5

Approximate Inference

In many settings, simple algorithms like variable elimination may be unreasonably slow. The general problem of exact probabilistic inference in Bayesian networks is NP-hard (Cooper, 1990), and many interesting classes of models may not admit exact polynomial-time solutions at all. For this reason, much research effort in machine learning is spent on developing algorithms that yield *approximate* solutions to the inference problem. Approximate inference algorithms generally forgo the accuracy of exact inference in exchange for computational efficiency. Still, approximating probability distributions remains a challenging task: even approximate probabilistic inference in Bayesian networks is NP-hard in the general case (Dagum and Luby, 1993), though special cases can be polynomial-time with respect to input size.

There exist two main families of approximate algorithms: *sampling* methods, which produce answers by repeatedly generating random numbers from a distribution of interest, and *variational* methods, which formulate inference as an optimization problem. Sampling methods can be used to perform both marginal and MAP inference queries. In addition, they can compute various interesting quantities, such as expectations $\mathbb{E}[f(X)]$ of random variables distributed according to a

given probabilistic model. Sampling methods have historically been the main way of performing approximate inference, although variational methods have emerged as viable (and often computationally superior) alternatives. Variational inference methods (sometimes called *variational Bayes*) take their name from the *calculus of variations*, which deals with optimizing functions that take other functions as arguments.

We begin this chapter by exploring general approximate inference methods. First, we introduce *sampling-based approaches*, covering popular Monte Carlo and Markov methods (Section 5.1). We then provide intuition for *inference as optimization*, reviewing the foundations of variational inference (Section 5.2). Finally, we conclude by revisiting special cases of *MAP inference* through the lens of approximate inference strategies (Section 5.3).

5.1 Sampling-Based Approximate Inference

5.1.1 Sampling from a Probability Distribution

As a warm-up, let's think for a moment about how we might sample from a multinomial distribution with k possible outcomes and associated probabilities $\theta_1, \ldots, \theta_k$.

Sampling, in general, is not an easy problem. Our computers can only generate samples from very simple distributions such as the uniform distribution over $[0, 1]$. Even those samples are not truly random. They are actually taken from a deterministic sequence whose statistical properties (e.g., running averages) are indistinguishable from a truly random one. We call such sequences *pseudorandom*. All sampling techniques involve calling some kind of simple subroutine multiple times in a clever way.

In our case, we may reduce sampling from a multinomial variable to sampling a single uniform variable by subdividing a unit interval into k regions with region i having size θ_i (Figure 5.1). We then sample uniformly from $[0, 1]$ and return the value of the region in which our sample falls.

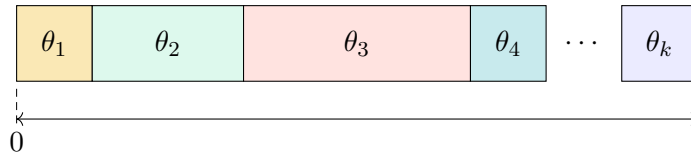

Figure 5.1: Reducing the problem of sampling from a multinomial distribution to sampling a uniform distribution in $[0, 1]$ by subdividing a unit interval into k regions, where each region i is size θ_i.

Forward Sampling

Our technique for sampling from multinomials naturally extends to Bayesian networks with multinomial-distributed variables, via a method called *ancestral* (or *forward*) sampling. In particular, to sample from a probabilistic model $p(\mathbf{x}) = p(x_1, \ldots, x_n)$ specified by a Bayesian network, we can sample variables in topological order. We start by sampling the variables with no parents; then we sample from the next generation by conditioning these variables' conditional probability distributions on values sampled at the first step. We proceed like this until all n variables have been sampled. Importantly, in a Bayesian network over n variables, forward sampling allows us to sample from the joint distribution $\mathbf{x} \sim p(\mathbf{x})$ in linear, $O(n)$, time by taking exactly 1 multinomial sample from each conditional probability distribution.

As an example, take our earlier model of student grades (Figure 3.1). Here, we would first sample an exam difficulty d' and an investment level i'. Then, once we have samples d' and i', we generate a student grade g' from $p(g \mid d', i')$. At each step, we simply perform standard multinomial sampling.

Forward sampling can also be performed efficiently on undirected graphical models if the model can be represented as a junction tree (Definition 4.1) with a small number of variables per node. We begin by performing message passing, until each clique holds a potential proportional to the marginal distribution over the variables in that clique, conditioned on any observed evidence. We then choose one of the cliques in the junction tree to act as the root. Sampling proceeds in a top-down manner through the tree. Starting at the root clique, we sample its variables sequentially, conditioning each sample on the previously

sampled variables within the clique. After sampling all variables in the root clique, we pass the sampled values of shared variables to a neighboring child clique. In that child clique, we condition on these shared variables to sample the remaining (unseen) variables in the clique. This process continues: at each clique, we condition on previously sampled variables, sample the remaining variables in the clique, and pass the samples of shared variables to the next neighboring cliques in the tree.

5.1.2 Monte Carlo Estimation

Sampling from a distribution lets us perform many useful tasks, including marginal inference, MAP inference, and computing integrals of the form

$$\mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})] = \sum_{\mathbf{x}} f(\mathbf{x})p(\mathbf{x}).$$

If $f(\mathbf{x})$ does not have special structure that matches the Bayesian network structure of p, this integral will typically be impossible to perform analytically. Instead, we will approximate it using a large number of samples from p. Algorithms that construct solutions based on a large number of samples from a given distribution are referred to as *Monte Carlo*[1] (MC) methods.

Monte Carlo integration is an important instantiation of the general Monte Carlo principle. This technique approximates a target expectation with

$$\mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})] \approx I_T = \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{x}^t),$$

where $\mathbf{x}^1, \ldots, \mathbf{x}^T$ are samples drawn according to p. It can be shown that

$$\mathbb{E}_{\mathbf{x}^1,\ldots,\mathbf{x}^T \overset{iid}{\sim} p}[I_T] = \mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})]$$
$$\mathrm{Var}_{\mathbf{x}^1,\ldots,\mathbf{x}^T \overset{iid}{\sim} p}[I_T] = \frac{1}{T}\mathrm{Var}_{\mathbf{x}\sim p}[f(\mathbf{x})].$$

[1] The name *Monte Carlo* refers to a famous casino in the city of Monaco. The term was originally coined as a codeword by physicists working on the atomic bomb as part of the secret Manhattan project.

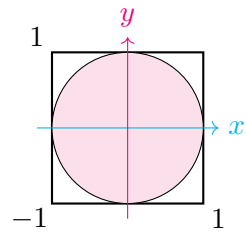

Figure 5.2: Graphical illustration of rejection sampling. We may compute the area of circle by drawing uniform samples from the square. The fraction of points that fall in the circle represents its area. This method breaks down if the size of the circle is small relative to the size of the square.

The first equation says that the MC estimate I_T is an unbiased estimator for $\mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})]$. The two equations together show that $I_T \to \mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})]$ as $T \to \infty$. In particular, the variance of I_T can be made arbitrarily small with enough samples.

Rejection Sampling

A special case of Monte Carlo integration is rejection sampling. We may use it to compute the area of a region γ by sampling in a larger region with a known area and recording the fraction of samples that falls within γ.

For example, suppose we have a Bayesian network over the set of variables $\mathbf{X} = \mathbf{Z} \cup \mathbf{E}$. We may use rejection sampling to compute marginal probabilities $p(\mathbf{E} = \mathbf{e})$. We can rewrite the probability as

$$p(\mathbf{E} = \mathbf{e}) = \sum_{\mathbf{z}} p(\mathbf{Z} = \mathbf{z}, \mathbf{E} = \mathbf{e}) = \sum_{\mathbf{x}} p(\mathbf{x})\mathbb{I}(\mathbf{E} = \mathbf{e}) = \mathbb{E}_{\mathbf{x}\sim p}[\mathbb{I}(\mathbf{E} = \mathbf{e})]$$

and then take the Monte Carlo approximation. In other words, we draw many samples from p and report the fraction of samples that are consistent with the value of the marginal.

Importance Sampling

Unfortunately, rejection sampling can be very wasteful. If $p(\mathbf{E} = \mathbf{e})$ equals, say, 1%, then we will discard 99% of all samples.

A better way of computing such integrals uses *importance sampling*. The main idea is to sample from a distribution q (hopefully with $q(\mathbf{x})$ roughly proportional to $f(\mathbf{x}) \cdot p(\mathbf{x})$), and then *reweigh* the samples in a principled way, so that their sum still approximates the desired integral.

More formally, suppose we are interested in computing $\mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})]$. We may rewrite this integral as

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})] &= \sum_{\mathbf{x}} f(\mathbf{x})p(\mathbf{x}) \\
&= \sum_{\mathbf{x}} f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x}) \\
&= \mathbb{E}_{\mathbf{x}\sim q}[f(\mathbf{x})w(\mathbf{x})] \\
&\approx \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{x}^t)w(\mathbf{x}^t)
\end{aligned}$$

where $w(\mathbf{x}) = \frac{p(\mathbf{x})}{q(\mathbf{x})}$ and the samples $\mathbf{x}^t$ are drawn independently from q. In other words, we may instead take samples from q and reweigh them with $w(\mathbf{x})$. The expected value of this Monte Carlo approximation will be the original integral. The variance of this new estimator is

$$\mathrm{Var}_{\mathbf{x}\sim q}[f(\mathbf{x})w(\mathbf{x})] = \mathbb{E}_{\mathbf{x}\sim q}[f^2(\mathbf{x})w^2(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim q}[f(\mathbf{x})w(\mathbf{x})]^2 \geq 0.$$

Note that we can set the variance equal to zero by choosing

$$q(\mathbf{x}) = \frac{|f(\mathbf{x})|p(\mathbf{x})}{\int |f(\mathbf{x})|p(\mathbf{x})d\mathbf{x}}.$$

If we can sample from this q (and evaluate the corresponding weight), then we only need a single Monte Carlo sample to compute the true value of our integral. Of course, sampling from such a q is NP-hard in general (its denominator $\mathbb{E}_{\mathbf{x}\sim p}[|f(\mathbf{x})|]$ is basically the quantity we're trying to estimate in the first place), but this at least gives us an indication for what to strive for.

In the context of our previous example for computing $p(\mathbf{E} = \mathbf{e})$, we may take q to be the uniform distribution and apply importance

sampling as follows:

$$\begin{aligned} p(\mathbf{E} = \mathbf{e}) &= \mathbb{E}_{\mathbf{z} \sim p}[p(\mathbf{e} \mid \mathbf{z})] \\ &= \mathbb{E}_{\mathbf{z} \sim q}\left[p(\mathbf{e} \mid \mathbf{z}) \frac{p(\mathbf{z})}{q(\mathbf{z})}\right] \\ &= \mathbb{E}_{\mathbf{z} \sim q}\left[\frac{p(\mathbf{e}, \mathbf{z})}{q(\mathbf{z})}\right] \\ &= \mathbb{E}_{\mathbf{z} \sim q}[w_{\mathbf{e}}(\mathbf{z})] \\ &\approx \frac{1}{T} \sum_{t=1}^{T} w_{\mathbf{e}}(\mathbf{z}^t) \end{aligned}$$

where $w_{\mathbf{e}}(\mathbf{z}) = p(\mathbf{e}, \mathbf{z})/q(\mathbf{z})$, and each $\mathbf{z}^t$ is sampled independently from q. Unlike rejection sampling, this will use all of the samples in our calculation. If $p(\mathbf{z} \mid \mathbf{e})$ is not too far from uniform, this will converge to the true probability after only a very small number of samples.

Normalized Importance Sampling

Unfortunately, unnormalized importance sampling is not suitable for estimating conditional probabilities of the form

$$p(\mathbf{Z}_i = \mathbf{z}_i \mid \mathbf{E} = \mathbf{e}) = \frac{p(\mathbf{Z}_i = \mathbf{z}_i, \mathbf{E} = \mathbf{e})}{p(\mathbf{E} = \mathbf{e})},$$

where $\mathbf{Z}_i \subseteq \mathbf{Z}$. Note that, using unnormalized importance sampling, we could estimate the numerator as

$$\begin{aligned} p(\mathbf{Z}_i = \mathbf{z}_i, \mathbf{E} = \mathbf{e}) &= \sum_{\mathbf{z}} \delta(\mathbf{z}) p(\mathbf{e}, \mathbf{z}) \\ &= \sum_{\mathbf{z}} \delta(\mathbf{z}) w_{\mathbf{e}}(\mathbf{z}) q(\mathbf{z}) \\ &= \mathbb{E}_{\mathbf{z} \sim q}[\delta(\mathbf{z}) w_{\mathbf{e}}(\mathbf{z})] \\ &\approx \frac{1}{T} \sum_{t=1}^{T} \delta(\mathbf{z}^t) w_{\mathbf{e}}(\mathbf{z}^t). \end{aligned}$$

where

$$\delta(\mathbf{z}) = \begin{cases} 1 & \text{if } \mathbf{z} \text{ is consistent with } \mathbf{Z}_i = \mathbf{z}_i \\ 0 & \text{otherwise.} \end{cases}$$

While the denominator is the same as the result we derived earlier:

$$p(\mathbf{E} = \mathbf{e}) \approx \frac{1}{T} \sum_{t=1}^{T} w_{\mathbf{e}}(\mathbf{z}^t),$$

where the $\mathbf{z}^t$ are drawn independently from q. If we estimate the numerator $p(\mathbf{Z}_i = \mathbf{z}_i, \mathbf{E} = \mathbf{e})$ and the denominator $p(\mathbf{E} = \mathbf{e})$ with different and independent samples of $\mathbf{z}^t \sim q$, then the errors in the two approximations may compound. For example, if the numerator is an under-estimate and the denominator is an over-estimate, the final probability could be a severe under-estimate.

However, if we instead use the same set of T samples $\mathbf{z}^1, \ldots, \mathbf{z}^T \sim q$ for both the numerator and denominator, we avoid this issue of compounding errors. Thus, the final form of the normalized importance sampling estimate can be written

$$\hat{p}_T(\mathbf{Z}_i = \mathbf{z}_i \mid \mathbf{E} = \mathbf{e}) = \frac{\frac{1}{T}\sum_{t=1}^{T} \delta(\mathbf{z}^t) w_{\mathbf{e}}(\mathbf{z}^t)}{\frac{1}{T}\sum_{t=1}^{T} w_{\mathbf{e}}(\mathbf{z}^t)},$$

where we use the notation $\hat{p}_T$ to denote the sampling-based approximation to the density of interest p, given T samples.

Unfortunately, there is one drawback to the normalized importance sampling estimator, which is that it is *biased*. If $T = 1$, then we have

$$\mathbb{E}_{\mathbf{z}\sim q}[\hat{p}_1(\mathbf{Z}_i = \mathbf{z}_i \mid \mathbf{E} = \mathbf{e})] = \mathbb{E}_{\mathbf{z}\sim q}[\delta(\mathbf{z})] \neq p(\mathbf{Z}_i = \mathbf{z}_i \mid \mathbf{E} = \mathbf{e})$$

Fortunately, because the numerator and denominator are both unbiased, the normalized importance sampling estimator remains *asymptotically unbiased*, meaning that

$$\lim_{T\to\infty} \mathbb{E}_{\mathbf{z}\sim q}[\hat{p}_T(\mathbf{Z}_i = \mathbf{z}_i \mid \mathbf{E} = \mathbf{e})] = p(\mathbf{Z}_i = \mathbf{z}_i \mid \mathbf{E} = \mathbf{e}).$$

5.1.3 Markov Chain Monte Carlo

We now turn our attention from computing expectations to performing marginal and MAP inference using sampling. We will solve these problems using a technique called *Markov chain Monte Carlo* (MCMC).

MCMC is another algorithm that was developed during the Manhattan project and eventually republished in the scientific literature some decades later. It is so impactful that it was recently named as one of the ten most important algorithms of the 20^{th} century (Cipra, 2000).

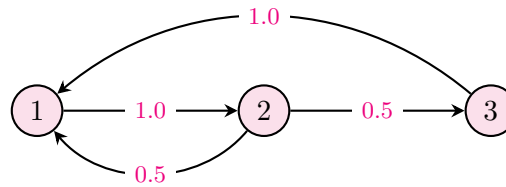

Figure 5.3: A Markov chain over three states. The weighted directed edges indicate probabilities of transitioning to a different state.

Markov Chains

A key concept in MCMC is that of a *Markov chain*. A (discrete-time) Markov chain is a sequence of random variables $(S_0, S_1, S_2, \ldots)$ with each random variable $S_i \in \{1, 2, \ldots, d\}$ taking one of d possible values, intuitively representing the state of a system. The initial state is distributed according to a probability $P(S_0)$. All subsequent states are generated from a conditional probability distribution that depends only on the previous random state. That is, S_i is distributed according to $P(S_i \mid S_{i-1})$.

The probability $P(S_i \mid S_{i-1})$ is the same at every step i. This means that the transition probabilities at any time in the entire process depend only on the given state and not on the history of how we got there. This is called the *Markov assumption*.

It is very convenient to represent the transition probability as a $d \times d$ matrix $\mathbf{T}$ where

$$\mathbf{T}_{ij} = P(S_{\text{new}} = i \mid S_{\text{prev}} = j).$$

If the initial state S_0 is drawn from a vector of probabilities p_0, we may represent the probability p_t of ending up in each state after t steps as

$$p_t = \mathbf{T}^t p_0,$$

where $\mathbf{T}^t$ denotes matrix exponentiation (i.e., we apply the matrix operator t times). The limit

$$\pi = \lim_{t \to \infty} p_t$$

(when it exists) is called a *stationary distribution* of the Markov chain. We will construct below a Markov chain with a stationary distribution

π that exists and is the same for all p_0. We will refer to such π as *the* stationary distribution of the chain.

Sufficient Conditions for Stationary Distributions

A sufficient condition for the existence of a stationary distribution for a Markov chain with transition probability $\mathbf{T}$ is *detailed balance*, defined as:

$$\pi(x')\mathbf{T}(x \mid x') = \pi(x)\mathbf{T}(x' \mid x) \quad \forall x.$$

It is easy to show that such a π must form a stationary distribution by summing both sides of the equation over x and simplifying. However, the reverse may not hold and indeed it is possible to have MCMC without satisfying detailed balance (Suwa and Todo, 2010).

The high-level idea of MCMC will be to construct a Markov chain whose states will be joint assignments to the variables in the model and whose stationary distribution will equal a probability distribution of interest. Suppose we have defined a Markov chain with a stationary distribution π. In addition to the existence of a stationary distribution, we need to ensure that the chain will converge to this stationary distribution from any initial state (as required in MCMC). In particular, we desire the Markov chain to be ergodic, i.e., to converge to a unique stationary distribution π over time. This turns out to be true under two sufficient conditions:

1. *Irreducibility*: It is possible to get from any state x to any other state x' with probability > 0 in a finite number of steps.

2. *Aperiodicity*: It is possible to return to any state at any time. That is, there exists an n such that for all i and all $n' \geq n$,

$$P(s_{n'} = i \mid s_0 = i) > 0.$$

The first condition is meant to prevent *absorbing states* (i.e., states from which we can never leave). In the example in Figure 5.4, if we start in states 1 or 2, we will never reach state 4. Conversely, if we start in state 4, then we will never reach states 1 or 2. If we start the chain in the middle (in state 3), then clearly it cannot have a single limiting

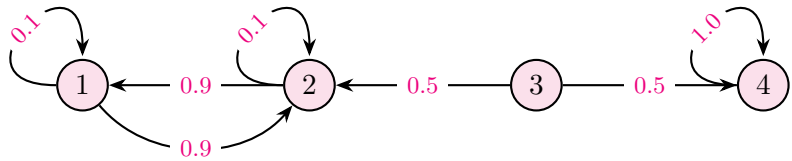

Figure 5.4: A reducible Markov chain over four states.

distribution. The second condition is necessary to rule out transition operators such as

$$\mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Note that this chain alternates forever between states 1 and 2 without ever settling in a stationary distribution.

Theorem 5.1. An irreducible and aperiodic finite-state Markov chain has a stationary distribution.

In the case of continuous state spaces, ensuring that a Markov chain is ergodic typically requires stronger conditions than just irreducibility and aperiodicity. For generality, we will simply assume that our Markov chains are ergodic.

High-Level Procedure

MCMC algorithms construct a Markov chain over the assignments to a probability function p. The chain will have a stationary distribution equal to p itself. By running the chain for some number of time steps, we will thus sample from p.

At a high level, MCMC algorithms will have the following structure. They take as argument a transition operator $\mathcal{T}$ specifying a Markov chain whose stationary distribution is p, and an initial assignment $\mathbf{x}_0$ to the variables of p. We then perform the following steps.

1. Run the Markov chain from $\mathbf{x}_0$ for B *burn-in* steps.

2. Run the Markov chain for N *sampling* steps and collect all the states that it visits.

Assuming B is sufficiently large, the latter collection of states will form samples from p. We may then use these samples for Monte Carlo integration (or importance sampling). We may also use them to produce Monte Carlo estimates of marginal probabilities. Finally, we may take the sample with the highest probability and use it as an estimate of the mode (i.e., to perform MAP inference).

The Metropolis-Hastings Algorithm

The Metropolis-Hastings (MH) algorithm is our first way to construct Markov chains within MCMC. The MH method constructs a transition operator $\mathcal{T}(\mathbf{x}' \mid \mathbf{x})$ from two components:

1. A transition kernel $\mathcal{Q}(\mathbf{x}' \mid \mathbf{x})$, specified by the user.

2. An acceptance probability for moves proposed by $\mathcal{Q}$, specified by the algorithm as

$$\mathcal{A}(\mathbf{x}' \mid \mathbf{x}) = \min\left(1, \frac{p(\mathbf{x}')\mathcal{Q}(\mathbf{x} \mid \mathbf{x}')}{p(\mathbf{x})\mathcal{Q}(\mathbf{x}' \mid \mathbf{x})}\right).$$

At each step of the Markov chain, we choose a new point $\mathbf{x}'$ according to $\mathcal{Q}$. Then, we either accept this proposed change (with probability α), or we remain at our current state with probability $1-\alpha$. Notice that the acceptance probability encourages us to move towards more likely points in the distribution p (imagine for example that $\mathcal{Q}$ is uniform). When $\mathcal{Q}$ suggests that we move into a low-probability region, we follow that move only a certain fraction of the time. In practice, the distribution $\mathcal{Q}$ is taken to be something simple, like a Gaussian centered at $\mathbf{x}$ if we are dealing with continuous variables.

Given any $\mathcal{Q}$, the MH algorithm will ensure that p will be a stationary distribution of the resulting Markov chain. More precisely, p will satisfy the detailed balance condition with respect to the MH Markov chain. To see that, first observe that if $\mathcal{A}(\mathbf{x}' \mid \mathbf{x}) < 1$, then

$$\frac{p(\mathbf{x})\mathcal{Q}(\mathbf{x}' \mid \mathbf{x})}{p(\mathbf{x}')\mathcal{Q}(\mathbf{x} \mid \mathbf{x}')} > 1$$

and thus $\mathcal{A}(\mathbf{x} \mid \mathbf{x}') = 1$. When $\mathcal{A}(\mathbf{x}' \mid \mathbf{x}) < 1$, this lets us write:

$$\begin{aligned}
\mathcal{A}(\mathbf{x}' \mid \mathbf{x}) &= \frac{p(\mathbf{x}')\mathcal{Q}(\mathbf{x} \mid \mathbf{x}')}{p(\mathbf{x})\mathcal{Q}(\mathbf{x}' \mid \mathbf{x})} \\
p(\mathbf{x}')\mathcal{Q}(\mathbf{x} \mid \mathbf{x}')\mathcal{A}(\mathbf{x} \mid \mathbf{x}') &= p(\mathbf{x})\mathcal{Q}(\mathbf{x}' \mid \mathbf{x})\mathcal{A}(\mathbf{x}' \mid \mathbf{x}) \\
p(\mathbf{x}')\mathcal{T}(\mathbf{x} \mid \mathbf{x}') &= p(\mathbf{x})\mathcal{T}(\mathbf{x}' \mid \mathbf{x}),
\end{aligned}$$

which is simply the detailed balance condition. We used $\mathcal{T}(\mathbf{x} \mid \mathbf{x}')$ to denote the full transition operator of MH (obtained by applying both $\mathcal{Q}$ and $\mathcal{A}$). Thus, if the MH Markov chain is ergodic, its stationary distribution will be p.

Gibbs Sampling

A widely-used special case of MH methods is Gibbs sampling. Given an ordered set of variables $(x_1, \ldots, x_n)$ and a starting configuration $\mathbf{x}^0 = (x_1^0, \ldots, x_n^0)$, Gibbs sampling iteratively samples each variable in turn from its conditional distribution, given the current values of all other variables. At iteration t, the algorithm updates each variable x_i in sequence by drawing

$$x_i^{(t)} \sim p\left(x_i \mid x_1^{(t)}, \ldots, x_{i-1}^{(t)}, x_{i+1}^{(t-1)}, \ldots, x_n^{(t-1)}\right)$$

i.e., conditioning on the most recent values of all other variables. We outline the full Gibbs sampling procedure in Algorithm 4.

We use $\mathbf{x}_{-i}$ to denote all variables in $\mathbf{x}$ except x_i. It is often very easy to performing each sampling step, since we only need to condition x_i on its Markov blanket (Definitions 3.7, 3.8), which is typically small. Note that when we update x_i, we *immediately* use its new value for sampling other variables x_j.

Gibbs sampling can be seen as a special case of MH with proposal

$$\mathcal{Q}(x_i', \mathbf{x}_{-i} \mid x_i, \mathbf{x}_{-i}) = p(x_i' \mid \mathbf{x}_{-i}).$$

It is easy check that the acceptance probability simplifies to one. Assuming the right transition operator, both Gibbs sampling and MH will eventually produce samples from their stationary distribution, which by construction is p. There exist simple ways of ensuring that this will be the case.

Algorithm 4 *Gibbs Sampling*

Repeat until convergence for $t = 1, 2, \ldots$:

1. Set $\mathbf{x} \leftarrow \mathbf{x}^{t-1}$.
2. For each variable x_i in the order we fixed:
   (a) Sample $x_i' \sim p(x_i \mid \mathbf{x}_{-i})$
   (b) Update $\mathbf{x} \leftarrow (x_1, \ldots, x_i', \ldots, x_n)$.
3. Set $\mathbf{x}^t \leftarrow \mathbf{x}$

1. To ensure *irreducibility*, the transition operator $\mathcal{Q}$ with MH should be able to move to every state. In the case of Gibbs sampling, we would like to make sure that every x_i' can be sampled from $p(x_i \mid \mathbf{x}_{-i})$.
2. To ensure *aperiodicity*, it is enough to let the chain transition stay in its state with some probability.

In practice, it is not difficult to ensure that these requirements are met.

Running Time of MCMC

A key parameter to this algorithm is the number of burn-in steps B. Intuitively, this corresponds to the number of steps needed to converge to our limit (stationary) distribution. This is called the *mixing time* of the Markov chain.

Unfortunately, this time may vary dramatically and may sometimes take (essentially) forever. For example, if the transition matrix is

$$\mathbf{T} = \begin{bmatrix} 1-\epsilon & \epsilon \\ \epsilon & 1-\epsilon \end{bmatrix},$$

then for small ϵ it will take a very long time to reach the stationary distribution, which is close to $(0.5, 0.5)$. At each step, we will stay in the same state with overwhelming probability. Very rarely, we will transition to another state, and then stay there for a very long time. The average

of these states will converge to $(0.5, 0.5)$, but the convergence will be very slow.

This problem will also occur with complicated distributions that have two distinct and narrow modes. With high probability, the algorithm will sample from a given mode for a very long time. These examples are indications that sampling is a hard problem in general, and MCMC does not give us a free lunch. Nonetheless, for many real-world distributions, sampling will produce very useful solutions.

Another – and perhaps more important – problem is that we may not know when to end the burn-in period, even if it is theoretically not very long. There exist many heuristics to determine whether a Markov chain has *mixed*. However, these heuristics typically involve plotting certain quantities and estimating them by eye. Even the quantitative measures are not significantly more reliable than this approach.

In summary, even though MCMC is able to sample from the right distribution (which in turn can be used to solve any inference problem), doing so may sometimes require a very long time. Furthermore, there is no easy way to judge the amount of computation that we will need to spend to find a good solution.

There are also a number of more-advanced MCMC methods that can improve on mixing performance by incorporating gradient information about the target distribution. These gradient-based MCMC algorithms include Langevin Monte Carlo (LMC), which augments random-walk proposals with steps in the direction of the gradient of the log-density, and Hamiltonian Monte Carlo (HMC), which simulates trajectories under a physical system that uses position and momentum variables to explore the space more efficiently. LMC and HMC can make more informed steps in high-dimensional parameter spaces and often achieve faster convergence with improved mixing. For more information on gradient-based MCMC, see Brooks *et al.* (2011) and Betancourt (2017).

Further Reading

Approximate Inference with Sampling

- Chapter 11 in C. M. Bishop. (2006). *Pattern Recognition and*

Machine Learning. Springer.

- Chapter 12 in D. Koller and N. Friedman. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.
- Chapters 23 & 24 in K. P. Murphy. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.
- F. Lindsten, T. B. Schön, *et al.* (2013). "Backward simulation methods for Monte Carlo statistical inference". *Foundations and Trends® in Machine Learning*. 6(1): 1–143.
- E. Angelino *et al.* (2016). "Patterns of Scalable Bayesian Inference". *Foundations and Trends® in Machine Learning*. 9(2): 119–247.
- C. A. Naesseth *et al.* (2019). "Elements of Sequential Monte Carlo". *Foundations and Trends® in Machine Learning*. 12(3): 187–306.
- S. Brooks *et al.* (2011). *Handbook of Markov chain Monte Carlo*. CRC Press.

5.2 Variational Methods for Approximate Inference

We have seen that inference in probabilistic models is often intractable, and we have learned about some algorithms that provide approximate solutions to the inference problem by using subroutines involving sampling random variables. Unfortunately, these sampling-based methods have several important shortcomings.

1. Although MCMC methods are guaranteed to find a globally optimal solution given enough time, it is difficult to tell how close they are to a good solution given the finite amount of time that they have in practice.
2. In order to quickly reach a good solution, MCMC methods require choosing an appropriate sampling technique (e.g., a good proposal

in MH). Choosing this technique can be an art in itself.

In this section, we will discuss an alternative approach to approximate inference called the *variational inference* family of algorithms.

5.2.1 Inference as Optimization

The main idea of variational inference methods is to cast inference as an optimization problem. Suppose we are given an intractable probability distribution p. Variational techniques will try to solve an optimization problem over a class of tractable distributions $\mathcal{Q}$ in order to find a $q \in \mathcal{Q}$ that is most similar to p. We will then query q (rather than p) in order to get an approximate solution.

The main differences between sampling and variational techniques include the following:

1. Unlike sampling-based methods, variational approaches will almost never find the globally optimal solution.

2. However, we will always know if they have converged. In some cases, we will even have bounds on their accuracy.

3. In practice, variational inference methods often scale better and are more amenable to techniques like stochastic gradient optimization, parallelization over multiple processors, and GPU acceleration.

Although sampling methods were historically invented first (in the 1940s), variational inference techniques have grown increasingly popular in recent years and are now commonly used in many applications.

5.2.2 The Kullback-Leibler Divergence

To formulate inference as an optimization problem, we need to choose an approximating family $\mathcal{Q}$ and an optimization objective $J(q)$. This objective needs to capture the similarity between q and p. The field of information theory provides us with a tool for this called the *Kullback-Leibler divergence*.

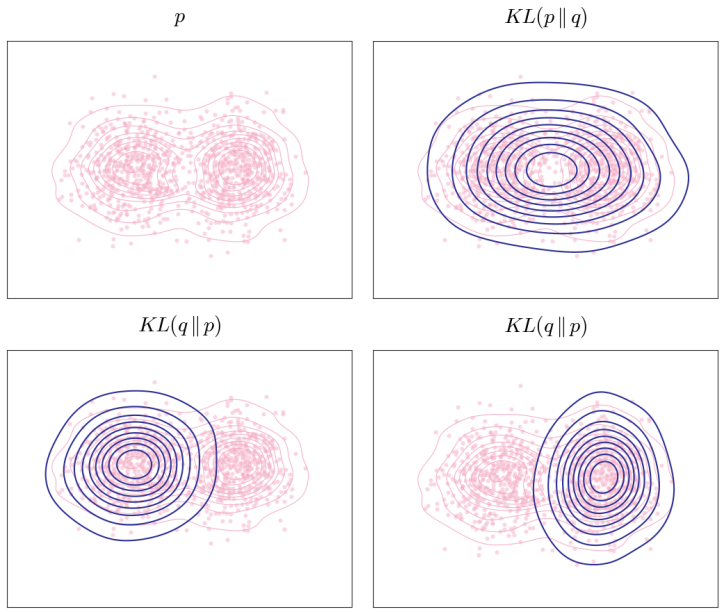

Figure 5.5: Fitting a unimodal approximating distribution q (blue) to a multimodal p (pink). Using $\mathrm{KL}(p \,\|\, q)$ leads to a q that tries to cover both modes. Using $\mathrm{KL}(q \,\|\, p)$ forces q to choose one of the two modes of p.

Definition 5.1 (Kullback-Leibler (KL) divergence)**.** The KL divergence between two distributions q and p with discrete support is defined as

$$\mathrm{KL}(q \,\|\, p) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})}. \tag{5.1}$$

In information theory, this function is used to measure differences in information contained within two distributions. The KL divergence has the following properties that make it especially useful in our setting:

Proposition 5.1. $\mathrm{KL}(q \,\|\, p) \geq 0$ for all q, p.

Proposition 5.2. $\mathrm{KL}(q \,\|\, p) = 0$ if and only if $q = p$.

These can be proven as an exercise. Note that $\mathrm{KL}(q \,\|\, p) \neq \mathrm{KL}(p \,\|\, q)$, i.e., the KL divergence is not symmetric. This is why we say that it's a *divergence*, but not a distance. We will come back to this distinction shortly.

5.2.3 The Variational Lower Bound

How do we perform variational inference with a KL divergence? First, let's fix a form for p. We'll assume that p is a general (discrete, for simplicity) undirected model of the form

$$p(x_1, \ldots, x_n; \theta) = \frac{\tilde{p}(x_1, \ldots, x_n; \theta)}{Z(\theta)} = \frac{1}{Z(\theta)} \prod_k \phi_k(\mathbf{x}_k; \theta),$$

where the ϕ_k are the factors and $Z(\theta)$ is the normalization constant. This formulation captures virtually all the distributions in which we might want to perform approximate inference, such as conditional distributions of directed models $p(\mathbf{x} \mid \mathbf{e}) = p(\mathbf{x}, \mathbf{e})/p(\mathbf{e})$ with evidence $\mathbf{e}$.

Given this formulation, optimizing $\mathrm{KL}(q \,\|\, p)$ directly is not possible because of the potentially intractable normalization constant $Z(\theta)$. In fact, even evaluating $\mathrm{KL}(q \,\|\, p)$ is not possible, because we need to evaluate p.

Instead, we will work with the following objective, which has the same form as the KL divergence, but only involves the unnormalized probability $\tilde{p}(\mathbf{x}) = \prod_k \phi_k(\mathbf{x}_k; \theta)$:

$$J(q) = \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{\tilde{p}(\mathbf{x})}.$$

This function is not only tractable, it also has the following important property.

Proposition 5.3.

$$\begin{aligned} J(q) &= \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{\tilde{p}(\mathbf{x})} \\ &= \sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} - \log Z(\theta) \\ &= \mathrm{KL}(q \,\|\, p) - \log Z(\theta). \end{aligned}$$

Since $\mathrm{KL}(q \,\|\, p) \geq 0$, we get by rearranging terms that

$$\log Z(\theta) = \mathrm{KL}(q \,\|\, p) - J(q) \geq -J(q).$$

Thus, $-J(q)$ is a *lower bound* on the log partition function $\log Z(\theta)$. In many cases, $Z(\theta)$ has an interesting interpretation. For example,

we may be trying to compute the conditional (posterior) probability $p(\mathbf{x} \mid \mathcal{D}) = p(\mathbf{x}, \mathcal{D})/p(\mathcal{D})$ of variables $\mathbf{x}$ given observed data $\mathcal{D}$ that plays the role of evidence. We assume that $p(\mathbf{x}, \mathcal{D})$ is directed. In this case, minimizing $J(q)$ amounts to maximizing a lower bound on the log-likelihood $\log p(\mathcal{D})$ of the observed data.

Because of this property, $-J(q)$ is called the *variational lower bound* or the *evidence lower bound*. It is often written in the following form.

Definition 5.2 (Evidence Lower Bound (ELBO))**.**

$$\log Z(\theta) \geq \mathbb{E}_{q(\mathbf{x})}[\log \tilde{p}(\mathbf{x}) - \log q(\mathbf{x})]. \tag{5.2}$$

Crucially, the difference between $\log Z(\theta)$ and $-J(q)$ is precisely $\mathrm{KL}(q \,\|\, p)$. Thus, by maximizing the evidence lower bound, we are minimizing $\mathrm{KL}(q \,\|\, p)$ by "squeezing" it between $-J(q)$ and $\log Z(\theta)$.

5.2.4 On the Choice of KL Divergence

To recap, we have just defined an optimization objective for variational inference (the variational lower bound) and we have shown that maximizing this lower bound leads to minimizing the divergence $\mathrm{KL}(q \,\|\, p)$.

Recall that $\mathrm{KL}(q \,\|\, p) \neq \mathrm{KL}(p \,\|\, q)$. Both divergences equal zero when $q = p$, but assign different penalties when $q \neq p$. This raises the question: why did we choose one over the other and how do they differ?

Perhaps the most important difference is computational: optimizing $\mathrm{KL}(q \,\|\, p)$ involves an expectation with respect to q, while $\mathrm{KL}(p \,\|\, q)$ requires computing expectations with respect to p, which is typically intractable even to evaluate.

However, choosing this particular divergence affects the returned solution when the approximating family $\mathcal{Q}$ does not contain the true p. Observe that $\mathrm{KL}(q \,\|\, p)$ — which is called the *I-projection* or *information projection* — is infinite if $p(\mathbf{x}) = 0$ and $q(\mathbf{x}) > 0$. This means that if $p(\mathbf{x}) = 0$ we must have $q(\mathbf{x}) = 0$. We say that $\mathrm{KL}(q \,\|\, p)$ is zero-forcing for q and it will typically under-estimate the support of p.

On the other hand, $\mathrm{KL}(p \,\|\, q)$ — known as the *M-projection* or the *moment projection* — is infinite if $q(\mathbf{x}) = 0$ and $p(\mathbf{x}) > 0$. Thus, if $p(\mathbf{x}) > 0$ we must have $q(\mathbf{x}) > 0$. We say that $\mathrm{KL}(p \,\|\, q)$ is zero-avoiding

for q and it will typically over-estimate the support of p. Figure 5.5 illustrates this phenomenon graphically.

Due to the above properties, we often call $\mathrm{KL}(p \,\|\, q)$ the *inclusive* KL divergence, and $\mathrm{KL}(q \,\|\, p)$ the *exclusive* KL divergence.

5.2.5 Mean Field Inference

The next step in our development of variational inference concerns the choice of approximating family $\mathcal{Q}$. The machine learning literature contains dozens of proposed ways to parametrize this class of distributions. These include exponential families, neural networks, Gaussian processes, latent variable models, and many other types of models.

However, one of the most widely used classes of distributions is simply the set of fully-factored $q(\mathbf{x}) = q_1(x_1)q_2(x_2)\cdots q_n(x_n)$. Here, each $q_i(x_i)$ is a categorical distribution over a one-dimensional discrete variable, which can be described as a one-dimensional table.

This choice of $\mathcal{Q}$ turns out to be easy to optimize over and works surprisingly well. It is perhaps the most popular choice when optimizing the variational bound. Variational inference with this choice of $\mathcal{Q}$ is called *mean field* variational inference. It consists in solving the following optimization problem:

$$\hat{q} = \operatorname*{argmin}_{q_1,\ldots,q_n} J(q).$$

The standard way of performing this optimization problem is via coordinate descent over the q_j: we iterate over $j = 1, 2, \ldots, n$ and for each j we optimize $\mathrm{KL}(q \,\|\, p)$ over q_j while keeping the other "coordinates" $q_{-j} = \prod_{i \neq j} q_i$ fixed.

Interestingly, the optimization problem for one coordinate has a simple closed form solution:

$$\log q_j(x_j) \leftarrow \mathbb{E}_{q_{-j}} \left[\log \tilde{p}(\mathbf{x})\right] + \text{const}.$$

Notice that both sides of the above equation contain univariate functions of x_j: we are thus replacing $q(x_j)$ with another function of the same form. The constant term is a normalization constant for the new distribution.

Notice also that on the right-hand side, we are taking an expectation of a sum of factors

$$\log \tilde{p}(\mathbf{x}) = \sum_k \log \phi(\mathbf{x}_k).$$

Of these, only factors belonging to the Markov blanket of x_j are a function of x_j, by the definition of the Markov blanket (Definitions 3.7, 3.8). The rest are constant with respect to x_j and can be pushed into the constant term.

This leaves us with an expectation over a much smaller number of factors. If the Markov blanket of x_j is small (as is often the case), we are able to analytically compute $q(x_j)$. For example, if the variables are discrete with K possible values, and there are F factors and N variables in the Markov blanket of x_j, then computing the expectation takes $O(KFK^N)$ time: for each value of x_j we sum over all K^N assignments of the N variables, and in each case, we sum over the F factors.

The result of this is a procedure that iteratively fits a fully-factored $q(\mathbf{x}) = q_1(x_1)q_2(x_2)\cdots q_n(x_n)$ that approximates p in terms of $\mathrm{KL}(q \,\|\, p)$. After each step of coordinate descent, we increase the variational lower bound, tightening it around $\log Z(\theta)$.

In the end, the factors $q_j(x_j)$ will not quite equal the true marginal distributions $p(x_j)$, but they will often be good enough for many practical purposes, such as determining $\max_{x_j} p(x_j)$.

5.2.6 Loopy Belief Propagation

As we have seen, the junction tree algorithm (Section 4.3) has a running time that is potentially exponential in the size of the largest cluster (since we need to marginalize all the cluster's variables). For many graphs, it will be difficult to find a good junction tree and applying the algorithm will not be possible. However, some use cases do not require the exact solution that the junction tree algorithm provides. Instead, we may be satisfied with a quick approximate solution. *Loopy belief propagation* (LBP) is a special case of variational inference that obtains approximate solutions for complex (non-tree structured) graphs (Frey and MacKay, 1997). Below, we briefly describe LBP.

Definition for Pairwise Models

Suppose that we are given an MRF with pairwise potentials. The main idea of LBP is to disregard loops in the graph and perform message passing anyway. In other words, given an ordering on the edges, at each

time $\mathcal{T}$ we iterate over a pair of adjacent variables x_i, x_j in that order and simply perform the update

$$m_{i\to j}^{t+1}(x_j) = \sum_{x_i} \phi(x_i)\phi(x_i, x_j) \prod_{\ell \in N(i)\setminus j} m_{\ell\to i}^{t}(x_i).$$

We keep performing these updates for a fixed number of steps or until convergence (the messages don't change). Messages are typically initialized uniformly.

Properties

This heuristic approach often works surprisingly well in practice. In general, however, it may not converge and its analysis is still an area of active research. We know for example that it provably converges on trees and on graphs with at most one cycle. If the method does converge, its beliefs may not necessarily equal the true marginals, although they will often be close in practice.

Further Reading

Variational Inference

- M. I. Jordan *et al.* (1999). "An introduction to variational methods for graphical models". *Machine learning*. 37: 183–233.
- K. P. Murphy *et al.* (1999). "Loopy belief propagation for approximate inference: an empirical study". In: *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. 467–475.
- M. J. Wainwright and M. I. Jordan. (2008). "Graphical Models, Exponential Families, and Variational Inference". *Foundations and Trends® in Machine Learning*. 1(1): 1–305. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000001.
- M. D. Hoffman *et al.* (2013). "Stochastic variational inference". *Journal of Machine Learning Research*.
- D. M. Blei *et al.* (2017). "Variational inference: A review for

statisticians". *Journal of the American statistical Association.* 112(518): 859–877.

5.3 Approximate MAP Inference

5.3.1 Linear Programming for Approximate MAP Inference

We conclude this chapter by describing efficient approximate inference procedures for special-cases of MAP inference—paralleling the final section of the previous chapter, which focused on efficient *exact* inference algorithms for MAP queries.

Although graph cut-based methods recover the exact MAP assignment (Section 4.4.2), they are only applicable in certain restricted classes of MRFs. The algorithms we see next solve the MAP problem approximately, but apply to much larger classes of graphical models.

Our first approximate inference strategy consists of reducing MAP inference to integer linear programming. Linear programming (LP) — also known as linear optimization — refers to a class of problems of the form

$$\begin{array}{ll} \text{minimize (over } \mathbf{x}) & \mathbf{c} \cdot \mathbf{x} \\ \text{subject to} & A\mathbf{x} \preceq \mathbf{b} \end{array}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the optimization variable, and $\mathbf{c}, \mathbf{b} \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ are problem parameters.

Problems of this form are found in almost every field of science and engineering. They have been extensively studied since the 1930s, which has led to extensive theory. A major breakthrough of applied mathematics in the 1980s was the development of polynomial-time algorithms for linear programming (Karmarkar, 1984; Adler *et al.*, 1989) and practical tools that can solve very large LP instances in a reasonable time (e.g., 100,000 variables or more; IBM 1987).

Integer linear programming (ILP) is an extension of linear programming in which we also require that $\mathbf{x} \in \{0, 1\}^n$. Unfortunately, this makes optimization considerably more difficult, and ILP is in general NP-hard. Nonetheless, there are many heuristics for solving ILP problems in practice, and commercial solvers can handle instances with

thousands of variables or more.

One of the main techniques for solving ILP problems is *rounding*. In rounding, we relax the requirement that $\mathbf{x} \in \{0,1\}^n$ into $0 \leq \mathbf{x} \leq 1$ and solve the resulting LP. Finally, we round the LP solution to its nearest integer value. This approach works surprisingly well in practice and has theoretical guarantees for some classes of ILPs.

Formulating MAP Inference as ILP

For simplicity, let's look at MAP in pairwise MRFs. Consider the MRF $\mathcal{G} = (\mathbf{V}, \mathbf{E})$. We can reduce the MAP objective to integer linear programming by introducing two types of indicator variables:

1. A variable $\mu_i(x_i)$ for each $i \in \mathbf{V}$ and state x_i.
2. A variable $\mu_{ij}(x_i, x_j)$ for each edge $(i,j) \in \mathbf{E}$ and pair of states x_i, x_j.

We can rewrite the MAP objective in terms of these variables as

$$\max_{\mu} \sum_{i \in \mathbf{V}} \sum_{x_i} \theta_i(x_i)\mu_i(x_i) + \sum_{i,j \in \mathbf{E}} \sum_{x_i, x_j} \theta_{ij}(x_i, x_j)\mu_{ij}(x_i, x_j).$$

We would like to optimize over these μ's. For that, we also need to introduce constraints. First, we need to force each cluster to choose a local assignment:

$$\begin{aligned} \mu_i(x_i) &\in \{0,1\} \; \forall \, i, x_i \\ \sum_{x_i} \mu_i(x_i) &= 1 \; \forall \, i \\ \mu_{ij}(x_i, x_j) &\in \{0,1\} \; \forall \, i, j \in \mathbf{E}, x_i, x_j \\ \sum_{x_i, x_j} \mu_{ij}(x_i, x_j) &= 1 \; \forall \, i, j \in \mathbf{E}. \end{aligned}$$

These assignments must also be consistent:

$$\begin{aligned} \sum_{x_i} \mu_{ij}(x_i, x_j) &= \mu_j(x_j) \; \forall \, i, j \in \mathbf{E}, x_j \\ \sum_{x_j} \mu_{ij}(x_i, x_j) &= \mu_i(x_i) \; \forall \, i, j \in \mathbf{E}, x_i. \end{aligned}$$

Together, these constraints along with the MAP objective yield an integer linear program whose solution equals the MAP assignment. This ILP is still NP-hard, but we have a simple way to transform this into an (easy to solve) LP via relaxation. This is the essence of the linear programming approach to MAP inference.

In general, this method will only give approximate solutions. An important special case are tree-structured graphs, in which the relaxation is guaranteed to always return integer solutions, which are in turn optimal (Koller and Friedman, 2009).

5.3.2 Dual Decomposition

Let us now look at another way to transform the MAP objective into a more amenable optimization problem. Suppose that we are dealing with an MRF of the form

$$\max_{\mathbf{x}} \sum_{i \in \mathbf{V}} \theta_i(x_i) + \sum_{f \in \mathcal{F}} \theta_f(\mathbf{x}_f),$$

where $f \in \mathcal{F}$ denote arbitrary factors (e.g., the edge potentials in a pairwise MRF). Let us use p^* to denote the optimal value of this objective and let $\mathbf{x}^*$ denote the optimal assignment. The above objective is difficult to optimize because the potentials are coupled. Consider for a moment an alternative objective where we optimize the potentials separately:

$$\sum_{i \in \mathbf{V}} \max_{x_i} \theta_i(x_i) + \sum_{f \in \mathcal{F}} \max_{\mathbf{x}^f} \theta_f(\mathbf{x}^f).$$

This would be easy to optimize, but would only give us an upper bound on the value of the true MAP assignment. To make our relaxation tight, we would need to introduce constraints that encourage consistency between the potentials:

$$x_i^f - x_i = 0 \; \forall f, \forall i \in f.$$

The dual decomposition approach consists in softening these constraints in order to achieve a middle ground between the two optimization objective defined above.

We will achieve this by first forming the *Lagrangian* for the constrained problem, which is

$$L(\delta, \mathbf{x}^f, \mathbf{x}) = \sum_{i \in \mathbf{V}} \theta_i(x_i) + \sum_{f \in \mathcal{F}} \theta_f(\mathbf{x}^f) + \sum_{f \in \mathcal{F}} \sum_{i \in f} \sum_{x'_i} \delta_{fi}(x'_i) \left(\mathbb{I}_{x'_i = x_i} - \mathbb{I}_{x'_i = x_i^f} \right).$$

Here, the δ variables are called *Lagrange multipliers*. Each of them is associated with a constraint, and they allow us to move the hard consistency constraints into the optimization objective. The Lagrangian is a standard technique from constrained optimization that converts a constrained problem into an unconstrained one via these multipliers. Intuitively, each multiplier $\delta_{fi}(x'_i)$ adjusts the objective to penalize disagreement between the local variable copies x_i^f and the global variable x_i. By optimizing the Lagrangian over both the original variables and the multipliers, dual decomposition seeks a solution that approximately satisfies the original constraints while breaking the problem into smaller, tractable components.

Observe that $\mathbf{x}, \mathbf{x}^f = \mathbf{x}^*$ is a valid assignment to the Lagrangian. Its value equals p^* for any δ, since the Lagrange multipliers are simply multiplied by zero. This shows that the Lagrangian is an upper bound on p^*:

$$L(\delta) := \max_{\mathbf{x}^f, \mathbf{x}} L(\delta, \mathbf{x}^f, \mathbf{x}) \geq p^* \ \forall \delta.$$

In order to get the tightest such bound, we may optimize $L(\delta)$ over δ. It turns out that by the theory of Lagarange duality, at the optimal δ^*, this bound will be exactly tight, i.e.,

$$L(\delta^*) = p^*.$$

It is actually not hard to prove this in our particular setting. To see that, note that we can reparametrize the Lagrangian as:

$$\begin{aligned} L(\delta) &:= \sum_{i \in \mathbf{V}} \max_{x_i} \left(\theta_i(x_i) + \sum_{f : i \in f} \delta_{fi}(x_i) \right) + \sum_{f \in \mathcal{F}} \max_{\mathbf{x}^f} \left(\theta_f(\mathbf{x}^f) + \sum_{i \in f} \delta_{fi}(x_i) \right) \\ &= \sum_{i \in \mathbf{V}} \max_{x_i} \bar{\theta}_i^{\delta}(x_i) + \sum_{f \in \mathcal{F}} \max_{\mathbf{x}^f} \bar{\theta}_f^{\delta}(\mathbf{x}^f). \end{aligned}$$

Suppose we can find dual variables $\bar{\delta}$ such that the local maximizers of $\bar{\theta}_i^{\bar{\delta}}(x_i)$ and $\bar{\theta}_f^{\bar{\delta}}(\mathbf{x}^f)$ agree. In other words, we can find a $\bar{\mathbf{x}}$ such that

$\bar{x}_i \in \arg\max_{x_i} \bar{\theta}_i^{\bar{\delta}}(x_i)$ and $\bar{\mathbf{x}}^f \in \arg\max_{\mathbf{x}^f} \bar{\theta}_f^{\bar{\delta}}(\mathbf{x}^f)$. Then we have that

$$L(\bar{\delta}) = \sum_{i \in \mathbf{V}} \bar{\theta}_i^{\bar{\delta}}(\bar{x}_i) + \sum_{f \in \mathcal{F}} \bar{\theta}_f^{\bar{\delta}}(\bar{\mathbf{x}}^f) = \sum_{i \in \mathbf{V}} \theta_i(\bar{x}_i) + \sum_{f \in \mathcal{F}} \theta_f(\bar{\mathbf{x}}^f).$$

The first equality follows by definition of $L(\delta)$, while the second follows because terms involving Lagrange multipliers cancel out when $\mathbf{x}$ and $\mathbf{x}^f$ agree.

On the other hand, we have by the definition of p^* that

$$\sum_{i \in \mathbf{V}} \theta_i(\bar{x}_i) + \sum_{f \in \mathcal{F}} \theta_f(\bar{\mathbf{x}}^f) \leq p^* \leq L(\bar{\delta})$$

which implies that $L(\bar{\delta}) = p^*$.

This argument has shown two things:

1. The bound given by the Lagrangian can be made tight for the right choice of δ.

2. To compute p^*, it suffices to find a δ at which the local subproblems agree with each other. This happens surprisingly often in practice.

Minimizing the Objective

There exist several ways of computing $L(\delta^*)$, of which we will give a brief overview.

Since the objective $L(\delta)$ is continuous and convex, we may minimize it using subgradient descent. Let

$$\bar{x}_i \in \operatorname*{argmax}_{x_i} \bar{\theta}_i^{\delta}(x_i)$$

and let

$$\bar{\mathbf{x}}^f \in \operatorname*{argmax}_{\mathbf{x}^f} \bar{\theta}_f^{\delta}(\mathbf{x}^f).$$

It can be shown that the gradient $g_{fi}(x_i)$ of $L(\delta)$ w.r.t. $\delta_{fi}(x_i)$ equals 1 if $\bar{x}_i^f \neq \bar{x}_i$ and zero otherwise; similarly, $g_{fi}(x_i^f)$ equals -1 if $\bar{x}_i^f \neq \bar{x}_i$ and zero otherwise. This expression has the effect of decreasing $\bar{\theta}_i^{\delta}(\bar{x}_i)$ and increasing $\bar{\theta}_f^{\delta}(\bar{\mathbf{x}}^f)$, thus bringing them closer to each other.

To compute these gradients we need to perform the operations $\bar{x}_i \in \operatorname{argmax}_{x_i} \bar{\theta}_i^\delta(x_i)$ and $\bar{\mathbf{x}}^f \in \operatorname{argmax}_{\mathbf{x}^f} \bar{\theta}_f^\delta(\mathbf{x}^f)$. This is possible if the scope of the factors is small, if the graph has small treewidth, if the factors are constant on most of their domain, and in many other useful special cases.

An alternative way of minimizing $L(\delta)$ is via block coordinate descent. A typical way of forming blocks is to consider all the variables $\delta_{fi}(x_i)$ associated with a fixed factor f. This results in updates that are very similar to loopy max-product belief propagation. In practice, this method may be faster than subgradient descent, is guaranteed to decrease the objective at every step, and does not require tuning a step-size parameter. Its drawback is that it does not find the global minimum (since the objective is not *strongly* convex).

Recovering the MAP Assignment

As we have seen above, if a solution $\mathbf{x}, \mathbf{x}^f$ agrees on the factors for some δ, then we can guarantee that this solution is optimal.

If the optimal $\mathbf{x}, \mathbf{x}^f$ do not agree, finding the MAP assignment from this solution is still NP-hard. However, this is usually not a big problem in practice. From the point of view of theoretical guarantees, if each $\bar{\theta}_i^{\delta^*}$ has a unique maximum, then the problem will be decodable. If this guarantee is not met by all variables, we can clamp the ones that can be uniquely decoded to their optimal values and use exact inference to find the remaining variables' values.

5.3.3 Other Methods

Local Search

A more heuristic-type solution consists in starting with an arbitrary assignment and perform "moves" on the joint assignment that locally increase the probability. This technique has no guarantees. However, we can often use prior knowledge to come up with highly effective moves. Therefore, in practice, local search may perform extremely well.

Branch and Bound

Alternatively, one may perform exhaustive search over the space of assignments, while pruning branches that can be provably shown not to contain a MAP assignment. The LP relaxation or its dual can be used to obtain upper bounds useful for pruning trees.

Simulated Annealing

A third approach is to use sampling methods (e.g., Metropolis-Hastings) to sample from $p_t(\mathbf{x}) \propto \exp(\frac{1}{t}\sum_{c\in\mathbf{C}}\theta_c(\mathbf{x}_c))$. The parameter T is called the temperature. When $t \to \infty$, p_t is close to the uniform distribution, which is easy to sample from. As $t \to 0$, p_t places more weight on $\arg\max_{\mathbf{x}}\sum_{c\in\mathbf{C}}\theta_c(\mathbf{x}_c)$, the quantity we want to recover. However, since this distribution is highly peaked, it is also very difficult to sample from.

The idea of simulated annealing is to run a sampling algorithm starting with a high T, and gradually decrease it, as the algorithm is being run. If the "cooling rate" is sufficiently slow, we are guaranteed to eventually find the mode of our distribution. In practice, however, choosing the rate requires a lot of tuning. This makes simulated annealing somewhat difficult to use in practice.

Further Reading

Approximate MAP Inference

- Chapter 8 in S. Sra *et al.* (2011). *Optimization for Machine Learning*. MIT Press.
- Chapter 13 in D. Koller and N. Friedman. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.
- M. J. Wainwright and M. I. Jordan. (2008). "Graphical Models, Exponential Families, and Variational Inference". *Foundations and Trends® in Machine Learning*. 1(1): 1–305. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000001.