Chapter 17

Monte Carlo Methods

Randomized algorithms fall into two rough categories: Las Vegas algorithms and Monte Carlo algorithms. Las Vegas algorithms always return precisely the correct answer (or report that they failed). These algorithms consume a random amount of resources, usually memory or time. In contrast, Monte Carlo algorithms return answers with a random amount of error. The amount of error can typically be reduced by expending more resources (usually running time and memory). For any fixed computational budget, a Monte Carlo algorithm can provide an approximate answer.

Many problems in machine learning are so difficult that we can never expect to obtain precise answers to them. This excludes precise deterministic algorithms and Las Vegas algorithms. Instead, we must use deterministic approximate algorithms or Monte Carlo approximations. Both approaches are ubiquitous in machine learning. In this chapter, we focus on Monte Carlo methods.

17.1 Sampling and Monte Carlo Methods

Many important technologies used to accomplish machine learning goals are based on drawing samples from some probability distribution and using these samples to form a Monte Carlo estimate of some desired quantity.

17.1.1 Why Sampling?

There are many reasons that we may wish to draw samples from a probability distribution. Sampling provides a flexible way to approximate many sums and

integrals at reduced cost. Sometimes we use this to provide a significant speedup to a costly but tractable sum, as in the case when we subsample the full training cost with minibatches. In other cases, our learning algorithm requires us to approximate an intractable sum or integral, such as the gradient of the log partition function of an undirected model. In many other cases, sampling is actually our goal, in the sense that we want to train a model that can sample from the training distribution.

17.1.2 Basics of Monte Carlo Sampling

When a sum or an integral cannot be computed exactly (for example the sum has an exponential number of terms and no exact simplification is known) it is often possible to approximate it using Monte Carlo sampling. The idea is to view the sum or integral as if it was an expectation under some distribution and to *approximate the expectation by a corresponding average*. Let

$$s = \sum_{\boldsymbol{x}} p(\boldsymbol{x}) f(\boldsymbol{x}) = E_p[f(\mathbf{x})] \tag{17.1}$$

or

$$s = \int p(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x} = E_p[f(\mathbf{x})] \tag{17.2}$$

be the sum or integral to estimate, rewritten as an expectation, with the constraint that p is a probability distribution (for the sum) or a probability density (for the integral) over random variable $\mathbf{x}$.

We can approximate s by drawing n samples $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}$ from p and then forming the empirical average

$$\hat{s}_n = \frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{x}^{(i)}). \tag{17.3}$$

This approximation is justified by a few different properties. The first trivial observation is that the estimator $\hat{s}$ is unbiased, since

$$\mathbb{E}[\hat{s}_n] = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}[f(\boldsymbol{x}^{(i)})] = \frac{1}{n} \sum_{i=1}^{n} s = s. \tag{17.4}$$

But in addition, the **law of large numbers** states that if the samples $\boldsymbol{x}^{(i)}$ are i.i.d., then the average converges almost surely to the expected value:

$$\lim_{n \to \infty} \hat{s}_n = s, \tag{17.5}$$

provided that the variance of the individual terms, $\text{Var}[f(\boldsymbol{x}^{(i)})]$, is bounded. To see this more clearly, consider the variance of $\hat{s}_n$ as n increases. The variance $\text{Var}[\hat{s}_n]$ decreases and converges to 0, so long as $\text{Var}[f(\mathbf{x}^{(i)})] <$:

$$\text{Var}[\hat{s}_n] = \frac{1}{n^2} \sum_{i=1}^{n} \text{Var}[f(\mathbf{x})] \tag{17.6}$$

$$= \frac{\text{Var}[f(\mathbf{x})]}{n}. \tag{17.7}$$

This convenient result also tells us how to estimate the uncertainty in a Monte Carlo average or equivalently the amount of expected error of the Monte Carlo approximation. We compute both the empirical average of the $f(\boldsymbol{x}^{(i)})$ and their empirical variance, and then divide the estimated variance by the number of samples n to obtain an estimator of $\text{Var}[\hat{s}_n]$. The **central limit theorem** tells us that the distribution of the average, $\hat{s}_n$, converges to a normal distribution with mean s and variance $\frac{\text{Var}[f(\ )]}{n}$. This allows us to estimate confidence intervals around the estimate $\hat{s}_n$, using the cumulative distribution of the normal density.

However, all this relies on our ability to easily sample from the base distribution $p(\mathbf{x})$, but doing so is not always possible. When it is not feasible to sample from p, an alternative is to use importance sampling, presented in section 17.2. A more general approach is to form a sequence of estimators that converge towards the distribution of interest. That is the approach of Monte Carlo Markov chains (section 17.3).

17.2 Importance Sampling

An important step in the decomposition of the integrand (or summand) used by the Monte Carlo method in equation 17.2 is deciding which part of the integrand should play the role the probability $p(\boldsymbol{x})$ and which part of the integrand should play the role of the quantity $f(\boldsymbol{x})$ whose expected value (under that probability distribution) is to be estimated. There is no unique decomposition because $p(\boldsymbol{x})f(\boldsymbol{x})$ can always be rewritten as

$$p(\boldsymbol{x})f(\boldsymbol{x}) = q(\boldsymbol{x})\frac{p(\boldsymbol{x})f(\boldsymbol{x})}{q(\boldsymbol{x})}, \tag{17.8}$$

where we now sample from q and average $\frac{pf}{q}$. In many cases, we wish to compute an expectation for a given p and an f, and the fact that the problem is specified

The unbiased estimator of the variance is often preferred, in which the sum of squared differences is divided by instead of .

from the start as an expectation suggests that this p and f would be a natural choice of decomposition. However, the original specification of the problem may not be the the optimal choice in terms of the number of samples required to obtain a given level of accuracy. Fortunately, the form of the optimal choice q^* can be derived easily. The optimal q^* corresponds to what is called optimal importance sampling.

Because of the identity shown in equation 17.8, any Monte Carlo estimator

$$\hat{s}_p = \frac{1}{n} \sum_{i=1,}^{n} \underset{\sim p}{} f(\boldsymbol{x}^{(i)}) \tag{17.9}$$

can be transformed into an importance sampling estimator

$$\hat{s}_q = \frac{1}{n} \sum_{i=1,}^{n} \underset{\sim q}{} \frac{p(\boldsymbol{x}^{(i)}) f(\boldsymbol{x}^{(i)})}{q(\boldsymbol{x}^{(i)})}. \tag{17.10}$$

We see readily that the expected value of the estimator does not depend on q:

$$\mathbb{E}_q[\hat{s}_q] = \mathbb{E}_q[\hat{s}_p] = s. \tag{17.11}$$

However, the variance of an importance sampling estimator can be greatly sensitive to the choice of q. The variance is given by

$$\mathrm{Var}[\hat{s}_q] = \mathrm{Var}[\frac{p(\mathbf{x}) f(\mathbf{x})}{q(\mathbf{x})}]/n. \tag{17.12}$$

The minimum variance occurs when q is

$$q^*(\boldsymbol{x}) = \frac{p(\boldsymbol{x}) f(\boldsymbol{x})}{Z}, \tag{17.13}$$

where Z is the normalization constant, chosen so that $q^*(\boldsymbol{x})$ sums or integrates to 1 as appropriate. Better importance sampling distributions put more weight where the integrand is larger. In fact, when $f(\boldsymbol{x})$ does not change sign, $\mathrm{Var}\left[\hat{s}_q\right] = 0$, meaning that *a single sample is sufficient* when the optimal distribution is used. Of course, this is only because the computation of q^* has essentially solved the original problem, so it is usually not practical to use this approach of drawing a single sample from the optimal distribution.

Any choice of sampling distribution q is valid (in the sense of yielding the correct expected value) and q^* is the optimal one (in the sense of yielding minimum variance). Sampling from q^* is usually infeasible, but other choices of q can be feasible while still reducing the variance somewhat.

Another approach is to use **biased importance sampling**, which has the advantage of not requiring normalized p or q. In the case of discrete variables, the biased importance sampling estimator is given by

$$\hat{s}_{BIS} = \frac{\sum_{i=1}^{n} \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} f(\boldsymbol{x}^{(i)})}{\sum_{i=1}^{n} \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}} \tag{17.14}$$

$$= \frac{\sum_{i=1}^{n} \frac{p(\boldsymbol{x})}{\tilde{q}(\boldsymbol{x})} f(\boldsymbol{x}^{(i)})}{\sum_{i=1}^{n} \frac{p(\boldsymbol{x})}{\tilde{q}(\boldsymbol{x})}} \tag{17.15}$$

$$= \frac{\sum_{i=1}^{n} \frac{\tilde{p}(\boldsymbol{x})}{\tilde{q}(\boldsymbol{x})} f(\boldsymbol{x}^{(i)})}{\sum_{i=1}^{n} \frac{\tilde{p}(\boldsymbol{x})}{\tilde{q}(\boldsymbol{x})}}, \tag{17.16}$$

where $\tilde{p}$ and $\tilde{q}$ are the unnormalized forms of p and q and the $\boldsymbol{x}^{(i)}$ are the samples from q. This estimator is biased because $\mathbb{E}[\hat{s}_{BIS}] = s$, except asymptotically when n and the denominator of equation 17.14 converges to 1. Hence this estimator is called asymptotically unbiased.

Although a good choice of q can greatly improve the efficiency of Monte Carlo estimation, a poor choice of q can make the efficiency much worse. Going back to equation 17.12, we see that if there are samples of q for which $\frac{p(\boldsymbol{x})|f(\boldsymbol{x})|}{q(\boldsymbol{x})}$ is large, then the variance of the estimator can get very large. This may happen when $q(\boldsymbol{x})$ is tiny while neither $p(\boldsymbol{x})$ nor $f(\boldsymbol{x})$ are small enough to cancel it. The q distribution is usually chosen to be a very simple distribution so that it is easy to sample from. When $\boldsymbol{x}$ is high-dimensional, this simplicity in q causes it to match p or pf poorly. When $q(\boldsymbol{x}^{(i)}) \quad p(\boldsymbol{x}^{(i)}) f(\boldsymbol{x}^{(i)})$, importance sampling collects useless samples (summing tiny numbers or zeros). On the other hand, when $q(\boldsymbol{x}^{(i)}) \quad p(\boldsymbol{x}^{(i)}) f(\boldsymbol{x}^{(i)})$, which will happen more rarely, the ratio can be huge. Because these latter events are rare, they may not show up in a typical sample, yielding typical underestimation of s, compensated rarely by gross overestimation. Such very large or very small numbers are typical when $\boldsymbol{x}$ is high dimensional, because in high dimension the dynamic range of joint probabilities can be very large.

In spite of this danger, importance sampling and its variants have been found very useful in many machine learning algorithms, including deep learning algorithms. For example, see the use of importance sampling to accelerate training in neural language models with a large vocabulary (section 12.4.3.3) or other neural nets with a large number of outputs. See also how importance sampling has been used to estimate a partition function (the normalization constant of a probability

distribution) in section 18.7, and to estimate the log-likelihood in deep directed models such as the variational autoencoder, in section 20.10.3. Importance sampling may also be used to improve the estimate of the gradient of the cost function used to train model parameters with stochastic gradient descent, particularly for models such as classifiers where most of the total value of the cost function comes from a small number of misclassified examples. Sampling more difficult examples more frequently can reduce the variance of the gradient in such cases (Hinton, 2006).

17.3 Markov Chain Monte Carlo Methods

In many cases, we wish to use a Monte Carlo technique but there is no tractable method for drawing exact samples from the distribution $p_{\text{model}}(\mathbf{x})$ or from a good (low variance) importance sampling distribution $q(\mathbf{x})$. In the context of deep learning, this most often happens when $p_{\text{model}}(\mathbf{x})$ is represented by an undirected model. In these cases, we introduce a mathematical tool called a **Markov chain** to approximately sample from $p_{\text{model}}(\mathbf{x})$. The family of algorithms that use Markov chains to perform Monte Carlo estimates is called **Markov chain Monte Carlo methods** (MCMC). Markov chain Monte Carlo methods for machine learning are described at greater length in Koller and Friedman (2009). The most standard, generic guarantees for MCMC techniques are only applicable when the model does not assign zero probability to any state. Therefore, it is most convenient to present these techniques as sampling from an energy-based model (EBM) $p(\boldsymbol{x}) \quad \exp(\ E(\boldsymbol{x}))$ as described in section 16.2.4. In the EBM formulation, every state is guaranteed to have non-zero probability. MCMC methods are in fact more broadly applicable and can be used with many probability distributions that contain zero probability states. However, the theoretical guarantees concerning the behavior of MCMC methods must be proven on a case-by-case basis for different families of such distributions. In the context of deep learning, it is most common to rely on the most general theoretical guarantees that naturally apply to all energy-based models.

To understand why drawing samples from an energy-based model is difficult, consider an EBM over just two variables, defining a distribution $p(\mathrm{a}, \mathrm{b})$. In order to sample a, we must draw a from $p(\mathrm{a} \quad \mathrm{b})$, and in order to sample b, we must draw it from $p(\mathrm{b} \quad \mathrm{a})$. It seems to be an intractable chicken-and-egg problem. Directed models avoid this because their graph is directed and acyclic. To perform **ancestral sampling** one simply samples each of the variables in topological order, conditioning on each variable's parents, which are guaranteed to have already been sampled (section 16.3). Ancestral sampling defines an efficient, single-pass method

of obtaining a sample.

In an EBM, we can avoid this chicken and egg problem by sampling using a Markov chain. The core idea of a Markov chain is to have a state $\boldsymbol{x}$ that begins as an arbitrary value. Over time, we randomly update $\boldsymbol{x}$ repeatedly. Eventually $\boldsymbol{x}$ becomes (very nearly) a fair sample from $p(\boldsymbol{x})$. Formally, a Markov chain is defined by a random state $\boldsymbol{x}$ and a transition distribution $T(\boldsymbol{x}' \mid \boldsymbol{x})$ specifying the probability that a random update will go to state $\boldsymbol{x}'$ if it starts in state $\boldsymbol{x}$. Running the Markov chain means repeatedly updating the state $\boldsymbol{x}$ to a value $\boldsymbol{x}'$ sampled from $T(\mathbf{x}' \mid \boldsymbol{x})$.

To gain some theoretical understanding of how MCMC methods work, it is useful to reparametrize the problem. First, we restrict our attention to the case where the random variable $\mathbf{x}$ has countably many states. We can then represent the state as just a positive integer x. Different integer values of x map back to different states $\boldsymbol{x}$ in the original problem.

Consider what happens when we run infinitely many Markov chains in parallel. All of the states of the different Markov chains are drawn from some distribution $q^{(t)}(x)$, where t indicates the number of time steps that have elapsed. At the beginning, $q^{(0)}$ is some distribution that we used to arbitrarily initialize x for each Markov chain. Later, $q^{(t)}$ is influenced by all of the Markov chain steps that have run so far. Our goal is for $q^{(t)}(x)$ to converge to $p(x)$.

Because we have reparametrized the problem in terms of positive integer x, we can describe the probability distribution q using a vector $\boldsymbol{v}$, with

$$q(\mathrm{x} = i) = v_i. \tag{17.17}$$

Consider what happens when we update a single Markov chain's state x to a new state x'. The probability of a single state landing in state x' is given by

$$q^{(t+1)}(x') = \sum_x q^{(t)}(x) T(x' \mid x). \tag{17.18}$$

Using our integer parametrization, we can represent the effect of the transition operator T using a matrix $\boldsymbol{A}$. We define $\boldsymbol{A}$ so that

$$A_{i,j} = T(\mathbf{x}' = i \mid \mathbf{x} = j). \tag{17.19}$$

Using this definition, we can now rewrite equation 17.18. Rather than writing it in terms of q and T to understand how a single state is updated, we may now use $\boldsymbol{v}$ and $\boldsymbol{A}$ to describe how the entire distribution over all the different Markov chains (running in parallel) shifts as we apply an update:

$$\boldsymbol{v}^{(t)} = \boldsymbol{A}\boldsymbol{v}^{(t-1)}. \tag{17.20}$$

Applying the Markov chain update repeatedly corresponds to multiplying by the matrix $\boldsymbol{A}$ repeatedly. In other words, we can think of the process as exponentiating the matrix $\boldsymbol{A}$:

$$\boldsymbol{v}^{(t)} = \boldsymbol{A}^t \boldsymbol{v}^{(0)}. \tag{17.21}$$

The matrix $\boldsymbol{A}$ has special structure because each of its columns represents a probability distribution. Such matrices are called **stochastic matrices**. If there is a non-zero probability of transitioning from any state x to any other state x' for some power t, then the Perron-Frobenius theorem (Perron, 1907; Frobenius, 1908) guarantees that the largest eigenvalue is real and equal to 1. Over time, we can see that all of the eigenvalues are exponentiated:

$$\boldsymbol{v}^{(t)} = \left(\boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}\right)^t \boldsymbol{v}^{(0)} = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})^t \boldsymbol{V}^{-1}\boldsymbol{v}^{(0)}. \tag{17.22}$$

This process causes all of the eigenvalues that are not equal to 1 to decay to zero. Under some additional mild conditions, $\boldsymbol{A}$ is guaranteed to have only one eigenvector with eigenvalue 1. The process thus converges to a **stationary distribution**, sometimes also called the **equilibrium distribution**. At convergence,

$$\boldsymbol{v}' = \boldsymbol{A}\boldsymbol{v} = \boldsymbol{v}, \tag{17.23}$$

and this same condition holds for every additional step. This is an eigenvector equation. To be a stationary point, $\boldsymbol{v}$ must be an eigenvector with corresponding eigenvalue 1. This condition guarantees that once we have reached the stationary distribution, repeated applications of the transition sampling procedure do not change the *distribution* over the states of all the various Markov chains (although transition operator does change each individual state, of course).

If we have chosen T correctly, then the stationary distribution q will be equal to the distribution p we wish to sample from. We will describe how to choose T shortly, in section 17.4.

Most properties of Markov Chains with countable states can be generalized to continuous variables. In this situation, some authors call the Markov Chain a **Harris chain** but we use the term Markov Chain to describe both conditions. In general, a Markov chain with transition operator T will converge, under mild conditions, to a fixed point described by the equation

$$q'(\mathbf{x}') = \mathbb{E}_{\mathbf{x}\sim q} T(\mathbf{x}' \mid \mathbf{x}), \tag{17.24}$$

which in the discrete case is just rewriting equation 17.23. When $\mathbf{x}$ is discrete, the expectation corresponds to a sum, and when $\mathbf{x}$ is continuous, the expectation corresponds to an integral.

Regardless of whether the state is continuous or discrete, all Markov chain methods consist of repeatedly applying stochastic updates until eventually the state begins to yield samples from the equilibrium distribution. Running the Markov chain until it reaches its equilibrium distribution is called "**burning in**" the Markov chain. After the chain has reached equilibrium, a sequence of infinitely many samples may be drawn from from the equilibrium distribution. They are identically distributed but any two successive samples will be highly correlated with each other. A finite sequence of samples may thus not be very representative of the equilibrium distribution. One way to mitigate this problem is to return only every n successive samples, so that our estimate of the statistics of the equilibrium distribution is not as biased by the correlation between an MCMC sample and the next several samples. Markov chains are thus expensive to use because of the time required to burn in to the equilibrium distribution and the time required to transition from one sample to another reasonably decorrelated sample after reaching equilibrium. If one desires truly independent samples, one can run multiple Markov chains in parallel. This approach uses extra parallel computation to eliminate latency. The strategy of using only a single Markov chain to generate all samples and the strategy of using one Markov chain for each desired sample are two extremes; deep learning practitioners usually use a number of chains that is similar to the number of examples in a minibatch and then draw as many samples as are needed from this fixed set of Markov chains. A commonly used number of Markov chains is 100.

Another difficulty is that we do not know in advance how many steps the Markov chain must run before reaching its equilibrium distribution. This length of time is called the **mixing time**. It is also very difficult to test whether a Markov chain has reached equilibrium. We do not have a precise enough theory for guiding us in answering this question. Theory tells us that the chain will converge, but not much more. If we analyze the Markov chain from the point of view of a matrix $\boldsymbol{A}$ acting on a vector of probabilities $\boldsymbol{v}$, then we know that the chain mixes when $\boldsymbol{A}^t$ has effectively lost all of the eigenvalues from $\boldsymbol{A}$ besides the unique eigenvalue of 1. This means that the magnitude of the second largest eigenvalue will determine the mixing time. However, in practice, we cannot actually represent our Markov chain in terms of a matrix. The number of states that our probabilistic model can visit is exponentially large in the number of variables, so it is infeasible to represent $\boldsymbol{v}$, $\boldsymbol{A}$, or the eigenvalues of $\boldsymbol{A}$. Due to these and other obstacles, we usually do not know whether a Markov chain has mixed. Instead, we simply run the Markov chain for an amount of time that we roughly estimate to be sufficient, and use heuristic methods to determine whether the chain has mixed. These heuristic methods include manually inspecting samples or measuring correlations between

successive samples.

17.4 Gibbs Sampling

So far we have described how to draw samples from a distribution $q(\boldsymbol{x})$ by repeatedly updating $\boldsymbol{x}$ $\boldsymbol{x}'$ $T(\boldsymbol{x}'$ $\boldsymbol{x})$. However, we have not described how to ensure that $q(\boldsymbol{x})$ is a useful distribution. Two basic approaches are considered in this book. The first one is to derive T from a given learned p_{model}, described below with the case of sampling from EBMs. The second one is to directly parametrize T and learn it, so that its stationary distribution implicitly defines the p_{model} of interest. Examples of this second approach are discussed in sections 20.12 and 20.13.

In the context of deep learning, we commonly use Markov chains to draw samples from an energy-based model defining a distribution $p_{\text{model}}(\boldsymbol{x})$. In this case, we want the $q(\boldsymbol{x})$ for the Markov chain to be $p_{\text{model}}(\boldsymbol{x})$. To obtain the desired $q(\boldsymbol{x})$, we must choose an appropriate $T(\boldsymbol{x}'$ $\boldsymbol{x})$.

A conceptually simple and effective approach to building a Markov chain that samples from $p_{\text{model}}(\boldsymbol{x})$ is to use **Gibbs sampling**, in which sampling from $T(\mathbf{x}'$ $\mathbf{x})$ is accomplished by selecting one variable x_i and sampling it from p_{model} conditioned on its neighbors in the undirected graph defining the structure of the energy-based model. It is also possible to sample several variables at the same time so long as they are conditionally independent given all of their neighbors. As shown in the RBM example in section 16.7.1, all of the hidden units of an RBM may be sampled simultaneously because they are conditionally independent from each other given all of the visible units. Likewise, all of the visible units may be sampled simultaneously because they are conditionally independent from each other given all of the hidden units. Gibbs sampling approaches that update many variables simultaneously in this way are called **block Gibbs sampling**.

Alternate approaches to designing Markov chains to sample from p_{model} are possible. For example, the Metropolis-Hastings algorithm is widely used in other disciplines. In the context of the deep learning approach to undirected modeling, it is rare to use any approach other than Gibbs sampling. Improved sampling techniques are one possible research frontier.

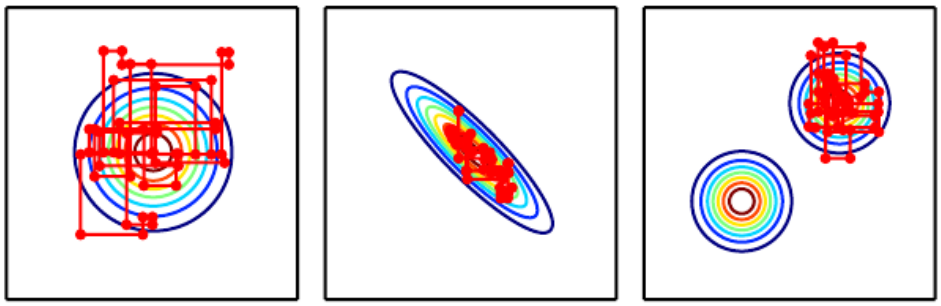
17.5 The Challenge of Mixing between Separated Modes

The primary difficulty involved with MCMC methods is that they have a tendency to **mix** poorly. Ideally, successive samples from a Markov chain designed to sample

from $p(\boldsymbol{x})$ would be completely independent from each other and would visit many different regions in $\boldsymbol{x}$ space proportional to their probability. Instead, especially in high dimensional cases, MCMC samples become very correlated. We refer to such behavior as slow mixing or even failure to mix. MCMC methods with slow mixing can be seen as inadvertently performing something resembling noisy gradient descent on the energy function, or equivalently noisy hill climbing on the probability, with respect to the state of the chain (the random variables being sampled). The chain tends to take small steps (in the space of the state of the Markov chain), from a configuration $\boldsymbol{x}^{(t-1)}$ to a configuration $\boldsymbol{x}^{(t)}$, with the energy $E(\boldsymbol{x}^{(t)})$ generally lower or approximately equal to the energy $E(\boldsymbol{x}^{(t-1)})$, with a preference for moves that yield lower energy configurations. When starting from a rather improbable configuration (higher energy than the typical ones from $p(\mathbf{x})$), the chain tends to gradually reduce the energy of the state and only occasionally move to another mode. Once the chain has found a region of low energy (for example, if the variables are pixels in an image, a region of low energy might be a connected manifold of images of the same object), which we call a mode, the chain will tend to walk around that mode (following a kind of random walk). Once in a while it will step out of that mode and generally return to it or (if it finds an escape route) move towards another mode. The problem is that successful escape routes are rare for many interesting distributions, so the Markov chain will continue to sample the same mode longer than it should.

This is very clear when we consider the Gibbs sampling algorithm (section 17.4). In this context, consider the probability of going from one mode to a nearby mode within a given number of steps. What will determine that probability is the shape of the "energy barrier" between these modes. Transitions between two modes that are separated by a high energy barrier (a region of low probability) are exponentially less likely (in terms of the height of the energy barrier). This is illustrated in figure 17.1. The problem arises when there are multiple modes with high probability that are separated by regions of low probability, especially when each Gibbs sampling step must update only a small subset of variables whose values are largely determined by the other variables.

As a simple example, consider an energy-based model over two variables a and b, which are both binary with a sign, taking on values -1 and 1. If $E(\mathrm{a}, \mathrm{b}) = -w\mathrm{ab}$ for some large positive number w, then the model expresses a strong belief that a and b have the same sign. Consider updating b using a Gibbs sampling step with $\mathrm{a} = 1$. The conditional distribution over b is given by $P(\mathrm{b} = 1 \mid \mathrm{a} = 1) = \sigma(w)$. If w is large, the sigmoid saturates, and the probability of also assigning b to be 1 is close to 1. Likewise, if $\mathrm{a} = -1$, the probability of assigning b to be -1 is close to 1. According to $P_{\text{model}}(\mathrm{a}, \mathrm{b})$, both signs of both variables are equally likely.

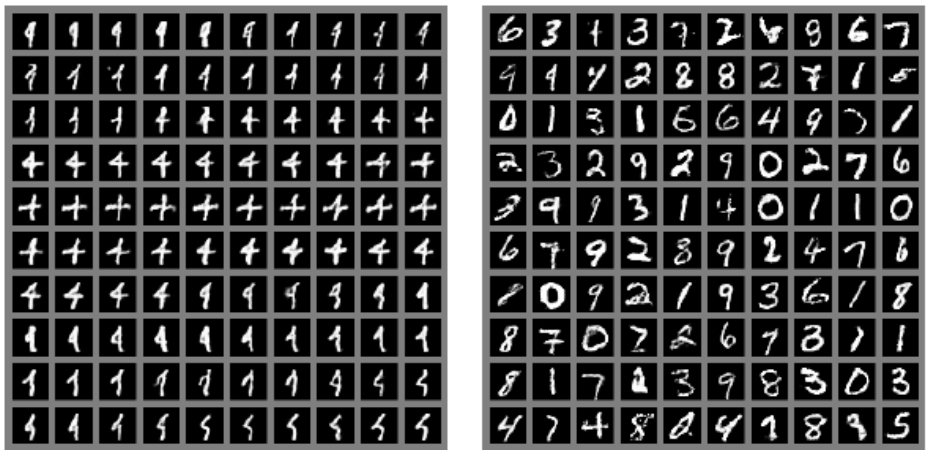


According to $P_{\text{model}}(\text{a} \quad \text{b})$, both variables should have the same sign. This means that Gibbs sampling will only very rarely flip the signs of these variables.

In more practical scenarios, the challenge is even greater because we care not only about making transitions between two modes but more generally between all the many modes that a real model might contain. If several such transitions are difficult because of the difficulty of mixing between modes, then it becomes very expensive to obtain a reliable set of samples covering most of the modes, and convergence of the chain to its stationary distribution is very slow.

Sometimes this problem can be resolved by finding groups of highly dependent units and updating all of them simultaneously in a block. Unfortunately, when the dependencies are complicated, it can be computationally intractable to draw a sample from the group. After all, the problem that the Markov chain was originally introduced to solve is this problem of sampling from a large group of variables.

In the context of models with latent variables, which define a joint distribution $p_{\text{model}}(\boldsymbol{x}, \boldsymbol{h})$, we often draw samples of $\boldsymbol{x}$ by alternating between sampling from $p_{\text{model}}(\boldsymbol{x} \quad \boldsymbol{h})$ and sampling from $p_{\text{model}}(\boldsymbol{h} \quad \boldsymbol{x})$. From the point of view of mixing



rapidly, we would like $p_{\text{model}}(\boldsymbol{h} \quad \boldsymbol{x})$ to have very high entropy. However, from the point of view of learning a useful representation of $\boldsymbol{h}$, we would like $\boldsymbol{h}$ to encode enough information about $\boldsymbol{x}$ to reconstruct it well, which implies that $\boldsymbol{h}$ and $\boldsymbol{x}$ should have very high mutual information. These two goals are at odds with each other. We often learn generative models that very precisely encode $\boldsymbol{x}$ into $\boldsymbol{h}$ but are not able to mix very well. This situation arises frequently with Boltzmann machines—the sharper the distribution a Boltzmann machine learns, the harder it is for a Markov chain sampling from the model distribution to mix well. This problem is illustrated in figure 17.2.

All this could make MCMC methods less useful when the distribution of interest has a manifold structure with a separate manifold for each class: the distribution is concentrated around many modes and these modes are separated by vast regions of high energy. This type of distribution is what we expect in many classification problems and would make MCMC methods converge very slowly because of poor mixing between modes.

17.5.1 Tempering to Mix between Modes

When a distribution has sharp peaks of high probability surrounded by regions of low probability, it is difficult to mix between the different modes of the distribution. Several techniques for faster mixing are based on constructing alternative versions of the target distribution in which the peaks are not as high and the surrounding valleys are not as low. Energy-based models provide a particularly simple way to do so. So far, we have described an energy-based model as defining a probability distribution

$$p(\boldsymbol{x}) \quad \exp\left(\quad E(\boldsymbol{x})\right). \tag{17.25}$$

Energy-based models may be augmented with an extra parameter β controlling how sharply peaked the distribution is:

$$p_\beta(\boldsymbol{x}) \quad \exp\left(\quad \beta E(\boldsymbol{x})\right). \tag{17.26}$$

The β parameter is often described as being the reciprocal of the **temperature**, reflecting the origin of energy-based models in statistical physics. When the temperature falls to zero and β rises to infinity, the energy-based model becomes deterministic. When the temperature rises to infinity and β falls to zero, the distribution (for discrete $\boldsymbol{x}$) becomes uniform.

Typically, a model is trained to be evaluated at $\beta = 1$. However, we can make use of other temperatures, particularly those where $\beta < 1$. **Tempering** is a general strategy of mixing between modes of p_1 rapidly by drawing samples with $\beta < 1$.

Markov chains based on **tempered transitions** (Neal, 1994) temporarily sample from higher-temperature distributions in order to mix to different modes, then resume sampling from the unit temperature distribution. These techniques have been applied to models such as RBMs (Salakhutdinov, 2010). Another approach is to use **parallel tempering** (Iba, 2001), in which the Markov chain simulates many different states in parallel, at different temperatures. The highest temperature states mix slowly, while the lowest temperature states, at temperature 1, provide accurate samples from the model. The transition operator includes stochastically swapping states between two different temperature levels, so that a sufficiently high-probability sample from a high-temperature slot can jump into a lower temperature slot. This approach has also been applied to RBMs (Desjardins *et al.*, 2010; Cho *et al.*, 2010). Although tempering is a promising approach, at this point it has not allowed researchers to make a strong advance in solving the challenge of sampling from complex EBMs. One possible reason is that there are **critical temperatures** around which the temperature transition must be very slow (as the temperature is gradually reduced) in order for tempering to be effective.

17.5.2 Depth May Help Mixing

When drawing samples from a latent variable model $p(\boldsymbol{h}, \boldsymbol{x})$, we have seen that if $p(\boldsymbol{h} \mid \boldsymbol{x})$ encodes $\boldsymbol{x}$ too well, then sampling from $p(\boldsymbol{x} \mid \boldsymbol{h})$ will not change $\boldsymbol{x}$ very much and mixing will be poor. One way to resolve this problem is to make $\boldsymbol{h}$ be a deep representation, that encodes $\boldsymbol{x}$ into $\boldsymbol{h}$ in such a way that a Markov chain in the space of $\boldsymbol{h}$ can mix more easily. Many representation learning algorithms, such as autoencoders and RBMs, tend to yield a marginal distribution over $\boldsymbol{h}$ that is more uniform and more unimodal than the original data distribution over $\boldsymbol{x}$. It can be argued that this arises from trying to minimize reconstruction error while using all of the available representation space, because minimizing reconstruction error over the training examples will be better achieved when different training examples are easily distinguishable from each other in $\boldsymbol{h}$-space, and thus well separated. Bengio *et al.* (2013a) observed that deeper stacks of regularized autoencoders or RBMs yield marginal distributions in the top-level $\boldsymbol{h}$-space that appeared more spread out and more uniform, with less of a gap between the regions corresponding to different modes (categories, in the experiments). Training an RBM in that higher-level space allowed Gibbs sampling to mix faster between modes. It remains however unclear how to exploit this observation to help better train and sample from deep generative models.

Despite the difficulty of mixing, Monte Carlo techniques are useful and are often the best tool available. Indeed, they are the primary tool used to confront the intractable partition function of undirected models, discussed next.