From the holdings of the Department of Special Collections , Princeton University Library, Princeton, NJ

These documents are either in the public domain, or can only be used for educational and research purposes (Fair use) as per U.S. Copyright law (text below). By accessing this file, all users agree that their use falls in the public domain or within fair use as defined by the copyrightlaw. We do not charge any permission or usage fees for the publication of images of material inour collections, including those provided by Princeton University Library, via our website, catalog, or directly from staff.For more information regarding copyright, credit, and citation please review the information onthe Copyright, Credit and Citation Guidelines page of our website.
https://library.princeton.edu/special-collections/policies/copyright-credit-and-citation-guidelines. If you have any questions, including how to order higher-quality images from the collection, contact Special Collections Public Services via our Ask Us! Form. https://library.princeton.edu/special-collections/ask-us

---

Master's Theses, 2022-2026

Computer Science, 2023

2023

Bayesian Filtering for Neural Networks

Bencomo, Gianluca Michele

Princeton, NJ : Princeton University

---

http://arks.princeton.edu/ark:/88435/dsp01d791sk48p

U.S. Copyright law: The copyright law of the United States (Title 17, §108, United States Code) governs the making of photocopies or other reproductions of copyrighted material. Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be used for any purpose other than private study, scholarship, or research. If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of fair use that user may be liable for copyright infringement. This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve a violation of copyright law. https://www.copyright.gov/title17/title17.pdf

Bayesian Filtering for Neural Networks

Gianluca M. Bencomo

Advisor: Thomas L. Griffiths

Reader: Ryan P. Adams

Submitted in partial fulfillment

of the requirements for the degree of

Masters of Science and Engineering

Department of Computer Science

Princeton University

May 2023

Abstract

Adaptability is a crucial component of intelligent systems. In dynamic environments, biological organisms continuously adjust their behavior to match the demands of the current situation. However, this fundamental feature of biological intelligence conflicts with a typical assumption of machine learning, which assumes that the data generating process is static over time. Bayesian filtering offers a rich toolkit for breaking this assumption, but the complex and high-dimensional space of neural networks poses several challenges. This thesis investigates these problems and presents progress for handling them. In the simple linear-Gaussian case, we show that we can effectively learn the dynamics and process noise of a time-varying hierarchical Bayesian linear regression model. This results in a simple linear regressor that can adapt its weights to fit lines without any data at forward timesteps. In the nonlinear case, we derive two methods for variational Kalman filtering, one based on conjugate gradients and the other on natural gradients. We find that we are able to produce promising results over simple networks but more work is required before we can learn dynamics and process noise in higher-dimensional, and more complex, parameter spaces.

Acknowledgements

I am very grateful to my advisor Tom Griffiths and my collaborator Jake Snell for their guidance in this work, bringing helpful insight and direction into every one of our discussions. Jake Snell was especially helpful in derivations of the conjugate gradient variational Kalman filter, formalizing connections with meta-learning, and implementations. I would also like to thank Ryan Adams for helpful feedback at the end of this thesis writing process and for identifying some possible directions for future research.

The work in this thesis was carried out at the Computational Cognitive Science Lab (CoCoSci) at Princeton University, led by Tom Griffiths. I am grateful for funding provided by an Assistantship in Instruction (AI) that allowed me to pursue coursework, teaching experience, and research during my time at Princeton University and summer funding provided by CoCoSci that allowed me to continue doing research over summer.

Contents

Abstract ii

Acknowledgements iii

List of Tables vi

List of Figures vii

1 Introduction **1**

2 Background **5**

2.1 Bayesian Filtering 8

2.1.1 Kalman Filtering 11

2.1.2 Variational Kalman Filtering 13

2.2 Bayesian Neural Networks 16

2.2.1 Stochastic Gradient Langevin Dynamics 17

2.2.2 Stochastic Variational Inference 19

3 Meta-Filtering in the Linear-Gaussian Case **22**

3.1 Time-varying hierarchical Bayesian linear regression 23

3.2 Dynamics and process noise estimation 27

3.3 Results 28

4 Meta-Filtering in the nonlinear case **31**

4.1 A few considerations 32

4.2 Conjugate gradient variational Kalman filtering 32
4.3 Natural gradient variational Kalman filtering 36
4.4 Results 41

5 Conclusion **46**

5.1 Approximate Inference 47
5.2 Dynamics Estimation 48
5.3 Process Noise Estimation 49

List of Tables

4.1 Yearbook Dataset Results. Sequential ERM and Batchwise ERM were the best performers for in-distribution (ID) and out-of-distribution (OOD) testing. All reported numbers are average and worst accuracies averaged over three different random seeds. The LOBPCG-VKF did not match the OOD performance of sequential ERM under any configuration tested. 45

List of Figures

1.1 Laptop evolution from 2004 to 2009 to 2014. In this time period, laptops got thinner and more powerful. This is not the only thing that evolved across the same time period. Cars, people, landscapes, and medicine are a few examples. Researchers should be interested in methods that can reason across time since it better reflects the evolving nature of our environment. 3

2.1 Diagrammatic representation of nonparametric variational inference as described in [6]. We start from some explicitly defined starting distribution q_0 and perform a series of transitions T, which generally represent an gradient-based optimization method. Since optimization is an entropy destroying process, the *implicitly* defined variational distributions $q_1, \ldots, q_T$ shrink over time. This creates a spectrum of distributions that range from underfitting to overfitting. See [6] for more. . . 6

2.2 Diagrammatic representation of nonparametric variational Kalman filtering extended from [6]. We start from some implicitly defined distribution q_N, which was determined by optimizing over data from a given timestep with gradient descent plus early stopping. Since we stopped early, we are not optimized to the MAP solution exactly but we are close (within some arbitrarily defined metric space). To optimize over the next time step, we apply some dynamics $\boldsymbol{A}$ and start a new task chain. The MAP solution is now a different location within our metric space and we again wish to optimize with early stopping. We can view the spectrum of underfitting and overfitting in this time-varying context as modulating the Kalman gain. 7

3.1 Drifting line dataset for 30 timesteps with no per datum noise. See generative process outlined in main text for more details. 28

3.2 (Left) Parameter tracking over all 40 timessteps. The first 30 times steps were used for training and are considered in-distribution. The next 10 timesteps, labeled with forward projection, are considered out-of-distribution and no data was used to fit those lines. The Gaussian Random Walk (GRW) and Empirical Risk Minimizer (ERM) assumed stationarity so the parameters stayed static in time while the true parameters and trained filter moved forward. (Right) Mean Squared Error over 40 timesteps. GRW and ERM show terrible generalization while the trained filter significantly reduces the accumulated error over time despite the lack of data. 30

4.1 Moons dataset visualisation under three training paradigms using a random seed. (left) Three timesteps were used as a training set for the three approaches shown. (right) Predictions were made on three test timesteps where the models were agnositic to the data. (top) Sequential empirical risk minimization. Gradient descent was performed until convergence for everything training time step indivually. (middle) Batchwise empirical risk minimization. Gradient descent was performed until convergence for stochastic mini-batches of size = 100 over all training time steps until convergence. (bottom) Proposed method (rank 40) with $\boldsymbol{Q}_t = \text{diag}(0.75)$. 42

4.2 Out-of-distribution performance per test timestamp on the moons dataset. Under all six configurations, three timesteps were used as a training set. 44

Chapter 1

Introduction

Our behavior can be summarized as actions made over an *approximate* understanding of our current environment. We did not have much of a choice in this. 3.5 billion years of evolution has consistently punished organisms who did not embody the physical requirements for life on Earth and most often rewarded organisms who reflected the most accurate understanding. Many microcosms of this idea exist today. Financial markets reward those who understand the true market forces underlying economic phenomena and punish those who do not [21]. Most car crashes boil down to something unknown or misconceived: an object not seen or a driving condition being misinterpreted as safe [7]. The best conversations often happen between those who understand each other, and the worst between those who do not. This ability to understand the current environmental conditions is an important hallmark of systems that behave intelligently.

Since Earth's environment is constantly changing, we can imagine that the only two types of organisms alive today are either adaptive or lucky. This adaptability has been a well documented in humans – we carry the cognitive ability of generalization and the many perturbations our environment undergoes over time rarely affects our ability to perform repeated tasks. Consider the task of driving a car: laws,

regulations, roads, and other drivers change day to day and country to country. Humans perform the task of driving in a seemingly trivial fashion, even though machine learning practitioners have massively struggled to solve this problem for autonomous vehicles. This is due to a number of reasons, one of which being the inherent *non-stationarity* associated with the problem, a kind of caution tape that surrounds many of the hardest problems in machine learning.

Non-stationary systems assume that the data generating process evolves over time. More specifically, the assumption is that the data is ordered and cannot be treated as a bag of data points or, less colloquially, independently and identically distributed (i.i.d.). This non-stationarity assumption can greatly simplify computation in many situations. For example, perceiving my living room is a lot easier to do if I have encoded the knowledge of where I was before, perhaps my kitchen. This reduces the search space of possible answers in the problem of "where am I?", which would be quite difficult to solve if the search space was every possible location in the universe. But this is at odds with the conventional treatment of neural networks, which often treats search spaces as completely unregularized, or loosely at best.

Let us denote $\mathcal{D} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$ as a set of N input-output pairs where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$ for every $i \in \{1, \ldots, N\}$. Neural networks attempt to learn some $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ by optimizing a parameter set $\boldsymbol{\theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K\}$ over some network architecture of K parameters. The most common approach to the problem of learning is to define some differentiable probability distribution over data, $p(\mathcal{D}|\boldsymbol{\theta})$, and use a gradient-based optimizer to find the parameters $\boldsymbol{\theta}^*$ such that

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta}). \tag{1.1}$$

This approach is often called *maximum likelihood estimation* (MLE) as the parameters $\boldsymbol{\theta}^*$ are set to maximize the likelihood of $\boldsymbol{\theta}$, given some distributional assumptions. If

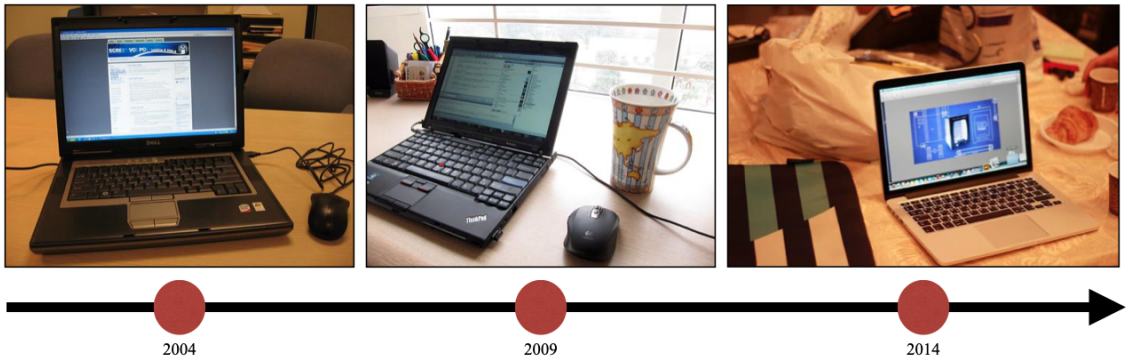


Figure 1.1: Laptop evolution from 2004 to 2009 to 2014. In this time period, laptops got thinner and more powerful. This is not the only thing that evolved across the same time period. Cars, people, landscapes, and medicine are a few examples. Researchers should be interested in methods that can reason across time since it better reflects the evolving nature of our environment.

we are dealing with large quantities of data and have no reason to consider an order across our N input-output pairs, this approach will likely be sufficient.

Problems with this approach begin to arise if the data is (1) ordered, (2) limited, and/or (3) some level of uncertainty quantification is desired. Humans are guaranteed to observe (1), often observe (2), and have (3) naturally built into our computations [19]. In the limited data regime (2), we can take advantage of Bayes' rule

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}, \tag{1.2}$$

and depending on whether or not our treatment is *fully Bayesian*, meaning every possible parameter is treated as a random variable with an associated prior distribution, we also achieve a sufficient level of uncertainty quantification over our system (3). This thesis will adopt the Bayesian perspective and, through this perspective, treat the problem of designing neural networks that carefully modify their behavior to appropriately handle ordered data (2) (see Figure 1.1 for a motivating example).

Throughout this thesis, we assume that the reader is familiar with the basic concepts of linear algebra, multivariate calculus, and probability. Chapter 2 will attempt

to prime the reader on some relevant background information, namely Bayesian filtering and two methods within the scope of Bayesian neural networks. Chapters 3 and 4 will cover the work done for this thesis in linear and nonlinear cases. The concluding chapter will focus on future directions for this work, as it remains largely incomplete.

Chapter 2

Background

The work presented in this thesis has been heavily influenced by a recently released benchmark on *temporal shift* [33]. Temporal shift is when the data generating distribution shifts over time, causing parameters that performed well on the training set to severely mismatch the parameters required to perform well on the test set. This phenomena requires either (1) methods that generate more generalizable parameter sets or (2) methods that evolve parameter sets over time without the observation of additional data. [33] showed that (2) can be accomplished by a method that closely resembles a Bayesian filter, which we call sequential empirical risk minimization (SERM) throughout this thesis.

Let $\boldsymbol{\theta}_t \in \mathbb{R}^D$ be the parameters of some neural network $f_{\boldsymbol{\theta}}$. At every time step t, consider the usage of first-order optimization to determine some

$$\boldsymbol{\theta}_t^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t; \boldsymbol{X}_t, \boldsymbol{y}_t), \tag{2.1}$$

where $\mathcal{L}(\cdot)$ is some differentiable loss function and $(\boldsymbol{X}_t, \boldsymbol{y}_t)$ is an input-output tuple from timestep t. Instead of optimizing $\mathcal{L}(\cdot)$ to completion, SERM stops prior to convergence, something known as early stopping [23]. As elucidated by [6], this can be seen as a form of nonparametric variational inference in the stationary case (see

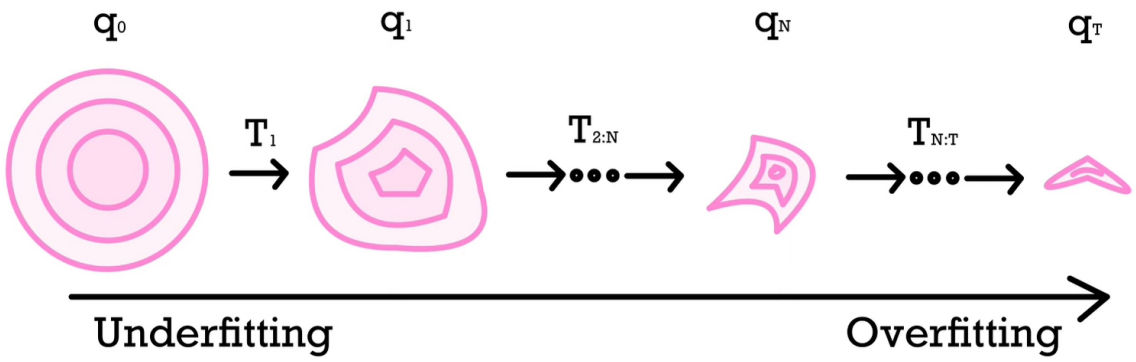


Figure 2.1: Diagrammatic representation of nonparametric variational inference as described in [6]. We start from some explicitly defined starting distribution q_0 and perform a series of transitions T, which generally represent an gradient-based optimization method. Since optimization is an entropy destroying process, the *implicitly* defined variational distributions $q_1, \ldots, q_T$ shrink over time. This creates a spectrum of distributions that range from underfitting to overfitting. See [6] for more.

Figure 2.1). In the non-stationary case, where we have a state-space that changes across time, we can view this same system as a nonparametric variational Kalman filter, where early stopping implicitly modulates the Kalman Gain (see Figure 2.2). This variational Kalman filter (VKF), is severely limited in several ways. First, we can only observe samples from the variational distributions rather than the variational distributions themselves. Second, we do not have access to an level of uncertainty quantification, which typically come baked into variational methods. Finally, the dynamics of SERM assume stationarity, which may not be appropriate in some cases.

Despite these limitations and its simplicity, SERM outperformed 13 different approaches explicitly designed for robustness and complex data generating distributions on benchmark datasets with temporal shift [33]. Since, at its core, SERM is an implicit Bayesian filter, we believe making it *explicit* will remove its limitations. In theory, an explicit Bayesian filter over neural network parameters would mean that we could add in dynamics and quantify our uncertainty while maintaining more generalizable representations. In reference to the two points listed earlier, this kind of method would accomplish both (1) and (2), creating a method that is highly desireable for

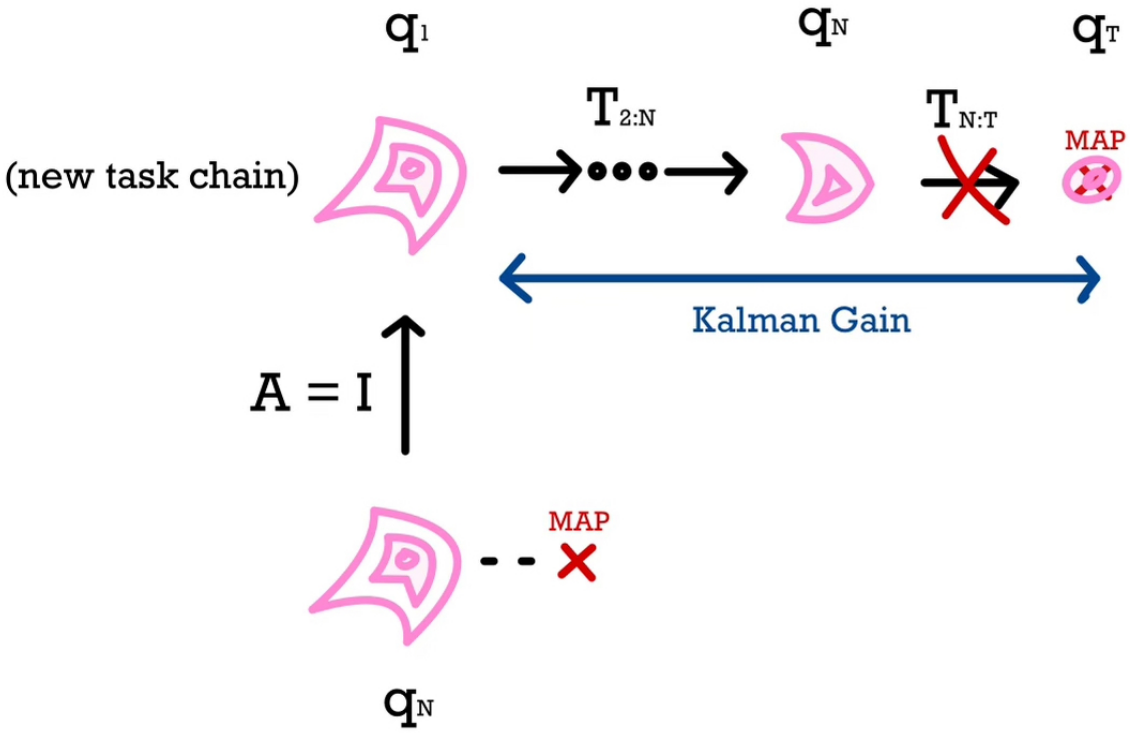


Figure 2.2: Diagrammatic representation of nonparametric variational Kalman filtering extended from [6]. We start from some implicitly defined distribution q_N, which was determined by optimizing over data from a given timestep with gradient descent plus early stopping. Since we stopped early, we are not optimized to the MAP solution exactly but we are close (within some arbitrarily defined metric space). To optimize over the next time step, we apply some dynamics $\boldsymbol{A}$ and start a new task chain. The MAP solution is now a different location within our metric space and we again wish to optimize with early stopping. We can view the spectrum of underfitting and overfitting in this time-varying context as modulating the Kalman gain.

machine learning practitioners.

In this section, we introduce two key ideas at a high-level: Bayesian filtering and Bayesian neural networks. In the section on Bayesian filtering, we present the special case of linear-Gaussian filtering known as the Kalman filter, and Variational Kalman filtering, which is the primary tool used in this thesis. The Bayesian neural networks section discusses Stochastic Gradient Langevin Dynamics (SGLD) and Stochastic Variational Inference (SVI), which should give the reader perspective on considerations made when applying Bayesian methods to neural networks. We have

purposely omitted a related works section since, to our knowledge, this is the only attempt at endowing neural networks with the ability to filter. For more motivation and context, we refer the reader to [33].

2.1 Bayesian Filtering

The purpose of Bayesian filtering is to estimate the marginal posterior distribution or filtering distribution of some state $\boldsymbol{\theta}_t$ at time step t given the history of measurements up to time step t. Suppose we maintain some state $\boldsymbol{\theta}_t \in \mathbb{R}^D \; \forall \; t \in \{1, \ldots, N\}$ and make observations $\boldsymbol{y}_t \in \mathbb{R}^K \; \forall \; t \in \{1, \ldots, N\}$. We assume

$$\boldsymbol{\theta}_t \sim p(\boldsymbol{\theta}_t | \boldsymbol{\theta}_{t-1}) \quad \text{and} \quad \boldsymbol{y}_t \sim p(\boldsymbol{y}_t | \boldsymbol{\theta}_t),$$

for $t \in \{1, \ldots, T\}$ where $p(\boldsymbol{\theta}_t | \boldsymbol{\theta}_{t-1})$ is the transition distribution, or dynamics model, and $p(\boldsymbol{y}_t | \boldsymbol{\theta}_t)$ is our measurement model. Here, we are assuming that the sequence of states form a first-order Markov chain and the measurements are conditionally independent. More formally, each state at time step t only depends on the previous state, such that

$$p(\boldsymbol{\theta}_t | \boldsymbol{\theta}_{1:T \setminus t}, \boldsymbol{y}_{1:T \setminus t}) = p(\boldsymbol{\theta}_t | \boldsymbol{\theta}_{t-1}), \tag{2.2}$$

and the current measurement only depends on the current state:

$$p(\boldsymbol{y}_t | \boldsymbol{\theta}_{1:T}, \boldsymbol{y}_{1:T \setminus t}) = p(\boldsymbol{y}_t | \boldsymbol{\theta}_t), \tag{2.3}$$

where $\boldsymbol{\theta}_{1:T} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T\}$, $\boldsymbol{y}_{1:T} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_T\}$, and $A \setminus B$ represents the set of elements that are in A but not in B. Given these two assumptions, we define the

joint prior distribution over states as

$$p(\boldsymbol{\theta}_{0:T}) = p(\boldsymbol{\theta}_0) \prod_{t=1}^{T} p(\boldsymbol{\theta}_t | \boldsymbol{\theta}_{t-1}), \tag{2.4}$$

where $p(\boldsymbol{\theta}_0)$ is some initial starting distribution and let the joint likelihood over measurements be

$$p(\boldsymbol{y}_{1:T} | \boldsymbol{\theta}_{0:T}) = \prod_{t=1}^{T} p(\boldsymbol{y}_t | \boldsymbol{\theta}_t). \tag{2.5}$$

As noted in [28], it is natural here to make an attempt at Bayes' theorem to yield our quantity of interest

$$p(\boldsymbol{\theta}_{0:T} | \boldsymbol{y}_{1:T}) = \frac{p(\boldsymbol{y}_{1:T} | \boldsymbol{\theta}_{0:T}) p(\boldsymbol{\theta}_{0:T})}{p(\boldsymbol{y}_{1:T})}, \tag{2.6}$$

but such a direct application of Bayes' theorem will immediately present difficulties. Namely, online applications will require a number of computations per time step that increase without bound as T increases. This necessitates a recursive solution that falls naturally from our two assumptions.

Consider the joint distribution between $\boldsymbol{\theta}_t$ and $\boldsymbol{\theta}_{t-1}$ given measurements up to but excluding time step t. We have

$$p(\boldsymbol{\theta}_t, \boldsymbol{\theta}_{t-1} | \boldsymbol{y}_{1:t-1}) = p(\boldsymbol{\theta}_t | \boldsymbol{\theta}_{t-1}) p(\boldsymbol{\theta}_{t-1} | \boldsymbol{y}_{t-1}), \tag{2.7}$$

due to the chain rule and our Markov assumption in equation 2.2. The marginal distribution of $\boldsymbol{\theta}_t$ given $\boldsymbol{y}_{1:t-1}$ can be obtained via integration

$$p(\boldsymbol{\theta}_t | \boldsymbol{y}_{1:t-1}) = \int p(\boldsymbol{\theta}_t | \boldsymbol{\theta}_{t-1}) p(\boldsymbol{\theta}_{t-1} | \boldsymbol{y}_{t-1}) d\boldsymbol{\theta}_{t-1}, \tag{2.8}$$

yielding the Chapman-Kolmogorov equation. We denote this as the *predictive distri-*

bution over possible states $\boldsymbol{\theta}_t$ at time step t. Alternatively, we can regard this as a recursive prior since we use this term to treat our incoming data sequentially. Since equation 2.3 assumes that our observations are independent of previous observations and states, a simple application of Bayes' rule yields our desired posterior over state $\boldsymbol{\theta}_t$

$$p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t}) = \frac{p(\boldsymbol{y}_t|\boldsymbol{\theta}_t)p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t-1})}{p(\boldsymbol{y}_t)}, \tag{2.9}$$

where

$$p(\boldsymbol{y}_t) = \int p(\boldsymbol{y}_t|\boldsymbol{\theta}_t)p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t-1})d\boldsymbol{\theta}_t. \tag{2.10}$$

The conditional independence of the measurements allows us to greatly simplify computation here. This application of Bayes' rule in this setting is often called the *update step*.

Taken together, equation 2.8 and equation 2.9 give us a simple formalism for inferring states over sequential observations. Knowledge of how our system transitions, $p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})$, allows us to update our previous posterior such that we receive a prior for the next time step, $p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t-1})$. If information is available at that time step, we can perform an update (equation 2.9) to incorporate that knowledge. When data in not available, we can rely on our predictive distribution to yield estimates of our state forward in time.

At this point, several things should be noted. First, a level of uncertainty quantification comes pre-baked into this filtering formalism as a virtue of Bayesian computation. Second, if all assumptions are linear and Gaussian, we receive all equations in closed-form, which we talk about next. Lastly, Bayesian filtering carries the conventional treatment machine learning treatment of data as a special case. As stated previously, the machine learning typically considers a collapsed state space where

every $\boldsymbol{\theta}_t = \boldsymbol{\theta}$. The quantity of interest is

$$p(\boldsymbol{\theta}|\boldsymbol{y}_{1:T}) \propto p(\boldsymbol{\theta})p(\boldsymbol{y}_{1:T}|\boldsymbol{\theta}) = p(\boldsymbol{\theta}) \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{\theta}), \tag{2.11}$$

which comes from the i.i.d. assumption. This assumption most notably assumes exchangability across data points, meaning that the data is not ordered. As such, this implies that the transition distribution $p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})$ is the identity function for every $t \in T$. Adding this in, we yield

$$p(\boldsymbol{\theta}_T|\boldsymbol{y}_{1:T}) \propto p(\boldsymbol{\theta}_0) \prod_{t=1}^{T} p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})p(\boldsymbol{y}_t|\boldsymbol{\theta}_t),$$

which is a Bayesian filter with stationary dynamics and zero process noise.

We will now treat two classes of Bayesian filters. First, the classical Kalman filter, which carries fully linear and Gaussian assumptions. Thereafter, we will treat the Variational Kalman filter, which will serve as our workhorse for filtering treatments of neural networks.

2.1.1 Kalman Filtering

The Kalman Filter [14] assumes linear and Gaussian measurement models and transition distributions such that

$$\boldsymbol{\theta}_t = \boldsymbol{A}_{t-1}\boldsymbol{\theta}_{t-1} + \boldsymbol{q}_{t-1}, \tag{2.12}$$

$$\boldsymbol{y}_t = \boldsymbol{X}_t\boldsymbol{\theta}_t + \boldsymbol{r}_t, \tag{2.13}$$

where $\boldsymbol{q}_{t-1} \sim \mathcal{N}(0, \boldsymbol{Q}_{t-1})$ is the process noise, $\boldsymbol{r}_t \sim \mathcal{N}(0, \boldsymbol{R}_t)$ is the measurement noise, $p(\boldsymbol{\theta}_0) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$, $\boldsymbol{A}_{t-1}$ denotes the dynamics matrix, and $\boldsymbol{X}_t$ is the measurement

model matrix. In probabilistic notation, equations 2.12 and 2.13 are equivalent to

$$p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}) = \mathcal{N}(\boldsymbol{A}_{t-1}\boldsymbol{\theta}_{t-1}, \boldsymbol{Q}_{t-1}), \tag{2.14}$$

$$p(\boldsymbol{y}_t|\boldsymbol{\theta}_t) = \mathcal{N}(\boldsymbol{X}_t\boldsymbol{\theta}_t, \boldsymbol{R}_t), \tag{2.15}$$

being exactly our transition distributions and measurement model from the previous section.

Note that we introduced two new quantities, $\boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_t$, which are the mean and covariance parameters of a Gaussian distribution over states, respectively. We can use these quantities to state the closed-form solutions for the prediction step and update step of a Kalman filter. The predict step is

$$\boldsymbol{\mu}_t^- = \boldsymbol{A}_{t-1}\boldsymbol{\mu}_{t-1}, \tag{2.16}$$

$$\boldsymbol{\Sigma}_t^- = \boldsymbol{A}_{t-1}\boldsymbol{\Sigma}_{t-1}\boldsymbol{A}_{t-1}^\top + \boldsymbol{Q}_{t-1}, \tag{2.17}$$

and the update step is

$$\boldsymbol{v}_t = \boldsymbol{y}_t - \boldsymbol{X}_t\boldsymbol{\mu}_t^-, \tag{2.18}$$

$$\boldsymbol{S}_t = \boldsymbol{X}_t\boldsymbol{\Sigma}_t^-\boldsymbol{X}_t^\top + \boldsymbol{R}_t, \tag{2.19}$$

$$\boldsymbol{K}_t = \boldsymbol{\Sigma}_t^-\boldsymbol{X}_t^\top\boldsymbol{S}_t^{-1}, \tag{2.20}$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_t^- + \boldsymbol{K}_t\boldsymbol{v}_t, \tag{2.21}$$

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_t^- - \boldsymbol{K}_t\boldsymbol{S}_t\boldsymbol{K}_t^\top, \tag{2.22}$$

where $\boldsymbol{K}_t$ is what is commonly referred to as the *Kalman Gain*. In probabilistic

notation, we have

$$p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t-1}) = \mathcal{N}(\boldsymbol{\theta}_t|\boldsymbol{\mu}_t^-, \boldsymbol{\Sigma}_t^-), \tag{2.23}$$

$$p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t}) = \mathcal{N}(\boldsymbol{\theta}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t), \tag{2.24}$$

which are the predictive distribution and filtering distribution. Please see [28] for more details and a full proof.

At this point, it is helpful to inject some intuition. At a high level, the predict step is using the dynamics matrix $\boldsymbol{A}_{t-1}$ to modify our understanding of the state given some domain knowledge. If this modification is trustworthy, we can reward it via a process noise matrix $\boldsymbol{Q}_{t-1} \to \mathbf{0}$. Otherwise, $\boldsymbol{Q}_{t-1}$ should be some value that accurately reflects our confidence over the dynamics of our system. The update step is comparing the estimates from the predict step with the quantity we would receive from MLE, and constraining that MLE estimate based on the confidence in our prediction.

Note that if $\boldsymbol{Q}_{t-1} = \mathbf{0}$ and $\boldsymbol{A}_{t-1} = \mathrm{I}$, we recover recursive linear regression and, thus, batchwise linear regression, due to the i.i.d. assumption. This is because every predict step is an identity function and the update step is simply the closed-form solution for MAP estimation. This offers a more concrete example of our dicussion in the previous section, being that Bayesian filtering is a general case of Bayesian inference over a stationary distribution. This also begs us to consider the outcome of $\boldsymbol{Q}_{t-1} \neq \mathbf{0}$ and $\boldsymbol{A}_{t-1} \neq \mathrm{I}$. As we will see in later sections, this allows us to inject a level of *inductive bias* as we move across time.

2.1.2 Variational Kalman Filtering

The variational Kalman filter (VKF) [1] is an alternate formulation of the traditional Kalman Filter that accommodates non-linear and non-Gaussian filtering dis-

tributions. Here, we assume the predictive distribution does not change under the assumptions of the VKF, so computing the predict step and predictive distribution fall from (2.23). We must only treat the filtering distribution (update step), which we will do now.

Let us begin by following the standard recipe of variational inference [4]. Assume that computing the true filtering distribution is intractable and we instead wish to optimize over some tractable family of densities $\mathcal{Q}$ such that we find some $q(\boldsymbol{\theta}_t) \in \mathcal{Q}$ that is closest in Kullback-Leibler (KL) divergence to the true filtering distribution $p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t})$,

$$\mathrm{KL}(q(\boldsymbol{\theta}_t)||p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t})) = \mathbb{E}_{q(\boldsymbol{\theta}_t)}\left[\log \frac{q(\boldsymbol{\theta}_t)}{p(\boldsymbol{y_t}|\boldsymbol{\theta}_t)p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t-1})}\right] + \log p(\boldsymbol{\theta}_t), \tag{2.25}$$

where the intractability comes from the marginal likelihood term, $\log p(\boldsymbol{\theta}_t)$. Due to the non-negativity of the KL divergence, we can instead maximize the Evidence Lower Bound (ELBO),

$$\mathcal{L}[q] = \mathbb{E}_{q(\boldsymbol{\theta}_t)}\left[\log p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t-1}))\right] + \mathbb{E}_{q(\boldsymbol{\theta}_t)}\left[\log p(\boldsymbol{y_t}|\boldsymbol{\theta}_t)\right] - \mathbb{E}_{q(\boldsymbol{\theta}_t)}\left[\log q(\boldsymbol{\theta}_t)\right], \tag{2.26}$$

which is equivalent to minimizing the KL divergence.

Let us now assume that $\mathcal{Q}$ is the family of Gaussian distributions. By the entropy of a Gaussian [13], we have

$$\mathcal{L}[q] = \mathbb{E}_{q(\boldsymbol{\theta}_t)}\left[\log p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t-1}))\right] + \mathbb{E}_{q(\boldsymbol{\theta}_t)}\left[\log p(\boldsymbol{y_t}|\boldsymbol{\theta}_t)\right] - \frac{D}{2}\left(1 + \log 2\pi\right) + \frac{1}{2}\log|\boldsymbol{\Sigma}|, \tag{2.27}$$

where D is the dimensionality of the Gaussian approximating family and $\boldsymbol{\Sigma}$, $\boldsymbol{\mu}$ are the variational parameters. If we take derivatives with respect to these parameters,

we receive

$$0 = \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[\log p(\boldsymbol{\theta}_t | \boldsymbol{y}_{1:t-1})) \right] + \nabla_{\boldsymbol{\mu}} \mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[\log p(\boldsymbol{y_t} | \boldsymbol{\theta}_t) \right], \tag{2.28}$$

$$\boldsymbol{\Sigma}^{-1} = -2 \left(\nabla_{\boldsymbol{\Sigma}} \mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[\log p(\boldsymbol{\theta}_t | \boldsymbol{y}_{1:t-1})) \right] + \nabla_{\boldsymbol{\Sigma}} \mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[\log p(\boldsymbol{y_t} | \boldsymbol{\theta}_t) \right] \right), \tag{2.29}$$

which, after applying the result shown in the Appendix of [22], we get

$$0 = \mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}_t | \boldsymbol{y}_{1:t-1})) \right] + \mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y_t} | \boldsymbol{\theta}_t) \right], \tag{2.30}$$

$$\boldsymbol{\Sigma}^{-1} = -\mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\boldsymbol{\theta}_t | \boldsymbol{y}_{1:t-1})) \right] - \mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\boldsymbol{y_t} | \boldsymbol{\theta}_t) \right]. \tag{2.31}$$

If we substitute the known form for $p(\boldsymbol{\theta}_t | \boldsymbol{y}_{1:t-1})$ based on $\boldsymbol{\mu}^-$ and $\boldsymbol{\Sigma}^-$ from the previous section, we recieve

$$0 = \mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[(\boldsymbol{\Sigma}^-)^{-1} (\boldsymbol{\theta}_t - \boldsymbol{\mu}^-) \right] + \mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y_t} | \boldsymbol{\theta}_t) \right], \tag{2.32}$$

$$\boldsymbol{\Sigma}^{-1} = -\mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[(\boldsymbol{\Sigma}^-)^{-1} \right] - \mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\boldsymbol{y_t} | \boldsymbol{\theta}_t) \right]. \tag{2.33}$$

This is the jumping off point for constructing different VKFs. By referencing the basic foundation established here, we can begin to think about different VKF variations and their associated assumptions. Namely, in the conjugate gradient variational Kalman filter (CG-VKF), we take unbiased estimators of the expectations. For natural gradient variational Kalman filtering (NG-VKF), we derive a similar result as just shown, but take advantage of information geometry as we optimize the variational parameters. As such, we can imagine that the CG-VKF makes low-rank covariance matrices much more accessible but the underlying predication on the Laplace approximation might make the uncertainty quantification poor for neural networks. NG-VKFs, due to their truly variational form, should be more reliable. We will discuss this further in Chapter 4.

2.2 Bayesian Neural Networks

Neural networks are extremely powerful and flexible function approximators. Most generally, a neural network consists of a set of nodes, or neurons, organized into layers. Each node takes in one or more inputs, performs a linear computation on those inputs, and then applies a nonlinearity to produce an output.

Formally, this can be written as

$$y_j = f\left(\sum_{i=1}^{n} w_{ij} x_i + b_j\right), \tag{2.34}$$

where y_j is the output of node j, x_i is the i^{th} input to neuron j, w_{ij} is the weight of the connection between node i and node j, b_j is the bias of node j, and $f(\cdot)$ denotes a nonlinearity, commonly referred to as an activation function.

Common activation functions include the sigmoid function, Rectified Linear Unit (ReLU), and Hyperbolic tangent (tanh). While neural networks generally embody equation 2.34, many variations exist including feedforward networks, convolutional neural networks (CNNs), recurrent neural networks (RNNs), and more. Rather than treating these variations explicitly, we will focus on imbuing these general subclasses of neural networks with Bayesian principles. For a comprehensive overview of neural networks, we recommend [11].

The goal of Bayesian neural networks (BNNs) is to infer the posterior distribution over the weights and biases given some data. The emphasis is on *inference*, rather than neural network *architecture*. Bayesian inference for neural networks comes in primarily two flavors: variational inference (VI) and Markov Chain Monte Carlo (MCMC). MCMC is a class of methods for generating samples from the posterior distribution over the weights and biases in a BNN while VI approximates the intractable posterior distribution by optimizing over a more tractable surrogate distribution instead. Here, we describe two important algorithms for BNNs: Stochastic Gradient Langevin Dy-

namics (SGLD) and Stochastic Variational Inference (SVI). This will put into context some of the considerations needed for applying Bayesian methods to neural networks, and the options we have.

2.2.1 Stochastic Gradient Langevin Dynamics

An exact Bayesian treatments tells us to compute posteriors as follows:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}'} p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}'}. \tag{2.35}$$

For neural networks, $p(\mathcal{D})$ is problematic because it does not yield a closed-form solution. MCMC considers approximations to $p(\mathcal{D})$ via

$$\int_{\boldsymbol{\theta}'} p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}' = \mathbb{E}_{p(\boldsymbol{\theta})}[p(\mathcal{D}|\boldsymbol{\theta})] \approx \frac{1}{N}\sum_{i=1}^{N} p(\mathcal{D}|\boldsymbol{\theta}_i), \tag{2.36}$$

where every $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta})$.

This is the starting point for Stochastic Gradient Langevin Dynamics (SGLD). The goal of SGLD is to modify stochastic gradient descent (SGD) such that unconverged gradient steps resemble standard SGD and the converged gradient steps behave like Metropolis-Hastings (MH). This is done via a small amount of noise injected into every gradient step. If we anneal the learning rate over time, we recover a nice property in that we can completely skip the MH accept-reject step since the number of rejections go asymptotically to zero as the learning rate decreases. We will now describe the algorithm in full, following the formalisms from the original paper [32].

Let θ denote a parameter vector for any differentiable probabilistic model, with $p(\theta)$ denoting the prior distribution and $p(x|\theta)$ the likelihood of the data x, given the parameterization θ. At each iteration t, for a subset of $n \leq N$ data items, we are

going to define Stochastic Gradient Descent (SGD) as the following update:

$$\Delta\theta_t = \frac{\epsilon_t}{2}\left(\nabla \log p\left(\theta_t\right) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log\left(x_{ti}|\theta_t\right)\right), \tag{2.37}$$

where ϵ_t is a sequence of step sizes. SGLD performs an identical gradient update, with the addition of a Gaussian noise term:

$$\Delta\theta_t = \frac{\epsilon_t}{2}\left(\nabla \log p\left(\theta_t\right) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log\left(x_{ti}|\theta_t\right)\right) + \eta_t, \tag{2.38}$$

$$\eta_t \sim \mathcal{N}\left(0, \epsilon_t\right). \tag{2.39}$$

To combat the variation in geometry across gradients at every time step, t, we can introduce a preconditioning matrix M_t:

$$\Delta\theta_t = \frac{\epsilon_t}{2} M_t \left(\nabla \log p\left(\theta_t\right) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log\left(x_{ti}|\theta_t\right)\right) + \eta_t, \tag{2.40}$$

$$\eta_t \sim \mathcal{N}\left(0, \epsilon_t M_t\right), \tag{2.41}$$

which can significantly improve convergence. Whether this algorithm is in the stochastic optimization phase or Langevin dynamics phase depends on the variance of the injected noise, $\epsilon_t M_t$, versus that of the stochastic gradient. We can estimate a covariance matrix for the stochastic gradients using the empirical covariance:

$$V\left(\theta_t\right) \approx \frac{N^2}{n^2}\sum_{i=1}^{n}\left(s_{ti} - s_t^*\right)\left(s_{ti} - s_t^*\right)^\top, \tag{2.42}$$

$$s_{ti} = \frac{1}{N}\nabla \log p\left(\theta_t\right) + \nabla \log\left(x_{ti}|\theta_t\right), \tag{2.43}$$

$$s_t^* = \frac{1}{n}\sum_{i=1}^{n} s_{ti}, \tag{2.44}$$

The variance of the stochastic gradient step is approximately $\frac{\epsilon_t^2}{4} M V\left(\theta_t\right) M$. To

get the injected noise to dominate in all directions, and ensure that we are sampling from the approximate posterior, we need the condition

$$\frac{\epsilon_t^2 N^2}{4n}\lambda_{\max}\left(M^{1/2}V_s M^{1/2}\right) = \alpha \ll 1, \tag{2.45}$$

where $\lambda_{\max}(A)$ is the largest eigenvalue of A and $V_s = \frac{n}{N^2}V(\theta_t)$.

It should be noted that convergence guarantees, and other technical assumptions, are contingent on the satisfaction of the properties

$$\sum_{t=1}^{\infty}\epsilon_t = \infty \qquad \sum_{t=1}^{\infty}\epsilon_t^2 < \infty \tag{2.46}$$

which are broken if the learning rate is held constant. However, in practice, the learning rate is typically held constant with this gradient descent procedure. There is some great literature on how to set up the preconditioning matrix [20] and theoretical properties of SGLD [5, 24, 30]. We refer the reader to these sources for further discussion. This algorithm likely represents the simplest way to construct a Bayesian neural network but it is extremely slow to converge.

2.2.2 Stochastic Variational Inference

Let $q_{\boldsymbol{\mu},\sigma}(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \sigma^2\boldsymbol{I})$ be a multivariate Gaussian that is our approximating distribution over the true posterior, $p(\boldsymbol{\theta}|\mathcal{D})$. In stochastic variational inference, our goal is to minimize the difference between $p(\boldsymbol{\theta}|\mathcal{D})$ and $q_{\boldsymbol{\mu},\sigma}(\boldsymbol{\theta})$ by optimizing the variational parameters $\phi = \{\boldsymbol{\mu}, \sigma\}$. Traditionally, this is done by maximizing the evidence lower bound (ELBO):

$$\ln p(\mathcal{D}) \geq \mathbb{E}_{q_\phi(\boldsymbol{\theta})}[\ln p(\boldsymbol{\theta}, \mathcal{D})] - \mathbb{E}_{q_\phi(\boldsymbol{\theta})}[\ln q_\phi(\boldsymbol{\theta})] = \mathcal{L}[q], \tag{2.47}$$

where $\ln q_{\boldsymbol{\mu},\sigma}(\boldsymbol{\theta}) = -\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\mu})^\top(\sigma^2\boldsymbol{I})^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu})$ in the mean-field, isotropic Gaussian case we are assuming here.

SVI handles the intractability of large datasets by optimizing the ELBO over stochastic subsets $\boldsymbol{X}, \boldsymbol{y} \sim \mathcal{D}$. We can be explicit, and restate the ELBO as

$$\mathcal{L}[q] = \mathbb{E}_{\boldsymbol{X},\boldsymbol{y}\sim\mathcal{D}}[\mathbb{E}_{q_{\mu,\sigma}(\boldsymbol{\theta})}[\ln p(\boldsymbol{\theta},\mathcal{D})] - \mathbb{E}_{q_{\boldsymbol{\mu},\sigma}(\boldsymbol{\theta})}[\ln q_{\boldsymbol{\mu},\sigma}(\boldsymbol{\theta})]], \tag{2.48}$$

where $\mathbb{E}_{\boldsymbol{X},\boldsymbol{y}\sim\mathcal{D}}[\cdot]$ is approximated by an average over mini-batches.

To optimize over the variational parameters, we must compute $\nabla_{\boldsymbol{\mu}}\mathcal{L}[q]$ and $\nabla_{\sigma}\mathcal{L}[q]$. This yields the following:

$$\nabla_{\boldsymbol{\mu},\sigma}\mathcal{L}[q] = \nabla_{\boldsymbol{\mu},\sigma}\mathbb{E}_{\boldsymbol{X},\boldsymbol{y}\sim\mathcal{D}}[\mathbb{E}_{q_{\mu,\sigma}(\boldsymbol{\theta})}[\ln p(\boldsymbol{\theta},\mathcal{D})] - \mathbb{E}_{q_{\boldsymbol{\mu},\sigma}(\boldsymbol{\theta})}[\ln q_{\boldsymbol{\mu},\sigma}(\boldsymbol{\theta})]], \tag{2.49}$$

which requires an average over mini-batches, as previously stated, and the reparameterization trick. Let us expand the joint, combine our two variational expectation terms, and insert an average over mini-batches:

$$\nabla_{\boldsymbol{\mu},\sigma}\mathcal{L}[q] \approx \frac{1}{N}\sum_{i=1}^{N}[\nabla_{\boldsymbol{\mu},\sigma}\mathbb{E}_{q_{\mu,\sigma}(\boldsymbol{\theta})}[\ln p(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)=\boldsymbol{y}_i|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta}) - \ln q_{\boldsymbol{\mu},\sigma}(\boldsymbol{\theta})]]. \tag{2.50}$$

Now, let us introduce the following reparameterzation: $\boldsymbol{\theta} = \boldsymbol{\mu} + \sigma\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. This allows us to bring the derivative inside the expectation and then approximate the derivative using Monte Carlo (MC) samples

$$\nabla_{\boldsymbol{\mu},\sigma}\mathcal{L}[q] \approx \frac{1}{N\cdot M}\sum_{i=1}^{N}\sum_{j=1}^{M}\nabla_{\boldsymbol{\mu},\sigma}[\ln p(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)=\boldsymbol{y}_i|\boldsymbol{\theta}(\boldsymbol{\mu},\sigma,\boldsymbol{\epsilon}_j)) + \ln p(\boldsymbol{\theta}(\boldsymbol{\mu},\sigma,\boldsymbol{\epsilon}_j))], \tag{2.51}$$

where M is the number of Monte-Carlo samples. This completes the simple derivation for mean-field SVI over an isotropic Gaussian family.

At the expense of increased computation and a more involved derivation, we can induce more structure into this approximation. In practice, mean-field SVI gives poor performance so more structured approaches are required. We only offered a brief introduction here, and we refer the reader to [4] for a more comprehensive tutorial.

Chapter 3

Meta-Filtering in the Linear-Gaussian Case

Before tackling the complex high-dimensional parameter spaces of neural networks, one might ask: why bother with this in the first place? To answer this, we constructed a *meta-filter* in the linear-Gaussian case and assessed its impact on modeling a simple non-stationary distribution over linear regression tasks. We use the term, *meta-filter*, for two reasons: (1) the simple model tested here is a non-stationary version of MAML based on hierarchical Bayesian linear regression [12, 8], and (2) the previous state of a Bayesian filter can be seen as an inductive bias for the current state estimation, which is in the spirit of meta-learning.

This chapter treats this simple Linear-Gaussian meta-filter fully, and also describes the procedure used for dynamics and process noise estimation. At the end, we show some results on a drifting linear regression task that clearly demonstrates how dynamics improve estimation across time, if there are meaningful dynamics to be learned. The aim of this chapter is to motivate Chapter 4, where we attempt to expand this simple case, and show the benefits of such an approach.

3.1 Time-varying hierarchical Bayesian linear regression

Let $\boldsymbol{\theta} \in \mathbb{R}^D$ be the parameters of a shared prior distribution over the task-specific parameters $\boldsymbol{\phi}_t \in \mathbb{R}^D$ across task indices $t \in \{0, \ldots, T\}$. For the sake of simplicity, we consider there to be N observed examples for each task. Let $\boldsymbol{X}_t \in \mathbb{R}^{N \times D}$ be the input features for the observed examples in task t. Let $\mathbf{y}_t \in \mathbb{R}^M$ be the observations for task t.

For any task t, the task-specific likelihood is

$$p\left(\mathbf{y}_t \,|\boldsymbol{X}_t, \boldsymbol{\theta}\right) = \int p\left(\mathbf{y}_t \,|\boldsymbol{X}_t, \boldsymbol{\phi}_t\right) p\left(\boldsymbol{\phi}_t \,|\boldsymbol{\theta}\right) d\boldsymbol{\phi}_t. \tag{3.1}$$

Assuming the tasks are i.i.d, the full likelihood over all tasks is

$$p\left(\mathbf{y}_{0:T} \,|\boldsymbol{X}_{0:T}, \boldsymbol{\theta}\right) = \prod_t \int p\left(\mathbf{y}_t \,|\boldsymbol{X}_t, \boldsymbol{\phi}_t\right) p\left(\boldsymbol{\phi}_t \,|\boldsymbol{\theta}\right) d\boldsymbol{\phi}_t. \tag{3.2}$$

By Bayes' rule, the posterior distribution can be computed as follows:

$$p\left(\boldsymbol{\theta} \,|\mathbf{y}_{0:T}, \boldsymbol{X}_{0:T}\right) \propto p\left(\boldsymbol{\theta}\right) p\left(\mathbf{y}_{0:T} \,|\boldsymbol{X}_{0:T}, \boldsymbol{\theta}\right) = p\left(\boldsymbol{\theta}\right) \prod_t p\left(\mathbf{y}_t \,|\boldsymbol{X}_t, \boldsymbol{\theta}\right). \tag{3.3}$$

By assuming that all quantities are linear and Gaussian, we get

$$p\left(\boldsymbol{\phi}_t \,|\boldsymbol{\theta}\right) = \mathcal{N}\left(\boldsymbol{\phi}_t \,|\boldsymbol{\theta}, \mathbf{W}_t\right), \tag{3.4}$$

and

$$p\left(\mathbf{y}_t \,|\boldsymbol{X}_t, \boldsymbol{\phi}_t\right) = \mathcal{N}\left(\mathbf{y}_t \,|\boldsymbol{X}_t \boldsymbol{\phi}_{\boldsymbol{t}}, \mathbf{V}_t\right), \tag{3.5}$$

for the definitions of $\mathbf{W}_t, \mathbf{V}_t$ that we will define shortly. By ([3], eq. B.44), we can

compute in closed-form:

$$p\left(\mathbf{y}_t \left| \boldsymbol{X}_t, \boldsymbol{\theta} \right.\right) = \mathcal{N}\left(\boldsymbol{y}_t \left| \boldsymbol{X}_t\boldsymbol{\theta}, \boldsymbol{X}_t\mathbf{W}_t\boldsymbol{X}_t^\top + \boldsymbol{V}_t \right.\right), \tag{3.6}$$

which we can use to simplify our application of Bayes' Theorem from before:

$$p\left(\boldsymbol{\theta} \left| \mathbf{y}_{0:T}, \boldsymbol{X}_{0:T} \right.\right) \propto \mathcal{N}\left(\boldsymbol{\theta}_0 | \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0\right) \prod_t \mathcal{N}\left(\boldsymbol{y}_t \left| \boldsymbol{X}_t\boldsymbol{\theta}_t, \boldsymbol{X}_t\mathbf{W}_t\boldsymbol{X}_t^\top + \boldsymbol{V}_t \right.\right). \tag{3.7}$$

We know that in this linear regression case, the task-specific parameters $\boldsymbol{\phi_j}$ have a MAP estimate that corresponds to the the global mode of the posterior $p\left(\boldsymbol{\phi_j} \left| \mathbf{y}_t, \boldsymbol{X}_t, \boldsymbol{\theta} \right.\right) \propto p\left(\mathbf{y}_t \left| \boldsymbol{X}_t, \boldsymbol{\phi}_t \right.\right) p\left(\boldsymbol{\phi}_t \left| \boldsymbol{\theta} \right.\right)$. We know that early stopping of an iterative gradient descent procedure to estimate $\boldsymbol{\phi_j}$ is exactly equivalent to MAP estimate, if the prior encodes both the number of gradient steps and the direction of each step. For a single gradient step, we have

$$\boldsymbol{\phi_t} = \boldsymbol{\theta} - \eta \nabla_\theta \left[\left\| \boldsymbol{y}_t - \boldsymbol{X}_t\boldsymbol{\theta} \right\|^2_{\boldsymbol{V}_t} \right], \tag{3.8}$$

$$= \boldsymbol{\theta} - \eta \boldsymbol{X}_t \boldsymbol{V}_t^{-1} \left(\boldsymbol{X}_t\boldsymbol{\theta} - \boldsymbol{y}_t \right), \tag{3.9}$$

which is equivalent to the regularized linear least squares problem

$$\min \left(\left\| \boldsymbol{y}_t - \boldsymbol{X}_t\boldsymbol{\theta} \right\|^2_{\boldsymbol{V}_t} + \left\| \boldsymbol{\theta} - \boldsymbol{\phi_t} \right\|^2_{\boldsymbol{W}_t} \right). \tag{3.10}$$

This minimization corresponds to the maximization of

$$p\left(\boldsymbol{\phi_j} \left| \mathbf{y}_t, \boldsymbol{X}_t, \boldsymbol{\theta} \right.\right) \propto \mathcal{N}\left(\mathbf{y}_t \left| \boldsymbol{X}_t\boldsymbol{\phi_t}, \mathbf{V}_t \right.\right) \mathcal{N}\left(\boldsymbol{\phi}_t \left| \boldsymbol{\theta}, \mathbf{W}_t \right.\right). \tag{3.11}$$

If we assume $\mathbf{V}_t = \mathrm{I}$ and $\boldsymbol{M} = \mathrm{diag}(\eta)$, Theorem 3.2 of [27] states that there exists a

nonsingular matrix $\boldsymbol{Y}$ such that

$$\boldsymbol{Y}^\top \boldsymbol{M}^{-1} \boldsymbol{Y} = \text{diag}(\frac{1}{\eta_1}, \ldots, \frac{1}{\eta_n}), \tag{3.12}$$

and

$$\boldsymbol{Y}^\top (\boldsymbol{X}_t^\top \boldsymbol{V}_t^{-1} \boldsymbol{X}_t) \boldsymbol{Y} = \boldsymbol{Y}^\top (\boldsymbol{X}_t^\top \boldsymbol{X}_t) \boldsymbol{Y} = \text{diag}(a_1, \ldots, a_n), \tag{3.13}$$

with $a_i \geq 0$ and $\eta_i > 0 \; \forall \; i \in \{1, \ldots, n\}$. We are interested in the quantity $\boldsymbol{W}_t$, defined as

$$\boldsymbol{W}_t = \boldsymbol{Y}^\top \text{diag}(\mu_i) \boldsymbol{Y}, \tag{3.14}$$

where

$$\mu_i = \begin{cases} \frac{1}{a_i} \left((1 - a_i \eta_i)^{-1} - 1 \right), & \text{if } a_i \neq 0 \\ 1, & \text{otherwise.} \end{cases} \tag{3.15}$$

which we can be computed easily in closed-form.

The derivation stated above specifies hierarchical Bayesian linear regression in the stationary case, in a form equivalent to MAML as shown by [12]. Following Section 2.1, casting hierarchical Bayesian linear regression as a Bayesian filter is extremely simple.

Let the measurement noise $\boldsymbol{R}_t = \boldsymbol{X}_t \mathbf{W}_t \boldsymbol{X}_t^\top + \boldsymbol{V}_t$ and the observation model be the input data $\boldsymbol{X}_t$. To break the stationarity assumption, we are going to drop the assumption of a collapsed state space ($\boldsymbol{\theta}_t = \boldsymbol{\theta} \; \forall \; t \in \{0, \ldots, T\}$) and introduce the transition distribution $p(\boldsymbol{\theta}_t | \boldsymbol{\theta}_{t-1})$. Since all assumptions are linear-Gaussian, this

transition distribution is defined as

$$p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}) = \mathcal{N}(\boldsymbol{\theta}_t|\boldsymbol{A}_{t-1}\boldsymbol{\theta}_{t-1}, \boldsymbol{Q}_{t-1}), \tag{3.16}$$

with the same definitions of $\boldsymbol{A}_{t-1}$ and $\boldsymbol{Q}_{t-1}$ as Section 2.1. To further simplify the system, we will assume the dynamics are time-invariant, that is $\boldsymbol{A}_{t-1} = \boldsymbol{A}$ and $\boldsymbol{Q}_{t-1} = \boldsymbol{Q}$. This yields a new posterior distribution for time step t that can be written as

$$p\left(\boldsymbol{\theta}_t \left|\mathbf{y}_{0:t}, \boldsymbol{X}_{0:t}\right.\right) \propto \mathcal{N}\left(\boldsymbol{\theta}_0|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0\right) \prod_t \mathcal{N}(\boldsymbol{\theta}_t|\boldsymbol{A}\boldsymbol{\theta}_{t-1}, \boldsymbol{Q}) \mathcal{N}\left(\boldsymbol{y}_t \left|\boldsymbol{X}_t\boldsymbol{\theta}_t, \boldsymbol{R}_t\right.\right), \tag{3.17}$$

where $\boldsymbol{A}$ and $\boldsymbol{Q}$ are typically unknown a priori. We treat their estimation in the next section.

As one final note, MAML can be seen as a special case of the above model with three primary modifications:

1. MAML learns a point estimate of $\boldsymbol{\theta}$ by type-II maximum likelihood (empirical Bayes) and does not define a prior distribution $p(\boldsymbol{\theta})$.

2. MAML approximates the task likelihood $p(\boldsymbol{y}_t|\boldsymbol{X}_t, \boldsymbol{\theta}) = \int p(\boldsymbol{y}_t|\boldsymbol{X}_t, \boldsymbol{\phi}_t) p(\boldsymbol{\phi}_t|\boldsymbol{\theta})\, d\boldsymbol{\theta}$ with a point estimate $\hat{\boldsymbol{\phi}}_{\boldsymbol{t}} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\phi}} \mathcal{L}_t(\boldsymbol{\phi}; \boldsymbol{y}_t, \boldsymbol{X}_t)|_{\boldsymbol{\phi}=\boldsymbol{\theta}}$.

3. MAML does not accommodate any non-stationarity via a transition distribution $p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1})$. MAML assumes stationarity over the task distribution and all computations are batchwise.

We can also generalize MAML under this Bayesian filtering formalism in the nonlinear case by using the same arguments shown in [12].

3.2 Dynamics and process noise estimation

Let $\Theta = \{\boldsymbol{A}, \boldsymbol{Q}\}$ be a parameter set consisting of the dynamics $\boldsymbol{A} \in \mathbb{R}^{D \times D}$ and process noise $\boldsymbol{Q} \in \mathbb{R}^{D \times D}$. We can pose the dynamics and process noise estimation problem as the following:

$$\Theta^* = \arg\max_{\Theta} \sum_{t=1}^{T} \log p(\boldsymbol{y}_t | \boldsymbol{y}_{1:t-1}), \tag{3.18}$$

where, by Gaussian assumptions,

$$p(\boldsymbol{y}_t | \boldsymbol{y}_{1:t-1}) = \mathcal{N}(\boldsymbol{y}_t | \boldsymbol{X}_t \boldsymbol{\mu}_t^-, \boldsymbol{X}_t \boldsymbol{\Sigma}_t^- \boldsymbol{X}_t^\top + \boldsymbol{R}_t), \tag{3.19}$$

with $\boldsymbol{\mu}_t^-$ and $\boldsymbol{\Sigma}_t^-$ denoting the mean and covariance estimations of the predict step, respectively. Recall the definitions of $\boldsymbol{\mu}_t^-$ and $\boldsymbol{\Sigma}_t^-$ from Section 2.1:

$$\boldsymbol{\mu}_t^- = \boldsymbol{A} \boldsymbol{\mu}_{t-1}, \tag{3.20}$$

$$\boldsymbol{\Sigma}_t^- = \boldsymbol{A} \boldsymbol{\Sigma}_{t-1} \boldsymbol{A}^\top + \boldsymbol{Q}. \tag{3.21}$$

Since the Gaussian distribution is differentiable, we can optimize for $\boldsymbol{A}$ and $\boldsymbol{Q}$ via any gradient-based method.

In the following section, we optimized $\boldsymbol{A}$ and $\boldsymbol{Q}$ using standard gradient descent with a learning rate $\epsilon = 1e-3$. For 5 iterations, we ran our filtering setup forward and computed gradients with the log-likelihood specified above. We initialized $\boldsymbol{A}$ and $\boldsymbol{Q}$ as $\boldsymbol{A} = \mathrm{I}$ and $\boldsymbol{Q} = \mathrm{I}$. Any attempts to perform gradient descent for more than 5 iterations decreased performance. This is due to extremely noisy gradients over the dynamics and process noise matrices, which make their estimation a challenge.

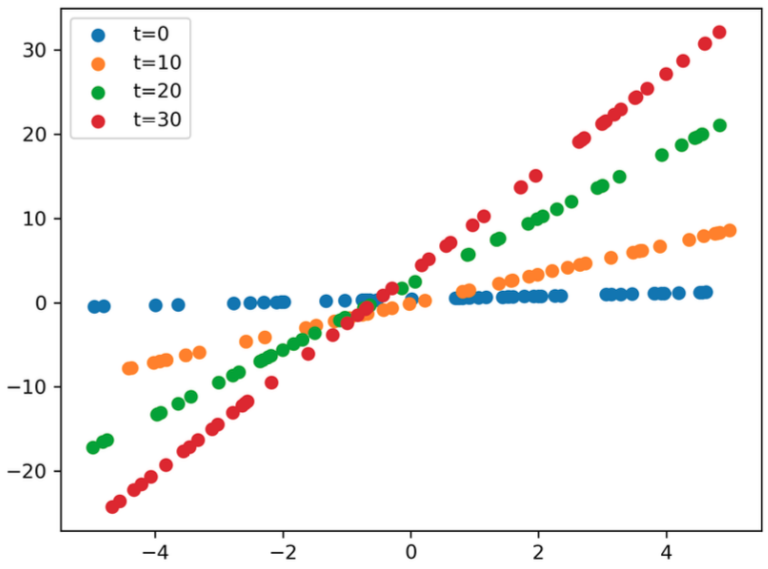


Figure 3.1: Drifting line dataset for 30 timesteps with no per datum noise. See generative process outlined in main text for more details.

3.3 Results

Drifting linear regression as shown in Figure 3.1 was the primary evaluation dataset for this simple linear-Gaussian meta-filter. Our goal was to create a synthetic setting where the data generating distribution shifted slightly over time, and ground truth dynamics and process noise were available. There are only two parameters to estimate in this simple model but estimation is hardly nontrivial. Sources of noise include a per datum perturbation that was applied to each data point at each time step and parameter perturbation that made the dynamics stochastic. Fitting lines using maximum likelihood estimation (MLE) when data is available is trivial, but using previous timesteps to fit lines without any data is extremely nontrivial. In these experiments, we use the first 30 timesteps as an in-distribution (ID) training set and the next 10 timesteps as an out-of-distribution (OOD) test set. We define the generative process for the data as follows.

Let $f(x_t) = \theta_1 x_t + \theta_0$ be the true underlying line at time step t with true parameters

$\boldsymbol{\theta} = [\theta_1, \theta_0]^\top$. At time step 0, let $\theta_1 = 0$ and $\theta_0 = 0$ and iterate the following for T time steps:

1. Sample $\theta_1^* \sim \mathcal{N}(\theta_1, 1)$ and $\theta_0^* \sim \mathcal{N}(\theta_0, 1)$
2. Sample $\boldsymbol{x} \sim \mathcal{U}[-5, 5]$ where $\boldsymbol{x} = \{x_1, \ldots, x_{50}\}$.
3. Let $\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 0.1)\ \forall\ x_t \in \boldsymbol{x}$.
4. Increase θ_1 by 1 and repeat step 1.

Three different models were evalutated on the data just described: (1) a Gaussian random walk (GRW) where $\boldsymbol{A}$ and $\boldsymbol{Q}$ are both the identity matrix, (2) empirical risk minimization (ERM) which is available in closed-form via the maximum likelihood estimator, and (3) a trained filter where $\boldsymbol{A}$ and $\boldsymbol{Q}$ were learned using the 30 ID timesteps and the estimation method described in Section 3.2. Both (1) and (2) reflect stationary dynamics whereas (3) attempted to learn the non-stationary generative process just described. Figure 3.2 shows these results.

We found that the trained filter significantly outperformed both the GRW model and ERM when tested on the 10 OOD timesteps. On ID data, the trained filter also outperformed both methods. The GRW model offered a slight performance boost, compared to ERM, on both ID and OOD timesteps. This is due to overfitting from the maximum likelihood estimator that was smoothed out by the Gaussian random walk. Interestingly, it seems that the GRW offered a slightly better inductive bias when moving to OOD timesteps. We believe that this is because the GRW model avoids fitting the noise on the timestep directly prior to starting OOD estimation. Of course, by knowing the dynamics and the true process noise, the trained filter was able to avoid fitting this time-varying noise even more.

Taken together, we see this application as a promising start for treating the problem of temporal shift. Since estimating the dynamics and the process noise is much

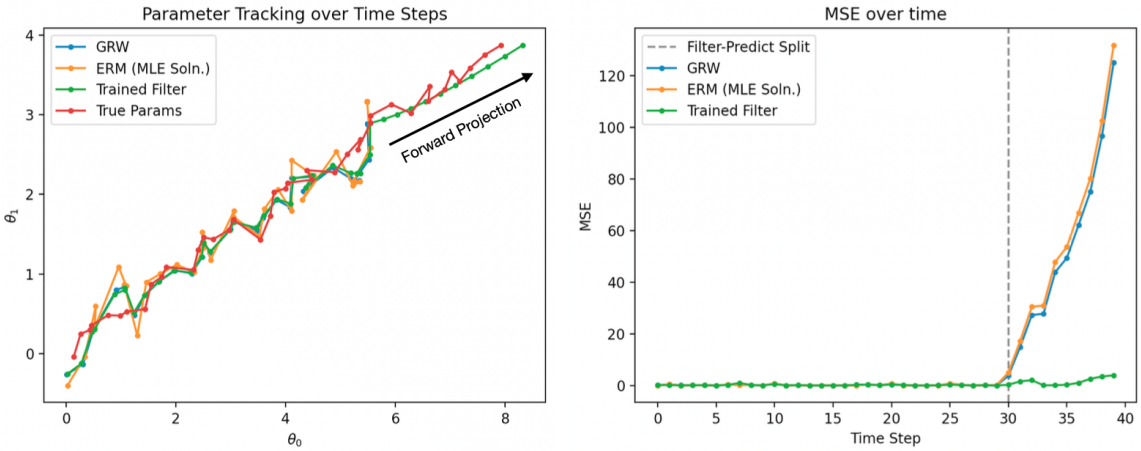


Figure 3.2: (Left) Parameter tracking over all 40 timessteps. The first 30 times steps were used for training and are considered in-distribution. The next 10 timesteps, labeled with forward projection, are considered out-of-distribution and no data was used to fit those lines. The Gaussian Random Walk (GRW) and Empirical Risk Minimizer (ERM) assumed stationarity so the parameters stayed static in time while the true parameters and trained filter moved forward. (Right) Mean Squared Error over 40 timesteps. GRW and ERM show terrible generalization while the trained filter significantly reduces the accumulated error over time despite the lack of data.

more difficult in neural networks, we report results using a Gaussian random walk as shown here. The hope was to provide a slight performance boost compared to ERM, as was done here.

Chapter 4

Meta-Filtering in the nonlinear case

In the previous chapter, we showed success in a simple two parameter case with drifting linear regression. In this chapter, we explore the much higher dimensional case that arises when working with neural networks. We should preface with the fact that we had very limited success in this domain. Bayesian methods can be extremely challenging for neural networks, and dynamics estimation in this setting still remains unsolved.

We use the first section of this chapter to informally discuss the challenges of applying Bayesian methods to neural networks. The following two sections will dive into two methods of approximate inference that we derived for variational Kalman filtering. In the final section, we conclude with preliminary results. Our goal here was to outline a few things researchers looking at these kinds of problems should think about and make our derivations publicly available.

4.1 A few considerations

Neural networks are notoriously high-dimensional, with parameter counts that can go into the billions depending on the domain. This necessitates low-rank covariance matrices or diagonal covariance matrices since high-dimensional matrices are often impossible to store, let alone compute or invert.

A second consideration is that of neural network geometry. The non-convexity of neural networks make them prone to Hessian matrices that are not positive semi-definite (PSD), due to negative eigenvalues. Since this is a challenge to covariance estimation, the Gauss-Newton Hessian $\boldsymbol{G} \approx \nabla^2_{\boldsymbol{\theta\theta}} \mathcal{L}(\boldsymbol{\theta})$ should be used since it is guarentted to be PSD.

The third and final consideration is the use of matrix-vector products (MVPs). We are often required to instantiate large matrices in second-order optimization. MVPs make the use of these second-order methods feasible. Methods like conjugate gradients naturally lend themselves to MVPs, so they should be considered when designing methods.

4.2 Conjugate gradient variational Kalman filtering

Recall from Section 2.1 that our assumption for the variational Kalman filter (VKF) is that the predict step remains the same as the linear-Gaussian case and only the update step changes. With that in mind, we will derive the VKF update step via the following assumption:

$$0 = \mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[(\boldsymbol{\Sigma}^-)^{-1} (\boldsymbol{\theta}_t - \boldsymbol{\mu}^-) \right] + \mathbb{E}_{q(\boldsymbol{\theta}_t)} \left[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y_t} | \boldsymbol{\theta}_t) \right] \tag{4.1}$$

$$\approx (\boldsymbol{\Sigma}^-)^{-1} (\boldsymbol{\theta}_t - \boldsymbol{\mu}^-) + \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y_t} | \boldsymbol{\theta}_t) = \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}), \tag{4.2}$$

where we are saying that the first derivative of our variational objective is roughly equivalent to that of its unbiased estimator. To be both explicit and pedagogical, we are assuming here that we can write the negative log posterior as

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta}_t - \boldsymbol{\mu}^-)^\top (\boldsymbol{\Sigma}^-)^{-1}(\boldsymbol{\theta}_t - \boldsymbol{\mu}^-) + \log p(\boldsymbol{y_t}|\boldsymbol{\theta}_t) + \text{const}, \tag{4.3}$$

and we are taking derivatives with respect to this objective. It follows that the first derivative is $\nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta})$ as defined above and the second derivative is

$$\nabla^2_{\boldsymbol{\theta\theta}}\mathcal{J}(\boldsymbol{\theta}) = (\boldsymbol{\Sigma}^-)^{-1} + \nabla^2_{\boldsymbol{\theta\theta}} \log p(\boldsymbol{y_t}|\boldsymbol{\theta}_t). \tag{4.4}$$

This yields the complete recipe for generating a corresponding quadratic Taylor approximation about $\boldsymbol{\theta}_t$:

$$\mathcal{Q}_t(\boldsymbol{\theta}) = \mathcal{J}(\boldsymbol{\theta}_t) + \nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta})^\top(\boldsymbol{\theta} - \boldsymbol{\theta}_t) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_t)^\top \nabla^2_{\boldsymbol{\theta\theta}}\mathcal{J}(\boldsymbol{\theta})(\boldsymbol{\theta} - \boldsymbol{\theta}_t), \tag{4.5}$$

where the gradient and Hessian are

$$\nabla_{\boldsymbol{\theta}}\mathcal{Q}_t(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}) + \nabla^2_{\boldsymbol{\theta\theta}}\mathcal{J}(\boldsymbol{\theta})(\boldsymbol{\theta} - \boldsymbol{\theta}_t), \tag{4.6}$$

$$\nabla^2_{\boldsymbol{\theta\theta}}\mathcal{Q}_t(\boldsymbol{\theta}) = \nabla^2_{\boldsymbol{\theta\theta}}\mathcal{J}(\boldsymbol{\theta}). \tag{4.7}$$

By setting $\nabla_{\boldsymbol{\theta}}\mathcal{Q}_t(\boldsymbol{\theta}) = 0$ and solving for $\boldsymbol{\theta}^*$, we get

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_t - ((\boldsymbol{\Sigma}^-)^{-1} + \nabla^2_{\boldsymbol{\theta\theta}} \log p(\boldsymbol{y_t}|\boldsymbol{\theta}_t))^{-1}((\boldsymbol{\Sigma}^-)^{-1}(\boldsymbol{\theta}_t - \boldsymbol{\mu}^-) + \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y_t}|\boldsymbol{\theta}_t)). \tag{4.8}$$

Now, the above minimization suggests an iterative procedure that we wish to

exploit. We can define this iterative procedure as follows:

$$\begin{aligned}\boldsymbol{\theta}_{t,k+1} &= \boldsymbol{\theta}_{t,k} + \boldsymbol{p}_k, \\ &= \boldsymbol{\theta}_{t,k} + \boldsymbol{A}_k^{-1}\boldsymbol{b}_k,\end{aligned}$$

where

$$\begin{aligned}\boldsymbol{A}_k^{-1} &= (\boldsymbol{\Sigma}^-)^{-1} + \nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\boldsymbol{y_t}|\boldsymbol{\theta}_{t,k}), \\ \boldsymbol{b}_k &= -((\boldsymbol{\Sigma}^-)^{-1}(\boldsymbol{\theta}_{t,k} - \boldsymbol{\mu}^-) + \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y_t}|\boldsymbol{\theta}_{t,k})),\end{aligned}$$

and $\boldsymbol{\theta}_{t,k}$ denotes parameters $\boldsymbol{\theta}$ at time step t and optimization iteration k. As such, we can use the conjugate gradient (CG) method to compute $\boldsymbol{p}_k$. When $\nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}_{t,k}) = 0$, $\boldsymbol{\theta}_{t,k}$ is a minimizer of the negative log posterior. We can use the modified procedure of [2] to reconstruct a low-rank approximation of $\boldsymbol{A}_k^{-1}$ from the descent directions of the inner CG procedure. This low-rank approximation will be propagated to the next time step of the filter.

Additional considerations include using the Gauss-Newton Hessian $\boldsymbol{G} \approx \nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$ as stated before and using a backtracking line-search to find an α_k such that $\mathcal{J}(\boldsymbol{\theta}_{t,k+1}) = \mathcal{J}(\boldsymbol{\theta}_{t,k} + \alpha_k \boldsymbol{p}_k)$ provides a sufficient decrease relative to $\mathcal{J}(\boldsymbol{\theta}_{t,k})$.

We can summarize all of this with the algorithm shown in Algorithm 1. This a simple procedure for both optimizing the negative log posterior and producing a low-rank covariance matrix.

Another option is to use Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) instead of the traditional CG algorithm [17]. LOBPCG can be used to do a low-rank eigendecomposition that then can be used to construct the low rank covariance matrices. Some benefits of LOBPCG are that it is more amendable to GPU usage and it can provide signficant speedups compared to its CG counterpart.

In the results section reported below, we used a LOBPCG-VKF instead of the CG-

Algorithm 1 Proposed CG-VKF Update Step

Input: $\boldsymbol{\mu}^-, \boldsymbol{\Sigma}^-$
Output: $\boldsymbol{\mu}^+, \boldsymbol{B}^+$ such that $\boldsymbol{B}^+(\boldsymbol{B}^+)^\top \approx \boldsymbol{\Sigma}^+$
$\boldsymbol{x}_0 \leftarrow \boldsymbol{\mu}^-$
$\boldsymbol{b}_0 \leftarrow -\nabla\mathcal{L}(\boldsymbol{x}_0)$
$\boldsymbol{G}_0 \leftarrow$ GAUSS-NEWTON-HESSIAN$(\mathcal{L}, \boldsymbol{x}_0)$
$\boldsymbol{A}_0 \leftarrow \boldsymbol{G}_0 + (\boldsymbol{\Sigma}^-)^{-1}$
$\boldsymbol{p}_0, \boldsymbol{B}_0 \leftarrow$ MODIFIED-CG$(\boldsymbol{A}_0, \boldsymbol{b}_0)$ ▷ {Solve $\boldsymbol{p}_0 \approx \boldsymbol{A}_0^{-1}\boldsymbol{b}_0$ and obtain low-rank approximation $\boldsymbol{B}_0\boldsymbol{B}_0^\top \approx \boldsymbol{A}_0^{-1}$}
$k \leftarrow 0$
while $\boldsymbol{p}_k \neq 0$ **do**
    $\boldsymbol{x}_{k+1} \leftarrow$ LINE-SEARCH$(\mathcal{J}, \boldsymbol{x}_k, \boldsymbol{p}_k)$ ▷ {Solve for α}
    $\boldsymbol{b}_{k+1} \leftarrow -(\nabla\mathcal{L}(\boldsymbol{x}_{k+1}) + (\boldsymbol{\Sigma}^-)^{-1}(\boldsymbol{x}_{k+1} - \boldsymbol{\mu}^-))$
    $\boldsymbol{G}_{k+1} \leftarrow$ GAUSS-NEWTON-HESSIAN$(\mathcal{L}, \boldsymbol{x}_{k+1})$
    $\boldsymbol{A}_{k+1} \leftarrow \boldsymbol{G}_{k+1} + (\boldsymbol{\Sigma}^-)^{-1}$
    $\boldsymbol{p}_{k+1}, \boldsymbol{B}_{k+1} \leftarrow$ MODIFIED-CG$(\boldsymbol{A}_{k+1}, \boldsymbol{b}_{k+1})$
    $k \leftarrow k + 1$
end while
$\boldsymbol{\mu}^+ \leftarrow \boldsymbol{x}_k$
$\boldsymbol{B}^+ \leftarrow \boldsymbol{x}_k$

VKF describe here. The methods are roughly identical, with the exception that we replaced the backtracking line search with a trust region method and the LOBPCG-VKF is roughly 10 times faster. We describe the CG-VKF method here instead of the LOBPCG-VKF method used to report results because the conjugate gradient method is significantly more familiar to the average reader and the derivation is considerably shorter for the purposes of this thesis.

One final note is that this method, in general, does not quantify the uncertainty of a neural network well. This should not be a surprise because we are constructing a low-rank Laplace approximation, which is only *locally* accurate. Due to this poor uncertainty quantification, we turn to the natural gradient variational Kalman filter.

4.3 Natural gradient variational Kalman filtering

Since the conjugate gradient VKF from the previous section does not accurately quantify uncertainty, we were motivated to explore other options. Natural Gradient Variational Inference (NVGI) is a promising new approach that promises variational inference at the implementation complexity of MLE. In this section, we provide a derivation of a Natural Gradient Variational Kalman Filter (NG-VKF) closely following the work of [15]. In the next section, we report results on this filter under the mean-field assumption that follows directly from the math shown here.

Suppose we are given some initialization distribution $p(\boldsymbol{\theta}_0)$, predicted distribution $p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t-1})$, and likelihood $p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{\theta}_t)$. Let $p(\boldsymbol{\theta}_0) = \mathcal{N}(\boldsymbol{\theta}_0|\mathbf{0}, \alpha^{-1}\boldsymbol{I})$, $p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t-1}) = \mathcal{N}(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}, \boldsymbol{\Sigma}_{t-1} + \boldsymbol{Q}_{t-1})$, and $p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{\theta}_t) = \mathcal{L}(\boldsymbol{\theta}_t; \boldsymbol{y}_t, \boldsymbol{x}_t)$, which is any arbitrary likelihood function assumed to be non-conjugate with the distribution from the predict step. This setup recovers what we referred to as a Variational Kalman Filter in section 2.1, with a transition distribution equal to the identity function.

We are interested in learning the parameters of a distribution $q(\boldsymbol{\theta}_t)$ for every time-step t that most accurately approximates our true filtering distribution $p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t})$. Let us assume that $q(\boldsymbol{\theta}_t)$ belongs to an exponential family such that

$$q(\boldsymbol{\theta}_t) = q(\boldsymbol{\theta}_t|\boldsymbol{\eta}) = h(\boldsymbol{\theta}_t) \exp\{\boldsymbol{\eta}^\top T(\boldsymbol{\theta}_t) - A(\boldsymbol{\eta})\}, \tag{4.9}$$

where $T(\boldsymbol{\theta}_t)$ is the sufficient statistic, $A(\boldsymbol{\eta})$ is the log-partition function, $\boldsymbol{\eta}$ are the natural parameters, and $h(\boldsymbol{\theta}_t)$ is the base measure. Note that we are writing the natural parameters of each time-step, $\boldsymbol{\eta}^{(t)}$, as $\boldsymbol{\eta}$ for notational convenience, and will be following this convention whenever the time-step t is clearly implied. Let us also assume a minimal exponential family, which yields a one-to-one mapping between $\boldsymbol{\eta}$

and mean parameters $\boldsymbol{m}$ such that

$$\boldsymbol{m} = \mathbb{E}_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})}\left[T(\boldsymbol{\theta}_t)\right] = \nabla_{\boldsymbol{\eta}} A(\boldsymbol{\eta}), \tag{4.10}$$

and

$$\nabla_{\boldsymbol{\eta}} \boldsymbol{m} = \nabla^2_{\boldsymbol{\eta}\boldsymbol{\eta}} A(\boldsymbol{\eta}) = \boldsymbol{F}(\boldsymbol{\eta}), \tag{4.11}$$

where $\boldsymbol{F}(\boldsymbol{\eta})$ is the Fisher information matrix (FIM) defined as

$$\boldsymbol{F}(\boldsymbol{\eta}) = \mathbb{E}_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})}\left[\nabla_{\boldsymbol{\eta}} \log q(\boldsymbol{\theta}_t|\boldsymbol{\eta}) \nabla_{\boldsymbol{\eta}} \log q(\boldsymbol{\theta}_t|\boldsymbol{\eta})^\top\right]. \tag{4.12}$$

We can find the best approximating distribution $q(\boldsymbol{\theta}_t|\boldsymbol{\eta})$ by maximising the Evidence Lower Bound (ELBO), which effectively minimizes the Kullback–Leibler (KL) divergence between the approximate distribution and true filtering distribution. The ELBO of a Variational Kalman Filter can be written as follows,

$$\mathcal{L}_{\text{ELBO}}(\boldsymbol{\eta}) = \mathbb{E}_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})}\left[\log p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{\theta}_t)\right] + \mathbb{E}_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})}\left[\log \frac{p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t-1})}{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})}\right], \tag{4.13}$$

which we aim to optimize with respect to $\boldsymbol{\eta}$. Note that $p(\boldsymbol{\theta}_1|\boldsymbol{y}_1) = \int p(\boldsymbol{\theta}_1|\boldsymbol{\theta}_0) p(\boldsymbol{\theta}_0) d\boldsymbol{\theta}_0$ when $t = 1$.

Given the ELBO of this setup, we are interested in optimizing with respect to $\boldsymbol{\eta}$. Standard natural-gradient descent follows the following iterative procedure:

$$\boldsymbol{\eta}_{k+1} = \boldsymbol{\eta}_k + \beta_k \boldsymbol{F}(\boldsymbol{\eta}_k)^{-1} \nabla_{\boldsymbol{\eta}} \mathcal{L}_{\text{ELBO}}(\boldsymbol{\eta})\big|_{\boldsymbol{\eta}=\boldsymbol{\eta}_k}, \tag{4.14}$$

where β_k is the learning rate. In the setting of Deep Neural Networks (DNN) or other highly parameterized models, the FIM is expensive to compute, store, and invert. To

avoid this, we utilize the equalities shown in (4.11),

$$\nabla_{\boldsymbol{\eta}}\mathcal{L}(\boldsymbol{\eta}_k) = [\nabla_{\boldsymbol{\eta}}\boldsymbol{m}_k]\nabla_{\boldsymbol{m}}\mathcal{L}_*(\boldsymbol{m}_k) = [\nabla^2_{\boldsymbol{\eta}\boldsymbol{\eta}}A(\boldsymbol{\eta}_k)]\nabla_{\boldsymbol{m}}\mathcal{L}_*(\boldsymbol{m}_k) = \boldsymbol{F}(\boldsymbol{\eta}_k)\nabla_{\boldsymbol{m}}\mathcal{L}_*(\boldsymbol{m}_k), \tag{4.15}$$

where $\mathcal{L}_*(\boldsymbol{m}_k) = \mathcal{L}(\boldsymbol{\eta}_k)$ is a reparameterized function expressed in terms of $\boldsymbol{m}$. Using this result, we can recast the iterative procedure for natural-gradient descent as

$$\boldsymbol{\eta}_{k+1} = \boldsymbol{\eta}_k + \beta_k \nabla_{\boldsymbol{m}}\mathcal{L}_{\text{ELBO}}(\boldsymbol{\eta})|_{\boldsymbol{\eta}=\boldsymbol{\eta}_k}. \tag{4.16}$$

Let $\boldsymbol{m}^{(t)} = \boldsymbol{m}$ for notational convenience. From (4.13), we have,

$$\begin{aligned}
\nabla_{\boldsymbol{m}}\mathbb{E}_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})}\left[\log\frac{p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t-1})}{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})}\right] &= \nabla_{\boldsymbol{m}}\mathbb{E}_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})}\left[(\boldsymbol{\eta}^{(t-1)} - \boldsymbol{\eta}^{(t)})^\top T(\boldsymbol{\theta}_t) + A(\boldsymbol{\eta}^{(t)}) + \text{const}\right] \\
&= \nabla_{\boldsymbol{m}}\left[(\boldsymbol{\eta}^{(t-1)} - \boldsymbol{\eta}^{(t)})^\top \boldsymbol{m}\right] + \nabla_{\boldsymbol{m}}A(\boldsymbol{\eta}^{(t)}) \\
&= \boldsymbol{\eta}^{(t-1)} - \boldsymbol{\eta}^{(t)} - \nabla_{\boldsymbol{m}}\left[\boldsymbol{\eta}^{(t)}\right]^\top \boldsymbol{m} + \nabla_{\boldsymbol{m}}A(\boldsymbol{\eta}^{(t)}) \\
&= \boldsymbol{\eta}^{(t-1)} - \boldsymbol{\eta}^{(t)} - \boldsymbol{F}(\boldsymbol{\eta}^{(t)})^{-1}\boldsymbol{m} + \boldsymbol{F}(\boldsymbol{\eta}^{(t)})^{-1}\boldsymbol{m} \\
&= \boldsymbol{\eta}^{(t-1)} - \boldsymbol{\eta}^{(t)},
\end{aligned}$$

where we utilize the product rule, equalities from (4.10) and (4.11), and the symmetry of the FIM. Now, we can update the iterative procedure in (4.16) using this simplification of the KL term in the ELBO:

$$\boldsymbol{\eta}^{(t)}_{k+1} = (1-\beta_k)\boldsymbol{\eta}^{(t)}_k + \beta_k\left(\boldsymbol{\eta}^{(t-1)}_k + \nabla_{\boldsymbol{m}}\mathbb{E}_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})}\left[\log p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{\theta}_t)\right]\right). \tag{4.17}$$

We will now make our first assumption: the variational family we wish to approximate our filtering distribution $p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t})$ with is Gaussian, such that $q(\boldsymbol{\theta}_t|\boldsymbol{\eta}) = \mathcal{N}(\boldsymbol{\theta}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ for some mean $\boldsymbol{\mu}_t$ and covariance $\boldsymbol{\Sigma}_t$ at time-step t. The minimal representation for a Gaussian family has two components to its natural parameters and

mean parameters,

$$\boldsymbol{\eta}^{(t)} = \begin{bmatrix} \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t \\ -\frac{1}{2}\boldsymbol{\Sigma}_t^{-1} \end{bmatrix}, \quad \boldsymbol{m}^{(t)} = \begin{bmatrix} \boldsymbol{\mu}_t \\ \boldsymbol{\mu}_t\boldsymbol{\mu}_t^\top + \boldsymbol{\Sigma}_t \end{bmatrix}.$$

Similarly, the minimal representation for the natural parameters and mean parameters of the predicted distribution $p(\boldsymbol{\theta}_t|\boldsymbol{y}_{1:t-1}) = \mathcal{N}(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}, \boldsymbol{\Sigma}_{t-1} + \boldsymbol{Q}_{t-1}) = \mathcal{N}(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}, \boldsymbol{\Sigma}_t^-)$ is

$$\boldsymbol{\eta}^{(t-1)} = \begin{bmatrix} [\boldsymbol{\Sigma}^-{}_t]^{-1}\boldsymbol{\theta}_{t-1} \\ -\frac{1}{2}[\boldsymbol{\Sigma}^-{}_t]^{-1} \end{bmatrix}, \quad \boldsymbol{m}^{(t-1)} = \begin{bmatrix} \boldsymbol{\theta}_{t-1} \\ \boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-1}^\top + [\boldsymbol{\Sigma}^-{}_t]^{-1} \end{bmatrix}.$$

Now, we want to write the natural-gradient updates for both parameters of the Gaussian, using equation (4.17). Here, we will take advantage of the chain rule, as shown in [22],

$$\nabla_{\boldsymbol{m}}\mathcal{F}_t = \begin{bmatrix} \nabla_{\boldsymbol{\mu}_t}\mathcal{F}_t - 2[\nabla_{\boldsymbol{\Sigma}_t}\mathcal{F}_t]\boldsymbol{\mu}_t \\ \nabla_{\boldsymbol{\Sigma}_t}\mathcal{F}_t \end{bmatrix}, \tag{4.18}$$

where $\mathcal{F}_t = \mathbb{E}_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})}[\log p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{\theta}_t)]$. We can start with the second component of $\boldsymbol{\eta}$ to yield

$$\boldsymbol{\Sigma}_{t,k+1}^{-1} = (1-\beta_k)\boldsymbol{\Sigma}_{t,k}^{-1} + \beta_k\left([\boldsymbol{\Sigma}^-{}_t]^{-1} - 2\nabla_{\boldsymbol{\Sigma}_t}\mathcal{F}_t\right), \tag{4.19}$$

where the double subscripts now correspond to optimization iteration step k for the parameters of time step t. Similarly, we can derive the update for mean $\boldsymbol{\mu}$ through the first component of $\boldsymbol{\eta}$,

$$\boldsymbol{\mu}_{t,k+1} = \boldsymbol{\mu}_{t,k} + \beta_k\boldsymbol{\Sigma}_{t,k+1}\left([\boldsymbol{\Sigma}^-{}_t]^{-1}(\boldsymbol{\theta}_{t-1} - \boldsymbol{\mu}_{t,k}) + \nabla_{\boldsymbol{\mu}_t}\mathcal{F}_t\right). \tag{4.20}$$

To further simplify, we can invoke Bonnet's and Price's theorems [22, 26],

$$\nabla_{\boldsymbol{\mu}_t} E_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})} \left[\log p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{\theta}_t)\right] = E_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})} \left[\nabla_{\boldsymbol{\theta}_t} \log p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{\theta}_t)\right] = -E_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})} \left[N\boldsymbol{g}(\boldsymbol{\theta_t})\right], \tag{4.21}$$

$$\nabla_{\boldsymbol{\Sigma}_t} E_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})} \left[\log p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{\theta}_t)\right] = \frac{1}{2} E_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})} \left[\nabla^2_{\boldsymbol{\theta}_t\boldsymbol{\theta}_t} \log p(\boldsymbol{y}_t|\boldsymbol{x}_t, \boldsymbol{\theta}_t)\right] = -\frac{1}{2} E_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})} \left[N\boldsymbol{H}(\boldsymbol{\theta}_t)\right], \tag{4.22}$$

where $\boldsymbol{g}(\cdot)$ and $\boldsymbol{H}(\cdot)$ are the average per-example gradient and Hessian of the negative log-likelihood over N examples, respectively. After substituting (4.21), (4.22) into (4.19), (4.20) we receive our final update equations:

$$\boldsymbol{\Sigma}^{-1}_{t,k+1} = (1-\beta_k)\boldsymbol{\Sigma}^{-1}_{t,k} + \beta_k \left(N E_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})} \left[\boldsymbol{H}(\boldsymbol{\theta}_t)\right] + [\boldsymbol{\Sigma}^{-}{}_t]^{-1}\right), \tag{4.23}$$

$$\boldsymbol{\mu}_{t,k+1} = \boldsymbol{\mu}_{t,k} - \beta_k \boldsymbol{\Sigma}_{t,k+1} \left(N E_{q(\boldsymbol{\theta}_t|\boldsymbol{\eta})} \left[\boldsymbol{g}(\boldsymbol{\theta}_t)\right] - [\boldsymbol{\Sigma}^{-}{}_t]^{-1}(\boldsymbol{\theta}_{t-1} - \boldsymbol{\mu}_{t,k})\right), \tag{4.24}$$

which is a non-stationary version of the Variational Online-Newton (VON) algorithm [15]. Estimating the Hessian of the negative log-likelihood using the Gauss-Newton Hessian yields the non-stationary version of the Variational Online-Gauss-Newton (VOGN) algorithm, which is what we use in the next section.

To conclude this section, we would like to note that this method provides better uncertainty quantification but we observed that the parameterizations are extremely poor. This is due to the geometry of the neural network being factored into the gradient steps. This method does not optimize to local minima but rather seeks robustness, avoiding regions of sharp curvature. Since neural networks are often sharp around their local minima, this can cause NG-VKFs to miss good parameterizations. We are exploring this problem in current research.

4.4 Results

In this final section, we describe some of the results we collected on these two methods using two different datasets. The first dataset, rotating moons, is a synthetic dataset with a data generating process that follows from placing two normal distributions on a two-dimensional plane, centered 0.5 away from each other with covariance $\mathbf{\Sigma}_1 = \mathbf{\Sigma}_2 = \epsilon \mathrm{I}$ for $\epsilon = 0.1$. From each distribution, 50 data points are sampled to create two classes. Sine and cosine nonlinearities are applied such that we transform each set of data points into shapes that resemble crescent moons (see Figure 4.1).

To generate a relationship that is time-varying, we can apply the following affine transformation to every data point:

$$\begin{bmatrix} \cos\frac{\pi}{6} & -\sin\frac{\pi}{6} \\ \sin\frac{\pi}{6} & \sin\frac{\pi}{6} \end{bmatrix},$$

at every time step. This recovers a system that is both spatially and temporally nonlinear.

For the second dataset we pulled the yearbook dataset from [33]. The idea is that social norms, fashion styles, and population demographics change over time. This notion is embodied by this yearbook dataset, which consists of 33,431 portrait photos from 1930 to 2013. Each photo is a $32 \times 32 \times 1$ grey-scale image corresponding to a label of male or female.

For the moons dataset analysis, we benchmarked 5 methods: our LOBPCG-VKF, batchwise ERM, SERM, NG-VKF, and the CG-VKF from [2]. Batchwise ERM takes all in-distribution timesteps and fits a classifier until convergence using those data points. SERM is as described in Chapter 2. Figure 4.1 shows a sample run that compares LOBPCG-VKF, batchwise ERM, and SERM.

All rotating moons results reported in this section used the average of 20 dif-

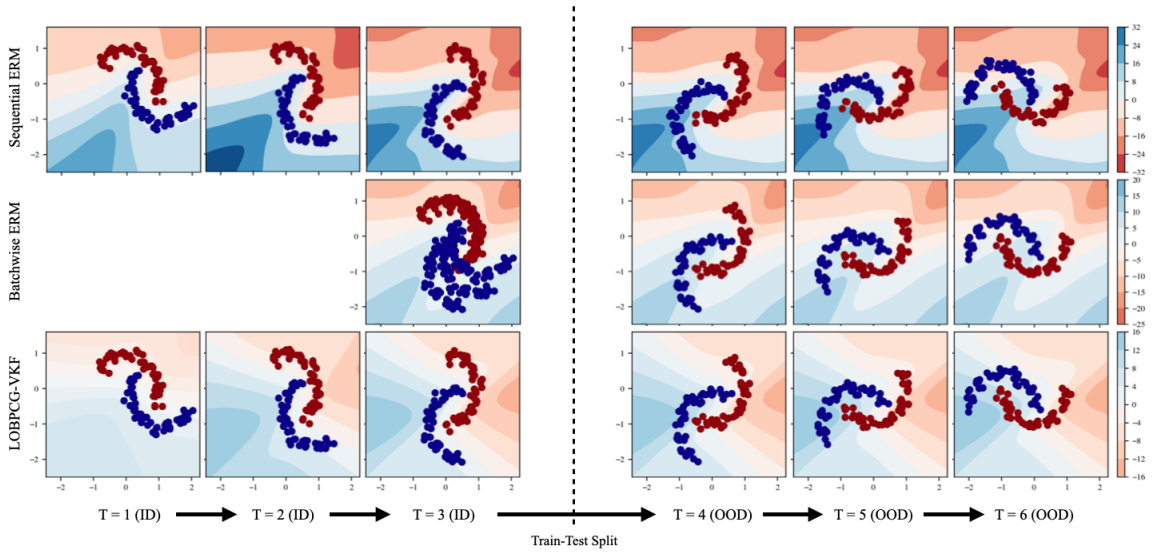


Figure 4.1: Moons dataset visualisation under three training paradigms using a random seed. (left) Three timesteps were used as a training set for the three approaches shown. (right) Predictions were made on three test timesteps where the models were agnositic to the data. (top) Sequential empirical risk minimization. Gradient descent was performed until convergence for everything training time step indivually. (middle) Batchwise empirical risk minimization. Gradient descent was performed until convergence for stochastic mini-batches of size = 100 over all training time steps until convergence. (bottom) Proposed method (rank 40) with $\boldsymbol{Q}_t = \text{diag}(0.75)$.

ferent seeds to generate the data and determine the network initialization weights. The network used in all experiments was a single layer dense network with a hidden layer width of 50 and tanh nonlinearity. For both ERM baselines, we use the Adam optimizer with a fixed learning rate of 10^{-3} and trained with a batch size of 100. Baselines were trained for 3000 iterations for both batchwise ERM and sequential ERM (per time step) settings. The LOBPCG-VKF was tested under 3 different covariance ranks: 10, 20, 40. The CG-VKF method from [2] was tested under covariance rank 40. We report results on the NG-VKF method separately, since they are not comparable. All methods assumed $\boldsymbol{A} = \text{I}$ and $\boldsymbol{Q} = \text{I}$. We do not report methods where the dynamics or process noise are learned in this section.

As shown in Figure 4.2, the LOBPCG-VKF was not affected adversely by the level of rank in the covariance matrix and matched performance of SERM at the

first two out-of-distribution time steps. On the third out-of-distribution time step, LOBPCG-VKF significantly outperformed all other methods. As seen in Figure 4.1, this is due to less overfitting and better generalization.

The CG-VKF from [2] has the worst performance since it assumed convexity over a network that is not convex. The NG-VKF derived in the previous section showed the most interesting negative results out of all methods tested.

We trained the NG-VKF under a wide range of configurations (learning rates, initializations, process noise values, etc.) and found that the best in-distribution performance was roughly 92 percent average accuracy, which is incomparable to the other 5 methods which all had roughly 99 percent in-distribution average accuracy. At briefly stated in the previous section, we believe that the incorporated information geometry prevents the NG-VKF from entering sharp local minima, which may be required to perform well on this task.

Out-of-distribution performance also suffered for the NG-VKF. The average $t+1$, $t+2$, and $t+3$ accuracies were 81 percent, 73 percent, and 45 percent, respectively. These we significantly worse than every method tested except the CG-VKF from [2].

To test whether or not this poor performance was a virtue of bad initializations, we ran experiments where we started each in-distribution timestep at the MLE solution, where average accuracy was 99 percent. Interestingly, after training, average accuracy recovered the 95 percent in-distribution accuracy as found before. This offered some intuition on how the NG-VKF might be escaping sharp local minima.

Despite this poor performance, we actually believe the NG-VKF is the way forward after a few modifications. First, we only had the time to test mean-field NG-VKF. A structured NG-VKF is something we are planning on deriving and implementing. We believe that this popping out of local minima problem might be better mitigated if the covariance matrices were not only diagonal. Second, modifying the NG-VKF such that it places slightly less weight on the information geometry near convergence

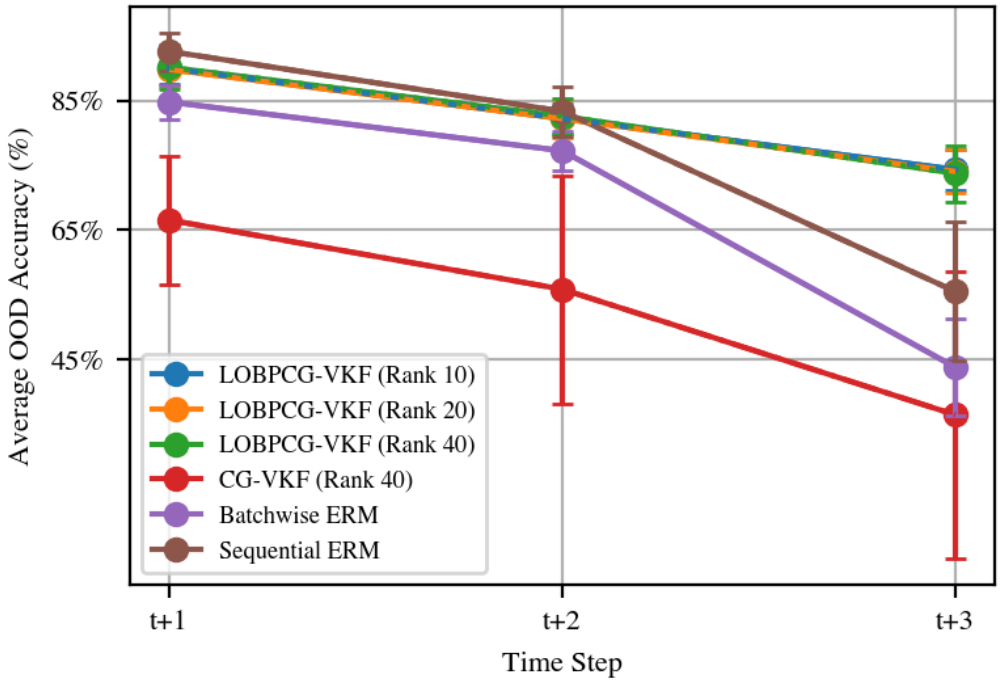


Figure 4.2: Out-of-distribution performance per test timestamp on the moons dataset. Under all six configurations, three timesteps were used as a training set.

may also help mitigate this problem. This is something that is relatively simple to do, and we plan to test. Third, it may be helpful to work with an approximation that in in between a Laplace approximation and a true variational approximation. Rather than taking Monte Carlo samples with respect to the variational distribution, we could take samples with respect to a more narrow variational distribution. This may be slightly unprincipled, but something we plan to try in the future.

With all of this said, let us turn to the yearbook data. We replicated the same procedures that we performed on rotating moons, but at a higher rank approximations to accommodate the increase in parameters. Following [33], we used a 4-layer convolutional network. Each convolutional layer had a kernel size 3×3, stride of 1×1, padding of size 1, 32 output channels, a spatial batch norm layer, ReLU activation, and a 2D max pool layer with kernel size 2×2. In this setting, we used

Method	ID (Average)	OOD (Average)	OOD (Worst)
Sequential ERM	94.13	**81.98**	**69.62**
Batchwise ERM	**97.99**	79.50	N/A
LOBPCG-VKF (rank 50)	92.68	78.44	66.17
LOBPCG-VKF (rank 100)	93.55	79.78	67.25
LOBPCG-VKF (rank 200)	93.34	80.56	67.45

Table 4.1: Yearbook Dataset Results. Sequential ERM and Batchwise ERM were the best performers for in-distribution (ID) and out-of-distribution (OOD) testing. All reported numbers are average and worst accuracies averaged over three different random seeds. The LOBPCG-VKF did not match the OOD performance of sequential ERM under any configuration tested.

the first 41 timesteps as in-distribution training data and the following timesteps as out-of-distribution test data.

In Table 4.1, we report the results of these experiments. Note that the CG-VKF from [2] and the NG-VKF are not included. The CG-VKF from [2] was too computationally intensive to produce results and the NG-VKF significantly underperformed on the simpler moons analysis. We did not conduct any analysis on the NG-VKF for this reason. We want to revisit the NG-VKF in future work.

For the yearbook dataset, Sequential ERM showed the best out-of-distribution performance and batchwise ERM showed the best in-distribution performance. Our proposed method was not as high performaning as it was in the rotating moons case. We believe this is because the convolutional neural network used on this dataset has four nonlinearities. This results in a loss landscape that is more complex than the loss landscape found in the rotating moons task. Since the Laplace approximation is only locally accurate, the risk of using it becomes higher as the loss landscape increases complexity.

Chapter 5

Conclusion

Bayesian filtering for neural networks is a promising approach for handling *temporal shift*, but significant further research is required before any of the methods described here could have an impact on the broader machine learning research community. In the linear-Gaussian case, we showed that the incorporation of learned dynamics and process noise can successfully treat temporal shift in a synthetic drifting linear regression task. To learn the dynamics and process noise, we used a simple gradient-based optimization approach that performed well. Unfortunately, in the higher dimensional and more complex parameter spaces of neural networks, this approach is impractical. More elegant solutions must be considered.

When testing on neural networks, we used two different variational Kalman filters with stationary dynamics and isotropic process noise. This was extremely limiting. Our hope with these assumptions were to provide a small performance boost against existing methods, namely the SERM described in Chapter 2. We found that we performed well in the simple rotating moons case but that performance did not generalize to the more complicated yearbook dataset case. This is because the Laplace approximation is only locally accurate. Further, the amount of computation required to use the LOBPCG-VKF method greatly inhibited its practicality. For these reasons, we

turned to the NG-VKF but found that performance was not better.

In light of the results reported here, we would like to use this concluding section to talk informally about future work we hope to do. This future work exists at three fronts: approximate inference, dynamics estimation, and process noise estimation. We believe this is the most useful way to conclude for any reader interested in iterating upon the work reported here.

5.1 Approximate Inference

The Bayesian filtering recipe requires a level of uncertainty quantication within every timestep, which is where approximate inference enters the picture. Variational Kalman filters (VKFs) are the most natural choice but optimizing a variational objective at hundreds of timesteps can be impractical. Considerations for VKFs in this setting include efficiency and accuracy. Since the uncertainty estimates also affect our ability to filter accurately, the stakes for approximate inference are high in this setting.

Natural gradient variational Kalman filters (NG-VKF) are promising because they are simple and they incorporate information geometry into their estimates. This is, however, a double-edged sword. It is important that we do not find ourselves in sharp local minima because they do not generalize well but wide local minima can be poor performers in some cases. In Chapter 4, we saw this with the rotating moons example. Figuring out the most appropriate way to incorporate information geometry is something we are interested in.

Another way forward is by following [6]. In [6], the authors state that the transition operator T is bijective. In most optimization cases, this is not true. However, finding a T that is bijective or conditionally bijective would mean we could construct a normalizing flow. This could allow us to build highly complex variational approx-

imations that may help our ability to perform approximate inference. Stochastic gradient descent, in particular, may be an interesting research avenue if put into the scope of normalizing flows [18, 25].

Lastly, the Bayesian Learning Rule (BLR) [16] may provide us with a principled way of improving our CG-VKF that is only locally accurate. The BLR states that all non-Bayesian learning algorithms are approximations of Bayesian ones. By following the BLR, we can take the CG-VKF derived here and make it more general via a number of subtle modifications. We hope to explore this in future work since conjugate gradients yield an easy way of recieving low-rank covariances.

5.2 Dynamics Estimation

In systems with meaningful dynamics, incorporation of this information is important. Since gradients are extremely noisy and expensive to compute, Bayesian optimization seems to be the best option for estimating the dyanmics matrix in future research [29].

The biggest problem with performing Bayesian optimization over neural networks is the number of parameters. It has been well documented that Bayesian optimization is extremely difficult in high dimensions. However, as pointed out in [29], it is only difficult if the number of *meaningful* dimensions is high.

[9] states the number of meaningful dimensions in a neural network is actually quite few in most settings. This may mean that Bayesian optimization over the dynamics matrix of a Bayesian filter for a neural network is tractable, as long as we can identify the meaningful dimensions efficiently and optimize over those. One interesting approach to high-dimensional Bayesian optimization is to use random embeddings [31]. In doing so, we can project ourselves onto a lower dimensional subspace where we can optimize our meaningful dimensions with very little inhibition.

This approach, and more recently proposed Bayesian optimization techniques [10], are avenues for estimating the dynamics matrix in future research. We hope to look into this subsequent to this thesis.

5.3 Process Noise Estimation

As mentioned informally at the beginning of Chapter 2, SERM can be seen as a nonparametric variational Kalman filter. This realization means that the Kalman gain is implicitly encoded by the number of gradient steps within a timestep. Since the Kalman gain is dependent on the process noise and measurement noise, we may be able to draw a connection between gradients and the process noise and use this connection to estimate the process noise from gradients alone.

Let us consider some common heuristics in machine learning. Typically, good parameterizations have small gradients. This is because local minima are defined by gradients equal to zero so any parameter set near a local minima should approach zero. Inversely, bad parameterizations usually have large gradients. Being far away from a local minima is almost always marked by nonzero gradients unless we are at a saddle point. Now, let us consider the difference between high process noise and low process noise.

High process noise means that we have a low probability of guessing the true parameters of our dynamical system using the dynamics alone. A low process noise means that we have a high probability of guessing the true parameters using the dynamics alone. We can view process noise as a metric for trustworthiness when considering our dynamics.

Taken together, we can see a correspondence between gradients and process noise if we realize that small gradients often mean trustworthy parameters and large gradients mean the parameters need to be adjusted. In settings where we can compute Jacobian

matrices, there may be a way to use these Jacobian matrices to compute the process noise in a fashion that is exact at the limit of infinite timesteps. This another idea we are currently interested in exploring in future work.

Bibliography

[1] H. Auvinen, J. M. Bardsley, H. Haario, and T. Kauranne. The variational kalman filter and an efficient implementation using limited memory bfgs. *International Journal for Numerical Methods in Fluids*, 64(3):314–335, 2010.

[2] J. M. Bardsley, A. Parker, A. Solonen, and M. Howard. Krylov space approximate kalman filtering. *Numerical Linear Algebra with Applications*, 20(2):171–184, 2013.

[3] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[4] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[5] N. Brosse, A. Durmus, and E. Moulines. The promises and pitfalls of stochastic gradient langevin dynamics. *Advances in Neural Information Processing Systems*, 31, 2018.

[6] D. Duvenaud, D. Maclaurin, and R. Adams. Early stopping as nonparametric variational inference. In *Artificial Intelligence and Statistics*, pages 1070–1077. PMLR, 2016.

[7] R. Factor, G. Yair, and D. Mahalel. Who by accident? the social morphology of car accidents. *Risk Analysis: An International Journal*, 30(9):1411–1423, 2010.

[8] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[9] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[10] R. Garnett. *Bayesian optimization*. Cambridge University Press, 2023.

[11] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[12] E. Grant, C. Finn, S. Levine, T. Darrell, and T. Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*, 2018.

[13] M. F. Huber, T. Bailey, H. Durrant-Whyte, and U. D. Hanebeck. On entropy approximation for gaussian mixture random vectors. In *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 181–188. IEEE, 2008.

[14] R. E. Kalman. A new approach to linear filtering and prediction problems. 1960.

[15] M. Khan, D. Nielsen, V. Tangkaratt, W. Lin, Y. Gal, and A. Srivastava. Fast and scalable bayesian deep learning by weight-perturbation in adam. In *International conference on machine learning*, pages 2611–2620. PMLR, 2018.

[16] M. E. Khan and H. Rue. The bayesian learning rule. *arXiv preprint arXiv:2107.04562*, 2021.

[17] A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM journal on scientific computing*, 23(2):517–541, 2001.

[18] I. Kobyzev, S. J. Prince, and M. A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):3964–3979, 2020.

[19] T. L Griffiths, C. Kemp, and J. B Tenenbaum. Bayesian models of cognition. 2008.

[20] C. Li, C. Chen, D. Carlson, and L. Carin. Preconditioned stochastic gradient langevin dynamics for deep neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[21] A. W. Lo. The adaptive markets hypothesis: Market efficiency from an evolutionary perspective. *Journal of Portfolio Management, Forthcoming*, 2004.

[22] M. Opper and C. Archambeau. The variational gaussian approximation revisited. *Neural computation*, 21(3):786–792, 2009.

[23] L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[24] M. Raginsky, A. Rakhlin, and M. Telgarsky. Non-convex learning via stochastic gradient langevin dynamics: a nonasymptotic analysis. In *Conference on Learning Theory*, pages 1674–1703. PMLR, 2017.

[25] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[26] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

[27] R. J. Santos. Equivalence of regularization and truncated iteration for general ill-posed problems. *Linear algebra and its applications*, 236:25–33, 1996.

[28] S. Särkkä. *Bayesian filtering and smoothing*. Number 3. Cambridge university press, 2013.

[29] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[30] Y. W. Teh, A. H. Thiery, and S. J. Vollmer. Consistency and fluctuations for stochastic gradient langevin dynamics. *Journal of Machine Learning Research*, 17, 2016.

[31] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, N. De Freitas, et al. Bayesian optimization in high dimensions via random embeddings. In *IJCAI*, volume 13, pages 1778–1784, 2013.

[32] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

[33] H. Yao, C. Choi, B. Cao, Y. Lee, P. W. W. Koh, and C. Finn. Wild-time: A benchmark of in-the-wild distribution shift over time. *Advances in Neural Information Processing Systems*, 35:10309–10324, 2022.