Chapter 18

Confronting the Partition Function

In Sec. 16.2.2 we saw that many probabilistic models (commonly known as undirected graphical models) are defined by an unnormalized probability distribution $\tilde{p}(\mathbf{x}; \theta)$. We must normalize $\tilde{p}$ by dividing by a partition function $Z(\boldsymbol{\theta})$ in order to obtain a valid probability distribution:

$$p(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \tilde{p}(\mathbf{x}; \boldsymbol{\theta}). \tag{18.1}$$

The partition function is an integral (for continuous variables) or sum (for discrete variables) over the unnormalized probability of all states:

$$\int \tilde{p}(\boldsymbol{x}) d\boldsymbol{x} \tag{18.2}$$

or

$$\sum_{\boldsymbol{x}} \tilde{p}(\boldsymbol{x}). \tag{18.3}$$

This operation is intractable for many interesting models.

As we will see in Chapter 20, several deep learning models are designed to have a tractable normalizing constant, or are designed to be used in ways that do not involve computing $p(\mathbf{x})$ at all. However, other models directly confront the challenge of intractable partition functions. In this chapter, we describe techniques used for training and evaluating models that have intractable partition functions.

18.1 The Log-Likelihood Gradient

What makes learning undirected models by maximum likelihood particularly difficult is that the partition function depends on the parameters. The gradient of the log-likelihood with respect to the parameters has a term corresponding to the gradient of the partition function:

$$\nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}; \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}; \boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} \log Z(\boldsymbol{\theta}). \tag{18.4}$$

This is a well-known decomposition into the *positive phase* and *negative phase* of learning.

For most undirected models of interest, the negative phase is difficult. Models with no latent variables or with few interactions between latent variables typically have a tractable positive phase. The quintessential example of a model with a straightforward positive phase and difficult negative phase is the RBM, which has hidden units that are conditionally independent from each other given the visible units. The case where the positive phase is difficult, with complicated interactions between latent variables, is primarily covered in Chapter 19. This chapter focuses on the difficulties of the negative phase.

Let us look more closely at the gradient of $\log Z$:

$$\nabla_{\boldsymbol{\theta}} \log Z \tag{18.5}$$

$$= \frac{\nabla_{\boldsymbol{\theta}} Z}{Z} \tag{18.6}$$

$$= \frac{\nabla_{\boldsymbol{\theta}} \sum_{\mathbf{x}} \tilde{p}(\mathbf{x})}{Z} \tag{18.7}$$

$$= \frac{\sum_{\mathbf{x}} \nabla_{\boldsymbol{\theta}} \tilde{p}(\mathbf{x})}{Z}. \tag{18.8}$$

For models that guarantee $p(\mathbf{x}) > 0$ for all $\mathbf{x}$, we can substitute $\exp\left(\log \tilde{p}(\mathbf{x})\right)$ for $\tilde{p}(\mathbf{x})$:

$$\frac{\sum_{\mathbf{x}} \nabla_{\boldsymbol{\theta}} \exp\left(\log \tilde{p}(\mathbf{x})\right)}{Z} \tag{18.9}$$

$$= \frac{\sum_{\mathbf{x}} \exp\left(\log \tilde{p}(\mathbf{x})\right) \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x})}{Z} \tag{18.10}$$

$$= \frac{\sum_{\mathbf{x}} \tilde{p}(\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x})}{Z} \tag{18.11}$$

$$= \sum_{\mathbf{x}} p(\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}) \tag{18.12}$$

$$= \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}). \tag{18.13}$$

This derivation made use of summation over discrete $\boldsymbol{x}$, but a similar result applies using integration over continuous $\boldsymbol{x}$. In the continuous version of the derivation, we use Leibniz's rule for differentiation under the integral sign to obtain the identity

$$\nabla_{\boldsymbol{\theta}} \int \tilde{p}(\mathbf{x}) d\boldsymbol{x} = \int \nabla_{\boldsymbol{\theta}} \tilde{p}(\mathbf{x}) d\boldsymbol{x}. \tag{18.14}$$

This identity is applicable only under certain regularity conditions on $\tilde{p}$ and $\nabla_{\boldsymbol{\theta}} \tilde{p}(\mathbf{x})$. In measure theoretic terms, the conditions are: (i) The unnormalized distribution $\tilde{p}$ must be a Lebesgue-integrable function of $\boldsymbol{x}$ for every value of $\boldsymbol{\theta}$; (ii) The gradient $\nabla_{\boldsymbol{\theta}} \tilde{p}(\mathbf{x})$ must exist for all $\boldsymbol{\theta}$ and almost all $\boldsymbol{x}$; (iii) There must exist an integrable function $R(\boldsymbol{x})$ that bounds $\nabla_{\boldsymbol{\theta}} \tilde{p}(\mathbf{x})$ in the sense that $\max_i |\frac{\partial}{\partial \theta_i} \tilde{p}(\mathbf{x})| \leq R(\boldsymbol{x})$ for all $\boldsymbol{\theta}$ and almost all $\boldsymbol{x}$. Fortunately, most machine learning models of interest have these properties.

This identity

$$\nabla_{\boldsymbol{\theta}} \log Z = \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}) \tag{18.15}$$

is the basis for a variety of Monte Carlo methods for approximately maximizing the likelihood of models with intractable partition functions.

The Monte Carlo approach to learning undirected models provides an intuitive framework in which we can think of both the positive phase and the negative phase. In the positive phase, we increase $\log \tilde{p}(\mathbf{x})$ for $\boldsymbol{x}$ drawn from the data. In the negative phase, we decrease the partition function by decreasing $\log \tilde{p}(\mathbf{x})$ drawn from the model distribution.

In the deep learning literature, it is common to parametrize $\log \tilde{p}$ in terms of an energy function (Eq. 16.7). In this case, we can interpret the positive phase as pushing down on the energy of training examples and the negative phase as pushing up on the energy of samples drawn from the model, as illustrated in Fig. 18.1.

18.2 Stochastic Maximum Likelihood and Contrastive Divergence

The naive way of implementing Eq. 18.15 is to compute it by burning in a set of Markov chains from a random initialization every time the gradient is needed. When learning is performed using stochastic gradient descent, this means the chains must be burned in once per gradient step. This approach leads to the

training procedure presented in Algorithm 18.1. The high cost of burning in the Markov chains in the inner loop makes this procedure computationally infeasible, but this procedure is the starting point that other more practical algorithms aim to approximate.

Algorithm 18.1 A naive MCMC algorithm for maximizing the log-likelihood with an intractable partition function using gradient ascent.

Set ϵ, the step size, to a small positive number.
Set k, the number of Gibbs steps, high enough to allow burn in. Perhaps 100 to train an RBM on a small image patch.
while not converged **do**
  Sample a minibatch of m examples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ from the training set.
  $\mathbf{g} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}^{(i)}; \boldsymbol{\theta})$.
  Initialize a set of m samples $\{\tilde{\mathbf{x}}^{(1)}, \ldots, \tilde{\mathbf{x}}^{(m)}\}$ to random values (e.g., from a uniform or normal distribution, or possibly a distribution with marginals matched to the model's marginals).
  for $i = 1$ to k **do**
    for $j = 1$ to m **do**
      $\tilde{\mathbf{x}}^{(j)} \leftarrow \text{gibbs\_update}(\tilde{\mathbf{x}}^{(j)})$.
    end for
  end for
  $\mathbf{g} \leftarrow \mathbf{g} - \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\tilde{\mathbf{x}}^{(i)}; \boldsymbol{\theta})$.
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon \mathbf{g}$.
end while

We can view the MCMC approach to maximum likelihood as trying to achieve balance between two forces, one pushing up on the model distribution where the data occurs, and another pushing down on the model distribution where the model samples occur. Fig. 18.1 illustrates this process. The two forces correspond to maximizing $\log \tilde{p}$ and minimizing $\log Z$. Several approximations to the negative phase are possible. Each of these approximations can be understood as making the negative phase computationally cheaper but also making it push down in the wrong locations.

Because the negative phase involves drawing samples from the model's distribution, we can think of it as finding points that the model believes in strongly. Because the negative phase acts to reduce the probability of those points, they are generally considered to represent the model's incorrect beliefs about the world. They are frequently referred to in the literature as "hallucinations" or "fantasy particles." In fact, the negative phase has been proposed as a possible explanation

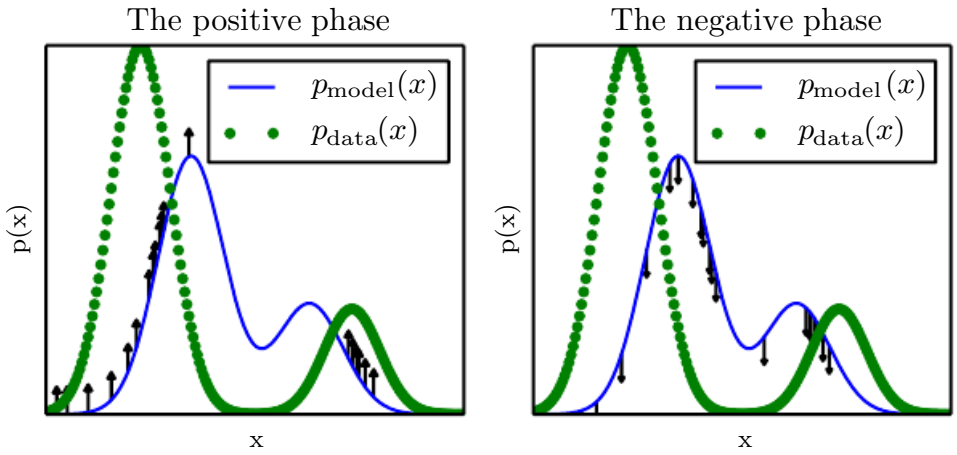

Figure 18.1: The view of Algorithm 18.1 as having a "positive phase" and "negative phase." *(Left)* In the positive phase, we sample points from the data distribution, and push up on their unnormalized probability. This means points that are likely in the data get pushed up on more. *(Right)* In the negative phase, we sample points from the model distribution, and push down on their unnormalized probability. This counteracts the positive phase's tendency to just add a large constant to the unnormalized probability everywhere. When the data distribution and the model distribution are equal, the positive phase has the same chance to push up at a point as the negative phase has to push down. When this occurs, there is no longer any gradient (in expectation) and training must terminate.

for dreaming in humans and other animals (Crick and Mitchison, 1983), the idea being that the brain maintains a probabilistic model of the world and follows the gradient of $\log \tilde{p}$ while experiencing real events while awake and follows the negative gradient of $\log \tilde{p}$ to minimize $\log Z$ while sleeping and experiencing events sampled from the current model. This view explains much of the language used to describe algorithms with a positive and negative phase, but it has not been proven to be correct with neuroscientific experiments. In machine learning models, it is usually necessary to use the positive and negative phase simultaneously, rather than in separate time periods of wakefulness and REM sleep. As we will see in Sec. 19.5, other machine learning algorithms draw samples from the model distribution for other purposes and such algorithms could also provide an account for the function of dream sleep.

Given this understanding of the role of the positive and negative phase of learning, we can attempt to design a less expensive alternative to Algorithm 18.1. The main cost of the naive MCMC algorithm is the cost of burning in the Markov chains from a random initialization at each step. A natural solution is to initialize the Markov chains from a distribution that is very close to the model distribution,

so that the burn in operation does not take as many steps.

The *contrastive divergence* (CD, or CD-k to indicate CD with k Gibbs steps) algorithm initializes the Markov chain at each step with samples from the data distribution (Hinton, 2000, 2010). This approach is presented as Algorithm 18.2. Obtaining samples from the data distribution is free, because they are already available in the data set. Initially, the data distribution is not close to the model distribution, so the negative phase is not very accurate. Fortunately, the positive phase can still accurately increase the model's probability of the data. After the positive phase has had some time to act, the model distribution is closer to the data distribution, and the negative phase starts to become accurate.

Algorithm 18.2 The contrastive divergence algorithm, using gradient ascent as the optimization procedure.

Set ϵ, the step size, to a small positive number.
Set k, the number of Gibbs steps, high enough to allow a Markov chain sampling from $p(\mathbf{x};\boldsymbol{\theta})$ to mix when initialized from p_{data}. Perhaps 1-20 to train an RBM on a small image patch.
while not converged **do**
  Sample a minibatch of m examples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ from the training set.
  $\mathbf{g} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}^{(i)};\boldsymbol{\theta})$.
  for $i = 1$ to m **do**
    $\tilde{\mathbf{x}}^{(i)} \leftarrow \mathbf{x}^{(i)}$.
  end for
  for $i = 1$ to k **do**
    for $j = 1$ to m **do**
      $\tilde{\mathbf{x}}^{(j)} \leftarrow \text{gibbs\_update}(\tilde{\mathbf{x}}^{(j)})$.
    end for
  end for
  $\mathbf{g} \leftarrow \mathbf{g} - \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\tilde{\mathbf{x}}^{(i)};\boldsymbol{\theta})$.
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon \mathbf{g}$.
end while

Of course, CD is still an approximation to the correct negative phase. The main way that CD qualitatively fails to implement the correct negative phase is that it fails to suppress regions of high probability that are far from actual training examples. These regions that have high probability under the model but low probability under the data generating distribution are called *spurious modes*. Fig. 18.2 illustrates why this happens. Essentially, it is because modes in the model distribution that are far from the data distribution will not be visited by

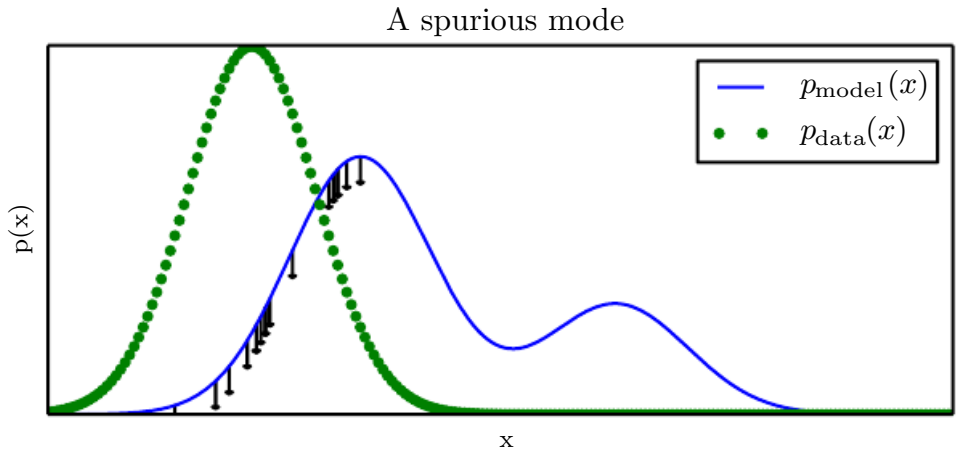

Figure 18.2: An illustration of how the negative phase of contrastive divergence (Algorithm 18.2) can fail to suppress spurious modes. A spurious mode is a mode that is present in the model distribution but absent in the data distribution. Because contrastive divergence initializes its Markov chains from data points and runs the Markov chain for only a few steps, it is unlikely to visit modes in the model that are far from the data points. This means that when sampling from the model, we will sometimes get samples that do not resemble the data. It also means that due to wasting some of its probability mass on these modes, the model will struggle to place high probability mass on the correct modes. For the purpose of visualization, this figure uses a somewhat simplified concept of distance—the spurious mode is far from the correct mode along the number line in $\mathbb{R}$. This corresponds to a Markov chain based on making local moves with a single x variable in $\mathbb{R}$. For most deep probabilistic models, the Markov chains are based on Gibbs sampling and can make non-local moves of individual variables but cannot move all of the variables simultaneously. For these problems, it is usually better to consider the edit distance between modes, rather than the Euclidean distance. However, edit distance in a high dimensional space is difficult to depict in a 2-D plot.

Markov chains initialized at training points, unless k is very large.

Carreira-Perpiñan and Hinton (2005) showed experimentally that the CD estimator is biased for RBMs and fully visible Boltzmann machines, in that it converges to different points than the maximum likelihood estimator. They argue that because the bias is small, CD could be used as an inexpensive way to initialize a model that could later be fine-tuned via more expensive MCMC methods. Bengio and Delalleau (2009) showed that CD can be interpreted as discarding the smallest terms of the correct MCMC update gradient, which explains the bias.

CD is useful for training shallow models like RBMs. These can in turn be stacked to initialize deeper models like DBNs or DBMs. However, CD does not provide much help for training deeper models directly. This is because it is difficult

to obtain samples of the hidden units given samples of the visible units. Since the hidden units are not included in the data, initializing from training points cannot solve the problem. Even if we initialize the visible units from the data, we will still need to burn in a Markov chain sampling from the distribution over the hidden units conditioned on those visible samples.

The CD algorithm can be thought of as penalizing the model for having a Markov chain that changes the input rapidly when the input comes from the data. This means training with CD somewhat resembles autoencoder training. Even though CD is more biased than some of the other training methods, it can be useful for pretraining shallow models that will later be stacked. This is because the earliest models in the stack are encouraged to copy more information up to their latent variables, thereby making it available to the later models. This should be thought of more of as an often-exploitable side effect of CD training rather than a principled design advantage.

Sutskever and Tieleman (2010) showed that the CD update direction is not the gradient of any function. This allows for situations where CD could cycle forever, but in practice this is not a serious problem.

A different strategy that resolves many of the problems with CD is to initialize the Markov chains at each gradient step with their states from the previous gradient step. This approach was first discovered under the name *stochastic maximum likelihood* (SML) in the applied mathematics and statistics community (Younes, 1998) and later independently rediscovered under the name *persistent contrastive divergence* (PCD, or PCD-k to indicate the use of k Gibbs steps per update) in the deep learning community (Tieleman, 2008). See Algorithm 18.3. The basic idea of this approach is that, so long as the steps taken by the stochastic gradient algorithm are small, then the model from the previous step will be similar to the model from the current step. It follows that the samples from the previous model's distribution will be very close to being fair samples from the current model's distribution, so a Markov chain initialized with these samples will not require much time to mix.

Because each Markov chain is continually updated throughout the learning process, rather than restarted at each gradient step, the chains are free to wander far enough to find all of the model's modes. SML is thus considerably more resistant to forming models with spurious modes than CD is. Moreover, because it is possible to store the state of all of the sampled variables, whether visible or latent, SML provides an initialization point for both the hidden and visible units. CD is only able to provide an initialization for the visible units, and therefore requires burn-in for deep models. SML is able to train deep models efficiently.

Marlin *et al.* (2010) compared SML to many of the other criteria presented in this chapter. They found that SML results in the best test set log-likelihood for an RBM, and that if the RBM's hidden units are used as features for an SVM classifier, SML results in the best classification accuracy.

SML is vulnerable to becoming inaccurate if the stochastic gradient algorithm can move the model faster than the Markov chain can mix between steps. This can happen if k is too small or ϵ is too large. The permissible range of values is unfortunately highly problem-dependent. There is no known way to test formally whether the chain is successfully mixing between steps. Subjectively, if the learning rate is too high for the number of Gibbs steps, the human operator will be able to observe that there is much more variance in the negative phase samples across gradient steps rather than across different Markov chains. For example, a model trained on MNIST might sample exclusively 7s on one step. The learning process will then push down strongly on the mode corresponding to 7s, and the model might sample exclusively 9s on the next step.

Algorithm 18.3 The stochastic maximum likelihood / persistent contrastive divergence algorithm using gradient ascent as the optimization procedure.

Set ϵ, the step size, to a small positive number.
Set k, the number of Gibbs steps, high enough to allow a Markov chain sampling from $p(\mathbf{x}; \boldsymbol{\theta} + \epsilon \mathbf{g})$ to burn in, starting from samples from $p(\mathbf{x}; \boldsymbol{\theta})$. Perhaps 1 for RBM on a small image patch, or 5-50 for a more complicated model like a DBM.
Initialize a set of m samples $\{\tilde{\mathbf{x}}^{(1)}, \ldots, \tilde{\mathbf{x}}^{(m)}\}$ to random values (e.g., from a uniform or normal distribution, or possibly a distribution with marginals matched to the model's marginals).
while not converged **do**
  Sample a minibatch of m examples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ from the training set.
  $\mathbf{g} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\mathbf{x}^{(i)}; \boldsymbol{\theta})$.
  for $i = 1$ to k **do**
    for $j = 1$ to m **do**
      $\tilde{\mathbf{x}}^{(j)} \leftarrow \text{gibbs\_update}(\tilde{\mathbf{x}}^{(j)})$.
    end for
  end for
  $\mathbf{g} \leftarrow \mathbf{g} - \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \log \tilde{p}(\tilde{\mathbf{x}}^{(i)}; \boldsymbol{\theta})$.
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon \mathbf{g}$.
end while

Care must be taken when evaluating the samples from a model trained with SML. It is necessary to draw the samples starting from a fresh Markov chain

initialized from a random starting point after the model is done training. The samples present in the persistent negative chains used for training have been influenced by several recent versions of the model, and thus can make the model appear to have greater capacity than it actually does.

Berglund and Raiko (2013) performed experiments to examine the bias and variance in the estimate of the gradient provided by CD and SML. CD proves to have lower variance than the estimator based on exact sampling. SML has higher variance. The cause of CD's low variance is its use of the same training points in both the positive and negative phase. If the negative phase is initialized from different training points, the variance rises above that of the estimator based on exact sampling.

All of these methods based on using MCMC to draw samples from the model can in principle be used with almost any variant of MCMC. This means that techniques such as SML can be improved by using any of the enhanced MCMC techniques described in Chapter 17, such as parallel tempering (Desjardins *et al.*, 2010; Cho *et al.*, 2010).

One approach to accelerating mixing during learning relies not on changing the Monte Carlo sampling technology but rather on changing the parametrization of the model and the cost function. *Fast PCD* or FPCD (Tieleman and Hinton, 2009) involves replacing the parameters $\boldsymbol{\theta}$ of a traditional model with an expression

$$\boldsymbol{\theta} = \boldsymbol{\theta}^{(\text{slow})} + \boldsymbol{\theta}^{(\text{fast})}. \tag{18.16}$$

There are now twice as many parameters as before, and they are added together element-wise to provide the parameters used by the original model definition. The fast copy of the parameters is trained with a much larger learning rate, allowing it to adapt rapidly in response to the negative phase of learning and push the Markov chain to new territory. This forces the Markov chain to mix rapidly, though this effect only occurs during learning while the fast weights are free to change. Typically one also applies significant weight decay to the fast weights, encouraging them to converge to small values, after only transiently taking on large values long enough to encourage the Markov chain to change modes.

One key benefit to the MCMC-based methods described in this section is that they provide an estimate of the gradient of $\log Z$, and thus we can essentially decompose the problem into the $\log \tilde{p}$ contribution and the $\log Z$ contribution. We can then use any other method to tackle $\log \tilde{p}(\mathbf{x})$, and just add our negative phase gradient onto the other method's gradient. In particular, this means that our positive phase can make use of methods that provide only a lower bound on $\tilde{p}$. Most of the other methods of dealing with $\log Z$ presented in this chapter are

incompatible with bound-based positive phase methods.

18.3 Pseudolikelihood

Monte Carlo approximations to the partition function and its gradient directly confront the partition function. Other approaches sidestep the issue, by training the model without computing the partition function. Most of these approaches are based on the observation that it is easy to compute ratios of probabilities in an undirected probabilistic model. This is because the partition function appears in both the numerator and the denominator of the ratio and cancels out:

$$\frac{p(\mathbf{x})}{p(\mathbf{y})} = \frac{\frac{1}{Z}\tilde{p}(\mathbf{x})}{\frac{1}{Z}\tilde{p}(\mathbf{y})} = \frac{\tilde{p}(\mathbf{x})}{\tilde{p}(\mathbf{y})}. \tag{18.17}$$

The pseudolikelihood is based on the observation that conditional probabilities take this ratio-based form, and thus can be computed without knowledge of the partition function. Suppose that we partition $\mathbf{x}$ into $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, where $\mathbf{a}$ contains the variables we want to find the conditional distribution over, $\mathbf{b}$ contains the variables we want to condition on, and $\mathbf{c}$ contains the variables that are not part of our query.

$$p(\mathbf{a} \mid \mathbf{b}) = \frac{p(\mathbf{a}, \mathbf{b})}{p(\mathbf{b})} = \frac{p(\mathbf{a}, \mathbf{b})}{\sum_{\mathbf{a},\mathbf{c}} p(\mathbf{a}, \mathbf{b}, \mathbf{c})} = \frac{\tilde{p}(\mathbf{a}, \mathbf{b})}{\sum_{\mathbf{a},\mathbf{c}} \tilde{p}(\mathbf{a}, \mathbf{b}, \mathbf{c})}. \tag{18.18}$$

This quantity requires marginalizing out $\mathbf{a}$, which can be a very efficient operation provided that $\mathbf{a}$ and $\mathbf{c}$ do not contain very many variables. In the extreme case, $\mathbf{a}$ can be a single variable and $\mathbf{c}$ can be empty, making this operation require only as many evaluations of $\tilde{p}$ as there are values of a single random variable.

Unfortunately, in order to compute the log-likelihood, we need to marginalize out large sets of variables. If there are n variables total, we must marginalize a set of size $n - 1$. By the chain rule of probability,

$$\log p(\mathbf{x}) = \log p(x_1) + \log p(x_2 \mid x_1) + \cdots + p(x_n \mid \mathbf{x}_{1:n-1}). \tag{18.19}$$

In this case, we have made $\mathbf{a}$ maximally small, but $\mathbf{c}$ can be as large as $\mathbf{x}_{2:n}$. What if we simply move $\mathbf{c}$ into $\mathbf{b}$ to reduce the computational cost? This yields the *pseudolikelihood* (Besag, 1975) objective function, based on predicting the value of feature x_i given all of the other features $\boldsymbol{x}_{-i}$:

$$\sum_{i=1}^{n} \log p(x_i \mid \boldsymbol{x}_{-i}). \tag{18.20}$$

If each random variable has k different values, this requires only $k \times n$ evaluations of $\tilde{p}$ to compute, as opposed to the k^n evaluations needed to compute the partition function.

This may look like an unprincipled hack, but it can be proven that estimation by maximizing the pseudolikelihood is asymptotically consistent (Mase, 1995). Of course, in the case of datasets that do not approach the large sample limit, pseudolikelihood may display different behavior from the maximum likelihood estimator.

It is possible to trade computational complexity for deviation from maximum likelihood behavior by using the *generalized pseudolikelihood* estimator (Huang and Ogata, 2002). The generalized pseudolikelihood estimator uses m different sets $\mathbb{S}^{(i)}, i = 1, \ldots, m$ of indices of variables that appear together on the left side of the conditioning bar. In the extreme case of $m = 1$ and $\mathbb{S}^{(1)} = 1, \ldots, n$ the generalized pseudolikelihood recovers the log-likelihood. In the extreme case of $m = n$ and $\mathbb{S}^{(i)} = \{i\}$, the generalized pseudolikelihood recovers the pseudolikelihood. The generalized pseudolikelihood objective function is given by

$$\sum_{i=1}^{m} \log p(\mathbf{x}_{\mathbb{S}^{(i)}} \mid \mathbf{x}_{-\mathbb{S}^{(i)}}). \tag{18.21}$$

The performance of pseudolikelihood-based approaches depends largely on how the model will be used. Pseudolikelihood tends to perform poorly on tasks that require a good model of the full joint $p(\mathbf{x})$, such as density estimation and sampling. However, it can perform better than maximum likelihood for tasks that require only the conditional distributions used during training, such as filling in small amounts of missing values. Generalized pseudolikelihood techniques are especially powerful if the data has regular structure that allows the $\mathbb{S}$ index sets to be designed to capture the most important correlations while leaving out groups of variables that only have negligible correlation. For example, in natural images, pixels that are widely separated in space also have weak correlation, so the generalized pseudolikelihood can be applied with each $\mathbb{S}$ set being a small, spatially localized window.

One weakness of the pseudolikelihood estimator is that it cannot be used with other approximations that provide only a lower bound on $\tilde{p}(\mathbf{x})$, such as variational inference, which will be covered in Chapter 19. This is because $\tilde{p}$ appears in the denominator. A lower bound on the denominator provides only an upper bound on the expression as a whole, and there is no benefit to maximizing an upper bound. This makes it difficult to apply pseudolikelihood approaches to deep models such as deep Boltzmann machines, since variational methods are one of the dominant approaches to approximately marginalizing out the many layers of hidden variables

that interact with each other. However, pseudolikelihood is still useful for deep learning, because it can be used to train single layer models, or deep models using approximate inference methods that are not based on lower bounds.

Pseudolikelihood has a much greater cost per gradient step than SML, due to its explicit computation of all of the conditionals. However, generalized pseudolikelihood and similar criteria can still perform well if only one randomly selected conditional is computed per example (Goodfellow *et al.*, 2013b), thereby bringing the computational cost down to match that of SML.

Though the pseudolikelihood estimator does not explicitly minimize $\log Z$, it can still be thought of as having something resembling a negative phase. The denominators of each conditional distribution result in the learning algorithm suppressing the probability of all states that have only one variable differing from a training example.

See Marlin and de Freitas (2011) for a theoretical analysis of the asymptotic efficiency of pseudolikelihood.

18.4 Score Matching and Ratio Matching

Score matching (Hyvärinen, 2005) provides another consistent means of training a model without estimating Z or its derivatives. The name score matching comes from terminology in which the derivatives of a log density with respect to its argument, $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$, are called its *score*. The strategy used by score matching is to minimize the expected squared difference between the derivatives of the model's log density with respect to the input and the derivatives of the data's log density with respect to the input:

$$L(\boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{2} ||\nabla_{\boldsymbol{x}} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}) - \nabla_{\boldsymbol{x}} \log p_{\text{data}}(\boldsymbol{x})||_2^2 \tag{18.22}$$

$$J(\boldsymbol{\theta}) = \frac{1}{2} \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})} L(\boldsymbol{x}, \boldsymbol{\theta}) \tag{18.23}$$

$$\boldsymbol{\theta}^* = \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{18.24}$$

This objective function avoids the difficulties associated with differentiating the partition function Z because Z is not a function of $\boldsymbol{x}$ and therefore $\nabla_{\mathbf{x}} Z = 0$. Initially, score matching appears to have a new difficulty: computing the score of the data distribution requires knowledge of the true distribution generating the training data, p_{data}. Fortunately, minimizing the expected value of $L(\boldsymbol{x}, \boldsymbol{\theta})$ is

equivalent to minimizing the expected value of

$$\tilde{L}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j=1}^{n} \left(\frac{\partial^2}{\partial x_j^2} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}) + \frac{1}{2} \left(\frac{\partial}{\partial x_j} \log p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}) \right)^2 \right) \tag{18.25}$$

where n is the dimensionality of $\boldsymbol{x}$.

Because score matching requires taking derivatives with respect to $\mathbf{x}$, it is not applicable to models of discrete data. However, the latent variables in the model may be discrete.

Like the pseudolikelihood, score matching only works when we are able to evaluate $\log \tilde{p}(\mathbf{x})$ and its derivatives directly. It is not compatible with methods that only provide a lower bound on $\log \tilde{p}(\mathbf{x})$, because score matching requires the derivatives and second derivatives of $\log \tilde{p}(\mathbf{x})$ and a lower bound conveys no information about its derivatives. This means that score matching cannot be applied to estimating models with complicated interactions between the hidden units, such as sparse coding models or deep Boltzmann machines. While score matching can be used to pretrain the first hidden layer of a larger model, it has not been applied as a pretraining strategy for the deeper layers of a larger model. This is probably because the hidden layers of such models usually contain some discrete variables.

While score matching does not explicitly have a negative phase, it can be viewed as a version of contrastive divergence using a specific kind of Markov chain (Hyvärinen, 2007a). The Markov chain in this case is not Gibbs sampling, but rather a different approach that makes local moves guided by the gradient. Score matching is equivalent to CD with this type of Markov chain when the size of the local moves approaches zero.

Lyu (2009) generalized score matching to the discrete case (but made an error in their derivation that was corrected by Marlin *et al.* (2010)). Marlin *et al.* (2010) found that *generalized score matching* (GSM) does not work in high dimensional discrete spaces where the observed probability of many events is 0.

A more successful approach to extending the basic ideas of score matching to discrete data is *ratio matching* (Hyvärinen, 2007b). Ratio matching applies specifically to binary data. Ratio matching consists of minimizing the average over examples of the following objective function:

$$L^{(\text{RM})}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j=1}^{n} \left(\frac{1}{1 + \frac{p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})}{p_{\text{model}}(f(\boldsymbol{x}), j); \boldsymbol{\theta})}} \right)^2, \tag{18.26}$$

where $f(\boldsymbol{x}, j)$ returns $\mathbf{x}$ with the bit at position j flipped. Ratio matching avoids the partition function using the same trick as the pseudolikelihood estimator: in a ratio of two probabilities, the partition function cancels out. Marlin *et al.* (2010) found that ratio matching outperforms SML, pseudolikelihood and GSM in terms of the ability of models trained with ratio matching to denoise test set images.

Like the pseudolikelihood estimator, ratio matching requires n evaluations of $\tilde{p}$ per data point, making its computational cost per update roughly n times higher than that of SML.

As with the pseudolikelihood estimator, ratio matching can be thought of as pushing down on all fantasy states that have only one variable different from a training example. Since ratio matching applies specifically to binary data, this means that it acts on all fantasy states within Hamming distance 1 of the data.

Ratio matching can also be useful as the basis for dealing with high-dimensional sparse data, such as word count vectors. This kind of data poses a challenge for MCMC-based methods because the data is extremely expensive to represent in dense format, yet the MCMC sampler does not yield sparse values until the model has learned to represent the sparsity in the data distribution. Dauphin and Bengio (2013) overcame this issue by designing an unbiased stochastic approximation to ratio matching. The approximation evaluates only a randomly selected subset of the terms of the objective, and does not require the model to generate complete fantasy samples.

See Marlin and de Freitas (2011) for a theoretical analysis of the asymptotic efficiency of ratio matching.

18.5 Denoising Score Matching

In some cases we may wish to regularize score matching, by fitting a distribution

$$p_{\text{smoothed}}(\boldsymbol{x}) = \int p_{\text{data}}(\boldsymbol{y}) q(\boldsymbol{x} \mid \boldsymbol{y}) d\boldsymbol{y} \tag{18.27}$$

rather than the true p_{data}. The distribution $q(\boldsymbol{x} \mid \boldsymbol{y})$ is a corruption process, usually one that forms $\boldsymbol{x}$ by adding a small amount of noise to $\boldsymbol{y}$.

Denoising score matching is especially useful because in practice we usually do not have access to the true p_{data} but rather only an empirical distribution defined by samples from it. Any consistent estimator will, given enough capacity, make p_{model} into a set of Dirac distributions centered on the training points. Smoothing by q helps to reduce this problem, at the loss of the asymptotic consistency property

described in Sec. 5.4.5. Kingma and LeCun (2010) introduced a procedure for performing regularized score matching with the smoothing distribution q being normally distributed noise.

Recall from Sec. 14.5.1 that several autoencoder training algorithms are equivalent to score matching or denoising score matching. These autoencoder training algorithms are therefore a way of overcoming the partition function problem.

18.6 Noise-Contrastive Estimation

Most techniques for estimating models with intractable partition functions do not provide an estimate of the partition function. SML and CD estimate only the gradient of the log partition function, rather than the partition function itself. Score matching and pseudolikelihood avoid computing quantities related to the partition function altogether.

Noise-contrastive estimation (NCE) (Gutmann and Hyvarinen, 2010) takes a different strategy. In this approach, the probability distribution estimated by the model is represented explicitly as

$$\log p_{\text{model}}(\mathbf{x}) = \log \tilde{p}_{\text{model}}(\mathbf{x}; \boldsymbol{\theta}) + c, \tag{18.28}$$

where c is explicitly introduced as an approximation of $-\log Z(\boldsymbol{\theta})$. Rather than estimating only $\boldsymbol{\theta}$, the noise contrastive estimation procedure treats c as just another parameter and estimates $\boldsymbol{\theta}$ and c simultaneously, using the same algorithm for both. The resulting $\log p_{\text{model}}(\mathbf{x})$ thus may not correspond exactly to a valid probability distribution, but will become closer and closer to being valid as the estimate of c improves.[1]

Such an approach would not be possible using maximum likelihood as the criterion for the estimator. The maximum likelihood criterion would choose to set c arbitrarily high, rather than setting c to create a valid probability distribution.

NCE works by reducing the unsupervised learning problem of estimating $p(\mathbf{x})$ to that of learning a probabilistic binary classifier in which one of the categories corresponds to the data generated by the model. This supervised learning problem is constructed in such a way that maximum likelihood estimation in this supervised

[1]NCE is also applicable to problems with a tractable partition function, where there is no need to introduce the extra parameter c. However, it has generated the most interest as a means of estimating models with difficult partition functions.

learning problem defines an asymptotically consistent estimator of the original problem.

Specifically, we introduce a second distribution, the *noise distribution* $p_{\text{noise}}(\mathbf{x})$. The noise distribution should be tractable to evaluate and to sample from. We can now construct a model over both $\mathbf{x}$ and a new, binary class variable y. In the new joint model, we specify that

$$p_{\text{joint}}(y = 1) = \frac{1}{2}, \tag{18.29}$$

$$p_{\text{joint}}(\mathbf{x} \mid y = 1) = p_{\text{model}}(\mathbf{x}), \tag{18.30}$$

and

$$p_{\text{joint}}(\mathbf{x} \mid y = 0) = p_{\text{noise}}(\mathbf{x}). \tag{18.31}$$

In other words, y is a switch variable that determines whether we will generate $\mathbf{x}$ from the model or from the noise distribution.

We can construct a similar joint model of training data. In this case, the switch variable determines whether we draw $\mathbf{x}$ from the *data* or from the noise distribution. Formally, $p_{\text{train}}(y = 1) = \frac{1}{2}$, $p_{\text{train}}(\mathbf{x} \mid y = 1) = p_{\text{data}}(\mathbf{x})$, and $p_{\text{train}}(\mathbf{x} \mid y = 0) = p_{\text{noise}}(\mathbf{x})$.

We can now just use standard maximum likelihood learning on the *supervised* learning problem of fitting p_{joint} to p_{train}:

$$\boldsymbol{\theta}, c = \arg\max_{\boldsymbol{\theta}, c} \mathbb{E}_{\mathbf{x},\text{y}\sim p_{\text{train}}} \log p_{\text{joint}}(y \mid \mathbf{x}). \tag{18.32}$$

The distribution p_{joint} is essentially a logistic regression model applied to the difference in log probabilities of the model and the noise distribution:

$$p_{\text{joint}}(y = 1 \mid \mathbf{x}) = \frac{p_{\text{model}}(\mathbf{x})}{p_{\text{model}}(\mathbf{x}) + p_{\text{noise}}(\mathbf{x})} \tag{18.33}$$

$$= \frac{1}{1 + \frac{p_{\text{noise}}(\mathbf{x})}{p_{\text{model}}(\mathbf{x})}} \tag{18.34}$$

$$= \frac{1}{1 + \exp\left(\log \frac{p_{\text{noise}}(\mathbf{x})}{p_{\text{model}}(\mathbf{x})}\right)} \tag{18.35}$$

$$= \sigma\left(-\log \frac{p_{\text{noise}}(\mathbf{x})}{p_{\text{model}}(\mathbf{x})}\right) \tag{18.36}$$

$$= \sigma\left(\log p_{\text{model}}(\mathbf{x}) - \log p_{\text{noise}}(\mathbf{x})\right). \tag{18.37}$$

NCE is thus simple to apply so long as $\log \tilde{p}_{\text{model}}$ is easy to back-propagate through, and, as specified above, p_{noise} is easy to evaluate (in order to evaluate p_{joint}) and sample from (in order to generate the training data).

NCE is most successful when applied to problems with few random variables, but can work well even if those random variables can take on a high number of values. For example, it has been successfully applied to modeling the conditional distribution over a word given the context of the word (Mnih and Kavukcuoglu, 2013). Though the word may be drawn from a large vocabulary, there is only one word.

When NCE is applied to problems with many random variables, it becomes less efficient. The logistic regression classifier can reject a noise sample by identifying any one variable whose value is unlikely. This means that learning slows down greatly after p_{model} has learned the basic marginal statistics. Imagine learning a model of images of faces, using unstructured Gaussian noise as p_{noise}. If p_{model} learns about eyes, it can reject almost all unstructured noise samples without having learned anything about other facial features, such as mouths.

The constraint that p_{noise} must be easy to evaluate and easy to sample from can be overly restrictive. When p_{noise} is simple, most samples are likely to be too obviously distinct from the data to force p_{model} to improve noticeably.

Like score matching and pseudolikelihood, NCE does not work if only a lower bound on $\tilde{p}$ is available. Such a lower bound could be used to construct a lower bound on $p_{\text{joint}}(y = 1 \mid \mathbf{x})$, but it can only be used to construct an upper bound on $p_{\text{joint}}(y = 0 \mid \mathbf{x})$, which appears in half the terms of the NCE objective. Likewise, a lower bound on p_{noise} is not useful, because it provides only an upper bound on $p_{\text{joint}}(y = 1 \mid \mathbf{x})$.

When the model distribution is copied to define a new noise distribution before each gradient step, NCE defines a procedure called *self-contrastive estimation*, whose expected gradient is equivalent to the expected gradient of maximum likelihood (Goodfellow, 2014). The special case of NCE where the noise samples are those generated by the model suggests that maximum likelihood can be interpreted as a procedure that forces a model to constantly learn to distinguish reality from its own evolving beliefs, while noise contrastive estimation achieves some reduced computational cost by only forcing the model to distinguish reality from a fixed baseline (the noise model).

Using the supervised task of classifying between training samples and generated samples (with the model energy function used in defining the classifier) to provide a gradient on the model was introduced earlier in various forms (Welling *et al.*, 2003b; Bengio, 2009).

Noise contrastive estimation is based on the idea that a good generative model should be able to distinguish data from noise. A closely related idea is that a good generative model should be able to generate samples that no classifier can distinguish from data. This idea yields generative adversarial networks (Sec. 20.10.4).

18.7 Estimating the Partition Function

While much of this chapter is dedicated to describing methods that avoid needing to compute the intractable partition function $Z(\boldsymbol{\theta})$ associated with an undirected graphical model, in this section we discuss several methods for directly estimating the partition function.

Estimating the partition function can be important because we require it if we wish to compute the normalized likelihood of data. This is often important in *evaluating* the model, monitoring training performance, and comparing models to each other.

For example, imagine we have two models: model $\mathcal{M}_A$ defining a probability distribution $p_A(\mathbf{x};\boldsymbol{\theta}_A) = \frac{1}{Z_A}\tilde{p}_A(\mathbf{x};\boldsymbol{\theta}_A)$ and model $\mathcal{M}_B$ defining a probability distribution $p_B(\mathbf{x};\boldsymbol{\theta}_B) = \frac{1}{Z_B}\tilde{p}_B(\mathbf{x};\boldsymbol{\theta}_B)$. A common way to compare the models is to evaluate and compare the likelihood that both models assign to an i.i.d. test dataset. Suppose the test set consists of m examples $\{\boldsymbol{x}^{(1)},\dots,\boldsymbol{x}^{(m)}\}$. If $\prod_i p_A(\mathrm{x}^{(i)};\boldsymbol{\theta}_A) > \prod_i p_B(\mathrm{x}^{(i)};\boldsymbol{\theta}_B)$ or equivalently if

$$\sum_i \log p_A(\mathrm{x}^{(i)};\boldsymbol{\theta}_A) - \sum_i \log p_B(\mathrm{x}^{(i)};\boldsymbol{\theta}_B) > 0, \tag{18.38}$$

then we say that $\mathcal{M}_A$ is a better model than $\mathcal{M}_B$ (or, at least, it is a better model of the test set), in the sense that it has a better test log-likelihood. Unfortunately, testing whether this condition holds requires knowledge of the partition function. Unfortunately, Eq. 18.38 seems to require evaluating the log probability that the model assigns to each point, which in turn requires evaluating the partition function. We can simplify the situation slightly by re-arranging Eq. 18.38 into a form where we need to know only the **ratio** of the two model's partition functions:

$$\sum_i \log p_A(\mathbf{x}^{(i)};\boldsymbol{\theta}_A) - \sum_i \log p_B(\mathbf{x}^{(i)};\boldsymbol{\theta}_B) = \sum_i \left(\log \frac{\tilde{p}_A(\mathbf{x}^{(i)};\boldsymbol{\theta}_A)}{\tilde{p}_B(\mathbf{x}^{(i)};\boldsymbol{\theta}_B)}\right) - m \log \frac{Z(\boldsymbol{\theta}_A)}{Z(\boldsymbol{\theta}_B)}. \tag{18.39}$$

We can thus determine whether $\mathcal{M}_A$ is a better model than $\mathcal{M}_B$ without knowing the partition function of either model but only their ratio. As we will see shortly,

we can estimate this ratio using importance sampling, provided that the two models are similar.

If, however, we wanted to compute the actual probability of the test data under either $\mathcal{M}_A$ or $\mathcal{M}_B$, we would need to compute the actual value of the partition functions. That said, if we knew the ratio of two partition functions, $r = \frac{Z(\boldsymbol{\theta}_B)}{Z(\boldsymbol{\theta}_A)}$, and we knew the actual value of just one of the two, say $Z(\boldsymbol{\theta}_A)$, we could compute the value of the other:

$$Z(\boldsymbol{\theta}_B) = rZ(\boldsymbol{\theta}_A) = \frac{Z(\boldsymbol{\theta}_B)}{Z(\boldsymbol{\theta}_A)} Z(\boldsymbol{\theta}_A). \tag{18.40}$$

A simple way to estimate the partition function is to use a Monte Carlo method such as simple importance sampling. We present the approach in terms of continuous variables using integrals, but it can be readily applied to discrete variables by replacing the integrals with summation. We use a proposal distribution $p_0(\mathbf{x}) = \frac{1}{Z_0}\tilde{p}_0(\mathbf{x})$ which supports tractable sampling and tractable evaluation of both the partition function Z_0 and the unnormalized distribution $\tilde{p}_0(\mathbf{x})$.

$$Z_1 = \int \tilde{p}_1(\mathbf{x})\ d\mathbf{x} \tag{18.41}$$

$$= \int \frac{p_0(\mathbf{x})}{p_0(\mathbf{x})} \tilde{p}_1(\mathbf{x})\ d\mathbf{x} \tag{18.42}$$

$$= Z_0 \int p_0(\mathbf{x}) \frac{\tilde{p}_1(\mathbf{x})}{\tilde{p}_0(\mathbf{x})}\ d\mathbf{x} \tag{18.43}$$

$$\hat{Z}_1 = \frac{Z_0}{K} \sum_{k=1}^{K} \frac{\tilde{p}_1\left(\mathbf{x}^{(k)}\right)}{\tilde{p}_0\left(\mathbf{x}^{(k)}\right)} \quad \text{s.t.} : \mathbf{x}^{(k)} \sim p_0 \tag{18.44}$$

In the last line, we make a Monte Carlo estimator, $\hat{Z}_1$, of the integral using samples drawn from $p_0(\mathbf{x})$ and then weight each sample with the ratio of the unnormalized $\tilde{p}_1$ and the proposal p_0.

We see also that this approach allows us to estimate the ratio between the partition functions as

$$\frac{1}{K} \sum_{k=1}^{K} \frac{\tilde{p}_1(\mathbf{x}^{(k)})}{\tilde{p}_0(\mathbf{x}^{(k)})} \quad \text{s.t.} : \mathbf{x}^{(k)} \sim p_0. \tag{18.45}$$

This value can then be used directly to compare two models as described in Eq. 18.39.

If the distribution p_0 is close to p_1, Eq. 18.44 can be an effective way of estimating the partition function (Minka, 2005). Unfortunately, most of the time

p_1 is both complicated (usually multimodal) and defined over a high dimensional space. It is difficult to find a tractable p_0 that is simple enough to evaluate while still being close enough to p_1 to result in a high quality approximation. If p_0 and p_1 are not close, most samples from p_0 will have low probability under p_1 and therefore make (relatively) negligible contribution to the sum in Eq. 18.44.

Having few samples with significant weights in this sum will result in an estimator that is of poor quality due to high variance. This can be understood quantitatively through an estimate of the variance of our estimate $\hat{Z}_1$:

$$\hat{\text{Var}}\left(\hat{Z}_1\right) = \frac{Z_0}{K^2}\sum_{k=1}^{K}\left(\frac{\tilde{p}_1(\mathbf{x}^{(k)})}{\tilde{p}_0(\mathbf{x}^{(k)})} - \hat{Z}_1\right)^2. \tag{18.46}$$

This quantity is largest when there is significant deviation in the values of the importance weights $\frac{\tilde{p}_1(\mathbf{x}^{(k)})}{\tilde{p}_0(\mathbf{x}^{(k)})}$.

We now turn to two related strategies developed to cope with the challenging task of estimating partition functions for complex distributions over high-dimensional spaces: annealed importance sampling and bridge sampling. Both start with the simple importance sampling strategy introduced above and both attempt to overcome the problem of the proposal p_0 being too far from p_1 by introducing intermediate distributions that attempt to *bridge the gap* between p_0 and p_1.

18.7.1 Annealed Importance Sampling

In situations where $D_{\text{KL}}(p_0\|p_1)$ is large (i.e., where there is little overlap between p_0 and p_1), a strategy called *annealed importance sampling* (AIS) attempts to bridge the gap by introducing intermediate distributions (Jarzynski, 1997; Neal, 2001). Consider a sequence of distributions $p_{\eta_0}, \ldots, p_{\eta_n}$, with $0 = \eta_0 < \eta_1 < \cdots < \eta_{n-1} < \eta_n = 1$ so that the first and last distributions in the sequence are p_0 and p_1 respectively.

This approach allows us to estimate the partition function of a multimodal distribution defined over a high-dimensional space (such as the distribution defined by a trained RBM). We begin with a simpler model with a known partition function (such as an RBM with zeroes for weights) and estimate the ratio between the two model's partition functions. The estimate of this ratio is based on the estimate of the ratios of a sequence of many similar distributions, such as the sequence of RBMs with weights interpolating between zero and the learned weights.

We can now write the ratio $\frac{Z_1}{Z_0}$ as

$$\frac{Z_1}{Z_0} = \frac{Z_1}{Z_0}\frac{Z_{\eta_1}}{Z_{\eta_1}} \cdots \frac{Z_{\eta_{n-1}}}{Z_{\eta_{n-1}}} \tag{18.47}$$

$$= \frac{Z_{\eta_1}}{Z_0}\frac{Z_{\eta_2}}{Z_{\eta_1}} \cdots \frac{Z_{\eta_{n-1}}}{Z_{\eta_{n-2}}}\frac{Z_1}{Z_{\eta_{n-1}}} \tag{18.48}$$

$$= \prod_{j=0}^{n-1} \frac{Z_{\eta_{j+1}}}{Z_{\eta_j}} \tag{18.49}$$

Provided the distributions p_{η_j} and p_{η_j+1}, for all $0 \leq j \leq n-1$, are sufficiently close, we can reliably estimate each of the factors $\frac{Z_{\eta_{j+1}}}{Z_{\eta_j}}$ using simple importance sampling and then use these to obtain an estimate of $\frac{Z_1}{Z_0}$.

Where do these intermediate distributions come from? Just as the original proposal distribution p_0 is a design choice, so is the sequence of distributions $p_{\eta_1} \ldots p_{\eta_{n-1}}$. That is, it can be specifically constructed to suit the problem domain. One general-purpose and popular choice for the intermediate distributions is to use the weighted geometric average of the target distribution p_1 and the starting proposal distribution (for which the partition function is known) p_0:

$$p_{\eta_j} \propto p_1^{\eta_j} p_0^{1-\eta_j} \tag{18.50}$$

In order to sample from these intermediate distributions, we define a series of Markov chain transition functions $T_{\eta_j}(\boldsymbol{x}' \mid \boldsymbol{x})$ that define the conditional probability distribution of transitioning to $\boldsymbol{x}'$ given we are currently at $\boldsymbol{x}$. The transition operator $T_{\eta_j}(\boldsymbol{x}' \mid \boldsymbol{x})$ is defined to leave $p_{\eta_j}(\boldsymbol{x})$ invariant:

$$p_{\eta_j}(\boldsymbol{x}) = \int p_{\eta_j}(\boldsymbol{x}') T_{\eta_j}(\boldsymbol{x} \mid \boldsymbol{x}') \, d\boldsymbol{x}' \tag{18.51}$$

These transitions may be constructed as any Markov chain Monte Carlo method (e.g., Metropolis-Hastings, Gibbs), including methods involving multiple passes through all of the random variables or other kinds of iterations.

The AIS sampling strategy is then to generate samples from p_0 and then use the transition operators to sequentially generate samples from the intermediate distributions until we arrive at samples from the target distribution p_1:

- for $k = 1 \ldots K$
  - Sample $\boldsymbol{x}_{\eta_1}^{(k)} \sim p_0(\mathbf{x})$

  - Sample $\boldsymbol{x}_{\eta_2}^{(k)} \sim T_{\eta_1}\left(\mathbf{x}_{\eta_2}^{(k)} \mid \boldsymbol{x}_{\eta_1}^{(k)}\right)$
  - ...
  - Sample $\boldsymbol{x}_{\eta_{n-1}}^{(k)} \sim T_{\eta_{n-2}}(\mathbf{x}_{\eta_{n-1}}^{(k)} \mid \boldsymbol{x}_{\eta_{n-2}}^{(k)})$
  - Sample $\boldsymbol{x}_{\eta_n}^{(k)} \sim T_{\eta_{n-1}}(\mathbf{x}_{\eta_n}^{(k)} \mid \boldsymbol{x}_{\eta_{n-1}}^{(k)})$
- end

For sample k, we can derive the importance weight by chaining together the importance weights for the jumps between the intermediate distributions given in Eq. 18.49:

$$w^{(k)} = \frac{\tilde{p}_{\eta_1}(\boldsymbol{x}_{\eta_1}^{(k)})}{\tilde{p}_0(\boldsymbol{x}_{\eta_1}^{(k)})} \frac{\tilde{p}_{\eta_2}(\boldsymbol{x}_{\eta_2}^{(k)})}{\tilde{p}_{\eta_1}(\boldsymbol{x}_{\eta_2}^{(k)})} \cdots \frac{\tilde{p}_1(\boldsymbol{x}_1^{(k)})}{\tilde{p}_{\eta_{n-1}}(\boldsymbol{x}_{\eta_n}^{(k)})}. \tag{18.52}$$

To avoid numerical issues such as overflow, it is probably best to compute $\log w^{(k)}$ by adding and subtracting log probabilities, rather than computing $w^{(k)}$ by multiplying and dividing probabilities.

With the sampling procedure thus defined and the importance weights given in Eq. 18.52, the estimate of the ratio of partition functions is given by:

$$\frac{Z_1}{Z_0} \approx \frac{1}{K} \sum_{k=1}^{K} w^{(k)} \tag{18.53}$$

In order to verify that this procedure defines a valid importance sampling scheme, we can show (Neal, 2001) that the AIS procedure corresponds to simple importance sampling on an extended state space with points sampled over the product space $[\boldsymbol{x}_{\eta_1}, \ldots, \boldsymbol{x}_{\eta_{n-1}}, \boldsymbol{x}_1]$. To do this, we define the distribution over the extended space as:

$$\tilde{p}(\boldsymbol{x}_{\eta_1}, \ldots, \boldsymbol{x}_{\eta_{n-1}}, \boldsymbol{x}_1) \tag{18.54}$$

$$= \tilde{p}_1(\boldsymbol{x}_1)\tilde{T}_{\eta_{n-1}}(\boldsymbol{x}_{\eta_{n-1}} \mid \boldsymbol{x}_1)\tilde{T}_{\eta_{n-2}}(\boldsymbol{x}_{\eta_{n-2}} \mid \boldsymbol{x}_{\eta_{n-1}}) \ldots \tilde{T}_{\eta_1}(\boldsymbol{x}_{\eta_1} \mid \boldsymbol{x}_{\eta_2}), \tag{18.55}$$

where $\tilde{T}_a$ is the reverse of the transition operator defined by T_a (via an application of Bayes' rule):

$$\tilde{T}_a(\boldsymbol{x}' \mid \boldsymbol{x}) = \frac{p_a(\boldsymbol{x}')}{p_a(\boldsymbol{x})} T_a(\boldsymbol{x} \mid \boldsymbol{x}') = \frac{\tilde{p}_a(\boldsymbol{x}')}{\tilde{p}_a(\boldsymbol{x})} T_a(\boldsymbol{x} \mid \boldsymbol{x}'). \tag{18.56}$$

Plugging the above into the expression for the joint distribution on the extended state space given in Eq. 18.55, we get:

$$\tilde{p}(\boldsymbol{x}_{\eta_1}, \ldots, \boldsymbol{x}_{\eta_{n-1}}, \boldsymbol{x}_1) \tag{18.57}$$

$$= \tilde{p}_1(\boldsymbol{x}_1)\frac{\tilde{p}_{\eta_{n-1}}(\boldsymbol{x}_{\eta_{n-1}})}{\tilde{p}_{\eta_{n-1}}(\boldsymbol{x}_1)}T_{\eta_{n-1}}(\boldsymbol{x}_1 \mid \boldsymbol{x}_{\eta_{n-1}})\prod_{i=1}^{n-2}\frac{\tilde{p}_{\eta_i}(\boldsymbol{x}_{\eta_i})}{\tilde{p}_{\eta_i}(\boldsymbol{x}_{\eta_{i+1}})}T_{\eta_i}(\boldsymbol{x}_{\eta_{i+1}} \mid \boldsymbol{x}_{\eta_i}) \tag{18.58}$$

$$= \frac{\tilde{p}_1(\boldsymbol{x}_1)}{\tilde{p}_{\eta_{n-1}}(\boldsymbol{x}_1)}T_{\eta_{n-1}}(\boldsymbol{x}_1 \mid \boldsymbol{x}_{\eta_{n-1}})\ \tilde{p}_{\eta_1}(\boldsymbol{x}_{\eta_1})\prod_{i=1}^{n-2}\frac{\tilde{p}_{\eta_{i+1}}(\boldsymbol{x}_{\eta_{i+1}})}{\tilde{p}_{\eta_i}(\boldsymbol{x}_{\eta_{i+1}})}T_{\eta_i}(\boldsymbol{x}_{\eta_{i+1}} \mid \boldsymbol{x}_{\eta_i}). \tag{18.59}$$

We now have means of generating samples from the joint proposal distribution q over the extended sample via a sampling scheme given above, with the joint distribution given by:

$$q(\boldsymbol{x}_{\eta_1}, \ldots, \boldsymbol{x}_{\eta_{n-1}}, \boldsymbol{x}_1) = p_0(\boldsymbol{x}_{\eta_1})T_{\eta_1}(\boldsymbol{x}_{\eta_2} \mid \boldsymbol{x}_{\eta_1}) \ldots T_{\eta_{n-1}}(\boldsymbol{x}_1 \mid \boldsymbol{x}_{\eta_{n-1}}). \tag{18.60}$$

We have a joint distribution on the extended space given by Eq. 18.59. Taking $q(\boldsymbol{x}_{\eta_1}, \ldots, \boldsymbol{x}_{\eta_{n-1}}, \boldsymbol{x}_1)$ as the proposal distribution on the extended state space from which we will draw samples, it remains to determine the importance weights:

$$w^{(k)} = \frac{\tilde{p}(\boldsymbol{x}_{\eta_1}, \ldots, \boldsymbol{x}_{\eta_{n-1}}, \boldsymbol{x}_1)}{q(\boldsymbol{x}_{\eta_1}, \ldots, \boldsymbol{x}_{\eta_{n-1}}, \boldsymbol{x}_1)} = \frac{\tilde{p}_1(\boldsymbol{x}_1^{(k)})}{\tilde{p}_{\eta_{n-1}}(\boldsymbol{x}_{\eta_{n-1}}^{(k)})} \cdots \frac{\tilde{p}_{\eta_2}(\boldsymbol{x}_{\eta_2}^{(k)})}{\tilde{p}_1(\boldsymbol{x}_{\eta_1}^{(k)})}\frac{\tilde{p}_{\eta_1}(\boldsymbol{x}_{\eta_1}^{(k)})}{\tilde{p}_0(\boldsymbol{x}_0^{(k)})}. \tag{18.61}$$

These weights are the same as proposed for AIS. Thus we can interpret AIS as simple importance sampling applied to an extended state and its validity follows immediately from the validity of importance sampling.

Annealed importance sampling (AIS) was first discovered by Jarzynski (1997) and then again, independently, by Neal (2001). It is currently the most common way of estimating the partition function for undirected probabilistic models. The reasons for this may have more to do with the publication of an influential paper (Salakhutdinov and Murray, 2008) describing its application to estimating the partition function of restricted Boltzmann machines and deep belief networks than with any inherent advantage the method has over the other method described below.

A discussion of the properties of the AIS estimator (e.g.. its variance and efficiency) can be found in Neal (2001).

18.7.2 Bridge Sampling

Bridge sampling Bennett (1976) is another method that, like AIS, addresses the shortcomings of importance sampling. Rather than chaining together a series of

intermediate distributions, bridge sampling relies on a single distribution p_*, known as the bridge, to interpolate between a distribution with known partition function, p_0, and a distribution p_1 for which we are trying to estimate the partition function Z_1.

Bridge sampling estimates the ratio Z_1/Z_0 as the ratio of the expected importance weights between $\tilde{p}_0$ and $\tilde{p}_*$ and between $\tilde{p}_1$ and $\tilde{p}_*$:

$$\frac{Z_1}{Z_0} \approx \sum_{k=1}^{K} \frac{\tilde{p}_*(\boldsymbol{x}_0^{(k)})}{\tilde{p}_0(\boldsymbol{x}_0^{(k)})} \Bigg/ \sum_{k=1}^{K} \frac{\tilde{p}_*(\boldsymbol{x}_1^{(k)})}{\tilde{p}_1(\boldsymbol{x}_1^{(k)})} \tag{18.62}$$

If the bridge distribution p_* is chosen carefully to have a large overlap of support with both p_0 and p_1, then bridge sampling can allow the distance between two distributions (or more formally, $D_{\mathrm{KL}}(p_0 \| p_1)$) to be much larger than with standard importance sampling.

It can be shown that the optimal bridging distribution is given by $p_*^{(opt)}(\mathbf{x}) \propto \frac{\tilde{p}_0(\boldsymbol{x})\tilde{p}_1(\boldsymbol{x})}{r\tilde{p}_0(\boldsymbol{x})+\tilde{p}_1(\boldsymbol{x})}$ where $r = Z_1/Z_0$. At first, this appears to be an unworkable solution as it would seem to require the very quantity we are trying to estimate, Z_1/Z_0. However, it is possible to start with a coarse estimate of r and use the resulting bridge distribution to refine our estimate iteratively (Neal, 2005). That is, we iteratively re-estimate the ratio and use each iteration to update the value of r.

Linked importance sampling Both AIS and bridge sampling have their advantages. If $D_{\mathrm{KL}}(p_0 \| p_1)$ is not too large (because p_0 and p_1 are sufficiently close) bridge sampling can be a more effective means of estimating the ratio of partition functions than AIS. If, however, the two distributions are too far apart for a single distribution p_* to bridge the gap then one can at least use AIS with potentially many intermediate distributions to span the distance between p_0 and p_1. Neal (2005) showed how his linked importance sampling method leveraged the power of the bridge sampling strategy to bridge the intermediate distributions used in AIS to significantly improve the overall partition function estimates.

Estimating the partition function while training While AIS has become accepted as the standard method for estimating the partition function for many undirected models, it is sufficiently computationally intensive that it remains infeasible to use during training. However, alternative strategies that have been explored to maintain an estimate of the partition function throughout training

Using a combination of bridge sampling, short-chain AIS and parallel tempering, Desjardins *et al.* (2011) devised a scheme to track the partition function of an

RBM throughout the training process. The strategy is based on the maintenance of independent estimates of the partition functions of the RBM at every temperature operating in the parallel tempering scheme. The authors combined bridge sampling estimates of the ratios of partition functions of neighboring chains (i.e. from parallel tempering) with AIS estimates across time to come up with a low variance estimate of the partition functions at every iteration of learning.

The tools described in this chapter provide many different ways of overcoming the problem of intractable partition functions, but there can be several other difficulties involved in training and using generative models. Foremost among these is the problem of intractable inference, which we confront next.