Chapter 16

Structured Probabilistic Models for Deep Learning

Deep learning draws on many modeling formalisms that researchers can use to guide their design efforts and describe their algorithms. One of these formalisms is the idea of **structured probabilistic models**. We discuss structured probabilistic models briefly in section 3.14. That brief presentation is sufficient to understand how to use structured probabilistic models as a language to describe some of the algorithms in part II. Now, in part III, structured probabilistic models are a key ingredient of many of the most important research topics in deep learning. To prepare to discuss these research ideas, in this chapter, we describe structured probabilistic models in much greater detail. This chapter is intended to be self-contained; the reader does not need to review the earlier introduction before continuing with this chapter.

A structured probabilistic model is a way of describing a probability distribution, using a graph to describe which random variables in the probability distribution interact with each other directly. Here we use "graph" in the graph theory sense—a set of vertices connected to one another by a set of edges. Because the structure of the model is defined by a graph, these models are often also referred to as **graphical models**.

The graphical models research community is large and has developed many different models, training algorithms, and inference algorithms. In this chapter, we provide basic background on some of the most central ideas of graphical models, with an emphasis on the concepts that have proved most useful to the deep learning research community. If you already have a strong background in graphical models, you may wish to skip most of this chapter. However, even a graphical model

expert may benefit from reading the final section of this chapter, section 16.7, in which we highlight some of the unique ways in which graphical models are used for deep learning algorithms. Deep learning practitioners tend to use very different model structures, learning algorithms and inference procedures than are commonly used by the rest of the graphical models research community. In this chapter, we identify these differences in preferences and explain the reasons for them.

We first describe the challenges of building large-scale probabilistic models. Next, we describe how to use a graph to describe the structure of a probability distribution. While this approach allows us to overcome many challenges, it is not without its own complications. One of the major difficulties in graphical modeling is understanding which variables need to be able to interact directly, that is, which graph structures are most suitable for a given problem. In section 16.5, we outline two approaches to resolving this difficulty by learning about the dependencies. Finally, we close with a discussion of the unique emphasis that deep learning practitioners place on specific approaches to graphical modeling, in section 16.7.

16.1 The Challenge of Unstructured Modeling

The goal of deep learning is to scale machine learning to the kinds of challenges needed to solve artificial intelligence. This means being able to understand high-dimensional data with rich structure. For example, we would like AI algorithms to be able to understand natural images,[1] audio waveforms representing speech, and documents containing multiple words and punctuation characters.

Classification algorithms can take an input from such a rich high-dimensional distribution and summarize it with a categorical label—what object is in a photo, what word is spoken in a recording, what topic a document is about. The process of classification discards most of the information in the input and produces a single output (or a probability distribution over values of that single output). The classifier is also often able to ignore many parts of the input. For example, when recognizing an object in a photo, it is usually possible to ignore the background of the photo.

It is possible to ask probabilistic models to do many other tasks. These tasks are often more expensive than classification. Some of them require producing multiple output values. Most require a complete understanding of the entire structure of the input, with no option to ignore sections of it. These tasks include the following:

[1] A **natural image** is an image that might be captured by a camera in a reasonably ordinary environment, as opposed to a synthetically rendered image, a screenshot of a web page, etc.

- **Density estimation**: Given an input $\boldsymbol{x}$, the machine learning system returns an estimate of the true density $p(\boldsymbol{x})$ under the data-generating distribution. This requires only a single output, but it also requires a complete understanding of the entire input. If even one element of the vector is unusual, the system must assign it a low probability.

- **Denoising**: Given a damaged or incorrectly observed input $\tilde{\boldsymbol{x}}$, the machine learning system returns an estimate of the original or correct $\boldsymbol{x}$. For example, the machine learning system might be asked to remove dust or scratches from an old photograph. This requires multiple outputs (every element of the estimated clean example $\boldsymbol{x}$) and an understanding of the entire input (since even one damaged area will still reveal the final estimate as being damaged).

- **Missing value imputation**: Given the observations of some elements of $\boldsymbol{x}$, the model is asked to return estimates of or a probability distribution over some or all of the unobserved elements of $\boldsymbol{x}$. This requires multiple outputs. Because the model could be asked to restore any of the elements of $\boldsymbol{x}$, it must understand the entire input.

- **Sampling**: The model generates new samples from the distribution $p(\boldsymbol{x})$. Applications include speech synthesis, that is, producing new waveforms that sound like natural human speech. This requires multiple output values and a good model of the entire input. If the samples have even one element drawn from the wrong distribution, then the sampling process is wrong.

For an example of a sampling task using small natural images, see figure 16.1.

Modeling a rich distribution over thousands or millions of random variables is a challenging task, both computationally and statistically. Suppose we wanted to model only binary variables. This is the simplest possible case, and yet already it seems overwhelming. For a small 32×32 pixel color (RGB) image, there are 2^{3072} possible binary images of this form. This number is over 10^{800} times larger than the estimated number of atoms in the universe.

In general, if we wish to model a distribution over a random vector $\mathbf{x}$ containing n discrete variables capable of taking on k values each, then the naive approach of representing $P(\mathbf{x})$ by storing a lookup table with one probability value per possible outcome requires k^n parameters!

This is not feasible for several reasons:

- *Memory—the cost of storing the representation*: For all but very small values of n and k, representing the distribution as a table will require too many values to store.

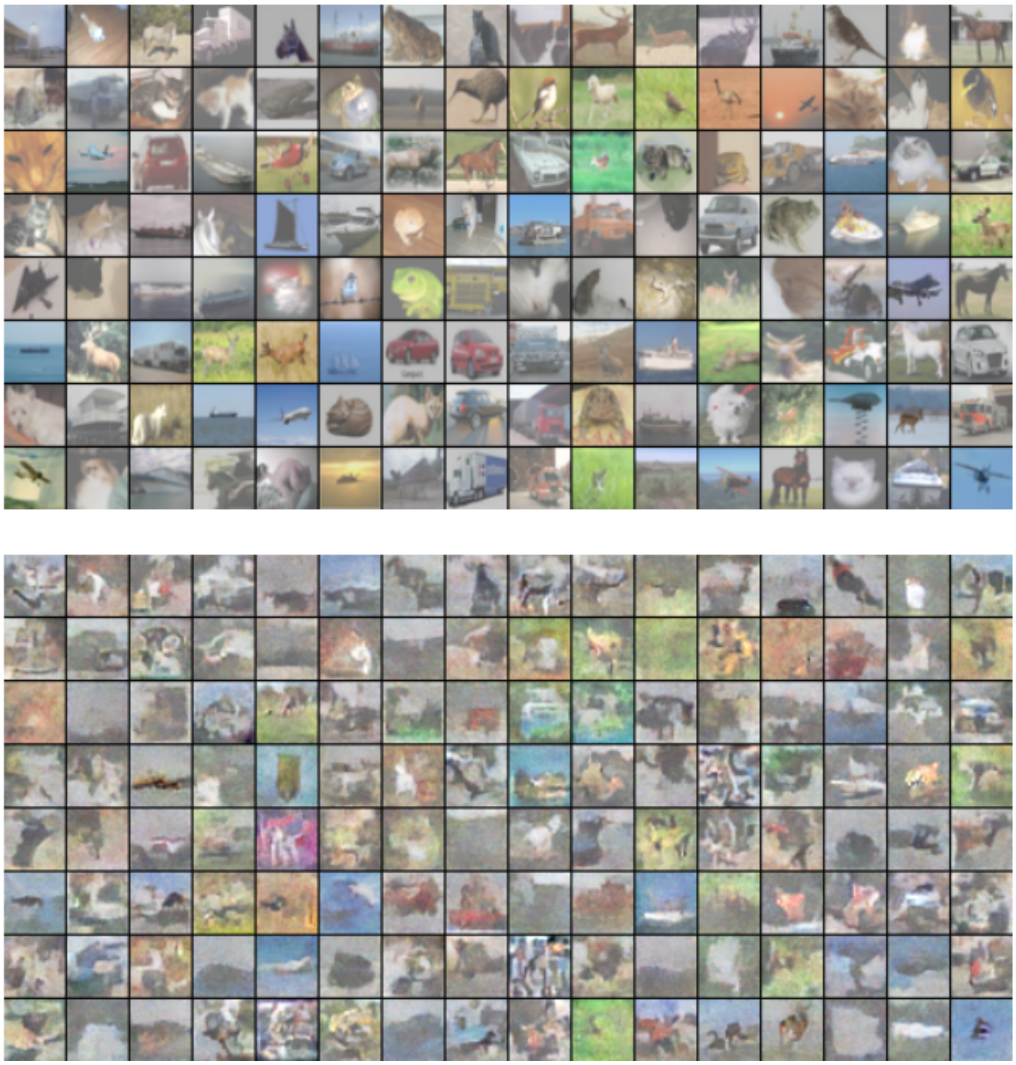

Figure 16.1: Probabilistic modeling of natural images. *(Top)* Example 32×32 pixel color images from the CIFAR-10 dataset (Krizhevsky and Hinton, 2009). *(Bottom)* Samples drawn from a structured probabilistic model trained on this dataset. Each sample appears at the same position in the grid as the training example that is closest to it in Euclidean space. This comparison allows us to see that the model is truly synthesizing new images, rather than memorizing the training data. Contrast of both sets of images has been adjusted for display. Figure reproduced with permission from Courville *et al.* (2011).

- *Statistical efficiency*: As the number of parameters in a model increases, so does the amount of training data needed to choose the values of those parameters using a statistical estimator. Because the table-based model has an astronomical number of parameters, it will require an astronomically large training set to fit accurately. Any such model will overfit the training set very badly unless additional assumptions are made linking the different entries in the table (as in back-off or smoothed n-gram models; section 12.4.1).

- *Runtime—the cost of inference*: Suppose we want to perform an inference task where we use our model of the joint distribution $P(\mathbf{x})$ to compute some other distribution, such as the marginal distribution $P(\mathrm{x}_1)$ or the conditional distribution $P(\mathrm{x}_2 \mid \mathrm{x}_1)$. Computing these distributions will require summing across the entire table, so the runtime of these operations is as high as the intractable memory cost of storing the model.

- *Runtime—the cost of sampling*: Likewise, suppose we want to draw a sample from the model. The naive way to do this is to sample some value $\mathrm{u} \sim U(0, 1)$, then iterate through the table, adding up the probability values until they exceed u and return the outcome corresponding to that position in the table. This requires reading through the whole table in the worst case, so it has the same exponential cost as the other operations.

The problem with the table-based approach is that we are explicitly modeling every possible kind of interaction between every possible subset of variables. The probability distributions we encounter in real tasks are much simpler than this. Usually, most variables influence each other only indirectly.

For example, consider modeling the finishing times of a team in a relay race. Suppose the team consists of three runners: Alice, Bob and Carol. At the start of the race, Alice carries a baton and begins running around a track. After completing her lap around the track, she hands the baton to Bob. Bob then runs his own lap and hands the baton to Carol, who runs the final lap. We can model each of their finishing times as a continuous random variable. Alice's finishing time does not depend on anyone else's, since she goes first. Bob's finishing time depends on Alice's, because Bob does not have the opportunity to start his lap until Alice has completed hers. If Alice finishes faster, Bob will finish faster, all else being equal. Finally, Carol's finishing time depends on both her teammates. If Alice is slow, Bob will probably finish late too. As a consequence, Carol will have quite a late starting time and thus is likely to have a late finishing time as well. However, Carol's finishing time depends only *indirectly* on Alice's finishing time via Bob's. If we already know Bob's finishing time, we will not be able to estimate Carol's

finishing time better by finding out what Alice's finishing time was. This means we can model the relay race using only two interactions: Alice's effect on Bob and Bob's effect on Carol. We can omit the third, indirect interaction between Alice and Carol from our model.

Structured probabilistic models provide a formal framework for modeling only direct interactions between random variables. This allows the models to have significantly fewer parameters and therefore be estimated reliably from less data. These smaller models also have dramatically reduced computational cost in terms of storing the model, performing inference in the model, and drawing samples from the model.

16.2 Using Graphs to Describe Model Structure

Structured probabilistic models use graphs (in the graph theory sense of "nodes" or "vertices" connected by edges) to represent interactions between random variables. Each node represents a random variable. Each edge represents a direct interaction. These direct interactions imply other, indirect interactions, but only the direct interactions need to be explicitly modeled.

There is more than one way to describe the interactions in a probability distribution using a graph. In the following sections, we describe some of the most popular and useful approaches. Graphical models can be largely divided into two categories: models based on directed acyclic graphs, and models based on undirected graphs.

16.2.1 Directed Models

One kind of structured probabilistic model is the **directed graphical model**, otherwise known as the **belief network** or **Bayesian network**[2] (Pearl, 1985).

Directed graphical models are called "directed" because their edges are directed, that is, they point from one vertex to another. This direction is represented in the drawing with an arrow. The direction of the arrow indicates which variable's probability distribution is defined in terms of the other's. Drawing an arrow from a to b means that we define the probability distribution over b via a conditional distribution, with a as one of the variables on the right side of the conditioning

[2] Judea Pearl suggested using the term "Bayesian network" when one wishes to "emphasize the judgmental" nature of the values computed by the network, i.e., to highlight that they usually represent degrees of belief rather than frequencies of events.

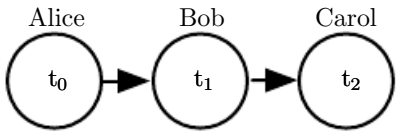

Figure 16.2: A directed graphical model depicting the relay race example. Alice's finishing time t_0 influences Bob's finishing time t_1, because Bob does not get to start running until Alice finishes. Likewise, Carol only gets to start running after Bob finishes, so Bob's finishing time t_1 directly influences Carol's finishing time t_2.

bar. In other words, the distribution over b depends on the value of a.

Continuing with the relay race example from section 16.1, suppose we name Alice's finishing time t_0, Bob's finishing time t_1, and Carol's finishing time t_2. As we saw earlier, our estimate of t_1 depends on t_0. Our estimate of t_2 depends directly on t_1 but only indirectly on t_0. We can draw this relationship in a directed graphical model, illustrated in figure 16.2.

Formally, a directed graphical model defined on variables $\mathbf{x}$ is defined by a directed acyclic graph $\mathcal{G}$ whose vertices are the random variables in the model, and a set of **local conditional probability distributions** $p(x_i \mid Pa_{\mathcal{G}}(x_i))$, where $Pa_{\mathcal{G}}(x_i)$ gives the parents of x_i in $\mathcal{G}$. The probability distribution over $\mathbf{x}$ is given by

$$p(\mathbf{x}) = \Pi_i p(x_i \mid Pa_{\mathcal{G}}(x_i)). \tag{16.1}$$

In our relay race example, this means that, using the graph drawn in figure 16.2,

$$p(t_0, t_1, t_2) = p(t_0)p(t_1 \mid t_0)p(t_2 \mid t_1). \tag{16.2}$$

This is our first time seeing a structured probabilistic model in action. We can examine the cost of using it, to observe how structured modeling has many advantages relative to unstructured modeling.

Suppose we represented time by discretizing time ranging from minute 0 to minute 10 into 6-second chunks. This would make t_0, t_1 and t_2 each be a discrete variable with 100 possible values. If we attempted to represent $p(t_0, t_1, t_2)$ with a table, it would need to store 999,999 values (100 values of t_0 $\times$ 100 values of t_1 $\times$ 100 values of t_2, minus 1, since the probability of one of the configurations is made redundant by the constraint that the sum of the probabilities be 1). If instead, we make a table only for each of the conditional probability distributions, then the distribution over t_0 requires 99 values, the table defining t_1 given t_0 requires 9,900 values, and so does the table defining t_2 given t_1. This comes to a total of 19,899 values. This means that using the directed graphical model reduced our number of parameters by a factor of more than 50!

In general, to model n discrete variables each having k values, the cost of the single table approach scales like $O(k^n)$, as we have observed before. Now suppose we build a directed graphical model over these variables. If m is the maximum number of variables appearing (on either side of the conditioning bar) in a single conditional probability distribution, then the cost of the tables for the directed model scales like $O(k^m)$. As long as we can design a model such that $m << n$, we get very dramatic savings.

In other words, as long as each variable has few parents in the graph, the distribution can be represented with very few parameters. Some restrictions on the graph structure, such as requiring it to be a tree, can also guarantee that operations like computing marginal or conditional distributions over subsets of variables are efficient.

It is important to realize what kinds of information can and cannot be encoded in the graph. The graph encodes only simplifying assumptions about which variables are conditionally independent from each other. It is also possible to make other kinds of simplifying assumptions. For example, suppose we assume Bob always runs the same regardless of how Alice performs. (In reality, Alice's performance probably influences Bob's performance—depending on Bob's personality, if Alice runs especially fast in a given race, this might encourage Bob to push hard and match her exceptional performance, or it might make him overconfident and lazy). Then the only effect Alice has on Bob's finishing time is that we must add Alice's finishing time to the total amount of time we think Bob needs to run. This observation allows us to define a model with $O(k)$ parameters instead of $O(k^2)$. However, note that t_0 and t_1 are still directly dependent with this assumption, because t_1 represents the absolute time at which Bob finishes, not the total time he spends running. This means our graph must still contain an arrow from t_0 to t_1. The assumption that Bob's personal running time is independent from all other factors cannot be encoded in a graph over t_0, t_1, and t_2. Instead, we encode this information in the definition of the conditional distribution itself. The conditional distribution is no longer a $k \times k - 1$ element table indexed by t_0 and t_1 but is now a slightly more complicated formula using only $k - 1$ parameters. The directed graphical model syntax does not place any constraint on how we define our conditional distributions. It only defines which variables they are allowed to take in as arguments.

16.2.2 Undirected Models

Directed graphical models give us one language for describing structured probabilistic models. Another popular language is that of **undirected models**, otherwise

known as **Markov random fields** (MRFs) or **Markov networks** (Kindermann, 1980). As their name implies, undirected models use graphs whose edges are undirected.

Directed models are most naturally applicable to situations where there is a clear reason to draw each arrow in one particular direction. Often these are situations where we understand the causality, and the causality flows in only one direction. One such situation is the relay race example. Earlier runners affect the finishing times of later runners; later runners do not affect the finishing times of earlier runners.

Not all situations we might want to model have such a clear direction to their interactions. When the interactions seem to have no intrinsic direction, or to operate in both directions, it may be more appropriate to use an undirected model.

As an example of such a situation, suppose we want to model a distribution over three binary variables: whether or not you are sick, whether or not your coworker is sick, and whether or not your roommate is sick. As in the relay race example, we can make simplifying assumptions about the kinds of interactions that take place. Assuming that your coworker and your roommate do not know each other, it is very unlikely that one of them will give the other an infection such as a cold directly. This event can be seen as so rare that it is acceptable not to model it. However, it is reasonably likely that either of them could give you a cold, and that you could pass it on to the other. We can model the indirect transmission of a cold from your coworker to your roommate by modeling the transmission of the cold from your coworker to you and the transmission of the cold from you to your roommate.

In this case, it is just as easy for you to cause your roommate to get sick as it is for your roommate to make you sick, so there is not a clean unidirectional narrative on which to base the model. This motivates using an undirected model. As with directed models, if two nodes in an undirected model are connected by an edge, then the random variables corresponding to those nodes interact with each other directly. Unlike directed models, the edge in an undirected model has no arrow and is not associated with a conditional probability distribution.

We denote the random variable representing your health as h_y, the random variable representing your roommate's health as h_r, and the random variable representing your colleague's health as h_c. See figure 16.3 for a drawing of the graph representing this scenario.

Formally, an undirected graphical model is a structured probabilistic model

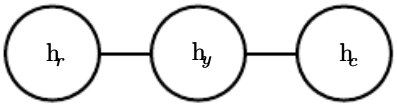

Figure 16.3: An undirected graph representing how your roommate's health h_r, your health h_y, and your work colleague's health h_c affect each other. You and your roommate might infect each other with a cold, and you and your work colleague might do the same, but assuming that your roommate and your colleague do not know each other, they can only infect each other indirectly via you.

defined on an undirected graph $\mathcal{G}$. For each clique $\mathcal{C}$ in the graph,[3] a **factor** $\phi(\mathcal{C})$ (also called a **clique potential**) measures the affinity of the variables in that clique for being in each of their possible joint states. The factors are constrained to be nonnegative. Together they define an **unnormalized probability distribution**

$$\tilde{p}(\mathbf{x}) = \Pi_{\mathcal{C} \in \mathcal{G}} \phi(\mathcal{C}). \tag{16.3}$$

The unnormalized probability distribution is efficient to work with so long as all the cliques are small. It encodes the idea that states with higher affinity are more likely. However, unlike in a Bayesian network, there is little structure to the definition of the cliques, so there is nothing to guarantee that multiplying them together will yield a valid probability distribution. See figure 16.4 for an example of reading factorization information from an undirected graph.

Our example of the cold spreading between you, your roommate, and your colleague contains two cliques. One clique contains h_y and h_c. The factor for this clique can be defined by a table and might have values resembling these:

	$\mathrm{h}_y = 0$	$\mathrm{h}_y = 1$
$\mathrm{h}_c = 0$	2	1
$\mathrm{h}_c = 1$	1	10

A state of 1 indicates good health, while a state of 0 indicates poor health (having been infected with a cold). Both of you are usually healthy, so the corresponding state has the highest affinity. The state where only one of you is sick has the lowest affinity, because this is a rare state. The state where both of you are sick (because one of you has infected the other) is a higher affinity state, though still not as common as the state where both are healthy.

[3] A clique of the graph is a subset of nodes that are all connected to each other by an edge of the graph.

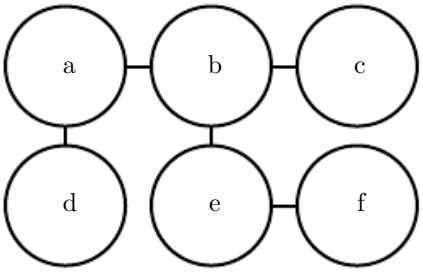

Figure 16.4: This graph implies that $p(\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}, \mathrm{f})$ can be written as $\frac{1}{Z}\phi_{\mathrm{a,b}}(\mathrm{a}, \mathrm{b})\phi_{\mathrm{b,c}}(\mathrm{b}, \mathrm{c})\phi_{\mathrm{a,d}}(\mathrm{a}, \mathrm{d})\phi_{\mathrm{b,e}}(\mathrm{b}, \mathrm{e})\phi_{\mathrm{e,f}}(\mathrm{e}, \mathrm{f})$ for an appropriate choice of the ϕ functions.

To complete the model, we would need to also define a similar factor for the clique containing h_y and h_r.

16.2.3 The Partition Function

While the unnormalized probability distribution is guaranteed to be nonnegative everywhere, it is not guaranteed to sum or integrate to 1. To obtain a valid probability distribution, we must use the corresponding normalized probability distribution[4]

$$p(\mathbf{x}) = \frac{1}{Z}\tilde{p}(\mathbf{x}), \tag{16.4}$$

where Z is the value that results in the probability distribution summing or integrating to 1:

$$Z = \int \tilde{p}(\mathbf{x})d\mathbf{x}. \tag{16.5}$$

You can think of Z as a constant when the ϕ functions are held constant. Note that if the ϕ functions have parameters, then Z is a function of those parameters. It is common in the literature to write Z with its arguments omitted to save space. The normalizing constant Z is known as the **partition function**, a term borrowed from statistical physics.

Since Z is an integral or sum over all possible joint assignments of the state $\mathbf{x}$, it is often intractable to compute. To be able to obtain the normalized probability distribution of an undirected model, the model structure and the definitions of the ϕ functions must be conducive to computing Z efficiently. In the context of deep learning, Z is usually intractable. Because of the intractability of computing Z

[4] A distribution defined by normalizing a product of clique potentials is also called a **Gibbs distribution**.

exactly, we must resort to approximations. Such approximate algorithms are the topic of chapter 18.

One important consideration to keep in mind when designing undirected models is that it is possible to specify the factors in such a way that Z does not exist. This happens if some of the variables in the model are continuous and the integral of $\tilde{p}$ over their domain diverges. For example, suppose we want to model a single scalar variable $\mathrm{x} \in \mathbb{R}$ with a single clique potential $\phi(x) = x^2$. In this case,

$$Z = \int x^2 dx. \tag{16.6}$$

Since this integral diverges, there is no probability distribution corresponding to this choice of $\phi(x)$. Sometimes the choice of some parameter of the ϕ functions determines whether the probability distribution is defined. For example, for $\phi(x; \beta) = \exp\left(-\beta x^2\right)$, the β parameter determines whether Z exists. Positive β results in a Gaussian distribution over x, but all other values of β make ϕ impossible to normalize.

One key difference between directed modeling and undirected modeling is that directed models are defined directly in terms of probability distributions from the start, while undirected models are defined more loosely by ϕ functions that are then converted into probability distributions. This changes the intuitions one must develop to work with these models. One key idea to keep in mind while working with undirected models is that the domain of each of the variables has dramatic effect on the kind of probability distribution that a given set of ϕ functions corresponds to. For example, consider an n-dimensional vector valued random variable $\mathbf{x}$ and an undirected model parametrized by a vector of biases $\boldsymbol{b}$. Suppose we have one clique for each element of $\mathbf{x}$, $\phi^{(i)}(\mathrm{x}_i) = \exp(b_i \mathrm{x}_i)$. What kind of probability distribution does this result in? The answer is that we do not have enough information, because we have not yet specified the domain of $\mathbf{x}$. If $\mathbf{x} \in \mathbb{R}^n$, then the integral defining Z diverges, and no probability distribution exists. If $\mathbf{x} \in \{0, 1\}^n$, then $p(\mathbf{x})$ factorizes into n independent distributions, with $p(\mathrm{x}_i = 1) = \text{sigmoid}(b_i)$. If the domain of $\mathbf{x}$ is the set of elementary basis vectors $(\{[1, 0, \ldots, 0], [0, 1, \ldots, 0], \ldots, [0, 0, \ldots, 1]\})$, then $p(\mathbf{x}) = \text{softmax}(\boldsymbol{b})$, so a large value of b_i actually reduces $p(\mathrm{x}_j = 1)$ for $j \neq i$. Often, it is possible to leverage the effect of a carefully chosen domain of a variable to obtain complicated behavior from a relatively simple set of ϕ functions. We explore a practical application of this idea in section 20.6.

16.2.4 Energy-Based Models

Many interesting theoretical results about undirected models depend on the assumption that $\forall \mathbf{x}, \tilde{p}(\mathbf{x}) > 0$. A convenient way to enforce this condition is to use an **energy-based model** (EBM) where

$$\tilde{p}(\mathbf{x}) = \exp(-E(\mathbf{x})), \tag{16.7}$$

and $E(\mathbf{x})$ is known as the **energy function**. Because $\exp(z)$ is positive for all z, this guarantees that no energy function will result in a probability of zero for any state $\mathbf{x}$. Being completely free to choose the energy function makes learning simpler. If we learned the clique potentials directly, we would need to use constrained optimization to arbitrarily impose some specific minimal probability value. By learning the energy function, we can use unconstrained optimization.[5] The probabilities in an energy-based model can approach arbitrarily close to zero but never reach it.

Any distribution of the form given by equation 16.7 is an example of a **Boltzmann distribution**. For this reason, many energy-based models are called **Boltzmann machines** (Fahlman *et al.*, 1983; Ackley *et al.*, 1985; Hinton *et al.*, 1984; Hinton and Sejnowski, 1986). There is no accepted guideline for when to call a model an energy-based model and when to call it a Boltzmann machine. The term Boltzmann machine was first introduced to describe a model with exclusively binary variables, but today many models such as the mean-covariance restricted Boltzmann machine incorporate real valued variables as well. While Boltzmann machines were originally defined to encompass both models with and without latent variables, the term Boltzmann machine is today most often used to designate models with latent variables, while Boltzmann machines without latent variables are more often called Markov random fields or log-linear models.

Cliques in an undirected graph correspond to factors of the unnormalized probability function. Because $\exp(a)\exp(b) = \exp(a+b)$, this means that different cliques in the undirected graph correspond to the different terms of the energy function. In other words, an energy-based model is just a special kind of Markov network: the exponentiation makes each term in the energy function correspond to a factor for a different clique. See figure 16.5 for an example of how to read the form of the energy function from an undirected graph structure. One can view an energy-based model with multiple terms in its energy function as being a **product of experts** (Hinton, 1999). Each term in the energy function corresponds to another factor in the probability distribution. Each term of the energy function can

[5] For some models, we may still need to use constrained optimization to make sure Z exists.

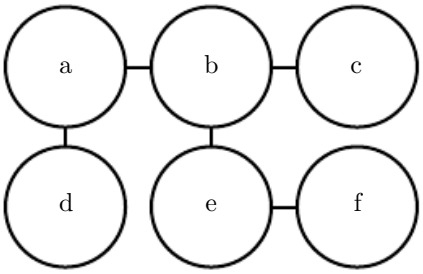

Figure 16.5: This graph implies that $E(\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}, \mathrm{f})$ can be written as $E_{\mathrm{a,b}}(\mathrm{a}, \mathrm{b}) + E_{\mathrm{b,c}}(\mathrm{b}, \mathrm{c}) + E_{\mathrm{a,d}}(\mathrm{a}, \mathrm{d}) + E_{\mathrm{b,e}}(\mathrm{b}, \mathrm{e}) + E_{\mathrm{e,f}}(\mathrm{e}, \mathrm{f})$ for an appropriate choice of the per-clique energy functions. Note that we can obtain the ϕ functions in figure 16.4 by setting each ϕ to the exponential of the corresponding negative energy, e.g., $\phi_{\mathrm{a,b}}(\mathrm{a}, \mathrm{b}) = \exp(-E(\mathrm{a}, \mathrm{b}))$.

be thought of as an "expert" that determines whether a particular soft constraint is satisfied. Each expert may enforce only one constraint that concerns only a low-dimensional projection of the random variables, but when combined by multiplication of probabilities, the experts together enforce a complicated high-dimensional constraint.

One part of the definition of an energy-based model serves no functional purpose from a machine learning point of view: the $-$ sign in equation 16.7. This $-$ sign could be incorporated into the definition of E. For many choices of the function E, the learning algorithm is free to determine the sign of the energy anyway. The $-$ sign is present primarily to preserve compatibility between the machine learning literature and the physics literature. Many advances in probabilistic modeling were originally developed by statistical physicists, for whom E refers to actual physical energy and does not have arbitrary sign. Terminology such as "energy" and "partition function" remains associated with these techniques, even though their mathematical applicability is broader than the physics context in which they were developed. Some machine learning researchers (e.g., Smolensky [1986], who referred to negative energy as **harmony**) have chosen to omit the negation, but this is not the standard convention.

Many algorithms that operate on probabilistic models need to compute not $p_{\mathrm{model}}(\boldsymbol{x})$ but only $\log \tilde{p}_{\mathrm{model}}(\boldsymbol{x})$. For energy-based models with latent variables $\boldsymbol{h}$, these algorithms are sometimes phrased in terms of the negative of this quantity, called the **free energy**:

$$\mathcal{F}(\boldsymbol{x}) = -\log \sum_{\boldsymbol{h}} \exp(-E(\boldsymbol{x}, \boldsymbol{h})). \tag{16.8}$$

In this book, we usually prefer the more general $\log \tilde{p}_{\mathrm{model}}(\boldsymbol{x})$ formulation.

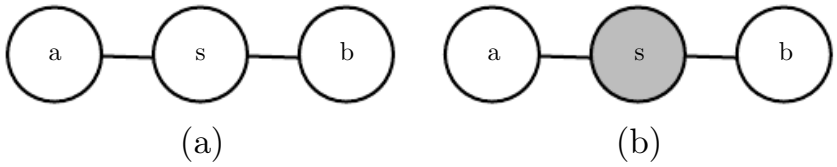

Figure 16.6: (a) The path between random variable a and random variable b through s is active, because s is not observed. This means that a and b are not separated. (b) Here s is shaded in, to indicate that it is observed. Because the only path between a and b is through s, and that path is inactive, we can conclude that a and b are separated given s.

16.2.5 Separation and D-Separation

The edges in a graphical model tell us which variables directly interact. We often need to know which variables *indirectly* interact. Some of these indirect interactions can be enabled or disabled by observing other variables. More formally, we would like to know which subsets of variables are conditionally independent from each other, given the values of other subsets of variables.

Identifying the conditional independences in a graph is simple for undirected models. In this case, conditional independence implied by the graph is called **separation**. We say that a set of variables $\mathbb{A}$ is **separated** from another set of variables $\mathbb{B}$ given a third set of variables $\mathbb{S}$ if the graph structure implies that $\mathbb{A}$ is independent from $\mathbb{B}$ given $\mathbb{S}$. If two variables a and b are connected by a path involving only unobserved variables, then those variables are not separated. If no path exists between them, or all paths contain an observed variable, then they are separated. We refer to paths involving only unobserved variables as "active" and paths including an observed variable as "inactive."

When we draw a graph, we can indicate observed variables by shading them in. See figure 16.6 for a depiction of how active and inactive paths in an undirected model look when drawn in this way. See figure 16.7 for an example of reading separation from an undirected graph.

Similar concepts apply to directed models, except that in the context of directed models, these concepts are referred to as **d-separation**. The "d" stands for "dependence." D-separation for directed graphs is defined the same as separation for undirected graphs: We say that a set of variables $\mathbb{A}$ is d-separated from another set of variables $\mathbb{B}$ given a third set of variables $\mathbb{S}$ if the graph structure implies that $\mathbb{A}$ is independent from $\mathbb{B}$ given $\mathbb{S}$.

As with undirected models, we can examine the independences implied by the graph by looking at what active paths exist in the graph. As before, two variables are dependent if there is an active path between them and d-separated if no such

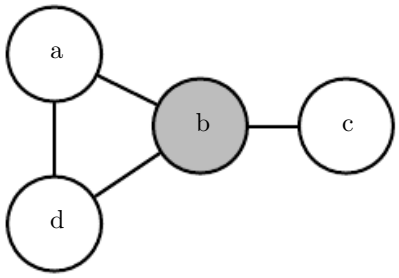

Figure 16.7: An example of reading separation properties from an undirected graph. Here b is shaded to indicate that it is observed. Because observing b blocks the only path from a to c, we say that a and c are separated from each other given b. The observation of b also blocks one path between a and d, but there is a second, active path between them. Therefore, a and d are not separated given b.

path exists. In directed nets, determining whether a path is active is somewhat more complicated. See figure 16.8 for a guide to identifying active paths in a directed model. See figure 16.9 for an example of reading some properties from a graph.

It is important to remember that separation and d-separation tell us only about those conditional independences *that are implied by the graph.* There is no requirement that the graph imply all independences that are present. In particular, it is always legitimate to use the complete graph (the graph with all possible edges) to represent any distribution. In fact, some distributions contain independences that are not possible to represent with existing graphical notation. **Context-specific independences** are independences that are present dependent on the value of some variables in the network. For example, consider a model of three binary variables: a, b and c. Suppose that when a is 0, b and c are independent, but when a is 1, b is deterministically equal to c. Encoding the behavior when $a = 1$ requires an edge connecting b and c. The graph then fails to indicate that b and c are independent when $a = 0$.

In general, a graph will never imply that an independence exists when it does not. However, a graph may fail to encode an independence.

16.2.6 Converting between Undirected and Directed Graphs

We often refer to a specific machine learning model as being undirected or directed. For example, we typically refer to RBMs as undirected and sparse coding as directed. This choice of wording can be somewhat misleading, because no probabilistic model is inherently directed or undirected. Instead, some models are most easily *described* using a directed graph, or most easily described using an undirected graph.

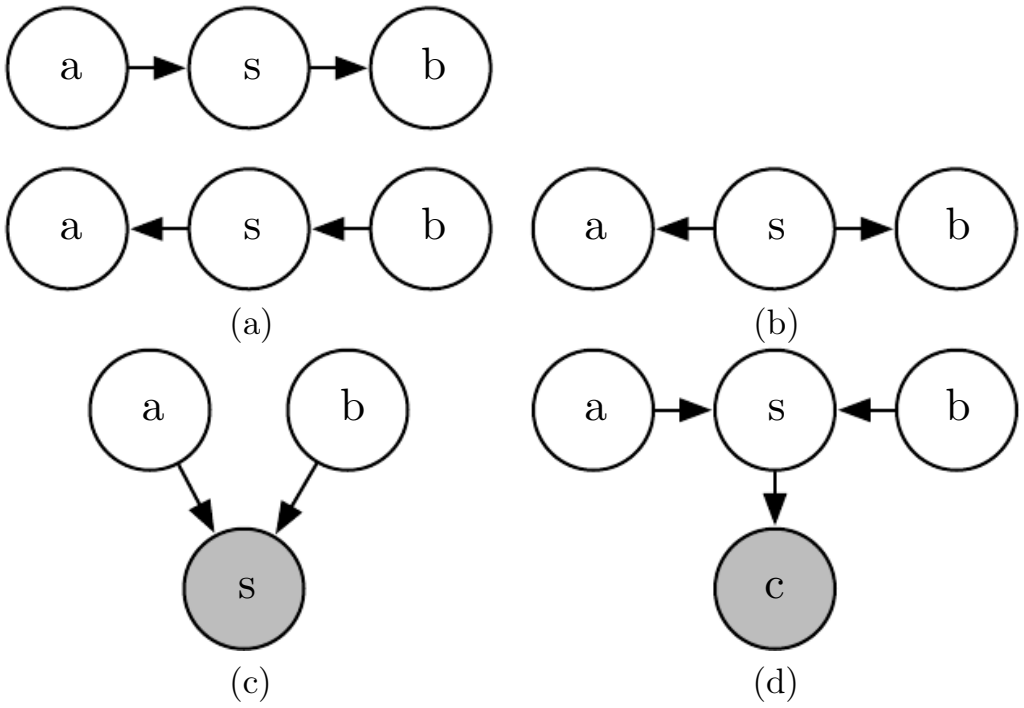

Figure 16.8: All the kinds of active paths of length two that can exist between random variables a and b. *(a)* Any path with arrows proceeding directly from a to b or vice versa. This kind of path becomes blocked if s is observed. We have already seen this kind of path in the relay race example. *(b)* Variables a and b are connected by a *common cause* s. For example, suppose s is a variable indicating whether or not there is a hurricane, and a and b measure the wind speed at two different nearby weather monitoring outposts. If we observe very high winds at station a, we might expect to also see high winds at b. This kind of path can be blocked by observing s. If we already know there is a hurricane, we expect to see high winds at b, regardless of what is observed at a. A lower than expected wind at a (for a hurricane) would not change our expectation of winds at b (knowing there is a hurricane). However, if s is not observed, then a and b are dependent, i.e., the path is active. *(c)* Variables a and b are both parents of s. This is called a **V-structure**, or **the collider case**. The V-structure causes a and b to be related by the **explaining away effect**. In this case, the path is actually active when s is observed. For example, suppose s is a variable indicating that your colleague is not at work. The variable a represents her being sick, while b represents her being on vacation. If you observe that she is not at work, you can presume she is probably sick or on vacation, but it is not especially likely that both have happened at the same time. If you find out that she is on vacation, this fact is sufficient to *explain* her absence. You can infer that she is probably not also sick. *(d)* The explaining away effect happens even if any descendant of s is observed! For example, suppose that c is a variable representing whether you have received a report from your colleague. If you notice that you have not received the report, this increases your estimate of the probability that she is not at work today, which in turn makes it more likely that she is either sick or on vacation. The only way to block a path through a V-structure is to observe none of the descendants of the shared child.

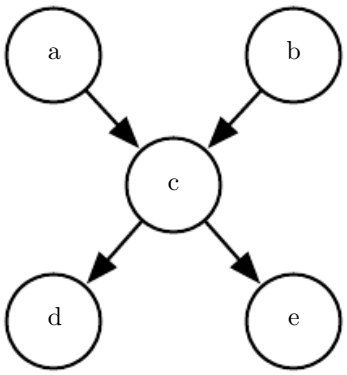

Figure 16.9: From this graph, we can read out several d-separation properties. Examples include:

- a and b are d-separated given the empty set
- a and e are d-separated given c
- d and e are d-separated given c

We can also see that some variables are no longer d-separated when we observe some variables:

- a and b are not d-separated given c
- a and b are not d-separated given d

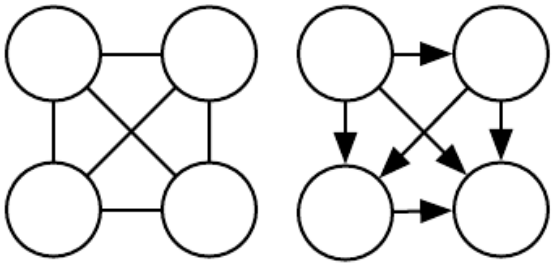

Figure 16.10: Examples of complete graphs, which can describe any probability distribution. Here we show examples with four random variables. *(Left)* The complete undirected graph. In the undirected case, the complete graph is unique. *(Right)* A complete directed graph. In the directed case, there is not a unique complete graph. We choose an ordering of the variables and draw an arc from each variable to every variable that comes after it in the ordering. There are thus a factorial number of complete graphs for every set of random variables. In this example, we order the variables from left to right, top to bottom.

Directed models and undirected models both have their advantages and disadvantages. Neither approach is clearly superior and universally preferred. Instead, we should choose which language to use for each task. This choice will partially depend on which probability distribution we wish to describe. We may choose to use either directed modeling or undirected modeling based on which approach can capture the most independences in the probability distribution or which approach uses the fewest edges to describe the distribution. Other factors can affect the decision of which language to use. Even while working with a single probability distribution, we may sometimes switch between different modeling languages. Sometimes a different language becomes more appropriate if we observe a certain subset of variables, or if we wish to perform a different computational task. For example, the directed model description often provides a straightforward approach to efficiently draw samples from the model (described in section 16.3), while the undirected model formulation is often useful for deriving approximate inference procedures (as we will see in chapter 19, where the role of undirected models is highlighted in equation 19.56).

Every probability distribution can be represented by either a directed model or an undirected model. In the worst case, one can always represent any distribution by using a "complete graph." For a directed model, the complete graph is any directed acyclic graph in which we impose some ordering on the random variables, and each variable has all other variables that precede it in the ordering as its ancestors in the graph. For an undirected model, the complete graph is simply a graph containing a single clique encompassing all the variables. See figure 16.10 for an example.

Of course, the utility of a graphical model is that the graph implies that some variables do not interact directly. The complete graph is not very useful because it does not imply any independences.

When we represent a probability distribution with a graph, we want to choose a graph that implies as many independences as possible, without implying any independences that do not actually exist.

From this point of view, some distributions can be represented more efficiently using directed models, while other distributions can be represented more efficiently using undirected models. In other words, directed models can encode some independences that undirected models cannot encode, and vice versa.

Directed models are able to use one specific kind of substructure that undirected models cannot represent perfectly. This substructure is called an **immorality**. The structure occurs when two random variables a and b are both parents of a third random variable c, and there is no edge directly connecting a and b in either direction. (The name "immorality" may seem strange; it was coined in the graphical models literature as a joke about unmarried parents.) To convert a directed model with graph $\mathcal{D}$ into an undirected model, we need to create a new graph $\mathcal{U}$. For every pair of variables x and y, we add an undirected edge connecting x and y to $\mathcal{U}$ if there is a directed edge (in either direction) connecting x and y in $\mathcal{D}$ or if x and y are both parents in $\mathcal{D}$ of a third variable z. The resulting $\mathcal{U}$ is known as a **moralized graph**. See figure 16.11 for examples of converting directed models to undirected models via moralization.

Likewise, undirected models can include substructures that no directed model can represent perfectly. Specifically, a directed graph $\mathcal{D}$ cannot capture all the conditional independences implied by an undirected graph $\mathcal{U}$ if $\mathcal{U}$ contains a **loop** of length greater than three, unless that loop also contains a **chord**. A loop is a sequence of variables connected by undirected edges, with the last variable in the sequence connected back to the first variable in the sequence. A chord is a connection between any two nonconsecutive variables in the sequence defining a loop. If $\mathcal{U}$ has loops of length four or greater and does not have chords for these loops, we must add the chords before we can convert it to a directed model. Adding these chords discards some of the independence information that was encoded in $\mathcal{U}$. The graph formed by adding chords to $\mathcal{U}$ is known as a **chordal**, or **triangulated**, graph, because all the loops can now be described in terms of smaller, triangular loops. To build a directed graph $\mathcal{D}$ from the chordal graph, we need to also assign directions to the edges. When doing so, we must not create a directed cycle in $\mathcal{D}$, or the result will not define a valid directed probabilistic model. One way to assign directions to the edges in $\mathcal{D}$ is to impose an ordering on the random variables, then

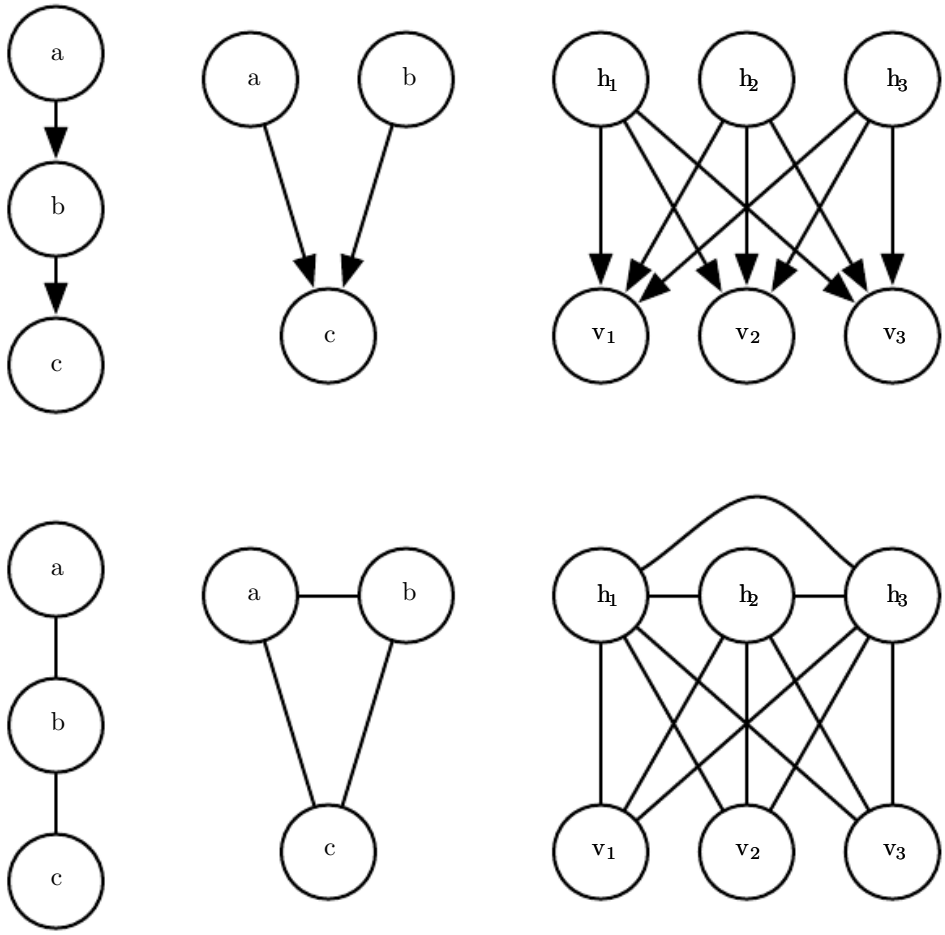

Figure 16.11: Examples of converting directed models (top row) to undirected models (bottom row) by constructing moralized graphs. *(Left)* This simple chain can be converted to a moralized graph merely by replacing its directed edges with undirected edges. The resulting undirected model implies exactly the same set of independences and conditional independences. *(Center)* This graph is the simplest directed model that cannot be converted to an undirected model without losing some independences. This graph consists entirely of a single immorality. Because a and b are parents of c, they are connected by an active path when c is observed. To capture this dependence, the undirected model must include a clique encompassing all three variables. This clique fails to encode the fact that a$\perp$b. *(Right)* In general, moralization may add many edges to the graph, thus losing many implied independences. For example, this sparse coding graph requires adding moralizing edges between every pair of hidden units, thus introducing a quadratic number of new direct dependences.

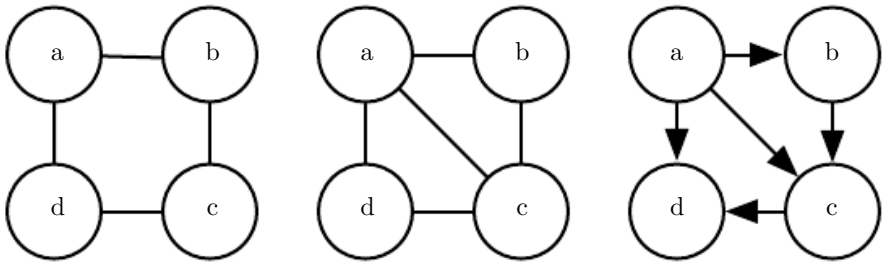

Figure 16.12: Converting an undirected model to a directed model. *(Left)* This undirected model cannot be converted to a directed model because it has a loop of length four with no chords. Specifically, the undirected model encodes two different independences that no directed model can capture simultaneously: $\mathrm{a} \perp \mathrm{c} \mid \{\mathrm{b}, \mathrm{d}\}$ and $\mathrm{b} \perp \mathrm{d} \mid \{\mathrm{a}, \mathrm{c}\}$. *(Center)* To convert the undirected model to a directed model, we must triangulate the graph, by ensuring that all loops of greater than length three have a chord. To do so, we can either add an edge connecting a and c or we can add an edge connecting b and d. In this example, we choose to add the edge connecting a and c. *(Right)* To finish the conversion process, we must assign a direction to each edge. When doing so, we must not create any directed cycles. One way to avoid directed cycles is to impose an ordering over the nodes, and always point each edge from the node that comes earlier in the ordering to the node that comes later in the ordering. In this example, we use the variable names to impose alphabetical order.

point each edge from the node that comes earlier in the ordering to the node that comes later in the ordering. See figure 16.12 for a demonstration.

16.2.7 Factor Graphs

Factor graphs are another way of drawing undirected models that resolve an ambiguity in the graphical representation of standard undirected model syntax. In an undirected model, the scope of every ϕ function must be a *subset* of some clique in the graph. Ambiguity arises because it is not clear if each clique actually has a corresponding factor whose scope encompasses the entire clique—for example, a clique containing three nodes may correspond to a factor over all three nodes, or may correspond to three factors that each contain only a pair of the nodes. Factor graphs resolve this ambiguity by explicitly representing the scope of each ϕ function. Specifically, a factor graph is a graphical representation of an undirected model that consists of a bipartite undirected graph. Some of the nodes are drawn as circles. These nodes correspond to random variables, as in a standard undirected model. The rest of the nodes are drawn as squares. These nodes correspond to the factors ϕ of the unnormalized probability distribution. Variables and factors may be connected with undirected edges. A variable and a factor are connected

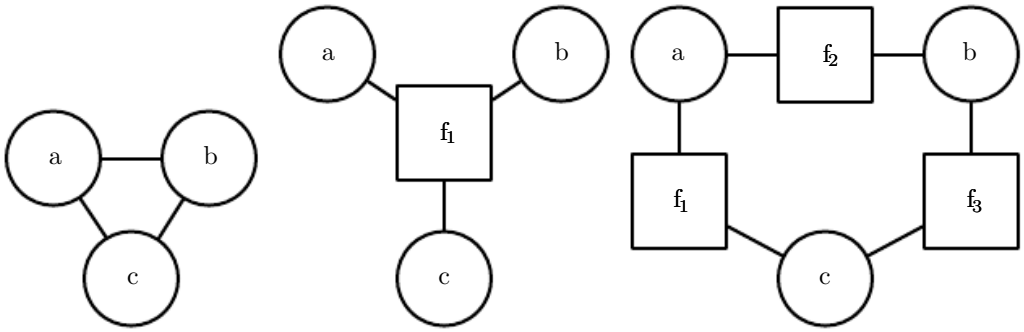

Figure 16.13: An example of how a factor graph can resolve ambiguity in the interpretation of undirected networks. *(Left)* An undirected network with a clique involving three variables: a, b and c. *(Center)* A factor graph corresponding to the same undirected model. This factor graph has one factor over all three variables. *(Right)* Another valid factor graph for the same undirected model. This factor graph has three factors, each over only two variables. Representation, inference, and learning are all asymptotically cheaper in this factor graph than in the factor graph depicted in the center, even though both require the same undirected graph to represent.

in the graph if and only if the variable is one of the arguments to the factor in the unnormalized probability distribution. No factor may be connected to another factor in the graph, nor can a variable be connected to a variable. See figure 16.13 for an example of how factor graphs can resolve ambiguity in the interpretation of undirected networks.

16.3 Sampling from Graphical Models

Graphical models also facilitate the task of drawing samples from a model.

One advantage of directed graphical models is that a simple and efficient procedure called **ancestral sampling** can produce a sample from the joint distribution represented by the model.

The basic idea is to sort the variables x_i in the graph into a topological ordering, so that for all i and j, j is greater than i if x_i is a parent of x_j. The variables can then be sampled in this order. In other words, we first sample $\mathrm{x}_1 \sim P(\mathrm{x}_1)$, then sample $P(\mathrm{x}_2 \mid Pa_{\mathcal{G}}(\mathrm{x}_2))$, and so on, until finally we sample $P(\mathrm{x}_n \mid Pa_{\mathcal{G}}(\mathrm{x}_n))$. So long as each conditional distribution $p(\mathrm{x}_i \mid Pa_{\mathcal{G}}(\mathrm{x}_i))$ is easy to sample from, then the whole model is easy to sample from. The topological sorting operation guarantees that we can read the conditional distributions in equation 16.1 and sample from them in order. Without the topological sorting, we might attempt to sample a variable before its parents are available.

For some graphs, more than one topological ordering is possible. Ancestral sampling may be used with any of these topological orderings.

Ancestral sampling is generally very fast (assuming sampling from each conditional is easy) and convenient.

One drawback to ancestral sampling is that it only applies to directed graphical models. Another drawback is that it does not support every conditional sampling operation. When we wish to sample from a subset of the variables in a directed graphical model, given some other variables, we often require that all the conditioning variables come earlier than the variables to be sampled in the ordered graph. In this case, we can sample from the local conditional probability distributions specified by the model distribution. Otherwise, the conditional distributions we need to sample from are the posterior distributions given the observed variables. These posterior distributions are usually not explicitly specified and parametrized in the model. Inferring these posterior distributions can be costly. In models where this is the case, ancestral sampling is no longer efficient.

Unfortunately, ancestral sampling is applicable only to directed models. We can sample from undirected models by converting them to directed models, but this often requires solving intractable inference problems (to determine the marginal distribution over the root nodes of the new directed graph) or requires introducing so many edges that the resulting directed model becomes intractable. Sampling from an undirected model without first converting it to a directed model seems to require resolving cyclical dependencies. Every variable interacts with every other variable, so there is no clear beginning point for the sampling process. Unfortunately, drawing samples from an undirected graphical model is an expensive, multipass process. The conceptually simplest approach is **Gibbs sampling**. Suppose we have a graphical model over an n-dimensional vector of random variables $\mathbf{x}$. We iteratively visit each variable x_i and draw a sample conditioned on all the other variables, from $p(\mathrm{x}_i \mid \mathrm{x}_{-i})$. Due to the separation properties of the graphical model, we can equivalently condition on only the neighbors of x_i. Unfortunately, after we have made one pass through the graphical model and sampled all n variables, we still do not have a fair sample from $p(\mathbf{x})$. Instead, we must repeat the process and resample all n variables using the updated values of their neighbors. Asymptotically, after many repetitions, this process converges to sampling from the correct distribution. It can be difficult to determine when the samples have reached a sufficiently accurate approximation of the desired distribution. Sampling techniques for undirected models are an advanced topic, covered in more detail in chapter 17.

16.4 Advantages of Structured Modeling

The primary advantage of using structured probabilistic models is that they allow us to dramatically reduce the cost of representing probability distributions as well as learning and inference. Sampling is also accelerated in the case of directed models, while the situation can be complicated with undirected models. The primary mechanism that allows all these operations to use less runtime and memory is choosing to not model certain interactions. Graphical models convey information by leaving edges out. Anywhere there is not an edge, the model specifies the assumption that we do not need to model a direct interaction.

A less quantifiable benefit of using structured probabilistic models is that they allow us to explicitly separate representation of knowledge from learning of knowledge or inference given existing knowledge. This makes our models easier to develop and debug. We can design, analyze, and evaluate learning algorithms and inference algorithms that are applicable to broad classes of graphs. Independently, we can design models that capture the relationships we believe are important in our data. We can then combine these different algorithms and structures and obtain a Cartesian product of different possibilities. It would be much more difficult to design end-to-end algorithms for every possible situation.

16.5 Learning about Dependencies

A good generative model needs to accurately capture the distribution over the observed, or "visible," variables $\mathbf{v}$. Often the different elements of $\mathbf{v}$ are highly dependent on each other. In the context of deep learning, the approach most commonly used to model these dependencies is to introduce several latent or "hidden" variables, $\mathbf{h}$. The model can then capture dependencies between any pair of variables v_i and v_j indirectly, via direct dependencies between v_i and $\mathbf{h}$, and direct dependencies between $\mathbf{h}$ and v_j.

A good model of $\mathbf{v}$ which did not contain any latent variables would need to have very large numbers of parents per node in a Bayesian network or very large cliques in a Markov network. Just representing these higher-order interactions is costly—both in a computational sense, because the number of parameters that must be stored in memory scales exponentially with the number of members in a clique, but also in a statistical sense, because this exponential number of parameters requires a wealth of data to estimate accurately.

When the model is intended to capture dependencies between visible variables with direct connections, it is usually infeasible to connect all variables, so the

graph must be designed to connect those variables that are tightly coupled and omit edges between other variables. An entire field of machine learning called **structure learning** is devoted to this problem. For a good reference on structure learning, see (Koller and Friedman, 2009). Most structure learning techniques are a form of greedy search. A structure is proposed and a model with that structure is trained, then given a score. The score rewards high training set accuracy and penalizes model complexity. Candidate structures with a small number of edges added or removed are then proposed as the next step of the search. The search proceeds to a new structure that is expected to increase the score.

Using latent variables instead of adaptive structure avoids the need to perform discrete searches and multiple rounds of training. A fixed structure over visible and hidden variables can use direct interactions between visible and hidden units to impose indirect interactions between visible units. Using simple parameter learning techniques, we can learn a model with a fixed structure that imputes the right structure on the marginal $p(\mathbf{v})$.

Latent variables have advantages beyond their role in efficiently capturing $p(\mathbf{v})$. The new variables $\mathbf{h}$ also provide an alternative representation for $\mathbf{v}$. For example, as discussed in section 3.9.6, the mixture of Gaussians model learns a latent variable that corresponds to the category of examples the input was drawn from. This means that the latent variable in a mixture of Gaussians model can be used to do classification. In chapter 14 we saw how simple probabilistic models like sparse coding learn latent variables that can be used as input features for a classifier, or as coordinates along a manifold. Other models can be used in this same way, but deeper models and models with different kinds of interactions can create even richer descriptions of the input. Many approaches accomplish feature learning by learning latent variables. Often, given some model of $\mathbf{v}$ and $\mathbf{h}$, experimental observations show that $\mathbb{E}[\mathbf{h} \mid \mathbf{v}]$ or $\operatorname{argmax}_{\boldsymbol{h}} p(\boldsymbol{h}, \boldsymbol{v})$ is a good feature mapping for $\boldsymbol{v}$.

16.6 Inference and Approximate Inference

One of the main ways we can use a probabilistic model is to ask questions about how variables are related to each other. Given a set of medical tests, we can ask what disease a patient might have. In a latent variable model, we might want to extract features $\mathbb{E}[\mathbf{h} \mid \mathbf{v}]$ describing the observed variables $\mathbf{v}$. Sometimes we need to solve such problems in order to perform other tasks. We often train our models using the principle of maximum likelihood. Because

$$\log p(\boldsymbol{v}) = \mathbb{E}_{\mathbf{h} \sim p(\mathbf{h} \mid \boldsymbol{v})} \left[\log p(\boldsymbol{h}, \boldsymbol{v}) - \log p(\boldsymbol{h} \mid \boldsymbol{v}) \right], \tag{16.9}$$

we often want to compute $p(\mathbf{h} \mid \boldsymbol{v})$ in order to implement a learning rule. All these are examples of **inference** problems in which we must predict the value of some variables given other variables, or predict the probability distribution over some variables given the value of other variables.

Unfortunately, for most interesting deep models, these inference problems are intractable, even when we use a structured graphical model to simplify them. The graph structure allows us to represent complicated, high-dimensional distributions with a reasonable number of parameters, but the graphs used for deep learning are usually not restrictive enough to also allow efficient inference.

It is straightforward to see that computing the marginal probability of a general graphical model is #P hard. The complexity class #P is a generalization of the complexity class NP. Problems in NP require determining only whether a problem has a solution and finding a solution if one exists. Problems in #P require counting the number of solutions. To construct a worst-case graphical model, imagine that we define a graphical model over the binary variables in a 3-SAT problem. We can impose a uniform distribution over these variables. We can then add one binary latent variable per clause that indicates whether each clause is satisfied. We can then add another latent variable indicating whether all the clauses are satisfied. This can be done without making a large clique, by building a reduction tree of latent variables, with each node in the tree reporting whether two other variables are satisfied. The leaves of this tree are the variables for each clause. The root of the tree reports whether the entire problem is satisfied. Because of the uniform distribution over the literals, the marginal distribution over the root of the reduction tree specifies what fraction of assignments satisfy the problem. While this is a contrived worst-case example, NP hard graphs commonly arise in practical real-world scenarios.

This motivates the use of approximate inference. In the context of deep learning, this usually refers to variational inference, in which we approximate the true distribution $p(\mathbf{h} \mid \boldsymbol{v})$ by seeking an approximate distribution $q(\mathbf{h}|\mathbf{v})$ that is as close to the true one as possible. This and other techniques are described in depth in chapter 19.

16.7 The Deep Learning Approach to Structured Probabilistic Models

Deep learning practitioners generally use the same basic computational tools as other machine learning practitioners who work with structured probabilistic models.

In the context of deep learning, however, we usually make different design decisions about how to combine these tools, resulting in overall algorithms and models that have a very different flavor from more traditional graphical models.

Deep learning does not always involve especially deep graphical models. In the context of graphical models, we can define the depth of a model in terms of the graphical model graph rather than the computational graph. We can think of a latent variable h_i as being at depth j if the shortest path from h_i to an observed variable is j steps. We usually describe the depth of the model as being the greatest depth of any such h_i. This kind of depth is different from the depth induced by the computational graph. Many generative models used for deep learning have no latent variables or only one layer of latent variables but use deep computational graphs to define the conditional distributions within a model.

Deep learning essentially always makes use of the idea of distributed representations. Even shallow models used for deep learning purposes (such as pretraining shallow models that will later be composed to form deep ones) nearly always have a single large layer of latent variables. Deep learning models typically have more latent variables than observed variables. Complicated nonlinear interactions between variables are accomplished via indirect connections that flow through multiple latent variables.

By contrast, traditional graphical models usually contain mostly variables that are at least occasionally observed, even if many of the variables are missing at random from some training examples. Traditional models mostly use higher-order terms and structure learning to capture complicated nonlinear interactions between variables. If there are latent variables, they are usually few in number.

The way that latent variables are designed also differs in deep learning. The deep learning practitioner typically does not intend for the latent variables to take on any specific semantics ahead of time—the training algorithm is free to invent the concepts it needs to model a particular dataset. The latent variables are usually not very easy for a human to interpret after the fact, though visualization techniques may allow some rough characterization of what they represent. When latent variables are used in the context of traditional graphical models, they are often designed with some specific semantics in mind—the topic of a document, the intelligence of a student, the disease causing a patient's symptoms, and so forth. These models are often much more interpretable by human practitioners and often have more theoretical guarantees, yet they are less able to scale to complex problems and are not reusable in as many different contexts as deep models are.

Another obvious difference is the kind of connectivity typically used in the deep learning approach. Deep graphical models typically have large groups of units that

are all connected to other groups of units, so that the interactions between two groups may be described by a single matrix. Traditional graphical models have very few connections, and the choice of connections for each variable may be individually designed. The design of the model structure is tightly linked with the choice of inference algorithm. Traditional approaches to graphical models typically aim to maintain the tractability of exact inference. When this constraint is too limiting, a popular approximate inference algorithm is called **loopy belief propagation**. Both approaches often work well with sparsely connected graphs. By comparison, models used in deep learning tend to connect each visible unit v_i to many hidden units h_j, so that $\mathbf{h}$ can provide a distributed representation of v_i (and probably several other observed variables too). Distributed representations have many advantages, but from the point of view of graphical models and computational complexity, distributed representations have the disadvantage of usually yielding graphs that are not sparse enough for the traditional techniques of exact inference and loopy belief propagation to be relevant. As a consequence, one of the most striking differences between the larger graphical models community and the deep graphical models community is that loopy belief propagation is almost never used for deep learning. Most deep models are instead designed to make Gibbs sampling or variational inference algorithms efficient. Another consideration is that deep learning models contain a very large number of latent variables, making efficient numerical code essential. This provides an additional motivation, besides the choice of high-level inference algorithm, for grouping the units into layers with a matrix describing the interaction between two layers. This allows the individual steps of the algorithm to be implemented with efficient matrix product operations, or sparsely connected generalizations, like block diagonal matrix products or convolutions.

Finally, the deep learning approach to graphical modeling is characterized by a marked tolerance of the unknown. Rather than simplifying the model until all quantities we might want can be computed exactly, we increase the power of the model until it is just barely possible to train or use. We often use models whose marginal distributions cannot be computed and are satisfied simply to draw approximate samples from these models. We often train models with an intractable objective function that we cannot even approximate in a reasonable amount of time, but we are still able to approximately train the model if we can efficiently obtain an estimate of the gradient of such a function. The deep learning approach is often to figure out what the minimum amount of information we absolutely need is, and then to figure out how to get a reasonable approximation of that information as quickly as possible.

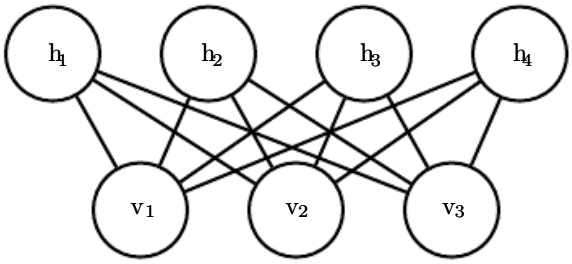

Figure 16.14: An RBM drawn as a Markov network.

16.7.1 Example: The Restricted Boltzmann Machine

The **restricted Boltzmann machine** (RBM) (Smolensky, 1986), or **harmonium**, is the quintessential example of how graphical models are used for deep learning. The RBM is not itself a deep model. Instead, it has a single layer of latent variables that may be used to learn a representation for the input. In chapter 20, we will see how RBMs can be used to build many deeper models. Here, we show how the RBM exemplifies many of the practices used in a wide variety of deep graphical models: its units are organized into large groups called layers, the connectivity between layers is described by a matrix, the connectivity is relatively dense, the model is designed to allow efficient Gibbs sampling, and the emphasis of the model design is on freeing the training algorithm to learn latent variables whose semantics were not specified by the designer. In section 20.2, we revisit the RBM in more detail.

The canonical RBM is an energy-based model with binary visible and hidden units. Its energy function is

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{b}^\top \boldsymbol{v} - \boldsymbol{c}^\top \boldsymbol{h} - \boldsymbol{v}^\top \boldsymbol{W} \boldsymbol{h}, \tag{16.10}$$

where $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{W}$ are unconstrained, real valued, learnable parameters. We can see that the model is divided into two groups of units: $\boldsymbol{v}$ and $\boldsymbol{h}$, and the interaction between them is described by a matrix $\boldsymbol{W}$. The model is depicted graphically in figure 16.14. As this figure makes clear, an important aspect of this model is that there are no direct interactions between any two visible units or between any two hidden units (hence "restricted"; a general Boltzmann machine may have arbitrary connections).

The restrictions on the RBM structure yield the nice properties

$$p(\mathbf{h} \mid \mathbf{v}) = \Pi_i p(\mathrm{h}_i \mid \mathbf{v}) \tag{16.11}$$

and

$$p(\mathbf{v} \mid \mathbf{h}) = \Pi_i p(\mathrm{v}_i \mid \mathbf{h}). \tag{16.12}$$

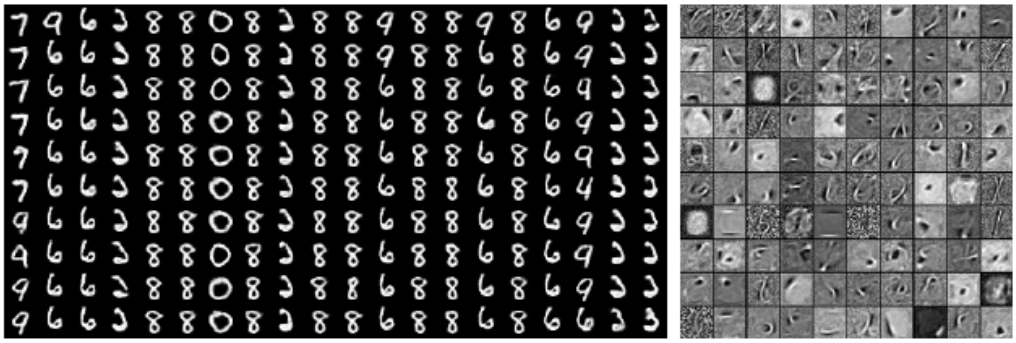

Figure 16.15: Samples from a trained RBM and its weights. *(Left)* Samples from a model trained on MNIST, drawn using Gibbs sampling. Each column is a separate Gibbs sampling process. Each row represents the output of another 1,000 steps of Gibbs sampling. Successive samples are highly correlated with one another. *(Right)* The corresponding weight vectors. Compare this to the samples and weights of a linear factor model, shown in figure 13.2. The samples here are much better because the RBM prior $p(\boldsymbol{h})$ is not constrained to be factorial. The RBM can learn which features should appear together when sampling. On the other hand, the RBM posterior $p(\boldsymbol{h} \mid \boldsymbol{v})$ is factorial, while the sparse coding posterior $p(\boldsymbol{h} \mid \boldsymbol{v})$ is not, so the sparse coding model may be better for feature extraction. Other models are able to have both a nonfactorial $p(\boldsymbol{h})$ and a nonfactorial $p(\boldsymbol{h} \mid \boldsymbol{v})$. Image reproduced with permission from LISA (2008).

The individual conditionals are simple to compute as well. For the binary RBM we obtain

$$P(\mathrm{h}_i = 1 \mid \mathbf{v}) = \sigma\left(\mathbf{v}^\top \boldsymbol{W}_{:,i} + c_i\right), \tag{16.13}$$

$$P(\mathrm{h}_i = 0 \mid \mathbf{v}) = 1 - \sigma\left(\mathbf{v}^\top \boldsymbol{W}_{:,i} + c_i\right). \tag{16.14}$$

Together these properties allow for efficient **block Gibbs sampling**, which alternates between sampling all of $\mathbf{h}$ simultaneously and sampling all of $\mathbf{v}$ simultaneously. Samples generated by Gibbs sampling from an RBM model are shown in figure 16.15.

Since the energy function itself is just a linear function of the parameters, it is easy to take its derivatives. For example,

$$\frac{\partial}{\partial W_{i,j}} E(\mathbf{v}, \mathbf{h}) = -\mathrm{v}_i \mathrm{h}_j. \tag{16.15}$$

These two properties—efficient Gibbs sampling and efficient derivatives—make training convenient. In chapter 18, we will see that undirected models may be trained by computing such derivatives applied to samples from the model.

Training the model induces a representation $\boldsymbol{h}$ of the data $\boldsymbol{v}$. We can often use $\mathbb{E}_{\mathbf{h} \sim p(\mathbf{h} \mid \boldsymbol{v})}[\boldsymbol{h}]$ as a set of features to describe $\boldsymbol{v}$.

Overall, the RBM demonstrates the typical deep learning approach to graphical models: representation learning accomplished via layers of latent variables, combined with efficient interactions between layers parametrized by matrices.

The language of graphical models provides an elegant, flexible and clear language for describing probabilistic models. In the chapters ahead, we use this language, among other perspectives, to describe a wide variety of deep probabilistic models.