Advanced Topics in Machine Learning: Bayesian Machine Learning

Tom Rainforth

Department of Computer Science

Hilary 2020

Contents

1 Introduction

How come a dog is able to catch a frisbee in mid-air? How come a batsman can instinctively predict the flight of a cricket ball, moving at over 100km/h, sufficiently accurately and quickly to hit it when there is not even time to consciously make a prediction? Clearly, neither can be based on a deep explicit knowledge of the laws of physics or some hard-coded model for the movement of objects; we are not even born with the knowledge that unsupported objects will fall down [Baillargeon, 2002]. The only reasonable explanation for these abilities is that the batsmen and the dog have *learned from experience*. We do not have all the knowledge we require to survive from birth, but we are born with the ability to learn and adapt, making observations about the world around us and using these to refine our cognitive models for everything from the laws of physics to social interaction. Classically the scientific method has relied on human interpretation of the world to formulate explicit models to explain our internal intuitions, which we then test through experimentation. However, even as a whole scientific society, our models are often terribly inferior to the subconscious models of animals and children, such as for most tasks revolving around social interaction. This leads one to ask, is there something fundamentally wrong with this hand-crafted modeling approach? Is there another approach that better mimics the way humans themselves learn?

Machine learning is an appealing alternative, and often complementary, approach that focuses on constructing algorithms and systems that can adapt, or learn, from data in order to make predictions that have not been explicitly programmed. This is exciting not only because of the potential it brings to automate and improve a wide array of computational tasks, but because it allows us to design systems capable of going beyond the boundaries of human understanding, reasoning about and making predictions for tasks we cannot solve directly ourselves. As a field, machine learning is very wide ranging, straddling computer science, statistics, engineering, and beyond. It is perhaps most closely related to the field of computational statistics, differing predominantly in its emphasis on prediction rather than understanding. Despite the current hype around the field, most of the core ideas have existed for some time, often under the guise of pattern recognition, artificial intelligence, or computational statistics. Nonetheless, the explosion

in the availability of data and in computational processing power in recent years has led to a surge of interest in machine learning by academia and industry alike, particularly in its application to real world problems. This interest alone is enough to forgive the hype, as the spotlight is not only driving the machine learning community itself forward, but helping identify huge numbers of applications where existing techniques can be transferred to fantastic effect. From autonomous vehicles [Lefèvre et al., 2014], to speech recognition [Jurafsky and Martin, 2014], and designing new drugs [Burbidge et al., 2001], machine learning is rapidly becoming a crucial component in many technological and scientific advancements.

In many machine learning applications, it is essential to use a principled *probabilistic* approach [Ghahramani, 2015], incorporating uncertainty and utilizing all the information at hand, particularly when data is scarce. The *Bayesian paradigm* provides an excellent basis upon which to do this: an area specialist constructs a probabilistic model for data generation, conditions this on the actual observations received, and, using Bayes' rule, receives an updated model incorporating this information. This allows information from both existing expertise and data to be combined in a statistically rigorous fashion. As such, it allows us to use machine learning to complement the conventional scientific approach, rather than directly replacing it: we can construct models in a similar way to that which is already done, but then improve and refine these using data.

Unfortunately, there are two key challenges that often make it difficult for this idealized view of the Bayesian machine learning approach to be realized in practice. Firstly, a process known as *Bayesian inference* is required to solve the specified problems. This is typically a challenging task, closely related to integration, which is often computationally intensive to solve. Secondly, it can be challenging to specify models that are true to the assumptions the user wishes to make and the prior information available. In particular, if the data is complex or high-dimensional, hand-crafting such models may not be feasible, such that we should instead also look to learn the model itself in a data-driven manner.

In this course, we will cover the fundamentals of the Bayesian machine learning approach and start to make inroads into how these challenges can be overcome. We will go through how to construct models and how to run inference in them, before moving on to showing how we can instead learn the models themselves. Our finishing point will be the recently emerged field of *deep generative models* [Kingma and Welling, 2014; Rezende et al., 2014; Goodfellow et al., 2014], wherein deep learning approaches are used to learn highly complex generative models directly from data.

1.1 A Note on Advanced Sections

Some sections of these notes are quite advanced and may be difficult to completely grasp given the constraints of what can be realistically covered in the course. They are clearly marked using [Advanced Topic] and some may not be covered in the lectures themselves. Understanding them may require you do additional reading or look up certain terms not fully discussed in the course. As such, you should not feel like you need to perfectly understand them to be able to complete the coursework, but they may prove helpful in providing a complete picture and deeper appreciation of the course's content.

2

A Brief Introduction to Probability

Before going into the main content of the course, we first provide a quick primer on probability theory, outlining some essential background, terminology and conventions. This should hopefully be predominantly a recap (with the likely exception of the concept of measures), but there are many subtleties with probability that can prove important for Bayesian machine learning.

2.1 Random Variables, Outcomes, and Events

A ***random variable*** is a variable whose realization is currently unknown, such that it can take on multiple different values or ***outcomes***. A set of one or more outcomes is known as an ***event***. For example, if we roll a fair six-sided dice then the result of the roll is a random variable , while rolling a 4 is both a possible outcome and a possible event. Rolling a number greater or equal to 5, on the other hand, is a possible event but not a possible outcome: it is a set of two individual outcomes, namely rolling a 5 and rolling a 6. Outcomes are ***mutually exclusive***, that is, it is not possible for two separate outcomes to occur for a particular trial, e.g. we cannot roll both a 2 and 4 with a single throw. Events, on the other hand, are not. For example, it is possible for both the events that we roll and even number and we roll a number greater than 3 to occur.

2.2 Probabilities

A ***probability*** is the chance of an event occurring. For example, if we denote the output of our dice roll as X, then we can say that $P(X = 4) = 1/6$ or that $P(X \leq 3) = 0.5$. Here $X = 4$ and $X \leq 3$ are events for the random variable X with probabilities of $1/6$ and 0.5 respectively. A probability of 0 indicates that an event has no chance of happening, for example the probability that we roll an 8, while a probability of 1 indicates it is certain to happen, for example, the probability that we roll a positive number. All probabilities must thus lie between 0 and 1 (inclusive). The ***distribution*** of a random variable provides the probabilities of each possible outcome for that random variable occurring.

Though, we will regularly use the shorthand $P(x)$ to denote the probability of the event $P(X = x)$, we reiterate the important distinction between the random variable X and the outcome x: the former has an unknown value (e.g. the result of the dice roll) and the latter

is a fixed possible realization of the random variable (e.g. rolling a 4). All the same, in later chapters we will often carefree about delineating between random variables and outcomes for simplicity, except for when the distinction is explicitly necessary.

Somewhat surprisingly, there are two competing (and sometimes incompatible) formal interpretations of probability. The ***frequentist*** interpretation of probability is that it is the *average proportion of the time an event will occur if a trial is repeated infinitely many times*. The ***Bayesian*** interpretation of probability is that it is the *subjective belief that an event will occur in the presence of incomplete information*. Both viewpoints have strengths and weaknesses and we will avoid being drawn into one of the biggest debates in science, noting only that the philosophical differences between the two are typically completely detached from the practical differences between the resulting machine learning or statistics methods (we will return to this in the next chapter), despite these philosophical differences all too often being used to argue the superiority of the resultant algorithms [Gelman et al., 2011; Steinhardt, 2012].

2.3 Conditioning and Independence

A ***conditional probability*** is the probability of an event given that another event has occurred. For example, the conditional probability that we roll a 4 with a dice given that we have rolled a 3 or higher is $P(X = 4|X \geq 3) = 0.25$. More typically, we will condition upon events that are separate but correlated to the event we care about. For example, the probability of dying of lung cancer is higher if you smoke. The process of updating a probability using the information from another event is known as conditioning on that event. For example, one can condition the probability that a football team will win the league on the results from their first few games.

Events are ***independent*** if the occurrence of one event does not affect the probability of the occurrence of the other event. Similarly, random variables are independent if the outcome of one random variable does not affect the distribution of the other. Independence of random variables indicates the probability of each variable is the same as the conditional probability given the other variable, i.e. if X and Y are independent, $P(X = x) = P(X = x|Y = y)$ for all possible y and x. Note that independence does not necessarily carry over when adding or removing a conditioning: if X and Y are independent, this does not necessarily mean that $P(X = x|A) = P(X = x|A, Y = y)$ for some event A. For example, the probability that the next driver to pass a speed camera is speeding and that the speed camera is malfunctioning can be reasonably presumed to be independent. However, conditioned on the event that the speed camera is triggered, the two are clearly not independent: if the camera is working and

triggered, this would indicate that the driver is speeding. If $P(X = x|A) = P(X = x|A, Y = y)$ holds, then X and Y are known as conditionally independent given A. In the same way that independence does not imply conditional independence, conditional independence does not imply non-conditional independence.

2.4 The Laws of Probability

Though not technically axiomatic, the mathematical laws of probability can be summarized by the ***product rule*** and the ***sum rule***. Remarkably, almost all of Bayesian statistics stems from these two simple rules.

The product rule states that the probability of two events occurring is the probability of one of the events occurring times the conditional probability of the other event happening given the first event happened, namely

$$P(A, B) := P(A \cap B) = P(A|B)P(B) = P(B|A)P(A) \tag{2.1}$$

where we have introduced $P(A, B)$ as a shorthand for the probability that both the events A and B occur. An immediate consequence of the product rule is Bayes' rule,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{2.2}$$

which we will to return at length throughout the course. Another is that, for independent random variables, the joint distribution is the product of the individual probabilities: $P(A, B) = P(A)P(B)$.

The sum rule has a number of different representations, the most general of which is that the probability that either A or B occurs, $P(A \cup B)$, is given by

$$P(A \cup B) = P(A) + P(B) - P(A, B). \tag{2.3}$$

The intuition of the sum rule is perhaps easiest to see by considering that

$$P(B) - P(A, B) = P(B)(1 - P(A|B)) = P(B, \neg A)$$

is the probability of B and not A. Now $A \cup B$ can only occur if A occurs or if B occurs and not A. As it is not possible for both these events to occur, the probability of either event must be the sum of the probability of each separate event, leading to (2.3).

There are a number of immediate consequences of the sum rule. For example, if A and B are mutually exclusive then $P(A \cup B) = P(A) + P(B)$. As outcomes are mutually exclusive, it follows from the sum rule and the axioms of probability that the sum of the probabilities for

each possible outcome is equal to 1. We can also use this to define the concept of ***marginalizing*** out a random variable Y as

$$P(X = x) = \sum_i P(X = x, Y = y_i) \tag{2.4}$$

where the sum is over all the possible outcomes of Y. Here $P(X = x)$ is known as the ***marginal probability*** of X and $P(X = x, Y = y)$ as the ***joint probability*** of X and Y.

Conditional probabilities follow the same key results as unconditional probabilities, but it should be noted that they do not define probability distributions over the conditioning term. For example, $P(A|B)$ is a probability distribution over A with all the corresponding requirements, but is not a distribution over B. Therefore, for example, it is possible to have $\sum_i P(A|B = b_i) > 1$. We instead refer to $P(A|B)$ as the ***likelihood*** of B, given the occurrence of event A.

2.5 Probability Densities

Thus far we have presumed that our random variables are discrete, i.e. that there is some fixed number of possible outcomes.[1] Things get somewhat more complicated if our variables are continuous. Consider for example the probability that a runner takes exactly π (i.e. $3.14159265\ldots$) hours to run a marathon $P(X = \pi)$. Clearly, the probability of this particular event is zero, $P(X = \pi) = 0$, as is the probability of the runner taking any other exact time to complete the race: we have an infinite number of possible outcomes, each with zero probability (presuming the runner finishes the race). Thankfully, the notion of an event that we previously introduced comes to our rescue. For example, the event that the runner takes between 3 and 4 hours has non-zero probability: $P(3 \leq X \leq 4) \neq 0$. Here our event includes an infinite number of possible outcomes and even though each individual outcome had zero probability, the combination of *uncountably infinitely* many such outcomes need not also have zero probability.

To more usefully characterize probability in such cases, we can define a ***probability density function*** which reflects the relative probability of areas of the space of outcomes. We can informally define this by considering the probability of being in some small area of the space of size δx. Presuming that the probability density $p_X(x)$ is roughly constant within our small area, we can say in one dimension that $p_X(x)\delta x \approx P(x \leq X < x + \delta x)$, thus giving the

[1] Technically speaking, discrete random variables can also take on a *countably infinite* number of values, e.g. the Poisson distribution is defined over $0, 1, 2, \ldots, \infty$. However, this countable infinity is much smaller than the *uncountably infinite* number of possible outcomes for continuous random variables.

informal definition $p_X(x) = \lim_{\delta \to 0} \frac{P(x \leq X < x+\delta x)}{\delta x}$. More precisely, and for multiple dimensions, we can define the probability density as satisfying

$$P(X \in \mathcal{A}) = \int_{x \in \mathcal{A}} p_X(x) dx \tag{2.5}$$

where $X \in \mathcal{A}$ means the event that X is in $\mathcal{A}$. For one-dimensional variables, we can similarly define the ***cumulative distribution function*** $P(X \leq x)$, which is the probability that X is less than equal to the outcome x

$$P(X \leq x) = \int_{-\infty}^{x} p_X(u) du, \tag{2.6}$$

where u is a dummy variable. The fundamental laws of probability discussed in the last section apply equally well to probability densities, replacing summations with integrals as and when required.

In the rest of these notes, we will drop the notation $p_X(x)$, using simply $p(x)$ instead. The main rationale for this is that we will regularly use probability density functions that we do not actually sample from. For example, in importance sampling, we will sample from one distribution but evaluate its density under another. In these scenarios, it may not be possible to link a random variable to each density. We will instead make it explicit what distribution a random variable is drawn from using the notation $X \sim p(x)$. However, we will regularly be carefree about distinguishing between random variables and outcomes by using loose notations such as $x \sim p(x)$ when the delineation is not necessary in the context.

2.6 Expectations and Variances

The ***expected value*** $\mathbb{E}[X]$, or ***mean***, of a random variable X is the average value that the variable will take if an infinite number of independent draws are made. Its definition is easiest to convey using probability density notation as follows

$$\mathbb{E}[X] = \int x p(x) dx. \tag{2.7}$$

In the discrete case this leads to $\mathbb{E}[X] = \sum_i x_i P(X = x_i)$. Because expectations are defined by a random variable (rather than a density), they average over all the contained randomness, e.g. $\mathbb{E}[f(X,Y)] = \iint f(x,y)p(x,y)dxdy$. However, if we wish to average only with respect to part of the randomness in a system, we can instead use a ***conditional expectation***, for example

$$\mathbb{E}[f(X,Y)|Y=y] = \int f(x,y)p(x|y)dx, \tag{2.8}$$

for which we will sometimes use the shorthand $\mathbb{E}[f(X,Y)|Y]$. It will also sometimes be convenient to define the random variable and conditioning for an expectation at the same time, for which we use the slightly loose notation

$$\mathbb{E}_{p(x|y)}\left[f(x,y,z)\right] = \int f(x,y,z)p(x|y)dx, \tag{2.9}$$

where we have implicitly defined the random variable $X \sim p(x|Y=y)$, we are calculating $\mathbb{E}\left[f(X,Y,z)|Y=y\right]$, and the resulting expectation is a function of z (which is treated as a deterministic variable). One can informally think about this as being the expectation of $f(X,y,z)$ with respect to $X \sim p(x|y)$: i.e. our expectation is only over the randomness associated with drawing from $p(x|y)$.

Denoting the mean of a random variable X as $\mu = \mathbb{E}[X]$, the ***variance*** of X is defined using any one of the following equivalent forms (replacing integrals with sums for discrete variables)

$$\mathrm{Var}(X) = \mathbb{E}\left[(X-\mu)^2\right] = \int (x-\mu)^2 p(x)dx = \mathbb{E}[X^2] - \mu^2 = \int x^2 p(x)dx - \mu^2. \tag{2.10}$$

In other words, it is the average squared distance of a variable from its mean. Its square root, the ***standard deviation***, informally forms an estimate of the average amount of variation of the variable from its mean and has units which are the same as the data. We will use the same notational conventions as for expectations when defining variances (e.g. $\mathrm{Var}_{p(x|y)}[f(x,y,z)]$).

The variance is a particular case of the more general concept of a ***covariance*** between two random variables X and Y. Defining $\mu_X = \mathbb{E}[X]$ and $\mu_Y = \mathbb{E}[Y]$, then the covariance is defined by any one of the following equivalent forms (again replacing integrals with sums for discrete variables)

$$\begin{aligned}\mathrm{Cov}(X,Y) &= \mathbb{E}\left[(X-\mu_X)(Y-\mu_Y)\right] = \iint (x-\mu_X)(y-\mu_Y)p(x,y)dxdy \\ &= \mathbb{E}\left[XY\right] - \mathbb{E}\left[X\right]\mathbb{E}\left[Y\right] = \iint xyp(x,y)dxdy - \left(\int xp(x)dx\right)\left(\int yp(y)dy\right).\end{aligned} \tag{2.11}$$

The covariance between two variables measures the joint variability of two random variables. It is perhaps easiest to interpret through the definition of correlation (or more specifically, Pearson's correlation coefficient) which is the correlation scaled by the standard deviation of each of the variables

$$\mathrm{Corr}(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}}. \tag{2.12}$$

The correlation between two variables is always in the range $[-1,1]$. Positive correlations indicate that when one variable is relatively larger, the other variable also tends to be larger. The higher the correlation, the more strongly this relationship holds: if the correlation is 1 then one variable

is ***linearly*** dependent on the other. The same holds for negative correlations except that when one variable increases, the other tends to decrease. Independent variables have a correlation (and thus a covariance) of zero, though the reciprocal is not necessarily true: variables with zero correlation need not be independent. Note that correlation is not causation.

2.7 Measures [Advanced Topic]

Returning to our marathon runner example from Section 2.5, consider now if there is also a probability that the runner does not finish the race, which we denote as the outcome $X = \text{DNF}$. As we have thus-far introduced them, neither the concept of a probability or a probability density seem to be suitable for this case: every outcome other than $X = \text{DNF}$ has zero probability, but $X = \text{DNF}$ seems to have infinite probability density.

To solve this conundrum we have to introduce the concept of a ***measure***. A measure can be thought of as something that assigns a size to a set of objects. ***Probability measures*** assign probabilities to events, remembering that events represent sets of outcomes, and thus are used to define a more formal notion of probability than we have previously discussed. The measure assigned to an event including all possible outcomes is thus 1, while the measure assigned to the empty set is 0.

We can generalize the concept of a probability density to arbitrary random variables by formalizing its definition as being with respect to an appropriate ***reference measure***. Somewhat confusingly, this reference measure is typically not a probability measure. Consider the case of the continuous densities examined in Section 2.5. Here we have implicitly used the ***Lebesgue measure*** as the reference measure, which corresponds to the standard Euclidean notion of size, coinciding with the concepts of length, area, and volume in 1, 2, and 3 dimensions respectively. In (2.5) then dx indicated integration with respect to a Lebesgue measure, with $\int_{x\in\mathcal{A}} dx$ being equal to the hypervolume of $\mathcal{A}$ (e.g. area of $\mathcal{A}$ in two dimensions). Our reference measure is, therefore, clearly not a probability measure as $\int_{x\in\mathbb{R}} dx = \infty$. Our probability measure here can be informally thought of as $p(x)dx$, so that $\int_{x\in\mathcal{A}} p(x)dx = P(x \in \mathcal{A})$.[2]

In the discrete case, we can define a probability density $p(x) = P(X = x)$ by using the notion of a ***counting measure*** for reference, which simply counts the number of outcomes which lead to a particular event.

[2]More formally, the density is derived from the probability measure and reference measure rather than the other way around: it is the Radon-Nikodym derivative of the probability measure with respect to the reference measure.

Note that we were not free to choose any arbitrary measure for any given random variable. We cannot use a counting measure as reference for continuous random variables or the Lebesgue measure for discrete random variables because, for example, the Lebesgue measure would assign zero measure to all possible events for the latter. In principle, the reference measure we use is not necessarily unique either (see below), but in practice it is rare we need to venture beyond the standard Lebesgue and counting measures. For notional convenience, we will refer to dx (or equivalent) as our reference measure elsewhere in the notes.

Returning to the example where the runner might not finish, we can now solve our problem by using a ***mixed measure***. Perhaps the easiest way to think about this is to think about the runner's time X as being generated through the following process:

Sample a discrete random variable $Y \in \{0, 1\}$ that dictates if the runner finishes
if $Y = 0$ **then**
    $X \leftarrow$ DNF
else
    Sample X conditioned on the runner finishing the race
end if

We can now define the probability of the event $X \in \mathcal{A}$ by marginalizing over Y:

$$\begin{aligned} P(X \in \mathcal{A}) &= \sum_{y \in \{0,1\}} P(X \in \mathcal{A}, Y = y) \\ &= P(X \in \mathcal{A}|Y = 0)P(Y = 0) + P(X \in \mathcal{A}|Y = 1)P(Y = 1). \end{aligned}$$

Here neither $P(X \in \mathcal{A}|Y = 0)$ nor $P(X \in \mathcal{A}|Y = 1)$ is problematic. Specifically, we have $P(X \in \mathcal{A}|Y = 0) = \mathbb{I}(\text{DNF} \in \mathcal{A})$, while $P(X \in \mathcal{A}|Y = 1)$ can be straightforwardly defined as the integral of a density defined with respect to a Lebesgue reference measure, namely

$$P(X \in \mathcal{A}|Y = 1) = \int_{x \in \mathcal{A}} p(x|Y = 1)dx$$

where dx is the Lebesgue measure.

By definition of a probability density, we also have that

$$P(X \in \mathcal{A}) = \int_{x \in \mathcal{A}} p(x)d\mu(x)$$

for density $p(x)$ with respect to measure $d\mu(x)$, where we switched notation from dx to $d\mu(x)$ to express the fact that the measure now depends explicitly on the value of x, i.e. our measure is non-stationary in x. This is often referred to as a ***mixture measure***. For $x \neq$ DNF then it is natural for $d\mu(x)$ to correspond to the Lebesgue measure as above. For $x =$ DNF it is natural for $d\mu(x)$ to correspond to counting measure.

Note though that care must be taken if optimizing a density when this is defined with respect to a mixed measure. In the above example, one could easily have that $\arg\max_x p(x) \neq$ DNF which could quickly lead to confusion given that $X =$ DNF is infinitely times more probable than any other X.

2.8 Change of Variables

We finish the chapter by considering the relationship between random variables which are deterministic functions of one another. This important case is known as a ***change of variables***. Imagine that a random variable $Y = g(X)$ is a deterministic and invertible function of another random variable X. Given a probability density function for X, we can define a probability density function on Y using

$$p(y)dy = p(x)dx = p(g^{-1}(y))dg^{-1}(y) \tag{2.13}$$

where $p(y)$ and $p(x)$ are the respective probability densities for Y and X with measures dx and dy. Here dy is known as a ***push-forward*** measure of dx. Rearranging we see that, for one-dimensional problems,

$$p(y) = \left|\frac{dg^{-1}(y))}{dy}\right| p(g^{-1}(y)). \tag{2.14}$$

For the multidimensional case, the derivative is replaced by the determinant of the Jacobian for the inverse mapping.

Note that by (2.5), changing variables does not change the value of actual probabilities or expectations (see Section 2.6). This is known as the ***law of the unconscious statistician*** and it effectively means that both the probability of an event and the expectation of a function, do not depend on how we parameterize the problem. More formally, if $Y = g(X)$ where g is a deterministic function (which need not be invertible), then

$$\mathbb{E}[Y] = \int yp(y)dy = \int g(x)p(x)dx = \mathbb{E}[g(X)], \tag{2.15}$$

such that we do not need to know $p(y)$ to calculate the expectation of Y: we can take the expectation with respect to X instead and use $p(x)$.

However, (2.13) still has the important consequence that the optimum of a probability distribution depends on the parameterization, i.e., in general,

$$x^* = \arg\max_x p(x) \neq g^{-1}\left(\arg\max_{g(x)} p(g(x))\right). \tag{2.16}$$

For example, parameterizing a problem as either X or $\log X$ will lead to a different x^*.

3
Machine Learning Paradigms

In this chapter, we will provide a high-level introduction to some of the core approaches to machine learning. We will discuss the most common ways in which data is used, such as supervised and unsupervised learning. We will distinguish between discriminative and generative approaches, outlining some of the key features that indicate when problems are more suited to one approach or the other. Our attention then settles on probabilistic generative approaches, which will be the main focus of the course. We will explain how the ***Bayesian paradigm*** provides a powerful framework for generative machine learning that allows us to combine data with existing expertise. We continue by introducing the main counterpart to the Bayesian approach—***frequentist*** approaches—and present arguments for why neither alone provides the full story. In particular, we will outline the fundamental underlying assumptions made by each approach and explain why the differing suitability of these assumptions to different tasks means that both are essential tools in the machine learning arsenal, with many problems requiring both Bayesian and frequentist elements in their analysis. We finish the chapter by discussing some of the key practical challenges for Bayesian modeling.

3.1 Learning From Data

Machine learning is all about learning from ***data***. In particular, we typically want to use the data to learn something that will allow us to make ***predictions*** at unseen datapoints. This emphasis on prediction is what separates machine learning from the field of ***computational statistics***, where the aim is typically more to learn about parameters of interest. Inevitably though, the line between these two is often blurry, and will be particular so for this Bayesian machine learning course. Like with most fields, the term machine learning does not have a precise infallible definition, it is more a continuum of ideas spanning a wide area of statistics, computer science, engineering, applications, and beyond.

With some notable exceptions that we will discuss later, the starting point for most machine learning algorithms is a ***dataset***. Most machine learning methods can be delineated based on the type of dataset they work with, and so we will first introduce some of the most common types of categorization. Note that these are not exhaustive.

3.1.1 Supervised Learning

Supervised learning is arguably the most natural and common machine learning setting. In supervised learning, our aim is to learn a ***predictive model*** f that takes an input $x \in \mathcal{X}$ and aims to predict its corresponding output $y \in \mathcal{Y}$. Learning f is done by using a ***training dataset*** $\mathcal{D}$ that is comprised of a set of input–output pairs: $\mathcal{D} = \{x_n, y_n\}_{n=1}^N$. This is sometimes referred to as ***labeled data***. The hope is that these example pairs can be used to "teach" f how to accurately make predictions.

The two most common types of supervised learning are ***regression*** and ***classification***. In regression, the outputs we are trying to predict are numerical, such that $\mathcal{Y} \subseteq \mathbb{R}$ (or, in the case of multi-variate regression, $\mathcal{Y} \subseteq \mathbb{R}^d$ for some $d \in \mathbb{N}^+$). Common examples of regression problems include curve fitting and many empirical scientific prediction models; example regression methods include linear regression, Gaussian processes, and deep learning. In classification, the outputs are categorical variables, such that $\mathcal{Y}$ is some discrete (and typically non-ordinal) set of possible output values. Common example classification problems include image classification and medical diagnosis; example classification methods include random forests, support vector machines, and deep learning. Note that many supervised machine learning methods are capable of handling both regression and classification.

3.1.2 Unsupervised Learning

Unlike supervised learning, ***unsupervised learning*** methods do not have a single unified predictive goal. They are generally categorized by the data having no clear output variable that we are attempting to predict, such that we just have a collection of example datapoints rather than explicit input–output pairs, i.e. $\mathcal{D} = \{x_n\}_{n=1}^N$. This is sometimes referred to as ***unlabeled data***.

In general, unsupervised learning methods look to exact some salient features for the dataset, such as underlying structure, patterns, or characteristics. They are also sometimes used to simplify datasets so that they can be more easily interacted with by humans or other algorithms. Common types of unsupervised learning include clustering, feature extraction, density estimation, representation learning, data visualization, data mining, data compression, and some model learning. A host of different methods are used to accomplish these tasks, with approaches that leverage deep learning becoming increasingly prominent.

3.1.3 Semi–Supervised Learning

As the name suggests, ***semi–supervised learning*** is somewhere in–between supervised and unsupervised learning. It is generally characterized by the presence of a dataset where not all the inputs variables have corresponding output; i.e. only some of a datapoints are *labeled*. In particular, many semi-supervised approaches focus on cases where there is a large amount of unlabeled data available, but only a small amount of labeled data (in cases were only a small number of labels are missing, one often just uses a suitable supervised algorithm instead).

The aim of semi–supervised learning can depend on the context. In many, and arguably most, cases, the aim is to use the unlabeled data to assist in learning a predictive model, e.g. through uncovering structure or patterns that aid in training the model or help to better generalize to unseen inputs. However, there are also some cases where the labels are instead used to try and help with more unsupervised–learning–orientated tasks, such as learning features with strong predictive power, or performing representation learning in a manner that emphasizes structure associated with the output labels.

3.1.4 Notable Exceptions

There are also a number of interesting machine learning settings where one does not start with a dataset, but must either gather the data during the learning process, or even simulate our own pseudo data.

Perhaps the most prominent example of this is ***reinforcement learning***. In reinforcement learning, one must "learn on the fly": there is an agent which must attempt an action or series of actions before receiving a reward for those choices. The agent then learns from these rewards to improve its actions over time, with the ultimate aim being to maximize the long-term reward (which can be either cumulative or instantaneous). Reinforcement learning still uses data through its updating based on previous rewards, but it must also learn how to collect that data as well, leading to quite distinct algorithms. It is often used in settings where an agent must interact with the physical world (e.g. self-driving cars) or a simulator (e.g. training AIs for games).

Other examples of machine learning frameworks that do not fit well into any of the aforementioned categories include experimental design, active learning, meta-learning, and collaborative filtering.

3.2 Discriminative vs Generative Machine Learning

3.2.1 Discriminative Machine Learning

In some machine learning applications, huge quantities of data are available that dwarf the information that can be provided from human expertise. In such situations, the main challenge is in processing and extracting all the desired information from the data to form a useful characterization, typically an artifact providing accurate predictions at previous unseen inputs. Such problems are typically suited to ***discriminative machine learning*** approaches [Breiman et al., 2001; Vapnik, 1998], such as neural networks [Rumelhart et al., 1986; Bishop, 1995], support vector machines [Cortes and Vapnik, 1995; Schölkopf and Smola, 2002], and decision tree ensembles [Breiman, 2001; Rainforth and Wood, 2015].

Discriminative machine learning approaches are predominantly used for supervised learning tasks. They focus on directly learning a predictive model: given training data $\mathcal{D} = \{x_n, y_n\}_{n=1}^N$, they learn a parametrized mapping f_θ from the inputs $x \in \mathcal{X}$ to the outputs $y \in \mathcal{Y}$ that can be used directly to make predictions for new inputs $x \notin \{x_n\}_{n=1}^N$. ***Training*** uses the data $\mathcal{D}$ to estimate optimal values of the parameters θ^*. ***Prediction*** at a new input x involves applying the mapping with an estimate of the optimal parameters $\hat{\theta}$ giving an estimate for the output $\hat{y} = f_{\hat{\theta}}(x)$. Some methods may also return additional prediction information instead of just the output itself. For example, in a classification task, we might predict the probability of each class, rather than just the class.

Perhaps the simplest example of discriminative learning is linear regression: one finds the hyperplane that best represents the data and then uses this hyperplane to interpolate or extrapolate to previously unseen points. As a more advanced example, in a neural network one uses training to learn the weights of the network, after which prediction can be done by running the network forwards.

There are many intuitive reasons to take a discriminative machine learning approach. Perhaps the most compelling is the idea that if our objective is prediction, then it is simplest to solve that problem directly, rather than try and solve some more general problem such as learning an underlying generative process [Vapnik, 1998; Breiman et al., 2001]. Furthermore, if sufficient data is provided, discriminant approaches can be spectacularly successful in terms of predictive performance. Discriminant methods are typically highly flexible and can capture intricate structure in the data that would be hard, or even impossible, to establish manually. Many

approaches can also be run with little or no input on behalf of the user, delivering state-of-the-art performance when used "out-of-the-box" with default parameters.

However, this black-box nature is also often their downfall. Discriminative methods typically make such weak assumptions about the underlying process that is difficult to impart prior knowledge or domain-specific expertise. This can be disastrous if insufficient data is available, as the data alone is unlikely to possess the required information to make adequate predictions. Even when substantial data is available, there may be significant prior information available that needs to be exploited for effective performance. For example, in time series modeling the sequential nature of the data is critically important information [Liu and Chen, 1998].

The difficultly in incorporating assumptions about the underlying process varies between different discriminative approaches. Much of the success of neural networks (i.e. deep learning) stems from the fact that they still provide a relatively large amount of flexibility to adapt the framework to a particular task, e.g. through the choice of architecture. At the other extreme, random forest approaches provide very little flexibility to adjust the approach to a particular task, but are still often the best performing approaches when we require a fully black-box algorithm that requires no human tuning to the specific task [Rainforth and Wood, 2015].

Not only does the black-box nature of many discriminative methods restrict the level of human input that can be imparted on the system, it often restricts the amount of insight and information that can be extracted *from* the system once trained. The parameters in most discriminative algorithms do not have physical meaning that can be queried by a user, making their operation difficult to interpret and hampering the process of improving the system through manual revision of the algorithm. Furthermore, this typically makes them inappropriate for more statistics orientated tasks, where it is the parameters themselves which are of interest, rather than the ability for the system itself to make predictions. For example, the parameters may have real-world physical interpretations which we wish to learn about.

Most discriminative methods also do not naturally provide realistic uncertainty estimates. Though many methods can produce uncertainty estimates either as a by-product or from a post-processing step, these are typically heuristic based, rather than stemming naturally from a statistically principled estimate of the target uncertainty distribution. A lack of reliable uncertainty estimates can lead to overconfidence and can make certain discriminative methods inappropriate in many scenarios, e.g. for any application where there are safety concerns. It can also reduce the composability of a methods within larger systems, as information is lost when only providing a point estimate.

3.2.2 Generative Machine Learning

These shortfalls with discriminative machine learning approaches mean that many tasks instead call for a ***generative machine learning*** approach [Ng and Jordan, 2002; Bishop, 2006]. Rather than directly learning a predictor, generative methods look to explain the observed data using a ***probabilistic model***. Whereas discriminative approaches aim only to make predictions, generative approaches model how the data is actually generated: they model the joint probability $p(X, Y)$ of the inputs X and outputs Y. By comparison, we can think of discriminative approaches as only modeling the outputs given the inputs $Y|X$. For this reason, most techniques for unsupervised learning are based on a generative approach, as here we have no explicit outputs.

Because they construct a joint probability model, generative approaches generally make stronger modeling assumptions about the problem than discriminative approaches. Though this can be problematic when the model assumptions are wrong and is often unnecessary in the limit of large data, it is essential for combining prior information with data and therefore for constructing systems that exploit application-specific expertise. In the eternal words of George Box [Box, 1979; Box et al., 1979],

> *All models are wrong, but some are useful.*

In a way, this is a self-fulfilling statement: a model for any real phenomena is, by definition, an approximation and so is never exactly correct, no matter how powerful. However, it is still an essential point that is all too often forgotten, particularly by academics trying to convince the world that only their approach is correct. Only in artificial situations can we construct exact models and so we must remember, particularly in generative machine learning, that the first, and often largest, error is in our original mathematical abstraction of the problem. On the other hand, real situations have access to finite and often highly restricted data, so it is equally preposterous to suggest that a method is superior simply due to better asymptotic behavior in the limit of large data, or that if our approach does not work then the solution always just to get more data.[1] As such, the ease of which domain-specific expertise can be included in generative approaches is often essential to achieving effective performance on real-world tasks.

To highlight the difference between discriminative and generative machine learning, we consider the example of the differences between logistic regression (a discriminative classifier) and naïve Bayes (a generative classifier). We will consider the binary classification case for

[1] It should, of course, be noted that the availability of data is typically the biggest bottleneck in machine learning: performance between machine learning approaches is often, if not usually, dominated by variations in the inherently difficulty of the problem, which is itself not usually known up front, rather than differences between approaches.

simplicity. Logistic regression is a linear classification method where the class label $y \in \{-1, +1\}$ is predicted from the input features $x \in \mathbb{R}^D$ using

$$p_{a,b}(y|x) = \frac{1}{1 + \exp(-y(a + b^T x))}, \tag{3.1}$$

and where $a \in \mathbb{R}^D$ and $b \in \mathbb{R}^D$ are the parameters of the model. The model is trained by finding the values for a and b that minimize a loss function on the training data. For example, a common approach is to find the *most likely* parameters a^* and b^* by minimizing cross-entropy loss function

$$\{a^*, b^*\} = \underset{a \in \mathbb{R}^D, b \in \mathbb{R}^D}{\arg\min} - \sum_{n=1}^{N} \log\left(p_{a,b}(y_n|x_n)\right). \tag{3.2}$$

Once found, a^* and b^* can be used with (3.1) to make predictions at any possible x. Logistic regression is a discriminative approach as we have directly calculated a characterization for the predictive distribution, rather than constructing a joint distribution on the inputs and outputs.

The naïve Bayes classifier, on the other hand, constructs a generative mode for the data. Namely it presumes that each data point is generated by sampling a class label $y_n \sim p_\psi(y)$ and then sampling the features given the class label $x_n \sim p_\phi(x|y_n)$. Here the so-called naïve Bayes assumption is that different data points are generated independently given the class label, namely

$$p_{\psi,\phi}(y_{1:N}|x_{1:N}) \propto p_\psi(y_{1:N}) \prod_{n=1}^{N} p_\phi(x_n|y_n). \tag{3.3}$$

We are free to choose the form for both $p_\psi(x|y)$ and $p_\phi(y)$ and we will use the data to learn their parameters ψ and ϕ. For example, we could take a maximum likelihood approach by calculating[2]

$$\{\psi^*, \phi^*\} = \underset{\psi,\phi}{\arg\max}\, p_{\psi,\phi}(y_{1:N}|x_{1:N}) = \underset{\psi,\phi}{\arg\max}\, p_\psi(y_{1:N}) \prod_{n=1}^{N} p_\phi(x_n|y_n) \tag{3.4}$$

and then using these parameters to make predictions $\tilde{y}$ at a given input $\tilde{x}$ at test time as follows

$$p_{\psi^*,\phi^*}(\tilde{y}|\tilde{x}) \propto p_{\psi^*}(\tilde{y}) p_{\phi^*}(\tilde{x}|\tilde{y}). \tag{3.5}$$

The freedom to choose the form for $p_\psi(x|y)$ and $p_\phi(y)$ is both a blessing and a curse of this generative approach: it allows us to impart our own knowledge about the problem on the model, but we may be forced to make assumptions without proper justification in the interest of tractability, for convenience, in error, or simply because it is challenging to specify a sufficiently general purpose model that can cover all possible cases. Further, even after the forms of $p_\phi(x|y)$ and $p_\psi(y)$ have been defined, there are still decisions to be made: do we take a Bayesian or frequentist approach for making predictions? What is the best way to calculate the information required to make predictions? We will go into these questions in more depth in Section 3.4.

[2]Note that the name naïve Bayes as potentially misleading here as we are not taking a fully Bayesian approach.

As we have shown, generative approaches are inherently probabilistic. This is highly convenient when it comes to calculating uncertainty estimates or gaining insight from our trained model. They are generally more intuitive than discriminative methods, as, in essence, they constitute an explanation for how the data is generated. As such, the parameters tend to have physical interpretation in the generative process and therefore provide not only prediction but also insight. Generative approaches will not always be preferable, particularly when there is an abundance of data available, but they provide a very powerful framework that is essential in many scenarios. Perhaps their greatest strength is in allowing the use of so-called Bayesian approaches, which we now introduce.

3.3 The Bayesian Paradigm

At its core, the Bayesian paradigm is simple, intuitive, and compelling: for any task involving learning from data, we start with some prior knowledge and then update that knowledge to incorporate information from the data. This process is known as ***Bayesian inference***. To give an example, consider the process of interviewing candidates for a job. Before we interview each candidate, we have some intuitions about how successful they will be in the advertised role, e.g. from their application form. During the interview we receive more information about their potential competency and we combine this with our existing intuitions to get an updated belief of how successful they will be.

To be more precise, imagine we are trying to reason about some variables or parameters θ. We can encode our initial belief as probabilities for different possible instances of θ, this is known as a ***prior*** $p(\theta)$. Given observed data $\mathcal{D}$, we can characterize how likely different values of θ are to have given rise to that data using a ***likelihood function*** $p(\mathcal{D}|\theta)$. These can then be combined to give a ***posterior***, $p(\theta|\mathcal{D})$, that represents our updated belief about θ once the information from the data has been incorporated by using ***Bayes' rule***:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta)p(\theta)d\theta} = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}. \tag{3.6}$$

Here the denominator, $p(\mathcal{D})$, is a normalization constant known as the ***marginal likelihood*** or ***model evidence*** and is necessary to ensure $p(\theta|\mathcal{D})$ is a valid probability distribution (or probability density for continuous problems). One can, therefore, think of Bayes' rule in the even simpler form of the posterior being proportional to the prior times the likelihood:

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta). \tag{3.7}$$

For such a fundamental theorem, Bayes' rule has a remarkably simple derivation, following directly from the product rule of probability as we showed in Chapter 2.

Another way of thinking about the Bayesian paradigm is in terms of constructing a generative model corresponding to the joint distribution on parameters and possible data $p(\theta, \mathcal{D})$, then ***conditioning*** that model by fixing $\mathcal{D}$ to the data we actually observe. This provides an updated distribution on the parameters $p(\theta|\mathcal{D})$ that incorporates the information from the data.

A key feature of Bayes' rule is that it can be used in a self-similar fashion where the posterior from one task becomes the prior when the model is updated with more data, i.e.

$$p(\theta|\mathcal{D}_1, \mathcal{D}_2) = \frac{p(\mathcal{D}_2|\theta, \mathcal{D}_1)p(\theta|\mathcal{D}_1)}{p(\mathcal{D}_2|\mathcal{D}_1)} = \frac{p(\mathcal{D}_2|\theta, \mathcal{D}_1)p(\mathcal{D}_1|\theta)p(\theta)}{p(\mathcal{D}_2|\mathcal{D}_1)p(\mathcal{D}_1)}. \tag{3.8}$$

As a consequence, there is something quintessentially human about the Bayesian paradigm: we learn from our experiences by updating our beliefs after making observations. Our model of the world is constantly evolving with time and is the cumulation of experiences over a lifetime. If we make an observation that goes against our prior experience, we do not suddenly make drastic changes to our underlying belief,[3] but if we see multiple corroborating observations our view will change. Furthermore, once we have developed a strong prior belief about something, we can take substantial convincing to change our mind, even if that prior belief is highly illogical.

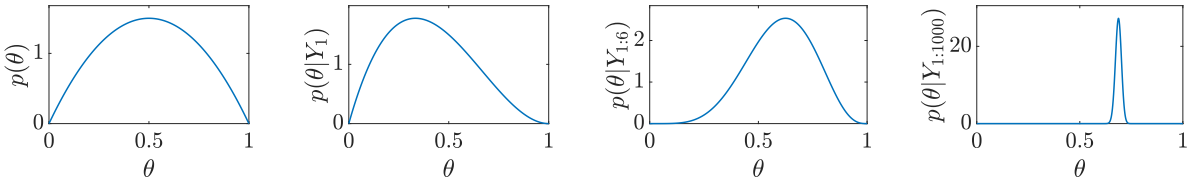
3.3.1 Worked Example: Predicting the Outcome of a Weighted Coin

To give a concrete example of a Bayesian analysis, consider estimating the probability of getting a heads from a weighted coin. Let's call this weighting $\theta \in [0, 1]$ such that the probability of getting a heads (H) when flipping the coin is $p(y = H|\theta) = \theta$ where y is the outcome of the flip. This will be our likelihood function, corresponding to a *Bernoulli* distribution, noting that the probability of getting a tails (T) is $p(y = T|\theta) = 1 - \theta$. Before seeing the coin being flipped we have some prior belief about its weighting. We can, therefore, define a prior $p(\theta)$, for which we will take the beta distribution

$$p(\theta) = \textsc{Beta}\,(\theta; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1 - \theta)^{\beta-1} \tag{3.9}$$

where $\Gamma(\cdot)$ is the gamma function and we will set $\alpha = \beta = 2$. A plot for this prior is shown in Figure 3.1a where we see that under our prior then it is more probable that θ is close to 0.5 than the extremes 0 and 1.

[3] This is not always quite true: as probabilities are multiplicative then a particularly unexpected observation can still drastically change our distribution.



(a) Prior **(b)** Posterior 1 flip **(c)** Posterior 6 flips **(d)** Posterior 1000 flips

Figure 3.1: Prior and posteriors for coin flip example after different numbers of observations.

We now flip the coin and get a tails (T). We can calculate the posterior using Bayes' rule

$$p(\theta|y_1 = T) = \frac{p(\theta)p(y_1 = T|\theta)}{\int p(\theta)p(y_1 = T|\theta)d\theta} = \frac{\theta(1-\theta)^2}{\int \theta(1-\theta)^2 d\theta} = \text{BETA}\left(\theta; 2, 3\right). \tag{3.10}$$

Here we have used the fact that a Beta prior is ***conjugate*** to a Bernoulli likelihood to give an analytic solution. Conjugacy means that the prior-likelihood combination gives a posterior that is of the same form as the prior distribution. More generally, for a prior of $\text{BETA}(\theta; \alpha, \beta)$ then the posterior will be $\text{BETA}(\theta; \alpha + 1, \beta)$ if we observe a heads and $\text{BETA}(\theta; \alpha, \beta + 1)$ if we observe a tails. Figure 3.1b shows that our posterior incorporates the information from the prior and the observed data. For example, our observation means that it becomes more probable that $\theta < 0.5$. The posterior also reflects the fact that we are still uncertain about the value of θ, it is not simply the empirical average of our observations which would give $\theta = 0$.

If we now flip the coin again, our previous posterior (3.10) becomes our prior and we can incorporate the new observations in the same way. Through our previous conjugacy result, then if we observe n_H heads and n_T tails and our prior is $\text{BETA}(\theta; \alpha, \beta)$ then our posterior is $\text{BETA}(\theta; \alpha + n_H, \beta + n_T)$. Thus if our sequence of new observations is $HTHHH$ then our new posterior is

$$p(\theta|y_1, \ldots, y_6) = \frac{p(y_2, \ldots, y_6|\theta)p(\theta|y_1)}{\int p(y_2, \ldots, y_6|\theta)p(\theta|y_1)d\theta} = \text{BETA}\left(\theta; 6, 4\right), \tag{3.11}$$

which is shown in Figure 3.1c. We see now that our belief for the probability of heads has shifted higher and that the uncertainty has reduced because of the increased number of observations. After seeing a total of 1000 observations as shown in Figure 3.1d, we find that the posterior has predominantly collapsed down to a small range of θ. We will return to how to use this posterior to make predictions at the end of the next section.

3.3.2 Using the Posterior

In some cases, the posterior is all we care about. For example, in many statistical applications θ is some physical parameter of interest and our core aim is to learn about it. Often though, the posterior will be a stepping stone to some ultimate task of interest.

One common task is making decisions; the Bayesian paradigm is rooted in decision theory. It is effectively the language of ***epistemic uncertainty***—that is uncertainty originating from

lack of information. As such, we can use it as a basis for making decisions in the presence of incomplete information. As we will show in Section 3.4, in a Bayesian decision framework we first calculate the posterior and then use this to make a decision by choosing the decision which has the lowest expected loss under this posterior.

Another common task, particularly in the context of Bayesian machine learning, is to use the posterior to make predictions for unseen data. For this, we use the so-called ***posterior predictive distribution***. Denoting the new data as $\mathcal{D}^*$, this is calculated by first introducing a predictive model for new data given θ, $p(\mathcal{D}^*|\theta, \mathcal{D})$, then taking the expectation of this over possible parameter values as dictated by posterior as follows

$$p(\mathcal{D}^*|\mathcal{D}) = \int p(\mathcal{D}^*, \theta|\mathcal{D})d\theta = \int p(\mathcal{D}^*|\theta, \mathcal{D})p(\theta|\mathcal{D})d\theta = \mathbb{E}_{p(\theta|\mathcal{D})}[p(\mathcal{D}^*|\theta, \mathcal{D})]. \tag{3.12}$$

Though the exact form of $p(\mathcal{D}^*|\theta, \mathcal{D})$ can vary depending on the context, it is equivalent to a likelihood term for the new data: if we were to observe $\mathcal{D}^*$ rather than predicting it, this is exactly the likelihood term we would use to update our posterior as per (3.8).

We further note that it is typical to assume that $p(\mathcal{D}^*|\theta, \mathcal{D}) = p(\mathcal{D}^*|\theta)$, such that data is assumed to be independent given θ. As we discuss in the next chapter, there are strong theoretical and practical motivations for this assumption, but it is important to appreciate that it is an assumption none the less: it effectively equates to assuming that all the information we want to use for predicting from our model is encapsulated in θ.

Returning to our coin flipping example, we can now make predictions using the posterior predictive distribution as follows

$$\begin{aligned} p(y_{N+1} = H|y_{1:N}) &= \int p(y_{N+1} = H, \theta|y_{1:N})d\theta = \int p(y_{N+1} = H|\theta)p(\theta|y_{1:N})d\theta \\ &= \int \theta \ \textsc{Beta}(\theta; \alpha + n_H, \beta + n_T)d\theta = \frac{\alpha + n_H}{\alpha + n_H + \beta + n_T} \end{aligned} \tag{3.13}$$

where we have used the known result for the mean of the Beta distribution. The role of the parameters α and β in our prior now become apparent – they take on the role of pseudo-observations. Our prediction is in line with the empirical average from seeing $\alpha + n_H$ heads and $\beta + n_T$ tails. The larger $\alpha + \beta$ is then the strong our prior compared to the observations, while we can skew towards heads or tails being more likely by changing the relative values of α and β.

3.4 Bayesianism vs Frequentism [Advanced Topic]

We have just introduced the Bayesian approach to generative modeling, but this is far from the only possible approach. In this section, we will briefly introduce and compare the alternative,

frequentist, approach. The actual statistical differences between the approaches are somewhat distinct from the more well-known philosophical differences we touched on in Section 2.2, even though the latter are often dubiously used for justification for the practical application of a particular approach. We note that whereas Bayesian methods are always, at least in theory, generative [Gelman et al., 2014, Section 14.1], frequentist methods can be either generative or discriminative.[4] As we have already discussed differences between generative and discriminative modeling in Section 3.2, we will mostly omit this difference from our subsequent discussion.

At their root, the statistical differences between Bayesian and frequentist methods[5] stem from distinct fundamental assumptions: frequentist modeling presumes fixed model parameters, Bayesian modeling assumes fixed data [Jordan, 2009]. In many ways, both of these assumptions are somewhat dubious. Why assume fixed parameters when we do not have enough information from the data to be certain of the correct value? Why ignore that fact that other data could have been generated by the same underlying true parameters? However, making such assumptions can sometimes be unavoidable for carrying out particular analyses.

To elucidate the different assumptions further and start looking into why they are made, we will now step into a decision-theoretic framework. Let's presume that the universe gives us some data $\mathcal{D}$ and some true parameter θ, the former of which we can access, but the latter of which is unknown. We can alternatively think in terms of $\mathcal{D}$ being some information that we actually receive and θ being some underlying truth or oracle from which we could make optimal predictions, noting that there is no need for θ to be some explicit finite parameterization. Any machine learning approach will take the data as input and return some artifact or decision, for example, predictions for previously unseen inputs. Let's call this process the decision rule d, which we presume, for the sake of argument, to be deterministic for a given dataset, producing decisions $d(\mathcal{D})$.[6] Presuming that our analysis is not frivolous, there will be some loss function $L(d(\mathcal{D}), \theta)$ associated with the action we take and the true parameter θ, even if this loss function is subjective or unknown. At a high level, our aim is always to minimize this loss, but what we mean by minimizing the loss changes between the Bayesian and frequentist settings.

[4]Note that this does not mean that we cannot be Bayesian *about* a discriminative model, e.g. a Bayesian neural network. Doing this though requires us to write a prior over the model parameters and thus requires us to convert the model it something which is (partially) generative.

[5]At least in their decision-theoretic frameworks. It is somewhat inevitable that delineation here and later will be a simplification on what is, in truth, not a clear-cut divide [Gelman et al., 2011].

[6]If we allow our predictions to be probability distributions this assumption is effectively equivalent to assuming we can solve any analysis required by our approach exactly.

In the frequentist setting, $\mathcal{D}$ is a random variable and θ is unknown but not a random variable. Therefore, one takes an expectation over possible data that could have been generated, giving the ***frequentist risk*** [Vapnik, 1998]

$$R(\theta, d) = \mathbb{E}_{p(\mathcal{D})}\left[L(d(\mathcal{D}), \theta)\right] \tag{3.14}$$

which is thus a function of θ and our decision rule. The frequentist focus is on *repeatability* (i.e. the generalization of the approach to different datasets that *could* have been generated); choosing the parameters θ is based on optimizing for the best average performance over all possible datasets.

In the Bayesian setting, θ is a random variable but $\mathcal{D}$ is fixed and known: the focus of the Bayesian approach is on generalizing over possible values of the parameters and using all the information at hand. Therefore one takes an expectation over θ to make predictions conditioned on the value of $\mathcal{D}$, giving the ***posterior expected loss*** [Robert, 2007]

$$\varrho(\mathcal{D}, d) = \mathbb{E}_{p(\theta|\mathcal{D})}[L(d(\mathcal{D}), \theta)], \tag{3.15}$$

where $p(\theta|\mathcal{D})$ is our posterior distribution on θ. Although $\varrho(\mathcal{D}, d)$ is a function of the data, the Bayesian approach takes the data as given (after all we have a particular dataset) and so for a model and decision rule, the posterior expected loss is a fixed value and, unlike in the frequentist case, further assumptions are not required to calculate the optimal decision rule d^*.

To see this, we can consider calculating the ***Bayes risk*** [Robert, 2007], also known as the ***integrated risk***, which averages over both data and parameters

$$r(d) = \mathbb{E}_{p(\mathcal{D})}\left[\varrho(\mathcal{D}, d)\right] = \mathbb{E}_{\pi(\theta|\mathcal{D})}\left[R(\theta, d)\right]. \tag{3.16}$$

Here we have noted that we could have equivalently taken the expectation of the frequentist risk over the posterior, such that, despite the name, the Bayes risk is neither wholly Bayesian nor frequentist. It is now straightforward to show that the decision function which minimizes $r(d)$ is obtained by, for each possible dataset $\mathcal{D} \in \mathbb{D}$, choosing the decision that minimizes the posterior expected loss, i.e. $d^*(\mathcal{D}) = \arg\min_{d(\mathcal{D})} \varrho(\mathcal{D}, d)$.

By comparison, because the frequentist risk is still a function of the parameters, it requires further work to define the optimal decision rule, e.g. by taking a *minimax* approach [Vapnik, 1998]. On the other hand, the frequentist risk does not require us to specify a prior $p(\theta)$, or even a generative model at all (though many common frequentist approaches will still be based around a likelihood function $p(\mathcal{D}|\theta)$). We note that the Bayesian approach can be relatively optimistic: it is constrained to choose decisions that optimize the expected loss, whereas the frequentist approach allows, for example, d to be chosen in a manner that optimizes for the worst case θ.

We now introduce some shortfalls that originate from taking each approach. We emphasize that we are only scratching the surface of one of the most hotly debated issues in statistics and do not even come close to doing the topic justice. The aim is less to provide a comprehensive explanation of the relative merits of the two approaches, but more to make you aware that there are a vast array of complicated, and sometimes controversial, issues associated with whether to use a Bayesian or frequentist approach, most of which have no simple objective conclusion.

3.4.1 Shortcomings of the Frequentist Approach [Advanced Topic]

One of the key criticisms of the frequentist approach is that predictions depend on the experimental procedure and can violate the ***likelihood principle***. The likelihood principle states that the only information relevant to the parameters θ conveyed by the data $\mathcal{D}$ is encoded through the likelihood function $p(\mathcal{D}|\theta)$ [Robert, 2007, Section 1.3.2]. In other words, the same data and the same likelihood model should always lead to the same inferences about θ. Though this sounds intuitively obvious, it is actually violated by taking an expectation over $\mathcal{D}$ in frequentist methods, as this introduces a dependency from the experimental procedure.

As a classic example, imagine that our data from flipping a coin is 3 heads and 9 tails. In a frequentist setting, we can reach different conclusions about whether the coin is biased depending on whether our data originated from flipping the coin 12 times and counting the number of heads, or if we flipped the coin until we got 3 heads. For example, at the 5% level of a ***significance test***, we can reject the ***null hypothesis*** that the coin is unbiased in the latter case, but not the former. This is obviously somewhat problematic, but it can be used to argue both for and against frequentist methods.

Using it to argue against frequentist methods, and in particular significance tests, is quite straightforward: the subjective differences in our experiment should not affect our conclusions about whether the coin is fair or not. We can also take things further and make the results change for absurd reasons. For example, imagine our experimenter had intended to flip until she got 3 heads, but was then attacked and killed by a bear while the twelfth flip was in the air, such that further flips would not have been possible regardless of the outcome. In the frequentist setting, this again changes our conclusions about whether the coin is biased. Clearly, it is ridiculous that the occurrence or lack of a bear attack during the experiment should change our conclusions on the biasedness of a coin, but that is need-to-know information for frequentist approaches.

As we previously suggested though, one can also use this example to argue for frequentist methods: one can argue that it actually suggests the likelihood principle is itself incorrect.

Although significance tests are a terribly abused tool whose misuse has had severe detrimental impact on many applied communities [Goodman, 1999; Ioannidis, 2005], they are not incorrect, and extremely useful, if interpreted correctly. If one very carefully considers the definition of a ***p-value*** as being the probability that a given, or more extreme, event is observed if the *experiment is repeated*, we see that our bear attack does actually affect the outcome. Namely, the chance of getting the same or more extreme data from repeating the experiment of "*flip the coin until you get* 3 *heads*" is different to the chance of getting the same or a more extreme result from repeating the experiment "*flip the coin until you get* 3 *heads or make* 12 *flips (at which point you will be killed by a bear)*". As such, one can argue that the apparently absurd changes in conclusions originate from misinterpreting the results and that, in fact, these changes actually demonstrate that the likelihood principle is flawed because, without a notion of an experimental procedure, we have no concept of repeatability.

This question is also far from superfluous. Imagine instead the more practical scenario where a suspect researcher stops their experiment early as it looks like the results are likely to support their hypothesis and they do not want to take the risk that if they keep it running as long as they intended, then the results might no longer be so good. Here the researcher has clearly biased their results in a way that ostensibly violates the likelihood principle.

Whichever view you take, two things are relatively indisputable. Firstly a number of frequentist concepts, such as p-values, are not compatible with the likelihood principle. Secondly, frequentist methods are not always ***coherent***, that is they can return answers that are not consistent with each other, e.g. probabilities that do not sum to one.

Another major criticism of the frequentist approach is that it takes a point estimate for θ, rather than averaging over different possible parameters. This can be somewhat inefficient in the finite data case, as it limits the information gathered from the learning process to that encoded by the calculated point estimate for θ, which is then wasteful when making predictions. Part of the reason that this is done is to actively avoid placing a prior distribution on the parameters, either because this prior distribution might be "wrong"[7] or because, at a more philosophical level, they are not random variables under the frequentist definition of probability. Some people thus object to placing a distribution over them at a fundamental level (we will see this objection mirrored by Bayesians for the data in the next section). For the Bayesian perspective (and a viewpoint

[7]Whether a prior can be wrong, or what that even means, is a somewhat philosophical question except in the case where it fails to put any probability mass (or density for continuous problems) on the ground truth value of the parameters.

we actively argue for elsewhere), this is itself also a weakness of the frequentist approach as incorporating prior information is often essential for effective modeling.

3.4.2 Shortcomings of the Bayesian Approach [Advanced Topic]

Unfortunately, the Bayesian approach is also not without its shortcomings. We have already discussed one key criticism in the last section in that the Bayesian approach relies on the likelihood principle which itself may not be sound, or at the very least ill-suited for some statistical modeling problems. More generally, it can be seen as foolhardy to not consider other possible datasets that might have been generated. Taking a very strict stance, then even checking the performance of a Bayesian method on test data is fundamentally frequentist, as we are assessing how well our model generalizes to other data. Pure Bayesianism, which is admittedly not usually carried out in practice, shuns empiricism as it, by definition, is rooted in the concept of repeated trials which is not possible if the data is kept fixed. The rationale typically given for this is that we should use all the information available in the data and by calculating a frequentist risk we are throwing some of this away. For example, cross-validation approaches only ever use a subset of the data when training the model. However, a common key aim of statistics is generalization and repeatability. Pure Bayesian approaches include no consideration for ***calibration***, that is, even if our likelihood model is correct, there is still no reason that any probabilities or confidence intervals must be also. This at odds with frequentist approaches, for which we can often derive absolute guarantees.

A related issue is that Bayesian approaches will often reach spectacularly different conclusions for ostensibly inconsequential changes between datasets.[8] At least when making the standard assumption of i.i.d. data in Bayesian analysis, then likelihood terms are multiplicative and so typically when one adds more data, the relative probabilities of two parameters quickly diverge. This divergence is necessary for Bayesian methods to converge to the correct ground truth parameters for data distributed exactly as per the model, but it also means any slight misspecifications in the likelihood model become heavily emphasized very quickly. As a consequence, Bayesian methods can chronically underestimate uncertainty in the parameters, particularly for large datasets, because they do not account for the ***unknown unknowns***. This means that ostensibly inconsequential features of the likelihood model can lead to massively different conclusions about the relative probabilities of different parameters. In terms of the posterior expected loss, this is often not much of a problem as the assumptions might be similarly inconsequential for predictions. However, if our aim is actually to learn about the parameters

[8] Though this is arguably more of an issue with generative approaches than Bayesian methods in particular.

themselves then this is quite worrying. At the very least it shows why we should view posteriors with a serious degree of skepticism (particularly in their uncertainty estimates), rather than taking them as ground truth.

Though techniques such as cross-validation can reduce sensitivity to model misspecification, generative frequentist methods still often do not fare much better than Bayesian methods for misspecified models (after all they produce no uncertainty estimates on θ). Discriminative methods, on the other hand, do not have an explicit model to specify in the same way and so are far less prone to the same issues. Therefore, though much of the criticisms of Bayesian modeling stem from the use of a prior, issues with model (i.e. likelihood) misspecification are often much more severe and predominantly shared with generative frequentist approaches [Gelman and Robert, 2013]. It is, therefore, often the question of discriminative vs generative machine learning that is most critical [Breiman et al., 2001].

Naturally one of the biggest concerns with Bayesian approaches is their use of a prior, with this being one of the biggest differences to generative frequentist approaches. The prior is typically a double-edged sword. On the one hand, it is necessary for combining existing knowledge and information from data in a principled manner, on the other hand, priors are inherently subjective and so all produced posteriors are similarly subjective. Given the same problem, different practitioners will use different priors and reach potentially different conclusions. In the Bayesian framework, there is no such thing as a correct posterior for a given likelihood (presuming finite data). Consequently, "all bets are off" for repeated experimentation with Bayesian methods as there is no quantification for how wrong our posterior predictive might be compared with the true generating distribution. This can mean they are somewhat inappropriate for tasks where the focus is on repeated use, e.g. providing reliable confidence intervals for medical trials, though many Bayesian methods retain good frequentist properties. Such cases both predicate a need for considering that there are many possible datasets that might occur and, ideally, an objective approach that means the conclusions are independent of the whims of the analyst.

In particular, there is no (objective) Bayesian alternative to frequentist ***falsification*** methods such as the aforementioned significance tests.[9] Both Bayesian and frequentist methods require assumptions and neither can ever truly prove that a particular model or prediction is correct, but

[9]This is not to say one cannot perform falsification as part of Bayesian modeling, in fact avoiding doing so would be ridiculous, but that providing objective statistical guarantees to this falsification requires the use of frequentist methods. For example, even the essential research process of informally rejecting and improving models undermines Bayesian coherence [Gelman et al., 2011]. On the other hand, the process of peer review effectively invalidates frequentist calibration by ensuring some studies never make it to the public domain.

frequentist methods do allow one to indisputably *disprove* hypotheses to within some confidence interval. The real power of this is realized when we consider disproving ***null hypotheses***, e.g. the hypothesis that an output is independent of a potential driving factor. This is why significance tests are so widely used through the sciences as, although they cannot be used to prove a model is correct (as much as people might try), they can certainly be used to show particular assumptions or models are wrong.

The use of a prior in Bayesian modeling can also be problematic because it is often easy to end up "using the data twice" [Gelman et al., 2008]. The Bayesian paradigm requires that the prior is independent from the data and this means that there should not be any human-induced dependency. In other words, strict adherence to the paradigm would require that the user sets their prior before observing the data they condition on, otherwise, the data will both influence their choice of prior and be incorporated through the likelihood. This is not a problem with Bayesian methods per se, but it can be a common shortfall in their application.

Another practical issue with Bayesian methods is their computational complexity at both train time and test time. Firstly, using Bayesian methods requires one to carry out inference, which, as we explain in the following chapters, is typically a challenging and computationally expensive process, often prohibitively so. Some frequentist approaches can also be similarly expensive at train time, but others can be substantially cheaper, particularly discriminative approaches. Further, Bayesian methods tend to also be somewhat expensive for making predictions with the posterior predictive itself being an integral. Frequentist methods tend to instead predict using point estimates for θ, such that prediction is typically much cheaper.

3.4.3 Practical Usage

Although Bayesianism and frequentism are both exact frameworks, their application to real problems is not. Both frameworks have their strengths and weaknesses and so perhaps the key question is not which framework is correct, but when should we use each. In particular, the Bayesian approach is often essential when working with small datasets but where we have substantial prior expertise. On the other hand, a frequentist approach is essential to providing guarantees and ensuring repeatability. Bayesian and frequentist methods are also by no means mutually exclusive and effective modeling often requires elements of both to be used concurrently. For example, one could be Bayesian about the results from a cross-validation test or look to calculate frequentist guarantees for a Bayesian model. In essence, Bayesian and frequentist analysis have different aims – Bayesianism is about updating subjective beliefs and frequentism

is about creating long run, or repeated application, guarantees. We often care about both. It is also worth noting that a number of Bayesian methods exhibit good frequentist properties, see e.g. McAllester [2013] and the references therein.

We finish by noting a critical assumption made by both Bayesian and generative frequentist methods—that there is some true underlying value for the parameters. Because all models are approximations of the real world, this is often a misguided and harmful assumption. This assumption is made is clear in the frequentist setting, but is somewhat subtle for Bayesian approaches. Bayesian methods allow for multiple hypotheses or parameter values, but this originates from our own uncertainty about which parameter or hypothesis is correct, thereby still implicitly assuming that *one* of them is correct. Namely, as we will show with the Bernstein-Von Mises theorem in Section 4.2, in the limit of large data, Bayesian methods with finite numbers of parameters will collapse to a point estimate. Consequently, a Bayesian approach does not fundamentally enrich the model space by averaging over parameters—it is still necessary that exactly one set of parameters lead to the data, but we are not exactly sure which one [Minka, 2000].

3.5 Further Reading

- Chapter 1 of K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012. Note that while most of the book is not freely available, this chapter is: `https://www.cs.ubc.ca/~murphyk/MLbook/pml-intro-22may12.pdf`.

- L. Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001

- Chapter 1 of C. Robert. *The Bayesian choice: from decision-theoretic foundations to computational implementation*. Springer Science & Business Media, 2007. A full version of this book can be found here: `https://www.researchgate.net/publication/41222434_The_Bayesian_Choice_From_Decision_Theoretic_Foundations_to_Computational_Implementation`. Note that this book is rather heavy reading unless you have a statistics background and mostly well beyond the scope of this course.

- Michael I Jordan. Are you a Bayesian or a frequentist? Video lecture, 2009. `http://videolectures.net/mlss09uk_jordan_bfway/`

4
Bayesian Modeling

The challenges involved in performing Bayesian machine learning are remarkably straightforward to identify. As all information is encoded through the prior and the likelihood, the "only" challenges are in specifying these and then carrying out the required Bayesian inference to estimate the posterior. In other words, we need to be able to specify good models and we need to be able to solve them. Though simple to quantify, actually overcoming both these challenges can be difficult in practice.

In this chapter, we consider the former problem of constructing models, discussing some of the most important assumptions and theorems of Bayesian modeling, how graphical models can be used a mechanism for constructing Bayesian models, and a short introduction to nonparametric models with a focus on Gaussian processes. Note that, unlike the last chapter, this chapter will not follow the lecture slides; the overlap between them is quite low. It is recommended that you digest and ensure you understand the content of Lectures 2 and 3 (with the exception of probabilistic programing) before going through this chapter. The aim of the notes is supplement the lectures; they do not amount to a self-contained introduction to the course's content by themselves.

4.1 A Fundamental Assumption

Though not strictly necessary, an assumption made by virtually all Bayesian models is that datapoints are conditionally independent given the parameter values. In other words, if our data is given by $\mathcal{D} = \{x_n\}_{n=1}^N$, then the likelihood factorizes as follows

$$p(\mathcal{D}|\theta) = \prod_{n=1}^{N} p(x_n|\theta). \tag{4.1}$$

There are two main motivations for the assumption, one practical, the other theoretical.

The practical motivation is that if datapoints are not conditionally independent given θ, this indicates our model is ignoring information present in the data. In principle, we want to choose models where our parameters encapsulate as much information that is useful for prediction as possible. If our likelihood model directly uses information from other data, e.g. $p(x_2|\theta, x_1)$, this indicates that there was useful predictive information in x_1 that we are not encapsulating through θ. This in turn indicates we could improve our model by incorporating

this information, e.g. by introducing an additional parameter into θ. As such, we can argue that if we are always using the best possible model given our our knowledge of the problem, this model should always satisfy (4.1). Furthermore, writing models in a manner that satisfies (4.1) is often highly convenient when designing models and conducting inference, particularly if the datapoints are identically distributed.

Note here that there is an important distinction between assuming that our likelihood model is correct, and assuming that it encapsulates all the information salient for prediction. For example, presume that θ represent a real–world, but unknown, parameter. It is perfectly possible (ignoring the fact that models are never perfect) that $p(x_n|\theta)$ represents the correct conditional distribution of x_n in the frequentist sense that this it is the distribution created by repeating the experiment with the same value of θ, but that θ also contains very little information about x_n, such that fixing θ has little influence on the produced x_n. Thus it is possible for our model to be well-specified, but also useless. Similarly, it is possible for a model to not be exactly correct, but still be very useful, a somewhat important fact given that all models are inevitably wrong anyway (see Section 3.2.2).

The theoretical motivation of assuming (4.1) is based on an important result known as ***de Finetti's Theorem*** [De Finetti, 1937]. Though the precise result is beyond the scope of this course, the high-level idea of de Finetti's Theorem is that if we have an infinite sequence of exchangeable random variables (i.e. their probability density is the same if we permute their order), then there exists some parameter which all these variables are *conditionally* independent with respect to. Informally, the upshot of this result in our context is that if the order of data does not matter and our data can be assumed to be a finite sample from an infinitely long sequence (both of which are often the case), then there is exists some set of latent variables (i.e. θ) for which (4.1) holds. Note that this not mean this (4.1) holds for any particular model, it simply shows that a model exists where it does hold. Intuitively, this model is also the most powerful possible model as it ensure θ encapsulates all information relevant to predicting a datapoint that can be provided by other datapoints. This, in turn, provides a motivation for making the assumption.

Though this assumption is so ingrained in Bayesian modeling that its presence is often overlooked (one could even make an argument that this assumption is part of the Bayesian approach itself), it is far from inconsequential. For example, one information–theoretic consequence is that, because all information for prediction must pass through θ, there is a finite amount of information that can be stored in the model if θ is finite dimensional, thereby placing a limit on it predictive power in the limit of large data (see also the Bernstein von-Mises Theorem

below). To try and cater for this, some *non-parametric* models are based around using an infinite dimensional θ (we will return to this in Section 4.5).

Unfortunately, it is well beyond the scope of this course to fully explore the consequences of this assumption. Our aim has instead been to simply make you aware that it is an explicit assumption being made by virtually all Bayesian methods, as well as explaining some of the justifications for why. At the end of the day though, it is important to remember that all models are inevitably approximations of the truth for any real–world scenario. As such, it is impossible to avoid making assumptions entirely.

4.2 The Bernstein-Von Mises Theorem

One of the important implications of the assumption discussed in the last section is the ***Bernstein–von Mises theorem***, which explains the behavior of Bayesian methods in the limit of large data. Assume, for the sake of argument, that (4.1) holds and our likelihood model $p(\mathcal{D}|\theta)$ is correct in the sense that the datapoints x_n (where $\mathcal{D} = x_{1:N}$) are all independent and identically distributed (i.i.d.) according to $p(x_n|\theta^*)$ where θ^* are a (finite) set of "ground truth" parameters and the prior $p(\theta)$ satisfies $p(\theta^*) > 0$. Informally speaking, the Bernstein–von Mises theorem now states that in the limit of large N, the posterior distribution $p(\theta|x_{1:N})$ converges to a normal distribution with mean θ^* and variance of order $O(1/N)$ (i.e. it decreases at a rate $1/N$) [Doob, 1949; Freedman, 1963].

This is a hugely important result in Bayesian statistics as it demonstrates that, in the (predominantly hypothetical) scenario that our model assumptions are correct, we converge to the true parameters. Because of this, it is sometimes referred to as the ***consistency*** of Bayesian methods. It also means that the posterior becomes independent of the prior when we are provided with sufficient data: the likelihood always dominates in parametric models if we provide enough data

It further transpires that when no such θ^* exists (i.e. our model is misspecified), the convergence is instead to the parameters $\hat{\theta}$ which minimize the Kullback-Leibler (KL) divergence[1] to the true data generating distribution $p^*(y_{1:N})$, namely

$$\hat{\theta} = \arg\min_{\theta} \mathrm{KL}\left(p^*(y_{1:N}) \| p(y_{1:N}|\theta)\right) = \arg\min_{\theta} \int p^*(y_{1:N}) \log\left(\frac{p^*(y_{1:N})}{p(y_{1:N}|\theta)}\right) dy_{1:N}. \quad (4.2)$$

See for example [Kleijn et al., 2012] and the references therein for further details.

[1] The KL divergence can informally be thought of as a measure of discrepancy between two distributions. Though it is not symmetric in its inputs, it is always non-negative and zero if and only if the two distributions are the same. We will refer to it in more detail later

The Bernstein–von Mises theorem can be both a blessing and a curse. On the one hand, it ensures we reach the correct conclusion with enough data and a model that is powerful enough to encapsulate the true data distribution. On the other hand, it also means that our uncertainty estimates will collapse to zero given enough data even if our model is misspecified and the answer we are collapsing is not correct: even if θ is a real parameter, it is perfectly possible that $\theta^* \neq \theta$ if our likelihood model is not exactly correct. This is closely linked to the fact that Bayesian models fail to capture the ***unknown unknowns***, such that their uncertainty estimates are usually overconfident: they fail to account for the fact that the model itself might not be correct. This can be a serious problem when think about the fact that our model is almost invariably an approximation of the truth.

4.3 Graphical Models

Generative models will typically have many variables and a complex ***dependency structure***. In other words, many variables will be conditionally independent of one another given values for other variables. ***Graphical models*** are a ubiquitously used method for representing and reasoning about generative models, with a particular focus on the dependency structure. As such, they are an important technique for working with Bayesian models.

At a high-level, graphical models capture how the joint probability distribution can be broken down into a product of different factors, each defined over a subset of the variables. They are formed of a number of connected nodes, where each node represents a random or observed variable in the model. Links between nodes represent dependencies: any two connected nodes have an explicit dependency, though unconnected nodes may still be dependent. Various independence assumptions can be deduced from the graphical model, though the exact nature of these deductions will depend on the type of graphical model; nodes without direct links between them will often still be dependent.

Graphical models can be separated into two distinct classes: ***directed graphical models*** and ***undirected graphical models***. Undirected graphical models, also known as Markov random fields, imply no ordering and are used only to express conditional independences between variables. They are used in scenarios where it is difficult to specify the target distribution in a generative way, e.g. Boltzmann machines [Ackley et al., 1985]. To give a more concrete example, if modeling whether it will rain at various locations, then there is a clear dependence between nearby locations, but not a natural ordering to the joint probability distribution of where it will rain. Independence in undirected graphical models can be deduced through the ***global***

Markov property which states that any two non-intersecting subsets of the variables A and B are conditionally independent given a third, separating, subset C if there is no path between A and B that does not pass through C. This means, for example, that each variable is conditionally independent of all the other variables given its neighbors.

Our main focus, though, will instead be on directed graphical models and in particular ***directed acyclic graphical models*** (DAGs), i.e. directed graphical models containing no cycles or loops one can follow and arrive back in the starting position. DAGs, also known as ***Bayesian networks***, are particularly useful in the context of Bayesian modeling because they can be used to express known conditional relationships. As such, they can be used as a piecewise explanation for how samples are generated from a distribution. This forms a natural means to describe and design models as we can carefully order the breakdown to factorize the distribution into only terms we know: we will generally not have access to all possible factorizations in an analytic form as otherwise there would be no need to perform inference. For example, to represent the joint $p(\theta, \mathcal{D})$ in a Bayesian model we would naturally first introduce the θ and then the data because we known $p(\theta)$ and $p(\mathcal{D}|\theta)$, but not $p(\mathcal{D})$ or $p(\theta|\mathcal{D})$. As a rule-of-thumb, when we define a model using a DAG, we need to be able to define the probability of each variable given its ***parents***, i.e. all the nodes with arrows, representing a link and its direction, pointing to the node the question.

To demonstrate this factorization more explicitly and give a concrete example of a DAG, imagine a medical diagnostic problem where we wish to predict if a patient has lung cancer. Let a denote lifestyle and genetic factors of the patient such as whether they smoke or have (potentially unknown) preexisting conditions. These will generally either be known or can reasonably be estimated by other means, e.g. using tests or by considering prevalence within the population, allowing definition of a prior marginal on a, $p(a)$. Given these factors, we can develop a model for the probability that a patient will develop lung cancer, which we can denote $p(b|a)$ where $b = 1$ indicates cancer is present. Given the lifestyle and genetic factors and the knowledge of whether lung cancer is present, we can predict what symptoms, c, might be observed, e.g. a persistent cough, encoding this using $p(c|a, b)$. We thus have the following breakdown of the joint distribution

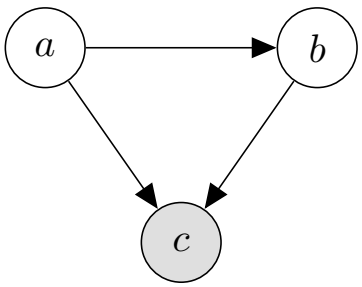


Figure 4.1: Simple example DAG corresponding to (4.3)

$$p(a, b, c) = p(a)p(b|a)p(c|a, b). \tag{4.3}$$

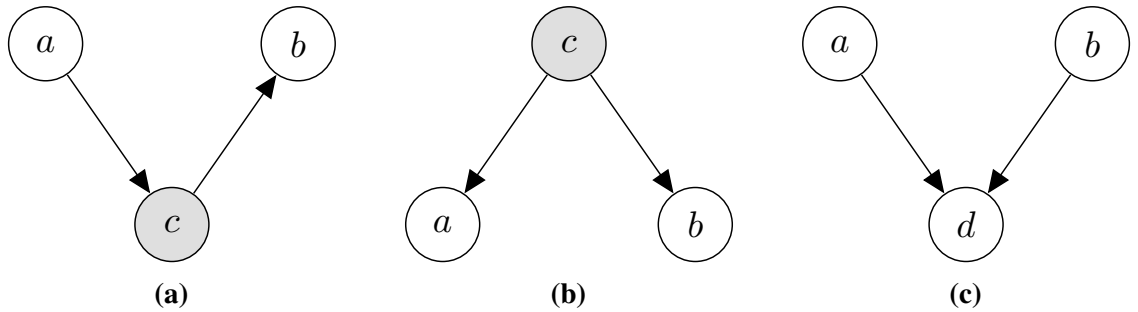


Figure 4.2: Examples of DAGs blocked between a and b. Here both example **(a)** and example **(b)** are blocked by the second rule of d-separation. More specifically, the only path between a and b in each case passes through an observed node c which respectively has one and both of the arrows pointing away from it in the two examples. Consequently, for **(a)** and **(b)** then a and b are conditionally independent given c, such that $p(b|a,c) = p(b|c)$ and $p(a|b,c) = p(a|c)$. **(c)** is instead an example of where a and b are *marginally* independent. Here the only possible path between a and b is blocked, even though there are no observed nodes, due to the first rule of d-separation with $n = d$. This is because the arrows meet head-to-head at d and neither d nor any of its descendants are observed, consequently $p(b|a) = p(b)$ and $p(a|b) = p(a)$. Note though that a and b are not conditionally independent given d as the path becomes unblocked if this is observed (see Figure 4.3).

which can be expressed using the DAG shown in Figure 4.1. Here we have shaded in c to express the fact that this is ***observed***. The graphical model expresses our dependency structure as we have the probability of each node given its parents. As shown in (4.3), the product of all these factors is equal to our joint distribution. The DAG has thus formed a convenient means of representing our generative process. Our aim for this problem was to find the probability cancer is present given the observed symptoms, i.e. $p(b|c)$, which will require Bayesian inference. In our previous simple examples, the posterior had an analytic form. However, in general this will not be the case and we will need to develop strategies for carrying out the inference as explained in the following Chapters. For these, knowing the dependency structure and, in particular, the independence relationships, of a model will often be very helpful information, for which DAGs can be very useful.

A natural question is now how can we deduce the independence relationships from a DAG? This can be done by introducing the notion of ***d-separation*** [Pearl, 2014]. Consider three arbitrary, non-intersecting, subsets A, B, and C of a DAG. A and B are conditionally independent given C if there are no ***unblocked*** paths from A to B (or equivalently from B to A) when C is observed, in which case A is said to be d-separated from B by C. Note that paths do not need to be in the same directions as the arrows in the DAG. A path is defined as blocked if

1. There is an observed node, n, in the path where both the arrows point towards n (i.e. the arrows meet head-to-head at n), and n also has no observed descendants, i.e. we cannot get to any observed nodes of by following arrows from n.

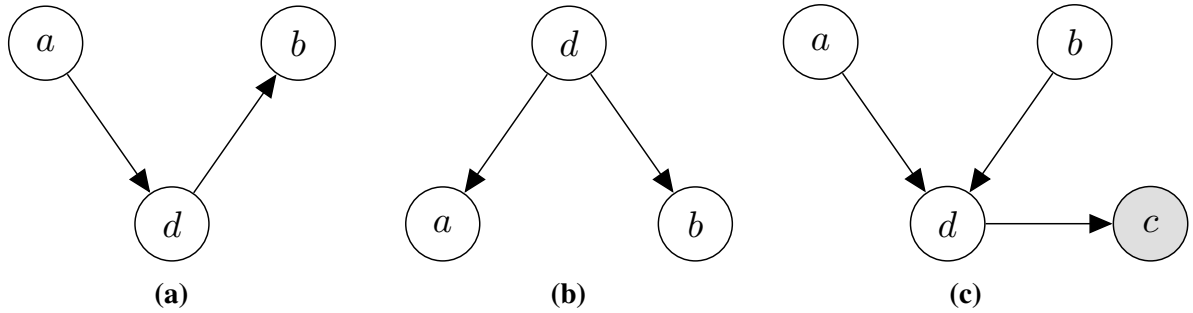


Figure 4.3: Examples of DAGs unblocked between a and b. For **(a)** then the path from a to b is not blocked by the d-separation rules. Perhaps more intuitively, we have that $p(b|a) = \int p(b, d|a)dd = \int p(b|d)p(d|a)dd \neq p(b)$ unless $p(d|a) = p(d)$. Similarly, there is an unblocked path for **(b)** from a to b as the path that does not pass through any observed nodes or nodes with both arrows pointing towards it. The path in **(c)** is unblocked because of the phenomenon of ***explaining away***. The first rule of d-separation does not apply here because c is an observed descendant of d. We thus have that a and b are marginally independent as per Figure 4.2c, but not conditionally independent given c (and or d). The rationale for explaining away can be thought of in terms of events needing an explanation—if two precursor events (here a and b) can give rise to a third event (here c), then the third event occurring but not the first precursor event implies that the second precursor event occurs. Thus the two precursor events are correlated because one *explains away* the other. As described in Section 2.3, one example of this is that if a speed camera is triggered and the camera is not malfunctioning, this implies the vehicle is speeding, even though the vehicle speeding and the camera malfunctioning are marginally independent.

2. Consecutive arrows meet at an observed node and either one or both of them points away from the node.

Note that only the first of these rules is necessary for establishing marginal independence between nodes because only this rule can satisfied when no nodes are observed. Examples of blocked paths are shown in Figure 4.2 while examples of unblocked paths are shown in Figure 4.3, explanations for which are given in the captions. For a more comprehensive introduction to establishing independence in DAGs, we refer the reader to [Bishop, 2006, Section 8.2].

4.4 Example Bayesian Models

4.4.1 Bayesian Linear Regression

In this example, we will consider the problem of a Bayesian linear regression from inputs $x \in \mathbb{R}^D$ to outputs $y \in \mathbb{R}$. For simplicity of notation (namely so we avoid the need for a separate intercept term), we will always consider the first value of x to be 1, such that if we actually care about regressing from $z \in \mathbb{R}^{D-1}$, we take $x = [1, z^T]^T$. Assume that we have N observations $\mathcal{D} = \{x_n, y_n\}_{n=1}^N$ and let $\mathbf{x} = [x_1, \ldots, x_N]^T$ and $\mathbf{y} = [y_1, \ldots, y_N]^T$ respectively be a $N \times D$ matrix and a column vector whose rows correspond to the different data points. Our generative regression model is of the form $y_n = x_n^T \mathbf{w} + \epsilon_n$ where $\mathbf{w} \in \mathbb{R}^D$ and each $\epsilon_n \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$, where

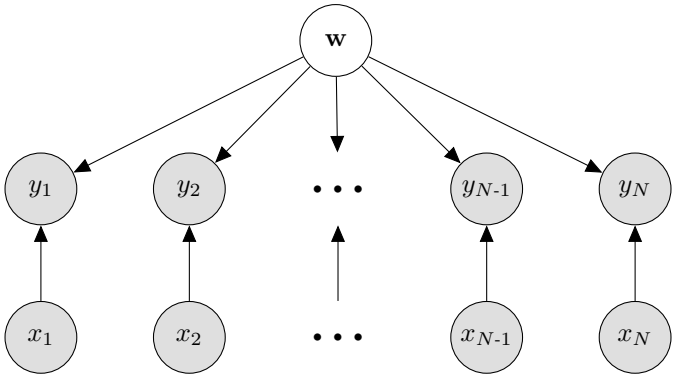


Figure 4.4: DAG for Bayesian linear regression.

$\mathcal{N}$ signifies a normal (i.e. Gaussian) distribution. This implies a likelihood of

$$p(\mathbf{y}|\mathbf{x}, \mathbf{w}, \sigma) = \prod_{n=1}^{N} p(y_n|x_n, \mathbf{w}, \sigma) = \prod_{n=1}^{N} \mathcal{N}(y_n; x_n^T\mathbf{w}, \sigma^2). \tag{4.4}$$

For simplicity, we will assume that σ is a known fixed parameter, but we will put a prior on $\mathbf{w}$—namely $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \mathbf{0}, C)$ where C is a fixed covariance matrix—in order to perform inference. The DAG for this model is given in Figure 4.4.

To make predictions, we first calculate the posterior

$$p(\mathbf{w}|\mathcal{D}, \sigma) = \mathcal{N}(\mathbf{w}; \mathbf{0}, C) \prod_{n=1}^{N} \mathcal{N}(y_n; x_n^T\mathbf{w}, \sigma^2) = \mathcal{N}(\mathbf{w}; m, S) \tag{4.5}$$

$$\text{where} \quad m = S^{-1}\mathbf{x}^T\mathbf{y}/\sigma^2 \quad \text{and} \quad S = \left(C^{-1} + \frac{\mathbf{x}^T\mathbf{x}}{\sigma^2}\right)^{-1}.$$

We have omitted the necessary linear algebra (see for example Bishop [2006] Sections 2.3.3 and 3.3) but note the conjugacy between the normal distribution and itself. Prediction now uses the posterior predictive, marginalizing over the parameters in the same manner as the coin flip example. Here though, we are interested in predicting the output $\tilde{y}$ at a particular input point $\tilde{x}$ for which we have

$$\begin{aligned} p(\tilde{y}|\tilde{x}, \mathcal{D}, \sigma) &= \int p(\tilde{y}|\tilde{x}, \mathbf{w}) p(\mathbf{w}|\mathcal{D}, \sigma) d\mathbf{w} = \int \mathcal{N}(\tilde{y}; \tilde{x}^T\mathbf{w}, \sigma^2)\, \mathcal{N}(\mathbf{w}; m, S)\, d\mathbf{w} \\ &= \mathcal{N}\left(\tilde{y}\ ;\ \tilde{x}^T m,\ \left(\tilde{x}^T S^{-1}\tilde{x} + \frac{1}{\sigma^2}\right)^{-1}\right) \end{aligned} \tag{4.6}$$

which again follows from standard results for Gaussian distributions. We, therefore, have an analytic predictive distribution at any possible input point. Though this linear regression example might seem overly simple for practical purposes, we will see in Section 4.6 that substantially more advanced models, such as Gaussian processes, can be viewed as linear regressions between a set of features on the inputs $\phi(x)$ and the output y.

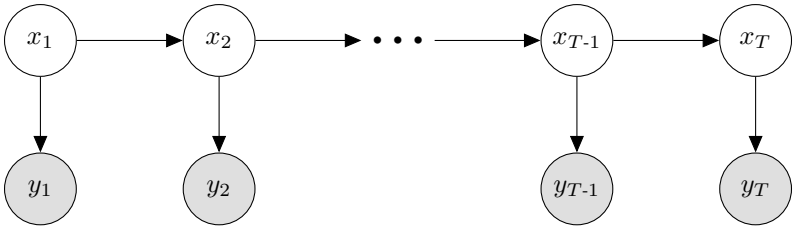


Figure 4.5: DAG for a hidden Markov model.

4.4.2 Hidden Markov Models

In the previous Bayesian linear regression example, there were no independence relationships (other than between the datapoints given the model) and so there was little to gain from working with the DAG. A more advanced example where there are substantial independence relationships which can be exploited is shown in Figure 4.5. This model is known as a hidden Markov model[2] (HMM) and has T latent variables $x_{1:T}$ and T observations $y_{1:T}$. The joint distribution is as follows

$$p(x_{1:T}, y_{1:T}) = p(x_1)p(y_1|x_1)\prod_{t=2}^{T} p(x_t|x_{t-1})p(y_t|x_t), \tag{4.7}$$

where each x_t is independent of $x_{1:t-2}$ and $y_{1:t-1}$ given x_{t-1} and of $x_{t+2:T}$ and $y_{t+1:T}$ given x_{t+1}. This is known as the Markov property and means that each latent variable only depends on the other variables and observations through its immediate neighboring states. In essence, the model has no memory as information is passed forwards (or backwards) only through the value of the previous (or next) latent state. A number of stochastic processes and dynamical systems obey the Markov property and HMMs and their extensions are extensively used for a number of tasks involving sequential data, such as DNA sequencing [Durbin et al., 1998] and tracking animals [Dhir et al., 2016, 2017] to name but a few.

A key part of the appeal of HMMs is that the structure of the DAG can be exploited to give analytic solutions to the resulting Bayesian inference whenever each $p(y_t|x_t)$ and $p(x_t|x_{t-1})$ are either a categorical or Gaussian distribution. Even when this does not hold, there are still a number of features of the dependency structure that can make the inference substantially easier. As we will show in later chapters, Bayesian inference is generally a challenging problem, often prohibitively so. Therefore the (fast) analytic inference for HMMs is highly convenient. However, it can mean that HMMs are perhaps overused. More generally, simplifying approximations or unjustified assumptions are often made by Bayesian practitioners for tractability, e.g. by using an off-the-shelf model like an HMM with known analytic solution. Though often necessary, this

[2]The use of the term HMM is overloaded in the literature. Sometimes authors simply mean the Markov structure we discuss here, but it can also used in a way that implies that the latent states are discrete variables, with the equivalent continuous model referred to as a (Markovian) state space model.

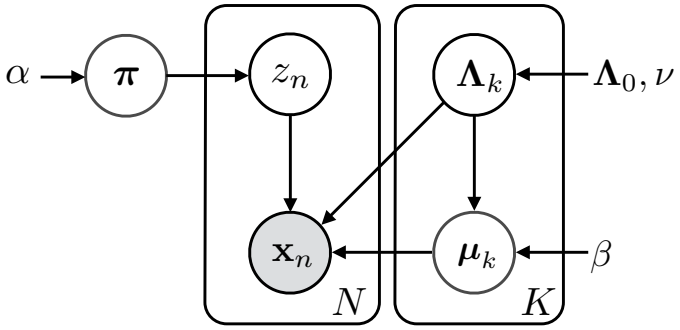


Figure 4.6: DAG for Bayesian Gaussian mixture model. Here the boxes are a standard graphical model ***plate notation*** and respectively indicated that the contents are repeated N and K times respectively. This is a concise way of constructing a DAG with a large number of similar nodes and could have equally been applied for Figure 4.4. Note that variables not inside nodes represent fixed parameters instead of random variables.

must be done with extreme care and the implications of the approximations should carefully considered. Unfortunately, quantifying the implications of approximations can be very difficult, given that they are typically made in the interest of tractability in the first place.

4.4.3 Bayesian Gaussian Mixture Model with Conjugate Priors

In the lectures we considered the example of a Gaussian mixture model with a fixed distribution over the mixture components, along with a fixed mean and covariance matrix for each mixture. Here we instead introduce a more complex variant where we also introduce priors over all of these, further generalizing to an arbitrary number of mixtures K. Namely we have

$$\pi \sim \text{Dirichlet}(\alpha) \tag{4.8}$$

$$\Lambda_k \sim \text{Wishart}(\Lambda_0, \nu) \quad \forall k \in \{1, \dots, K\} \tag{4.9}$$

$$\mu_k \Big| \, \Lambda_k \sim \mathcal{N}\Big(\mathbf{0}, (\beta\Lambda_k)^{-1}\Big) \quad \forall k \in \{1, \dots, K\} \tag{4.10}$$

and treat each of these as latent variables we wish to do inference over. Note that the Dirichlet and Wishart distributions are common distributions over categorical probability distribution and precision matrices respectively. Each mixture then has mean μ_k and precision Λ_k, with

$$z_n \Big| \, \pi \sim \text{Categorical}(\pi) \quad \forall n \in \{1, \dots, N\} \tag{4.11}$$

$$x_n \Big| \, z_n = k, \; \mu_k, \; \Lambda_k \sim \mathcal{N}(\mu_k, \Lambda_k^{-1}) \quad \forall n \in \{1, \dots, N\}. \tag{4.12}$$

Note here that we have $\theta = \{\pi, \mu_{1:K}, \Lambda_{1:K}, z_{1:N}\}$, $\mathcal{D} = \{x_n\}_{n=1}^N$, and α, ν, β, and Λ_0 are fixed parameters. The full likelihood of the model is given by

$$p(\mathcal{D}|\theta) = p(\pi; \alpha) \left(\prod_{n=1}^{N} p(z_n|\pi) p(x_n|\mu_{z_n}, \Lambda_{z_n}) \right) \left(\prod_{k=1}^{K} p(\Lambda_k; \Lambda_0, \nu) p(\mu_k|\Lambda_k; \beta) \right). \tag{4.13}$$

The corresponding graphical model is shown in Figure 4.6.

Note that the priors have been chosen very carefully here to allow ***conjugacy*** relationships to be exploited through known results for conjugate exponential family distributions. Namely, it turns out that all conditional distributions of the form $p(z_n = k|z_{-n}, x_{1:N})$ can be calculated analytically. This allows one to construct a highly efficient Gibbs sampling scheme[3] to perform inference in the model. Though we will not go into more details here, those interested to read more should check out the following graduate course homework assignment designed by Frank Wood: `http://www.robots.ox.ac.uk/~fwood/teaching/AIMS_CDT_ML_2016/homework/HW_3_sampling/`. This provides additional details, links, and coding problem based around running inference in this model.

4.5 Nonparametric Bayesian Models

So far we have mostly implicitly assumed that we are using a ***parametric*** Bayesian model, i.e. that our model has a finite number of parameters. In many cases this can actually be a highly restrictive assumption. In particular, such parametric models can only encode a limited amount of ***information***, such that when the total dimension of the data exceeds that of the parameters, our model (informally speaking) cannot capture all of the information available. For example, in our Bayesian linear regression example, our model only stores information about the possible linear fits of the data; in the limit of large data it will collapse to a single line that contains little information relative to that which was actually present in the data (presuming that the true generative process was more complicated than this simple model). More generally, any parametric Bayesian model will ***underfit*** given enough data.

Nonparametric Bayesian methods are an approach that allows us to get around this problem by either using an infinite dimensional parameterization, or a model where the number of parameters grows with the size of the data. This allows us to construct models which can become increasingly complex as the amount of data available to them grows. As such, many nonparametric models are very flexible and powerful modeling approaches. A key motivation for many nonparametric methods is to try and let the data dictate the complexity of the model, for example, the number of clusters in a clustering problem.

A downside to nonparametric models is that because they generally work with infinite dimensional spaces, we can usually only use very particular classes of models where we can carry out analytic marginalizations of the parameters to allow tractable calculations. Note that this is not the same as, and much more restrictive than, the more general intractability of

[3] We will introduce Gibbs sampling in Lecture 5

Bayesian inference: if we cannot (at least approximately) marginalize out these dimensions, we cannot work with the model at all. As such, many nonparametric methods are actually somewhat less flexible than we might hope, because these tractability assumptions often enforce other strong assumptions to be made.

Though it is beyond the scope of this course to cover nonparametric methods in detail (e.g. many approaches require far more advanced probability theory than we have covered), we will introduce one of the most common nonparametric approaches, ***Gaussian processes***, in the next section. For those interested in other nonparametric approaches, we refer them to [Gershman and Blei, 2012] for an accessible introduction that stays clear of their many theoretical complexities.

4.6 Gaussian Processes

Informally one can think of a Gaussian processes (GPs) [Rasmussen and Williams, 2006] as being a nonparametric distribution[4] over functions. Equivalently, one can think of it as a generalization of a Gaussian distribution to (uncountably) infinite dimensions. Practically, they are powerful tools for regression, classification, and feature extraction [Kuss and Rasmussen, 2005; Lawrence, 2004].

GPs are extremely effective predictive models, especially in low dimensions and in data-scarce settings; they allow for the use of prior distributions and provide a measure for uncertainty quantification. We focus here on regression as it is generally the simplest usage. Once trained, prediction from a Gaussian Processes corresponds to computing the mean and variance of the Gaussian predictive distribution in the target space $y \in \mathcal{Y}$. Inference for GPs is made possible by the self-conjugacy of the Gaussian distribution.

In the rest of this section we will first introduce GPs from a ***function–space*** perspective, before showing how we can derive them more formally. In particular, we will show how they can be viewed as an extension of the Bayesian linear regression example we introduced earlier, where we first map our inputs into an infinite dimensional feature space before showing how this can be analytically marginalized out.

[4]Technically speaking they are stochastic processes, not distributions.

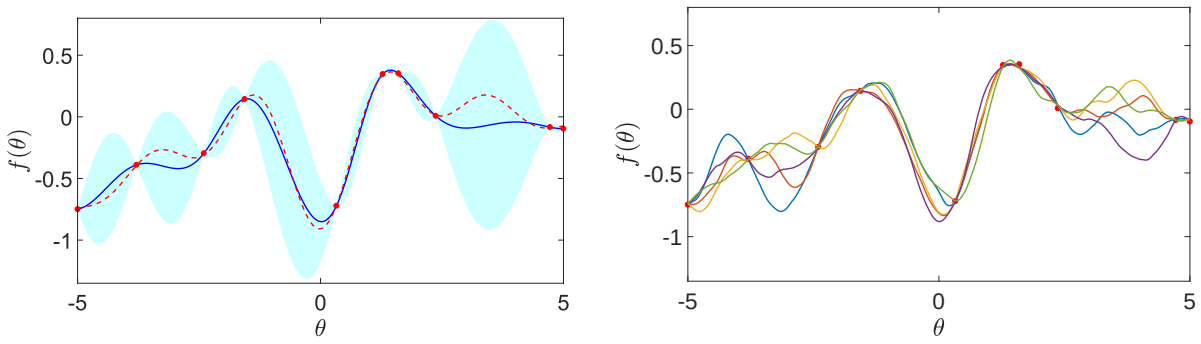


Figure 4.7: Example GP regression for points shown in red dots. Shown left is the GP posterior where the solid dark blue line is the mean of the GP and the light blue shading is the mean ± 2 standard deviations. Shown right is five example functions drawn from this posterior. I apologize for the inconsistent notation of the axis labels (θ should be x): they have been recycled from earlier work using notation.

4.6.1 Function–Space View

A GP is fully specified by a ***mean function*** $\mu \colon \mathcal{X} \to \mathbb{R}$ and a symmetric ***covariance function*** $k \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, the latter of which must be a bounded (i.e. $k(x, x') < \infty,\ \forall x, x' \in \mathcal{X}$) and reproducing kernel (we will return to this in depth later in Section 4.6.3). We can informally describe a function f as being distributed according to a GP:

$$f(x) \sim GP(\mu(x), k(x, x')) \tag{4.14}$$

which by definition means that the functional evaluations realized at any finite number of sample points is distributed according to a multivariate Gaussian. For example, if we consider three points $x = 0$, $x = 1$, and $x = 3$, then we have that

$$\begin{bmatrix} f(0) \\ f(1) \\ f(3) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(0) \\ \mu(1) \\ \mu(3) \end{bmatrix}, \begin{bmatrix} k(0,0) & k(0,1) & k(0,3) \\ k(1,0) & k(1,1) & k(1,3) \\ k(3,0) & k(3,1) & k(3,3) \end{bmatrix} \right) \tag{4.15}$$

Note that the inputs to μ and k need not be numeric: a GP can be defined over anything for which a suitable mean function and kernel can be defined.

Figure 4.7 shows an example GP regression with a small number of data points $\{x_j, v_j\}_{j=1:m}$. Here we wish to regress a function f from x to y. To do this, we place a GP prior on f

$$f_{\text{prior}}(x) \sim \text{GP}\left(\mu_{\text{prior}}(x), k_{\text{prior}}(x, x')\right),$$

where it is normal to take $\mu_{\text{prior}}(x) = 0\ \forall x$, noting the generality of this choice as one can simply regress $y - \mu_{\text{prior}}(x)$ if a different μ_{prior} is desired. Lots of options are viable for the covariance function, we will return to how these are chosen in the next section. For now, we simply note that the choice of covariance function and its parameters will heavily influence the

nature of functions generated by the GP, dictating things such as smoothness and characteristic length scales of variation.

An important property of a GP that we now exploit is that it is conjugate with a Gaussian likelihood. Let $\mathbf{x} = \{x_j\}_{j=1:m}$ and $\mathbf{v} = \{v_j\}_{j=1:m}$ be the set of observed inputs and outputs respectively. We use a separable Gaussian likelihood function

$$p(\mathbf{v}|\mathbf{x}, f) = \prod_{j=1}^{m} p(v_j|f(x_j)) = \prod_{j=1}^{m} \frac{1}{\sigma_n\sqrt{2\pi}} \exp\left(-\frac{(v_j - f(x_j))^2}{2\sigma_n^2}\right) \tag{4.16}$$

where σ_n is an observation noise, set to 0.001 in our example. Combining this with the GP prior we previously defined leads to GP process posterior and predictive distribution.

To see this we can consider the joint distribution between the points we have considered so far, and a new set of points $\{\mathbf{x}^*, \mathbf{v}^*\}$. Introducing the shorthand $k_{\text{prior}}(\mathbf{x}, \mathbf{x}^*) = \begin{bmatrix} k_{\text{prior}}(x_1,x_1^*) & k_{\text{prior}}(x_1,x_2^*) & \dots \\ k_{\text{prior}}(x_2,x_1^*) & k_{\text{prior}}(x_2,x_2^*) & \dots \\ \dots & \dots & \dots \end{bmatrix}$ then we have by the fact that any finite realizations of points is Gaussian

$$\begin{bmatrix} \mathbf{v} \\ f^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k_{\text{prior}}(\mathbf{x}, \mathbf{x}) + \sigma_n^2 I & k_{\text{prior}}(\mathbf{x}, \mathbf{x}^*) \\ k_{\text{prior}}(\mathbf{x}^*, \mathbf{x}) & k_{\text{prior}}(\mathbf{x}^*, \mathbf{x}^*) \end{bmatrix}\right) \tag{4.17}$$

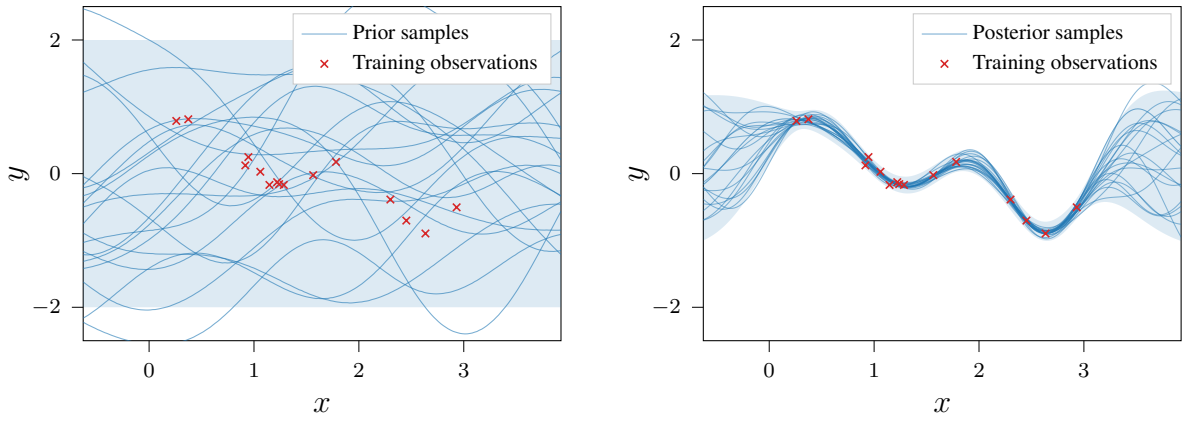
where I is the identity matrix, $\mathbf{0}$ is a vector of zeros, and f^* is the true function values at $\mathbf{x}^*$ (such that $\mathbf{v}^* - f^* \sim \mathcal{N}(0, \sigma_n^2 I)$). We can now use standard results for Gaussians (see e.g. Petersen et al. [2008]) to get the conditional distribution for f^* given all the other variables

$$\begin{aligned} f^*|\mathbf{x}, \mathbf{v}, \mathbf{x}^* &\sim \mathcal{N}\left(\mu_{\text{post}}\left(\mathbf{x}^*\right), k_{\text{post}}\left(\mathbf{x}^*, \mathbf{x}^*\right)\right) \quad \text{where} \\ \mu_{\text{post}}\left(\mathbf{x}^*\right) &= k_{\text{prior}}\left(\mathbf{x}^*, \mathbf{x}\right)\left[k_{\text{prior}}\left(\mathbf{x}, \mathbf{x}\right) + \sigma_n^2 I\right]^{-1}\mathbf{v} \\ k_{\text{post}}\left(\mathbf{x}^*, \mathbf{x}^*\right) &= k_{\text{prior}}\left(\mathbf{x}^*, \mathbf{x}^*\right) - k_{\text{prior}}\left(\mathbf{x}^*, \mathbf{x}\right)\left[k_{\text{prior}}\left(\mathbf{x}, \mathbf{x}\right) + \sigma_n^2 I\right]^{-1} k_{\text{prior}}\left(\mathbf{x}, \mathbf{x}^*\right). \end{aligned} \tag{4.18}$$

Now as $\mathbf{x}^*$ are arbitrary points, this corresponds to our predictive distribution for the function (note that to convert this to the posterior predictive for the outputs y, we need to add back in the likelihood noise to the covariance matrix, i.e. add a term $\sigma_n^2 I$ onto k_{post}). Further, as this predictive distribution is still a GP, we can refer to our model as having a GP posterior

$$f_{\text{post}}(x)|\mathbf{x}, \mathbf{v} \sim \text{GP}\left(\mu_{\text{post}}(x), k_{\text{post}}(x, x')\right).$$

Going back to Figure 4.7, we see the result of this process. Our GP regression gives us a posterior mean function that represents the expected value at every possible input points, along with a variance that represents our subjective uncertainty in the value of the function at that point. Figure 4.7 also shows that we can draw from the GP by choosing some set of evaluation points $\mathbf{x}^*$ and then drawing from (4.18). We see that there is a larger variation in the function values away from the points that have been evaluated, as would be expected. An alternative visualization of GP training is shown in Figure 4.8.



(a) Before Training **(b)** After Training

Figure 4.8: Samples from a Gaussian Process distribution before and after training (i.e. from the GP prior and posterior). On the left, we show samples coming from a zero-mean GP prior with a radial basis function kernel. On the right, samples from a GP posterior that has been trained on the data displayed as red crosses. The shaded areas area corresponds to $\mu \pm 2\sigma$ and is thus roughly a 95% confidence internal for the function values. *Figure credit: Gabriele Abbati*

An important point of note is that the marginal likelihood for our model is also analytic, namely

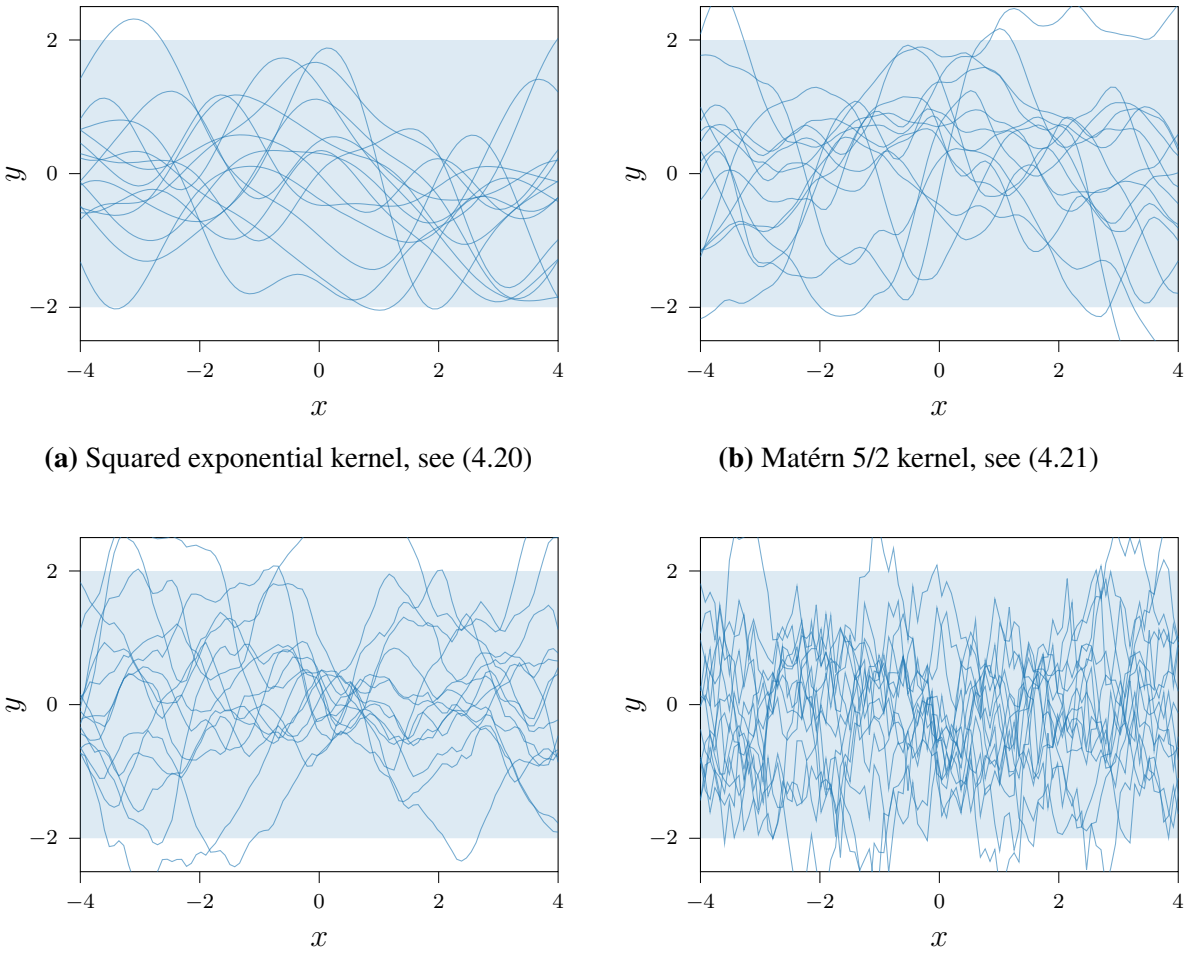
$$\log p(\mathbf{v}|\mathbf{x}) = -\frac{1}{2}\mathbf{v}^T \left[k_{\text{prior}}(\mathbf{x},\mathbf{x}) + \sigma_n^2 I\right]^{-1} \mathbf{v} - \frac{1}{2}\log\left|k_{\text{prior}}(\mathbf{x},\mathbf{x}) + \sigma_n^2 I\right| - \frac{m}{2}\log 2\pi \quad (4.19)$$

where m is the number of points in $\mathbf{x}$. This is important as it means it will be tractable to optimize or do inference over the GP hyperparameters.

4.6.2 Kernels and Hyperparameters

As we showed in the last section, GPs form powerful and expressive priors over functions. In this section, we show that their practical behavior varies substantially depending on the choice of the covariance function, aka ***kernel***, and the hyperparameters. Informally, we can think of the kernel as expressing the similarity between the evaluation of the function at two different input points x and x'. When $k(x, x')$ is large, these evaluations are strongly correlated such that the evaluations will have similar values. When it is small, there is little correlation between the points and so their evaluations will be roughly independent. Note that it is possible for $k(x, x') < 0$ provided $x \neq x'$, but as kernels must be positive definition functions as explained in the next section, $k(\mathbf{x}, \mathbf{x})$ is always a positive definite matrix. Though it is not necessary for the kernel to be stationary (i.e. that it only depends on $x - x'$ rather than the absolute values), we will not consider non-stationary cases further here (see Rasmussen and Williams [2006] for further information).

Figure 4.9 shows some simple example kernels. The figure shows that the qualitative behavior of the GP changes substantially with changes to the kernel. The choice of kernel is therefore



(a) Squared exponential kernel, see (4.20)

(b) Matérn 5/2 kernel, see (4.21)

(c) Matérn 3/2 kernel, see (4.21)

(d) Exponential kernel $\sigma_f^2 \exp(-\|x - x'\|/\rho)$

Figure 4.9: Samples from Gaussian process priors with different covariance functions. *Figure credit: Gabriele Abbati*

critical to the performance of GPs. Though there is work that examines methods for learning the kernel directly [Duvenaud et al., 2013; Lloyd et al., 2014; Wilson et al., 2014; Janz et al., 2016], this is typically too computationally intensive to carry out in practice. One must therefore either use prior knowledge about the problem, or use a relatively general purpose kernel to ensure that the nuances of the data are not overwhelmed.

A particularly common problem in the choice of kernel is in the desired level of ***smoothness***. At the one extreme, one can use the ***squared exponential kernel***

$$k_{\text{se}}(x, x') = \sigma_f^2 \exp\left(-\frac{\|x - x'\|_2^2}{2\rho^2}\right) \tag{4.20}$$

which is infinitely differentiable. Here σ_f^2 is the "signal standard deviation"—a hyperparameter that affects the scaling of the output from the GP: the larger σ_f, the higher the average standard deviation of the function. Meanwhile, ρ is a hyperparameter that dictates the ***characteristic***

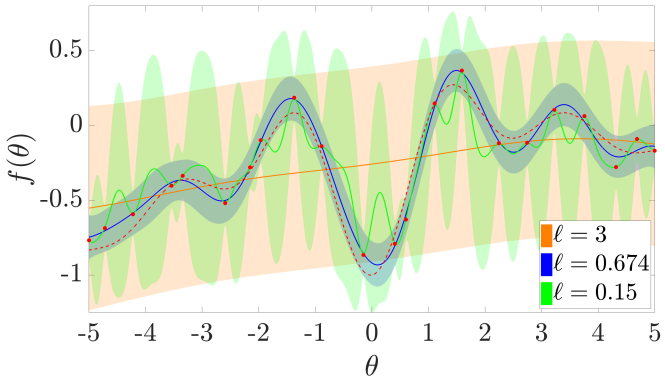


Figure 4.10: Effective of changing the length scale for Matérn-5/2 kernel. Again I apologize for the consistent notation of the axis labels.

length scale of variation for the function—the higher ρ is, the slower the correlation decreases as one moves away from the current point, and therefore the less "wiggly" the function is. Both these hyperparameters are shared by most commonly used kernels. Although the presented kernel is isotropic, this is easily relaxed by having a different ρ for each dimension.

In many practical problems, the squared exponential kernel is too smooth. One possible alternative in these scenarios are the ***Matérn kernels***. The Matérn-ν kernel, given by

$$k_\nu(x, x') = \sigma_f \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{\rho} \|x - x'\| \right)^\nu K_\nu \left(\frac{\sqrt{2\nu}}{\rho} \|x - x'\| \right) \tag{4.21}$$

where K_ν is the modified Bessel function of second kind order ν and $\Gamma(\cdot)$ is the gamma function, is $\lfloor \nu - 1 \rfloor$ times differentiable and therefore different values of ν can be used to express different levels of smoothness. Typically ν is set to $n + 1/2$ for some integer n, for which it has a simple closed form [Rasmussen and Williams, 2006].

Though the choice of kernel function is critical for a GP, the choice of hyperparameters such as the scaling and length scale can have equally significant impact on the behavior. Figure 4.10 shows the effect of changing the length scale of a Matérn-5/2 kernel. When the length scale is too large, as shown in orange, the regressed function is overly smooth and does not capture the data. When the length scale is too small, as shown in green, wild fluctuations are permitted in between the datapoints such that there is very high uncertainty between points and also potentially overfitting as the regressor can model all the fluctuations in the data, including those originating simply from noisy evaluations. As the hyperparameters are rarely known upfront, one must typically either optimize them, or perform inference over them to produce a mixture of GPs.

4.6.3 Weight–Space View & Reproducing Kernel Hilbert Space [Advanced Topic]

One of the key advantages of GPs is that they can be used effectively without needing to know the intricacies for why they work. The derivations we introduced in the last section and in particular the calculations required to derive the GP posterior were spectacularly simple, albeit computationally intensive (the need for a matrix inversion to calculate (4.18) means that training is $O(m^3)$ for m training points). However, what is going on beneath the surface is substantially more complicated, which is perhaps not surprising when one remembers that they are defined over uncountably infinite many variables. To truly understand the underlying rationale and assumptions for GPs, and to properly appreciate the restrictions on possible kernels, it is necessary to delve into GPs more formally, taking a so–called ***weight–space*** view. In this section, we, therefore, outline a more formal derivation of a GP from the viewpoint of reproducing kernels [Hofmann et al., 2008] and show how they can be thought of as Bayesian linear regression using an infinite number of features. We start with the following definitions.

Definition 4.1. *An **inner product** $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is a function $\mathcal{H} \times \mathcal{H} \rightarrow \mathbb{R}$ associated with a vector space $\mathcal{H}$ that is symmetric $\langle u, v \rangle_{\mathcal{H}} = \langle v, u \rangle_{\mathcal{H}}$, linear $\langle au_1 + bu_2, v \rangle_{\mathcal{H}} = a\langle u_1, v \rangle_{\mathcal{H}} + b\langle u_2, v \rangle_{\mathcal{H}}$, and positive definitive $\langle u, u \rangle_{\mathcal{H}} \geq 0,\ \langle u, u \rangle_{\mathcal{H}} = 0 \ \Leftrightarrow \ u = \mathbf{0}$.*

Definition 4.2. *A **Hilbert space**, $\mathcal{H}$, is a, potentially uncountably infinite, vector space associated with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ which is complete with respect to the norm $\|u\|_{\mathcal{H}} = \sqrt{\langle u, u \rangle_{\mathcal{H}}}$.*

Less formally, we can think of Hilbert spaces as being a generalization of Euclidean space (i.e. the space vectors live in) to include functions. Using Hilbert spaces, we can think of a function as being an uncountably long vector defining the value of the function at each possible input point.

Consider now the linear (in the weights $\mathbf{w}$) model

$$f(x) = \mathbf{w}^T Z(x) + \mu(x), \quad \mathbf{w} \sim \mathcal{N}(\mathbf{0}, C) \tag{4.22}$$

where $Z(\cdot) = [\zeta_1(\cdot), \ldots, \zeta_m(\cdot)]^T$, $\zeta_a \colon \mathcal{X} \rightarrow \mathbb{R}$ for $a = 1, \ldots, m$ is a feature map in m-dimensional Hilbert space $\mathcal{H}$. For any set of points $x_1, \ldots, x_t$ then $F = [f(x_1), \ldots, f(x_t)]^T$ will be distributed according to a $t-$dimensional Gaussian with mean $[\mu(x_1), \ldots, \mu(x_t)]^T$ and covariance $k(x_i, x_j) = Z(x_i)^T C Z(x_j), \ \forall i, j \in \{1, \ldots, t\}$. Note that

$$Z(x)^T C Z(x') = \sum_{a=1}^{m} \sum_{b=1}^{m} C_{ab} \zeta_a(x) \zeta_b(x') \geq 0, \quad \forall x, x' \in \mathcal{X} \tag{4.23}$$

as C must be positive semi-definite for the distribution on $\mathbf{w}$ to be well defined[5]. This also means that the Cholesky decomposition $C^{1/2}$ exists and we can define $\Psi(\cdot) = C^{1/2}Z(\cdot) = [\psi_1(\cdot), \ldots, \psi_m(\cdot)]^T$ to give a covariance function $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ of the form

$$k(x, x') = \langle \Psi(x), \Psi(x') \rangle_{\mathcal{H}} = \sum_{a=1}^{m} \psi_a(x)\, \psi_a(x') \leq \sqrt{\sum_{a=1}^{m} (\psi_a(x))^2} \sqrt{\sum_{a=1}^{m} (\psi_a(x'))^2} \quad (4.24)$$

where the inequality trivially follows using Cauchy-Schwarz. Now consider the case where m is uncountably infinite such that $\mathcal{H}$ represents a functional space along with an inner product. If $\Psi(\cdot)$ remains in L^2 space, i.e.

$$\sum_{a=1}^{\infty} (\psi_a(x))^2 < \infty, \quad \forall x \in \mathcal{X}, \quad (4.25)$$

then $k(x, x') < \infty,\ \forall x, x' \in \mathcal{X}$ using the inequality from equation (4.24), and our covariance will remain bounded with infinitely many basis functions.

The key realization is now that the distribution of F only depends on Z and C through the inner product $\langle \Psi(x), \Psi(x') \rangle_{\mathcal{H}}$ such that we need never compute the (potentially infinite) mapping $\Psi(x)$ if we can find a way to implicitly compute the inner product, for example if k has the ***reproducing property***

$$\langle u(\cdot), k(\cdot, x) \rangle_{\mathcal{H}} = u(x), \quad \forall x \in \mathcal{X},\ \forall u(\cdot) \in \mathcal{H}. \quad (4.26)$$

This is called the ***kernel trick***, where we have used the representation $\Psi(x) = k(\cdot, x)$ noting that any feature map can be thought of as parameterizing a mapping $\mathcal{H} \to \mathbb{R}$ through the inner product. We can now introduce the notion of ***reproducing kernel Hilbert space*** (RKHS) [Aronszajn, 1950].

Definition 4.3. *Given a Hilbert space $\mathcal{H}$, then a function $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a reproducing kernel and $\mathcal{H}$ is a reproducing kernel Hilbert space if $k(\cdot, x) \in \mathcal{H},\ \forall x \in \mathcal{X}$ and the reproducing property described in equation* (4.26) *is satisfied.*

The significance of RKHSs for GPs is that their covariance functions correspond to reproducing kernels and so a GP is defined over an RKHS. Most RKHSs do not contain all possible functions—e.g. the RKHS associated with the squared exponential kernel contains only infinitely differentiable functions—and so the choice of GP kernel will dictate the range functions it is capable of encapsulating. Note that as $k(x, x') = \langle k(\cdot, x), k(\cdot, x') \rangle = k(x', x)$, all reproducing kernels are symmetric.

[5] $Z(x_i)^T C Z(x_j) = 0$ is only possible if $\mathbf{w}$ lives in a lower dimensional subspace of $\mathbb{R}^m$

Going back to our GP derivation, then for the realization of $k(x, x')$ at a finite number of points to be a valid covariance matrix we further require it to be positive definite, i.e:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\beta_i\beta_j k(x_i, x_j) \geq 0, \quad \forall n \geq 1,\ \forall\{\beta_1, \ldots, \beta_n\} \in \mathbb{R}^n,\ \forall\{x_1, \ldots, x_n\} \in \mathcal{X}^n \tag{4.27}$$

which can be proved to be the case given the restrictions we have placed on k by noting

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\beta_i\beta_j k(x_i, x_j) = \sum_{i=1}^{n}\beta_i\Psi(x_i)^T\sum_{j=1}^{n}\beta_j\Psi(x_j) = \left\|\sum_{i=1}^{n}\beta_i\Psi(x_i)\right\|_{\mathcal{H}}^2 \geq 0. \tag{4.28}$$

Consequently, (4.25) is a sufficient condition for the distribution of F to be a multivariate $t-$dimensional Gaussian with a valid and finite covariance matrix when we consider an uncountable infinite number of basis functions. Furthermore, we note that many choices for Ψ will lead to closed-form analytic expressions of $\langle\Psi(x), \Psi(x')\rangle_{\mathcal{H}}$ without the need to ever calculate $\Psi(x)$ explicitly.

To show a simple example of this consider the one-dimensional case where $x \in \mathbb{R}$ and basis functions of the form $\psi_a = \sigma_w \exp\left(-\frac{(x-c_a)^2}{2\rho^2}\right)$ where $c_a \in [c_{\min}, c_{\max}]$ represents the center of the basis function and ρ is a common length scale. Further specify that $\sigma_w^2 = \frac{\beta(c_{\max}-c_{\min})}{m}$, i.e. that σ_w^2 is proportional to the number of functions per unit length, then we have

$$k(x, x') = \sum_{a=1}^{m}\frac{\beta(c_{\max}-c_{\min})}{m}\exp\left(-\frac{(x-c_a)^2}{2\rho^2}\right)\exp\left(-\frac{(x'-c_a)^2}{2\rho^2}\right). \tag{4.29}$$

If we assume that the basis functions are evenly space in $[c_{\min}, c_{\max}]$, then in the limit $m \to \infty$ we recover the integral

$$k(x, x') = \beta\int_{c_{\min}}^{c_{\max}}\exp\left(-\frac{(x-c)^2}{2\rho^2}\right)\exp\left(-\frac{(x'-c)^2}{2\rho^2}\right)dc. \tag{4.30}$$

Now if we further take $c_{\min} \to -\infty$ and $c_{\max} \to \infty$ then we have the analytic solution

$$k(x, x') = \sqrt{\pi}\rho\beta\exp\left(-\frac{(x-x')^2}{4\rho^2}\right), \tag{4.31}$$

which is the common squared exponential kernel in one dimension. Thus we have managed to analytically marginalize over all basis functions and are left with a simple expression that can be used to evaluate the covariance between any two given points that implicitly uses an uncountably infinite number of basis functions.

More generally, the Moore-Aronszajin theorem [Aronszajn, 1950] states that

Theorem 4.1. *Any symmetric, positive definite kernel $k\colon \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ has a unique RKHS for which k is a reproducing kernel.*

In other words, if we define a kernel function that is symmetric $k(x, x') = k(x', x)$ and positive definite as per (4.27), then at least one corresponding feature map $\Psi : \mathcal{X} \to \mathcal{H}$ must exist. Therefore a corresponding $Z : \mathcal{X} \to \mathcal{H}$ and C must also exist (for example we can trivially take C as the identity matrix and $Z = \Psi$), and if we further ensure that k is bounded $k(x, x') < \infty, \ \forall x, x' \in \mathcal{X}$, then finite realizations $[f(x_1), \ldots, f(x_t)]^T$ of equation (4.22) must be distributed to a finite multivariate Gaussian distribution with mean $[\mu(x_1), \ldots, \mu(x_t)]^T$ and covariance $k(x_i, x_j) \ \forall i, j \in 1, \ldots, t$.

We can, therefore, think of GP regression as Bayesian linear regression using a feature mapping to an RKHS. This shows the power of a GP—for appropriate choices of kernel it uses an uncountable number of features—but also highlights their assumptions: the co-Gaussianity of the weights and reliance of the target function to fall in RKHS represented by the covariance function.

4.7 Further Reading

- Chapters 1-3 and 8 of C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006
- Chapter 5 of K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012
- Zoubin Ghahramani on Bayesian machine learning (there are various alternative variations of this talk): `https://www.youtube.com/watch?v=y0FgHOQhG4w`
- Iain Murray on Probabilistic Modeling: `https://www.youtube.com/watch?v=pOtvyVYAuW4`
- Eric Xing's course on Probabilistic Graphical Model: `http://www.cs.cmu.edu/~epxing/Class/10708-14/lecture.html`
- C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. Available at: `http://www.gaussianprocess.org/gpml/chapters/RW.pdf`
- S. J. Gershman and D. M. Blei. A tutorial on Bayesian nonparametric models. *Journal of Mathematical Psychology*, 56(1):1–12, 2012
- Peter Orbanz and Yee Whye Teh on Bayesian nonparametrics: `https://www.youtube.com/watch?v=F0_ih7THV94`.

5
Probabilistic Programming

Probabilistic programming systems (PPSs) allow probabilistic models to be represented in the form of a generative model and statements for conditioning on data [Gordon et al., 2014; Goodman et al., 2008b]. Informally, one can think of the generative model as the definition of a prior, the conditioning statements as the definition of a likelihood, and the output of the program as samples from a posterior distribution. Their core philosophy is to ***decouple model specification and inference***, the former corresponding to the ***user-specified program code***, which implicitly defines a distribution on random variables, and the latter to an ***inference engine*** capable of operating on arbitrary programs. Removing the need for users to write inference algorithms significantly reduces the burden of developing new models and makes effective statistical methods accessible to non-experts. The inference/model abstraction barrier further means that some systems allow the definition of models that would be hard, or even impossible, to convey using conventional frameworks like graphical models.

Two key challenges for PPS are providing the syntax and semantics to allow easy definition of models, and in designing the solvers, i.e. inference engines, to provide effective inference for those models. In this Chapter, we focus on the former of these, providing an introduction to probabilistic programming from a user's perspective. We we will outline PPSs how it can be used for, and to extend, conventional Bayesian modeling and how it can also be used to reinterpret many computational simulation techniques not usually associated with Bayesian modeling in a more Bayesian mindset. We will mostly ignore the rather major issue of how to construct inference engines for PPS. Instead, we will focus on how general purpose PPSs aim to provide the ***flexibility*** to define wide-ranging and potentially obscure models, the ***expressivity*** of a framework for model definition that is more in line with conventional scientific simulation than mainstream statistical approaches, and the ***automation*** to run any problem the user might write by decoupling model specification and inference. Together these characteristics produce a framework that allows researchers whose expertise lies elsewhere, to carry out powerful statistical analyses for their application specific tasks. This framework also aids in the development of both inference

algorithms and models for those within the machine learning and statistics communities, by removing many of the complications of one while developing the other.

We note that it will be necessary at times during this chapter to refer briefly to some Bayesian inference algorithms that will not be properly introduced until later chapters or which are potentially not even covered in the course at all. We have situated this chapter before those partly in order to emphasize the point that one should not need an intricate knowledge of inference methods to *use* PPSs. Though it is difficult to introduce PPSs while completely omitting reference to inference methods, readers who are not familiar with them should be able to safely ignore which methods are be referred to at a first pass, noting only that different inference algorithms have different requirements and sets of problems they perform well on, and thus that the design of a PPS is often intricately linked to the inference method(s) used.

In general, this chapter is quite advanced, particularly the latter sections. Do not worry if you are struggling to follow: later parts of the course will not be building further on the content of this chapter. It has instead been provided because PPSs may prove a useful tool for some of the assignment papers, one of which is explicitly on probabilistic programming.

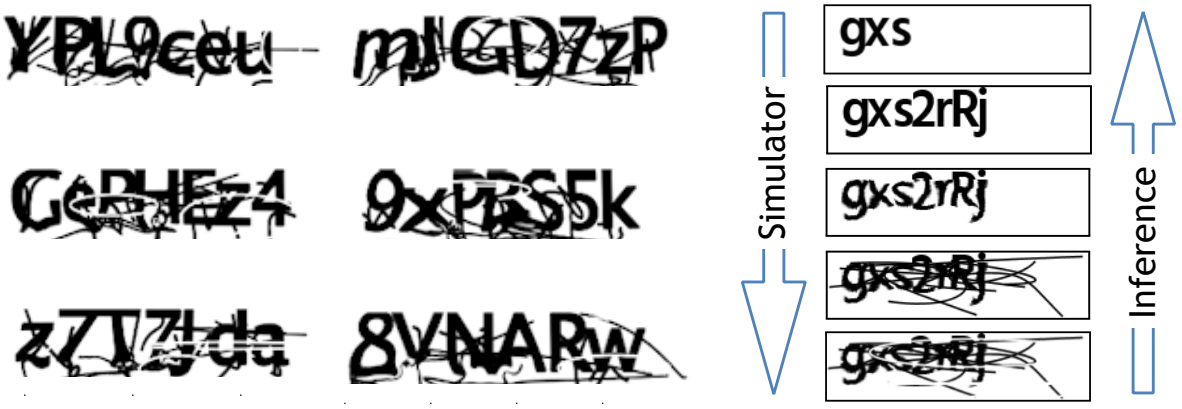
5.1 Inverting Simulators

Though the use of Bayesian modeling through the sciences and engineering is widespread, it is still dwarfed by the use of ***simulators*** more generally. Some simulations are inherently probabilistic, such as many of those used in statistical physics [Landau and Binder, 2014], financial modeling [Jäckel, 2002], and weather prediction [Evensen, 1994]. Others are deterministic approximations of a truly stochastic world, such as lap time simulation for formula one cars [Perantoni and Limebeer, 2014] and finite element simulations for fluid dynamics [Versteeg and Malalasekera, 2007]. In many of these scenarios, real data is also available, but not in sufficient quantities that the carefully constructed simulations can be done away with entirely and replaced by a purely data–driven discriminative machine learning approach. Imagine the potential utility of general–purpose methods for incorporating real data into these simulators to improve them, without having to throw away the existing carefully constructed models. What about if we could even find methods for automatically inverting these simulators? Given a target lap time, we could return the best car setup; given observations of, and a simulator for, human behavior, we could learn about the underlying cognitive processes; given a climate change model and measurements, we might infer what the driving factors are.

An ambitious long-term aim of ***probabilistic programming*** is to solve such problems and to do so in an automated fashion so that it requires little or no statistical expertise on the behalf of the user, allowing for simple, widespread usage across many fields. The key realization is that stochastic simulators implicitly define probability distributions. They, therefore, represent generative models and using probabilistic programming we can reason about, and work with, these generative models explicitly. One typically thinks of Bayesian modeling in terms of the prior and the posterior, but one can also think about it in terms of defining a joint distribution over both parameters and data, then fixing the latter to the actual observations to get a conditional distribution from this joint. Simulators implicitly define such joint distributions, with the outputs of the simulator corresponding to the data, and the inputs and internal variables the parameters. Probabilistic programming allows us to turn this on its head, using the same code as the original simulator, but instead providing the observed data as the input and then inverting the simulator through inference to learn about possible input parameters and other variables sampled during the program's forward execution. As well as the clear direct utility of allowing such inversion, this process also allows us to improve our simulator using real data, by calculating the posterior predictive distribution that incorporates both the original model and the information from the data.

To explain what we mean by inverting simulators more precisely, we will now consider the example of inferring ***Captchas*** [Mansinghka et al., 2013]. Even if the name is not familiar, everyone should hopefully have come across Captchas before when a website shows us an image, such as those in Figure 5.1a, and asks us to type the characters in that image to demonstrate we are not a robot. We now ask the question: how might we write an algorithm that breaks this Captcha by automatically predicting these characters directly from the image? In other words, can we build a robot that mimics a human on a task specifically designed to distinguish between the two? If we had access to large supply of training examples, i.e. character-image pairs, we could, of course, use an off-the-shelf discriminative algorithm: neural networks have been used to try and solve the problem in exactly this way with reasonable success [Von Ahn et al., 2008]. However, without access to an abundance of data, this is a rather challenging task and we will need to exploit our prior knowledge about the problem.

Doing this process in reverse, i.e. simulating a Captcha, on the other hand, is a substantially less daunting process. The true Captchas are actually generated by a simulator and so we can attempt to mimic this original simulation process. For example, as shown in Figure 5.1b we might first sample a number of characters, then a symbol for each character, apply manipulations such as rotations and warpings, simulate some obscuring lines or other noise, and finally render



(a) Examples of real Captchas **(b)** Example simulation of a Captcha

Figure 5.1: Solving Captchas by simulator inversion. **(a)** gives examples of real Facebook Captchas taken from Le et al. [2017b]. Here the corresponding generating strings going to clockwise from the top left are YPL9ceu, mJGD7zP, 9xPBS5k, 8VNARw, z7T7Jda, and GePHEz4. A user is asked to type in these strings when shown the corresponding image to show they are not a robot. **(b)** gives an example of the process of simulating Captchas taken by Le et al. [2017a]. Here we see that we can generate a Captcha by first simulating a series of characters, then simulating appropriate manipulations to those characters such as warping, rotating, and adding noise. Inverting this simulation process corresponds to an inference problem, where we want to find out the characters that lead to a particular image.

our simulated components into an image. Though admittedly it might take some time and effort to construct a high fidelity simulator that well matches the produced images, the technical requirements for doing this (other than possibly the final rendering) are minimal and so it could be carried out by most people with a reasonable degree of experience in scientific programming. The number of people able to write such a simulator should certainly be substantially larger than the number of people with sufficient experience in Bayesian modeling to construct an equivalent graphical model or direct mathematical formulation. The number of people with the expertise to then write an appropriate inference scheme for the problem is even smaller. In a PPS, writing this simulator and providing the data is all that is required. Given these, the PPS will carry out the inference required to invert the simulator automatically, inferring the characters from the image. More generally, we are estimating the inputs and internal variables of our simulator, given target values for the outputs.

There are two key factors to realizing our aim of inverting simulators. Firstly we need to provide a language which easily allows users to write down simulators and which has semantics that allows the compiler to extract an appropriate representation of the joint distribution. In other words, we need our language to be sufficiently general purpose and easy to use to not burden the user, while at the same time having syntax and semantics that ensure the corresponding joint distribution is well defined and can be converted into a form where we can run inference.

Doing this will require the introduction of means for ***conditioning*** on data and of specially defined ***random primitives*** whose behavior can be controlled at run time, rather than just always sampling from the same predefined distribution as they would in an ordinary programming language. The latter can be thought of as defining terms in the prior and the former as defining terms in the likelihood as we will discuss in Section 5.3.

For certain cases, one can alternatively think of conditioning as applying ***constraints*** to the program. For example, we can think of a probabilistic program as defining a simulator and a set of constraints that must be satisfied; this is exactly how the PPS Church [Goodman et al., 2008b] is designed. An important distinction here though is between ***hard*** and ***soft*** conditioning. Hard conditioning is as per the conventional interpretation of a constraint—we condition on the fact that an event occurs exactly. Soft conditioning instead assigns a weight to the program based on the probability (or probability density) of a given event occurring. Though hard conditioning is a particular case of soft conditioning (for which the weight is either 1 or 0), one can, at least semantically, use it to specify a soft conditioning, say $p(Y = y|X = x)$, by sampling $Y \sim p(y|X = x)$ and then imposing the constraint $Y = y$. However, only supporting hard conditioning in a PPS is somewhat restrictive for practical use, as one cannot effectively condition on continuous data because there is zero probability of satisfying the resulting constraint.

The second key factor is that our language needs a general purpose inference(s) engine capable of working on any program the user writes. Bayesian models are fully defined by their joint distribution and the data. Therefore, once a user has written their simulator and provided the data, this uniquely defines a posterior and the only problem is in solving the resulting Bayesian inference. If we can now construct inference engines capable of working on arbitrary code, we can automate inference on any simulator or model the user defines, creating an abstraction barrier between model definition and drawing inferences from that model.

If we can construct a system that can successfully carry out these tasks, the huge potential applications this could provide should be clear. We would have a system where the user requires no expertise in inference or conventional Bayesian modeling in order to write application-specific models and have them solved automatically. Instead, they need only have the ability to write stochastic simulators for the process they wish to model, a skill possessed by most of the scientific community and many of those outside it as well. In a hypothetical future where scientists code all their simulators in extremely powerful PPSs, tasks such as inverting those simulators and improving the simulator by incorporating real data would be automated in the same way current compilers convert high-level coding languages to machine code. However, this ability is not

completely hypothetical—many such problems can already be handled by existing systems. An important current research challenge is improving and scaling such systems to deal effectively with more difficult and more wide-ranging models in a tractable manner. The need for such systems to work in an automated manner for a wide array of possible problems makes this a very difficult problem; after all, we are somewhat flaunting the no-free-lunch theorem. However, there is a key component that provides hope that this may be possible: we have access to the target source code of the simulator itself, rather than needing to treat it as a black-box.

5.2 Differing Approaches

Rather than being a clearly defined method, probabilistic programming is more of an umbrella term that covers a spectrum of different approaches, varying from inference toolboxes through to universal probabilistic programming languages (PPLs) that allow arbitrary probabilistic code to be written. Often there is a trade-off between efficiency and expressivity: the more restricted one makes the language, the more those restrictions can be exploited to improve the efficiency of the inference. This lends itself to two distinct philosophies when developing a system. Firstly one can start with a particular inference algorithm and then design a system around making it as easy as possible to write models for which that inference algorithm is suitable. Secondly one can start with a general purpose language that allows as many models as possible to be written and then try to construct inference engines that are capable of working in such a general framework. Both approaches have their merits and drawbacks, with the distinction typically coming down to the intended use. We will now elucidate each approach more precisely.

5.2.1 Inference Driven Systems

Though there is a plethora of bespoke inference algorithms designed for particular models, the vast majority of these are based around a relatively small number of foundational methods such as importance sampling, sequential Monte Carlo, Metropolis-Hastings, Gibbs sampling, message passing, and variational inference (we will introduce a number of these in the coming chapters). The extensive use of these core inference approaches throughout Bayesian statistics and machine learning means that it makes clear sense to write packages for automating them and which make it easy for the user to define appropriate graphical models for which the inference can be automated. This both improves the efficiency of constructing models and reduces barriers to entry by reducing the required expertise for users. This inference-first philosophy is taken by a number of successful PPSs and inference toolboxes (the distinguishing line between which can be a little blurry), a small number of which we now briefly outline.

BUGS (Bayesian inference Using Gibbs Sampling) [Spiegelhalter et al., 1996] and its extensions [Lunn et al., 2000; Plummer et al., 2003; Todeschini et al., 2014] allow finite DAGs to be specified using declarative code or pictorially using a graphical user interface. These are converted to a form that is suitable for inference, the exact nature of which depends on the implementation, with the original work being based on Gibbs sampling.

Infer.Net [Minka et al., 2010] is modeling language for defining, and automating approximate inference in both DAGs and Markov random fields, using predominantly message-passing algorithms. Distributions are generally, though not exclusively, restricted to be exponential families. Branching (i.e. `if`) is allowed but requires enumeration of all possible paths at run time.

LibBi [Murray, 2013] is a package for doing Bayesian inference for state-space models, using particle-based inference methods. It has a strong focus on scalable computation, providing support for multi-core architectures and graphics processing units.

PyMC3 [Salvatier et al., 2016] is a python framework for carrying out MCMC and variational inference, using Theano [Bergstra et al., 2010] to calculate the gradients required by some inference methods.

Stan [Carpenter et al., 2015] is a PPS with interfaces to many difference languages and a focus on performing Hamiltonian Monte Carlo inference [Duane et al., 1987; Hoffman and Gelman, 2014], though other inference methods such as variational inference are also provided [Kucukelbir et al., 2015]. As with PyMC3, automatic differentiation [Baydin et al., 2015] is used to calculate the required gradients. The need to take derivatives means that there is limited support for discrete variables or branching.

Edward [Tran et al., 2016] is a PPS based around Tensorflow [Abadi et al., 2016] that supports directed graphical models, neural networks, and combinations of both. It supports both Monte Carlo and variational inference methods (again using automatic differentiation) and has a strong emphasis on model criticism.

These systems do not allow users to write models that would be difficult (at least for an expert) to code without a PPS—in general, they can all be thought of as defining a graphical model or sometimes a factor graph—but they offer substantial utility through ease of model exposition and automating inference.

5.2.2 Universal Probabilistic Programming

As useful as these inference-driven systems are, they do not fit very well with the notion of inverting simulators we introduced in Section 5.1. They are still closely tied to graphical models and are more toolboxes for streamlining the Bayesian modeling process than a means of writing models that would be problematic to define by conventional means. Achieving the aforementioned long-term ambitious aim of making general purpose systems for conducting inference on arbitrary simulators requires us to take a somewhat different approach that instead starts with a general-purpose language and then attempts to design inference algorithms capable of working on arbitrary models and code. Such systems need to support models where the set of random variables is dynamically typed, such that it is possible to write programs in which this set, and thus potentially the number of random variables, differs from execution to execution. To avoid hindering the user or restricting the models which can be defined, it will important to allow things such as branching, recursion, higher-order functions, conditional existence of variables, and arbitrary deterministic functions. Ideally, one would like to provide no restrictions on the code that the user can write, except for eliminating programs that do not define valid probability distributions, such as those that have a non-zero probability of never terminating. In practice catching such invalid cases can be hard or even impossible and so many systems actually adopt a philosophy of applying no restrictions, such that it is perfectly possible to define invalid models. General purpose PPSs actually bring up new theoretical questions about what constitutes a valid probability model [Heunen et al., 2017] and the set of valid definable models is a strict superset of those definable by graphical models for many systems [Goodman, 2013].

These systems are sometimes known as *universal* PPLs [Goodman et al., 2008a; Staton et al., 2016], so-called because they are based on *Turing complete* languages that can specify any computable distribution [Goodman, 2013]. Here we will briefly discuss some prominent higher order PPLs.

Church is a PPL based on Scheme [Goodman et al., 2008a]. The original seminal paper and accompanying system form a foundation on which many of the prominent existing systems are built, through its demonstration that higher-order probabilistic programs define valid probability models, even in the presence of infinite recursion. However, Church predominantly only allows hard conditioning,[1] namely a model in Church comprises of a generative sampler and a separate predicate procedure which returns true if the desired conditions are satisfied. In addition to the

[1] Some very limited support for soft-conditioning is provided in current implementations through a "noisy equals" that equates to a Gaussian likelihood.

aforementioned issues of hard conditioning, this complete separation of the generative process and the conditioning can also be wasteful in not allowing the structure of a model to be exploited. Later systems therefore mostly allow soft conditioning statements to be interleaved through the generative progress (in an analogous manner to likelihood terms), increasing the range of (solvable) models that can be encoded and the potential efficiency of inference algorithms. Inference in Church (and its direct derivatives) is typically carried out using either rejection sampling or MCMC. Church places a particularly strong emphasis on the ability to carry out ***nested inference*** [Rainforth, 2018].

Venture [Mansinghka et al., 2014] is a probabilistic programming platform providing a flexible system for both specification of models and inference methods. It has a strong emphasis on being extensible and for allowing the hosting of external applications. For example, it allows the user to provide proposals for the inference engine or reprogram the inference strategy entirely. Venture is predominantly used via the VentureScript PPL [Mansinghka et al., 2014].

Anglican [Wood et al., 2014] is a universal PPL integrated into *Clojure* [Hickey, 2008]. Anglican inherits most of the syntax of Clojure, but extends it with the key special forms **`sample`** and **`observe`** [Tolpin et al., 2015, 2016; Rainforth, 2017], defined in the same way as our example language setup in the next section. Despite using predominantly the same syntax, Anglican has different *probabilistic* semantics to Clojure, i.e. code is written in the same way, but is interpreted differently. Anglican was the first PPL to introduce particle based inference schemes such as sequential Monte Carlo (SMC) [Doucet et al., 2001] and particle MCMC [Andrieu et al., 2010], which was a key advancement for universal PPSs because it allows the structure of the query to be exploited, often providing substantially more efficient inference then previous approaches.

WebPPL [Goodman and Stuhlmüller, 2014] is a PPL built using a purely functional subset of Javascript, conveniently allowing for embedding in web pages. It combines the ability to write a generative process using sampling statements and to add in likelihood terms through a `factor` primitive that is analogous to the **`observe`** primitive that we will introduce in Section 5.3.1. At its back end, WebPPL provides a number of different inference algorithms, such as SMC and MCMC methods.

Pyro [Bingham et al., 2017] and ProbTorch [Siddharth et al., 2017] are two more recent PPLs based on PyTorch [Paszke et al., 2017] that share many design characteristics. Similarly to Edward, they allow modeling of neural networks, but differ in that they construct gradients dynamically, allowing things such as stochastic control and recursion.

The price for the expressivity of these general purpose systems is a substantial extra burden on the inference engine. In general, inference methods for such systems must be formulated in such a manner that they are applicable to models where the density function is intractable and can only be evaluated during forwards simulation of the program. For example, it may not be possible to know if a variable is continuous or discrete except by running the program, while some variables will only exist conditioned on the values of others. This required generality of the inference engine will naturally lead to a drop in performance compared to custom written inference code, but this is often a price worth paying for generality and automation, particularly when considering models that would be challenging to express, let alone do inference in, using more conventional frameworks.

5.3 Bayesian Models as Program Code [Advanced Topic]

In Section 5.1 we showed how one can think of PPSs as inverting simulators, predicting internal variables given the outputs. In this section, we will take a different perspective and show how we can translate Bayesian modeling into the framework of program code. In the previous chapters, we showed how a Bayesian model is defined by a prior over parameters and a likelihood function for those parameters given the data. This viewpoint will mostly translate into the probabilistic programming setting by equating between the prior and sampling statements and between the likelihood and conditioning statements. However, in Section 5.3.2 we will why explain why this is not always exactly true.

A key point to note throughout this section is that probabilistic programs define models rather than procedures. We refer to these models as ***queries*** [Goodman et al., 2008b] which are analogous to functions in a ordinary language. In a standard programming language, functions take in inputs and then run through a series of commands in order until termination is reached.[2] Likewise, random sampling statements like `rand`, `randn` etc, make a single independent draw from the same distribution each time they appear in the execution trace. Neither is the case for a probabilistic program query. Instead a query defines a model which is compiled to a form that can be interpreted by an inference engine which then outputs some characterization of the posterior such as a series of samples. Perhaps the easiest way to think about how a probabilistic programming language works (though not necessarily what happens for all systems) is that

[2] In functional programming languages operations are not necessarily carried out in the order they are defined, but it still holds that the function takes inputs and then carries out a series of actions until the desired output is calculated.

the query is, or sometimes parts of the query are, run many times and the exact behavior of this running is control by the inference engine.

5.3.1 A Simplified Probabilistic Programming Setup

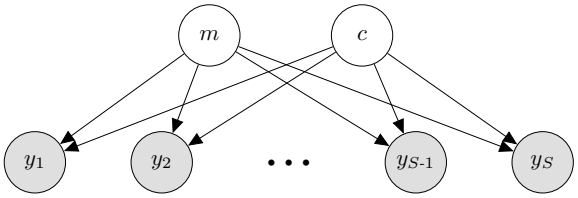
We first consider the case of constructing a restricted hypothetical example PPL. We emphasize that this is by no means the only setup one can use, with design choices made in the interest of exposition. We will presume that our PPL has no branching (i.e. there are no `if` statements or equivalent), recursion, or memoization (i.e. functions are always re-evaluated from scratch when called); is first order (i.e. variables cannot be functions); and that it does not allow any conditioning on internally sampled variables. We will give our language two special constructs, **`sample`** and **`observe`**, between which the distribution of the query is defined.

Informally, **`sample`** will be used to specify terms in the prior and **`observe`** terms in the likelihood. More precisely, **`sample`** will be used to make random draws $x_t \sim f_t(x_t|\Xi_t)$, where Ξ_t is a subset of the variables in scope at the point of sampling, and **`observe`** will be used to condition on data y_s, factoring the program density by $g_s(y_s|\Lambda_s)$, with Λ_s defined in the same way as Ξ_t. For our inference, it will be necessary to control the sampling and so we define the syntax of **`sample`** to take a ***distribution object*** as its only input and for **`observe`** to take a distribution object and an observation as input. We further define each distribution object as containing a sampling procedure and a density function that can be evaluated exactly. Our language will be provided with a number of ***elementary random procedures*** in the form of distribution classes for common sampling distributions such as the normal and Poisson distributions, but will also provide the ability for users to define their own distribution classes. These classes allow a distribution object to be constructed when provided with the required parameters, such that the distribution is fully defined before being passed to a **`sample`** or **`observe`**. We complete our syntactic definition of **`sample`** and **`observe`** by defining them to return a sample and `nil` respectively. We will presume here and throughout that, other than the effects of **`sample`** and **`observe`**, functions in our PPL are ***pure***, such that they always provide the same outputs when called with the same inputs. This restriction naturally means that queries should not have any random components other than dictated by **`sample`** and **`observe`**, but also suggests that they should be free from side effects such as modifications of global variables.

For our simplified setup, we distinguish between two types of inputs to our queries: external parameters ϕ and data $y_{1:S}$. The external parameters are defined as the inputs that are not "observed" at any point but can affect the conditioning through Ξ_t and Λ_s. We presume that

Inputs: Student-t degrees of freedom ν, error scale σ, data $y_{1:S} = \{u_s, v_s\}_{s=1}^S$

$m \leftarrow$ **sample** (**normal** (0,1))
$c \leftarrow$ **sample** (**normal** (0,1))
obs-dist $\leftarrow$ **student-t** (ν)
for $s = 1, \ldots, S$ **do**
    $d \leftarrow (v_s - mu_s - c)/\sigma$
    observe (obs-dist, d)
end for
return $m,\ c$

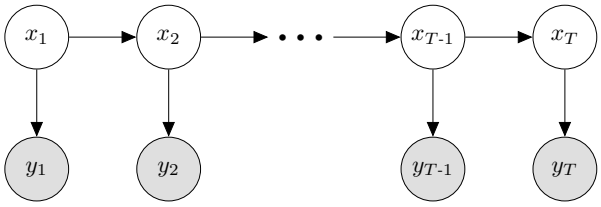


$$p(m, c, y_{1:S}|\nu, \sigma) = \mathcal{N}(m; 0, 1)\, \mathcal{N}(c; 0, 1) \prod_{s=1}^{S} \text{STUDENT-T}\left(\frac{v_s - mu_s - c}{\sigma}; \nu\right)$$

(a) Bayesian linear regression model with student-t likelihood, namely $v_s = mu_s + c + \sigma\epsilon_s$ where $\epsilon_s \sim \text{STUDENT-T}(\nu)$. We presume that the scaling of the error σ and the number of degrees of freedom ν are fixed input parameters (i.e. $\phi = \{\nu, \sigma\}$), that our fixed data is $y_{1:S} = \{u_s, v_s\}_{t=1}^S$, and that we are trying to infer the slope m and intercept c (we thus have $x_1 = m$, $x_2 = c$ in our general notation), both of which are assigned a unit Gaussian as a prior. Our query first samples m and c (note that **normal** (0,1) generates a unit Gaussian distribution object) and constructs a student-t distribution object obs-dist. It then cycles over each datapoint and observes $(v_s - mu_s - c)/\sigma$ using obs-dist, before finally returning m and c as outputs. Note that if we instead wished to directly predict the outputs at some untested inputs points $u_{S+1:S+n}$ then we could predict these anywhere in the query (after m and c have been defined) and return them as outputs along with, or instead of, m and c.

Inputs: Transition std-dev σ, output shape α, output rate β, data $y_{1:T}$

$x_0 \leftarrow 0$
tr-dist $\leftarrow$ **normal** $(0, \sigma)$
obs-dist $\leftarrow$ **gamma** (α, β)
for $t = 1, \ldots, T$ **do**
    $x_t \leftarrow x_{t-1}$+**sample** (tr-dist)
    observe (obs-dist, $y_t - x_t$)
    $z_t \leftarrow \mathbb{I}(x_t > 4)$
end for
return $z_{1:T}$



$$p(x_{1:T}, y_{1:T}|\sigma, \alpha, \beta) = \mathcal{N}(x_1; 0, \sigma^2)\ \text{GAMMA}(y_1 - x_1; \alpha, \beta) \prod_{t=2}^{T} \mathcal{N}(x_t - x_{t-1}; 0, \sigma^2)\ \text{GAMMA}(y_t - x_t; \alpha, \beta)$$

(b) State-space model with Gaussian transition and Gamma emission distributions. It is a form of the HMM model given in (4.7) with $p(x_1) = \mathcal{N}(x_1; 0, \sigma^2)$, $p(x_t|x_{t-1}) = \mathcal{N}(x_t - x_{t-1}; 0, \sigma^2)$, and $p(y_t|x_t) = \text{GAMMA}(y_t - x_t; \alpha, \beta)$ with shape parameter α and scale parameter β. We assume that the input parameters $\phi = \{\sigma, \alpha, \beta\}$ are fixed and we want to sample from $p(z_{1:T}|y_{1:T}, \phi)$ given data $y_{1:T}$, where each z_t is an indicator for if x_t exceeds a threshold of 4. Our query, exploiting the equivalence between $p(x_1)$ and $p(x_t|x_{t-1} = 0)$, first initializes $x_0 = 0$ and creates distribution objects for the transitions tr-dist and emissions obs-dist. It then loops over time steps, sampling each x_t given x_{t-1}, observing y_t given x_t, and deterministically calculating z_t. Finally the $z_{1:T}$ are returned as the desired output.

Figure 5.2: Example pseudo-queries for our simplified probabilistic programming setup with corresponding graphical models and joint distributions.

the data terms, defined as the inputs we observe, appear in neither Ξ_t or Λ_s. We now define both **sample** and **observe** from the probability model perspective as adding a factor to the joint distribution which is therefore given by

$$p(x_{1:T}, y_{1:S}|\phi) = \prod_{t=1}^{T} f_t(x_t|\Xi_t) \prod_{s=1}^{S} g_s(y_s|\Lambda_s). \tag{5.1}$$

The two vary in whether they define a new random variable (**sample**) or affect the probability of the execution given particular instances of the other random variables (**observe**). Our presumptions for this simplified setup that no y_s terms are present in the Ξ_t or Λ_s and that we do not condition on internally sampled variables, means that the product of the **sample** terms correspond exactly to our prior $\prod_{t=1}^{T} f_t(x_t|\Xi_t) =: p(x_{1:T}|\phi)$ and that the product of the **observe** terms corresponds exactly to our likelihood $\prod_{s=1}^{S} g_s(y_s|\Lambda_s) =: p(y_{1:S}|x_{1:T}, \phi)$. Consequently, for our simplified setup, each query defines a finite DAG (see Section 4.3) where the conditional relationships are defined through the definitions of f_t and g_s. This breakdown into a prior and likelihood and the equivalence to graphical models will not hold in the more general cases we consider later. Our aim will be to perform inference to provide a characterization of $p(x_{1:T}|y_{1:S}, \phi)$ (or the posterior for some deterministic mapping of $x_{1:T}$), typically in the form of (approximate) samples. Figure 5.2 shows two example queries along with the corresponding graphical models and joint distributions they define.

Other than **sample** and **observe** statements, the rest of our query is, by construction, totally deterministic. Therefore, though it may contain random variables other than $x_{1:T}$, these random variables are deterministic functions of the "raw" random draws $x_{1:T}$ and inputs ϕ and $y_{1:S}$. We can therefore define the outputs of our query as $\Omega := h(x_{1:T}, y_{1:S}, \phi)$ for some deterministic function h. As we explained in Section 2.7, this change of variables means that the density function on Ω, $p(\Omega|y_{1:S}, \phi)$ can have a different form to the posterior implied by our query, namely $p(x_{1:T}|y_{1:S}, \phi)$. Though this is a serious complication in the context of optimization (we may not in general be able to find $\arg\max_\Omega p(\Omega|y_{1:S}, \phi)$ or even evaluate $p(\Omega|y_{1:S}, \phi)$ exactly), it is perfectly acceptable in the context of calculating expectations as the law of the unconscious statistician tells us that

$$\int f(\Omega)p(\Omega|y_{1:S}, \phi)d\Omega = \int f(h(x_{1:T}, y_{1:S}, \phi))p(x_{1:T}|y_{1:S}, \phi)dx_{1:T} \tag{5.2}$$

for implicitly defined reference measures $d\Omega$ and $dx_{1:T}$. One consequence of this is that we can express any expectation calculated by our query as an expectation over $p(x_{1:T}|y_{1:S}, \phi)$ which is fully defined by the joint (5.1). Another is that, provided we are not worried about carrying

out optimization, we do not need to explicitly worry about the implicit measures defined by the definition of our query, other than any potential effects on the inference scheme and the assumption that suitable measures exist [Staton et al., 2016]. In particular, if our aim is to generate samples from $p(\Omega|y_{1:S}, \phi)$ then we can simply generate samples from $p(x_{1:T}|y_{1:S}, \phi)$ and deterministically convert each sample to the space of Ω. In other words, our inference engine does not need to worry about the consequences of changes of variables if our intended output is just a sequence of samples.

An important point to note is that (5.1) shows that all of our `sample` and `observe` statements are exchangeable, in the sense that their order can be moved around and still define the same joint distribution, up to restrictions about all the required variables existing and being in scope. For example, if all variables remain in scope and are not redefined, we can generally move all our `observe` statements to the end of the query without changing the joint distribution. Therefore the query given in Figure 5.2b would define the same model if all the x_t were sampled upfront before making any observations. Nonetheless, the position of the `observe` statements can often be important from the perspective of the performance of the inference engine. This exchangeability result will carry over to the non-simplified case.

5.3.2 A General Probabilistic Programming Setup [Very Advanced Topic]

Perhaps surprisingly, we do not need to do anything to our language to extend it to a universal PPL other than to relax a number of the restrictions made for our simplified case. Namely, we will allow branching, higher order functions, (potentially infinite) recursion, stochastic memoization [Goodman et al., 2008b], conditioning on internally sampled variables, and the use of the "data" inputs $y_{1:T}$ in the definition of our generative model (instead of just allowing them to be observed). One assumption we will make is that our program terminates with probability 1, asserting that any program that does not satisfy this assumption does not define a valid model. We will maintain the syntax of our simplified setup, along with the assumption that functions are pure other than the effect of `sample` and `observe`. An important point to note though for why such a language is universal, is that arbitrary distributions with countable parameters can be defined through a series of uniform $[0, 1]$ draws followed by an arbitrary deterministic mapping—after all, this is effectively how all probability distributions are defined from a measure theoretic point of view.[3]

[3]Interestingly, this viewpoint breaks down for distributions over functions for which measure theory itself somewhat breaks down Heunen et al. [2017]. As such, universal PPSs actually go beyond the standard measure-theoretic view of probability.

Despite their ostensibly modest nature, these generalizations will have a substantial effect on the intuitions relating our universal PPL to the Bayesian framework, the range of models we can define, and the difficulty of performing general purpose inference. For example, as y_s terms can appear in the Ξ_t terms, it can be the case that $p(x_{1:T}|\phi) \neq \prod_{t=1}^{T} f(x_t|\Xi_t)$, such that the latter no longer explicitly corresponds to a conventional definition of a prior. Some variables may change type (e.g. between continuous and discrete) or even not exist depending on the value of other variables. The number of variables may change from one execution to the next and it could even be the case that the number of latent variables is unbounded provided that any possible execution has a finite number of variables is with probability 1, such as is the case for certain Bayesian non-parametric models such as the Dirichlet process [Ferguson, 1973; Teh, 2011; Bloem-Reddy et al., 2017]. Note that the number **`sample`** and **`observe`** statements lexically defined within our query must, for obvious reasons, be finite, but recursion or looping may mean that they are evaluated an infinite number of times. It is also possible for a variable within a query to itself encode an infinite number of parameters, for example, one might include a Gaussian process within the query (see Section 4.6). These complications make it difficult to mathematically reason about the joint distribution defined by a query in a universal PPS, let alone reason about its breakdown into a prior and a likelihood, or represent the query using a graphical model (which might actually not even be possible).

One way to conceptually overcome these difficulties is to reason in terms of ***execution paths***. Even though we might not know upfront which **`sample`** and **`observe`** statements are invoked by a particular query output, if we execute the query in order, the draws of the **`sample`** statements made thus far in an ***execution trace*** uniquely identify which **`sample`** statement will be invoked next and the value of all the variables up to that **`sample`** statement. In other words, given the outcome of the first t evaluated **`sample`** statements, which **`sample`** statement corresponds to the $(t+1)$-th to be evaluated is uniquely defined and the query up to that **`sample`** statement is completely deterministic, including which path to take at each `if` statement, which variables are defined and their values, and the probability arising from the **`observe`** conditioning statements.[4] Thus although the distribution of the query is not necessarily statically determinable, it can still be ***evaluated through execution*** as each **`sample`** statement provides the information we require, in addition to the information from the previous **`sample`** statements, to deterministically evaluate up to the next **`sample`** statement. Consequently, we can evaluate the probability of

[4]Note that our inference algorithm might induce probabilistic behavior at **`observe`** statements, but we can safely ignore this in terms of defining the distribution represented by the query.

any particular trace as we run it, even though it might be challenging to evaluate the probability of a predefined configuration of the variables.

Using this idea of execution paths, we can define an expression for the conditional probability a query defines, albeit in a complex and somewhat abstract manner that only really retains meaning in our calculation through evaluation mindset, by defining the probability of a particular trace. First let λ denote the value of all the inputs to the query (including both parameters and data as we will now have no explicit distinction between the two). Further let n_x and n_y be the number of **sample** and **observe** statements respectively invoked by the trace, noting that n_x and n_y may themselves be random variables and could potentially be unbounded (e.g. n_x could follow a Poisson distribution). We can now define, for any given execution, $x_{1:n_x} = x_1, \ldots, x_{n_x}$ and $y_{1:n_y} = y_1, \ldots, y_{n_y}$ respectively as the outputs of our n_x **sample** statements and observation inputs for our n_y **observe** statements. To avoid complications regarding variable reassignment, we will not consider the x_j as being variables in our program, such that calling $a \leftarrow$**sample**`(dist)` effectively creates an internal protected variable x_j (whose value can never be reassigned) and then immediately assigns the value of x_j to a. All variables generated before x_j are a deterministic function of $x_{1:j-1}$ and λ. Therefore, even though all variables in our program might be random, including the observations y_k, all are deterministically calculable given $x_{1:n_x}$ and λ. Even n_x itself is a deterministic function of the output of the **sample** statements, as $\{x_{1:j}, \lambda\}$ deterministically dictates whether there will be any further **sample** statements encountered. Consequently, in the same way we only needed to worry about $x_{1:n_x}$ to reason about the distribution over the program outputs for our simplified case, we can reason about the distribution on all variables in program (for a given λ) through considering $x_{1:n_x}$ in our general case. We can thus think of our trace as being defined by $x_{1:n_x}$.

We continue by defining $f_1, \ldots, f_{n_s}$ and $g_1, \ldots, g_{n_o}$ respectively as the densities (with respect to an implicitly defined reference measure) associated with the n_s **sample** and n_o **observe** statements defined lexically within the program (i.e. the distinct **sample** and **observe** statements appearing anywhere in the raw program code), noting that n_s and n_o are ***not*** random variables. To express these densities, we use the notation $f_i(x_j|\eta_j)$ to indicate the density of lexical **sample** statement i returning outputs x_j when provided with distribution object η_j, which will typically be a random variable itself. Note here that parameters of the density are encoded through the distribution object, so, for example, we can think of **sample** (**normal** $(\mu, 1)$) for a random variable μ as first creating the random distribution object $\eta_j =$**normal** $(\mu, 1)$ before passing this to the **sample** call. We similarly use the notation $g_i(y_k|\psi_k)$ to express the density of using

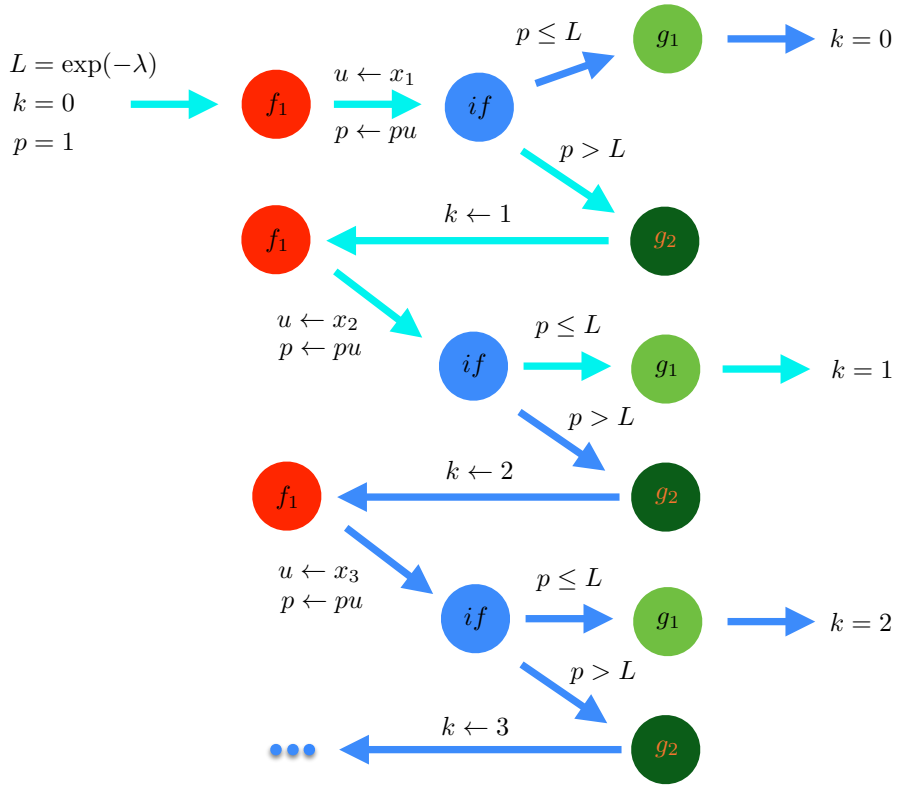
lexical **observe** g_i to observe output y_k with the (random) distribution object ψ_k. We define $a_j \in \{1, \ldots, n_s\},\ \forall j \in \{1, \ldots, n_x\}$ and $b_k \in \{1, \ldots, n_o\},\ \forall k \in \{1, \ldots, n_y\}$ as the random variables respectively used to index which of the lexical **sample** and **observe** statements correspond to the j^{th} and k^{th} execution trace **sample** and **observe** statements.

To encapsulate the notion of a valid trace, i.e. a $x_{1:n_x}$ that can be generated by the program and for which all the density terms are well-defined, we introduce the deterministic boolean function $\mathcal{B}(x_{1:n_x}, \lambda)$ which returns 1 if the trace is valid and 0 otherwise. This is something we do not need to worry about if we take an evaluation based inference approach, whereby we rely on methods that only propose from the generative model, but it is necessary to ensure that the density is well defined if we try to evaluate it at a predetermined point, chosen externally to the program. For example, $\mathcal{B}(x_{1:n_x}, \lambda)$ is necessary to ensure that $x_{1:n_x}$ is a "complete" trace: it may be that a certain $x_{1:n_x}$ implies that further **sample** statements are still to be invoked after the n_x^{th}, in which case the trace is not valid as these additional outputs are undefined. For example, if the program defines the distribution $\mathcal{N}(x_1; 0, 1)\mathcal{N}(x_2; 0, 1)\mathcal{N}(4; x_2, x_1)$, then the trace $x_1 = 3.2$ is incomplete. Similarly, we can use $\mathcal{B}(x_{1:n_x}, \lambda)$ to encode that in our example $P(n_x > 2) = 0$. Meanwhile, $\mathcal{B}(x_{1:n_x}, \lambda)$ also ensures that all terms within the trace probability are well defined. For example, an invalid trace might provide parameters of the wrong type to one of the distribution objects, meaning that density of that distribution object is undefined.

We are now finally ready to define the conditional distribution on the trace $\mathcal{T}$ implied by our query as $p(\mathcal{T} = x_{1:n_x}|\lambda) \propto \gamma(x_{1:n_x}, \lambda)$ where

$$\gamma(x_{1:n_x}, \lambda) = \begin{cases} \prod_{j=1}^{n_x} f_{a_j}(x_j|\eta_j) \prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k) & \text{if} \quad \mathcal{B}(x_{1:n_x}, \lambda) = 1 \\ 0 \quad \text{otherwise} \end{cases}, \tag{5.3}$$

remembering that although the a_j, η_j, etc are random variables, they are all deterministically calculable from $x_{1:n_x}$ for a given query and λ. As a consequence, our program implies a well defined, normalized, conditional distribution, or "posterior", on the produced traces (presuming the normalization constant is finite and non-zero). Note that this definition is explicitly on outcomes of the trace and so, in general, $p(\mathcal{T} = x_{1:j}|\lambda) \neq \int p(\mathcal{T} = x_{1:n_x}|\lambda) dx_{j+1:n_x}$. In fact, because $x_{1:j}$ deterministically dictates whether x_{j+1} exists, it is not possible to have $\mathcal{B}(x_{1:n_x}, \lambda) = \mathcal{B}(x_{1:j}, \lambda) = 1$, i.e. we cannot have that both $x_{1:j}$ and $\{x_{1:j}, x_{j+1:n_x}\}$ are complete traces. Consequently, it is always necessary that at least one of $p(\mathcal{T} = x_{1:j}|\lambda)$ and $p(\mathcal{T} = x_{1:n_x}|\lambda)$ are equal to zero. Note though that we can still define a program that places non-zero weight on, for example, both the ***outputs*** $[2.3, 3.5]$ and $[2.3, 3.5, 1.2]$, remembering that $x_{1:n_x}$ relates to the ***raw draws*** from the **sample** statements.

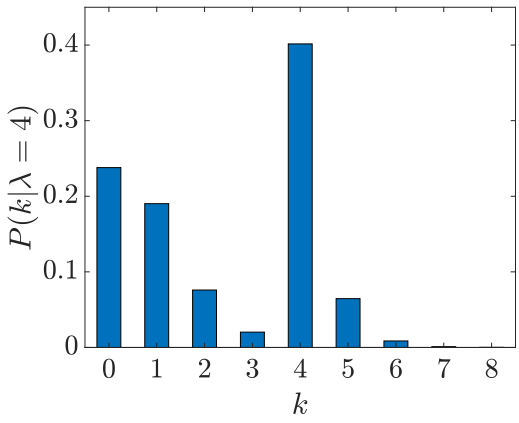


(a) Possible traces for our query. **`sample`** statements are shown in red, **`observe`** statements are shown in shades of green, and deterministic computations (given the outputs of the **`sample`** statements) are shown in shades of blue. Subscripts for the sample and observe statements show the lexical index (i.e. $a_j \in \{1\}$ and $b_k \in \{1, 2\}$), as do the different shades. The cyan arrows correspond to a particular execution path which gives output $k = 1$ and has $n_x = 2$, $n_y = 2$, $b_1 = 2$, and $b_2 = 1$. Valid traces for this particular path must have $x_1 > \exp(-\lambda)$ and $x_2 \leq \exp(-\lambda)/x_1$.

Inputs: Event rate λ

$L \leftarrow \exp(-\lambda),\ k \leftarrow 0,\ p \leftarrow 1$
while $p > L$ **do**
  $u \leftarrow$ **`sample`** (**`uniform`** (0,1))
  $p \leftarrow pu$
  if $p \leq L$ **then**; **break while**; **end if**
  `observe` (**`bernoulli`** (0.2), 1)
  $k \leftarrow k + 1$
end while
`observe` (**`bernoulli`** (0.99), $\mathbb{I}(k{>}3)$)
return k

(b) Query code for warped Poisson sampler



(c) Conditional distribution on k for $\lambda = 4$.

Figure 5.3: Demonstration of stochastic execution traces using a warper Poisson sampler, adapted from [Paige, 2016, Figure 3.3]. The query defined in **(b)** would output k as per a Poisson distribution with event rate λ if it were not for the **`observe`** statements. As shown in **(c)** though, these **`observe`** statements warp the distribution to give a lighter tail while discouraging $k \leq 3$. Here both $n_x \in \mathbb{N}^+$ and $n_y \in \mathbb{N}^+$ depend on the trace and are unbounded. Which **`observe`** we see first is also probabilistic, while some traces will be invalid, e.g. $x_{1:n_x} = [1, 0.5]$ as $x_1 = 1$ indicates there will be only a single **`sample`** encountered. Nonetheless, we can calculate the probability of a trace by following its path to completion while accumulating **`sample`** and **`observe`** factors.

An illustrative example for calculating the probability of a program through execution traces is shown in Figure 5.3. This corresponds to a problem that cannot not be expressed using our simplified PPL or a graphical model, but which is simple to write in our universal framework. Imagine we want to calculate the unnormalized trace probability $\gamma(x_{1:n_x} = [0.2, 0.07], \lambda = 4)$ which we can do by stepping through the program and accumulating terms (for reference we will be following the cyan path in Figure 5.3a). On our path we first hit $f_1(x_1|\eta_1) = \textsc{Uniform}(0.2; 0, 1) = 1$, we fix $u \leftarrow x_1$ and $p \leftarrow 1 \cdot u$, and test if $p = 0.2 \leq \exp(-4) \approx 0.0183$. This gives false and so we do not break the while loop, instead encountering $g_2(y_1|\psi_1) = \textsc{Bernoulli}(1; 0.2) = 0.2$ and so our running value for $\gamma(x_{1:n_x} = [0.2, 0.07], \lambda = 4)$ is $1 \times 0.2 = 0.2$. Updating $k \leftarrow 1$ and going back to the start of the while loop we encounter $f_1(x_2|\eta_2) = \textsc{Uniform}(0.07; 0, 1) = 1$, so our running score is $1 \times 0.2 \times 1 = 0.2$. Reassigning u and p we see that $p = 0.014 \leq \exp(-4)$ is now true and so we take the opposite branch to before with our if statement, thus breaking the while loop. To finish the program we pass through $g_1(y_2|\psi_2) = \textsc{Bernoulli}(k > 3; 0.99) = 0.01$ giving the final unnormalized density of $\gamma(x_{1:n_x} = [0.2, 0.07], \lambda = 4) = 0.01 \times 0.2 = 0.002$ and an output of $k = 1$. We finish by checking that our trace was valid, which we can easily see is the case because no terms were undefined and we generated the correct number of sample outputs (i.e. $n_x = 2$ as required).

An important point of note is that, in general, $\gamma(x_{1:n_x}, \lambda)$ is not a normalized joint distribution. This is firstly, and most obviously, because λ might contain terms, referred to as ϕ before, that are not observed and so have no implied density. Secondly, and more critically, even if $\phi = \emptyset$, $\gamma(x_{1:n_x}, \lambda)$ is not necessarily correctly normalized, because of the ability to observe sampled variables and condition the **`sample`** statements on the observations. As a simple example, our model might consist of a $x_1 \leftarrow$**`sample`** (**`normal`** (0,1)) term followed by an **`observe`** (**`normal`** $(-1, 2, x_1)$) term. This does not directly define any properly normalized joint distribution on any particular variables (noting that $\mathcal{N}(x_1; 0, 1)\mathcal{N}(x_1; 0, 2)$ is an unnormalized distribution with only the variable x_1). Consequently, there is no means of writing down a normalized joint distribution for a general query in our universal PPL setup in closed form—we might actually need to empirically estimate the normalization constant to evaluate the joint. This actually steps outside the conventional Bayesian modeling framework and raises a number of interesting theoretical questions. However, from a practical perspective, we can note that provided the implicitly defined normalization constant is finite, the query also implicitly defines a correctly normalized conditional distribution (noting that $\gamma(x_{1:n_x}, \lambda) \geq 0$). This is analogous to knowing

the joint but not the posterior in Bayesian inference, though it is not exactly equivalent because the normalization constant is no longer the marginal likelihood.

For a simple example of why it is important to be able to define models up to an unnormalized joint distribution, consider a model where a and b are each sampled from discrete distributions and we want to condition on the value of a and b being equal. Here the combination of the sample statements and the observation that the two are equal clearly does not lead to a correctly normalized joint (we do not even really have a conventional notion of a likelihood), but it clearly defines an unnormalized conditional distribution as

$$P(a, b|\mathbb{I}(a = b)) = \frac{P(a)P(b|a)\mathbb{I}(a = b)}{\sum_a \sum_b P(a)P(b|a)\mathbb{I}(a = b)}.$$

In such cases where our observation is a hard constraint, we can view the normalizing constant for the conditional defined by the query as the probability of our constraint being satisfied. For more general queries of discrete variables, we might have, for example,

$$P(x|\lambda) \propto f(x|\lambda)g(y = \kappa(x, \lambda)|x, \lambda)$$

for some deterministic function κ. Here we can view the marginalization as being the probability of the event $y = \kappa(x, \lambda)$ under the joint $P(x, y) = f(x|\lambda)g(y|x, \lambda)$. For our previous example we have $x := a$, $y := b$, $\kappa(a, \lambda) := a$, $f(a) := p(a)$, and $g(b = a) := p(b = a|a)$. The same intuition applies to continuous cases where we now have the density of the event $y = \kappa(x, \lambda)$. We can also think of $f(x|\lambda)g(\kappa(x, \lambda)|x, \lambda)$ as defining an unnormalized distribution on x whose normalization constant is $\int f(x|\lambda)g(\kappa(x, \lambda)|x, \lambda)dx$. In general, our normalization constant will be a combination of conventional marginal likelihood terms and contributions from these "doubly defined" terms. We refer to this normalization constant as the ***partition function***[5] and note that it is given by

$$\begin{aligned} Z(\lambda) &= \mathbb{E}\left[\prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k)\middle|\lambda\right] = \int \gamma(x_{1:n_x}, \lambda)dx_{1:n_x} \\ &= \int_{x_{1:n_x} \in \{X:\mathcal{B}(X,\lambda)=1\}} \prod_{j=1}^{n_x} f_{a_j}(x_j|\eta_j) \prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k)dx_{1:n_x} \end{aligned} \tag{5.4}$$

where the expectation is under running the query forwards (i.e. the distribution implied by simulating from an equivalent query with all the `observe` statements removed) and all terms in the integral are deterministically calculable for the query given λ and $x_{1:n_x}$. For our query

[5]Note this is not a term that is usually used in the probabilistic programming literature, where it is usually just referred to as a marginal likelihood. Similarly, it is common in the literature to refer to a query as defining a "normalized joint" density of $p_\phi(x_{1:n_x}, y_{1:n_y}) = \prod_{j=1}^{n_x} f_{a_j}(x_j|\eta_j) \prod_{k=1}^{n_y} g_{b_k}(y_k|\psi_k)$. We have deviated from both of these because, as our examples demonstrate, this viewpoint is actually an approximation. For an even more rigorous treatment of the distributions defined by PPSs, we refer the reader to Staton et al. [2016].

to represent a well-defined conditional distribution, it is necessary to have $0 < Z(\lambda) < \infty$. Although $Z(\lambda)$ does not correspond exactly to a marginal likelihood, we can still think of it in terms of representing a ***model evidence*** in the same way, it just might not be a correctly normalized density in the same way a marginal likelihood is.

We now see that we can draw a direct analogy to the Bayesian framework whereby the product of the `sample` terms is analogous to the prior, the product of the `observe` terms is analogous to the likelihood, and the partition function is analogous to the marginal likelihood. If observed variables are not sampled within or used elsewhere in the query (e.g. being used as parameters in distribution objects later used for sampling), then this analogy becomes exact as per our simplified setup. However, as we have explained, it will often be desirable to go beyond this framework to specify some models. When we do, we still have an implicit Bayesian model, in the same way that Bayes' rule means that a prior and a likelihood implicitly define a posterior, but we may not actually have access to our implicitly defined prior and likelihood.

Given our query is now defined to specify an unnormalized conditional distribution rather than a normalized joint distribution, it is natural to ask whether it is necessary for each `observe` term to correspond to a correctly normalized density for its output, instead of just restricting it to be a positive semi-definite function representing an arbitrary soft constraint. The answer is that it is not. In fact, from a theoretical perspective, it perhaps easier to not think of the density defined by the query as being a combination of sampling and conditioning terms, but instead the product of a correctly normalized generative model density and a positive semi-definite ***score function*** that applies an unnormalized weighting to any possible sample the generative model can produce. This is exactly the approach taken by [Staton et al., 2016] and [Goodman and Stuhlmüller, 2014] who explicitly use such score functions, calling them `score` and `factor` respectively.

5.4 Further Reading

The main recommendation for further reading here is to go investigate some individual probabilistic programming systems—try searching for some of those listed in Section 5.2. I would recommend Pyro as a system that covers at lot of the relevant ideas, is relatively easy to use, and is well documented.

- Video tutorial on probabilistic programming by Frank Wood: `https://www.youtube.com/watch?v=Te7A5JEm5UI&t=500s`
- Full conference of talks on probabilistic programming: `https://www.youtube.com/channel/UCTFDb7aQY1ewBYwJJrpKp6Q`

6

Foundations of Bayesian Inference and Monte Carlo Methods

In the previous chapters we introduced the concept of Bayesian modeling and showed how we can combine prior information $p(\theta)$ and a likelihood model $p(\mathcal{D}|\theta)$ using Bayes' rule (i.e. (3.6)), to produce a posterior $p(\theta|\mathcal{D})$ on variables θ that characterizes both our prior information and information from the data $\mathcal{D}$. We now consider the problem of how to calculate (or more typically approximate) this posterior, a process known as ***Bayesian inference***. At first, this may seem like a straightforward problem: by Bayes' rule we have that $p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$ and so we already know the relative probability of any one value of θ compared to another. In practice, this could hardly be further from the truth. Bayesian inference for the general class of graphical models is, in fact, an NP-hard problem [Cooper, 1990; Dagum and Luby, 1993]. In this chapter, we will outline these challenges and introduce foundational methods for overcoming them in the form of ***inference algorithms***. We will focus, in particular, on ***Monte Carlo*** methods, for which we will introduce some key underlying results, before introducing a number of foundational methods that form the building blocks for the more advanced strategies discussed in the next chapter.

6.1 The Challenge of Bayesian Inference

We can break down Bayesian inference into two key challenges: calculating the normalization constant $p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta)d\theta$ and providing a useful characterization of the posterior, for example, a set of approximate samples. Interestingly, each of these constitutes a somewhat distinct problem and many inference methods focus only on solving the latter problem. However, this breakdown will still prove useful in illustrating the intuitions about the difficulties presented by Bayesian inference.

6.1.1 The Normalization Constant

Calculating the normalization constant in Bayesian inference is essentially a problem of integration. Our target, $p(\mathcal{D})$, is the expectation of the likelihood under the prior, hence the name ***marginal likelihood***. When $p(\mathcal{D})$ is known, the posterior can be evaluated exactly at any possible input point using (3.6) directly. When it is unknown, we lack a scaling in the evaluation of

any point and so we have no concept of how relatively significant that point is relative to the distribution as a whole. For example, for a discrete problem then if we know the normalization constant, we can evaluate the exact probability of any particular θ by evaluating that point alone. If we do not know the normalization constant, we do not know if there are other substantially more probable events that we have thus–far missed, which would, in turn, imply that the queried point has a negligible chance of occurring.

To give a more explicit example, consider a model where $\theta \in \{1, 2, 3\}$ with a corresponding uniform prior $P(\theta) = 1/3$ for each θ. Now presume that for some reason we are only able to evaluate the likelihood at $\theta = 1$ and $\theta = 2$, giving $p(\mathcal{D}|\theta = 1) = 1$ and $p(\mathcal{D}|\theta = 2) = 10$ respectively. Depending on the marginal likelihood $p(\mathcal{D})$, the posterior probability of $P(\theta = 2|\mathcal{D})$ will vary wildly. For example, $p(\mathcal{D}) = 4$ gives $P(\theta = 2|\mathcal{D}) = 5/6$, while $p(\mathcal{D}) = 1000$ gives $P(\theta = 2|\mathcal{D}) = 1/100$.

Though this example may seem far-fetched, this lack of knowledge of the marginal likelihood is almost always seen in practice for realistic models, at least those with non-trivial solutions. Typically it is not possible to enumerate all the possible values of θ in a reasonable time and we are left wondering: how much probability mass is left that we have not seen? The problem is even worse in the setting where θ is continuous, for which it is naturally impossible to evaluate all possible values for θ. Knowing the posterior only up to a normalization constant is deceptively unhelpful: we never know how much of the probability mass we have missed and therefore whether the probability (or probability density) where we have looked so far is tiny compared to some other dominant region we are yet to explore. At its heart, the problem of Bayesian inference is a problem of where to concentrate our finite computational resources so that we can effectively characterize the posterior. If $p(\mathcal{D})$ is known, then we generally have some idea of whether we are looking in the right place or whether there are places left to look that we are yet to find.[1] This brings us onto our second challenge: knowing the posterior in closed form is often not enough.

6.1.2 Characterizing the Posterior

Once we have the normalization constant, it might seem that we are done; after all, we now have the exact form of the posterior using Bayes' rule. Unfortunately, it tends to be the case, particularly when θ is continuous, that this is insufficient to carry out most tasks that we might

[1] For continuous problems, it might still be difficult to fully calibrate this even when $p(\mathcal{D})$ is known because we may not know what the effective scaling of the input space is. For example, different problem parameterizations will lead to different posterior densities.

want to use our posterior for. There are a number of different, often overlapping, reasons for wanting to calculate a posterior including

- To calculate the posterior probability or probability density for one or more particular instances of the variables.
- To calculate the expected value of some function, $\mu_f = \mathbb{E}_{p(\theta|\mathcal{D})}[f(\theta)]$. For example, we might want to calculate the expected values of the variables themselves $\mu_\theta = E_{p(\theta|\mathcal{D})}[\theta]$.
- To make predictions as per the posterior predictive distribution introduced in Section 3.3.2.
- To find the most probable variable values $\theta^* = \arg\max_\theta p(\theta|\mathcal{D})$. This is known as maximum a posteriori estimation.
- To produce a useful representation of the posterior, e.g. a set of samples, for passing on to another part of a computational pipeline or to be directly observed by a user.

If θ is continuous, or some elements of θ are continuous, then only the first of these can be carried out directly using the form of the posterior provided by Bayes' rule with known normalization constant. We see, therefore, that knowing the normalization alone will not be enough to fully solve the Bayesian inference problem in a useful manner. In particular, it will generally not be sufficient in order to be able to ***sample*** from the posterior. As we will see later, the ability to sample will be at the core of most practical uses for the posterior as it allows use of Monte Carlo methods [Metropolis and Ulam, 1949; Robert, 2004; Rubinstein and Kroese, 2016], which can, in turn, be used to carry out many of the outlined tasks.

To further demonstrate why knowing the normalization constant is insufficient for most Bayesian inference tasks, we consider the following simple example

$$p(\theta) = \textsc{Gamma}(\theta; 3, 1) = \frac{\theta^2 \exp(-\theta)}{2}, \quad \theta \in (0, \infty), \tag{6.1a}$$

$$p(y = 5|\theta) = \textsc{Student-t}(\theta - 5; 2) = \frac{\Gamma(1.5)}{\sqrt{2\pi}} \left(1 + \frac{(\theta - 5)^2}{2}\right)^{-3/2}, \tag{6.1b}$$

$$p(\theta|y = 5) \approx 5.348556\, \theta^2 \exp(-\theta) \left(2 + (5 - \theta)^2\right)^{-3/2}. \tag{6.1c}$$

Here we have that the prior on θ is distributed according to a gamma distribution with shape parameter 3 and rate parameter 1. The likelihood function is a student-t distribution on the difference between θ and the output $y = 5$. Using a numerical integration over θ, the normalization constant can be calculated to a high accuracy, giving the provided closed-form equation for the posterior shown in (6.1c). This posterior, along with the prior and likelihood are shown Figure 6.1.

Here knowing the marginal likelihood means that we have a closed-form equation for the posterior. Imagine though that we wish to sample from it. As it does not correspond to a standard distribution, there is, in fact, no way to directly sample from it without doing further calculations. For example, if we also know the inverse of the cumulative density function of the posterior

$$P(\Theta \leq \theta | y = 5) = \int_{\Theta=0}^{\Theta=\theta} p(\theta = \Theta | y = 5) d\Theta, \quad (6.2)$$

then we can sample from the posterior by first sampling $\hat{u} \sim \text{UNIFORM}(0, 1)$ and then taking as our exact sample $\hat{\theta} = P^{-1}(\hat{u})$ such that $\hat{u} = P(\Theta \leq \hat{\theta} | y = 5)$. However, the cumulative distribution function and its inverse cannot, in general, be calculated analytically. Though in this simple one-dimensional problem they can be easily estimated numerically, this will prove prohibitively difficult for most problems where θ has more than a few dimensions. Similarly, if we wish to estimate an expectation with respect to this posterior we could do this relatively easily numerically for this simple problem, for example using Simpson's rule, but in higher dimensions, this will be impractical.

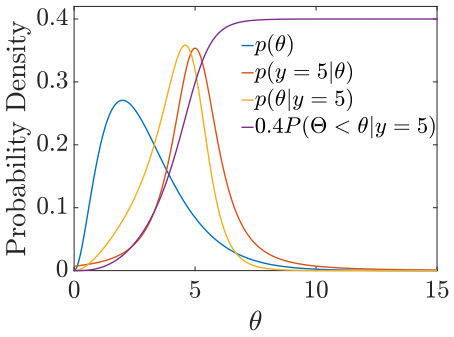


Figure 6.1: Example inference for problem given in (6.1). Also shown is the cumulative distribution for the posterior as per (6.2). This is scaled by a factor of 0.4 for visualization.

There are a number of indirect methods we could use instead to sample from the posterior such as rejection sampling, importance sampling, and MCMC. However, as we will show in Section 6.4, these all only require that we can evaluate an unnormalized version of target distribution, such that they side-step the need to calculate the marginal likelihood. Nonetheless, knowledge of the marginal likelihood can still be helpful in a number of scenarios (e.g. in adapting our inference algorithm) for the reasons outlined in Section 6.1.1.

6.2 Deterministic Approximations

One of the simplest approaches to Bayesian inference is to effectively ignore the problem completely and instead resort to a heuristic approximation. The simplest such deterministic approximation is just to take a ***point estimate*** $\tilde{\theta}$ for θ and then approximate the posterior predictive distribution using only this value:

$$p(\mathcal{D}^*|\mathcal{D}) \approx p(\mathcal{D}^*|\tilde{\theta}). \quad (6.3)$$

In general, finding $\tilde{\theta}$ requires only an ***optimization*** problem to be solved, which is generally far easier than the ***integration*** problem posed by estimating the full posterior predictive or an

expectation with respect to the posterior. The two most ways of choosing a $\tilde{\theta}$ are to take either a ***maximum likelihood*** (ML) estimate or ***maximum a posteriori*** (MAP) estimate.

ML estimation is not a Bayesian approach at all: it completely ignores the prior and instead tries to find the parameters that maximize the likelihood. Namely it takes

$$\tilde{\theta}_{\text{ML}} = \arg\max_{\theta \in \vartheta} p(\mathcal{D}|\theta). \tag{6.4}$$

Many classical statistical and machine learning techniques on based on ML estimation as it is intuitively the most probable set of parameters given data and a model. However, this can be prone to overfitting and so it typically requires some form of regularization [Hastie et al., 2001]. Many of the relative advantages and disadvantages of maximum likelihood estimation relative to a Bayesian approach are the same of those of frequentist versus Bayesian methods as discussed in Section 3.4 (note though that ML estimation is far from the only frequentist approach). In particular, it fails to incorporate prior information (thing back to the sun exploding example in the lectures) and fails to capture uncertainty in θ.

MAP estimation corresponds to maximizing the posterior probability, which is equivalent to extending ML estimation to also include a prior term, noting that $p(\mathcal{D})$ is a constant, such that we have

$$\tilde{\theta}_{\text{MAP}} = \arg\max_{\theta \in \vartheta} p(\theta|\mathcal{D}) = \arg\max_{\theta \in \vartheta} p(\mathcal{D}|\theta)p(\theta). \tag{6.5}$$

This provides regularization compared to ML estimation (indeed many priors and ML regularization methods are exactly equivalent [Bishop, 2006]), but still has a number of drawbacks compared to full inference when the latter is possible. For example, the position of the MAP estimate is dependent of the parametrization of the problem (see Section 2.7). Using a MAP estimate also, of course, incorporates less information into the predictive distribution than using a fully Bayesian approach. Nonetheless, MAP estimation is still an important tool of Bayesian machine learning, necessary when a single estimate is required because the parameter has some real world meaning we are trying to learn about, or when inference is infeasible. Note that when ϑ is bounded, then ML estimation is recovered from MAP estimation by using a uniform prior.

When θ is continuous and $p(\theta|\mathcal{D})$ is twice differentiable, the MAP estimate can often be refined by instead using the ***Laplace approximation*** of the posterior. This is based on approximating the posterior with Gaussian centered at the MAP estimate, with covariance dictated by the curvature of the density around this point. Specifically it uses

$$p(\theta|\mathcal{D}) \approx \mathcal{N}\left(\theta; \tilde{\theta}_{\text{MAP}}, \left(\Lambda_{\text{MAP}}\right)^{-1}\right) \tag{6.6}$$

where Λ_{MAP} is the negative Hessian of the log joint density evaluated at the MAP, i.e.

$$\Lambda_{\text{MAP}} = -\nabla^2_\theta \log\left(p(\theta, \mathcal{D})\right)|_{\theta=\tilde{\theta}_{\text{MAP}}}. \tag{6.7}$$

The derivation of this stems from taking a Taylor expansion of $\log p(\theta|\mathcal{D})$ about $\tilde{\theta}_{\text{MAP}}$, see Bishop [2006, Section 4.4] for more details. The Laplace approximation can be quite effective if the true posterior is close to Gaussian, which is actually the case more often than one might expect when their are a large number of observations (due to the Bernstein–von Mises theorem as per 4.2)). However, it, unsurprisingly, performs poorly when this is not the case, particularly when the posterior is not unimodal.

6.3 Monte Carlo

In many scenarios, the deterministic posterior approximations discussed in the last section are not sufficient. In particular, for cases where uncertainty is important or where a significant proportion of the posterior mass is in its tails, these approaches can lead to highly unsatisfactory solutions. In the rest of this chapter we therefore consider more a flexible class of methods that allow arbitrary good approximation of the posterior: ***Monte Carlo*** methods. We start by introducing the concept of Monte Carlo itself and some of its fundamental properties, before moving onto how it can be used for Bayesian inference in the next section.

Monte Carlo [Metropolis and Ulam, 1949] is the characterization of a probability distribution through random sampling; it forms the underlying principle for all stochastic computation. It is the foundation for a huge array of methods for numerical integration, optimization, and, most notably for our purposes, Bayesian inference. Monte Carlo provides us with a means of dealing with complex models and problems in a statistically principled manner. As we will show, it is a highly composable framework that will allow the output of one system to be input directly to another. For example, the Monte Carlo samples from a joint distribution will also have the correct marginal distribution over any of its individual components, while sampling from the marginal distribution then sampling from the conditional distribution given these samples, will give samples distributed according to the joint. These characteristics prove to be hugely important as they mean that Monte Carlo can be used as a mechanism for ***unbiasedly*** propagating information: passing our beliefs through samples allows us to avoid the ***flaw of averages***. In other words, if we take some fixed approximation of a random variable (e.g. its mean) and pass this onto another part of the computational pipeline this induces biases through the fact that, in

general, $f(\mathbb{E}[\theta]) \neq \mathbb{E}[f(\theta)]$ [Rainforth et al., 2018]. The composability of Monte Carlo means that this can be avoided if we avoid making approximations and pass on samples instead.

The critical importance of Monte Carlo estimation stems from the fact that most of the example target tasks laid out in 6.1.2 can be formulated as ***expectations***. Even when our intention is simply to generate samples from a target distribution, we can usually think of this as being an implicit expectation of an, as yet unknown, target function. Here our implicit aim is to minimize the bias and variance of whatever process the samples are eventually used for, even if that process is simply visual inspection.

As such, we will, for now, digress from directly thinking about how to approximate the posterior and instead think about the ***implications*** of approximating it with a set of samples. Through this we will demonstrate what we desire from our approximations, before returning to how we can do this.

6.3.1 Monte Carlo Estimates

Consider the problem of calculating the expectation of some function $f(\theta)$ under the distribution $\theta \sim \pi(\theta)$ ($= p(\theta|\mathcal{D})$ for the Bayesian inference case), which we will denote as

$$I := \mathbb{E}_{\pi(\theta)}\left[f(\theta)\right] = \int f(\theta)\pi(\theta)d\theta. \tag{6.8}$$

This can be approximated using the ***Monte Carlo estimator*** I_N where

$$I \approx I_N := \frac{1}{N}\sum_{n=1}^{N} f(\hat{\theta}_n) \quad \text{and} \quad \hat{\theta}_n \sim \pi(\theta) \tag{6.9}$$

are independent draws from $\pi(\theta)$. In other words, the Monte Carlo estimator estimates an expectation by making a number of draws from the reference distribution, evaluating the target function for each of these samples, then taking the empirical average of these evaluations.

The first result we note is that (6.9) is an ***unbiased*** estimator for I, i.e. we have

$$\mathbb{E}\left[I_N\right] = \mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N} f(\hat{\theta}_n)\right] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\left[f(\hat{\theta}_n)\right] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\left[f(\hat{\theta}_1)\right] = I \tag{6.10}$$

where we have first moved the sum outside of expectation using linearity,[2] then the fact that each $\hat{\theta}_n$ is identically distributed to note that each $\mathbb{E}\left[f(\hat{\theta}_n)\right] = \mathbb{E}\left[f(\hat{\theta}_1)\right]$, and finally that $\mathbb{E}\left[f(\hat{\theta}_1)\right] = I$ by the definition of I and the distribution on $\hat{\theta}_1$. This is an important result as it means that Monte Carlo does not introduce any systematic error, i.e. ***bias***, into the approximation: in expectation, it does not pathologically overestimate or underestimate the target. This is not

[2]Note that this presumes that N is independent of the samples. This is usually the case, but care is necessary in some situations, namely when the number of samples taken is adaptively chosen based on the sample values.

to say though that it is equally likely to overestimate or underestimate as it may, for example, typically underestimate by a small amount and then rarely overestimate by a large amount. Instead, it means that if we were to repeat the estimation an infinite number of times and average the results, we would get the true value of I. This now hints at another important question: do we also recover the true value of I when we conduct one infinitely large estimation, namely if we take $N \to \infty$? This is known as ***consistency*** of a statistical estimator, which we will now consider next.

6.3.2 The Law of Large Numbers

A key mathematical idea underpinning many Monte Carlo methods is the ***law of large numbers*** (LLN). Informally, the LLN states that the empirical average of ***independent and identically distributed*** (i.i.d.) random variables converges to the true expected value of the underlying process as the number of samples in the average increases. We can, therefore, use it to prove the consistency of Monte Carlo estimators where the samples are drawn independently from the same distribution. The high-level idea for the LLN can be shown by considering the ***mean squared error*** of a Monte Carlo estimator as follows

$$\begin{aligned}
\mathbb{E}\left[(I_N - I)^2\right] &= \mathbb{E}\left[\left(\frac{1}{N}\sum_{n=1}^{N} f(\hat{\theta}_n) - I\right)^2\right] = \frac{1}{N^2}\mathbb{E}\left[\left(\sum_{n=1}^{N}\left(f(\hat{\theta}_n) - I\right)\right)^2\right] \\
&= \frac{1}{N^2}\sum_{n=1}^{N}\mathbb{E}\left[\left(f(\hat{\theta}_n) - I\right)^2\right] + \frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1,m\neq n}^{N}\mathbb{E}\left[(f(\hat{\theta}_n) - I)(f(\hat{\theta}_m) - I)\right] \\
&= \frac{1}{N^2}\sum_{n=1}^{N}\mathbb{E}\left[\left(f(\hat{\theta}_1) - I\right)^2\right] + \frac{1}{N^2}\sum_{n=1}^{N}\sum_{m=1,m\neq n}^{N}\cancelto{0}{\left(\mathbb{E}\left[(f(\hat{\theta}_1) - I)\right]\right)^2} \\
&= \frac{\sigma_\theta^2}{N} \quad \text{where} \quad \sigma_\theta^2 := \mathbb{E}\left[\left(f(\hat{\theta}_1) - I\right)^2\right] = \mathrm{Var}\left[f(\theta)\right].
\end{aligned} \tag{6.11}$$

Here the second line follows from the first simply by expanding the square and using linearity to move the sum outside of the expectation as in the unbiasedness derivation. The first term in the third line follows from the equivalent term in the second line by again noting that each $\hat{\theta}_n$ has the same distribution. The second term in the third line follows from the assumption that the samples are drawn independently such that

$$\mathbb{E}\left[(f(\hat{\theta}_n) - I)(f(\hat{\theta}_m) - I)\right] = \mathbb{E}\left[(f(\hat{\theta}_n) - I)\right]\mathbb{E}\left[(f(\hat{\theta}_m) - I)\right] = 0.$$

by unbiasedness of the estimator. The last line simply notes that $\mathbb{E}\left[\left(f(\hat{\theta}_1) - I\right)^2\right]$ is a constant, namely the variance of $f(\theta)$ when $\theta \sim \pi(\theta)$.

Our final result has a simple and intuitive form:the mean squared error for our estimator using N samples is $1/N$ times the mean squared error of an estimator that only uses a single

sample, which is itself equal to the variance of $f(\hat{\theta})$. As $N \to \infty$, we thus have that our expected error goes to 0.

A key upshot of this result is that the difference between our empirical estimate and the true value (i.e. $I_N - I$) should be of order $O(1/\sqrt{N})$. In some ways this is rather slow: deterministic numerical integration schemes often have much faster theoretical convergence rates. For example, Simpson's rule has a convergence rate of $O(1/N^4)$ for one-dimensional functions [Owen, 2013, Chapter 7]. As such, Monte Carlo is often an inferior way of estimating integrals for smooth functions in low dimensions. However, these deterministic numerical integration schemes require smoothness assumptions on f and, more critically, their convergence rates diminish rapidly (typically exponentially) with the dimensionality. By comparison, the dimensionality only affects the Monte Carlo convergence rate through changes in the constant factor σ_θ and though this will typically increase with the dimensionality, this scaling will usually be substantially more graceful than deterministic numerical methods.

6.3.3 The Central Limit Theorem [Advanced Topic]

The ***central limit theorem*** (CLT) is another core result in the study of Monte Carlo methods. In its simplest form, it states that the empirical mean of N i.i.d. random variables tends towards a normal distribution in the limit $N \to \infty$. While the LLN demonstrated the convergence of the average of i.i.d. random variables towards the true mean, the CLT gives us information about how this convergence occurs. Furthermore, it has variants that do not require that the variables are i.i.d., meaning we can use it do demonstrate convergence in scenarios where samples are correlated, such as occurs when doing MCMC inference as we will consider in the next chapter. In the i.i.d. sampling case, the CLT is as follows.

Theorem 6.1 (Central Limit Theorem)**.** *Assume $X_1, \ldots, X_N$ is a sequence of i.i.d. random variables and let $I_N := \frac{1}{N}\sum_{n=1}^{N} X_n$ be the sample average of this sequence. If $\mathbb{E}[X_1] = I$ and $\mathbb{E}[X_1^2] = \sigma < \infty$, then the following result holds*

$$\frac{\sqrt{N}\,(I_N - I)}{\sigma} \to \mathcal{N}(0,1) \quad as \quad N \to \infty \tag{6.12}$$

where $\mathcal{N}(0,1)$ represents the unit normal distribution.

Proof. See, for example, [Durrett, 2010, Chapter 3]. □

The above assumes that our random variables are i.i.d.. In fact, neither the assumption of being identically distributed nor that of independence is actually necessary for the CLT to

hold. One can instead use the concept of ***strong mixing***, namely that variables sufficiently far apart in the sequence are independent, to generalize beyond the i.i.d. setting [Jones et al., 2004]. This is critical for the numerous Monte Carlo inference schemes, such as MCMC methods, that do not produce independent samples but instead rely on the samples converging in distribution to a target distribution.

6.4 Foundational Monte Carlo Inference Methods

In this section, we introduce the key methods that form the basis upon which most Monte Carlo inference schemes are based. The key idea at the heart of all Monte Carlo inference methods is to use some form of ***proposal distribution*** that we can easily sample from and then make appropriate adjustments to achieve (typically approximate) samples from the posterior. Most methods will require only an ***unnormalized*** distribution

$$\gamma(\theta) = \pi(\theta)Z \tag{6.13}$$

as a target where $Z = \int \gamma(\theta)d\theta$. As such they will apply to any situation where we desire to sample from an unnormalized (or in some cases normalized) distribution, for which the Bayesian inference setting is a particular case where

$$\gamma(\theta) = p(\theta, \mathcal{D}) = p(\mathcal{D}|\theta)p(\theta) \quad \text{and} \quad Z = p(\mathcal{D}). \tag{6.14}$$

Nonetheless, knowing Z can be useful in a number of scenarios, e.g. allowing for unbiased importance sampling estimators. The methods we introduce will, in general, vary only on how samples are proposed and the subsequent adjustments that are made. However, this will lead to a plethora of different approaches, varying substantially in their motivation, theoretical justification, algorithmic details, and the scenarios for which they are effective.

To aid linking this material to the rest of the course, we will presume a Bayesian inference setting from here on in. However, we note that the methods introduced all apply more generally for unnormalized targets $\gamma(\theta)$.

6.4.1 Rejection Sampling

Rejection sampling is one of the simplest Monte Carlo inference methods and one of the only ones to produce exact samples from the target. Before going into the method itself, we first consider an example to demonstrate the underlying intuition. Imagine we want to generate samples distributed uniformly over some arbitrary two-dimensional shape. One simple way of doing this would be to sample uniformly from a box enclosing the shape and then only taking the samples which fall

within the shape. An example of such sampling by rejection is shown in Figure 6.2. As all the samples within the box are distributed uniformly, they are also uniformly distributed on any subset of the space. Therefore if we sample from the box and then only take the samples that fall within the desired shape, we will generate samples uniformly over that shape. We can also use this method to estimate the area of the shape by using the fact that the probability of any one sample falling within the shape is equal to the ratio of the areas of the shape and the bounding box, namely

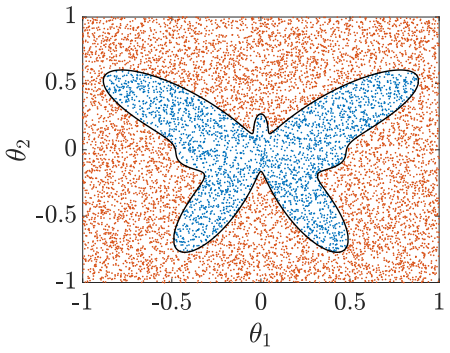


Figure 6.2: Sampling uniformly from an arbitrary shape by rejection. Samples are proposed uniformly from the $[-1, 1]$ square. Any sample falling within the black outline is accepted (blue), otherwise it is rejected (red).

$$
\begin{aligned}
A_{\text{shape}} &= A_{\text{box}} P(\theta \in \text{shape}) \\
&\approx \frac{A_{\text{box}}}{N} \sum_{n=1}^{N} \mathbb{I}(\hat{\theta}_n \in \text{shape}) \quad \text{where} \quad \hat{\theta}_n \sim \textsc{Uniform}(\text{box})
\end{aligned}
$$

and we have used a Monte Carlo estimator for $P(\theta \in \text{shape})$. Note that the value of $P(\theta \in \text{shape})$ will dictate the efficiency of our estimation as it represents the ***acceptance rate*** of our samples. In other words, we need to generate on average $1/P(\theta \in \text{shape})$ samples from our proposal for each sample created in the target area. As we will show later, $P(\theta \in \text{shape})$ typically becomes very small as θ becomes high-dimensional, so this approach will typically only be effective in low dimensions.

The underlying idea to extend this approach to rejection sampling more generally, is that we can sample from any distribution by sampling uniformly from the hyper–volume under its unnormalized probability density function. Though formally this is effectively axiomatic by the definition of a probability density function with respect to the Lebesgue measure, we can get a non measure-theoretic intuition for this by considering augmenting a target distribution with a new variable u such that $p(u|\theta, \mathcal{D}) = \textsc{Uniform}(0, p(\theta, \mathcal{D}))$. Sampling $\hat{\theta} \sim p(\theta|\mathcal{D})$ and then $\hat{u} \sim p(u|\theta, \mathcal{D})$ corresponds to sampling uniformly from the hyper-volume under the probability density function, while we clearly have that the marginal distribution on θ is $p(\theta|\mathcal{D})$.

Using this idea, we can sample from any unnormalized distribution by sampling from an appropriate bounding as per Figure 6.2 and then accepting only samples that fall within the hyper-volume of the probability density function. More specifically, we define a proposal distribution $q(\theta)$ which completely envelopes a scaled version of the unnormalized target distribution

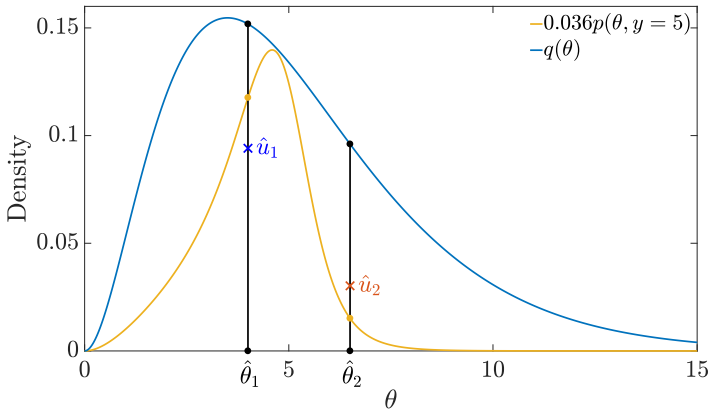


Figure 6.3: Demonstration of rejection sampling for problem shown in (6.1). We first sample $\hat{\theta} \sim q(\theta)$, corresponding to sampling from the distribution shown in blue, and then sample $\hat{u} \sim \text{UNIFORM}(0, q(\theta))$, corresponding to sampling a point uniformly along the black lines for the two shown example values of $\hat{\theta}$. The point is accepted if $\hat{u} \leq Cp(\theta, y = 5)$ (i.e. if it below the yellow curve), where we have taken $C = 0.036$ to ensure $Cp(\theta, y = 5) \leq q(\theta)$ for all theta. Here the example sample pair $\{\hat{\theta}_1, \hat{u}_1\}$ is accepted, while $\{\hat{\theta}_2, \hat{u}_2\}$ is rejected. The resulting accepted sample pairs will be uniformly sampled from the region under the unnormalized target distribution given by the yellow curve and therefore the accepted $\hat{\theta}$ will correspond to exact samples from the posterior $p(\theta|y = 5)$.

$Cp(\theta, \mathcal{D})$, for some fixed C, such that $q(\theta) \geq Cp(\theta, \mathcal{D})$ for all values of θ. We then sample a pair $\{\hat{\theta}, \hat{u}\}$ by first sampling $\hat{\theta} \sim q(\theta)$ and then $\hat{u} \sim \text{UNIFORM}(0, q(\theta))$. The sample is accepted if

$$\hat{u} \leq Cp(\hat{\theta}, \mathcal{D}) \tag{6.15}$$

which occurs with an acceptance rate $Cp(\mathcal{D})$ (note that $q(\theta) \geq Cp(\theta, \mathcal{D})\ \forall\theta$ ensures that $C \leq 1/p(\mathcal{D})$). This can be used to estimate the normalization constant $p(\mathcal{D})$, corresponding to the marginal likelihood for Bayesian models, by calculating the empirical estimate of the acceptance rate and dividing this by C. A graphical demonstration of the rejection sampling process is shown in Figure 6.3.

Rejection sampling can be a highly effective sampling or inference method in low dimensions. In particular, the fact that it generates exact samples from the target distribution can be very useful. This very rare characteristic is used to construct efficient samplers for many common distributions such as in the ziggurat algorithm for generating Gaussian random variables [Marsaglia et al., 2000]. More generally, whenever one wishes to construct a sampler for a non-standard low dimensional distribution, rejection sampling is the clear go-to approach because it produces exact samples and because one can usually engineer a very efficient sampler. However, the efficiency of rejection sampling is critically dependent on the value of C, because C is directly proportional to the acceptance rate. By proxy, it is also critically dependent on the proposal $q(\theta)$ as this dictates the minimum possible value of C, namely $C_{\text{min}} = \min_\theta q(\theta)p(\mathcal{D})/p(\theta|\mathcal{D})$.

Consequently, it is very prone to the ***curse of dimensionality*** as we discuss in the next chapter, meaning performance cannot be maintained for higher dimensional problems.

6.4.2 Importance Sampling

Importance sampling is another common sampling method that it the cornerstone for many more advanced inference schemes. It is closely related to rejection sampling in that it uses a proposal, i.e. $\hat{\theta} \sim q(\theta)$, but instead of going through an accept/reject step, it assigns an ***importance weight*** to each sample. These importance weights act like correction factors to account for the fact that we sampled from $q(\theta)$ rather than our target $p(\theta|\mathcal{D})$.

To demonstrate the key idea, consider the problem of calculating an expectation as per (6.8). We will assume for now that we can evaluate $p(\theta|\mathcal{D})$ exactly, but not draw samples from it, such that we cannot form a direct Monte Carlo estimate as per (6.9). Here, we can rearrange the form of our expectation to generate a different Monte Carlo estimator which we can evaluate directly as follows

$$\begin{aligned} I := \mathbb{E}_{p(\theta|\mathcal{D})}\left[f(\theta)\right] &= \int f(\theta)p(\theta|\mathcal{D})d\theta = \int f(\theta)\frac{p(\theta|\mathcal{D})}{q(\theta)}q(\theta)d\theta \\ &\approx \frac{1}{N}\sum_{n=1}^{N}\frac{p(\hat{\theta}_n|\mathcal{D})}{q(\hat{\theta}_n)}f(\hat{\theta}_n) \quad \text{where} \quad \hat{\theta}_n \sim q(\theta) \end{aligned} \tag{6.16}$$

where $\frac{p(\hat{\theta}_n|\mathcal{D})}{q(\hat{\theta}_n)} =: w_n$ is known as an importance weight. The key trick we have applied is to multiply the integrand by $\frac{q(\theta)}{q(\theta)}$, which equals 1 for all points where $q(\theta) \neq 0$. Thus if $q(\theta) \neq 0$ for all θ for which $p(\theta|\mathcal{D}) \neq 0$ (to avoid infinite importance weights), this has no effect on the expectation. However, we can informally view the new formulation as being the expectation of $f(\theta)\frac{p(\theta|\mathcal{D})}{q(\theta)}$ under the distribution $q(\theta)$. We can now construct a Monte Carlo estimator for this new formulation, by choosing $q(\theta)$ to be a distribution we sample from. A graphical demonstration of importance sampling is given in Figure 6.4 in the more general setting where we do not have access to the $p(\theta|\mathcal{D})$ exactly, but only an unnormalized version $p(\theta, \mathcal{D}) = p(\theta|\mathcal{D})p(\mathcal{D})$. As we will show in detail in Section 6.4.2.1, we can still use importance sampling in this case by ***self-normalizing*** the weights.

Importance sampling has a number of desirable properties as an inference method. In particular, it is both ***unbiased*** and ***consistent***. The former can be shown as follows

$$\begin{aligned} \mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N}\frac{p(\hat{\theta}_n|\mathcal{D})}{q(\hat{\theta}_n)}f(\hat{\theta}_n)\right] &= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{q(\theta_n)}\left[\frac{p(\hat{\theta}_n|\mathcal{D})}{q(\hat{\theta}_n)}f(\hat{\theta}_n)\right] \\ &= \mathbb{E}_{q(\theta_1)}\left[\frac{p(\hat{\theta}_1|\mathcal{D})}{q(\hat{\theta}_1)}f(\hat{\theta}_1)\right] = \mathbb{E}_{p(\theta|\mathcal{D})}\left[f(\theta)\right] \end{aligned} \tag{6.17}$$

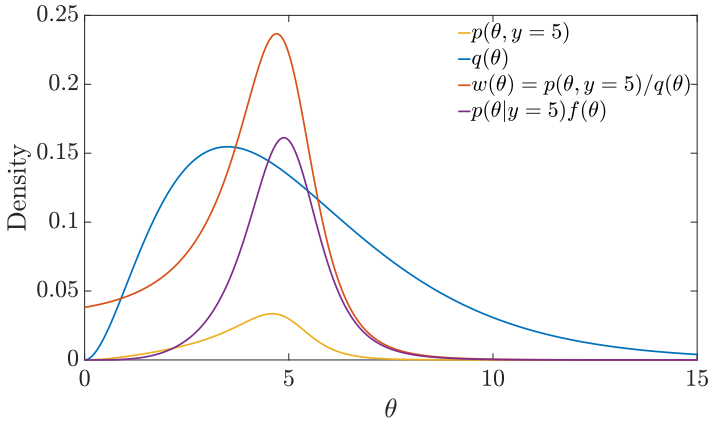


Figure 6.4: Demonstration of importance sampling for problem shown in (6.1). We are trying to estimate $\mathbb{E}_{p(\theta|\mathcal{D})}[f(\theta)]$: the expectation of the function $f(\theta) := \theta^2/50$ under the posterior $p(\theta|\mathcal{D}) := p(\theta|y = 5)$ defined as per (6.1c). We assume the setting where $p(\theta|\mathcal{D})$ is only known up to a normalization constant (see Section 6.4.2.1), namely we only have access to $p(\theta, \mathcal{D}) := p(\theta)p(y = 5|\theta)$ as shown in yellow. Our procedure is to draw samples independently $\hat{\theta}_n \sim q(\theta)$ and then evaluate their weight $w_n = w(\hat{\theta}_n) = p(\hat{\theta}_n, \mathcal{D})/q(\hat{\theta}_n)$. This produces a set of weighted samples which can then be used to estimate to estimate the expectation using (6.24) (one can also use (6.16) if the normalized $p(\theta|\mathcal{D})$ is used instead of $p(\theta, \mathcal{D})$).

where we have effectively stepped backward through (6.16).[3] Consistency, on the other hand, follows from applying the LLN in the same manner as (6.11), but replacing each $f(\hat{\theta}_n)$ with $\frac{p(\hat{\theta}_n|\mathcal{D})}{q(\hat{\theta}_n)} f(\hat{\theta}_n)$, leading to the same result except that σ_θ is now

$$\sigma_\theta^2 = \mathbb{E}_{q(\theta)}\left[\left(\frac{p(\hat{\theta}_1|\mathcal{D})}{q(\hat{\theta}_1)} f(\hat{\theta}_1) - I\right)^2\right] = \mathrm{Var}_{q(\theta)}\left[\frac{p(\theta|\mathcal{D})}{q(\theta)} f(\theta)\right]. \tag{6.18}$$

The form of this variance further gives insight into how to best to set the proposal: the lower that $\mathrm{Var}_{q(\theta)}\left[\frac{p(\theta|\mathcal{D})}{q(\theta)} f(\theta)\right]$ is, the better the expected performance of our estimator. One obvious question is what is the optimal proposal $q^*(\theta)$? It turns out that [Kahn and Marshall, 1953; Owen, 2013]

$$q^*(\theta) = \frac{p(\theta|\mathcal{D})\,|f(\theta)|}{\int p(\theta|\mathcal{D})\,|f(\theta)|\,d\theta}, \tag{6.19}$$

which can be shown as follows where we will make use of Jensen's inequality and comparing to an arbitrary proposal $q(\theta)$

$$\begin{aligned}
\mathrm{Var}_{q^*(\theta)}\left[\frac{p(\theta|\mathcal{D})}{q^*(\theta)} f(\theta)\right] &= \mathbb{E}_{q^*(\theta)}\left[\left(\frac{p(\theta|\mathcal{D})}{q^*(\theta)} f(\theta)\right)^2\right] - \left(\mathbb{E}_{q^*(\theta)}\left[\frac{p(\theta|\mathcal{D})}{q^*(\theta)} f(\theta)\right]\right)^2 \\
&= \int \frac{p(\theta|\mathcal{D})^2 f(\theta)^2}{q^*(\theta)} d\theta - I^2 = \left(\int p(\theta|\mathcal{D})\,|f(\theta)|\,d\theta\right)^2 - I^2 \qquad (6.20) \\
&\leq \int \left(\frac{p(\theta|\mathcal{D}) f(\theta)}{q(\theta)}\right)^2 q(\theta) d\theta - I^2 = \mathrm{Var}_{q(\theta)}\left[\frac{p(\theta|\mathcal{D})}{q(\theta)} f(\theta)\right]. \qquad (6.21)
\end{aligned}$$

[3] Note that this unbiasedness result does not pass over to the self-normalized variant given in Section 6.4.2.1.

Here we have shown that the variance for $q^*(\theta)$ is less than or equal to the variance using an arbitrary $q(\theta)$. It must, therefore, be the optimal proposal. A further point of note, is that if $f(\theta) \geq 0 \ \forall\theta$ (or $f(\theta) \leq 0 \ \forall\theta$), then (6.20) will equal zero giving a zero variance estimator: each importance weight will be equal to the $I/f(\theta)$ and thus I can be calculated by evaluating a single point.

Though it will typically be impossible to find $q^*(\theta)$ in practice, it still provides a guide as to what constitutes a good proposal: we want $p(\theta|\mathcal{D})\,|f(\theta)|\,/q(\theta)$ to be as close to constant as possible. In particular, we need to be careful to avoid scenarios where $\frac{p(\theta|\mathcal{D})|f(\theta)|}{\int p(\theta|\mathcal{D})|f(\theta)|d\theta} \gg q(\theta)$ as this will cause the ratio to explode, leading to high variances. A consequence of this is that we want $q(\theta)$ to have ***light tails*** compared to $p(\theta|\mathcal{D})\,|f(\theta)|$ to ensure that the ratio does not systematically increase as θ moves away from the modes of $q(\theta)$.

Aside from the clear practical issues, if this requirement does not hold, then it can easily be the case that $\sigma_\theta = \infty$ and thus that the estimator has infinite variance. Consider, for example, the case where $p(\theta|\mathcal{D}) = \mathcal{N}(\theta; 0, 1)$, $f(\theta) = \theta$, and $q(\theta) = \mathcal{N}(\theta; 0, s^2)$ (Example 9.1 from Owen [2013]). Noting that the mean, I, is zero by symmetry and defining $\nu = \frac{1}{2s^2} - 1$, we have that

$$\sigma_\theta^2 = \int_{-\infty}^{\infty} \theta^2 \frac{\left(\exp\left(-\theta^2/2\right)/\sqrt{2\pi}\right)^2}{\exp\left(-\theta^2/\left(2s^2\right)\right)/\sqrt{2\pi s^2}} d\theta - I^2 = \frac{s}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \theta^2 \exp\left(\theta^2\nu\right) d\theta.$$

Now this integral is clearly only finite for $\nu < 0$ (as otherwise the integrand is $+\infty$ and $\theta = \pm\infty$ and finite elsewhere). Therefore, σ_θ is only finite when $s^2 > 1/2$. In other words, we only get a finite estimator in this case if the proposal variance is at least half that of the target distribution $p(\theta|\mathcal{D})$. This highlights the pitfalls of having insufficiently heavy tails on our proposal. Overcoming these will typically require careful setup of the proposal on a case-by-case basis, for example, choosing a distribution type for the proposal that is known to have heavier tails than $p(\theta|\mathcal{D})$.

6.4.2.1 Self-Normalized Importance Sampling

In the previous section, we presumed that we have access to a normalized version of the posterior $p(\theta|\mathcal{D})$. Typically this will not be the case and we will only have access to an unnormalized target, namely the joint $p(\theta, \mathcal{D}) = p(\theta|\mathcal{D})p(\mathcal{D})$. We now show how one can still use importance sampling in these scenarios, by ***self-normalizing*** the importance weights.

The key idea for self-normalized importance sampling (SNIS) is that the weights provide an unbiased and consistent estimator of the marginal likelihood

$$Z_N := \frac{1}{N}\sum_{n=1}^{N} w_n, \tag{6.22}$$

$$\mathbb{E}[Z_N] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[w_n] = \mathbb{E}_{q(\hat{\theta}_1)}\left[\frac{p(\hat{\theta}_1,\mathcal{D})}{q(\hat{\theta}_1)}\right] = p(\mathcal{D}). \tag{6.23}$$

Now as $\mathbb{E}_{q(\theta)}\left[\frac{p(\theta,\mathcal{D})}{q(\theta)}f(\theta)\right] = E_{q(\theta)}\left[\frac{p(\theta|\mathcal{D})}{q(\theta)}p(\mathcal{D})f(\theta)\right] = p(\mathcal{D})\ \mathbb{E}_{p(\theta|\mathcal{D})}\left[f(\theta)\right]$, we can use our samples to construct Monte Carlo estimators for both $p(\mathcal{D})$ and $p(\mathcal{D})\ \mathbb{E}_{p(\theta|\mathcal{D})}\left[f(\theta)\right]$ and use the ratio of our estimates to get an estimate for $E_{p(\theta|\mathcal{D})}\left[f(\theta)\right]$ as follows

$$\mathbb{E}_{p(\theta|\mathcal{D})}\left[f(\theta)\right] \approx \frac{\frac{1}{N}\sum_{n=1}^{N} w_n f(\hat{\theta}_n)}{\frac{1}{N}\sum_{n=1}^{N} w_n} \quad \text{where} \quad \hat{\theta}_n \sim q(\theta), \quad w_n = \frac{p(\hat{\theta}_n,\mathcal{D})}{q(\hat{\theta}_n)}. \tag{6.24}$$

This can alternatively be expressed as $\mathbb{E}_{p(\theta|\mathcal{D})}\left[f(\theta)\right] \approx \sum_{n=1}^{N}\bar{w}_n f(\hat{\theta}_n)$ where $\bar{w}_n = \frac{w_n}{\sum_n w_n}$ are the normalized importance weights such that $\sum_{n=1}^{N}\bar{w}_n = 1$.

The consistency of (6.24) follows from the individual consistency of both the numerator and the denominator to $p(\mathcal{D})\ \mathbb{E}_{p(\theta|\mathcal{D})}\left[f(\theta)\right]$ and $p(\mathcal{D})$ respectively. However, unlike (6.16), (6.24) is a ***biased*** estimator for finite N. This is because a) the numerator and denominator are correlated and b) even though Z_N is an unbiased estimator of $p(\mathcal{D})$, $1/Z_N$ is not an unbiased estimator of $1/p(\mathcal{D})$. The latter follows directly from Jensen's inequality noting that inversion is a convex function for strictly positive inputs,

$$\mathbb{E}\left[\frac{1}{\frac{1}{N}\sum_{n=1}^{N} w_n}\right] \geq \frac{1}{\mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N} w_n\right]} = \frac{1}{p(\mathcal{D})}, \tag{6.25}$$

where equality holds if and only if Z_N is a zero variance estimator for $p(\mathcal{D})$ (which typically happens only in the limit $N \to \infty$). However, it can be shown that the bias decreases at a rate $O(1/N)$ (see e.g. Doucet and Johansen [2009]), whereas the standard deviation of the estimate decreases at a rate $O(1/\sqrt{N})$. Thus the bias becomes dominated as $N \to \infty$.

Note that the optimal proposal in the SNIS case varies slightly from the $q^*(\theta)$ derived earlier in the Section and is instead [Hesterberg, 1988]

$$q^*_{\text{SNIS}}(\theta) = \frac{p(\theta|\mathcal{D})\,|f(\theta) - I|}{\int p(\theta|\mathcal{D})\,|f(\theta) - I|\,d\theta}. \tag{6.26}$$

As a consequence, there is a minimum variance for the SNIS estimator, unlike in the pre-normalized case where $q^*(\theta)$ was a zero variance estimator if $f(\theta) \geq 0\ \forall\theta$.

6.4.2.2 Importance Sampling for Approximating the Posterior

So far we have assumed that we are using importance sampling to calculate an expectation of a known function. In practice, there will be many scenarios, where there is no explicit $f(\theta)$ or it is not known ahead of time, such that we instead just desire to generate samples from a posterior, potentially using these to calculate expectations at a later data.

When no $f(\theta)$ is specified, we can carry out importance sampling in the same fashion, sampling from $q(\theta)$ and returning a set of *weighted* samples $\{\hat{\theta}_n, w_n\}_{n=1:N}$ where the weights are equal to $p(\hat{\theta}_n, \mathcal{D})/q(\hat{\theta}_n)$ as before. Here we can think of importance sampling as approximating the posterior with a series of deltas functions, namely

$$p(\theta|\mathcal{D}) \approx \hat{p}(\theta|\mathcal{D}) := \sum_{n=1}^{N} \bar{w}_n \delta_{\hat{\theta}_n}(\theta) \tag{6.27}$$

where $\delta_{\hat{\theta}_n}(\theta)$ are delta functions centered at $\hat{\theta}_n$. This is known as an ***empirical measure*** and as $N \to \infty$ it becomes an "exact" approximation of the distribution in the sense that the Monte Carlo estimates formed using it converge to the target expectation.

Importance weights are multiplicative when doing conditional sampling: if we sample $\hat{\theta}_n \sim q_1(\theta)$ then $\hat{\phi}_n|\hat{\theta}_n \sim q_2(\phi|\hat{\theta}_n)$ when targeting $p(\theta|\mathcal{D})p(\phi|\theta, \mathcal{D})$ then the importance weight is

$$\frac{p(\hat{\theta}_n|\mathcal{D})p(\hat{\phi}_n|\hat{\theta}_n, \mathcal{D})}{q_1(\hat{\theta}_n)q_2(\hat{\phi}_n|\hat{\theta}_n)} = \frac{p(\hat{\theta}_n|\mathcal{D})}{q_1(\hat{\theta}_n)} \times \frac{p(\hat{\phi}_n|\hat{\theta}_n, D)}{q_2(\hat{\phi}_n|\hat{\theta}_n)} = w_{n,1} \times w_{n,2}. \tag{6.28}$$

This is known as ***sequential importance sampling*** and means that we can propagate importance weighted samples through a computational system and retain a valid importance sampler with the standard properties such as unbiasedness (presuming the weights are not self-normalized) and consistency. In other words, correctly weighted Monte Carlo samples share the key desirable theoretical properties of standard Monte Carlo samples.

Again a natural question in this "unknown f" setting is what is the optimal proposal $q^*(\theta)$? Though answering this question in a theoretically rigorous manner is beyond the scope of this course, for most purposes one can assume that the optimal proposal is simply the posterior, that is $q^*(\theta) = p(\theta|\mathcal{D})$

6.4.2.3 Effective Sample Size [Advanced Topic]

In this section, we consider an important diagnostic for the performance of importance sampling based schemes, the ***effective sample size*** (ESS). The ESS informally provides an estimated measure for the amount of information stored in our weighted sample set. The more information stored in the samples, the better our approximation of the posterior, and, at a high level, the more evenly balanced our weights, the more information they encode. Therefore, the ESS is a

measure of how many unweighted samples would be required to convey the same information about the posterior as the weighted sample set.

The weighted average of N_e independent evaluations $\{f_n\}_{n=1}^N$, where $f_n = f(\hat{\theta}_n)$, of some arbitrary function $f(\theta)$, each with individual variance σ^2, has variance σ^2/N_e as we showed in (6.11). Therefore, we can calculate the ESS of a set of weighted samples by comparing the variance of our weighted estimate to the variance of an estimate using a set of unweighted evaluations. More specifically, the ESS will be the number of unweighted evaluations N_e that gives an equivalent variance to our weighted estimate as follows, where we will make use of the assumption that the f_n are independent

$$\frac{\sigma^2}{N_e} = \text{Var}\left[\frac{\sum_{n=1}^N w_n f_n}{\sum_{n=1}^N w_n}\middle|\{w_n\}_{n=1}^N\right] = \sum_{n=1}^N \left(\frac{w_n}{\sum_{n=1}^N w_n}\right)^2 \text{Var}[f_n] = \frac{\sigma^2 \sum_{n=1}^N w_n^2}{\left(\sum_{n=1}^N w_n\right)^2}. \tag{6.29}$$

Now rearranging for N_e we get

$$N_e = \frac{\left(\sum_{n=1}^N w_n\right)^2}{\sum_{n=1}^N w_n^2} = \frac{1}{\sum_{n=1}^N \bar{w}_n^2} \tag{6.30}$$

which completes our definition for the effective sample size. It transpires that N_e is independent of $f(\theta)$, so we can still use the ESS as a diagnostic when f is unknown. It is straightforward to show using Jensen's inequality we have that $N_e \leq N$ with equality holding if and only if all the weights are equal. On the other hand, if all but one of the weights is zero, then $N_e = 1$. These two extremes respectively occur when the proposal is equal to the target, $q(\theta) = p(\theta|\mathcal{D})$, and when the proposal provides a very poor representation of the target. The ESS is often therefore used for ***proposal adaptation*** [Bugallo et al., 2017], as a larger value of the ESS generally indicates a better proposal.

However, the ESS is far from a perfect measure of sample quality. For example, if the proposal perfectly matches one of the modes of the target but completely misses another larger mode, the ESS will usually be very high, even though the samples provide a very poor representation of the target. It is not uncommon in practice to see the ESS drop drastically as more samples are added, due to the addition of a new dominating sample, typically indicating a region of significant target probability mass that had previously been missed. Nonetheless, the ESS is still a very useful performance metric and is usually a reliable indicator for whether our importance sampling is struggling. In particular, though the possibility of missing modes means that it is possible for the ESS to be high even when the approximation of the posterior is poor, if the ESS is low then the approximation of the posterior will always be poor and any subsequent estimates will usually be high variance.

6.4.2.4 Resampling [Advanced Topic]

A useful feature of SNIS is that it can be used to produce unweighted samples by ***sampling with replacement*** from the set of produced samples in proportion to the sample weights. This procedure is typically known as resampling, because we are resampling samples from the empirical distribution of our original samples. Resampling allows us to generate unweighted samples with importance sampling which are approximately distributed according to $p(\theta|\mathcal{D})$, with this approximation becoming exact in the limit $N \to \infty$. Resampling on its own always leads to a higher variance estimator than using (6.24) directly. However, there are many scenarios where unweighted samples are required or more convenient.

Mathematically, we can express resampling as producing a set of unweighted resampled samples $\left\{\tilde{\theta}_n\right\}_{n=1}^N$ using

$$\tilde{\theta}_n = \hat{\theta}_{a_n} \quad \text{where} \quad a_n \sim \textsc{Discrete}\left(\{\bar{w}_n\}_{n=1}^N\right). \tag{6.31}$$

Here $\{a_n\}_{n=1}^N$ are known as ancestor indices as they indicate which ancestor in the original sample set each unweighted sample originated from. Note that the a_n need not be drawn independently and typically are not; (6.31) instead conveys the required marginal distribution for each a_n.

Considering the approximation of the posterior provided by importance sampling given in (6.27), we can view resampling as producing the approximation

$$\tilde{p}(\theta|\mathcal{D}) := \sum_{n=1}^N \frac{k_n}{N}\delta_{\hat{\theta}_n}(\theta) \tag{6.32}$$

where k_n is the number times the sample $\hat{\theta}_n$ appears in the resampled sample set $\left\{\tilde{\theta}_n\right\}_{n=1}^N$. Provided that $\mathbb{E}[k_n|\{w_n\}_{n=1}^N] = Nk_n$, then it directly follows that $\tilde{p}(\theta|\mathcal{D})$ is an unbiased estimator for $p(\theta|\mathcal{D})$. Consequently, the convergence of SNIS with resampling follows directly from the convergence of SNIS.

There are a number of different methods one can use for generating the a_n in resampling [Douc and Cappé, 2005]. They all share in common the requirements above but vary in correlations between the a_n. The simplest method, ***multinomial resampling***, simply involves sampling each a_n independently, such that the k_n have a multinomial distribution. Though simple, this method is generally inadvisable as it adds unnecessary variation to the resampling compared to methods using randomized quasi-Monte Carlo [L'Ecuyer and Lemieux, 2005], such as systematic resampling [Carpenter et al., 1999; Whitley, 1994],[4] or other variance reduction techniques, such

[4]Note that systematic resampling, as it is now known, is somewhat confusingly referred to as stratified resampling in the former of these papers and universal sampling in the latter.

as stratified resampling [Kitagawa, 1996] and residual resampling [Whitley, 1994]. Though residual resampling is a little more complicated (see Douc and Cappé [2005]), stratified and systematic resampling only require small changes on the underlying random number draws made in multinomial resampling. See [Douc and Cappé, 2005] for more details.

6.5 Further Reading

- Chapters 1, 2, 7, and 9 of Art Owen's online book on Monte Carlo: `https://statweb.stanford.edu/~owen/mc/`
- Chapter 23 of K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012
- M. F. Bugallo, V. Elvira, L. Martino, D. Luengo, J. Miguez, and P. M. Djuric. Adaptive importance sampling: the past, the present, and the future. *IEEE Signal Processing Magazine*, 34(4):60–79, 2017
- David MacKay on Monte Carlo methods `http://videolectures.net/mackay_course_12/`

7 Advanced Inference Methods

In the last chapter we outlined the problem posed by Bayesian inference and outlined some of the fundamental building blocks and algorithms we can use to perform it. In this chapter, we show how the ***curse of dimensionality*** means that these simple approaches typically become ineffective in higher dimensions (they typically cannot be performed successfully above around 10 dimensions). We then show how the curse of dimensionality can be overcome (or at least mitigated) through two of the most common Bayesian inference approaches used in practice: Markov chain Monte Carlo (MCMC) and variational inference.

7.1 The Curse of Dimensionality

In this section, we digress from introducing specific inference methods themselves to talk about a common problem faced by most inference methods, the *curse of dimensionality* [Bellman, 1961]. At a high-level, the curse of dimensionality is a tendency of modeling and numerical procedures to get substantially harder as the dimensionality increases, often at an exponential rate. If not managed properly, it can cripple the performance of inference methods and it is the main reason the two procedures discussed so far, rejection sampling and importance sampling, are in practice only used for very low dimensional problems. At its core, it stems from an increase of the size (in an informal sense) of a problem as the dimensionality increases. This is easiest to see for discrete problems. Imagine we are calculating an expectation over a discrete distribution of dimension D, where each dimension has K possible values. The cost of enumerating all the possible combinations scales as K^D and thus increases exponentially with D; even for modest values for K and D this will be prohibitively large.

However, the curse of dimensionality extends far beyond problems of enumeration. It will be felt, to some degree or another, by almost all approaches for inference and modeling more generally, but its effect will be most pronounced for methods that try to explicitly model the target space. As a geometrical demonstration of this, we consider a rejection sampling example. For simplicity, we will presume that the target shape is a hypersphere and that the bounding shape is the smallest hypercube that encloses that hypersphere. Our acceptance rate, and thus the efficiency of the algorithm, will be equal to the ratio of the two hyper-volumes. For an even

number of dimensions, the hyper-volume of the D-dimensional hypersphere with radius r is $V_s = \frac{\pi^{D/2} r^D}{(D/2)!}$ and the hyper-volume of the enclosing hypercube is $V_c = (2r)^D$, giving a ratio of $\frac{V_s}{V_c} = \frac{\pi^{D/2}}{(D/2)!2^D} = \left(\frac{\sqrt{\pi}}{2}\right)^D \frac{1}{(D/2)!}$. The first of these terms decreases exponentially and second super-exponentially with D (noting that $(D/2)! > (D/6)^{(D/2)}$). For example, $D = 10, 20$, and 100 respectively gives ratios of approximately 2.5×10^{-3}, 2.5×10^{-8}, and 1.9×10^{-70}. Consequently, our acceptance rate will diminish super-exponentially with the number of dimensions and our approach will quickly become infeasible in higher dimensions.

An immediate possible criticism of this analysis would be to suggest that the approximation of the target shape provided by our bounding shape is simply increasingly poor as the dimensionality increases and that we should choose a better approximation. Although this is true, the key realization is that achieving a good approximation is increasingly difficult in high dimensions, typically exponentially so. To demonstrate this, imagine we instead use an arbitrary bounding shape defined in polar co-ordinates, such that the proportional difference in the radius at the any given point is at most ε (i.e. the radius of our bounding shape is between r and $r(1+\varepsilon)$ at all points). The hyper-volume in which our approximation might live for an even number of dimensions is given by

$$V_\varepsilon = \frac{\pi^{D/2} r^D (1+\varepsilon)^D}{(D/2)!} - \frac{\pi^{D/2} r^D}{(D/2)!} = V_s \left((1+\varepsilon)^D - 1\right). \tag{7.1}$$

Consequently, we have that the ratio $\frac{V_\varepsilon}{V_s}$ increases exponentially with D and that for sufficiently large D and a fixed ε the amount of space in our tolerance region will become substantially larger than the target hyper-volume, again leading to very low acceptance rates.

Flipping this on its head, we can ask the question how does ε need to vary to ensure that V_ε / V_s remains constant? A quick manipulation shows that $\varepsilon = \left(\left(\frac{V_\varepsilon}{V_s} + 1\right)^{1/D} - 1\right)$ and therefore that $\frac{\log\left(\frac{V_\varepsilon}{V_s}+1\right)}{D} \geq \log(1+\varepsilon) \approx \varepsilon$ for small values of ε. Thus we only need to decrease ε roughly in proportion to $\frac{1}{D}$ to achieve a fixed ratio. Initially, this would not seem so bad. For example, if $D = 1000$ we roughly need $\varepsilon \leq 6.9 \times 10^{-4}$ to get $V_\varepsilon / V_s \leq 1$. However, this misses the key difficulty caused by (7.1): the higher D is, the more difficult it is to accurately model the target shape and keep ε small. It follows from (7.1) that as D increases, the more the hyper-volume of the sphere is concentrated at the surface. This generalizes to non-spherical targets and means that accurate modeling the surface of our target is essential in high dimensions. Unfortunately, this task becomes rapidly more difficult with increasing dimensionality.

Consider, for example, regressing the surface of the target by using a number of inducing points spread over the surface. As the dimensionality increases, these become increasingly far

apart from one another and so the more points we need to accurately model the surface. For example, the probability of two points uniformly distributed on the surface of a sphere being within some target distance of one another decreases exponentially with the dimensionality of the sphere. This can be seen by noting that a necessary condition for two points to be within d of each other, is that the discrepancy of each individual dimension must be less than d. In other words, if we denote the overall distance as δ and the distance in each dimension as δ_i then we have

$$P(\delta \leq d) \leq P(\delta_1 \leq d)P(\delta_2 \leq d) \dots P(\delta_D \leq d)$$

and so $P(\delta \leq d)$ must decrease exponentially with D. If the correlation between points is proportional to their euclidean distance, then we will subsequently need an exponentially large number of points in the dimension to model the surface to a given accuracy. Consequently, we see that not only are we increasingly punished for any discrepancies between our approximation and the target as the dimensionality increases, it rapidly becomes harder to avoid these discrepancies in the first place.

One can informally think of the proposals we have introduced thus-far as being approximations to the target distribution: complications with tail behavior aside, it will generally be the case that the better the proposal approximates the target, the better the inference will perform. This typically leads to catastrophically bad performance for importance sampling and rejection sampling in high dimensions, for which this approximation breaks down for the reasons we have just outlined. To give a simple example, imagine that our target is an isotropic unit Gaussian and we use an independent student-t distribution with $\nu = 2$ in each dimension as the proposal. We have that the weights are as follows

$$w(\theta) = \frac{\pi(\theta)}{q(\theta)} = \prod_{i=1}^{D} \frac{\exp(-\theta_d^2/2)/\sqrt{2\pi}}{\frac{\Gamma(1.5)}{\sqrt{2\pi}}\left(1+\theta_d^2/2\right)^{-3/2}} = \prod_{i=1}^{D} \frac{2\exp(-\theta_d^2/2)\left(1+\theta_d^2/2\right)^{3/2}}{\sqrt{\pi}}. \tag{7.2}$$

It follows that the variance of the weights under the proposal increases exponentially with D as

$$\begin{aligned}\mathrm{Var}_{q(\theta)}\left[w(\theta)\right] &= \int w^2(\theta)q(\theta)d\theta - \left(\int w(\theta)q(\theta)d\theta\right)^2 = -1 + \prod_{i=1}^{D}\int_{-\infty}^{\infty} w^2(\theta_d)q(\theta_d)d\theta_d \\ &= -1 + \prod_{i=1}^{D}\int_{-\infty}^{\infty} \frac{\sqrt{2}\exp(-\theta_d^2)\left(1+\theta_d^2/2\right)^{(3/2)}}{\pi} d\theta_d \approx 1.1455^D - 1 \end{aligned} \tag{7.3}$$

where we have used the fact that the integral has a closed form solution. We thus see that our effective sample size will drop exponentially quickly with D and that our inference will break down if the dimensionality is too high.

It is now natural to ask whether we can overcome the curse of dimensionality. Thankfully, the answer in many scenarios is that we can. In many high-dimensional scenarios, the target

distribution will only have significant mass in a small proportion of the total area, often concentrated around a lower dimensional manifold of the larger space. This means that if we use inference methods that in some way exploit the structure of the target distribution and only search the small subset of the space with significant mass, then effective inference can still be performed. When this is not the case, practical inference will typically be futile in high dimensions anyway and so many inference algorithms are geared towards exploiting a particular type of structure. As we will show in the next section, the effectiveness of MCMC methods is mostly based on exploiting single modality in the target by making local moves that cause the algorithm to have a hill-climbing style behaviour away from the mode and then sticking close to the mode once it is found. Sequential Monte Carlo methods [Doucet and Johansen, 2009] rely on using the structure of the target more explicitly, by using a series of stepping-stone distributions and adaptively allocating resources. Variational methods often make assumptions or approximations about the structure of the model to break the inference problem down into a number of small problems that can then be combined into an overall estimate. Arguably the key to all advanced inference methods is how well they can exploit structure in higher dimensions, while the relative performance of different methods tends to come down to how suited the target is to their particular form of structure exploitation.

7.2 Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) methods [Metropolis et al., 1953; Hastings, 1970; Gilks et al., 1995] form one of the key approaches to circumventing the curse of dimensionality and are perhaps the most widely used class of algorithms for Bayesian inference, though they are also used extensively outside the Bayesian inference setting. The key idea is to construct a valid Markov chain that has the target distribution as its equilibrium distribution. They are suitable for Bayesian inference because this can still be done when the target distribution is only known up to a normalization constant.

The reason that they are often able to overcome, or at least alleviate, the curse of dimensionality, is that rather than trying to independently sample from the target distribution at each iteration, they instead make ***local*** moves from their current position. As with rejection sampling and importance sampling, they use a proposal distribution, but unlike these alternatives, the proposal is defined conditionally on the current location, namely, they propose according to $\theta' \sim q(\theta'|\theta)$ where θ is the current state and θ' is the new sampled state. The underlying intuition behind this is that in high dimensions the proportion of the space with significant probability

mass is typically very small. Therefore, if the target is single modal (or we have a proposal that is carefully designed to jump between modes), then once we have a sample in the mode, all the other points with significant mass should be close to that point. Therefore we can explore the mode by restricting ourselves to local moves, overcoming the curse of dimensionality by predominantly ignoring the majority of the space which has insignificant probability mass. As the dimensionality increases, the proportion of the space with significant mass decreases, counteracting many of the other complications that arise from the increasing dimension. When away from a mode, MCMC methods often behave like hill-climbing algorithms, emphasizing their close links with simulated annealing methods for optimization [Aarts and Korst, 1988]. Therefore, they can be highly effective for both finding the mode of a posterior and then sticking to that mode.

7.2.1 Markov Chains

We first introduced the concept of the ***Markov property*** in Section 4.4.2 in the concept of a hidden Markov model, where we explained how in a Markovian system each state is independent of all the previous states given the last state, i.e.

$$p(\Theta_n = \theta_n|\Theta_1 = \theta_1, \ldots, \Theta_{n-1} = \theta_{n-1}) = p(\Theta_n = \theta_n|\Theta_{n-1} = \theta_{n-1}). \tag{7.4}$$

In other words, the system transitions based only on its current state. Here the series $\Theta_1, \ldots, \Theta_n, \ldots$ is known as a Markov chain. We see that a probability of a Markov chain is fully defined by the probability of its initial state $p(\Theta_1 = \theta_1)$ and the probability of its transitions $p(\Theta_n = \theta_n|\Theta_{n-1} = \theta_{n-1})$. If each transition has the same distribution, i.e.

$$p(\Theta_{n+1} = \theta'|\Theta_n = \theta) = p(\Theta_n = \theta'|\Theta_{n-1} = \theta), \tag{7.5}$$

then the Markov chain is known as ***homogeneous***. Most MCMC methods are based on homogeneous Markov chains and so we will assume that (7.5) holds from now on. In such situations, $p(\Theta_{n+1} = \theta_{n+1}|\Theta_n = \theta_n)$ is typically known as a ***transition kernel*** $T(\theta_{n+1} \leftarrow \theta_n)$.

For a Markov chain to converge to a target distribution $p(\theta|\mathcal{D})$, we will need that $\lim_{n\to\infty} p(\Theta_n = \theta) = p(\theta|\mathcal{D})$ for any possible starting position θ_1, i.e. that the chain converges in distribution to the target for all possible starting positions. For this to happen we need two things: $p(\theta|\mathcal{D})$ must be a ***stationary distribution*** of the Markov chain, such that if $p(\Theta_n = \theta) = p(\theta|\mathcal{D})$ then $p(\Theta_{n+1} = \theta) = p(\theta|\mathcal{D})$, and all possible starting points θ_1 must converge to this distribution. The former of these will be satisfied if

$$\int T(\theta' \leftarrow \theta)p(\theta|\mathcal{D})d\theta = p(\theta'|\mathcal{D}) \tag{7.6}$$

where we see that the target distribution is ***invariant*** under the application of the transition kernel. Thus if $p(\theta_n) = p(\theta|\mathcal{D})$ for some n, all subsequent points will have the desired distribution. The requirement that all starting points converge to the desired target distribution is known as ***ergodicity***, which guarantees both the uniqueness of the stationary distribution and that all points converge to this distribution. Ergodicity requires that the Markov chain is ***irreducible***, i.e. all points with non-zero probability can be reached in a finite number of steps, and ***aperiodic***, i.e. that no states can only be reached at certain periods of time. We will not delve into the specifics of ergodicity in depth, but note only that homogeneous Markov chains that satisfy (7.6) can be shown to be ergodic under very weak conditions, see for example Neal [1993]; Tierney [1994].

7.2.2 Detailed Balance

A common sufficient (but not necessary) condition used for constructing valid Markov chains is to ensure that the chain satisfies the condition of ***detailed balance***. Chains that satisfy detailed balance are known as ***reversible***. For a target $p(\theta|\mathcal{D})$, detailed balanced is defined as

$$p(\theta|\mathcal{D})T(\theta' \leftarrow \theta) = p(\theta'|\mathcal{D})T(\theta \leftarrow \theta'). \tag{7.7}$$

It is straightforward to see that Markov chains satisfying detailed balance will admit $p(\theta|\mathcal{D})$ as a stationary distribution by noting that

$$\int T(\theta' \leftarrow \theta)p(\theta|\mathcal{D})d\theta = \int T(\theta \leftarrow \theta')p(\theta'|\mathcal{D})d\theta = p(\theta'|\mathcal{D}). \tag{7.8}$$

Thus any ergodic Markov chain we construct that satisfies (7.7) will converge to the target distribution. From an inference perspective, this means that we can eventually generate samples according to our desired target by choosing an arbitrary start point Θ_1 and then repeatedly sampling from our transition kernel $T(\Theta_n \leftarrow \Theta_{n-1})$.

7.2.3 Metropolis Hastings

One of the simplest and most widely used MCMC methods is ***Metropolis Hastings*** (MH) [Hastings, 1970]. Given an unnormalized target $p(\theta, \mathcal{D})$, then at each iteration of the MH algorithm, one samples a new point θ' according to the a proposal $\theta' \sim q(\theta'|\theta_n)$ conditioned on the current point θ_n and then accepts the new sample with probability

$$P(\text{Accept}) = \min\left(1, \frac{p(\theta', \mathcal{D})q(\theta_n|\theta')}{p(\theta_n, \mathcal{D})q(\theta'|\theta_n)}\right). \tag{7.9}$$

At iteration n then we set $\theta_{n+1} \leftarrow \theta'$ if the sample is accepted and otherwise set $\theta_{n+1} \leftarrow \theta_n$. Critically this process does not require access to the normalized posterior $p(\theta|\mathcal{D})$. We can show

that (7.9) satisfies detailed balance and therefore produces a valid Markov chain as follows

$$\begin{aligned} p(\theta_n|\mathcal{D})T(\theta_{n+1} \leftarrow \theta_n) &= \min\left(1, \frac{p(\theta_{n+1},\mathcal{D})q(\theta_n|\theta_{n+1})}{p(\theta_n,\mathcal{D})q(\theta_{n+1}|\theta_n)}\right) p(\theta_n|\mathcal{D})q(\theta_{n+1}|\theta_n) \\ &= \min\left(p(\theta_n,\mathcal{D})q(\theta_{n+1}|\theta_n), p(\theta_{n+1},\mathcal{D})q(\theta_n|\theta_{n+1})\right)/p(\mathcal{D}) \\ &= \min\left(\frac{p(\theta_n,\mathcal{D})q(\theta_{n+1}|\theta_n)}{p(\theta_{n+1},\mathcal{D})q(\theta_n|\theta_{n+1})}, 1\right) p(\theta_{n+1}|\mathcal{D})q(\theta_n|\theta_{n+1}) \\ &= p(\theta_{n+1}|\mathcal{D})T(\theta_n \leftarrow \theta_{n+1}). \end{aligned}$$

Though MH is valid for any reasonable choice of the proposal distribution [Tierney, 1994], the practical performance will depend heavily on this choice. For example, if $q(\theta'|\theta)$ is independent of θ then no information is passed from one iteration to the next and one gets an algorithm (known as the Metropolis algorithm, i.e. the Hasting term in MH relates to the use of local moves) that is strictly worse than importance sampling: samples are independently generated in the same way, but information is lost in the accept-reject step. Instead, one will generally want to propose points close to the current point so the advantages of *local moves* can be exploited, namely through the hill climbing behavior we previously discussed. However, this has complications as explained in Section 7.2.5, while choosing a proposal with the right characteristics is still rather challenging. For example, if the variance of our proposal is too high then we will rarely propose good points and so the acceptance rate will become very low, giving few distinct samples. If the variance is too low, the Markov chain will move very slowly as it can only take small steps. This will increase correlation between all our samples and reduce the fidelity of our estimates. Chains that quickly cover the full probability space are said to ***mix*** quickly.

7.2.4 Gibbs Sampling [Advanced Topic]

Gibbs sampling is an important special case of Metropolis-Hastings that looks to update only some subset of variables in a joint distribution at each iteration. Imagine we have a D distributional posterior $p(\theta|\mathcal{D})$ where $\theta = \{\theta_1, \theta_2, \ldots, \theta_D\}$. Gibbs sampling incrementally updates one or more of the variables θ_d at each iteration conditioned on the value of the others. Thus it uses proposals of the form $\theta'_d \sim p(\theta'_d|\theta\backslash\theta_d, \mathcal{D})$ with $\theta\backslash\theta_d$ kept constant from one iteration to the next. There are two reasons for wanting to do this. Firstly changing only one of the variables at a time is a form of local proposal and can be a beneficial way to make updates, particularly if a random walk proposal is inappropriate. Secondly, if we have access to $p(\theta_d|\theta\backslash\theta_d, \mathcal{D})$ exactly, then we

will actually accept every sample as because, noting that $\theta' \backslash \theta'_d = \theta \backslash \theta_d$, we have

$$\begin{aligned}\frac{p(\theta|\mathcal{D})p(\theta'_d|\theta\backslash\theta_d,\mathcal{D})}{p(\theta'|\mathcal{D})p(\theta_d|\theta'\backslash\theta'_d,\mathcal{D})} &= \frac{p(\theta_d|\theta\backslash\theta_d,\mathcal{D})p(\theta\backslash\theta_d|\mathcal{D})p(\theta'_d|\theta\backslash\theta_d,\mathcal{D})}{p(\theta'_d|\theta'\backslash\theta'_d,\mathcal{D})p(\theta'\backslash\theta'_d|\mathcal{D})p(\theta_d|\theta'\backslash\theta'_d,\mathcal{D})} \\ &= \frac{p(\theta_d|\theta\backslash\theta_d,\mathcal{D})p(\theta\backslash\theta_d|\mathcal{D})p(\theta'_d|\theta\backslash\theta_d,\mathcal{D})}{p(\theta'_d|\theta\backslash\theta_d,\mathcal{D})p(\theta\backslash\theta_d|\mathcal{D})p(\theta_d|\theta\backslash\theta_d,\mathcal{D})} = 1\end{aligned}$$

such that the acceptance probability is always 1. In a number of common models, it will be possible to sample from $\theta'_d \sim p(\theta'_d|\theta\backslash\theta_d,\mathcal{D})$ exactly and thus carry out Gibbs sampling steps exactly. We can then cycle through each of the θ_d, either in a random order or in sequence, and apply the appropriate updates.

The effectiveness of this approach will depend on the level of correlation between the different variables. The more correlated each variable, the smaller the updates will be for each variable conditioned on the values of the others and the slower the chain will mix. In extreme cases, this does impose stricter conditions for convergence for Gibbs samplers than MH [Roberts and Smith, 1994]. For example, consider an exclusive OR style problem where $p(\theta_1,\theta_2|\mathcal{D}) = 1$ if $0 \leq \theta_1 \leq 1$ and $0 \leq \theta_2 \leq 1$ or $-1 \leq \theta_1 < 0$ and $-1 \leq \theta_2 < 0$, and $p(\theta_1,\theta_2|\mathcal{D}) = 0$ otherwise. Here there is no way to move from the $[0,1]^2$ square to the $[-1,0]^2$ square by updating only one of the variables at a time. As such, a Gibbs sampler would end up stuck in either the positive or negative square.

In the more general case—where it is not possible to sample from the conditional distributions exactly, or if it is only possible for some of the variables—we can instead use a ***Metropolis-within-Gibbs*** approach, also known as ***component-wise Metropolis-Hastings***, where we approximate one or more $p(\theta'_d|\theta\backslash\theta_d,\mathcal{D})$ with an appropriate MH transition kernel. Namely, we introduce a proposal for that specific variable, sample from it, and then perform an accept-reject step. Though the convergence of this approach has been shown by, for example, Jones et al. [2014], additional assumptions are required compared to the standard Gibbs or MH cases (these are beyond the scope of this course).

7.2.5 Intuitions, Complications, and Practical Considerations

Though MCMC methods can be exceptionally effective, they are not without their weaknesses. Most of these weakness stem from the fact that all the generated samples are correlated, leading to, for example, biased estimates. Correlation reduces the amount of distinct information conveyed by each sample and this will reduce the accuracy of the estimator. However, it also causes more fundamental issues.

Most of the convergence results we have presented so far have relied on samples being generated in a i.i.d. fashion, which is clearly not the case in the MCMC setting. MCMC

methods therefore require their own unique convergence proofs, based in general on ergodic theory (see e.g. [Durrett, 2010, Chapter 6]). Furthermore, whereas importance sampling and rejection sampling lead to unbiased estimates of the marginal likelihood, MCMC produces no natural estimate and methods that do produce marginal likelihood estimates for MCMC are often extremely biased [Chib and Jeliazkov, 2001].

The aforementioned convergence results mean that the bias of estimates made using MCMC samples tends to zero as the number of iterations tends to infinity, but it is often very difficult to estimate the magnitude of the bias for a finite numbers of iterations. Whereas importance sampling and rejection sampling had reasonable diagnostics for the performance of the inference, such as the effective sample size and the acceptance rate respectively, estimating the bias from MCMC samplers is typically fiendishly difficult and it can often look like an MCMC sampler is performing well (e.g. in terms of its acceptance rate) when in fact it is doing disastrously.

One of the most common ways this is manifested is in the sampler becoming stuck in a particular mode of the target. Using localized proposals can make it prohibitively difficult to move between modes. Though valid MCMC samplers must eventually visit every mode infinitely often, it can take arbitrarily long to even visit each mode once. Even worse, getting the correct estimate relies on spending the correct relative proportion of time in each mode, which will typically take many orders of magnitude more time to get a reasonable estimate for, than it will just to have the sampler visit each significant mode at least once. The issues associated with multiple modes provides a demonstration of why it is difficult to estimate the bias of an MCMC sampler: we do not generally know if we have missed another mode or whether our sampler has spent an appropriate amount of time in each mode.

Because of these drawbacks, using MCMC on multi-modal problems is dangerous unless an appropriate mechanism for transitioning between the modes can be found. One also tends to throw away some of the earlier samples in the Markov chain to allow the chain to ***burn in***, remembering that samples are only distributed according to the target of interest asymptotically and so the earlier samples, which have marginal distributions very far away from the distribution of interest, can add substantial bias to the resultant estimator. Thankfully, there are a surprisingly wide array of models that actually fit these restrictions, particularly in high dimensions or if we can find an appropriate parameterization of the model.

Remembering from Section 2.7 that changing the parameterization of a model changes its probability density function in a non-trivial manner, the performance of MCMC methods is often critically dependent on their parameterization. Changing the parameterization will change the

concept of what parameter values are close to which other parameter values. In an ideal world, we would make moves in the raw sample space where all points are equally probable. Typically this is not practical, but it is still usually the case that some parameterizations will tend to be more single-modal and more generally have all points of interest close together in the parameter space. Note that there is often an equivalence here between a good proposal and a good warping of the space to one where an isotropic proposal will be effective. One possible way of achieving a good parameterization is through the use of ***auxiliary variables*** [Higdon, 1998; Andrieu et al., 2010], which can improve mixing by allowing more degrees of freedom in the proposal, decreasing the chance of getting "stuck", e.g. in a particular mode. Somewhat counterintuitively, projecting to higher dimensional spaces can actually substantially improve the mixing of an MCMC sampler.

As a concrete example of this, ***Hamiltonian Monte Carlo*** (HMC) [Duane et al., 1987; Neal, 2011] uses derivatives of the density function and an auxiliary variable to make effective long distance proposals. Given that much of the behavior of MCMC is based on hill-climbing effects, it would perhaps be intuitive to presume that these gradients are used to hasten the hill-climbing behavior or try to move between modes. In practice, the intention is exactly the opposite. In high dimensions, most of the mass of a mode is not at its peak but in a thin strip around that peak known as a ***typical set*** [Betancourt, 2017]. Classical random walk MH methods will both rarely propose samples in the right direction to stay in this typical set, giving a low acceptance rate, and be very slow to move around the typical set because the reversibility of the proposals mean that this only happens slowly through drift. By moving perpendicular to the gradient, HMC makes proposals that are more likely to stay within the typical set, while also allowing large moves to be made in a single step. Together these mean that it can explore a particular mode much faster than random walk MH strategies. See, for example, Neal [2011]; Betancourt [2017] for a more complete introduction.

7.3 Variational Inference

Though our focus in these notes has mostly been on Monte Carlo inference methods, we finish by noting that these are far from the only viable approaches. Two key advantages of Monte Carlo methods are their ubiquitous nature, i.e. many can almost always be applied, and that most commonly used Monte Carlo methods are asymptotically exact, such that given enough time, we can always achieve a required level of accuracy. However, in some scenarios, Monte Carlo methods can be problematically slow to converge and so alternative, asymptotically approximate,

methods can be preferable such as ***variational inference*** [Blei et al., 2016] and ***messaging passing*** methods [Lauritzen and Spiegelhalter, 1988].

Of these, variational inference has become an increasingly popular approach. Its key idea is to reformulate the inference problem to an optimization, by learning parameters of an approximation to the posterior. Typically this involves defining some family of distributions within which the posterior approximation can live, e.g. an exponential distribution family, and then optimizing an ***evidence lower bound*** (ELBO) with respect to the parameters of this approximation. Doing this implicitly minimizes the Kullback-Leiber divergence between the approximation the target. Variational inference often forms a highly efficient means of calculating a posterior approximation, but, in addition to the obvious bias from using a particular family of distributions for the approximation, it typically requires strong structural assumptions to be made about the form of the posterior. Namely most methods make a so-called ***mean-field*** assumption that presumes that the posterior factorizes over all latent variables. Its effectiveness is thus critically dependent on the reasonableness of these assumptions.

We will not cover variational inference further, or our subsequent topic of variational auto-encoders [Kingma and Welling, 2014; Rezende et al., 2014], in these notes. You are instead referred to the lecture slides and suggested further reading. Note that the slides now available online are more comprehensive than those used in the lectures themselves.

7.4 Further Reading

- Iain Murray on MCMC: `https://www.youtube.com/watch?v=_v4Eb09qp7Q`
- Demo of various MCMC methods: `https://chi-feng.github.io/mcmc-demo/app.html?algorithm=RandomWalkMH&target=banana`
- Chapters 21, 22, and 23 of K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012
- D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *arXiv preprint arXiv:1601.00670*, 2016
- NeurIPS tutorial on variational inference that accompanies the previous paper: `https://www.youtube.com/watch?v=ogdv_6dbvVQ`
- Training VAEs in Pyro: `https://pyro.ai/examples/vae.html` and `https://www.youtube.com/watch?v=vgFWeEyen6Y&t=1058s`

- Tutorial paper on VAEs: C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016
- Video tutorial on deep generative models by Shakir Mohamed and Danilo Rezende `https://www.youtube.com/watch?v=JrO5fSskISY`
- GANs, one of the main alternatives to VAEs: I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014

Bibliography

E. Aarts and J. Korst. Simulated annealing and Boltzmann machines. 1988.

M. Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2010.

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, pages 337–404, 1950.

R. Baillargeon. The acquisition of physical knowledge in infancy: A summary in eight lessons. *Blackwell handbook of childhood cognitive development*, 1(46-83):1, 2002.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *arXiv preprint arXiv:1502.05767*, 2015.

R. E. Bellman. *Adaptive control processes: a guided tour*. Princeton university press, 1961.

J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: A CPU and GPU math compiler in Python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

M. Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*, 2017.

Bingham, J. P. Eli, Chen, M. Jankowiak, T. Karaletsos, F. Obermeyer, N. Pradhan, R. Singh, P. Szerlip, and N. Goodman. Pyro, 2017. URL `https://github.com/uber/pyro`.

C. M. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *arXiv preprint arXiv:1601.00670*, 2016.

B. Bloem-Reddy, E. Mathieu, A. Foster, T. Rainforth, H. Ge, M. Lomelí, Z. Ghahramani, and Y. W. Teh. Sampling and inference for discrete random probability measures in probabilistic programs. *NIPS Workshop on Advances in Approximate Bayesian Inference*, 2017.

G. E. Box. Robustness in the strategy of scientific model building. *Robustness in statistics*, 1: 201–236, 1979.

G. E. Box, W. G. Hunter, and J. S. Hunter. *Statistics for experimenters: an introduction to design, data analysis, and model building*, volume 1. John Wiley and Sons, 1979.

L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

L. Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

M. F. Bugallo, V. Elvira, L. Martino, D. Luengo, J. Miguez, and P. M. Djuric. Adaptive importance sampling: the past, the present, and the future. *IEEE Signal Processing Magazine*, 34(4):60–79, 2017.

R. Burbidge, M. Trotter, B. Buxton, and S. Holden. Drug design by machine learning: support vector machines for pharmaceutical data analysis. *Computers & chemistry*, 26(1):5–14, 2001.

B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. A. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: a probabilistic programming language. *Journal of Statistical Software*, 2015.

J. Carpenter, P. Clifford, and P. Fearnhead. Improved particle filter for nonlinear problems. *IEE Proceedings-Radar, Sonar and Navigation*, 146(1):2–7, 1999.

S. Chib and I. Jeliazkov. Marginal likelihood from the Metropolis–Hastings output. *Journal of the American Statistical Association*, 96(453):270–281, 2001.

G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial intelligence*, 42(2-3):393–405, 1990.

C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial intelligence*, 60(1):141–153, 1993.

B. De Finetti. La prévision: ses lois logiques, ses sources subjectives. In *Annales de l'institut Henri Poincaré*, volume 7, pages 1–68, 1937.

N. Dhir, Y. Perov, M. Wijers, F. Wood, A. Markham, P. Trethowan, B. du Preez, A. Loveridge, and D. Macdonald. Tracking african lions with nonparametric hierarchical models using probabilistic programming. In *Proceedings of the International Society of Bayesian Analysis (ISBA) 2016 World Meeting*, 2016.

N. Dhir, M. Vákár, M. Wijers, A. Markham, F. Wood, P. Trethowan, B. du Preez, A. Loveridge, and D. Macdonald. Interpreting lion behaviour with nonparametric probabilistic programs. In *UAI*, 2017.

C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

J. L. Doob. Application of the theory of Martingales. *Le calcul des probabilites et ses applications*, pages 23–27, 1949.

R. Douc and O. Cappé. Comparison of resampling schemes for particle filtering. In *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, pages 64–69. IEEE, 2005.

A. Doucet and A. M. Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12:656–704, 2009.

A. Doucet, N. De Freitas, and N. Gordon. An introduction to sequential Monte Carlo methods. In *Sequential Monte Carlo methods in practice*, pages 3–14. Springer, 2001.

S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics letters B*, 1987.

R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

R. Durrett. *Probability: theory and examples*. Cambridge university press, 2010.

D. Duvenaud, J. R. Lloyd, R. Grosse, J. B. Tenenbaum, and Z. Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. *arXiv preprint arXiv:1302.4922*, 2013.

G. Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. *Journal of Geophysical Research: Oceans*, 99, 1994.

T. S. Ferguson. A Bayesian analysis of some nonparametric problems. *The annals of statistics*, pages 209–230, 1973.

D. A. Freedman. On the asymptotic behavior of Bayes' estimates in the discrete case. *The Annals of Mathematical Statistics*, pages 1386–1403, 1963.

A. Gelman and C. P. Robert. "Not only defended but also applied": The perceived absurdity of Bayesian inference. *The American Statistician*, 67(1):1–5, 2013.

A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian data analysis*, volume 2. CRC press Boca Raton, FL, 2014.

A. Gelman et al. Objections to Bayesian statistics. *Bayesian Analysis*, 2008.

A. Gelman et al. Induction and deduction in Bayesian data analysis. *Rationality, Markets and Morals*, 2(67-78):1999, 2011.

S. J. Gershman and D. M. Blei. A tutorial on Bayesian nonparametric models. *Journal of Mathematical Psychology*, 56(1):1–12, 2012.

Z. Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 2015.

W. R. Gilks, S. Richardson, and D. Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC press, 1995.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

N. Goodman, V. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. In *UAI*, pages 220–229, 2008a.

N. D. Goodman. The principles and practice of probabilistic programming. *ACM SIGPLAN Notices*, 48(1):399–402, 2013.

N. D. Goodman and A. Stuhlmüller. *The Design and Implementation of Probabilistic Programming Languages*. 2014.

N. D. Goodman, V. K. Mansinghka, D. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. 2008b.

S. N. Goodman. Toward evidence-based medical statistics. 1: The p value fallacy. *Annals of internal medicine*, 130(12):995–1004, 1999.

A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*. ACM, 2014.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2001.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

T. C. Hesterberg. *Advances in importance sampling*. PhD thesis, Stanford University, 1988.

C. Heunen, O. Kammar, S. Staton, and H. Yang. A convenient category for higher-order probability theory. *arXiv preprint arXiv:1701.02547*, 2017.

R. Hickey. The clojure programming language. In *Proceedings of the 2008 symposium on Dynamic languages*, page 1. ACM, 2008.

D. M. Higdon. Auxiliary variable methods for Markov chain Monte Carlo with applications. *Journal of the American Statistical Association*, 93(442):585–595, 1998.

M. D. Hoffman and A. Gelman. The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *JMLR*, 15(1):1593–1623, 2014.

T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

J. P. Ioannidis. Why most published research findings are false. *PLoS medicine*, 2(8):e124, 2005.

P. Jäckel. *Monte Carlo methods in finance*. J. Wiley, 2002.

D. Janz, B. Paige, T. Rainforth, J.-W. van de Meent, and F. Wood. Probabilistic structure discovery in time series data. *NIPS Workshop on Artificial Intelligence for Data Science*, 2016.

G. L. Jones, G. O. Roberts, and J. S. Rosenthal. Convergence of conditional Metropolis-Hastings samplers. *Advances in Applied Probability*, 46(2):422–445, 2014.

G. L. Jones et al. On the Markov chain central limit theorem. *Probability surveys*, 2004.

M. I. Jordan. Are you a Bayesian or a frequentist? 2009. URL `http://videolectures.net/mlss09uk_jordan_bfway/`.

D. Jurafsky and J. H. Martin. *Speech and language processing*, volume 3. Pearson London, 2014.

H. Kahn and A. W. Marshall. Methods of reducing sample size in Monte Carlo computations. *Journal of the Operations Research Society of America*, 1(5):263–278, 1953.

D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

G. Kitagawa. Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of computational and graphical statistics*, 5(1):1–25, 1996.

B. Kleijn, A. Van der Vaart, et al. The Bernstein-von-Mises theorem under misspecification. *Electronic Journal of Statistics*, 6:354–381, 2012.

A. Kucukelbir, R. Ranganath, A. Gelman, and D. Blei. Automatic variational inference in Stan. In *NIPS*, pages 568–576, 2015.

M. Kuss and C. E. Rasmussen. Assessing approximate inference for binary Gaussian process classification. *JMLR*, 6(Oct):1679–1704, 2005.

D. P. Landau and K. Binder. *A guide to Monte Carlo simulations in statistical physics*. Cambridge university press, 2014.

S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 157–224, 1988.

N. D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *NIPS*, pages 329–336, 2004.

T. A. Le, A. G. Baydin, and F. Wood. Inference compilation and universal probabilistic programming. In *20th AISTATS*, 2017a.

T. A. Le, A. G. Baydin, R. Zinkov, and F. Wood. Using synthetic data to train neural networks is model-based reasoning. *arXiv preprint arXiv:1703.00868*, 2017b.

S. Lefèvre, D. Vasquez, and C. Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *Robomech Journal*, 1(1):1, 2014.

J. S. Liu and R. Chen. Sequential Monte Carlo methods for dynamic systems. *Journal of the American statistical association*, 93(443):1032–1044, 1998.

J. R. Lloyd, D. K. Duvenaud, R. B. Grosse, J. B. Tenenbaum, and Z. Ghahramani. Automatic construction and natural-language description of nonparametric regression models. In *AAAI*, pages 1242–1250, 2014.

D. J. Lunn, A. Thomas, N. Best, and D. Spiegelhalter. Winbugs-a Bayesian modelling framework: concepts, structure, and extensibility. *Statistics and computing*, 10(4):325–337, 2000.

P. L'Ecuyer and C. Lemieux. Recent advances in randomized quasi-Monte Carlo methods. *Modeling uncertainty*, pages 419–474, 2005.

V. Mansinghka, D. Selsam, and Y. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint arXiv:1404.0099*, 2014.

V. K. Mansinghka, T. D. Kulkarni, Y. N. Perov, and J. Tenenbaum. Approximate Bayesian image interpretation using generative probabilistic graphics programs. In *NIPS*, pages 1520–1528, 2013.

G. Marsaglia, W. W. Tsang, et al. The ziggurat method for generating random variables. *Journal of statistical software*, 5(8):1–7, 2000.

D. McAllester. A pac-Bayesian tutorial with a dropout bound. *arXiv preprint arXiv:1307.2118*, 2013.

N. Metropolis and S. Ulam. The Monte Carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

T. Minka, J. Winn, J. Guiver, and D. Knowles. Infer .NET 2.4, Microsoft Research Cambridge, 2010.

T. P. Minka. Bayesian model averaging is not model combination. *Available electronically at http://www. stat. cmu. edu/minka/papers/bma. html*, 2000.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

L. M. Murray. Bayesian state-space modelling on high-performance hardware using libbi. *arXiv preprint arXiv:1306.3277*, 2013.

R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. 1993.

R. M. Neal. MCMC using Hamiltonian dynamics. *Handbook of MCMC*, 2, 2011.

A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, pages 841–848, 2002.

A. B. Owen. *Monte Carlo theory, methods and examples*. 2013.

B. Paige. *Automatic inference for higher-order probabilistic programs*. PhD thesis, PhD thesis, University of Oxford, 2016.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. *NIPS Workshop on AutoDiff*, 2017.

J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. 2014.

G. Perantoni and D. J. Limebeer. Optimal control for a formula one car with variable parameters. *Vehicle System Dynamics*, 52(5):653–678, 2014.

K. B. Petersen, M. S. Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7: 15, 2008.

M. Plummer et al. Jags: A program for analysis of Bayesian graphical models using Gibbs sampling. In *Proceedings of the 3rd international workshop on distributed statistical computing*, 2003.

T. Rainforth. *Automating inference, learning, and design using probabilistic programming*. PhD thesis, University of Oxford, 2017.

T. Rainforth. Nesting probabilistic programs. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

T. Rainforth and F. Wood. Canonical correlation forests. *arXiv preprint arXiv:1507.05444*, 2015.

T. Rainforth, R. Cornish, H. Yang, A. Warrington, and F. Wood. On Nesting Monte Carlo Estimators. *International Conference on Machine Learning (ICML)*, 2018.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.

C. Robert. *The Bayesian choice: from decision-theoretic foundations to computational implementation*. Springer Science & Business Media, 2007.

C. P. Robert. *Monte Carlo methods*. Wiley Online Library, 2004.

G. O. Roberts and A. F. Smith. Simple conditions for the convergence of the Gibbs sampler and Metropolis-Hastings algorithms. *Stochastic processes and their applications*, 49(2):207–216, 1994.

R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo method*, volume 10. 2016.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

J. Salvatier, T. V. Wiecki, and C. Fonnesbeck. Probabilistic programming in python using pymc3. *PeerJ Computer Science*, 2:e55, 2016.

B. Schölkopf and A. J. Smola. Learning with kernels: support vector machines, regularization, optimization, and beyond, 2002.

N. Siddharth, B. Paige, J.-W. van de Meent, A. Desmaison, F. Wood, N. D. Goodman, P. Kohli, P. H. Torr, et al. Learning disentangled representations with semi-supervised deep generative models. *arXiv preprint arXiv:1706.00400*, 2017.

D. Spiegelhalter, A. Thomas, N. Best, and W. Gilks. Bugs 0.5: Bayesian inference using Gibbs sampling manual (version ii). *MRC Biostatistics Unit, Cambridge*, 1996.

S. Staton, H. Yang, F. Wood, C. Heunen, and O. Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 525–534. ACM, 2016.

J. Steinhardt. Beyond Bayesians and frequentists. 2012.

Y. W. Teh. Dirichlet process. In *Encyclopedia of machine learning*, pages 280–287. Springer, 2011.

L. Tierney. Markov chains for exploring posterior distributions. *The Annals of Statistics*, 1994.

A. Todeschini, F. Caron, M. Fuentes, P. Legrand, and P. Del Moral. Biips: software for Bayesian inference with interacting particle systems. *arXiv preprint arXiv:1412.3779*, 2014.

D. Tolpin, J.-W. van de Meent, and F. Wood. Probabilistic programming in Anglican. Springer International Publishing, 2015.

D. Tolpin, J.-W. van de Meent, H. Yang, and F. Wood. Design and implementation of probabilistic programming language Anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages*, page 6. ACM, 2016.

D. Tran, A. Kucukelbir, A. B. Dieng, M. Rudolph, D. Liang, and D. M. Blei. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.

V. N. Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

H. K. Versteeg and W. Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson Education, 2007.

L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. Recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

D. Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

A. Wilson, E. Gilboa, J. P. Cunningham, and A. Nehorai. Fast kernel learning for multidimensional pattern extrapolation. In *NIPS*, pages 3626–3634, 2014.

F. Wood, J. W. van de Meent, and V. Mansinghka. A new approach to probabilistic programming inference. In *AISTATS*, pages 2–46, 2014.