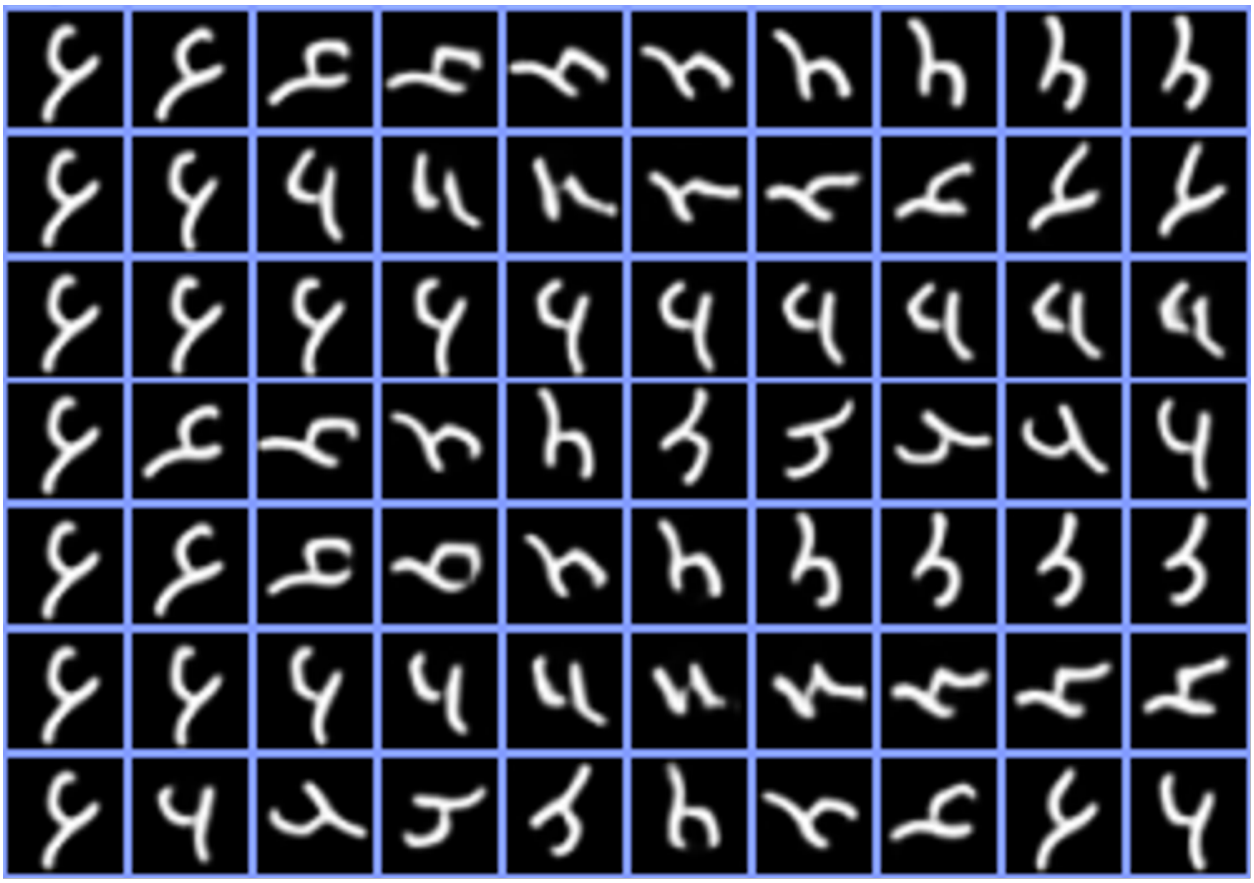
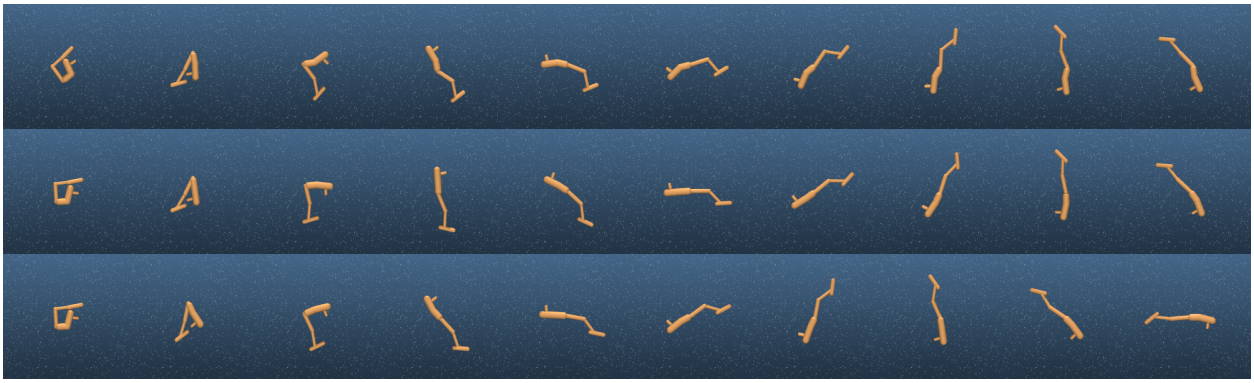
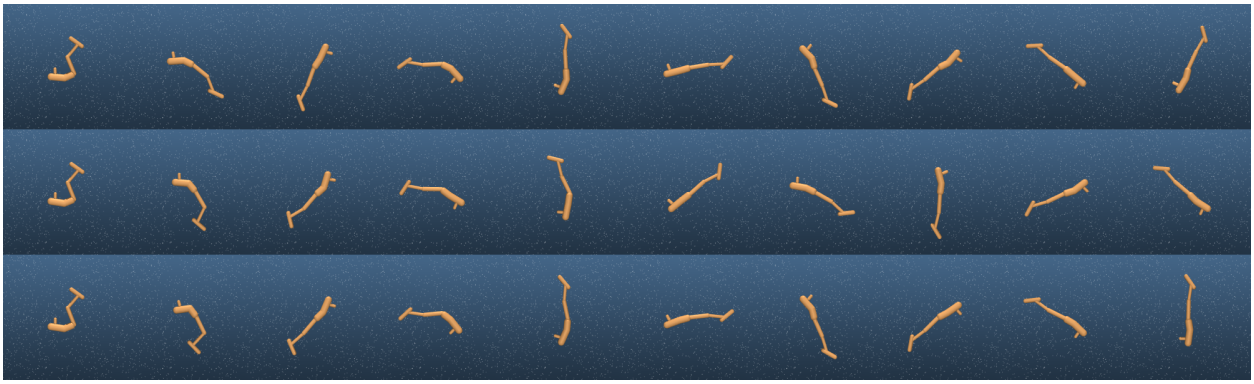
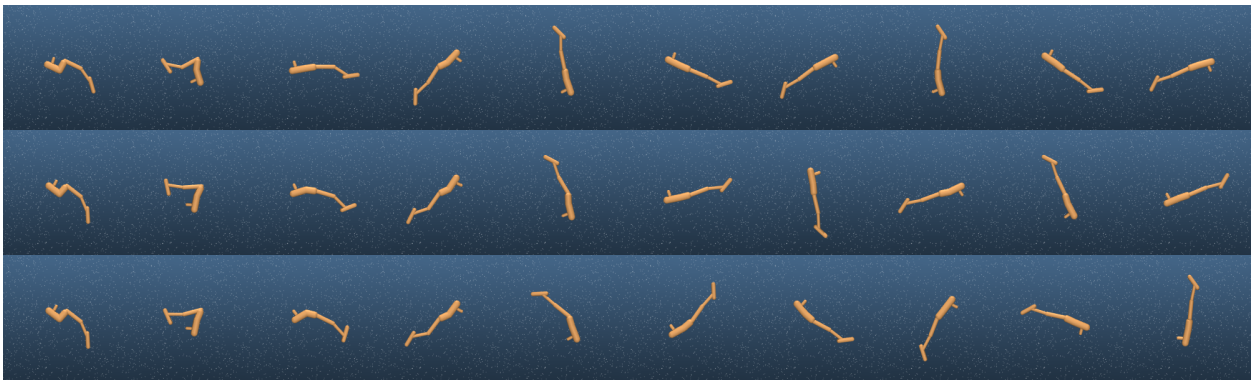
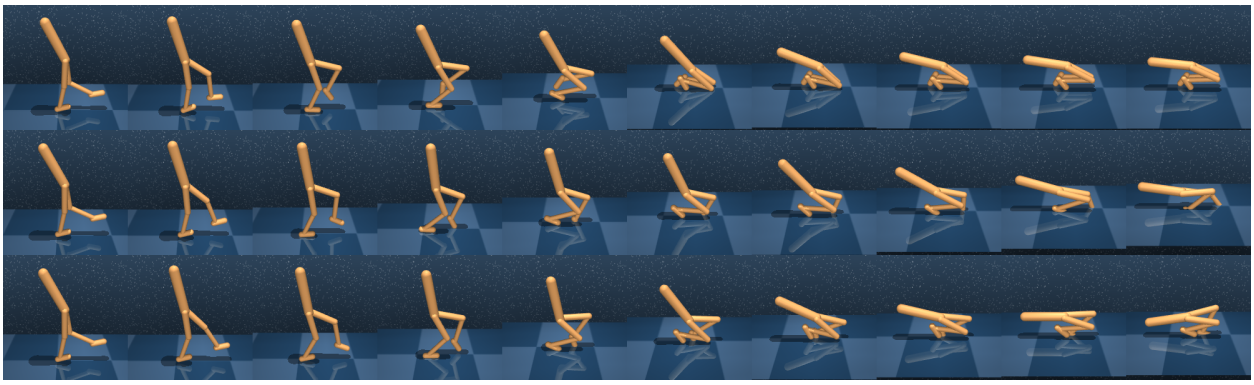


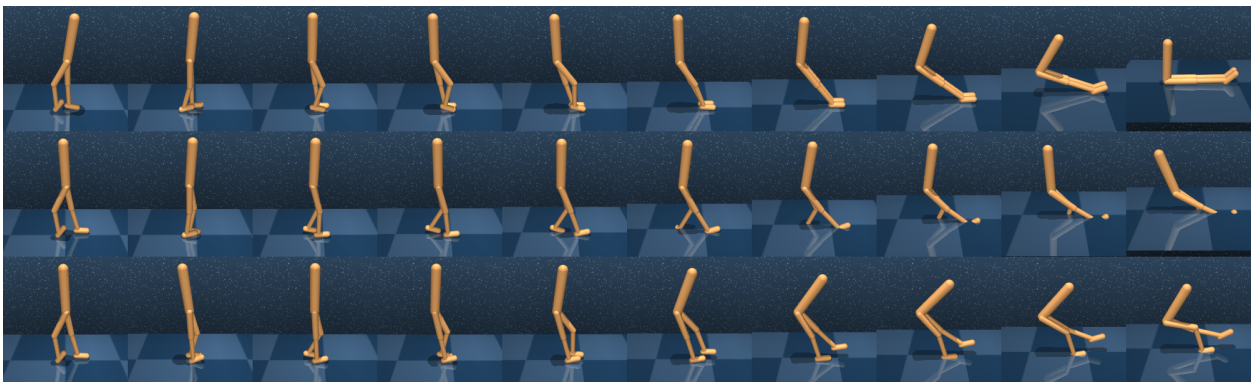
Figure D.2: Variation of samples for our model. Notice the difference in the angle of rotation given the same initial value, but varying the representation of transition function γ.

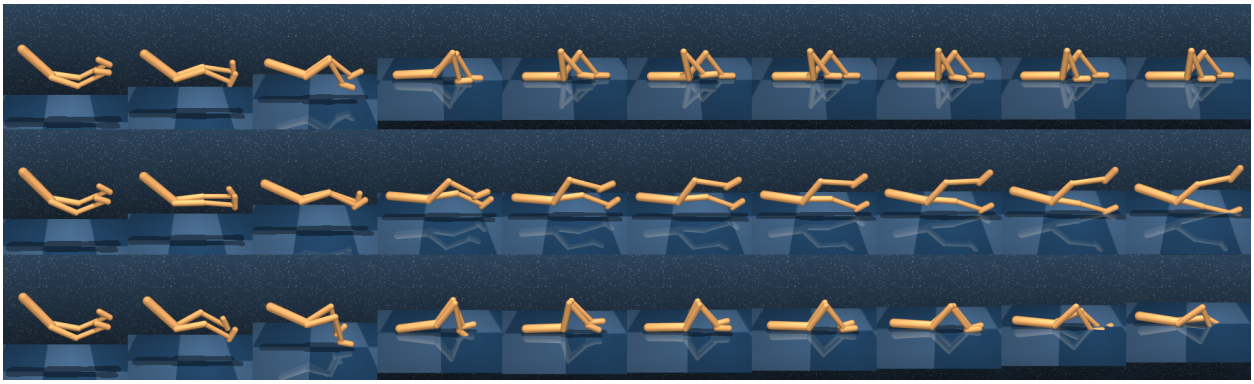


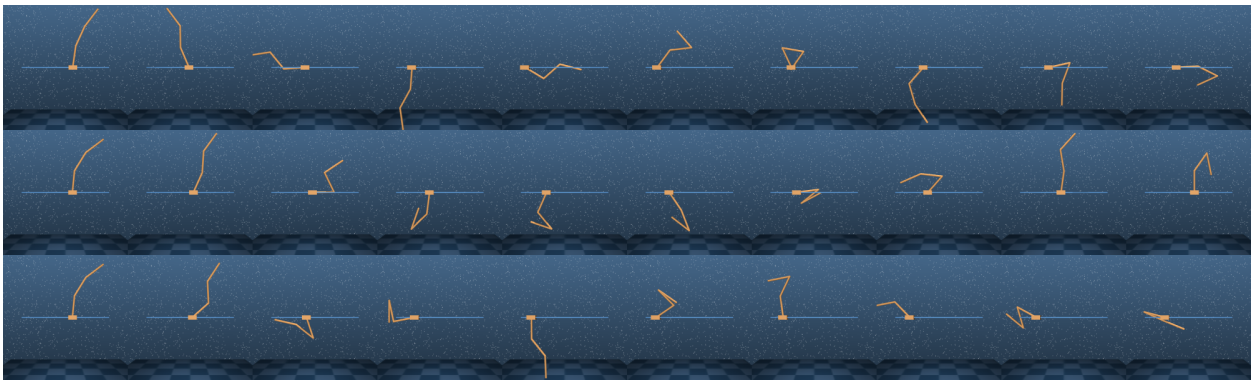


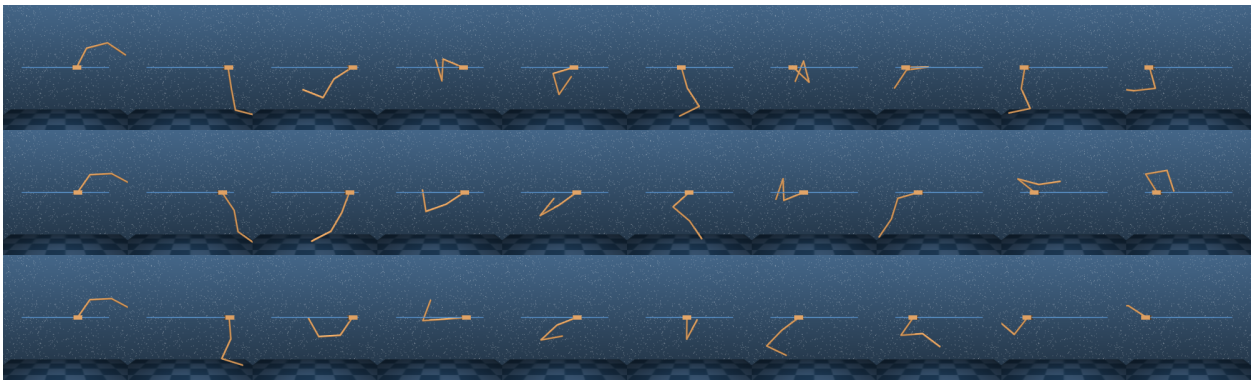


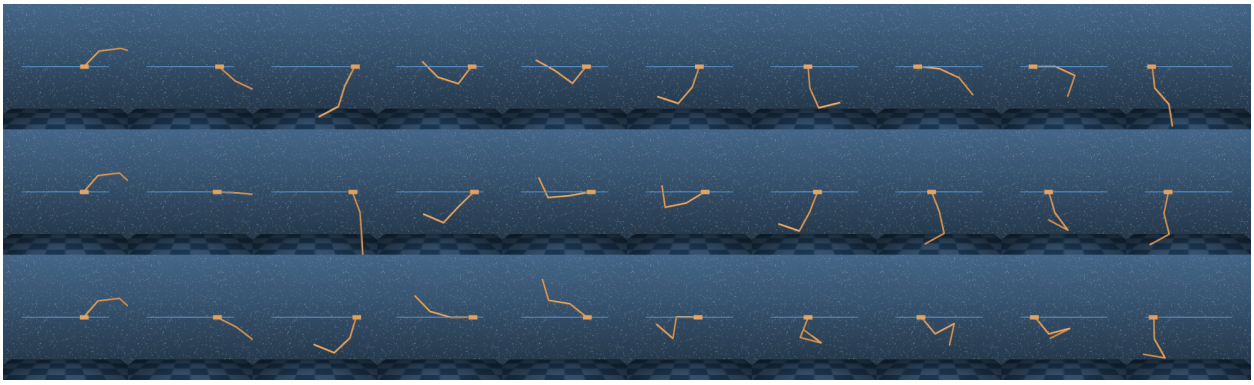












References

Adler, Robert J, Kevin Bartz, Sam C Kou, and Anthea Monod. 2017. Estimating thresholding levels for random fields via Euler characteristics. *arXiv preprint arXiv:1704.08562*.

Adler, Robert J, Jonathan E Taylor, Keith J Worsley, and Keith Worsley. 2007. Applications of random fields and geometry: Foundations and case studies. In *In preparation, available on R. Adler's homepage*.

Akkus, Zeynettin, Alfiia Galimzianova, Assaf Hoogi, Daniel L Rubin, and Bradley J Erickson. 2017. Deep learning for brain MRI segmentation: state of the art and future directions. *Journal of digital imaging* 30(4):449–459.

Almalioglu, Yasin, Muhamad Risqi U Saputra, Pedro PB de Gusmao, Andrew Markham, and Niki Trigoni. 2019. GANVO: Unsupervised deep monocular visual odometry and depth estimation with generative adversarial networks. In *International Conference on Robotics and Automation*. IEEE.

Altuğlu, Tuğçe Ballı, Barış Metin, Emine Elif Tülay, Oğuz Tan, Gökben Hızlı Sayar, Cumhur Taş, Kemal Arikan, and Nevzat Tarhan. 2020. Prediction of treatment resistance in obsessive compulsive disorder patients based on EEG complexity as a biomarker. *Clinical Neurophysiology* 131(3):716–724.

Arjovsky, Martin, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*. PMLR.

Aron, Arthur, and Elaine N Aron. 1999. *Statistics for psychology*. Prentice-Hall, Inc.

Ashburner, John. 2007. A fast diffeomorphic image registration algorithm. *Neuroimage* 38(1):95–113.

Bai, Jincheng, Qifan Song, and Guang Cheng. 2020. Efficient variational inference for sparse deep learning with theoretical guarantee. *Advances in Neural Information Processing Systems* 33.

Bakshi, Ainesh, and Pravesh K Kothari. 2021. List-Decodable Subspace Recovery: Dimension Independent Error in Polynomial Time. In *ACM-SIAM Symposium on Discrete Algorithms* . SIAM.

Baltrušaitis, Tadas, Chaitanya Ahuja, and Louis-Philippe Morency. 2018. Multimodal machine learning: A survey and taxonomy. *Transactions on Pattern Analysis and Machine Intelligence* 41(2).

Barlow, Roger J. 1993. *Statistics: a guide to the use of statistical methods in the physical sciences*, vol. 29. John Wiley & Sons.

Betthauser, Tobey J, Murat Bilgel, Rebecca L Koscik, Bruno M Jedynak, Yang An, Kristina A Kellett, Abhay Moghekar, Erin M Jonaitis, Charles K Stone, Corinne D Engelman, et al. 2022. Multi-method investigation of factors influencing amyloid onset and impairment in three cohorts. *Brain* 145(11):4065–4079.

Blasch, Erik, Robert Cruise, Alexander Aved, Uttam Majumder, and Todd Rovito. 2019. Methods of AI for multimodal sensing and action for complex situations. *AI Magazine* 40(4):50–65.

Blott, Gregor, Masato Takami, and Christian Heipke. 2018. Semantic segmentation of fisheye images. In *European Conference on Computer Vision workshop*.

Blundell, Charles, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. 2015. Weight uncertainty in neural network. In *International Conference on Machine Learning*.

Bouriaud, Olivier, G Stefan, and Laurent Saint-André. 2019. Comparing local calibration using random effects estimation and Bayesian calibrations: a case study with a mixed effect stem profile model. *Annals of Forest Science* 76(3):1–12.

Briot, Jean-Pierre, Gaëtan Hadjeres, and François-David Pachet. 2017. Deep learning techniques for music generation–a survey. *arXiv preprint arXiv:1709.01620*.

Brockwell, Peter J. 2001. Continuous-time ARMA processes. *Handbook of statistics* 19: 249–276.

Brunton, Steven L, Bernd R Noack, and Petros Koumoutsakos. 2020. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics* 52:477–508.

Cao, Hu, Yueyue Wang, Joy Chen, Dongsheng Jiang, Xiaopeng Zhang, Qi Tian, and Manning Wang. 2023. Swin-unet: Unet-like pure transformer for medical image segmentation. In *European Conference on Computer Vision workshop*.

Cao, J, and KJ Worsley. 2001. Applications of random fields in human brain mapping. In *Spatial Statistics: Methodological Aspects and Applications*, 169–182. Springer.

Carlson, Alexandra, Katherine A Skinner, Ram Vasudevan, and Matthew Johnson-Roberson. 2018. Modeling camera effects to improve visual learning from synthetic data. In *European Conference on Computer Vision workshop*.

Carvalho, Carlos M, Nicholas G Polson, and James G Scott. 2009. Handling sparsity via the horseshoe. In *Artificial Intelligence and Statistics*.

Chatfield, Chris. 1995. Model uncertainty, data mining and statistical inference. *Journal of the Royal Statistical Society: Series A (Statistics in Society)* 158(3):419–444.

Chen, Jianwei, and Hulin Wu. 2008. Efficient local estimation for time-varying coefficients in deterministic dynamic models with applications to HIV-1 dynamics. *Journal of the American Statistical Association* 103(481):369–384.

Chen, Liang-Chieh, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. 2018a. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *Transactions on Pattern Analysis and Machine Intelligence* 40(4).

Chen, Liang-Chieh, George Papandreou, Florian Schroff, and Hartwig Adam. 2017. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*.

Chen, Ricky TQ, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018b. Neural ordinary differential equations. *Advances in Neural Information Processing Systems* 31.

Chen, Xi, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2016. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*.

Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Choi, Yunjey, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. 2020. Stargan v2: Diverse image synthesis for multiple domains. In *Conference on Computer Vision and Pattern Recognition*.

Chung, Junyoung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. 2015. A recurrent latent variable model for sequential data. *Advances in Neural Information Processing Systems* 28.

Cui, Henggang, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. 2019. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In *International Conference on Robotics and Automation*.

Dai, Bin, Yu Wang, John Aston, Gang Hua, and David Wipf. 2018. Connections with robust PCA and the role of emergent sparsity in variational autoencoder models. *The Journal of Machine Learning Research* 19(1).

Danielsson, Jon. 1994. Stochastic volatility in asset prices estimation with simulated maximum likelihood. *Journal of Econometrics* 64(1-2):375–400.

De'ath, Glenn, and Katharina E Fabricius. 2000. Classification and regression trees: a powerful yet simple technique for ecological data analysis. *Ecology* 81(11).

Del Moral, Pierre, Arnaud Doucet, and Ajay Jasra. 2012. An adaptive sequential Monte Carlo method for approximate Bayesian computation. *Statistics and Computing* 22(5): 1009–1020.

Demidenko, Eugene. 2013. *Mixed models: theory and applications with R*. John Wiley & Sons.

Deng, Wei, Xiao Zhang, Faming Liang, and Guang Lin. 2019. An adaptive empirical bayesian method for sparse deep learning. *Advances in Neural Information Processing Systems* 2019.

Derksen, Shelley, and Harvey J Keselman. 1992. Backward, forward and stepwise automated subset selection algorithms: Frequency of obtaining authentic and noise variables. *British Journal of Mathematical and Statistical Psychology* 45(2):265–282.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dillon, Joshua V, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. 2017. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*.

Ditlevsen, Susanne, and Andrea De Gaetano. 2005. Mixed effects in stochastic differential equation models. *REVSTAT-Statistical Journal* 3(2):137–153.

Dupont, Emilien, Hyunjik Kim, SM Eslami, Danilo Rezende, and Dan Rosenbaum. 2022. From data to functa: Your data point is a function and you should treat it like one. *arXiv preprint arXiv:2201.12204*.

Dusenberry, Michael, Ghassen Jerfel, Yeming Wen, Yian Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran. 2020. Efficient and scalable bayesian neural nets with rank-1 factors. In *International Conference on Machine Learning*.

Eisenhauer, Elizabeth A, Patrick Therasse, Jan Bogaerts, Lawrence H Schwartz, Danielle Sargent, Robert Ford, Janet Dancey, S Arbuck, Steve Gwyther, Margaret Mooney, et al. 2009. New response evaluation criteria in solid tumours: revised RECIST guideline (version 1.1). *European journal of cancer* 45(2):228–247.

Erden, Sacide, Hande Mefkure Ozkaya, Sabiha Banu Denizeri, and Emrah Karabacak. 2016. The effects of home blood pressure monitoring on blood pressure control and treatment planning. *Postgraduate medicine* 128(6):584–590.

Escobar, Michael D, and Mike West. 1995. Bayesian density estimation and inference using mixtures. *Journal of the American statistical association* 90(430):577–588.

Fang, Yun, Hulin Wu, and Li-Xing Zhu. 2011. A two-stage estimation method for random coefficient differential equation models with application to longitudinal HIV dynamic data. *Statistica Sinica* 21(3):1145.

Farquhar, Sebastian, Michael A Osborne, and Yarin Gal. 2020. Radial bayesian neural networks: Beyond discrete support in large-scale bayesian deep learning. In *International Conference on Artificial Intelligence and Statistics*.

Fearnhead, Paul, and Dennis Prangle. 2010. Semi-automatic approximate Bayesian computation. *Arxiv preprint arXiv*.

Fitzmaurice, Garrett M, Nan M Laird, and James H Ware. 2012. *Applied longitudinal analysis*. John Wiley & Sons.

Fortuin, Vincent, Adrià Garriga-Alonso, Florian Wenzel, Gunnar Ratsch, Richard E Turner, Mark van der Wilk, and Laurence Aitchison. 2020. Bayesian Neural Network Priors Revisited. In *Advances in Neural Information Processing Systems workshop*.

Franckena, Martine, Richard Canters, Fabian Termorshuizen, Jacoba Van Der Zee, and Gerard Van Rhoon. 2010. Clinical implementation of hyperthermia treatment planning guided steering: A cross over trial to assess its current contribution to treatment quality. *International Journal of Hyperthermia* 26(2):145–157.

Frison, Lars, and Stuart J Pocock. 1992. Repeated measures in clinical trials: analysis using mean summary statistics and its implications for design. *Statistics in medicine* 11(13): 1685–1704.

Fu, Huan, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. 2018. Deep ordinal regression network for monocular depth estimation. In *Conference on Computer Vision and Pattern Recognition*.

Fuglede, Bent, and Flemming Topsoe. 2004. Jensen-Shannon divergence and Hilbert space embedding. In *International Symposium on Information Theory*. IEEE.

Gaidon, Adrien, Qiao Wang, Yohann Cabon, and Eleonora Vig. 2016. Virtual worlds as proxy for multi-object tracking analysis. In *Conference on Computer Vision and Pattern Recognition*.

Gal, Yarin, and Zoubin Ghahramani. 2015. Bayesian convolutional neural networks with Bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*.

———. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*.

Gashler, Michael S, and Stephen C Ashmore. 2014. Training deep fourier neural networks to fit time-series data. In *International Conference on Intelligent Computing*. Springer.

Geiger, Andreas, Philip Lenz, Christoph Stiller, and Raquel Urtasun. 2013. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research*.

Ghosh, Partha, Mehdi S. M. Sajjadi, Antonio Vergari, Michael Black, and Bernhard Scholkopf. 2020. From variational to deterministic autoencoders. In *International Conference on Learning Representations*.

Ghosh, Soumya, and Finale Doshi-Velez. 2017. Model selection in Bayesian neural networks via horseshoe priors. *arXiv preprint arXiv:1705.10388*.

Goan, Ethan, and Clinton Fookes. 2020. Bayesian neural networks: An introduction and survey. In *Case Studies in Applied Bayesian Data Science*, 45–87. Springer.

Godard, Clément, Oisin Mac Aodha, and Gabriel J Brostow. 2017. Unsupervised monocular depth estimation with left-right consistency. In *Conference on Computer Vision and Pattern Recognition*.

Goodfellow, Ian J, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*.

Graves, Alex. 2011. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*.

Griewank, Andreas. 2012. Who invented the reverse mode of differentiation. *Documenta Mathematica, Extra Volume ISMP* 389–400.

Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*.

Gustafsson, Fredrik K, Martin Danelljan, and Thomas B Schon. 2020. Evaluating scalable bayesian deep learning methods for robust computer vision. In *Conference on Computer Vision and Pattern Recognition workshop*.

Hairer, Martin, and Étienne Pardoux. 2015. A Wong-Zakai theorem for stochastic PDEs. *Journal of the Mathematical Society of Japan* 67(4):1551–1604.

Hartigan, John A, and Manchek A Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)* 28(1).

He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask R-CNN. In *International Conference on Computer Vision*.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016a. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*.

———. 2016b. Identity mappings in deep residual networks. In *European Conference on Computer Vision*.

Heckerman, David. 1997. Bayesian networks for data mining. *Data mining and knowledge discovery* 1(1).

Hedeker, Donald, and Robert D Gibbons. 2006. *Longitudinal data analysis*, vol. 451. John Wiley & Sons.

Hernández-Lobato, José Miguel, and Ryan Adams. 2015. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*.

Hershey, John R, and Peder A Olsen. 2007. Approximating the Kullback Leibler divergence between Gaussian mixture models. In *International Conference on Acoustics, Speech and Signal Processing*, vol. 4. IEEE.

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

Hodgkinson, Liam, Chris van der Heide, Fred Roosta, and Michael W Mahoney. 2020. Stochastic normalizing flows. *arXiv preprint arXiv:2002.09547*.

Holgersson, HET, and Ghazi Shukur. 2004. Testing for multivariate heteroscedasticity. *Journal of Statistical Computation and Simulation* 74(12):879–896.

Huang, Chin-Wei, Shawn Tan, Alexandre Lacoste, and Aaron C Courville. 2018. Improving explorability in variational inference with annealed variational objectives. *Advances in Neural Information Processing Systems* 31.

Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Conference on Computer Vision and Pattern Recognition*.

Huang, Yangxin, Tao Lu, et al. 2008. Modeling long-term longitudinal HIV dynamics with application to an AIDS clinical study. *The Annals of Applied Statistics* 2(4):1384–1408.

Huang, Zhiheng, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Hubschneider, Christian, Robin Hutmacher, and J Marius Zöllner. 2019. Calibrating uncertainty models for steering angle estimation. In *Intelligent Transportation Systems Conference*.

Hyun, Jung Won, Yimei Li, Chao Huang, Martin Styner, Weili Lin, Hongtu Zhu, Alzheimer's Disease Neuroimaging Initiative, et al. 2016. STGP: Spatio-temporal Gaussian process models for longitudinal neuroimaging data. *Neuroimage* 134:550–562.

Iserles, Arieh, Hans Z Munthe-Kaas, Syvert P Nørsett, and Antonella Zanna. 2000. Lie-group methods. *Acta numerica* 9:215–365.

Iyyer, Mohit, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. *arXiv preprint arXiv:1804.06059*.

Jacot, Arthur, Franck Gabriel, and Clément Hongler. 2018. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in Neural Information Processing Systems* 31.

Jaderberg, Max, Karen Simonyan, Andrew Zisserman, et al. 2015. Spatial transformer networks. *Advances in Neural Information Processing Systems* 28.

Joshi, Gargi, Rahee Walambe, and Ketan Kotecha. 2021. A review on explainability in multimodal deep neural nets. *IEEE Access* 9:59800–59821.

Kant, Yash, Abhinav Moudgil, Dhruv Batra, Devi Parikh, and Harsh Agrawal. 2021. Contrast and classify: Training robust vqa models. In *International Conference on Computer Vision*.

Katsev, Sergei, and Ivan L'Heureux. 2003. Are Hurst exponents estimated from short or irregular time series meaningful? *Computers & Geosciences* 29(9):1085–1089.

Kendall, Alex, and Yarin Gal. 2017. What uncertainties do we need in bayesian deep learning for computer vision? *arXiv preprint arXiv:1703.04977*.

Kidger, Patrick, James Morrill, James Foster, and Terry Lyons. 2020. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems* 33.

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

Kingma, Diederik P, and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, Durk P, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. 2016. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, ed. D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, vol. 29. Curran Associates, Inc.

Kingma, Durk P, Tim Salimans, and Max Welling. 2015. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*.

Kiser, Kendall J, Clifton D Fuller, and Valerie K Reed. 2019. Artificial intelligence in radiation oncology treatment planning: a brief overview. *J Med Artif Intell* 2(9).

Kocić, Jelena, Nenad Jovičić, and Vujo Drndarević. 2018. Sensors and sensor fusion in autonomous vehicles. In *Telecommunications Forum* .

Konda, Kishore Reddy, and Roland Memisevic. 2015. Learning visual odometry with a convolutional network. In *VISAPP*.

Kononenko, Igor. 1989. Bayesian neural networks. *Biological Cybernetics* 61(5):361–370.

Koopman, Bernard Osgood. 1936. On distributions admitting a sufficient statistic. *Transactions of the American Mathematical society* 39(3):399–409.

Kreindler, David M, and Charles J Lumsden. 2006. The Effects of the Irregular Sample and Missing Data in Time Series Analysis. *Nonlinear dynamics, psychology, and life sciences*.

Krishnan, Ranganath, Mahesh Subedar, and Omesh Tickoo. 2019. Efficient Priors for Scalable Variational Inference in Bayesian Deep Neural Networks. In *International Conference on Computer Vision workshop*.

Kruschke, John K, and Torrin M Liddell. 2018. The Bayesian New Statistics: Hypothesis testing, estimation, meta-analysis, and power analysis from a Bayesian perspective. *Psychonomic bulletin & review* 25(1):178–206.

Kuang, Dongyang. 2019. Cycle-Consistent training for reducing negative Jacobian determinant in deep registration networks. In *International workshop on Simulation and Synthesis in Medical Imaging*.

Kumar, Varun Ravi, Ciarán Eising, Christian Witt, and Senthil Yogamani. 2023. Surround-View Fisheye Camera Perception for Automated Driving: Overview, Survey & Challenges. *Transactions on Intelligent Transportation Systems*.

Lai, Guokun, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. 2018. Modeling long-and short-term temporal patterns with deep neural networks. In *International ACM SIGIR conference on research & development in information retrieval*.

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*.

Lee, Jaehoon, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. 2017. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*.

Legendre, Adrien Marie. 1806. *Nouvelles méthodes pour la détermination des orbites des comètes; par AM Legendre...* chez Firmin Didot, libraire pour lew mathematiques.

Lewis, Mike, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Li, Xuechen, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. 2020. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*. PMLR.

Li, Yingzhen, and Richard E Turner. 2017. Gradient estimators for implicit models. *arXiv preprint arXiv:1705.07107*.

Liang, Hua, and Yao Yu. 2013. Parameter estimation for HIV ODE models incorporating longitudinal structure. *Statistics and Its Interface* 6(1):9–18.

Lin, Tsung-Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*.

Liu, Xuanqing, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. 2019. Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*.

Liu, Ziwei, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep learning face attributes in the wild. In *International Conference on Computer Vision*.

Loeve, M. 1963. *Probability Theory II*. Springer-Verlag.

Lu, Dan, Siyan Liu, and Daniel Ricciuto. 2019. An efficient Bayesian method for advancing the application of deep learning in earth science. In *International Conference on Data Mining workshop*. IEEE.

Lu, George Y, and David W Wong. 2008. An adaptive inverse-distance weighting spatial interpolation technique. *Computers & geosciences* 34(9):1044–1055.

Luo, Huaishao, Lei Ji, Botian Shi, Haoyang Huang, Nan Duan, Tianrui Li, Jason Li, Taroon Bharti, and Ming Zhou. 2020. Univl: A unified video and language pre-training model for multimodal understanding and generation. *arXiv preprint arXiv:2002.06353*.

Lusch, Bethany, J Nathan Kutz, and Steven L Brunton. 2018. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications* 9(1):1–10.

Lütkepohl, Helmut. 1985. Comparison of criteria for estimating the order of a vector autoregressive process. *Journal of Time Series Analysis* 6(1).

Lv, Yisheng, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. 2014. Traffic flow prediction with big data: a deep learning approach. *Transactions on Intelligent Transportation Systems* 16(2):865–873.

Ma, Lin, Zhengdong Lu, Lifeng Shang, and Hang Li. 2015. Multimodal convolutional neural networks for matching image and sentence. In *International Conference on Computer Vision*.

MacKay, David JC. 1995. Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 354(1):73–80.

MaCurdy, Thomas E. 1982. The use of time series processes to model the error structure of earnings in a longitudinal data analysis. *Journal of econometrics* 18(1):83–114.

Mao, Junhua, Wei Xu, Yi Yang, Jiang Wang, and Alan L Yuille. 2014. Explain images with multimodal recurrent neural networks. *arXiv preprint arXiv:1410.1090*.

Mao, Xiaojiao, Chunhua Shen, and Yu-Bin Yang. 2016. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. *Advances in Neural Information Processing Systems* 29.

Marin, Jean-Michel, Pierre Pudlo, Christian P Robert, and Robin J Ryder. 2012. Approximate Bayesian computational methods. *Statistics and Computing* 22(6):1167–1180.

Marinescu, Razvan V, Neil P Oxtoby, Alexandra L Young, Esther E Bron, Arthur W Toga, Michael W Weiner, Frederik Barkhof, Nick C Fox, Stefan Klein, Daniel C Alexander, et al. 2018. Tadpole challenge: Prediction of longitudinal evolution in Alzheimer's disease. *arXiv preprint arXiv:1805.03909*.

McCulloch, J Huston. 1985. Miscellanea on Heteros* edasticity. *Econometrica (pre-1986)* 53(2):483.

McCulloch, Warren S, and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* 5(4):115–133.

McGregor, Felix, Arnu Pretorius, Johan du Preez, and Steve Kroon. 2019. Stabilising priors for robust Bayesian deep learning. *arXiv preprint arXiv:1910.10386*.

Menard, Scott. 2002. *Applied logistic regression analysis*, vol. 106. Sage.

Meyer, Joseph S, Christopher G Ingersoll, Lyman L McDonald, and Marks S Boyce. 1986. Estimating uncertainty in population growth rates: jackknife vs. bootstrap techniques. *Ecology* 67(5):1156–1166.

Miller, Andrew, Nick Foti, Alexander D'Amour, and Ryan P Adams. 2017. Reducing reparameterization gradient variance. In *Advances in Neural Information Processing Systems*.

Minaee, Shervin, Yuri Y Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. 2021. Image segmentation using deep learning: A survey. *Transactions on Pattern Analysis and Machine Intelligence*.

Morrill, James, Cristopher Salvi, Patrick Kidger, and James Foster. 2021. Neural rough differential equations for long time series. In *International Conference on Machine Learning*.

Mueller, Susanne G., Michael W. Weiner, Leon J. Thal, Ronald C. Petersen, Clifford Jack, William Jagust, John Q. Trojanowski, Arthur W. Toga, and Laurel Beckett. 2005. The alzheimer's disease neuroimaging initiative. *Neuroimaging Clinics of North America* 15(4): 869–877. Alzheimer's Disease: 100 Years of Progress.

Nazarovs, Jurijs, Rudrasis Chakraborty, Songwong Tasneeyapant, Sathya Ravi, and Vikas Singh. 2021a. A variational approximation for analyzing the dynamics of panel data. In *Uncertainty in Artificial Intelligence*.

Nazarovs, Jurijs, Zhichun Huang, Songwong Tasneeyapant, Rudrasis Chakraborty, and Vikas Singh. 2022a. Understanding uncertainty maps in vision with statistical testing. In *Conference on Computer Vision and Pattern Recognition*.

Nazarovs, Jurijs, Ronak R Mehta, Vishnu Suresh Lokhande, and Vikas Singh. 2021b. Graph Reparameterizations for Enabling 1000+ Monte Carlo Iterations in Bayesian Deep Neural Networks. In *Uncertainty in Artificial Intelligence*, vol. 2021.

Nazarovs, Jurijs, Jack W Stokes, Melissa Turcotte, Justin Carroll, and Itai Grady. 2022b. Radial spike and slab bayesian neural networks for sparse data in ransomware attacks. *arXiv preprint arXiv:2205.14759*.

Newton, Michael, Nicholas G Polson, and Jianeng Xu. 2018. Weighted Bayesian Bootstrap for Scalable Bayes. *arXiv preprint arXiv:1803.04559*.

Nilsson, David, and Cristian Sminchisescu. 2018. Semantic video segmentation by gated recurrent flow propagation. In *Conference on Computer Vision and Pattern Recognition*.

Novak, Roman, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Greg Yang, Jiri Hron, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. 2018. Bayesian deep convolutional networks with many channels are gaussian processes. *arXiv preprint arXiv:1810.05148*.

Öhrnell, Carl. 2020. Lie Groups and PDE.

Paisley, John, David Blei, and Michael Jordan. 2012. Variational Bayesian inference with stochastic search. *arXiv preprint arXiv:1206.6430*.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*. Curran Associates, Inc.

Pearce, Tim, Mohamed Zaki, and Andy Neely. 2018. Bayesian Neural Network Ensembles. *arXiv preprint arXiv:1811.12188*.

Pettigrew, Corinne, Jurijs Nazarovs, Anja Soldan, Vikas Singh, Jiangxia Wang, Timothy Hohman, Logan Dumitrescu, Julia Libby, Brian Kunkle, Alden L Gross, et al. 2023. Alzheimer's disease genetic risk and cognitive reserve in relationship to long-term cognitive trajectories among cognitively normal individuals. *Alzheimer's Research & Therapy* 15(1):1–16.

Pierson, Emma, Pang Wei Koh, Tatsunori Hashimoto, Daphne Koller, Jure Leskovec, Nicholas Eriksson, and Percy Liang. 2019. Inferring multidimensional rates of aging from cross-sectional data. *Proceedings of machine learning research* 89:97.

Poplin, Ryan, Avinash V Varadarajan, Katy Blumer, Yun Liu, Michael V McConnell, Greg S Corrado, Lily Peng, and Dale R Webster. 2018. Prediction of cardiovascular risk factors from retinal fundus photographs via deep learning. *Nature Biomedical Engineering* 2(3): 158–164.

Pratt, Harry, Bryan Williams, Frans Coenen, and Yalin Zheng. 2017. Fcnn: Fourier convolutional neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer.

Pu, Yunchen, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. 2016. Variational autoencoder for deep learning of images, labels and captions. *arXiv preprint arXiv:1609.08976*.

PyTorch. 2023. *Autograd mechanics*.

Qi, Charles R, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Conference on Computer Vision and Pattern Recognition*.

Radford, Alec, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Ramachandran, Prajit, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. 2019. Stand-alone self-attention in vision models. *Advances in Neural Information Processing Systems* 32.

Ranganath, Rajesh, Sean Gerrish, and David Blei. 2014. Black box variational inference. In *Artificial Intelligence and Statistics*, 814–822.

Razavi, Ali, Aaron van den Oord, and Oriol Vinyals. 2019. Generating diverse high-fidelity images with vq-vae-2. In *Advances in neural information processing systems*, 14866–14876.

Rombach, Robin, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Conference on Computer Vision and Pattern Recognition*.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*.

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

Rousseau, François, Lucas Drumetz, and Ronan Fablet. 2020. Residual networks as flows of diffeomorphisms. *Journal of Mathematical Imaging and Vision* 62(3):365–375.

Royo, Santiago, and Maria Ballesta-Garcia. 2019. An overview of lidar imaging systems for autonomous vehicles. *Applied sciences* 9(19):4093.

Rubanova, Yulia, Ricky T. Q. Chen, and David K Duvenaud. 2019. Latent Ordinary Differential Equations for Irregularly-Sampled Time Series. In *Advances in Neural Information Processing Systems*, vol. 32.

Ruck, Dennis W, Steven K Rogers, Matthew Kabrisky, Mark E Oxley, and Bruce W Suter. 1990. The multilayer perceptron as an approximation to a Bayes optimal discriminant function. *IEEE transactions on neural networks* 1(4).

Rudy, Samuel H, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. 2017. Data-driven discovery of partial differential equations. *Science advances* 3(4):e1602614.

Rüschendorf, Ludger. 1985. The Wasserstein distance and approximation theorems. *Probability Theory and Related Fields* 70(1).

Särkkä, Simo, and Arno Solin. 2014. Lecture notes on applied stochastic differential equations, 2014. *Version as of December* 4.

———. 2019. *Applied stochastic differential equations*, vol. 10. Cambridge University Press.

Sarzynska-Wawer, Justyna, Aleksander Wawer, Aleksandra Pawlak, Julia Szymanowska, Izabela Stefaniak, Michal Jarkiewicz, and Lukasz Okruszek. 2021. Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research* 304: 114135.

Scott, Ian, and Deborah Mazhindu. 2014. *Statistics for healthcare professionals: An introduction*. Sage.

Shahroudy, Amir, Jun Liu, Tian-Tsong Ng, and Gang Wang. 2016. Ntu rgb+ d: A large scale dataset for 3d human activity analysis. In *Conference on Computer Vision and Pattern Recognition*.

Simonyan, Karen, and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Song, Yang, and Stefano Ermon. 2019. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems* 32.

Song, Yang, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2020. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*.

Storey, John D, and Robert Tibshirani. 2003. Statistical significance for genomewide studies. *National Academy of Sciences* 100(16):9440–9445.

Streiner, David L, and Geoffrey R Norman. 2011. Correction for multiple testing: is there a resolution? *Chest* 140(1):16–18.

Strike, Lachlan T, Narelle K Hansell, Kai-Hsiang Chuang, Jessica L Miller, Greig I de Zubicaray, Paul M Thompson, Katie L McMahon, and Margaret J Wright. 2022. The Queensland Twin Adolescent Brain Project, a longitudinal study of adolescent brain development. *bioRxiv*.

Taylor, Jonathan, Akimichi Takemura, Robert J Adler, et al. 2005. Validity of the expected Euler characteristic heuristic. *Annals of Probability* 33(4):1362–1396.

Taylor, Jonathan E, and Keith J Worsley. 2007. Detecting sparse signals in random fields, with an application to brain mapping. *Journal of the American Statistical Association* 102(479): 913–928.

Taylor, Jonathan E, et al. 2006. A Gaussian kinematic formula. *Annals of probability* 34(1): 122–158.

Todorov, Emanuel, Tom Erez, and Yuval Tassa. 2012. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*.

Trick, Susanne, Dorothea Koert, Jan Peters, and Constantin A Rothkopf. 2019. Multimodal uncertainty reduction for intention recognition in human-robot interaction. In *International Conference on Intelligent Robots and Systems*.

Trumpler, Robert Julius, and Harold F Weaver. 1953. *Statistical astronomy*. Univ of California Press.

Tzen, Belinda, and Maxim Raginsky. 2019. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*.

Vahdat, Arash, and Jan Kautz. 2020. Nvae: A deep hierarchical variational autoencoder. In *Advances in Neural Information Processing Systems*, ed. H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, vol. 33. Curran Associates, Inc.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* 30.

van Velzen, Sanne GM, Steffen Bruns, Jelmer M Wolterink, Tim Leiner, Max A Viergever, Helena M Verkooijen, and Ivana Išgum. 2022. AI-based quantification of planned radiation therapy dose to cardiac structures and coronary arteries in patients with breast cancer. *International Journal of Radiation Oncology* Biology* Physics* 112(3):611–620.

Vempala, Santosh S. 2005. *The random projection method*, vol. 65. American Mathematical Soc.

Vincent, Pascal, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*.

Vo, Tien, Akshay Mishra, Vamsi Ithapu, Vikas Singh, and Michael A Newton. 2021. Dimension constraints improve hypothesis testing for large-scale, graph-associated, brain-image data. *Biostatistics*. Kxab001.

Wang, L, Jiguo Cao, JO Ramsay, DM Burger, CJL Laporte, and JK Rockstroh. 2014. Estimating mixed-effects differential equation models. *Statistics and Computing* 24(1):111–121.

Wang, Xiaolong, and Abhinav Gupta. 2016. Generative image modeling using style and structure adversarial networks. In *European conference on computer vision*. Springer.

Wenzel, Florian, Kevin Roth, Bastiaan S Veeling, Jakub Świątkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. 2020. How good is the bayes posterior in deep neural networks really? *arXiv preprint arXiv:2002.02405*.

Whitaker, Gavin A, Andrew Golightly, Richard J Boys, Chris Sherlock, et al. 2017. Bayesian inference for diffusion-driven mixed-effects models. *Bayesian Analysis* 12(2):435–463.

Wilkinson, Richard David. 2013. Approximate Bayesian computation (ABC) gives exact results under the assumption of model error. *Statistical applications in genetics and molecular biology* 12(2):129–141.

Williams, Christopher, and Carl Rasmussen. 1995. Gaussian processes for regression. *Advances in Neural Information Processing Systems* 8.

Wong, Eugene, and Moshe Zakai. 1965. On the convergence of ordinary integrals to stochastic integrals. *The Annals of Mathematical Statistics* 36(5):1560–1564.

Worsley, Keith J, Alan C Evans, Sean Marrett, and P Neelin. 1992. A three-dimensional statistical analysis for CBF activation studies in human brain. *Journal of Cerebral Blood Flow & Metabolism* 12(6):900–918.

Wu, A, S Nowozin, E Meeds, RE Turner, JM Hernández-Lobato, and AL Gaunt. 2019. Deterministic variational inference for robust Bayesian neural networks. In *International Conference on Learning Representations*.

Xiong, Yunyang, Hyunwoo J Kim, Bhargav Tangirala, Ronak Mehta, Sterling C Johnson, and Vikas Singh. 2019. On Training Deep 3D CNN Models with Dependent Samples in Neuroimaging. In *International Conference on Information Processing in Medical Imaging*. Springer.

Xu, Zhi-Qin John, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. 2019. Frequency principle: Fourier analysis sheds light on deep neural networks. *arXiv preprint arXiv:1901.06523*.

Yildiz, Cagatay, Markus Heinonen, and Harri Lahdesmaki. 2019. ODE2VAE: Deep generative second order ODEs with Bayesian neural networks. In *Advances in Neural Information Processing Systems*.

Yosida, Kösaku. 2012. *Functional analysis*. Springer Science & Business Media.

Yu, Xinli, Mohsen Malmir, Xin He, Jiangning Chen, Tong Wang, Yue Wu, Yue Liu, and Yang Liu. 2021. Cross interaction network for natural language guided video moment retrieval. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Zähle, Martina. 2011. Lipschitz-Killing curvatures of self-similar random fractals. *Transactions of the American Mathematical Society* 363(5):2663–2684.

Zhang, Muhan, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. 2019. D-vae: A variational autoencoder for directed acyclic graphs. *arXiv preprint arXiv:1904.11088*.

Zhou, Bolei, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. 2016. Learning deep features for discriminative localization. In *Conference on Computer Vision and Pattern Recognition*.

Zhou, Ding-Xuan. 2020. Universality of deep convolutional neural networks. *Applied and computational harmonic analysis* 48(2):787–794.