A Review of Mathematical Techniques in Machine Learning

Owen Lewis

May 26, 2010

Introduction

This thesis is a review of an array topics in machine learning. The topics chosen are somewhat miscellaneous; they are chosen to give a broad overview of the field rather than to give a complete picture of any one area.

One thing that all the topics do have in common is that the techniques involved are of mathematical interest. Topics like inference, graphical models and nonparametic Bayesian methods all draw on results in statistics, and material on kernel methods is almost entirely in the language of functional analysis. The approach this thesis takes is somewhere between engineering and mathematics. On the one hand, we don't discuss purely practical matters like implementation details of the algorithms we present. On the other hand, we believe that the material presented, most of which is very new, is still a work in progress, so we often avoid formal proofs, reasoning that these are better explored after techniques are more established. Overall, the emphasis is on giving a presentation of the material that is rigorous and general, but that does not get bogged down in mathematical details.

Chapter 1 begins by discussing inference and sampling, tools that are important for the realization of the models we discuss in subsequent chapters. Chapter 2 discusses graphical models. It presents the three main types of graphical models, but most of the emphasis is on belief propagation, an elegant form of inference that is specifc to graphical models. Chapter 3 presents nonparametric Bayesian techniques, which are relatively new and very popular. The emphasis in this chapter is on ways to define probabaility distributions over infinite dimensional objects like functions, measures and infinite matrices. Chapter 4 is on kernel methods, which now might be regarded as a classical topic in machine learning. This chapeter gives some theoretical justification of the use of kernels, and discusses how kernels relate to regularization. The 5th and last chapter presents some topics in reinforcement learning, giving a perspective on the learning problem that is different than the ones presented in earlier chapters. This chapter surveys some classical techniques in reinforcement learning, and looks at a few approaches to hierarchical reinforcement learning, which remains an exciting research area.

In general, we will use the following notational conventions:

- Bold lowercase letters (e.g. $\mathbf{x}, \mathbf{y}$) will denote variables; bold uppercase letters will denote collections of variables (e.g. $\mathbf{X} = \mathbf{x}_1, \mathbf{x}_2, \ldots$).
- Matrices will generally be plain uppercase letters, (e.g. A.)
- An uppercase P will denote the probability mass function of a discrete probability distribution; a lowercase p will denote the probability density function of a continuous probability distribution.
- In graphical models, filled circles will correspond to observed variables, and open circles will correspond to unobserved ones.

Chapter 1

Inference

As we will see in later chapters, many problems in machine learning can be formulated probabilistically. Unfortunately, many of these formulations involve intractable probability distributions. This chapter presents techniques for overcoming this intractability.

We begin by presenting the EM algorithm, which shows how to make inference possible by the introduction of hidden variables. The EM algorithm can be interpreted as an instance of variational inference, which is a way to approximate an intractable distribution with one that is easier to deal with. Following the EM algorithm, we present variational techniques in general. The chapter concludes by presenting sampling methods. Rather than trying to actually compute distributions, these techniques simply sample from them, and then use the samples to compute quantities of interest. Although this idea is very simple, it often difficult to get obtain the samples. We discuss two methods to deal with this, Gibbs samping and the Metropolis Hastings algorithm, which make sampling possible.

1.1 The EM algorithm

Consider the problem of fitting data to a Gaussian mixture model. We assume that we have n data points $\mathbf{Y} = \mathbf{y}_1, \ldots, \mathbf{y}_n$ that are drawn from a mixture of m Gaussian distributions. The model is specified by a choice of parameters $(\mu_j, \sigma_j)_{j=1,\ldots,m}$ for each of these Gaussians, and by m mixture proportions a_i, so that the each point $\mathbf{y}_i$ is drawn from the mixture:

$$\sum_{j=1}^{m} a_j \mathcal{N}(\mu_j, \sigma_j^2) \tag{1.1.1}$$

We call the collection of parameters θ, with $\theta_j := (a_j, \mu_j, \sigma_j^2)$.

Our goal is to find a maximum likelihood estimate of θ. The problem is that, as currently posed, the likelihood is difficult to optimize. However, let us introduce cluster membership parameters $\mathbf{x}_1, \ldots, \mathbf{x}_n$ where $\mathbf{x}_i = k$ if $\mathbf{y}_i$ was generated by the kth Gaussian.

Now, it is easy to evaluate the likelihood function $L(\theta \mid \mathbf{X}, \mathbf{Y})$ because all we have to do is go through the points $\mathbf{y}_i$, and check the probability that $\mathbf{y}_i$ was drawn from the Gaussian that $\mathbf{x}_i$ assigns it to. Of course, though, we don't actually know $\mathbf{X}$. But if we knew the parameters θ, we could easily infer a distribution over $\mathbf{X}$: $p(\mathbf{x}_i = k \mid \mathbf{Y}, \theta)$ is proportional to the likelihood that $\mathbf{x}_i$ was drawn from $\mathcal{N}(\mu_k, \sigma_k^2)$. Note that this is essentially the same calculation that we used to find the likelihood of the parameters given the hidden variables.

So we essentially have a chicken and egg problem: if we know the parameters, we can compute a distribution over the hidden variables. Conversely, if we know the hidden variables, we can find the ML estimates of the parameters by computing and maximizing the likelihood given the hidden and observed variables. This suggests an iterative scheme wherein we guess the parameters, then guess the distribution of hidden variables based on these estimates, and then update the estimates of the parameters based this estimated distribution of the hidden variables. In fact, this is exactly what the EM algorithm does.

1.1.1 The EM Algorithm in General

The general setting of the EM algorithm is much like the setting in the example: We have visible data $\mathbf{Y}$ and hidden data $\mathbf{X}$, and that we want to maximize the likelihood of a set of parameters $L(\theta \mid \mathbf{Y})$. We assume that maximizing this likelihood directly is intractable, but that, as in the example above, we can compute

- $p(\mathbf{X} \mid \mathbf{Y}, \theta)$
- $L(\theta \mid \mathbf{X}, \mathbf{Y})$

Note that these are not really different calculations since $L(\theta \mid \mathbf{X}, \mathbf{Y}) = p(\mathbf{X}, \mathbf{Y} \mid \theta)$, but the logic of the algorithm, and particularly the chicken and egg nature of the problem is made clearer by presenting them separately.

The EM algorithm has two steps, an (E)xpectation step and a (M)aximization step. Given an estimate $\theta^{(k)}$ of the parameters, by the first point above, we can compute a distribution q^k over the hidden variables.

$$\text{E Step: } q^{(k+1)}(\mathbf{X}) = p(\mathbf{X} \mid \mathbf{Y}, \theta^{(k)}) \tag{1.1.2}$$

With this new knowledge of the hidden variables, we can update our estimate of the parameters to $\theta^{(k+1)}$. Of course, we don't actually have an estimate of the values of the hidden variables; we have an estimate of a distribution over them. This means that we can't directly compute the ML estimates of the parameters. However, we can compute and maximize the *expected* likelihood based on our estimated distribution. This is what we do in the M step:

$$\text{M Step: } \theta^{(k+1)} = \text{argmax}_{\theta} \mathbb{E}_{q^{k+1}}[L(\theta \mid \mathbf{X}, \mathbf{Y})] \tag{1.1.3}$$

Note that this exposition is somewhat unorthodox: The usual presentation has the E step computing the expectation that we put in the M step, and the M step simply maximizing it with respect to θ. We choose this presentation because it facilitates the interpretation of the EM algorithm as variational inference that we will give later on.

1.2 Variational Inference

Variational inference is a technique for approximating difficult distributions with easier ones. The variational framework can be applied in a variety of situations to obtain approximations with many different types of distribution. In this section, we will explore the case in which the approximating distribution is rendered tractable by virtue of the fact that it factorizes. First, though, we develop a situation in which this sort of approximation can be applied. The example we develop, and the subsequent sections, closely mirror the presentation given in [3].

1.2.1 Bayesian Linear Regression

The setup is a standard one in regression problems. Namely, we have a set of input-output pairs $(\mathbf{s}_i, \mathbf{t}_i)_{i=1:n}$, and we assume that $\mathbf{t}_i$ depends on $\mathbf{s}_i$. Since this is linear regression, we assume that:

$$\mathbf{t}_i = \sum_j \mathbf{w}_j \, \phi_j(\mathbf{s}_i) + \epsilon, \tag{1.2.1}$$

where the ϕ are a set of possibly non-linear basis functions, and ϵ is a noise term that we assume follows the Gaussian distribution:

$$\epsilon \sim \mathcal{N}(0, \beta^{-1}) \tag{1.2.2}$$

Note that for notational convenience, we have omitted an offset term or intercept term in (1.2.1). This is not a serious limitation though, since it is possible to imagine that $\mathbf{s}_1$ is a "dummy" data point, $\mathbf{s}_1 = 1$, which would make $\mathbf{w}_1$ the offset.

Since we are doing regression in a Bayesian framework, our goal is to infer the posterior distribution of the weights, $p(\mathbf{W} \mid \mathbf{S}, \mathbf{T})$ assuming some prior distribution on $\mathbf{W}$. Because we have assumed that the noise ϵ follows a Gaussian distribution, we know that $p(\mathbf{T} \mid \mathbf{W}, \mathbf{S})$ is Gaussian, so we can get conjugacy by choosing the prior on $\mathbf{W}$ to be Gaussian as well. Because we assume our variables are i.i.d., it seems to be reasonable to make the prior isotropic and centered at zero:

$$\mathbf{W} \mid \alpha \sim \mathcal{N}(0, \alpha^{-1} I) \tag{1.2.3}$$

We can also place a prior on α. We choose the conjugate prior, as discussed later. The conjugate prior for the inverse variance of a Gaussian with known mean is a Gamma, so we choose

$$\alpha \sim \text{Gam}(a, b) \tag{1.2.4}$$

(The reason that we are working with α^{-1} is that working with the variance directly would give us a messier inverse Gamma prior.)

So our goal is to find the joint posterior $p(\mathbf{W}, \alpha \mid \mathbf{T}, \mathbf{S})$. (For notational convenience, we will call this distribution p). There are a number of ways to do this, but for the purposes of this exposition, we will use variational methods to approximate the true distribution p with a more manageable factorial distribution $q = q_1(\alpha)q_2(\mathbf{W})$. The question now is how to choose the q distributions. We can do this by first applying a quite general result that shows how to use variational methods to approximate an arbitrary posterior distribution with a factorial distribution. Before proceeding, though, we have to introduce some background:

1.2.2 Information Theory

Entropy

Qualitatively, entropy is a measure of uncertainty about the outcome of a discrete random variable that takes values $\mathbf{x}_i$ with probability $P(\mathbf{x}_i)$. We can define entropy as by requiring that it satisfy three axioms:

1. For a fixed number of outcomes, uncertainty should be maximal when the distribtution is uniform. This makes sense, because a uniform distribution is the distribution for which we are least able to make an accurate guess of the outcome (for a peaked distribution, we could guess the peak and expect to be right relatively frequently), and hence the distribution about we are most uncertain. The same reasoning justifies the requirement the entropy of a uniform random variable increase with the number values it can take on.

2. The uncertainty should remain the same if we shuffle the probabilities and assign them to different outcomes. In other words, the entropy should depend only on the probabilities and not on the outcome values.

3. The uncertainty of two independent random variables should be the sum of the uncertainties about each.

It can be shown that the only measure of uncertainty that satisfies the above requirements is:

$$H(X) = -\sum_i P(\mathbf{x}_i)\log P(\mathbf{x}_i) \tag{1.2.5}$$

It can be shown that the entropy as defined above can be interpreted as the expected minimum number of yes/no questions that a questioner following the optimal asking strategy will have to pose in order to determine the value of the random variable from someone who knows its outcome.

Conditional Entropy

Let $\mathbf{x}$ and $\mathbf{y}$ be two discrete random variables. The symbol $H(\mathbf{y} \mid \mathbf{x})$ denotes the conditional entropy of $\mathbf{y}$ given $\mathbf{x}$. $H(\mathbf{y} \mid \mathbf{x})$ tells us how much uncertainty is left in $\mathbf{y}$ if we know the value of $\mathbf{x}$.

$$H(\mathbf{y} \mid \mathbf{x}) = \sum_i P(\mathbf{x}_i) H(\mathbf{y} \mid \mathbf{x} = \mathbf{x}_i) \tag{1.2.6}$$

We have $H(\mathbf{y}) \geq H(\mathbf{y} \mid \mathbf{x})$, with equality iff $\mathbf{x}$ and $\mathbf{y}$ are independent.

1.2.3 Mutual Information

We define the mutual information between random variables $\mathbf{x}$ and $\mathbf{y}$ to be the reduction in uncertainty about $\mathbf{x}$ we get if we know $\mathbf{y}$:

$$I(\mathbf{x}, \mathbf{y}) = H(\mathbf{x}) - H(\mathbf{x} \mid \mathbf{y}) \tag{1.2.7}$$

We see that $I(\mathbf{x}, \mathbf{y}) = 0$ if and only if $H(\mathbf{x}) = H(\mathbf{x} \mid \mathbf{y})$ iff knowing $\mathbf{y}$ does not reduce the uncertainty of $\mathbf{x}$ at all, which happens if and only if X and Y are independent. The term ‘mutual’ is justified by the fact that $I(X, Y) = I(Y, X)$.

Kullback-Leibler (KL) Divergence

The KL divergence is a way of measuring the distance between two probability distributions (note: the word 'distance' is somewhat misleading, because the KL divergence is not symmetric, and therefore does not define a metric). The KL diveregence is defined by:

$$\mathrm{KL}(P(\mathbf{x})||Q(\mathbf{x})) = \sum_i P(\mathbf{x})\log\frac{P(\mathbf{x})}{Q(\mathbf{x})} \tag{1.2.8}$$

An alternative characterization of the mutual information between $\mathbf{x}$ and $\mathbf{y}$ is that

$$I(\mathbf{x},\mathbf{y}) = KL(P(\mathbf{x},\mathbf{y})||P(\mathbf{x})P(\mathbf{y})) \tag{1.2.9}$$

In other words, the mutual information between two variables is high if they are far from being independent.

1.2.4 Variational Approximation with Factorial Distributions

We will frame the general problem as approximating the true posterior distribution $p(\mathbf{X} \mid \mathbf{Y})$ of hidden variables $\mathbf{X}$ conditioned on visible variables $\mathbf{Y}$ with a factorial distribution with two factors: $q(\mathbf{x}_1, \mathbf{x}_2) = q_1(\mathbf{x}_1)q_2(\mathbf{x}_2)$. One idea is to choose q to have minimal KL divergence from the true posterior distribution. However, the whole reason that q is on the table is that we are assuming the true posterior is intractable. So presumably it is not practical to compute a KL divergence involving it. The first step to avoiding this problem is to write

$$\mathrm{KL}(q||p) = -\int q(\mathbf{X})\log\left[\frac{p(\mathbf{X} \mid \mathbf{Y})}{q(\mathbf{X})}\right]d\,\mathbf{X} \tag{1.2.10}$$

Splitting the log gives:

$$= -\int q(\mathbf{X})\big(\log p(\mathbf{X} \mid \mathbf{Y}) - \log q(\mathbf{X})\big)d\,\mathbf{X} \tag{1.2.11}$$

Using the definition of conditional probability and splitting the log again:

$$= -\int q(\mathbf{X})\big(\log p(\mathbf{X},\mathbf{Y}) - \log p(\mathbf{Y}) - \log q(\mathbf{X})\big)d\,\mathbf{X} \tag{1.2.12}$$

Distributing $q(\mathbf{X})$:

$$= \int q(\mathbf{X})\log p(\mathbf{Y})d\,\mathbf{X} - \int q(\mathbf{X})\big(\log p(\mathbf{X},\mathbf{Y}) - \log q(\mathbf{X})\big)d\,\mathbf{X} \tag{1.2.13}$$

Pulling out a constant:

$$= \log p(\mathbf{Y}) \int q(\mathbf{X}) d\,\mathbf{X} - \int q(\mathbf{X}) \log \left[\frac{p(\mathbf{X}, \mathbf{Y})}{q(\mathbf{X})} \right] d\,\mathbf{X} \tag{1.2.14}$$

Since q integrates to 1,

$$\mathrm{KL}(q||p) = \log p(\mathbf{Y}) - F(q), \tag{1.2.15}$$

Where we have defined

$$F(q) := \int q(\mathbf{X}) \log \left[\frac{p(\mathbf{X}, \mathbf{Y})}{q(\mathbf{X})} \right] d\,\mathbf{X} \tag{1.2.16}$$

F is called the *variational free energy*. The key observation is that since $\mathbf{Y}$ is observed, $\log p(\mathbf{Y})$ is fixed. Therefore, the KL divergence may be "squeezed" between this constant and $F(q)$: we may therefore minimize the KL divergence by maximizing F. This is significant because F involves the (assumed to be tractable) joint distribution $p(\mathbf{X}, \mathbf{Y})$ and not the intractable posterior $p(\mathbf{X} \mid \mathbf{Y})$. Therefore, we may approximate the posterior without ever needing to compute it!

In light of our factorizeability assumption, we have

$$F(q) = \int \int q_1(\mathbf{x}_1) q_2(\mathbf{x}_2) \log \left[\frac{p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{Y})}{q_1(\mathbf{x}_1) q_2(\mathbf{x}_2)} \right] d\,\mathbf{x}_1\, d\,\mathbf{x}_2 \tag{1.2.17}$$

Expanding the logarithm and distributing the double integral gives

$$F(q) = \int \int q_1(\mathbf{x}_1) q_2(\mathbf{x}_2) \log p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{Y}) d\,\mathbf{x}_1\, d\,\mathbf{x}_2 \tag{1.2.18}$$

$$- \int \int q_1(\mathbf{x}_1) q_2(\mathbf{x}_2) \log q_1(\mathbf{x}_1) d\,\mathbf{x}_1\, d\,\mathbf{x}_2 - \int \int q_1(\mathbf{x}_1) q_2(\mathbf{x}_2) \log q_2\, \mathbf{x}_2\, d\,\mathbf{x}_1\, d\,\mathbf{x}_2$$

After pulling constants out of integrals, we get

$$F(q) = \int q_2(\mathbf{x}_2) \left[\int q_1(\mathbf{x}_1) \log p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{Y}) d\,\mathbf{x}_1 \right] d\,\mathbf{x}_2 \tag{1.2.19}$$

$$- \int q_2(\mathbf{x}_2) \left[\int q_1(\mathbf{x}_1) \log q_1(\mathbf{x}_1) d\,\mathbf{x}_1 \right] d\,\mathbf{x}_2 - \int q_2(\mathbf{x}_2) \log q_2(\mathbf{x}_2) \left[\int q_1(\mathbf{x}_1) d\,\mathbf{x}_1 \right] d\,\mathbf{x}_2$$

Consider for the moment only the dependence on $q_2(\mathbf{x}_2)$. We make a few observations. First, the inner integral in the second term is constant with respect to $q_2(\mathbf{x}_2)$, so for the purposes of opimization with respect to q_2, we can drop it. Further, since $q_2(\mathbf{x}_2)$ is a probability distribution, it integrates to 1. Thus, for the purposes of optimization, we drop the second term entirely. Last, $q_1(\mathbf{x}_1)$ also integrates to 1, so we can drop the inner integral from the third term. We are left with

$$F(q) = \int q_2(\mathbf{x}_2)\left[\int q_1(\mathbf{x}_1)\log p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{Y})d\,\mathbf{x}_1\right]d\,\mathbf{x}_2 - \int q_2(\mathbf{x}_2)\log q_2(\mathbf{x}_2)d\,\mathbf{x}_2 \tag{1.2.20}$$

Now, we recognize the second term as the negative entropy of the distribution $q_2(\mathbf{x}_2)$, $H(q_2)$. We can also recognize the first term as the negative cross entropy of q_2 with the distribution $s(\mathbf{x}_1, \mathbf{Y})$ defined by

$$\log\, s(\mathbf{x}_2, \mathbf{Y}) = \int q_1(\mathbf{x}_1)\log\, p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{Y})d\,\mathbf{x}_1 = \mathbb{E}_{q_1}[\log p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{Y})] \tag{1.2.21}$$

Therefore, our optimization amounts to maximizing

$$-H(q_2, s) + H(q_2) = -\mathrm{KL}(q_2||s) \tag{1.2.22}$$

So, in order to maximize F with respect to q_2, it suffices to minimize the KL divergence between q_2 and s. Further, we know that this minimization occurs when $q_2 = s$. Thus, we set:

$$\log q_2(\mathbf{x}_2) = \mathbb{E}_{q_1}[\log p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{Y})] \tag{1.2.23}$$

Everything in our derivation was symmetric, so that we also have:

$$\log q_1(\mathbf{x}_1) = \mathbb{E}_{q_2}[\log p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{Y})] \tag{1.2.24}$$

The obvious problem is that these expressions tell us how to estimate one unknown distribution in terms of another unknown distribution So we again have a chicken and egg problem. As before, we use an iteration to alternatingly estimate the two distributions.

1.2.5 Back to Bayesian Linear Regression

Substituting α for $\mathbf{x}_1$ and $\mathbf{W}$ for $\mathbf{x}_2$ in our general results (1.2.23) and (1.2.24), we get:

$$\ln q_2(\mathbf{W}) = \mathbb{E}_{q_1}[\log p(\alpha, \mathbf{W}, \mathbf{T})] \tag{1.2.25}$$

$$\ln q_1(\alpha) = \mathbb{E}_{q_2}[\log p(\alpha, \mathbf{W}, \mathbf{T})] \tag{1.2.26}$$

Note that we have dropped $\mathbf{S}$ from the distribution. The reason is that since $\mathbf{S}$ contains the input variables in the regression problem, it is not influenced by any of the variables we're interested in. We first estimate $\log(q_1(\alpha))$. We have

$$\mathbb{E}_{q_2}[\log p(\alpha, \mathbf{W}, \mathbf{S}, \mathbf{T})] = \mathbb{E}_{q_2}[\log p(\mathbf{T}, \mathbf{W}, \alpha)] \tag{1.2.27}$$

$$= \mathbb{E}_{q_2}[\log\big[p(\mathbf{T} \mid \mathbf{W}, \alpha)p(\mathbf{W} \mid \alpha)p(\alpha)] \tag{1.2.28}$$

But α influences $\mathbf{T}$ only through $\mathbf{W}$, so $\mathbf{T} \perp \alpha \mid \mathbf{W}$, and we have:

$$\mathbb{E}_{q_2}[\log p(\alpha, \mathbf{W}, \mathbf{S}, \mathbf{T})] = \mathbb{E}_{q_2}[\log\big[p(\mathbf{T} \mid \mathbf{W})p(\mathbf{W} \mid \alpha)p(\alpha)\big]] \tag{1.2.29}$$

$$= \mathbb{E}_{q_2}[\log p(\mathbf{T} \mid \mathbf{W}) + \log p(\mathbf{W} \mid \alpha) + \log p(\alpha)] \tag{1.2.30}$$

We are deriving a distribution over α, so we can drop the first term, which is constant with respect to α. Reading off the distributions of the second two terms from (1.2.4) and (1.2.3), we see that we want to evaluate:

$$\mathbb{E}_{q_2}[\log \mathcal{N}(0, \alpha^{-1}I) + \log \mathrm{Gam}(a, b) + k] \tag{1.2.31}$$

Where k is constant with respect to α.

$$= \mathbb{E}_{q_2}\left[\log\left(\frac{1}{(2\pi)^{N/2} \mid \alpha^{-1}I \mid^{1/2}} e^{-\frac{1}{2}\mathbf{W}^\top(\alpha^{-1}I)^{-1}\mathbf{W}}\right) + \log\left(\frac{1}{\Gamma(a)} b^a \alpha^{a-1} e^{-b\alpha}\right)\right] \tag{1.2.32}$$

Now, using the special structure of the matrix $\alpha^{-1}I$ (and recalling that I is $N \times N$) gives us

$$\mathbb{E}_{q_2}\left[\log\left(\frac{1}{(2\pi)^{N/2}}\right) + \frac{N}{2}\log(\alpha) - \frac{\alpha}{2}\mathbf{W}^\top\mathbf{W} + \log\left(\frac{1}{\Gamma(a)} b^a\right) + (a-1)\log\alpha - b\alpha + k\right] \tag{1.2.33}$$

Lumping $\log\left(\frac{1}{(2\pi)^{N/2}}\right)$ and $\log\left(\frac{1}{\Gamma(a)} b^a\right)$ into k, using the linearity of expectation, and the fact that the expectation of a constant is that constant gives:

$$\frac{N}{2}\log\alpha - \frac{\alpha}{2}\mathbb{E}_{q_2}[\mathbf{W}^\top\mathbf{W}] + (a-1)\log\alpha - b\alpha + k \tag{1.2.34}$$

Now, this is the log of a Gamma distribution:

$$q_1(\alpha) = \mathrm{Gam}(a', b') \tag{1.2.35}$$

$$a' := a + \frac{N}{2} \tag{1.2.36}$$

$$b' := b + \frac{1}{2}\mathbb{E}_{q_2}[\mathbf{W}^\top\mathbf{W}]) \tag{1.2.37}$$

Now the task is evaluating the expectation with respect to the unknown distribution q_2. From (1.2.2), (1.2.1) and the i.i.d assumption, we get that $\mathbf{T} \mid \mathbf{W} \sim$

$\prod_{i=1}^{N} \mathcal{N}(\sum_j \mathbf{w}_j\, \phi_j(\mathbf{s}_i), \beta^{-1})$. This piece of information allows us to do an analogous calculation to the one above, and obtain:

$$q_2(\mathbf{W}) = \mathcal{N}(m, v) \tag{1.2.38}$$

Using the notation Φ for the matrix whose i, jth cell is $\phi_j(\mathbf{x}_i)$,

$$m := \beta s \Phi^\top \mathbf{T} \tag{1.2.39}$$

$$v = [\mathbb{E}_{q_1}[\alpha] I + \beta \Phi^\top \Phi]^{-1} \tag{1.2.40}$$

Therefore, the expectation that we need to compute q_1 is $\mathbb{E}_{q_2}[\mathbf{W}^\top \mathbf{W}]) = mm^\top + v$, and the expectation that we need to compute q_2 is $\mathbb{E}_{q_1}[\alpha] = a'/b'$. This then gives the two expectations in terms of one another, showing how the iteration my be performed.

1.2.6 The EM Algorithm As Variational Inference

Having given a general example of the way in which variational inference may be applied, we return the EM algorithm, and show how it can be reframed in a way that closely resembles variational inference. This interpretation was first given by Radford Neal and Geoff Hinton in [13]. This example is interesting, because although the mathematics in this section looks very similar to the mathematics in the previous section, there are several important interpretive differences that show the flexibility of the variational framework.

Equation (1.2.15) is equivalent to the following decomposition:

$$\text{log} p(\mathbf{Y}) = F(q) + \text{KL}(q||p), \tag{1.2.41}$$

where p was the posterior distribution $p(\mathbf{X} \mid \mathbf{Y})$, and q was an approximating distribution, also over the hidden variables $\mathbf{X}$.

For application to the EM algorithm, we condition all the distributions above on θ, giving:

$$\text{log} p(\mathbf{Y} \mid \theta) = F(q, \theta) + \text{KL}(q||p(\mathbf{X} \mid \mathbf{Y}, \theta)), \tag{1.2.42}$$

recognizing the likelihood,

$$\ell(\theta) = F(q, \theta) + \text{KL}\big(q||p(\mathbf{X} \mid \mathbf{Y}, \theta)\big). \tag{1.2.43}$$

There are two important expressions for F. First, a simple rearrangment gives

$$F(q, \theta) = \ell(\theta) - \text{KL}\big(q||p(\mathbf{X} \mid \mathbf{Y}, \theta)\big) \tag{1.2.44}$$

Second, transforming (1.2.16) gives

$$F(q,\theta) = \int_{\mathbf{X}} q(\mathbf{X})\log\left[\frac{P(\mathbf{Y},\mathbf{X} \mid \theta)}{q(\mathbf{X})}\right]d\,\mathbf{X} = \mathbb{E}_q[p(\mathbf{X},\mathbf{Y} \mid \theta)] + H(q), \tag{1.2.45}$$

where $H(q)$ is the entropy of q.

There is an important interpretive difference between the variational inference we do here, and what we did in the factorial approximation case. In the factorial approximation case, we imagined "squeezing" the KL divergence between the fixed probability of the observed data, and the free energy. In this case, we will be changing θ, so the likelihood is no longer fixed. In this case, we use the non-negativity of the KL divergence to see that (1.2.44) shows that the free energy is a lower bound for the likelihood of θ. Therefore, we imagine pushing the likelihood higher by increasing F.

There is also the question of how to interpret F now that it depends on two variables. One answer is to view it as "q-approximate likelihood": (1.2.44) shows that F differs from $\ell(\theta)$ to the extent that q differs from p (as measured by the KL divergence).

We maximize q by alternatingly maximizing it with respect to each of its two variables. We call the two maximizations the E step and the M step, for reasons that will become clear.

$$\text{E Step: } q^{(k+1)} = \text{argmax}_q F(q,\theta^{(k)}) \tag{1.2.46}$$

$$\text{M Step: } \theta^{(k+1)} = \text{argmax}_\theta F(q^{(k+1)},\theta) \tag{1.2.47}$$

We can be more explicit about how to perform these two steps. First, looking back at (1.2.42), and noting that $\ell(\theta)$ is independent of q, we see that maximizing F with respect to q is accomplished by minimizing $\text{KL}(q||p)$. Recalling that since the second argument of F is $\theta^{(k)}$, p is shorthand for $p(\mathbf{X} \mid \mathbf{Y},\theta^{(k)})$, so the E-Step is accomplished by setting $q^{(k+1)} = p(\mathbf{X} \mid \mathbf{Y},\theta^{(k)})$, which is the unique choice that makes the KL divergence vanish. (Since the the KL divergence is always positive, this is the best we can hope to do).

Next, looking at the second equality in (1.2.43) and noting that the entropy of q is independent of θ, we see that the M step is accomplished by setting $\theta^{(k+1)}$ to the value that maximizes $\mathbb{E}_{q^{(k+1)}}[p(\mathbf{X}, fY \mid \theta)]$. Therefore we have:

$$\text{E Step: } q^{(k+1)} = p(\mathbf{X} \mid \mathbf{Y},\theta^{(k)}) \tag{1.2.48}$$

$$\text{M Step: } \theta^{(k+1)} = \text{argmax}_\theta \mathbb{E}_{q^{(k+1)}}[p(\mathbf{X},\mathbf{Y} \mid \theta)] \tag{1.2.49}$$

By comparing with (1.1.2) and (1.1.3), we see that the iteration that we have just derived is the same as the classical EM algorithm that we discussed in the previous section.

The interpretation is that at the E step we increase the lower bound on the likelihood of the current parameters until this bound is tight: because after the kth E step, $\mathrm{KL}(q^{(k+1)}||p(\mathbf{X} \mid \mathbf{Y}, \theta^{(k)})) = 0$, so that $F(q^{(k+1)}, \theta^{(k)}) = \ell(\theta)^{(k)}$. Then, on the M step, θ is adjusted, therefore allowing for a larger lower bound, which is then obtained in the subsequent E step.

Now, since F never decreases, and the lower bound on $\ell(\theta)$ is tight after E step, $\ell(\theta)$ must never decrease. So this formulation of the EM algorithm shows that the algorithm accomplishes the goal of choosing a sequence of parameters with increasing (or at least non-decreasing) likelihoods. It is also possible to prove the stronger statement that if the sequence of F values converges to a local maximum at a certain point, then this point is also a local maximum for the likelihood function.

1.3 Sampling

The variational techniques discussed in the previous sections are appealing because they allow us to compute, at least approximately, a true distribution of interest. There are some cases, though, in which we do not actually need to know the true distribution. For example, we might just be interested in the expectation of the distribution. In cases like this, we can often compute the information we want from samples from the distribution, as we do when approximating an expectation with a sample mean.

The most obvious issue to address is how to obtain these samples. After all, we are assuming that the true distribution is intractable, so presumably it is impossible to sample from it directly. Fortunately, it is often possible to take advantage of some structure in the problem at hand to obtain approximate samples. In the following sections, we present two algorithms that can do this.

1.3.1 Gibbs Sampling

Suppose we have some number of variables $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_n$ and we're interested in the joint distribution $p(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. Often this joint distribution is intractable, since we have to reason about the behavior of all the variables together. However, in many cases, it is relatively easy to infer the distribution of one of the variables $\mathbf{x}_j$ if we assume that all the other variables are known. That is, the conditional distributions $p(\mathbf{x}_j \mid \mathbf{x}_{i,i\neq j})$ are tractable. This suggests an idea for an iterative sampling scheme: initialize all the variables in some way, then follow this loop:

```
for $t = 1$ to numSamples do
    choose $i$
    draw $s \sim p(\mathbf{x}_i \mid \mathbf{x}_{j \neq i})$
    $\mathbf{x}_i \leftarrow s$
  end for
```

The choice of i in the second line can be done in a variety of ways, for example by cycling through the variables sequentially.

This algorithm is intuitively appealing, but we would like a more rigorous theoretical analysis. The first step in this analysis is to view the samples from a Gibbs sampler as forming a Markov chain. Observe that each iteration of the Gibbs sampling algorithm, we obtain a collection of vectors $\mathbf{X}_t$, where $\mathbf{X}_{t+1}$ and $\mathbf{X}_t$ differ in the ith spot. It is easy to see that the transition from $\mathbf{X}_t$ to $\mathbf{X}_{t+1}$ does not on anything that happened before step s. Therefore, we may safely take the $\mathbf{X}_t$ variables to be our Markov chain. (Note that we use the capital letter $\mathbf{X}_t$ because $\mathbf{X}_t$ is the collection of the values of our original variables $\mathbf{x}$ at step t).

For what follows, we will use the idea of the *stationary distribution* of a Markov chain. To get the intuition behind this idea, suppose we have a probability distribution π over states. If we know this distribution at time t, we can calculate what it will be at time $t+1$ by:

$$\pi_{t+1}(\mathbf{X}) = \sum_{\mathbf{X}'} \pi_t(\mathbf{X}')T(\mathbf{X}', \mathbf{X}), \tag{1.3.1}$$

where T is the transition probability. In other words, the transition probabilities tell us how the distribution changes over time. We say that π is a stationary distribution for the Markov chain defined by T, if $\pi_t = \pi_{t+1}$ for all t. Intuitively, this amounts to requiring T to hold π constant.

It is possible to check if a distribution is stationary by checking if it satisfies the so-called *detailed balance* equation. π is a stationary distribution for the Markov chain defined by T if

$$\pi(\mathbf{X})T(\mathbf{X}, \mathbf{X}') = \pi(\mathbf{X}')T(\mathbf{X}', \mathbf{X}) \tag{1.3.2}$$

Intuitively, this says that the probability of starting out in a state $\mathbf{X}$ and moving to $\mathbf{X}'$ is the same as the probability of starting out in $\mathbf{X}'$ and moving to $\mathbf{X}$.

It is easy to check that detailed balance implies the π is stationary. We have

$$\pi_{t+1}(\mathbf{X}) = \sum_{\mathbf{X}'} \pi_t(\mathbf{X}')T(\mathbf{X}', \mathbf{X}), \tag{1.3.3}$$

This sum is over the right hand side of (1.3.2), so we may substitute in the sum over the left hand side:

$$\pi_{t+1}(\mathbf{X}) = \sum_{\mathbf{X}'} \pi(\mathbf{X})T(\mathbf{X}, \mathbf{X}') \tag{1.3.4}$$

$$= \pi(\mathbf{X}) \sum_{\mathbf{X}'} T(\mathbf{X}, \mathbf{X}') \tag{1.3.5}$$

$$= \pi(\mathbf{X}) \cdot 1, \tag{1.3.6}$$

where we have used the fact that $T(\mathbf{X}, \mathbf{X}')$ is a probability distribution over $\mathbf{X}'$ and must sum to one.

We will also need another definition. A Markov chain is said to be *ergodic* if there exists some integer m such that it is possible for any state to transition to any other state in m steps. Ergodic chains are important for our purposes because the distribution of states in an ergodic chain will converge to a stationary distribution if the chain is run for long enough. This points the way to a proof of the convergence of the Gibbs sampler: check that the chain defined by the $\mathbf{X}_t$ is ergodic, and then check that the joint distribution $p(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ satisfies the detailed balance condition. This will show that the joint distribution is the stationary distribution to which ergodicity shows that the chain will converge.

We begin by checking detailed balance. Let $\pi := p(\mathbf{X}) = p(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. By the way we constructed the Markov chain produced by the Gibbs sampler, we know that a state $\mathbf{X}$ and its succesor $\mathbf{X}'$ differ in only one coordinate. Suppose this coordinate is the ith one. Then we have:

$$T(\mathbf{X}, \mathbf{X}') = T\big([\mathbf{x}_i, \mathbf{x}_{j\neq i}], [\mathbf{x}'_i, \mathbf{x}_{j\neq i}]\big) = p(\mathbf{x}'_i \mid \mathbf{x}_{i\neq j}) \tag{1.3.7}$$

Now, by definition of π,

$$\pi(\mathbf{X}) T(\mathbf{x}, \mathbf{x}') = p(\mathbf{X}) p(\mathbf{x}'_i \mid \mathbf{x}_{i\neq j}) \tag{1.3.8}$$

$$= p(\mathbf{x}_i, \mathbf{x}_{j\neq i}) p(\mathbf{x}'_i \mid \mathbf{x}_{i\neq j}) \tag{1.3.9}$$

$$= p(\mathbf{x}_i \mid \mathbf{x}_{j\neq i}) p(\mathbf{x}_{j\neq i}) p(\mathbf{x}'_i \mid \mathbf{x}_{i\neq j}) \tag{1.3.10}$$

But the product of the last two terms is $p(\mathbf{X}')$. So after reordering terms,

$$\pi(\mathbf{X}) T(\mathbf{x}, \mathbf{x}') = p(\mathbf{X}') p(\mathbf{x}_i \mid \mathbf{x}_{i\neq j}) = p(\mathbf{X}') p(\mathbf{x}_i \mid \mathbf{x}'{}_{i\neq j}) = \pi(\mathbf{X}') T(\mathbf{X}', \mathbf{X}) \tag{1.3.11}$$

So detailed balance is satisfied, and we have the desired stationary distribution.

It remains to check ergodicity. In general, ergodicity will depend on the specific distributions involved. But ergodicity is easily verified in the case that the conditional distributions from which we sample at each step are supported everywhere. If this condition is met, then, after all points have been sampled, conditioned on the others, there is a non-zero probability of ending up at any given joint configuration. Plainly, this is sufficient for ergodicity.

1.3.2 The Metropolis Hastings algorithm

Algorithms like Gibbs sampling that construct a Markov chain is the distribution from which we want to sample are called Markov chain Monte Carlo (MCMC) algrotihms.The Metropolis Hastings algorithm is perhaps the most widely used and important MCMC algorithm. As we shall see, it includes Gibbs sampling as a special case. The Metropolis Hastings algorithm is applicable in cases in which we wish to sample from a probability distribution that is difficult to compute explicitly, but that is proportional to a function that is easily tractable. This may appear to be a strange situation, but it is one that in fact arises frequently. We give two examples:

Example 1.3.1. Suppose we apply Bayes' theorem to variables $\mathbf{x}$ and $\mathbf{y}$ to obtain

$$p(\mathbf{x} \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})}{p(\mathbf{y})} = \frac{p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})}{\int p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})d\,\mathbf{x}} \tag{1.3.12}$$

Often, the likelihood term will be specified by the model, and we usually assume that the prior over $\mathbf{x}$ is known. However, the integral in the denominator may be intractable. In this case, the conditions for the application of the Metropolis Hastings algorithm are met: the distribution is proportional to the numerator, which we can evaluate, but the constant of proportionality (given by the integral) is unknown.

Example 1.3.2. In statistical mechanics, one often encounters distributions of the following form:

$$p(\mathbf{s}) = \frac{1}{Z}e^{-\frac{1}{k_B T}\beta E(\mathbf{s})} \tag{1.3.13}$$

Here, s is a microstate of a physical system (i.e. a detailed description of properties of individual molecules), and E is the energy associated with this microstate. The distribution, then, assigns low probability to states with high energy. However, this effect is lessened at high temperatures T. This accords with our intuition that systems at high temperatures should be more likely to have high energy than those at low temperatures. Z is a normalizing constant called the partition function and defined by

$$Z = \sum_{\mathbf{s}'} e^{-\frac{1}{k_B T}E(\mathbf{s}')} \tag{1.3.14}$$

Since this sum is over all possible states of the system, it is generally not tractable. Even in the case of an Ising model in which each molecule may have only two states, the exponential growth in the number of possible states with respect to the number of particles quickly makes calculation of the partition function prohibitively costly. However, usually it is easy to calculate the energy associated with any given state. So here again we have a distribution that is easy to evaluate apart from a normalizing constant, so we have another situation in which the Metropolis Hastings algorithm may be profitably used.

The Metropolis Hastings algorithm first appeared in a somewhat simplified form as the Metropolis algorithm. We will present the Metropolis algorithm first, and then present the full Metropolis Hastings algorithm. We first observe that since we can evaluate $p(\mathbf{x})$ up to a constant, we can evaluate $\frac{p(\mathbf{x}_1)}{p(\mathbf{x}_2)}$ for any pair of points $\mathbf{x}_1$ and $\mathbf{x}_2$. Now, suppose we have some way of generating "candidate" samples from p. The algorithm procedes by generating a first candidate $\mathbf{x}_1$, and then a second candidate $\mathbf{x}_2$, which may be accepted or rejected as a sample. If $\frac{p(\mathbf{x}_2)}{p(\mathbf{x}_1)} > 1$, then $p(\mathbf{x}_2) > p(\mathbf{x}_1)$, and $\mathbf{x}_2$ is at least as good as $\mathbf{x}_2$, so it is accepted. If $\frac{p(\mathbf{x}_2)}{p(\mathbf{x}_1)} < 1$, however, we do not automatically reject $\mathbf{x}_2$, since even if we were able to generate true samples from p, some of these would have low probability.

It is helpful at this point to return to the statistical mechanics example in (1.3.13). If we are in a state $\mathbf{s}_1$, we can calculate explicitly that the probability of transitioning to s_2 with energy $E(s_2) > E(s_1)$. This probability is proportional to $e^{-\frac{1}{k_B T}(E(s_2)-E(s_1))} = \frac{P(s_2)}{P(S_1)}$ (this makes intuitive sense because if we assume that we are already in state s_1, the probability of moving to a new state should depend only on the difference in energies between s_1 and s_2 and on the temperature, not on the absolute energy of s_2.) Therefore, it makes sense to accept $\mathbf{x}_2$ as a sample with this same probability. This motivates the following general acceptance criterion:

$$p(\text{accept }\ \mathbf{x}_2) = \min\left(1, \frac{p(\mathbf{x}_2)}{p(\mathbf{x}_1)}\right) \tag{1.3.15}$$

The algorithm then continues to generate samples by replacing $\mathbf{x}_1$ in (1.3.15) with the next accepted sample, and then replacing this sample, and so on.

It remains to specify how we generate the candidate samples. In the statistical mechanics example, we imagined starting in a state $\mathbf{s}_1$ and transitioning to a new state. We use the same intuition in the general case by taking our current sample into account when choosing our next sample. The motivation for this is that the sampling algorithm will be very inefficient if we reject too many of the potential samples that we generate. If we encourage the subsequent sample to be relatively close to our current (accepted) sample, we can hope that the ratio in (1.3.15) will be close to one. (Of course, we don't want each sample to be *too* close to the previous one, as this would lead to slow convergence of the Markov chain). With these considerations in mind, we say that s_2 is generated from a *proposal distribution* q that depends on s_1:

$$s_2 \sim q(s_2 \mid s_1) \tag{1.3.16}$$

The specific form of the proposal distribution will depend on the application at hand.

As we did with Gibbs sampling, we can check the convergence of the Metropolis algorithm by imagining that the accepted samples form a Markov chain and checking that the distribution p satisfies the detailed balance condition. Recall that checking detailed balance amounts to checking that:

$$p(\mathbf{x})T(\mathbf{x},\mathbf{x}') = p(\mathbf{x}')T(\mathbf{x}',\mathbf{x}) \tag{1.3.17}$$

Substituting in the transition probability from (1.3.15) gives:

$$p(\mathbf{x})q(\mathbf{x} \mid \mathbf{x}')\mathrm{min}\left(1, \frac{p(\mathbf{x}')}{p(\mathbf{x})}\right) = p(\mathbf{x}')q(\mathbf{x}' \mid \mathbf{x})\mathrm{min}\left(1, \frac{p(\mathbf{x})}{p(\mathbf{x}')}\right) \tag{1.3.18}$$

$$\mathrm{min}\left(p(\mathbf{x})q(\mathbf{x} \mid \mathbf{x}'), p(\mathbf{x})q(\mathbf{x} \mid \mathbf{x}')\frac{p(\mathbf{x}')}{p(\mathbf{x})}\right) = \mathrm{min}\left(p(\mathbf{x}')q(\mathbf{x}' \mid \mathbf{x}), p(\mathbf{x}')q(\mathbf{x}' \mid \mathbf{x})\frac{p(\mathbf{x})}{p(\mathbf{x}')}\right) \tag{1.3.19}$$

$$\mathrm{min}\left(p(\mathbf{x})q(\mathbf{x} \mid \mathbf{x}'), q(\mathbf{x} \mid \mathbf{x}')p(\mathbf{x}')\right) = \mathrm{min}\left(p(\mathbf{x}')q(\mathbf{x}' \mid \mathbf{x}), q(\mathbf{x}' \mid \mathbf{x})p(\mathbf{x})\right) \tag{1.3.20}$$

Plainly this is satisfied if and only if $q(\mathbf{x} \mid \mathbf{x}') = q(\mathbf{x}' \mid \mathbf{x})$. So we make this symmetry a requirement of the metropolis algorithm. An example of such a symmetric proposal distribution is the Gaussian $q(\mathbf{x}' \mid \mathbf{x}) = \mathcal{N}(\mathbf{x}, \sigma^2)$, where σ^2 is held constant.

Last, we point out that ergodicity is guaranteed if q is nonzero everywhere that p is nonzero.

The Metropolis Hastings Algorithm

The Metropolis-Hastings Algorithm is an extension of the Metropolis algorithm that is designed to able to work with proposal distributions that are not symmetric. The rejection rule for the Metropolis-Hastings algorithm is

$$p(\text{accept } \mathbf{x}_2) = \mathrm{min}\left(1, \frac{p(\mathbf{x}_2)q(\mathbf{x}_1 \mid \mathbf{x}_2)}{p(\mathbf{x}_1)q(\mathbf{x}_2 \mid \mathbf{x}_1)}\right) \tag{1.3.21}$$

It is easy to check for convergence in the same way that we did for the Metropolis algorithm.

As a last note in this section, we observe that Gibbs sampling be derived as special case of the Metropolis-Hasting algorithm. For a step of Gibbs sampling in which the ith coordinate of $\mathbf{X}$ is to be changed, the proposal $q(\mathbf{X}' \mid \mathbf{X}) = p(\mathbf{x}'_j \mid \mathbf{x}_{j\neq i})$. We did not mention rejection in our discussion of Gibbs sampling, so in order to be consistent, we prove that no samples are rejected in the Metropolis Hasting version of Gibbs sampling. For any $\mathbf{X}'$generated from the proposal distribution, we have

$$p(\text{accept}\,\mathbf{X}') = \frac{p(\mathbf{X}')p(\mathbf{x}_j \mid \mathbf{x}'_{j\neq i})}{p(\mathbf{X})p(\mathbf{x}'_j \mid \mathbf{x}_{j\neq i})} \tag{1.3.22}$$

Expanding $p(\mathbf{X}') = p(\mathbf{x}'_j \mid \mathbf{x}'_{j\neq i})p(\mathbf{x}'_{j\neq i})$ and $p(\mathbf{X}) = p(\mathbf{x}_j \mid \mathbf{x}_{j\neq i})p(\mathbf{x}_{j\neq i})$ gives

$$p(\text{accept}\,\mathbf{X}') = \frac{p(\mathbf{x}'_j \mid \mathbf{x}'_{j\neq i})p(\mathbf{x}'_{j\neq i})p(\mathbf{x}_j \mid \mathbf{x}'_{j\neq i})}{p(\mathbf{x}_j \mid \mathbf{x}_{j\neq i})p(\mathbf{x}_{j\neq i})p(\mathbf{x}'_j \mid \mathbf{x}_{j\neq i})} \tag{1.3.23}$$

But $\mathbf{x}'_{j\neq i} = \mathbf{x}_{j\neq i}$ so this is

$$p(\text{accept}\,\mathbf{X}') = \frac{p(\mathbf{x}'_j \mid \mathbf{x}_{j\neq i})p(\mathbf{x}_{j\neq i})p(\mathbf{x}_j \mid \mathbf{x}_{j\neq i})}{p(\mathbf{x}_j \mid \mathbf{x}_{j\neq i})p(\mathbf{x}_{j\neq i})p(\mathbf{x}'_j \mid \mathbf{x}_{j\neq i})} = 1, \tag{1.3.24}$$

as required.

Chapter 2

Nonparametric Bayes

Rather than trying to give a general definition of nonparametric Bayesian methods, we simply characterize them as methods for defining probability distributions on infinite dimensional objects. Although this definition may not apply to all nonparametric Bayesian models, it adequately describes the ones discussed in this chapter.

Perhaps the prototypical example of an infinite dimensional object is a function, so we begin by discussing Gaussian processes, which are a way to define distributions on functions. Next, we present Dirichlet processes, which give distributions on probability measures. Distributions on more general measures can be obtained by Beta and Gamma processes, and we discuss these next. Last, we present the Indian Buffet process, which defines a distribution on infinite dimensional binary matrices.

2.1 Gaussian Processes

Understanding of Gaussian processes, and indeed almost all nonparametric Bayesian models, is aided by a review of a few definitions from probability theory.

Recall that a *stochastic process* is a collection of random variables on a probability space Ω indexed by variable t that is usually interpreted as time. In other words, a stochastic process is a collection of functions

$$X(\omega, t) : \Omega \times T \to \mathbb{R} \tag{2.1.1}$$

Thus, if we pick $\omega \in \Omega$ randomly according to some probability measure on Ω, then we get a random function

$$f(t) = X(\omega, \cdot) : T \to \mathbb{R}. \tag{2.1.2}$$

A Gaussian process (GP) is a way of generating random functions on $\mathbb{R}$, so we will take T above to be $\mathbb{R}$. Taking any finite dimensional vector $\mathbf{X} \in \mathbb{R}^n$ and applying

f gives a (row) vector $\mathbf{Y} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)]$. One way to specify the behavior of the stochastic process is by requiring that $\mathbf{Y}$ follow some distribution, thereby using a manageable finite dimensional object as a proxy for a thornier infinite dimensional one. The Gaussian process gets its name from the fact that we require that $\mathbf{Y}$ follow an n dimensional Gaussian distribution. For the remainder of this exposition, we will assume that this distribution has mean zero. This assumption means that the distribution is entirely specified by its covariance matrix $Q = \mathbb{E}[\mathbf{Y}^\top \mathbf{Y}]$.

We can make the following observation: Suppose that $\mathbf{x}$ and $\mathbf{x}'$ are two points in $\mathbb{R}$. When we are drawing random vectors as above, the sample size n is arbitrary, as long as it is finite. However, no matter the size of sample, if the points $\mathbf{x}$ and $\mathbf{x}'$ appear together in the sample, the outer product structure of the covariance matrix implies that their contribution to Q will always be the constant $\mathbb{E}[f(\mathbf{x})f(\mathbf{x}'))]$. We make the following definition:

$$\mathbb{E}[y(\mathbf{x})y(\mathbf{x})')] := c(\mathbf{x}, \mathbf{x}') \tag{2.1.3}$$

This definition means that for any sample $\mathbf{x}_1, \ldots \mathbf{x}_n$, the covariance matrix will satisfy

$$Q_{ij} = c(\mathbf{x}_i, \mathbf{x}_j) \tag{2.1.4}$$

Thus, a mean-zero Gaussian process is entirely specified by c, which we will call its *covariance function.* Following the notation for a Gaussian distribution, we will write GP(0,c) for a Gaussian process with mean 0 and covariance function c.

The importance of this (2.1.4) is that, in a manner somewhat reminiscent of the use of kernels in SVMs, we can define the function c *a priori.* If we do this, we immediately have a way to evaluate a function drawn from a GP at a set of points. Suppose this set of points is $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_n$. Then we have:

$$f \sim GP(0, c) \Rightarrow f(\mathbf{x}_1, \ldots, \mathbf{x}_n) \sim \mathcal{G}(0, Q) \tag{2.1.5}$$

$$Q_{ij} = c(\mathbf{x}_i, \mathbf{x}_j) \tag{2.1.6}$$

For example, the three GP draws visualized in figure 2.1 were obtained by setting $\mathbf{X} = [-3 : .01 : 5]$, and $c = 2\text{exp}\left[\frac{-(\mathbf{x} - \mathbf{x}')^2}{2}\right]$.

We can pause at this point to give some intuition about what is going on in figure 2.1. Our choice of a covariance function of the form $k \cdot \exp\left[\frac{-(x-x')^2}{2}\right]$ is a common one, that can be justified by the observation that this covariance function will give covariances that decay rapidly as the distance between the associated input values grows. Conversely, the covariance between the output values corresponding to input values that are close together will be large. This last fact implies that the

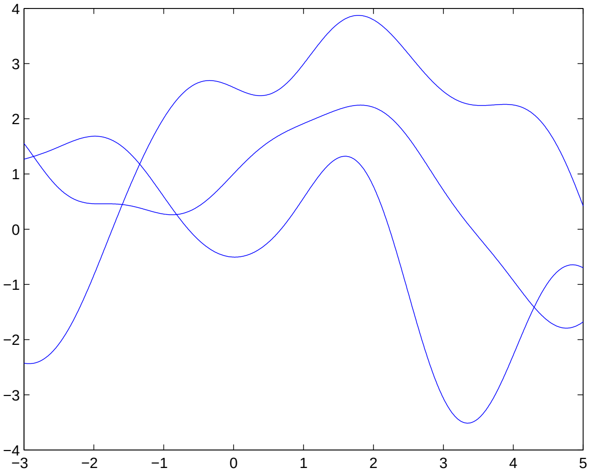

Figure 2.1: Three draws from $\mathrm{GP}(0, 2\exp\left[\frac{-(\mathbf{x}-\mathbf{x}')^2}{2}\right])$.

output values corresponding to input values that are close together will not in general differ very greatly. This property encourages functions drawn from a GP with this covariance function to be smooth. In general, the choice of covariance functions gives the programmer control over the properties that are drawn from the corresponding GP. For example, if there is reason to believe that one's data will be best modeled with a periodic function, one could choose a covariance function that will give this property.

2.1.1 Gaussian Process Regression

Gaussian processes are used in a variety of applications. One that is particularly important for machine learning is regression. Suppose we are given a training set with inputs $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_n$ and outputs $\mathbf{T} = \mathbf{t}_1, \ldots, \mathbf{t}_n$. If we are given a new input point $\mathbf{x}_{n+1}$, our task is to infer the corresponding output $\mathbf{t}_{n+1}$ or, better yet, a distribution over $\mathbf{t}_{n+1}$. In particular, we are interested in the conditional distribution $p(\mathbf{t}_{n+1} \mid \mathbf{T}, \mathbf{X})$. Leaving the dependence on $\mathbf{X}$ implicit, we have:

$$p(\mathbf{t}_{n+1} \mid \mathbf{T}) = \frac{p(\mathbf{T}, \mathbf{t}_{n+1})}{p(\mathbf{T})} \tag{2.1.7}$$

Since training points $\mathbf{T}$ are observed, the likelihood $p(\mathbf{T})$ is a constant. Moreover, we are using a GP model so we have the distribution of the vector $[\mathbf{T}, \mathbf{t}_{n+1}]$. So we have:

$$P(t_{N+1} \mid \mathbf{t}_N) \propto \exp\left[-\frac{1}{2}[\mathbf{T}, \mathbf{t}_{n+1}] Q_{n+1}^{-1} [\mathbf{T}, \mathbf{t}_{n+1}]^T \right], \tag{2.1.8}$$

where, extending the definition of the covariance function to vectors in the obvious way,

$$Q_{n+1} = c([\mathbf{X}, \mathbf{x}_{n+1}], [\mathbf{X}, \mathbf{x}_{n+1}]) \tag{2.1.9}$$

$$= \left[\begin{array}{cc} Q_n & \mathbf{k} \\ \mathbf{k}^T & \kappa \end{array} \right] \tag{2.1.10}$$

where $Q_n = c(\mathbf{X}, \mathbf{X})$, $\mathbf{k} = c(\mathbf{X}, \mathbf{x}_{n+1})$, and $\kappa = c(\mathbf{x}_{n+1}, \mathbf{x}_{n+1})$. In order to actually use (2.1.8) for prediction, we have to invert Q_{n+1}. Fortunately, the partitioned representation of Q_{n+1} gives an easy formula for Q_{n+1}^{-1}:

$$Q_{n+1}^{-1} = \left[\begin{array}{cc} \mathbf{M} & \mathbf{m} \\ \mathbf{m}^T & m \end{array} \right] \tag{2.1.11}$$

$$m = (\kappa - \mathbf{k}^T Q_N^{-1} \mathbf{k})^{-1} \tag{2.1.12}$$

$$\mathbf{m} = -m Q_N^{-1} \mathbf{k} \tag{2.1.13}$$

$$\mathbf{M} = Q_N^{-1} + \frac{1}{m} \mathbf{m}\mathbf{m}^T \tag{2.1.14}$$

Note that these expressions have the advantage that the only brute-force matrix inversion required is of the covariance matrix for the training points Q_n. This inverse can be reused in order to make predictions at multiple test points.

Now we can substitute the inverse we have just obtained into the distribution (2.1.8). This gives

$$P(t_{N+1} \mid \mathbf{t}_N) \propto \exp\left[-\frac{1}{2} \frac{(t_{N+1} - \hat{t}_{N+1})^2}{\sigma_{\hat{t}_{N+1}}^2} \right] \tag{2.1.15}$$

The mean of this distribution is

$$\hat{t}_{N+1} = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{t}_N \tag{2.1.16}$$

and its variance is

$$\sigma_{\hat{t}_{N+1}}^2 = \kappa - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{k} \tag{2.1.17}$$

Therefore, our prediction of the new target value t_{n+1} will be, as the notation suggests, $\hat{t}_{n+1}$. The variance can be used to give error bars for this prediction.

2.2 The Dirichlet Process

The Gaussian process gave us a way to get distributions over functions; the Dirichlet process gives us a way to get a distribution over probability measures. Because probability measures are functions on σ algebras, not on the sets that underly them, in order to generate distribution on a set Ω, we need to set T in (2.1.2) to be a σ algebra of subsets of Ω.

Just as we defined Gaussian processes by specifying the distribution of the function values of finite collections of points, we now define Dirichlet processes by specifying their behavior on finite collections of subsets, or rather on finite partitions. A Dirichlet process has two parameters, a real number α called the *concentration parameter* and probability distribution H Ω called the *base distribution.* We say that a distribution G is drawn from $\mathrm{DP}(\alpha, H)$ if for any finite partition of Ω into measurable sets $A_1, \ldots, A_n$ we have

$$(G(A_1), \ldots, G(A_n)) \sim \mathrm{Dir}(\alpha H(A_1), \ldots, \alpha H(A_n)) \tag{2.2.1}$$

This definition does not guarantee that stochastic processes with the required properties exist. We will prove existence by giving a concrete construction, but first we will derive some of the properties of DPs, taking their existence for granted. Our exposition from this point on follows [19]. First we examine the posterior and predictive distributions of a draw from a DP.

Let G be a draw from a Dirichlet process, and let $\omega_1, \ldots, \omega_m$ be draws from it. We are interested in inferring G from these draws. As before, we consider only partitions, so let $A_1, \ldots, A_n$ be a finite partition of Ω. We are interested in the posterior distribution

$$p(G(A_1), \ldots, G(A_n) \mid \omega_1, \ldots, \omega_m). \tag{2.2.2}$$

Note that for determining the measures of the sets A_i, the actual identities of the points ω_i are irrelevant: all that matters are the number points that lie in each set in the partition. Therefore, we define a vector $\mathbf{c}$ of counts, so that c_i is the number of points ω that lie in A_i. This reduces our task to finding the posterior $p(G(A_1), \ldots, G(A_n) \mid \mathbf{c})$.

We proceed by evaluating the likelihood $p(\mathbf{c} \mid G(A_1), \ldots, G(A_n))$. It is easy to see that:

$$\mathbf{c} \mid [G(A_1), \ldots, G(A_n)]) \sim \mathrm{Mult}([G(A_1), \ldots, G(A_n)]) \tag{2.2.3}$$

So, we have a multinomial likelihood, and, by the definition of the Dirichlet process, we have a Dirichlet prior. Therefore, we can use the conjugacy between these two distributions and simply look up the posterior. It is:

$$(G(A_1), \ldots, G(A_n)) \mid \omega_1, \ldots, \omega_m \sim \mathrm{Dir}(\alpha H(A_1) + c_1, \ldots, \alpha H(A_n) + c_n) \tag{2.2.4}$$

Moreover, looking back at the definition, (2.2.1), we see that the posterior distribution of G is again a Dirichlet process. We see that the unnormalized base measure is $\alpha H + \sum_i \delta_{\omega_i}$, with a concentration parameter of 1. So the normalized base measure is $\frac{1}{\alpha+m}\big(\alpha H + \sum_i \delta_{\omega_i}\big)$. We correct for the normalization by rechoosing the concentration parameter to be $\alpha + m$:

$$G \mid \omega_1, \ldots, \omega_m \sim \mathrm{DP}\bigg(\alpha + m, \frac{1}{\alpha+m}\big(\alpha H + \sum_i \delta_{\omega_i}\big)\bigg) \tag{2.2.5}$$

For the purposes of what follows, the main application of (2.2.5) is in determining the predictive distribution of ω_{m+1}. We first observe that $p(\omega_{m+1} \in A \mid \omega_1, \ldots, \omega_m) = \mathbb{E}[G(A)] \mid \omega_1, \ldots, \omega_m$, where the expectation is taken over draws from the Dirichlet process. In order to evaluate this expectation, we use (2.2.5) with the partition A, A^c and then apply the definition of the Dirichlet process to get:

$$G(A), G(A^c) \mid \omega_1, \ldots, \omega_m \sim \mathrm{Dir}\bigg(\frac{1}{\alpha+m}\big(\alpha H(A) + \sum_i \delta_{\omega_i}(A), \frac{1}{\alpha+m}\big(\alpha H(A^c) + \sum_i \delta_{\omega_i}(A^c)\big)\bigg) \tag{2.2.6}$$

So the expectation we seek is just the expectation of the first component of a draw from the left hand side above. So we have:

$$P(\omega_{m+1} \in A \mid \omega_1, \ldots, \omega_m) = \frac{1}{\alpha+m}\big(\alpha H(A) + \sum_i \delta_{\omega_i}(A)\big) \tag{2.2.7}$$

Since the set A was an arbitrary measurable set, we have:

$$\omega_{m+1} \mid \omega_1, \ldots, \omega_m \sim \frac{1}{\alpha+n}\big(\alpha H + \sum_i \delta_{\omega_i}\big) \tag{2.2.8}$$

We can make two observations. First, we can now interpret the parameter α. If α is large, the H will dominate the predictive distribution - hence the name concentration parameter: the larger α is, the more concentrated G will be around H. Second, because of the inclusion of the δ measures, we see that the probability that ω_{m+1} will be equal to one of the points already drawn is nonzero. This shows that with probability 1, G is a discrete probability distribution. In fact, it is possible to show that the expected number of unique points among m drawn from G is only $O(\alpha \mathrm{log} m)$.

The fact that the draws $\omega_1, \ldots, \omega_m$ may not be unique, motivates us to rewrite the predictive distribution in the following equivalent form:

$$\omega_{m+1} \mid \omega_1, \ldots, \omega_m \sim \frac{1}{\alpha+n}\big(\alpha H + \sum_j c_j \delta_{\omega_j^*}\big), \tag{2.2.9}$$

where ω_j^* are the unique values, and c_j is the number of times that the value ω_j^* appeared in the draw.

The Chinese Restaurant Process

Equation (2.2.9) is the basis for a useful metaphor for understanding Dirichlet processes. We can imagine generating a draw from a DP in the following way: First, pick the parameters of the DP: α and H, where H. Then imagine the following scenario. Customers enter a (Chinese) restaurant and seat themselves at tables, of which there are infinitely many. The first customer sits at the first table. Subsequent customers can either sit at an already occupied table, or sit at a new table. Specifically, suppose that m have at least one customer sitting at them, and that for $i = 1, \dots, m$, the ith table has N_i customers at it. Then $(n+1)$st will sit at table i with probability kN_i, and at a new table with probability $k\alpha$. Here k is a normalizing constant chosen so that the probabilities sum to 1, $k = \frac{1}{n+\alpha}$.

Stick-Breaking Construction

We now give a construction that proves the existence of the Dirichlet process. We have shown that draws from a DP are discrete with probability 1, so we have the representation:

$$G = \sum_i w_i \delta_{\omega_i} \tag{2.2.10}$$

Therefore, it suffices to choose the w_i and the ω_i. We do this as follows: start with a stick of length one, and break it a point b_1 that is drawn from a Beta(1,α) distribution. Keep the piece on the left; its length is w_1. Now draw a point according to H; this is ω_1. We would like to repeat this process with what remains of the stick. The problem is that the remaining piece does not have length 1, so we need a correction. We draw another value b_2 from Beta(1,α) as before. The correction is then to shrink it so that it is in the required range $[0, 1 - b_1]$. We keep the result of the shrinking as w_2.

$$w_2 = (1 - b_1)b_2 \tag{2.2.11}$$

Since $b_2 \leq 1$, this multiplication has the desired effect. As before, we draw ω_2 from H.

We then repeat this process ad infinitum.

$$w_n = b_n \prod_{i=1}^{n-1} (1 - b_i) \quad b_n \sim \text{Beta}(1, \alpha) \tag{2.2.12}$$

$$\omega_n \sim H \tag{2.2.13}$$

Plugging these into (2.2.10) gives the desired distribution.

2.2.1 Dirichlet Process Mixture Models

One important use of Dirichlet processes is to do mixture modeling. In traditional mixture modeling, one tries to model data as being produced by a distribution that is a mixture of a finite and fixed number of component distributions. The problem here is that, in general, one does not know the right number of components to use. As we shall see, we can use Dirichlet processes to avoid this problem.

A *Dirichlet process mixture model* is used to model data $\mathbf{X} = \{\mathbf{x}_i\}_{i=1,\ldots,n}$ as a mixture of parametrized distributions $F(\theta)$ via the following generative process.

$$\mathbf{x}_i \sim F(\theta_i) \tag{2.2.14}$$

$$\theta_i \sim G \tag{2.2.15}$$

$$G \sim \mathrm{DP}(\alpha, H) \tag{2.2.16}$$

For example, we might have $F(\theta_i) = \mathcal{N}(\theta_i, \sigma^2)$, where we assume that σ^2 is known. We would then want choose the right number of Gaussians to model our data, and to choose the means of these Gaussians.

Looking at (2.2.14)-(2.2.16), we see that each point is allowed have its own mixture component. Since it is nonsensical for there to be more components than data points, this amounts to having an unbounded number of mixture components available. *A priori*, it would seem that this scheme would lead to trivial models, since giving each its mixture component is unlikely to tell us anything useful. However, we note that the θ_i are draws from a draw from a DP, and that we therefore expect only $O(\alpha \mathrm{log} n)$ of the θ_i to be unique. Thus, although an unbounded number of mixture components are available, we expect only a relatively small number of them to actually appear. Importantly, this small number is not set beforehand, meaning that we have achieved the desired flexibility with respect to the number of components.

Now, (2.2.14)-(2.2.16) tells us how to generate a set of data points. If we want to model data that is already given, we have to invert this process. This inversion can be done with the sampling methods discussed earlier.

Gibbs Sampling for Dirichlet Process Mixture Models

In [14], Radford Neal explores a variety of ways to sample for inference in Dirichlet process mixture models. We choose to present the simplest one here, we note though that Neal stresses the fact that it is not a very effective method in practice.

We are interested in the posterior distribution $p(\theta \mid \mathbf{X})$. To use Gibbs sampling, we have to compute the conditional distributions $P(\theta_i \mid \theta_{j \neq i}, \mathbf{X})$. Fortunately, this can be done fairly easily. From the discussion of the Chinese restaurant process and of the predictive distribution of draws from a DP, we know that:

$$p(\theta_i \mid \theta_{j \neq i}) = \frac{1}{n-1+\alpha} \sum_{j \neq i} \delta_{\theta_j}(\theta_i) + \frac{\alpha}{n-1+\alpha} H, \tag{2.2.17}$$

where we have used the notation $\delta\theta_j$ for the δ function supported at θ_j. Since the posterior we are interested in is also conditioned on the data $\mathbf{X}$, we also have to compute the likelihood $P(\theta_i \mid \mathbf{X})$:

$$p(\theta_i \mid \mathbf{X}) = F(\mathbf{x}_i; \theta_i), \tag{2.2.18}$$

where we have used the fact that each data point interacts only with its own parameter. Therefore, the overall conditional distribution is a mixture of this likelihood and (2.2.17):

$$p(\theta_i \mid \theta_{j \neq i}, \mathbf{X}) = b\Bigg(\sum_{j \neq i} F(x_i; \theta_j) \delta_{\theta_j}(\theta_i) + \alpha F(x_i; \theta_i) H(\theta_i) \Bigg) \tag{2.2.19}$$

Here, b is a normalizing constant into which we have absorbed the factor $\frac{1}{n-1+\alpha}$. There is one point to note. The reason that θ_j, rather than θ_i appears in the sum is that the δ function ensures that contribution of a term to the summand is non-zero only if $\theta_i = \theta_j$. The substitution enables us to interpret the sum as evaluating the likelihood we would obtain by setting θ_i to each of the fixed θ_j. We can now evaluate the normalizing constant b:

$$b = \left(\int \left[\sum_{j \neq i} F(x_i; \theta_j) \delta_{\theta_j}(\theta_i) + \alpha F(x_i; \theta_i) H(\theta_i) \right] d\theta_i \right)^{-1} \tag{2.2.20}$$

This is:

$$b = \left(\sum_{j \neq i} F(x_i; \theta_j) + \alpha \int F(x_i; \theta) H(\theta) d\theta \right)^{-1} \tag{2.2.21}$$

The only problem is the evaluation of the integral $\int F(x_i; \theta) H(\theta_i) d\theta$. In order to keep things simple here, we will pick the base measure H to be conjugate to F. With this in place, this Gibbs sampling is completely straightforward. This concludes our discussion of Gibbs sampling for Dirichlet Process mixture models, but we give an example below that shows how the integral can be computed.

2.2.2 Conjugate Priors

Bayes' theorem says,

$$p(\mathbf{x} \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})}{p(\mathbf{y})} \tag{2.2.22}$$

This expression is useful because we often know all the quantities on the right hand side when we do not know the term on the left. Sometimes, though, it is difficult to compute the right hand side explicitly. There is, however, one special case in which the calculation is easy, indeed trivial. This is the case in which the prior distribution $p(\mathbf{x})$ is *conjugate* to the likelihood $p(\mathbf{y} \mid \mathbf{x})$. In this case, the posterior is of the same form as the likelihood; only the parameters change.

Example 2.2.1. Suppose that we want to model a set of data $\mathbf{X} = \{\mathbf{x}_i\}_{i=1,\ldots,n}$ with a Gaussian for which we know the variance. The mean μ is unknown, so we place a prior over it. By Bayes' theorem:

$$p(\mu \mid \mathbf{X}) = \frac{p(\mathbf{x} \mid \mu)p(\mu)}{p(\mathbf{X})} = \frac{\mathcal{N}(\mathbf{X}; \mu, \sigma^2)p(\mu)}{p(\mathbf{X})} \tag{2.2.23}$$

Now, most choices of $p(\mu)$ will lead to an intractable posterior. However, the conjugate distribution to a normal distribution is again normal. So let us choose $p(\mu) = \mathcal{N}(\mu; \mu_0, \sigma_0^2)$. Then we can simply look up the posterior and see that

$$p(\mu \mid \mathbf{x}) = \mathcal{N}(\mu; \mu_2, \sigma_2^2) \tag{2.2.24}$$

$$\mu_2 = \left(\frac{\mu_0}{\sigma_0^2} + \frac{\sum_{i=1}^n x_i}{\sigma^2}\right) \Big/ \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right) \tag{2.2.25}$$

$$\sigma_2^2 = \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right)^{-1} \tag{2.2.26}$$

Another reason that conjugate priors are useful is that they let us compute integrals that would otherwise be intractable.

Example 2.2.2. In the above discussion of Dirichlet Process mixture models, we had to calculate: $\int F(\mathbf{x}_i \mid \theta)H(\theta)d\theta$. F is known ahead of time, and we are free to choose H. The key observation for this problem comes directly from rewriting Bayes' theorem, using F as the likelihood and H as the prior on θ.

$$F(\theta \mid \mathbf{x}_i) = \frac{F(\mathbf{x}_i \mid \theta)H(\theta)}{p(x_i)} = \frac{F(\mathbf{x}_i \mid \theta)H(\theta)}{\int F(\mathbf{x}_i \mid \theta)H(\theta)d\theta} \tag{2.2.27}$$

So the integral we want to evaluate is now in the denominator. Now, if we choose H conjugate to F, then we know the posterior, which means that we know the normalizing constant of the posterior, which is exactly the integral we want to compute. For example, if we have $F(\theta_i) = \mathcal{N}(\theta_i, \sigma^2)$, and we choose the base distribution H to be $\mathcal{N}(\mu; \mu_0, \sigma_0^2)$, then we can apply the results of (2.2.24) - (2.2.26) to conclude the integral is the normalizing constant for the posterior normal distribution, which is

$$\int F(\mathbf{x}_i \mid \theta) H(\theta) d\theta = \frac{1}{\sqrt{2\pi\sigma_0^2}} \tag{2.2.28}$$

2.2.3 The Hierarchical Dirichlet Process

We have shown how to model a group of data points with a Dirichlet process mixture model. Suppose now that we have multiple groups of data from a common origin. We want to model each group with a Dirichlet process mixture model, but we want to encode our knowledge about the origin of the data by sharing some of the parameters θ_i across the groups.

We could try to do this sharing by drawing the parameters for each group from the same Dirchlet process. In some applications, though, we would like a weaker way to share information. We can accomplish this by adding a hierarchical layer to the Dirichlet process mixture model framework. To see what this means, we will introduce some notation. The setup is that we have J groups, with the jth containing I data points $\mathbf{x}_{ji}$. We have:

$$\mathbf{x}_{ji} \sim F(\theta_{ji}) \tag{2.2.29}$$

$$\theta_{ji} \sim G_j \tag{2.2.30}$$

$$G_j \sim DP(\alpha, G_0) \tag{2.2.31}$$

The question now is how to define G_0. It would defeat our purpose to make G_0 a continuous distribution, because this would mean that, with probability 1, the G_j would be supported at different atoms, which would mean that none of the θ values would be shared across groups. Therefore, we have to make G_0 a discrete distribution. In this case, all the G_j will be weighted sums of δ measures that are supported at the same points; only weights will differ. However, all the usual choices of discrete distribution seem too restrictive. In particular, we do not want to limit the support of G_0.

The solution proposed in [20] is to make G_0 itself a draw from a DP: with the right choice of base distribution, this will give us a discrete distribution that meets our needs. In this case, it is a discrete distribution that is not limited to any particular form and that does have restricted support. The complete model is then:

$$\mathbf{x}_{ji} \sim F(\theta_{ji}) \tag{2.2.32}$$

$$\theta_{ji} \sim G_j \tag{2.2.33}$$

$$G_j \sim DP(\alpha, G_0) \tag{2.2.34}$$

$$G_0 \sim DP(\alpha', H) \tag{2.2.35}$$

In order to get the desiderata of the previous paragraph, we generally choose H to be continuous with wide support.

The Chinese Restaurant Franchise

As the Chinese restaurant process is to Dirichlet processes, the Chinese restaurant franchise is to hierarchical Dirichlet processes. The idea is that we now have J restaurants (corresponding to groups of data), all sharing a menu, on which the dishes are the parameters θ. Customers in each group seat themselves in their own restaurants, using a CRP with concentration parameter α. Next, completely abandoning narrative plausibility, the now-occupied tables sit themselves at master tables, according to a CRP with concentration parameter α'. Each master table is then assigned a draw from H.

The metaphor in [20] makes more sense as a story but this one may be somewhat more straightforward.

2.2.4 Completely Random Measures

Completely random measures have caught on in machine learning more recently than the other methods we have discussed. We present here some results reviewed in [9].

A *completely random measure* on a set X with σ algebra F is a measure μ on X such that whenever $A_1, A_2 \in F$ are disjoint, $\mu(A_1)$ and $\mu(A_2)$ are independent as random variables. An immediate consequence of this definition is that no probability measure can be completely random, because for any measurable set A, A and A^c are clearly disjoint, but $\mu(A)$ determines $\mu(A^c)$, as $\mu(A^c) = 1 - \mu(A)$.

We will now give one way to construct completely random discrete measures. Discreteness means that our completely random measure μ will be of the form

$$\mu = \sum_{i=1}^{\infty} p_i \delta_{\omega_i}, \tag{2.2.36}$$

meaning that all we have to do is specify p_i and ω_i. We do this via a generalization of the Poisson process. In an ordinary Poisson process, the number of events occuring in an interval is a Poisson random variable with rate equal to the Lebesgue measure of that interval. To generalize this setup to arbitrary spaces on which the Lebesgue measure will not in general be defined, we say that number of events occuring in a measurable set will be a Poisson random variable with rate equal to the measure of the set, where the measure is now allowed to an arbitrary σ finite measure. This measure is called the *intensity measure* of the process.

So, in order to define the points ω_i, we put an arbitrary measure on Ω, and run a Poisson process with respect to this measure. The event points at which the events of this process occur are the points we require. We do exactly the same thing to define the weights p_i: we choose a sigma finite measure on $\mathbb{R}$, and run the associated Poisson process, keeping the points at which events occur and using them as the p_i.

It is easy to see that the measure so defined is completely random, since the number of events of a Poisson process that occur in two disjoint subsets are independent.

We have remarked that a probability measure can never be completely random, which implies that a draw from a Dirichlet process can never be a completely random measure. However, it is interesting to note that the draws from a Dirichlet process can be obtained by normalizing completely random measures that are generated by a process called the *Gamma process*. This process generates the weights p_i by a Poisson process whose intensity measure is the *improper Gamma distribution*:

$$cp^{-1}e^{-cp}dp \tag{2.2.37}$$

where c is a parameter. The measure on Ω used for the Poisson process to generate the support of μ is allowed to be an arbitrary σ finite measure that we will call G_0.

We will spend more time on a second way of generating completely random measures called the *Beta process*. It is defined similarly to the Gamma process, but with the improper Gamma distribution replaced by the *improper Beta distribution*

$$cp^{-1}(1-p)^{c-1}dp, \tag{2.2.38}$$

In this case, we will call the base distribution on Ω B_0 and write $\mu \sim \mathrm{BP}(c, B_0)$ for a measure generated by the Beta process.

One interesting aspect of the Beta process is its relationship to the Indian buffet process, another commonly-encountered entity in nonparametric Bayesian models. In order to establish this relationship, we need some probabilistic background.

Exchangeability and de Finetti's Theorem

A sequence of random variables $\mathbf{x}_1, \mathbf{x}_2, \ldots$ is said to be *infinitely exchangeable* if the joint probability of any finite subset is invariant under permutations. That is, if $S \subset \mathbb{N}$ is a set of size n, then

$$p(\mathbf{x}_{S(1)}, \ldots, \mathbf{x}_{S(n)}) = p(\mathbf{x}_{\sigma(S(1))}, \ldots, \mathbf{x}_{\sigma(S(n))}), \tag{2.2.39}$$

where σ is any permutation. In general, the notion of exchangeability is stronger is stronger than the notion of identical distribution but weaker than independence. However, a result called *de Finetti's theorem* says that any exchangeable collection of random variables is conditionally independent given some entity $\mathbf{y}$.

For any sequence, we have

$$p(\mathbf{x}_1, \dots, \mathbf{x}_n) = \int p(\mathbf{x}_1, \dots, \mathbf{x}_n \mid \mathbf{y}) dp(\mathbf{y}) \tag{2.2.40}$$

If the sequence is exchangeable, de Finetti's theorem says that we get conditional independence with respect to $\mathbf{y}$, which means that we have

$$p(\mathbf{x}_1, \dots, \mathbf{x}_n) = \int \prod_{i=1}^{n} p(\mathbf{x}_i \mid \mathbf{y}) dp(\mathbf{y}) \tag{2.2.41}$$

In the literature the distribution $p(\mathbf{y})$ is known as the *de Finetti mixing distribution* [21].

An example of the how de Finetti's theorem is applied is given by the CRP.

The CRP and Exchangeability

As we have seen, the Chinese restaurant process will produce sequences of values ω_i, in which many of the values are repeated. For example, a draw might be

$$\omega_1, \omega_2, \omega_1, \omega_3, \omega_2, \dots \tag{2.2.42}$$

An important fact is the probability of a sequence of points produced by the CRP *depends only on the number of times each unique point appears.* Put another way, the the only things that determines the probability of a seating arrangement produced from a CRP is the number of customers at each table (i.e an arrangement with three customers at the first table and four at the second has the same probability as the arrangement with four customers at the first table and three at the second). We can present a brief example that should make the idea clear. Consider a situation in which we draw only four values, among which there are only two unique points ω_1 and ω_2. If our claim is correct, we should have

$$p(\omega_1, \omega_1, \omega_1, \omega_2) = p(\omega_1, \omega_2, \omega_1, \omega_1) \tag{2.2.43}$$

(Note that these two sequences were chosen arbitrarily: the claim should hold for any two sequences such that ω_i appears the same number of times in the both for $i = 1, 2$.) We can evaluate the probabilities explicitly by using (2.2.9). We see that:

$$p(\omega_2, \omega_1, \omega_1, \omega_1) = 1 \cdot \frac{\alpha}{1+\alpha} \cdot \frac{1}{2+\alpha} \cdot \frac{2}{3+\alpha} = \frac{1 \cdot \alpha \cdot 1 \cdot 2}{1(1+\alpha)(2+\alpha)(3+\alpha)} \tag{2.2.44}$$

And

$$p(\omega_1, \omega_1, \omega_2, \omega_1) = 1 \cdot \frac{1}{1+\alpha} \cdot \frac{\alpha}{2+\alpha} \cdot \frac{2}{3+\alpha} = \frac{1 \cdot 1 \cdot \alpha \cdot 2}{1(1+\alpha)(2+\alpha)(3+\alpha)} \qquad (2.2.45)$$

Commutativity then gives the desired equality. The point that this example illustrates is that permuting the draws from a CRP only permutes the numerators in (2.2.9): the denominators only depend on α the number of points drawn. But permutation of the numerators is plainly irrelevant, since they appear together in a product.

Now that we have shown that the CRP gives exchangeable sequences, it is clear that the condition for de Finetti's theorem is met. Further, we know that points generated from the CRP are i.i.d draws from a draw from a DP. Therefore, in the case of the CRP, $\mathbf{y}$ in (2.2.41) is a draw from a DP, and $p(\mathbf{y})$ is the DP itself. Thus, the de Finetti mixing distribution of the CRP is a DP. The result we describe in the next section is that the Beta Process is the de Finetti mixing distribution for the India Buffet Process (IBP). First we present the IBP, and then we explore its relationship to the Beta process.

2.2.5 The Indian Buffet Process

As the Gaussian process defines a probability distribution over functions, and the Dirichlet process defines a distribution over distributions, the Indian buffet process, introduced in [6] gives a distribution over infinite sparse matrices. The original motivation for the IBP was as a way to create latent feature models, the idea being that each row of an infinite matrix is an object, and each column is a feature. The ijth entry of the matrix is one if object i has feature j, otherwise it is zero. The reason that the IBP belongs with the other models that we have been discussing in this chapter is that it allows the modeller to remain agnostic about the number of features to be used (just as the DPMM allows agnosticism about the number of components): the matrix has infinitely many columns, so there are an unbounded number of features available, but the model will generally select some small subset of them.

An intuitive way to derive the IBP is as the limit of a finite model. We may define a generative model for finite binary matrices in the following way. First, for each feature, choose the probability that it is active. This probability is constant across objects. In other words, fill the matrix with values between zero and one that are constant within each column. The binary matrix is then generated by, for each cell, flipping a coin whose probability of landing heads is given by the number in the cell. If the coin lands heads, set the cell to one, otherwise set it to zero.

Mathematically, this corresponds to the following model. For each row i and column j, of the matrix A we have:

$$p_j \sim \text{Beta}(\alpha, 1) \tag{2.2.46}$$

$$A_{ij} \sim \text{Bernoulli}(p_j) \tag{2.2.47}$$

Note that the Beta distribution is an appropriate choice for the initial parameters because it is conjugate to the Bernoulli distribution.

After some mathematical manipulation, this model can expressed in a form that makes it possible to take a limit and obtain a corresponding model for infinite matrices. There is one issue: when the limit is taken, the probability assigned to any given infinite matrix is zero. Therefore, it is desirable to define probabilities for equivalence classes of matrices rather than for the matrices themselves. The equivalence relation that Griffiths and Ghahrmani use says that two matrices are equivalent iff they have the same left ordered form, where the left ordered form of a matrix is obtained by assigning to each column a binary number, where the top spot in the column is the largest place. The columns are then ordered according the magnitude of this binary number, with columns with larger numbers being placed farther left.

As its name suggests, the IBP, like the CRP can be understood via a culinary metaphor. This time, the customers are at an Indian Buffet, and their job is to choose dishes (not to seat themselves.) The standard IBP has one parameter, α. The first customer to come to the buffet chooses a Poisson(α) number of dishes. Subsequently, the ith customer samples dish j with probability $\frac{n_j}{i}$, where n_j is the number of customers that have previously chosen dish j. After having chosen whether or not to sample each of the already-tried dishes, he samples a Poisson($\frac{\alpha}{i}$). The matrix is formed by placing a one in cell i, j if customer i choses dish j. Note that because the probability that a customer samples increases with number of customers that have previously sampled the dish, the IBP has the same "rich get richer" property that we saw with the CRP. In order to connect these idea with our previous discussion, we observe that the Indian Buffet process generates individual matrices. However, in line with the earlier discussion, we only look at these matrices modulo the left-ordered equivalence relation.

An application

The IBP has found a number of applications; we will discuss only one here. We present an application to independent components analysis (ICA). We motivate ICA via the well known *cocktail party* problem [8]. For our purposes, this will consist of two people having a conversation that is being recorded by two microphones. Each microphone records a time series of data that we will imagine has been discretized.

The first microphone records the time series $\mathbf{y}_1(t)$ and the second records the time series $\mathbf{y}_2(t)$. We also have two time series corresponding to the two speakers: $\mathbf{x}_1(t)$ represents the sounds made by the first speaker, and $\mathbf{x}_2(t)$ the sounds made by the second. We assume that the microphone data is a linear supposition of the output of the two speakers. This means that we have, for all t

$$\begin{bmatrix} \mathbf{y}_1(t) \\ \mathbf{y}_2(t) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1(t) \\ \mathbf{x}_2(t) \end{bmatrix} + \begin{bmatrix} \epsilon_1(t) \\ \epsilon_2(t) \end{bmatrix}, \tag{2.2.48}$$

where the ϵ values are noise. Written more compactly this is,

$$\mathbf{Y} = A\,\mathbf{X} + E \tag{2.2.49}$$

The goal of ICA is to recover the hidden data $\mathbf{X}$ from the visible (or in this case recorded) data $\mathbf{Y}$. The key assumption is that the hidden sources produce signals that are statistically independent. As long as this assumption holds, quite good recovery is possible.

However, there is a problem with this setup. Suppose that, in the above example, rather than only two speakers, we had a large number of people talking, but that only a few of them are talking at any one time. Further suppose that one does not know in advance the total number of people in the room. The issue here is that we want to view the observed data as a linear combination of different hidden sources at different times, and that we don't want to specify, or even bound, the number of hidden sources ahead of time. The article [12] shows how the IBP lets us build models for this scenario. The setup is:

$$\mathbf{Y} = A(Z \odot \mathbf{X}) + E \tag{2.2.50}$$

$$Z \sim IBP(\alpha), \tag{2.2.51}$$

where $\odot$ is the pointwise product.

The idea here is that we allow the matrix $\mathbf{X}$ to have an infinite number of rows corresponding to an infinite number of sources. The binary matrix $\mathbf{Z}$ then acts as a masking matrix: each row of $\mathbf{Z}$ will have only a finite number of non-zero entries meaning that for each observed variable, only a finite number of the hidden sources will be active: the rest are set to zero by the pointwise product with $\mathbf{Z}$. We will not describe any more of the details of the model beyond mentioning the fact that inference is done using some of the sampling techniques described earlier.

The Indian Buffet Process and the Beta Process

If we think about the Indian buffet process in terms of (2.2.46) and (2.2.47), it is clear that the rows of the matrix are exchangeable. Exchangeability becomes problematic when we consider the restaurant metaphor: the probability with which the dish-tasting procedure generates matrices in is in fact not invariant with respect to row permutations. However, exchangeability is preserved by the fact that we are only interested in this matrix up to left-ordered equivalence. Given exchangeability it is reasonable to ask about the de Finetti mixing distribution for the IBP. We have already said that this distribution is the Beta process, a result first proved in [21]. In this section we try to give some intuition about this fact.

Recall that the CRP gives draws from a draw from a DP, (simply a draw from a DP). Whereas a draw from a draw from a DP was a single point, it is more natural to take a draw from a draw from a Beta process to a be a collection of points. If $\mu = \sum_i p_i \delta_{\omega_i}$ is drawn from a Beta process, a draw from μ will consist of a subset of the points $\{\omega_i\}$, where a particular point is chosen with probability p_i. We can imagine making these choices for each ω_i by flipping a coin with probability of landing heads equal to p_i. We represent this draw as an infinite binary vector $\mathbf{v}$, with $v_i = 1$ if ω is picked, and $v_i = 0$ otherwise. We can make this more formal by defining a *Bernoulli Process* BeP to do the coin flipping for us. Given μ as above we define BeP(μ) as the process that generates a measure $\sum_i z_i \delta_{\omega_i}$, where $z_i \sim$Bernoulli(p_i). it is easy to see that these weights $\{z_i\}$ give the binary vector v.

We can generate an infinite matrix A by generating each of its rows A_i by:

$$\mu \sim \mathrm{BP}(c, B_0) \tag{2.2.52}$$

$$A_i \sim \mathrm{BeP}(\mu) \tag{2.2.53}$$

Comparing with (2.2.46) and (2.2.47) then gives the desired analogy with the Indian buffet process.

Chapter 3

Graphical Models

3.1 Directed Graphical Models

Directed graphical models, also called Bayes nets, are away an informative and visual way to represent the joint distribution of a collection of random variables.

We know that for any collection of random variables $\mathbf{x}_1, \ldots, \mathbf{x}_n$, we can write:

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{m=1}^{n} P(\mathbf{x}_m \mid \mathbf{x}_{m+1}, \ldots, \mathbf{x}_n) \tag{3.1.1}$$

However, this representation does not necessarily respect the relationships that may exist between the variables in a particular problem. In particular, we may have conditional independence relationships that imply that for some m,

$$p(x_m \mid x_{m+1} \ldots, x_n) = p(x_m \mid x_{s(1)} \ldots, x_{s(k)}). \tag{3.1.2}$$

where s is some proper subset of size k of the indices $m+1, \ldots, n$. In other words, the full factorization may introduce dependencies that do not really exist for a given problem. We would like to replace each term in the full factorization with a term that only includes the necessary dependencies. For each $\mathbf{x}_m$, denote by $\text{pa}(m)$ the smallest set of indices such that that we may decompose the joint distribution as follows:

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{m=1}^{n} P(\mathbf{x}_m \mid \text{pa}(m)) \tag{3.1.3}$$

This representation is useful both because it eliminates clutter and because it exposes the dependencies that exist in our data.

The expression (3.1.3) suggests a way to represent our joint distribution in a graphical way: we make a graph with node for each variable, and we draw a directed edge from the node $\mathbf{x}_k$ to $\mathbf{x}_m$ iff $\mathbf{x}_k \in$pa(m). An example of a directed graphical model with four nodes is shown in 3.1. Applying (3.1.3) to this model shows that the joint distribution factors into $p(\mathbf{x}_4 \mid \mathbf{x}_3)p(\mathbf{x}_3 \mid \mathbf{x}_2, \mathbf{x}_1)p(\mathbf{x}_2)p(\mathbf{x}_1)$.

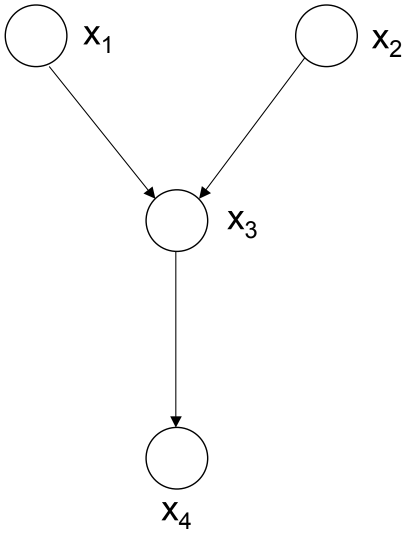

Figure 3.1: A directed graphical model with four variables. The structure of the graph tells us that the joint distribution factors as $p(\mathbf{x}_4 \mid \mathbf{x}_3)p(\mathbf{x}_3 \mid \mathbf{x}_2, \mathbf{x}_1)p(\mathbf{x}_2)p(\mathbf{x}_1)$.

3.2 Undirected Graphical Models

The directed connections in a Bayes net are often interpreted as representing a causal relationships in which a parent node influences its children. It is easy to find problems in which no such causal relationships should be assumed to exist. This is one of the motivations for *undirected graphical models*.

We introduce undirected graphical models via an example, borrowed from [22]. The data we are given are the noisy pixel values $\mathbf{y}_i$ of an image $\mathbf{Y}$, and we want to infer the corresponding to noise-free pixel values $\mathbf{x}_i$ of the clean image $\mathbf{X}$. We expect the $\mathbf{x}_i$ to depend probabilistically on the $\mathbf{y}_i$, where the probability is determined by the process by which the noise was introduced. Specifically, we assume that $\mathbf{x}_i$ and $\mathbf{y}_i$ should not differ very greatly. Given the regularities present in natural images, it is also reasonable to assume the each $\mathbf{x}_i$ is similar to its spatial neighbors: as humans, we are able to pick out the corrupted pixels in a noisy image in part by finding the pixels that stand out from their neighbors. We represent these assumed dependencies graphically, as shown in Figure 3.2. This figure shows a 4x4 image patch. The hidden clean pixels (open circles) interact with their spatial neighbors and the their corresponding noisy pixels (filled circles).

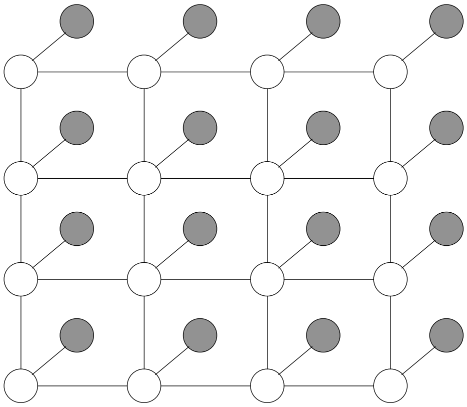

Figure 3.2: An undirected model representing the dependencies assumed to exist among observed noisy pixel values (filled circles) and hidden clean pixel values (open circles).

We can define the joint probability of a pair $\mathbf{X}$ and $\mathbf{Y}$ in terms of well the pair satisfies the criteria given above i.e. that $\mathbf{x}_i$ is close to $\mathbf{y}_i$ and $\mathbf{x}_i$ is close in to its neighbors. To do this, we introduce potential functions ψ and ϕ that measure how well the criteria are met. One plausible choice is:

$$\psi_{ij}(\mathbf{x}_i, \mathbf{x}_j) = e^{\frac{-1}{2\sigma_\psi^2}(\mathbf{x}_i - \mathbf{x}_j)^2}, \tag{3.2.1}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are neighbors.

$$\psi_{ij}(\mathbf{y}_i, \mathbf{x}_i) = e^{\frac{-1}{2\sigma_\phi^2}(\mathbf{y}_i - \mathbf{x}_i)^2} \tag{3.2.2}$$

Note that the variances are allowed to be different. The joint distribution is then:

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \prod_{\text{neighbors } ij} \psi_{ij}(\mathbf{x}_i, \mathbf{y}_j) \prod_i \phi(\mathbf{y}_i, \mathbf{x}_i) \tag{3.2.3}$$

Z is a normalizing constant.

Equation (3.2.3) points to the similarity in motivation between directed and undirected graphical models: in both cases we decompose the distribution of a large number of random variables into a product of dependencies that only exist locally on the graph.

3.3 Factor Graphs

We now introduce a third kind of graphical model, called a factor graph. We showed that both directed and undirected graphical models can be seen as representing the joint distribution of a collection of variables as the product of functions of subsets of the variables. We can write this decomposition in a general way as:

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{k=1}^{K} f_k(\mathbf{X}_k), \tag{3.3.1}$$

where each $\mathbf{X}_k$ is a subset of $\mathbf{X}$. Factor graphs are yet another way of representing this decomposition graphically. A factor graph is a bipartite graph. One set of vertices are the variables $\mathbf{X}$, and the other set of nodes contains K *factors*. The kth factor is connected by undirected edges to each of nodes in the vector $\mathbf{X}_k$ in (3.3.1). An illustrative example comes from coding theory.

Coding Theory

Coding theory is concerned with the problem of transmitting information over a noisy channel. The idea is that we want to send a collection of bits, some subset of which will change value during transmission due to noise present in the transmission process. In order to correct for this corruption, we augment our original message with some number of extra bits. A naive use of these extra bits would be to simply send several copies of our original message, hoping to use this redundancy to average out the effects of noise. Not surprisingly, there are much better ways to use the extra bits. One of these that has particularly nice properties is a system called *low density parity check coding.*

As explained, we send our original message of n bits, and in addition send k extra bits, making our codewords be of total length $n + k = m$. In low density parity check codes is to choose a collection of small subsets f_i of the m bits in the codeword, and require these subsets to contain an even number of ones (which is equivalent to requiring them to sum to zero modulo 2.) The idea is that if we have the n bits of data that we want to transmit, we choose the remaining k bits in such a way that these parity check constraints are met. (Obviously certain consistency requirements must be met for this to be possible.) The receiver can detect noise by seeing whether or not the parity check conditions are met. Further, the receiver can denoise a message by modifying to be more consistent with the parity requirements.

We can represent low density parity check codes as factor graphs, as shown in 3.3. The variables are the bits of the message and the factors are the parity checks: a group of variables that participates in a factor has joint probability one if it satisfies the parity parity requirement and probability zero otherwise. For example, in 3.3, $\mathbf{x}_1, \mathbf{x}_4, \mathbf{x}_5$ participate in the factor f_1, meaning that their values must sum to zero mod 2. In the notation of (3.3.1), we have:

$$f_k(\mathbf{X}_k) = \begin{cases} 1 & \sum_i \mathbf{X}_k(i) \equiv 0 \bmod 2 \\ 0 & \text{else} \end{cases} \tag{3.3.2}$$

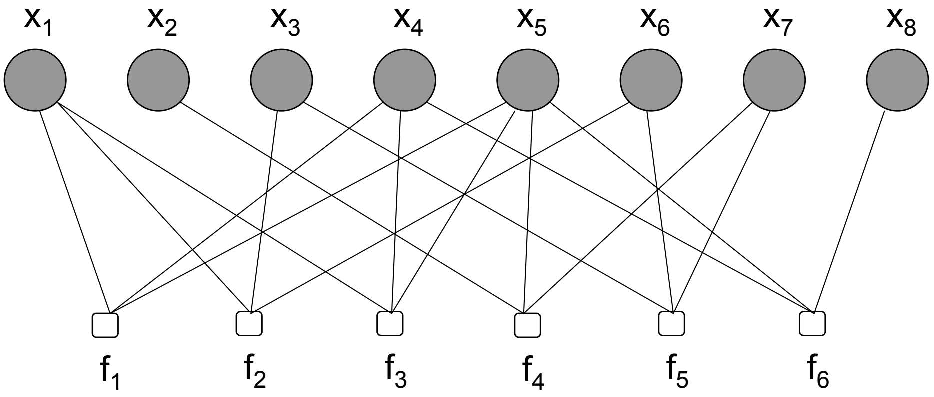

Figure 3.3: In this figure, the eight circular nodes are the transmitted bits, and the six square nodes are the parity checks in which they participate. For example, f_1 says that there must be an even number of ones among $\mathbf{x}_1$, $\mathbf{x}_4$ and $\mathbf{x}_5$.

Error correction can be done in using an approach similar to the one we used in the image denoising example: inferred sent bits are chosen to compatible with one another as determined by the factors, and with the noisy received bits as determined by the channel's bit-flipping probability.

3.4 Belief Propagation

In the image denoising and coding examples, we want to find configurations of the hidden and visible variables that have high probability. The process of finding these configurations is what we call inference. We will now present a very general and elegant algorithm for doing inference in undirected graphical models and factor graphs. This algorithm goes by a variety of names: belief propagation, message passing and the sum product algorithm. We will call it belief propagation. We begin by describing the belief propagation for the Markov random field example given in Equation (3.2.3). We will then define it for factor graphs and give an example involving parameter inference in hidden Markov models.

Consider the hidden node $\mathbf{y}_i$. As we discussed, the state of this node is influenced both by the corresponding visible node, and by the neighboring hidden nodes. The states of these neighboring hidden nodes will in turn be influenced by the visible nodes to which they correspond and by their neighbors. We therefore approach the inference problem by imagining "messages" being passed between nodes. A message from node j to node i encodes node j's "belief" about the state that node i should be in.

A message m_{ji} from node j to node i is a vector whose length is equal to the number of states that it is possible for node j to take. The following notation is standard in the literature but its somewhat confusing: $m_{ji}(\mathbf{x}_i)$ is proportional to how likely node j thinks it is that node i has value $\mathbf{x}_i$. Of course this is not correct: m_{ji} is a vector, but $\mathbf{x}_i$ is not an index. However, what this notation lacks in intuitive clarity, it makes up for in compactness, which is a virtue that will be valuable later.

After node i has received all its incoming messages, it can form a "belief" about its own state. This belief will again be a vector b_i of length equal to the number of values that node i can be in. Using the notation introduced in the previous paragraph, $b_i(\mathbf{x}_i)$ is the current estimate that node i has value $\mathbf{x}_i$. In accordance with the intuition that node i's state should be influenced by its neighbors and its corresponding visible node, we define:

$$b_i(\mathbf{x}_i) = \frac{1}{Z}\phi_i(\mathbf{y}_i, \mathbf{x}_i) \prod_{j \in \mathcal{N}(i)} m_{ji}(\mathbf{x}_i) \tag{3.4.1}$$

Here $\mathcal{N}(i)$ denotes the set of hidden nodes that neighbor i, and Z is a normalizing constant. To make the interpretation absolutely explicit: $\phi_i(\mathbf{y}_i, \mathbf{x}_i)$ is visible node i's estimate of how likely i is to be in state x_i, and $m_{ji}(\mathbf{x}_i)$ are the corresponding estimates from the neighboring hidden nodes.

How are the messages m_{ji} determined? The messages from node j is defined in terms of the messages coming into node j:

$$m_{ji}(\mathbf{x}_i) = \sum_{\mathbf{x}_j} \psi_{ji}(\mathbf{x}_j, \mathbf{x}_i)\phi_j(\mathbf{y}_j, \mathbf{x}_j) \prod_{k \in \mathcal{N}(j), k \neq i} m_{kj}(\mathbf{x}_j) \tag{3.4.2}$$

Observe that if $\mathbf{x}_i$ was not excluded from the product, we would have

$$m_{ji}(\mathbf{x}_i) = \sum_{\mathbf{x}_j} \psi_{ji}(\mathbf{x}_j, \mathbf{x}_i) b_j(\mathbf{x}_j). \tag{3.4.3}$$

Abusing notation, we can interpret this as

$$m_{ji}(\mathbf{x}_i) = \mathbb{E}_{b_j}[\psi_{ji}(\mathbf{x}_j, \mathbf{x}_i)]. \tag{3.4.4}$$

In other words, node j wants node i to be in state $\mathbf{x}_i$ to the extent that node j expects $\mathbf{x}_i$ to be compatible with its own value as estimated by b_j. Of course, this whole discussion is hypothetical, since $\mathbf{x}_i$ was *is* must excluded from the product to avoid circularity, and (3.4.3) doesn't actually hold. Nevertheless, if we can view (3.4.2) as the closest approximation possible to (3.4.3) subject to the avoidance of circularity, we can hold on to the intuition behind the interpretation given above.

We have given a recursive definition of the messages. How is this recursion initialized? In the case in which the graph is a tree there will be leaf nodes that have no incoming messages, and the recursion may be initialized at these nodes. In this tree case, belief propagation can be shown to perform exact inference. When the graph is not a tree, the nodes may be initialized heuristically. Although inference is no longer provably exact when the graph is not a tree, it is often acceptable in practice. Belief propagation on graphs that are not trees is called "loopy belief propagation," by virtue of the fact that graphs that are not trees contain cycles, or loops.

3.4.1 Belief Propagation in Factor Graphs

In factor graphs, we need two types of messages. One type is passed from variable nodes to factor nodes, and the other type is passed from factor nodes to variable nodes. (Because a factor graph is bipartite, no messages will be passed between nodes of the same type.) We will us the following notation: $m_{f_j \mathbf{x}_i}$ is a message from the jth factor to the ith variable, and $m_{\mathbf{x}_i f_j}$ is a message from the ith variable to the jth factor.

First we note that, as was the case with undirected graphical models, a variable node $\mathbf{x}_i$'s belief about its state is determined by the messages it receives. We assume there are no visible nodes, so that, unlike in the denoising example for unidrected graphical models, evidence from these variables does not enter:

$$b_i(\mathbf{x}_i) = \prod_{j \in \mathcal{N}(i)} m_{f_j\,\mathbf{x}_i}(\mathbf{x}_i) \tag{3.4.5}$$

Here, $\mathcal{N}$ is the set of factors that neighbor node i.

One key point about factor graphs is that the state of a factor node is directly determined by the states of its neighboring variable nodes. This means that the most information that a variable node can pass about the state of neighboring factor node is its belief about its own state. Therefore, we define

$$m_{\mathbf{x}_i\,f_j} = \prod_{k \in \mathcal{N}(i), k \neq j} m_{f_j\,\mathbf{x}_i}, \tag{3.4.6}$$

where the product is taken coordinate-wise. As with our our interpretation of the messages passed in undirected graphical models, we can interpret $\prod_{k \in \mathcal{N}(i), k \neq} m_{ji}$ as the best noncircular approximation to b_i.

Now we define the messages from passed from factor nodes to variable nodes. Let f_j be a factor node sending a message to variable node $\mathbf{x}_i$. Suppose that variable nodes $\mathbf{x}_1, \ldots, \mathbf{x}_m$ also participate in (i.e. neighbor) f_j. Then the message from f_j to x_i is defined as

$$m_{f_j\,\mathbf{x}_i}(\mathbf{x}_i) = \sum_{\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}} f(\mathbf{x}_i, \mathbf{x}_1, \ldots, \mathbf{x}_M) \prod_{n=1}^{M} m_{\mathbf{x}_n\,f_j} \tag{3.4.7}$$

In an undirected graphical model, outgoing messages assigned high probabilities to states that were compatible with states of the sending node that had high likelihood. The main difference here is that the outgoing message now says that a state of the target node will be likely if it is expected to be compatible with the states of the other variables in the factor. Here, expectation is determined by these other variables' own estimates of their state. Heuristically, these variables estimate their states and request that $\mathbf{x}_i$ be compatible with these estimates, where compatibility is defined by f_j.

As with belief propagation in undirected graphical models, belief propogation for factor graphs can either be done in trees or in graphs with loops. In the tree case, we designate one node as the root, and then initialize the recursion at the leaves. If the leaf is a factor node, the message that it passes out to a variable $\mathbf{x}$ is $f(\mathbf{x})$, which makes sense because the product in (3.4.7) will empty at initialization, and, as a leaf, the factor node will neighbor only one variable node. If the leaf is a variable node, the message it passes to factors are constant vectors of ones.

3.5 Hidden Markov Models

We now proceed to an extended example, which involves the application of several inference techniques we have discussed (including belief propagtion in factor graphs) to parameter estimation in hidden Markov models. Our treatment of this analysis follows [5] and [3].

For our purposes, a hidden Markov model (HMM) will be constructed as follows.

- There is a Markov chain of hidden variables $\mathbf{X} = (\mathbf{x}_t)_{t=1}^T$ that we will assume that hidden variables take values in a discrete set of size N. Transitions in the hidden Markov chain are governed by the transition matrix A.

- The Markov chain is initialized with a distribution π from which the state of $\mathbf{x}_1$ is drawn.

- At each time t, the hidden variable $\mathbf{x}_t$ "emits" a visible variable $\mathbf{y}_t$. We will assume that the visible variables are also discrete over a common finite alphabet (which may be distinct from the hidden one), meaning that we may describe the emission probabilities with a matrix E. The sequence of visible variables is $\mathbf{Y} = (\mathbf{Y}_t)_{t=1}^T$.

This setup is shown as a directed graphical model in 3.4.

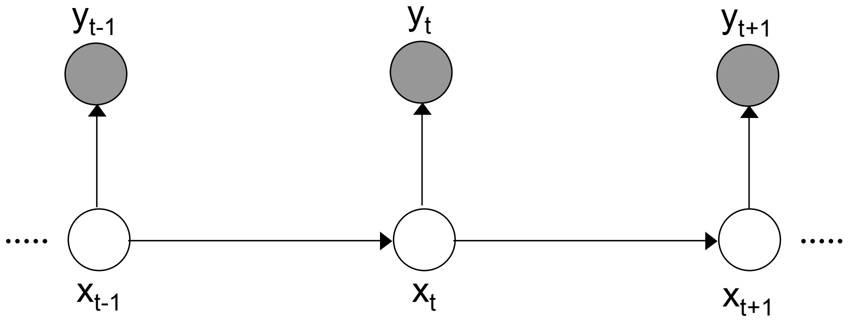

Figure 3.4: In an HMM, hidden variables $\mathbf{x}_t$ transition as a Markov chain, and at each step emit a visible variable $\mathbf{y}_t$. The Markov property means that each hidden node only interacts with its predecessor.

3.5.1 The EM Algorithm for Parameter Estimation in HMMs

HMMs present a number of problems. The one we will examine here is the task of inferring the model parameters A, E and π, collectively denoted θ from the visible data $\mathbf{Y}$. The presence of hidden variables makes the application of the EM algorithm intuitively appealing. This idea is reinforced by the fact that the likelihood in the presence of the hidden variables is easy to compute:

$$\log P(\mathbf{X}, \mathbf{Y} \mid \theta) = \log P(\mathbf{x}_1) + \sum_{t=1}^{T} \log P(\mathbf{y}_t \mid \mathbf{x}_t) + \sum_{t=2}^{T} \log P(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \tag{3.5.1}$$

We will use the variational interpretation of the EM algorithm given above. In the language of HMMs, the two steps of the EM algorithm are:

$$\text{E Step: } Q^{(k+1)} = P(\mathbf{X} \mid \mathbf{Y}, \theta^{(k)}) \tag{3.5.2}$$

$$\text{M Step: } \theta^{(k+1)} = \text{argmax}_\theta \mathbb{E}_{Q_{k+1}}[\log P(\mathbf{X}, \mathbf{Y} \mid \theta)] \tag{3.5.3}$$

In order to see how to perform these steps, we will rewrite (3.5.1) in a more useful way. As specified before, we will limit ourselves to the case in which both the hidden and observed variables are discrete. This allows us to write, for each t, $\mathbf{y}_t$ and $\mathbf{x}_t$ as column vectors whose length is equal to the number of possible states of the variables. This vector has a 1 in the spot corresponding to the value that the variable takes and is zero elsewhere. With this notation, (3.5.1) is

$$\log P(\mathbf{X}, \mathbf{Y} \mid \theta) = \mathbf{x}_1^\top \log\pi + \sum_{t=1}^{T} \mathbf{y}_t^\top (\log E)\, \mathbf{x}_t + \sum_{t=2}^{T} \mathbf{x}_t^\top (\log A)\, \mathbf{x}_{t-1} \tag{3.5.4}$$

Using the linearity of expectation, and the fact each parameter appears in only one term, we can write the maximization in the E-step as finding

$$\text{argmax}_\theta \mathbb{E}_{Q^{(k+1)}}[\log P(\mathbf{X}, \mathbf{Y} \mid \theta)] = \text{argmax}_\pi \mathbb{E}_{Q^{(k+1)}}[\mathbf{x}_1^\top \log\pi] \tag{3.5.5}$$

$$+\text{argmax}_E \mathbb{E}_{Q^{(k+1)}}[\sum_{t=1}^{T} \mathbf{y}_t^\top (\log E)\, \mathbf{x}_t] \tag{3.5.6}$$

$$+\text{argmax}_A \mathbb{E}_{Q^{(k+1)}}[\sum_{t=2}^{T} \mathbf{x}_t^\top (\log A)\, \mathbf{x}_{t-1}] \tag{3.5.7}$$

Because the quantities with respect to which we are trying to maximize define probabilities, the maximization must be done subject to the appropriate sum-to-one constraints. It is therefore necessary to use lagrange multipliers. The results are as follows:

$$\pi^{(k+1)} = \mathbb{E}_{Q_{k+1}}[\mathbf{x}_1] \tag{3.5.8}$$

$$A^{(k+1)} = \frac{1}{Z_1} \sum_{t=2}^{T} \mathbb{E}_{Q_{k+1}}[\mathbf{x}_t\, \mathbf{x}_{t-1}^\top] \tag{3.5.9}$$

Here, $\mathbb{E}_{Q_{k+1}}[\mathbf{x}_t\, \mathbf{x}_{t-1}^\top]$is the matrix whose i, jth cell contains the probability that the X was in state i at time $t-1$ and state j at time t.

$$E^{(k+1)} = \frac{1}{Z_2} \sum_{t=2}^{T} \mathbb{E}_{Q_{k+1}}[\mathbf{y}_t\, \mathbf{x}_t^\top] = \frac{1}{Z_2} \sum_{t=2}^{T} \mathbf{y}_t\, \mathbb{E}_{Q_{k+1}}[\mathbf{x}_t^\top] \tag{3.5.10}$$

Z_1 and Z_2 are normalizing constants. Expectations are taken component-wise.

How do we compute $\mathbb{E}_{Q_{k+1}}[\mathbf{x}_t]$? The key is to observe that $\mathbf{x}_t$ is multinomial random variable. So, by definition of the expectation of a multinomial random variable, we see that $\mathbb{E}_{Q_{k+1}}[\mathbf{x}_t]$ (as a vector) is simply equal to the vector representing the marginal distribution of $\mathbf{x}_t$ under $Q^{(k+1)}$. The analogous conclusion applies to $\mathbb{E}_{Q_{k+1}}[\mathbf{x}_t\, \mathbf{x}_{t-1}^\top]$. So in order to implement the EM algorithm for HMMs, we need to compute two pieces of information.

- To compute A in (3.5.9), we need for each i, j and $t > 1$ the joint probability $P(\mathbf{x}_{t-1}(i) = 1, \&\, \mathbf{x}_t(j) = 1)$.
- To compute π in (3.5.8) and E in (3.5.10), for i and each t, we need the probability that $X_t(i) = 1$. (Note that since we know $\mathbf{Y}$, this is the only information that we need to compute E.)

Because we are interested in expectations with respect to $Q_{k+1}(\mathbf{X}) = P(\mathbf{X} \mid \mathbf{Y}, \theta_k)$, the above probabilities are to be computed conditioned on the observed variables and the model parameters. So for our calculations below, we will assume that these quantities are known.

3.5.2 The Forward-Backward Algorithm

It turns out that the right way to approach the calculations we need is a form of belief propagation. However, it has a classical presentation that we give first before giving the belief propagation interpretation. We will show in detail how to do the second calculation, finding the distribution of $\mathbf{x}_t$ for any t. The basic approach for finding the transition probabilities is the same, so we won't present it in detail.

The first step is to write:

$$P(\mathbf{x}_t \mid \mathbf{Y}) = \frac{P(\mathbf{Y} \mid \mathbf{x}_t)P(\mathbf{x}_t)}{P(\mathbf{Y})} \tag{3.5.11}$$

Looking at the graphical model for the HMM, we see that paths between visible units before and after time t are blocked by the hidden $\mathbf{x}_t$ node. This gives the following conditional independence relation:

$$P(\mathbf{Y} \mid \mathbf{x}_t) = P(\mathbf{y}_1, \ldots, \mathbf{y}_t \mid \mathbf{x}_t)P(\mathbf{y}_{t+1}, \ldots, \mathbf{y}_T \mid \mathbf{x}_t) \tag{3.5.12}$$

Plugging this into (3.5.11) gives

$$P(\mathbf{x}_t \mid \mathbf{Y}) = \frac{P(\mathbf{y}_1, \ldots, \mathbf{y}_t \mid \mathbf{x}_t)P(\mathbf{y}_{t+1}, \ldots, \mathbf{y}_T \mid \mathbf{x}_t)P(\mathbf{x}_t)}{P(\mathbf{Y})} \tag{3.5.13}$$

Applying the definition of conditional probability to the first factor in the numerator gives:

$$P(\mathbf{x}_t \mid \mathbf{Y}) = \frac{[P(\mathbf{y}_1, \ldots, \mathbf{y}_t, \mathbf{x}_t)/P(\mathbf{x}_t)]P(\mathbf{y}_{t+1}, \ldots, \mathbf{y}_T \mid \mathbf{x}_t)P(\mathbf{x}_t)}{P(\mathbf{Y})} \tag{3.5.14}$$

Then cancel $P(\mathbf{x}_t)$:

$$P(\mathbf{x}_t \mid \mathbf{Y}) = \frac{P(\mathbf{y}_1, \ldots, \mathbf{y}_t, \mathbf{x}_t)P(\mathbf{y}_{t+1}, \ldots, \mathbf{y}_T \mid \mathbf{x}_t)}{P(\mathbf{Y})} \tag{3.5.15}$$

$$:= \frac{\alpha_t(\mathbf{x}_t)\beta_t(\mathbf{x}_t)}{P(\mathbf{Y})}, \tag{3.5.16}$$

where we have defined:

$$\alpha(\mathbf{x}_t) := P(\mathbf{y}_1, \ldots, \mathbf{y}_t, \mathbf{x}_t) \tag{3.5.17}$$

$$\beta(\mathbf{x}_t) := P(\mathbf{y}_{t+1}, \ldots, \mathbf{y}_T \mid \mathbf{x}_t) \tag{3.5.18}$$

This shows how to perform the second calculation needed for the application the EM algorithm. The first can be done with similar methods.

The key to the effectiveness of the forward-backward algorithm are recurrence relations for α and β that allow them to be computed efficiently. To obtain $\alpha(\mathbf{x}_t)$ from $\alpha(\mathbf{x}_{t-1})$, we first have to include the next hidden variable to get $P(\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \mathbf{x}_t)$.

$$P(\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \mathbf{x}_t) = \sum_{\mathbf{x}_{t-1}} P(\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \mathbf{x}_{t-1})P(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \tag{3.5.19}$$

$$\Rightarrow P(\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \mathbf{x}_t) = \sum_{\mathbf{x}_{t-1}} \alpha(\mathbf{x}_{t-1}) P(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \tag{3.5.20}$$

We then include the next visible state by simple multiplication to get the recursion:

$$\alpha(\mathbf{x}_t) := P(\mathbf{y}_t \mid \mathbf{x}_t) \sum_{\mathbf{x}_{t-1}} \alpha(\mathbf{x}_{t-1}) P(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \tag{3.5.21}$$

The analogous recursion for β is

$$\beta(\mathbf{x}_t) = \sum_{\mathbf{x}_{t+1}} \beta(\mathbf{x}_{t+1}) P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) P(\mathbf{y}_{t+1} \mid \mathbf{x}_{t+1}) \tag{3.5.22}$$

Since we are assuming that the model parameters are known, we know every term in the above expressions aside from $\beta(\mathbf{x}_{t+1})$ and $\alpha(\mathbf{x}_{t-1})$, and these can be computed recursively, after specifying values for α_1 and β_T. We initialize the recursion with:

$$\alpha(\mathbf{x}_1) = P(\mathbf{y}_1) P(\mathbf{y}_1 \mid \mathbf{x}_1) \tag{3.5.23}$$

The intuition here is that, by (3.5.19), $\alpha(\mathbf{x}_1) = P(\mathbf{y}_1, \mathbf{x}_1)$. The definition conditional probability gives (3.5.23). The initialization for β is

$$\beta(\mathbf{x}_t) = 1 \tag{3.5.24}$$

3.5.3 The Forward-Backward Algorithm as Belief Propagation

Following [3], we will use the forward-backward algorithm as an example of belief propogation in factor graphs, and show that the α and β quantities we defined above are equal to the messages.

So the first step is to reformulate the HMM graphical model in figure 3.4 as a factor graph. Because of the restricted nature of interactions in the HMM, this graph will have a particularly simple form. Each hidden node only interacts with the preceding hidden node, as dictated by the Markov property, and each visible node only interacts with its corresponding hidden node. We do, however, have to specify the treatment of the visible variables in the HMM. First, the values of these nodes are constant, so we don't do inference on them. Second, the messages that they pass into a factor will always be constant, and may incorporate these constant terms into the factor itself, thereby eliminating altogether all inputs to factors from visible nodes. Altogether then, it is safe to treat the visible nodes entirely implicitely. We obtain the factor graph shown in 3.5, with factors given by

$$f_t(\mathbf{x}_t, \mathbf{x}_{t-1}) = P(\mathbf{x}_t \mid \mathbf{x}_{t-1}) P(\mathbf{y}_t \mid \mathbf{x}_t) \tag{3.5.25}$$

It might be disturbing that $\mathbf{x}_{t-1}$ appears in the same factor as $\mathbf{y}_t$ given that these two variables do not interact. But we see that the way the factor is defined means that no such interaction is implied.

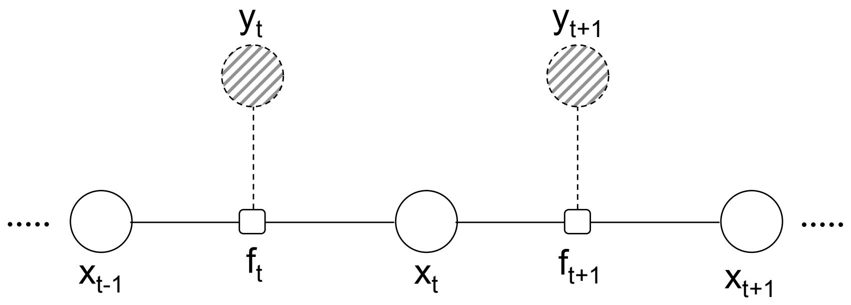

Figure 3.5: In this reformulation of an HMM as a factor graph, we take advantage of the fact that the visible variables do not play a meaningful part in the message passing, which lets include them only implicitly, as indicated by the dashed lines.

Having constructed a factor graph, we compute the massages using (3.4.6) and (3.4.7). First we look at the messages that are passed forward from factors to variables. Using (3.4.7), we have:

$$m_{f_t,\mathbf{x_t}} = \sum_{x_{t-1}} f_t(\mathbf{x}_t, \mathbf{x}_{t-1}) m_{\mathbf{x}_{t-1}\,f_t} \tag{3.5.26}$$

We can now use (3.4.6) to expand $m_{\mathbf{x}_{t-1}\,f_t}$ as $m_{f_{t-1}\,\mathbf{x}_{t-1}}$. Substituting this in gives:

$$m_{f_t,\mathbf{x_t}} = \sum_{\mathbf{x}_{t-1}} f_t(\mathbf{x}_t, \mathbf{x}_{t-1}) m_{f_{t-1}\,\mathbf{x}_{t-1}} \tag{3.5.27}$$

This gives a recursive definition for the tth message. If we write $\alpha(\mathbf{x}_t) = m_{f_t,\mathbf{x_t}}$, and subsitute in the definition from (3.5.25) we have:

$$\alpha(\mathbf{x}_t) = \sum_{\mathbf{x}_{t-1}} P(\mathbf{x}_t \mid \mathbf{x}_{t-1}) P(\mathbf{y}_t \mid \mathbf{x}_t) \alpha(\mathbf{x}_{t-1}) = P(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \sum_{\mathbf{x}_{t-1}} \alpha(\mathbf{x}_{t-1}) P(\mathbf{y}_t \mid \mathbf{x}_t) \tag{3.5.28}$$

But this is exactly the recursion (3.5.21). Therefore, to show that the α's are equal to the forward messages, it suffices to check that the initializations are the same.

Recalling the initial conditions we have given for general factor graphs, we see that the initialization is $f(x_1)$, which according to (3.5.25) is $P(\mathbf{x}_1)P(\mathbf{x}_1 \mid \mathbf{y}_1)$. But comparing to (3.5.23), we see that we have equality. This shows that the messages passed forward from factors to variables are equal to the α's. We now check that the messages passed backward are equal to the β's.

According to (3.4.6), we have:

$$m_{f_{t+1},\mathbf{x}_t} = \sum_{\mathbf{x}_{t+1}} f(\mathbf{x}_{t+1}, \mathbf{x}_t) m_{\mathbf{x}_{t+1}\, f_{t+1}} \tag{3.5.29}$$

By (3.4.6) this is

$$\sum_{\mathbf{x}_{t+1}} f(\mathbf{x}_{t+1}, \mathbf{x}_t) m_{f_{t+2}\, \mathbf{x}_{t+1}} \tag{3.5.30}$$

We can write $\beta(\mathbf{x}_t) = m_{f_{t+1},x_t}$ and expand f using (3.5.25). This gives:

$$\beta_t = \sum_{\mathbf{x}_{t+1}} P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) P(\mathbf{y}_{t+1} \mid \mathbf{x}_{t+1}) \beta(\mathbf{x}_{t+1}) = \sum_{\mathbf{x}_{t+1}} \beta(\mathbf{x}_{t+1}) P(\mathbf{x}_{t+1} \mid \mathbf{x}_t) P(\mathbf{y}_{t+1} \mid \mathbf{x}_{t+1}) \tag{3.5.31}$$

As before, we see that this is the same recursion that defines the sequence of βs. And also as before, it remains only to check that the initialization is the same. By (3.5.24) we see that the initialization is 1, which is the same initialization we gave for factor graphs when the leaf node is a variable.

Chapter 4

Reproducing Kernel Hilbert Spaces

Many machine learning algorithms are most naturally formulated in linear terms, but it is often desirable to be able to handle nonlinear data. In order to preserve the structure of the original algorithms, we map the original input space $\mathcal{X}$ to another space $\mathcal{H}$ via the non-linear map $\Phi : \mathcal{X} \to \mathcal{H}$. If we have chosen Φ and $\mathcal{H}$ correctly, we may then apply our linear algorithms in $\mathcal{H}$.

This process is made much easier by the fact that many linear algorithms are linear only because they involve terms of the form $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{X}}$ for $\mathbf{x}, \mathbf{y} \in \mathcal{X}$, where $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{X}}$ denotes the inner product in $\mathcal{X}$. After applying the map Φ, these terms become $\langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle_{\mathcal{H}}$. Therefore, if we can find a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that

$$k(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle_{\mathcal{H}}, \tag{4.0.1}$$

then we can avoid actually calculating and applying Φ. The function k is called a *kernel*.

In practice, one never actually considers Φ and $\mathcal{H}$. Rather, one starts by constructing the kernel k, and Φ and $\mathcal{H}$ are implicit. In order for this to work, we need to know which functions k are admissable kernels, i.e. we need to know for which kernels k there exits Φ, $\mathcal{H}$ such that (4.0.1) holds. This information is furnished by the following theorem, which we will not prove; a proof can be found in [7].

Definition 4.0.1. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *positive kernel* if for any finite set of points $\mathbf{x}_1 \dots x_n \in \mathcal{X}$, the matrix K defined by $K_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$ is positive definite.

Theorem 4.0.2. *Suppose $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive kernel, and define the map $\Phi_k : \mathcal{X} \to \mathbb{R}^{\mathcal{X}}$ by $\Phi_k(\mathbf{x}) = k(\mathbf{x}, \cdot)$. Then define a space $\mathcal{H}_k$ by closing the image of Φ_k*

under sums of the form $\sum_{i=1}^{n} a_i k(\mathbf{x}_i, \cdot)$*, and equipping it with the inner product:*

$$\langle f, g\rangle_{\mathcal{H}_k} = \langle \sum_{i=1}^{n} a_i k(\mathbf{x}_i, \cdot), \sum_{j=1}^{m} b_i k(\mathbf{y}_j, \cdot)\rangle_{\mathcal{H}_k} = \sum_{i=1}^{n}\sum_{j=1}^{m} a_i b_j k(\mathbf{x}_i, \mathbf{y}_j) \tag{4.0.2}$$

Then $\mathcal{H}_k$ *is a Hilbert space and we have*

$$k(\mathbf{x}, \mathbf{y}) = \langle \Phi_k(\mathbf{x}), \Phi_k(\mathbf{y})\rangle_{\mathcal{H}_k}, \tag{4.0.3}$$

which shows that k *is an admissable kernel.*

As an immediate consequence of the construction of $\mathcal{H}_k$, we see that k has the following *reproducing property*:

$$\langle k(\mathbf{x}, \cdot), k(\mathbf{y}, \cdot)\rangle_{\mathcal{H}} = k(\mathbf{x}, \mathbf{y}) \tag{4.0.4}$$

and

$$\langle f(\cdot), k(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} = f(\mathbf{x}) \tag{4.0.5}$$

The second equality comes from

$$f(\cdot) = \sum_{i=1}^{n} k(\mathbf{y}_i, \cdot) \Rightarrow \langle f(\cdot), k(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} = \sum_{i=1}^{n} k(\mathbf{y}_i, \mathbf{x}) = f(\mathbf{x}) \tag{4.0.6}$$

We call $\mathcal{H}_k$ the *reproducing kernel Hilbert space* (RKHS) associated with the kernel k.

4.1 Regulariziation

Generally, when one is training a supervised learning algorithm, one wants to keep the error on the training set small. However, unless corrective steps are taken, the model with minimal error on the training data will often overfit the data. Overfitting occurs when a model is too finely tuned to the peculiarities of a particular training set and fails to generalize well to data on which it has not been trained. Although much of what follows is true for both classification and regression algorithms, we will restrict our discussion to classification.

One sign that a classification algorithm has overfit the data is that its decision function is highly non-smooth: this indicates that the function is "contorted" in a way that lets it fit the training set perfectly but that may impair generalization. With this in mind, we may seek to prevent overfitting by requiring that our decision function be smooth. This is called *regularization.*

Mathematically, a regularized learning algorithm will have the following form: Find the decision function f that minimizes:

$$c(f, \{\mathbf{x}_i, \mathbf{y}_i\})) + L(f), \tag{4.1.1}$$

where c is a function that measures error on the training set and L is a function that measures the "roughness" of f: $L(f)$ is large if f is highly non-smooth. In this section, we explore the nature of the function L. In particular, we show that we we may choose $L(f)$ to be the norm of f in certain function space. The remarkable fact is that this function space arises naturally from the learning problem: it is the reproducing kernel Hilbert space we get when we use a kernelized learning algorithm.

In section 4.2, we recall some facts about kernelized algorithms and reproducing kernel Hilbert spaces. In sections 4.3 and 4.4, we show from two points of view that minimizing the RKHS norm of a function f amounts to regularizing f. In section 4.5, we show that support vector machines are an example of a regularized learning algorithm. Section 4.6 contains the proof of the representer theorem, a result that characterizes the solution f to a regularized minimization problem of the form (4.1.1). Spefically, the representer theorem shows that if f minimizes (4.1.1), then f may be represented as a finite linear combination of kernels associated with training points. Section 4.7 concludes.

4.2 Regularization via Penalization of Large Derivatives

For any differentiable function, a large derivative indicates rapid changes in value. Thus, a natural way to encourage smoothness is to penalize large derivatives. In other words, if L is a differential operator, then we want to minimize $||Lf||_{L^2}$. In this section, we show that minimizing $||f||_{\mathcal{H}_k}$ accomplishes this.

This section explores the differential regularization problem from two directions. First we show that if we are given a differential operator L, then it is possible to find a kernel k that satisfies the kernel requirements, and for which we have

$$||Lf||_{L^2} = ||f||_{\mathcal{H}_k}. \tag{4.2.1}$$

Second, we start with two common SVM kernels, the Gaussian kernel and the polynomial kernel and display differential operators L such that (4.2.1) is satisfied.

We will need the following definition:

Definition 4.2.1. Given a linear operator L, a *Green's function* of L is a function k that satisfies:

$$Lk(\mathbf{x}, \mathbf{y}) = \delta(\mathbf{x} - \mathbf{y}) \tag{4.2.2}$$

where δ is the Dirac delta function.

We observe that for any function f, we have:

$$\langle f(\cdot), Lk(\mathbf{x}, \cdot)\rangle_{L^2} = \int f(t)\delta(\mathbf{x} - t)dt = f(\mathbf{x}). \tag{4.2.3}$$

Now we can prove the main proposition.

Proposition 4.2.2. *Let L be a differential operator, and let $k(\mathbf{x}, \mathbf{y})$ be a Green's function of L^*L. Then k is a positive definite kernel, and for any function f in the RKHS $\mathcal{H}$ associated with k, we have $||Lf||_{L^2} = ||f||_{\mathcal{H}}$.*

Proof. Note that to prove that k is positive definite, it suffices to exhibit a Hilbert space $\mathcal{H}$ and a function $\Phi : \mathcal{X} \to \mathcal{H}$ for which:

$$k(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y})\rangle_{\mathcal{H}}. \tag{4.2.4}$$

To this end, define Φ by $\mathbf{x} \mapsto k(\mathbf{x}, \cdot)$, and define the $\mathcal{H}$ inner product to be $\langle f, g\rangle_{\mathcal{H}} = \langle Lf, Lg\rangle_{L^2}$. The space $\mathcal{H}$ itself can be constructed by taking the correct linear combinations and completions. Then we have:

$$\langle \Phi(\mathbf{x}), \Phi(y)\rangle_{\mathcal{H}} = \langle Lk(\mathbf{x}, \cdot), Lk(\mathbf{y}, \cdot)\rangle_{L^2} = \langle k(\mathbf{x}, \cdot), L^*Lk(\mathbf{y}, \cdot)\rangle_{L^2} = k(\mathbf{x}, \mathbf{y}), \tag{4.2.5}$$

where the last equality is justified by (4.2.3).

The fact that $||Lf||_{L^2} = ||f||_{\mathcal{H}}$ is immediate by the construction of $\langle \cdot, \cdot\rangle_{\mathcal{H}}$.

□

Now we go the other direction: we start with the kernel k and find the corresponding differential operator L. Obviously, we can't prove a general theorem here. Rather, we report the results given in [17] and [16] for two common SVM kernels, the radial basis/Gaussian kernel and the polynomial kernel.

For the RBF kernel, we have (reprinted exactly from [17]):

$$||Lf|| = \int d\mathbf{x} \sum_m \frac{\sigma^{2m}}{m!2^m}(\hat{O}f(\mathbf{x}))^2 \tag{4.2.6}$$

where: σ is the standard deviation of the Gaussian kernel, $\hat{O}^{2m} = \Delta^m$ and $\hat{O}^{2m+1} = \nabla\Delta^m$, with ∇ as the Laplacian and Δ as the gradient. More informally, because of the infinite sum, using an RBF kernel penalizes all derivatives, leading to a very smooth decision function f.

For the polynomial kernel, we have, again directly reprinted, this time from [16]:

$$L = \sum_{|m|=p} e_m \binom{p}{m}^{\frac{1}{2}} D_0^m \tag{4.2.7}$$

Here, m is a mulitindex, p is the degree of the polynomial kernel, and D_0 is defined by

$$D_0^m f = \frac{1}{m_1!}\partial_{\mathbf{x}_1}^{m_1}, \ldots, \frac{1}{m_n!}\partial_{\mathbf{x}_n}^{m_n} \tag{4.2.8}$$

4.3 Regularization via Fourier Analysis

Above, we have considered encouraging smoothness by penalizing large derivatives. Here we consider another another way to encourage smoothness, namely penalizing high frequencies. For this reason, we consider the Fourier transform $\hat{f}$ of f. In order to penalize the correct frequencies, we introduce a real valued function $\upsilon(\omega)$ that is symmetric, non-negative and converges to 0 as $\mid \omega \mid \rightarrow 0$. Our regularization term is then:

$$\text{minimize} \quad \lambda \int \frac{\mid \hat{f}(\omega) \mid^2}{\upsilon(\omega)} d\omega \tag{4.3.1}$$

Because υ converges to 0 as $\mid \omega \mid \rightarrow 0$, the high frequencies ($\omega$ close to 0) must be small to keep the whole expression minimal.

In the previous section, we showed that given a differential operator L, there was a kernel k such that $||Lf|| = ||f||_{\mathcal{H}_k}$. Here we do the analogous thing by showing that given a function $\upsilon(\omega)$, there exists a reproducing kernel k such that:

$$\lambda \int \frac{\mid \hat{f}(\omega) \mid^2}{\upsilon(\omega)} d\omega = ||f||_{\mathcal{H}_k} \tag{4.3.2}$$

To do this, we reduce the problem to the one we considered the previous section. That is, we define an operator L for which:

$$||Lf||^2 = \lambda \int \frac{\mid \hat{f}(\omega) \mid^2}{\upsilon(\omega)} d\omega, \tag{4.3.3}$$

and then we find the corresponding Green's function. By the results in the previous section this Green's function will be the kernel k that we seek. Skipping the derivation, this Green's function is given (cf. [17], [10])by:

$$k(x, y) = \frac{1}{2\pi} \int e^{i\omega(\mathbf{x}-\mathbf{y})} \upsilon(\omega) d\omega \tag{4.3.4}$$

By the results in the previous section, we're done.

4.3.1 Connecting Fourier and Differential Regularization

Following [10], we show that the formulation of regularization given in this section can be used to reconstruct the differential operator formulation given previously. The key observation lies in the relationship between differentiation and the Fourier transform. We have:

$$\hat{f}'(\mathbf{x}) = i\omega\hat{f}(\mathbf{x}) \tag{4.3.5}$$

Let us choose $\upsilon(\omega) = \frac{1}{\omega^2}$. Then we have:

$$\frac{1}{2\pi}\int \frac{\mid \hat{f}(\omega) \mid^2}{\upsilon(\omega)} d\omega = \int \omega^2 \mid \hat{f}(\omega) \mid^2 d\omega = \omega^2 \langle \hat{f}, \hat{f} \rangle_{L^2} = -\langle i\omega\hat{f}, i\omega\hat{f} \rangle_{L^2} = \langle \hat{f}', \hat{f}' \rangle_{L^2} \tag{4.3.6}$$

Now, by Parseval's theorem, we have

$$\langle \hat{f}', \hat{f}' \rangle_{L^2} = \langle f', f' \rangle_{L^2} = \int \mid f(x) \mid^2 d\,\mathbf{x} = ||f'(\mathbf{x})||^2_{L^2} \tag{4.3.7}$$

Putting this together with the previous string of equalities gives:

$$\frac{1}{2\pi}\int \frac{\mid \hat{f}(\omega) \mid^2}{\upsilon(\omega)} d\omega = \int \omega^2 \mid \hat{f}(\omega) \mid^2 d\omega = ||f'(\mathbf{x})||^2_{L^2} = ||Lf||^2_{L^2} \tag{4.3.8}$$

This shows that by choosing $\upsilon(\omega) = \frac{1}{\omega^2}$ gives the same result as doing the analysis of the previous section with the differential operator $Lf = \frac{d}{d\mathbf{x}} f(\mathbf{x})$.

Similarly, a choice of $\upsilon(\omega) = \frac{1}{\omega^4}$ corresponds to a choice of $Lf = \frac{d^2}{dx^2} f(\mathbf{x})$.

4.4 Support Vector Machines

We now have all the information necessary to see how the idea of regularization plays out in the context of a specific learning algorithm, namely support vector classification.

One the face of it, it is rather suprising that support vector machines do not overfit training data. After all, for certain common parameter choices, SVMs are capable of perfectly or nearly perfectly fitting all training sets. This should immediately make us suspect overfitting. However, SVMs generalize very well in practice, indicating that overfitting does not in fact occur. It has been suggested [17] that the explanation of the phenomenon is that SVMs incorporate regularization. We explore this idea here. Our discussion follows [11].

We will show that support vector machines seek the decision function f that minimizes

$$c(f, \{\mathbf{x}_i, \mathbf{y}_i\})) + ||f||_{\mathcal{H}_k}, \tag{4.4.1}$$

where $\mathcal{H}_k$ is the RKHS corresponding the SVM kernel. Given that we have shown in sections 3 and 4 that $||f||_{\mathcal{H}_k}$ can be seen as a regularization term, this amounts to showing that SVM is a regularized learning algorithm.

Recall that the decision function for an SVM with kernel k is given by

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \cdot) \tag{4.4.2}$$

(The reason that the offset b does not appear here is that it amounts to a simple shift and may be incorporated in a choice of threshold). Since for each i, $\alpha_i y_i$ is a constant, so f is a linear combination of $k(x_i, \cdot)$. Therefore f is in the RKHS corresponding to k.

Now observe that:

$$||f||_{\mathcal{H}}^2 = \langle f, f \rangle_{\mathcal{H}} = \sum_{i=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_j \, \mathbf{y}_i \, \mathbf{y}_j \, k(\mathbf{x}_i, \mathbf{x}_j) \tag{4.4.3}$$

But now look back at the dual form of the optimization problem we solve to find f. This is:

$$\text{maximize} \quad \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{4.4.4}$$

$$\text{subject to} \quad \sum_{i=1}^{N} \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i > 0 \ \forall i \tag{4.4.5}$$

Comparing this with (4.4.3) we get:

$$\text{maximize} \quad \sum_{i=1}^{N} \alpha_i - ||f||_{\mathcal{H}}^2 \tag{4.4.6}$$

So in other words, we are looking for the acceptable f with smallest RKHS norm, which, as we have seen, amounts to seeking a smooth f.

4.5 The Representer Theorem

In the preceding sections, we have shown that in order to prevent overfitting, it is wise to minimize the sum of a loss function and a regularization term. We have also shown that the regularization properties of a function f are well characterized by $||f||_{\mathcal{H}}$. Therefore, we arrive at the optimization problem:

$$\text{minimize} \quad c(f, \{\mathbf{x}_i, \mathbf{y}_i\})) + \Omega(||f||_{\mathcal{H}}), \tag{4.5.1}$$

where Ω is non-decreasing. We will assume that the loss function c is pointwise. That is, we assume that c to depends only on the values $\{f(\mathbf{x}_i)\}$. This assumption is actually quite natural and should not be seen as restrictive.

This optimization problem is made tractable even if $\mathcal{H}$ is infinite dimensional by the following result which is known as the representer theorem. Intuitively, the representer theorem says that the solution to the optimization problem may be expressed in terms of only those kernels associated with training points, even if the space $\mathcal{H}$ is very large. Our proof follows the presentation in [1].

Theorem 4.5.1. *In the minimization problem (4.5.1), if the function Ω is strictly increasing, then any solution f has a representation of the form:*

$$f(x) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) \tag{4.5.2}$$

If Ω is non-decreasing then the optimization problem has least one solution that can be represented in the form (4.5.2).

Proof. Define $\mathcal{K} \subset \mathcal{H}$ by $\mathcal{K} = \operatorname{span}\{k(\cdot, \mathbf{x}_i)\}$. We have $\mathcal{H} = \mathcal{K} \oplus \mathcal{K}^{\perp}$, so for any function $f \in \mathcal{H}$, we have $f = f_{||} + f_{\perp}$, where $f_{||} \in \mathcal{K}$ and $f_{\perp} \in \mathcal{K}^{\perp}$. Suppose that f solves 4.5.1. We will prove that $f_{||}$ also solves 4.5.1, and that if Ω is strictly increasing then $f = f_{||}$. Noting that by definition of $\mathcal{K}$, $f_{||}$ admits a representation of the form 4.5.2, we see that this will prove the theorem.

By the reproducing property of k, we have, for all i:

$$f(\mathbf{x}_i) = \langle f, k(\mathbf{x}_i, \cdot\rangle_{\mathcal{H}_k} \tag{4.5.3}$$

$$= \langle f_{||}, k(\mathbf{x}_i, \cdot\rangle_{\mathcal{H}_k} + \langle f_{\perp}, k(\mathbf{x}_i, \cdot)\rangle_{\mathcal{H}_k} \tag{4.5.4}$$

$$= \langle f_{||}, k(\mathbf{x}_i, \cdot\rangle_{\mathcal{H}_k} \tag{4.5.5}$$

$$= f_{||}(\mathbf{x}_i). \tag{4.5.6}$$

Therefore, by the assumption that c depends only on the values $\{f(\mathbf{x}_i)\}$, we have

$$c(f_{||}) = c(f). \tag{4.5.7}$$

Furthermore, we have:

$$||f||^2_{\mathcal{H}} = ||f_{||}||^2_{\mathcal{H}} + ||f_{\perp}||^2_{\mathcal{H}} \tag{4.5.8}$$

$$\Rightarrow ||f_{||}||_{\mathcal{H}} \leq ||f||_{\mathcal{H}}. \tag{4.5.9}$$

The inequality is strict if $f_\perp \neq 0$. Combining (4.5.7) and (4.5.9) gives:

$$c(f_{||}, \{\mathbf{x}_i, y_i\})) + \Omega(||f_{||}||_{\mathcal{H}}) \leq c(f, \{\mathbf{x}_i, y_i\})) + \Omega(||f||_{\mathcal{H}}) \tag{4.5.10}$$

This immediately shows that if f solves (4.5.1), then so does $f_{||}$. Furthermore, it is easy to see that the inequality is strict if Ω is strictly increasing and $f_\perp \neq 0$. This shows that if Ω is strictly increasing, then $f_\perp = 0 \Rightarrow f = f_{||}$.

□

Chapter 5

Reinforcement Learning

In reinforcement learning, we imagine an agent interacting with the world. The agent can perform actions which change the state of its environment. Some state-action pairs are desirable and are rewarded, while others are not desirable and are punished. The agent's goal is to interact with its environment in a way that maximizes some function of its reward. We formalize these ideas with the following setup:

- The world consists of a set of states $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$. We will assume that the number of states is finite.

- The agent interacts with the world at discrete time points. At each point t it chooses an action a_t from a finite set of possible actions. Subsequently, two things happen:

- The agent receives a stochastic real-valued reward $r_t = r(a_t, \mathbf{x}_t)$, where $\mathbf{x}_t$ is the state at time t, and a_t is the action taken by the agent. This function has expected value $\bar{r}_t = \bar{r}(a_t, \mathbf{x}_t)$ Also,

- The state of the world stochastically transitions to $\mathbf{x}_{t+1}$ according to the distribution $P(\mathbf{x}_{t+1} \mid a_t, \mathbf{x}_t)$.

In the setup above, transitions to a new state only depend on the action and state on the previous time step. Given these, the new state is independent of everything that occurred at earlier times. The setup is therefore called a *Markov decision process (MDP)*. One point to note is that, as the notation above suggests, we make the simplifying assumption of stationarity: the reward and transition probabilities are independent of time.

The goal of an agent in an MDP is learn from its experience to choose good actions. Of course, we have to specify what we mean by "good". Assume that the current time index is t. Then we will define an optimal sequence of actions as one that is expected to produce a sequence of rewards $(r_{t+u})_{u=0}^{\infty}$ that maximizes the following *discounted return*:

$$\mathbb{E}\left[\sum_{u=0}^{\infty} \gamma^u r_{t+u}\right] \tag{5.0.1}$$

Recall that the quantities r_{t+u} are expected rewards. Also, $\gamma \in [0,1]$ is a *discount factor* that assigns lower values to rewards that are obtained later in time: "A bird in the hand is worth two in the bush." The presence of γ can be justified in a number of ways. First, it is mathematically necessary in order to make the sum converge. More intuitively it reflects the possibility that unforeseen events may disrupt the policy down the road: we don't want to sacrifice too much of our short-term reward for a long-term reward that may or may not materialize.

It is useful to introduce the notion of a *policy* that guides an agent's actions. A policy is a function

$$\pi : \mathbf{a} \times \mathbf{X} \to [0,1] \tag{5.0.2}$$

that assigns probabilities to action-state pairs ($\mathbf{a}$ is set of all possible actions). For a state $\mathbf{x}$, $\pi(\cdot, \mathbf{x})$ is a probability distribution over actions from which the agent samples when it finds itself in state $\mathbf{x}$. The goal of the agent can now be reformulated as finding an optimal policy π^*, where an optimal policy is one that chooses a sequence of actions that maximizes (5.0.1). How do we find a good policy? We will describe two ways. The first is called *Q learning.*

5.1 Q Learning

It would be easier to find good actions if we had some way of defining the *quality* of taking an action a in a state s, since we could then just choose actions with high quality. The problem is that if we are using (5.0.1) as our metric of performance, any reasonable notion of quality will depend not only on $\mathbf{x}_t$ and a_t, but on whatever happens afterwards as well. In order to correct for this, we use a "best case scenario" notion of quality in which quality is the discounted reward we expect to recieve from performing action a_t in state $\mathbf{x}_t$ and thereafter choosing a sequence of actions that is optimal in the sense of (5.0.1):

$$Q^*(a_t, \mathbf{x}_t) = \mathbb{E}\left[r(a_t, \mathbf{x}_t) + \sum_{u=1}^{\infty} \gamma^u r(a_{t+u}^*, \mathbf{x}_{t+u})\right] \tag{5.1.1}$$

$$= \bar{r}(a_t, \mathbf{x}_t) + \sum_{u=1}^{\infty} \gamma^u \, \bar{\mathrm{r}}(a_{t+u}^*, \mathbf{x}_{t+u}) \tag{5.1.2}$$

The star indicates optimality, so that (a^*_{t+u}) is the sequence of optimal actions. One note: we have described Q^* as a function, but since the set of states and the set of actions are both finite, it is often more helpful to think of it as a table whose columns are states and whose rows are actions, so that the ijth cell contains the quality of taking action i in state j.

To see why these quality functions are useful, recall that we are trying to find an optimal policy. Suppose that have already determined the optimal probability distribution over actions for all states other than $\mathbf{x}_t$. The quality function can then enable us to determine what to do in $\mathbf{x}_t$. It is easy to see that the best action a_t is the one with the highest quality:

$$a^*_t = \text{argmax}_{a_t} Q^*(a_t, \mathbf{x}_t) \tag{5.1.3}$$

The optimal policy is then defined in the obvious way as the deterministic map that assigns probability 1 to action a^*_t in state $\mathbf{x}_t$ with probability 1.

The obvious problem here is that we can't evaluate (5.0.4) because we don't know the optimal actions to take following a_t. The first step in overcoming this difficulty is to write down the so-called *Bellman equation* for Q^*. We can peel off the first term of the infinite sum in (5.0.4) to get

$$Q^*(a_t, \mathbf{x}_t) = \bar{\mathrm{r}}(a_t, \mathbf{x}_t) + \gamma\, \bar{\mathrm{r}}(a^*_{t+1}, \mathbf{x}_{t+1}) + \sum_{u=2}^{\infty} \gamma^u\, \bar{\mathrm{r}}(a^*_{t+u}, \mathbf{x}_{t+u}) \tag{5.1.4}$$

Which is

$$Q^*(a_t, \mathbf{x}_t) = \bar{\mathrm{r}}(a_t, \mathbf{x}_t) + \gamma Q^*(a^*_{t+1}, \mathbf{x}_{t+1}) \tag{5.1.5}$$

This problem is that this expression assumes that $\mathbf{x}_{t+1}$ can be known at time t - because of the stochastic nature of the transitions, this is impossible. The trick to overcoming this randomness is to take a probabilistic average, which gives:

$$Q^*(a_t, \mathbf{x}_t) = \bar{\mathrm{r}}(a_t, \mathbf{x}_t) + \gamma \sum_{s'} P(s' \mid a_t, \mathbf{x}_t) \text{max}_{a_{t+1}} Q^*(a_{t+1}, s') \tag{5.1.6}$$

This gives us a recursive definition of the Q^* values. Since the set of possible actions and the set of possible states are both finite, (5.0.8) gives us, in principle, a way to find the value of Q^* for every action-state pair by solving a system of linear equations with coefficients $P(s' \mid a_t, \mathbf{x}_t)$. In practice, though, the coefficients are not known, so we need yet another work-around.

Observe that to make evaluation of (5.0.8) practical, we need to find the expected value of the function $\text{max}_{a_{t+1}} Q^*(a_{t+1}, s')$ with respect to the distribution $P(s' \mid a_t, \mathbf{x}_t)$. The trick is to approximate this expected value with samples - in fact with one sample.

Obtaining this sample is easy: if the agent is in state $\mathbf{x}_t$ and performs action a_t, it can observe the state $\mathbf{x}_{t+1}$ that results, and $\max_{a_{t+1}} Q^*(a_{t+1}, \mathbf{x}_{t+1})$ is the desired sample. Substituting this into (5.0.8) we get:

$$Q^*(a_t, \mathbf{x}_t) \approx r(a_t, \mathbf{x}_t) + \gamma \max_{a_{t+1}} Q^*(a_{t+1}, \mathbf{x}_{t+1}) \tag{5.1.7}$$

Of course, this is *still* not a practical formula, since it includes the quantities Q^* that we want to define. However, we can use an iterative scheme: For each state action pair, we initialize an estimate $\hat{Q}^*(a, s)$ to some value, and then at each timestep use the following update rule:

$$\hat{Q}^*(\mathbf{x}_t, a_t) \leftarrow \hat{Q}^*(\mathbf{x}_t, a_t) + \epsilon(r(a_t, \mathbf{x}_t) + \gamma \max_{a_{t+1}} \hat{Q}^*(a_{t+1}, \mathbf{x}_{t+1}) - \hat{Q}^*(a, s)) \tag{5.1.8}$$

Here α is a learning rate. The interpretation is that is that iteration brings the current estimate closer to the estimate given by , to an extent determined by α.

The above iteration requires the agent to choose actions before it knows the optimal policy, so we have to specify the policy by which it does this. An important idea that motivates this choice is the so-called *exploration exploitation tradeoff.* The idea is that the agent wants to use its current knowledge to obtain high rewards (exploitation), but it also wants to try many different actions so that it will be better able to learn an optimal policy in the long-run (exploration). With this tradeoff in mind, we use an α-greedy policy, in which the agent chooses the action that is optimal with respect to its current quality estimates with probability $1 - \alpha$, and a random action with probability α. It is possible to elaborate this policy by decreasing α over time, the idea being that the estimates should be better, and less exploration should be needed, later on.

5.2 Actor-critic learning

We now discuss another learning scheme. As the name suggests, actor-critic methods have two components. The actor components performs actions according to a policy, and the critic evaluates these actions and suggests improvements to the policy.

We need to introduce the following *value* function, which will be recognized as being very similar to the Q^* functions defined earlier. The value of a state with respect to a policy π is defined as the discounted return that is expected to result from following policy π in state s.

$$V^\pi(\mathbf{x}_t) = \mathbb{E}\left[\sum_{u=0}^{\infty} r(\pi(\mathbf{x}_{t+u}), \mathbf{x}_{t+u})\right]. \tag{5.2.1}$$

Where we use $\pi(\mathbf{x}_{t+u})$ to denote the action recommended by the policy π in state $\mathbf{x}_{t+u}$.

The critic's job is to keep estimates of these value functions and to use them to evaluate and improve the policy. We will first describe how estimates are obtained, and then show how they can be used to improve the policy that is currently being followed.

The first step is to write down the Bellman equation for the value function, which can be obtained using the same reasoning that we used to deduce the Bellman equation for the Q^* functions. We have

$$V^\pi(\mathbf{x}_t) = \sum_a \pi(\mathbf{x}_t, a)[\bar{\mathrm{r}}(\mathbf{x}_t, a) + \gamma \sum_{s'} P(s' \mid \mathbf{x}_t, a) V^\pi(s')] \tag{5.2.2}$$

As a side note, we can give an intuitive way to understand (5.0.12). The sequence of possible actions and states in an MDP can be represented as a tree (like a game tree in classical AI). If we imagine that the tree is finite with known values at the leaves, then we can propagate the values up the tree by labeling the nodes adjacent to the roots with a likelihood-weighted average of the root values. We can repeat this process until labels have been assigned to all nodes in the tree. One can notice that this scheme is formalized by (5.0.12).

As we did with Q^* learning, we replace the averages with single samples. This entails taking one action from the distribution given by π, and observing the state to which it leads. The estimate is then

$$V^\pi(\mathbf{x}_t) = r(\pi(\mathbf{x}_t), \mathbf{x}_t) + \gamma V^\pi(\mathbf{x}_{t+1}), \tag{5.2.3}$$

where, as before, $\pi(\mathbf{x}_t)$, is the action recommended by π at time t. As with Q^* learning, we use an iterative scheme, and initialize estimates $\hat{V}^\pi(\mathbf{x})$ for each state $\mathbf{x}$. The update is

$$\hat{V}^\pi(\mathbf{x}_t) \leftarrow \hat{V}^\pi(\mathbf{x}_t) + \alpha[r(\pi(\mathbf{x}_t), \mathbf{x}_t) + \gamma \hat{V}^\pi(\mathbf{x}_{t+1}) - \hat{V}^\pi(\mathbf{x}_t)] \tag{5.2.4}$$

This completes the acquisition of value estimates.

Having obtained estimates $\hat{V}^\pi(\mathbf{x})$ of the value of all states, we can describe how to use these estimates to improve our policy. It is convenient to introduce parameters $m(\mathbf{x}, a)$ for each state $\mathbf{x}$ and action a and to assume that:

$$\pi(\mathbf{x}, a) = \frac{e^{m(\mathbf{x}, a)}}{e^{\sum_{a'} m(\mathbf{x}, a')}} \tag{5.2.5}$$

We can then adjust these parameters to make it more likely that the policy π chooses actions that are good according the estimate of V. Define the following

$$\delta_t = r(\pi(\mathbf{x}_t), \mathbf{x}_t) + \gamma \hat{V}^\pi(\mathbf{x}_{t+1}) - \hat{V}^\pi(\mathbf{x}_t) \tag{5.2.6}$$

Looking at (5.0.13), we see that the sum of the two left-hand terms can be interpreted as the estimate of the value of state $\mathbf{x}_t$ obtained after the action a_t has been performed $r(\pi(\mathbf{x}_t), \mathbf{x}_t)$ has been observed. This is likely to be somewhat better than the prior estimate $\hat{V}^{\pi}(\mathbf{x}_t)$, since it includes the reward $r(\pi(\mathbf{x}_t), \mathbf{x}_t)$ that was actually obtained. Therefore δ_t can be interpreted as the difference in the predicted value $\hat{V}^{\pi}(\mathbf{x}_t)$ and the updated prediction $r(t) + \gamma \hat{V}^{\pi}(\mathbf{x}_{t+1})$. If δ_t is positive, then update is greater than original estimate, and we conclude that a_t turned out to be better than expected. Conversely, if δ_t is negative, then the a_t was worse than expected. Thus, we can punish actions that were worse than expected and encourage actions that were better than expected by using the update

$$m(s(t), a(t)) \leftarrow m(s(t), a(t)) + \epsilon_2 \delta(t), \tag{5.2.7}$$

where ϵ_2 is another learning rate.

Note that actor-critic learning requires updates both to the estimated value function, as in (5.0.14) and to the policy π, as in (5.0.17) The most theoretically justified way to balance these two factors is wait until the value estimate is as exact as possible before updating the policy. In practice, though, it is more common to alternate the two updates.

5.3 Hierarchical Reinforcement Learning

The reinforcement learning framework presented above has some serious drawbacks that make it computationally expensive and cognitively unrealistic. The most glaring of these is best illustrated with an example. When I walk to school in the morning, I don't plan each step I take individually. Rather, I plan at a larger scale: get to the top of the hill, turn left, walk straight for one block, etc. In other words, I don't plan in terms of primitive actions (steps); I plan in terms of blocks of actions. Looking back the previous section, though, we see that an agent in an MDP plans only in terms of steps: a choice of action as to be made at every single time step.

Hierarchical reinforcement learning is designed to overcome this problem by grouping actions in various ways, and then planning in terms of these groups. A number of approaches to hierarchical reinforcement learning have been proposed. We follow the approach taken in [2].

5.3.1 The Options Framework

In the options framework [18], we still have a Markov decision process with ordinary actions, but these actions are organized hierarchically by objects called *options*. An option is a triple, $o = (I, \mu, T)$. I is a set of states from which the option may be initiated. For example, if I have a "climb the hill" option as part of my walk to school, this option can be initiated only at the bottom of the hill. Next, T is a of termination conditions, $T : \mathbf{X} \to [0, 1]$, where $T(\mathbf{x})$ is the probability that the option terminates in state $\mathbf{x}$. The "climb the hill" option might be defined to have probability one of terminating at the top of the hill, and probability zero of terminating elsewhere.

The most important part of the option is μ, which is a policy *over options*. That is, μ is a function

$$\mu : \mathbf{o} \times \mathbf{X} \to [0, 1], \tag{5.3.1}$$

where $\mathbf{o}$ is the set of all possible options. The interpretation here is the same as the interpretation of standard policies: μ in a given state μ assigns a probability to each option. The idea is that while it is active, each option o chooses other options, which in turn choose other options, and so on. When an option terminates, control returns to its parent, and depending on its own termination conditions may terminate or may choose another option. Obviously, for the agent to actually accomplish anything, this process must eventually "ground out" in primitive actions. In order to accomplish this while maintaining the fact that options always choose other options, we simply include the set of primitive actions in the set options, so that each action is an option that, with probability one, terminates in all states.

We would like to be able to define a version of Q-learning in the options setup, where the Q^* functions are now interpreted as the quality of initiating option o in state $\mathbf{x}$ and choosing optimal options thereafter. Recall that the Q-learning update rule is

$$\hat{Q}^*(\mathbf{x}_t, a_t) \leftarrow \hat{Q}^*(\mathbf{x}_t, a_t) + \epsilon(r(a_t, \mathbf{x}_t) + \gamma \text{max}_{a_{t+1}} \hat{Q}^*(a_{t+1}, \mathbf{x}_{t+1}) - \hat{Q}^*(a, s)) \tag{5.3.2}$$

We would like to simply replace each a_t with o_t. The problem is the presence of the reward $r(a_t, \mathbf{x}_t)$. Choosing an option results in the execution of a whole sequence of primitive actions, each with their own reward. Therefore, in the options framework, there is no "instantaneous" reward obtained by an option at each time step. There is a fairly natural way to define the symbol $r(o_t, \mathbf{x}_t)$, however. We take our definition to be the *total* discounted reward accumulated while the option o was active. So, if o was active for $\tau + 1$ timesteps after after taking effect at time t, we define

$$r(o_t, \mathbf{x}_t) = \sum_{u=0}^{\tau} \gamma^u r_{t+u} \tag{5.3.3}$$

This definition enables us to straightforwardly extend Q-learning to the options framework. The update is

$$\hat{Q}^*(\mathbf{x}_t, o_t) \leftarrow \hat{Q}^*(\mathbf{x}_t, o_t) + \epsilon(r(o_t, \mathbf{x}_t) + \gamma^\tau \text{max}_{o_{t+\tau}} \hat{Q}^*(o_{t+\tau}, \mathbf{x}_{t+\tau}) - \hat{Q}^*(o_t, \mathbf{x}_t)) \tag{5.3.4}$$

As before, the estimates are initialized before the iteration begins. As this exposition illustrates, one of the chief advantages of the options framework is that it has many features in common with ordinary, non-heierarchical reinforcement learning, and many ideas from non-hierarachical reinforcement learning can be easily generalized to the options case.

5.3.2 Hierarchies of Abstract Machines

Like the options framework, the hierarchies of abstract machines (HAM) framework [15] consists of a standard MDP with some superstructure. This time, rather than options, we have a collection of finite state machines, each of which has a specific function (e.g. "climb the hill"). The internal states of these machines are of four types. In an **action** state, the machine performs an action in the base MDP. In a **call** state, the machine calls another machine. In a **terminate** state, the machine terminates, and passes control back to the machine that called it. In a **choice** state, the machine transitions (stochastically) to another internal state. After each step, the machine transitions to a new state based on its internal transition function and on the state of the base MDP.

In the walking to school example, an agent might have a machine for traveling along a particular section of the route. When the agent is simply walking, the machine stays in the action states necessary to take steps. When the agent gets to a traffic light, though, the machine might transition to a choice state that enables it to transition to either of two call states. In one of these it can call a machine designed to let the agent cross at the crosswalk, and in the other it can call a machine to let it jaywalk. Once the street is crossed, the internal state of whichever street-crossing machines was called will transition to a terminate state, and control will return to the original machine.

At this point, we have two types of states: internal states in the machines, and states of the environment in the base MDP. The first learning in the HAM framework is to combine these two sets of states into one two-dimensional state space. Call the set of all possible internal states of all possible machines $\mathbf{S}$. Then the two-dimensional state space is $\mathbf{S} \times \mathbf{X}$, where, as before, $\mathbf{X}$ is the set of states of the base MDP. We also have transitions in $\mathbf{S} \times \mathbf{X}$, defined by the transition functions of the machine and the base MDP. (Note that following a "choice" machine state, the state of the MDP does not change). It is possible to prove that the two-dimesnional state space and transition functions define an MDP. The goal now is to apply Q learning in this MDP.

The observation is that any given machine is "on autopilot" in all states other than choice states. Therefore, we define Q learning in a way similar to the approach we took when defining Q learning in the options framework. Let $[s_c^t, \mathbf{x}_c^t] \in \mathbf{S} \times \mathbf{X}$ be a choice point encountered at time t, and let a_c^t be the action taken by the machine. Then we perform the following update:

$$\hat{Q}^*([s_c^t, \mathbf{x}_c^t], a_c^t) \leftarrow \hat{Q}^*([s_c^t, \mathbf{x}_c^t], a_c) + \epsilon(r + \gamma^\tau \mathrm{max}_{a_c^{t+\tau}} \hat{Q}^*([s_c^{t+\tau}, \mathbf{x}_c^{t+\tau}]a') - \hat{Q}^*([s_c^t, \mathbf{x}_c^t], a_c^t)) \tag{5.3.5}$$

Here, τ is the number of timesteps until the next choice point, $[s_c^{t+\tau}, \mathbf{x}_c^{t+\tau}]$, and r is the discounted reward accumulated between the two choice points.

5.3.3 MAXQ

As with the other frameworks we have discussed, the MAXQ framework [4] starts with a decomposition of the overall task into subtasks, each with their own policies that allow them to call their children, and conditions that specify when they terminate and return control to their parent tasks. As with the options framework, MAXQ includes the performance of primitive actions in the set of subtasks. The important difference between MAXQ and other approaches is that MAXQ is designed to learn programs that can solve subtasks in a way that allows these programs to be reused in the context of different supertasks.

As a consequence of this idea, where the other frameworks we have discussed are designed to find an optimal solution to the master task, MAXQ is designed to find optimal solutions to each of the subtasks, while allowing that patching together these optimal solutions may yield a suboptimal solution to the master task. This idea can be defined more rigorously: given a set of subtasks arranged in hierarchy, a set of policies for these subtasks is called *recursively optimal* if the solution to each subtask is optimal given the solutions to its child subtasks.

Following this idea of treating each subtask as an important learning problem in its own right, we define value functions and quality functions that are specific to each subtask. First we need some preliminary definitions. We have said that the subtasks in MAXQ each have their own policies according to which they call their children. It is convenient to compile these policies into a single *hierarchical policy* π, in which π_i is the policy for subtask i. Next, consider a subtask i. As we have discussed, the analogue of an action taken by i in the MAXQ framework is a request that a child subtask a be completed. We define the transition probability $P_i(\mathbf{x}' \mid \mathbf{x}, a)$ to be the probability that the environment will be in state $\mathbf{x}'$, after subtask a has been completed and control is returned to i. Similarly, we define the reward function $r_i(\mathbf{x}, a)$ to be the reward accrued during the period of time while a is being completed, until control returns to i.

Now we can define the value functions. When we defined h the value of state $\mathbf{x}$ to subtask i under hierarchical policy π, V_i^π is the reward that is expected to accrue during the period until subtask i terminated if hierarchical policy π is followed. With these definitions, it is easy to see that $r_i(a, \mathbf{x})$ under hierarchical policy π is $V_a^\pi(\mathbf{x})$.

This enables us to make the following recursive definition of the quality functions:

$$Q_i^\pi(\mathbf{x}, a) = V_a^\pi(\mathbf{x}) + \sum_{s'} P_i(\mathbf{x}' \mid \mathbf{x}, a) V_i^\pi(\mathbf{x}') \tag{5.3.6}$$

We define the leftmost term as $C_i^\pi(\mathbf{x}, a) = \sum_{\mathbf{x}'} P_i(\mathbf{x}' \mid s, a) V_i^\pi(\mathbf{x}')$, which we interpret as the reward associated with completing subtask i from state $\mathbf{x}$ after performing action a. So we can summarize our results as follows:

$$Q_i^\pi(\mathbf{x}, a) = V_a^\pi(\mathbf{x}) + C_i^\pi(\mathbf{x}, a) \tag{5.3.7}$$

$$V_a^\pi(\mathbf{x}) = \begin{cases} Q_i^\pi(s, \pi_a(\mathbf{x})) & a \text{ composite} \\ r(a, \mathbf{x}) & a \text{ primitive} \end{cases} \tag{5.3.8}$$

This shows that in order to compute quantities of interest, such as value and quality functions, it suffices to have estimates of the completion functions. For example, if we substitute (5.1.8) in (5.1.7), we get

$$Q_i^\pi(\mathbf{x}, a) = Q_a^\pi(s, \pi_a(\mathbf{x})) + C_i^\pi(\mathbf{x}, a) = V_{\pi_a(\mathbf{x})}^\pi(\mathbf{x}) + C_a^\pi(\mathbf{x}, \pi_a(\mathbf{x})) + C_i^\pi(\mathbf{x}, a) \tag{5.3.9}$$

We can then perform another substitution to expand $V_{\pi_a(s)}^\pi(s)$. We can continue in this way until we reach a primitive action. Thus, we can express each Q function as a sum of terms involving completion functions, and value functions corresponding to primitive actions.

Before showing how to obtain estimates of the completion functions, we introduce one more feature of the MAXQ framework. In many situations, a subtask may be solved in a variety of different ways, some of which will be better than others in the context of the master task. Therefore, it is desirable to have a mechanism by which the programmer can bias subtasks to be completed in particular ways. This is accomplished by introducing subtask-specific pseudoreward functions $\tilde{r}_i$. In general, these are defined only on the set of sets the subtask may terminate, and are set to a large positive value for a set of "goal states" and a large negative value otherwise.

We can now define the updates. Recall that $C_i^\pi(s, a)$ is the expected reward from completing task i after performing a in s. We can therefore express it as the sum of the obtained instantaneous reward, and the reward we expect when we continue from state s'. As we did with Q^* learning , we assume a "best case" scenario in which we perform the best action a^* next. The difference here is that the definition of best depends on the pseudoreward function:

$$a^* = \text{argmax}_{a'}[\tilde{C}_i^\pi(s', a') + V_{a'}(s')] \tag{5.3.10}$$

The interpretation is that a^* maximizes the genuine reward that we will get from perfoming, and the pseudoreward we can expect to receive when subtask i terminates. So assuming that a^* is the next action a, the reward we expect after the instantaneous reward from a is $C_i^\pi(s', a^*) + V_{a^*}^\pi(s')$. (This follows immediately from the fact that $V_{a^*}^\pi(s')$ gives the immediate reward and $C_i^\pi(s', a^*)$ gives the rest of the reward. So the update is:

$$C_i^\pi(s,a) \leftarrow C_i^\pi(s,a) + \alpha_t(i)[r(a,s) + C_i^\pi(s', a^*) + V_{a^*}^\pi(s') - C_i^\pi(s,a)] \tag{5.3.11}$$

where $\alpha_t(i)$ is a learning rate that requires to decrease to zero as time goes to infinity. Note that V value can be obtained recursively via (5.1.7) and (5.1.8).

We also have to update the estimate of $\tilde{C}_i^\pi(s,a)$. The idea of this update is similar, except that both genuine and pseudoreawrd terms are included:

$$\tilde{C}_i^\pi(s,a) \leftarrow \tilde{C}_i^\pi(s,a) + \alpha_t(i)[\tilde{r}_i(s') + r(a,s) + C_i^\pi(s', a^*) + V_{a^*}^\pi(s') - \tilde{C}_i^\pi(s,a)] \tag{5.3.12}$$

If a is a primitive action, the update is:

$$V_a^\pi(s) = V_a(s) + \alpha_t(a)[r(a,s) - V_a^\pi(s)] \tag{5.3.13}$$

As with Q learning, we still have to specify how the agent uses these estimates to behave. It turns out that in this setup, it is desirable for the agent to follow a greedy in the limit with infinite exploration (GLIE) policy, for example an ϵ-greedy policy in which ϵ decreases as the inverse of time.

Bibliography

[1] Peter Bartlett and R. Honicky. Representer theorem and construction kernels: Lecture given to cs281b, 2006. `http://www.cs.berkeley.edu/ bartlett/courses/281b-sp06/lecture09.ps`.

[2] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] Thomas G. Dietterich. The maxq method for hierarchical reinforcement learning. In *ICML*, pages 118–126, 1998.

[5] Zoubin Ghahramani. An introduction to hidden markov models and bayesian networks. *IJPRAI*, 15(1):9–42, 2001.

[6] Thomas L. Griffiths and Zoubin Ghahramani. Infinite latent feature models and the indian buffet process. In *In NIPS*, pages 475–482. MIT Press, 2005.

[7] T. Hofmann, B. Schlkopf, and A. J. Smola. Kernel methods in machine learning. *Annals of Statistics*, 36(3):1171–1220, 06 2008.

[8] A. Hyvarinen and E. Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.

[9] M. I. Jordan. Hierarchical models, nested models and completely random measures. In H. Geffner R. Dechter and J. Halpern, editors, *Heuristics, Probability and Causality: A Tribute to Judea Pearl*. College Publications, 2010.

[10] Michael Jordan and Amol Deshpande. Fourier perspective on regularization: Lecture given to cs281b, 2001. `http://www.cs.berkeley.edu/ jordan/courses/281B-spring01/lectures/fourier.ps`.

[11] Michael Jordan and Nemanja Isailovic. Reproducing kernel hilbert spaces 2: Lecture given to cs281b, 2004. `http://www.cs.berkeley.edu/ jordan/courses/281B-spring04/lectures/rkhs2.ps`.

[12] David Knowles and Zoubin Ghahramani. Infinite sparse factor analysis and infinite independent components analysis.

[13] Radford Neal and Geoffrey E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, 1998.

[14] Radford M. Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9:249–265, 2000.

[15] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *NIPS '97: Proceedings of the 1997 conference on Advances in neural information processing systems 10*, pages 1043–1049, Cambridge, MA, USA, 1998. MIT Press.

[16] Alex J. Smola, Bernhard Schölkopf, and Klaus-Robert Müller. The connection between regularization operators and support vector kernels. *Neural Netw.*, 11(4):637–649, 1998.

[17] Alexander J. Smola and Bernhard Schölkopf. From regularization operators to support vector kernels. In *NIPS '97: Proceedings of the 1997 conference on Advances in neural information processing systems 10*, pages 343–349, Cambridge, MA, USA, 1998. MIT Press.

[18] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999.

[19] Y. W. Teh. Dirichlet processes. In *Encyclopedia of Machine Learning*. Springer, 2010.

[20] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006.

[21] Romain Thibaux and Michael I. Jordan. Hierarchical beta processes and the indian buffet process. this volume. Technical report, In Practical Nonparametric and Semiparametric Bayesian Statistics, 2007.

[22] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. *Understanding belief propagation and its generalizations*, pages 239–269. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.