Practical Algorithms for Latent Variable Models

Gregory W. Gundersen

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN THE CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY.

RECOMMENDED FOR ACCEPTANCE BY
THE DEPARTMENT OF
COMPUTER SCIENCE

Adviser: Barbara E. Engelhardt

September 2021

© Copyright by Gregory W. Gundersen, 2021.

All rights reserved.

Abstract

Latent variables allow researchers and engineers to encode assumptions into their statistical models. A latent variable might, for example, represent an unobserved covariate, measurement error, or a missing class label. Inference is challenging because one must account for the conditional dependence structure induced by these variables, and marginalization is often intractable. In this thesis, I present several practical algorithms for inferring latent structure in probabilistic models used in computational biology, neuroscience, and time-series analysis.

First, I present a multi-view framework that combines neural networks and probabilistic canonical correlation analysis to estimate shared and view-specific latent structure of paired samples of histological images and gene expression levels. The model is trained end-to-end to estimate all parameters simultaneously, and we show that the latent variables capture interpretable structure, such as tissue-specific and morphological variation. Next, I present a family of nonlinear dimension-reduction models that use random features to support non-Gaussian data likelihoods. By approximating a nonlinear relationship between the latent variables and observations with a function that is linear with respect to random features, we induce closed-form gradients of the posterior distribution with respect to the latent variables. This allows for gradient-based nonlinear dimension-reduction models for a variety of data likelihoods. Finally, I discuss lowering the computational cost of online Bayesian filtering of time series with abrupt changes in structure, called changepoints. We consider settings in which a time series has multiple data sources, each with an associated cost. We trade the cost of a data source against the quality or *fidelity* of that source and how its fidelity affects the estimation of changepoints. Our framework makes cost-sensitive decisions about which data source to use based on minimizing the information entropy of the posterior distribution over changepoints.

Acknowledgments

Yongli Zhang hired me as a research technician when, with two degrees in architecture, I had no business working in his biophysics lab. I will always be grateful for his belief in me.

Before returning to school, I learned computer science and mathematics using free online resources. Thank you to these educators for their commitment to open-access education: Hal Abelson, Alex Aiken, Ana Bell, Brian Caffo, Eric Grimson, John Guttag, Steve Huffman, Sal Khan, Philip Klein, Andrew Ng, Barbara Oakley, Martin Odersky, Robert Sedgewick, Jeffrey Ullman, Kevin Wayne, and Jennifer Widom.

I thank Barbara Engelhardt for being a great advisor and mentor. She created an environment that combined intellectual freedom and academic integrity with kindness and inclusion. I also thank Ryan Adams, who made time for me, let me hang around his group, and taught me how to approach research problems with both creativity and rigor. Also, thanks to the other members of my committee for their time and thoughtful feedback: Tom Griffiths, Jonathan Pillow, and Brandon Stewart. Thanks to everyone in both Barbara's and Ryan's groups, especially my collaborators and office mates, with whom I had many helpful conversations. In particular, thanks to Diana Cai, Bianca Dumitrascu, and Michael Zhang.

I wish I could thank my grandmothers, Cleo and Eva, for their caregiving and insistence on an education. Finally, I am grateful for Jillian. Graduate school is hard for a lot of reasons. Her advice, patience, and love made the time enjoyable.

To explain all nature is too difficult a task for any one man or even for any one age. ‘Tis much better to do a little with certainty, & leave the rest for others that come after you, than to explain all things by conjecture without making sure of any thing.

Isaac Newton, 1704.

Contents

List of Figures

List of Tables

Mathematical conventions

Unless stated otherwise, I will use the following notational conventions:

x	Scalar
$\mathbf{x}$	Column vector
$\mathbf{X}$	Matrix
$\mathbf{x}_i$ or x_i	ith component of vector $\mathbf{x}$
$\mathbf{x}_{ni}$ or x_{ni}	ith component of the nth instance of vector $\mathbf{x}_n$
$\mathbf{X}_{ij}$ or X_{ij}	ith row and jth column of matrix $\mathbf{X}$
$\lVert\mathbf{x}\rVert_p$	p-norm of vector $\mathbf{x}$
$\mathbf{X}^\top$	Matrix transpose
$\mathbf{X}^{-1}$	Matrix inverse
$\det(\mathbf{X})$ or $\lvert\mathbf{X}\rvert$	Determinant of matrix $\mathbf{X}$
$\operatorname{tr}(\mathbf{X})$	Trace of matrix $\mathbf{X}$
$\operatorname{diag}(\mathbf{X})$	Diagonal elements of matrix $\mathbf{X}$
$\mathbf{0}$	Null vector or null matrix of all zeros
$\mathbf{I}$	Identity matrix
$p(\mathbf{x})$ or $\mathbb{P}(\mathbf{x})$	Probability of $\mathbf{x}$
$p(\mathbf{x} \mid \mathbf{y})$ or $\mathbb{P}(\mathbf{x} \mid \mathbf{y})$	Probablity of $\mathbf{x}$ given $\mathbf{y}$
$f(\mathbf{x}; \mathbf{w})$ or $f_{\mathbf{w}}(\mathbf{x})$	Function f with input $\mathbf{x}$ and parameters $\mathbf{w}$
$\mathbf{x} \sim P_{\boldsymbol{\theta}}$	Random variable $\mathbf{x}$ with distribution P with parameters $\boldsymbol{\theta}$
$p_{\boldsymbol{\theta}}(\mathbf{x})$	Probability function of a distribution with parameters $\boldsymbol{\theta}$

Chapter 1

Introduction

1.1 Scientific data analysis with latent variables

The goal of science is to understand the physical world through observation, hypothesis, and experimentation. Data analysis aids this process by discovering patterns in data. Scientists can then use those patterns to formulate and test hypotheses. In 1854, for example, London suffered a severe cholera outbreak. The physician John Snow mapped the locations of the cases and compared death-rates between water districts to find the root cause, germ-contaminated water [Snow, 1855]. In other words, Snow used data visualization and summary statistics to interpret his data and develop a testable scientific hypothesis.

Today, with emerging challenges such as climate change, widespread disinformation, and global pandemics, we need data-based decision-making more than ever. With more data and faster computers, machine learning, or the study of computer programs that automatically learn patterns from data and then act on those inferences [Murphy, 2012], has the potential to help address these types of global challenges. Machine learning can provide the tools that researchers, engineers, policy-makers, and other practitioners need in order to make sense of massive data sets that are often heterogeneous, non-stationary, richly structured, and growing rapidly.

The idea that machines might learn from data is as old as the modern computer. In the first half of the twentieth century, mathematicians and scientists were already asking questions such as whether neural activity could be described using propositional logic [McCulloch and Pitts, 1943] or whether machines could think [Turing, 1950]. The now-famous "Dartmouth workshop"[1] in the summer of 1956 is widely considered to be the founding event of the field of artificial intelligence, and it was held only a decade after the first electromechanical computers were built. However, in the last few decades, machine learning, a subfield of artificial intelligence, has emerged as the front-runner of the field. This prominence is due to successes in computer vision [Krizhevsky et al., 2012], board and video game play [Campbell et al., 2002, Mnih et al., 2013, Silver et al., 2016], natural language processing [Vaswani et al., 2017], protein folding [Senior et al., 2020], and healthcare [Gulshan et al., 2016], to name a few. Additionally, many other related research agendas have made important advances to address the scale, velocity, and heterogeneity of data in the modern world, such as randomized algorithms for numerical linear algebra [Halko et al., 2011, Mahoney, 2016], signal processing techniques for reconstructing signals from inaccurate measurements [Candes et al., 2006], approximation methods for scaling kernel machines [Snelson and Ghahramani, 2006, Rahimi and Recht, 2007, Wang et al., 2019], Bayesian nonparametric methods that can handle an infinite number of parameters [Ferguson, 1973, Teh et al., 2006], generative models for richly structured data [Kingma and Welling, 2013, Goodfellow et al., 2014], and improved methods for automatic differentiation [Baydin et al., 2018] and automatic integration [Metropolis et al., 1953, Hastings, 1970, Duane et al., 1987, Hoffman and Gelman, 2014]. These research agendas are bolstered by the widespread adoption and support of publicly available data sets [Deng et al., 2009, Geiger et al., 2012], open-source software libraries [Harris et al., 2020, Virtanen et al., 2020], programming languages [Ihaka, 1998, Bezanson et al., 2017], and new programming paradigms [Carpenter et al., 2017, Maclaurin et al., 2015, Paszke et al., 2017, Dillon et al., 2017]. In short, machine learning today holds immense promise for many fields, and this promise is grounded in decades of work by researchers across fields. The recent accelerants are massive data, faster computers, and algorithms that automatically learn from data.

While *learning from data* could and often does refer to computer programs building internal representations from data that can be used on downstream tasks, *learning from data* can also refer to building representations that help humans understand their data better. This latter sense of the phrase is particularly suited for scientific work. Scientists bring

[1]Officially, the *Dartmouth Summer Research Project on Artificial Intelligence.*

significant prior knowledge that, at this time, is difficult to encode into algorithms. In Leo Breiman's famous paper on statistical modeling [Breiman et al., 2001], he posits two camps or cultures on learning from data. One camp is interested in algorithmic modeling, without considering data-generating mechanisms. Put simply, if it works, it works. The other camp treats data as observations from stochastic processes and attempts to understand their data by modeling those processes. One can then interpret their model's inferences through an explicit mathematical model. This thesis is firmly in the second camp. It takes a probabilistic approach to machine learning [Hastie et al., 2009, Murphy, 2012, Ghahramani, 2015] because such models have many desirable properties for scientific data analysis.[2]

The pillar of a probabilistic approach to machine learning is the *statistical model* mentioned above. In a statistical model, data are viewed as observations from a random process, which is governed by some statistical or *generative* parameters. The goal is to infer these generative parameters. For example, if we have a coin which we suspect is unfair, we can flip it many times. These data can then be viewed as realizations from a Bernoulli process, and statistical inference estimates the Bernoulli process's bias parameter. While this is a simple probabilistic model, the power of such an approach is that we can use all the tools from probability theory to compose larger models in a mathematically principled way.

However, complex data often exhibit simpler but unobserved patterns that are not captured by statistical parameters. To model these patterns, we can introduce hidden or *latent* variables into our models. A *latent variable model*, then, is simply a statistical model with such unobserved variables. In the simplest inferential setting, a latent variable is distinct from a parameter in that it is still a random variable, i.e. it is a draw from a distribution rather than a quantity that specifies a distribution. However, the Bayesian statistician makes no real distinction between latent variables and parameters, treating both as unknown random variables. (I will discuss the Bayesian philosophy of statistics [Bayes, 1763, Laplace, 1820] later.) Another distinction, and one which persists under a Bayesian perspective, is that often the number of parameters in a model is fixed, while the number of latent variables

[2]This view of machine learning is closely related to computational statistics, which bends the traditional focus of statistical methods towards problems with large and heterogeneous data sets. However, I do not find the distinction between statistics and machine learning to be particularly useful. Much like the distinction between theory and practice, there exists a rich interplay between these two agendas that will likely increase as the complexity and scale of problems grow.

often grows with the number of observations.

We can appreciate the value of latent variable modeling through an example. Consider the task of learning patterns from sequential data. Imagine that every day, a man flips a coin to decide whether or not to take his umbrella to work, i.e. if the coin is heads with some probability $\theta \in [0, 1]$, he takes his umbrella. However, he selects the coin from a set of two coins depending on the weather. If it's cloudy, he uses a coin with bias $\theta_{\mathsf{cloudy}} = 0.8$. If it's sunny, he uses a coin with bias $\theta_{\mathsf{sunny}} = 0.05$. However, we only observe whether or not he takes his umbrella to work. Here, the statistical parameter of interest is still the Bernoulli bias for each day t, call this $\theta_t \in \{\theta_{\mathsf{cloud}}, \theta_{\mathsf{sunny}}\}$. However, the latent variable on day t, call this z_t, indexes into a set of states, specifying which bias to use. We have now built a more complex model than the simple Bernoulli statistical model discussed above and can infer more complex structure than can be modeled with just one bias parameter. We can interpret our inferences of both the parameters and the latent variables through the context of this mathematical model.

Scientists have been interested in latent variable models for over a century. Early pioneering work in the field include factor analysis, developed by Charles Spearman in his study of human intelligence [Spearman, 1904]; principal component analysis (PCA), developed by Karl Pearson for simplifying data sets [Pearson, 1901]; and canonical correlation analysis (CCA), developed by Harold Hotelling for joint analysis of two multivariate data sets [Hotelling, 1936]. Today, these factor models are still pervasive. See Engelhardt and Stephens [2010] for just one example: using a sparse factor model to infer simpler patterns from complex and high-dimensional genomics data. Latent variable models are particularly appealing for applied tasks in science and engineering precisely because of their interpretability. Practitioners can make sense of their data by inferring parameters and latent variables and then understanding those quantities through the conditional dependency structure of their models. See Blei [2014] for a more detailed overview and further discussion.

The examples above are all *linear–Gaussian factor models* because the observations are assumed to be linear functions of Gaussian-distributed latent variables. Due to nice computational properties of the Gaussian distribution (see Bishop [2006] for a discussion), these models have been explored deeply, and these modeling assumptions connect a variety of seemingly unrelated methods [Roweis and Ghahramani, 1999]. PCA is just factor analysis with isotropic variance. CCA is factor analysis with two data sets being generated from shared latent variables. A dynamic form of a linear–Gaussian factor model is the Kalman filter [Kalman, 1960], and the discrete-state version of the Kalman filter is the hidden Markov model (HMM) [Baum and Petrie, 1966]. (See Minka [1999] for details on this connection.) From another perspective, linear–Gaussian state space models such as the Kalman filter can be reformulated as Gaussian random fields [Hartikainen and Särkkä, 2010]. In addition to linear–Gaussian factor models, latent variable models have been used for nonlinear dimension reduction [Lawrence, 2004], clustering [Duda et al., 1973], topic modeling [Blei et al., 2003], and non-stationary time series analysis [Adams and MacKay, 2007, Fearnhead and Liu, 2007], to name just a few examples.

1.2 Practical algorithms for latent variable models

Despite their long history and use in statistics, psychology, genetics, and a variety of other fields, latent variable models have mostly remained within the regime of linear–Gaussian models. However, relatively recently, more flexible models have been developed. For example, the Gaussian process latent variable model (GPLVM) [Lawrence, 2004] is a generalization of PCA in which the observations are nonlinear rather than linear functions of the latent variables. The autoencoder [Baldi and Hornik, 1989] was introduced as PCA with neural networks. Methods such CCA and HMMs have also been extended to nonlinear settings using neural networks [Andrew et al., 2013, Krishnan et al., 2017], and Bayesian filtering algorithms for changepoint detection [Chib, 1998, Fearnhead, 2006, Chopin, 2007] are similar to other dynamic programming approaches to time series data analysis such as HMMs, while modeling a particular type of non-stationary behavior with possibly non-Gaussian data.

Inference is challenging in more flexible latent variable models because we must still account for the conditional dependencies induced by the latent variables. The probabilistic solution to this problem would be to marginalize out the latent variables, but this is often intractable (see Bishop [2006] or Blei et al. [2017] for examples). In this thesis, I present several practical algorithms for inferring latent structure with more flexible latent variable models for problems in computational biology, neuroscience, and time-series analysis.

The outline of my thesis is as follows. In Chapter 2, I present the necessary background to understand this thesis, focusing on probabilistic modeling (as opposed to algorithmic modeling), latent variable modeling, and Bayesian inference. In Chapter 3, I present deep probabilistic canonical correlation analysis (DP-CCA). DP-CCA composes deep neural networks with probabilistic CCA [Bach and Jordan, 2005] to infer shared variation in paired samples of histology images and molecular features. The inference challenge is to learn both the neural network parameters and the latent variables jointly, using end-to-end inference similar to a variational autoencoder [Kingma and Welling, 2013]. In Chapter 4, I present random feature latent variable models (RFLVMs). RFLVMs are a class of nonlinear dimension reduction methods that use random features [Rahimi and Recht, 2007] to induce analytic tractability for GPLVMs with non-Gaussian observations. This allows for well-specified dimension reduction on non-Gaussian data, such as images and text. Finally, in Chapter 5, I present a multi-fidelity approach to Bayesian online changepoint detection (BOCD) [Adams and MacKay, 2007, Fearnhead and Liu, 2007]. Standard BOCD is a Bayesian online filtering algorithm for non-stationary time series data. We extend BOCD to the multi-fidelity setting, in which we have multiple data sources, each with an associated quality metric or "fidelity", as well as a cost. We propose an active fidelity selection strategy based on maximizing the information rate (gain over cost) with respect to an important latent variable in BOCD that models the current behavior of the time series. In Chapter 6, I discussion several future directions and summarize my contributions.

The theme of the work in this thesis is the construction of mathematically principled latent variable models and their application to scientifically interesting problems. By developing betters tools for more scalable, tractable, and flexible latent variable models, we can accelerate the scientific process by automatically extracting interpretable insights from data.

1.3 How to read this

In my experience, the hardest part of graduate school was developing a deep enough understanding of my field such that I could think creatively. The challenge was two-fold. First, I had understand ideas rigorously, and then I had to move past rigor into intuition [Tao, 2009]. I suspect this struggle is common, but in my mind, the writing style of scientific papers inadvertently obfuscates this struggle for others because it was designed to highlight and convey novelty. Any concept that is not a main contribution is cited and then taken as a given. This is good; scientific papers would constantly rehash prior work otherwise. However, a novice might mistake this writing style for how scientists actually think and produce new ideas.

My aim in writing this thesis was to include the process, not just the products, of research. As a consequence, the background material in Chapter 2 is broader and more elementary than usual. Each main chapter contains extended background sections and even lengthier appendices. The appendices contain many detailed notes, derivations, and computer programs that I made along the way, often circuitously, to the main ideas of the chapter. The reader is encouraged to skip and re-read as desired. To quote Richard Feynman, "It is not complicated. It is just a lot of it" [Dowling and Gassner, 1976].

I imagine the ideal reader is a curious person with a good grasp of calculus, probability theory, and linear algebra—or my future self. For brevity, I take for granted that the reader understands the basics of machine learning. I won't, for example, define *clustering*, *supervised learning*, or other elementary topics. If any ideas in this thesis are unfamiliar and unexplained, please consult a textbook. My favorite machine-learning textbooks are Bishop [2006] and MacKay [2003], since these explain ideas from first principles. Hastie et al. [2009] and Murphy [2012] are excellent as references, while Gelman et al. [2013] is good for more advanced topics in Bayesian data analysis. For advanced topics in numerical linear algebra, which is useful for understanding the computational primitives upon which much of modern data analysis is built, I recommend Trefethen and Bau III [1997].

Given that this thesis contains many detailed technical notes, written by and for myself to learn new material and without review from others, it inevitably contains errors. My apologies to the reader. The goal of these notes was to convince myself, and I suggest the reader do the same.

Chapter 2

Background

In this chapter, I present the main ideas required to understand this thesis. I first formalize a probabilistic approach to machine learning [Ghahramani, 2015, Murphy, 2012], comparing two well-known algorithms and formalizing the notion of statistical inference. I then discuss latent variable modeling [Blei, 2014], or statistical modeling with unobserved structure. I give an example of such a model and present an important algorithm for inferring parameters and latent variables in such models. Finally, I discuss the Bayesian philosophy towards inferential statistics [Bayes, 1763, Laplace, 1820], in which parameters are themselves random variables. Methods for Bayesian inference can be broadly categorized into two groups, exact and approximate, and I discuss a number of important ideas for both groups.

2.1 Probabilistic modeling

In a probabilistic approach to machine learning, our data $\mathbf{X} \coloneqq \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ are viewed as realizations or observations from a random process. Formally, let $\mathcal{P} \coloneqq \{P_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \boldsymbol{\Theta}\}$ denote a class of models. This notation means that the distribution $P_{\boldsymbol{\theta}}$ is from a family of distributions, which are indexed by parameters $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. For example, consider the family of Poisson distributions. Here, θ is the Poisson's rate parameter, $\boldsymbol{\Theta}$ is the positive real numbers, and each particular choice of $\theta \in \mathbb{R}_{>0}$ induces a specific Poisson distribution.

In statistical modeling, the goal is to estimate optimal parameters $\boldsymbol{\theta}^{\star}$ from data, such that

$$\mathbf{x} \sim P_{\boldsymbol{\theta}^{\star}}, \tag{2.1}$$

is a good approximation of the assumed underlying data generating process, which in turn is an approximation of nature.

Consider, for example, a standard binary classification task. Our data set $\mathcal{D}$ is a collection of N targets or labels, $y_n \in \{0, 1\}$, and N independent variables or features, $\mathbf{x}_n \in \mathbb{R}^D$. Denote these data as $\mathcal{D} := \{(y_1, \mathbf{x}_1), \dots, (y_N, \mathbf{x}_N)\}$. Logistic regression is a statistical model such that the marginal distribution of our targets is

$$y_n \mid \mathbf{x}_n, \boldsymbol{\theta} \sim \text{Bernoulli}\left(p_n\right), \qquad p_n := \frac{\exp\left\{\boldsymbol{\theta}^\top \mathbf{x}_n\right\}}{1 + \exp\left\{\boldsymbol{\theta}^\top \mathbf{x}_n\right\}}. \tag{2.2}$$

This model assumes that our observations are realizations from a Bernoulli process, and that the bias of that process is a function of the independent variables. The coefficients $\boldsymbol{\theta}$ define a hyperplane in $\mathbb{R}^D$, and we can interpret the importance of each feature using the magnitude of the associated coefficient (component in $\boldsymbol{\theta}$). The logistic function, $f(\mathbf{x}) = e^{\mathbf{x}}/(1 + e^{\mathbf{x}})$, is necessary to map $\boldsymbol{\theta}^\top \mathbf{x}_n$ onto the support of the Bernoulli bias, ensuring $p_n \in [0, 1]$.

This example demonstrates how statistical modeling is not simply inferring parameters from distributions. Rather, we can build and compose more complex models using probability theory.

2.1.1. Example: Gaussian mixture model

To motivate these ideas, I want to compare two approaches to clustering data: *K-means* and a *Gaussian mixture model* (GMM). These models perform the same task in similar ways, but GMMs are probabilistic. This comparison highlights the distinction between algorithmic and probabilistic modeling, as discussed in Chapter 1 and in Breiman et al. [2001].

K-means. K-means is an unsupervised learning algorithm for partitioning N data points into K clusters. Each cluster is represented by a single parameter $\boldsymbol{\mu}_k$, which is a D-vector representing the cluster's kth mean. Given independent variables without labels, $\mathbf{X} := \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, the goal is to find cluster assignments such that the total intra-cluster squared Euclidean distance from the cluster means is minimized. We can also view this as finding clusters with low variance.

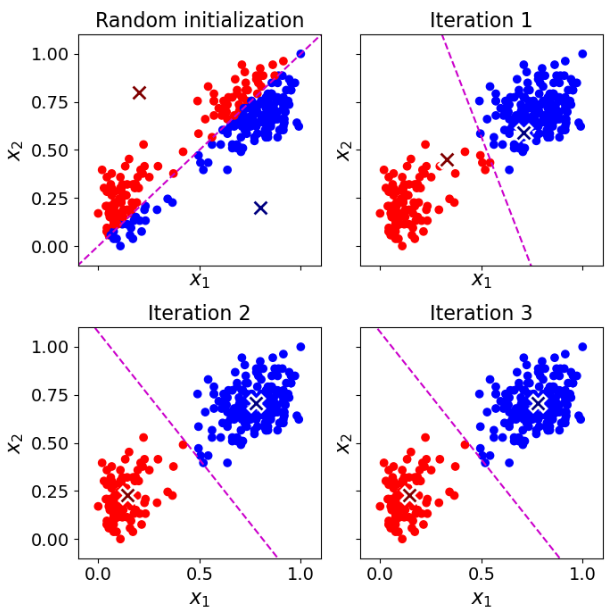

Figure 2.1: A visualization of the K-means algorithm ($K = 2$) on feature-scaled Old Faithful data. First, the means are randomly initialized. Then each datum is assigned to one of two clusters until convergence.

To express this as an objective function, we introduce one-hot vectors $\mathbf{z}_n$ such that variable $z_{nk} = 1$ if the cluster assignment is k for the nth data point. Then the objective is to minimize the quantity J,

$$J := \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|_2^2. \tag{2.3}$$

Our goal is to learn the mean parameters and cluster assignments. The algorithm for fitting this model is an iterative procedure. At a high-level, the algorithm iteratively updates the cluster assignments $\mathbf{Z} := \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$ by minimizing J with respect to (w.r.t.) to $\mathbf{Z}$ while keeping the cluster means $\mathbf{U} = \{\boldsymbol{\mu}_k\}_{k=1}^{K}$ fixed. Next, it updates $\mathbf{U}$ by minimizing J w.r.t. to $\mathbf{U}$ while keeping $\mathbf{Z}$ fixed [Bishop, 2006]. The algorithm stops when it *converges*, meaning when the value of J does not change between iterations[1]. I have visualized K-means on Old Faithful data[2] (Figure 2.1[3]).

[1]Typically, this is quantified as when the difference in values for J between iterations is less than some small value ε.

[2]`http://www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat`

[3]This figure and Figure 2.3 were inspired by Bishop [2006].

Once the algorithm has converged, we have a K-parameter model that allows us to cluster or label new, unseen data. We assign a new datum $\mathbf{x}_{N+1}$ to one of the clusters by computing the closest cluster mean, as quantified by the 2-norm $\|\cdot\|_2$.

Gaussian mixture model. Note that nothing about K-means is probabilistic. There are no statistical parameters or learned densities. Instead, K-means is a completely algorithmic approach to the task of clustering.

To approach the problem from a probabilistic perspective, we can use a *mixture model*. A mixture model represents a density as a mixture of weighted densities. For example, a *Gaussian mixture model* (GMM) is a mixture model in which the weighted densities are all Gaussian. This idea is easy to visualize with univariate Gaussian distributions (Figure 2.2).

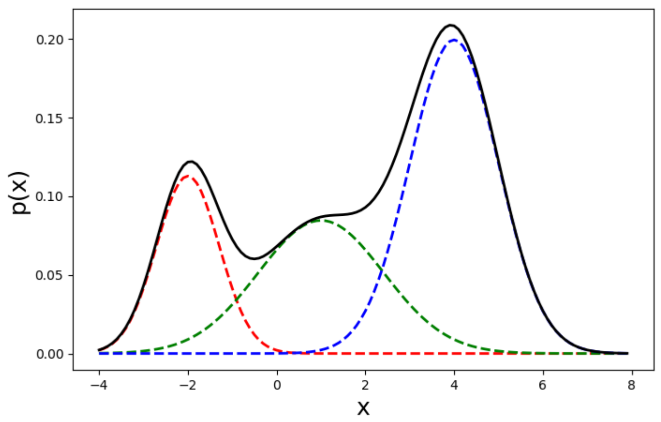

Figure 2.2: A Gaussian mixture model that is a weighted combination of univariate Gaussian distributions.

We can formalize a Gaussian mixture model as the sum of K weighted Gaussian densities, called *components*, each with its own mean $\boldsymbol{\mu}_k$, covariance $\boldsymbol{\Sigma}_k$, and mixture weight α_k, writing the density function as

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{k=1}^{K} \alpha_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \qquad \alpha_k \geq 0, \tag{2.4}$$

where $\boldsymbol{\theta} = \{(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\}_{k=1}^K$. For example, the mixture model in Figure 2.2 can be written as

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = 0.3\mathcal{N}(\mathbf{x} \mid 1, 2) + 0.2\mathcal{N}(\mathbf{x} \mid -2, 0.5) + 0.5\mathcal{N}(\mathbf{x} \mid 4, 1). \tag{2.5}$$

We can see that the mixture weights must sum to unity for this to be a valid density function:

$$1 = \int_{\mathcal{X}} p(\mathbf{x}) = \int_{\mathcal{X}} \sum_{k=1}^{K} \alpha_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \sum_{k=1}^{K} \alpha_k \int_{\mathcal{X}} \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \sum_{k=1}^{K} \alpha_k. \tag{2.6}$$

Estimating the GMM cluster assignment variables and parameters is more complicated than inference for K-means. However, it is also an iterative procedure, and I will discuss it in Section 2.2.1.

To help visualize GMMs, I created a small ($N = 500$) synthetic data set[4] with three multivariate Gaussian densities, fit a GMM to the data, and then visualized the model at various iterations (Figure 2.3).

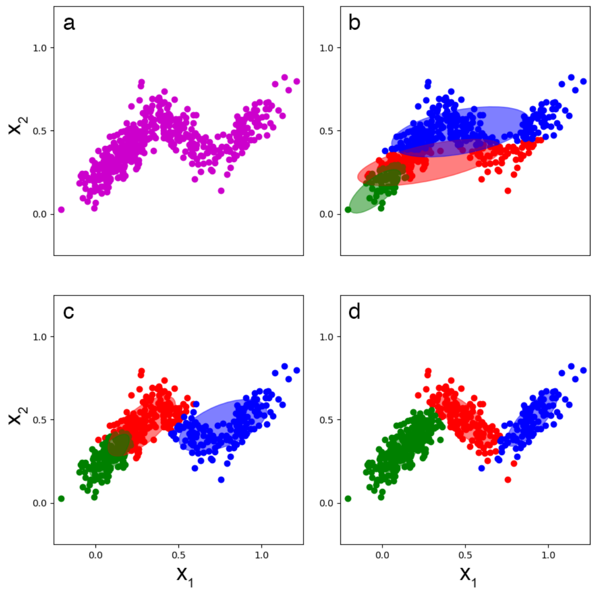

Figure 2.3: A visualization of fitting a Gaussian mixture model ($K = 3$) on (a) synthetic data. (b) The components are randomly initialized. (c-d) The parameters of the model are iteratively updated using an algorithm discussed in Section 2.2.1.

Let's now discuss a few of the benefits of probabilistic modeling. See Ghahramani [2015]

[4]I am not using the Old Faithful data set here because I want to demonstrate how GMMs handle overlapping clusters.

for a detailed discussion.

Handling uncertainty. In the algorithmic approach of K-means, the best parameters $\boldsymbol{\theta}^\star := \{\boldsymbol{\mu}_k^\star\}_{k=1}^K$ given our data are fixed and do not encode any uncertainty about the parameters. But with a probabilistic approach, modeling our data given this uncertainty has a natural mathematical solution, namely *marginalization*:

$$p(\mathbf{x}) = \int_{\boldsymbol{\Theta}} p_{\boldsymbol{\theta}}(\mathbf{x}) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}. \tag{2.7}$$

This marginal distribution allows us to remove our uncertainty about $\boldsymbol{\theta}$ by considering all possible parameter values. We will see in Section 2.3 that a Bayesian approach is particularly interested in parameter marginalization and uncertainty quantification.

Modeling generative processes. A probabilistic model is sometimes referred to as a *generative model* because we can generate new data using our inferred parameters. Generative modeling enriches our understanding of what the model has learned.

For example, with a GMM, we can sample a new assignment variable $\mathbf{z}_{N+1}$, since this one-hot vector can be viewed as a categorical random variable with event probabilities $\boldsymbol{\alpha}^\star$, which are the estimated mixture weights:

$$\mathbf{z}_{N+1} \sim \text{Categorical}(\boldsymbol{\alpha}^\star). \tag{2.8}$$

This ensures that, in expectation, the number of generated data points for each cluster is proportional to what we observed. Next, for the sampled cluster assignment $\mathbf{z}_{N+1}$, we can sample from a multivariate Gaussian with our estimated parameters $\boldsymbol{\mu}^\star_{\mathbf{z}_{N+1}}$ and $\boldsymbol{\Sigma}^\star_{\mathbf{z}_{N+1}}$:

$$\mathbf{x}_{N+1} \sim \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}^\star_{\mathbf{z}_{N+1}}, \boldsymbol{\Sigma}^\star_{\mathbf{z}_{N+1}}\right). \tag{2.9}$$

I have visualized the estimated density and some generated samples using this procedure in Figure 2.4.

In my mind, this visualization really emphasizes the mixture weights. The figure visual-

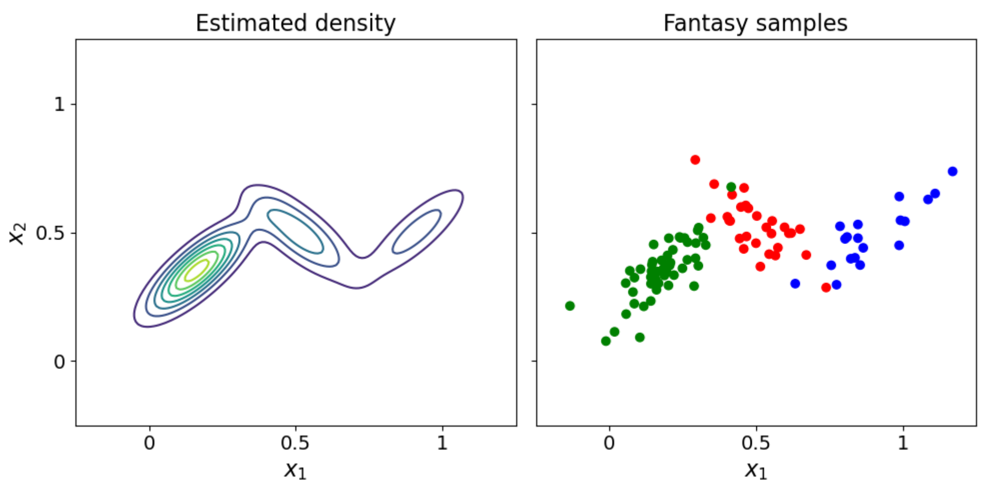

Figure 2.4: (Left) The inferred GMM density given our observations in Figure 3. (Right) One hundred generated samples from our inferred GMM density using the sampling procedure described in the text.

izes the mathematical fact that the blue density has the least amount of mass and that, as a consequence, it is the least likely to be sampled from. We can inspect the model's estimated mixture weights and see that this is true:

$$\begin{aligned} \text{green } \alpha &\to 0.511 \\ \text{red } \alpha &\to 0.295 \\ \text{blue } \alpha &\to 0.194 \end{aligned} \tag{2.10}$$

The actual mixture weights for the synthetic data set are 0.5, 0.3, and 0.2, in the same order as above.

Compositionality. An appealing property of a probabilistic approach is that simple distributions can be used as building blocks for more complex models. In other words, we can judiciously compose random variables to encode conditional dependencies that model the real world. The dominant paradigm for composing probability distributions in this way is graphical modeling [Wainwright and Jordan, 2008]; other paradigms include Bayesian networks [Pearl, 1985], Markov networks [Kindermann, 1980], and mixed graphs [Beck et al., 2013]. The benefit of composing probabilistic building blocks is that we can use probability theory to interpret and understand larger models. Contrast this with the compositionality of neural networks. While neural networks can also be readily composed from simpler building blocks, they are harder to reason about rigorously and their behavior is often less well-defined. It is not surprising, then, that a probabilistic approach is often appreciated in scientific applications, where interpretability is the goal.

2.1.2. Maximum likelihood estimation

Let's now discussion how to infer the parameters of a statistical model, sometimes called *fitting* the model. The standard approach is *maximum likelihood estimation* (MLE)[5]. The idea is to find parameters $\boldsymbol{\theta}_{\mathsf{MLE}}$ such that our observations $\mathbf{X}$ have the highest probability under the induced distribution $P_{\boldsymbol{\theta}_{\mathsf{MLE}}}$. First, we define the *likelihood function* $\mathcal{L}_N$ as the joint probability function over N independent and identically distributed (i.i.d.) observations,

$$\mathcal{L}_N(\boldsymbol{\theta}) := p(\mathbf{x}_1, \dots, \mathbf{x}_N \mid \boldsymbol{\theta}) \overset{\star}{=} \prod_{n=1}^{N} p_{\boldsymbol{\theta}}(\mathbf{x}_n), \tag{2.11}$$

where $p_{\boldsymbol{\theta}}$ is the probability function of the distribution $P_{\boldsymbol{\theta}}$. Step $\star$ holds because we assume our observations are i.i.d. In other words, if we know our model's parameters $\boldsymbol{\theta}$, we don't need to know any other observation $\mathbf{x}_m$ in order to reason about $\mathbf{x}_n$.

The MLE is the value that maximizes this likelihood function[6]:

$$\boldsymbol{\theta}_{\mathsf{MLE}} := \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathcal{L}_N(\boldsymbol{\theta}). \tag{2.12}$$

Note that Equation (2.12) does not specify *how* to calculate the maximum likelihood estimator $\boldsymbol{\theta}_{\mathsf{MLE}}$. Instead, MLE is a framework and associated theory around the estimator $\boldsymbol{\theta}_{\mathsf{MLE}}$. We can compute $\boldsymbol{\theta}_{\mathsf{MLE}}$ in many different ways, such as gradient descent [Cauchy et al., 1847], quasi-Newton methods [Broyden, 1967] such as the Broyden–Fletcher–Goldfarb–Shanno algorithm [Broyden, 1970, Fletcher, 1970, Goldfarb, 1970, Shanno, 1970], or expectation–maximization [Dempster et al., 1977].

In its modern formulation, maximum likelihood estimation has been studied by statis-

[5] To my knowledge, the "E" in "MLE" stands for either "estimation" or "estimator" depending on context.
[6] Often, we work with the log likelihood for practical reasons, such as numerical stability and ease of differentiation. I will denote the log likelihood explicitly as $\log \mathcal{L}$.

ticians for over a hundred years, at least since Ronald Fisher, and therefore has a rich and well-developed theory [Stigler et al., 2007]. Under certain regularity conditions, for example, the MLE is asymptotically efficient, meaning that in the limit of infinite data, the estimator $\boldsymbol{\theta}_{\mathsf{MLE}}$ achieves minimum possible variance. See a discussion of the Cramér–Rao lower bound in Shao [2003] for a detailed discussion.

Example: MLE for a Bernoulli model. Let's examine the MLE in a simple setting. Consider the task of estimating the bias θ of a Bernoulli process with observations $\mathbf{x} := \{x_1, \dots, x_N\}$. Let $M \geq 0$ be the number of successes. The probability of M successes in N trials is a binomial random variable, and the likelihood function is

$$p(\mathbf{x} \mid \theta) = \prod_{n=1}^{N} p_\theta(x_n) = \prod_{n=1}^{N} \binom{N}{M} \theta^{x_n}(1-\theta)^{1-x_n} = \binom{N}{M} \theta^{M}(1-\theta)^{N-M}. \tag{2.13}$$

To solve for the value of θ that maximizes our likelihood, we compute the derivative of $\log p_\theta(\mathbf{x})$ with respect to θ, set it equal to 0, and solve for θ. The derivative is

$$\begin{aligned} \frac{\partial}{\partial\theta} \log p_\theta(\mathbf{x}) &= \frac{\partial}{\partial\theta} \log \binom{N}{M} + \frac{\partial}{\partial\theta} M \log\theta + \frac{\partial}{\partial\theta}(N-M)\log(1-\theta) \\ &\propto \frac{M}{\theta} - \frac{N-M}{1-\theta}. \end{aligned} \tag{2.14}$$

The constant of proportionality is with respect to θ. The normalizer $\binom{N}{M}$ disappears since it does not effect the maximum of the function. Solving for θ when the derivative is equal to 0, we get

$$\theta_{\mathsf{MLE}} = \frac{M}{N} = \frac{1}{N}\sum_{n=1}^{N} x_n. \tag{2.15}$$

This worked, but only because we worked with a simple and analytically tractable distribution. More generally, one can compute or approximate the MLE using gradient-based optimization. See Vanderbei et al. [2015] for an introduction to the optimization theory.

Example: MLE for a Gaussian model. As a second example, let's perform maximum likelihood estimation on real data, course grades with integer values in the range [0, 19] from

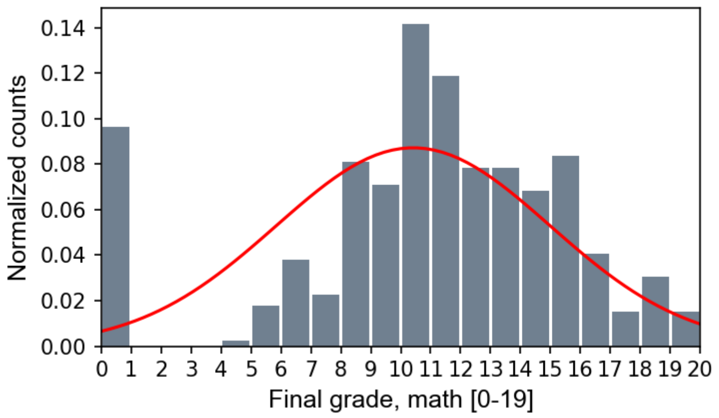

Figure 2.5: Gaussian model, $\mathcal{N}(\mathbf{x} \mid 11.9, 3.2)$, fit to final math grades from the Student Performance data set. The distribution's mode is the mean parameter's MLE.

the Student Performance data set[7]. Now let's fit a Gaussian model. This means that we want to find the parameters for the normal distribution, the mean μ and variance σ^2, that best fits the data. The log likelihood is:

$$\log \mathcal{L}_N(\mu, \sigma^2) = -\frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{n=1}^{N} (x_n - \mu)^2. \tag{2.16}$$

Again, we compute the derivative of $\log \mathcal{L}_N$ with respect to its two parameters, μ and σ^2, and then solve for μ and σ^2 when these derivatives are set to 0. It is straightforward to show that these operations produce the following estimators:

$$\begin{aligned} \mu_{\mathsf{MLE}} &\coloneqq \frac{1}{N} \sum_{n=1}^{N} x_n, \\ \sigma^2_{\mathsf{MLE}} &\coloneqq \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu_{\mathsf{MLE}})^2. \end{aligned} \tag{2.17}$$

Note that we first computed μ_{MLE} and then plugged that value into the equation for σ^2_{MLE}. Finally, we can calculate these parameters from the data and plot the resulting Gaussian distribution, $\mathcal{N}(\mathbf{x} \mid 11.9, 3.2)$ (Figure 2.5).

[7] https://archive.ics.uci.edu/ml/datasets/Student+Performance

2.2 Latent variable modeling

Now that we understand a probabilistic approach to machine learning, let's discuss a natural extension of the idea, latent variable modeling. The central assumptions of latent variable models are that complex data often exhibit simpler patterns and that these simpler patterns are not necessarily captured by statistical parameters. Consider the GMM in Section 2.1.1, for example. The generative parameters are $\boldsymbol{\theta} = \{(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\}_{k=1}^K$. However, what are the cluster assignment variables, $\mathbf{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_N\}$? We call these *latent variables*. A latent variable is an unobserved or hidden variable, as opposed to observed variables. As the GMM example illustrates, latent variables allow us to explicitly encode richer structure into our model. Again, a major benefit to latent variable models is that we can interpret the inferred latent variables in terms of an explicit mathematical model.

Perhaps the simplest latent variable model is factor analysis [Spearman, 1904, Lawley and Maxwell, 1962]:

$$\begin{aligned}
\mathbf{x}_n &= \mathbf{W}\mathbf{z}_n + \mathbf{u}_n, \\
\mathbf{z}_n &\overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}), \\
\mathbf{u}_n &\overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \boldsymbol{\Psi}).
\end{aligned} \tag{2.18}$$

Here, $\boldsymbol{\Psi}$ is a diagonal but not necessarily scalar matrix. Let $\mathbf{x}_n$ and $\mathbf{z}_n$ be D- and K-vectors respectively. Then $\mathbf{W}$ is a $D \times K$ matrix or a linear transformation from $\mathbb{R}^K$ to $\mathbb{R}^D$. As we can see, the modeling assumption is that our observations $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ are linear with respect to the latent variables $\mathbf{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_N\}$. The additive Gaussian noise $\mathbf{u}_n$ induces a Gaussian assumption on our observations,

$$\mathbf{x}_n \mid \mathbf{W}, \mathbf{z}_n, \mathbf{u}_n \sim \mathcal{N}(\mathbf{W}\mathbf{z}_n, \boldsymbol{\Psi}). \tag{2.19}$$

Each component of the low-dimensional vector $\mathbf{z}_n$ is a *factor* and can be interpreted as the low-dimensional counterpart to the data's features. Notice we have no labels $\mathbf{Y}$. Thus, this is a form of unsupervised learning, and a probabilistic version of dimension reduction. Perhaps the most common latent variable model probabilistic principal component analysis (PCA) [Tipping and Bishop, 1999], which I discuss in Section 4.3. Now we can see why this model is from the family of linear–Gaussian factor models.

2.2.1. Expectation–maximization

Inference for latent variable models is challenging, due to the conditional dependencies induced by the latent variables. To perform maximum likelihood inference, we need to compute the log likelihood $\log \mathcal{L}_N(\boldsymbol{\theta}) = \prod_{n=1}^{N} \log p_{\boldsymbol{\theta}}(\mathbf{x})$. However, our modeling assumption is that we have some latent variables $\mathbf{Z}$, which we must account for in some way. We could handle these latent variables by marginalizing them out,

$$\sum_{n=1}^{N} \log p_{\boldsymbol{\theta}}(\mathbf{x}_n) = \sum_{n=1}^{N} \sum_{\mathbf{z}} \log p_{\boldsymbol{\theta}}(\mathbf{x}_n, \mathbf{z}). \tag{2.20}$$

However, this may be intractable. For example, in the Gaussian mixture model with N observations with K mixture components, Equation (2.20) has K^N terms.

There is a standard solution to this general problem: *expectation–maximization* (EM) [Dempster et al., 1977]. EM relies on the fact that in many statistical problems, maximizing the *complete log likelihood* $\log p_{\boldsymbol{\theta}}(\mathbf{X}, \mathbf{Z})$ is actually easier than maximizing the log likelihood. So rather than optimizing Equation (2.20) directly, EM iteratively optimizes a lower bound. As we will see, this lower bound is *tight*, meaning no greater value is also a lower bound, and maximizing the lower bound guarantees that we maximize the likelihood. EM, then, is a specific algorithm for performing maximum likelihood estimation.

First, let's derive the lower bound,

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}) &= \log \sum_{\mathbf{z}} p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) \\
&= \log \sum_{\mathbf{z}} q(\mathbf{z}) \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \\
&= \log \left(\mathbb{E}_{q(\mathbf{z})} \left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right] \right) \\
&\geq \mathbb{E}_{q(\mathbf{z})} \left[\log \left(\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right) \right].
\end{aligned} \tag{2.21}$$

The inequality holds because log is a concave function, allowing us to invoke Jensen's inequality (see Section 2A.1 for a discussion) for concave functions, $\log(\mathbb{E}[a]) \geq \mathbb{E}[\log(a)]$. We can then use $\log(a/b) = \log(a) - \log(b)$ and the linearity of expectation to write,

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \underbrace{\mathbb{E}_{q(\mathbf{z})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\right]}_{\text{Expected complete LL}} \underbrace{- \mathbb{E}_{q(\mathbf{z})}\left[\log q(\mathbf{z})\right]}_{\text{Entropy of } q}. \tag{2.22}$$

Notice that we have arbitrarily introduced a density, $q(\mathbf{z})$. Which density should we choose if we want a tight lower bound? Jensen's inequality holds with equality if the concave function $f(\cdot)$ is a constant or if

$$\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} = \text{non-random}. \tag{2.23}$$

For the fraction to be a constant, we know it must be true that $q(\mathbf{z}) \propto p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$. And since $q(\mathbf{z})$ is a density and must normalize to one, we have

$$\begin{aligned} q(\mathbf{z}) &= \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{\sum_{\mathbf{z}'} p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}')} \\ &= \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})} \\ &= p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x}). \end{aligned} \tag{2.24}$$

Thus, we have found the ideal $q(\mathbf{z})$ in our lower-bound approximation of the log likelihood. We can rewrite the inequality in Equation (2.22) as an equality,

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\right] - \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})\right]. \tag{2.25}$$

We are almost ready to construct the EM algorithm. However, since EM is an iterative algorithm, let's first introduce some notation based on iteration indexes. Let $\boldsymbol{\theta}_t$ be the parameter estimates at iteration t. Then let

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(\mathbf{x}) &= \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\right] - \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})\right] \\ &:= Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t) + H(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t). \end{aligned} \tag{2.26}$$

where $Q(\cdot)$ equal to the first expectation and $H(\cdot)$ equal to the negative of the second expectation. This notation is meant to express an expectation of $\boldsymbol{\theta}$ with respect to some other parameter value $\boldsymbol{\theta}_t$.

Now carefully consider the following reasoning. By Gibbs' inequality (see MacKay [2003] for a discussion), we know that

$$H(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t) \geq H(\boldsymbol{\theta}_t \mid \boldsymbol{\theta}_t). \tag{2.27}$$

This is intuitive. The cross entropy (left-hand side) is always greater or equal to the entropy (right-hand side). (See MacKay [2003] for a discussion of information entropy.) The measure of "surprise" can only go up. Alternatively—and this is how many other authors explain EM—you can note that $H(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t) - H(\boldsymbol{\theta}_t \mid \boldsymbol{\theta}_t)$ is a Kullback–Leibler (KL) divergence (see Section 2A.2 for a discussion), which is nonnegative.

The above logic implies,

$$\begin{aligned}\log p_{\boldsymbol{\theta}}(\mathbf{x}) - \log p_{\boldsymbol{\theta}_t}(\mathbf{x}) &= Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t) - Q(\boldsymbol{\theta}_t \mid \boldsymbol{\theta}_t) + \overbrace{H(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t) - H(\boldsymbol{\theta}_t \mid \boldsymbol{\theta}_t)}^{\geq 0} \\ &\geq Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t) - Q(\boldsymbol{\theta}_t \mid \boldsymbol{\theta}_t).\end{aligned} \tag{2.28}$$

The inequality in Equation (2.28) demonstrates that if we maximize $Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)$, then we maximize the log likelihood $\log p_{\boldsymbol{\theta}}(\mathbf{x})$ by the same amount.

Thus, EM works by iteratively optimizing the expected complete log likelihood $Q(\cdot)$ rather than $\log p_{\boldsymbol{\theta}}(\mathbf{x})$. It consists of two eponymous steps:

$$\begin{aligned}\textbf{E-step:} \quad & Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t) = \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\mathbf{z} \mid \mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\right] \\ \textbf{M-step:} \quad & \boldsymbol{\theta}_{t+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t).\end{aligned} \tag{2.29}$$

The *E-step* is so-called because it constructs the *expectation* of the complete log likelihood. The *M-step* is so-called because it then *maximizes* that quantity. Intuitively we can think of the E-step as constructing the desired lower bound, and the M-step as optimizing that bound.

2.2.2. *Example: EM for factor analysis*

To illustrate the EM algorithm, I will derive the EM updates for the factor analysis model discussed in the introduction of this section. Let's also rewrite Equation (2.18) by vectorizing across N independent samples as

$$\mathbf{X} = \mathbf{W}\mathbf{Z} + \mathbf{U}, \tag{2.30}$$

where $\mathbf{X} \in \mathbb{R}^{D \times N}$, $\mathbf{Z} \in \mathbb{R}^{K \times N}$ and $\mathbf{U} \in \mathbb{R}^{P \times N}$.

To compute the EM updates, we first write down the expected complete log likelihood, only including terms that depend on $\mathbf{Z}$ since we're maximizing the parameters $\boldsymbol{\theta} = \{\mathbf{W}, \boldsymbol{\Psi}\}$ w.r.t. the log posterior $\log p_{\boldsymbol{\theta}}(\mathbf{Z} \mid \mathbf{X})$:

$$\begin{aligned}
Q &= \mathbb{E}\left[-\frac{1}{2}\sum_{n=1}^{N}(\mathbf{x}_n - \mathbf{W}\mathbf{z}_n)^\top \boldsymbol{\Psi}^{-1}(\mathbf{x}_n - \mathbf{W}\mathbf{z}_n) - \frac{1}{2}\ln|\boldsymbol{\Psi}| + C\right] \\
&\propto -\frac{1}{2}\sum_{n=1}^{N}\mathbb{E}\left[\mathbf{x}_n^\top \boldsymbol{\Psi}^{-1}\mathbf{x}_n - 2\mathbf{z}_n^\top \mathbf{W}^\top \boldsymbol{\Psi}^{-1}\mathbf{x}_n + \mathbf{z}_n^\top \mathbf{W}^\top \boldsymbol{\Psi}^{-1}\mathbf{W}\mathbf{z}_n\right] - \frac{N}{2}\ln|\boldsymbol{\Psi}|. \\
&= -\frac{1}{2}\sum_{n=1}^{N}\Bigg[\underbrace{\mathbf{x}_n^\top \boldsymbol{\Psi}^{-1}\mathbf{x}_n}_{A} - 2\underbrace{\mathbb{E}[\mathbf{z}_n]^\top \mathbf{W}^\top \boldsymbol{\Psi}^{-1}\mathbf{x}_n}_{B} + \underbrace{\operatorname{tr}\left(\mathbf{W}^\top \boldsymbol{\Psi}^{-1}\mathbf{W}\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^\top]\right))}_{C}\Bigg] \\
&\quad \underbrace{-\frac{N}{2}\ln|\boldsymbol{\Psi}|}_{D}.
\end{aligned} \tag{2.31}$$

To compute EM updates for either $\mathbf{W}$ or $\boldsymbol{\Psi}$, we compute the derivative of Equation (2.31) w.r.t. each parameter, set the equation equal to zero, and solve for the parameter. We can take derivatives of the terms labeled A, B, C, and D w.r.t. both parameters using the excellent *Matrix Cookbook* [Petersen et al., 2008]:

$$\begin{aligned}
\frac{\partial B}{\partial \mathbf{W}} &= \boldsymbol{\Psi}^{-1}\mathbf{x}_n\mathbb{E}[\mathbf{z}_n]^\top, && \text{MC.71} \\
\frac{\partial C}{\partial \mathbf{W}} &= 2\boldsymbol{\Psi}^{-1}\mathbf{W}\mathbb{E}\left[\mathbf{z}_n\mathbf{z}_n^\top\right], && \text{MC.117}
\end{aligned} \tag{2.32}$$

and

$$\begin{aligned} \frac{\partial A}{\partial \mathbf{\Psi}^{-1}} &= \mathbf{x}_n \mathbf{x}_n^\top, && \text{MC.72} \\ \frac{\partial B}{\partial \mathbf{\Psi}^{-1}} &= \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^\top \mathbf{W}^\top, && \text{MC.70} \\ \frac{\partial C}{\partial \mathbf{\Psi}^{-1}} &= \mathbf{W} \mathbb{E}\left[\mathbf{z}_n \mathbf{z}_n^\top\right] \mathbf{W}^\top, && \text{MC.102} \\ \frac{\partial D}{\partial \mathbf{\Psi}^{-1}} &= -\frac{N}{2} \mathbf{\Psi}. && \text{MC.56} \end{aligned} \tag{2.33}$$

MC.X denotes Equation X in Petersen et al. [2008]. First, we can solve for the optimal $\mathbf{W}^\star$:

$$\begin{aligned} \frac{\partial Q}{\partial \mathbf{W}} &= -\sum_{n=1}^{N} \mathbf{\Psi}^{-1} \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^\top + \sum_{n=1}^{N} \mathbf{\Psi}^{-1} \mathbf{W} \mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top], \\ &\Downarrow \\ \mathbf{W}^\star &= \left(\sum_{n=1}^{N} \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^\top \right) \left(\sum_{n=1}^{N} \mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top] \right)^{-1}. \end{aligned} \tag{2.34}$$

Next, we plug $\mathbf{W}^\star$ into the expected complete log likelihood in Equation (2.31) and solve for $\mathbf{\Psi}$:

$$\begin{aligned} \frac{\partial Q}{\partial \mathbf{\Psi}^{-1}} &= -\frac{1}{2} \sum_{n=1}^{N} \left[\mathbf{x}_n \mathbf{x}_n^\top - 2 \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^\top [\mathbf{W}^\star]^\top + \mathbf{W}^\star \mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top] [\mathbf{W}^\star]^\top \right] + \frac{N}{2} \mathbf{\Psi}, \\ &\Downarrow \\ \mathbf{\Psi}^\star &= \frac{1}{N} \sum_{n=1}^{N} \left[\mathbf{x}_n \mathbf{x}_n^\top - 2 \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^\top [\mathbf{W}^\star]^\top + \mathbf{W}^\star \mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top] [\mathbf{W}^\star]^\top \right] \\ &= \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^\top - \frac{1}{N} \sum_{n=1}^{N} 2 \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^\top [\mathbf{W}^\star]^\top + \frac{1}{N} \sum_{n=1}^{N} \mathbf{W}^\star \mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top] [\mathbf{W}^\star]^\top. \end{aligned} \tag{2.35}$$

We can simplify Equation (2.35) with the following observation based on the equality of Equation (2.34):

$$\frac{1}{N} \sum_{n=1}^{N} \mathbf{W}^\star \mathbb{E}[\mathbf{z}_n \mathbf{z}_n^\top] [\mathbf{W}^\star]^\top = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^\top [\mathbf{W}^\star]^\top. \tag{2.36}$$

So we can simplify Equation (2.35) as

$$\mathbf{\Psi}^\star = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^\top - \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^\top [\mathbf{W}^\star]^\top. \tag{2.37}$$

To derive the posterior moments in Equation (2.32) and Equation (2.33), note that the Gaussian assumptions on both $\mathbf{z}_n$ and $\mathbf{u}_n$ induce a conditionally Gaussian assumption on $\mathbf{x}_n$,

$$\mathbf{x}_n \mid \mathbf{W}, \mathbf{z}_n, \mathbf{u}_n \sim \mathcal{N}(\mathbf{W}\mathbf{z}_n, \boldsymbol{\Psi}), \tag{2.38}$$

Since our prior on $\mathbf{z}_n$ is also Gaussian, the posterior is Gaussian:

$$\mathbf{z}_n \mid \mathbf{x}_n \sim \mathcal{N}(\mathbf{M}\mathbf{W}^\top \boldsymbol{\Psi}^{-1}\mathbf{x}_n, \mathbf{M}), \tag{2.39}$$

where

$$\mathbf{M} = (\mathbf{I} + \mathbf{W}^\top \boldsymbol{\Psi}^{-1}\mathbf{W})^{-1}. \tag{2.40}$$

See Bishop [2006] for details on the Gaussian distribution. We can use the covariance and mean to solve for the second moment:

$$\mathbb{E}\left[\mathbf{z}_n\mathbf{z}_n^\top\right] = \mathbf{M} + \mathbb{E}[\mathbf{z}_n]\mathbb{E}[\mathbf{z}_n]^\top, \tag{2.41}$$

where the first moment is clearly

$$\mathbb{E}[\mathbf{z}_n] = \mathbf{M}\mathbf{W}^\top \boldsymbol{\Psi}^{-1}\mathbf{x}_n. \tag{2.42}$$

Now the EM updates in Equation (2.34) and Equation (2.37) match those given in Ghahramani et al. [1996]. However, we can vectorize them as follows. First, let's define the vectorized quantity $\mathbf{S}$ as

$$\mathbf{S} \coloneqq \left[\begin{array}{c|c|c} \mathbb{E}[\mathbf{z}_1] & \dots & \mathbb{E}[\mathbf{z}_N] \end{array}\right] = \mathbf{M}\mathbf{W}^\top \boldsymbol{\Psi}^{-1}\mathbf{X}. \tag{2.43}$$

So $\mathbf{S} \in \mathbb{R}^{K \times N}$ is a matrix of first posterior moments. Then the vectorized EM updates are

$$\begin{aligned}
\mathbf{W}^\star &= \left(\sum_{n=1}^N \mathbf{x}_n \mathbb{E}[\mathbf{z}_n]^\top\right)\left(\sum_{n=1}^N \mathbb{E}[\mathbf{z}_n\mathbf{z}_n^\top]\right)^{-1} \\
&= \left(\sum_{n=1}^N \mathbf{x}_n\mathbf{x}_n^\top \boldsymbol{\Psi}^{-1}\mathbf{W}\mathbf{M}\right)\left(\sum_{n=1}^N \left[\mathbf{M} + \mathbf{M}\mathbf{W}^\top\boldsymbol{\Psi}^{-1}\mathbf{x}_n\mathbf{x}_n^\top\boldsymbol{\Psi}^{-1}\mathbf{W}\mathbf{M}\right]\right)^{-1} \\
&= \mathbf{X}\mathbf{S}^\top(N\mathbf{M} + \mathbf{S}\mathbf{S}^\top)^{-1},
\end{aligned} \tag{2.44}$$

and

$$
\begin{aligned}
\boldsymbol{\Psi}^{\star} &= \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^{\top} - \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n\mathbb{E}[\mathbf{z}_n]^{\top}[\mathbf{W}^{\star}]^{\top} \\
&= \tilde{\boldsymbol{\Sigma}} - \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n\mathbf{x}_n^{\top}\boldsymbol{\Psi}^{-1}\mathbf{W}\mathbf{M}[\mathbf{W}^{\star}]^{\top} \\
&= \tilde{\boldsymbol{\Sigma}} - \tilde{\boldsymbol{\Sigma}}\boldsymbol{\Psi}^{-1}\mathbf{W}\mathbf{M}[\mathbf{W}^{\star}]^{\top},
\end{aligned} \tag{2.45}
$$

where $\tilde{\boldsymbol{\Sigma}} = (N-1)^{-1}\mathbf{X}\mathbf{X}^{\top}$. In summary, we can write the EM algorithm for factor analysis as:

$$
\begin{aligned}
&\text{E-step:} \\
&\qquad \mathbf{M} = (\mathbf{I} + \mathbf{W}^{\top}\boldsymbol{\Psi}^{-1}\mathbf{W})^{-1}, \\
&\qquad \mathbf{S} = \mathbf{M}\mathbf{W}^{\top}\boldsymbol{\Psi}^{-1}\mathbf{X}, \\
&\qquad \mathbf{A} = N\mathbf{M} + \mathbf{S}\mathbf{S}^{\top}. \\
&\text{M-step:} \\
&\qquad \mathbf{W}^{\star} = \mathbf{X}\mathbf{S}^{\top}\mathbf{A}^{-1}, \\
&\qquad \boldsymbol{\Psi}^{\star} = \operatorname{diag}\left(\tilde{\boldsymbol{\Sigma}} - \tilde{\boldsymbol{\Sigma}}\boldsymbol{\Psi}^{-1}\mathbf{W}\mathbf{M}[\mathbf{W}^{\star}]^{\top}\right).
\end{aligned} \tag{2.46}
$$

We need the diagonalization operator because when finding the optimal value of $\boldsymbol{\Psi}$ since we only care about the diagonal elements. In other words, this is a logical rather than linear algebraic constraint.

Summary. EM is an iterative algorithm for estimating the parameters in a latent variable model. It works by constructing a lower bound of model's likelihood and then optimizing that bound. See Wu [1983] for a proof of convergence of this algorithm. EM is also related to another common technique for approximate inference in Bayesian models (discussed next), called *variational inference* (VI). For completeness, I have included a short description of VI in Section 2A.3. See Blei et al. [2017] for a more thorough introduction.

2.3 Bayesian modeling

The second extension to the basic probabilistic framework used in this thesis is the Bayesian philosophy of inferential statistics. To motivate a Bayesian approach, consider the following low-data scenario. Imagine we flip a coin three times and observe three heads in a row. Assume heads is denoted with unity and tails is denoted with zero. Is the coin biased? The maximum likelihood estimate of the bias parameter, given by Equation (2.15), is $\theta_{\mathsf{MLE}} = (3/3)\sum_{n=1}^{3} x_n = 1$. However, intuitively, this is overfitting because we have prior knowledge that most coins are fair. The Bayesian statistician resolves this issue by assuming that the parameter $\boldsymbol{\theta}$ is itself a random variable and to then apply Bayes' theorem.

For any two events $\mathcal{A}$ and $\mathcal{B}$, Bayes' theorem [Bayes, 1763, Laplace, 1820] can be derived from the definition of joint probability. Since

$$\mathbb{P}(\mathcal{A}, \mathcal{B}) = \mathbb{P}(\mathcal{B}, \mathcal{A}) \quad \Longrightarrow \quad \mathbb{P}(\mathcal{A} \mid \mathcal{B})\mathbb{P}(\mathcal{B}) = \mathbb{P}(\mathcal{B} \mid \mathcal{A})\mathbb{P}(\mathcal{A}), \tag{2.47}$$

then we can divide both sides in Equation (2.47) by $\mathbb{P}(\mathcal{A})$, provided $\mathbb{P}(\mathcal{A}) \neq 0$, to get

$$\mathbb{P}(\mathcal{B} \mid \mathcal{A}) = \frac{\mathbb{P}(\mathcal{A} \mid \mathcal{B})\mathbb{P}(\mathcal{B})}{\mathbb{P}(\mathcal{A})}, \qquad \mathbb{P}(\mathcal{A}) \neq 0. \tag{2.48}$$

Often, the denominator in Equation (2.48) is rewritten via marginalization:

$$\mathbb{P}(\mathcal{A}) = \sum_{\mathcal{B}'} \mathbb{P}(\mathcal{A} \mid \mathcal{B}')\mathbb{P}(\mathcal{B}'). \tag{2.49}$$

Equation (2.48) quantifies the probability of event $\mathcal{B}$ given event $\mathcal{A}$ in terms of the probability of $\mathcal{A}$ given $\mathcal{B}$ and *prior probabilities.* We refer to $\mathbb{P}(\mathcal{A})$ and $\mathbb{P}(\mathcal{B})$ as prior probabilities because they specify the probability of events $\mathcal{A}$ and $\mathcal{B}$ without conditioning on any other event. In words, we might say: what we know about $\mathcal{B}$ given $\mathcal{A}$ depends on what we knew beforehand about $\mathcal{A}$ and $\mathcal{B}$, as well as how $\mathcal{A}$ depends on $\mathcal{B}$. See Section 1.2 in Bishop [2006] for a more detailed discussion.

For our purposes, Bayes' theorem provides a principled way to do inference when our parameters $\boldsymbol{\theta}$ are random variables. Since $\boldsymbol{\theta}$ is random, we must assume a distribution Π for $\boldsymbol{\theta}$, called the *prior* distribution. We can then use Bayes' theorem to compute a probability function $\pi(\boldsymbol{\theta} \mid \mathbf{X})$ which specifies how likely $\boldsymbol{\theta}$ is given our observations $\mathbf{X}$. This probability function is associated with a distribution, called the *posterior* distribution, which I'll denote as $\Pi_{\boldsymbol{\theta}|\mathbf{X}}$. Thus, rather than inferring a *point estimate* or single value for $\boldsymbol{\theta}$, such as $\boldsymbol{\theta}_{\mathsf{MLE}}$, we infer a distribution over $\boldsymbol{\theta}$ that depends on the data we observe. The numerator in Bayes' theorem is then just our likelihood (Equation (2.11)) times our prior probability function. The denominator is the probability of our data independent of the parameters and is often called the *evidence*. The evidence is typically ignored since the posterior mode does not depend on this constant or *normalizer*, since it ensures the probability distribution integrates or sums to unity.

Putting this all together, we have

$$\overbrace{\pi(\boldsymbol{\theta} \mid \mathbf{x}_1, \dots, \mathbf{x}_N)}^{\text{Posterior}} = \frac{\overbrace{\prod_{n=1}^{N} p_{\boldsymbol{\theta}}(\mathbf{x}_n)}^{\text{Likelihood}} \; \overbrace{\pi_{\boldsymbol{\alpha}}(\boldsymbol{\theta})}^{\text{Prior}}}{\underbrace{p(\mathbf{x}_1, \dots, \mathbf{x}_N)}_{\text{Evidence}}}. \tag{2.50}$$

The prior distribution has its own parameters, which we refer to as *hyperparameters* and which I've denoted with $\boldsymbol{\alpha}$ here. Hyperparameters are assumed, not inferred.

Because we have inferred a distribution over our parameters, we must account for uncertainty about the actual value. The probabilist's tool for dealing with uncertainty is marginalization. Thus, Bayesian modeling is also concerned with a variety of integrals or expectations. The *marginal likelihood* is the likelihood function with the parameters marginalized out:

$$p_{\boldsymbol{\alpha}}(\mathbf{X}) = \int_{\boldsymbol{\Theta}} \left[\prod_{n=1}^{N} p_{\boldsymbol{\theta}}(\mathbf{x}_n) \right] \pi_{\boldsymbol{\alpha}}(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}. \tag{2.51}$$

Intuitively, Equation (2.51) captures the likelihood of the data without considering the model-specific parameters. This has a variety of applications, such providing the Bayesian statistician the tools to compare two models without considering the parameters. (See the discussion of *Bayesian model comparison* in [Gelman et al., 2013] for details.) The *posterior predictive* is the distribution over unobserved data given observed data:

$$p(\mathbf{x}_{N+1} \mid \mathbf{X}) = \int_{\boldsymbol{\Theta}} p_{\boldsymbol{\theta}}(\mathbf{x}_{N+1}) \pi_{\boldsymbol{\alpha}}(\boldsymbol{\theta} \mid \mathbf{X}) \mathrm{d}\boldsymbol{\theta}. \tag{2.52}$$

Notice that we do not condition on $\mathbf{X}$ in the term $p_{\boldsymbol{\theta}}(\mathbf{x}_{N+1})$. This is because the standard assumption is that unseen data $\mathbf{x}_{N+1}$ is conditionally independent of everything else given the generative parameters $\boldsymbol{\theta}$. The posterior predictive distribution underscores the value of modeling uncertainty in the parameter estimate. By integrating out $\boldsymbol{\theta}$, we weigh the statistical model $p_{\boldsymbol{\theta}}(\mathbf{x}_{N+1})$ by every possible value of $\boldsymbol{\theta}$ using the posterior distribution's probability function.

Bayesian inference can be challenging because the posterior distribution (Equation (2.50)) may have no analytical form. If we multiply two probability functions together, it is not obvious what the normalizing constant of the resultant unnormalized function is. And even when the posterior distribution does have an analytical form, this may not be the case for the marginal likelihood (Equation (2.51)) or the posterior predictive (Equation (2.52)) distributions.

Given these challenges, Bayesian inference can be grouped into three broad categories: *maximum a posteriori* estimation (Section 2.3.1), which is the Bayesian analog to the MLE; exact inference, which typically requires a property called *conjugacy* (Section 2.3.3); and approximate inference, which requires numerical approximations such as Markov chain Monte Carlo (Section 2.3.4) or variational inference (Section 2A.3). I introduce these and related ideas in this order next.

2.3.1. Maximum a posterior estimation

Recall that the MLE of a parameter is simply the parameter value that maximizes the likelihood function:

$$\boldsymbol{\theta}_{\mathsf{MLE}} := \operatorname*{argmax}_{\boldsymbol{\theta}} \mathcal{L}_N(\boldsymbol{\theta}). \tag{2.53}$$

However, in a Bayesian approach, $\boldsymbol{\theta}$ is random, and we account for this through a prior distribution $\Pi_{\boldsymbol{\theta}}$ with density function $\pi(\boldsymbol{\theta})$. The *maximum a posteriori* (MAP) estimate is

$$\begin{aligned} \boldsymbol{\theta}_{\mathsf{MAP}} &:= \operatorname*{argmax}_{\boldsymbol{\theta}} \pi(\boldsymbol{\theta} \mid \mathbf{X}) \\ &\propto \operatorname*{argmax}_{\boldsymbol{\theta}} \mathcal{L}_N(\boldsymbol{\theta})\pi(\boldsymbol{\theta}), \end{aligned} \tag{2.54}$$

where proportionality is w.r.t. $\boldsymbol{\theta}$. We can drop the evidence because it is always positive and does not depend on $\boldsymbol{\theta}$. It is constant for all possible values of $\boldsymbol{\theta}$ and can be ignored in any method that optimizes Equation (2.54).

Both the MLE and MAP estimates are point estimates. One view of the MAP estimate is that it is the MLE regularized by the prior Π. However, a MAP estimate does not give us uncertainty quantification of $\boldsymbol{\theta}$. Therefore, all things being equal, we would prefer to use exact or approximate inference to infer the full posterior distribution $\pi(\boldsymbol{\theta} \mid \mathbf{X})$. Exact inference is possible when we use a conjugate prior, which I'll discuss next.

2.3.2. *Conjugate priors*

Consider a prior distribution Π. If prior's density function $\pi(\boldsymbol{\theta})$ times the likelihood $\mathcal{L}_N(\boldsymbol{\theta})$ is the same functional form as the posterior's density function $\pi(\boldsymbol{\theta} \mid \mathbf{X})$, the prior is called a *conjugate prior*. This means that the two boxed terms in Bayes' formula below have the same functional form:

$$\boxed{\pi(\boldsymbol{\theta} \mid \mathbf{X})} \propto \mathcal{L}_N(\boldsymbol{\theta}) \boxed{\pi(\boldsymbol{\theta})}. \tag{2.55}$$

Conjugacy is a desirable property because it implies that the posterior has an analytical form.

Furthermore, conjugacy allows for simple and interpretable Bayesian updating. First, we begin with a prior $\pi(\boldsymbol{\theta})$. Then we observe a set of N observations $\mathbf{X}$. We analytically infer a posterior $\pi(\boldsymbol{\theta} \mid \mathbf{X})$. Now imagine we observe $\mathbf{x}_{N+1}$. Since the posterior is the same functional form as the prior, we simply update the posterior in the same way that we updated the prior, resulting in a new posterior $\pi(\boldsymbol{\theta} \mid \mathbf{X}, \mathbf{x}_{N+1})$.

Example: Beta–Bernoulli model. Let's look at an example. Recall the task of inferring the bias of a Bernoulli distribution (Section 2.1.2). We said that the MLE was problematic in low-data scenarios because it could easily overfit. The Bayesian approach is to treat the bias as a random variable and infer a posterior distribution rather than a point estimate.

What sort of prior should we place on the Bernoulli bias θ? We need a distribution with support over the range $[0, 1]$ and that can be shaped via hyperparameters. (Contrast this

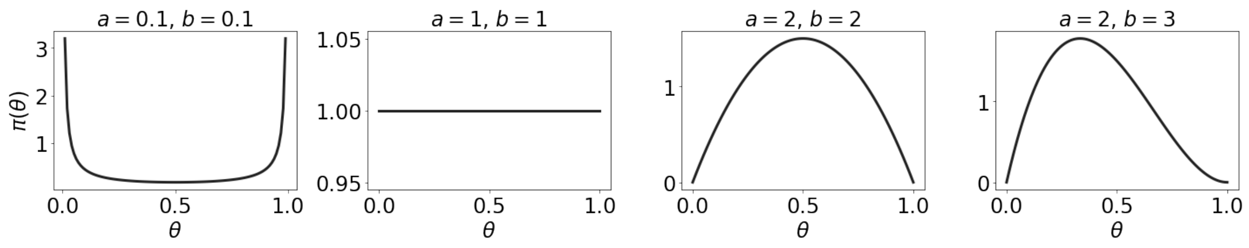

Figure 2.6: The beta distribution Beta(a, b) for a variety of hyperparameters a and b.

idea with the uniform distribution over $[0, 1]$.) One distribution that has these properties is the beta distribution (Figure 2.6), given by

$$\text{Beta}(\mu \mid a, b) = \frac{1}{\text{B}(\alpha, \beta)} \theta^{a-1}(1-\theta)^{b-1}. \tag{2.56}$$

$\text{B}(\cdot)$ is the beta function,

$$\text{B}(\alpha, \beta) := \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}, \tag{2.57}$$

where $\Gamma(\cdot)$ is the gamma function,

$$\Gamma(x) := \int_0^\infty \theta^{x-1} e^{-\theta} \text{d}\theta. \tag{2.58}$$

At a high level, the gamma function can be viewed as a factorial function for complex numbers. Intuitively, it appears in the normalizer for a variety of distributions because the normalizing constant often requires some amount of bookkeeping as to the number of possible outcomes.

Now let's see how the beta distribution is a conjugate prior for the Bernoulli likelihood. If we multiply our likelihood (Equation (2.13)) by our prior (Equation (2.56)), we get a posterior that has the same functional form as the prior:

$$\begin{aligned} \pi(\theta \mid \mathbf{X}) &\propto \mathcal{L}_N(\theta)\pi(\theta) \\ &= \left(\prod_{n=1}^{N} \theta^{x_n}(1-\theta)^{1-x_n}\right)\left(\frac{1}{\text{B}(\alpha, \beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}\right) \\ &\propto \theta^{\sum_n x_n + \alpha - 1}(1-\theta)^{N - \sum_n x_n + \beta - 1}. \end{aligned} \tag{2.59}$$

The first constant of proportionality is the evidence, $p(\mathbf{X})$. The second constant of proportionality is the normalizer $1/\mathrm{B}(\alpha, \beta)$. Both can be dropped because they do not depend on θ. Thus, we see that our posterior is another beta distribution:

$$\begin{aligned}\pi(\theta \mid \mathbf{X}) &= \text{Beta}(\alpha_N, \beta_N), \\ \alpha_N &:= \sum_{n=1}^{N} x_n + \alpha, \\ \beta_N &:= N - \sum_{n=1}^{N} x_n + \beta.\end{aligned} \tag{2.60}$$

This example underscores how we can interpret the posterior in certain simple settings. In this case, the posterior density $\pi(\theta \mid \mathbf{X})$ is the same functional form as the prior density $\pi(\theta)$. Both are density functions for beta distributions. The only difference between the two distributions is that the posterior's parameters account for the data. $\sum_n x_n$ is the number of successes (ones), while $N - \sum_n x_n$ is the number of failures (zeros). The hyperparameters α and β therefore capture prior knowledge about the *a priori* number of successes or failures. As we observe more data, the effect of the posterior updates "overwhelms" the prior's hyperparameters.

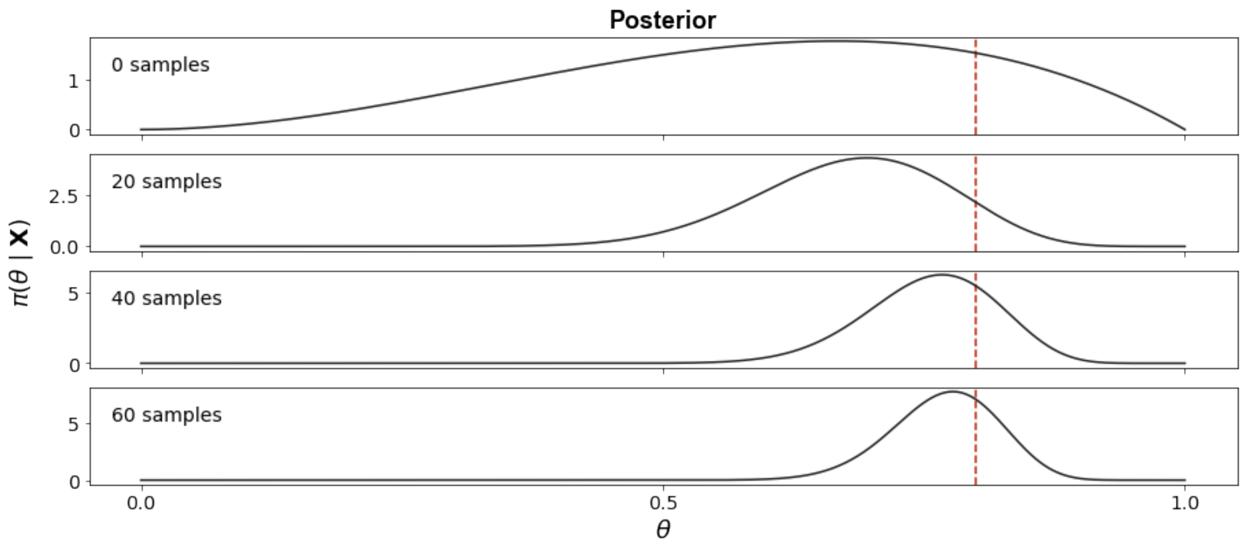

Figure 2.7: Visualizing the posterior distribution for n samples with $n \in \{0, 10, 20, 30\}$. As the model accounts for more data, the posterior places more density around the true bias from the generative process. And after each datum, the posterior becomes the new prior. Note that the y-axes are at different scales and therefore the last frame is even more peaked than the second-to-last frame.

To see this in detail, consider a sequential setting, where we observe each datum in $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ in order, updating our posterior after each observation. Here, the posterior at step n becomes the prior at step $n+1$. For example, imagine our model processes coin flips one at a time. On the nth coin flip, we see a success or failure and increment our posterior parameters at step n, α_n and β_n. Ideally, as we observe more data, the posterior *concentrates* or becomes more peaked around the true parameter value (Figure 2.7).[8]

Now that we understand conjugacy, let's address the problem of overfitting discussed in Section 2.1.2. Does our Bayesian model resolve the issue of overfitting and if so, how? Because this new posterior is in a tractable form, it is straightforward to compute θ_{MAP}. To do this, we could compute the derivative of the posterior, set it equal to zero, and then solve for θ_{MAP}. However, we know that the posterior is a beta distribution with parameters α_N and β_N as defined in Equation (2.60). The beta distribution has a single mode:

$$\frac{\alpha_N - 1}{\alpha_N + \beta_N - 2}. \tag{2.61}$$

Notice that if the number of successes and failures are both zero, if we have observed no data, the MAP estimate is

$$\theta_{\mathsf{MAP}} = \frac{\alpha - 1}{\alpha + \beta - 2}. \tag{2.62}$$

For example, if $\alpha = \beta = 2$, then the MAP estimate without data is $1/2$. (In other words, when $\alpha = \beta = 2$, the prior is that most coins are fair.) However, what about the case where we observe three successes or heads in a row? Then the MAP estimate is

$$\theta_{\mathsf{MAP}} = \frac{3 + \alpha - 1}{3 + \alpha + \beta - 2} = \frac{4}{5}. \tag{2.63}$$

Thus, despite seeing three heads in a row, the MAP estimate does not overfit to $\theta_{\mathsf{MAP}} = 1$. The reason why is simply numeric: the prior's hyperparameters prevent this. We need to see more data before we overwhelm the prior values. This example demonstrates why the prior is especially important for parameter estimation with small data and how it helps to prevent overfitting.

[8] A large body of work in Bayesian statistics is concerned with how and when the posterior distribution concentrates around the true parameter value as the model conditions on more data.

As a final note, conjugacy often allows for simple forms for the marginal likelihood (Equation (2.51)) and posterior predictive (Equation (2.52)) distributions. In the beta–Bernoulli example, these two distributions are relatively easy to compute. To compute the marginal likelihood, we just leverage the integral definition of the beta function:

$$\begin{aligned}
p(\mathbf{X}) &= \int_0^1 \left[\prod_{n=1}^{N} p_\theta(x_n)\right] \pi(\theta)\mathrm{d}\theta \\
&= \int_0^1 \frac{1}{\mathrm{B}(\alpha,\beta)} \theta^{\sum_n x_n+\alpha-1}(1-\theta)^{N-\sum_n x_n+\beta-1}\mathrm{d}\theta \\
&= \frac{1}{\mathrm{B}(\alpha,\beta)} \int_0^1 \theta^{\alpha_N-1}(1-\theta)^{\beta_N-1}\mathrm{d}\theta \\
&= \frac{\mathrm{B}(\alpha_N,\beta_N)}{\mathrm{B}(\alpha,\beta)}.
\end{aligned} \tag{2.64}$$

To compute the posterior predictive over an unseen observation x_{N+1}, we integrate out our uncertainty about the parameters θ. The easiest way to compute this is to just use our previous derivation for the marginal likelihood. Since the prior and posterior are both beta distributed, we know this should work:

$$\begin{aligned}
p(x_{N+1} \mid \mathbf{X}) &= \int_0^1 p_\theta(x_{N+1})\pi(\theta \mid \mathbf{X})\mathrm{d}\theta \\
&= \frac{1}{\mathrm{B}(\alpha_N,\beta_N)} \int_0^1 \theta^{\hat{x}+\alpha_N-1}(1-\theta)^{1-x_{N+1}+\beta_N-1}\mathrm{d}\theta \\
&= \frac{\mathrm{B}(x_{N+1}+\alpha_N, 1-x_{N+1}+\beta_N)}{\mathrm{B}(\alpha_N,\beta_N)}.
\end{aligned} \tag{2.65}$$

2.3.3. Exponential family

Given the importance of conjugacy in exact Bayesian inference, it is useful to know when this property holds. A family of distributions that all have conjugate priors, as well as a variety of other important properties, is the *exponential family* [Pitman, 1936, Koopman, 1936, Darmois, 1935, Wainwright and Jordan, 2008]. Many commonly used distributions are members of the exponential family, such as the Gaussian, exponential, gamma, chi-squared, beta, Dirichlet, Bernoulli, categorical, Poisson, Wishart, inverse Wishart, and geometric distributions. The basic idea of the exponential family is that any distribution that can be expressed in exponential family form has some nice mathematical properties.

The general form for any member of the exponential family is

$$p_{\boldsymbol{\eta}}(\mathbf{x}) = h(\mathbf{x})g(\boldsymbol{\eta}) \exp\left\{\boldsymbol{\eta}^\top u(\mathbf{x})\right\}, \tag{2.66}$$

where, in exponential family terminology, $\boldsymbol{\eta}$ is the *natural parameter*, $h(\mathbf{x})$ is the *underlying measure*, $u(\mathbf{x})$ is the *sufficient statistic* of the data, and $g(\boldsymbol{\eta})$ is the *normalizer*, ensuring that

$$g(\boldsymbol{\eta}) \int h(\mathbf{x}) \exp\left\{\boldsymbol{\eta}^\top u(\mathbf{x})\right\} \mathrm{d}\mathbf{x} = 1. \tag{2.67}$$

Equation (2.66) is sometimes written with the normalizer pushed inside the exponent. In this case, the term is called the *log normalizer*. Let's look at an example.

Example: Poisson distribution. Let x be a Poisson random variable with parameter $\theta > 0$. The probability function of the Poisson is

$$p_\theta(x) = \frac{e^{-\theta}\theta^x}{x!} \tag{2.68}$$

and can be written in exponential family form as

$$\begin{aligned} p_\theta(x) &= \frac{1}{x!}\exp(-\theta)\exp(x\log\theta) \\ &= h(x)g(\eta)\exp(\eta x) \end{aligned} \tag{2.69}$$

where

$$\begin{aligned} \eta = \log\mu, \quad h(x) &= \frac{1}{x!}, \\ u(x) = x, \qquad g(\eta) &= \exp(-\exp(\eta)). \end{aligned} \tag{2.70}$$

Sufficient statistics. Why is $u(\mathbf{x})$ called the *sufficient statistic*? Loosely speaking, $\sum_i u(\mathbf{x}_i)$ is all we need to compute the MLE for the natural parameters. To see this, let's maximize the log likelihood. For any distribution in exponential family form, the likelihood is

$$\mathcal{L}_N(\boldsymbol{\eta}) = \prod_{n=1}^N p_{\boldsymbol{\eta}}(\mathbf{x}_n) = \prod_{n=1}^N h(\mathbf{x}_n)g(\boldsymbol{\eta})\exp\left\{\boldsymbol{\eta}^\top u(\mathbf{x}_n)\right\}. \tag{2.71}$$

And the log likelihood is

$$\log \mathcal{L}_N(\boldsymbol{\eta}) = \log \left(\prod_{n=1}^{N} h(\mathbf{x}_n) \right) + N \log g(\boldsymbol{\eta}) + \boldsymbol{\eta}^\top \sum_{n=1}^{N} u(\mathbf{x}_n). \tag{2.72}$$

To compute $\boldsymbol{\eta}_{\mathsf{MLE}}$, we compute the derivative of Equation (2.72), set it equal to 0, and solve for $\boldsymbol{\eta}$:

$$\begin{aligned}
\nabla \log p(\mathbf{X} \mid \boldsymbol{\eta}) &= \nabla N \log g(\boldsymbol{\eta}) + \nabla \boldsymbol{\eta}^\top \sum_{n=1}^{N} u(\mathbf{x}_n) \\
&\Downarrow \\
-\nabla \log g(\boldsymbol{\eta}) &= \nabla \boldsymbol{\eta}^\top \frac{1}{N} \sum_{n=1}^{N} u(\mathbf{x}_n) \\
-\nabla \log g(\boldsymbol{\eta}_{\mathsf{MLE}}) &= \frac{1}{N} \sum_{n=1}^{N} u(\mathbf{x}_n)
\end{aligned} \tag{2.73}$$

Thus, the optimal natural parameters $\boldsymbol{\eta}_{\mathsf{MLE}}$ only depend upon a function of $\sum_n u(\mathbf{x}_n)$. This all works because, by construction, $g(\boldsymbol{\eta})$ has no dependence on $\mathbf{X}$.

Conjugate priors. Every exponential family member has a conjugate prior of the form

$$\pi_{\boldsymbol{\chi},\nu}(\boldsymbol{\eta}) = f(\boldsymbol{\chi}, \nu) g(\boldsymbol{\eta})^{\nu} \exp\left\{\boldsymbol{\eta}^\top \boldsymbol{\chi}\right\}, \tag{2.74}$$

where ν and $\boldsymbol{\chi}$ are hyperparameters and $f(\boldsymbol{\chi}, \nu)$ depends on the form of the exponential family member. To verify conjugacy, recall that we must verify that the posterior has the same functional form as the prior. We can see that this holds:

$$\begin{aligned}
&\pi(\boldsymbol{\eta} \mid \mathbf{X}, \boldsymbol{\chi}, \nu) \\
&\propto \pi_{\boldsymbol{\chi},\nu}(\boldsymbol{\eta}) \prod_{n=1}^{N} p_{\boldsymbol{\eta}}(\mathbf{x}_n) \\
&= \left[\left(\prod_{n=1}^{N} h(\mathbf{x}_n) \right) g(\boldsymbol{\eta})^N \exp\left\{ \boldsymbol{\eta}^\top \sum_{n=1}^{N} u(\mathbf{x}_n) \right\} \right] \left[f(\boldsymbol{\chi}, \nu) g(\boldsymbol{\eta})^{\nu} \exp\left\{\boldsymbol{\eta}^\top \boldsymbol{\chi}\right\} \right] \\
&= \left(\prod_{n=1}^{N} h(\mathbf{x}_n) \right) f(\boldsymbol{\chi}, \nu) g(\boldsymbol{\eta})^{N+\nu} \exp\left\{ \boldsymbol{\eta}^\top \sum_{n=1}^{N} u(\mathbf{x}_n) + \boldsymbol{\eta}^\top \boldsymbol{\chi} \right\}.
\end{aligned} \tag{2.75}$$

Since the first $N+1$ terms are constant w.r.t. $\boldsymbol{\eta}$, we can write this as

$$\pi_{\boldsymbol{\chi},\nu}(\boldsymbol{\eta} \mid \mathbf{X}) \propto g(\boldsymbol{\eta})^{N+\nu} \exp\left\{\boldsymbol{\eta}^{\top}\left(\sum_{n=1}^{N} u(\mathbf{x}_n) + \boldsymbol{\chi}\right)\right\}. \tag{2.76}$$

Note that this posterior (Equation (2.76)) is the same exponential family form as the prior (Equation (2.74)), just with different hyperparameters:

$$\begin{aligned} \nu &\coloneqq \nu_{\text{prior}} + N, \\ \boldsymbol{\chi} &\coloneqq \boldsymbol{\chi}_{\text{prior}} + \sum_{n=1}^{N} u(\mathbf{x}_n). \end{aligned} \tag{2.77}$$

As mentioned in Section 2.3.3, conjugacy lends itself nicely to sequential learning. Let subscript t index the parameters at time t. Then sequential updates for the exponential family are

$$\begin{aligned} \nu_t &\coloneqq \begin{cases} \nu_{\text{prior}} & \text{if } t = 0, \\ \nu_{t-1} + 1 & \text{if } t > 0, \end{cases} \\ \boldsymbol{\chi}_t &\coloneqq \begin{cases} \boldsymbol{\chi}_{\text{prior}} & \text{if } t = 0, \\ \boldsymbol{\chi}_{t-1} + u(\mathbf{x}_{t-1}) & \text{if } t > 0. \end{cases} \end{aligned} \tag{2.78}$$

2.3.4. Markov chain Monte Carlo

Let's review the logic of this section so far. In Bayesian inference, given data $\mathbf{X} \coloneqq \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ and parameters $\boldsymbol{\theta}$, the posterior

$$\pi(\boldsymbol{\theta} \mid \mathbf{X}) = \frac{\prod_{n=1}^{N} p_{\boldsymbol{\theta}}(\mathbf{x}_n)\pi_{\boldsymbol{\alpha}}(\boldsymbol{\theta})}{p(\mathbf{X})} \tag{2.79}$$

is generally unavailable in closed form. In the last section, we showed how Bayesian posterior inference is tractable when we use conjugate priors, a property available to a large class of distributions called the exponential family. However, using only conjugate priors limits the range and flexibility of the distributions at our disposal. Because of this, most contemporary Bayesian inference research is concerned with numerical methods to approximate the posterior in some way. A broad class of algorithms with good theoretical guarantees that allow for approximate inference of arbitrary densities is called *Markov chain Monte Carlo* (MCMC) methods[9]. In the next section, I introduce the basic ideas about Markov chains to understand MCMC. Please skip to Section 2.3.4 if you are familiar with Markov chains.

Markov chains. A Markov chain is a stochastic model that represents a set of states and probabilities of transitioning between states. We can "walk" the Markov chain by starting at an initial state and then changing states based on these transition probabilities. Speaking loosely, imagine a frog jumping around on a lily pad field as walking a Markov chain; the location of the frog on the field is the Markov chain's state, and the transition probabilities define how likely the frog is to jump to any other pad (state). Surprisingly, this simple model can describe a wide range of problems. Markov chains were first presented by Andrey Markov in his analysis of a poem by Eugene Onegin [Markov, 1913].

Let us formalize this concept.

Definition 2.3.1. *Let $\mathcal{D}$ be a finite set. A random process $\mathbf{x}_1, \mathbf{x}_2, \ldots$ with values in $\mathcal{D}$ is called a* Markov chain *if*

$$\mathbb{P}\{\mathbf{x}_{n+1}, \mid \mathbf{x}_n, \ldots, \mathbf{x}_0\} = \mathbb{P}\{\mathbf{x}_{n+1} \mid \mathbf{x}_n\}. \tag{2.80}$$

We can think of $\mathbf{x}_n$ as a random state at time n, and the Markovian assumption is that the probability of transitioning from $\mathbf{x}_n$ to $\mathbf{x}_{n+1}$ only depends on $\mathbf{x}_n$. In words, the future depends only on the present. A Markov chain can be defined by a transition probability matrix.

Definition 2.3.2. *Let p_{ij} be the probability of transitioning from state i to state j. The matrix $\mathbf{P} = [p_{ij}]_{i,j \in \mathcal{D}}$ is called the* transition probability matrix.

Consider a simple Markov chain modeling the weather. We assume the weather has two states: rainy and sunny. Let $\mathcal{D} = \{r, s\}$. Therefore $\mathbf{x}_n \in \mathcal{D}$ represents the weather on day n. The transition probabilities are as follows. If today is rainy (denoted r), tomorrow it is sunny

[9]This strange name is a combination of "Markov chain", which is the mathematical object used in this framework, and "Monte Carlo", the name of a famous casino in Monaco

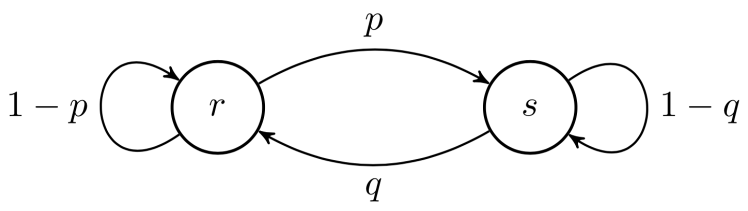

Figure 2.8: State diagram for a Markov chain with states $\mathcal{D} = \{r, s\}$. The probability of moving from r to s is p and from s to r is q. The remaining probabilities can be computed because the probabilities of the outgoing edges must sum to one.

(denoted s) with probability p. If today it is sunny, tomorrow it is rainy with probability q. The transition matrix is then

$$\mathbf{P} = \begin{array}{|c|cc|} \hline & r & s \\ \hline r & 1-p & p \\ s & q & 1-q \\ \hline \end{array} \tag{2.81}$$

Note that the rows of the matrix must sum to unity. The state diagram of this Markov chain is Figure 2.8. We are interested in *ergodicity*, which is a property of a random process in which its time average is the same as its probability space average. Formally, let the notation $\mathbb{P}_i\{\mathbf{x}_n = j\}$ mean $\mathbb{P}\{\mathbf{x}_n = j \mid \mathbf{x}_0 = i\}$. Then

Definition 2.3.3. *A Markov chain* $\mathbf{x}_1, \mathbf{x}_2, \ldots$ *is called* ergodic *if the limit*

$$\boldsymbol{\pi}(j) \coloneqq \lim_{n \to \infty} \mathbb{P}_i\{\mathbf{x}_n = j\} \tag{2.82}$$

exists for every state j *and does not depend on the initial state* i*. The* D*-vector* $\boldsymbol{\pi}$ *is called the* stationary probability*.*

Intuitively, the probability $\boldsymbol{\pi}(j)$ of being on state j after a long time is *independent* of the initial state i.[10] In the frog analogy, an ergodic lily pad field means that regardless of where the frog starts, they will always end up on the same pads with the same probability, *eventually*.

[10] Typically, I use Π and $\pi(\cdot)$ to denote the prior distribution and probability functions respectively. However, in this section, these will refer to the stationary distribution and probability function, as this follows convention.

Ergodicity has two conditions. If a Markov chain is *irreducible*, meaning any state is reachable from any other state, and if it *aperiodic*, meaning the same state is not reached on a fixed frequency, then it is ergodic. While we will not prove this fact, consider the following intuition[11]: imagine two frogs are jumping around on a lily pad field, and that their transition probabilities follow the same Markov chain. If the transition probabilities are such that the frogs enter no periodic loops or subsets of the field from which they cannot escape, they will eventually meet. At that point in time, since they are walking the same Markov chain, they will be forever coupled. In this romantic view, if the two frogs are hopping around on an ergodic lily pad field, they are fated to eventually meet, and after that moment, never part.

This proof technique is called *coupling*, since it allows one to "force" two unrelated variables to be related in some way. To my knowledge, it was first used by Doeblin and Fortet [1937] to prove the ergodicity of certain Markov chains.[12]

Implicit Markov chain construction. Now that we understand Markov chains, let's unpack MCMC algorithms. The first and most important MCMC algorithm is *Metropolis–Hastings*. While there are a number of other MCMC algorithms, such as Gibbs sampling and Hamiltonian Monte Carlo, Metropolis–Hastings contains all the foundational ideas required to understand MCMC. Therefore, in this section, I will introduce MCMC by introducing this algorithm.

Metropolis–Hastings (MH) is an elegant algorithm that is based on a truly deep idea.

[11] I first heard this intuition from Ramon van Handel at Princeton University.

[12] Wolfgang Doeblin had a tragic but noteworthy life. He was a born in Berlin in 1915 to Alfred and Erna Döblin. His father was a famous writer. They were Jewish-German; and Alfred and his family sans Wolfgang fled Berlin a few days after the Reichstag fire in February 1933. Wolfgang stayed behind until April to complete his gymnasium studies. The family was reunited and settled in Paris later that year; and Wolfgang continued his mathematical studies at the Sorbonne.

In the summer of 1936, Doeblin submitted his work on coupling [Doeblin and Fortet, 1937, Doeblin, 1939] for publication. In March 1938, Doeblin defended his thesis—more or less [Doeblin, 1937]—and was drafted into the French army in November. After the start of World War II in September 1939, Doeblin was moved into active duty. In April 1940, Doeblin's last mathematical note was filed with l'Academie des Sciences by Émile Borel. The note was submitted as a pli cacheté (sealed envelope), which was designed to remain sealed, granting its author intellectual priority if necessary without disclosing the work.

In 1940, Doeblin lost contact with his company while on a mission to the village Housseras. With Nazi troops a few minutes away, Doeblin burned his mathematical notes and took his own life. Doeblin's pli cacheté was opened in 2000, revealing he had already proven a result (Itô's lemma) for random processes. See Bru and Yor [2002] further discussion.

Suppose we want to sample from a *target distribution* Π^*. Assume that we can evaluate the probability function $\pi^*(\cdot)$ associated with the distribution Π^*, but we cannot sample from Π^*. MH performs a random walk according to a Markov chain whose stationary distribution is Π^*. At each step in the chain, a new state is proposed and either accepted or rejected according to a dynamically calculated probability, called the *acceptance criteria*. The Markov chain is never explicitly constructed; we cannot save the transition probability matrix to disk or print one of its rows. However, if our implicit Markov chain is ergodic and if the MH algorithm is run for long enough, then the probability of being on a given state in the chain is equal to the probability of the associated sample. Thus, walking the Markov chain and recording states is, in the long-run, like sampling from Π^*.

Consider a Markov chain with transition kernel $P(\mathbf{x}, \mathcal{A})$ where $\mathbf{x} \in \mathbb{R}^D$ and $\mathcal{A}$ is a subset of our sample space. A transition kernel is the generalization of a transition matrix from finite to infinite state-spaces. So in words, $P(\mathbf{x}, \mathcal{A})$ is a conditional distribution function of moving from $\mathbf{x}$ to a point in the set $\mathcal{A}$. Naturally, $P(\mathbf{x}, \mathbb{R}^D) = 1$ and self-loops are allowed, meaning that $P(\mathbf{x}, \{\mathbf{x}\})$ is not necessarily zero.

The stationary distribution of a Markov chain can be defined through the probability function π^* where

$$\pi^*(\mathrm{d}\mathbf{y}) = \pi(\mathbf{y})\mathrm{d}y = \int_{\mathbb{R}^D} P(\mathbf{x}, \mathrm{d}\mathbf{y})\pi(\mathbf{x})\mathrm{d}\mathbf{x}. \tag{2.83}$$

This is just the continuous state-space analog of the discrete case. In a discrete Markov chain $\{\mathbf{x}_n\}_{n=1}^{\infty}$ taking values in $\mathcal{D}$, if the transition matrix is $\mathbf{P} = [p_{ij}]_{i,j\in\mathcal{D}}$, then stationary distribution is

$$\boldsymbol{\pi}^* = \boldsymbol{\pi}^*\mathbf{P} \tag{2.84}$$

where $\boldsymbol{\pi}^*$ is a D-dimensional row vector. In words, the Markov chain has *mixed* when the probability of being on a given state no longer changes as we walk the chain. Using the romantic analogy from Section 2.3.4, the Markov chain has mixed when the two frogs meet.

The nth iterate or nth application of the transition kernel is given by

$$\begin{aligned} P^{(1)}(\mathbf{x}, \mathcal{A}) &\coloneqq P(\mathbf{x}, \mathcal{A}), \\ P^{(n)}(\mathbf{x}, \mathcal{A}) &\coloneqq \int_{\mathbb{R}^D} P^{(n-1)}(\mathbf{x}, \mathrm{d}\mathbf{y})P(\mathbf{y}, \mathcal{A}). \end{aligned} \tag{2.85}$$

As n goes to infinity, the nth iterate converges to the stationary distribution or

$$\pi^*(\mathcal{A}) = \lim_{n\to\infty} P^{(n)}(\mathbf{x}, \mathcal{A}). \tag{2.86}$$

The above is an alternative definition of $\pi^*(\cdot)$.

The MCMC framework approaches the problem of sampling from Π^* in a beautiful but nonobvious way. Let's imagine that Π^* is the stationary distribution of a particular Markov chain. If we could randomly walk that Markov chain, then we could sample from Π^* eventually. Thus, we need to construct a transition kernel $P(\mathbf{x}, \mathcal{A})$ which converges to Π^* in the limit.

Suppose we represent the transition kernel as

$$P(\mathbf{x}, \mathrm{d}\mathbf{y}) = p(\mathbf{x}, \mathbf{y})\mathrm{d}\mathbf{y} + r(\mathbf{x})\mathbb{1}(\mathbf{x} \in \mathrm{d}\mathbf{y}), \tag{2.87}$$

where $p(\mathbf{x}, \mathbf{y})$ is some function, $\mathbb{1}(c)$ is an indicator random variable taking the value unity if condition c is true and zero otherwise, and $r(\mathbf{x})$ is defined as

$$r(\mathbf{x}) = 1 - \int_{\mathbb{R}^D} p(\mathbf{x}, \mathbf{y})\mathrm{d}\mathbf{y}. \tag{2.88}$$

Then alternatively, we can write the transition kernel as

$$P(\mathbf{x}, \mathrm{d}\mathbf{y}) = \begin{cases} 1 - \int_{\mathbb{R}^D} p(\mathbf{x}, \mathbf{y})\mathrm{d}\mathbf{y} & \text{if } \mathbf{x} \in \mathrm{d}\mathbf{y}, \\ p(\mathbf{x}, \mathbf{y})\mathrm{d}\mathbf{y} & \text{else.} \end{cases} \tag{2.89}$$

Thus, $r(\mathbf{x})$ is the probability that the Markov chain remains at $\mathbf{x}$, and $\int_{\mathbb{R}^D} p(\mathbf{x}, \mathbf{y})\mathrm{d}\mathbf{y}$ is not necessarily one because $r(\mathbf{x})$ is not necessarily zero.

Now consider the following *reversibility constraint*,

$$\pi(\mathbf{x})p(\mathbf{x}, \mathbf{y}) = \pi(\mathbf{y})p(\mathbf{y}, \mathbf{x}). \tag{2.90}$$

If the function $p(\mathbf{x}, \mathbf{y})$ adheres to this constraint, then $\pi(\cdot)$ is the probability function of the stationary distribution Π of the transition kernel $P(\mathbf{x}, \cdot)$. To see this, consider the following derivation [Tierney, 1994, Chib and Greenberg, 1995]:

$$\begin{aligned}
&\int P(\mathbf{x}, \mathcal{A})\pi(\mathbf{x})\mathrm{d}\mathbf{x} \\
&= \int \left[\int_{\mathcal{A}} p(\mathbf{x}, \mathbf{y})\mathrm{d}\mathbf{y}\right] \pi(\mathbf{x})\mathrm{d}\mathbf{x} + \int r(\mathbf{x})\mathbb{1}(\mathbf{x} \in \mathcal{A})\pi(\mathbf{x})\mathrm{d}\mathbf{x} \\
&= \int_{\mathcal{A}} \left[\int p(\mathbf{x}, \mathbf{y})\pi(\mathbf{x})\mathrm{d}\mathbf{x}\right] \mathrm{d}\mathbf{y} + \int_{\mathcal{A}} r(\mathbf{x})\pi(\mathbf{x})\mathrm{d}\mathbf{x} \\
&\overset{\star}{=} \int_{\mathcal{A}} \left[\int p(\mathbf{y}, \mathbf{x})\pi(\mathbf{y})\mathrm{d}\mathbf{x}\right] \mathrm{d}\mathbf{y} + \int_{\mathcal{A}} r(\mathbf{x})\pi(\mathbf{x})\mathrm{d}\mathbf{x} \\
&= \int_{\mathcal{A}} (1 - r(\mathbf{y}))\pi(\mathbf{y})\mathrm{d}\mathbf{y} + \int_{\mathcal{A}} r(\mathbf{x})\pi(\mathbf{x})\mathrm{d}\mathbf{x} \\
&= \int_{\mathcal{A}} \pi(\mathbf{x})\mathrm{d}\mathbf{x}.
\end{aligned} \tag{2.91}$$

Step $\star$ is the key. It only holds because of the reversibility constraint, and it's what allows us to cancel all terms except $\pi(\mathbf{y})\mathrm{d}\mathbf{y}$.

Let's review. We want to sample from some target distribution Π^*. We imagine that this distribution is the stationary distribution of some Markov chain, but we don't know the chain's transition kernel $P(\mathbf{x}, \mathcal{A})$. The above derivation demonstrates that if we define $P(\mathbf{x}, \mathcal{A})$ as Equation (2.87) and ensure that $p(\mathbf{x}, \mathbf{y})$ adheres to the reversibility constraint in Equation (2.90), then we will have found the transition kernel for a chain whose stationary distribution is our target distribution. This is the essence of Metropolis–Hastings. As we will see, MH's acceptance criteria is constructed to ensure that the reversibility constraint is met.

Metropolis–Hastings. We want to construct the function $p(\mathbf{x}, \mathbf{y})$ such that it is reversible. Consider a candidate generating distribution $Q_{\mathbf{y}|\mathbf{x}}$ with probability function $q(\mathbf{y} \mid \mathbf{x})$. This distribution generates candidate samples $\mathbf{y}$ conditioned on $\mathbf{x}$ that will be either rejected or accepted depending on some criteria. Note that if $\mathbf{x}$ and $\mathbf{y}$ are states, this is a Markov process because no past states are considered. The future only depends on the present. Since $Q_{\mathbf{y}|\mathbf{x}}$ is a distribution, $\int q(\mathbf{y} \mid \mathbf{x})\mathrm{d}\mathbf{y} = 1$. If the following were true,

$$q(\mathbf{y} \mid \mathbf{x})\pi(\mathbf{x}) = q(\mathbf{x} \mid \mathbf{y})\pi(\mathbf{y}) \tag{2.92}$$

then we're done. We've satisfied Equation (2.90). But most likely this is not the case. For example, we might find that

$$q(\mathbf{y} \mid \mathbf{x})\pi(\mathbf{x}) \geq q(\mathbf{x} \mid \mathbf{y})\pi(\mathbf{y}). \tag{2.93}$$

Our random process moves from $\mathbf{x}$ to $\mathbf{y}$ more often than from $\mathbf{y}$ to $\mathbf{x}$. MH ensures equilibrium or reversibility by restricting some moves according to an acceptance criterion, $\alpha(\mathbf{x}, \mathbf{y}) \leq 1$. Thus, if a move is not made, the process returns to $\mathbf{x}$. Intuitively, this is what allows us to balance $q(\mathbf{y} \mid \mathbf{x})\pi(\mathbf{x})$ with $q(\mathbf{x} \mid \mathbf{y})\pi(\mathbf{y})$. Let's assume that moves from $\mathbf{x}$ to $\mathbf{y}$ happen more often than the reverse. Then our criteria would be

$$\begin{aligned} q(\mathbf{y} \mid \mathbf{x})\pi(\mathbf{x})\alpha(\mathbf{x}, \mathbf{y}) &= q(\mathbf{x} \mid \mathbf{y})\pi(\mathbf{y}) \\ \alpha(\mathbf{x}, \mathbf{y}) &= \frac{q(\mathbf{x} \mid \mathbf{y})\pi(\mathbf{y})}{q(\mathbf{y} \mid \mathbf{x})\pi(\mathbf{x})}. \end{aligned} \tag{2.94}$$

Of course, the probabilities could be reversed, but we can handle both cases this with a single expression:

$$\alpha(\mathbf{x}, \mathbf{y}) = \min\left\{1, \frac{q(\mathbf{x} \mid \mathbf{y})\pi(\mathbf{y})}{q(\mathbf{y} \mid \mathbf{x})\pi(\mathbf{x})}\right\}. \tag{2.95}$$

If MH moves from $\mathbf{x}$ to $\mathbf{y}$ more than the reverse, then the numerator is greater than the denominator, and the probability of accepting a move to $\mathbf{y}$ from $\mathbf{x}$ goes down. If we move from $\mathbf{y}$ to $\mathbf{x}$ more than the reverse, the sample (move from $\mathbf{x}$ to $\mathbf{y}$) is accepted with probability one.

We have found our function $p(\mathbf{x}, \mathbf{y})$. It is

$$p_{\mathsf{MH}}(\mathbf{x}, \mathbf{y}) = \alpha(\mathbf{x}, \mathbf{y})q(\mathbf{y} \mid \mathbf{x}), \qquad \mathbf{x} \neq \mathbf{y}. \tag{2.96}$$

And while it is unnecessary to write down, for completeness the full transition kernel $P_{\mathsf{MH}}(x, y)$ is

$$P_{\mathsf{MH}}(\mathbf{x}, \mathbf{y}) = \overbrace{\alpha(\mathbf{x}, \mathbf{y})q(\mathbf{y} \mid \mathbf{x})}^{\text{Probability of leaving } \mathbf{x}} + \overbrace{\left[1 - \int_{\mathbb{R}^D} \alpha(\mathbf{x}, \mathbf{y})q(\mathbf{y} \mid \mathbf{x})\mathrm{d}\mathbf{y}\right] \mathbb{1}(\mathbf{x} \in \mathrm{d}\mathbf{y})}^{\text{Probability of staying on } \mathbf{x}}. \tag{2.97}$$

In summary, if we sample from the conditional distribution $Q_{\mathbf{y}|\mathbf{x}}$ and accept the proposed sample according to $\alpha(\mathbf{x}, \mathbf{y})$, then we'll be randomly walking according to a Markov chain whose stationary distribution is Π^*.

Metropolis. Notice that for a symmetric conditional distribution, so when $q(\mathbf{y} \mid \mathbf{x}) = q(\mathbf{x} \mid \mathbf{y})$, Equation (2.95) simplifies to

$$\alpha(\mathbf{x}, \mathbf{y}) = \min\left\{1, \frac{\pi(\mathbf{y})}{\pi(\mathbf{x})}\right\}. \tag{2.98}$$

This is the predecessor to Metropolis–Hastings, the *Metropolis algorithm* [Metropolis et al., 1953]. In 1970, Wilfred Hastings extended Equation (2.98) to the more general asymmetrical case in Equation (2.95) [Hastings, 1970]. If we assume, for example, that $Q_{\mathbf{y}|\mathbf{x}}$ is a conditional Gaussian distribution, then running the Metropolis–Hastings algorithm is equivalent to running the Metropolis algorithm.

Example: Rosenbrock density. Imagine we want to sample from the Rosenbrock distribution (Figure 2.9) with the following probability function:

$$\pi^*(\theta_1, \theta_2; a, b) \propto \exp\left\{-\frac{(a - \theta_1)^2 + b(\theta_2 - \theta_1^2)^2}{20}\right\}. \tag{2.99}$$

The Rosenbrock function [Rosenbrock, 1960] is a well-known test function in optimization because while finding a minimum is relatively easy, finding the global minimum at (1, 1) is less trivial. Goodman and Weare [2010] adapted the function to serve as a benchmark for MCMC algorithms.

We must provide a candidate transition kernel $q(\theta^\star \mid \theta)$. The only requirement is that we can sample conditionally from it and that $\int q(\theta^\star \mid \theta) \mathrm{d}\theta^\star = 1$ for all θ. Perhaps the simplest possible kernel is to simply add Gaussian noise to θ. Thus, our candidate transition kernel is the distribution

$$\theta^\star \mid \theta \sim \mathcal{N}(\theta, \sigma^2), \tag{2.100}$$

where the variance σ^2 controls how big each step is. Since the distribution is symmetric, we can implement the simpler Metropolis algorithm. We can accept a proposal sample with

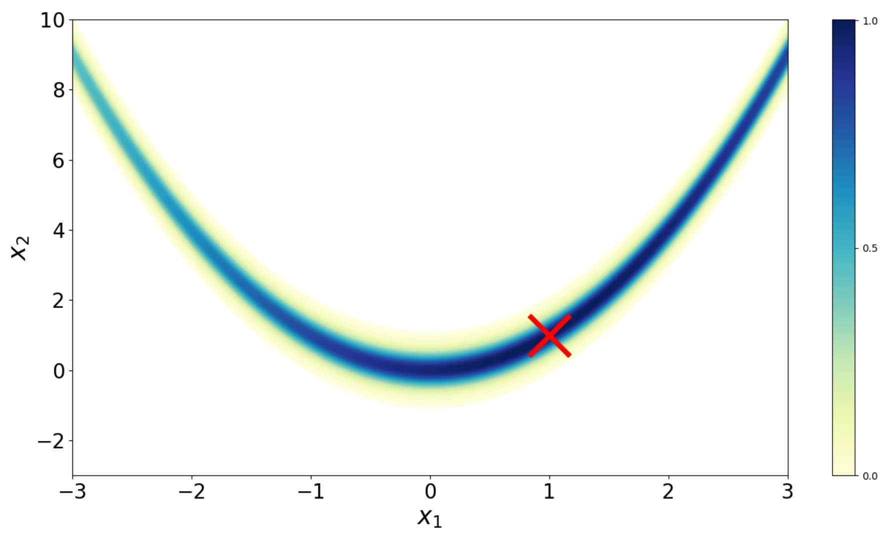

Figure 2.9: The Rosenbrock density (Equation (2.99)) with $a = 1$ and $b = 100$. Darker colors indicate higher probability. The global minimum is at $(x, y) = (a, a^2) = (1, 1)$ and denoted with a red "X".

probability $\alpha(\theta, \theta^{\star})$ by drawing a uniform random variable with support $[0, 1]$, and checking if it is less than the acceptance criteria. We can write MH in Python as the following:

```
import numpy as np
from   numpy.random import multivariate_normal as mvn
import matplotlib.pyplot as plt

n_iters    = 1000
samples    = np.empty((n_iters, 2))
samples[0] = np.random.uniform(low=[-3, -3], high=[3, 10], size=2)
rosen      = lambda x, y: np.exp(-((1 - x)**2 + 100*(y - x**2)**2) / 20)

for i in range(1, n_iters):
    curr  = samples[i-1]
    prop  = curr + mvn(np.zeros(2), np.eye(2) * 0.1)
    alpha = rosen(*prop) / rosen(*curr)
    if np.random.uniform() < alpha:
        curr = prop
    samples[i] = curr

plt.plot(samples[:, 0], samples[:, 1])
plt.show()
```

That's it. Despite its conceptual depth, Metropolis–Hastings is surprisingly simple to implement, and it is not hard to imagine writing a more general implementation that handles Equation (2.95) for any arbitrary π and $q(\cdot)$.

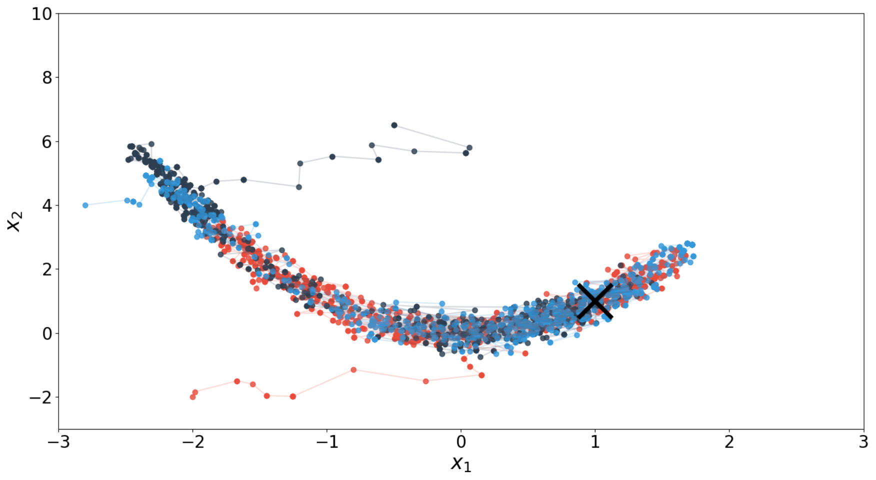

Figure 2.10: Three randomly initialized Markov chains run on the Rosenbrock density (Equation (2.99)) using the Metropolis–Hastings algorithm. After mixing, each chain walks in regions where the probability is high. The global minimum is at $(x, y) = (a, a^2) = (1, 1)$ and denoted with a black "X".

The above code is the basis for Figure 2.10, which runs three Markov chains from randomly initialized starting points. This example highlights two important implementation details for Metropolis–Hastings. First, step size (or σ^2 for our candidate distribution) is important. If the step size were too large relative to the support of the Rosenbrock distribution, it would be difficult to sample near the distribution's mode. Second, typically, the initial samples from an MCMC sampler are discarded. This is because samples before the chain starts to mix are not representative of the stationary distribution. This period of discarding samples is called the *burn-in period*.

2.3.5. Gibbs sampling

Gibbs sampling is a special case of Metropolis–Hastings in which the newly proposed state is always accepted with probability one. It is fairly straightforward to see this. Consider a D-dimensional posterior with parameters $\boldsymbol{\theta} = \{\theta_1, \dots, \theta_D\}$. The basic idea of Gibbs sampling is to iteratively sample from the conditional distribution $P_{\theta_d | \mathbf{X}, \boldsymbol{\theta}_{-d}}$ where $\boldsymbol{\theta}_{-d}$ is $\boldsymbol{\theta}$ without the dth parameter. Let $\theta_k^{(t)}$ denote the value of the kth component of $\boldsymbol{\theta}$ on the tth iteration. Then Gibbs sampling is Algorithm 1.

Algorithm 1: Gibbs sampling

```
for $t = 1, \dots T$ do
    $\theta_1^{(t+1)} := \theta_1^\star \sim P_{\theta_1^{(t)} | \mathbf{X}, \theta_2^{(t)}, \theta_3^{(t)}, \dots, \theta_D^{(t)}}$
    $\theta_2^{(t+1)} := \theta_2^\star \sim P_{\theta_2^{(t)} | \mathbf{X}, \theta_1^{(t+1)}, \theta_3^{(t)}, \dots, \theta_D^{(t)}}$
    $\theta_d^{(t+1)} := \theta_d^\star \sim P_{\theta_d^{(t)} | \mathbf{X}, \theta_1^{(t+1)}, \dots, \theta_{d-1}^{(t+1)}, \theta_d^{(t)}, \dots, \theta_D^{(t)}}$
        $\vdots$
    $\theta_D^{(t+1)} := \theta_D^\star \sim P_{\theta_D^{(t)} | \mathbf{X}, \theta_1^{(t+1)}, \theta_2^{(t+1)}, \dots, \theta_{D-1}^{(t+1)}}$
```

To see why this works, first note that

$$p(\boldsymbol{\theta} \mid \mathbf{X}) = p(\theta_d, \boldsymbol{\theta}_{-d} \mid \mathbf{X}) = p(\theta_d \mid \mathbf{X}, \boldsymbol{\theta}_{-d}) p(\boldsymbol{\theta}_{-d} \mid \mathbf{X}). \tag{2.101}$$

Ignoring the iterates' notation, the probability of a transition can be written as

$$\begin{aligned} \alpha(\boldsymbol{\theta}^\star \mid \boldsymbol{\theta}) &= \min\left\{1, \frac{p(\theta^\star \mid \mathbf{X}) p(\theta_d \mid \mathbf{X}, \boldsymbol{\theta}_{-d})}{p(\theta \mid \mathbf{X}) p(\theta_d^\star \mid \mathbf{X}, \boldsymbol{\theta}^\star_{-d})}\right\} \\ &= \min\left\{1, \frac{p(\theta_d^\star \mid \mathbf{X}, \boldsymbol{\theta}^\star_{-d}) p(\boldsymbol{\theta}^\star_{-d} \mid \mathbf{X}) p(\theta_d \mid \mathbf{X}, \boldsymbol{\theta}_{-d})}{p(\theta_d \mid \mathbf{X}, \boldsymbol{\theta}_{-d}) p(\boldsymbol{\theta}_{-d} \mid \mathbf{X}) p(\theta_d^\star \mid \mathbf{X}, \boldsymbol{\theta}^\star_{-d})}\right\} \\ &= 1. \end{aligned} \tag{2.102}$$

In Equation (2.102), I have color-coded the terms that cancel. In particular, the terms in red cancel because $\boldsymbol{\theta}^\star_{-d} = \boldsymbol{\theta}_{-d}$. In other words, in each step of the Gibbs sampling algorithm, we are performing a Metropolis–Hastings-like random walk in which the proposed next state always adheres to the reversibility constraint.

The primary advantage of Gibbs sampling is simple: proposals are always accepted. The primary disadvantage is that we need to be able to derive the above conditional probability distributions. This is tractable when dealing with conditionally conjugate models.

2.4 Summary

To summarize, a probabilistic approach to machine learning is appealing because such models can handle parameter uncertainty through marginalization, be interpreted through explicit generative processes, and be composed with other models using probability theory. Latent variable models are a class of statistical models which introduce unobserved or latent variables to model more complex phenomena. These models are particularly useful in scientific applications because one can interpret the latent variables through the conditional dependency structure of the model. The Bayesian philosophy of inferential statistics uses Bayes' theorem to infer a posterior distribution over parameters and latent variables, rather than estimating a single value. To the Bayesian, there is no real distinction between statistical parameters and latent variables; both are random variables to be inferred through Bayes' theorem.

Broadly, methods for Bayesian inference can divided into exact and approximate techniques. Conjugate models are useful because we have exact analytic updates of the posterior. However, conjugate models are typically from the exponential family and are limited in flexibility and representational power. Approximate Bayesian inference is a broad class of techniques for inferring posteriors in the absence of a tractable posterior. Methods such as expectation–maximization, Markov chain Monte Carlo, and variational inference are all approximate inference methods.

2A Appendix

2A.1 Jensen's inequality

Let f be a convex function. Then Jensen's inequality is

$$f\left(\sum_{n=1}^{N} \alpha_n \mathbf{x}_n\right) \leq \sum_{n=1}^{N} \alpha_n f(\mathbf{x}_n) \tag{2.103}$$

where $\alpha_n \geq 0$ and $\sum_{n=1}^{N} \alpha_n = 1$. (The inequality is reversed if f is concave.)

Proof. The proof is by induction. First, consider the base case:

$$f(\alpha_1 \mathbf{x}_1 + \alpha_2 \mathbf{x}_2) \leq \alpha_1 f(\mathbf{x}_1) + \alpha_2 f(\mathbf{x}_2). \tag{2.104}$$

This clearly holds because f is convex. Now for the inductive case, we want to show that

$$f\left(\sum_{n=1}^{K} \alpha_n \mathbf{x}_n\right) \leq \sum_{n=1}^{K} \alpha_n f(\mathbf{x}_n) \implies f\left(\sum_{n=1}^{K+1} \alpha_n \mathbf{x}_n\right) \leq \sum_{n=1}^{K+1} \alpha_n f(\mathbf{x}_n). \tag{2.105}$$

First, let's start with our inductive hypothesis and add $\alpha_{K+1} f(\mathbf{x}_{K+1})$ to both sides:

$$f\left(\sum_{n=1}^{K} \alpha_n \mathbf{x}_n\right) + \alpha_{K+1} f(\mathbf{x}_{K+1}) \leq \sum_{n=1}^{K+1} \alpha_n f(\mathbf{x}_n). \tag{2.106}$$

Now the α terms on the right-hand side no longer sum to unity. Let's normalize both sides of the equation by multiplying by $\frac{1}{1+\alpha_{K+1}}$:

$$\overbrace{\frac{1}{1+\alpha_{K+1}}}^{A} f\left(\sum_{n=1}^{K} \alpha_n \mathbf{x}_n\right) + \overbrace{\frac{\alpha_{K+1}}{1+\alpha_{K+1}}}^{B} f(\mathbf{x}_{K+1}) \leq \frac{1}{1+\alpha_{K+1}} \sum_{n=1}^{K+1} \alpha_n f(\mathbf{x}_n). \tag{2.107}$$

This normalization constant makes sense because $\sum_{n=1}^{K} \alpha_n = 1 \iff \sum_{n=1}^{K+1} \alpha_n = 1 + \alpha_{K+1}$. Now note that the terms labeled A and B above sum to unity. And since f is convex, we can say

$$\begin{aligned} &f\left(\frac{1}{1+\alpha_{K+1}} \sum_{n=1}^{K} \alpha_n \mathbf{x}_n + \frac{\alpha_{K+1}}{1+\alpha_{K+1}} \mathbf{x}_{K+1}\right) \\ &\leq \frac{1}{1+\alpha_{K+1}} f\left(\sum_{n=1}^{K} \alpha_n \mathbf{x}_n\right) + \frac{\alpha_{K+1}}{1+\alpha_{K+1}} f(\mathbf{x}_{K+1}). \end{aligned} \tag{2.108}$$

At this point, we're basically done. The left-hand side of the above inequality can be simplified to

$$f\left(\frac{1}{1+\alpha_{K+1}} \sum_{n=1}^{K+1} \alpha_n \mathbf{x}_n\right), \tag{2.109}$$

which we have already shown is less than or equal to

$$\frac{1}{1+\alpha_{K+1}} \sum_{n=1}^{K+1} \alpha_n f(\mathbf{x}_n), \tag{2.110}$$

as desired.

Jensen's inequality for random variables. Now consider this: since $\alpha_n \geq 0$ and $\sum_i \alpha_n = 1$, we can interpret α_n as the probability of a random variable $\mathbf{x}$ taking on a specific value, giving us

$$f\left(\mathbb{E}[\mathbf{x}]\right) \leq \mathbb{E}\left[f(\mathbf{x})\right], \tag{2.111}$$

which for discrete distributions is equivalent to

$$f\left(\sum_{\mathbf{x}} \mathbf{x} p(\mathbf{x}) \mathrm{d}\mathbf{x}\right) \leq \sum_{\mathbf{x}} f(\mathbf{x}) p(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{2.112}$$

2A.2 KL divergence

The Kullback–Leibler (KL) divergence is a metric for how similar two probability distributions are. A standard formulation is the following. Given two probability distributions P and Q with probability functions $p(\cdot)$ and $q(\cdot)$ respectively, the KL divergence is the integral

$$D_{\mathrm{KL}}[P \| Q] = \int_{-\infty}^{\infty} p(\mathbf{x}) \log\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right) \mathrm{d}\mathbf{x}. \tag{2.113}$$

There are many ways to interpret the KL divergence. I'll present just one; the KL divergence can be viewed as a measure of *relative information entropy* between two distributions. Let's rewrite Equation (2.113) in terms of entropy $H(\cdot)$ and cross-entropy $H(\cdot,\cdot)$:

$$\begin{aligned} D_{\mathrm{KL}}[P \| Q] &= \int_{-\infty}^{\infty} p(\mathbf{x}) \log\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right) \mathrm{d}\mathbf{x} \\ &= \mathbb{E}_{p(\mathbf{x})}\left[\log\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right)\right] \\ &= \mathbb{E}_{p(\mathbf{x})}\left[\log p(\mathbf{x}) - \log q(\mathbf{x})\right] \\ &= \mathbb{E}_{p(\mathbf{x})}\left[-\log q(\mathbf{x})\right] - \mathbb{E}_{p(\mathbf{x})}\left[-\log p(\mathbf{x})\right] \\ &= H(P, Q) - H(P), \end{aligned} \tag{2.114}$$

As we can see, the KL divergence is the difference between cross-entropy and entropy (see MacKay [2003] for a discussion of information entropy). In other words, one interpretation of the KL divergence is that it captures the *relative information* or *relative entropy* between two distributions P and Q. Also note that the KL divergence is not symmetric, i.e. $D_{\mathrm{KL}}[P \| Q] \neq D_{\mathrm{KL}}[Q \| P]$ in general.

2A.3 Variational inference

In Bayesian inference, we are often interested in the posterior distribution $P_{\mathbf{Z}|\mathbf{X}}$ with probability function $p(\mathbf{Z} \mid \mathbf{X})$ where $\mathbf{X} := \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ are observations, and $\mathbf{Z} := \{\mathbf{z}_1, \dots, \mathbf{z}_N\}$ are latent variables. However, in many practical models of interest, this posterior is intractable because we cannot compute the evidence or denominator of Bayes' theorem, $p(\mathbf{X})$. This

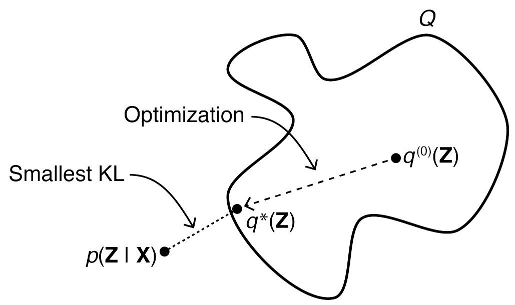

Figure 2A.3.1: Diagram of VI. A family of distributions $\mathcal{Q}$ is visualized as a blob. VI starts with some initial distribution $q^{(0)}(\mathbf{Z}) \in \mathcal{Q}$ and then iteratively minimizes the KL divergence between the approximating distribution at iteration t, call this $q^{(t)}(\mathbf{Z})$, and the desired posterior $p(\mathbf{Z} \mid \mathbf{X})$. The goal is to find an optimal distribution $q^*(\mathbf{Z})$ where optimality is defined as having the smallest possible KL divergence between $q^*(\mathbf{Z})$ and $p(\mathbf{Z} \mid \mathbf{X})$.

evidence is hard to compute because we have introduced latent variables that must now be marginalized out. Such integrals are often *intractable* in the sense that (1) we do not have an analytic expression for them or (2) they are computationally intractable. See Blei et al. [2017] for examples.

The main idea of variational inference (VI) is to use optimization to find a simpler or more tractable distribution $Q_{\mathbf{Z}}$ with probability function $q(\mathbf{Z})$ from a family of distributions $\mathcal{Q}$ such that it is close to the desired posterior distribution $p(\mathbf{Z} \mid \mathbf{X})$ (Figure 2A.3.1[13]). In VI, we define "close to" using the Kullback–Leibler (KL) divergence. Thus, the desired VI objective is

$$q^*(\mathbf{Z}) := \arg \min_{q(\mathbf{Z}) \in \mathcal{Q}} D_{\mathrm{KL}}[q(\mathbf{Z}) \| p(\mathbf{Z} \mid \mathbf{X})]. \tag{2.115}$$

Minimizing the KL divergence can be interpreted as minimizing the relative entropy between the two distributions. See Section 2A.2 for a discussion.

2A.3.1. Evidence-lower bound

The main challenge with the variational inference objective in Equation (2.115) is that it implicitly depends on the evidence, $p(\mathbf{X})$. Thus, we have not yet gotten around the intractability discussed above. To see this dependence, let's write out the definition of the

[13]I saw a version of this diagram in a talk by David Blei at Princeton University.

KL divergence:

$$\begin{aligned}
D_{\text{KL}}[q(\mathbf{Z}) \| p(\mathbf{Z}|\mathbf{X})] &= \int_q q(\mathbf{Z}) \log \frac{q(\mathbf{Z})}{p(\mathbf{Z}|\mathbf{X})} \\
&= \mathbb{E}_{q(\mathbf{Z})}\left[\log \frac{q(\mathbf{Z})}{p(\mathbf{Z}|\mathbf{X})}\right] \\
&= \underbrace{\mathbb{E}_{q(\mathbf{Z})}[\log q(\mathbf{Z})] - \mathbb{E}_{q(\mathbf{Z})}[\log p(\mathbf{Z}, \mathbf{X})]}_{-\text{ELBO}(q)} + \log p(\mathbf{X}).
\end{aligned} \tag{2.116}$$

Because we cannot compute the desired KL divergence, we optimize a different objective that is equivalent to this KL divergence up to constant. This new objective is called the *evidence lower bound* (ELBO):

$$\text{ELBO}(q) := \mathbb{E}_{q(\mathbf{Z})}[\log p(\mathbf{Z}, \mathbf{X})] - \mathbb{E}_{q(\mathbf{Z})}[\log q(\mathbf{Z})]. \tag{2.117}$$

This is a negation of the left two terms in Equation (2.116). We can rewrite Equation (2.116) as

$$\log p(\mathbf{X}) = \text{ELBO}(q) + D_{\text{KL}}[q(\mathbf{Z}) \| p(\mathbf{Z}|\mathbf{X})]. \tag{2.118}$$

Why is the ELBO so-named? Since the KL divergence is non-negative, we know

$$\log p(\mathbf{X}) \geq \text{ELBO}(q). \tag{2.119}$$

In other words, the log evidence $\log p(\mathbf{X})$, a fixed quantity for any set of observations $\mathbf{X}$, cannot be less than the ELBO. So if we maximize the ELBO, we minimize the desired KL divergence. This is VI in a nutshell.

2A.3.2. Relationship to EM

It's fun to observe the relationship between VI and EM discussed in Section 2.2.1. EM maximizes the expected log likelihood when $q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X})$, i.e.

$$\log p(\mathbf{X}) = \overbrace{\text{ELBO}(q)}^{\text{EM maximizes this}} + \overbrace{\text{KL}[q(\mathbf{Z}) \| p(\mathbf{Z}|\mathbf{X})]}^{\text{Since this is zero}}. \tag{2.120}$$

To be a bit more pedantic, at iteration t with current parameters estimates $\boldsymbol{\theta}^{(t)}$, EM optimizes this expected complete log likelihood inside the ELBO:

$$\log p(\mathbf{X} \mid \boldsymbol{\theta}) = \overbrace{\mathbb{E}_{p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}^{(t)})}\left[\log p(\mathbf{X}, \mathbf{Z} \mid \boldsymbol{\theta})\right]}^{\text{EM maximizes this}} - \mathbb{E}_{p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}^{(t)})}\left[\log p(\mathbf{Z} \mid \mathbf{X})\right]. \tag{2.121}$$

See Section 2.2.1 for why EM ignores the right term in Equation (2.121).

Chapter 3

Deep probabilistic CCA

Probabilistic canonical correlation analysis (P-CCA) [Bach and Jordan, 2005] is a multivariate statistical method that assumes two paired observations are generated from a single shared latent variable. This probabilistic model is a reformulation of classic CCA [Hotelling, 1936], which finds linear projections of two data sets such that projected data pairs (paired latent variables) are maximally correlated. In this first chapter, I present an end-to-end inference framework for *deep probabilistic CCA* (DP-CCA). The goal of DP-CCA is to infer shared structure in a way that drives the representation learning of neural networks.

DP-CCA was motivated by an important problem in computational biology. Medical pathology images are visually evaluated by experts for disease diagnosis, but the connection between image features and the state of the cells in an image is typically unknown. To understand this relationship, we developed DP-CCA to estimate the shared latent structure of joint gene expression levels and medical image features. The challenge was to leverage the powerful representation learning of models such as convolutional neural networks with more mathematically well-understood methods such as P-CCA. Thus, DP-CCA is trained end-to-end so that the P-CCA and neural network parameters are estimated simultaneously. I demonstrate the utility of this method in constructing image features that are predictive of gene expression levels on simulated data and the Genotype-Tissue Expression data, and demonstrate that the latent variables are interpretable by disentangling the latent subspace through shared and modality-specific views.

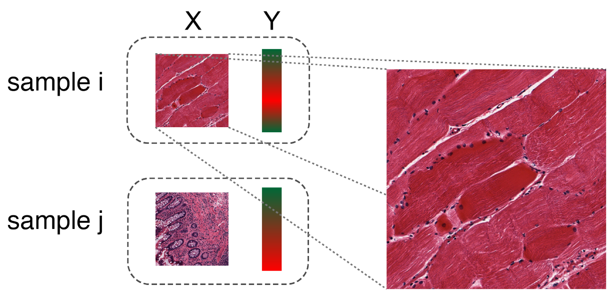

Figure 3.1.1: A paired GTEx sample consists of a whole tissue slide (cropped) along with gene expression levels from the same tissue and donor.

3.1 Introduction

Many diseases are diagnosed by pathologists using morphological features in tissue imaging data. But the genes that capture the internal state of cells and that are associated with a specific tissue morphology are typically unknown and hard to assay in a particular sample. The Genotype-Tissue Expression (GTEx) Consortium [Consortium et al., 2017, Carithers et al., 2015] has collected data from over 948 autopsy subjects (donors), including standardized whole tissue histology slides, giving us images of each sample, and bulk RNA-sequencing, giving us gene expression levels for each sample, from approximately 50 different human tissues (Figure 3.1.1). These multi-subject, multi-view data provide an opportunity to develop computational tools that capture the relationship between cell state (observable in gene expression data) and morphological features (observable in histology images).

Historically, modeling data across two views with the goal of extracting shared signal has been performed by some version of canonical correlation analysis (CCA) [Hotelling, 1936]. Given two random vectors, CCA aims to find two linear projections into a shared latent space for which the projected vectors are maximally correlated. Probabilistic CCA (P-CCA) is particularly attractive for medical applications with small sample sizes. (See Ghahramani [2015] or Section 2.1 for a discussion of the value of probabilistic modeling in low-data scenarios.)

In its most general form, P-CCA will ignore possibly important nonlinear structure in data such as images; this nonlinear structure could be extracted first with computer vision techniques such as convolutional neural networks [LeCun et al., 1998] that have achieved excellent performance on medical imaging tasks [Bar et al., 2015, Shah et al., 2017, Esteva et al., 2017, Geras et al., 2017, Gulshan et al., 2016]. (See Section 3A.5 for a detailed introduction to the convolution operator and convolutional neural networks.) To this end, two recent studies trained models in a two-stage approach, first embedding the imaging data using convolutional models and then fitting CCA to the lower-dimensional embedded data [Ash et al., 2021, Subramanian et al., 2018]. Another two-stage approach computed the principal components of single views and then computed cross-view correlations [Barry et al., 2018], while Cooper et al. [2012] first clustered image features and then looked for genomic associations. However, two-stage approaches decouple the image feature learning from estimating the shared latent subspace of the data views, leading to image features that capture minimal variation in the shared subspace, and projections of the two views that are difficult to interpret. While "interpretability" is a broad concept [Lipton, 2016], here we mean specifically that we can identify small subsets of correlated features across modalities, here, gene expression levels representing cell state that are associated with specific image features.

To address these challenges, we propose a multimodal modeling and inference framework, deep probabilistic CCA (DP-CCA). DP-CCA estimates end-to-end the nonlinear embeddings and shared latent structure of paired high-dimensional data sets—here, histology images and paired gene expression data. Our model is *probabilistic* in that it is generative and models parameter uncertainty, *interpretable* in that it uses sparse P-CCA to associate sets of genes with image features, and *nonlinear* in that it learns convolutional features for high-dimensional observations. Our training procedure makes use of automatic differentiation of a single loss function; this avoids the difficulties of implementing joint inference over probabilistic models and neural networks via conjugacy and message passing [Johnson et al., 2016, Lin et al., 2018].

The impact of solving these challenges is twofold. First, our DP-CCA model will include latent factors capturing shared variation across the paired data modalities but also latent factors that capture modality-specific variation; our end-to-end inference will substantially increase the shared variation captured in the latent space by identifying embeddings for each data view that maximize the shared variation. Second, this generative framework allows cross-mode imputation: given a fitted model, we can guess at the values of gene expression data for a held-out histology image, for example. This is particularly important given a discrepancy in cost between the two data modalities, since it is much more expensive to assay gene expression in a tissue sample than to stain and image that sample.

We illustrate the behavior of our method for simultaneous embedding and latent space modeling of multiple data modalities on both the MNIST data [LeCun et al., 2010], where we paired Gaussian-distributed vectors with specific digits, and on the GTEx v6 data. We compare the results from our approach against results from related multimodal approaches in order to illustrate the additional gains in the variation captured in the shared latent space and also the interpretability of the end-to-end inference of embeddings and shared space. We validate these results in the GTEx v6 data using additional held out biological data that correlate with signal identified in the inferred latent space. We conclude with thoughts on further improvements to our model.

Contributions: The fundamental contributions of DP-CCA to the field of simultaneous embedding and joint modeling of high-dimensional multimodal data include addressing three main methodological and domain-specific challenges. First, in our end-to-end training procedure, the shared latent subspace drives the convolutional image embeddings. Compared with a standard autoencoder that learns embeddings that minimize a reconstruction loss, the P-CCA backend encourages image embeddings that maximally explain variation in the other data modalities. Second, the shared and modality-specific latent variables provide three views into variation that can be mined for domain-specific patterns of interest, making our model interpretable with respect to the data domain. Finally, the shared latent variables represent a *composite phenotype* between tissue morphology and gene expression—sets of genes representing cell state and image features that covary together. These composite phenotypes can be used for many downstream tasks, including identifying paired phenotypic differences between sick and healthy patients and testing for associations with other modalities, such as genotype.

3.2 Background

The original realization of CCA was more recently reframed as a probabilistic model. This model is known as inter-battery factor analysis in the statistics community [Tucker, 1958, Browne, 1979] and was re-derived as probabilistic CCA [Bach and Jordan, 2005] in the machine learning community. An important feature of P-CCA is the allowance for view-specific noise. If P-CCA assumed independent noise, that would mean that any view-specific variation in the data would have to be modeled as shared variation. The model would have no other way of explaining that variance given that it assumes noise is independent.

CCA has also been extended to nonlinear settings using kernel methods [Akaho, 2006, Hardoon et al., 2004]. Variants of combining CCA with neural networks also exist. Deep CCA (DCCA) estimates linear projections on the outputs of neural networks [Andrew et al., 2013]. Deep variational CCA (DVCCA) is a variational approximation of CCA using a single encoder, while we learn pairs of embeddings with view-specific encoders [Wang et al., 2016]. While DCCA learns embeddings that capture shared structure, it does not explicitly model view-specific noise as in P-CCA. We demonstrate that this is an important benefit of our model in Section 3.4. Furthermore, learning linear maps as in CCA and P-CCA is key to the interpretation of covarying data features from a given latent variable.

Deep multimodal learning without the notion of correlation maximization is another related body of work [Ngiam et al., 2011]. However, a multimodal autoencoder that learns a shared lower-dimensional representation explicitly optimizes a reconstruction loss, but it does not disentangle the latent space across views, which is an essential component of our model. Another related model worth mentioning is oi-VAE [Ainsworth et al., 2018], which uses multiple decoders over the same latent variables, with the goal of having interpretable factors of the same data view, not accounting for multiple views.

We note that our domain-specific problem is related to other domains such as image captioning in computer vision and neural machine translation in natural language processing. A major distinction in language-based modeling is that they cannot make the same Gaussian assumptions about their data.

Table 3.2.1: We preprocessed the GTEx v6 data to only include samples from which 1000×1000 pixels crops could be taken and that had both tissue slides and gene expression levels. After preprocessing, we obtained 2221 paired samples from 29 tissue types. The data are both small and class-imbalanced.

Tissue	Count	Tissue	Count
Adipose Tissue	5	Nerve	9
Adrenal Gland	134	Ovary	88
Bladder	4	Pancreas	166
Blood Vessel	47	Pituitary	51
Brain	172	Prostate	53
Breast	5	Salivary Gland	10
Cervix Uteri	7	Skin	28
Colon	81	Small Intestine	59
Esophagus	134	Spleen	103
Fallopian Tube	4	Stomach	106
Heart	188	Testis	44
Kidney	12	Thyroid	65
Liver	115	Uterus	69
Lung	76	Vagina	17
Muscle	369	TOTAL	2221

3.2.1. Problem setup

We index N paired samples using $n \in \{1, 2, \dots, N\}$, and we index two data views a and b using $j \in \{a, b\}$. The nth paired sample is a tuple $(\mathbf{x}_n^a, \mathbf{x}_n^b)$. Here, an image $\mathbf{x}_n^a$ is a multidimensional array with dimensions for the number of channels, image height, and image width; for ease of notation we can flatten this multidimensional array into a vector with dimensionality $\mathbb{R}^{Q^a}$. A gene expression sample $\mathbf{x}_n^b$ is a $\mathbb{R}^{Q^b}$-vector for Q^b genes. We embed each data view before performing P-CCA, and we refer to these view-specific embeddings as P^a- and P^b-dimensional vectors $\mathbf{y}^a$ and $\mathbf{y}^b$. Here we use a convolutional autoencoder for the image vector and a linear embedding for the gene expression vector. Each paired sample comes from a single donor and one of 29 human tissues after preprocessing (Table 3.2.1). The sample tissue labels are held out to be used as biological signal to validate the model, which we explore in Section 3.4.

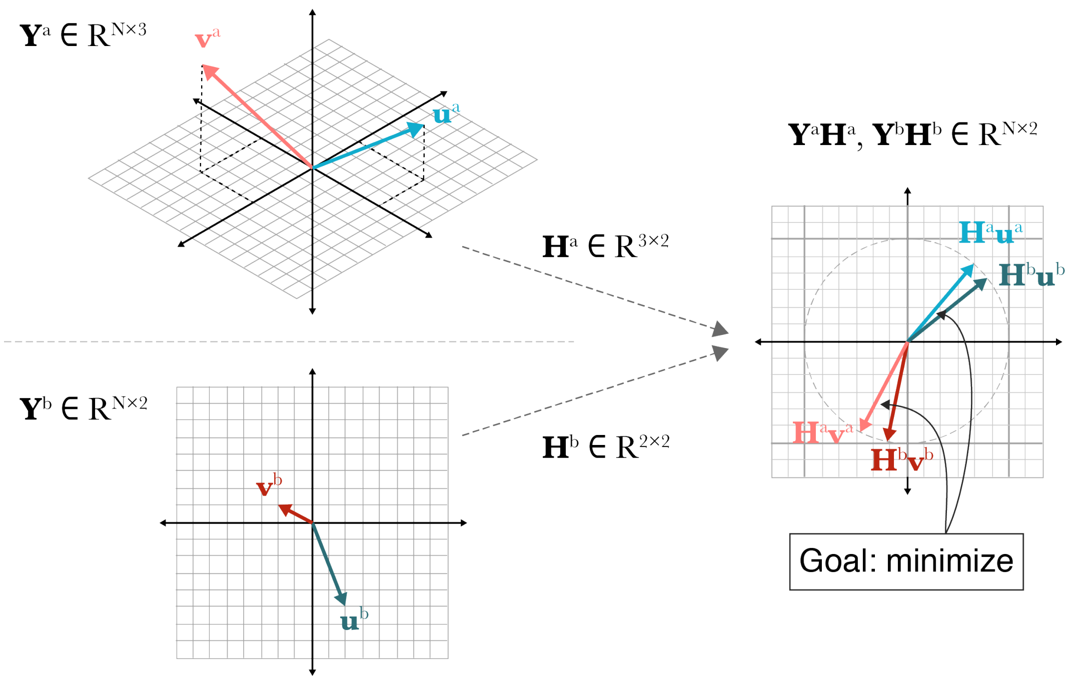
$\mathbf{Y}^a \in R^{N\times3}$
$\mathbf{v}^a$
$\mathbf{u}^a$
$\mathbf{Y}^b \in R^{N\times2}$
$\mathbf{v}^b$
$\mathbf{u}^b$
$\mathbf{H}^a \in R^{3\times2}$
$\mathbf{H}^b \in R^{2\times2}$
$\mathbf{Y}^a\mathbf{H}^a, \mathbf{Y}^b\mathbf{H}^b \in R^{N\times2}$
$\mathbf{H}^a\mathbf{u}^a$
$\mathbf{H}^b\mathbf{u}^b$
$\mathbf{H}^a\mathbf{v}^a$
$\mathbf{H}^b\mathbf{v}^b$
Goal: minimize

Figure 3.2.1: Diagram of canonical correlation analysis. Let $N = 2$ be the number of observations. Two data sets $\mathbf{Y}^a \in \mathbb{R}^{N\times3}$ and $\mathbf{Y}^b \in \mathbb{R}^{N\times2}$ are transformed by projections $\mathbf{H}^a \in \mathbb{R}^{3\times2}$ and $\mathbf{H}^b \in \mathbb{R}^{2\times2}$ such that the paired embeddings, $(\mathbf{v}_a, \mathbf{v}_b)$ and $\mathbf{u}_a, \mathbf{u}_b)$, are maximally correlated with unit length in the projected space.

3.2.2. Canonical correlation analysis

Canonical correlation analysis (CCA) is a multivariate statistical method for finding two linear projections, one for each set of observations in a paired data set, such that the projected pairs are maximally correlated. Provided the data are mean-centered, this procedure can be visualized fairly easily (Figure 3.2.1). CCA finds linear projections such that paired data points are close to each other in a lower-dimensional space. And since mean-centered points are vectors with tails at the origin $\mathbf{0}$, CCA can be thought of as minimizing the cosine distance between paired points.

To formalize this, consider two paired data sets, $\mathbf{Y}^a \in \mathbb{R}^{N\times P^a}$ and $\mathbf{Y}^b \in \mathbb{R}^{N\times P^b}$. By "paired", we mean that the nth sample consist of two row vectors, $\left(\mathbf{y}_n^a, \mathbf{y}_n^b\right)$, with dimensionality P^a and P^b respectively. Thus, there are N paired empirical multivariate observations. We assume the data are mean-centered.

Let $\mathbf{h}^a$ and $\mathbf{h}^b$ denote linear transformations of the data, P^a- and P^b-vectors respectively. These are vectors in the matrices in Figure 3.2.1. A pair of *canonical variables* $\mathbf{z}^a \in \mathbb{R}^N$ and

$\mathbf{z}^b \in \mathbb{R}^N$ are the images of $\mathbf{Y}^a$ and $\mathbf{Y}^b$ after projection, i.e.:

$$\mathbf{z}^a = \mathbf{Y}^a \mathbf{h}^a \qquad \mathbf{z}^b = \mathbf{Y}^b \mathbf{h}^b. \tag{3.1}$$

The goal of CCA is to learn $\mathbf{h}^a$ and $\mathbf{h}^b$ such that this pair of canonical variables are maximally correlated. Let $\mathbf{\Sigma}^{ij}$ denote the covariance matrix between data sets $\mathbf{Y}^i$ and $\mathbf{Y}^j$ for $i, j \in \{a, b\}$. Then the CCA objective is

$$\begin{aligned}
(\mathbf{h}^a)^\star, (\mathbf{h}^b)^\star &= \underset{\mathbf{h}^a, \mathbf{h}^b}{\operatorname{argmax}} \operatorname{corr}(\mathbf{Y}^a \mathbf{h}^a, \mathbf{Y}^b \mathbf{h}^b) \\
&= \underset{\mathbf{h}^a, \mathbf{h}^b}{\operatorname{argmax}} \frac{(\mathbf{h}^a)^\top \mathbf{\Sigma}^{ab} \mathbf{h}^b}{\sqrt{(\mathbf{h}^a)^\top \mathbf{\Sigma}^{aa} \mathbf{h}^a} \sqrt{(\mathbf{h}^b)^\top \mathbf{\Sigma}^{bb} \mathbf{h}^b}}, \\
\text{s.t.} \quad & \|\mathbf{z}^a\|_2 = 1, \quad \|\mathbf{z}^b\|_2 = 1.
\end{aligned} \tag{3.2}$$

Alternatively, since

$$\begin{aligned}
(\mathbf{h}^i)^\top \mathbf{\Sigma}^{ij} \mathbf{h}^j &= (\mathbf{h}^i)^\top \left(\frac{1}{N-1} (\mathbf{Y}^i)^\top \mathbf{Y}^j \right) \mathbf{h}^j \\
&= \frac{1}{N-1} (\mathbf{Y}^i \mathbf{h}^i)^\top \mathbf{Y}^j \mathbf{h}^j \\
&= \frac{1}{N-1} (\mathbf{z}^i)^\top \mathbf{z}^j,
\end{aligned} \tag{3.3}$$

we can rewrite the objective as finding the minimum angle θ between the two canonical variables $\mathbf{z}^a$ and $\mathbf{z}^b$,

$$\max_{\mathbf{z}^a, \mathbf{z}^b} \overbrace{\frac{(\mathbf{z}^a)^\top \mathbf{z}^b}{\sqrt{(\mathbf{z}^a)^\top \mathbf{z}^a} \sqrt{(\mathbf{z}^b)^\top \mathbf{z}^b}}}^{\cos\theta} = \max_{\mathbf{z}^a, \mathbf{z}^b} (\mathbf{z}^a)^\top \mathbf{z}^b. \tag{3.4}$$

This second formulation has a nice geometric interpretation. We want projections such that our embedded data are vectors pointing in the same direction in lower-dimensional space, i.e., are near each other (Figure 3.2.1).

This CCA objective in Equation (3.2) applies to the first pair of canonical variables. If we would like to find a second pair, an additional constraint is that these must be orthogonal to the first pair. In general, for the ith and jth pair of canonical variables, we must have:

$$\left(\mathbf{z}^a_i\right)^\top \mathbf{z}^a_j = 0, \quad \left(\mathbf{z}^b_i\right)^\top \mathbf{z}^b_j = 0. \tag{3.5}$$

Finally, the number of pairs of canonical variables, call this R, cannot be greater than the minimum of P^a and P^b:

$$R \coloneqq \min(P^a, P^b). \tag{3.6}$$

Putting this objective and the constraints in one place, we have

$$\begin{gathered}(\mathbf{z}^a)^\star, \left(\mathbf{z}^b\right)^\star = \underset{\mathbf{z}^a, \mathbf{z}^b}{\operatorname{argmax}} \left\{ (\mathbf{z}^a)^\top \mathbf{z}^b \right\}, \\ \|\mathbf{z}^a\|_2 = \sqrt{(\mathbf{h}^a)^\top \mathbf{\Sigma}^{aa} \mathbf{h}^a} = 1, \quad \left\|\mathbf{z}^b\right\|_2 = \sqrt{(\mathbf{h}^b)^\top \mathbf{\Sigma}^{bb} \mathbf{h}^b} = 1, \\ (\mathbf{z}_i^a)^\top \mathbf{z}_j^a = 0, \quad \left(\mathbf{z}_i^b\right)^\top \mathbf{z}_j^b = 0, \qquad \forall j \neq i : i, j \in \{1, 2, \dots, R\}.\end{gathered} \tag{3.7}$$

With this notation, we define the goal of CCA as finding R linear projections, $(\mathbf{h}_r^a, \mathbf{h}_r^a)$ for $r \in \{1, 2, \dots, R\}$, that satisfy the above constraints. If we pack these vectors into a matrix, we get the matrices $\mathbf{H}^a \in \mathbb{R}^{P^a \times N}$ and $\mathbf{H}^b \in \mathbb{R}^{P^b \times N}$.

The solution to this optimization problem can be found analytically by solving the standard eigenvalue problem [Hotelling, 1936, Hardoon et al., 2004]. The geometric interpretation is that we estimate two linear maps that project both views into a shared subspace. See Section 3A.1 for a discussion of how to solve for the canonical variables in this objective.

3.2.3. Probabilistic CCA

A probabilistic interpretation of CCA (P-CCA) extends these ideas to a model that shares properties with factor analysis and PCA. P-CCA as proposed by Bach and Jordan [2005] can be written as

$$\begin{aligned}\mathbf{z}_n &\overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}), \\ \mathbf{y}_n^a &\sim \mathcal{N}(\mathbf{W}^a \mathbf{z}_n, \mathbf{\Psi}^a), \\ \mathbf{y}_n^b &\sim \mathcal{N}(\mathbf{W}^b \mathbf{z}_n, \mathbf{\Psi}^b),\end{aligned} \tag{3.8}$$

where $\mathbf{\Psi}^a$ and $\mathbf{\Psi}^b$ are full-rank matrices. This property is important because it allows for P-CCA to model variation that is *not* shared, i.e. view-specific variation. A different representation of this same model is sometimes called *inter-battery factor analysis* (IBFA) [Browne, 1979]:

$$
\begin{aligned}
\mathbf{z}_n^c &\overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}), \\
\mathbf{z}_n^a, \mathbf{z}_n^b &\overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}), \\
\mathbf{y}_n^a &\sim \mathcal{N}(\mathbf{\Lambda}^a \mathbf{z}_n^c + \mathbf{B}^a \mathbf{z}_n^a, \mathbf{\Psi}^a), \\
\mathbf{y}_n^b &\sim \mathcal{N}(\mathbf{\Lambda}^b \mathbf{z}_n^c + \mathbf{B}^b \mathbf{z}_n^b, \mathbf{\Psi}^b),
\end{aligned}
\tag{3.9}
$$

where now $\mathbf{\Psi}^j := (\sigma^j)^2 \mathbf{I}$ and where $\mathbf{\Lambda}^j \in \mathbb{R}^{P^j \times K}$, $\mathbf{B}^j \in \mathbb{R}^{P^j \times K}$, and $\mathbf{\Psi}^j \in \mathbb{R}^{P^j \times P^j}$. Note that we have isotropic noise, and therefore the *view-specific* latent variables $\mathbf{z}_n^a$ and $\mathbf{z}_n^b$ account for view-specific variation. The *shared* latent variable $\mathbf{z}_n^c$ captures shared variation (covariation) across the two views.

Note that P-CCA can be viewed as factor analysis with appropriately tiled data and parameters,

$$
\begin{aligned}
\mathbf{y} &= \begin{bmatrix} \mathbf{y}^a \\ \mathbf{y}^b \end{bmatrix} & \mathbf{z} &= \begin{bmatrix} \mathbf{z}^c \\ \mathbf{z}^a \\ \mathbf{z}^b \end{bmatrix} \\
\mathbf{\Lambda} &= \begin{bmatrix} \mathbf{\Lambda}^a & \mathbf{B}^a & \mathbf{0} \\ \mathbf{\Lambda}^b & \mathbf{0} & \mathbf{B}^b \end{bmatrix} & \mathbf{\Psi} &= \begin{bmatrix} \mathbf{\Psi}^a & \mathbf{0} \\ \mathbf{0} & \mathbf{\Psi}^b \end{bmatrix},
\end{aligned}
\tag{3.10}
$$

This immediately suggests EM for inference in P-CCA, using the EM parameter updates for factor analysis given the above tiling [Ghahramani et al., 1996]:

$$
\begin{aligned}
\mathbf{\Lambda}^\star &= \sum_n \left(\mathbf{y}_n \mathbb{E}_{\mathbf{z} \mid \mathbf{y}_n} \left[\mathbf{z} \mid \mathbf{y}_n \right]^\top \right) \left(\mathbb{E}_{\mathbf{z} \mid \mathbf{y}_n} \left[\mathbf{z}\mathbf{z}^\top \mid \mathbf{y}_n \right] \right)^{-1} \\
\mathbf{\Psi}^\star &= \sum_i \frac{1}{N} \operatorname{diag} \left(\mathbf{y}_n \mathbf{y}_n^\top - \mathbf{\Lambda}^\star \mathbb{E}_{\mathbf{z} \mid \mathbf{y}_n} \left[\mathbf{z} \mid \mathbf{y}_n \right] \mathbf{y}_n^\top \right).
\end{aligned}
\tag{3.11}
$$

See Section 3A.3 for detailed derivations for EM for Equation (3.8). In this framing, $\mathbf{y} \in \mathbb{R}^P$ where $P := P^a + P^b$ and $\mathbf{z} \in \mathbb{R}^K$ where $K := K^c + K^a + K^b$, the dimensions of $\mathbf{z}^c$, $\mathbf{z}^a$, and $\mathbf{z}^b$ respectively. Thus, $\mathbf{y} \in \mathbb{R}^P$, $\mathbf{\Lambda} \in \mathbb{R}^{P \times K}$, and $\mathbf{\Psi} \in \mathbb{R}^{P \times P}$. In contrast to CCA, P-CCA does not constrain the latent variables to be orthonormal. See Section 3A.2 for further discussion.

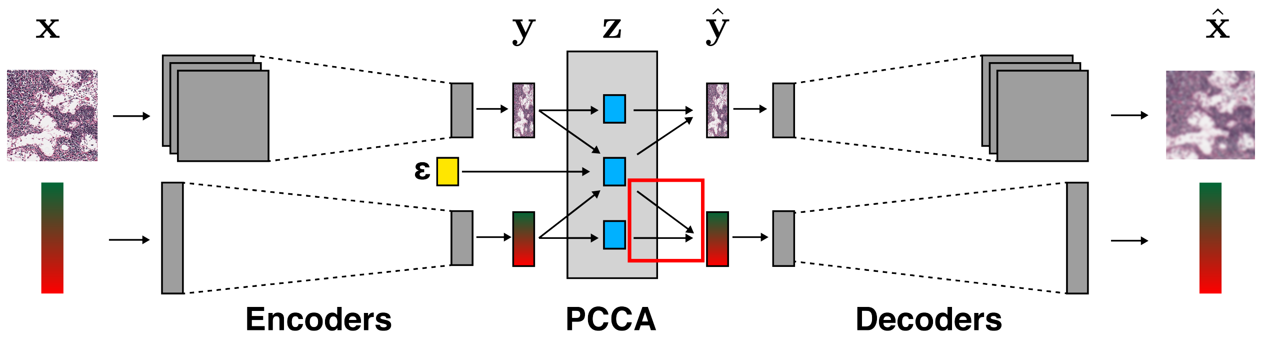

Figure 3.3.1: The input is a paired set of histology images and gene expression levels. The model is trained by fitting P-CCA to embeddings from convolutional (images) and linear (gene expression) autoencoders (gray). Then we sample from the P-CCA model using the reparameterization trick ($\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ is in yellow), and then we backpropagate through the model using the reconstruction loss. The model learns shared and modality-specific latent variables (blue). Sparsity is induced on the P-CCA parameters for the gene expression levels (red box).

3.3 End-to-end training of DP-CCA

DP-CCA is a deep generative model that fits P-CCA to the embeddings of two autoencoders. Additionally, the model has an ℓ_1 penalty on the P-CCA gene weights ($\boldsymbol{\Lambda}^b$ and $\mathbf{B}^b$ in Equation (3.9)) to encourage sparsity in the factors for the gene expression levels. The DP-CCA model is trained end-to-end with backpropogation through the reconstruction loss (Figure 3.3.1). (See Section 3A.4 for a discussion of the backpropagation algorithm [Rumelhart et al., 1986].)

In detail, given a paired sample $(\mathbf{x}_n^a, \mathbf{x}_n^b)$, each encoder $E^j(\cdot)$ with parameters $\mathbf{W}_e^j$ embeds its respective views into a vector, $\mathbf{y}_n^j \in \mathbb{R}^{Pj}$. Each embedding is view-specific: here we use a convolutional encoder for the images and a linear projection for the genes. The embedded vectors $\mathbf{y}_n^a$ and $\mathbf{y}_n^b$ are then fit by P-CCA using the parameter updates in Equation (3.11) with a sparsity-inducing prior on the gene-specific parameters. This results in shared and view-specific latent variables $\mathbf{z}_n = \begin{bmatrix} \mathbf{z}_n^c & \mathbf{z}_n^a & \mathbf{z}_n^b \end{bmatrix}^\top$. Embedded samples $\hat{\mathbf{y}}_n^j$ are obtained from the generative process of the model through sampling from the low-dimensional P-CCA representation $\hat{\mathbf{y}}_n^j \sim \mathcal{N}(\boldsymbol{\Lambda}^{j\star}\mathbf{z}_n^c + \mathbf{B}^{j\star}\mathbf{z}_n^j; \boldsymbol{\Psi}^{j\star})$ using the reparameterization trick similar to Kingma and Welling [2013]. This reparameterization is needed so that the Monte Carlo estimate of the expectation is differentiable with respect to the encoders' parameters

Algorithm 2: End-to-end training of DP-CCA

```
1: Initialize P-CCA parameters, image encoder and decoder parameters, and gene encoder
   and decoder parameters ($\mathbf{\Lambda}$, $\mathbf{\Psi}$, $\mathbf{W}_e^a$, $\mathbf{W}_d^a$, $\mathbf{W}_e^b$, $\mathbf{W}_d^b$).
2: while epoch < # epochs do
3:    For $M$ paired samples, $\mathcal{B} = \{(\mathbf{x}_m^a, \mathbf{x}_m^b)\}_{m=1}^M$.
4:    for $(\mathbf{x}^a, \mathbf{x}^b) \in \mathcal{B}$ do
5:       Encode the $j$th view as $E^j(\mathbf{x}^j) \to \mathbf{y}^j$.
6:       Compute $\mathbf{\Lambda}^\star$ and $\mathbf{\Psi}^\star$ using Equation (3.11).
7:       Sample $\hat{\mathbf{y}}^j \sim \mathcal{N}(\mathbf{\Lambda}^{j\star}\mathbf{z}^c + \mathbf{B}^{j\star}\mathbf{z}^j; \mathbf{\Psi}_j^\star)$ using the reparameterization trick.
8:       Decode the $j$th view as $D^j(\hat{\mathbf{y}}^j) \to \hat{\mathbf{x}}^j$.
9:    end for
10:   Compute loss (Equation (3.13)) and backpropagate to compute $\nabla\mathcal{L}_{\mathbf{\Theta}}$.
11: end while
```

(see Section 3A.6 for a discussion).

Each sampled P-CCA embedding $\hat{\mathbf{y}}_n^a$ and $\hat{\mathbf{y}}_n^b$ is then decoded into reconstructions $\hat{\mathbf{x}}_n^a$ and $\hat{\mathbf{x}}_n^b$ using view-specific decoders with parameters $\mathbf{W}_d^j$ (Figure 3.3.1). Finally, let $\mathcal{L}$ be the reconstruction loss and $\mathbf{\Theta}$ be both the P-CCA and neural network parameters, or

$$\mathbf{\Theta} = \{\mathbf{\Lambda}, \mathbf{\Psi}\} \cup \{\mathbf{W}_e^j, \mathbf{W}_d^j\}_{j\in\{a,b\}}. \tag{3.12}$$

To estimate the parameters $\mathbf{\Theta}$, we perform stochastic gradient descent, where the gradient at each step is $\nabla_{\mathbf{\Theta}}\mathcal{L}$ with

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}\left(\|\hat{\mathbf{x}}_n^a - \mathbf{x}_n^a\|_2^2 + \|\hat{\mathbf{x}}_n^b - \mathbf{x}_n^b\|_2^2\right) + \gamma\left(\|\mathbf{\Lambda}^b\|_1 + \|\mathbf{\Lambda}^{bc}\|_1\right). \tag{3.13}$$

The hyperparameter γ is the ℓ_1 coefficient. This procedure is summarized in Algorithm 2.

3.4 Experiments

In this section, we explore the strengths of our model in two settings: an expanded version of the MNIST handwritten digit data [LeCun et al., 2010], and the GTEx v6 data [Consortium et al., 2017, Carithers et al., 2015] that includes publicly available paired histology images and gene expression data. We implemented our model in PyTorch [Paszke et al., 2017] and

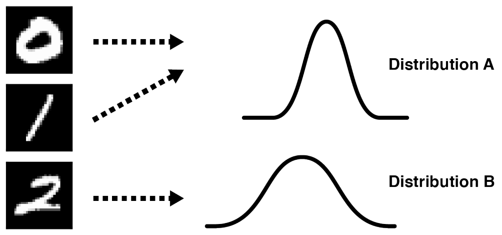

Figure 3.4.1: Multimodal MNIST: Each image from one of three classes $(0, 1, 2)$ is paired with a continuous random variable (*pseudogene*) drawn from one of two multivariate normal distributions with separate means and separate diagonal covariance matrices. Digits of 0s and 1s are paired with samples from the first distribution. Digits of 2s are paired with samples from the second distribution.

Table 3.4.1: Baseline experiments comparing an image-only autoencoder (AE), a multimodal autoencoder (MAE), P-CCA, and DP-CCA on image and pseudogene reconstructions of multimodal MNIST. Each error is an average of five independent trials; standard deviations are shown parenthetically. Our method performs comparably to an MAE and outperforms P-CCA at reconstructing both views.

	Image MSE	Pseudogene MSE
Image AE	0.0196 (0.0019)	NA
MAE	0.0435 (0.0015)	2.287 (0.0117)
P-CCA	0.1207 (0.0032)	33.749 (0.648)
DP-CCA	0.0518 (0.0121)	2.3098 (0.0137)

used the Adam optimizer for all experiments [Kingma and Ba, 2014]. Our code is available online[1] to encourage more work in this important area.

3.4.1. Baselines and multimodal MNIST

We first wanted to study the performance of our model, and compare our results with results from related work using a simple data set. To do this, we built a *multimodal MNIST* data set using the MNIST handwritten digits. MNIST consists of 60,000 training and 10,000 testing images, each with 28×28 pixels with values ranging between 0 (black) and 255 (white). The images are handwritten digits between 0 and 9 and have corresponding class labels in $\{0, 1, \ldots 9\}$.

We augmented MNIST in the following way[2]. First, we removed all images with labels not

[1]https://github.com/gwgundersen/dpcca
[2]See `data/mnist/generate.py` in the repository.

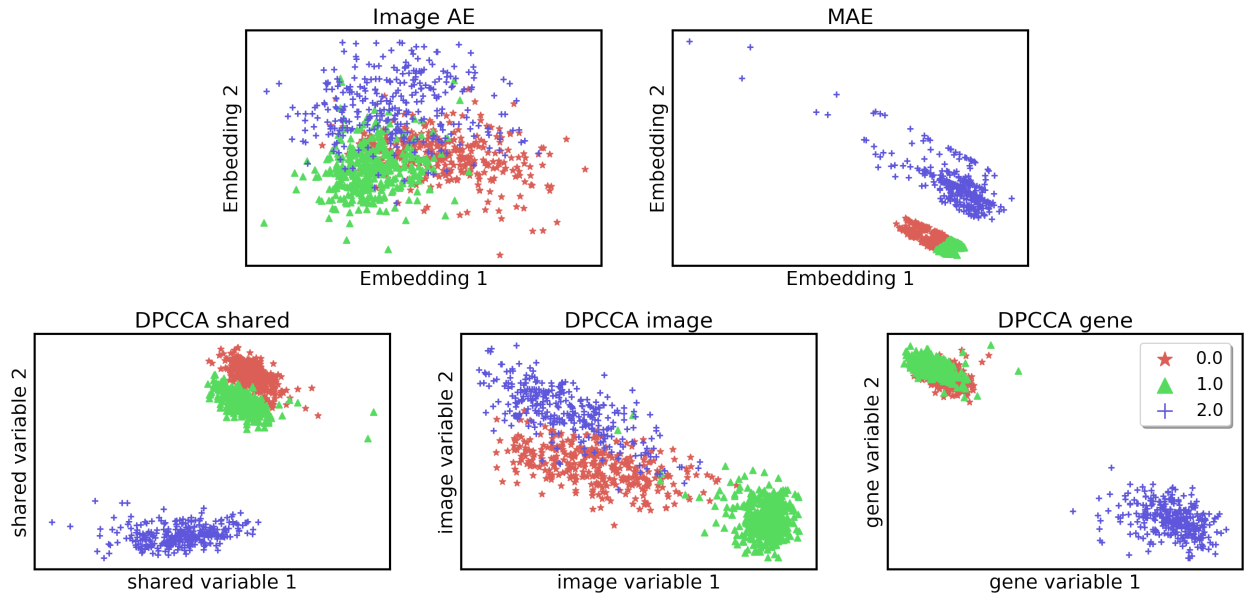

Figure 3.4.2: Two-dimensional embeddings from models trained on multimodal MNIST. (Top row) Embeddings from an autoencoder (left) and multimodal autoencoder (right). (Bottom row) The shared (left), image-specific (center), and gene-specific (right) embeddings from DP-CCA.

in $[0-2]$. For each remaining image, we created an associated *pseudogene* expression vector by sampling from one of two multivariate normal distributions, depending on the image label. The distributions had separate means and separate diagonal covariance matrices. If the image was a 0 or 1, we sampled from the first distribution. If the image was a 2, we sampled from the second distribution (Figure 3.4.1).

Our model should ideally be able to reconstruct both modalities using the latent variables, including the modality-specific variation and the shared variation. We can also examine $\mathbf{Z}$ to ensure that it captures the shared information we encoded in the data: namely, the relationship between the (0,1) images and the 2 images with their respective multivariate normal distributions rather than image digit label, which, for (0,1), are not distinguished by pseudogenes.

As baselines, we fit a single-view autoenoder (AE) on just images, a multimodal autoencoder (MAE) on both data views, and standard P-CCA to both data views. We found that DP-CCA can reconstruct both modalities well relative to these baselines (Table 3.4.1). The AE and MAE are better than DP-CCA at reconstruction, which is expected since our method also must optimize P-CCA in an inner loop. Standard P-CCA performs worse in reconstructing both views. However, neither the MAE nor the AE incorporate both shared and view-specific latent variables, which is crucial to the interpretability of our framework.

Second, we found that the shared and view-specific latent variables contained appropriate shared and view-specific information. To illustrate this, we compared the latent space of DP-CCA to the embeddings of the single-view AE and an MAE (Figure 3.4.2, top). In this experiment, we set K, the dimensionality of the embeddings, to 2 because we have empirically found that the AE with two-dimensional embeddings can reconstruct MNIST well. Recall that, in our model, $\mathbf{z} = \begin{bmatrix} \mathbf{z}^c & \mathbf{z}^a & \mathbf{z}^b \end{bmatrix}^\top$. DP-CCA's shared latent variables $\mathbf{z}^c$ primarily capture the relationship between the two views rather than distinguishing between digits (Figure 3.4.2, bottom left). For comparison, the AE trained on images alone distinguishes digit label, while the MAE captures the shared view without distinguishing 0s and 1s (Figure 3.4.2, top).

We compared the shared and view-specific latent variables of our model to understand the signal captured by each set. The shared latent variables do not distinguish digits 0 and 1 but instead distinguish 0s and 1s versus 2s (Figure 3.4.2, top right). The image-specific latent variables capture information that distinguishes the three digits; this makes sense since the MNIST images are digits (Figure 3.4.2, bottom center). This view is similar to the image-only AE. The pseudogene-specific latent variables, like the shared latent variables, do not distinguish 0s and 1s because the pseudogene variables corresponding to both 0s and 1s are drawn from the same distribution (Figure 3.4.2, bottom right). These results suggest that DP-CCA can estimate embeddings that maximize the correlation of the two views, and that together these shared and view-specific embeddings capture meaningful signals and information contained in held-out class labels better than autoencoders alone.

3.4.2. GTEx data

In experiments on the GTEx data [Consortium et al., 2017, Carithers et al., 2015], we wanted to show that our model can be applied to these data, that it captures interesting held-out biological information such as tissue type, and that the shared latent variables model variation in both images and gene expression levels. To show this, we analyzed the latent factors of our model—a "factor" being a row vector of $\mathbf{Z} \in \mathbb{R}^{K' \times n}$ where K' may be K,

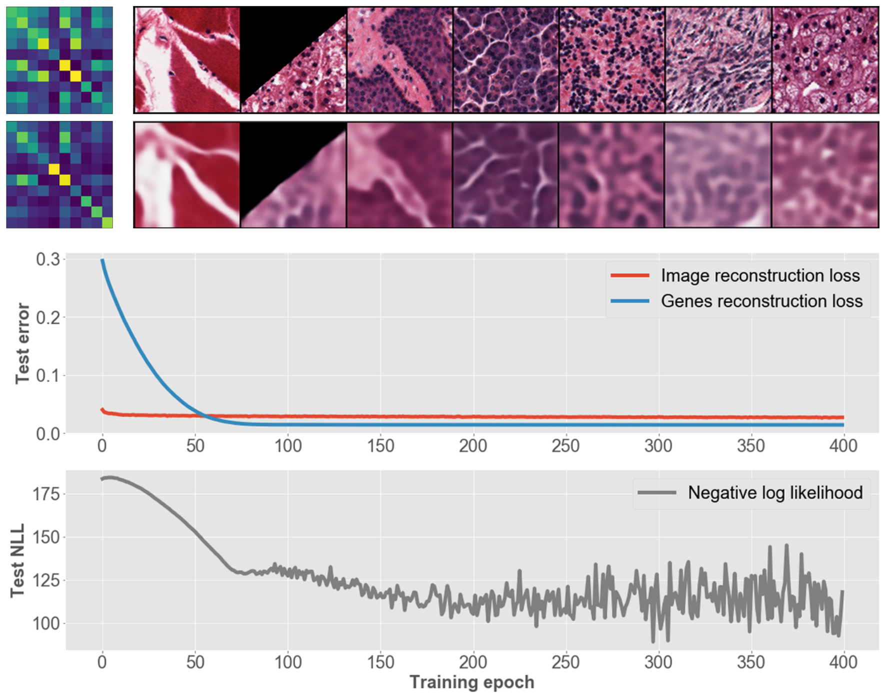

Figure 3.4.3: (Top.) Examples of original data (top row) and reconstructions (bottom row) of gene expression covariance matrices (left) and histology slides (right). For clarity, we show the top 10 columns with the highest variance in the original data. (Bottom.) (Top) View-specific test error over training on the test data from the GTEx data. (Bottom) Negative log likelihood (Test NLL) of DP-CCA over training on the test data from the GTEx data.

K^c, K^a, or K^b depending on context—and found tissue-specific information and variation in images that covaries with changes in factor value. We used held-out genotypes known to be associated with specific genes to identify genotypes associated with tissue morphology using the shared factors. While these results are preliminary from a biological perspective, they are evidence that our model may be a useful tool for joint analysis of paired data.

To this end, we trained our model on 2221 samples from the GTEx v6 study. Each whole tissue slide was subsampled once down to a 1000×1000 pixel RGB image. The crops were chosen as follows. A slide was scanned for tiles in which the mean gray values of the tile and its neighboring tiles were darker than 180 out of 255. The final crop was chosen uniformly at random from suitable tiles. To augment the data, the model was trained on 128×128 pixel crops with random rotations and reflections. The image autoencoder is based on the DCGAN architecture [Radford et al., 2015], and the gene autoencoder is two linear layers. The gene expression measurements are approximately 18,000-dimensional real-valued

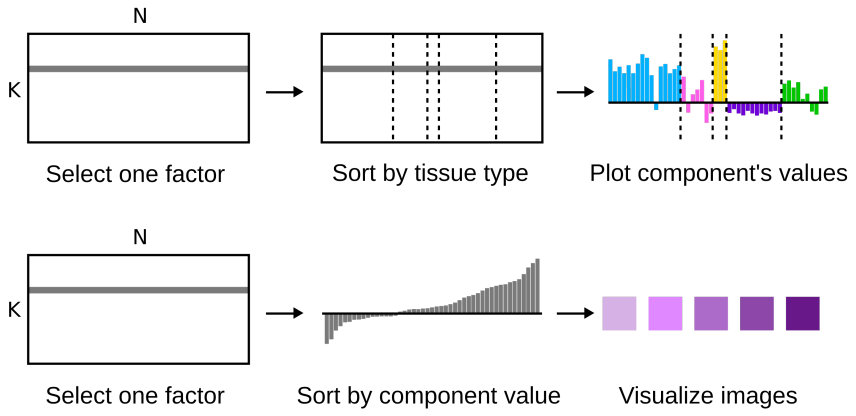

Figure 3.4.4: Analyzing a latent factor. (Top) Given a single N-dimensional latent factor (left), we sort the samples or columns by tissue type (middle) and then plot the value of the factor for each sample. (Bottom) Given a single N-dimensional latent factor (left), we sort the samples based on the factor's value (middle). Then we visualize each sample by its associated histology slide in the same order as the sorted factors.

vectors [Hubbard et al., 2002]. For the number of latent variables for each of the three latent variables types, we swept over values in $\{2, 10, 20, 50, 100, 500\}$ and used the smallest number, 10, that resulted in high-quality image and gene reconstructions. Thus, $K^c = K^a = K^b = 10$ and $K = 30$.

Before using our model for biomedical data analysis, we wanted to verify two important properties. First, we wanted to show that our model could reconstruct both data modalities from a shared latent variable. To show this, we saved reconstructions of the images and reconstructions of the gene covariance matrices during training. We found that our model was able to reconstruct both modalities (Figure 3.4.3, top) and that the test error for both views decreases throughout training (Figure 3.4.3, bottom). This suggests that the shared latent variables carry sufficient information to recapitulate both views.

Second, we wanted to verify our end-to-end training procedure. With a model composed of both neural networks and P-CCA, we might ask whether one of the sub-models is ignored due to an imbalance in the numbers of parameters. To test this, we computed the expected complete negative log-likelihood of held-out test data and found that it decreased over training (Figure 3.4.3, bottom). Taken together, these results suggest that the neural networks and P-CCA are jointly learning parameters for embedding and reconstructing data from nonlinear observations while minimizing the negative log-likelihood of the generative model.

3.4.3. Tissue-specific associations

Next, we wanted to see if our latent factors captured meaningful, held-out biological information: the tissue type of the sample. We did this by sorting the samples (latent variables) by tissue type and plotting the value of a latent factor for each sample (Figure 3.4.4, top). We found that the model's latent factor capture tissue-specific information, and quantified this using a one-sample two-sided t-test (Figure 3.4.5, top). This measures the extent to which the different subsets of the latent variable's factors pull out tissue-specific information.

Our analysis demonstrates that tissue-specific structure is shared across images and genes, and is captured both in the shared factors and also in the gene-specific factors, but less so in the image specific factors. We hypothesize that the tissue-specific signal in images is captured in the shared latent space, which is why no tissue-specific signal is observed in the image-specific latent space. Put differently, there is no tissue-specific variation in the images that is not shared in the genes. Biologically, this makes sense.

3.4.4. Image-specific variation

Next, we wanted to see if DP-CCA captures interpretable morphological information about the images. We did this by visualizing the image associated with each sample after sorting by a single latent factor (Figure 3.4.4, bottom). We found that our model's latent factors capture variations in images that are visible to the human eye (Figure 3.4.5, bottom). In some cases, this variation is a feature of the image itself. For example, cropped images with black chunks are toward one end of the spectrum. But in other cases, this variation is related to tissue morphology. For example, more striated muscle tissue and cerebellar granule cells are both captured by the factor value.

3.4.5. Downstream analysis: Image QTLs

The shared latent variables from our method can be integrated into established genomics pipelines such as quantitative trait loci (QTL) mapping. A cornerstone of quantitative ge-

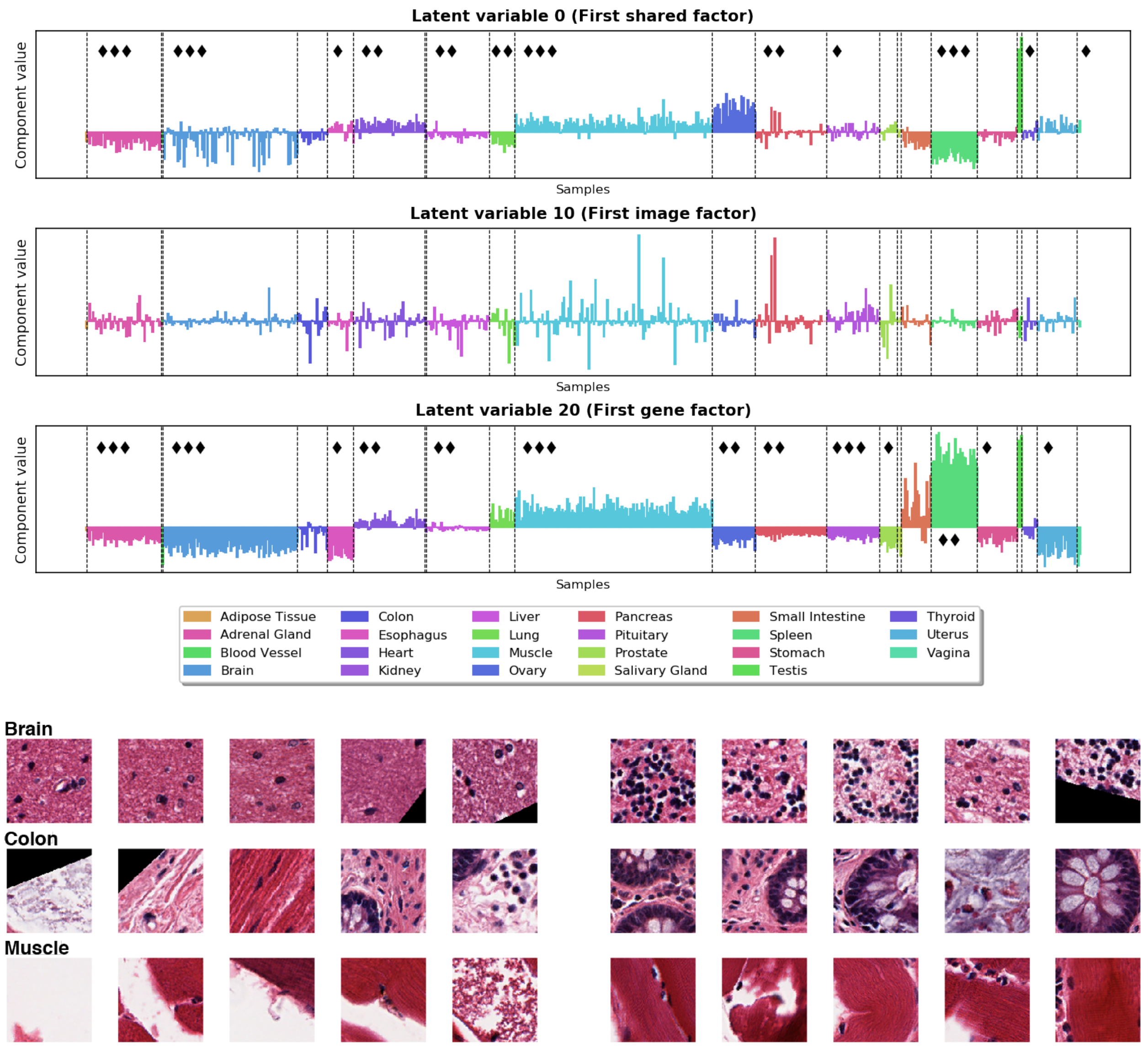

Figure 3.4.5: (Top) Analysis of shared (top row), image-specific (middle row), and gene-specific (bottom row) latent factors. The x-axis (samples) is sorted by tissue type and the y-axis is factor value. We performed a one-sample two-sided t-test on the latent factor values for all samples of the same tissue type. We applied Bonferroni correction to a p-value threshold of 0.05. Tissue samples that reject the null hypothesis are marked with diamonds. We ranked the significant p-values and then uniformly partitioned them from most (3 diamonds) to least (1 diamond) significant. (Bottom) Visualization of the variation in the latent factors. The x-axis (samples) is sorted by factor value, and the images associated with the five most extreme positive and negative values are shown for three tissues.

nomic analysis [Consortium et al., 2017], this method aims to identify associations between genetic variants (genotypes) and quantitative human traits such as height, weight, or gene expression levels (phenotypes), using false discovery rate (FDR)-corrected linear regression

Table 3.4.2: Top genotypes hypothesized to affect the composite phenotype capturing gene expression and morphology across different tissues.

Tissue	SNP	P-value	FDR
Adrenal Gland	chr10_5330574_G_T_b38	3.49E-10	2.40E-3
Adrenal Gland	chr13_33247934_C_T_b38	7.56E-10	3.81E-3
Adrenal Gland	chr9_18386575_C_G_b38	1.65E-9	9.57E-3
Brain Cerebellum	chr1_14376802_A_G_b38	1.82E-10	1.83E-3
Brain Cerebellum	chr9_133727220_C_A_b38	4.43E-10	2.57E-3
Colon Sigmoid	chr10_49206009_T_G_b38	2.02E-10	1.37E-3
Colon Sigmoid	chr12_82520469_A_G_b38	2.29E-9	5.30E-3
Esophagus Mucosa	chr13_68558539_A_T_b38	3.45E-10	4.51E-5
Muscle Skeletal	chr6_150866897_G_A_b38	6.14E-11	5.62E-4
Uterus	chr11_6991683_G_A_b38	1.89E-9	6.20E-3

within each tissue. The shared latent factors estimated using DP-CCA constitute a phenotype describing how the morphology of cells in a tissue (how cells appear) covaries with gene expression levels (characterizing cellular state). These resulting composite phenotypes allow researchers to study the close relationship between a cell's appearance and a cell's state at a macroscopic scale, with the goal of using a cell's appearance to infer its state at a high resolution. QTL analysis takes these ideas one step further to query whether population variation, in the form of differences in genotypes at a particular genetic locus, leads to differences in cellular morphology or cellular state.

To do this, we performed QTL analysis using linear regression (MatrixEQTL) [Shabalin, 2012] with a Benjamini–Hochberg corrected FDR threshold of 0.05 between the 30 composite phenotypes and over $400,000$ genomic loci per tissue across 635 individuals. We found over $20,000$ associations (Table 3.4.2). While validating these associations and elucidating the biological mechanisms behind them is beyond the scope of this work, we note that some of these associations are recurrent in the biological literature. For example, the genotype on chromosome 1 at position 14376802 appears to regulate expression levels of the gene *KAZN* or Kazrin in cerebellum in the brain. This gene has previously been found to affect changes in cell shape across various species [Cho et al., 2011].

3.5 Discussion

In this chapter, we developed a model and associated end-to-end inference method for learning shared structure in paired samples, specifically, histology images and gene expression levels. While our framework combines the power of neural networks for nonlinear embeddings with probabilistic models for interpretable dimension reduction, inference is gradient-based and can be implemented using frameworks leveraging automatic differentiation such as PyTorch [Paszke et al., 2017] and TensorFlow [Abadi et al., 2016].

We demonstrated that the latent factors estimated by DP-CCA revealed tissue-specific structure, despite withholding tissue labels from the model, as well as view-specific structure such as color and tissue attenuation for the images. We further validated our results using QTL analysis.

3A Appendix

3A.1 CCA objective

There are multiple ways to solve for the projections $\mathbf{h}^a$ and $\mathbf{h}^b$. I will discuss the one proposed by Hotelling [1936]. This approach frames the problem as a standard eigenvalue problem and solves for eigenvector $\mathbf{v}_i$ and eigenvalue λ_i given a matrix $\mathbf{A}$:

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i, \tag{3.14}$$

or equivalently, solving the characteristic equation

$$\det(\mathbf{A} - \lambda_i \mathbf{I}) = 0 \quad \text{s.t.} \quad (\mathbf{A} - \lambda_i \mathbf{I})\mathbf{v}_i = \mathbf{0}. \tag{3.15}$$

We can frame this optimization problem using Lagrange multipliers:

$$\mathcal{L} = \overbrace{(\mathbf{h}^a)^\top \boldsymbol{\Sigma}^{ab} \mathbf{h}^b}^{\text{Optimize}} - \overbrace{\frac{\rho_1}{2}((\mathbf{h}^a)^\top \boldsymbol{\Sigma}^{aa} \mathbf{h}^a - 1)}^{\text{Constraint}} - \overbrace{\frac{\rho_2}{2}((\mathbf{h}^b)^\top \boldsymbol{\Sigma}^{bb} \mathbf{h}^b - 1)}^{\text{Constraint}}, \tag{3.16}$$

where ρ_1 and ρ_2 are the Lagrange multipliers, which we divide by 2 to make taking the derivative easier. Taking the derivative of the loss with respect to $\mathbf{h}^a$, we get:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{h}^a} = \frac{\partial}{\partial \mathbf{h}^a}\left[(\mathbf{h}^a)^\top \boldsymbol{\Sigma}^{ab} \mathbf{h}^b\right] - \frac{\partial}{\partial \mathbf{h}^a}\left[\frac{\rho_1}{2}((\mathbf{h}^a)^\top \boldsymbol{\Sigma}^{aa} \mathbf{h}^a - 1)\right] \\ - \frac{\partial}{\partial \mathbf{h}^a}\left[\frac{\rho_2}{2}((\mathbf{h}^b)^\top \boldsymbol{\Sigma}^{bb} \mathbf{h}^b - 1)\right] \end{aligned} \tag{3.17}$$

Let's solve each term separately, and then apply the linearity of differentiation:

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{h}^a}\left((\mathbf{h}^a)^\top \boldsymbol{\Sigma}^{ab}\mathbf{h}^b\right) &= \boldsymbol{\Sigma}^{ab}\mathbf{h}^b, \\
\frac{\partial}{\partial \mathbf{h}^a}\left(\frac{\rho_1}{2}((\mathbf{h}^a)^\top \boldsymbol{\Sigma}^{aa}\mathbf{h}^a - 1)\right) &= \rho_1 \boldsymbol{\Sigma}^{aa}\mathbf{h}^a, \\
\frac{\partial}{\partial \mathbf{h}^a}\left(\frac{\rho_2}{2}((\mathbf{h}^b)^\top \boldsymbol{\Sigma}^{bb}\mathbf{h}^b - 1)\right) &= 0.
\end{aligned} \tag{3.18}$$

Putting it all together, we get:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}^a} = \boldsymbol{\Sigma}^{ab}\mathbf{h}^b - \rho_1 \boldsymbol{\Sigma}^{aa}\mathbf{h}^a = \mathbf{0}. \tag{3.19}$$

It's straightforward to see that the derivative w.r.t. $\mathbf{h}^b$ is

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}^b} = \boldsymbol{\Sigma}^{ba}\mathbf{h}^a - \rho_2 \boldsymbol{\Sigma}^{bb}\mathbf{h}^b = \mathbf{0}. \tag{3.20}$$

Now, note that the Lagrange multipliers ρ_1 and ρ_2 are the same. If we multiply Equation (3.19) by $(\mathbf{h}^a)^\top$, we get:

$$(\mathbf{h}^a)^\top \left(\boldsymbol{\Sigma}^{ab}\mathbf{h}^b - \rho_1 \boldsymbol{\Sigma}^{aa}\mathbf{h}^a\right) = (\mathbf{h}^a)^\top \boldsymbol{\Sigma}^{ab}\mathbf{h}^b - \rho_1 \underbrace{(\mathbf{h}^a)^\top \boldsymbol{\Sigma}^{aa}\mathbf{h}^a}_{\|\mathbf{z}_a\|_2 = 1}. \tag{3.21}$$

The same logic applies if we multiply $\mathbf{h}^b$ by Equation (3.20). Putting these two equations together, we get:

$$\begin{aligned}
0 &= (\mathbf{h}^a)^\top \boldsymbol{\Sigma}^{ab}\mathbf{h}^b - \rho_1, \\
0 &= (\mathbf{h}^b)^\top \boldsymbol{\Sigma}^{ba}\mathbf{h}^a - \rho_2, \\
&\Downarrow \\
\rho_1 &= \rho_2.
\end{aligned} \tag{3.22}$$

So clearly $\rho_1 = \rho_2$. Define $\rho \coloneqq \rho_1 = \rho_2$. Now we have two equations, Equation (3.19) and Equation (3.20), and three unknowns, $\mathbf{h}^a$ and $\mathbf{h}^b$ and ρ. Let's solve for $\mathbf{h}^a$ in Equation (3.19) and then substitute it into Equation (3.20). First, solving for $\mathbf{h}^a$:

$$\begin{aligned} \mathbf{0} &= \mathbf{\Sigma}^{ab}\mathbf{h}^b - \rho\mathbf{\Sigma}^{aa}\mathbf{h}^a \\ \mathbf{h}^a &= \frac{(\mathbf{\Sigma}^{aa})^{-1}\mathbf{\Sigma}^{ab}\mathbf{h}^b}{\rho} \end{aligned} \tag{3.23}$$

Next, substituting $\mathbf{h}^a$ into Equation (3.20) so we have one equation and two unknowns:

$$\begin{aligned} \mathbf{0} &= \mathbf{\Sigma}^{ba}\mathbf{h}^a - \rho\mathbf{\Sigma}^{bb}\mathbf{h}^b \\ &= \mathbf{\Sigma}^{ba}\left(\frac{(\mathbf{\Sigma}^{aa})^{-1}\mathbf{\Sigma}^{ab}\mathbf{h}^b}{\rho}\right) - \rho\mathbf{\Sigma}^{bb}\mathbf{h}^b \\ \rho\mathbf{\Sigma}^{bb}\mathbf{h}^b &= \mathbf{\Sigma}^{ba}\left(\frac{(\mathbf{\Sigma}^{aa})^{-1}\mathbf{\Sigma}^{ab}\mathbf{h}^b}{\rho}\right) \\ \rho^2\mathbf{\Sigma}^{bb}\mathbf{h}^b &= \mathbf{\Sigma}^{ba}(\mathbf{\Sigma}^{aa})^{-1}\mathbf{\Sigma}^{ab}\mathbf{h}^b \end{aligned} \tag{3.24}$$

The standard eigenvalue method assumes $\mathbf{\Sigma}^{bb}$ is invertible, which should hold since $\mathbf{\Sigma}^{bb}$ is a covariance matrix, and we have:

$$\begin{aligned} \rho^2\mathbf{\Sigma}^{bb}\mathbf{h}^b &= \mathbf{\Sigma}^{ba}(\mathbf{\Sigma}^{aa})^{-1}\mathbf{\Sigma}^{ab}\mathbf{h}^b, \\ \rho^2\mathbf{h}^b &= \left((\mathbf{\Sigma}^{bb})^{-1}\mathbf{\Sigma}^{ba}(\mathbf{\Sigma}^{aa})^{-1}\mathbf{\Sigma}^{ab}\right)\mathbf{h}^b. \end{aligned} \tag{3.25}$$

We can see that this is the standard eigenvalue problem where

$$\begin{aligned} \lambda &:= \rho^2, \\ \mathbf{A} &:= (\mathbf{\Sigma}^{bb})^{-1}\mathbf{\Sigma}^{ba}(\mathbf{\Sigma}^{aa})^{-1}\mathbf{\Sigma}^{ab}, \end{aligned} \tag{3.26}$$

which implies that $\mathbf{h}^b$ is an eigenvector that satisfies this equation:

$$\left((\mathbf{\Sigma}^{bb})^{-1}\mathbf{\Sigma}^{ba}(\mathbf{\Sigma}^{aa})^{-1}\mathbf{\Sigma}^{ab} - \rho^2\mathbf{I}\right)\mathbf{h}^b = \mathbf{0}. \tag{3.27}$$

We can solve for $\mathbf{h}^a$ using Equation (3.23) after solving for $\mathbf{h}^b$.

3A.2 CCA versus P-CCA

It is worth thinking about how the properties of CCA are converted to probabilistic assumptions in P-CCA. First, in CCA, $\mathbf{z}^a$ and $\mathbf{z}^b$ are a pair of embeddings that we correlate. The assumption is that both data sets have similar low-rank approximations. In P-CCA, this property is modeled by having a shared latent variable $\mathbf{z}$ or $\mathbf{z}^c$.

Furthermore, in CCA, we require the canonical variables to be orthogonal. In P-CCA, there is no such orthogonality constraint. Instead, we assume the latent variables are independent with an isotropic covariance matrix, $\mathbf{z} \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I})$. This independence assumption is the probabilistic analog to orthogonality.

The final constraint of the CCA objective is that the vectors have unit length. In probabilistic terms, this is analogous to unit variance.

3A.3 EM for linear–Gaussian models

In this section, I show how the EM updates (see Section 2.2.1) for factor analysis (Section 2.2.2) are related to the EM updates for probabilistic PCA [Tipping and Bishop, 1999] and probabilistic CCA [Bach and Jordan, 2005].

3A.3.1. EM for probabilistic PCA

The generative model for probabilistic PCA is

$$\begin{aligned} \mathbf{y}_n &= \mathbf{W}\mathbf{z}_n + \mathbf{u}_n, \\ \mathbf{z}_n &\overset{\text{iid}}{\sim} \mathcal{N}_K(\mathbf{0}, \mathbf{I}), \\ \mathbf{u}_n &\overset{\text{iid}}{\sim} \mathcal{N}_P(\mathbf{0}, \sigma^2\mathbf{I}). \end{aligned} \tag{3.28}$$

The only difference between Equation (3.28) with Equation (2.18) is that the noise covariance in probabilistic PCA is isotropic. Therefore, the updates for $\mathbf{W}$ are exactly the same as in factor analysis:

$$\mathbf{W}^\star = \mathbf{Y}\mathbf{S}^\top\mathbf{A}^{-1}. \tag{3.29}$$

However, we need a slightly different derivation for σ^2, since it is a scalar. In probabilistic PCA, the expected complete log likelihood is

$$Q = \sum_{n=1}^{N} \left[-\frac{1}{2\sigma^2}\mathbf{y}_n^\top \mathbf{y}_n + \frac{1}{\sigma^2}\mathbb{E}[\mathbf{z}_n]^\top \mathbf{W}^\top \mathbf{y}_n - \frac{1}{2\sigma^2}\text{tr}\left(\mathbf{W}^\top \mathbf{W}\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^\top]\right) - \frac{K}{2}\log\sigma^2 \right]. \tag{3.30}$$

Let's take the derivative of Q w.r.t. σ^2, set it equal to zero, and solve for σ^2, using the optimal value of $\mathbf{W}$:

$$(\sigma^2)^\star = \frac{1}{NK}\sum_{n=1}^{N}\left[\mathbf{y}_n^\top\mathbf{y}_n - 2\mathbb{E}[\mathbf{z}_n]^\top[\mathbf{W}^\star]^\top\mathbf{y}_n + \text{tr}\left([\mathbf{W}^\star]^\top\mathbf{W}^\star\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^\top]\right)\right] \tag{3.31}$$

Notice that

$$\begin{aligned} \sum_{n=1}^{N}\text{tr}([\mathbf{W}^\star]^\top\mathbf{W}^\star\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^\top]) &= \text{tr}([\mathbf{W}^\star]^\top\mathbf{W}^\star\sum_{n=1}^{N}\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^\top]) \\ &= \text{tr}([\mathbf{W}^\star]^\top\mathbf{W}^\star\mathbf{A}) \\ &= \text{tr}([\mathbf{W}^\star]^\top\mathbf{Y}\mathbf{S}^\top), \end{aligned} \tag{3.32}$$

and

$$\begin{aligned} \sum_{n=1}^{N}\mathbb{E}[\mathbf{z}_n]^\top[\mathbf{W}^\star]^\top\mathbf{y}_n &= \sum_{n=1}^{N}\text{tr}(\mathbf{y}_n\mathbb{E}[\mathbf{z}_n]^\top[\mathbf{W}^\star]^\top) \\ &= \text{tr}\left(\sum_{n=1}^{N}\mathbf{y}_n\mathbb{E}[\mathbf{z}_n]^\top[\mathbf{W}^\star]^\top\right) \\ &= \text{tr}([\mathbf{W}^\star]^\top\mathbf{Y}\mathbf{S}^\top). \end{aligned} \tag{3.33}$$

So we can write the optimal value for σ^2 as

$$\begin{aligned} (\sigma^2)^\star &= \frac{1}{NK}\sum_{n=1}^{N}\left\{\mathbf{y}_n^\top\mathbf{y}_n - 2\mathbb{E}[\mathbf{z}_n]^\top[\mathbf{W}^\star]^\top\mathbf{y}_n + \text{tr}\left(\mathbf{W}^\top[\mathbf{W}^\star]\mathbb{E}[\mathbf{z}_n\mathbf{z}_n^\top]\right)\right\} \\ &= \frac{1}{NK}\left\{\text{tr}(\mathbf{X}\mathbf{X}^\top) - \text{tr}([\mathbf{W}^\star]^\top\mathbf{Y}\mathbf{S}^\top)\right\} \\ &= \frac{1}{K}\text{tr}\left(\tilde{\mathbf{\Sigma}} - \tilde{\mathbf{\Sigma}}\mathbf{W}\frac{1}{\sigma^2}\mathbf{M}[\mathbf{W}^\star]^\top\right). \end{aligned} \tag{3.34}$$

This matches the EM updates in Tipping and Bishop [1999].

3A.3.2. *EM for probabilistic CCA*

Now we will see that the EM updates for probabilistic CCA are essentially the updates for factor analysis, with some block structure enforced. The generative model for probabilistic CCA is

$$\begin{aligned} \mathbf{z}_n &\overset{\text{iid}}{\sim} \mathcal{N}_K(\mathbf{0}, \mathbf{I}), \quad K \leq \min(P^a, P^b), \\ \mathbf{y}_n^a \mid \mathbf{z}_n &\sim \mathcal{N}_{P^a}(\mathbf{W}^a \mathbf{z}_n, \mathbf{\Psi}^a), \\ \mathbf{y}_n^b \mid \mathbf{z}_n &\sim \mathcal{N}_{P^b}(\mathbf{W}^b \mathbf{z}_n, \mathbf{\Psi}^b). \end{aligned} \tag{3.35}$$

Note that in probabilistic CCA, the covariance matrices $\mathbf{\Psi}^a$ and $\mathbf{\Psi}^b$ are not constrained to be diagonal. If we appropriately tile the data,

$$\mathbf{y}_n = \begin{bmatrix} \mathbf{y}_n^a \\ \mathbf{y}_n^b \end{bmatrix}, \qquad \mathbf{W} = \begin{bmatrix} \mathbf{W}^a \\ \mathbf{W}^b \end{bmatrix}, \qquad \mathbf{\Psi} = \begin{bmatrix} \mathbf{\Psi}^a & \mathbf{0} \\ \mathbf{0} & \mathbf{\Psi}^b \end{bmatrix}, \quad \tilde{\mathbf{\Sigma}} = \begin{bmatrix} \tilde{\mathbf{\Sigma}}^{aa} & \tilde{\mathbf{\Sigma}}^{ab} \\ \tilde{\mathbf{\Sigma}}^{21} & \tilde{\mathbf{\Sigma}}^{bb} \end{bmatrix}, \tag{3.36}$$

where $P := P^a + P^b$, $\mathbf{y}_n \in \mathbb{R}^P$, and $\mathbf{W} \in \mathbb{R}^{P \times K}$, then the EM updates for probabilistic CCA in Bach and Jordan [2005] would be equivalent to those of factor analysis. To see this, the optimal update for $\mathbf{W}$, again given by our formula for factor analysis in Equation (2.43), is

$$\begin{aligned} \mathbf{W}^{\star} &= \mathbf{Y}\mathbf{S}^{\top}\mathbf{A}^{-1} \\ &= \mathbf{Y}(\mathbf{M}\mathbf{W}^{\top}\mathbf{\Psi}^{-1}\mathbf{Y})^{\top}(N\mathbf{M} + \mathbf{S}\mathbf{S}^{\top})^{-1} \\ &= \mathbf{Y}\mathbf{Y}^{\top}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}\left(N\mathbf{M} + \mathbf{M}\mathbf{W}^{\top}\mathbf{\Psi}^{-1}\mathbf{Y}\mathbf{Y}^{\top}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}\right)^{-1} \\ &= \frac{1}{N}\mathbf{Y}\mathbf{Y}^{\top}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}\left(\mathbf{M} + \mathbf{M}\mathbf{W}^{\top}\mathbf{\Psi}^{-1}\frac{1}{N}\mathbf{Y}\mathbf{Y}^{\top}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}\right)^{-1} \\ &= \tilde{\mathbf{\Sigma}}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}\left(\mathbf{M} + \mathbf{M}\mathbf{W}^{\top}\mathbf{\Psi}^{-1}\tilde{\mathbf{\Sigma}}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}\right)^{-1}. \end{aligned} \tag{3.37}$$

In Equation (3.37), we use the fact that $(c\mathbf{A})^{-1} = c^{-1}\mathbf{A}^{-1}$ for any constant c. This works because the block structure of $\mathbf{\Psi}$ ensures we update each block of $\mathbf{W}$ correctly, e.g.:

$$\overbrace{\begin{bmatrix} (\mathbf{W}^a)^{\top} & (\mathbf{W}^b)^{\top} \end{bmatrix}}^{\mathbf{W}^{\top}} \overbrace{\begin{bmatrix} \mathbf{\Psi}^a & \mathbf{0} \\ \mathbf{0} & \mathbf{\Psi}^b \end{bmatrix}}^{\mathbf{\Psi}} = \begin{bmatrix} (\mathbf{W}^a)^{\top}\mathbf{\Psi}^a & (\mathbf{W}^b)^{\top}\mathbf{\Psi}^b \end{bmatrix}. \tag{3.38}$$

The update for $\mathbf{\Psi}$ is trickier only because we need to maintain the diagonal block structure. For $\mathbf{\Psi}_i^{\star}$ with $i \in \{a, b\}$ indexing each data modality, we have:

$$(\mathbf{\Psi}^i)^{\star} = \tilde{\mathbf{\Sigma}}^{ii} - \tilde{\mathbf{\Sigma}}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}^i[\mathbf{W}_i^{\star}]^{\top}. \tag{3.39}$$

In other words, the EM updates for probabilistic CCA immediately fall out of the EM updates for factor analysis:

$$\begin{aligned}
\mathbf{W}^{\star} &:= \tilde{\mathbf{\Sigma}}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}\left(\mathbf{M} + \mathbf{M}\mathbf{W}^{\top}\mathbf{\Psi}^{-1}\tilde{\mathbf{\Sigma}}\mathbf{\Psi}^{-1}\mathbf{W}\mathbf{M}\right)^{-1}, \\
\mathbf{\Psi}^{\star} &:= \begin{bmatrix} \tilde{\mathbf{\Sigma}}^{aa} - \tilde{\mathbf{\Sigma}}^{aa}(\mathbf{\Psi}^a)^{-1}\mathbf{W}^a\mathbf{M}_1[(\mathbf{W}^a)^{\star}]^{\top} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{\Sigma}}^{bb} - \tilde{\mathbf{\Sigma}}^{bb}(\mathbf{\Psi}^b)^{-1}\mathbf{W}^b\mathbf{M}_2[(\mathbf{W}^b)^{\star}]^{\top} \end{bmatrix}.
\end{aligned} \tag{3.40}$$

How I think of this is that by restricting the off-diagonal block matrices to be all zero, we enforce our modeling assumption that there is no correlation between the noise terms for each modality, $\mathbf{Y}^a$ and $\mathbf{Y}^b$.

3A.4 Backpropagation

The goal of backpropagation (backprop) [Rumelhart et al., 1986], a special case of automatic differentiation [Baydin et al., 2018], is to efficiently compute $\partial f / \partial \theta_i$ for every parameter or weight θ_i in a function $f(\cdot)$. Here, I'll assume $f(\cdot)$ is a fully-connected feedforward neural network, and I'll assume the reader is familiar with these models. Please see Goodfellow et al. [2016] if needed. To frame the problem, let's reason about an arbitrary weight θ_1 and node v somewhere in $f(\cdot)$ (Figure 3A.4.1[3]). The node v refers to the output value of the node after passing the weighted sum of its inputs $[t_1, \dots, t_n]$ through an activation function $\sigma(\cdot)$:

$$\begin{aligned}
u &:= \theta_1 t_1 + \theta_2 t_2 + \cdots + \theta_n t_n, \\
v &:= \sigma(u).
\end{aligned} \tag{3.41}$$

[3]Note that each layer in the neural network has the same width n. This is for ease of notation. It would not be hard, although it would be notationally clumsier, to present this material with layer-specific widths.

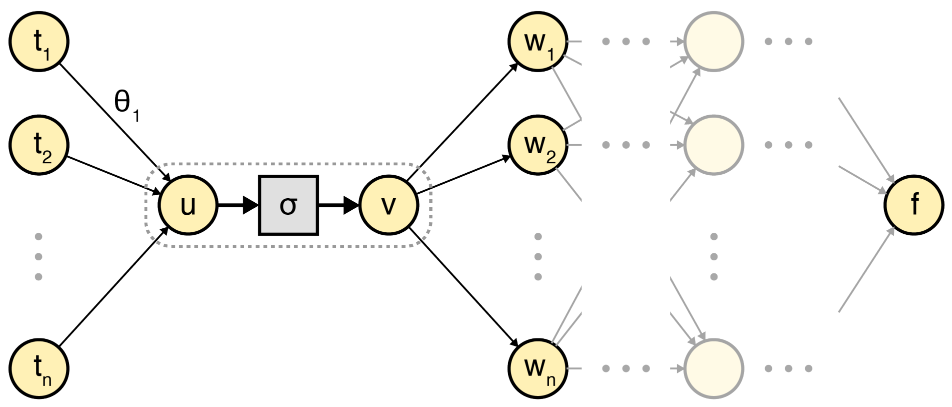

Figure 3A.4.1: A neural network $f(\cdot)$ with an arbitrary node with inputs $[t_1, \dots, t_n]$, outputs $[w_1, \dots, w_p]$, and activation function $\sigma(\cdot)$.

Note that in a typical diagram, and indeed for the other nodes in Figure 3A.4.1, u, $\sigma(\cdot)$, and v would all be a single node, denoted by the dashed line in Figure 3A.4.1. In my mind, the most important observation needed to understand backprop is that most of computing $\partial f / \partial \theta_1$ can be done locally because of the chain rule:

$$\frac{\partial f}{\partial \theta_1} = \frac{\partial f}{\partial v} \frac{\partial v}{\partial u} \frac{\partial u}{\partial \theta_1}. \tag{3.42}$$

We can compute $\partial v / \partial u$ analytically; it just depends on the definition of $\sigma(\cdot)$. And we know that $\partial u / \partial \theta_1 = t_1$ (see Equation (3.41)). So at every node v, if we knew $\partial f / \partial v$, we could compute $\partial f / \partial \theta_1$. The challenge with computing $\partial f / \partial v$ is that downstream nodes depend on the value of v. Intuitively, we can't quantify how a change in v results in a change f without knowing about these downstream interactions (Figure 3A.4.2). As we'll see, we can

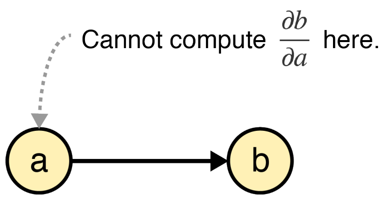

Figure 3A.4.2: Computing the derivative at a single node in a computational graph. The derivative $\partial b / \partial a$ cannot be computed before computing the value of b.

use the multivariable chain rule for this term.

For didactic purposes, let's first consider backpropagation in a "forward pass", meaning computing the gradient of the function from left-to-right in Figure 3A.4.1. We'll see that this is computationally prohibitive.

3A.4.1. Repeated terms

We want a forward propagating algorithm that can compute the partial derivative $\partial f/\partial\theta_1$ for an arbitrary weight θ_1. We showed above that at node v, this is equivalent to:

$$\frac{\partial f}{\partial \theta_1} = \frac{\partial f}{\partial v}\frac{\partial v}{\partial \theta_1}. \tag{3.43}$$

In our setup, for every downstream node w_i that depends on a node v, it is impossible to compute $\partial w_i/\partial v$ at node v. Therefore, in order to compute $\partial f/\partial v$, we must decompose the term using the multivariable chain rule and pass the other terms needed to compute $\partial f/\partial\theta_i$ forward to each node w_i that depends on v:

$$\frac{\partial f}{\partial \theta_i} = \left(\sum_i \frac{\partial f}{\partial w_i}\overbrace{\frac{\partial w_i}{\partial v}}^{\substack{\text{Compute}\\ \text{on } w_i}}\right)\overbrace{\frac{\partial v}{\partial \theta_i}}^{\substack{\text{Pass}\\ \text{forward}}}. \tag{3.44}$$

We can see that such an algorithm blows up computationally because we're forward propagating the same message many times over. For example, if we want to compute $\partial f/\partial\theta_i$ and $\partial f/\partial\theta_k$ where θ_i and θ_k are different weights in the same layer, we need to compute $\partial v/\partial\theta_i$ and $\partial v/\partial\theta_k$ separately, but all the other terms are repeated:

$$\begin{aligned}\frac{\partial f}{\partial \theta_i} &= \overbrace{\left(\sum_j\left(\sum_k \frac{\partial f}{\partial z_k}\frac{\partial z_k}{\partial w_j}\right)\frac{\partial w_j}{\partial v}\right)}^{\text{Repeated terms}}\frac{\partial v}{\partial \theta_i},\\ \frac{\partial f}{\partial \theta_k} &= \left(\sum_j\left(\sum_k \frac{\partial f}{\partial z_k}\frac{\partial z_k}{\partial w_j}\right)\frac{\partial w_j}{\partial v}\right)\frac{\partial v}{\partial \theta_k}.\end{aligned} \tag{3.45}$$

I think this the key insight to backprop: if we already had access to downstream terms, for example $\partial w_j/\partial v$, then we could message pass those terms backwards to node v in order to compute $\partial f/\partial v$. Since each node is just passing its own local term, the backward pass could be done in linear time with respect to the number of nodes.

3A.4.2. A backward pass

I hope this explanation clarifies how you might get to backprop from first principles when trying to compute derivatives in a fully-connected feedforward neural network. On a given node b that depends on a node a, we simply message pass $\partial b/\partial a$ back to a. The multivariable chain rule helps prove the correctness of backprop. For any node v with downstream weights w_j, if v simply sums the messages that are propagating backwards, then it will compute the desired derivative:

$$\frac{\partial f}{\partial v} = \sum_j \frac{\partial f}{\partial w_j} \frac{\partial w_j}{\partial v}. \tag{3.46}$$

Once you understand the main computational problem backprop solves (repeated terms), I think the standard explanation of backpropagating errors makes much more sense. This process is can be viewed as a solution to a kind of credit assignment problem: each node tells its upstream neighbors what they did wrong. But the reason the algorithm works this way is because a naive, forward propagating solution would have quadratic runtime in the number of nodes.

3A.5 Convolutional neural networks

In this appendix section, I assume the reader is familiar with neural networks and how to train them via backpropagation (Section 3A.4). If not, please consult a textbook such as Goodfellow et al. [2016].

Convolutional neural networks (CNNs) [LeCun et al., 1989] are specialized neural networks for detecting visual patterns in images. They are *location invariant*, recognizing a detected pattern regardless of its location in the image, and they are robust to small variations in the detected patterns. They work by automatically learning *image kernels* or sets of shared weights for pattern recognition.

To see why we might want a specialized neural network for computer vision, consider a fully connected neural network with an input of three-channel color images. If each image is just $3 \times 100 \times 100$ pixels (channels by height by width), then a single neuron in the input layer would have $30,000$ weights. By making a few biologically plausible, simplifying assumptions, CNNs dramatically reduce the number of model parameters.

3A.5.1. Convolutions and kernels

As their name suggests, the distinguishing idea behind convolutional neural networks is the convolution operator, which is related to how image kernels work. I'll first present convolutions and kernels and then explain how CNN architectures implement these ideas.

Discrete convolutions in 1-dimension. A *convolution* is a mathematical operation on two functions f and g that outputs a function $(f * g)$ that represents how one function reshapes the other. Since it is sufficient for our purposes, I will only discuss the discrete convolution operator, but Goodfellow et al. [2016] has a more general discussion. A discrete convolution between two single-variable functions f and g, denoted with an asterisk ($*$), is defined as:

$$(f * g)(x) \coloneqq \sum_{i=-\infty}^{\infty} f(i) \cdot g(x - i). \tag{3.47}$$

My intuition is that we are "sliding" the function g across the function f and outputting a new function in the process.

To see this, let's work through an example. Imagine that we have a 1-dimensional input signal with noise,

$$f = \begin{bmatrix} 10 & 9 & 10 & 9 & 10 & 1 & 10 & 9 & 10 & 8 \end{bmatrix}, \tag{3.48}$$

visualized in Figure 3A.5.1 (gray curve). The function f is a mapping from a time point (the zero-based index of the vector) to a signal value. Let the function's output be zero if it is otherwise undefined. If we want to smooth this signal to eliminate deviations like $f(5) = 1$, then we can define a second function g as

$$g \coloneqq \begin{bmatrix} 1/3 & 1/3 & 1/3 \end{bmatrix}, \tag{3.49}$$

and then convolve f with g. We can see that $y = (f * g)$ is a smoothed version of f

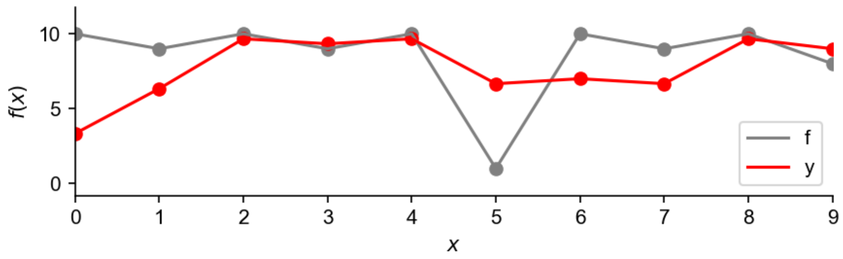

Figure 3A.5.1: A synthetic, 1-dimensional signal f (gray) smoothed via convolution into $y = (f * g)$ (red) with $g := [1/3, 1/3, 1/3]$.

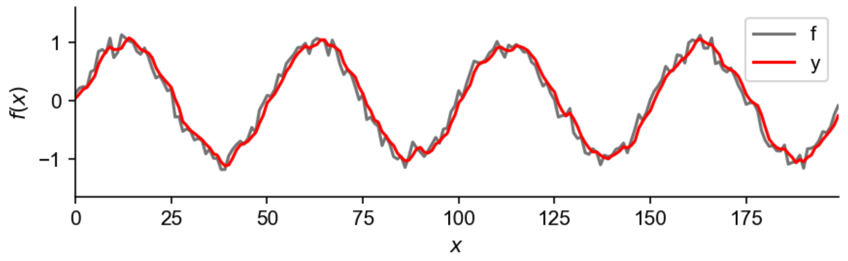

Figure 3A.5.2: A synthetic sine wave f (gray) smoothed via convolution into $y = (f * g)$ (red) with $g := [1/3, 1/3, 1/3]$.

(Figure 3A.5.1, red curve). The graph of y begins at $3\frac{1}{3}$ because g is only smoothing one data point. But once $x = 2$, g sums three elements of f and then divides by 3, i.e. computes the average. Once $f(5) = 1$ is encountered, y dips but not as extremely as f. It takes three steps (left-to-right) for the effects of $f(5)$ to disappear from y because the size of g's domain is three. This smoothing effect is more clear on a larger scale (Figure 3A.5.2).

Here is the Python code used to make Figure 3A.5.1:

```python
import matplotlib.pyplot as plt

# F and G are the two functions we want to convolve.
F = [10, 9, 10, 9, 10, 1, 10, 9, 10, 8]
G = [1./3, 1./3, 1./3]
N = len(F)
Y = list(range(N))

# Convolve F with G.
for x in range(N):
    total = 0
    for i in range(len(F)+1):
        if len(F) > i >= 0 and len(G) > x-i >= 0:
            total += F[i] * G[x-i]
    Y[x] = total

plt.plot(range(N), F, color='gray', label='f')
plt.plot(range(N), Y, color='red', label='y')
plt.show()
```

Discrete convolutions in 2-dimensions. Now let's consider 2-dimensions. The equation for a 2-dimensional discrete convolution is:

$$(f * g)(x_1, x_2) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i, j) \cdot g(x_1 - i, x_2 - j). \tag{3.50}$$

For example, if f and g are

$$f = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \end{bmatrix}, \quad g = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, \tag{3.51}$$

then to convolve f with g at $(x_1, x_2) = (1, 1)$ give us

$$\begin{aligned} (f * g)(1, 1) &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i, j) \cdot g(1 - i, 1 - j) \\ &= f(0, 0)g(1, 1) + f(0, 1)g(1, 0) + f(1, 0)g(0, 1) + f(1, 1)g(0, 0) \\ &= 1e + 2d + 6b + 7a. \end{aligned} \tag{3.52}$$

The sum has only four terms because f and g are defined for only those values. Notice that g has been flipped twice, once over the x-axis and once over the y-axis.[4] The resulting function y is also a matrix.

[4]This flipping is irrelevant for kernels that are symmetric, but it is important to be consistent with the 1-dimensional definition.

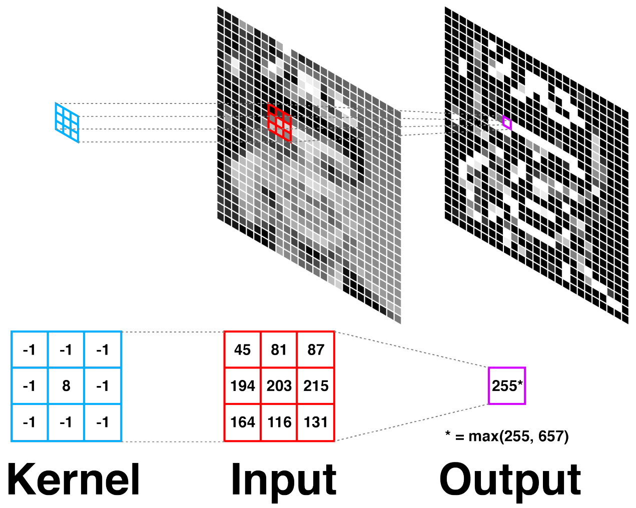

Figure 3A.5.3: Edge detection with a kernel g (blue) defined as in Equation (3.53) on an image of Grace Hopper (a section of the image is shown in red). The output pixel value (purple) is the convolution between the kernel and the image section.

Kernels. Now that we understand convolutions, we can discuss image kernels. A *kernel* is a matrix or function used to process an image. Using the definitions in Equation (3.51), f is a 1-channel image and g is a kernel.

As an example, consider the following function:

$$g := \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}. \tag{3.53}$$

If we convolved this kernel with an image, the resulting image would have its edges highlighted (edge detected). Why? If the sum of a pixel's neighbors' values is roughly equal to the center pixel's value[5], the output will be a number close to 0. If there is a strong difference in pixel values, the output will be positive. The result will be an image of white edges on a black field (Figure 3A.5.3).

The visual effect of convolving a kernel with an image is what we might call *filtering*. I mean nothing deep by this word; it is the same idea as applying a filter in photo-editing

[5]When used in image processing, the convention is that the center of the image kernel g is defined as $(0, 0)$. For symmetric kernels, the convolution operation is equivalent to taking the Hadamard product of an image section and a kernel and summing the values in the resulting matrix.

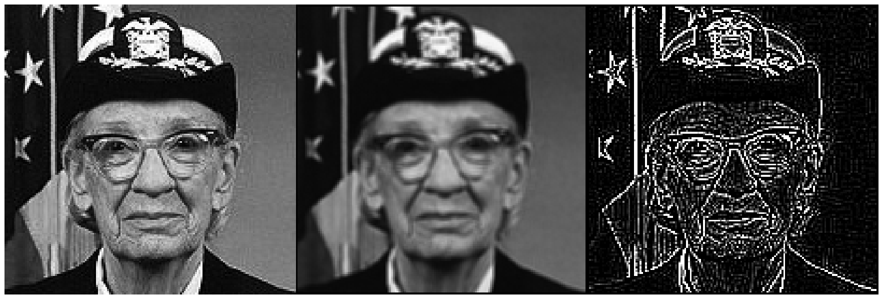

Figure 3A.5.4: Two image kernels convolved with an image of Grace Hopper. From left-to-right: original, blurred or smoothed (a 2D version of Equation (3.49)), and edge-detected.

software. For example, Figure 3A.5.4 shows two kernels applied to a photograph of Grace Hopper. There are many types of kernels and methods to achieve the same effect. I have implemented a few kernels and put them on GitHub[6].

3A.5.2. Properties of image kernels

Before discussing how CNNs learn image kernels, let's discuss two important properties of them.

Location invariance. This idea is visually obvious, but it is worth stating: kernels detect patterns in images in a way that is location invariant, e.g. if a kernel detects edges, it will detect edges anywhere. This property of kernels represents an assumption about the data, but one that makes sense for images and is clearly related to how humans see the world. Later, when we discuss the architecture of CNNs, we will demonstrate how it takes fewer parameters to model this assumption.

Robustness to small variations. The second important idea behind CNNs is *pooling* or *downsampling*. This is a straightforward idea that is explained well in a variety of places, e.g. [Karpathy, 2016]. The basic idea is to compute some function (maximum, minimum, average, etc.) of a region of input neurons and produce a single output neuron (Figure 3A.5.5). Pooling is easy to understand, but there is an important benefit that is good to visualize: pooling discards fluctuations and variations in images.

[6]`https://github.com/gwgundersen/kernels`

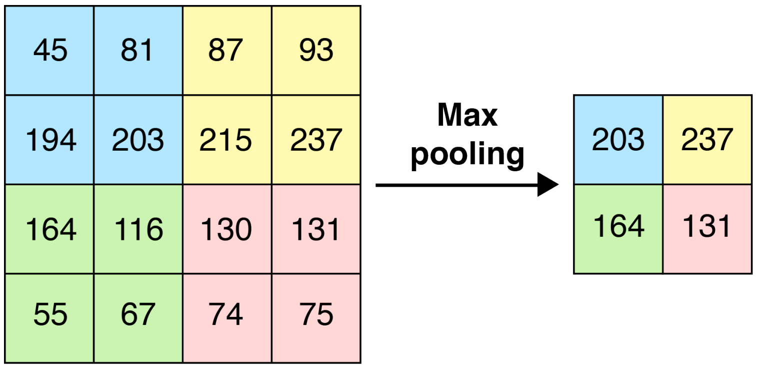

Figure 3A.5.5: Illustration of max pooling. Each section of the image (blue, yellow, green, red) is reduced to a single pixel using the maximum value in each region. *Credit*: I saw a version of this figure in Karpathy [2016].

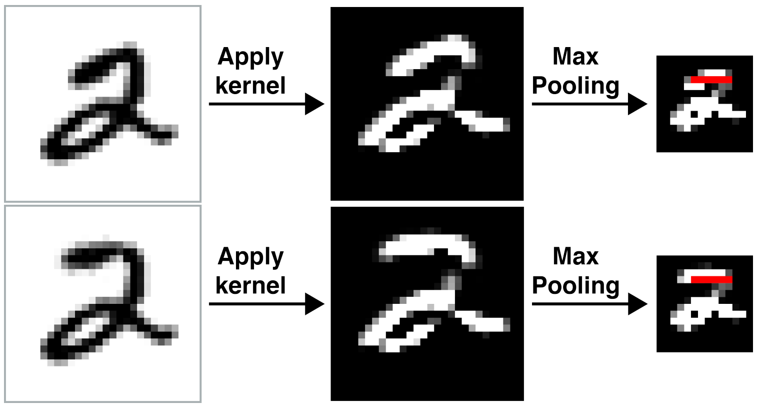

Figure 3A.5.6: Illustration of pooling on (top) an MNIST digit and (bottom) a second MNIST digit that is a modified version of the first MNIST digit.

I want to demonstrate this robustness using an imaginary CNN that can correctly classify MNIST digits [LeCun et al., 2010]. First, imagine our CNN has learned some kernels to correctly classify the digit 2, such as a kernel for detecting horizontal edges. After we convolve this kernel and use max pooling, imagine that we get Figure 3A.5.6 (top). I have drawn a red horizontal bar to indicate one of many patterns that the CNN uses to classify twos. Whenever the CNN detects that red bar, in addition to other patterns of "two-ness", it will correctly classify the digit. Next, imagine we see another 2 digit. In Figure 3A.5.6 (bottom), I have edited the top digit so that the top-left line in the digit is more horizontal than before. What happens? The kernel can still detect the red horizontal line. This is because max pooling reduces the significance of small variations in digits.

This is an example of how CNNs are robust to small changes in patterns. This robustness is due to pooling, not due to more data. Thus, we get this robustness for free. By combining kernels and pooling, CNNs can detect the same patterns anywhere on the visual field and are robust to small variations in those patterns.

Now that we understand kernels, we can succinctly state what CNNs do: CNNs automatically learn kernels from image data in order to extract discriminating patterns. For me, understanding kernels also makes it easier to imagine how someone invented the modern CNN. Previously, image kernels would have been designed by hand, e.g. Denker et al. [1989]. The title of LeCun et al. [1989], "Backpropagation applied to handwritten zip code recognition", makes sense in this context. The authors are proposing to use automatic differentiation to automatically learn these kernels from data, rather than engineering the image kernels by hand.

3A.5.3. Network architecture

Now that we understand kernels and pooling, we are ready to understand the architecture of CNNs. CNNs typically have three types of layers: convolutional, pooling, and fully-connected. Convolutional layers learn kernels for discriminating features and do so in a way that is invariant to translations in the patterns; pooling layers perform downsampling to compress the representation; and the fully-connected layers are typically at the end of the network for classification or dimension reduction, converting low-level image patterns into classification labels or embeddings.

Now given what we know so far, I think the most interesting question is: how do we implement learnable kernels in the convolutional layers? Personally, I found this connection to be the least well-explained aspect of CNNs—or at least most difficult for me to understand—so I'm going to be as explicit as possible. The answer to my question is *shared weights*.

Shared weights. Let's demonstrate how shared weights can be used to model an image kernel. To simplify things, let's move back to 1-dimension. Let our input signal f be:

$$f = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 & 10 & 10 & 10 & 10 & 10 & 7 & 7 & 7 & 7 & 7 \end{bmatrix}. \tag{3.54}$$

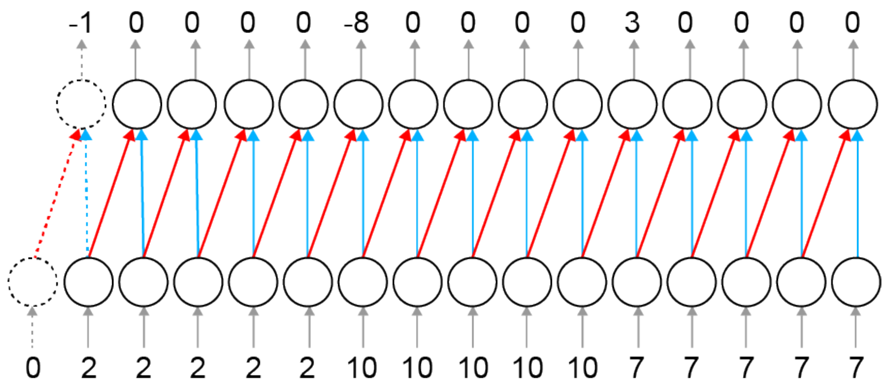

Figure 3A.5.7: A simple two-layer neural network with shared weights. The input (bottom row) is f; the output (top row) is $(f * g)$. Red lines have weights of 1; blue lines have weights of -1. Padding neurons are denoted with dashed lines.

This time, our 1-dimensional kernel g will detect edges:

$$g \coloneqq \begin{bmatrix} -1 & 1 \end{bmatrix}. \tag{3.55}$$

We can express the convolution operator between f and g as a special kind of neural network layer in which the weights are shared. In Figure 3A.5.7, the red weights (left) are 1 and the blue weights (right) are -1. (Remember that the convolution operator inverts g.) The input to the network is f and the output is $y = (f * g)$. In other words, by using shared weights, we have a network layer that is functionally equivalent to convolving a 1-dimension kernel with our input. If this point is unclear, I suggest the reader calculate the convolution by hand to confirm it matches the output of the layer in Figure 3A.5.7. Note that to prevent the domain of y from being smaller than f, we can add an extra input neuron. This is called *padding*. In 2-dimensions, when the origin of function is in the center of the matrix, we would add the padding symmetrically.

Of course, Figure 3A.5.7 is only for a 1-dimensional input. Figure 3A.5.8 is my attempt at a visualization for a 2-dimensional input. Shared weights are the same color. The key point is that the downstream neurons (right) each take an input signal from a $N \times M$ grid (in this case, $N = M = 3$) from the previous layer. The shared weights are functionally equivalent to convolving a kernel across the entire image.

In summary, the set of shared weights that define a convolutional layer are functionally

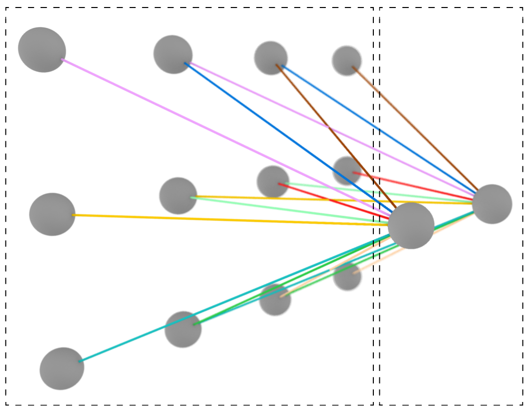

Figure 3A.5.8: 3D visualization of shared weights. Shared weights are the same color.

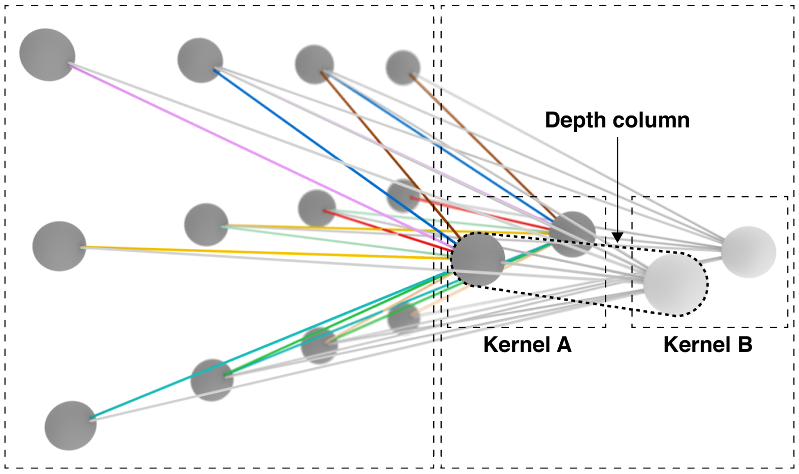

Figure 3A.5.9: 3D visualization of two kernels in the same convolutional layer. Kernel A's weights are shared if they share a color. Kernel B's weights have no color for ease of visualization.

equivalent to the kernel used in traditional image processing. They allow the CNN to detect features regardless of their location in an image and reduce the number of parameters that must be learned by the network.

Layers as volumes. It would be useful to for a CNN to learn a set of shared weights (a set of kernels) for a single layer. In fact, convolutional layers are typically discussed as three-dimensional volumes of neurons. The first two dimensions are the plane of the image. The third dimension is the number of kernels that can be learned in a single layer. This is not the easiest thing to diagram, but Figure 3A.5.9 is my attempt. The point is that a single convolutional layer has depth, and that depth indicates the number of kernels that can be learned by that layer. In Figure 3A.5.9, we have a convolutional layer that learns two kernels. The set of neurons all connected to the same region of input is called the *depth column*.

Importantly, RGB images have three layers (instead of channels) of values. Therefore, the input to a CNN is often $W \times H \times 3$ where W is the width, H is the height, and 3 is the depth of the image. In that case, the convolutional layer's depth would be a fourth dimension. For simplicity and consistency, I have only discussed grayscale images.

3A.6 Reparameterization trick

Imagine we want to take the gradient w.r.t. a parameter $\boldsymbol{\theta}$ of the expectation $\mathbb{E}_{p(\mathbf{z})}[f_{\boldsymbol{\theta}}(\mathbf{z})]$, where P is the distribution on $\mathbf{z}$ with probability function $p(\cdot)$. Provided we can differentiate $f_{\boldsymbol{\theta}}(\mathbf{z})$, we can easily Monte Carlo approximate the gradient:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p(\mathbf{z})}[f_{\boldsymbol{\theta}}(\mathbf{z})] &= \nabla_{\boldsymbol{\theta}} \left[\int_{\mathbf{z}} p(\mathbf{z}) f_{\boldsymbol{\theta}}(\mathbf{z}) \mathrm{d}\mathbf{z} \right] \\
&= \int_{\mathbf{z}} p(\mathbf{z}) \left[\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{z}) \right] \mathrm{d}\mathbf{z} \\
&= \mathbb{E}_{p(\mathbf{z})} \left[\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{z}) \right] \\
&\approx \frac{1}{L} \sum_{\ell=1}^{L} \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{z}_{\ell}).
\end{aligned} \tag{3.56}$$

In other words, the gradient of the expectation is equal to the expectation of the gradient. But what happens if our distribution is $P_{\boldsymbol{\theta}}$ with probability function $p_{\boldsymbol{\theta}}(\cdot)$, i.e. the distribution on $\mathbf{z}$ is also parameterized by $\boldsymbol{\theta}$?

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z})}[f_{\boldsymbol{\theta}}(\mathbf{z})] &= \nabla_{\boldsymbol{\theta}} \left[\int_{\mathbf{z}} p_{\boldsymbol{\theta}}(\mathbf{z}) f_{\boldsymbol{\theta}}(\mathbf{z}) \mathrm{d}\mathbf{z} \right] \\
&= \int_{\mathbf{z}} \nabla_{\boldsymbol{\theta}} \left[p_{\boldsymbol{\theta}}(\mathbf{z}) f_{\boldsymbol{\theta}}(\mathbf{z}) \right] \mathrm{d}\mathbf{z} \\
&= \int_{\mathbf{z}} f_{\boldsymbol{\theta}}(\mathbf{z}) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{z}) \mathrm{d}\mathbf{z} + \int_{\mathbf{z}} p_{\boldsymbol{\theta}}(\mathbf{z}) \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{z}) \mathrm{d}\mathbf{z} \\
&= \underbrace{\int_{\mathbf{z}} f_{\boldsymbol{\theta}}(\mathbf{z}) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{z}) \mathrm{d}\mathbf{z}}_{\text{What about this?}} + \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z})} \left[\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{z}) \right].
\end{aligned} \tag{3.57}$$

The first term of the last line of Equation (3.57) is not guaranteed to be an expectation.

Now that we have an understanding of the problem, let's see what happens when we apply the reparameterization trick to our simple example. The basic idea is to sample a new random variable $\boldsymbol{\varepsilon}$ and then parameterize the random variable we care about, in this case $\mathbf{z}$, as a differentiable function of $\boldsymbol{\varepsilon}$ that is also parameterized by $\boldsymbol{\theta}$, call this function $g_{\boldsymbol{\theta}}(\cdot)$:

$$\begin{aligned}
&\boldsymbol{\varepsilon} \sim p(\boldsymbol{\varepsilon}), \\
&\mathbf{z} = g_{\boldsymbol{\theta}}(\boldsymbol{\varepsilon}, \mathbf{y}).
\end{aligned} \tag{3.58}$$

We can now Monte Carlo approximate the gradient. Why? Notice that the expectation of interest can be rewritten as

$$\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z})} \left[f(\mathbf{y}) \right] = \mathbb{E}_{p(\boldsymbol{\varepsilon})} \left[f(g_{\boldsymbol{\theta}}(\boldsymbol{\varepsilon}, \mathbf{y})) \right]. \tag{3.59}$$

So when computing the derivative of this expectation, the integral can "push inside" the expectation as desired because the new expectation is w.r.t. the distribution on $\boldsymbol{\varepsilon}$, not on $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z})}[f(\mathbf{z})] = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{p(\boldsymbol{\varepsilon})} \left[f(g_{\boldsymbol{\theta}}(\boldsymbol{\varepsilon}, \mathbf{y})) \right] \tag{3.60}$$

$$= \mathbb{E}_{p(\boldsymbol{\varepsilon})} \left[\nabla_{\boldsymbol{\theta}} f(g_{\boldsymbol{\theta}}(\boldsymbol{\varepsilon}, \mathbf{y})) \right] \tag{3.61}$$

$$\approx \frac{1}{L} \sum_{\ell=1}^{L} \nabla_{\boldsymbol{\theta}} f(g_{\boldsymbol{\theta}}(\boldsymbol{\varepsilon}_\ell, \mathbf{y})), \qquad \boldsymbol{\varepsilon}_\ell \overset{\text{iid}}{\sim} P_{\boldsymbol{\varepsilon}}. \tag{3.62}$$

In words, we use the reparameterization trick to express a gradient of an expectation as an expectation of a gradient. Provided the function $g_{\boldsymbol{\theta}}(\cdot)$ is differentiable, then we can then use Monte Carlo methods to estimate $\nabla_{\boldsymbol{\theta}}\mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{z})}[f(\mathbf{z})]$.

Chapter 4

Random feature latent variable models

Gaussian process (GP)-based latent variable models [Lawrence, 2004] are flexible and theoretically grounded tools for nonlinear dimension reduction, but generalizing to non-Gaussian data likelihoods within this nonlinear framework is statistically challenging. In this chapter, I present the use random features to develop a family of nonlinear dimension reduction models that are easily extensible to non-Gaussian data likelihoods. We call this family *random feature latent variable models* (RFLVMs). By approximating a nonlinear relationship between the latent space and the observations with a function that is linear with respect to random features, we induce closed-form gradients of the posterior distribution with respect to the latent variables. This allows the RFLVM framework to support computationally tractable nonlinear latent variable models for a variety of data likelihoods in the exponential family without specialized derivations.

4.1 Introduction

Many dimension reduction techniques, such as principal component analysis [Pearson, 1901, Tipping and Bishop, 1999] and factor analysis [Spearman, 1904, Lawley and Maxwell, 1962], make two modeling assumptions: (1) the observations are Gaussian distributed, and (2) the latent structure is a linear function of the observations. However, for many applications, proper analysis requires us to break these assumptions. For example, in computational neuroscience, scientists collect firing rates for thousands of neurons simultaneously. These data are observed as counts, and neuroscientists believe that the biologically relevant latent structure is nonlinear with respect to the data [Cunningham and Byron, 2014].

To capture nonlinear relationships in latent variable models, one approach is to assume that the mapping between the latent manifold and observations is GP-distributed. A GP is a prior over the space of real-valued functions, and posterior inference is tractable when the GP prior is conjugate to the likelihood. This leads to the Gaussian process latent variable model (GPLVM) [Lawrence, 2004].

The basic GPLVM model with a radial basis function (RBF) kernel has nice statistical properties that allow for exact, computationally tractable inference methods to be used when the number of observations is a reasonable size. Deviating from this basic model, however, leads to challenges with inference.

In GPLVMs with Poisson data likelihoods, for example, we cannot integrate out the GP-distributed functional map, and we no longer have closed form expressions for the gradient of the posterior with respect to the latent space. This renders MAP estimation (discussed in Section 2.3.1) difficult, leading to solutions at poor local optima. (See Wu et al. [2017] for a discussion.)

Random Fourier features (RFFs) [Rahimi and Recht, 2008] were developed to avoid working with $N \times N$ dimensional matrices when fitting kernel machines. RFFs accelerate kernel machines by using a low-dimensional, randomized approximation of the inner product associated with a given shift-invariant kernel. For this approximation, RFFs induce a nonlinear map using a linear function of random features.

We propose to use RFFs to approximate the kernel function in a GPLVM to create a flexible, tractable, and modular framework for fitting GP-based latent variable models. In the context of GPLVMs, RFF approximations allow for closed-form gradients of the objective function with respect to the latent variable. Using RFFs solve a fundamental statistical problem with GPLVMs in non-Gaussian settings. With a Gaussian likelihood, we can obtain closed-form gradients by integrating out the GP-distributed maps, but this cannot be done in the non-Gaussian case. Our solution is to induce these closed-form gradients by making the data likelihood depend on the latent variables as a linear function of the random features. In addition, we can tractably explore the space of stationary covariance functions by using a Dirichlet process mixture prior for the spectral distribution of frequencies [Oliva et al., 2016], leading to a flexible latent variable model.

Contributions: This chapter makes the following contributions to the space of nonlinear latent variable models: (1) we represent the nonlinear mapping in GPLVMs using a linear function of random Fourier features; (2) we leverage this representation to generalize GPLVMs to non-Gaussian likelihoods and derive an MCMC sampler for a wide variety of count-data likelihoods, such as the Poisson, binomial, negative binomial, and multinomial distributions; (3) we place a prior on the random features to allow data-driven exploration over the space of shift-invariant kernels, to avoid putting restrictions on the kernel's functional form. While implementing GPs with RFFs has been done before [Lázaro-Gredilla et al., 2007, Hensman et al., 2017, Cutajar et al., 2017], it has always been motivated by *scalability*. However, we motivate RFFs with statistical *tractability* of non-conjugate GPLVMs. Thus, our contributions focus on tractability and generality rather than scalability or state-of-the-art results for specialized models.

We validate our approach on diverse simulated data sets, and show how results from RFLVMs compare with state-of-the-art methods on a variety of image, text, and scientific data sets. We release a Python library[1] with modular code for reproducing and building on our work.

4.2 Background

In this section, I introduce three main ideas that are required for understanding RFLVMs: Gaussian processes (Section 4.2.1), Gaussian process latent variable models (Section 4.2.2), and random feature approximations for kernel methods (Section 4.2.3). The reader can comfortably skip to Section 4.3 or skim this material as desired.

4.2.1. Gaussian processes

Rasmussen and Williams [2006] define a Gaussian process (GP) as:

[1] `https://github.com/gwgundersen/rflvm`

Definition 4.2.1. *A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

This definition is fairly abstract, but we can concretize it with an example. Following the outlines of Rasmussen and Williams [2006] and Bishop [2006], I introduce GPs by using Bayesian linear regression as an example and showing both the weight- and function-space interpretations of this model.

Weight-space view. In standard linear regression, we have

$$y_n = \mathbf{w}^\top \mathbf{x}_n, \tag{4.1}$$

where our predictor $y_n \in \mathbb{R}$ is just a linear combination of the covariates $\mathbf{x}_n \in \mathbb{R}^D$ for the nth sample out of N observations. We can make this model more flexible with M fixed basis functions,

$$f(\mathbf{x}_n) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_n), \tag{4.2}$$

where

$$\boldsymbol{\phi}(\mathbf{x}_n) = \begin{bmatrix} \phi_1(\mathbf{x}_n) \\ \vdots \\ \phi_M(\mathbf{x}_n) \end{bmatrix}. \tag{4.3}$$

Note that in Equation (4.1), $\mathbf{w} \in \mathbb{R}^D$, while in Equation (4.2), $\mathbf{w} \in \mathbb{R}^M$. Now consider a Bayesian treatment of linear regression that places prior on $\mathbf{w}$,

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \alpha^{-1}\mathbf{I}), \tag{4.4}$$

where $\alpha^{-1}\mathbf{I}$ is a diagonal precision matrix. In my mind, Bishop [2006] is clear in linking this prior to the notion of a Gaussian process. He writes in his Section 6.4.1, "For any given value of $\mathbf{w}$, the definition [our Equation (4.2)] defines a particular function of $\mathbf{x}$. The probability distribution over $\mathbf{w}$ defined by [our Equation (4.3)] therefore induces a probability distribution over functions $f(\mathbf{x})$." In other words, if $\mathbf{w}$ is random, then $\mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_n)$ is random as well. This example demonstrates how we can think of Bayesian linear regression as a distribution over functions. Thus, we can either talk about a random variable $\mathbf{w}$ or a random function f induced by $\mathbf{w}$.

In principle, we can imagine that f is an infinite-dimensional function since we can imagine infinite data and an infinite number of basis functions. However, in practice, we are really only interested in a finite collection of N data points. Let

$$\mathbf{y} = \begin{bmatrix} f(\mathbf{x}_1) \\ \vdots \\ f(\mathbf{x}_N) \end{bmatrix} \tag{4.5}$$

and let $\mathbf{\Phi}$ be a matrix such that $\mathbf{\Phi}_{nk} = \phi_k(\mathbf{x}_n)$. Then we can rewrite $\mathbf{y}$ as

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w} = \begin{bmatrix} \phi_1(\mathbf{x}_1) & \dots & \phi_M(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_N) & \dots & \phi_M(\mathbf{x}_N) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_M \end{bmatrix}. \tag{4.6}$$

Recall that if random variables $\mathbf{z}_1, \dots, \mathbf{z}_N$ are independent Gaussian random variables, then the linear combination $a_1\mathbf{z}_1 + \dots + a_N\mathbf{z}_N$ is also Gaussian for every $a_1, \dots, a_N \in \mathbb{R}$, and we say that $\mathbf{z}_1, \dots, \mathbf{z}_N$ are *jointly Gaussian*. Since each component of $\mathbf{y}$ (each y_n) is a linear combination of independent Gaussian-distributed variables ($\mathbf{w} := [w_1, \dots, w_M]^\top$), the components of $\mathbf{y}$ are jointly Gaussian. Therefore, we can uniquely specify the normal distribution of $\mathbf{y}$ by computing its mean vector and covariance matrix, which we can do:

$$\begin{aligned} \mathbb{E}[\mathbf{y}] &= \mathbf{0}, \\ \operatorname{Cov}(\mathbf{y}) &= \frac{1}{\alpha}\mathbf{\Phi}\mathbf{\Phi}^\top. \end{aligned} \tag{4.7}$$

If we define $\mathbf{K} := \operatorname{Cov}(\mathbf{y})$, then we can say that $\mathbf{K}$ is a Gram matrix such that

$$\mathbf{K}_{nm} = \frac{1}{\alpha}\langle \phi(\mathbf{x}_n), \phi(\mathbf{x}_m) \rangle_{\mathcal{V}} = k(\mathbf{x}_n, \mathbf{x}_m) \tag{4.8}$$

where $k(\mathbf{x}_n, \mathbf{x}_m)$ is called a *covariance* or *kernel function*,

$$k : \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}. \tag{4.9}$$

Note that this lifting of the input space into feature space $\mathcal{V}$ by replacing $\mathbf{x}^\top \mathbf{x}$ with $k(\mathbf{x}, \mathbf{x})$ is the kernel trick. (I will discuss kernel methods in more detail in Section 4.2.3.) Also, keep in mind that we did not explicitly choose $k(\cdot, \cdot)$; it simply fell out of the way we setup the problem. In other words, Bayesian linear regression is a specific instance of a Gaussian process, and we will see that we can choose different mean and kernel functions to get different types of GPs.

Rasmussen and Williams's presentation of this section is similar to Bishop's, except they derive the posterior $p(\mathbf{w} \mid \mathbf{x}_1, \dots \mathbf{x}_N)$, and show that this is Gaussian, whereas Bishop relies on the definition of jointly Gaussian. I prefer the latter approach, since it relies more on probabilistic reasoning and less on computation.

Function-space view. Following the outline of Rasmussen and Williams, let's connect the weight-space view from the previous section with a view of GPs as functions. These two interpretations are equivalent, but I found it helpful to connect the traditional presentation of GPs as functions with a familiar method, Bayesian linear regression.

Now, let us ignore the weights $\mathbf{w}$ and instead focus on the function $\mathbf{y} = f(\mathbf{x})$. Furthermore, let's talk about variables $\mathbf{f}$ instead of $\mathbf{y}$ to emphasize our interpretation of functions as random variables. We noted in the previous section that a jointly Gaussian random variable $\mathbf{f}$ is fully specified by a mean vector and covariance matrix. Alternatively, we can say that the function $f(\mathbf{x})$ is fully specified by a *mean function* $m(\mathbf{x})$ and covariance function $k(\mathbf{x}_n, \mathbf{x}_m)$ such that

$$\begin{aligned} m(\mathbf{x}_n) &= \mathbb{E}[y_n] \\ &= \mathbb{E}[f(\mathbf{x}_n)] \end{aligned} \tag{4.10}$$

and

$$\begin{aligned} k(\mathbf{x}_n, \mathbf{x}_m) &= \mathbb{E}[(y_n - \mathbb{E}[y_n])(y_m - \mathbb{E}[y_m])^\top] \\ &= \mathbb{E}[(f(\mathbf{x}_n) - m(\mathbf{x}_n))(f(\mathbf{x}_m) - m(\mathbf{x}_m))^\top]. \end{aligned} \tag{4.11}$$

This is the standard presentation of a Gaussian process, and we denote it as

$$\mathbf{f} \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{4.12}$$

At this point, Definition 4.2.1, which was a bit abstract when presented *ex nihilo*, begins to make more sense. With a concrete instance of a GP in mind, we can map this definition onto concepts we already know. The collection of random variables is $\mathbf{y}$ or $\mathbf{f}$, and it can be infinite because we can imagine infinite or endlessly increasing data. And we have already seen how a finite collection of the components of $\mathbf{y}$ can be jointly Gaussian and are therefore uniquely defined by a mean vector and covariance matrix. We can model more flexible functions by constructing the covariance matrix with different kernel functions.

Since we are thinking of a GP as a distribution over functions, let's sample functions from it (Equation (4.12)). To do so, we need to define mean and covariance functions. Let's use $m : \mathbf{x} \mapsto \mathbf{0}$ for the mean function, and instead focus on the effect of varying the kernel. Consider these three kernels,

$$\begin{aligned}
k(\mathbf{x}_n, \mathbf{x}_m) &= \exp\left\{\frac{1}{2}\|\mathbf{x}_n - \mathbf{x}_m\|_1^2\right\} && \text{squared exponential,} \\
k(\mathbf{x}_n, \mathbf{x}_m) &= \sigma_p^2 \exp\left\{-\frac{2\sin^2(\pi\|\mathbf{x}_n - \mathbf{x}_m\|_1/p)}{\ell^2}\right\} && \text{periodic,} \\
k(\mathbf{x}_n, \mathbf{x}_m) &= \sigma_b^2 + \sigma_v^2(\mathbf{x}_n - c)(\mathbf{x}_m - c) && \text{linear.}
\end{aligned} \tag{4.13}$$

taken from David Duvenaud's *Kernel Cookbook*[2]. Let our data be 400 evenly spaced real numbers between -5 and 5. Note that GPs are often used on sequential data, but it is not necessary to view the index n for $\mathbf{x}_n$ as either a temporal or spatial index nor do our inputs need to be evenly spaced. To sample from the GP, we first build the Gram matrix $\mathbf{K}$. Let K denote the kernel function evaluated on a set of data points rather than a single observation, let $\mathbf{X} \coloneqq \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ be training data, and let $\mathbf{X}_*$ be test data. Then sampling from the

[2]https://www.cs.toronto.edu/~duvenaud/cookbook/

GP prior is simply,

$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, K(\mathbf{X}_*, \mathbf{X}_*)). \tag{4.14}$$

In the absence of data, *test data* is loosely “everything” because we haven’t seen any data points yet. Figure 4.2.1 shows 10 samples of functions defined by the three kernels above.

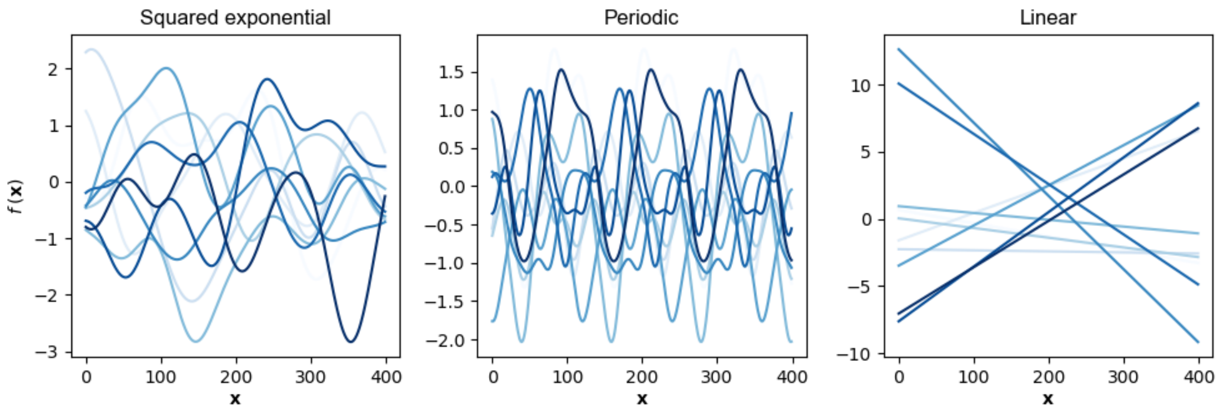

Figure 4.2.1: Ten samples from a GP with a (left) squared exponential kernel, (middle) periodic kernel, and (right) linear kernel. A single sample is a function in which the x-axis is the test data and the y-axis is the predicted value.

In my mind, Figure 4.2.1 makes clear that the kernel is hyperparameter or inductive bias. Given the same data, different kernels specify completely different types of functions.

Prediction. Ultimately, we are interested in prediction or generalization to unseen test data given training data. Intuitively, what this means is that we do not want just any functions sampled from our prior; rather, we want functions that “agree” with our training data (Figure 4.2.2).

There is an elegant solution to this modeling challenge: *conditionally Gaussian random variables.* Recall that a GP is an infinite collection of random variables, any finite number of which are jointly Gaussian. This means the the model of the concatenation of $\mathbf{f}$ and $\mathbf{f}_*$ is

$$\begin{bmatrix} \mathbf{f}_* \\ \mathbf{f} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} K(\mathbf{X}_*, \mathbf{X}_*) & K(\mathbf{X}_*, \mathbf{X}) \\ K(\mathbf{X}, \mathbf{X}_*) & K(\mathbf{X}, \mathbf{X}) \end{bmatrix} \right) \tag{4.15}$$

where for ease of notation, we assume $m(\cdot) = \mathbf{0}$. Using basic properties of multivariate

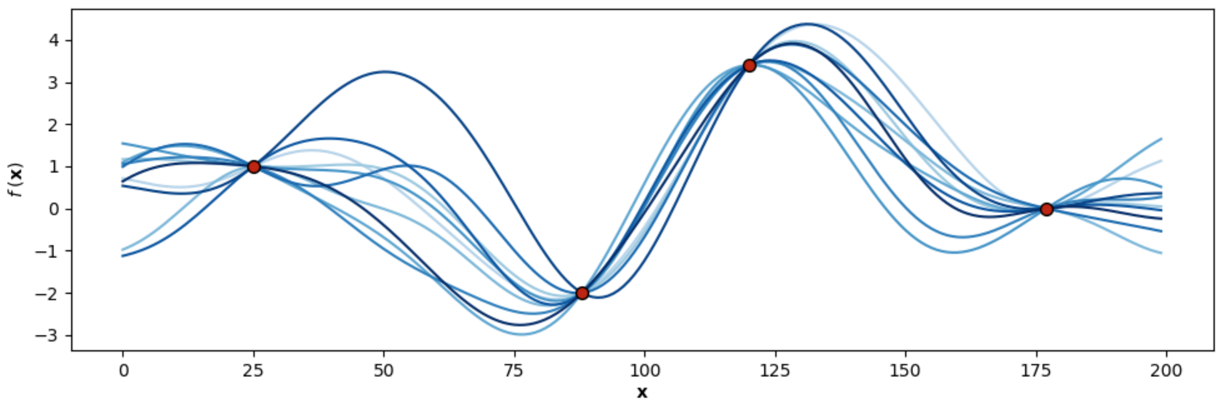

Figure 4.2.2: Ten samples from a GP posterior with a squared exponential kernel and four noise-free observations. A conditional Gaussian distribution has zero variance at the observed data points.

Gaussian distributions, we can compute

$$\begin{aligned}\mathbf{f}_* \mid \mathbf{f} \sim \mathcal{N}(&K(\mathbf{X}_*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}\mathbf{f}, \\ &K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}K(\mathbf{X}, \mathbf{X}_*)).\end{aligned} \tag{4.16}$$

While we are still sampling *random functions* $\mathbf{f}_*$, these functions “agree” with the training data. To see why, consider the scenario when $\mathbf{X}_* = \mathbf{X}$; the mean and variance in Equation (4.15) would be

$$\begin{aligned}\mathbf{f} &= K(\mathbf{X}, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}\mathbf{f}, \\ \mathbf{0} &= K(\mathbf{X}, \mathbf{X}) - K(\mathbf{X}, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}K(\mathbf{X}, \mathbf{X})).\end{aligned} \tag{4.17}$$

In other words, the variance at the training data points is $\mathbf{0}$ (non-random) and therefore the random samples are exactly our observations $\mathbf{f}$. To summarize, GP inference is conceptually straightforward; we simply compute the GP posterior, which is a conditionally Gaussian distribution with a covariance matrix such that the variance is zero at the observations.

Noisy observations. In Figure 4.2.2, we assumed each observation was noiseless—that our measurements of some phenomenon were perfect—and fit it exactly. But in practice, we might want to model noisy observations,

$$y_n = f(\mathbf{x}_n) + \varepsilon_n \tag{4.18}$$

where ε_n is i.i.d. Gaussian noise or $\varepsilon_n \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$. Then Equation (4.15) becomes

$$\begin{bmatrix} \mathbf{f}_* \\ \mathbf{f} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} K(\mathbf{X}_*, \mathbf{X}_*) & K(\mathbf{X}_*, \mathbf{X}) \\ K(\mathbf{X}, \mathbf{X}_*) & K(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I} \end{bmatrix} \right) \tag{4.19}$$

while Equation (4.16) becomes

$$\mathbf{f}_* \mid \mathbf{f} \sim \mathcal{N}(\mathbb{E}[\mathbf{f}_*], \text{Cov}(\mathbf{f}_*)) \tag{4.20}$$

where

$$\begin{aligned} \mathbb{E}[\mathbf{f}_*] &= K(\mathbf{X}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}]^{-1} \mathbf{y} \\ \text{Cov}(\mathbf{f}_*) &= K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}]^{-1} K(\mathbf{X}, \mathbf{X}_*)). \end{aligned} \tag{4.21}$$

Mathematically, the diagonal noise adds "jitter" such that $k(\mathbf{x}_n, \mathbf{x}_n) \neq 0$. In other words, the variance for the training data is greater than zero.

Uncertainty. An important property of Gaussian processes is that they explicitly model *uncertainty* or the variance associated with an observation. This is because the diagonal of the covariance matrix captures the variance for each data point. This diagonal is, of course, defined by the kernel function. For example, the squared exponential is clearly unity when $\mathbf{x}_n = \mathbf{x}_m$, while the periodic kernel's diagonal depends on the parameter σ_p^2. However, recall that the variance of the conditional Gaussian decreases around the training data, meaning the uncertainty is clamped, speaking visually, around our observations.

One way to understand this is to visualize two times the standard deviation (95% confidence interval) of a GP fit to more and more data from the same generative process (Figure 4.2.3). We can see that in the absence of much data (left), the GP falls back on its prior, and the model's uncertainty is high. However, as the number of observations increases (middle, right), the model's uncertainty in its predictions decreases. If we modeled noisy

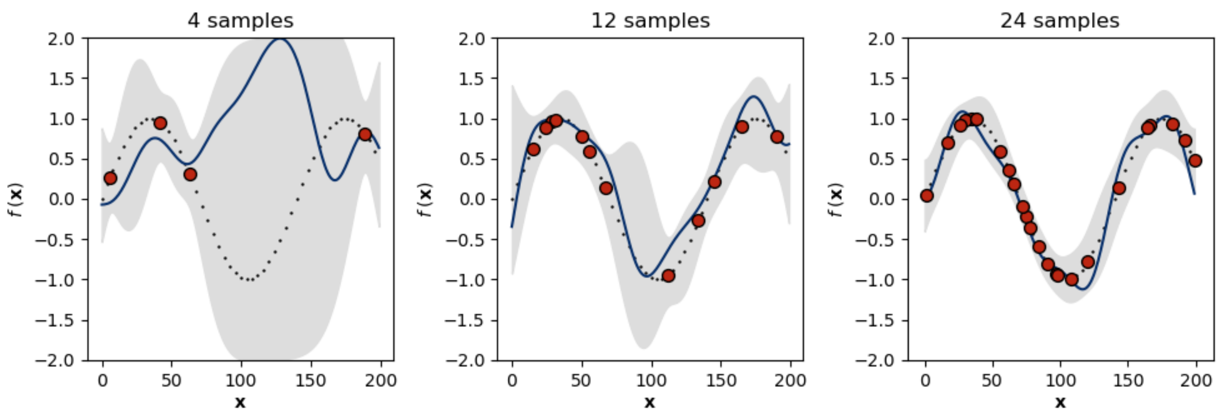

Figure 4.2.3: A visualization of twice the standard deviation (95% confidence interval) of a GP posterior conditioned on (left) 4, (middle) 12, and (right) 24 noisy observations. As the number of samples increases, the model's posterior uncertainty decreases, as quantified by a decrease in a conditional Gaussian's variance.

observations, as in Figure 4.2.3, then the uncertainty around the training data would also be greater than 0 and could be controlled by the hyperparameter σ^2.

Summary. There is a lot more to Gaussian processes. I did not, for example, discuss mean functions, hyperparameters, GP classification, or inducing points for computational efficiency [Quinonero-Candela and Rasmussen, 2005, Snelson and Ghahramani, 2006], to name a few. See Rasmussen and Williams [2006] for a thorough discussion of GPs. For now, this background should be sufficient to understand the GPLVM (discussed next) and ultimately RFLVMs.

4.2.2. *Gaussian process latent variable models*

We now discuss a latent variable model that uses Gaussian processes as a prior on the functions between latent and observation space. This model is called the *Gaussian process latent variable model* (GPLVM) [Lawrence, 2004].

PCA to GPLVM. In probabilistic principal component analysis (PCA)[3] [Tipping and Bishop, 1999], we assume our J-dimensional observations $\mathbf{Y} := [\mathbf{y}_1, \dots, \mathbf{y}_N]^\top$ are a linear

[3]I use "PCA" to refer to both standard PCA and probabilistic PCA and will specify when the distinction is not clear from context.

function of D-dimensional latent variables $\mathbf{X} \coloneqq [\mathbf{x}_1, \dots, \mathbf{x}_N]^\top$, with some additive Gaussian noise:

$$\mathbf{y}_n = \mathbf{W}\mathbf{x}_n + \boldsymbol{\varepsilon}_n, \tag{4.22}$$

where $D \ll J$ and $\boldsymbol{\varepsilon}_n \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ is random noise. The linear map $\mathbf{W}$ is a $J \times D$ matrix relating the two sets of variables. We further assume

$$\mathbf{x}_n \overset{\text{iid}}{\sim} \mathcal{N}_D(\mathbf{0}, \mathbf{I}). \tag{4.23}$$

This model is similar to factor analysis (Section 2.2) but with isotropic rather than non-isotropic Gaussian noise. In the language of factor analysis, $\mathbf{x}_n$ are latent variables, the columns of $\mathbf{X}$ are factors (latent features), and $\mathbf{W}$ are loadings.

If we assume that each observation is i.i.d., then the marginal distribution of $\mathbf{y}_n$ is also Gaussian,

$$p(\mathbf{y}_n \mid \mathbf{W}) = \mathcal{N}_J(\mathbf{0}, \underbrace{\mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I}}_{\mathbf{C}}). \tag{4.24}$$

The proof of this claim just relies on the fact that since

$$p(\mathbf{y}_n, \mathbf{x}_n \mid \mathbf{W}) = p(\mathbf{y}_n \mid \mathbf{x}_n, \mathbf{W}) p(\mathbf{x}_n \mid \mathbf{W}) \tag{4.25}$$

is jointly Gaussian, the marginal $p(\mathbf{y}_n \mid \mathbf{W})$ is also Gaussian. In effect, we have integrated out the latent variables $\mathbf{X}$. The log likelihood $\mathcal{L}_N(\mathbf{W})$ of Equation (4.24) is:

$$\begin{aligned}
\mathcal{L}_N(\mathbf{W}) &= \log p(\mathbf{Y} \mid \mathbf{W}) \\
&= \sum_{n=1}^{N} \log p(\mathbf{y}_n \mid \mathbf{W}) \\
&= \sum_{n=1}^{N} \left[\log \left(\frac{1}{\sqrt{(2\pi)^D |\mathbf{C}|}} \right) - \frac{1}{2} \mathbf{y}_n^\top \mathbf{C}^{-1} \mathbf{y}_n \right] \\
&= \sum_{n=1}^{N} \left[-\frac{D}{2} \log 2\pi - \frac{1}{2} \log |\mathbf{C}| - \frac{1}{2} \mathbf{y}_n^\top \mathbf{C}^{-1} \mathbf{y}_n \right] \\
&= -\frac{ND}{2} \log 2\pi - \frac{N}{2} \log |\mathbf{C}| - \frac{1}{2} \sum_{n=1}^{N} \mathbf{y}_n^\top \mathbf{C}^{-1} \mathbf{y}_n.
\end{aligned} \tag{4.26}$$

Often, the term $\sum_{n=1}^{N} \mathbf{y}_n^\top \mathbf{C}^{-1} \mathbf{y}_n$ is written as,

$$\begin{aligned} \sum_{n=1}^{N} \mathbf{y}_n^\top \mathbf{C}^{-1} \mathbf{y}_n &= \sum_{n=}^{N} \operatorname{tr}\left(\mathbf{C}^{-1} \mathbf{y}_n \mathbf{y}_n^\top\right) \\ &= \operatorname{tr}\left(\mathbf{C}^{-1} \sum_{n=}^{N} \mathbf{y}_n \mathbf{y}_n^\top\right) \\ &= \operatorname{tr}(\mathbf{C}^{-1} \mathbf{Y}^\top \mathbf{Y}). \end{aligned} \tag{4.27}$$

This just uses a trace trick,

$$\mathbf{a}^\top \mathbf{M} \mathbf{a} = \operatorname{tr}(\mathbf{M} \mathbf{a} \mathbf{a}^\top), \tag{4.28}$$

and the linearity of the trace operator. To fit probabilistic PCA, we optimize the nonlinear map $\mathbf{W}$ with respect to this marginal log likelihood. Tipping and Bishop [1999] proved that the maximum likelihood estimate of $\mathbf{W}$ is equivalent, up to scale and rotation, to the eigenvalue-based solution of standard PCA. (As we saw in Section 3A.3.1, we can also fit probabilistic PCA using EM.)

GPLVM. This marginalize-then-optimize process is reversed for a GPLVM. Rather than integrating out the latent variable $\mathbf{X}$ and then optimizing $\mathbf{W}$, we integrate out $\mathbf{W}$ and optimize $\mathbf{X}$. This requires that we rewrite Equation (4.22) as

$$\mathbf{y}_j = \mathbf{X} \mathbf{w}_j + \boldsymbol{\varepsilon}_j, \tag{4.29}$$

where now j indexes the columns of $\mathbf{Y}$ and $\mathbf{W}$ and where $\boldsymbol{\varepsilon}_j$ is now an N- rather than D-vector. Recall that $\mathbf{W}$ is a $J \times D$ matrix. If we assume that its rows are i.i.d., meaning

$$p(\mathbf{W}) = \prod_{j=1}^{J} \mathcal{N}_D(\mathbf{0}, \alpha^{-1} \mathbf{I}), \tag{4.30}$$

then we can copy the logic from the previous section exactly, just switching a couple variables. Now the marginal distribution of $\mathbf{y}_j$ is

$$p(\mathbf{y}_j \mid \mathbf{X}) = \mathcal{N}_N(\mathbf{0}, \underbrace{\alpha^{-1} \mathbf{X} \mathbf{X}^\top + \sigma^2 \mathbf{I}}_{\mathbf{K}_X}). \tag{4.31}$$

And the log likelihood is

$$\begin{aligned}
\mathcal{L}_N(\mathbf{X}) &= \log p(\mathbf{Y} \mid \mathbf{X}) \\
&= \sum_{j=1}^{J} \log p(\mathbf{y}_j \mid \mathbf{X}) \\
&= \sum_{j=1}^{J} \left[\log \left(\frac{1}{\sqrt{(2\pi)^N |\mathbf{K}_X|}} \right) - \frac{1}{2} \mathbf{y}_j^\top \mathbf{K}_X^{-1} \mathbf{y}_j \right] \\
&= \sum_{j=1}^{J} \left[-\frac{N}{2} \log 2\pi - \frac{1}{2} \log |\mathbf{K}_X| - \frac{1}{2} \mathbf{y}_j^\top \mathbf{K}_X^{-1} \mathbf{y}_j \right] \\
&= -\frac{JN}{2} \log 2\pi - \frac{J}{2} \log |\mathbf{K}_X| - \frac{1}{2} \sum_{j=1}^{J} \mathbf{y}_j^\top \mathbf{K}_X^{-1} \mathbf{y}_j.
\end{aligned} \tag{4.32}$$

If we rewrite the final sum in Equation (4.32) using the trace trick in Equation (4.28), we get the same formulation as in Lawrence [2004]:

$$\mathcal{L}_N(\mathbf{X}) = -\frac{JN}{2} \log 2\pi - \frac{J}{2} \log |\mathbf{K}_X| - \frac{1}{2} \mathrm{tr}(\mathbf{K}_X^{-1} \mathbf{Y} \mathbf{Y}^\top). \tag{4.33}$$

Nonlinear kernel functions. Optimizing Equation (4.33) can be viewed as the simplest form of a GPLVM. This might be surprising because there is no kernel function. Where is the GP? Notice that $\mathbf{K}_X = \alpha^{-1} \mathbf{X} \mathbf{X}^\top + \sigma^2 \mathbf{I}$ can be viewed as the covariance matrix induced by the linear kernel in Equation (4.13). A natural question is then: what if we used a nonlinear kernel? Recall that a GP is an infinite collection of random variables, defined by a mean function $m(\cdot)$ and kernel function $k(\cdot, \cdot)$, such that any finite collection of points is Gaussian distributed. This is why GPLVMs are often written in the following generative form:

$$\begin{aligned}
\mathbf{y}_j &\sim \mathcal{N}_N(f_j(\mathbf{X}), \sigma_j^2 \mathbf{I}), \\
f_j(\mathbf{X}) &\sim \mathcal{GP}(\mathbf{0}, \mathbf{K}_X), \\
\mathbf{x}_n &\sim \mathcal{N}_D(\mathbf{0}, \mathbf{I}),
\end{aligned} \tag{4.34}$$

where $f_j(\mathbf{X}) \coloneqq [f_j(\mathbf{x}_1), \dots, f_j(\mathbf{x}_N)]^\top$ and where we now have column- or feature-specific variance σ_j^2. In words, by choosing a specific nonlinear kernel function, we induce a nonlinear relationship between the latent variable $\mathbf{X}$ and our observations $\mathbf{Y}$. This nonlinear relationship can be viewed as placing a GP prior on each column of the $N \times J$ matrix $\mathbf{Y}$.

Summary. In a GPLVM, we optimize Equation (4.33) with respect to $\mathbf{X}$ rather than optimizing Equation (4.26) with respect to $\mathbf{W}$ as in probabilistic PCA. One can prove that the solution solved by optimizing Equation (4.33) with a linear kernel is equivalent the solution in PCA.

We cannot find the optimal $\mathbf{X}$ analytically, but various approximations have been proposed. We can obtain a MAP estimate by integrating out the GP-distributed maps and then optimizing $\mathbf{X}$ with respect to the posterior using scaled conjugate gradients [Lawrence, 2004, Lawrence and Hyvärinen, 2005], where computation scales as $\mathcal{O}(N^3)$. To scale inference, we may use sparse inducing point methods where the computational complexity is $\mathcal{O}(NM^2)$, for $M \ll N$ inducing points [Lawrence, 2007].

Alternatively, we can introduce a variational Bayes approximation of the posterior and minimize the Kullback–Leibler divergence between the posterior and the variational approximation with the latent variables $\mathbf{X}$ marginalized out. However, integrating out $\mathbf{X}$ in the approximate marginal likelihood is only tractable when we assume that we have Gaussian observations and when we use an RBF kernel with automatic relevance determination, which limits its flexibility. This variational approach, called a *Bayesian GPLVM* [Titsias and Lawrence, 2010, Damianou et al., 2016], may be scaled using sparse inducing point methods.

4.2.3. Random Fourier features

Finally, we review random Fourier features [Rahimi and Recht, 2008] to motivate a randomized approximation of the GP-distributed maps in GPLVMs.

Kernel methods. Consider a learning problem with data and targets $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$ where $\mathbf{x}_n \in \mathcal{X}$ and $y_n \in \mathcal{Y}$. Ignoring the bias, a linear model finds a hyperplane $\boldsymbol{\beta}$ such that the optimal decision function

$$f^*(\mathbf{x}) = \boldsymbol{\beta}^\top \mathbf{x} \tag{4.35}$$

is linear w.r.t. to the data, where the notion of optimality may be problem- or model-specific. For example, in logistic regression, we compute the logistic function of $f(\mathbf{x})$, and then threshold the output probability to produce a binary classifier with $\mathcal{Y} = \{0, 1\}$. Obviously, linear models break down when our data are not linearly separable for classification (Figure 4.2.4, left) or do not have a linear relationship between the features and targets for regression.

In a *kernel machine* or a *kernel method*—such as Gaussian processes in Section 4.2.1—, the input domain $\mathcal{X}$ is mapped into another space $\mathcal{V}$ in which the targets may be a linear function of the data. The dimension of $\mathcal{V}$ may be high or even infinite, but kernel methods avoid operating explicitly in this space using the kernel trick: if $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is a positive definite kernel function, then by Mercer's theorem there exists a *basis function* or *feature map* $\phi : \mathcal{X} \mapsto \mathcal{V}$ such that

$$k(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle_{\mathcal{V}}. \tag{4.36}$$

Here, $\langle \cdot, \cdot \rangle_{\mathcal{V}}$ is an inner product in $\mathcal{V}$. Using the kernel trick (see Section 4A.1 for a discussion) and a representer theorem [Kimeldorf and Wahba, 1971], kernel methods construct nonlinear models of $\mathcal{X}$ that are linear in $k(\cdot, \cdot)$,

$$f^*(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n k(\mathbf{x}, \mathbf{x}_n) = \langle \boldsymbol{\beta}, \phi(\mathbf{x}) \rangle_{\mathcal{V}}. \tag{4.37}$$

In Equation (4.37), $f^*(\cdot)$ denotes the optimal $f(\cdot)$. Taken together, Equation (4.36) and Equation (4.37) say that provided we have a positive definite kernel function $k(\cdot, \cdot)$, we can avoid operating in the possibly infinite-dimensional space $\mathcal{V}$ and instead only compute over N data points. This works because the optimal decision rule can be expressed as an expansion in terms of the training samples. See Schölkopf et al. [2001] for a detailed treatment on this topic.

The main point is that kernel methods allow for flexible modeling. Compare the results of the the support vector machine in Figure 4.2.4 (right) with the linear model (left), for example. Any algorithm that can be represented as a dot product between pairs of samples can be converted into a kernel method using Equation (4.36), e.g. kernel regression [Nadaraya, 1964, Watson, 1964] and kernel PCA [Schölkopf et al., 1998].

If this representer theorem is new to the reader, note that you have probably already seen other versions of representer theorems. For example, the optimal coefficients in linear

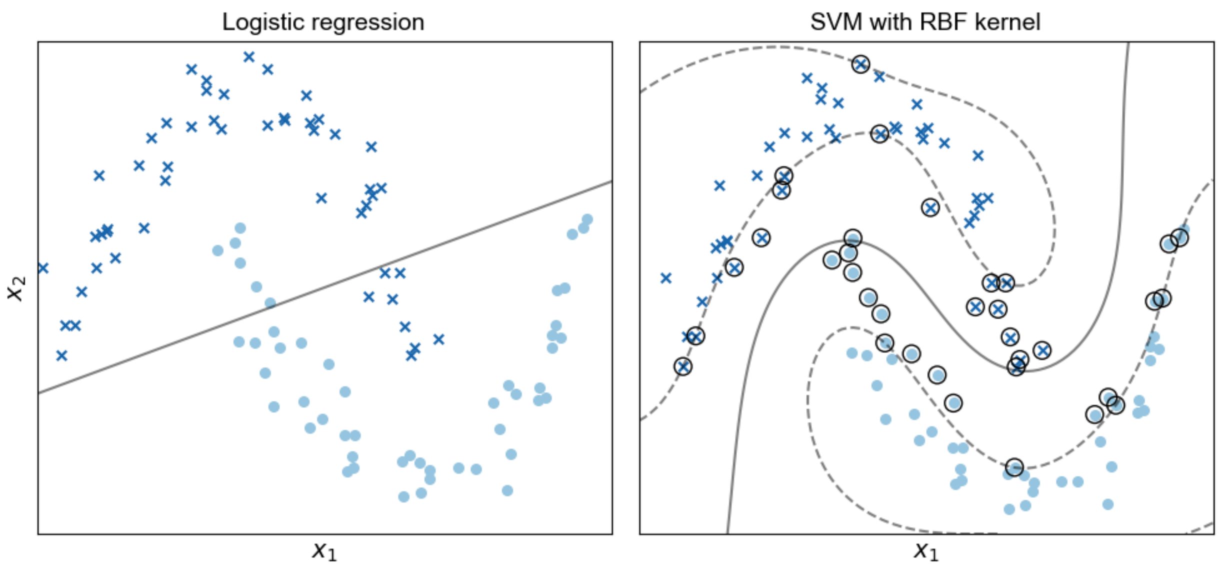

Figure 4.2.4: Logistic regression (left) with decision boundary denoted with a solid line and SVM with RBF kernel (right) on the Scikit-learn half-circles[4] data set. Support vectors are denoted with circles, and the margins are denoted with dashed lines.

regression are

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}, \tag{4.38}$$

where $\mathbf{X}$ is an $N \times D$ matrix of training data. We make predictions on held out data $\mathbf{X}_*$ as $\mathbf{y}_* = \mathbf{X}_* \hat{\boldsymbol{\beta}}$. In other words, the optimal prediction for a linear model can be viewed as an expansion in terms of the training samples. As another example, the predictive mean in GP regression (Equation (4.16)) implicitly uses the representer theorem and kernel trick.

While kernel methods are powerful, they do not scale well on large data sets (for huge N). This is because the machine must operate on a covariance matrix $\mathbf{K}_X$, induced by a particular kernel function, that is $N \times N$. To be explicit, this matrix is

$$\mathbf{K}_X := \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_N, \mathbf{x}_2) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}. \tag{4.39}$$

Random features. Rahimi and Recht [2007] proposed a way to scale kernel machines: approximate the above inner product in Equation (4.36) with a randomized map $\varphi : \mathbb{R}^D \mapsto$

$\mathbb{R}^M$ where ideally $M \ll N$:

$$k(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle_{\mathcal{V}} \approx \varphi(\mathbf{x})^\top \varphi(\mathbf{y}). \tag{4.40}$$

Why does this work, and why is it it a good idea? The representer theorem tells us that the optimal solution is a weighted sum of the kernel evaluated at our observations. If we have a good approximation of $\phi(\cdot)$, then

$$\begin{aligned} f^*(\mathbf{x}) &= \sum_{n=1}^{N} \alpha_n k(\mathbf{x}_n, \mathbf{x}) \\ &= \sum_{n=1}^{N} \alpha_n \langle \phi(\mathbf{x}_n), \phi(\mathbf{x}) \rangle_{\mathcal{V}} \\ &\approx \sum_{n=1}^{N} \alpha_n \varphi(\mathbf{x}_n)^\top \varphi(\mathbf{x}) \\ &= \boldsymbol{\beta}^\top \varphi(\mathbf{x}). \end{aligned} \tag{4.41}$$

In other words, provided $\varphi(\cdot)$ is a good approximation of $\phi(\cdot)$, then we can simply project our data using $\varphi(\cdot)$ and then use fast linear models in $\mathbb{R}^M$ rather than $\mathbb{R}^N$ (both $\boldsymbol{\beta}$ and $\varphi(\cdot)$ are M-vectors). So the task at hand is to find a random projection $\varphi(\cdot)$ such that it well-approximates the corresponding nonlinear kernel machine.

According to a blog post by Rahimi[5], this idea was inspired by the following observation. Let $\mathbf{w}$ be a random D-dimensional vector such that

$$\mathbf{w} \overset{\text{iid}}{\sim} \mathcal{N}_D(\mathbf{0}, \mathbf{I}). \tag{4.42}$$

Now define h as

$$h : \mathbf{x} \mapsto \exp(i\mathbf{w}^\top \mathbf{x}). \tag{4.43}$$

Above, i is the imaginary unit. Let the superscript $*$ denote the complex conjugate, and

[5]http://www.argmin.net/2017/12/05/kitchen-sinks/

recall that the complex conjugate of e^{ix} is e^{-ix}. Then note

$$\begin{aligned}\mathbb{E}_{\mathbf{w}}[h(\mathbf{x})h(\mathbf{y})^*] &= \mathbb{E}_{\mathbf{w}}[\exp(i\mathbf{w}^\top(\mathbf{x}-\mathbf{y}))] \\ &= \int_{\mathbb{R}^D} p(\mathbf{w}) \exp(i\mathbf{w}^\top(\mathbf{x}-\mathbf{y}))\mathrm{d}\mathbf{w} \\ &= \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{y})^\top(\mathbf{x}-\mathbf{y})\right).\end{aligned} \tag{4.44}$$

In other words, the expected value of $h(\mathbf{x})h(\mathbf{y})^*$ is the Gaussian kernel. See Section 4A.2 for a complete derivation.

This is quite interesting, and it is actually a specific instance of a more general result, Bochner's theorem [Bochner, 1959, Rudin, 1962]. Quoting Rahimi and Recht's version of Bochner's theorem with small modifications for consistent notation, the theorem is:

Theorem 4.2.2. Bochner's theorem: *A continuous kernel $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{x}-\mathbf{y})$ on $\mathbb{R}^D$ is positive definite if and only if $k(\Delta)$ is the Fourier transform of a non-negative measure.*

The Fourier transform of a non-negative measure is

$$k(\mathbf{\Delta}) = \int p(\mathbf{w}) \exp(i\mathbf{w}\mathbf{\Delta})\mathrm{d}\mathbf{w}, \tag{4.45}$$

where $p(\mathbf{w})$ is the probability function of a distribution $P_{\mathbf{w}}$. Rahimi and Recht observe that many popular kernels such as the Gaussian (or radial basis function), Laplace, and Cauchy kernels are shift-invariant. The kernel $k(\mathbf{x}-\mathbf{y})$ depends on the non-negative density function $p(\mathbf{w})$ (or vice versa).

This gives us a general framework to approximate any shift-invariant kernel by re-defining $h(\cdot)$ in Equation (4.43) to depend on $\mathbf{w}$ from any distribution $P_{\mathbf{w}}$, not just the spherical Gaussian in Equation (4.42). Furthermore, if we sample M i.i.d. realizations $\{\mathbf{w}_m\}_{m=1}^M$, we can lower the variance of this approximation:

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{y}) &= k(\mathbf{x} - \mathbf{y}) \\
&= \int p(\mathbf{w}) \exp(i\mathbf{w}^{\top}(\mathbf{x} - \mathbf{y})) \mathrm{d}\mathbf{w} \\
&= \mathbb{E}_{\mathbf{w}} \left[\exp(i\mathbf{w}^{\top}(\mathbf{x} - \mathbf{y})) \right] \\
&\overset{1}{\approx} \frac{1}{M} \sum_{m=1}^{M} \exp(i\mathbf{w}_m^{\top}(\mathbf{x} - \mathbf{y})) \\
&= \begin{bmatrix} \frac{1}{\sqrt{M}} \exp(i\mathbf{w}_1^{\top}\mathbf{x}) \\ \frac{1}{\sqrt{M}} \exp(i\mathbf{w}_2^{\top}\mathbf{x}) \\ \vdots \\ \frac{1}{\sqrt{M}} \exp(i\mathbf{w}_R^{\top}\mathbf{x}) \end{bmatrix}^{\top} \begin{bmatrix} \frac{1}{\sqrt{M}} \exp(-i\mathbf{w}_1^{\top}\mathbf{y}) \\ \frac{1}{\sqrt{M}} \exp(-i\mathbf{w}_2^{\top}\mathbf{y}) \\ \vdots \\ \frac{1}{\sqrt{M}} \exp(-i\mathbf{w}_R^{\top}\mathbf{y}) \end{bmatrix} \\
&\overset{2}{:=} \mathbf{h}(\mathbf{x})^{\top}\mathbf{h}(\mathbf{y})^{*}.
\end{aligned}
\tag{4.46}
$$

Step 1 is a Monte Carlo approximation of the expectation. Step 2 is the definition of a random map $\mathbf{h} : \mathbb{R}^D \mapsto \mathbb{R}^M$, or an M-vector of normalized $h(\cdot)$ transformations (Equation (4.43)).

Note that we mentioned a dot product $\varphi(\mathbf{x})^{\top}\varphi(\mathbf{y})$ in Equation (4.41), but above we have $\mathbf{h}(\mathbf{x})^{\top}\mathbf{h}(\mathbf{y})^{*}$. As we will see in the next section, the imaginary part of our random map will disappear, and the new transform is what Rahimi and Recht define as $\varphi(\cdot)$.[6]

Fine tuning. Now that we understand the big idea of a low-dimensional, randomized map and why it might work, let's get into the weeds. First, note that if we restrict ourselves to real-valued kernel functions, we can write

$$
\begin{aligned}
\exp(i\mathbf{w}^{\top}(\mathbf{x} - \mathbf{y})) &\overset{\dagger}{=} \cos(\mathbf{w}^{\top}(\mathbf{x} - \mathbf{y})) - \cancel{i \sin(\mathbf{w}^{\top}(\mathbf{x} - \mathbf{y}))} \\
&= \cos(\mathbf{w}^{\top}(\mathbf{x} - \mathbf{y})).
\end{aligned}
\tag{4.47}
$$

[6]Rahimi and Recht [2008] use the notation $\mathbf{z}(\cdot)$ for this map, but we use the notation $\varphi(\cdot)$ since $\mathbf{z}$ will be used elsewhere.

Step † is Euler's formula. We can then define $\zeta_{\mathbf{w}}(\mathbf{x})$—note that this is still not yet the function $\varphi(\mathbf{x})$—without the imaginary unit as

$$\begin{aligned} \mathbf{w} &\overset{\text{iid}}{\sim} P_{\mathbf{w}} \\ b &\overset{\text{iid}}{\sim} \text{Uniform}(0, 2\pi) \\ \zeta_{\mathbf{w}}(\mathbf{x}) &= \sqrt{2}\cos(\mathbf{w}^\top \mathbf{x} + b). \end{aligned} \tag{4.48}$$

This works because

$$\begin{aligned} \mathbb{E}_{\mathbf{w}}[\zeta_{\mathbf{w}}(\mathbf{x})\zeta_{\mathbf{w}}(\mathbf{y})] &= \mathbb{E}_{\mathbf{w}}[\sqrt{2}\cos(\mathbf{w}^\top \mathbf{x} + b)\sqrt{2}\cos(\mathbf{w}^\top \mathbf{y} + b)] \\ &\overset{\star}{=} \mathbb{E}_{\mathbf{w}}[\cos(\mathbf{w}^\top(\mathbf{x}+\mathbf{y}) + 2b)] + \mathbb{E}_{\mathbf{w}}[\cos(\mathbf{w}^\top(\mathbf{x}-\mathbf{y}))] \\ &\overset{\dagger}{=} \mathbb{E}_{\mathbf{w}}[\cos(\mathbf{w}^\top(\mathbf{x}-\mathbf{y}))]. \end{aligned} \tag{4.49}$$

Step $\star$ is just trigonometry. See Section 4A.3 for a derivation. Step † uses the fact that since $b \sim \text{Uniform}(0, 2\pi)$, the expectation with respect to b is zero:

$$\mathbb{E}_{\mathbf{w}}[\cos(\mathbf{w}^\top(\mathbf{x}+\mathbf{y}) + 2b)] = \mathbb{E}_{\mathbf{w}}[\mathbb{E}_b[\cos(\mathbf{w}^\top(\mathbf{x}+\mathbf{y}) + 2b) \mid \mathbf{w}]] = 0. \tag{4.50}$$

See Section 4A.4 for a complete derivation. We are now ready to define the random map $\varphi : \mathbb{R}^D \mapsto \mathbb{R}^M$ such that Equation (4.40) holds. Let

$$\varphi(\mathbf{x}) = \begin{bmatrix} \frac{1}{\sqrt{M}}\zeta_{\mathbf{w}_1}(\mathbf{x}) \\ \frac{1}{\sqrt{M}}\zeta_{\mathbf{w}_2}(\mathbf{x}) \\ \vdots \\ \frac{1}{\sqrt{M}}\zeta_{\mathbf{w}_R}(\mathbf{x}) \end{bmatrix}, \tag{4.51}$$

and therefore

$$\begin{aligned}\varphi(\mathbf{x})^\top\varphi(\mathbf{y}) &= \frac{1}{M}\sum_{m=1}^{M}\zeta_{\mathbf{w}_m}(\mathbf{x})\zeta_{\mathbf{w}_m}(\mathbf{y}) \\ &= \frac{1}{M}\sum_{m=1}^{M} 2\cos(\mathbf{w}_m^\top\mathbf{x}+b_r)\cos(\mathbf{w}_m^\top\mathbf{y}+b_r) \\ &= \frac{1}{M}\sum_{m=1}^{M}\cos(\mathbf{w}_m^\top(\mathbf{x}-\mathbf{y})) \\ &\approx \mathbb{E}_{\mathbf{w}}[\cos(\mathbf{w}^\top(\mathbf{x}-\mathbf{y}))] \\ &= k(\mathbf{x},\mathbf{y}).\end{aligned} \tag{4.52}$$

We now have a simple algorithm to estimate a shift invariant, positive definite kernel. Draw M samples of $\mathbf{w} \overset{\text{iid}}{\sim} P_{\mathbf{w}}$ and $b \overset{\text{iid}}{\sim} \text{Uniform}(0, 2\pi)$ and then compute $\varphi(\mathbf{x})^\top\varphi(\mathbf{y})$.

Example: kernel ridge regression. Let's see concretely why random Fourier features are efficient by looking at kernel (ridge) regression [Nadaraya, 1964, Watson, 1964]. (See Welling [2013] for an introduction to the model.) Equation (4.41) tells us that $f^*(\mathbf{x})$ is linear in $\varphi(\mathbf{x})$. Therefore, we just need to convert our input $\mathbf{x}$ into random features and apply linear methods. Concretely, we just want to solve for the coefficients $\boldsymbol{\beta}$ in

$$\hat{\boldsymbol{\beta}} = (\underbrace{\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda\mathbf{I}_M}_{\mathbf{A}})^{-1}\boldsymbol{\Phi}^\top\mathbf{y}, \tag{4.53}$$

where $\boldsymbol{\Phi} := [\varphi(\mathbf{x}_1), \dots, \varphi(\mathbf{x}_N)]^\top \in \mathbb{R}^{N\times M}$. Above, λ is the ridge regression regularization parameter. See Figure 4.2.5 for the results of comparing Gaussian kernel regression with random Fourier feature regression.

With Equation (4.53) in mind, it is clear why random Fourier features are efficient: inverting $\mathbf{A}$ has time complexity $\mathcal{O}(M^3)$ rather than $\mathcal{O}(N^3)$. If $M \ll N$, then we can have big savings. What is not shown is that even on this small data set, random Fourier feature regression is over an order of magnitude faster than kernel regression with a Gaussian kernel. Since kernel machines scale poorly in N, it is easy to make this multiplier larger by increasing N while keeping M fixed. See Section 4A.6 for a Python implementation.

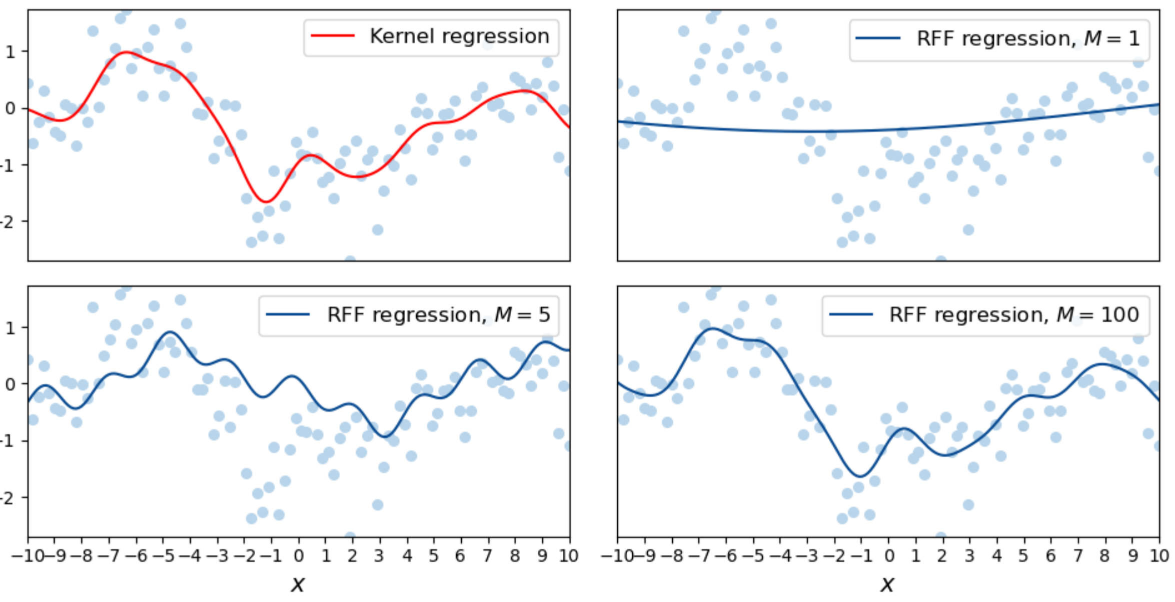

Figure 4.2.5: Comparison of Gaussian kernel ridge regression (top left) with ridge regression using random Fourier features (RFF regression) on $N = 100$ data points with $M \in \{1, 5, 100\}$.

Alternative definition. For the remainder of this chapter, we will use this alternative definition of Equation (4.51):

$$\mathbf{w}_m \overset{\text{iid}}{\sim} P_{\mathbf{w}}, \qquad \varphi(\mathbf{x}) := \sqrt{\frac{2}{M}} \begin{bmatrix} \sin(\mathbf{w}_1^\top \mathbf{x}) \cos(\mathbf{w}_1^\top \mathbf{x}) \\ \vdots \\ \sin(\mathbf{w}_{M/2}^\top \mathbf{x}) \cos(\mathbf{w}_{M/2}^\top \mathbf{x}) \end{bmatrix}. \tag{4.54}$$

In other words, we draw $M/2$ samples from $P_{\mathbf{w}}$, and the definition in Equation (4.54) doubles the number of random features to M. Finally, we use the notation $\mathbf{\Phi}$ to denote $\varphi(\cdot)$ applied to all observations, i.e. $\mathbf{\Phi} := [\varphi(\mathbf{x}_1), \dots, \varphi(\mathbf{x}_N)]^\top$. See Section 4A.5 for a detailed derivation of this formulation.

4.3 Random feature latent variable models

We now have the necessary background in GPs, the GPLVM, and random Fourier features to develop RFLVMs. Recall that the main idea is that we will approximate the GP-distributed functions in the GPLVM with random features, inducing closed-form gradients of the posterior with respect to the latent variable. RFFs have been used to reduce the computational costs of fitting GP regression models from $\mathcal{O}(N^3)$ to $\mathcal{O}(NM^2)$ [Lázaro-Gredilla et al., 2010, Hensman et al., 2017]. However, RFFs have not yet been used to make GPLVMs more computationally tractable.

4.3.1. Generative model for RFLVMs

The generative model of an RFLVM takes the form:

$$
\begin{aligned}
\mathbf{y}_j &\sim \mathcal{L}\left(g\left(\boldsymbol{\Phi}\boldsymbol{\beta}_j\right), \boldsymbol{\theta}\right), & \boldsymbol{\theta} &\sim P_{\boldsymbol{\theta}}, \\
\boldsymbol{\beta}_j &\sim \mathcal{N}_M(\boldsymbol{\beta}_0, \mathbf{B}_0), & \mathbf{x}_n &\sim \mathcal{N}_D(\mathbf{0}, \mathbf{I}), \\
\mathbf{w}_m &\sim \mathcal{N}_D(\boldsymbol{\mu}_{z_m}, \boldsymbol{\Sigma}_{z_m}), & z_m &\sim \mathrm{CRP}(\alpha), \\
\alpha &\sim \mathrm{Gamma}(a_\alpha, b_\alpha), & (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) &\sim \mathrm{NIW}(\boldsymbol{\mu}_0, \nu_0, \lambda_0, \boldsymbol{\Psi}_0).
\end{aligned}
\tag{4.55}
$$

Here, $\mathcal{L}(\cdot)$ is a likelihood function, $g(\cdot)$ is an invertible link function that maps the real numbers onto the likelihood parameters' support, and $\boldsymbol{\theta}$ are other likelihood-specific parameters. Following Wilson and Adams [2013] and Oliva et al. [2016], we assume the distribution on $\mathbf{w}$, denoted $P_{\mathbf{w}}$, is a Dirichlet process mixture of Gaussians (DP-GMM) [Ferguson, 1973, Antoniak, 1974]. By sampling from the posterior of $\mathbf{w}$, we can explore the space of stationary kernels and estimate the kernel hyperparameters in a Bayesian way. We assign each $\mathbf{w}_m$ in $\mathbf{W} \coloneqq [\mathbf{w}_1, \dots, \mathbf{w}_{M/2}]^\top$ to a mixture component with the variable z_m, which is distributed according to a Chinese restaurant process (CRP) [Aldous, 1985] with concentration parameter α. This prior introduces additional random variables: the mixture means $\{\boldsymbol{\mu}_k\}_{k=1}^K$, and the mixture covariance matrices $\{\boldsymbol{\Sigma}_k\}_{k=1}^K$ where K is the number of clusters in the current Gibbs sampling iteration. (The Gibbs sampler is discussed in Section 4.3.2.)

The randomized map in Equation (4.54) allows us to approximate the original GPLVM in Equation (4.34) as

$$
\begin{aligned}
\mathbf{y}_j &\sim \mathcal{N}_N(\boldsymbol{\Phi}\boldsymbol{\beta}_j, \sigma_j^2\mathbf{I}), \\
\boldsymbol{\beta}_j &\sim \mathcal{N}_M(\mathbf{b}_0, \mathbf{B}_0), \\
\mathbf{x}_n &\sim \mathcal{N}_D(\mathbf{0}, \mathbf{I}).
\end{aligned}
\tag{4.56}
$$

We approximate $f_j(\mathbf{X})$ in Equation (4.34) as $\boldsymbol{\Phi}\boldsymbol{\beta}_j$. This is a Gaussian RFLVM when $\mathcal{L}(\cdot)$ is a Gaussian distribution and $g(\cdot)$ is the identity function. Because the prior distribution on the mapping weights $\boldsymbol{\beta}_j$ is Gaussian, the model is analogous to Bayesian linear regression given $\boldsymbol{\Phi}$; if we integrate out $\boldsymbol{\beta}_j$, we recover a marginal likelihood that approximates the GPLVM's marginal likelihood. (See Section 4A.8 for a detailed derivation.)

We use this representation to generalize the RFLVM to other observation types in the exponential family. For example, a Poisson RFLVM takes the following form:

$$\begin{aligned}
\mathbf{y}_j &\sim \text{Poisson}(\exp(\boldsymbol{\Phi}\boldsymbol{\beta}_j)), \\
\boldsymbol{\beta}_j &\sim \mathcal{N}_M(\mathbf{b}_0, \mathbf{B}_0), \\
\mathbf{x}_n &\sim \mathcal{N}_D(\mathbf{0}, \mathbf{I}).
\end{aligned} \tag{4.57}$$

For distributions including the Bernoulli, binomial, and negative binomial, the functional form of the data likelihood is

$$\begin{aligned}
&\mathcal{L}(\boldsymbol{\Phi}, \boldsymbol{\beta}_j, a(\mathbf{y}_j), b(\mathbf{y}_j), c(\mathbf{y}_j)) \\
&= \prod_{n=1}^{N} c(y_{nj}) \frac{(\exp(\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j))^{a(y_{nj})}}{(1+\exp(\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j))^{b(y_{nj})}},
\end{aligned} \tag{4.58}$$

for some functions of the data $a(\cdot)$, $b(\cdot)$, and $c(\cdot)$. The general form of this logistic RFLVM is then:

$$\begin{aligned}
\mathbf{y}_j &\sim \mathcal{L}(\boldsymbol{\Phi}, \boldsymbol{\beta}_j, a(\mathbf{y}_j), b(\mathbf{y}_j), c(\mathbf{y}_j)), \\
\boldsymbol{\beta}_j &\sim \mathcal{N}_M(\mathbf{b}_0, \mathbf{B}_0), \\
\mathbf{x}_n &\sim \mathcal{N}_D(\mathbf{0}, \mathbf{I}).
\end{aligned} \tag{4.59}$$

For example, by setting $a(y_{nj}) = y_{nj}$, $b(y_{nj}) = y_{nj} + r_j$, and $c(y_{nj}) = \binom{y_{nj}+r_j-1}{y_{nj}}$, we get the negative binomial RFLVM with feature-specific dispersion parameter r_j.

4.3.2. Inference for RFLVMs

We now present a general Gibbs sampling framework for all RFLVMs. A consequence of the linearization induced by the random features is that we can use all available techniques for Gibbs sampling in linear models that are otherwise not possible for GPLVMs; and unlike other sampling methods such as HMC, Gibbs samplers do not require tuning parameters.

First, we write the Gibbs sampling steps to estimate the posterior of the covariance kernel. Next, we describe estimating the latent variable $\mathbf{X}$ by taking the MAP estimate. Then, we sample the data likelihood-specific parameters $\boldsymbol{\theta}$ and linear coefficients $\boldsymbol{\beta}_j$.

Variables subscripted with zero, e.g., $\boldsymbol{\theta}_0$, denote hyperparameters. While the number of mixture components may change across sampling iterations, let K denote the number of components in the current Gibbs sampling step. We initialize all the parameters in our model by drawing from the prior, except for $\mathbf{X}$, which we initialize with PCA.

First, we sample z_m following Algorithm 8 from Neal [2000]. We choose to use a sampling method that integrates out the Dirichlet process-distributed mixture weights because such samplers can better propose new features [Dubey et al., 2020] and are therefore more effective at exploring the posterior behavior of the covariance kernel. Let $n_k = \sum_\ell \delta(z_\ell = k)$, and let n_k^{-m} denote the same sum with z_m excluded. Then we sample the posterior of z_m from the following discrete distribution for $k = 1, 2, \dots, K$:

$$p(z_m = k \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{W}, \alpha) = \begin{cases} \frac{n_k^{-m}}{M-1+\alpha} \mathcal{N}(\mathbf{w}_m \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) & n_k^{-m} > 0 \\ \frac{\alpha}{M-1+\alpha} \int \mathcal{N}(\mathbf{w}_m \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \mathrm{NIW}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \mathrm{d}\boldsymbol{\mu} \mathrm{d}\boldsymbol{\Sigma} & n_k^{-m} = 0. \end{cases} \tag{4.60}$$

Given assignments $\mathbf{z} := [z_1, \dots, z_{M/2}]^\top$ and RFFs $\mathbf{W}$, the posterior of $\boldsymbol{\Sigma}_k$ is inverse-Wishart distributed. Given $\boldsymbol{\Sigma}_k$, the posterior of $\boldsymbol{\mu}_k$ is normally distributed [Gelman et al., 2013]:

$$\begin{aligned} &\boldsymbol{\Sigma}_k \sim \mathcal{W}^{-1}(\boldsymbol{\Psi}_k, \nu_k), \quad \boldsymbol{\mu}_k \sim \mathcal{N}(\mathbf{m}_k, \frac{1}{\lambda_k} \boldsymbol{\Sigma}_k). \\ &\boldsymbol{\Psi}_k := \boldsymbol{\Psi}_0 + \sum_{m:z_m=k}^{M/2} (\mathbf{w}_m - \bar{\mathbf{w}}^{(k)})(\mathbf{w}_m - \bar{\mathbf{w}}^{(k)})^\top + \frac{\lambda_0 n_k}{\lambda_0 + n_k} (\mathbf{w}_m - \boldsymbol{\mu}_0)(\mathbf{w}_m - \boldsymbol{\mu}_0)^\top \\ &\bar{\mathbf{w}}^{(k)} := \frac{1}{n_k} \sum_{m:z_m=k}^{M} \mathbf{w}_m, \quad \nu_k := \nu_0 + n_k, \\ &\mathbf{m}_k := \frac{\lambda_0 \boldsymbol{\mu}_0 + n_k \bar{\mathbf{w}}^k}{\lambda_0 + n_k}, \quad \lambda_k := \lambda_0 + n_k. \end{aligned} \tag{4.61}$$

We cannot sample from the full conditional distribution of $\mathbf{W}$, but prior work suggested a Metropolis–Hastings (MH) sampler in which the proposal distribution's density function

$q(\mathbf{W})$ is set to the prior density function $p(\mathbf{W} \mid \mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}_D(\boldsymbol{\mu}_{z_m}, \boldsymbol{\Sigma}_{z_m})$ (Equation (4.55)) and acceptance ratio $\rho_{\texttt{MH}}$ [Oliva et al., 2016]:

$$\mathbf{w}_m^\star \sim q(\mathbf{W}) \coloneqq p(\mathbf{W} \mid \mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}), \qquad \rho_{\texttt{MH}} \coloneqq \min\left\{1, \frac{p(\mathbf{Y} \mid \mathbf{X}, \mathbf{w}_m^\star, \boldsymbol{\theta})}{p(\mathbf{Y} \mid \mathbf{X}, \mathbf{w}_m, \boldsymbol{\theta})}\right\}. \tag{4.62}$$

Finally, we sample the DP-GMM concentration parameter α [Escobar and West, 1995]. We augment the model with variable η to make sampling α conditionally conjugate:

$$\begin{aligned}
\eta &\sim \text{Beta}(\alpha + 1, M), \\
\frac{\pi_\eta}{1 - \pi_\eta} &= \frac{a_\alpha + K - 1}{M(b_\alpha - \log(\eta))}, \quad K = |\{k : n_k > 0\}|, \\
\alpha &\sim \pi_\eta \text{Gamma}(a_\alpha + K, b_\alpha - \log(h)) \\
&\quad + (1 - \pi_\eta)\text{Gamma}(a_\alpha + K - 1, b_\alpha - \log(\eta)).
\end{aligned} \tag{4.63}$$

For the Gaussian RFLVM (Equation (4.56)), let $\mathbf{B}_0 = \sigma^{-2}\mathbf{S}_0$. We integrate out $\boldsymbol{\beta}_j$ and σ^{-2} in closed form to obtain a marginal likelihood,

$$p(\mathbf{y}_j \mid \mathbf{X}, \mathbf{W}) = \frac{1}{(2\pi)^{N/2}} \cdot \sqrt{\frac{|\mathbf{S}_0|}{|\mathbf{S}_N|}} \cdot \frac{b_0^{a_0}}{b_N^{a_N}} \cdot \frac{\Gamma(a_N)}{\Gamma(a_0)}, \tag{4.64}$$

where

$$\begin{aligned}
\mathbf{S}_N &\coloneqq \boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \mathbf{S}_0, \\
\boldsymbol{\beta}_N &\coloneqq \mathbf{S}_N^{-1}(\boldsymbol{\beta}_0^\top \mathbf{S}_0 + \boldsymbol{\Phi}^\top \mathbf{y}_j), \\
a_N &\coloneqq a_0 + N/2, \\
b_N &\coloneqq b_0 + (1/2)(\mathbf{y}_j^\top \mathbf{y}_j + \boldsymbol{\beta}_0^\top \mathbf{S}_0 \boldsymbol{\beta}_0 - \boldsymbol{\beta}_N^\top \mathbf{S}_N \boldsymbol{\beta}_N).
\end{aligned} \tag{4.65}$$

(See Section 4A.8 or Minka [2000] for details.) However, inference can be slow because marginalizing out $\boldsymbol{\beta}_j$ introduces dependencies between the latent variables, and the complexity becomes $\mathcal{O}(NM^2)$. Alternatively, we can Gibbs sample $\boldsymbol{\beta}_j$ and take the MAP estimate of $\mathbf{X}$ using the original log likelihood where the complexity is $\mathcal{O}(NM)$.

In the Poisson RFLVM (Equation (4.57)), we no longer have the option of marginalizing out $\boldsymbol{\beta}_j$. Instead, we take iterative MAP estimates of $\boldsymbol{\beta}_j$ and $\mathbf{X}$. Given $\boldsymbol{\Phi}$, inference for $\boldsymbol{\beta}_j$ is analogous to Bayesian inference for a Poisson generalized linear model (GLM). In

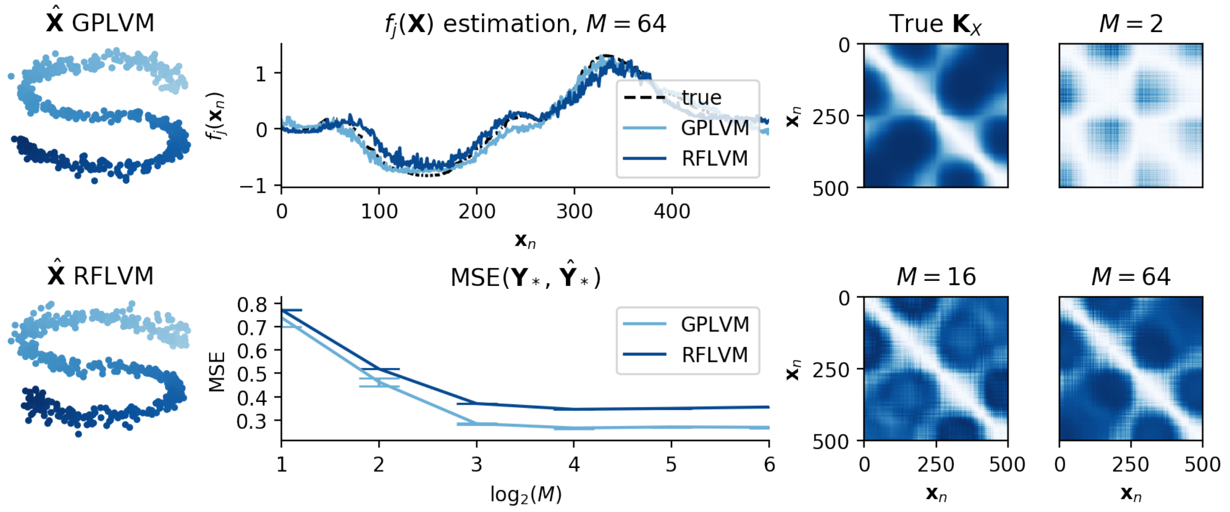

Figure 4.3.1: Simulated data with Gaussian emissions. (Left) Inferred latent variables for both a GPLVM and Gaussian RFLVM. (Upper middle) Comparison of estimated $f_j(\mathbf{X})$ for a single feature as estimated by GPLVM and RFLVM. (Lower middle) Comparison of MSE reconstruction error on held out $\mathbf{Y}_*$ for increasing M, where M is the number of inducing points for GPLVM and random Fourier features for RFLVM. (Right) Ground truth covariance matrix $\mathbf{K}_X$ compared with the RFLVM estimation for increasing M.

Section 4.4.1 and Section 4.4.2, we show that, by inducing closed-form gradients with respect to $\mathbf{X}$ through RFFs, this iterative MAP procedure produces results that are competitive with benchmarks on count data. For logistic RFLVMs (Equation (4.59)), we use Pólya–gamma augmentation [Polson et al., 2013] to make inference tractable. A random variable ω is Pólya–gamma distributed with parameters $b > 0$ and $c \in \mathbb{R}$, denoted $\omega \sim \text{PG}(b, c)$, if

$$\omega \overset{d}{=} \frac{1}{2\pi^2} \sum_{k=1}^{\infty} \frac{g_k}{(k - 1/2)^2 + c^2/(4\pi^2)}, \tag{4.66}$$

where $\overset{d}{=}$ denotes equality in distribution and $g_k \sim \text{Gamma}(b, 1)$ are independent gamma random variables. The identity critical for Pólya–gamma augmentation is

$$\frac{(e^{\psi_{nj}})^{a_{nj}}}{(1 + e^{\psi_{nj}})^{b_{nj}}} = 2^{-b_{nj}} e^{\kappa_{nj}\psi_{nj}} \int_0^{\infty} e^{-\omega\psi_{nj}^2/2} p(\omega) \mathrm{d}\omega, \tag{4.67}$$

where $\kappa_{nj} = a_{nj} - b_{nj}/2$ and $p(\omega) = \text{PG}(\omega \mid b_{nj}, 0)$. If we define $\psi_{nj} := \varphi(\mathbf{x}_n)^\top \boldsymbol{\beta}_j$, then Equation (4.67) allows us to rewrite the likelihood in Equation (4.58) as proportional to a Gaussian. Furthermore, we can sample ω conditioned on ψ_{nj} as $p(\omega \mid \psi_{nj}) \sim \text{PG}(b_{nj}, \psi_{nj})$.

This enables convenient, closed-form Gibbs sampling steps of $\boldsymbol{\beta}_j$, conditioned on Pólya–gamma augmentation variables ω_{nj}:

$$
\begin{aligned}
\omega_{nj} \mid \boldsymbol{\beta}_j &\sim \mathrm{PG}(b_{nj}, \varphi(\mathbf{x}_n)^\top \boldsymbol{\beta}_j), \\
\boldsymbol{\beta}_j \mid \boldsymbol{\Omega}_j &\sim \mathcal{N}(\mathbf{m}_{\boldsymbol{\omega}_j}, \mathbf{V}_{\boldsymbol{\omega}_j}), \\
\mathbf{V}_{\boldsymbol{\omega}_j} &:= (\boldsymbol{\Phi}^\top \boldsymbol{\Omega}_j \boldsymbol{\Phi} + \mathbf{B}_0^{-1})^{-1}, \\
\mathbf{m}_{\boldsymbol{\omega}_j} &:= \mathbf{V}_{\boldsymbol{\omega}_j}(\boldsymbol{\Phi}^\top \boldsymbol{\kappa}_j + \mathbf{B}_0^{-1}\boldsymbol{\beta}_0),
\end{aligned}
\tag{4.68}
$$

where $\boldsymbol{\Omega}_j = \mathrm{diag}([\omega_{1j} \dots \omega_{Nj}])$ and $\boldsymbol{\kappa}_j = [\kappa_{1j} \dots \kappa_{Nj}]^\top$. (See Section 4A.7 for a more thorough discussion of this technique.) This technique has been used to derive Gibbs samplers for binomial regression [Polson et al., 2013], negative binomial regression [Zhou et al., 2012], and correlated topic models [Chen et al., 2013, Linderman et al., 2015]. Here, we use it to derive samplers for logistic RFLVMs. (See Section 4A.9 for detailed derivations of the negative binomial RFLVM.)

RFLVMs are identifiable up to the rotation and scale of $\mathbf{X}$. As a result, MAP estimates of $\mathbf{X}$ between iterations are unaligned as they can be arbitrarily rescaled and rotated through inference. Thus, a point estimate of $\mathbf{X}$, denote this as $\hat{\mathbf{X}}$, that is a function of the Monte Carlo samples of $\mathbf{X}$, e.g., the expectation of $\hat{\mathbf{X}}$ across the MCMC samples, will not be meaningful. To this end, we arbitrarily fix the rotation of $\hat{\mathbf{X}}$ by taking the singular value decomposition (SVD) of the MAP estimate, $\hat{\mathbf{X}} := \mathbf{USV}^T$, and setting $\hat{\mathbf{X}}$ to be the left singular vectors corresponding to the D largest singular values, $\hat{\mathbf{X}} := [\mathbf{u}_1, \dots, \mathbf{u}_D]$ where $\mathrm{diag}(\mathbf{S}) = [s_1 \, \dots \, s_D]$ and $s_1 \geq s_2 \geq \dots \geq s_D$. Then, we rescale $\hat{\mathbf{X}}$ so that the covariance of the latent space is the identity matrix. This has the effect of enforcing orthogonality, and does not allow heteroskedasticity in the latent dimensions. This operation is analogous to the covariance adjustment in parameter-expanded expectation–maximization [Liu et al., 1998] and has been used to fix the rotation in Bayesian factor analysis models [Ročková and George, 2016].

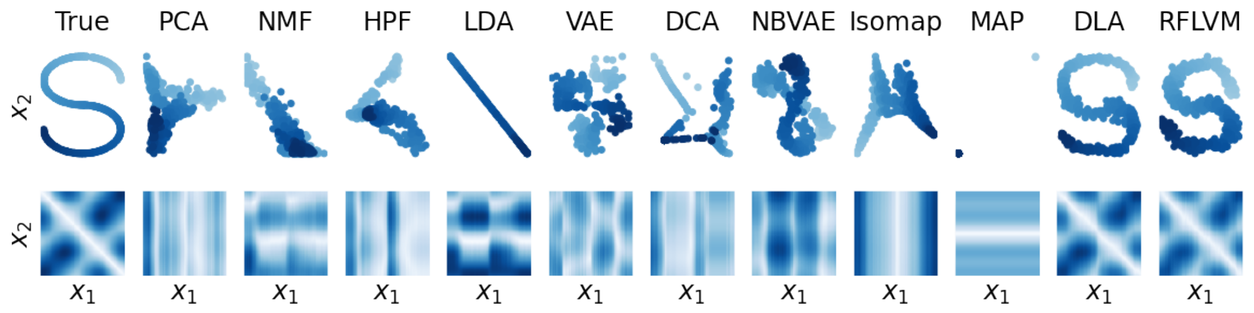

Figure 4.4.1: Simulated data with Poisson emissions. (Top) True latent variable $\mathbf{X}$ compared with inferred latent variables $\hat{\mathbf{X}}$ from benchmarks (see text for abbreviations) and a Poisson RFLVM. (Bottom) Distance matrices between true $\mathbf{X}$ and $\hat{\mathbf{X}}$ from the above benchmark (darker is farther away).

4.4 Experiments

In our results, we refer to the Gaussian-distributed GPLVM using inducing point methods for inference as *GPLVM* [Titsias and Lawrence, 2010][7]. We fit all GPLVM experiments using the `GPy` package [GPy, 2012]. We refer to the Poisson-distributed GPLVM using a double Laplace approximation as *DLA-GPLVM* [Wu et al., 2017]. DLA-GPLVM is designed to model multi-neuron spike train data, and the code[8] initializes the latent space using the output of a Poisson linear dynamical system [Macke et al., 2011], and places a GP prior on $\mathbf{X}$. To make the experiments comparable for all GPLVM experiments, we initialize DLA-GPLVM with PCA and assume $\mathbf{x} \overset{\text{iid}}{\sim} \mathcal{N}_D(\mathbf{0}, \mathbf{I})$. We refer to our GPLVM with random Fourier features as *RFLVM* and explicitly state the assumed distribution. In Section 4.4.1, we use a Gaussian RFLVM with the linear coefficients $\{\boldsymbol{\beta}_j\}_{j=1}^J$ marginalized out (Equation (4.64)) for a fairer comparison with the GPLVM.

Since hyperparameter tuning our model on each data set would be both time-consuming and unfair without also tuning the baselines, we fixed the hyperparameters across experiments. We used 2000 Gibbs sampling iterations with 1000 burn-in steps, $M = 100$, and $D = 2$. We initialized $K = 20$ and $\alpha = 1$. In Section 4.4.2, we used $D = 3$ and visualized $\hat{\mathbf{X}}$ after the best affine transformation onto 2-D rat positions following Wu et al. [2017].

[7]We used GPy's implementation `BayesianGPLVMMiniBatch`, which supports inducing points and prediction on held out data.
[8]https://github.com/waq1129/LMT

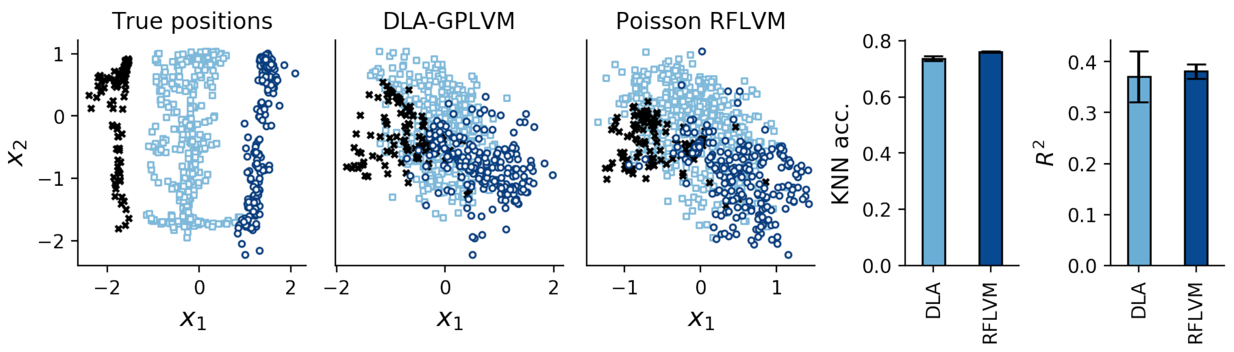

Figure 4.4.2: Hippocampal place cells. (Left three plots) Inferred latent space for the DLA-GPLVM and the Poisson RFLVM. The points are colored by three major regions of the true rat position in a W-shaped maze. (Right two plots) KNN accuracy using 5-fold cross validation and R^2 performance of the best affine transformation from $\hat{\mathbf{X}}$ onto the rat positions $\mathbf{X}$. Error bars computed using five trials.

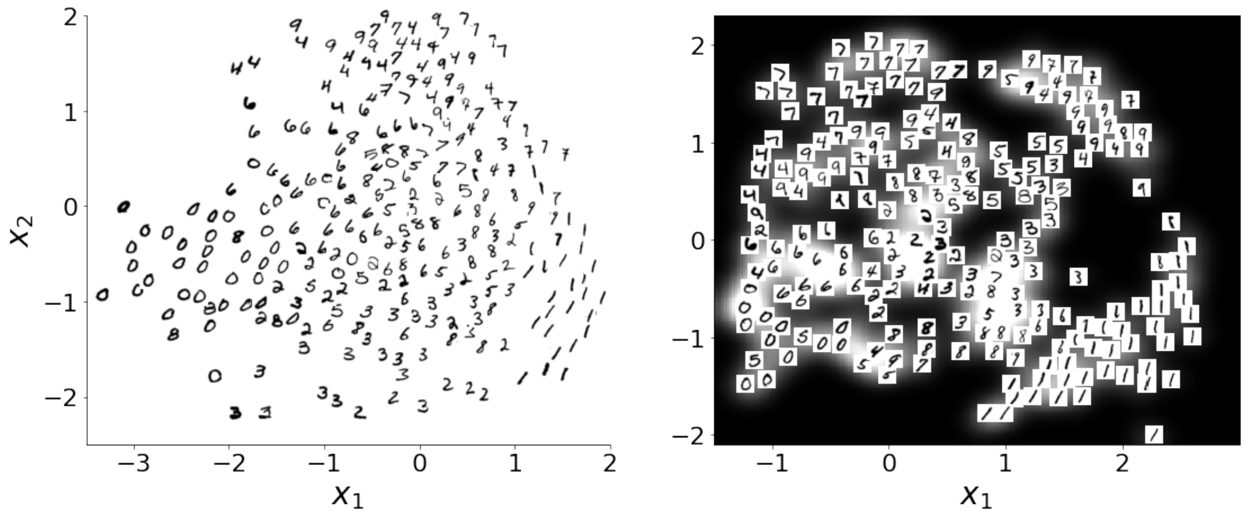

Figure 4.4.3: MNIST digits visualized in 2-D latent space inferred from DLA-GPLVM (left) and Poisson RFLVM (right). Following Lawrence [2004], we plotted images in a random order while not plotting any images that result in an overlap. The RFLVM's latent space is visualized as a histogram of 1000 draws after burn-in. The plotted points are the sample posterior mean.

For computational reasons, MNIST and CIFAR-10 were subsampled (see Section 4A.11 for details).

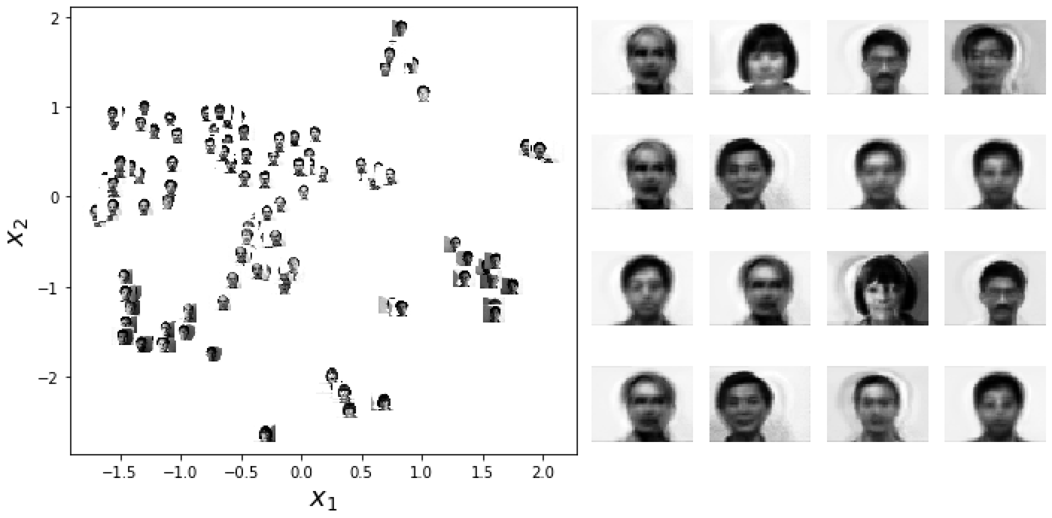

Figure 4.4.4: Yale faces data visualized in 2-D latent space using a Poisson RFLVM (left). Synthetic faces for the Yale data set sampled from the posterior data generating process using a Poisson RFLVM (right).

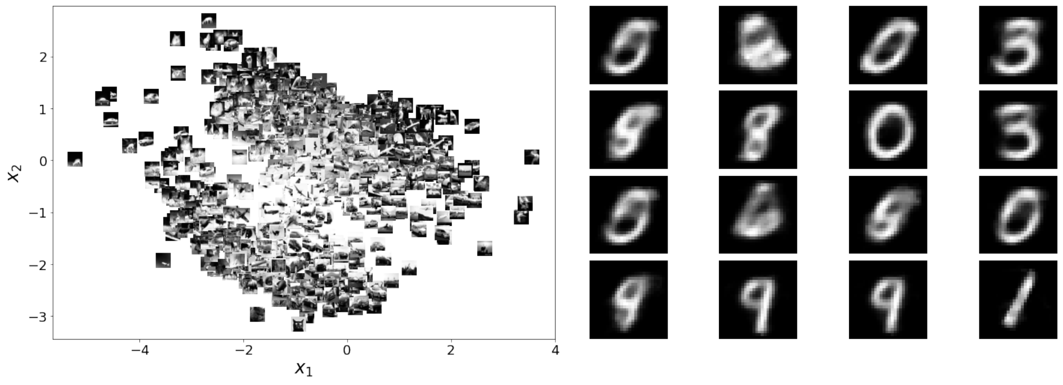

Figure 4.4.5: CIFAR-10 image data set visualized in 2-D latent space using a Poisson RFLVM (left). Synthetic digits for MNIST sampled from the posterior data generating process using a Poisson RFLVM (right).

4.4.1. Simulated data

We first evaluate RFLVM on simulated data. We set $\mathbf{X}$ to be a 2-D S-shaped manifold, sampled functions $\mathbf{F} \coloneqq \{f_j(\mathbf{X})\}_{j=1}^{J}$ from a Gaussian process with an RBF kernel, and then generated observations for Gaussian emissions (Equation (4.34)) and Poisson emissions (Equation (4.57)). For all simulations, we used $N = 500, J = 100$, and $D = 2$.

For these experiments, we computed the mean-squared error (MSE) between test set observations, $\mathbf{Y}_*$, and predicted observations $\hat{\mathbf{Y}}_*$, where we held out 20% of the observations for the test set. To evaluate our latent space results, we projected the estimated latent space, $\hat{\mathbf{X}}$, onto the hyperplane that minimizes the squared error with the ground truth, $\|\mathbf{X}-\hat{\mathbf{X}}\mathbf{A}\|_2^2$, and calculated the R^2 value between the true $\mathbf{X}$ and the projected latent space $\hat{\mathbf{X}}\mathbf{A}$. We evaluated our model's ability to estimate the GP outputs $f_j(\mathbf{X}) \approx \mathbf{\Phi}\boldsymbol{\beta}_j$ by comparing the MSE between the estimated $\varphi(\hat{\mathbf{X}})\boldsymbol{\beta}_j$ and the true generating $f_j(\mathbf{X})$. We computed the mean and standard deviation of the MSE and R^2 results by running each experiment five times.

We compared the performance of a Gaussian RFLVM to the GPLVM. We ran these experiments across multiple values of M, where M denotes the number of random features for the RFLVM and the number of inducing points for the GPLVM. Both models recovered the true latent variable $\mathbf{X}$ accurately and estimated the nonlinear maps, $\mathbf{F}$, well (Figure 4.3.1, upper middle). Empirically, a GPLVM shows better performance for estimating $\mathbf{Y}_*$ than the RFLVM (Figure 4.3.1, lower middle). We hypothesize that this is because Nyström's method has better generalization error bounds than RFFs when there is a large gap in the eigenspectrum [Yang et al., 2012], which is the case for $\mathbf{K}_X$. However, we see that the RFLVM approximates the true $\mathbf{K}_X$ given enough random features (Figure 4.3.1, right), though perhaps less accurately than the GPLVM (Figure 4.3.1, lower middle).

To demonstrate the utility of our model beyond Gaussian-distributed data, we compared results for simulated count data from a Poisson RFLVM with the following benchmarks: PCA, nonnegative matrix factorization (NMF) [Lee and Seung, 1999], hierarchical Poisson factorization (HPF) [Gopalan et al., 2015], latent Dirichlet allocation (LDA) [Blei et al., 2003], variational autoencoder (VAE) [Kingma and Welling, 2013], deep count autoencoder (DCA) [Eraslan et al., 2019], negative binomial VAE (NB-VAE) [Zhao et al., 2020], and Isomap [Balasubramanian et al., 2002]. Additionally, we compared results to our own naive implementation of the Poisson GPLVM that performs coordinate ascent on $\mathbf{X}$ and $\mathbf{F}$ by iteratively taking MAP estimates without using RFFs. We refer to this method as *MAP-GPLVM*. We found that the Poisson RFLVM infers a latent variable that is more similar to the true latent structure than other methods (Figure 4.4.1). Linear methods such as PCA and NMF lack the flexibility to capture this nonlinear space, while nonlinear but Gaussian methods such as Isomap and VAEs recover smooth latent spaces that lack the original structure. The MAP-GPLVM appears to get stuck in poor local modes because we do not

Table 4.4.1: Classification accuracy evaluated by fitting a KNN classifier ($K = 1$) with five-fold cross validation. Mean accuracy and standard deviation were computed by running each experiment five times.

	PCA	NMF	HPF	LDA	VAE	DCA
Bridges	0.8469 ± 0.0067	$\mathbf{0.8664 \pm 0.0164}$	0.7860 ± 0.0328	0.6747 ± 0.0412	0.8141 ± 0.0301	0.7093 ± 0.0317
CIFAR-10	0.2651 ± 0.0019	0.2450 ± 0.0028	0.2516 ± 0.0074	0.2248 ± 0.0040	0.2711 ± 0.0083	0.2538 ± 0.0178
Congress	0.5558 ± 0.0098	0.5263 ± 0.0108	0.6941 ± 0.0537	0.7354 ± 0.1018	0.6563 ± 0.0314	0.5917 ± 0.0674
MNIST	0.3794 ± 0.0146	0.2764 ± 0.0197	0.3382 ± 0.0370	0.2176 ± 0.0387	$\mathbf{0.6512 \pm 0.0228}$	0.1620 ± 0.0976
Montreal	0.6802 ± 0.0099	0.6878 ± 0.0207	0.6144 ± 0.1662	0.6238 ± 0.0271	0.6702 ± 0.0325	0.6601 ± 0.0997
Newsgroups	0.3896 ± 0.0043	0.3892 ± 0.0042	0.3921 ± 0.0122	0.3261 ± 0.0193	0.3926 ± 0.0113	0.4000 ± 0.0153
Spam	0.8454 ± 0.0037	0.8237 ± 0.0040	0.8719 ± 0.0353	0.8699 ± 0.0236	0.9028 ± 0.0128	0.8920 ± 0.0414
Yale	0.5442 ± 0.0129	0.4739 ± 0.0135	0.5200 ± 0.0071	0.3261 ± 0.0193	0.6327 ± 0.0209	0.2861 ± 0.0659

	NB-VAE	Isomap	DLA-GPLVM	Poisson RFLVM	Neg. binom. RFLVM	Multinomial RFLVM
Bridges	0.7485 ± 0.0613	0.8375 ± 0.0240	0.8578 ± 0.0101	0.8440 ± 0.0165	$\mathbf{0.8664 \pm 0.0191}$	0.7984 ± 0.0102
CIFAR-10	0.2671 ± 0.0048	0.2716 ± 0.0056	0.2641 ± 0.0063	$\mathbf{0.2789 \pm 0.0080}$	0.2656 ± 0.0048	0.2652 ± 0.0024
Congress	$\mathbf{0.8541 \pm 0.0074}$	0.5239 ± 0.0178	0.7815 ± 0.0185	0.7673 ± 0.0109	0.8093 ± 0.0154	0.6516 ± 0.0385
MNIST	0.2918 ± 0.0174	0.4408 ± 0.0192	0.3820 ± 0.0121	0.6494 ± 0.0210	0.4463 ± 0.0313	0.3794 ± 0.0153
Montreal	0.7246 ± 0.0131	0.7049 ± 0.0098	0.2885 ± 0.0001	$\mathbf{0.8158 \pm 0.0210}$	0.7530 ± 0.0478	0.7555 ± 0.0784
Newsgroups	0.4079 ± 0.0080	0.4021 ± 0.0098	0.3687 ± 0.0077	$\mathbf{0.4144 \pm 0.0029}$	0.4045 ± 0.0044	0.4076 ± 0.0039
Spam	$\mathbf{0.9570 \pm 0.0045}$	0.8272 ± 0.0047	0.9521 ± 0.0069	0.9515 ± 0.0023	0.9443 ± 0.0035	0.9397 ± 0.0015
Yale	0.5261 ± 0.0346	0.5891 ± 0.0155	0.4788 ± 0.0991	$\mathbf{0.6894 \pm 0.0295}$	0.5394 ± 0.0117	0.5441 ± 0.0059

have gradients of the posterior in closed form. (See Wu et al. [2017] for a discussion.) Both DLA-GPLVM and RFLVM, however, do have closed-form gradients and approximate the true manifold with similar R^2 and MSE values for $\hat{\mathbf{X}}$ and $\hat{f}_j(\mathbf{X})$ (not shown).

4.4.2. Hippocampal place cell data

Next, we checked whether a non-Gaussian RFLVM recovers an interpretable latent space when applied to a scientific problem. In particular, we use an RFLVM to model hippocampal place cell data [Wu et al., 2017]. Place cells, a type of neuron, are activated when an animal enters a particular place in its environment. Here, $\mathbf{Y}$ is an $N \times J$ matrix of count-valued spikes where n indexes time and j indexes neurons. These data were jointly recorded while measuring the position of a rat in a W-shaped maze. We are interested in reconstructing the latent positions of the rat with $\mathbf{X}$.

We quantified goodness-of-fit of the latent space by assessing how well the RFLVM captures known structure, in the form of held-out sample labels, in the low-dimensional space. After estimating $\hat{\mathbf{X}}$, we performed K-nearest neighbors (KNN) classification on $\hat{\mathbf{X}}$ with $K = 1$. We ran this classification five times using 5-fold cross validation. We report the mean and standard deviation of KNN accuracy across five experiments.

The Poisson RFLVM and DLA-GPLVM have similar performance in terms of how well they cluster samples in the latent space as measured by KNN accuracy using regions of the maze as labels. Furthermore, the models have similar performance in recovering the true rat positions $\mathbf{X}$, measured by R^2 performance (Figure 4.4.2). While this clustering would not be impressive for many benchmark data sets such as MNIST, dimension reduction for large-scale neural recordings is an open problem [Cunningham and Byron, 2014, Linderman et al., 2016, Wu et al., 2017]. These results suggest that our generalized RFLVM framework finds structure even in empirical, complex, non-Gaussian data and is competitive with models built for this specific task.

4.4.3. *Text and image data*

Finally, we examine whether an RFLVM captures the latent space of text, image, and empirical data sets. We hold out the labels and use them to evaluate the estimated latent space using the same KNN evaluation from Section 4.4.2. Across all eight data sets, the Poisson and negative binomial RFLVMs infer a low-dimensional latent variable $\hat{\mathbf{X}}$ that generally captures the latent structure as well as or better than linear methods like PCA and NMF. Moreover, adding nonlinearity but retaining a Gaussian data likelihood—as with real-valued models like Isomap [Tenenbaum et al., 2000], a variational autoencoder (VAE) [Kingma and Welling, 2013], and the Gaussian RFLVM, or even using the Poisson-likelihood DLA-GPLVM—perform worse than the Poisson and negative binomial RFLVMs (Table 4.4.1, Figure 4.4.3, Figure 4.4.4, Figure 4.4.5). The point of these results is not that RFLVMs are the best method for every data set, a spurious claim given "no free lunch" theorems [Wolpert and Macready, 1997], but rather that our framework allows for the easy implementation of a large number of practical non-conjugate GPLVMs. Thus, RFLVMs are useful when first performing nonlinear dimension reduction on non-Gaussian data. We posit that our improved performance is because the generating process from the latent space to the observations for these data sets is (in part) nonlinear, non-RBF, and integer-valued. See Section 4A.11.2 for wall-time experiments for the models in Table 4.4.1.

4.5 Discussion

We presented a framework that uses random Fourier features to induce computational tractability between the latent variables and GP-distributed maps in Gaussian process latent variable models. Our approach allows the Gaussian model to be extended to arbitrary distributions, and we derived an RFLVM for Gaussian, Poisson and logistic distributions. We described distribution-specific inference techniques for each posterior sampling step. Our empirical results showed that each was competitive in downstream analyses with existing distribution-specific approaches on diverse data sets including synthetic, image, text, and multi-neuron spike train data. We are particularly interested in exploring extensions of our generalized RFLVM framework to more sophisticated models such as extending GP dynamic state-space models [Ko and Fox, 2011] to count data and neuroscience applications, which assume temporal structure in $\mathbf{X}$.

RFLVMs have a number of limitations that motivate future work. First, the latent variables are unidentifiable up to scale and rotation. Our rescaling procedure (Section 4.3.2) does not allow heteroscedastic dimensions and enforces orthogonality between the Gaussian latent variables. This prevents the use of more structured priors, such as a GP prior on $\mathbf{X}$, since any inferred structure is eliminated between iterations. We are interested in adopting constraints from factor analysis literature to address the identifiability issues without a restrictive rescaling procedure [Erosheva and Curtis, 2011, Millsap, 2001, Ghosh and Dunson, 2009]. Second, label switching in mixture models is a well-studied challenge that is present in our model. Enforcing identifiability may improve inference and model interpretability [Stephens, 2000]. In this work, we focused on distributions in the exponential family because this class is both ubiquitous and well-studied. However, we do not see obvious obstacles to extending our approach to data likelihoods outside the exponential family, as we only need closed-form gradients to learn the latent space. Finally, our model has a number of hyperparameters such as the latent dimension, the number of random Fourier features, and the number of Gibbs sampling iterations. Both simplifying the model and estimating these hyperparameters from data are two important directions to improve the usability of RFLVMs.

4A Appendix

4A.1 Kernel trick

4A.1.1. Implicit inner products

Imagine we have some data for a classification problem that is not linearly separable. A classic example is Figure 4A.1.1 (left). We would like to use a linear classifier. How might we do this? One idea is to augment our data's features so that we can "lift" it into a higher dimensional space in which our data are linearly separable (Figure 4A.1.1, right).

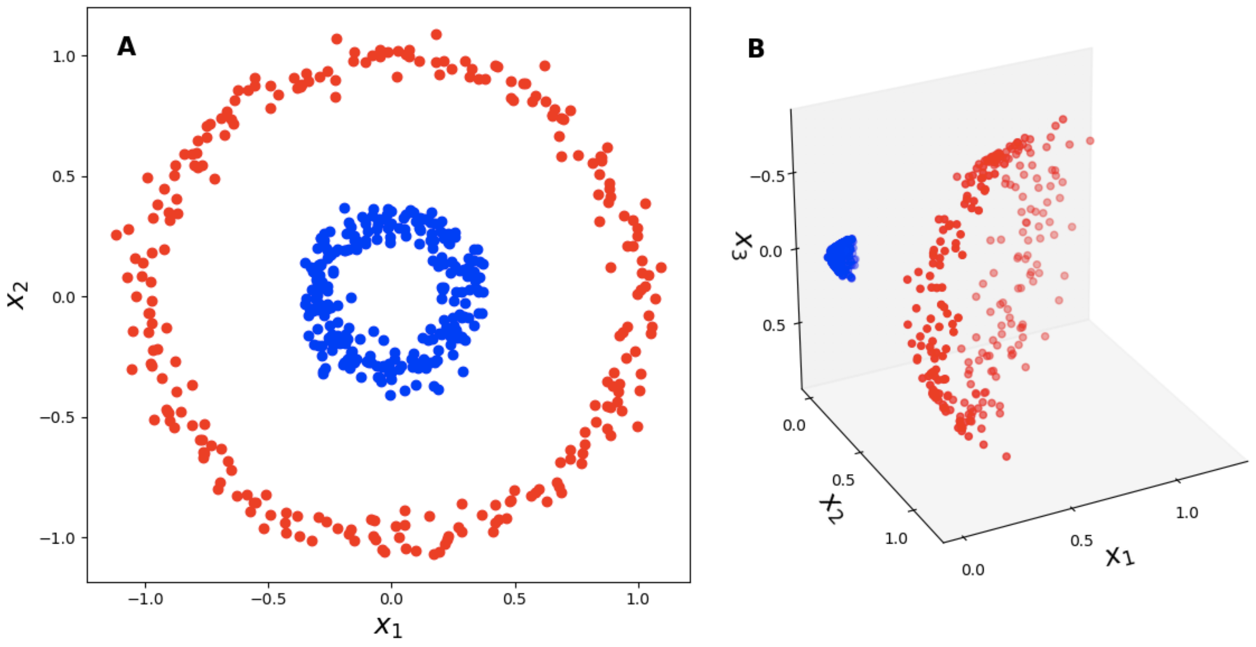

Figure 4A.1.1: The "lifting trick". (a) A binary classification problem that is not linearly separable in $\mathbb{R}^2$. (b) A lifting of the data into $\mathbb{R}^3$ using a polynomial kernel, $\phi([x_1 \;\; x_2]) = [x_1^2 \;\; x_2^2 \;\; \sqrt{2}x_1x_2]^\top$.

Let's formalize this approach. Let our data be $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$ where $\mathbf{x}_n \in \mathbb{R}^D$ in general. Now consider $D = 2$ and a single data point

$$\mathbf{x}_n = \begin{bmatrix} x_{n1} \\ x_{n2} \end{bmatrix}. \tag{4.69}$$

We might transform each data point with a function (the polynomial kernel),

$$\phi(\mathbf{x}_n) = \begin{bmatrix} x_{n1}^2 \\ x_{n2}^2 \\ \sqrt{2} x_{n1} x_{n2} \end{bmatrix}. \tag{4.70}$$

Since our new data, $\phi(\mathbf{x}_n)$, is in $\mathbb{R}^3$, we might be able to find a hyperplane $\boldsymbol{\beta}$ in 3D to separate our observations,

$$\boldsymbol{\beta}^\top \phi(\mathbf{x}_n) = \beta_0 + \beta_1 x_{n1}^2 + \beta_2 x_{n2}^2 + \beta_3 \sqrt{2} x_{n1} x_{n2} = 0. \tag{4.71}$$

This idea, while powerful, is not the kernel trick, but it deserves a name. Let us call it *the lifting trick*.[9]

In order to find this hyperplane, we need to run a classification algorithm on our data *after* it has been lifted into three-dimensional space. At this point, we could be done. We take $\mathbb{R}^D$, perform our lifting trick into $\mathbb{R}^J$ where $D \ll J$, and then use a method like logistic regression to try to linearly classify it. However, this might be expensive for an expressive enough $\phi(\cdot)$. For N data points lifted into J dimensions, we need NJ operations just to preprocess the data. But we can avoid computing $\phi(\cdot)$ entirely while still doing linear classification in this lifted space if we're clever. This second trick is the kernel trick.

[9]I am not aware of a name for this trick, but I find naming things useful.

Notice that for $\phi(\cdot)$, we have

$$
\begin{aligned}
\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m) &= \begin{bmatrix} x_{n1}^2 & x_{n2}^2 & \sqrt{2} x_{n1} x_{n2} \end{bmatrix} \cdot \begin{bmatrix} x_{m1}^2 \\ x_{m2}^2 \\ \sqrt{2} x_{m1} x_{m2} \end{bmatrix} \\
&= x_{n1}^2 x_{m1}^2 + x_{n2}^2 x_{m2}^2 + 2 x_{n1} x_{n2} x_{m1} x_{m2}.
\end{aligned} \tag{4.72}
$$

We would then need to compute this for all our N data points. As we discussed, the problem with this approach is scalability. However, consider the following derivation,

$$
\begin{aligned}
(\mathbf{x}_m^\top \mathbf{x}_m)^2 &= \left(\begin{bmatrix} x_{n1} & x_{n2} \end{bmatrix} \cdot \begin{bmatrix} x_{m1} \\ x_{m2} \end{bmatrix} \right)^2 \\
&= (x_{n1} x_{m1} + x_{n2} x_{m2})^2 \\
&= (x_{n1} x_{m1})^2 + (x_{n2} x_{m2})^2 + 2(x_{n1} x_{m1})(x_{n2} x_{m2}) \\
&= \phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m).
\end{aligned} \tag{4.73}
$$

What just happened? Rather than lifting our data into $\mathbb{R}^3$ and computing an inner product, we just computed an inner product in $\mathbb{R}^2$ and then squared the sum. While both derivations have a similar number of mathematical symbols, the actual number of operations is much smaller for the second approach. This is because a inner product in $\mathbb{R}^2$ is two multiplications and a sum. The square is just the square of a scalar, so 4 operations. The first approach squared three components of two vectors (6 operations), then performed an inner product (3 multiplications, 1 sum) for 9 operations.

This is the kernel trick: we can avoid expensive operations in high dimensions by finding an appropriate kernel function $k(\mathbf{x}_n, \mathbf{x}_m)$ that is equivalent to the inner product in higher dimensional space. In our example above, $k(\mathbf{x}_n, \mathbf{x}_m) = (\mathbf{x}_n^\top \mathbf{x}_m)^2$. In other words, the kernel trick performs the lifting trick for cheap.

4A.1.2. Mercer's theorem

The mathematical basis for the kernel trick was proven by James Mercer [Mercer, 1909]. Mercer proved that any positive definition function $k(\mathbf{x}_n, \mathbf{x}_m)$ with $\mathbf{x}_n, \mathbf{x}_m \in \mathbb{R}^D$ defines an inner product of another vector space $\mathcal{V}$. Thus, if you have a function $\phi(\cdot)$ such that $\langle \phi(\mathbf{x}_n), \phi(\mathbf{x}_m) \rangle_{\mathcal{V}}$ is a valid inner product in $\mathcal{V}$, you know a kernel function exists that can perform the lifting trick for cheap. Alternatively, if you have a positive definite kernel, you can deconstruct its implicit basis function $\phi(\cdot)$.

This idea is formalized in Mercer's Theorem[10]:

Definition 4A.1.1. Mercer's Theorem: *A symmetric function $k(\mathbf{x}, \mathbf{y})$ can be expressed as an inner product*

$$k(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle \tag{4.74}$$

for some $\phi(\cdot)$ if and only if $k(\mathbf{x}, \mathbf{y})$ is positive semidefinite, i.e.

$$\int k(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0, \qquad \forall g \tag{4.75}$$

or, equivalently, if

$$\begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix} \tag{4.76}$$

is positive semidefinite for any collection $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$.

This theorem is if and only if, meaning we could explicitly construct a kernel function $k(\cdot, \cdot)$ for a given $\phi(\cdot)$ or we could take a kernel function and use it without having an explicit representation of $\phi(\cdot)$.

If we assume everything is real-valued, then we can demonstrate this fact easily. Let $\mathbf{K}$ be the positive semidefinite Gram matrix above. Since it is real and symmetric, it has an

[10]This definition was taken from Michael Jordan's lecture notes: https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf

eigendecomposition of the form

$$\mathbf{K} = \mathbf{U}^\top \mathbf{\Lambda} \mathbf{U} \tag{4.77}$$

where $\mathbf{\Lambda} = \text{diag}([\lambda_1, \dots, \lambda_N])$. Since $\mathbf{K}$ is positive definite, then $\lambda_n \geq 0$ and the square root is real-valued. We can write an element of $\mathbf{K}$ as

$$K_{ij} = \begin{bmatrix} \mathbf{\Lambda}^{1/2} & \mathbf{U}_{:,i} \end{bmatrix} \begin{bmatrix} \mathbf{\Lambda}^{1/2} \\ \mathbf{U}_{:,j} \end{bmatrix}. \tag{4.78}$$

Define $\phi(\mathbf{x}_i) := \mathbf{\Lambda}^{1/2} \mathbf{U}_{:,i}$. Therefore, if our kernel function is positive semidefinite—if it defines a Gram matrix that is positive semidefinite—then there exists a function $\phi : \mathcal{X} \mapsto \mathcal{V}$ such that

$$k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^\top \phi(\mathbf{y}) \tag{4.79}$$

where $\mathcal{X}$ is the space of samples.

4A.1.3. Infinite-dimensional feature spaces

An interesting consequence of the kernel trick is that kernel methods, equipped with the appropriate kernel function, can be viewed as operating in possibly infinite-dimensional feature space. As an example, consider the radial basis function (RBF) kernel,

$$k_{\text{RBF}}(\mathbf{x}, \mathbf{y}) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{y}\|_2^2\right). \tag{4.80}$$

Let's take it for granted that this is a valid positive semidefinite kernel. Let $k_{\texttt{poly(r)}}$ denote a polynomial kernel of degree r, and let $\gamma = 1/2$. Then

$$\begin{aligned}
k_{\texttt{RBF}}(\mathbf{x}, \mathbf{y}) &= \exp\left(-\frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2\right) \\
&= \exp\left(-\frac{1}{2}\langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y}\rangle\right) \\
&\overset{\star}{=} \exp\left(-\frac{1}{2}\left[\langle \mathbf{x}, \mathbf{x} - \mathbf{y}\rangle - \langle \mathbf{y}, \mathbf{x} - \mathbf{y}\rangle\right]\right) \\
&\overset{\star}{=} \exp\left(-\frac{1}{2}\left[\langle \mathbf{x}, \mathbf{x}\rangle - \langle \mathbf{x}, \mathbf{y}\rangle - [\langle \mathbf{y}, \mathbf{x}\rangle - \langle \mathbf{y}, \mathbf{y}\rangle]\rangle\right]\right) \\
&= \exp\left(-\frac{1}{2}\left[\langle \mathbf{x}, \mathbf{x}\rangle + \langle \mathbf{y}, \mathbf{y}\rangle - 2\langle \mathbf{x}, \mathbf{y}\rangle\right]\right) \\
&= \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right) \exp\left(-\frac{1}{2}\|\mathbf{y}\|^2\right) \exp\left(-2\langle \mathbf{x}, \mathbf{y}\rangle\right)
\end{aligned} \tag{4.81}$$

Above, the two steps labeled $\star$ leverage the fact that

$$\langle \mathbf{u} + \mathbf{v}, \mathbf{w}\rangle = \langle \mathbf{u}, \mathbf{w}\rangle + \langle \mathbf{v}, \mathbf{w}\rangle \tag{4.82}$$

in general for inner products. Now let C be a constant,

$$C \coloneqq \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right) \exp\left(-\frac{1}{2}\|\mathbf{y}\|^2\right), \tag{4.83}$$

and note that the Taylor expansion of $e^{f(x)}$ is

$$e^{f(x)} = \sum_{r=0}^{\infty} \frac{[f(x)]^r}{r!}. \tag{4.84}$$

We can write the RBF kernel as

$$\begin{aligned}
k_{\texttt{RBF}}(\mathbf{x}, \mathbf{y}) &= C \exp\left(-2\langle \mathbf{x}, \mathbf{y}\rangle\right) \\
&= C \sum_{r=0}^{\infty} \frac{\langle \mathbf{x}, \mathbf{y}\rangle^r}{r!} \\
&= C \sum_{r}^{\infty} \frac{k_{\texttt{poly(r)}}(\mathbf{x}, \mathbf{y})}{r!}.
\end{aligned} \tag{4.85}$$

So the RBF kernel can be viewed as an infinite sum over polynomial kernels. As r increases, each polynomial kernel lifts the data into higher dimensions, and the RBF kernel is an infinite sum over these kernels.

4A.2 Gaussian kernel derivation

Let $\boldsymbol{\delta} = \mathbf{x} - \mathbf{y}$. Then:

$$\begin{aligned}
&\mathbb{E}_{\mathbf{w}}[h(\mathbf{x})h(\mathbf{y})^*] \\
&= \mathbb{E}_{\mathbf{w}}[\exp(i\mathbf{w}^\top \mathbf{x}) \exp(-i\mathbf{w}^\top \mathbf{y}))] \\
&= \mathbb{E}_{\mathbf{w}}[\exp(i\mathbf{w}^\top \boldsymbol{\delta})] \\
&= \int_{\mathbb{R}^D} p(\mathbf{w}) \exp(i\mathbf{w}^\top \boldsymbol{\delta}) \mathrm{d}\mathbf{w} \\
&= (2\pi)^{-D/2} \int_{\mathbb{R}^D} \exp\left(-\frac{1}{2}\mathbf{w}^\top \mathbf{w}\right) \exp(i\mathbf{w}^\top \boldsymbol{\delta}) \mathrm{d}\mathbf{w} \\
&= (2\pi)^{-D/2} \int_{\mathbb{R}^D} \exp\left(-\frac{1}{2}\mathbf{w}^\top \mathbf{w} - i\mathbf{w}^\top \boldsymbol{\delta}\right) \mathrm{d}\mathbf{w} \qquad (4.86) \\
&= (2\pi)^{-D/2} \int_{\mathbb{R}^D} \exp\left(-\frac{1}{2}\left(\mathbf{w}^\top \mathbf{w} - 2i\mathbf{w}^\top \boldsymbol{\delta} - \boldsymbol{\delta}^\top \boldsymbol{\delta}\right) - \frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{\delta}\right) \mathrm{d}\mathbf{w} \\
&= (2\pi)^{-D/2} \exp\left(-\frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{\delta}\right) \underbrace{\int_{\mathbb{R}^D} \exp\left(-\frac{1}{2}\left(\mathbf{w} - i\boldsymbol{\delta}\right)^\top \left(\mathbf{w} - i\boldsymbol{\delta}\right)\right) \mathrm{d}\mathbf{w}}_{(2\pi)^{D/2}} \\
&= \exp\left(-\frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{\delta}\right) \\
&= k(\boldsymbol{\delta}).
\end{aligned}$$

In other words, $k(\cdot)$ is the Gaussian kernel when $P_{\mathbf{w}}$ is the spherical Gaussian distribution.

4A.3 Proof of $\cos(x + y) = \cos(x)\cos(y) - \sin(x)\sin(y)$

Recall from trigonometry[11] that

$$\cos(x + y) = \cos(x)\cos(y) - \sin(x)\sin(y). \qquad (4.87)$$

[11]See this Khan Academy video for a proof: https://www.youtube.com/watch?v=0VBQnR2h8XM

Furthermore, note that $\cos(-x) = \cos(x)$ since the cosine function is symmetric about $x = 0$. And the sine function has odd symmetry, $\sin(-x) = -\sin(x)$. Thus, with a little clever manipulation, we can write

$$\begin{aligned}
&\cos(x+y) + \cos(x-y) \\
&= \cos(x+y) + \cos(x+(-y)) \\
&= [\cos(x)\cos(y) - \sin(x)\sin(y)] + [\cos(x)\cos(-y) - \sin(x)\sin(-y)] \\
&= [\cos(x)\cos(y) - \cancel{\sin(x)\sin(y)}] + [\cos(x)\cos(y) + \cancel{\sin(x)\sin(y)}] \\
&= 2\cos(x)\cos(y).
\end{aligned} \tag{4.88}$$

4A.4 Expectation of $\cos(\mathbf{t} + b)$

Note that

$$\mathbb{E}_{\mathbf{w}}\left[\cos(\mathbf{w}^\top(\mathbf{x}+\mathbf{y}) + 2b)\right] = \mathbb{E}_{\mathbf{w}}\left[\mathbb{E}_b\left[\cos(\mathbf{w}^\top(\mathbf{x}+\mathbf{y}) + 2b) \mid \mathbf{w}\right]\right] \tag{4.89}$$

holds by the law of total expectation. We claim the inner conditional expectation is zero. To ease notation, let $\mathbf{t} = \mathbf{w}^\top(\mathbf{x}-\mathbf{y})$. Then

$$\begin{aligned}
\mathbb{E}_b\left[\cos(\mathbf{t}+2b) \mid \mathbf{w}\right] &= \int_0^{2\pi} \frac{\cos(\mathbf{t}+2b)}{2\pi}\mathrm{d}b \\
&= \frac{1}{2\pi}\int_0^{2\pi} \cos(\mathbf{t}+2b)\mathrm{d}b \\
&= \frac{1}{2\pi}\left[\sin(\mathbf{t}+2b)\Big|_0^{2\pi}\right] \\
&= \frac{1}{2\pi}\left[\sin(\mathbf{t}) - \sin(\mathbf{t}+4\pi)\right] \\
&= 0.
\end{aligned} \tag{4.90}$$

The last step holds because $\sin(\mathbf{t}) = \sin(\mathbf{t} \pm 2\pi k)$ for any integer k.

4A.5 Alternative definition of random features

Consider this alternative definition of the random map:

$$\zeta_{\mathbf{w}_m}(\mathbf{x}) = \begin{bmatrix} \cos(\mathbf{w}_m^\top \mathbf{x}) \\ \sin(\mathbf{w}_m^\top \mathbf{x}) \end{bmatrix}. \tag{4.91}$$

Draw $M' = M/2$ samples

$$\mathbf{w}_m \overset{\mathsf{iid}}{\sim} P_{\mathbf{w}}. \tag{4.92}$$

Then

$$\begin{aligned} \frac{1}{M'} \sum_{m=1}^{M'} \zeta_{\mathbf{w}_m}(\mathbf{x})^\top \zeta_{\mathbf{w}_m}(\mathbf{y}) &:= \frac{2}{M} \sum_{m=1}^{M/2} \left(\begin{bmatrix} \cos(\mathbf{w}_m^\top \mathbf{x}) \\ \sin(\mathbf{w}_m^\top \mathbf{x}) \end{bmatrix}^\top \begin{bmatrix} \cos(\mathbf{w}_m^\top \mathbf{y}) \\ \sin(\mathbf{w}_m^\top \mathbf{y}) \end{bmatrix} \right) \\ &= \frac{2}{M} \sum_{m=1}^{M/2} \cos(\mathbf{w}_m^\top \mathbf{x}) \cos(\mathbf{w}_m^\top \mathbf{y}) + \sin(\mathbf{w}_m^\top \mathbf{x}) \sin(\mathbf{w}_m^\top \mathbf{y}) \\ &\overset{\star}{=} \frac{2}{M} \sum_{m=1}^{M/2} \cos(\mathbf{w}_m^\top \mathbf{x} - \mathbf{w}_m^\top \mathbf{y}) \\ &\approx \mathbb{E}_{\mathbf{w}}[\cos(\mathbf{w}^\top (\mathbf{x} - \mathbf{y}))] \\ &= k(\mathbf{x}, \mathbf{y}). \end{aligned} \tag{4.93}$$

Step $\star$ just applies the product identities from trigonometry:

$$\begin{aligned} 2\sin(x)\sin(y) &= \cos(x-y) - \cancel{\cos(x+y)}, \\ 2\cos(x)\cos(y) &= \cos(x-y) + \cancel{\cos(x+y)}. \end{aligned} \tag{4.94}$$

The right-most terms above cancel in Equation (4.93), and we get $2\cos(x - y)$.

4A.6 Code for kernel ridge regression with random features

```
import numpy as np
from   sklearn.linear_model import Ridge

```

```python
class RFFRidgeRegression:

    def __init__(self, rff_dim=1, alpha=1.0, sigma=1.0):
        """Kernel ridge regression using random Fourier features.

        rff_dim : Dimension of random feature.
        alpha :   Regularization strength. Should be a positive float.
        """
        self.fitted  = False
        self.rff_dim = rff_dim
        self.sigma   = sigma
        self.lm      = Ridge(alpha=alpha)
        self.b_      = None
        self.W_      = None

    def fit(self, X, y):
        """Fit model with training data X and target y.
        """
        Z, W, b = self._get_rffs(X, return_vars=True)
        self.lm.fit(Z.T, y)
        self.b_ = b
        self.W_ = W
        self.fitted = True
        return self

    def predict(self, X):
        """Predict using fitted model and testing data X.
        """
        Z = self._get_rffs(X, return_vars=False)
        return self.lm.predict(Z.T)

    def _get_rffs(self, X):
        """Return random Fourier features based on data X, as well as random
        variables W and b.
        """
        N, D = X.shape
        if self.W_ is not None:
            W, b = self.W_, self.b_
        else:
            W = np.random.normal(loc=0, scale=1, size=(self.rff_dim, D))
            b = np.random.uniform(0, 2*np.pi, size=self.rff_dim)
        B    = np.repeat(b[:, np.newaxis], N, axis=1)
        norm = 1./ np.sqrt(self.rff_dim)
        Z    = norm * np.sqrt(2) * np.cos(self.sigma * W @ X.T + B)
        return Z, W, b
```

4A.7 Pólya–gamma augmentation

Consider the task of Bayesian inference for models with binomial likelihoods parameterized by log-odds. Two well-known examples of such models are logistic regression and negative binomial regression. For example, in logistic regression (also discussed in Section 2.1), the dependent variables are assumed to be i.i.d. from a Bernoulli distribution with parameter $\mathbf{p} := [p_1, \dots, p_N]$, and therefore the likelihood function is

$$\mathcal{L}_N(\mathbf{p}) \propto \prod_{n=1}^{N} p_n^{y_n}(1-p_n)^{1-y_n} = p_n^{\sum y_n}(1-p_n)^{N-\sum y_n}. \tag{4.95}$$

The observations interact with the response through a linear relationship with the log-odds,

$$\log\left(\frac{p_n}{1-p_n}\right) = \beta_0 + x_{n1}\beta_1 + \cdots + x_{nD}\beta_D = \boldsymbol{\beta}^\top \mathbf{x}_n. \tag{4.96}$$

If we solve for p_n in Equation (4.96), we get

$$p_n = \frac{\exp(\boldsymbol{\beta}^\top \mathbf{x}_n)}{1+\exp(\boldsymbol{\beta}^\top \mathbf{x}_n)}, \tag{4.97}$$

and we can represent the likelihood—now in terms of $\boldsymbol{\beta}$ instead of $\mathbf{p}$—as

$$\mathcal{L}_N(\boldsymbol{\beta}) \propto \frac{[\exp(\boldsymbol{\beta}^\top \mathbf{x})]^{\sum y_n}}{[1+\exp(\boldsymbol{\beta}^\top \mathbf{x})]^N}. \tag{4.98}$$

Due to this functional form, Bayesian inference for logistic regression is intractable [Bishop, 2006]. This is because the evidence would require normalizing the product of a prior distribution (e.g. a Gaussian prior on $\boldsymbol{\beta}$) times the likelihood function in Equation (4.98). A similar problem arises for other models with binomial likelihoods parameterized by log-odds.

However, Polson et al. [2013] introduced a new method called *Pólya–gamma augmentation* that allows for constructing simple Gibbs samplers for these models. The goal of this section is to discuss their main results in detail, understand the derivations, and implement this Gibbs sampler.

4A.7.1. Pólya–gamma random variables

If ω is a Pólya–gamma distributed random variable with parameters $b > 0$ and $c \in \mathbb{R}$, denoted $\omega \sim \text{PG}(b, c)$, then

$$p(\omega \mid b, c) = \frac{1}{2\pi^2} \sum_{k=1}^{\infty} \frac{g_k}{(k - 1/2)^2 + c^2/(4\pi^2)} \tag{4.99}$$

where $g_k \sim \text{Gamma}(b, 1)$ are independent gamma random variables. While this density function is complicated, Polson et al. [2013] show that all the finite moments of ω can be written in closed form. For example, the expectation can be calculated immediately,

$$\mathbb{E}[\omega] = \frac{b}{2c} \tanh(c/2). \tag{4.100}$$

In particular, Polson et al. [2013] proved two useful properties of Pólya–gamma variables. First,

$$\frac{(e^{\psi})^a}{(1 + e^{\psi})^b} = 2^{-b} e^{\kappa\psi} \int_0^{\infty} e^{-\omega\psi^2/2} p(\omega) \mathrm{d}\omega \tag{4.101}$$

where $\kappa = a - b/2$ and $p(\omega) = \text{PG}(\omega \mid b, 0)$. And second,

$$p(\omega \mid \psi) \sim \text{PG}(b, \psi). \tag{4.102}$$

While the proof of Equation (4.101) is a few lines in the paper, it is dense. See Section 4A.7.5 for the proof with details. See Section 4A.7.6 for a derivation of Equation (4.102).

4A.7.2. Logistic regression with PG augmentation

It may not be immediately obvious why the right-hand side of Equation (4.101) is useful. Its utility is that we can construct Gibbs samplers of logistic models or models with likelihoods of the form in Equation (4.103). To be concrete, consider Bayesian inference for logistic regression. Recall that the nth observation's contribution to the likelihood Equation (4.98), denoted $\mathcal{L}_n(\boldsymbol{\beta})$, is

$$\mathcal{L}_n(\boldsymbol{\beta}) = \frac{\left(\exp(\boldsymbol{\beta}^\top \mathbf{x}_n)\right)^{y_n}}{1+\exp(\boldsymbol{\beta}^\top \mathbf{x}_n)}. \tag{4.103}$$

Using Equation (4.101), we can express this likelihood contribution as

$$\begin{aligned} \frac{\left(\exp(\boldsymbol{\beta}^\top \mathbf{x}_n)\right)^{y_n}}{1+\exp(\boldsymbol{\beta}^\top \mathbf{x}_n)} &= -2\exp\left\{\kappa_n \boldsymbol{\beta}^\top \mathbf{x}_n\right\} \int_0^\infty \exp\left\{-\omega_n(\boldsymbol{\beta}^\top \mathbf{x}_n)^2/2\right\} p(\omega_n \mid 1, 0)\mathrm{d}\omega_n \\ &= -2\exp\left\{\kappa_n \boldsymbol{\beta}^\top \mathbf{x}_n\right\} \mathbb{E}_{p(\omega_n \mid 1,0)}\left[\exp(-\omega_n(\boldsymbol{\beta}^\top \mathbf{x}_n)^2/2)\right], \end{aligned} \tag{4.104}$$

where $\kappa_n := y_n - 1/2$. Note that if we condition on ω_n, the likelihood contribution in Equation (4.104) is Gaussian in $\boldsymbol{\beta}$:

$$\begin{aligned} p(\boldsymbol{\beta} \mid \boldsymbol{\Omega}, \mathbf{y}) &= p(\boldsymbol{\beta}) \prod_{n=1}^N \mathcal{L}_n(\boldsymbol{\beta} \mid \omega_n) \\ &\overset{\ddagger}{\propto} p(\boldsymbol{\beta}) \prod_{n=1}^N \exp\left\{\kappa_n \boldsymbol{\beta}^\top \mathbf{x}_n\right\} \exp\left\{-\omega_n(\boldsymbol{\beta}^\top \mathbf{x}_n)^2/2\right\} \\ &\overset{\star}{\propto} p(\boldsymbol{\beta}) \prod_{n=1}^N \exp\left\{-\frac{\omega_n}{2}\left(\boldsymbol{\beta}^\top \mathbf{x}_n - \frac{\kappa_n}{\omega_n}\right)^2\right\} \\ &= p(\boldsymbol{\beta}) \exp\left\{-\frac{1}{2}(\mathbf{z} - \mathbf{X}\boldsymbol{\beta})^\top \boldsymbol{\Omega} (\mathbf{z} - \mathbf{X}\boldsymbol{\beta})\right\} \\ &\overset{\dagger}{=} p(\boldsymbol{\beta}) \exp\left\{-\frac{1}{2}\left(\boldsymbol{\beta} - \mathbf{X}^{-1}\mathbf{z}\right)^\top \mathbf{X}^\top \boldsymbol{\Omega} \mathbf{X}\left(\boldsymbol{\beta} - \mathbf{X}^{-1}\mathbf{z}\right)\right\}, \end{aligned} \tag{4.105}$$

where $\mathbf{z} := \langle \kappa_1/\omega_1, \dots, \kappa_N/\omega_N \rangle$ and $\boldsymbol{\Omega} := \operatorname{diag}([\omega_1, \dots, \omega_N])$. Step $\ddagger$ holds because the expectation in Equation (4.104) is constant if we condition on ω_n. Step $\star$ works by completing the square (see Section 4A.7.7), while step $\dagger$ is just a little algebra (see Section 4A.7.8).

In summary, if our prior on $\boldsymbol{\beta}$ is Gaussian (quadratic in $\boldsymbol{\beta}$), then Equation (4.105) is tractable because the posterior can be written as Gaussian ($\boldsymbol{\beta}$). This suggests that we can construct a Gibbs sampler, where we repeatedly sample $\boldsymbol{\Omega}$ given $\boldsymbol{\beta}$ and then $\boldsymbol{\beta}$ given $\boldsymbol{\Omega}$.

4A.7.3. PG-augmented Gibbs sampler

To perform Gibbs sampling with two parameters, we repeatedly fix one parameter while conditionally sampling from the other. Concretely for us, we first initialize $\boldsymbol{\beta}$. Then we for $t = 1, \dots T$, we sample

$$\begin{aligned} \boldsymbol{\Omega}^{(t+1)} &\sim P_{\boldsymbol{\Omega} \mid \boldsymbol{\beta}^{(t)}}, \\ \boldsymbol{\beta}^{(t+1)} &\sim P_{\boldsymbol{\beta} \mid \boldsymbol{\Omega}^{(t+1)}}. \end{aligned} \tag{4.106}$$

Provided we can compute each density above, we're done. The first density comes from Equation (4.102). We know that

$$\omega_n \mid \boldsymbol{\beta} \sim \text{PG}(1, \boldsymbol{\beta}^{\top} \mathbf{x}_n). \tag{4.107}$$

In other words, we sample each element along the diagonal of $\boldsymbol{\Omega}$ using Equation (4.107). The second equation is a bit trickier. If the prior on $\boldsymbol{\beta}$ is $\mathcal{N}(\mathbf{b}, \mathbf{B})$, then the conditional distribution $P_{\boldsymbol{\beta} \mid \boldsymbol{\Omega}, \mathbf{y}}$ is

$$\boldsymbol{\beta} \mid \boldsymbol{\Omega}, \mathbf{y} \sim \mathcal{N}(\mathbf{m}_{\boldsymbol{\omega}}, \mathbf{V}_{\boldsymbol{\omega}}), \tag{4.108}$$

where

$$\begin{aligned} \mathbf{V}_{\boldsymbol{\omega}} &\coloneqq (\mathbf{X}^{\top} \boldsymbol{\Omega} \mathbf{X} + \mathbf{B}^{-1})^{-1}, \\ \mathbf{m}_{\boldsymbol{\omega}} &\coloneqq \mathbf{V}_{\boldsymbol{\omega}} (\mathbf{X}^{\top} \boldsymbol{\kappa} + \mathbf{B}^{-1} \mathbf{b}), \end{aligned} \tag{4.109}$$

where $\boldsymbol{\kappa} \coloneqq \langle \kappa_1, \dots, \kappa_N \rangle$. The derivation just requires the matrix formula for completing the square and a bit of algebra. (See Section 4A.7.9 for details; it is worth skimming this derivation to confirm that the reason it works is because $\boldsymbol{\beta}$ is Gaussian.) Thinking algorithmically, if we can sample ω_n, we can use this reparameterization to get a conditionally Gaussian likelihood centered at $\mathbf{X}^{-1}\mathbf{z}$ with covariance $\mathbf{X}^{\top} \boldsymbol{\Omega} \mathbf{X}$.

4A.7.4. Demo

Section 4 of Polson et al. [2013] discusses simulating PG random variables. The details of this are beyond the scope of this thesis, and thankfully Scott Linderman has already created a

Cython port[12] of Jesse Windle's code[13] for sampling PG random variables. Using this library, we can easily construct a Gibbs sampler for logistic regression using PG augmentation:

```python
import matplotlib.pyplot as plt
import numpy as np
from   numpy.linalg import inv
import numpy.random as npr
from   pypolyagamma import PyPolyaGamma

def sigmoid(x):
    """Numerically stable sigmoid function.
    """
    return np.where(x >= 0,
                    1 / (1 + np.exp(-x)),
                    np.exp(x) / (1 + np.exp(x)))

def multi_pgdraw(pg, B, C):
    """Utility function for calling `pgdraw` on every pair in vectors B, C.
    """
    return np.array([pg.pgdraw(b, c) for b, c in zip(B, C)])

def gen_bimodal_data(N, p):
    """Generate bimodal data for easy sanity checking.
    """
    y     = npr.random(N) < p
    X     = np.empty(N)
    X[y]  = npr.normal(0, 1, size=y.sum())
    X[~y] = npr.normal(4, 1.4, size=(~y).sum())
    return X, y.astype(int)

# Set priors and create data.
N_train = 1000
N_test  = 1000
b       = np.zeros(2)
B       = np.diag(np.ones(2))
```

[12]https://github.com/slinderman/pypolyagamma
[13]https://github.com/jwindle/BayesLogit

```
X_train, y_train = gen_bimodal_data(N_train, p=0.3)
X_test, y_test   = gen_bimodal_data(N_test, p=0.3)
# Prepend 1 for the bias \beta_0.
X_train = np.vstack([np.ones(N_train), X_train])
X_test  = np.vstack([np.ones(N_test), X_test])

# Peform Gibb sampling for T iterations.
pg         = PyPolyaGamma()
T          = 100
Omega_diag = np.ones(N_train)
beta_hat   = npr.multivariate_normal(b, B)
k          = y_train - 1/2.

for _ in range(T):
    # \omega ~ PG(1, x*\beta).
    Omega_diag = multi_pgdraw(pg, np.ones(N_train), X_train.T @ beta_hat)
    # \beta ~ N(m, V).
    V          = inv(X_train @ np.diag(Omega_diag) @ X_train.T + inv(B))
    m          = np.dot(V, X_train @ k + inv(B) @ b)
    beta_hat   = npr.multivariate_normal(m, V)

y_pred = npr.binomial(1, sigmoid(X_test.T @ beta_hat))
bins = np.linspace(X_test.min()-3., X_test.max()+3, 100)
plt.hist(X_test.T[y_pred == 0][:, 1],    color='r', bins=bins)
plt.hist(X_test.T[~(y_pred == 0)][:, 1], color='b', bins=bins)
plt.show()
```

We can see in Figure 4A.7.1 that the method works nicely. The only data points that are misclassified are where the two Gaussian distributions overlap.

4A.7.5. Proof of main result

We want to prove Equation (4.101) or

$$\frac{(e^{\psi})^a}{(1+e^{\psi})^b} = 2^{-b} e^{\kappa\psi} \int_0^{\infty} e^{-\omega\psi^2/2} p(\omega) \mathrm{d}\omega, \tag{4.110}$$

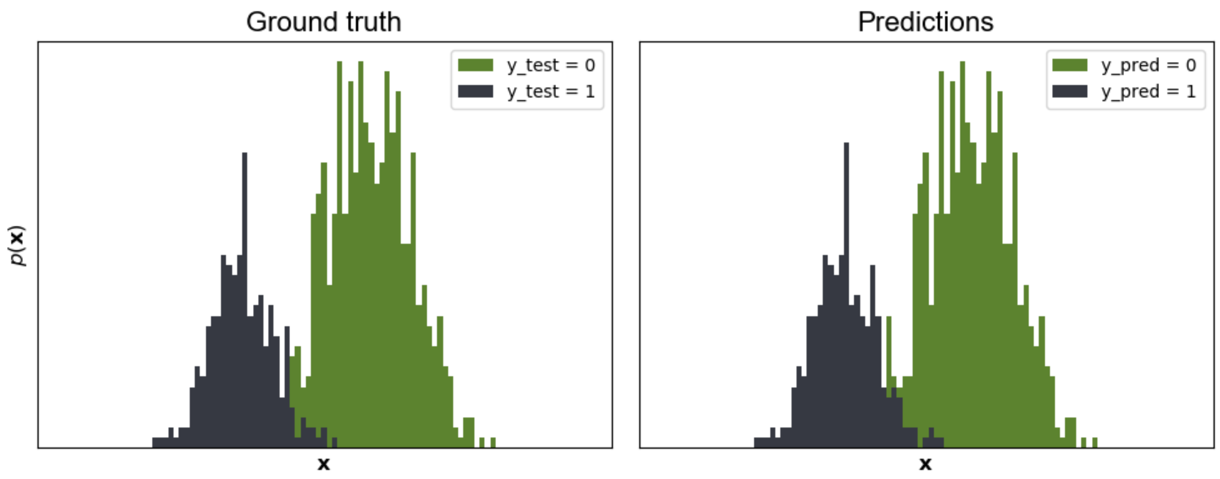

Figure 4A.7.1: (Left) Test data from a bimodal distribution colored based on ground truth binary labels. (Right) Test data colored based on predictions from a Bayesianb logistic regression model using PG-augmented Gibb sampling.

where ω is a PG random variable with PDF (Equation (4.99)). First, plug $a = \kappa + b/2$ into the left-hand side of Equation (4.101),

$$\begin{aligned}\frac{(e^{\psi})^{a}}{(1+e^{\psi})^{b}} &= \frac{(e^{\psi})^{\kappa+b/2}}{(1+e^{\psi})^{b}} \\ &= \frac{(e^{\psi})^{\kappa}(e^{\psi})^{b/2}}{(1+e^{\psi})^{b}} \\ &= \frac{(e^{\psi})^{\kappa}}{(1+e^{\psi})^{b}(e^{-\psi/2})^{b}} \\ &= \frac{(e^{\psi})^{\kappa}}{\left(\frac{1+e^{\psi}}{e^{\psi/2}}\right)^{b}}\end{aligned} \tag{4.111}$$

We can introduce the hyperbolic cosine through the identity

$$\cosh(x) = \frac{e^{x}+e^{-x}}{2}. \tag{4.112}$$

Plugging $\psi/2$ in for x and working backwards on the denominator in the last line of Equation (4.111), we have

$$\begin{aligned}\left(\frac{1+e^{\psi}}{e^{\psi/2}}\right)^b &= \left(\frac{e^{\psi}}{e^{\psi/2}}+\frac{1}{e^{\psi/2}}\right)^b \\ &= \left(e^{\psi/2}-e^{-\psi/2}\right)^2 \\ &= [2\cosh(\psi/2)]^b.\end{aligned} \tag{4.113}$$

This recapitulates the first step in Polson's proof, the line starting after the phrase, "Appealing to (3), we may write the left-hand side of (7) as...":

$$\frac{(e^{\psi})^a}{(1+e^{\psi})^b} = \frac{(e^{\psi})^{\kappa}}{\left(\frac{1+e^{\psi}}{e^{\psi/2}}\right)^b} = \frac{(e^{\psi})^{\kappa}}{[2\cosh(\psi/2)]^b} = \frac{2^{-b}(e^{\psi})^{\kappa}}{\cosh^n(\psi/2)}. \tag{4.114}$$

The next step uses Equation 4 from Polson et al. [2013]:

$$\mathbb{E}[\exp(-\omega t)] = \frac{1}{\cosh^b(\sqrt{t/2})} \tag{4.115}$$

and where we just set $t = \psi^2/2$:

$$\mathbb{E}\left[\exp\left(-\omega\frac{\psi^2}{2}\right)\right] = \frac{1}{\cosh^b(\psi/2)}. \tag{4.116}$$

Finally, we plug Equation (4.116) into Equation (4.114), and we're done:

$$\frac{(e^{\psi})^a}{(1+e^{\psi})^b} = \frac{2^{-b}(e^{\psi})^{\kappa}}{\cosh^n(\psi/2)} = 2^{-b}e^{\psi\kappa}\mathbb{E}\left[\exp\left(-\omega\frac{\psi^2}{2}\right)\right]. \tag{4.117}$$

The expectation is with respect to $\omega \sim \mathrm{PG}(b,0)$. Just apply the definition of expectation to Equation (4.117), and we're done.

4A.7.6. Proof of secondary result

By the definitions of a PG random variable and expectation,

$$\mathbb{E}[\exp(-\omega)\psi^2/2] = \int_0^{\infty} \exp(-\omega\psi^2/2)p(\omega)\mathrm{d}\omega. \tag{4.118}$$

Plug this into Polson's Equation (5) with $c = \psi$.

4A.7.7. Completing the square

This derivation relies on the univariate case of completing the square:

$$\begin{aligned}
\exp\left\{\kappa\boldsymbol{\beta}^\top\mathbf{x}\right\}\exp\left\{-\omega\frac{(\boldsymbol{\beta}^\top\mathbf{x})^2}{2}\right\} &= \exp\left\{\kappa\boldsymbol{\beta}^\top\mathbf{x} - \omega\frac{(\boldsymbol{\beta}^\top\mathbf{x})^2}{2}\right\} \\
&= \exp\left\{-\frac{\omega}{2}\left((\boldsymbol{\beta}^\top\mathbf{x})^2 - \frac{2\kappa}{\omega}(\boldsymbol{\beta}^\top\mathbf{x})\right)\right\} \\
&= \exp\left\{-\frac{\omega}{2}\left((\boldsymbol{\beta}^\top\mathbf{x})^2 - \frac{2\kappa}{\omega}(\boldsymbol{\beta}^\top\mathbf{x}) + \frac{\kappa^2}{\omega^2} - \frac{\kappa^2}{\omega^2}\right)\right\} \\
&\propto \exp\left\{-\frac{\omega}{2}\left(\boldsymbol{\beta}^\top\mathbf{x} - \frac{\kappa}{\omega}\right)^2\right\}.
\end{aligned} \tag{4.119}$$

4A.7.8. Making the likelihood quadratic in $\boldsymbol{\beta}_j$

We can show

$$(\mathbf{z} - \mathbf{X}\boldsymbol{\beta})^\top \boldsymbol{\Omega}\,(\mathbf{z} - \mathbf{X}\boldsymbol{\beta}) = \big(\boldsymbol{\beta} - \mathbf{X}^{-1}\mathbf{z})^\top\mathbf{X}^\top\boldsymbol{\Omega}\mathbf{X}(\boldsymbol{\beta} - \mathbf{X}^{-1}\mathbf{z}\big) \tag{4.120}$$

with a little algebra:

$$\begin{aligned}
&(\mathbf{z} - \mathbf{X}\boldsymbol{\beta})^\top\boldsymbol{\Omega}(\mathbf{z} - \mathbf{X}\boldsymbol{\beta}) \\
&= [\mathbf{X}(\boldsymbol{\beta} - \mathbf{X}^{-1}\mathbf{z})]^\top\boldsymbol{\Omega}[\mathbf{X}(\boldsymbol{\beta} - \mathbf{X}^{-1}\mathbf{z})] \\
&= (\boldsymbol{\beta} - \mathbf{X}^{-1}\mathbf{z})^\top\mathbf{X}^\top\boldsymbol{\Omega}\mathbf{X}(\boldsymbol{\beta} - \mathbf{X}^{-1}\mathbf{z}).
\end{aligned} \tag{4.121}$$

4A.7.9. Sum of two quadratic forms in $\mathbf{x}$

Note that the sum of two quadratic forms in $\mathbf{x}$ can be written as a single quadratic form plus a constant term that is independent of $\mathbf{x}$. Consider the equation

$$(\mathbf{x} - \boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) + (\mathbf{x} - \boldsymbol{\nu})^\top\boldsymbol{\Psi}^{-1}(\mathbf{x} - \boldsymbol{\nu}). \tag{4.122}$$

First, expand each quadratic term out:

$$\begin{aligned}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) &= \mathbf{x}^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} - 2\boldsymbol{\mu}^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} + \boldsymbol{\mu}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}, \\ (\mathbf{x}-\boldsymbol{\nu})^\top \boldsymbol{\Psi}^{-1}(\mathbf{x}-\boldsymbol{\nu}) &= \mathbf{x}^\top \boldsymbol{\Psi}^{-1}\mathbf{x} - 2\boldsymbol{\nu}^\top \boldsymbol{\Psi}^{-1}\mathbf{x} + \boldsymbol{\nu}^\top \boldsymbol{\Psi}^{-1}\boldsymbol{\nu}.\end{aligned} \tag{4.123}$$

If we combine similar terms and distribute, we get

$$\mathbf{x}^\top(\boldsymbol{\Sigma}^{-1}+\boldsymbol{\Psi}^{-1})\mathbf{x} - 2(\boldsymbol{\mu}^\top \boldsymbol{\Sigma}^{-1} + \boldsymbol{\nu}^\top \boldsymbol{\Psi}^{-1})\mathbf{x} + (\boldsymbol{\mu}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \boldsymbol{\nu}^\top \boldsymbol{\Psi}^{-1}\boldsymbol{\nu}) \tag{4.124}$$

which is again quadratic in $\mathbf{x}$. If we set

$$\begin{aligned}\mathbf{V} &:= \boldsymbol{\Sigma}^{-1} + \boldsymbol{\Psi}^{-1}, \\ \mathbf{m} &:= \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \boldsymbol{\Psi}^{-1}\boldsymbol{\nu}, \\ R &:= \boldsymbol{\mu}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \boldsymbol{\nu}^\top \boldsymbol{\Psi}^{-1}\boldsymbol{\nu},\end{aligned} \tag{4.125}$$

and apply completing the square, then we can write the above as

$$\mathbf{x}^\top \mathbf{V}\mathbf{x} - 2\mathbf{m}^\top \mathbf{x} + R = (\mathbf{x} - \mathbf{V}^{-1}\mathbf{m})\mathbf{V}(\mathbf{x} - \mathbf{V}^{-1}\mathbf{m}) - \mathbf{m}^\top \mathbf{V}^{-1}\mathbf{m} + R. \tag{4.126}$$

This is proportional to a Gaussian kernel with mean $\mathbf{V}^{-1}\mathbf{m}$ and covariance $\mathbf{V}^{-1}$. We can ignore the remainder terms $\mathbf{m}^\top \mathbf{V}^{-1}\mathbf{m}$ and R since they do not depend on $\boldsymbol{\beta}$.

This is the trick used in the paper. Using the notation in the paper, both Gaussians are quadratic in $\boldsymbol{\beta}$; one has mean $\mathbf{b}$ and covariance $\mathbf{B}$, and the other has mean $\mathbf{X}^{-1}\mathbf{z}$ and covariance $\mathbf{X}^\top \boldsymbol{\Omega}^{-1}\mathbf{X}$. Doing a little pattern matching, we get

$$\begin{aligned}\mathbf{V} &:= (\mathbf{B}^{-1} + \mathbf{X}^\top \boldsymbol{\Omega}\mathbf{X})^{-1}, \\ \mathbf{m} &:= \mathbf{B}^{-1}\mathbf{b} + (\mathbf{X}^\top \boldsymbol{\Omega}\mathbf{X})\mathbf{X}^{-1}\mathbf{z}, \\ &= \mathbf{B}^{-1}\mathbf{b} + \mathbf{X}^\top \boldsymbol{\Omega}\mathbf{z}, \\ &\overset{\star}{=} \mathbf{B}^{-1}\mathbf{b} + \mathbf{X}^\top \boldsymbol{\kappa}.\end{aligned} \tag{4.127}$$

Step $\star$ holds because if we multiply each value in $\boldsymbol{\Omega}$ by the definition of $\mathbf{z}$, we get back $\boldsymbol{\kappa}$. Thus, we have shown that $P_{\boldsymbol{\beta}|y,\omega}$ is Gaussian with mean $\mathbf{V}^{-1}\mathbf{m}$ and covariance $\mathbf{V}^{-1}$.

4A.8 Marginal likelihood in Bayesian linear regression

For ease of notation, we drop the j subscript, and therefore $\mathbf{Y}$ is now $\mathbf{y} \coloneqq [y_1 \dots y_N]^\top$ and $\boldsymbol{\beta}_j$ is now $\boldsymbol{\beta}$. Consider the linear regression model

$$\mathbf{y} = \boldsymbol{\Phi}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \overset{\text{iid}}{\sim} \mathcal{N}_N(\mathbf{0}, \sigma^2 \mathbf{I}), \tag{4.128}$$

where $\boldsymbol{\Phi} \coloneqq [\varphi(\mathbf{x}_1) \dots \varphi(\mathbf{x}_N)]^\top$, an $N \times M$ matrix. A common conjugate prior on $\boldsymbol{\beta}$ is a normal–inverse–gamma distribution,

$$\begin{aligned} \boldsymbol{\beta} \mid \sigma^2 &\sim \mathcal{N}_M(\boldsymbol{\beta}_0, \sigma^2 \mathbf{S}_0^{-1}), \\ \sigma^2 &\sim \text{InvGamma}(a_0, b_0). \end{aligned} \tag{4.129}$$

We can write the functional form of the posterior and prior terms in Equation (4.129) as

$$\begin{aligned} p(\mathbf{y} \mid \boldsymbol{\Phi}, \boldsymbol{\beta}, \sigma^2) &= (2\pi\sigma^2)^{-N/2} \exp\left(-\frac{1}{2\sigma^2}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta})^\top(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta})\right), \\ p(\boldsymbol{\beta} \mid \sigma^2) &= (2\pi\sigma^2)^{-M/2} \left|\mathbf{S}_0\right|^{1/2} \exp\left(-\frac{1}{2\sigma^2}(\boldsymbol{\beta} - \boldsymbol{\beta}_0)^\top \mathbf{S}_0 (\boldsymbol{\beta} - \boldsymbol{\beta}_0)\right), \\ p(\sigma_\beta^2) &= \frac{b_0^{a_0}}{\Gamma(a_0)} (\sigma_\beta^2)^{-(a_0+1)} \exp\left(\frac{-b_0}{\sigma^2}\right). \end{aligned} \tag{4.130}$$

We can combine the likelihood's Gaussian kernel with the prior's kernel in the following way:

$$\begin{aligned} &(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta})^\top(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\beta}) + (\boldsymbol{\beta} - \boldsymbol{\beta}_0)^\top \mathbf{S}_0 (\boldsymbol{\beta} - \boldsymbol{\beta}_0) \\ &= \mathbf{y}^\top \mathbf{y} + \boldsymbol{\beta}_0^\top \mathbf{S}_0 \boldsymbol{\beta}_0 - \boldsymbol{\beta}_N^\top \mathbf{S}_N \boldsymbol{\beta}_N + (\boldsymbol{\beta} - \boldsymbol{\beta}_N)^\top \mathbf{S}_N (\boldsymbol{\beta} - \boldsymbol{\beta}_N). \end{aligned} \tag{4.131}$$

where $\boldsymbol{\beta}_N$ and $\mathbf{S}_N$ are defined as

$$\begin{aligned} \mathbf{S}_N &\coloneqq \boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \mathbf{S}_0, \\ \boldsymbol{\beta}_N &\coloneqq \mathbf{S}_N^{-1}(\boldsymbol{\beta}_0^\top \mathbf{S}_0 + \boldsymbol{\Phi}^\top \mathbf{y}). \end{aligned} \tag{4.132}$$

Now our posterior can be written as

$$\begin{aligned} p(\mathbf{y} \mid \mathbf{\Phi}, \boldsymbol{\beta}, \sigma^2) \propto (2\pi)^{-M/2} |\mathbf{S}_0|^{1/2} \exp\left(-\frac{1}{2\sigma^2}\left[(\boldsymbol{\beta} - \boldsymbol{\beta}_N)^\top \mathbf{S}_N (\boldsymbol{\beta} - \boldsymbol{\beta}_N)\right]\right) \\ (2\pi\sigma^2)^{-N/2} \exp\left(-\frac{1}{2\sigma^2}\left[\mathbf{y}^\top \mathbf{y} + \boldsymbol{\beta}_0^\top \mathbf{S}_0 \boldsymbol{\beta}_0 - \boldsymbol{\beta}_N^\top \mathbf{S}_N \boldsymbol{\beta}_N\right]\right) \\ \frac{b_0^{a_0}}{\Gamma(a_0)} (\sigma^2)^{-(a_0+1)} \exp\left(\frac{-b_0}{\sigma^2}\right). \end{aligned} \tag{4.133}$$

We can see that we have an M-variate normal distribution on the first line. If we ignore $(2\pi)^{-N/2}$ and inverse–gamma prior normalizer, we can combine the bottom two lines to be proportional to an inverse–gamma distribution,

$$(\sigma^2)^{-(a_0+N/2+1)} \exp\left(-\frac{1}{\sigma^2}\left[b_0 + \frac{1}{2}\left\{\mathbf{y}^\top \mathbf{y} + \boldsymbol{\beta}_0^\top \mathbf{S}_0 \boldsymbol{\beta}_0 - \boldsymbol{\beta}_N^\top \mathbf{S}_N \boldsymbol{\beta}_N\right\}\right]\right). \tag{4.134}$$

Now define a_N and b_N as

$$\begin{aligned} a_N &:= a_0 + \frac{N}{2} \\ b_N &:= b_0 + \frac{1}{2}(\mathbf{y}^\top \mathbf{y} + \boldsymbol{\beta}_0^\top \mathbf{S}_0 \boldsymbol{\beta}_0 - \boldsymbol{\beta}_N^\top \mathbf{S}_N \boldsymbol{\beta}_N). \end{aligned} \tag{4.135}$$

Then we can write our posterior as

$$p(\boldsymbol{\beta}, \sigma^2 \mid \mathbf{\Phi}, \mathbf{y}) \propto p(\boldsymbol{\beta} \mid \mathbf{\Phi}, \mathbf{y}_j, \sigma^2) p(\sigma^2 \mid \mathbf{\Phi}, \mathbf{y}), \tag{4.136}$$

where

$$\begin{aligned} \boldsymbol{\beta} \mid \mathbf{\Phi}, \mathbf{y}, \sigma^2 &\sim \mathcal{N}_M(\boldsymbol{\beta}_N, \mathbf{S}_N), \\ \sigma^2 \mid \mathbf{y}, \mathbf{\Phi} &\sim \text{InvGamma}(a_N, b_N). \end{aligned} \tag{4.137}$$

Now to compute the log marginal likelihood, we want

$$p(\mathbf{y} \mid \mathbf{\Phi}, a_0, b_0) = \iint p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\beta}, \sigma^2) p(\boldsymbol{\beta}), \sigma^2 \mid a_0, b_0)\, \mathrm{d}^M \boldsymbol{\beta}\, \mathrm{d}\sigma^2. \tag{4.138}$$

Using the definitions in Equation (4.132) and Equation (4.135), we can write the joint as

$$\begin{aligned} p(\mathbf{y}, \boldsymbol{\beta}, \sigma^2) = (2\pi\sigma^2)^{-P/2} \left|\mathbf{S}_0\right|^{1/2} & \exp\left(-\frac{1}{2\sigma^2}\left[(\boldsymbol{\beta} - \boldsymbol{\beta}_N)^\top \mathbf{S}_N (\boldsymbol{\beta} - \boldsymbol{\beta}_N)\right]\right) \\ & (\sigma^2)^{-(a_N+1)} \exp\left(-\frac{b_N}{\sigma^2}\right) \\ & (2\pi)^{-N/2} \frac{b_0^{a_0}}{\Gamma(a_0)}. \end{aligned} \tag{4.139}$$

The integral over $\boldsymbol{\beta}$ is only over the Gaussian kernel, which allows us to compute it immediately:

$$(2\pi\sigma^2)^{M/2} \left|\mathbf{S}_N\right|^{-1/2} = \int \exp\left(-\frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta}_N)^\top \left[\frac{1}{\sigma^2}\mathbf{S}_N\right](\boldsymbol{\beta} - \boldsymbol{\beta}_N)\right) \mathrm{d}^M \boldsymbol{\beta}. \tag{4.140}$$

The terms $(2\pi\sigma^2)^{M/2}$ in Equation (4.139) cancel, and the first line of Equation (4.139) reduces to

$$\sqrt{\frac{|\mathbf{S}_0|}{|\mathbf{S}_N|}}. \tag{4.141}$$

We can compute the second integral in Equation (4.138) because we know the normalizing constant of the gamma kernel,

$$\frac{\Gamma(a_N)}{b_N^{a_N}} = \int (\sigma^2)^{-(a_N+1)} \exp\left(-\frac{b_N}{\sigma^2}\right) \mathrm{d}\sigma^2. \tag{4.142}$$

Putting everything together, we see that the marginal likelihood's probability function is

$$p(\mathbf{y} \mid \boldsymbol{\Phi}, a_0, b_0) = \frac{1}{(2\pi)^{N/2}} \cdot \sqrt{\frac{|\mathbf{S}_0|}{|\mathbf{S}_N|}} \cdot \frac{b_0^{a_0}}{b_N^{a_N}} \cdot \frac{\Gamma(a_N)}{\Gamma(a_0)}. \tag{4.143}$$

4A.9 Negative binomial Gibbs sampler updates

4A.9.1. Sampling $\boldsymbol{\beta}_j$

Let ω be a Pólya–gamma distributed random variable with parameters $b > 0$ and $c \in \mathbb{R}$, denoted $\omega \sim \mathrm{PG}(b, c)$. Now consider an NB likelihood on $\mathbf{Y}$,

$$\mathcal{L}_N(\boldsymbol{\beta}) = \prod_{n=1}^{N} \prod_{j=1}^{J} \frac{(\exp\{\boldsymbol{\beta}_j^\top \varphi(\mathbf{x}_n)\})^{y_{nj}}}{(1 + \exp\{\boldsymbol{\beta}_j^\top \varphi(\mathbf{x}_n)\})^{y_{nj}+r_j}}. \tag{4.144}$$

Using Equation (4.101), we can express the nj-th term in the negative binomial likelihood using the following variable substitutions,

$$\begin{aligned} \psi &= \boldsymbol{\beta}_j^\top \varphi(\mathbf{x}_n), \quad & a &= y_{nj}, \\ b &= y_{nj} + r_j, \quad & \kappa &= \frac{y_{nj} - r_j}{2}. \end{aligned} \tag{4.145}$$

This gives us

$$\begin{aligned} &\frac{(\exp\{\boldsymbol{\beta}_j^\top \varphi(\mathbf{x}_n)\})^{y_{nj}}}{(1 + \exp\{\boldsymbol{\beta}_j^\top \varphi(\mathbf{x}_n)\})^{y_{nj}+r_j}} \\ &\propto \exp\left\{\frac{y_{nj} - r_j}{2} \boldsymbol{\beta}_j^\top \varphi(\mathbf{x}_n)\right\} \int_0^\infty \exp\left\{-\omega_{nj} \frac{(\boldsymbol{\beta}_j^\top \varphi(\mathbf{x}_n))^2}{2}\right\} p(\omega_{nj}) \mathrm{d}\omega_{nj} \\ &= \exp\left\{-\frac{\omega_{nj}}{2} \left(\boldsymbol{\beta}_j^\top \varphi(\mathbf{x}_n) - z_{nj}\right)^2\right\}, \end{aligned} \tag{4.146}$$

where

$$z_{nj} \coloneqq \frac{y_{nj} - r_j}{2\omega_{nj}}. \tag{4.147}$$

Finally, note that

$$\omega \mid \Psi \sim \mathrm{PG}(b, \Psi) \implies \omega_{nj} \mid \boldsymbol{\beta}_j \sim \mathrm{PG}\left(y_{nj} + r_j, \boldsymbol{\beta}_j^\top \varphi(\mathbf{x}_n)\right). \tag{4.148}$$

If we vectorize across N, we can sample each $\boldsymbol{\beta}_j$ following Polson et al. [2013]'s proposed Gibbs sampler:

$$\begin{aligned} \boldsymbol{\beta}_j \mid \boldsymbol{\omega}_j &\sim \mathcal{N}(\mathbf{m}_{\boldsymbol{\omega}_j}, \mathbf{V}_{\boldsymbol{\omega}_j}) \\ \boldsymbol{\omega}_j \mid \boldsymbol{\beta}_j &\sim \mathrm{PG}(\mathbf{y}_j + r_j, \boldsymbol{\Phi}\boldsymbol{\beta}_j) \end{aligned} \tag{4.149}$$

where

$$\begin{aligned} \boldsymbol{\Omega}_j &:= \mathrm{diag}([\boldsymbol{\omega}_{1j}, \dots, \boldsymbol{\omega}_{Nj}]), \\ \mathbf{V}_{\boldsymbol{\omega}_j} &:= (\boldsymbol{\Phi}^\top \boldsymbol{\Omega}_j \boldsymbol{\Phi} + \mathbf{B}_0^{-1})^{-1}, \\ \mathbf{m}_{\boldsymbol{\omega}_j} &:= \mathbf{V}_{\boldsymbol{\omega}_j}(\boldsymbol{\Phi}^\top \boldsymbol{\kappa}_j + \mathbf{B}_0^{-1}\boldsymbol{\beta}_0), \\ \boldsymbol{\kappa}_j &:= (\mathbf{y}_j - r_j)/2. \end{aligned} \tag{4.150}$$

4A.9.2. Sampling r_j

Consider the hierarchical model

$$\begin{aligned} y_{nj} &\sim \mathrm{NB}(r_j, p_{nj}), \\ r_j &\sim \mathrm{Gamma}(a_0, 1/h), \\ h &\sim \mathrm{Gamma}(b_0, 1/g_0). \end{aligned} \tag{4.151}$$

Zhou and Carin [2012] showed we can sample r as follows:

$$r_j \sim \mathrm{Gamma}\left(L_j, \frac{1}{-\sum_{n=1}^N \log(\max(1 - p_{nj}, -\infty))}\right). \tag{4.152}$$

where

$$L_j := \sum_{n=1}^N \sum_{t=1}^{\ell_j} u_{n\ell}, \qquad u_{n\ell} \sim \log(p_{nj}), \qquad \ell_j \sim \mathrm{Poisson}(-r_j \ln(1 - p_{nj})). \tag{4.153}$$

Zhou has released code.[14]

[14] https://mingyuanzhou.github.io/Softwares/LGNB_Regression_v0.zip

4A.10 Multinomial Gibbs sampler updates

In order to derive a Gibbs sampler for the multinomial likelihood, we first must use the reparameterization of the likelihood developed in Holmes et al. [2006]. We may rewrite the likelihood as

$$\begin{aligned}\mathcal{L}_N(\boldsymbol{\beta}) &= \prod_{n=1}^{N} \frac{\Gamma\left(\sum_{j=1}^{J} y_{nj}+1\right)}{\prod_{j=1}^{J}\Gamma\left(y_{nj}+1\right)} \prod_{j=1}^{J}\left(\frac{\exp\left\{\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j\right\}}{\sum_{j=1}^{J}\exp\left\{\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j\right\}}\right)^{y_{nj}} \\ &\propto \prod_{n=1}^{N}\prod_{j=1}^{J} \frac{\left(\exp\left\{\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j-\boldsymbol{\xi}_{nj}\right\}\right)^{y_{nj}}}{\left(1+\exp\left\{\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j-\boldsymbol{\xi}_{nj}\right\}\right)^{y_{nj}+\sum_{j=1}^{J} y_{nj}}},\end{aligned} \tag{4.154}$$

where $\boldsymbol{\xi}_{nj} = \log\sum_{j'\neq j}\exp\{\varphi(\mathbf{x}_n)\boldsymbol{\beta}_{j'}\}$. By convention and for identifiability purposes, we set $\boldsymbol{\beta}_J = 0$. We let $\kappa_{nj} = y_{nj} - \sum_{j=1}^{J} y_{nj}/2$. Now that we have written the likelihood in this form, we may use the Pólya–gamma augmentation trick again:

$$\begin{aligned}&\frac{\left(\exp\left\{\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j-\boldsymbol{\xi}_{nj}\right\}\right)^{y_{nj}}}{\left(1+\exp\left\{\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j-\boldsymbol{\xi}_{nj}\right\}\right)^{y_{nj}+\sum_{j=1}^{J} y_{nj}}} \\ &\propto \exp\left\{\kappa_{nj}\left(\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j-\boldsymbol{\xi}_{nj}\right) - \frac{\omega_{nj}}{2}\left(\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j-\boldsymbol{\xi}_{nj}\right)^2\right\}.\end{aligned} \tag{4.155}$$

So this gives us a posterior w.r.t. $\boldsymbol{\beta}_j$ as

$$\begin{aligned}&p(\boldsymbol{\beta}_j \mid \mathbf{y}_j, \mathbf{X}) \\ &\propto p(\boldsymbol{\beta}_j)\prod_{n=1}^{N}\exp\left\{\boldsymbol{\kappa}_n\left(\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j-\boldsymbol{\xi}_j\right) - \frac{1}{2}\left(\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j-\boldsymbol{\xi}_j\right)^T\boldsymbol{\Omega}_n\left(\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j-\boldsymbol{\xi}_j\right)\right\},\end{aligned} \tag{4.156}$$

which we can rewrite into a closed-form update as

$$\boldsymbol{\beta}_j \mid \boldsymbol{\omega}_j \sim \mathcal{N}(\mathbf{m}_{\boldsymbol{\omega}_j}, \mathbf{V}_{\boldsymbol{\omega}_j}), \tag{4.157}$$

where

$$\begin{aligned}\boldsymbol{\Omega}_j &:= \operatorname{diag}([\boldsymbol{\omega}_{1j}, \dots, \boldsymbol{\omega}_{Nj}]) \\ \mathbf{V}_{\boldsymbol{\omega}_j} &:= (\boldsymbol{\Phi}^\top \boldsymbol{\Omega}_j \boldsymbol{\Phi} + \mathbf{B}_0^{-1})^{-1}, \\ \mathbf{m}_{\boldsymbol{\omega}_j} &:= \mathbf{V}_{\boldsymbol{\omega}_j}(\boldsymbol{\Phi}^\top(\boldsymbol{\kappa}_j + \boldsymbol{\xi}_j^T \boldsymbol{\Omega}_j) + \mathbf{B}_0^{-1}\boldsymbol{\beta}_0), \\ \boldsymbol{\kappa}_j &:= \mathbf{y}_j - \frac{1}{2}\sum_{j=1}^{J} y_{nj}.\end{aligned} \tag{4.158}$$

And we sample $\boldsymbol{\Omega}_j$ with

$$\boldsymbol{\omega}_j \mid \boldsymbol{\beta}_j \sim \operatorname{PG}\left(\sum_{j=1}^{J} y_{nj}, \boldsymbol{\Phi}\boldsymbol{\beta}_j - \boldsymbol{\xi}_j\right). \tag{4.159}$$

Although we can sample the $\boldsymbol{\beta}_j$ parameters in closed form with the Pólya–gamma augmentation, we still face a problem with obtaining the MAP of $\mathbf{X}$ through optimization when we assume the likelihood is multinomial. Baker [1994] discusses the optimization problem of learning the MLE of the regression parameters in a multinomial logistic link regression problem. It is difficult to optimize parameters with respect to an objective function where the parameters are pushed through the normalization constant of a softmax function. To avoid this problem, we may write the nth contribution to the likelihood,

$$\mathcal{L}_n(\boldsymbol{\beta}) \propto \prod_{j=1}^{J}\left(\frac{\exp\{\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j\}}{\sum_{j=1}^{J}\exp\{\varphi(\mathbf{x}_n)\boldsymbol{\beta}_j\}}\right)^{y_{nj}}, \tag{4.160}$$

as a Poisson probability mass function with an additional N-dimensional nuisance parameter, $\mathbf{h}$, that we must learn through optimization,

$$\mathcal{L}_n(\boldsymbol{\beta}) \propto \prod_{j=1}^{J}(\exp\{\mathbf{h} + \varphi(\mathbf{x}_n)\boldsymbol{\beta}_j\})^{y_{nj}} \exp\{-\exp\{\mathbf{h} + \varphi(\mathbf{x}_n)\boldsymbol{\beta}_j\}\}, \tag{4.161}$$

where the MLE for the parameters in this Poisson reparamaterization are equal to the MLE learned in the original multinomial likelihood. In our implementation, we use this Poisson parameterization to learn the MAP of $\mathbf{X}$.

4A.11 Experiment details

4A.11.1. Data descriptions and preprocessing

- **Bridges:** We used the number of bicycle crossing per day over four East River bridges in New York City[15]. Since these data are unlabeled, we used weekday versus weekend as binary labels since such information is correlated with bicycle counts (Figure 4A.11.1, left).

- **CIFAR-10:** We limited the classes to $[1-5]$ and subampled 400 images for each class for a final data set of size 2000. We converted the images to grayscale and resized them from 32×32 down to 20×20 pixels.

- **Congress:** These data are word frequency counts from individual members of the 109th Congress from Gentzkow and Shapiro [2010]. Labels are political party: *Democrat*, *Independent*, *Republican*.

- **MNIST:** We limited the data set size by randomly subsampling 1000 images.

- **Montreal:** We use the number of cyclists per day on eight bicycle lanes in Montreal[16]. Since these data are unlabeled, we used the four seasons as labels, since seasonality is correlated with bicycle counts (Figure 4A.11.1, right).

- **Newsgroups:** The 20 Newsgroups data set[17]. We limited the classes to *sci.med*, *alt.atheism*, and *comp.sys.mac.hardware*, and limited the vocabulary to words with document frequencies in the range $10-90\%$.

- **Spam:** The SMS Spam data set from the UCI Machine Learning Repository[18]. Emails are labeled *spam* or *ham* (not spam).

[15] `https://data.cityofnewyork.us/Transportation/Bicycle-Counts-for-East-River-Bridges/gua4-p9wg`

[16] `http://donnees.ville.montreal.qc.ca/dataset/f170fecc-18db-44bc-b4fe-5b0b6d2c7297/resource/64c26fd3-0bdf-45f8-92c6-715a9c852a7b`

[17] `http://qwone.com/~jason/20Newsgroups/`

[18] `https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection`

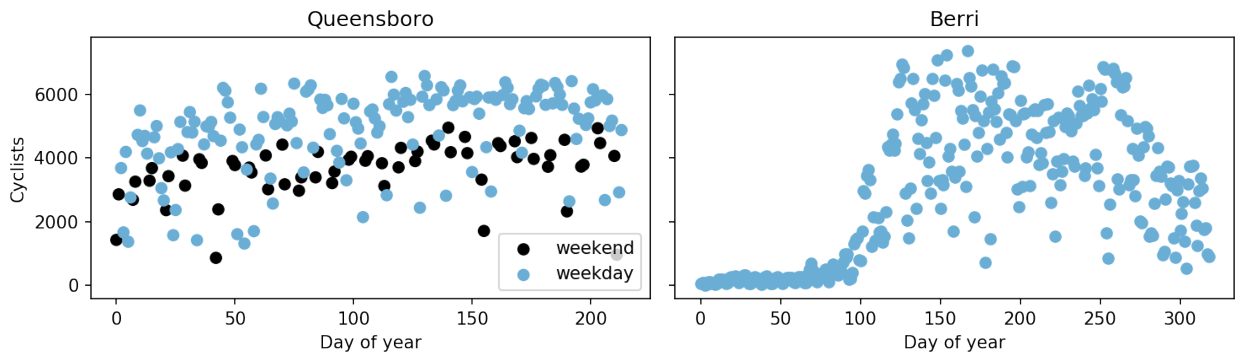

Figure 4A.11.1: (Left) The number of bicycle crossings over the Queensboro Bridge from April through November 2017. (Right) The number of cyclists on Berri St. in Montreal throughout 2015.

- **Yale:** The Yale Faces data set[19]. We used subject IDs as labels.

4A.11.2. Scalability experiments

To assess scalability of RFLVMs, we computed the wall-time in minutes required to fit both RFLVMs and the benchmarks (Table 4A.11.1). For both the VAE and deep count autoencoder, we trained the neural networks for 2000 iterations (default used in software package[20]). For DLA-GPLVM, we ran the optimizer for 50 iterations (default used in software package[21]). For RFLVMs, we ran the Gibbs samplers for 100 iterations. While results in Table 4.4.1 were run for 2000 Gibbs sampling iterations to ensure convergence for all data sets, we found empirically that reducing the number of iterations to 100 did not significantly change the results. We find that RFLVMs are indeed slower than most methods, but not significantly so. For example, on the CIFAR-10 data set, a VAE takes 23.7 minutes, while a Poisson RFLVM takes 22.9 minutes and a negative binomial RFLVM takes 55.7 minutes. The DLA-GPLVM is slowest, taking 69.8 minutes.

[19] `http://vision.ucsd.edu/content/yale-face-database`
[20] `https://github.com/theislab/dca`
[21] `https://github.com/waq1129/LMT`

Table 4A.11.1: Wall-time in minutes for model fitting. Mean and standard error were computed by running each experiment five times.

	PCA	NMF	HPF	LDA	VAE	DCA	NB-VAE	Isomap	DLA-GPLVM	Poisson RFLVM	Neg. binom. RFLVM	Multinomial RFLVM
Bridges	0.0186 ± 0.0005	0.0182 ± 0.0012	0.0273 ± 0.0002	0.0528 ± 0.0067	1.8193 ± 0.0708	0.5740 ± 0.0255	0.0867 ± 0.0157	0.0098 ± 0.0018	0.5182 ± 0.0206	0.3318 ± 0.0135	0.4915 ± 0.0502	0.5715 ± 0.0473
CIFAR-10	0.4398 ± 0.0743	0.4151 ± 0.0123	1.0894 ± 0.0500	0.8674 ± 0.0199	23.6770 ± 0.3789	1.1341 ± 0.0540	2.1002 ± 0.0594	0.4366 ± 0.0034	69.7889 ± 4.2406	22.9299 ± 1.2624	55.6701 ± 2.6837	59.8926 ± 9.9910
Congress	0.0244 ± 0.0002	0.0245 ± 0.0007	0.7296 ± 0.0824	0.0846 ± 0.0221	4.2919 ± 0.0539	0.5448 ± 0.0134	1.5898 ± 0.0725	0.0236 ± 0.0005	45.8584 ± 22.9771	9.8935 ± 0.1041	20.4514 ± 0.3995	94.0656 ± 2.7319
MNIST	0.2368 ± 0.0064	0.2522 ± 0.0273	1.0004 ± 0.1880	0.3264 ± 0.0237	15.3385 ± 1.8402	0.8719 ± 0.0618	2.1104 ± 0.1020	0.2148 ± 0.0019	26.4795 ± 1.5429	17.8148 ± 0.0493	33.8967 ± 4.1385	74.3100 ± 2.1778
Montreal	0.0171 ± 0.0008	0.0164 ± 0.0001	0.0523 ± 0.0350	0.0632 ± 0.0065	2.0585 ± 0.0947	0.5028 ± 0.0120	0.0819 ± 0.0009	0.0080 ± 0.0001	0.8723 ± 0.0237	0.5006 ± 0.0143	0.9291 ± 0.0434	0.8769 ± 0.0376
Newsgroups	0.0219 ± 0.0006	0.0227 ± 0.0000	0.1757 ± 0.0215	0.1163 ± 0.0344	6.8089 ± 0.7869	0.8551 ± 0.0527	0.7432 ± 0.0248	0.0721 ± 0.0008	1088.2659 ± 35.5089	2.6502 ± 0.4063	3.2600 ± 0.0892	2.8393 ± 0.1525
Spam	0.0230 ± 0.0004	0.0235 ± 0.0012	0.3039 ± 0.0419	0.1262 ± 0.0381	6.8448 ± 0.7796	0.7146 ± 0.0453	1.8411 ± 0.0283	0.0795 ± 0.0036	440.5963 ± 26.7444	10.6939 ± 0.4018	17.9958 ± 2.8573	19.0018 ± 2.4612
Yale	0.0884 ± 0.0003	0.0984 ± 0.0064	0.3774 ± 0.0181	0.1381 ± 0.0072	5.5177 ± 0.1645	0.6410 ± 0.0223	0.7931 ± 0.0589	0.0402 ± 0.0026	6.7210 ± 0.1193	9.8992 ± 0.5530	21.6030 ± 0.8839	45.4209 ± 4.4139

Chapter 5

Active multi-fidelity Bayesian online changepoint detection

Online algorithms for detecting changepoints, or abrupt shifts in the behavior of a time series, are often deployed with limited resources, e.g., to edge computing settings such as mobile phones or industrial sensors. In these scenarios, it may be beneficial to trade the cost of collecting an environmental measurement against the quality or *fidelity* of this measurement and how the measurement affects changepoint estimation. For instance, one might decide between inertial measurements or GPS to determine changepoints for motion. A Bayesian approach to changepoint detection [Adams and MacKay, 2007, Fearnhead and Liu, 2007] is particularly appealing because we can represent our posterior uncertainty about changepoints and make active, cost-sensitive decisions about data fidelity to reduce this posterior uncertainty. Moreover, the total cost could be dramatically lowered through active fidelity switching, while remaining robust to changes in data distribution. In this chapter, I present *active multi-fidelity Bayesian online changepoint detection* (MF-BOCD). MF-BOCD is a multi-fidelity approach that makes cost-sensitive decisions about which data fidelity to collect based on maximizing information gain with respect to changepoints. We evaluate this framework on synthetic, video, and audio data and show that this information-based approach results in accurate predictions while reducing total cost.

5.1 Introduction

Sequential data are rarely stationary. For example, a stock's volatility might increase or a text stream's topics might shift due to world events. A changepoint is an abrupt change in the generative parameters of sequential data. The goal of changepoint detection is to discover these structural changes, and thereby partition the data into regimes. Changepoint detection is a broad class of algorithms, including the classic CUSUM algorithm [Page, 1954], hidden Markov models with a changing transition matrix [Braun and Muller, 1998], Poisson processes with varying rates [Ritov et al., 2002], two-phase linear regression [Lund and Reeves, 2002], and Gaussian process changepoint models [Saatçi et al., 2010]. The Bayesian approach is appealing due to the ability to specify priors and represent posterior uncertainty [Chib, 1998, Fearnhead, 2006, Chopin, 2007]. For streaming applications, exact filtering algorithms allow for online Bayesian detection of changepoints without retrospective smoothing [Fearnhead and Liu, 2007, Adams and MacKay, 2007].

Many applications of online changepoint detection are in real-time settings with limited resources for sensing and computation, such as content delivery networks [Akhtar et al., 2018], autonomous vehicles [Ferguson et al., 2015], and smart home and internet-of-things devices [Aminikhanghahi et al., 2018, Lee et al., 2018, Munir et al., 2019]. In such resource-constrained settings, the observations for a changepoint detector are typically environmental measurements, for example heart-rate data [Villarroel et al., 2017]. Trading the cost of collecting these data against their quality or fidelity may be useful, depending on how these fidelities affect changepoint estimation.

For example, since scaling up neural network capacity is an effective approach to improving model performance [Arora et al., 2018, Kaplan et al., 2020, Mahajan et al., 2018], a high-fidelity observation model might be a large but expensive-to-evaluate neural network. Retraining a smaller architecture or using compression algorithms such as distillation [Hinton et al., 2015], quantization [Gong et al., 2014, Hubara et al., 2017], or pruning [Frankle and Carbin, 2018] could produce a low-fidelity observation model. If the output of these neural networks is the input to a changepoint detector, then the fidelity of the networks will impact the quality of changepoint detection.

In such situations, the cost of Bayesian online changepoint detection (BOCD) could be reduced by making decisions about the fidelity of the observations. One view of BOCD is as a model-based version of an exponentially-weighted moving average, estimating the weights from data rather than selecting them *a priori*. It determines which of the recent data matter for the current state. This view motivates our multi-fidelity approach: if changepoints are easily identified and the data can be partitioned into stationary regimes, there is no need for expensive high-fidelity observations when BOCD's posterior confidence about changepoints is high.

In our framing of the problem, we must choose which data fidelity to use and pay a fixed cost to make this choice. In the neural network example, we can evaluate either an expensive or cheap neural network to obtain a high- or low-fidelity representation of a raw measurement. To make this choice, we propose an information-theoretic approach, similar to the active data collection strategy proposed by MacKay [1992] and to approaches used in Bayesian optimization [Hernández-Lobato et al., 2014], preference learning [Houlsby et al., 2012], and Bayesian quadrature [Gessner et al., 2020]. We choose the data fidelity with maximal *information rate* (gain over cost) for the posterior distribution over changepoints. This results in policies that use lower-fidelity data in regimes with higher posterior certainty.

Contributions: First, we formulate a new version of an important problem: online changepoint detection with multiple data sources of varying cost and quality. The task is to choose which fidelity to use at each time point to make accurate predictions while minimizing costs. Second, we propose active selection of each datum's fidelity based on the expected informativeness of observations from each fidelity, and choose the one that maximizes the information rate for the posterior distribution over changepoints. Finally, we demonstrate the empirical performance of our algorithm on both synthetic and real-world data. We show that in many real-world scenarios, despite the extra step of computing information gain, our model reduces the total computational budget while maintaining good predictive accuracy.

5.2 Background

We begin by reviewing the BOCD algorithm [Adams and MacKay, 2007, Fearnhead and Liu, 2007]. Our data are a contiguous sequence of observations in time, $\mathbf{X}_{1:T} \coloneqq \{\mathbf{x}_1, \dots, \mathbf{x}_T\}$ where $\mathbf{x}_t \in \mathbb{R}^D$. Assume that the data can be partitioned such that, within each partition, the data are i.i.d. and governed by partition-specific parameters $\boldsymbol{\theta}$ [Barry and Hartigan, 1992]. The transition from one partition into another results in an abrupt change from one set of parameters to another. This transition is referred to as a *changepoint*.

Denote the parameters at time t as $\boldsymbol{\theta}_t$. In the changepoint process, these parameters are determined in one of two ways: either a changepoint has occurred at time t, in which case the parameters are drawn afresh from a prior distribution Π, or a changepoint has not occurred and the parameters are $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1}$, i.e., they stay the same. We model the arrival of changepoints as a discrete time Bernoulli process with hazard rate $1/\beta$, resulting in a geometric distribution over partition lengths with mean $\beta \in \mathbb{R}_{>0}$.

In the online setting, the primary quantity of interest is the time since the last changepoint, which we refer to as the *run length*. We denote the run length at time t as r_t, which takes values in the non-negative integers. Thus, a changepoint at t means $r_t = 0$. At time t, the BOCD algorithm estimates the posterior marginal distribution over the run length $p(r_t | \mathbf{X}_{1:t})$. We refer to this distribution as the *run-length posterior*. Online updating of the run-length posterior is made easy via a recursion that is essentially the same as the message-passing (dynamic programming) approach to hidden Markov models, particularly the forward algorithm [Baum and Petrie, 1966, Rabiner, 1989]. (See Section 5A.1 for a discussion of the forward–backward algorithm.) Here, the filtering recursion is:

$$\begin{aligned}
p(r_t \mid \mathbf{X}_{1:t}) &\propto p(r_t, \mathbf{X}_{1:t}) \\
&= \sum_{r_{t-1}} p(r_t, \mathbf{x}_t \mid r_{t-1}, \mathbf{X}_{1:t-1}) p(r_{t-1}, \mathbf{X}_{1:t-1}) \\
&= \sum_{r_{t-1}} p(r_t \mid r_{t-1}, \cancel{\mathbf{X}_{1:t-1}}) p(\mathbf{x}_t \mid r_t, \cancel{r_{t-1}}, \mathbf{X}_{1:t-1}) p(r_{t-1}, \mathbf{X}_{1:t-1}) \\
&= \sum_{r_{t-1}} \underbrace{p(r_t \mid r_{t-1})}_{\substack{\text{Bernoulli} \\ \text{process prior}}} \underbrace{p(\mathbf{x}_t \mid r_t, \mathbf{X}_{1:t-1})}_{\substack{\text{Posterior} \\ \text{predictive}}} \underbrace{p(r_{t-1}, \mathbf{X}_{1:t-1})}_{\substack{\text{Previous} \\ \text{estimate}}}, \qquad (5.1)
\end{aligned}$$

where the cancellations arise from Markovian assumptions we have made: 1) the probability of a changepoint at time t is independent of data before t, given knowledge of r_{t-1}, and 2) the predictive distribution over the data $\mathbf{x}_t$ at time t is independent of past run lengths, given knowledge of the current run length r_t. The three terms within the sum have a convenient interpretation as the prior, the predictive distribution, and the estimated joint distribution from the previous time step. These are the only ingredients necessary for a straightforward online filtering algorithm.

The Bernoulli process prior above is in an unconventional form that represents the time since the last changepoint:

$$p(r_t \mid r_{t-1}) = \begin{cases} 1/\beta & \text{if } r_t = 0, \\ 1 - 1/\beta & \text{if } r_t = r_{t-1} + 1, \\ 0 & \text{otherwise.} \end{cases} \qquad (5.2)$$

In other words, the run length r_t must either increase by one from the previous time point or drop to zero.

The construction so far has not depended on the specifics of the data-generating distribution $P_{\boldsymbol{\theta}_t}$, which appears as a part of the posterior predictive distribution in Equation (5.1):

$$p(\mathbf{x}_t \mid r_t = \ell, \mathbf{X}_{1:t-1}) = \int_{\boldsymbol{\Theta}} p_{\boldsymbol{\theta}_t}(\mathbf{x}_t) \, \pi(\boldsymbol{\theta}_t \mid \mathbf{X}^{(\ell)}) \, \mathrm{d}\boldsymbol{\theta}_t \,, \qquad (5.3)$$

where $p_{\boldsymbol{\theta}_t}(\cdot)$ is the probability density function associated with the distribution $P_{\boldsymbol{\theta}_t}$, $\pi(\boldsymbol{\theta} \mid \cdot)$ is the probability density function associated with the posterior distribution w.r.t. $\boldsymbol{\theta}$, and $\mathbf{X}^{(\ell)} := \mathbf{X}_{t-\ell:t-1}$ denotes the most recent ℓ data. This is a key property of the BOCD algorithm: conditioning on $r_t = \ell$ means that only the most recent ℓ data need to be accounted for in the posterior distribution. When the data distribution $P_{\boldsymbol{\theta}_t}$ is chosen to allow for a conjugate prior for Π, then the computations necessary for the recursion are relatively simple: it is only necessary to maintain a set of sufficient statistics for each r_t hypothesis. These statistics can be easily updated via addition, and the posterior predictive is often available in closed form. (See Adams and MacKay [2007] for further discussion.) When more complicated models are used, approximate inference or numerical integration are necessary.

Given the run-length posterior, we can compute a predictive distribution to make online predictions that are robust to changepoints by marginalizing out the run length, i.e., by computing a mixture of posterior predictive distributions—which are already available from the recursion—under the run-length posterior:

$$p(\mathbf{x}_{t+1} \mid \mathbf{X}_{1:t}) = \mathbb{E}_{p(r_t \mid \mathbf{X}_{1:t})}[p(\mathbf{x}_{t+1} \mid r_t = \ell, \mathbf{X}^{(\ell)})]\,. \tag{5.4}$$

Equation (5.4) underscores the value of modeling the run-length in this construction: it provides a model-based approach to decide which data are currently relevant for predicting the next observation. That is, the value of r_t explicitly captures the size of the current partition, i.e., what recent data share the same parameters.

The basic framework for BOCD has been extended in a number of ways, such as learning the changepoint prior [Wilson et al., 2010], adding Thompson sampling for multi-armed bandits with changing rewards [Mellor and Shapiro, 2013], estimating uncertainty bounds on the number and location of changepoints [Ruggieri and Antonellis, 2016], and using β-divergences for robustness against outliers [Knoblauch et al., 2018]. While changepoint detection has been explored in the context of active data selection [Osborne et al., 2010, Hayashi et al., 2019], to our knowledge, the BOCD framework has not been considered in multi-fidelity settings.

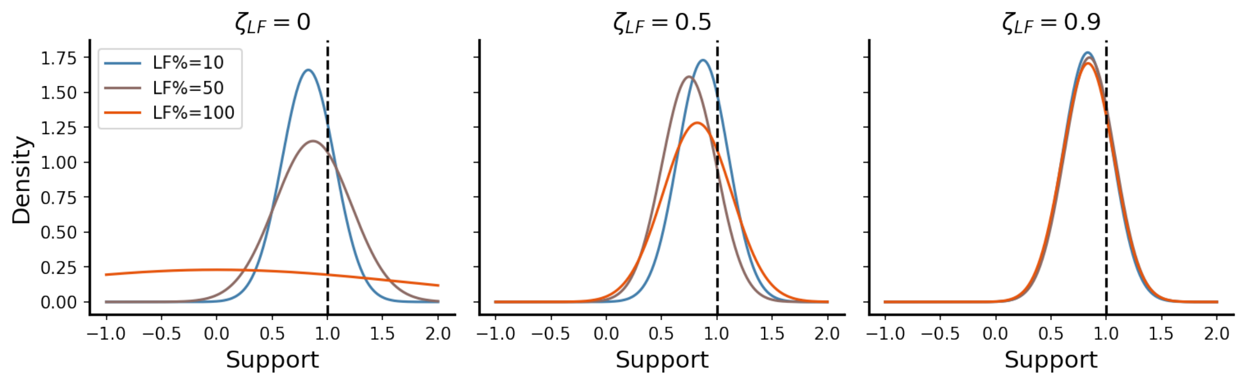

Figure 5.3.1: MF-posteriors $\pi(\theta_T \mid \mathbf{D}_{1:T})$ for varying low-fidelity weight $\zeta_{\mathsf{LF}} \in \{0, 0.5, 0.9\}$ but fixed high-fidelity weight $\zeta_{\mathsf{HF}} = 1$. The data are $T = 20$ i.i.d. samples $x_t \sim \mathcal{N}(1, 1)$. The prior is $\pi(\theta) = \mathcal{N}(0, 3)$. Within each panel, the percentage of (low-fidelity) weighted data likelihoods (LF%) varies. When $\zeta_{\mathsf{LF}} = 0$ and LF% $= 100$, (left panel, orange curve) the MF-posterior reduces to the prior $\pi(\theta)$. The MF-posterior becomes more concentrated when either ζ_{LF} increases (right two panels) or LF% decreases (blue curves).

5.3 Multi-fidelity changepoint detection

We now extend the BOCD framework to the multi-fidelity setting, referring to our algorithm as MF-BOCD. Our central assumption is that, at any time point t, we choose the quality of our observation, with higher fidelity (lower noise) having greater cost. We generally take this cost to be computational, but it could also be quantified in terms of resources such as money or energy. Given the selected data fidelities, we can again recursively compute a run-length posterior (Section 5.3.2). Given this multi-fidelity run-length posterior, the algorithm then selects the data fidelity that maximizes a cost-sensitive information rate objective (Section 5.3.4).

5.3.1. Multi-fidelity posterior predictive

Again, suppose we have a distribution $P_{\boldsymbol{\theta}_t}$ and prior Π, and the task is to estimate the parameter $\boldsymbol{\theta}_t$ in the presence of changepoints. Our data are again the contiguous sequence $\mathbf{X}_{1:T}$.

However, we now assume each observation $\mathbf{x}_t$ has an associated value $\zeta_t \in [0, 1]$, which we call the *fidelity*. The fidelities $\mathbf{z}_{1:T} := \{\zeta_1, \dots, \zeta_T\}$ are non-random and take values from a set $\mathcal{Z}$. In the experiments, we only consider the case when the cardinality of $\mathcal{Z}$ is two, i.e., we only have low- and high-fidelities, but this is not a necessary restriction. Let our sequence of observations and chosen fidelities be $\mathbf{D}_{1:T} \coloneqq \{(\mathbf{x}_1, \zeta_1), \dots, (\mathbf{x}_T, \zeta_T)\}$. The role of the fidelity ζ_t is to re-weight the associated probability function $p_{\boldsymbol{\theta}_t}(\mathbf{x})$ in a *multi-fidelity posterior* (MF-posterior). At time t, the MF-posterior is:

$$\pi(\boldsymbol{\theta}_t \mid \mathbf{D}_{1:t}) \propto \pi(\boldsymbol{\theta}_t) \prod_{i=1}^{t} p_{\boldsymbol{\theta}_t}(\mathbf{x}_i)^{\zeta_i}. \tag{5.5}$$

Here, $\pi(\cdot)$ is the probability density function associated with the prior distribution Π.

Intuitively, the effect of data re-weighting on the MF-posterior is a density that concentrates as if the contribution of T samples were $\sum_{t=1}^{T} \zeta_t$ number of data points instead of T data points. Figure 5.3.1 illustrates the MF-posterior of a conjugate Gaussian model with known variance (discussed in Section 5.3.3). Here the data are generated from a standard normal distribution, and the MF-posterior $\pi(\theta_T \mid \mathbf{D}_{1:T})$ is visualized for varying ζ_{LF} and fixed $\zeta_{\mathsf{HF}} = 1$. As ζ_{LF} decreases, the MF-posterior becomes less concentrated with a larger variance and increased influence from the prior.

Re-weighting terms in the likelihood has been considered under various names, such as safe Bayes [Heide et al., 2020, Grünwald et al., 2017], generalized posteriors [Walker and Hjort, 2001, Bissiri et al., 2016], coarsened posteriors [Miller and Dunson, 2018], and Bayesian data re-weighting [Wang et al., 2017]. In our framing of this model, we must choose each fidelity ζ_t of our observation $\mathbf{x}_t$, paying a fixed cost to make this choice.

When using a member of the exponential family with a conjugate prior, one has analytical expressions of the MF-posterior and MF-posterior predictive distributions. Let the distributions on $\mathbf{x}$ and $\boldsymbol{\theta}_t$ have the following functional forms:

$$p_{\boldsymbol{\theta}_t}(\mathbf{x}) = h_1(\mathbf{x}) \exp\left\{\boldsymbol{\theta}_t^\top u(\mathbf{x}) - a_1(\boldsymbol{\theta}_t)\right\}, \tag{5.6}$$

$$\pi_{\boldsymbol{\chi},\nu}(\boldsymbol{\theta}_t) = h_2(\boldsymbol{\theta}_t) \exp\left\{\boldsymbol{\theta}_t^\top \boldsymbol{\chi} - \nu a_1(\boldsymbol{\theta}_t) - a_2(\boldsymbol{\chi}, \nu)\right\}, \tag{5.7}$$

where, using exponential family terminology, $\boldsymbol{\theta}_t$ are now natural parameters, $u(\mathbf{x})$ are sufficient statistics, $a_1(\cdot)$ and $a_2(\cdot,\cdot)$ are log normalizers, and $h_1(\cdot)$ and $h_2(\cdot)$ are base measures.

Then the MF-posterior is

$$\pi_{\boldsymbol{\chi},\nu}(\boldsymbol{\theta}_t) \prod_{i=1}^{t} p_{\boldsymbol{\theta}_t}(\mathbf{x}_i)^{\zeta_i} \propto h_2(\boldsymbol{\theta}_t) \exp\left\{\boldsymbol{\theta}_t^\top \boldsymbol{\chi}_t - \nu_t a_1(\boldsymbol{\theta}_t)\right\}, \tag{5.8}$$

where $\boldsymbol{\chi}_t \coloneqq \boldsymbol{\chi} + \sum_{i=1}^{t} \zeta_i u(\mathbf{x}_i)$ and $\nu_t \coloneqq \nu + \sum_{i=1}^{t} \zeta_i$. The effect of the $\zeta_i < 1$ is to down-weight the sufficient statistics of $\mathbf{x}_i$. When $\zeta_i = 1$ for all i, Equation (5.8) reduces to the standard posterior for exponential family models.

We can now construct a multi-fidelity version of Equation (5.3): a posterior predictive distribution that depends on data fidelities. Let $\mathbf{D}^{(\ell)} \coloneqq \mathbf{D}_{t-\ell:t-1}$ denote the most recent ℓ data and associated fidelities (i.e., run length $r_t = \ell$), and let the associated parameter estimates be:

$$\boldsymbol{\chi}_\ell \coloneqq \boldsymbol{\chi} + \sum_{\tau=t-\ell}^{t-1} \zeta_\tau u(\mathbf{x}_\tau), \qquad \nu_\ell \coloneqq \nu + \sum_{\tau=t-\ell}^{t-1} \zeta_\tau. \tag{5.9}$$

Then the MF-posterior predictive is

$$\begin{aligned} p(\mathbf{x}_t \mid r_t = \ell, \zeta_t, \mathbf{D}^{(\ell)}) &= \int_{\boldsymbol{\Theta}} p_{\boldsymbol{\theta}_t}(\mathbf{x}_t)^{\zeta_t} \pi(\boldsymbol{\theta}_t \mid \mathbf{D}^{(\ell)}) \mathrm{d}\boldsymbol{\theta}_t \\ &= h_1(\mathbf{x}_t)^{\zeta_t} \frac{\exp(a_2(\zeta_t u(\mathbf{x}_t) + \boldsymbol{\chi}_\ell, \zeta_t + \nu_\ell))}{\exp(a_2(\boldsymbol{\chi}_\ell, \nu_\ell))}, \end{aligned} \tag{5.10}$$

provided $h_1(\mathbf{x}_i)^{\zeta_i}$ induces a distribution whose normalizer we can compute. See Section 5A.2 for a proof. Equation (5.10) can be interpreted as a traditional posterior predictive distribution for exponential family models but with the sufficient statistics weighted by the fidelities. Since BOCD is amenable to fast online updates for exponential families, inference using fidelities is often no harder than using the ordinary posterior.

Note that for some multi-fidelity models, the MF-posterior $p(\boldsymbol{\theta}_t \mid r_t = \ell, \mathbf{D}^{(\ell)})$ may not have an analytic form even when $p(\boldsymbol{\theta}_t \mid \mathbf{X}^{(\ell)})$ does. In this chapter, we only consider models in the exponential family, since this restriction often allows for efficient online updates. However, our approach may also extend to conditionally conjugate models (see Miller and Dunson [2018] for a discussion); in such settings, we could apply online variational inference to approximate predictive distributions [Turner et al., 2013]. As in standard BOCD, computing this predictive distribution without conjugate priors requires numerical approximations.

5.3.2. Multi-fidelity run-length posterior estimation

To accommodate multi-fidelity observations, we must modify the online posterior estimation procedure for the run lengths. We now condition the recursion on both the observations and data fidelities:

$$\begin{aligned} p(r_t = \ell \mid \mathbf{D}_{1:t}) &\propto p(r_t, \mathbf{X}_{1:t} \mid \mathbf{z}_{1:t}) \\ &= \sum_{r_{t-1}} p(r_t \mid r_{t-1}) p(\mathbf{x}_t \mid r_t, \zeta_t, \mathbf{D}^{(\ell)}) p(r_{t-1}, \mathbf{X}_{1:t-1} \mid \mathbf{z}_{1:t-1}). \end{aligned} \tag{5.11}$$

Similar to Equation (5.1), in the multi-fidelity case, the joint distribution of Equation (5.11) decomposes into a changepoint prior $p(r_t \mid r_{t-1})$, a predictive distribution, and the previous message. The latter two are now conditioned on fidelities. Thus, we can efficiently update the run length posterior in a recursive manner.

5.3.3. Examples

Before discussing how we choose fidelities, we demonstrate our approach with two examples of multi-fidelity models, which we use in Section 5.4. To simplify notation, we ignore the run length in this section, since it only specifies which data need to be accounted for in the MF-posterior distribution. See Section 5A.2 for more detailed derivations.

Multi-fidelity Gaussian. Consider a univariate Gaussian model[1] with known variance σ_x^2,

$$x_i \overset{\text{iid}}{\sim} \mathcal{N}(\theta_t, \sigma_x^2), \quad \theta_t \sim \mathcal{N}(\mu_0, \sigma_0^2). \tag{5.12}$$

The multi-fidelity likelihood is

$$\prod_{i=1}^{t} p_{\theta_t}(x_i)^{\zeta_i} \propto \prod_{i=1}^{t} \exp\left\{ -\frac{\zeta_i}{2\sigma_x^2}(x_i - \theta_t)^2 \right\}, \tag{5.13}$$

[1]It is straightforward to extend this result to the multivariate Gaussian.

and the MF-posterior is the product of $t+1$ independent Gaussian densities, which is again a Gaussian:

$$\begin{aligned}\pi(\theta_t \mid \mathbf{D}_{1:t}) &\propto \mathcal{N}(\theta_t \mid \mu_0, \sigma_0^2) \prod_{i=1}^{t} \mathcal{N}(x_i \mid \theta_t, \sigma_x^2/\zeta_i) \\ &\propto \mathcal{N}(\theta_t \mid \mu_t, \sigma_t^2),\end{aligned} \tag{5.14}$$

where

$$\frac{1}{\sigma_t^2} \coloneqq \frac{1}{\sigma_0^2} + \sum_{i=1}^{t} \frac{\zeta_i}{\sigma_x^2}, \qquad \mu_t \coloneqq \sigma_t^2 \left(\frac{\mu_0}{\sigma_0^2} + \sum_{i=1}^{t} \frac{\zeta_i x_i}{\sigma_x^2} \right). \tag{5.15}$$

The MF-posterior predictive distribution can be computed by integrating out θ_t. This is a convolution of two Gaussians—the posterior in Equation (5.14) and the prior $\pi(\theta_t)$—which is again Gaussian:

$$p(x_{t+1} \mid \zeta_{t+1}, \mathbf{D}_{1:t}) = \mathcal{N}\left(x_{t+1} \,\middle|\, \mu_t, \frac{\sigma_x^2}{\zeta_{t+1}} + \sigma_t^2 \right). \tag{5.16}$$

In this example, the fidelity ζ_i has the natural interpretation of increasing the posterior variance when $\zeta_i < 1$. In Equation (5.11), this has the effect that the multi-fidelity run length posterior is less concentrated. Any confidence in a changepoint is by definition lower.

Multi-fidelity Bernoulli. Consider a Bernoulli model,

$$x_i \overset{\text{iid}}{\sim} \text{Bernoulli}(\theta_t), \quad \theta_t \sim \text{Beta}(\alpha_0, \beta_0). \tag{5.17}$$

The MF-posterior is proportional to a beta distribution $\pi(\theta_t \mid \mathbf{D}_{1:t}) = \text{Beta}(\alpha_t, \beta_t)$ with parameters

$$\alpha_t \coloneqq \alpha_0 + \sum_{i=1}^{t} \zeta_i x_i, \quad \beta_t \coloneqq \beta_0 + \sum_{i=1}^{t} \zeta_i (1 - x_i). \tag{5.18}$$

The multi-fidelity posterior predictive distribution is the same as for a standard beta–Bernoulli model with α_t and β_t and additional re-weighting due to ζ_{t+1}:

$$p(x_{t+1} \mid \zeta_{t+1}, \mathbf{D}_{1:t}) = \frac{\mathrm{B}\left(\zeta_{t+1} x_{t+1} + \alpha_t, \zeta_{t+1}(1 - x_{t+1}) + \beta_t\right)}{\mathrm{B}(\alpha_t, \beta_t)}, \tag{5.19}$$

where $\mathrm{B}(\cdot, \cdot)$ is the beta function. When $\zeta_i < 1$, the fidelity has the natural effect of discounting count observations. (See Section 2.3.2 for a discussion of the single-fidelity beta–Bernoulli model.)

5.3.4. Active fidelity selection

So far, we have only discussed modeling data with multiple fidelities. However, in our framing of the problem, we must actively decide the fidelity of our observation $\mathbf{x}_t$, i.e., we must pick $\zeta_t \in \mathcal{Z}$. We propose maximizing the information rate of the multi-fidelity run length distribution. After observing $\mathbf{D}_{1:t-1}$ observations and fidelities, our current information about r_t is the Shannon entropy $\mathbb{H}[p(r_t \mid \mathbf{D}_{1:t-1})]$. Since we must choose a fidelity without observing $\mathbf{x}_t$, we want to choose the one that minimizes the expected entropy with respect to the predictive distribution in Equation (5.4). Thus, we choose the fidelity that maximizes the information gain of the run length posterior. The *utility* of ζ_t is therefore

$$\mathcal{U}(\zeta_t) = \mathbb{H}[r_t \mid \mathbf{D}_{1:t-1}] - \mathbb{E}_{\mathbf{x}_t}[\mathbb{H}[r_t \mid \mathbf{D}_{1:t-1}, \mathbf{x}_t, \zeta_t]]. \tag{5.20}$$

At time t, the left term in Equation (5.20) is easy to compute, since we have already computed the posterior distribution $p(r_{t-1} \mid \mathbf{D}_{1:t-1})$. We simply roll our estimation forward in time according to the changepoint process and without conditioning on new data. Furthermore, this value is the same for all fidelities, and therefore an equivalent formulation is to minimize the expected run length entropy, the right term in Equation (5.20). This entropy term is easy to compute because it is with respect to a discrete distribution that we can estimate at time t. The expectation is with respect to the predictive distribution (Equation (5.4)) and must be approximated in general.

However, we are not interested in the fidelity that just maximizes information gain regardless of cost. If this were the case, we would simply always use the highest fidelity. Let $\lambda(\zeta_t)$ denote the cost of fidelity ζ_t. In general, $\lambda(\cdot)$ could be a function of the input domain, but here we assume it is a scalar constant that is known, e.g., wall-time, energy usage, or floating point operations. Then the *information rate* of fidelity ζ_t at time t is $\alpha(\zeta_t) \coloneqq \mathcal{U}(\zeta_t)/\lambda(\zeta_t)$. However, given the interaction of fixed costs and estimated fidelities, it is possible that the maximum information rate is always achieved using the highest (or lowest) fidelity. In this case, we may still want some amount of low-fidelity (or high-fidelity) usage depending on data set size and computational budget. To address this, consider arbitrary weights $w(\zeta_t) \geq 0$. Our decision rule is then: use fidelity $\zeta_t^\star$ that maximizes the weighted information rate:

$$\zeta_t^\star \coloneqq \operatorname*{argmax}_{\zeta_t \in \mathcal{Z}} w(\zeta_t)\alpha(\zeta_t). \tag{5.21}$$

Note that the weights can be tuned on held-out data to achieve a desired expected budget. Introducing weights is useful because we do not lose $\lambda(\zeta_t)$, which may represent an interpretable quantity such as floating point operations.

5.3.5. Practical considerations

Analyzing costs. Since we are motivated by real-time decision-making, a sensible question is whether our decision-making algorithm is cheaper than using only high-fidelity observations. Here, we give a complete example of the cost for the beta–Bernoulli model. Since the predictive distribution is easy to work with, a useful reformulation of Equation (5.20) is

$$\mathcal{U}(\zeta_t) = \mathbb{H}[\mathbf{x}_t \mid \mathbf{D}_{1:t-1}] - \mathbb{E}_{r_t}[\mathbb{H}[\mathbf{x}_t \mid \mathbf{D}_{1:t-1}, r_t, \zeta_t]], \tag{5.22}$$

which uses the symmetry of information gain. (See Section 5A.3 for a proof.) At time t, the cost in floating point operations (flops) of computing Equation (5.22) is $32t + 1$ flops. The cost grows linearly with time because computing information gain requires summing over the run length posterior $p(r_t \mid \mathbf{D}_{1:t-1})$, and the support of this distribution grows linearly with time. However, Fearnhead and Liu [2007] proposed an optimal resampling algorithm, similar to particle filtering, that enables efficient approximate inference. This allows for a fixed cost to compute information gain. For example, with 10,000 particles, computing the information gain for the Bernoulli model requires 0.32 million flops. For comparison, consider MobileNets, which are a class of efficient neural networks designed for mobile and embedded vision applications [Howard et al., 2017]. The smallest reported MobileNet requires 41 million multi-adds (82 million flops). Thus, computing the beta–Bernoulli information gain twice (when the cardinality of $\mathcal{Z}$ is 2) is 140 times cheaper than evaluating the smallest MobileNet, while still using 10,000 particles in the run length posterior estimation.

Estimating fidelities in $\mathcal{Z}$. A second practical consideration is estimating the ζ values in $\mathcal{Z}$. In the Gaussian case with known variance σ_x^2, we can estimate ζ/σ_x^2 using the sample variance of held-out data and then calculate the value for ζ. In the Bernoulli case, we use model accuracy as a proxy for ζ. For example, if a binary classifier has a true positive rate of 90%, we treat an observation of 1 as a 0.9 using $\zeta = 0.9$.

5.4 Experiments

In this section, we empirically evaluate our algorithm on synthetic, video, and audio data, and compare performance of MF-BOCD against BOCD using only low- or high-fidelity data, as well as a randomized baseline. Please see Section 5A.4 for didactic code and the repository for a complete implementation.[2]

To evaluate our framework, we define two metrics. Let $\bar{\mathbf{X}}_{1:T} \coloneqq \{\bar{\mathbf{x}}_1, \dots, \bar{\mathbf{x}}_T\}$ denote the mean of the predictive distribution, Equation (5.4), of BOCD or MF-BOCD for all time points. Then the reported mean squared error (MSE) is between $\bar{\mathbf{X}}_{1:T}$ from the evaluated model and $\bar{\mathbf{X}}_{1:T}$ from BOCD using only high-fidelity data. Now let $\mathbf{R}_{1:T}$ denote a lower triangular matrix denoting the run length posterior at all time points. The L_1 distance is between $\mathbf{R}_{1:T}$ from the evaluated model and $\mathbf{R}_{1:T}$ from BOCD using only high-fidelity data. In other words, for both metrics, we compare the evaluated model to the best it could have done in practice. As a baseline, we compare MF-BOCD with a model that randomly switches between fidelities and which uses roughly the same percentage of high-fidelity data as MF-BOCD. This comparison isolates the question: is it *when* a multi-fidelity model uses high-fidelity data that improves performance or just the presence of high-fidelity data at all?

[2]https://github.com/gwgundersen/mf-bocd

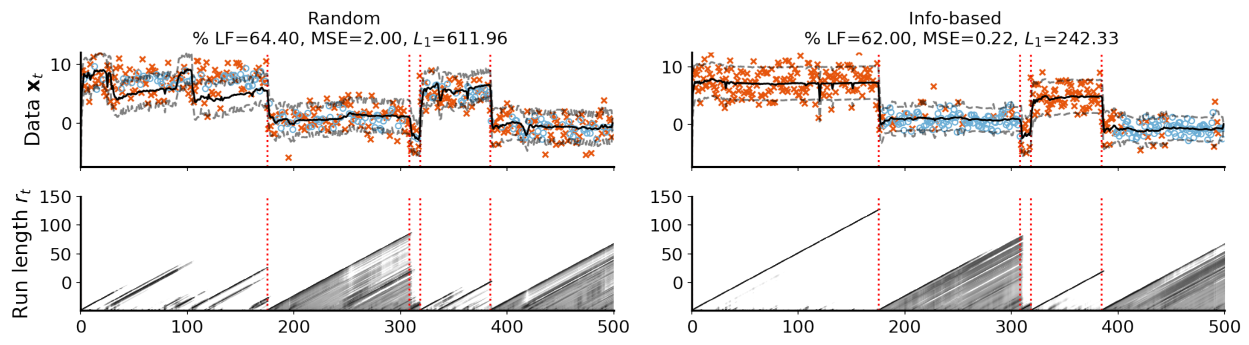

Figure 5.4.1: Comparison of two multi-fidelity models. Orange x marks and blue circles denote low- and high-fidelity data respectively. (Left two panels) A multi-fidelity model with random switching between fidelities. The probability of switching to low-fidelity data was chosen to be the fraction of low-fidelity observations used in the experiment in the right column. (Right two panels) A multi-fidelity model that actively selects the fidelity based on information rate.

5.4.1. Numerical experiments

The purpose of these experiments is to demonstrate that information rate is a useful decision rule and to build intuition about the model's behavior in a controlled setting. Consider a synthetic univariate signal with two fidelities. We assume data are i.i.d. Gaussian within each partition, and we use the Gaussian multi-fidelity model described in Section 5.3.3. When a changepoint occurs, the parameter θ_t is drawn from a prior $\mathcal{N}(1, 3)$. The data is then drawn from a distribution $x_t \sim \mathcal{N}(\theta_t, \zeta/\sigma_x^2)$ where $\sigma_x^2 = 1$. Our fidelities are from the set $\mathcal{Z} = \{\zeta_{\mathsf{HF}}, \zeta_{\mathsf{LF}}\}$. We set the higher fidelity to $\zeta_{\mathsf{HF}} = 1$ and the lower fidelity to $\zeta_{\mathsf{LF}} = 1/2$. Thus, low-fidelity data have twice the variance. Costs are arbitrary in this setting, and we set them to $\lambda(\zeta_{\mathsf{HF}}) = 2$ and $\lambda(\zeta_{\mathsf{LF}}) = 1$. We simulated the data using $T = 500$ observations with a changepoint prior with $1/\beta = 1/100$.

This experiment illustrates information rate as a decision rule as described in Section 5.3.4. In regions in which the model is confident about the run length posterior, low-fidelity data is preferred because both fidelities provide sufficient information. However, when the model is uncertain about the run length posterior, the high-fidelity observations are preferred (Figure 5.4.1). In contrast to information-based switching, the multi-fidelity model with random switching has both higher MSE and L_1 metrics. This suggests that while just using some high-fidelity data is useful, choosing when to use that high-fidelity data can improve performance. While this result is illustrative, we also include two randomized ablation experiments in Section 5A.5.

5.4.2. Cambridge video data

The numerical experiments provide a useful illustration of the role of information gain in a controlled setting. However, the fidelities and costs are contrived. In this section, we present a complete example of MF-BOCD with observation models and associated costs for the purpose of real-time detection of changepoints in streaming video data.

The Cambridge-driving Labeled Video Database (CamVid) is a collection of over ten minutes of video footage with object class semantic labels from 32 classes [Brostow et al., 2009]. The videos have been manually labeled at 1 frame per second, for just over 700 images. Each frame is 320×480 pixels. For observation models, we used pretrained V3 MobileNets [Howard et al., 2017, 2019]. The high-fidelity model is larger and more accurate (Table 5A.6.1).

The output of each observation model is a segmentation mask, which we converted to a binary signal depending on whether or not a given class is in the image. In particular, we used the "fence" signal because fences go in and out of the frame but typically remain in a sequence of frames for a brief period. We then fit the multi-fidelity Bernoulli model (Section 5.3.3) to the CamVid test set. We used the predictive version of information gain, Equation (5.22). We arbitrarily set the low-fidelity model's cost to 1 and the high-fidelity model's cost as function of that, $36.7/19.5 \approx 1.9$, using the number of flops (in billions) as a proxy for cost (Table 5A.6.1). The high-fidelity model used $\zeta_{\mathsf{HF}} = 1$. The low-fidelity model's fidelity is a function of the difference in mean intersection-over-union for each model, $\zeta_{\mathsf{LF}} = 1 - (0.723 - 0.674) \approx 0.95$.

We found that the output of low- and high-capacity neural networks were a reasonable proxy for low- and high-fidelity data. Standard BOCD using only high-fidelity observations estimates a run-length posterior that captures more groundtruth changepoints and has a predictive mean with smaller MSE and L_1 distance than BOCD using only low-fidelity data. The multi-fidelity model's decision-rule weights were tuned to approximate a total computational cost of 50% low-fidelity data using cross-validation data, and the randomized approach

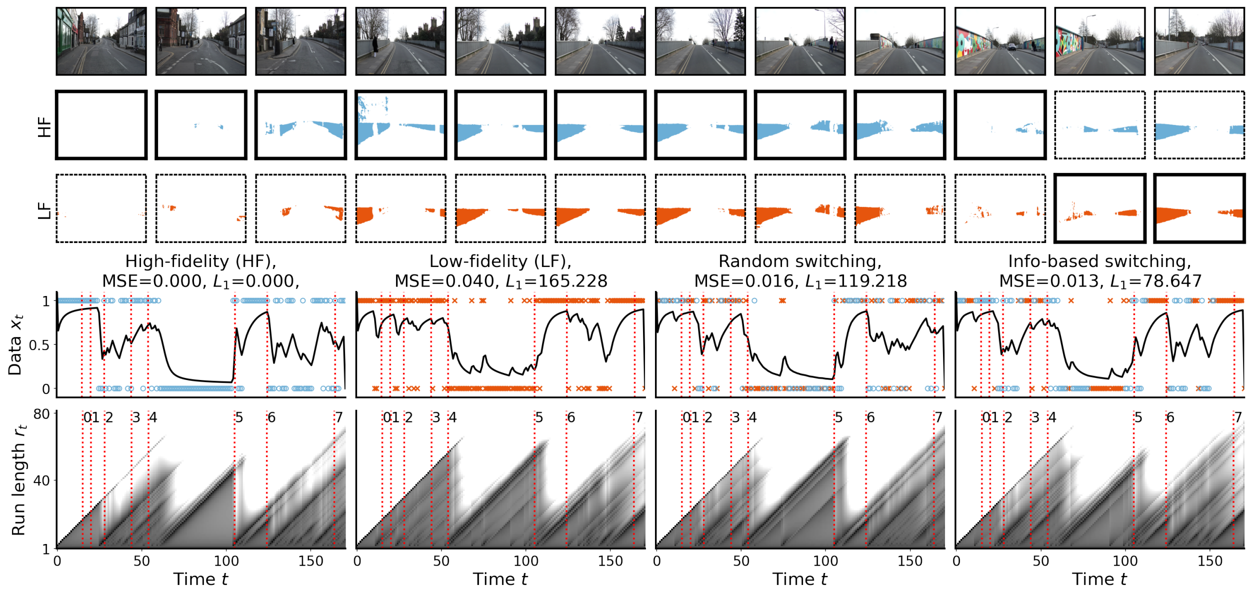

Figure 5.4.2: MF-BOCD on CamVid video stream. (Top three rows) A sequence of video frames as a camera-mounted vehicle approaches a bridge with fences on either side. The high- and low-fidelity masks are shown in middle and bottom rows respectively. A solid black frame indicates which fidelity was used by MF-BOCD. (Main center row) Binarized output from MobileNets for high-, low-, and multi-fidelity models. The solid black lines are predictive means. (Bottom row) Run length posteriors along with changepoints manually labeled from the groundtruth masks.

flips a fair coin to choose the data fidelity. On test data, MF-BOCD estimated a run length posterior that still closely matched the high-fidelity run-length posterior (Figure 5.4.2). The information-based approach results in a better predictive mean (MSE) and better run length posterior estimation (L_1 distance) than both the low-fidelity and randomized versions.

Finally, we estimated the computational cost of MF-BOCD relative to baselines. With roughly 50% low-fidelity data, the costs in billions of flops for MF-BOCD was 4827, for BOCD using just low-fidelity data was 3333, and for BOCD using just high-fidelity data was 6303. The cost of decision-making was marginal, requiring 0.00046 billion flops. (See-Section 5A.6 for details on these calculations.) As these calculations demonstrate, making a decision between high- and low-capacity neural networks can be significantly cheaper than evaluating either model. So while random usage of low-fidelity data is a reasonable approach to lowering the computational budget, decision-making can improve inference and predictions with marginal added cost.

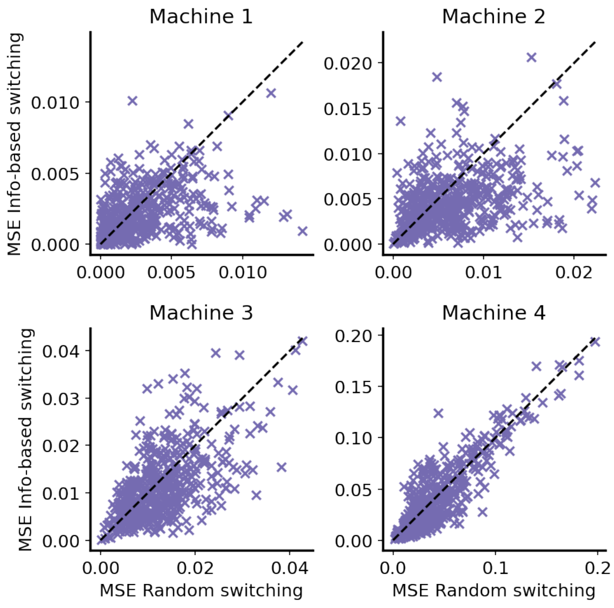

Figure 5.4.3: Comparison of information-based versus random switching on the MIMII data set. Under the line is better for MF-BOCD. See Table 5.4.1 for means and standard errors.

5.4.3. MIMII audio data

Next, we evaluated MF-BOCD on the sound data set for Malfunctioning Industrial Machine Investigation and Inspection (MIMII) [Purohit et al., 2019]. The raw data are 10-second audio clips recorded from 4 different industrial machines (slide rails in this experiment) during either normal or anomalous operation. For example, anomalous conditions might involve rail damage, a loose belt, or no grease. The high-fidelity observation model is a depth-wise separable convolutional neural network (MicroNets) [Banbury et al., 2020]. The low-fidelity observation model is a two-layer fully-connected neural network. Both models take frames of log-Mel spectrograms of audio signals as inputs and return an anomaly score as output. They were pretrained on audio clips of normal behavior. Then each 10-second test set clip was converted to 14 anomaly scores using these observation models. The anomaly score is a number between 0 and 1, with 0 indicating normal. We thresholded the anomaly scores to produce binary labels. We picked machine- and model-specific thesholds using ROC curves. (See Section 5A.6 for details.)

To randomly generate audio files with changepoints, we sampled a sequence of Bernoulli random variables $\mathbf{y}_{1:T}$. Then for each y_t, we chose a normal (anomalous) audio clip uniformly at random with replacement if $y_t = 0$ ($y_t = 1$). We converted clips to low- (high-) fidelity data by evaluating the low- (high-) neural network and computing the median anomaly score for that clip. As in Section 5.4.2, we used a Bernoulli model with $\zeta_{\mathsf{HF}} = 1$ and ζ_{LF} set to the low-fidelity model's true positive rate relative to the high-fidelity model. For each machine, we randomly generated 500 data sets with changepoints and computed the MSE and L_1 distances for low-fidelity BOCD and for MF-BOCD with both random and information-based switching. We found that the information-based approach to switching had lower MSE and L_1 distance than BOCD using just low-fidelity data and had better performance than randomized switching on the first three machines (Table 5.4.1). An interesting negative result is that MF-BOCD does not do significantly better than random on machine 4. We hypothesize that this is due to the poor quality of the low-fidelity observation model, which has an AUC < 0.5 (Figure 5A.6.3). With these data, MF-BOCD is making hard decisions (argmax) with bad information. And in general, a randomized approach can sometimes do well (Figure 5.4.3). An interesting direction for future work would be to soften the decision rule via sampling, perhaps controlled by a temperature.

As in the CamVid experiments, we found that the total cost of decision-making was marginal; the neural network costs dominated the calculations (Table 5.4.1). Thus, MF-BOCD offers a useful way to trade off detection accuracy for computational savings.

5.5 Discussion

We have extended Bayesian online changepoint detection to the multi-fidelity setting in which observations have associated fidelities and costs. We found that choosing the data fidelity based on maximal information rate with respect to the run-length posterior yields interpretable policies that lower computational costs while still maintaining good performance in terms of parameter and run-length posterior estimation. In simple models, decision-making is cheap relative to the cost of evaluating even tiny neural networks designed for commodity microcontrollers. Savings in number of operations can be translated to energy savings [Banbury et al., 2020], which is crucial for resource-constrained applications.

While we focus on the online and resource-constrained setting, this framework could be

Table 5.4.1: Comparison between low-fidelity BOCD (LF), random switching (RN), and MF-BOCD (IG). Mean and two standard errors were computed over 500 randomly generated MIMII data sets with changepoints, using the method described in the text. Cost is in millions of flops. Bold numbers indicate statistically significant using 95% confidence intervals. Reported %LF is the average across all data sets.

		Machine 1	Machine 2	Machine 3	Machine 4
MSE	LF	0.0060 (0.0004)	0.0195 (0.0008)	0.0347 (0.0012)	0.1743 (0.0042)
	RN	0.0026 (0.0002)	0.0063 (0.0004)	0.0126 (0.0006)	0.0411 (0.0028)
	IG	**0.0020** (0.0002)	**0.0045** (0.0003)	**0.0112** (0.0006)	0.0393 (0.0030)
L_1	LF	101.87 (3.28)	167.73 (3.61)	192.49 (4.02)	242.85 (4.63)
	RN	57.61 (3.14)	97.79 (3.66)	132.06 (3.63)	178.86 (4.97)
	IG	61.79 (3.02)	92.98 (3.65)	130.17 (3.88)	173.27 (4.95)
Ops	LF	100	”	”	”
	RN	14447.58	16109.38	12867.76	13357.11
	IG	14448.22	16110.02	12868.40	13357.74
	HF	24940	”	”	”
%LF		42	36	48	46

extended to scenarios in which observations take a long time to compute, such as changepoint detection in protein-folding [Fan et al., 2015] or engineering design [Robinson et al., 2008]. In such settings, expensive approximations of the posterior predictive distribution or information gain may be tolerable, as well as retrospective smoothing of the run-length distributions.

Alternative decision rules should also be explored, as these will induce different policies. Gessner et al. [2020] discuss how any monotonic transformation of Equation (5.21) gives rise to the same policy because the global maximum is the same even if the value at that maximum is not. However, this is not necessarily true after dividing the decision rule by a cost. Furthermore, a probabilistic decision-rule might be useful in scenarios where the difference between low- and high-fidelity observation models is marginal.

5A Appendix

5A.1 Forward–backward algorithm

The Baum–Welch algorithm [Baum and Petrie, 1966] is a special case of EM (Section 2.2.1) for hidden Markov models (HMMs). Baum–Welch uses the forward–backward algorithm in the E-step. The goal of this section is to derive just the forward–backward algorithm, as it is quite similar to the recursive algorithm used in BOCD (Equation (5.1)). For brevity, I assume the reader is familiar with HMMs. Please see Bishop [2006] for a detailed description.

To introduce notation, let $\mathbf{X} := \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ be N sequential observations where $\mathbf{x}_n \in \mathbb{R}^D$ and let $\mathbf{Z} := \{\mathbf{z}_1, \dots, \mathbf{z}_N\}$ denote the latent variables. Finally, let $\boldsymbol{\theta}^{(t)}$ denote the HMM parameters (initial state probabilities, transition probabilities, and emission probabilities) at EM iteration t. See Figure 5A.1.1 for the standard graphical model for an HMM.

The forward–backward algorithm computes the probabilities $p(\mathbf{z}_n \mid \mathbf{X}, \boldsymbol{\theta}^{(t)})$ and $p(\mathbf{z}_{n-1}, \mathbf{z}_n \mid \mathbf{X}, \boldsymbol{\theta}^{(t)})$, which are required by the posterior moments in the expected complete log likelihood:

$$
\begin{aligned}
&\mathbb{E}_{p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}^{(t)})}\left[\log p(\mathbf{Z}, \mathbf{X} \mid \boldsymbol{\theta}^{(t)})\right] \\
&= \mathbb{E}\left[\log p(\mathbf{z}_1)\right] + \mathbb{E}\left[\sum_{n=2}^{N} \log p(\mathbf{z}_n \mid \mathbf{z}_{n-1})\right] + \mathbb{E}\left[\sum_{n=1}^{N} \log p(\mathbf{x}_n \mid \mathbf{z}_n)\right].
\end{aligned}
\tag{5.23}
$$

(For the remainder of the section, let's ignore $\boldsymbol{\theta}^{(t)}$ since it is not necessary to understand the main idea.) More specifically, the algorithm computes these two terms,

$$
\alpha(\mathbf{z}_n) := p(\mathbf{z}_n, \mathbf{X}_{1:n}) \qquad \text{(forward pass),} \tag{5.24}
$$

$$
\beta(\mathbf{z}_n) := p(\mathbf{X}_{n+1:N} \mid \mathbf{z}_n) \qquad \text{(backward pass),} \tag{5.25}
$$

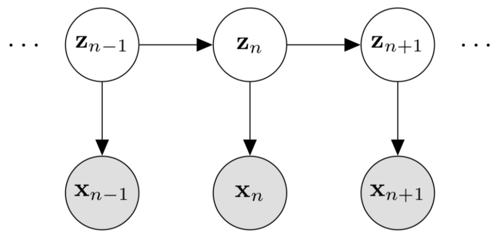

Figure 5A.1.1: Graphical model for a hidden Markov model. The hidden state variables $\mathbf{z}_1, \dots, \mathbf{z}_N$ follow a Markov process, while each observation $\mathbf{x}_n$ is conditionally independent from other data given its associated hidden state variable $\mathbf{z}_n$.

because the posterior mean and posterior second moment can be written using them. The first moment requires

$$\begin{aligned} p(\mathbf{z}_n \mid \mathbf{X}) &= \frac{p(\mathbf{X}, \mathbf{z}_n)}{p(\mathbf{X})} \\ &= \frac{p(\mathbf{X}_{1:n}, \mathbf{X}_{n+1:N}, \mathbf{z}_n)}{p(\mathbf{X})} \\ &\overset{\star}{=} \frac{p(\mathbf{X}_{n+1:N} \mid \mathbf{z}_n) p(\mathbf{z}_n, \mathbf{X}_{1:n})}{p(\mathbf{X})} \\ &= \frac{\alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)}{p(\mathbf{X})}. \end{aligned} \tag{5.26}$$

And the second moment requires

$$\begin{aligned} p(\mathbf{z}_{n-1}, \mathbf{z}_n \mid \mathbf{X}) &= \frac{p(\mathbf{X} \mid \mathbf{z}_{n-1}, \mathbf{z}_n) p(\mathbf{z}_{n-1}, \mathbf{z}_n)}{p(\mathbf{X})} \\ &= \frac{p(\mathbf{X} \mid \mathbf{z}_{n-1}, \mathbf{z}_n) p(\mathbf{z}_{n-1}, \mathbf{z}_n)}{p(\mathbf{X})} \\ &\overset{\star}{=} \frac{p(\mathbf{x}_{1:n-1} \mid \mathbf{z}_{n-1}) p(\mathbf{x}_n \mid \mathbf{z}_n) p(\mathbf{x}_{n+1:N} \mid \mathbf{z}_n) p(\mathbf{z}_n \mid \mathbf{z}_{n-1}) p(\mathbf{z}_{n-1})}{p(\mathbf{X})} \\ &= \frac{\alpha(\mathbf{z}_{n-1})\beta(\mathbf{z}_n) p(\mathbf{x}_n \mid \mathbf{z}_n) p(\mathbf{z}_n \mid \mathbf{z}_{n-1})}{p(\mathbf{X})}. \end{aligned} \tag{5.27}$$

In the steps labeled $\star$, we apply our modeling assumptions: that $\mathbf{Z}$ are Markov and that future observations only depend on the current latent variable (Figure 5A.1.1). In the language of HMMs, the forward–backward algorithm does both *filtering* (Figure 5A.1.2) and *smoothing* (Figure 5A.1.3). Computing $\alpha(\mathbf{z}_n)$ is effectively filtering, and computing the first posterior

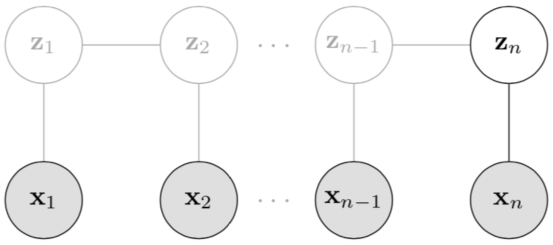

Figure 5A.1.2: Filtering or posterior inference of $\mathbf{z}_n$ given all the observations up until that time point, $\mathbf{x}_1, \dots, \mathbf{x}_n$.

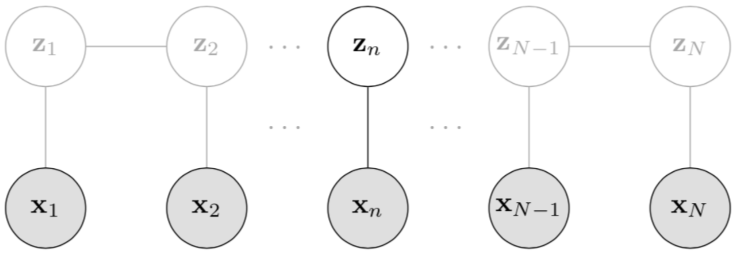

Figure 5A.1.3: Smoothing or posterior inference of $\mathbf{z}_n$ given all the observations $\mathbf{x}_1, \dots, \mathbf{x}_N$.

moment is smoothing:

$$\overbrace{p(\mathbf{z}_n \mid \mathbf{X}_{1:n})}^{\text{Filtering}} \propto p(\mathbf{z}_n, \mathbf{X}_{1:n}) = \alpha(\mathbf{z}_n), \tag{5.28}$$

$$\overbrace{p(\mathbf{z}_n \mid \mathbf{X})}^{\text{Smoothing}} \propto p(\mathbf{X}_{n+1:N} \mid \mathbf{z}_n) p(\mathbf{z}_n, \mathbf{X}_{1:n}) = \alpha(\mathbf{z}_n)\beta(\mathbf{z}_n). \tag{5.29}$$

I think it's fair to think of computing the posterior second moment as a kind of smoothing as well. Now let's look at the two passes.

5A.1.1. Forward pass

The main idea of the forward pass is to marginalize over the previous latent variable to develop a recursive message-passing algorithm. This will allow us to use dynamic programming for efficient computation:

$$\begin{aligned}
\alpha(\mathbf{z}_n) &= p(\mathbf{z}_n, \mathbf{X}_{1:n}) \\
&= \sum_{\mathbf{z}_{n-1}} p(\mathbf{z}_n, \mathbf{z}_{n-1}, \mathbf{X}_{1:n}) \\
&= \sum_{\mathbf{z}_{n-1}} p(\mathbf{z}_n, \mathbf{z}_{n-1}, \mathbf{x}_n, \mathbf{X}_{1:n-1}) \\
&= \sum_{\mathbf{z}_{n-1}} p(\mathbf{x}_n \mid \mathbf{z}_n, \cancel{\mathbf{z}_{n-1}}, \cancel{\mathbf{X}_{1:n-1}}) p(\mathbf{z}_n, \mathbf{z}_{n-1}, \mathbf{X}_{1:n-1}) \\
&= \sum_{\mathbf{z}_{n-1}} p(\mathbf{x}_n \mid \mathbf{z}_n) p(\mathbf{z}_n \mid \mathbf{z}_{n-1}, \cancel{\mathbf{X}_{1:n-1}}) p(\mathbf{z}_{n-1}, \mathbf{X}_{1:n-1}) \\
&= p(\mathbf{x}_n \mid \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} p(\mathbf{z}_n \mid \mathbf{z}_{n-1}) \alpha(\mathbf{z}_{n-1}).
\end{aligned} \tag{5.30}$$

Terms cancel due to modeling assumptions. Look at the graphical model (Figure 5A.1.1) for the conditional dependence structure.

5A.1.2. Backward pass

The backward pass is a similar idea, but rather than marginalizing over the previous hidden state, we marginalize over the next hidden state:

$$\begin{aligned}
\beta(\mathbf{z}_n) &= p(\mathbf{X}_{n+1:N} \mid \mathbf{z}_n) \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{z}_{n+1}, \mathbf{X}_{n+1:N} \mid \mathbf{z}_n) \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{z}_{n+1}, \mathbf{x}_{n+1}, \mathbf{x}_{n+2:N} \mid \mathbf{z}_n) \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+2:N} \mid \mathbf{z}_{n+1}, \cancel{\mathbf{x}_{n+1}}, \cancel{\mathbf{z}_n}) p(\mathbf{z}_{n+1}, \mathbf{x}_{n+1} \mid \mathbf{z}_n) \\
&= \sum_{\mathbf{z}_{n+1}} p(\mathbf{x}_{n+2:N} \mid \mathbf{z}_{n+1}) p(\mathbf{x}_{n+1} \mid \mathbf{z}_{n+1}, \cancel{\mathbf{z}_n}) p(\mathbf{z}_{n+1} \mid \mathbf{z}_n) \\
&= \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1} \mid \mathbf{z}_{n+1}, \cancel{\mathbf{z}_n}) p(\mathbf{z}_{n+1} \mid \mathbf{z}_n).
\end{aligned} \tag{5.31}$$

Once again, this is a recursive algorithm in which we can message pass a previously computed quantity backward.

5A.1.3. Initial conditions and evidence

Finally, we need to initialize our recursive algorithm. Since we compute the $\alpha(\mathbf{z})$ terms in a forward filtering pass, we need a value for $\alpha(\mathbf{z}_1)$. This is easy to compute:

$$\alpha(\mathbf{z}_1) = p(\mathbf{x}_1, \mathbf{z}_1) = p(\mathbf{x}_1 \mid \mathbf{z}_1)p(\mathbf{z}_1) = \prod_{k=1}^{K} \{\pi_k p_{\boldsymbol{\phi}_k}(\mathbf{x}_1)\}^{z_{1k}}, \tag{5.32}$$

where π_k is the initial probability of being on state k, and $p_{\boldsymbol{\phi}_k}(\cdot)$ is the probability function (emissions probability) for the kth distribution with parameters $\boldsymbol{\phi}_k$. In words, this is the probability of $\mathbf{x}_1$ for each state, weighted by the initial probability of that state.

However, what are the initial conditions for $\beta(\mathbf{z}_N)$? Recall that the recursion starts at the last observation since we compute the $\beta(\mathbf{z})$ terms in reverse. Now notice that if we set $n = N$ and apply the definition of $\alpha(\mathbf{z})$ in Equation (5.26), we have

$$p(\mathbf{z}_N \mid \mathbf{X}) = \frac{p(\mathbf{z}_N, \mathbf{X})\beta(\mathbf{z}_N)}{p(\mathbf{X})}. \tag{5.33}$$

Thus, it's clear that $\beta(\mathbf{z}_N) = 1$ for each state k; otherwise, we would not have properly normalized distributions.

Finally, we can compute the evidence by summing over both sides of Equation (5.26):

$$\begin{aligned}
p(\mathbf{z}_n \mid \mathbf{X}) &= \frac{\alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)}{p(\mathbf{X})} \\
\sum_{k=1}^{K} p(\mathbf{z}_n = k \mid \mathbf{X}) &= \sum_{k=1}^{K} \frac{\alpha(\mathbf{z}_n = k)\beta(\mathbf{z}_n = k)}{p(\mathbf{X})} \\
1 &= \sum_{k=1}^{K} \frac{\alpha(\mathbf{z}_n = k)\beta(\mathbf{z}_n = k)}{p(\mathbf{X})} \\
p(\mathbf{X}) &= \sum_{k=1}^{K} \alpha(\mathbf{z}_n = k)\beta(\mathbf{z}_n = k).
\end{aligned} \tag{5.34}$$

5A.2 Model derivations

5A.2.1. MF-posterior predictive for exponential family models

In multi-fidelity BOCD, we desire the posterior predictive distribution conditioned on the run length,

$$p(\mathbf{x}_t \mid r_t = \ell, \zeta_t, \mathbf{D}_{t-\ell:t-1}). \tag{5.35}$$

Assume this is an exponential family model with the following likelihood and prior density functions:

$$p_{\boldsymbol{\theta}_t}(\mathbf{x}) = h_1(\mathbf{x}) \exp\left\{\boldsymbol{\theta}_t^\top u(\mathbf{x}) - a_1(\boldsymbol{\theta}_t)\right\}, \tag{5.36}$$

$$\pi_{\boldsymbol{\chi},\nu}(\boldsymbol{\theta}_t) = h_2(\boldsymbol{\theta}_t) \exp\left\{\boldsymbol{\theta}_t^\top \boldsymbol{\chi} - \nu a_1(\boldsymbol{\theta}_t) - a_2(\boldsymbol{\chi}, \nu)\right\}. \tag{5.37}$$

See Section 5.3 or Equation (5.7) for a description of these terms. We introduce the following notation to denote the data and parameter estimates for the previous ℓ observations, associated with the run length hypothesis $r_t = \ell$:

$$\mathbf{D}^{(\ell)} \coloneqq \mathbf{D}_{t-\ell:t-1}, \tag{5.38}$$

$$\boldsymbol{\chi}_\ell \coloneqq \boldsymbol{\chi} + \sum_{\tau=t-\ell}^{t-1} \zeta_\tau u(\mathbf{x}_\tau), \tag{5.39}$$

$$\nu_\ell \coloneqq \nu + \sum_{\tau=t-\ell}^{t-1} \zeta_\tau. \tag{5.40}$$

Then the posterior predictive is

$$\begin{aligned}
&p(\mathbf{x}_t \mid r_t = \ell, \zeta_t, \mathbf{D}^{(\ell)}) \\
&= \int_{\boldsymbol{\Theta}} p_{\boldsymbol{\theta}}(\mathbf{x}_t)^{\zeta_t} \pi_{\boldsymbol{\chi}_\ell, \nu_\ell}(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} \\
&= \int_{\boldsymbol{\Theta}} [h_1(\mathbf{x}_t)]^{\zeta_t} \exp\left\{\boldsymbol{\theta}^\top \zeta_t u(\mathbf{x}_t) - \zeta_t a_1(\boldsymbol{\theta})\right\} \\
&\quad h_2(\boldsymbol{\theta}) \exp\left\{\boldsymbol{\theta}^\top \boldsymbol{\chi}_\ell - \nu_\ell a_1(\boldsymbol{\theta}) - a_2(\boldsymbol{\chi}_\ell, \nu_\ell)\right\} \mathrm{d}\boldsymbol{\theta} \qquad (5.41) \\
&= [h_1(\mathbf{x}_t)]^{\zeta_t} \frac{\int_{\boldsymbol{\Theta}} h_2(\boldsymbol{\theta}) \exp\left\{\boldsymbol{\theta}^\top [\zeta_t u(\mathbf{x}_t) + \boldsymbol{\chi}_\ell] - a_1(\boldsymbol{\theta}) [\zeta_t + \nu_\ell]\right\} \mathrm{d}\boldsymbol{\theta}}{\exp\{a_2(\boldsymbol{\chi}_\ell, \nu_\ell)\}} \\
&\overset{\star}{=} [h_1(\mathbf{x}_t)]^{\zeta_t} \frac{\exp\{a_2(\zeta_t u(\mathbf{x}_t) + \boldsymbol{\chi}_\ell, \zeta_t + \nu_\ell)\}}{\exp\{a_2(\boldsymbol{\chi}_\ell, \nu_\ell)\}} \\
&= [h_1(\mathbf{x}_t)]^{\zeta_t} \exp\{a_2(\zeta_t u(\mathbf{x}_t) + \boldsymbol{\chi}_\ell, \zeta_t + \nu_\ell) - a_2(\boldsymbol{\chi}_\ell, \nu_\ell)\}
\end{aligned}$$

Step $\star$ follows from the previous line because we know the normalizer for the integral.

5A.2.2. Multi-fidelity Gaussian model

To simplify notation, we ignore the run length in this section, since it only specifies which data need to be accounted for in the MF-posterior distribution. Consider a univariate Gaussian model with known variance.

$$x_i \overset{\text{iid}}{\sim} \mathcal{N}(\theta_t, \sigma_x^2), \quad \theta_t \sim \mathcal{N}(\mu_0, \sigma_0^2). \qquad (5.42)$$

The multi-fidelity likelihood is

$$\begin{aligned}
\prod_{i=1}^{t} p_{\theta_t}(x_i)^{\zeta_i} &= \prod_{i=1}^{t} \left[\frac{1}{\sqrt{2\pi\sigma_x^2}} \exp\left\{-\frac{1}{2\sigma_x^2}(x_i - \theta_t)^2\right\}\right]^{\zeta_i} \qquad (5.43) \\
&\propto \prod_{i=1}^{t} \exp\left\{-\frac{\zeta_i}{2\sigma_x^2}(x_i - \theta_t)^2\right\} \qquad (5.44)
\end{aligned}$$

When $\zeta_i < 1$, the variance of $\mathcal{N}(x_i \mid \sigma_x^2/\zeta_i)$ increases, and the fidelity hyperparameter has the natural interpretation of increasing the variance of our model.

The multi-fidelity posterior is the product of $t+1$ independent Gaussian densities, which is itself Gaussian:

$$\pi(\theta_t \mid \mathbf{D}_{1:t}) \propto \mathcal{N}(\theta_t \mid \mu_0, \sigma_0^2) \prod_{i=1}^{t} \mathcal{N}(x_i \mid \theta_t, \sigma_x^2/\zeta_i) \tag{5.45}$$

$$\propto \mathcal{N}(\theta_t \mid \mu_t, \sigma_t^2), \tag{5.46}$$

where

$$\frac{1}{\sigma_t^2} := \frac{1}{\sigma_0^2} + \sum_{i=1}^{t} \frac{\zeta_i}{\sigma_x^2}, \tag{5.47}$$

$$\mu_t := \sigma_t^2 \left(\frac{\mu_0}{\sigma_0^2} + \sum_{i=1}^{t} \frac{\zeta_i x_i}{\sigma_x^2} \right). \tag{5.48}$$

The MF-posterior predictive can be computed by integrating out θ_t. This is a convolution of two Gaussians, the posterior in Equation (5.14) and the prior $\pi(\theta) = \mathcal{N}(\theta \mid \mu_0, \sigma_0^2)$, which is again Gaussian:

$$p(x_{t+1} \mid \zeta_{t+1}, \mathbf{D}_{1:t}) = \int_{\Theta} [\mathcal{N}(x_{t+1} \mid \theta_t, \sigma_x^2)]^{\zeta_{t+1}} \mathcal{N}(\theta_t \mid \mu_t, \sigma_t^2) \mathrm{d}\theta_t \tag{5.49}$$

$$= \mathcal{N}\left(x_{t+1} \mid \mu_t, \frac{\sigma_x^2}{\zeta_{t+1}} + \sigma_t^2 \right). \tag{5.50}$$

With a single fidelity and $\zeta = 1$, this results reduces to the standard result for Gaussian models with known variance [Murphy, 2007].

5A.2.3. Multi-fidelity Bernoulli model

To simplify notation, we ignore the run length in this section, since it only specifies which data need to be accounted for in the MF-posterior distribution. Consider a beta–Bernoulli model

$$x_i \overset{\text{iid}}{\sim} \text{Bernoulli}(\theta_t), \quad \theta_t \sim \text{Beta}(\alpha_0, \beta_0). \tag{5.51}$$

The multi-fidelity likelihood is

$$\prod_{i=1}^{t} p_{\theta_t}(x_i)^{\zeta_i} = \prod_{i=1}^{t} \left[\theta_t^{x_i}(1-\theta_t)^{1-x_i}\right]^{\zeta_i} \tag{5.52}$$

$$= \prod_{i=1}^{t} \theta_t^{\zeta_i x_i}(1-\theta_t)^{\zeta_i(1-x_i)}. \tag{5.53}$$

Therefore the MF-posterior is

$$\pi(\theta_t) \prod_{i=1}^{t} p_{\theta_t}(x_i)^{\zeta_i} \propto \frac{1}{\mathrm{B}(\alpha_0, \beta_0)} \theta_t^{\alpha_0 - 1}(1-\theta_t)^{\beta_0 - 1} \prod_{i=1}^{t} \theta_t^{\zeta_i x_i}(1-\theta_t)^{\zeta_i(1-x_i)} \tag{5.54}$$

$$\propto \theta_t^{\alpha_0 - 1 + \sum_t \zeta_i x_i}(1-\theta_t)^{\beta_0 - 1 + \sum_t \zeta_i - x_i \zeta_i}. \tag{5.55}$$

So the MF-posterior is proportional to a beta distribution

$$\pi(\theta_t \mid \mathbf{D}_{1:t}) = \mathrm{Beta}(\alpha_t, \beta_t), \tag{5.56}$$

$$\alpha_t \coloneqq \alpha_0 + \sum_{i=1}^{t} \zeta_i x_i, \tag{5.57}$$

$$\beta_t \coloneqq \beta_0 + \sum_{i=1}^{t} \zeta_i (1 - x_i). \tag{5.58}$$

The MF-posterior predictive is:

$$\begin{aligned}
&p(x_{t+1} \mid \zeta_{t+1}, \mathbf{D}_{1:t}) \\
&= \int_0^1 p_{\theta_t}(x_{t+1})^{\zeta_{t+1}} p(\theta_t \mid \mathbf{D}_{1:t}) \mathrm{d}\theta_t \\
&= \int_0^1 \left(\theta_t^{x_{t+1}}(1-\theta_t)^{1-x_{t+1}}\right)^{\zeta_{t+1}} \left(\frac{1}{\mathrm{B}(\alpha_t, \beta_t)} \theta_t^{\alpha_t - 1}(1-\theta_t)^{\beta_t - 1}\right) \mathrm{d}\theta_t \\
&= \frac{1}{\mathrm{B}(\alpha_t, \beta_t)} \int_0^1 \theta_t^{\zeta_{t+1} x_{t+1} + \alpha_t - 1}(1-\theta_t)^{\zeta_{t+1}(1-x_{t+1}) + \beta_t - 1} \mathrm{d}\theta_t \\
&= \frac{\mathrm{B}(\alpha_t + \zeta_{t+1} x_{t+1}, \beta_t + \zeta_{t+1}(1 - x_{t+1}))}{\mathrm{B}(\alpha_t, \beta_t)}.
\end{aligned} \tag{5.59}$$

The last step as, as in the general case, depends on knowing the normalizer of the beta distribution. Notice that the base measure $h_1(x_t)$ of the Bernoulli distribution is one, and therefore $[h_1(x_t)]^{\zeta_t} = 1$.

5A.3 Proof that mutual information is symmetric

The mutual information (MI) between two random variables captures how much information entropy is obtained about one random variable by observing the other. Since that definition does not specify which is the observed random variable, we might suspect this is a symmetric quantity. In fact, it is. The goal of this section is to show why this definition is indeed symmetric. The proof will highlight a useful interpretation of MI.

Let X and Y be continuous random variables with probability functions $p_X(x)$ and $p_Y(y)$ respectively. The MI of X and Y is

$$\begin{aligned}
&\mathrm{MI}(X,Y) \\
&= \mathbb{H}[X] - \mathbb{E}_Y\left[\mathbb{H}[X \mid Y = y]\right] \\
&= -\int_x p_X(x)\ln p_X(x)\mathrm{d}x + \int_y p_Y(y)\int_x p_{X|Y}(x,y)\ln p_{X|Y}(x,y)\mathrm{d}x\mathrm{d}y \\
&= -\int_y\int_x p_{X,Y}(x,y)\ln p_X(x)\mathrm{d}x\mathrm{d}y + \int_y p_Y(y)\int_x p_{X|Y}(x,y)\ln p_{X|Y}(x,y)\mathrm{d}x\mathrm{d}y \\
&= -\int_y\int_x p_{X,Y}(x,y)\ln p_X(x)\mathrm{d}x\mathrm{d}y + \int_y\int_x p_{X,Y}(x,y)\ln p_{X|Y}(x,y)\mathrm{d}x\mathrm{d}y \\
&= \int_y\int_x p_{X,Y}(x,y)\Big(\ln p_{X|Y}(x,y) - \ln p_X(x)\Big)\mathrm{d}x\mathrm{d}y \\
&= \int_y\int_x p_{X,Y}(x,y)\ln\left[\frac{p_{X|Y}(x,y)}{p_X(x)}\right]\mathrm{d}x\mathrm{d}y \\
&= \int_y\int_x p_{X,Y}(x,y)\ln\left[\frac{p_{X|Y}(x,y)}{p_X(x)} \times \frac{p_Y(y)}{p_Y(y)}\right]\mathrm{d}x\mathrm{d}y \\
&= \int_y\int_x p_{X,Y}(x,y)\ln\left[\frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)}\right]\mathrm{d}x\mathrm{d}y \\
&= D_{\mathrm{KL}}[p_{X,Y} \| p_X \otimes p_Y].
\end{aligned} \tag{5.60}$$

Since the KL divergence is non-negative (Section 2A.2), mutual information is also non-negative. Furthermore, the mutual information is zero if and only if X and Y are independent. This makes intuitive sense: if two random variables are independent, observing one tells you nothing about the other.

Finally, it's pretty easy to see that we can simply reverse our calculations to get the mutual information of Y and X:

$$\begin{aligned}
&\mathrm{MI}(Y, X) \\
&= \mathbb{H}[Y] - \mathbb{E}_X\left[\mathbb{H}[Y \mid X = x]\right] \\
&= -\int_y p_Y(y) \ln p_Y(y) \mathrm{d}y + \int_x p_X(x) \int_y p_{Y|X}(y, x) \ln p_{Y|X}(y, x) \mathrm{d}y \mathrm{d}x \\
&= -\int_x \int_y p_{X,Y}(x, y) \ln p_Y(y) \mathrm{d}y \mathrm{d}x + \int_x p_X(x) \int_y p_{Y|X}(y, x) \ln p_{Y|X}(y, x) \mathrm{d}y \mathrm{d}x \\
&= -\int_x \int_y p_{X,Y}(x, y) \ln p_Y(y) \mathrm{d}y \mathrm{d}x + \int_x \int_y p_{X,Y}(x, y) \ln p_{Y|X}(y, x) \mathrm{d}y \mathrm{d}x \\
&= \int_x \int_y p_{X,Y}(x, y) \Big(\ln p_{Y|X}(x, y) - \ln p_Y(y) \Big) \mathrm{d}y \mathrm{d}x \\
&= \int_x \int_y p_{X,Y}(x, y) \ln \left[\frac{p_{Y|X}(x, y)}{p_Y(y)} \right] \mathrm{d}y \mathrm{d}x \\
&= \int_x \int_y p_{X,Y}(x, y) \ln \left[\frac{p_{Y|X}(x, y)}{p_Y(y)} \times \frac{p_X(x)}{p_X(x)} \right] \mathrm{d}y \mathrm{d}x \\
&= \int_x \int_y p_{X,Y}(x, y) \ln \left[\frac{p_{X,Y}(x, y)}{p_X(x) p_Y(y)} \right] \mathrm{d}y \mathrm{d}x \\
&= \mathrm{KL}[p_{X,Y} \| p_X \otimes p_Y].
\end{aligned} \tag{5.61}$$

It's easy to see that these derivations hold if X and Y are both discrete. The tricky case is if X is discrete and Y is continuous. Certainly, the derivations still work if we can interchange integrals and sums, which is true for finite sums. However, when the sums are infinite, we are effectively interchanging limits and integration. I don't know enough measure theory to know when this is possible.

5A.4 MF-BOCD algorithm in didactic code

This Python code is a didactic example of the MF-BOCD algorithm. At each time step, the algorithm (1) chooses a data fidelity using maximal information rate; (2) observes a datum of the chosen fidelity; (3-4) computes the posterior predictive and run-length posterior distributions; (5) updates the model parameters; and (6) makes a prediction. Please see the code repository[3] for a complete example.

```python
import numpy as np
from   scipy.special import logsumexp

def mf_bocd(data, model, hazard, costs):
    J, T        = data.shape
    log_message = np.array([1])
    log_R       = np.ones((T+1, T+1))
    log_R[0, 0] = 1
    pmean       = np.zeros(T)
    igs         = np.empty(J)
    choices     = np.empty(T)

    for t in range(1, T+1):

        # 1. Choose fidelity.
        rl_post = np.exp(log_R[t-1, :t])
        for j in range(J):
            igs[j] = compute_info_gain(t, model, rl_post, log_message, hazard, j)
        j_star = np.argmax(igs / costs)
        choices[t-1] = j_star

        # 2. Observe new datum.
        x = data[j_star, t-1]

        # 3. Compute predictive probabilities.
        log_pis = model.log_pred_prob(t, x, j_star)
```

[3]https://github.com/gwgundersen/mf-bocd

```

        # 4. Estimate run length distribution.
        log_growth_probs = log_pis + log_message + np.log(1 - hazard)
        log_cp_prob      = logsumexp(log_pis + log_message + np.log(hazard))
        new_log_joint    = np.append(log_cp_prob, log_growth_probs)
        log_R[t, :t+1]   = new_log_joint
        log_R[t, :t+1]  -= logsumexp(new_log_joint)

        # 5. Update model parameters and message pass.
        model.update_params(t, x, j_star)
        log_message = new_log_joint

        # 6. Predict.
        pmean[t-1] = np.sum(model.mean_params[:t] * rl_post)

    return choices, np.exp(log_R), pmean
```

5A.5 Ablation studies

Here, we report the results of an ablation study for the multi-fidelity Gaussian and multi-fidelity Bernoulli models described in Section 5.3.3. For varying costs, a multi-fidelity model using information gain-based switching was run on data generated from their respective data generating proceses. The percentage of low-fidelity observations was recorded; call this P_{low}. Then a randomized multi-fidelity model was run on the same data set. At each time step, the randomized model chose low-fidelity data based on a Bernoulli random variable with bias P_{low}. The goal of this experiment is to demonstrate that when the model switches to high-fidelity data is important to model performance, not just the fact that some percentage of high-fidelity data are used. We found that for both Gaussian (Table 5A.5.1) and Bernoulli data (Table 5A.5.2), choosing when to switch fidelities was often useful.

Table 5A.5.1: Ablation study for multi-fidelity Gaussian models. "LF only" is BOCD using only low-fidelity data. Mean and two standard errors, representing 95% confidence intervals, are reported over 200 trials. Bold numbers indicate statistically significant using 95% confidence intervals.

	MSE			L_1		
LF (%)	LF only	Random	Info-based	LF only	Random	Info-based
1	0.879 (0.034)	0.046 (0.056)	0.003 (0.001)	270.87 (8.35)	**5.98** (3.06)	73.92 (9.61)
2		0.125 (0.073)	0.111 (0.046)		**18.18** (7.37)	77.68 (9.79)
38		0.680 (0.118)	**0.494** (0.059)		162.31 (12.97)	161.05 (11.08)
53		0.702 (0.091)	**0.483** (0.066)		183.40 (11.36)	173.01 (10.46)
60		0.752 (0.140)	**0.452** (0.037)		186.11 (10.89)	174.95 (10.13)
67		0.665 (0.075)	**0.466** (0.036)		187.72 (10.01)	173.41 (9.91)
74		0.643 (0.064)	**0.480** (0.043)		182.18 (9.48)	175.88 (9.36)
80		0.656 (0.087)	**0.492** (0.044)		184.66 (9.20)	175.70 (9.20)
97		0.547 (0.028)	0.537 (0.028)		176.76 (9.40)	175.34 (9.33)

Table 5A.5.2: Ablation study for multi-fidelity Bernoulli models. "LF only" is BOCD using only low-fidelity data. Mean and two standard errors, representing 95% confidence intervals, are reported over 200 trials. Bold numbers indicate statistically significant using 95% confidence intervals.

	MSE			L_1		
LF (%)	LF only	Random	Info-based	LF only	Random	Info-based
9	0.123 (0.009)	0.003 (0.001)	**0.002** (0.000)	186.27 (7.02)	45.55 (6.04)	40.31 (5.43)
21		0.008 (0.001)	0.009 (0.002)		76.42 (7.68)	71.88 (7.54)
25		0.011 (0.002)	0.011 (0.002)		84.47 (8.11)	80.61 (7.89)
46		0.025 (0.003)	0.021 (0.003)		124.80 (7.07)	117.34 (7.31)
61		0.040 (0.004)	0.034 (0.005)		143.87 (6.34)	139.88 (7.23)
68		0.050 (0.005)	**0.040** (0.005)		158.48 (6.34)	149.79 (7.02)
73		0.057 (0.005)	0.048 (0.006)		163.68 (6.39)	158.08 (6.66)
83		0.077 (0.006)	**0.064** (0.007)		174.01 (6.21)	170.26 (6.49)
90		0.098 (0.007)	**0.082** (0.007)		184.46 (6.11)	178.86 (6.28)

5A.6 Experiment details

5A.6.1. CamVid experiment details

The pretrained MobileNets were downloaded from the Fastseg Python library.[4]

We can estimate the computational cost of MF-BOCD (λ_{MF}) relative to BOCD using only high- (λ_{HF}) and low- (λ_{LF}) fidelity data. We used 85 low- and 86 high- fidelity observations. The low- (high-) fidelity observation model required 19.48 (36.89) billion flops (Table 5A.6.1). Computing the information gain required 465,291 flops. The total cost of our algorithm in billions of flops is

$$\begin{aligned}
\lambda_{\mathsf{LF}} &= 171 \times 19.5 \approx 3333, \\
\lambda_{\mathsf{HF}} &= 171 \times 36.9 \approx 6303, \\
\lambda_{\mathsf{MF}} &= 0.00046 + (85 \times 19.5) + (86 \times 36.7) \approx 4827.
\end{aligned}$$

As we can see, decision-making has a marginal cost.

Table 5A.6.1: Observation model details for CamVid and MIMII experiments. (CamVid) The high-fidelity model has roughly twice times the number of flops and higher accuracy as measured by intersection-over-union (IoU) on the Cityscapes data set. (MIMII) The high-fidelity model requires roughly 250 times as many floating point operations (ops). "FC", "M", and "B" mean fully-connected, millions, and billions respectively.

	Fidelity	Model	Ops	Accuracy
CamVid	HF	V3-large	36.86 B	72.3 (IoU%)
	LF	V3-small	19.48 B	67.4 (IoU%)
MIMII	HF	MicroNet-AD(M)	124.7 M	96.15 (AUC%)
	LF	Two-layer FC	0.5 M	86.7 (AUC%)

5A.6.2. MIMII experiment details

In the MIMII experiment, the output of the observation models (Table 5A.6.1) is a scalar anomaly score in the range $[0, 1]$, where 0 indicates normal machine operation. An illustration

[4]https://github.com/ekzhang/fastseg

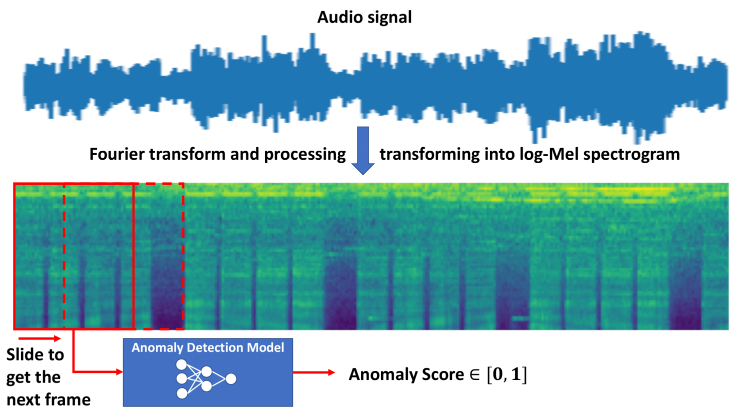
Audio signal
Fourier transform and processing
transforming into log-Mel spectrogram
Slide to get the next frame
Anomaly Detection Model
Anomaly Score ∈ [0, 1]

Figure 5A.6.1: Illustration of pipeline to generate anomaly scores from log-Mel spectrograms using deep neural networks.

of how these scores are obtained for an audio clip is shown in Figure 5A.6.1. To convert these anomaly scores to binary numbers for a Bernoulli multi-fidelity posterior predictive model, we thresholded the scores to integers in $\{0, 1\}$. The quality of the observation models depends on the choice of threshold. For examples of these data, see Figure 5A.6.2. To select the appropriate threshold, we used the intersection of the false negative and false positive rate curves, which corresponds to the top-left corner of the receiver operating characteristic (ROC) curves for each machine and each observation model (Figure 5A.6.3).

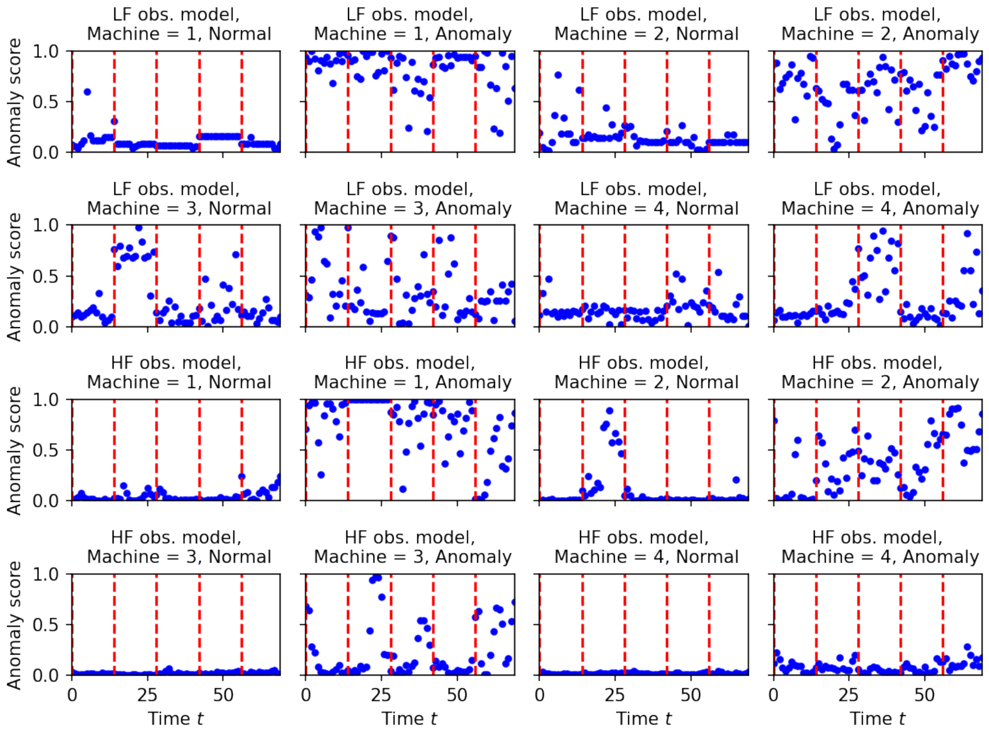

Figure 5A.6.2: Examples of MIMII anomaly scores, five audio clips for each machine. Dashed red lines separate audio clips.

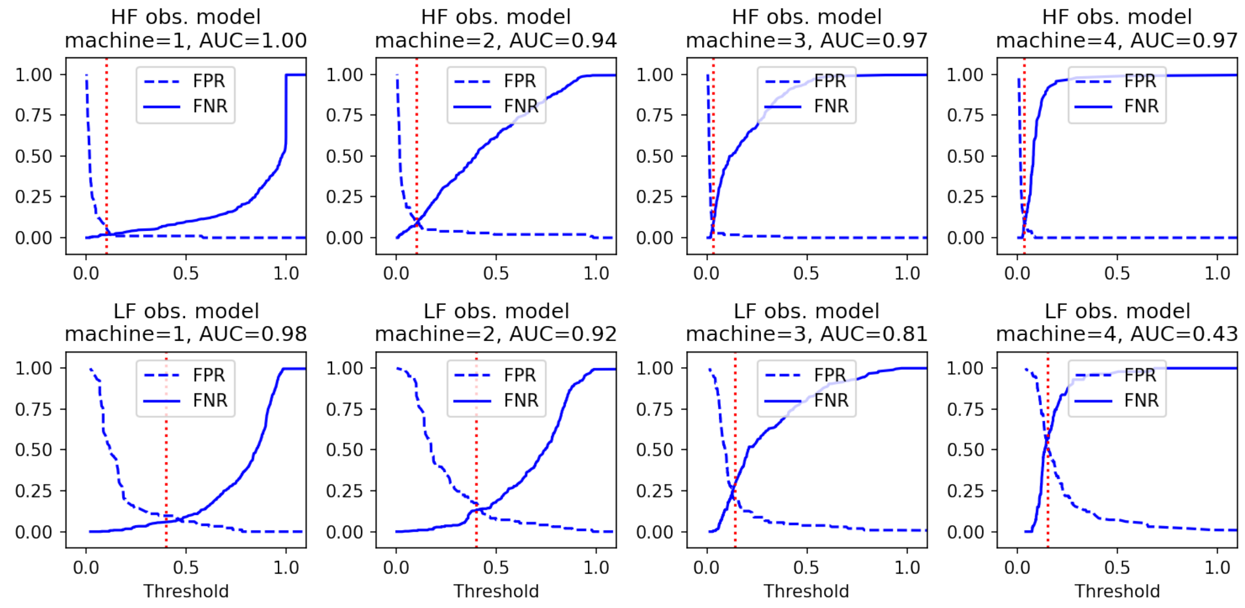

Figure 5A.6.3: False positive (FPR) and false negative rates (FNR) for high- (HF) and low- (LF) fidelity observation models on MIMII cross-validation data. Vertical dashed lines indicated the chosen threshold

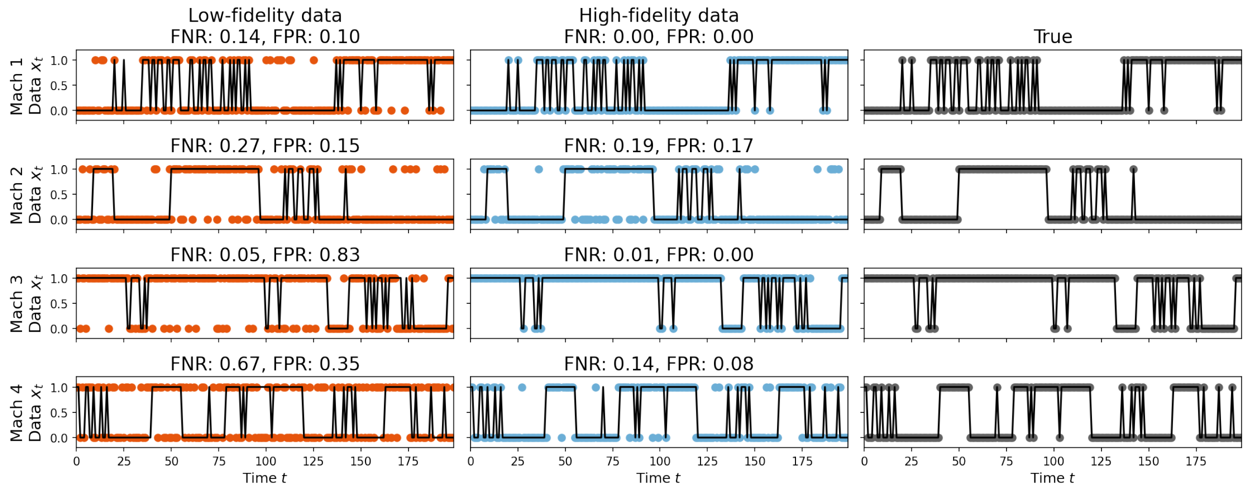

Figure 5A.6.4: Illustration of MIMII data after converting log-Mel spectrograms to binary numbers with machine- and observation model-specific thresholds. The true binary value is denoted with a black line.

Chapter 6

Conclusion

In this final chapter, I discuss future directions for the three lines of work proposed in this thesis. I conclude by briefly reviewing the main contributions of my work.

6.1 Future directions

6.1.1. Joint analysis of GTEx v8 data

Due to a data embargo, our work on deep probabilistic CCA used the GTEx v6 (rather than v8) data set, which contains 2221 paired samples across 29 tissue types. (See Section 3.1 for details.) In follow-up work, we are extending our joint analysis to the larger GTEx v8 data set [Consortium et al., 2020]. This contains bulk gene expression profiles, genotype data, and histological images for 935 donors across 55 tissues, for a total of 13,360 samples across all three modalities. This is roughly six times as many samples as in GTEx v6. Joint analysis of this larger data set will provide an unprecedented view into the relationship between molecular features and histological image features.

6.1.2. Structured priors for RFLVMs

As discussed in Section 4.3.2, RFLVMs are identifiable up to the rotation and scale of the latent variables. To accelerate the Gibbs sampler, we arbitrarily fixed the scale of the latent variables by adjusting the covariance matrix [Liu et al., 1998, Ročková and George, 2016]. However, this covariance adjustment prevents heteroscedasticity in the latent features and correlation between the latent variables. This in turn limits the use of more structured priors on the latent space. In particular, our results in Section 4.4.2 were suboptimal relative to a Poisson GPLVM with a GP prior on the latent space [Wu et al., 2017]. In that experiment, a smooth prior on the latent variables is desirable, since we hypothesize that the latent structure of hippocampal place cells should encode spatial information. A natural extension to the RFLVM framework, and one that we are currently pursuing, is to forgo this covariance adjustment step in favor of more structured priors, such as the dynamic prior used in Gaussian process dynamic models [Wang et al., 2005] or a GP prior with a Matérn kernel used in Wu et al. [2017]. This would enable modelers to encode more domain-specific structure into the inferred latent variables and broaden the applicability of RFLVMs.

6.1.3. Generalized MF-posterior framework

In Section 5.3, we proposed the MF-posterior, which re-weights the terms in the data likelihood based on fidelity-specific hyperparameters. This idea was explored because raising data likelihoods to a power is an established technique [Heide et al., 2020, Grünwald et al., 2017, Walker and Hjort, 2001, Bissiri et al., 2016, Miller and Dunson, 2018, Wang et al., 2017] and when dealing with models in the exponential family, often induces new distributions that are still tractable [Miller and Dunson, 2018]. This is appealing for models such as BOCD, which require estimating the posterior predictive distribution many times. However, the MF-posterior is an orthogonal idea to changepoint detection. A natural extension to the MF-BOCD framework would be to explore the MF-posterior in isolation, with more theoretical analysis on how the posterior concentrates along two axes, the data set size and values of the fidelity hyperparameters. This might allow for novel and tractable multi-fidelity modeling for a broader class of models.

6.2 Summary of contributions

Scientists have been interested in latent variable models since the pioneering work on methods such as factor analysis [Spearman, 1904], PCA [Pearson, 1901], and CCA [Hotelling, 1936]. These linear–Gaussian factor models have been reformulated as state-space models such as the Kalman filter [Kalman, 1960] and HMMs [Baum and Petrie, 1966] and as probabilistic models [Tipping and Bishop, 1999, Bach and Jordan, 2005]. While these linear–Gaussian factor models are still popular today, more flexible latent variable models have only recently received more attention from the research community, such as the autoencoder [Baldi and Hornik, 1989], the variational autoencoder [Kingma and Welling, 2013], the GPLVM [Lawrence, 2004], BOCD [Adams and MacKay, 2007, Fearnhead and Liu, 2007], deep CCA [Andrew et al., 2013], and deep HMMs [Krishnan et al., 2017]. Inference for flexible latent variable models is challenging because the latent variables induce complex dependencies between the hidden and observed variables and because flexible models are typically not conjugate.

In this thesis, I contributed the first end-to-end inference framework for composing deep neural networks with probabilistic CCA and showed that the model inferred interpretable and biologically meaningful latent structure. I then proposed the use of random features to extend the GPLVM framework to non-Gaussian data likelihoods and showed that we could build competitive nonlinear dimension reduction models for a variety of data likelihoods in the exponential family. In particular, we were able to infer the position of a rat given neural recordings of its hippocampal place cells. And finally, I proposed the first use of multi-fidelity modeling to accelerate inference for BOCD. We showed that we could lower the computational cost of BOCD by actively selecting each datum's fidelity based on maximizing the information about the posterior distribution over changepoints. Even when using state-of-the-art neural network architectures designed for commodity hardware, the cost of decision-making was insignificant relative to the cost of the cheapest observation models.

Latent variable models allow scientists to build, infer, and critique statistical models of their data using mathematically principled tools and techniques. While the work in this thesis spans a broad range of models and applications, the underlying theme is an ambition to aid the work of researchers and engineers through practical inference algorithms for flexible, tractable, and scalable latent variable models.

Appendix A

Prior Presentations and Publications

1. Active multi-fidelity Bayesian online changepoint detection.
   Gregory W. Gundersen, Diana Cai, Chuteng Zhu, Barbara E. Engelhardt, Ryan P. Adams.
   Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence, 2021.

2. Latent variable modeling with random features.
   Gregory W. Gundersen*, Michael Minyi Zhang*, Barbara E. Engelhardt.
   24th International Conference on Artificial Intelligence and Statistics, 2021.

3. End-to-end training of deep probabilistic CCA for joint modeling of paired biomedical observations.
   Gregory W. Gundersen, Bianca Dumitrascu, Jordan T. Ash, Barbara E. Engelhardt.
   Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence, 2019.

4. End-to-end training of deep probabilistic CCA for joint modeling of paired biomedical observations.
   Gregory W. Gundersen, Bianca Dumitrascu, Jordan T. Ash, Barbara E. Engelhardt.
   Third workshop on Bayesian Deep Learning, *Advances in Neural Information Processing Systems 31*, 2018.

** Denotes equal contribution.*

Bibliography

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

Ryan Prescott Adams and David JC MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

Samuel K Ainsworth, Nicholas J Foti, Adrian KC Lee, and Emily B Fox. oi-vae: Output interpretable vaes for nonlinear group factor analysis. In *International Conference on Machine Learning*, pages 119–128, 2018.

Shotaro Akaho. A kernel method for canonical correlation analysis. *arXiv preprint cs/0609071*, 2006.

Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. Oboe: auto-tuning video ABR algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 44–58, 2018.

David J Aldous. Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII—1983*, pages 1–198. Springer, 1985.

Samaneh Aminikhanghahi, Tinghui Wang, and Diane J Cook. Real-time change point detection with application to smart home time series data. *IEEE Transactions on Knowledge and Data Engineering*, 31(5):1010–1023, 2018.

Galen Andrew, Raman Arora, Jeff Bilmes, and Karen Livescu. Deep canonical correlation analysis. In *International Conference on Machine Learning*, pages 1247–1255, 2013.

Charles E Antoniak. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *The Annals of Statistics*, pages 1152–1174, 1974.

Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *International Conference on Machine Learning*, pages 244–253. PMLR, 2018.

Jordan T Ash, Gregory Darnell, Daniel Munro, and Barbara E Engelhardt. Joint analysis of expression levels and histological images identifies genes associated with tissue morphology. *Nature Communications*, 12(1):1–12, 2021.

Francis R Bach and Michael I Jordan. A probabilistic interpretation of canonical correlation analysis. 2005.

Stuart G Baker. The multinomial-Poisson transformation. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 43(4):495–504, 1994.

Mukund Balasubramanian, Eric L Schwartz, Joshua B Tenenbaum, Vin de Silva, and John C Langford. The Isomap algorithm and topological stability. *Science*, 295(5552):7–7, 2002.

Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakkar, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul N Whatmough. MicroNets: Neural network architectures for deploying TinyML applications on commodity microcontrollers. *arXiv preprint arXiv:2010.11267*, 2020.

Yaniv Bar, Idit Diamant, Lior Wolf, Sivan Lieberman, Eli Konen, and Hayit Greenspan. Chest pathology detection using deep learning with non-medical training. In *Biomedical Imaging (ISBI), 2015 IEEE 12th International Symposium on*, pages 294–297. IEEE, 2015.

Daniel Barry and John A Hartigan. Product partition models for change point problems. *The Annals of Statistics*, pages 260–279, 1992.

Joseph D Barry, Maud Fagny, Joseph N Paulson, Hugo JWL Aerts, John Platig, and John Quackenbush. Histopathological image qtl discovery of immune infiltration variants. *iScience*, 5:80–89, 2018.

Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966.

Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18, 2018.

FRS Bayes. An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, 1763.

Matthias Beck, Daniel Blado, Joseph Crawford, Taïna Jean-Louis, and Michael Young. On weak chromatic polynomials of mixed graphs. *Graphs and Combinatorics*, 2013.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

Christopher M Bishop. Pattern recognition and machine learning. 2006.

Pier Giovanni Bissiri, Chris C Holmes, and Stephen G Walker. A general framework for updating belief distributions. *Journal of the Royal Statistical Society. Series B, Statistical methodology*, 78(5):1103, 2016.

David M Blei. Build, compute, critique, repeat: Data analysis with latent variable models. *Annual Review of Statistics and Its Application*, 1:203–232, 2014.

David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

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

Salomon Bochner. *Lectures on Fourier integrals*, volume 42. Princeton University Press, 1959.

Jerome V Braun and Hans-Georg Muller. Statistical methods for DNA sequence segmentation. *Statistical Science*, pages 142–162, 1998.

Leo Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

Gabriel J Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009.

Michael W Browne. The maximum-likelihood solution in inter-battery factor analysis. *British Journal of Mathematical and Statistical Psychology*, 32(1):75–86, 1979.

Charles G Broyden. Quasi-newton methods and their application to function minimisation. *Mathematics of Computation*, 21(99):368–381, 1967.

Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

Bernard Bru and Marc Yor. Comments on the life and mathematical legacy of wolfgang doeblin. *Finance and Stochastics*, 6(1):3–47, 2002.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

Emmanuel J Candes, Justin K Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 59(8):1207–1223, 2006.

Latarsha J Carithers, Kristin Ardlie, Mary Barcus, Philip A Branton, Angela Britton, Stephen A Buia, Carolyn C Compton, David S DeLuca, Joanne Peter-Demchok, Ellen T Gelfand, et al. A novel approach to high-quality postmortem tissue procurement: the gtex project. *Biopreservation and biobanking*, 13(5):311–319, 2015.

Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus A Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: a probabilistic programming language. *Grantee Submission*, 76(1):1–32, 2017.

Augustin Cauchy et al. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

Jianfei Chen, Jun Zhu, Zi Wang, Xun Zheng, and Bo Zhang. Scalable inference for logistic-normal topic models. In *Advances in Neural Information Processing Systems*, pages 2445–2453, 2013.

Siddhartha Chib. Estimation and comparison of multiple change-point models. *Journal of econometrics*, 86(2):221–241, 1998.

Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

Kyucheol Cho, Moonsup Lee, Dongmin Gu, William A Munoz, Hong Ji, Malgorzata Kloc, and Pierre D McCrea. Kazrin, and its binding partners arvcf-and delta-catenin, are required for xenopus laevis craniofacial development. *Developmental Dynamics*, 240(12): 2601–2612, 2011.

Nicolas Chopin. Dynamic detection of change points in long time series. *Annals of the Institute of Statistical Mathematics*, 59(2):349–366, 2007.

GTEx Consortium et al. Genetic effects on gene expression across human tissues. *Nature*, 550(7675):204, 2017.

GTEx Consortium et al. The gtex consortium atlas of genetic regulatory effects across human tissues. *Science*, 369(6509):1318–1330, 2020.

Lee AD Cooper, Jun Kong, David A Gutman, Fusheng Wang, Jingjing Gao, Christina Appin, Sharath Cholleti, Tony Pan, Ashish Sharma, Lisa Scarpace, et al. Integrated morphologic analysis for the identification and characterization of disease subtypes. *Journal of the American Medical Informatics Association*, 19(2):317–323, 2012.

John P Cunningham and M Yu Byron. Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience*, 17(11):1500–1509, 2014.

Kurt Cutajar, Edwin V Bonilla, Pietro Michiardi, and Maurizio Filippone. Random feature expansions for deep Gaussian processes. In *International Conference on Machine Learning*, pages 884–893. PMLR, 2017.

Andreas C Damianou, Michalis K Titsias, and Neil Lawrence. Variational inference for latent variables and uncertain inputs in gaussian processes. 2016.

Georges Darmois. Sur les lois de probabilitéa estimation exhaustive. *CR Acad. Sci. Paris*, 260(1265):85, 1935.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

John S Denker, WR Gardner, Hans Peter Graf, Donnie Henderson, Richard E Howard, W Hubbard, Lawrence D Jackel, Henry S Baird, and Isabelle Guyon. Neural network recognizer for hand-written zip code digits. In *Advances in neural information processing systems*, pages 323–331. Citeseer, 1989.

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

Wolfgang Doeblin. Sur les propriétés asymptotiques de mouvement régis par certains types de chaines simples. *Bulletin mathématique de la Société roumaine des sciences*, 39(1): 57–115, 1937.

Wolfgang Doeblin. Sur les sommes d'un grand nombre de variables aléatoires indépendantes. *Bull. Sci. Math*, 63(2):23–32, 1939.

Wolfgang Doeblin and Robert Fortet. Sur des chaînes à liaisons complètes. *Bulletin de la Société Mathématique de France*, 65:132–148, 1937.

John Dowling and Edward Gassner. Take the world from another point of view (film), produced by wgbh-tv boston. reviewers. *American Journal of Physics*, 44(9):900–901, 1976.

Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

Kumar Avinava Dubey, Michael Minyi Zhang, Eric Xing, and Sinead Williamson. Distributed, partially collapsed MCMC for Bayesian nonparametrics. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108, pages 3685–3695. PMLR, 2020. URL http://proceedings.mlr.press/v108/dubey20a.html.

Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973.

Barbara E Engelhardt and Matthew Stephens. Analysis of population structure: a unifying framework and novel methods based on sparse factor analysis. *PLoS Genet*, 6(9):e1001117, 2010.

Gökcen Eraslan, Lukas M Simon, Maria Mircea, Nikola S Mueller, and Fabian J Theis. Single-cell RNA-seq denoising using a deep count autoencoder. *Nature communications*, 10(1):1–14, 2019.

Elena A Erosheva and S McKay Curtis. Dealing with rotational invariance in Bayesian confirmatory factor analysis. *Department of Statistics, University of Washington, Seattle, Washington, USA*, 2011.

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

Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

Zhou Fan, Ron O Dror, Thomas J Mildorf, Stefano Piana, and David E Shaw. Identifying localized changes in large systems: Change-point detection for biomolecular simulations. *Proceedings of the National Academy of Sciences*, 112(24):7454–7459, 2015.

Paul Fearnhead. Exact and efficient Bayesian inference for multiple changepoint problems. *Statistics and computing*, 16(2):203–213, 2006.

Paul Fearnhead and Zhen Liu. On-line inference for multiple changepoint problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):589–605, 2007.

Sarah Ferguson, Brandon Luders, Robert C Grande, and Jonathan P How. Real-time predictive modeling and robust avoidance of pedestrians with uncertain, changing intentions. In *Algorithmic Foundations of Robotics XI*, pages 161–177. Springer, 2015.

Thomas S Ferguson. A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, pages 209–230, 1973.

Roger Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13 (3):317–322, 1970.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 2013.

Matthew Gentzkow and Jesse M Shapiro. What drives media slant? Evidence from US daily newspapers. *Econometrica*, 78(1):35–71, 2010.

Krzysztof J Geras, Stacey Wolfson, S Kim, Linda Moy, and Kyunghyun Cho. High-resolution breast cancer screening with multi-view deep convolutional neural networks. *arXiv preprint arXiv:1703.07047*, 2017.

Alexandra Gessner, Javier Gonzalez, and Maren Mahsereci. Active multi-information source Bayesian quadrature. In *Uncertainty in Artificial Intelligence*, pages 712–721. PMLR, 2020.

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521 (7553):452, 2015.

Zoubin Ghahramani, Geoffrey E Hinton, et al. The em algorithm for mixtures of factor analyzers. Technical report, Technical Report CRG-TR-96-1, University of Toronto, 1996.

Joyee Ghosh and David B Dunson. Default prior distributions and efficient posterior computation in Bayesian factor analysis. *Journal of Computational and Graphical Statistics*, 18(2):306–320, 2009.

Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26, 1970.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Jonathan Goodman and Jonathan Weare. Ensemble samplers with affine invariance. *Communications in applied mathematics and computational science*, 5(1):65–80, 2010.

Prem Gopalan, Jake M Hofman, and David M Blei. Scalable recommendation with hierarchical Poisson factorization. In *UAI*, pages 326–335, 2015.

GPy. GPy: A Gaussian process framework in Python. http://github.com/SheffieldML/GPy, 2012.

Peter Grünwald, Thijs Van Ommen, et al. Inconsistency of Bayesian inference for misspecified linear models, and a proposal for repairing it. *Bayesian Analysis*, 12(4):1069–1103, 2017.

Varun Gulshan, Lily Peng, Marc Coram, Martin C Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, et al. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

David R Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical correlation analysis: An overview with application to learning methods. *Neural computation*, 16(12):2639–2664, 2004.

Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

Jouni Hartikainen and Simo Särkkä. Kalman filtering and smoothing solutions to temporal gaussian process regression models. In *2010 IEEE international workshop on machine learning for signal processing*, pages 379–384. IEEE, 2010.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media, 2009.

W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

Shogo Hayashi, Yoshinobu Kawahara, and Hisashi Kashima. Active change-point detection. In *Asian Conference on Machine Learning*, pages 1017–1032. PMLR, 2019.

Rianne Heide, Alisa Kirichenko, Peter Grunwald, and Nishant Mehta. Safe-Bayesian generalized linear regression. In *International Conference on Artificial Intelligence and Statistics*, pages 2623–2633. PMLR, 2020.

James Hensman, Nicolas Durrande, and Arno Solin. Variational Fourier features for Gaussian processes. *Journal of Machine Learning Research*, 18(1):5537–5588, 2017.

José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems*, pages 918–926, 2014.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

Chris C Holmes, Leonhard Held, et al. Bayesian auxiliary variable models for binary and multinomial regression. *Bayesian Analysis*, 1(1):145–168, 2006.

Harold Hotelling. Relations between two sets of variates. *Biometrika*, 28(3/4):321–377, 1936.

Neil Houlsby, Ferenc Huszar, Zoubin Ghahramani, and Jose Hernández-lobato. Collaborative Gaussian processes for preference learning. *Advances in neural information processing systems*, 25:2096–2104, 2012.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

Tim Hubbard, Daniel Barker, Ewan Birney, Graham Cameron, Yuan Chen, L Clark, Tony Cox, J Cuff, Val Curwen, Thomas Down, et al. The ensembl genome database project. *Nucleic acids research*, 30(1):38–41, 2002.

Ross Ihaka. R: Past and future history. *Computing Science and Statistics*, 392396, 1998.

Matthew Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in neural information processing systems*, pages 2946–2954, 2016.

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. *Neural networks*, 1, 2016.

George Kimeldorf and Grace Wahba. Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95, 1971.

Ross Kindermann. Markov random fields and their applications. *American mathematical society*, 1980.

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

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Jeremias Knoblauch, Jack Jewson, and Theodoros Damoulas. Doubly robust Bayesian inference for non-stationary streaming data with β-divergences. *arXiv preprint arXiv:1806.02261*, 2018.

Jonathan Ko and Dieter Fox. Learning GP-BayesFilters via Gaussian process latent variable models. *Autonomous Robots*, 30(1):3–23, 2011.

Bernard Osgood Koopman. On distributions admitting a sufficient statistic. *Transactions of the American Mathematical society*, 39(3):399–409, 1936.

Rahul Krishnan, Uri Shalit, and David Sontag. Structured inference networks for nonlinear state space models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25: 1097–1105, 2012.

Pierre Simon Laplace. *Théorie analytique des probabilités*. Courcier, 1820.

Derrick Norman Lawley and Albert Ernest Maxwell. Factor analysis as a statistical method. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 12(3):209–229, 1962.

Neil Lawrence and Aapo Hyvärinen. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of machine learning research*, 6(11), 2005.

Neil D Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *Advances in Neural Information Processing Systems*, pages 329–336, 2004.

Neil D Lawrence. Learning for larger datasets with the gaussian process latent variable model. In *Artificial intelligence and statistics*, pages 243–250. PMLR, 2007.

Miguel Lázaro-Gredilla, Joaquin Quinonero-Candela, and Anıbal Figueiras-Vidal. Sparse spectral sampling gaussian processes. Technical report, Technical report, Microsoft Research, 2007.

Miguel Lázaro-Gredilla, Joaquin Quiñonero-Candela, Carl Edward Rasmussen, and Aníbal R Figueiras-Vidal. Sparse spectrum Gaussian process regression. *Journal of Machine Learning Research*, 11:1865–1881, 2010.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

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

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

Wei-Han Lee, Jorge Ortiz, Bongjun Ko, and Ruby Lee. Time series segmentation through automatic feature learning. *arXiv preprint arXiv:1801.05394*, 2018.

Wu Lin, Nicolas Hubacher, and Mohammad Emtiyaz Khan. Variational message passing with structured inference networks. *arXiv preprint arXiv:1803.05589*, 2018.

Scott Linderman, Matthew J Johnson, and Ryan P Adams. Dependent multinomial models made easy: Stick-breaking with the Pólya-gamma augmentation. In *Advances in Neural Information Processing Systems*, pages 3456–3464, 2015.

Scott W Linderman, Matthew J Johnson, Matthew A Wilson, and Zhe Chen. A Bayesian nonparametric approach for uncovering rat hippocampal population codes during spatial navigation. *Journal of Neuroscience Methods*, 263:36–47, 2016.

Zachary C Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

Chuanhai Liu, Donald B Rubin, and Ying Nian Wu. Parameter expansion to accelerate EM: the PX-EM algorithm. *Biometrika*, 85(4):755–770, 1998.

Robert Lund and Jaxk Reeves. Detection of undocumented changepoints: A revision of the two-phase regression model. *Journal of Climate*, 15(17):2547–2554, 2002.

David JC MacKay. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992.

David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

Jakob H Macke, Lars Buesing, John P Cunningham, M Yu Byron, Krishna V Shenoy, and Maneesh Sahani. Empirical models of spiking in neural populations. In *Advances in Neural Information Processing Systems*, pages 1350–1358, 2011.

Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML Workshop*, volume 238, page 5, 2015.

Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens Van Der Maaten. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 181–196, 2018.

Michael W Mahoney. Lecture notes on randomized linear algebra. *arXiv preprint arXiv:1608.04481*, 2016.

Andrey A Markov. An example of statistical study on text of eugeny onegin illustrating the linking of events to a chain. *Izvestiya Akademii Nauk*, 6:153–162, 1913.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Joseph Mellor and Jonathan Shapiro. Thompson sampling in switching environments with Bayesian online change detection. In *Artificial Intelligence and Statistics*, pages 442–450. PMLR, 2013.

J Mercer. Functions ofpositive and negativetypeand theircommection with the theory ofintegral equations. *Philosophicol Trinsdictions ofthe Rogyal Society*, pages 4–415, 1909.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

Jeffrey W Miller and David B Dunson. Robust Bayesian inference via coarsening. *Journal of the American Statistical Association*, 2018.

Roger E Millsap. When trivial constraints are not trivial: The choice of uniqueness constraints in confirmatory factor analysis. *Structural Equation Modeling*, 8(1):1–17, 2001.

Thomas Minka. Bayesian linear regression. Technical report, 2000.

Tom Minka. From hidden markov models to linear dynamical systems. Technical report, Citeseer, 1999.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mohsin Munir, Shoaib Ahmed Siddiqui, Muhammad Ali Chattha, Andreas Dengel, and Sheraz Ahmed. Fusead: unsupervised anomaly detection in streaming sensors data by fusing statistical and deep learning models. *Sensors*, 19(11):2451, 2019.

Kevin P Murphy. Conjugate Bayesian analysis of the Gaussian distribution. *def*, 1($2\sigma 2$):16, 2007.

Kevin P Murphy. *Machine learning: a probabilistic perspective.* Cambridge, MA, 2012.

Elizbar A Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9 (1):141–142, 1964.

Radford M Neal. Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9(2):249–265, 2000.

Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011.

Junier B Oliva, Avinava Dubey, Andrew G Wilson, Barnabás Póczos, Jeff Schneider, and Eric P Xing. Bayesian nonparametric kernel-learning. In *Artificial Intelligence and Statistics*, pages 1078–1086, 2016.

Michael A Osborne, Roman Garnett, and Stephen J Roberts. Active data selection for sensor networks with faults and changepoints. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 533–540. IEEE, 2010.

Ewan S Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Judea Pearl. Bayesian netwcrks: A model cf self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine, CA, USA*, pages 15–17, 1985.

Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11): 559–572, 1901.

Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7(15):510, 2008.

Edwin James George Pitman. Sufficient statistics and intrinsic accuracy. In *Mathematical Proceedings of the cambridge Philosophical society*, volume 32, pages 567–579. Cambridge University Press, 1936.

Nicholas G Polson, James G Scott, and Jesse Windle. Bayesian inference for logistic models using Pólya–gamma latent variables. *Journal of the American Statistical Association*, 108 (504):1339–1349, 2013.

Harsh Purohit, Ryo Tanabe, Kenji Ichige, Takashi Endo, Yuki Nikaido, Kaori Suefusa, and Yohei Kawaguchi. Mimii dataset: Sound dataset for malfunctioning industrial machine investigation and inspection. *arXiv preprint arXiv:1909.09347*, 2019.

Joaquin Quinonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *The Journal of Machine Learning Research*, 6: 1939–1959, 2005.

Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2007.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pages 1177–1184, 2008.

Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 2. MIT Press Cambridge, MA, 2006.

Yaacov Ritov, A Raz, and H Bergman. Detection of onset of neuronal activity by allowing for heterogeneity in the change points. *Journal of neuroscience methods*, 122(1):25–42, 2002.

TD Robinson, Michael S Eldred, Karen E Willcox, and R Haimes. Surrogate-based optimization using multifidelity models with variable parameterization and corrected space mapping. *AIAA journal*, 46(11):2814–2822, 2008.

Veronika Ročková and Edward I George. Fast Bayesian factor analysis via automatic rotations to sparsity. *Journal of the American Statistical Association*, 111(516):1608–1622, 2016.

HoHo Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

Sam Roweis and Zoubin Ghahramani. A unifying review of linear gaussian models. *Neural computation*, 11(2):305–345, 1999.

Walter Rudin. *Fourier analysis on groups*, volume 121967. Wiley Online Library, 1962.

Eric Ruggieri and Marcus Antonellis. An exact approach to Bayesian sequential change point detection. *Computational Statistics & Data Analysis*, 97:71–86, 2016.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

Yunus Saatçi, Ryan D Turner, and Carl E Rasmussen. Gaussian process change point models. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 927–934. Citeseer, 2010.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A generalized representer theorem. In *International Conference on Computational Learning Theory*, pages 416–426. Springer, 2001.

Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792): 706–710, 2020.

Andrey A Shabalin. Matrix eqtl: ultra fast eqtl analysis via large matrix operations. *Bioinformatics*, 28(10):1353–1358, 2012.

Manan Shah, Dayong Wang, Christopher Rubadue, David Suster, and Andrew Beck. Deep learning assessment of tumor proliferation in breast cancer histological images. In *Bioinformatics and Biomedicine (BIBM), 2017 IEEE International Conference on*, pages 600–603. IEEE, 2017.

David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

Jun Shao. Mathematical statistics. 2003.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. *Advances in Neural Information Processing Systems*, 18:1259–1266, 2006.

John Snow. *On the mode of communication of cholera*. John Churchill, 1855.

C Spearman. General intelligence objectively determined and measured. *American Journal of Psychology*, 15:107–197, 1904.

Matthew Stephens. Dealing with label switching in mixture models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62(4):795–809, 2000.

Stephen M Stigler et al. The epic story of maximum likelihood. *Statistical Science*, 22(4): 598–620, 2007.

Vaishnavi Subramanian, Benjamin Chidester, Jian Ma, and Minh N Do. Correlating cellular features with gene expression using cca. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 805–808. IEEE, 2018.

Terence Tao. There's more to mathematics than rigour and proofs. *What's New blog. http://terrytao. wordpress. com/career-advice/theres-more-to-mathematics-than-rigour-and-proofs*, 2009.

Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Hierarchical dirichlet processes. *Journal of the american statistical association*, 101(476):1566–1581, 2006.

Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

Luke Tierney. Markov chains for exploring posterior distributions. *the Annals of Statistics*, pages 1701–1728, 1994.

Michael E Tipping and Christopher M Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

Michalis Titsias and Neil D Lawrence. Bayesian Gaussian process latent variable model. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 844–851, 2010.

Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

Ledyard R Tucker. An inter-battery method of factor analysis. *Psychometrika*, 23(2):111–136, 1958.

A. M. Turing. Computing Machinery and Intelligence. *Mind*, LIX(236):433–460, 10 1950. ISSN 0026-4423. doi: 10.1093/mind/LIX.236.433. URL https://doi.org/10.1093/mind/LIX.236.433.

Ryan Turner, Steven Bottone, and Clay Stanek. Online variational approximations to non-exponential family change point models: with application to radar tracking. In *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 1*, pages 306–314, 2013.

Robert J Vanderbei et al. *Linear programming*, volume 3. Springer, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Mauricio Villarroel, João Jorge, Chris Pugh, and Lionel Tarassenko. Non-contact vital sign monitoring in the clinic. In *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, pages 278–285. IEEE, 2017.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17 (3):261–272, 2020.

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

Stephen Walker and Nils Lid Hjort. On Bayesian consistency. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(4):811–821, 2001.

Jack M Wang, David J Fleet, and Aaron Hertzmann. Gaussian process dynamical models. In *NIPS*, volume 18, page 3. Citeseer, 2005.

Ke Alexander Wang, Geoff Pleiss, Jacob R Gardner, Stephen Tyree, Kilian Q Weinberger, and Andrew Gordon Wilson. Exact gaussian processes on a million data points. *arXiv preprint arXiv:1903.08114*, 2019.

Weiran Wang, Xinchen Yan, Honglak Lee, and Karen Livescu. Deep variational canonical correlation analysis. *arXiv preprint arXiv:1610.03454*, 2016.

Yixin Wang, Alp Kucukelbir, and David M Blei. Robust probabilistic modeling with Bayesian data reweighting. In *International Conference on Machine Learning*, pages 3646–3655. PMLR, 2017.

Geoffrey S Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372, 1964.

Max Welling. Kernel ridge regression. *Max Welling's Classnotes in Machine Learning*, pages 1–3, 2013.

Andrew Wilson and Ryan Adams. Gaussian process kernels for pattern discovery and extrapolation. In *Proceedings of International Conference on Machine Learning*, pages 1067–1075, 2013.

Robert C Wilson, Matthew R Nassar, and Joshua I Gold. Bayesian online learning of the hazard rate in change-point problems. *Neural computation*, 22(9):2452–2476, 2010.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

Anqi Wu, Nicholas A Roy, Stephen Keeley, and Jonathan W Pillow. Gaussian process based nonlinear latent structure discovery in multivariate spike train data. In *Advances in Neural Information Processing Systems*, pages 3496–3505, 2017.

CF Jeff Wu. On the convergence properties of the em algorithm. *The Annals of statistics*, 11(1):95–103, 1983.

Tianbao Yang, Yu-Feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi-Hua Zhou. Nyström method vs random Fourier features: A theoretical and empirical comparison. In *Advances in Neural Information Processing Systems*, pages 476–484, 2012.

He Zhao, Piyush Rai, Lan Du, Wray Buntine, Dinh Phung, and Mingyuan Zhou. Variational autoencoders for sparse and overdispersed discrete data. In *International Conference on Artificial Intelligence and Statistics*, pages 1684–1694. PMLR, 2020.

Mingyuan Zhou and Lawrence Carin. Augment-and-conquer negative binomial processes. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2012.

Mingyuan Zhou, Lingbo Li, David Dunson, and Lawrence Carin. Lognormal and gamma mixed negative binomial regression. In *Proceedings of International Conference on Machine Learning*, volume 2012, page 1343, 2012.