Chapter 1

Introduction

In this thesis, I explore algorithms for pattern classification, data compression, and channel coding. At first, it may be hard to imagine how three so different research areas can be brought into focus under a single theme that is both novel and of practical value. My hope is to convince the reader that these three problems can be attacked in an interesting and fruitful manner using a recently developed class of algorithms that make use of *probabilistic structure*. These algorithms take advantage of a graphical description of the dependencies between random variables. For example, the forward-backward algorithm [Baum and Petrie 1966] (a.k.a. the "BCJR" algorithm [Bahl et al. 1974]) makes use of a chain-like Markovian relationship between random variables. As another example, stochastic multi-resolution algorithms [Chou, Willsky and Benveniste 1994a; Chou, Willsky and Benveniste 1994b] make use of a tree-like relationship between random variables.

The roots of probabilistic structure reach far back to the beginning of the 20th century. In 1921, Sewal Wright [1921] developed "path analysis" as a means to study statistical relationships in biological data. Few new developments were made until the 1960's when statisticians began using graphs to describe restrictions in statistical models called "log-linear models" [Vorobev 1962; Goodman 1970]. In 1963, the idea of probabilistic structure briefly reared its head in the engineering research community when Gallager [1963] invented an error-control decoding algorithm based on a graphical description of the probabilistic relationships between variables involved in channel coding. Most likely because of the primitive computers available at the time, his algorithm was quickly overlooked by his peers, only to be rediscovered nearly 35 years later independently by at least three research groups, and to be shown to yield unprecedented performance in error-control coding applications [Berrou, Glavieux and Thitimajshima 1993; Wiberg, Loeliger and Kötter 1995; MacKay and Neal 1995]. Probabilistic structure has been most extensively developed in the artificial intelligence literature, with applications ranging from taxonomic hierarchies [Woods 1975; Schubert 1976] to medical diagnosis [Spiegelhalter 1990]. In 1986, Pearl [1986] presented a general algorithm based on a graphical representation of probabilistic structure. In the neural networks area, inverse graphical models have been recently proposed as a means to avoid computationally burdensome algorithms [Hinton et al. 1995; Dayan et al. 1995].

1.1 A probabilistic perspective

Probabilistic relationships are based on uncertainty, and offhand it is not obvious that pattern classification, data compression, and channel coding can be described usefully by such relationships. Given a segment of speech, shouldn't a classifier make a deterministic decision, say, as to whether or not the speaker has security clearance? If we require a measure of reliability regarding the decision, surely it too is deterministically related to the speech segment. Given a sequence of source symbols, shouldn't a data compression algorithm deterministically produce a sequence of codeword symbols? Given the output of a noisy telephone line, shouldn't a channel decoder (modem) make a deterministic decision about the contents of the transmitted data file? In the majority of practical cases, the answer to all three questions is "yes". Consequently, there has been a great deal of work done in all three areas without considering probabilistic structure *in detail*. However, if we probe a little further into the nature of these problems, we'll find that there are sophisticated probability models underlying each problem.

1.1.1 Pattern classification

A soft-decision *classifier* estimates the probability that a given pattern $\mathbf{v}$ belongs to each class $j \in \{0, \ldots, J-1\}$. That is, the classifier estimates

$$P_r(j|\mathbf{v}). \tag{1.1}$$

If these probabilities can be accurately estimated, Bayes decision theory tells us that a minimum rate of error can be achieved by choosing the class j that maximizes $P_r(j|\mathbf{v})$ [Chow 1957; Duda and Hart 1973].

We could use a regression model to estimate $P_r(j|\mathbf{v})$. For example, regression using a flexible model has been somewhat successfully used to classify individual digits extracted from handwritten United States ZIP codes [Le Cun et al. 1989]. However, this approach ignores the *causal structure* of the physical process of producing handwritten digits.

In order to faithfully capture the actual physical process that produces each digit, we first ought to specify an *a priori* distribution $P(j)$ over the digit classes $j \in \{0, \ldots, 9\}$ — maybe some digits are more common than others. Next, for a given class of digit j, we expect there to be a distribution $P(\mathbf{r}|j)$ over a set of digit attributes $\mathbf{r}$. Each element of $\mathbf{r}$ might specify the presence or absence of a particular line segment or flourish. Given a set of features $\mathbf{r}$, we expect there to be a distribution $P(\mathbf{v}|\mathbf{r})$ over possible images — this distribution models the way in which features combine to make an image, as well as noise such as ink spots. The joint distribution given by this model of the real world can be written

$$P(j, \mathbf{r}, \mathbf{v}) = P(j)P(\mathbf{r}|j)P(\mathbf{v}|\mathbf{r}), \tag{1.2}$$

and the distribution over classes given a pattern can be obtained by marginalizing out $\mathbf{r}$ and using Bayes' rule:

$$P(j|\mathbf{v}) = \frac{\sum_{\mathbf{r}} P(j, \mathbf{r}, \mathbf{v})}{\sum_{j'} \sum_{\mathbf{r}} P(j', \mathbf{r}, \mathbf{v})}. \tag{1.3}$$

So, it appears that to properly model the physics of the problem we need a more sophisticated probabilistic description than (1.1). In general, a *correct* model of this sort, where $P(j|\mathbf{v}) \approx P_r(j|\mathbf{v})$ will perform optimally in terms of classification error.

1.1.2 Data compression

A block *source code* maps each block of B source patterns $\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^B$ to a codeword $\mathbf{u}$, such that for each valid $\mathbf{u}$ there is a unique block $\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^B$. I will consider i.i.d. sources for which each pattern $\mathbf{v}^k$ is drawn independently from an identical distribution $P_r(\mathbf{v})$. The purpose of noiseless source coding, or lossless data compression, is to losslessly represent the source patterns by codewords, so that the average codeword length is as low as possible. Shannon's noiseless source coding theorem [Shannon 1948] states that the average codeword length per source pattern cannot be less than the entropy of the source:

$$\mathrm{E}[\ell(\mathbf{u})]/B \geq \mathcal{H}, \tag{1.4}$$

where $\ell(\mathbf{u})$ is the length of codeword $\mathbf{u}$ in bits, and $\mathcal{H}$ is the entropy of the source in bits:

$$\mathcal{H} = -\sum_{\mathbf{v}} P_r(\mathbf{v}) \log_2 P_r(\mathbf{v}). \tag{1.5}$$

Arithmetic coding [Rissanen and Langdon 1976; Witten, Neal and Cleary 1987] is a practical algorithm for producing near-optimal codewords when the source distribution $P_r(\mathbf{v})$ is known. Sometimes, *e.g.*, if $\mathbf{v}$ is binary-valued, these probabilities can be easily estimated from the source. Often, however, the distribution is too complex, and so a more sophisticated parametric model or flexible model must be used to estimate the probabilities. For example, consider a high-dimensional binary image $\mathbf{v}$ that is produced by the physical process described above, so that

$$P(j, \mathbf{r}, \mathbf{v}) = P(j)P(\mathbf{r}|j)P(\mathbf{v}|\mathbf{r}), \tag{1.6}$$

The probabilities used by the arithmetic encoder are obtained by marginalizing out j and $\mathbf{r}$:

$$P(\mathbf{v}) = \sum_j \sum_{\mathbf{r}} P(j, \mathbf{r}, \mathbf{v}). \tag{1.7}$$

We see that a probabilistic description can also be very useful for source coding.

1.1.3 Channel coding

A block *channel code* maps a vector of information symbols $\mathbf{u}$ to a vector of codeword symbols $\mathbf{x}$. This mapping adds error-control checks to $\mathbf{u}$ in order to protect the block against channel noise. After $\mathbf{x}$ is transmitted across the channel, the decoder receives a noise-corrupted version $\mathbf{y}$ and produces an estimate of the information block $\hat{\mathbf{u}}$. We say that a *block error* or a *word error* has occurred if $\hat{\mathbf{u}} \neq \mathbf{u}$. Shannon's channel coding theorem [Shannon 1948] states that for any given channel, there *exists*[1] a code that can achieve an arbitrarily low probability of block error when the signal-to-noise ratio is greater than a channel-dependent threshold called the Shannon limit. Roughly speaking, the codewords are kept far apart in codeword symbol space, so that when a moderately noise-corrupted codeword is received, it is still possible to determine with high probability which codeword was transmitted.

From a probabilistic perspective, the decoder can minimize the word error rate (WER) by choosing an estimate $\hat{\mathbf{u}}$ that maximizes $P_r(\hat{\mathbf{u}}|\mathbf{y})$, or minimize the symbol error rate (*e.g.*, BER for bit error rate) by choosing an estimate $\hat{\mathbf{u}}$ that maximizes $\prod_k P_r(\hat{u}_k|\mathbf{y})$. A probabilistic model can be constructed by examining the encoding process and the channel. We first specify a (usually uniform) distribution for the information blocks, $P(\mathbf{u})$. Often, the encoder uses a set of *state* variables $\mathbf{s}$ in order to produce the codeword. These variables are determined from the information block using a distribution $P(\mathbf{s}|\mathbf{u})$ — although this relationship is usually deterministic, this probabilistic description will come in handy later

[1] Shannon was quite the tease. He proved the code exists, but did not give us a way to decode it in practice.

on when we study probabilistic decoding. The transmitted codeword is determined from the information block and state variables by $P(\mathbf{x}|\mathbf{u}, \mathbf{s})$. Finally, the received word is related to the transmitted codeword by a distribution $P(\mathbf{y}|\mathbf{x})$ that models the channel. The joint distribution given by the model is

$$P(\mathbf{u}, \mathbf{s}, \mathbf{x}, \mathbf{y}) = P(\mathbf{u})P(\mathbf{s}|\mathbf{u})P(\mathbf{x}|\mathbf{u}, \mathbf{s})P(\mathbf{y}|\mathbf{x}), \tag{1.8}$$

and the distribution over information symbol u_k given the channel output can be obtained by marginalizing out $u_j \forall j \neq k$, $\mathbf{s}$ and $\mathbf{x}$ and using Bayes' rule:

$$P(u_k|\mathbf{y}) = \frac{\sum_{u_j \forall j \neq k} \sum_{\mathbf{s}} \sum_{\mathbf{x}} P(\mathbf{u}, \mathbf{s}, \mathbf{x}, \mathbf{y})}{\sum_{u_j \forall j} \sum_{\mathbf{y}'} \sum_{\mathbf{s}} \sum_{\mathbf{x}} P(\mathbf{u}, \mathbf{s}, \mathbf{x}, \mathbf{y}')}. \tag{1.9}$$

Although this probabilistic formulation may seem strange compared to many of the strongly algebraic traditional approaches, it is this formulation that I view as the foundation of the recently proposed high-performance *turbo-codes* [Berrou and Glavieux 1996].

1.1.4 Probabilistic inference

As presented above, pattern classification, data compression, and channel coding are all similar in that some type of marginal (and possibly conditioned) distribution is sought for a given joint distribution. Consider a set of random variables $\mathbf{z} = (z_1, z_2, \ldots, z_N)$ that covary according to a joint distribution $P(z_1, z_2, \ldots, z_N)$. For any two subset of variables $\mathbf{z}_1 \subseteq \mathbf{z}$ and $\mathbf{z}_2 \subseteq \mathbf{z}$, I will refer to the computation or estimation of $P(\mathbf{z}_1|\mathbf{z}_2)$, or a decision based on $P(\mathbf{z}_1|\mathbf{z}_2)$, as *probabilistic inference*. Examples of probabilistic inference include the computation of the class probabilities for pattern classification (1.3), the computation of the input probability for data compression (1.7), and the information symbol decisions based on the information symbol probabilities for channel coding (1.9).

1.2 Bayesian networks

Often, the joint distribution associated with a probabilistic inference problem can be decomposed into locally interacting factors. For example, the joint distributions involved in the computations of (1.3), (1.7), and (1.9) can be expressed in the forms given in (1.2), (1.6), and (1.8). By taking advantage of such *probabilistic structure*, we can design more efficient inference algorithms.

1.2.1 Probabilistic structure

Probabilistic structure can be characterized by a set of *conditional independence* relationships. (Note that this structural description says nothing about the values of the probabilities.) For example, in the case of channel coding, we can use the chain rule of probability to write out the joint distribution:

$$P(\mathbf{u}, \mathbf{s}, \mathbf{x}, \mathbf{y}) = P(\mathbf{u})P(\mathbf{s}|\mathbf{u})P(\mathbf{x}|\mathbf{u}, \mathbf{s})P(\mathbf{y}|\mathbf{u}, \mathbf{s}, \mathbf{x}). \tag{1.10}$$

The last factor can be simplified, since the received vector $\mathbf{y}$ is conditionally independent of the information vector $\mathbf{u}$ and the state vector $\mathbf{s}$, given the transmitted codeword $\mathbf{x}$:

$$P(\mathbf{y}|\mathbf{u}, \mathbf{s}, \mathbf{x}) = P(\mathbf{y}|\mathbf{x}). \tag{1.11}$$

By substituting this conditional independency relationship into (1.10), we obtain the more structured form of the joint distribution given in (1.8).

The general idea is to express the joint distribution as a product of factors, where each factor depends on a subset of the random variables. In the simplest case, each factor depends on a single random variable, making marginalization easy. Most distributions that describe practical problems cannot be broken up in this way, and the subsets overlap. In this case, the most we can hope for is that the overlap is minimal.

It turns out that graph theory provides a succinct way to represent probabilistic structure. A graphical representation for probabilistic structure, along with functions that can be used to derive the joint distribution, is called a *graphical model*. Examples of graphical models include *Markov random fields* [Kinderman and Snell 1980], *Bayesian networks* [Pearl 1988], and *chain graphs* [Lauritzen and Wermuth 1989]. Here, I consider Bayesian networks.

1.2.2 Definition of a Bayesian network

The conditional independency relationships for a distribution can be described graphically. Not only does the graphical representation concisely capture probabilistic structure, but it forms a computational framework for computing useful probabilities. A *Bayesian network* for a set of random variables $\mathbf{z} = (z_1, z_2, \ldots, z_N)$ consists of a directed acyclic graph[2], with one vertex for each variable, and a set of probability functions $P(z_k|a(z_k))$, $k = 1, \ldots, N$,

[2] I will use the term "acyclic" to refer to a graph without any *directed* cycles; "singly-connected" to refer to a graph without any directed or *undirected* cycles; and "multiply-connected" to refer to a graph that has at least one directed or undirected cycle.

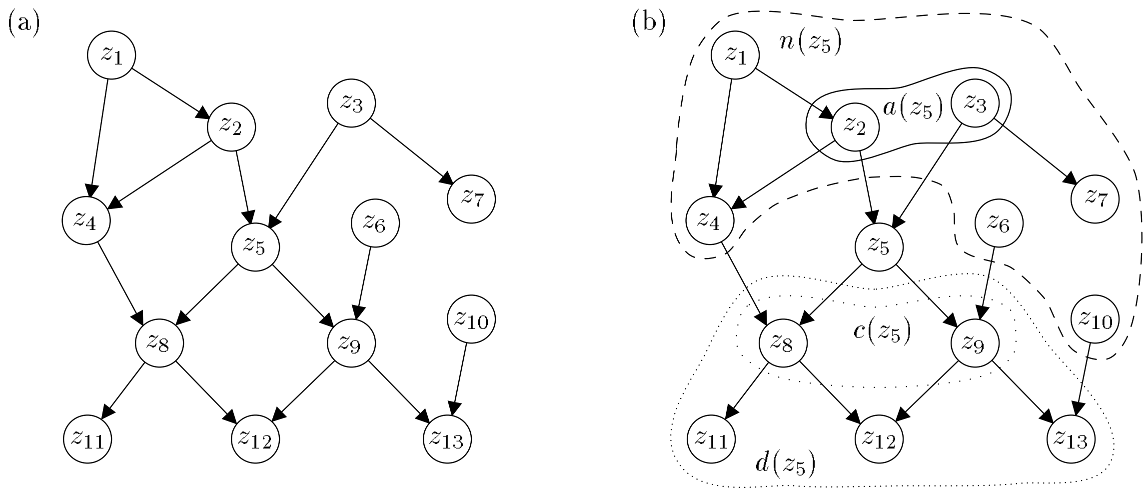

Figure 1.1: (a) An example of a Bayesian network. (b) The parents $a(z_5)$ of z_5 are shown by a solid loop; the children $c(z_5)$ are shown by a sparse dotted loop; the descendents $d(z_5)$ are shown by a dense dotted loop; the nondescendents $n(z_5)$ are shown by a dashed loop.

where the *parents* $a(z_k)$ of z_k are the variables that have directed edges connecting *to* z_k. (For simplicity of prose, I will often refer to a vertex by its variable name.) For now, we can think of the function $P(z_k|a(z_k))$ as a properly normalized exhaustive list of real numbers corresponding to the possible configurations of z_k and $a(z_k)$. Together, the graph and the probability functions are referred to as the *network specification.*

Several definitions will help to understand how a Bayesian network describes the probabilistic structure for a joint distribution $P(\mathbf{z})$. The *children* $c(z_k)$ of z_k are the variables that have directed edges connecting *from* z_k. The *descendents* $d(z_k)$ of z_k are its children, its children's children, *etc.* The *nondescendents* $n(z_k)$ of z_k are the variables in $\{z_1, z_2, \ldots, z_{k-1}, z_{k+1}, \ldots, z_N\}$ that are not in $d(z_k)$. Note that $n(z_k) \neq \overline{d(z_k)}$, since z_k is *not* included in the nondescendents. From these definitions, it follows that $a(z_k) \subseteq n(z_k)$. z_k is a *root* if $a(z_k) = \varnothing$, and z_k is a *leaf* if $c(z_k) = \varnothing$. If z_k is connected by a single edge, I will say that z_k is *singly-connected.*

Figure 1.1 shows an example of a Bayesian network, along with the parents, children, descendents and nondescendents of variable z_5. The roots of this network are z_1, z_3, z_6, z_{10}; the leaves of this network are $z_7, z_{11}, z_{12}, z_{13}$. Variables z_6, z_7, z_{10}, z_{11} are singly-connected.

A Bayesian network implies that given the parents of z_k, the distribution over z_k will not change if any combination of the nondescendents of z_k are also given:

$$P(z_k|a(z_k), W) = P(z_k|a(z_k)), \quad \forall\, W \subseteq n(z_k). \tag{1.12}$$

In other words, z_k is conditionally independent of any combination of its nondescendents, given its parents. To take the family hierarchy (not necessarily a tree) analogy further, given the genetic code of Susan's parents, determining the genes of her siblings, her grandparents, her grandparents' children, her children's other parents or any combination of the above does not influence our prediction of Susan's genetic make-up. This is not true for descendents. For example, determining the genes of Susan's grandchildren *does* influence our prediction of her genetic make-up, even though determining the genes of those parents of Susan's grandchildren who are not her children does not (notice that the latter are nondescendents).

The joint distribution for a Bayesian network can be written in a structured form simply by writing out the product of the distributions for individual variables, where each variable is conditioned on its parents:

$$P(\mathbf{z}) = \prod_{k=1}^{N} P(z_k | a(z_k)). \tag{1.13}$$

This form follows from (1.12) in the following way. Since a Bayesian network contains no directed cycles, it is always possible to choose an *ancestral ordering* $z_{\pi(1)}, z_{\pi(2)}, \ldots, z_{\pi(N)}$, where $\pi(\cdot)$ is a permutation map, so that the descendents of each variable come later in the ordering: $d(z_{\pi(k)}) \subseteq \{z_{\pi(k+1)}, \ldots, z_{\pi(N)}\}$. For example, the variables in the network shown in Figure 1.1 were assigned so that $z_1, z_2, \ldots, z_{13}$ is an ancestral ordering. Using the general chain rule of probability applied to the ancestral ordering, we have

$$P(\mathbf{z}) = \prod_{k=1}^{N} P(z_{\pi(k)} | z_{\pi(1)}, \ldots, z_{\pi(k-1)}). \tag{1.14}$$

From the definition of the ancestral ordering, it follows that the set of variables that precede $z_{\pi(k)}$ is a subset of its nondescendents and that the parents of $z_{\pi(k)}$ are a subset of the variables that precede $z_{\pi(k)}$:

$$a(z_{\pi(k)}) \subseteq \{z_{\pi(1)}, \ldots, z_{\pi(k-1)}\} \subseteq n(z_{\pi(k)}). \tag{1.15}$$

For this reason, $\{z_{\pi(1)}, \ldots, z_{\pi(k-1)}\} = a(z_{\pi(k)}) \cup (\{z_{\pi(1)}, \ldots, z_{\pi(k-1)} \setminus a(z_{\pi(k)}))$, and taking $W = \{z_{\pi(1)}, \ldots, z_{\pi(k-1)}\} \setminus a(z_{\pi(k)})$ in (1.12), we have

$$\begin{aligned} P(z_{\pi(k)} | \{z_{\pi(1)}, \ldots, z_{\pi(k-1)}\}) &= P(z_{\pi(k)} | a(z_{\pi(k)}), \{z_{\pi(1)}, \ldots, z_{\pi(k-1)}\} \setminus a(z_{\pi(k)})) \\ &= P(z_{\pi(k)} | a(z_{\pi(k)})). \end{aligned} \tag{1.16}$$

Inserting this result into (1.14), we obtain the form given in (1.13).

If the probability functions for a Bayesian network are not specified, the network is

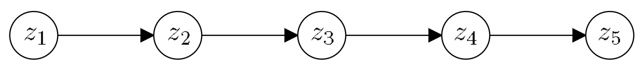

Figure 1.2: An example of a chain-type Bayesian network, or a *Markov chain*.

meant to represent *all* distributions that can be written in the form given in (1.13). For the network with ancestral ordering $z_1, z_2, \ldots, z_{13}$ shown in Figure 1.1, (1.13) gives a joint distribution

$$\begin{aligned} P(\mathbf{z}) &= P(z_1)P(z_2|z_1)P(z_3)P(z_4|z_1, z_2)P(z_5|z_2, z_3)P(z_6)P(z_7|z_3) \\ &\quad \cdot P(z_8|z_4, z_5)P(z_9|z_5, z_6)P(z_{10})P(z_{11}|z_8)P(z_{12}|z_8, z_9)P(z_{13}|z_9, z_{10}). \end{aligned} \tag{1.17}$$

This product could have been written in any order, but using an ancestral ordering helps clarify the dependencies. In this equation, each variable z_k is conditioned on variables whose distributions appear to the *left* of the distribution for z_k. Note that a Bayesian network may have more than one ancestral ordering. In this case, $z_{10}, z_6, z_3, z_7,\ z_1, z_2, z_4, z_5, z_8, z_{11}, z_9, z_{13}, z_{12}$ is also an ancestral ordering.

An interesting special case of a Bayesian network is the chain-type network shown in Figure 1.2, also known as a first-order *Markov chain*. Applying (1.13) to this network, we obtain

$$P(\mathbf{z}) = P(z_1)P(z_2|z_1)P(z_3|z_2)P(z_4|z_3)P(z_5|z_4). \tag{1.18}$$

This type of structure is frequently used to model time series data, where it is often assumed that the next state of a physical system depends only on the previous state. Comparing this network to the more complex networks that appear later in this thesis, the Bayesian network can be thought of as a generalization of the Markov chain.

1.2.3 Ancestral simulation

It is often practically impossible to draw a vector $\mathbf{z}$ from a joint distribution $P(\mathbf{z})$. However, if the joint distribution can be described by a Bayesian network, and if a value for each z_k can be drawn from its conditional probability $P(z_k|a(z_k))$ in a practical manner, then the ancestral ordering can be used to draw an entire vector. Starting with $k = 1$, we draw $z_{\pi(1)}$ from $P(z_{\pi(1)})$. We continue to draw $z_{\pi(k)}$ from $P(z_{\pi(k)}|a(z_{\pi(k)}))$ for $k = 2, \ldots, N$ until an entire vector $\mathbf{z}$ has been drawn. In this way, the probabilistic structure implied by the graph allows us to decompose the simulation problem into local pieces.

1.2.4 Example 1: Recursive convolutional codes and turbo-codes

One very popular class of channel codes can nicely be described using Bayesian networks. The encoder for a *recursive convolutional code* is simply a finite state machine (FSM) that takes information bits as inputs and produces codeword bits as outputs [Berrou and Glavieux 1996]. The details of the FSM are not essential at this point and are discussed in more detail in Chapter 4 and in [Berrou and Glavieux 1996]. To encode a vector of information bits $\mathbf{u}$, we feed them into the FSM. The codeword $\mathbf{x}$ with length N is produced by interleaving the original vector of information bits and the output of the FSM. When the information bits are included in the codeword like this, the code is called *systematic*. Since the FSM outputs one bit per time step, the codeword will have length $N = 2K$ if there are K information bits. Codeword signals are then transmitted (say by sending $+1$ if $x_k = 1$ and -1 if $x_k = 0$) and the decoder receives a noise-corrupted version $\mathbf{y}$ of the transmitted signals.

Figure 1.3a shows the Bayesian network for an example of a recursive systematic convolutional code. Normally, the number of information bits K is much larger than 6 (typical numbers range from 100 to 100,000 bits). s_k is the state of the FSM at time k. The state and state output distributions, which are actually deterministic, are

$$P(s_0|u_0) = \begin{cases} 1.0 & \text{if } s_0 \text{ follows the initial state of the FSM for input } u_0 \\ 0.0 & \text{otherwise,} \end{cases}$$

$$P(s_k|u_k, s_{k-1}) = \begin{cases} 1.0 & \text{if } s_k \text{ follows } s_{k-1} \text{ in the FSM for input } u_k \\ 0.0 & \text{otherwise,} \end{cases}$$

$$P(x_{2k}|u_k) = \begin{cases} 1.0 & \text{if } x_{2k} = u_k \\ 0.0 & \text{otherwise,} \end{cases}$$

$$P(x_{2k+1}|s_k) = \begin{cases} 1.0 & \text{if } x_{2k+1} \text{ is the output of the FSM for state } s_k \\ 0.0 & \text{otherwise.} \end{cases} \tag{1.19}$$

Assuming that the channel simply adds independent Gaussian noise with variance σ^2 to the

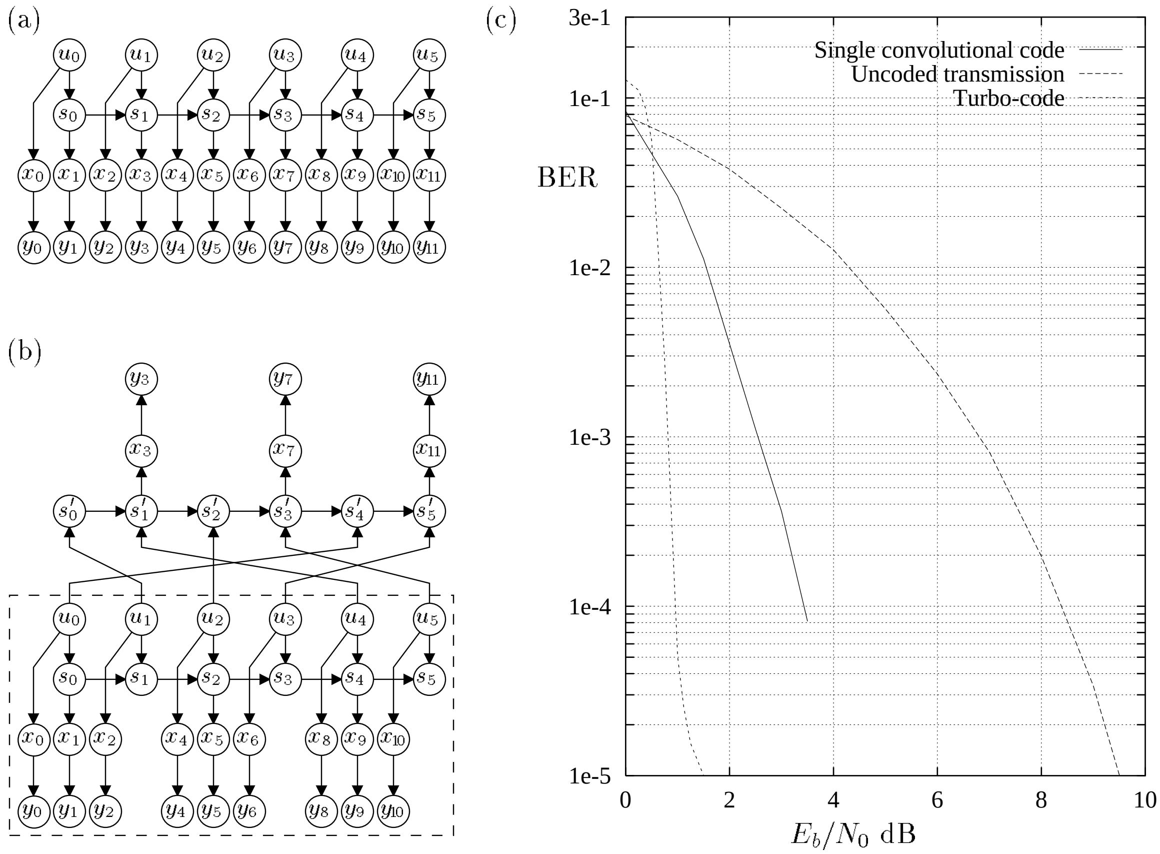

Figure 1.3: (a) The Bayesian network for a recursive systematic convolutional code. (b) The Bayesian network for an example of the recently proposed turbo-code. (c) The bit error rate (BER) performance for instances of these two codes, plus other competitive codes.

codeword signals, the conditional distributions for the received signals are

$$P(y_k|x_k) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_k-1.0)^2/2\sigma^2} & \text{if } x_k = 1 \\ \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_k+1.0)^2/2\sigma^2} & \text{if } x_k = 0. \end{cases} \tag{1.20}$$

Given an information vector $\mathbf{u}$, encoding and channel transmission can be simulated by one sweep of ancestral simulation. For example, we first directly copy u_0 into x_0, which is then used to draw a noisy channel output value y_0. Then, we use u_0 to determine s_0, which is then used to determine x_1, which is then used to draw a noisy channel output value y_1. Then, we directly copy u_1 into x_2 and so on until the entire channel output vector $\mathbf{y}$ has been obtained.

The decoder sees only the vector $\mathbf{y}$, and would like to infer the most likely value of each information bit, *i.e.*, determine for each k the u_k that maximizes $P(u_k|\mathbf{y})$. This task can be accomplished using the forward-backward (a.k.a. BCJR) algorithm [Baum and Petrie

1966; Bahl et al. 1974], which is just a special case of the general *probability propagation* algorithm discussed in Section 1.3. Once the block is decoded, we can compare the decoded information bit values with the true ones to determine the number of bit errors made for the block transmission. If we simulate the transmission of many blocks, we can obtain an estimate of the bit error rate (BER).

This procedure was carried out using a FSM with a 7-bit state memory. The z-transform $X(z)/U(z) = 1 + z^1 + z^2 + z^4 + z^5 + z^6 + z^7/1 + z^2 + z^5$ of the FSM was chosen to maximize the minimum Hamming distance between all pairs of codewords [Viterbi and Omura 1979; Lin and Costello 1983] ($d_{\min} = 10$). The information vector length was $K = 5000$ (giving a codeword length of $N = 10000$), and 5000 vectors (25×10^6 information bits in all) were transmitted for a fixed noise variance. It is common practice to give BER results as a function of the noise level measured by a signal-to-noise ratio E_b/N_0 in decibels. For any system with $N = 2K$, E_b/N_0 is related to σ^2 by $E_b/N_0 = -10\log_{10}\sigma^2$. Figure 1.3c shows the BER as a function of E_b/N_0 for this recursive systematic convolutional code. Notice that as E_b/N_0 increases (σ^2 decreases), the BER drops. I also give the BER curve for uncoded transmission, where each information bit is transmitted twice so that the same total number of signals are transmitted per block for the uncoded and coded systems (maintaining $N = 2K$). For the channel used in these experiments, low BERs (say, less than 10^{-3}) can only be achieved when $E_b/N_0 > 0.0$ dB. This limit follows from Shannon's coding theorem and can be worked out in a straight-forward manner by computing the mutual information between the channel input and the channel output [Cover and Thomas 1991]. So, for low BERs, systems of interest give performance points that lie between 0.0 dB and the curve for uncoded transmission Performance points to the left of 0.0 dB are impossible, and performance points to the right of the curve for uncoded transmission are inferior to simply sending the uncoded data.

Recently a code and decoding algorithm have been discovered that give unprecedented BER performance. It turns out that the *turbo-decoding* algorithm for these *turbo-codes* [Berrou and Glavieux 1996] is just the probability propagation algorithm discussed in Section 1.3 applied to a code network like the one shown in Figure 1.3b [Frey and Kschischang 1996; Kschischang and Frey 1997]. This Bayesian network contains two recursive convolutional code networks that are connected to the information bits in different ways. The information bits feed directly into one of the chains, but feed into the second chain in a *permuted order* as shown. In order to produce the same number of codeword bits per codeword as would be produced by the recursive systematic convolutional encoder described above, every second output of each FSM is alternately not transmitted (a procedure called *puncturing*).

Figure 1.3c shows the BER performance for a turbo-code system with $K = 5000$ and $N = 10000$. (Again, 5000 vectors were transmitted to determine the BER for each noise level). The two FSMs had the same z-transform, $X(z)/U(z) = 1 + z^4/1 + z^1 + z^2 + z^3 + z^4$. A memory size of 4 was chosen to make the decoding time equal to the decoding time for the single convolutional code described above. The information bit permuter was chosen at random. Note that the FSM z-transforms and the permuter were *not* chosen to optimize performance for the given memory size. The z-transforms were proposed in [Berrou, Glavieux and Thitimajshima 1993] for a $K = 65536$ system with a highly structured permuter. Even so, this turbo-code system clearly outperforms the computationally comparable single convolutional code system. At a BER of 10^{-5}, the turbo-code system is tolerant to roughly 2.5 dB more noise than the single convolutional code system.

In Chapter 4, I explore some of the exciting new applications of Bayesian networks to channel coding problems, with a focus on using the probability propagation algorithm discussed in Section 1.3 for inference.

1.2.5 Parameterized Bayesian networks

It is sometimes convenient to represent the conditional distributions $P(z_k|a(z_k))$ in parametric form. That is, the distribution over z_k given its parents $a(z_k)$ is specified not by an exhaustive list of values, but by a function of z_k, $a(z_k)$, and a set parameters $\boldsymbol{\theta}_{z_k}$. (The subscript z_k indicates that $\boldsymbol{\theta}_{z_k}$ is the set of parameters associated with z_k and should not be interpreted as a variable in this context.) In this case, we write the conditional distribution as $P(z_k|a(z_k), \boldsymbol{\theta}_{z_k})$. The total set of parameters is $\boldsymbol{\theta} = (\boldsymbol{\theta}_{z_1}, \boldsymbol{\theta}_{z_2}, \dots, \boldsymbol{\theta}_{z_N})$, and the parameterized joint distribution is expressed as $P(\mathbf{z}|\boldsymbol{\theta})$. Such a parametric form can be useful in applications such as density estimation, pattern classification, and data compression, where the distribution $P(\mathbf{z}|\boldsymbol{\theta})$ is to be estimated from a data set. In this case, the parametric form can act as a regularizer; since the number of possible configurations of each z_k and $a(z_k)$ is usually quite large, we'd need an extremely large data set in order to estimate each value accurately. On the other hand, using the parametric form, we need only estimate each parameter. As described in Section 1.5, a parametric form is also useful when formulating variational inference algorithms.

A common parametric Bayesian network is the binary sigmoidal Bayesian network [Neal 1992; Jordan 1995; Saul, Jaakkola and Jordan 1996], whose random variables are all binary-valued. The conditional probability function $P(z_k|a(z_k), \boldsymbol{\theta}_{z_k})$ can be viewed as a regression model that is meant to predict z_k from a set of attributes $a(z_k)$. A standard statistical method for predicting a binary-valued variable is *logistic regression* [McCullagh and Nelder

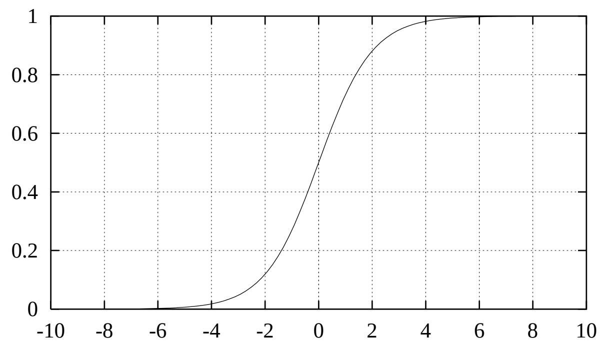

Figure 1.4: The logistic function $g(x) = 1/(1+\exp[-x])$.

1983]. The conditional probability for z_k given $a(z_k)$ is

$$P(z_k|a(z_k), \boldsymbol{\theta}_{z_k}) = \begin{cases} 1/(1+\exp[-\theta_{z_k z_0} - \sum_{\forall j: z_j \in a(z_k)} \theta_{z_k z_j} z_j]) & \text{if } z_k = 1, \\ 1 - 1/(1+\exp[-\theta_{z_k z_0} - \sum_{\forall j: z_j \in a(z_k)} \theta_{z_k z_j} z_j]) & \text{if } z_k = 0, \end{cases} \tag{1.21}$$

where the parameters $\theta_{z_k z_0}$ and $\{\theta_{z_k z_j}\}_{\forall j: z_j \in a(z_k)}$ essentially relate the probability that z_k has the value 1 *linearly* to the values of it parents. ($\theta_{z_k z_0}$ is just a bias in the exponent.) The logistic function $g(x) = 1/(1+\exp[-x])$ is used to restrict the probability to lie between 0 and 1. (This function is shown in Figure 1.4.) In the log-odds domain, we have

$$\log \frac{P(z_k = 1|a(z_k), \boldsymbol{\theta}_{z_k})}{P(z_k = 0|a(z_k), \boldsymbol{\theta}_{z_k})} = \theta_{z_k z_0} + \sum_{\forall j: z_j \in a(z_k)} \theta_{z_k z_j} z_j, \tag{1.22}$$

which shows how each parent $z_j \in a(z_k)$ independently increases or decreases the log-odds for z_k, depending on the sign of $\theta_{z_k z_j}$.

Sometimes, for the sake of notational simplicity, I will assume that the set of parents for each variable z_k is specified by some parameter constraints. Assume without loss of generality that for a given network, the random variables $\mathbf{z}$ have an ancestral ordering $z_1, z_2, \ldots, z_N$. I take $P(z_k|a(z_k), \boldsymbol{\theta}_{z_k}) = P(z_k|\{z_j\}_{j=1}^{k-1}, \boldsymbol{\theta}_{z_k})$, where in the second expression the parameters are constrained so that the function does not depend on nonparents. Also, in order to succinctly account for the bias, I will usually assume that there is a dummy variable z_0 that is set to $z_0 = 1$. (Thus the notation $\theta_{z_k z_0}$ for the bias in the summations above.) Using these notational simplifications and using $g(\cdot)$ for the logistic function, the sigmoidal model described above can be written

$$P(z_k|\{z_j\}_{j=1}^{k-1}, \boldsymbol{\theta}_{z_k}) = z_k g(\textstyle\sum_{j=0}^{k-1} \theta_{z_k z_j} z_j) + (1 - z_k)(1 - g(\sum_{j=0}^{k-1} \theta_{z_k z_j} z_j)), \tag{1.23}$$

where in the parameter vector $\boldsymbol{\theta}_{z_k} = (\theta_{z_k z_0}, \theta_{z_k z_1}, \ldots, \theta_{z_k z_N})$, we constrain $\theta_{z_k z_j} = 0$ for each nonparent z_j.

1.2.6 Example 2: The bars problem

Bayesian networks provide a useful framework for specifying *generative models*. A generative model can be used to generate data vectors that exhibit interesting structure. The generative models discussed in this thesis can also be used for pattern classification and data compression, in the fashion described in Sections 1.1.1 and 1.1.2. If the Bayesian network is parameterized, we can estimate the parameters of the network from a training set by making the generative distribution "close" (say, in the Kullback-Leibler pseudo-distance) to the training set distribution. We hope that in this fashion, we can extract the "true" underlying generative process, or at least one that is equally efficient at describing the data.

For example, the 4×4 binary images shown in Figure 1.5a were generated by first selecting an orientation (horizontal or vertical) with equal probability, and then randomly instantiating each of the four possible bars with probability 0.5. Using the Helmholtz machine and the wake-sleep algorithm (described in Sections 1.6 and 2.4.3), I fit the parameterized network shown in Figure 1.5b to a large training set of 2×10^6 images produced in this way. The network has three layers of binary variables: 1 in the top layer, 8 in the middle layer, and 16 in the visible layer (the image). The variables in adjacent layers are fully-connected, and the conditional distributions are modelled using logistic regression, as described in the previous section. After parameter estimation (see [Hinton et al. 1995] for details), ancestral simulation of the network produces output images that are indistinguishable from the training images.

After learning, the bias for the top-layer variable is nearly zero, so that under the joint distribution it has the value 1 as often as it has the value 0. The values of the other parameters are depicted in Figure 1.5c. The eight large blocks on the left show the parameters associated with the connections that feed into and out of the middle-layer variables. The bias for a variable is shown by the small black or white square on the top right of the block for each middle-layer variable. Positive parameters are white, negative parameters are black, and the area of the square is proportional to the magnitude of the parameter. (The largest parameter shown in the figure is 14.1.) The parameter associated with the connection from the top-layer variable to a middle-layer variable is shown by the small square on the top left of the block for each middle-layer variable. Finally, the parameters associated with the connections from a middle-layer variable to the visible variables are shown by the 4×4 grid of squares in the block for each middle-layer variable. The biases for the 16 visible variables are shown by the 4×4 grid of squares on the far right of Figure 1.5c.

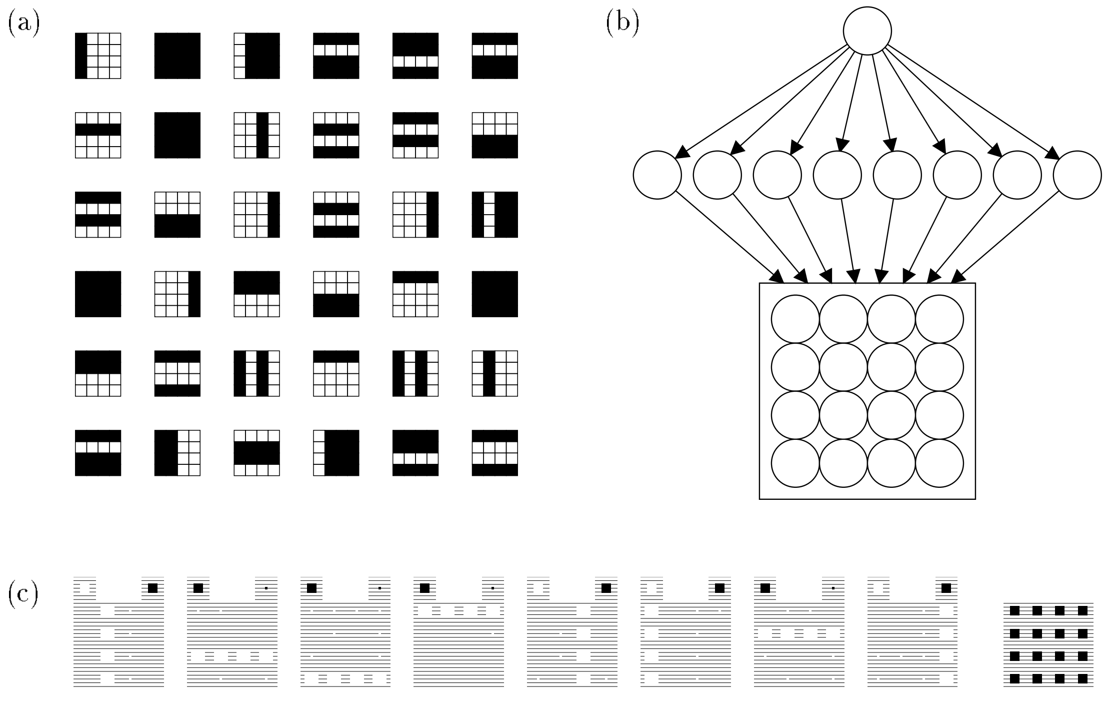

Figure 1.5: (a) Examples of training images typical of the "bars problem". (b) The graph for a parameterized Bayesian network that was estimated from a large training set using the wake-sleep algorithm. Edges that terminate on a box are connected to all vertices within the box. (c) The parameters of the Bayesian network clearly show that the network has learned the notion of horizontal and vertical bars (see the main text for a more complete description).

It is clear from these parameters that each middle-layer variable represents the presence (value of 1) or absence (value of 0) of a particular horizontal or vertical bar. If the top-layer variable is 1, the probability that a horizontal bar is present is nearly zero, since the biases for these variables are nearly zero and the parameters that connect these variables to the top-layer variable are large and negative. On the other hand, if the top-layer variable is 0, the probability that a horizontal bar is present is 0.5. In this way, the network captures the true generative model that produced the training data.

In Chapters 2 and **??**, I show how Monte Carlo inference, variational inference, and Helmholtz machines can be used to fit Bayesian networks to training data for the purposes of pattern classification, unsupervised learning, and data compression.

1.2.7 Dependency separation

In Section 1.2.2, I used the notion of conditional independence to define a Bayesian network. In particular, a Bayesian network implies that variable z_k is conditionally independent of any subset of its nondescendents, given its parents. This is expressed mathematically by

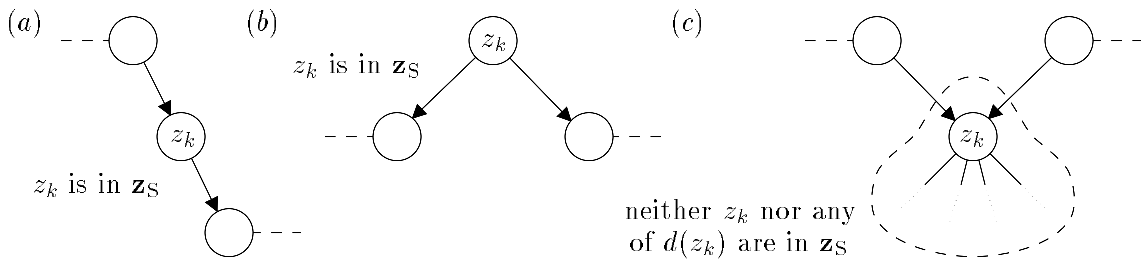

Figure 1.6: The three ways in which a path may be blocked. If all paths from all variables in $\mathbf{z}_{\mathrm{A}}$ to all variables in $\mathbf{z}_{\mathrm{B}}$ are blocked, then $\mathbf{z}_{\mathrm{A}}$ and $\mathbf{z}_{\mathrm{B}}$ are dependency-separated by $\mathbf{z}_{\mathrm{S}}$.

(1.12). A convenient way to describe this scenario is to say that "z_k is dependency-separated from any combination of its nondescendents by its parents".

In general, I will say that $\mathbf{z}_{\mathrm{A}}$ is dependency-separated from $\mathbf{z}_{\mathrm{B}}$ by $\mathbf{z}_{\mathrm{S}}$ ("S" for separation), if and only if [Pearl 1988]

$$P(\mathbf{z}_{\mathrm{A}}|\mathbf{z}_{\mathrm{S}}, \mathbf{z}_{\mathrm{B}}) = P(\mathbf{z}_{\mathrm{A}}|\mathbf{z}_{\mathrm{S}}), \quad \text{or, equivalently} \quad P(\mathbf{z}_{\mathrm{A}}, \mathbf{z}_{\mathrm{B}}|\mathbf{z}_{\mathrm{S}}) = P(\mathbf{z}_{\mathrm{A}}|\mathbf{z}_{\mathrm{S}})P(\mathbf{z}_{\mathrm{B}}|\mathbf{z}_{\mathrm{S}}). \tag{1.24}$$

Notice that dependency-separation is symmetrical with respect to $\mathbf{z}_{\mathrm{A}}$ and $\mathbf{z}_{\mathrm{B}}$. The case of dependency-separation that I used to define a Bayesian network is special, in that $\mathbf{z}_{\mathrm{S}}$ was the set of parents of the single variable $\mathbf{z}_{\mathrm{A}}$, and $\mathbf{z}_{\mathrm{B}}$ was a subset of the nondescendents of $\mathbf{z}_{\mathrm{A}}$.

It is possible to ascertain dependency separation in the general case simply by inspecting the Bayesian network. If $\mathbf{z}_{\mathrm{A}}$, $\mathbf{z}_{\mathrm{B}}$, and $\mathbf{z}_{\mathrm{S}}$ are three disjoint subsets of $\mathbf{z}$, then $\mathbf{z}_{\mathrm{S}}$ dependency-separates $\mathbf{z}_{\mathrm{A}}$ from $\mathbf{z}_{\mathrm{B}}$ if, in every path connecting any variable in $\mathbf{z}_{\mathrm{A}}$ to any variable in $\mathbf{z}_{\mathrm{B}}$, there is at least one variable z_k that satisfies one or more of the following three conditions:

1. z_k acts as both a parent and a child in the path and z_k is in $\mathbf{z}_{\mathrm{S}}$ (Figure 1.6a), or

2. z_k acts as the parent of two variables in the path and z_k is in $\mathbf{z}_{\mathrm{S}}$ (Figure 1.6b), or

3. z_k acts as the child of two variables in the path and neither z_k nor any of its descendents are in $\mathbf{z}_{\mathrm{S}}$ (Figure 1.6c).

A path for which one of these conditions is met is said to be *blocked*. Note that in order to ascertain dependency-separation, we need to consider only paths that do not intersect themselves, since those conditions that hold for any given path will also hold for the path when extra variables that form a loop are considered.

For example, in Figure 1.1a we have the following dependency separation relationships. z_6 is dependency-separated from z_{10} by z_9, by condition 1 applied to the sole path z_6, z_9, z_{13}, z_{10}. This means that if z_9 is observed, then z_6 and z_{10} are independent. z_6 is not dependency-separated from z_{10} by z_{13}, since there exists a path z_6, z_9, z_{13}, z_{10} for which none of the conditions can be met. This means that once z_{13} is observed, z_6 and z_{10} may still be dependent. z_2 is dependency-separated from z_9 by z_5, since path z_2, z_5, z_9 is blocked by z_5 in condition 1, paths $z_2, z_4, z_8, z_{12}, z_9$ and $z_2, z_1, z_4, z_8, z_{12}, z_9$ are blocked by z_{12} in condition 3, paths z_2, z_4, z_8, z_5, z_9 and $z_2, z_1, z_4, z_8, z_5, z_9$ are blocked by z_5 in condition 2 *and* by z_8 in condition 3, path $z_2, z_5, z_8, z_{12}, z_9$ is blocked by z_5 in condition 1 *and* by z_{12} in condition 3. z_2 is dependency-separated from (z_3, z_7) (by nothing), since the paths z_2, z_5, z_3 and z_2, z_5, z_3, z_7 are blocked by z_5 in condition 3. This means that in the absence of observations z_2 and (z_3, z_7) are independent. z_2 is not dependency-separated from (z_3, z_7) by z_{12}, since there exists a path z_2, z_5, z_3 for which none of the conditions can be met. Condition 3 applied to z_5 fails, because z_{12} is a descendent of z_5. This means that once z_{12} is observed, z_2 and (z_3, z_7) may *become* dependent.

1.3 Exact inference in singly-connected Bayesian networks

In 1986, Pearl [1986; 1988] developed an exact algorithm called *probability propagation* for inferring the distributions over individual variables in singly-connected Bayesian networks. A *singly-connected* network has only a single path (ignoring edge directions) connecting any two vertices (see Figure 1.7 for an example). Not only does the algorithm make use of the probabilistic structure implied by a Bayesian network, but it also uses the network as a circuit that specifies data flow (or *message passing*) channels for inference computations. In a singly-connected network, cutting an edge breaks the network into two pieces. So, each edge acts as a message passing bottle-neck for communicating information regarding one "side" of the network to the other. This leads to the idea of *articulation*.

1.3.1 Articulation

Consider the edge connecting z_4 to z_7 in Figure 1.7. We wish to pass messages bidirectionally on this edge in order to assist probabilistic inference. Since the network is singly-connected, there is only one path between z_4 and z_7. So, the message travelling from z_4 to z_7 must contain all relevant information in the subnetwork that contains z_4 when the edge is removed. In this sense, each vertex *articulates* for its neighbors the relevant information in the respective subnetworks. For example, z_4 articulates for z_7 relevant information in the subnetwork $N_{z_4 \mathbf{A} z_7}$ shown by the large dashed loop in Figure 1.7. (The notation $N_{z_4 \mathbf{A} z_7}$ is

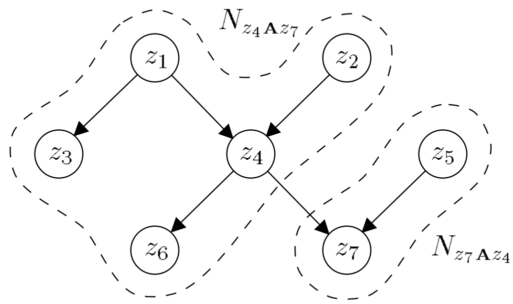

Figure 1.7: A singly-connected Bayesian network.

to be read "the network that z_4 articulates for z_7".) Also shown is the network $N_{z_7 \mathbf{A} z_4}$ that z_7 articulates for z_4.

In the next section, I show how messages of reasonable size can be combined at each node with a reasonable number of computations in order to perform exact probabilistic inference for individual variables. It turns out that each message is a vector with as many elements as there are states for the parent variable of the edge.

1.3.2 Probability propagation (the generalized forward-backward algorithm)

Probability propagation is a method of propagating the effects of one or more observations across a Bayesian network so that each unobserved node can compute the probability for its variable *conditioned* on the observed variables. Figure 1.8 shows a network fragment consisting of a variable r, its parents $\{q_i\}_{\forall i}$, and its children $\{v_j\}_{\forall j}$. Some of the other variables not shown have been observed and we wish to determine the effect of these observations on the probability of r; that is, we want to compute $P(r|\mathbf{O})$ where $\mathbf{O}$ is the set of observed variables. Since the network is singly-connected, each parent and child of r articulates the observations in the respective subnetwork. Let $\mathbf{O}_{v_j \mathbf{A} r}$ be the observations that v_j articulates for r. Similarly, let $\mathbf{O}_{q_i \mathbf{A} r}$ be the observations that q_i articulates for r.

To begin with, assume that the probability messages $\{P(q_i|\mathbf{O}_{q_i \mathbf{A} r})\}_{\forall i}$ and $\{P(\mathbf{O}_{v_j \mathbf{A} r}|r)\}_{\forall j}$ entering r have already somehow been computed. Given these messages and a *local* part of the network specification $P(r|\{q_i\}_{\forall i})$, we can compute $P(r|\mathbf{O})$ using the following formula:

$$P(r|\mathbf{O}) \overset{r}{\propto} \Big[\prod_{\forall j} P(\mathbf{O}_{v_j \mathbf{A} r}|r)\Big] \sum_{\{q_i\}_{\forall i}} \Big[P(r|\{q_i\}_{\forall i}) \prod_{\forall k} P(q_k|\mathbf{O}_{q_k \mathbf{A} r})\Big], \tag{1.25}$$

where the symbol "$\overset{r}{\propto}$" indicates equality up to a constant that is independent of r. (See

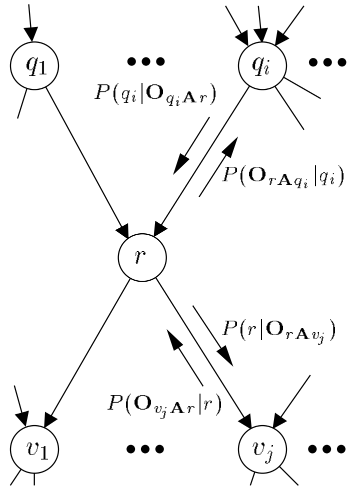

Figure 1.8: A network fragment shows the parents $\{q_i\}_{\forall i}$ and the children $\{v_j\}_{\forall j}$ of variable r in addition to some of the incoming and outgoing probability messages.

[Pearl 1988] for derivations.) In order to get the actual value of $P(r|\mathbf{O})$, the quantity computed from the right hand side of (1.25) must be scaled so that $\sum_r P(r|\mathbf{O}) = 1$. The most significant computational burden in obtaining $P(r|\mathbf{O})$ from the incoming messages is usually the summation over all the possible configurations $\{q_i\}_{\forall i}$ of the parents of r.

The outgoing parent-child messages $\{P(r|\mathbf{O}_{r\mathbf{A}v_j})\}_{\forall j}$ are computed as follows:

$$P(r|\mathbf{O}_{r\mathbf{A}v_j}) \overset{r}{\propto} P(r|\mathbf{O})/P(\mathbf{O}_{v_j\mathbf{A}r}|r). \tag{1.26}$$

Again, this message can be easily normalized. Notice that once $P(r|\mathbf{O})$ has been computed, the computation of this message is very simple.

Up to a scaling factor, the outgoing child-parent messages $\{P(\mathbf{O}_{r\mathbf{A}q_i}|q_i)\}_{\forall i}$ are

$$P(\mathbf{O}_{r\mathbf{A}q_i}|q_i) \overset{q_i}{\propto} \sum_r \left(\left[\prod_{\forall j} P(\mathbf{O}_{v_j\mathbf{A}r}|r) \right] \sum_{\{q_k\}_{\forall k \neq i}} \left[P(r|\{q_k\}_{\forall k}) \prod_{\forall l \neq i} P(q_l|\mathbf{O}_{q_l\mathbf{A}r}) \right] \right). \tag{1.27}$$

In this case, since q_i is the condition for $P(\mathbf{O}_{r\mathbf{A}q_i}|q_i)$, this probability can't be easily normalized. (To normalize it would require a summation over all the possible observation configurations $\mathbf{O}_{r\mathbf{A}q_i}$ that r articulates for q_i.) However, it turns out that these child-parent messages can be used in un-normalized form. They are used to compute (1.25) and (1.26), which can be normalized.

These update equations provide a consistent way to propagate conditional probability messages across a singly-connected network. When the network is *multiply-connected* (*i.e.*, not singly-connected), the propagation rules can still be applied, but there is no guarantee as to the quality of the results. In some applications, the results although approximate are still very useful. Most notably, the new and very impressive "turbo-decoding" error-correcting decoding algorithm is an example of probability propagation in multiply-connected Bayesian networks.

1.3.3 Buffered messages

The above update rules do not prescribe a *schedule* for passing the messages. When r receives a message from one of its children, must r immediately and in parallel propagate messages to its other children and its parents? It turns out that when the network is singly-connected, the propagation schedule does not matter, as long as all of the messages are eventually *absorbed*, as described below. Consequently, it is often useful to buffer messages. For example, if we know that r will soon receive messages from all the other children, we can pass fewer messages in total by waiting for those others to arrive before applying the update rules (1.25), (1.26) and (1.27). Pictorially, when a message arrives on an edge, but is not propagated on to the other neighbors, we draw a small dot on each of the other edges. These dots can be turned into arrows (indicating a message is being passed) at any time.

1.3.4 Message creation and absorption

Suppose that in Figure 1.8, r is observed and has the value r'. This observation will cause new messages to be sent to the parents and the children of r. Obviously, $P(r|\mathbf{O}) = \delta(r, r')$, where the indicator function $\delta(\cdot)$ satisfies $\delta(r, r') = 1$ if $r = r'$ and 0 otherwise. It follows from (1.26) that the normalized parent-child messages $\{P(r|\mathbf{O}_{r\mathbf{A}v_j})\}_{\forall j}$ are determined from

$$P(r|\mathbf{O}_{r\mathbf{A}v_j}) = \delta(r, r'). \tag{1.28}$$

In order to determine the newly created child-parent messages $\{P(\mathbf{O}_{r\mathbf{A}q_i}|q_i)\}_{\forall i}$, we pretend that r is not observed and imagine a fictitious child node z that *is* observed and to which r is connected deterministically: $P(z|r) = \delta(z, r)$. The authentic observation is equivalent to the fictitious observation, $z = r'$. When determining the child-parent messages from (1.27), the term $\prod_j P(\mathbf{O}_{v_j\mathbf{A}r}|r)$ will now include the fictitious child z. Consequently, this term will be zero for all but the case $r = r'$. The child-parent messages are thus determined using

$$P(\mathbf{O}_{r\mathbf{A}q_i}|q_i) \overset{q_i}{\propto} \sum_{\{q_k\}\forall k\neq i} \Big[P(r = r'|\{q_k\}) \prod_{\forall l\neq i} P(q_l|\mathbf{O}_{q_l\mathbf{A}r})\Big]. \tag{1.29}$$

Once created, these messages will continue to propagate across the network, creating new ones as they go. As mentioned above, it turns out that this process eventually terminates. When r receives a message from one of its parents or children, new messages are sent to all *other* parents and children. So, if r is a vertex that is connected by a single edge, the received message will be *absorbed*.

1.3.5 Network initialization

Before propagating the effects of observations across the network, we must produce initial messages that correspond to the *a priori* status of the network. We obtain the initial child-parent messages by noting that $\mathbf{O} = \varnothing$ and so

$$P(\mathbf{O}_{r\mathbf{A}q_i}|q_i) = P(\varnothing|q_i) = 1. \tag{1.30}$$

Also,

$$P(r|\mathbf{O}_{r\mathbf{A}v_j}) = P(r|\varnothing) = P(r) = \sum_{\{q_i\}\forall i} \Big[P(r|\{q_i\}) \prod_{\forall k} P(q_k) \Big]. \tag{1.31}$$

So, if r is a root node, it simply passes its marginal probability $P(r)$ to its children. If r is not a root node, it uses the marginal probabilities of its parents to compute $P(r)$. These messages propagate from parent to child until each variable has an initial probability.

1.3.6 Grouping variables and duplicating variables

Often, it is possible to convert a multiply-connected Bayesian network to a singly-connected Bayesian network, so that probability propagation can then be applied in a practical manner. To do this, we *group* variables, until there are no more multiple paths in the network. Graphically, two variables z_{j_1} and z_{j_2} are grouped by removing from the graph z_{j_1} and z_{j_2} as well as the edges to which they are connected, and then introducing a new vector variable (z_{j_1}, z_{j_2}). The set of parents of the new vector variable is the union of the sets of parents of the two old variables: $a((z_{j_1}, z_{j_2})) = a(z_{j_1}) \cup a(z_{j_2})$. Similarly for the children: $c((z_{j_1}, z_{j_2})) = c(z_{j_1}) \cup c(z_{j_2})$. New edges are introduced to reflect these relationships. This grouping operation will produce a valid Bayesian network as long as z_{j_2} is not an indirect descendent of z_{j_1} and *vice versa*. Otherwise, a directed cycle will result from the grouping, violating the requirement that a Bayesian network have no directed cycles. Note that if z_{j_2} is a child of z_{j_1}, and at the same time *not* an indirect descendent, the grouping is still valid, since no directed cycles are produced.

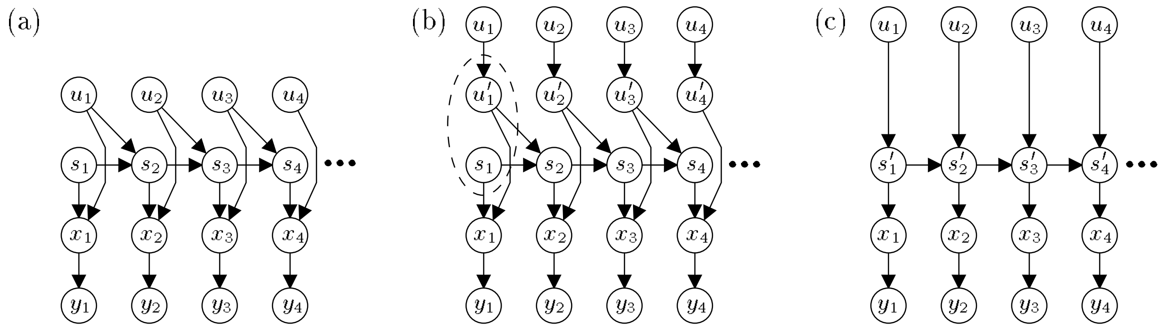

Figure 1.9: Transforming the multiply-connected Bayesian network for a recursive convolutional code (a) into a singly-connected network (c), involves variable *duplication* (a)-(b) and *grouping* (b)-(c).

As shown in Section A.2, this grouping operation also preserves the representational capacity of the network. Any distribution represented by the old network can be represented by the new one. In fact, all of the conditional probabilities $P(z_k|a(z_k))$ in the new network will be the same as in the old network, except the ones that involve either of the grouped variables. The latter conditional probabilities can quite easily be derived from the old ones.

Note that although grouping variables may help to produce a singly-connected network to which probability propagation can be applied, the grouping operation also hides the structure that makes probability propagation an attractive inference method in the first place. So, it is important to produce minimal groupings and retain as much of the structure as possible. Not surprisingly, by grouping variables, *any* network can be made singly-connected simply by grouping all variables together into a single vertex. However, extreme groupings of this sort usually eliminate too much structure. "Probability propagation" for the single vertex is equivalent to manipulating the joint distribution, which in most practical cases in unwieldy.

Another useful operation is *duplicating* variables. A variable z_j can be duplicated by removing it from the graph and inserting two new variables z_j' and z_j''. We create the following new parent-child relationships: $a(z_j') = a(z_j)$, $a(z_j'') = z_j'$, and $c(z_j'') = c(z_j)$. This procedure is especially useful in combination with grouping, since although we may wish to group z_{j_1} and z_{j_2} in order to make the network singly-connected, we may also wish to graphically distinguish z_{j_1} from the vector variable (z_{j_1}, z_{j_2}).

For example, the recursive convolutional code network shown in Figure 1.3a can be derived from the more natural recursive convolutional code network shown in Figure 1.9a. The latter network explicitly shows the dependence of the encoder state variable s_k on the previous information symbol u_{k-1} and the previous state s_{k-1}, as well as the dependence of the encoder output x_k on u_k and s_k. As with the former network, the memoryless channel is indicated by the dependence of each received signal y_k on *only* one transmitted symbol x_k. However, this network is multiply-connected, so probability propagation cannot be used to compute $P(u_k|\mathbf{y})$ for maximum *a posteriori* information symbol decoding. To convert the network to a singly-connected one, we first duplicate the information symbols (so that they are graphically distinguished in the final network) as shown in Figure 1.9b. Then, we group pairs of information symbols and state variables as shown by the dashed loop, producing the singly-connected network shown in Figure 1.9c. Note that by grouping variables in this way, the number of values that each new state s'_k can take on is increased.

Although in many cases grouping can be used to produce a tractable network, there are cases where it is impossible to find an appropriate grouping. In fact, it turns out that in general, inference in multiply-connected networks is a very difficult problem.

1.3.7 Exact inference in multiply-connected networks is NP-hard

Probability propagation is an exact method of inference for *singly-connected* Bayesian networks. Cooper [1990] has shown that probabilistic inference in Bayesian networks is *in general* NP-hard. Summations relevant to inference, such as the ones in (1.3), (1.7), and (1.9), contain an exponential number of terms that in general cannot be simplified. Researchers have thus focused on developing exact inference algorithms for restricted classes of networks (*e.g.*, probability propagation for singly-connected networks), and on developing approximate inference algorithms for networks that are intractable.

One obvious approach to approximate inference in a multiply-connected Bayesian network is to use the probability propagation algorithm while ignoring the fact that the network is multiply-connected. Each vertex articulates messages as if the network were singly-connected. In this case, the propagation procedure will never terminate, because there will be loops in which messages will endlessly circulate. Although this method has provided excellent results in the area of channel coding, it is not favored in many other research areas because there is little theoretical understanding of what this iterative procedure converges to.

Another disadvantage of the probability propagation algorithm is that it is not suited to inferring the *joint* distribution over several variables (*e.g.*, u_1 and u_4 in Figure 1.9c). Of course, we can group the variables we are interested in and then apply probability propagation. However, in many cases the grouping operation will produce a network that is no longer singly-connected. (*E.g.*, if we group u_1 and u_4 in Figure 1.9c, we will obtain the cycle (u_1, u_4)—s'_1—s'_2—s'_3—s'_4—(u_1, u_4).) Again, we could apply the probability propagation algorithm anyway, while ignoring any emergent cycles.

In the following sections, I describe several more principled approaches to approximate probabilistic inference, including Monte Carlo, variational inference, and Helmholtz machines.

1.4 Monte Carlo inference

The Monte Carlo method [Hammersley and Handscomb 1964; Kalos and Whitlock 1986; Ripley 1987] makes use of random numbers in order to perform computations. If we can somehow obtain a reasonably large sample from the distribution $P(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$ over some unobserved *hidden* variables (denoted by "H") in a Bayesian network given some observed *visible* variables (denoted by "V"), then relative frequencies can be used for approximate inference.

One brute force Monte Carlo approach is to simply simulate the network using the ancestral ordering. Then, we extract from the sample all those vectors that have the desired value for the component $\mathbf{z}_\mathrm{V}$. Next, we compile a frequency histogram for the different values that $\mathbf{z}_\mathrm{H}$ can take on. Although this approach is sometimes useful (notably, when using a small network to verify that a more sophisticated inference method works), in general it is not computationally efficient. The problem is that the value of $\mathbf{z}_\mathrm{V}$ that we wish to condition on may occur *extremely* rarely, so that an inordinate sample size must be used in order to obtain results.

1.4.1 Inference by ancestral simulation

In some cases, it so happens that ancestral simulation can be used to obtain a sample from $P(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$ in an efficient manner. In general, if the parents of the visible variables are dependency-separated (see Section 1.2.7) from the hidden variables by the visible variables, then ancestral simulation can be used to obtain a sample from $P(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$ (see Section **??** for a proof). (Note that if the visible variables have no parents, then it follows trivially that the variables in this null set are dependency-separated from the hidden variables.) Using the ancestral ordering, a value is drawn for each *hidden* variable given its parents. After one complete sweep, the value for $\mathbf{z}_\mathrm{H}$ will be an unbiased draw from $P(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$.

For example, suppose that in the multiply-connected network with ancestral ordering $z_1, z_2, z_3, z_4, z_5, z_6, z_7$ shown in Figure 1.10, the set of visible variables is (z_1, z_4). Suppose also that we'd like to infer the distribution over the set of hidden variables (z_6, z_7). Since (z_6, z_7) is dependency-separated by (z_1, z_4) from the parents (z_2) of (z_1, z_4), we can estimate

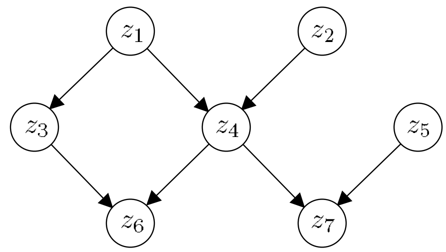

Figure 1.10: An example of a Bayesian network.

$P(z_6, z_7|z_1, z_4)$ by ancestral simulation. We draw a value for z_2, then for z_3 given z_1, then for z_5, then for z_6 given z_3 and z_4, then for z_7 given z_4 and z_5. We can estimate $P(z_6, z_7|z_1, z_4)$ by repeating this procedure over and over while building up a histogram for (z_6, z_7).

Notice that in the above example, it was not really necessary to draw a value for z_2, since for the ancestral simulation method to work, it was *required* that (z_2) be dependency-separated from (z_6, z_7) by (z_1, z_4); *i.e.*, given (z_1, z_4), z_2 does not influence (z_6, z_7). In general, values need only be drawn for those variables in the ancestral ordering that are *not* dependency-separated from $\mathbf{z}_\mathrm{H}$ by $\mathbf{z}_\mathrm{O}$. If these can be easily isolated, then simulation computations can be saved.

1.4.2 Gibbs sampling

When inference by ancestral simulation is not possible, Markov chain Monte Carlo is often used (see an excellent review of these methods by Neal [1993]). Given $\mathbf{z}_\mathrm{V}$, a temporal sequence of the hidden variables $\mathbf{z}_\mathrm{H}^{(1)}, \mathbf{z}_\mathrm{H}^{(2)}, \ldots$ is produced by simulating a Markov chain whose stationary distribution is carefully constructed to be equal to $P(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$. An approximate sample $\mathbf{z}_\mathrm{H}^{(t_1)}, \mathbf{z}_\mathrm{H}^{(t_2)}, \ldots$ is obtained by choosing time indices $t_1, t_2, \ldots$ so that the sample elements are as independent as possible. Often, a moderate degree of dependence must be tolerated in order to keep the simulation time within reason. Once collected, the sample can be used to produce a frequency histogram of the subset $\mathbf{z}_\mathrm{H}$ of $\mathbf{z}_\mathrm{H}$.

The *Gibbs sampling* algorithm is the simplest of the Markov chain Monte Carlo methods and has been successfully applied to Bayesian networks [Pearl 1987; Pearl 1988; Neal 1992] as well as other graphical models [Geman and Geman 1984; Hinton and Sejnowski 1986]. In this algorithm, each successive state $\mathbf{z}_\mathrm{H}^{(t)}$ is chosen by modifying only a single variable in the previous state $\mathbf{z}_\mathrm{H}^{(t-1)}$. The variables are usually modified in sequence, although they may be visited in other ways, including random order. If at time t, we have decided to modify

z_k, then we draw a value $z_k^{(t)}$ from

$$P(z_k|\{z_j = z_j^{(t-1)}\}_{\forall j \neq k}). \tag{1.32}$$

Usually, we cannot obtain this distribution exactly, but instead can only compute the joint probability which is proportional to the conditional probability:

$$P(z_k|\{z_j = z_j^{(t-1)}\}_{\forall j \neq k}) \overset{z_k}{\propto} P(z_k, \{z_j = z_j^{(t-1)}\}_{\forall j \neq k}). \tag{1.33}$$

The joint probability usually can be computed easily from (1.13). If z_k is discrete, we compute the joint probability for each value that it can take on, normalize these values, and then randomly draw a value $z_k^{(t)}$ from its normalized discrete distribution. However, if z_k is a continuous random variable, it may not be an easy task to draw a value from its distribution. Efficient methods for drawing values from several special types of continuous parametric distribution are given in [Devroye 1986] and [Ripley 1987]. In order to draw vales from other types of distribution, more sophisticated techniques such as rejection sampling [Gilks and Wild 1992] must be used.

1.4.3 The burglar alarm problem

In order to illustrate how Gibbs sampling works, in this section I use a variant of the "burglar alarm" network described by Pearl [1988]. The network describes a shoddy burglar alarm that is sensitive not only to burglars, but also to earthquakes. The three binary random variables in the network are b for "burglar", e for "earthquake", and a for "alarm"; a value of 0 indicates the cause is absent or the alarm is off, whereas a value of 1 indicates the cause is present or the alarm is ringing. Figure 1.11a shows the network, which has the following conditional probability relationships:

$$\begin{gathered} P(b=1) = 0.1, \qquad P(e=1) = 0.1, \\ P(a=1|b=0, e=0) = 0.001, \quad P(a=1|b=1, e=0) = 0.368, \\ P(a=1|b=0, e=1) = 0.135, \quad P(a=1|b=1, e=1) = 0.607. \end{gathered} \tag{1.34}$$

Suppose that while away at a conference, the burglar alarm contacts us by cell phone and informs us that the alarm is ringing ($a=1$). We would like to infer the distribution over the two causes. Since this network is quite simple, we can apply Bayes rule $P(b, e|a) =$

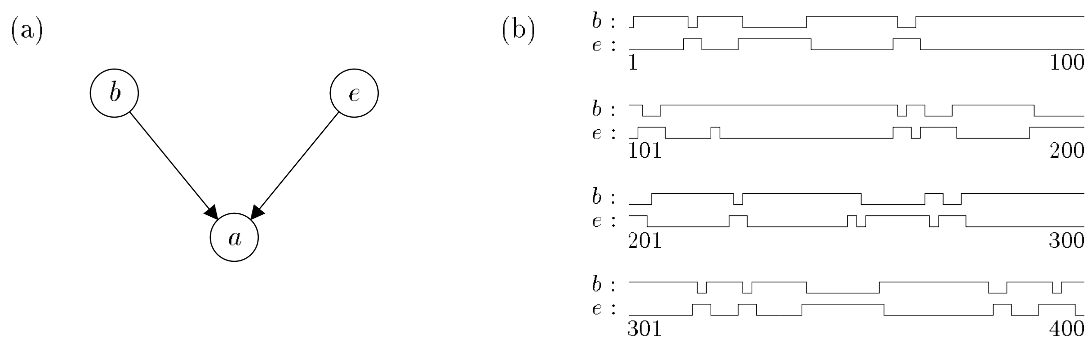

Figure 1.11: (a) The Bayesian network for the "burglar alarm" problem, with variables b ("burglar"), e ("earthquake") and a ("alarm"). (b) 400 steps of Gibb's sampling for the variables b and e when the alarm is observed to be ringing ($a = 1$).

$P(a|b,e)P(b)P(e)/\sum_{b',e'} P(a|b',e')P(b')P(e')$ to obtain the exact solution:

$$\begin{aligned}
P(b\!=\!0, e\!=\!0|a\!=\!1) &= 0.00081/0.05215 = 0.016, \\
P(b\!=\!1, e\!=\!0|a\!=\!1) &= 0.03312/0.05215 = 0.635, \\
P(b\!=\!0, e\!=\!1|a\!=\!1) &= 0.01215/0.05215 = 0.233, \\
P(b\!=\!1, e\!=\!1|a\!=\!1) &= 0.00607/0.05215 = 0.116. \qquad (1.35)
\end{aligned}$$

The most likely explanation for the ringing alarm is that a burglary took place. Notice, however, that although an earthquake is also a likely explanation, it is relatively *unlikely* that both a burglar and an earthquake were simultaneously the cause of the alarm. If we apply probability propagation to this network, we obtain $P(b = 1|a = 1) = 0.751$ and $P(e\!=\!1|a\!=\!1) = 0.349$, which tells us nothing about the covariance between the two causes. The Gibb's sampling method can reveal this relationship.

In order to perform Gibb's sampling, we need the probabilities for each of the hidden variables conditioned on *all* the other variables. Since these conditional probabilities are proportional to the joint probabilities, we can compute them in the following way, using

$P(b=1|e=0,a=1)$ as an example:

$$\begin{aligned} P(b=1|e=0,a=1) &= \frac{P(b=1,e=0,a=1)}{P(e=0,a=1)} \\ &= \frac{P(b=1,e=0,a=1)}{P(b=0,e=0,a=1)+P(b=1,e=0,a=1)} \\ &= \frac{0.03312}{0.00081+0.03312} = 0.976, \end{aligned} \tag{1.36}$$

where $P(b,e,a) = P(a|b,e)P(b)P(e)$ is the joint distribution. Similarly,

$$\begin{aligned} P(b=1|e=1,a=1) &= \frac{0.00607}{0.01215+0.00607} = 0.333, \\ P(e=1|b=0,a=1) &= \frac{0.01215}{0.00081+0.01215} = 0.938, \\ P(e=1|b=1,a=1) &= \frac{0.00607}{0.03312+0.00607} = 0.155. \end{aligned} \tag{1.37}$$

Gibbs sampling proceeds by alternately visiting b and e, while sampling from $P(b|e,a=1)$ and $P(e|b,a=1)$ using the above formulas. Figure 1.11b shows the values of b and e for 400 steps of Gibbs sampling, starting from an initial configuration $(b=0,e=0)$. The Markov chain shows that the configurations $(b=0,e=0)$ and $(b=1,e=1)$ are unlikely compared to $(b=1,e=0)$ and $(b=0,e=1)$. The correct probabilities in (1.35) can be approximated using the relative frequencies computed from this chain:

$$\begin{aligned} \hat{P}(b=0,e=0|a=1) &= 0.010, \\ \hat{P}(b=1,e=0|a=1) &= 0.674, \\ \hat{P}(b=0,e=1|a=1) &= 0.228 \\ \hat{P}(b=1,e=1|a=1) &= 0.088. \end{aligned} \tag{1.38}$$

These are quite close to the correct values. Normally, an initial segment of the Markov chain is discarded when computing these statistics. The motivation for this procedure is that we'd like to have samples that are typical of the *equilibrium* distribution, not the initial configuration.

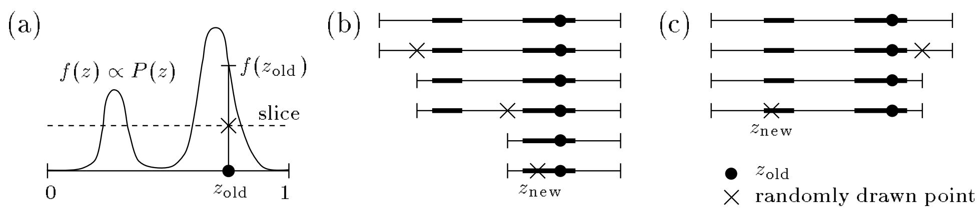

Figure 1.12: Slice sampling. After obtaining a random slice from the density (a), random values are drawn until one is accepted. (b) and (c) show two such sequences.

1.4.4 Slice sampling

In networks with continuous variables, it is often not an easy task to sample from the conditional distribution of each hidden variable, as Gibbs sampling requires. Unlike the case for discrete variables, it is not possible to compute the joint distribution for every configuration of a hidden variable. There are infinitely many configurations, and it is often practically impossible to determine an effective discretization. Methods for sampling from continuous distributions include the Metropolis algorithm [Metropolis et al. 1953; Neal 1993] and hybrid methods that use "momentum" in order to help search the configuration space [Duane et al. 1987; Neal 1993]. In this section, I review a technique called *slice sampling* [Neal 1997], that can be used for drawing a value z from a univariate distribution $P(z)$ — in the context of inference, $P(z)$ is the conditional distribution $P(z_k|\{z_j = z_j^{(t-1)}\}_{\forall j \neq k})$. Slice sampling does not directly produce values distributed according to $P(z)$, but instead produces a Markov chain that is guaranteed to converge to $P(z)$. At each step in the sequence, the old value z_{old} is used as a guide for where to pick the new value z_{new}.

To perform slice sampling, all that is needed is an efficient way to evaluate a function $f(z)$ that is *proportional* to $P(z)$ — in this application, the easily computed joint probability $P(z_k, \{z_j = z_j^{(t-1)}\}_{\forall j \neq k})$ is appropriate. Figure 1.12a shows an example of a univariate distribution, $P(z)$. The version of slice sampling discussed here requires that all of the density lies within a bounded *interval* as shown. To obtain z_{new} from z_{old}, $f(z_{\text{old}})$ is first computed and then a uniform random value is drawn from $[0, f(z_{\text{old}})]$. The distribution is then horizontally "sliced" at this value, as shown in Figure 1.12a. Any z for which $f(z)$ is greater than this value is considered to be part of the slice, as indicated by the bold line segments in the picture shown at the top of Figure 1.12b. Ideally, z_{new} would now be drawn uniformly from the slice. However, determining the line segments that comprise the slice is not easy, for although it is easy to determine whether a particular z is in the slice, it is much more difficult to determine the line segment boundaries, especially if the distribution is multimodal. Instead, a uniform value is drawn from the original interval as shown in the second picture of Figure 1.12b. If this value is in the slice it is accepted as z_{new} (note that this decision requires an evaluation of $f(z)$). Otherwise either the left or the right interval boundary is moved to this new value, while keeping z_{old} in the interval. This procedure is repeated until a value is accepted. For the sequence in Figure 1.12b, the new value is in the same mode as the old one, whereas for the sequence in Figure 1.12c, the new value is in the other mode. Once z_{new} is obtained, it is used as z_{old} for the next step.

1.5 Variational inference

In contrast to both the rather unprincipled approach of applying probability propagation to multiply-connected networks, and the computationally intensive stochastic approach of Monte Carlo, variational inference is a nonstochastic technique that directly addresses the quality of inference. Historically, variational inference methods [Saul, Jaakkola and Jordan 1996; Ghahramani and Jordan 1996; Jaakkola, Saul and Jordan 1996] were introduced as an alternative variation on the central theme of Helmholtz machines [Hinton et al. 1995; Dayan et al. 1995], which are described in Section 1.6. However, I will present variational inference first, because it is simpler to understand.

Suppose we are given a set of visible variables $\mathbf{z}_{\mathrm{V}}$, and that this set includes different variables from case to case. In order to solve the inference problem of estimating $P(\mathbf{z}_{\mathrm{H}}|\mathbf{z}_{\mathrm{V}})$, we introduce a parameterized variational distribution $Q(\mathbf{z}_{\mathrm{H}}|\boldsymbol{\xi})$ that is meant to approximate $P(\mathbf{z}_{\mathrm{H}}|\mathbf{z}_{\mathrm{V}})$. The most appropriate form of this distribution will depend on many factors, including the network specification and the quality of inference desired. Next, the distance between $P(\mathbf{z}_{\mathrm{H}}|\mathbf{z}_{\mathrm{V}})$ and $Q(\mathbf{z}_{\mathrm{H}}|\boldsymbol{\xi})$ (*e.g.*, Euclidean, relative entropy) is minimized with respect to $\boldsymbol{\xi}$, either directly or by using an optimization technique such as a Newton-like method or the conjugate gradient method [Fletcher 1987]. Once optimized, the distribution $Q(\mathbf{z}_{\mathrm{H}}|\boldsymbol{\xi})$ is used as an approximation to $P(\mathbf{z}_{\mathrm{H}}|\mathbf{z}_{\mathrm{V}})$.

The main advantage of variational inference over probability propagation in multiply-connected networks is the explicit choice of a distance measure that is minimized. Although probability propagation is optimal for singly-connected networks, there is very little known theoretically about the quality of inference that results when the network is multiply-connected. On the other hand, there is no general guarantee that in multiply-connected networks, variational methods will perform better than probability propagation. An example where probability propagation in multiply-connected networks works very well is the celebrated turbo-decoding algorithm for error-correcting coding [Berrou, Glavieux and Thitimajshima 1993].

Compared to Monte Carlo, variational inference may provide the designer with a more

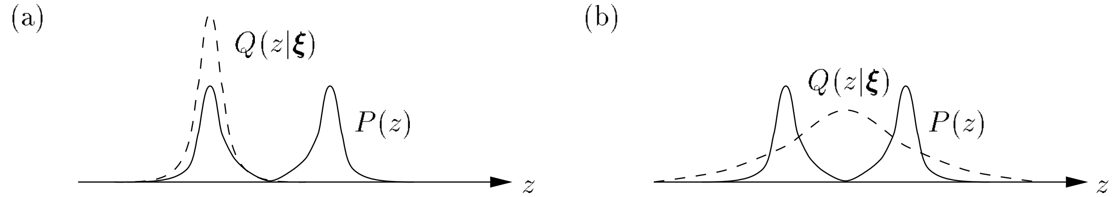

Figure 1.13: The effect of using (a) $D(Q(z|\boldsymbol{\xi}) \parallel P(z))$ versus (b) $D(P(z) \parallel Q(z|\boldsymbol{\xi}))$ when fitting a variational distribution $Q(z|\boldsymbol{\xi})$ to a univariate distribution $P(z)$.

structured approach to choosing a computationally tolerable approximation to $P(\mathbf{z}_{\mathrm{H}}|\mathbf{z}_{\mathrm{V}})$. However, variational methods do not usually provide a means to obtain exact inference. Also, variational inference can only be applied when the network is well-tailored to a sensible distance measure along with a fruitful form of variational distribution. (For example, the majority of work on variational methods for Bayesian networks to date has focussed on networks that are parameterized.) In contrast, Monte Carlo methods are guaranteed to converge to the correct solution, and can be applied to any Bayesian network.

1.5.1 Choosing the distance measure

Depending on the particular problem, different measures of distance may be appropriate. For example, in the case of hard-decision classification and hard-decision channel coding, a binary distance is ideal. Under this distance, the distributions are identical if they lead to the same decisions. Otherwise, the distance is incremented for each incorrect decision. In practice, this distance must be softened in order to use continuous optimization methods.

As another example, we'll see in Chapters 2 and **??** that for density estimation and data compression, the appropriate "distance" is the Kullback-Leibler divergence, or relative entropy, between $Q(\mathbf{z}_{\mathrm{H}}|\boldsymbol{\xi})$ and $P(\mathbf{z}_{\mathrm{H}}|\mathbf{z}_{\mathrm{V}})$:

$$D_{Q||P} = \sum_{\mathbf{z}_{\mathrm{H}}} Q(\mathbf{z}_{\mathrm{H}}|\boldsymbol{\xi}) \log \frac{Q(\mathbf{z}_{\mathrm{H}}|\boldsymbol{\xi})}{P(\mathbf{z}_{\mathrm{H}}|\mathbf{z}_{\mathrm{V}})}. \tag{1.39}$$

Notice that this is not a true distance since it is not symmetric: $D_{Q||P} \neq D_{P||Q}$.

The choice of whether to use $D_{Q||P}$ or $D_{P||Q}$ depends on our objective. The former places emphasis on not inferring unlikely values of $\mathbf{z}_{\mathrm{H}}$ at the cost of not inferring some of the likely values, whereas the latter places emphasis on inferring all likely values of $\mathbf{z}_{\mathrm{H}}$ at the cost of inferring some of the unlikely values. For example, consider a real-valued univariate distribution $P(z)$ that has two modes, as shown in Figure 1.13. Suppose the variational distribution $Q(z|\boldsymbol{\xi})$ is a Gaussian with $\boldsymbol{\xi}$ consisting of a mean and a variance. Figure 1.13a shows the optimum variational distribution that is obtained by minimizing $D_{Q||P}$, whereas Figure 1.13b shows the optimum variational distribution that is obtained by minimizing $D_{P||Q}$.

Notice that in order to compute $D_{Q||P}$ in (1.39), we need $P(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$, which is what we were after in the first place. So, in practice, we actually minimize the following function:

$$\begin{aligned} F_{Q||P} &= D_{Q||P} - \log P(\mathbf{z}_\mathrm{V}) \\ &= \sum_{\mathbf{z}_\mathrm{H}} Q(\mathbf{z}_\mathrm{H}|\boldsymbol{\xi}) \log \frac{Q(\mathbf{z}_\mathrm{H}|\boldsymbol{\xi})}{P(\mathbf{z}_\mathrm{H}, \mathbf{z}_\mathrm{V})}. \end{aligned} \tag{1.40}$$

Notice that minimizing $F_{Q||P}$ with respect to $\boldsymbol{\xi}$ gives the same set of parameters as minimizing $D_{Q||P}$, since $\log P(\mathbf{z}_\mathrm{V})$ does not depend on $\boldsymbol{\xi}$. In order to compute $F_{Q||P}$, we only need $P(\mathbf{z}_\mathrm{H}, \mathbf{z}_\mathrm{V})$, which is readily available.

1.5.2 Choosing the form of $Q(\mathbf{z}_\mathrm{H}|\boldsymbol{\xi})$

The form of $Q(\mathbf{z}_\mathrm{H}|\boldsymbol{\xi})$ will strongly influence the quality of the variational inference as well as the tractability of computing the distance and its derivatives (which may be needed for the optimization procedure). Exact inference can be achieved in principle by associating one parameter $\xi_{\mathbf{z}_\mathrm{H}}$ with each state of the hidden variables $\mathbf{z}_\mathrm{H}$, where $\xi_{\mathbf{z}_\mathrm{H}}$ is meant to be an estimate of $P(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$. However, computing the distance will require an explicit summation over all possible states of the hidden variables. The number of terms in this sum equals the number of possible configurations of the hidden variables, so this approach will only be tractable when there are not many configurations of the hidden variables. In fact, in most cases the above procedure will not be any more computationally efficient than directly computing $P(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$ using Bayes rule.

We'd like to choose $Q(\mathbf{z}_\mathrm{H}|\boldsymbol{\xi})$ so that the effect of the hidden variables $\mathbf{z}_\mathrm{H}$ in the distance measure can be integrated out either analytically or using a reasonably small number of computations. In this way, the distance and its gradients can be determined without having to numerically examine each possible state of the hidden variables $\mathbf{z}_\mathrm{H}$.

1.5.3 Variational inference for the burglar alarm problem

In this section, I illustrate variational inference using the burglar alarm network described in Section 1.4.3. One type of variational distribution that is often used is the product form distribution. Under this variational distribution, the hidden variables are independent. For continuous variables, further assumptions may be needed regarding the distributions for each hidden variable (*e.g.*, see Section 2.8). For the binary burglar b and earthquake e variables in the burglar alarm network, we can specify an arbitrary product-form distribution using the parameters ξ_{B} and ξ_{E} for the probabilities that $b=1$ and $e=1$ respectively. That is,

$$Q(b,e|\boldsymbol{\xi}) = Q(b|\boldsymbol{\xi})Q(e|\boldsymbol{\xi}) = \xi_{\mathrm{B}}^{b}(1-\xi_{\mathrm{B}})^{1-b}\xi_{\mathrm{E}}^{e}(1-\xi_{\mathrm{E}})^{1-e}. \tag{1.41}$$

Inserting this variational distribution into (1.40), we get

$$\begin{aligned}
F_{Q\|P} &= \sum_{\substack{b=0,1\\e=0,1}} Q(b,e|\boldsymbol{\xi}) \log \frac{Q(b,e|\boldsymbol{\xi})}{P(b,e,a=1)} \\
&= \sum_{\substack{b=0,1\\e=0,1}} Q(b,e|\boldsymbol{\xi})[b\log\xi_{\mathrm{B}} + (1-b)\log(1-\xi_{\mathrm{B}}) + e\log\xi_{\mathrm{E}} + (1-e)\log(1-\xi_{\mathrm{E}})] \\
&\quad - \sum_{\substack{b=0,1\\e=0,1}} Q(b,e|\boldsymbol{\xi}) \log P(b,e,a=1) \\
&= \xi_{\mathrm{B}}\log\xi_{\mathrm{B}} + (1-\xi_{\mathrm{B}})\log(1-\xi_{\mathrm{B}}) + \xi_{\mathrm{E}}\log\xi_{\mathrm{E}} + (1-\xi_{\mathrm{E}})\log(1-\xi_{\mathrm{E}}) \\
&\quad - \sum_{\substack{b=0,1\\e=0,1}} Q(b,e|\boldsymbol{\xi}) \log P(b,e,a=1).
\end{aligned} \tag{1.42}$$

Notice that the product form of $Q(b,e|\boldsymbol{\xi})$ simplified the expectation of $\log Q(b,e|\boldsymbol{\xi})$.

At this point, without any further restrictions, we have not gained any computational advantage by using the variational approach. To compute $F_{Q\|P}$ and its derivatives, we must still examine all possible configurations of the hidden variables to compute the expectation of $\log P(b,e,a=1)$. In order to make profitable use of variational inference, $\log P(b,e,a=1)$ must have a form that makes the computation of $F_{Q\|P}$ easy. It turns out that the conditional probabilities (1.34) for the burglar alarm network were obtained from

$$P(a=1|b,e) = \exp[6b + 5e - 4.5be - 7]. \tag{1.43}$$

So, the joint distribution $P(b,e,a=1)$ can be written

$$P(b,e,a=1) = P(a=1|b,e)P(b)P(e) = \exp[6b+5e-4.5be-7]0.1^{b}0.9^{1-b}0.1^{e}0.9^{1-e}. \tag{1.44}$$

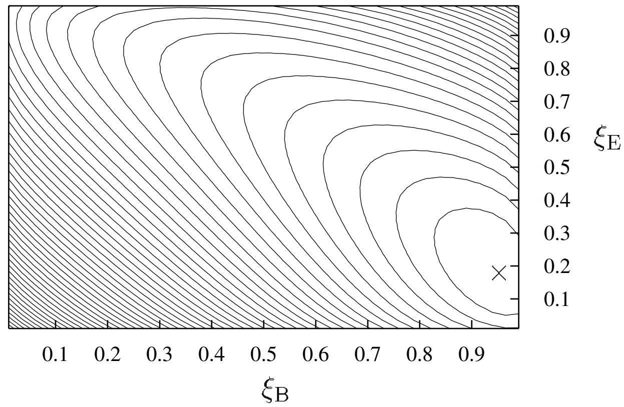

Figure 1.14: The contours of $F_{Q||P}$ for a variational technique applied to the burglar alarm problem. The global minimum uccurs at $\xi_B = 0.951$, $\xi_E = 0.186$.

Substituting this into (1.42), we get

$$F_{Q||P} = \xi_B \log \xi_B + (1 - \xi_B) \log(1 - \xi_B) + \xi_E \log \xi_E + (1 - \xi_E) \log(1 - \xi_E)$$

$$- \sum_{\substack{b=0,1 \\ e=0,1}} Q(b, e|\boldsymbol{\xi})[6b + 5e - 4.5be - 7 + b \log 0.1 + (1 - b) \log 0.9 + e \log 0.1 + (1 - e) \log 0.9]$$

$$= \xi_B \log \xi_B + (1 - \xi_B) \log(1 - \xi_B) + \xi_E \log \xi_E + (1 - \xi_E) \log(1 - \xi_E)$$

$$-6\xi_B - 5\xi_E + 4.5\xi_B\xi_E + 7 - \xi_B \log 0.1 - (1 - \xi_B) \log 0.9 - \xi_E \log 0.1 - (1 - \xi_E) \log 0.9. \tag{1.45}$$

Notice that the hidden variables b and e do not appear in this final expression. Because of the product form of $Q(b, e|\boldsymbol{\xi})$ and the exponential form of $P(b, e, a\!=\!1)$, we were able to integrate them out analytically.

Figure 1.14 shows a plot of the contours of $F_{Q||P}$ as a function of ξ_B and ξ_E. The global minimum occurs at $\xi_B = 0.951$, $\xi_E = 0.186$, which means the inference estimates are $\hat{P}(b = 1|a\!=\!1) = 0.951$ and $\hat{P}(e = 1|a\!=\!1) = 0.186$. These estimates clearly favor a burglar as the cause of the alarm. Recall that Gibbs sampling allowed us to estimate covariance statistics between the two hidden variables. Variational inference does not readily produce those estimates. However, compared to the probabilities $P(b = 1|a = 1) = 0.751$ and $P(e\!=\!1|a\!=\!1) = 0.349$ produced by the probability propagation algorithm, the variational method places more emphasis on the more likely cause b. In this sense, the variational technique reveals covariance more than probability propagation does. For example, using only the probabilities produced by propagation, we might conclude that the probability that *both* a burglar and an earthquake occurred is $0.751 \times 0.349 = 0.262$. In contrast, using the probabilities produced by the variational method gives $0.951 \times 0.186 = 0.177$, which is closer to the correct value of 0.116 given in (1.35).

In this case, because the burglar alarm network is so small, the analytic form of $F_{Q||P}$ in (1.45) is not much simpler than the expression that would be obtained if (1.41) were substituted into (1.42) and numerical integration over all values of b and e were performed. However, for larger networks, exponential computational savings may be made by using conditional distributions that lead to simple forms of $\log P(\mathbf{z}_\mathrm{H}, \mathbf{z}_\mathrm{V})$.

1.5.4 Bounds and extended representations

In practice, the form of $\log P(\mathbf{z}_\mathrm{H}, \mathbf{z}_\mathrm{V})$ is often not simple, so that a straight-forward variational approach cannot be attempted. In these cases, it may be possible to derive an upper *bound* on the distance that does not depend on $\mathbf{z}_\mathrm{H}$, and then try to minimize the bound instead of the distance itself [Saul, Jaakkola and Jordan 1996]. Effectively, we approximate $\log 1/P(\mathbf{z}_\mathrm{H}, \mathbf{z}_\mathrm{V})$ with an upper found that *can* be integrated analytically.

Alternatively, we may express each conditional distribution $P(z_k|a(z_k))$ in terms of conditional distributions over an extended set of variables [Jaakkola, Saul and Jordan 1996]. For example, $P(z_k|a(z_k))$ might be the marginal distribution of $P(z_k, y_k|a(z_k))$, where y_k is part of the extended representation. Having extended the representation, it is sometimes possible to introduce a variational distribution $Q(\mathbf{z}_\mathrm{H}, \mathbf{y}_\mathrm{H}|\boldsymbol{\xi})$ over the extended representation for which $\mathbf{z}_\mathrm{H}$ and $\mathbf{y}_\mathrm{H}$ *can* be integrated out in the distance measure.

1.6 Helmholtz machines

One of the main drawbacks of Markov chain Monte Carlo inference and variational inference is that for complex networks, each time a set of variables is observed, either a computationally taxing Markov chain must be simulated, or a high-dimensional optimization must be performed to find the best variational distribution. The essential problem, of course, is that the optimal distribution over $\mathbf{z}_\mathrm{H}$ is different for different values $\mathbf{z}_\mathrm{V}$ of the visible variables. A *Helmholtz machine* [Dayan et al. 1995; Hinton et al. 1995] tackles this problem by coupling the original *generative* network with a *recognition* Bayesian network that is meant to be capable of quickly producing an estimate of, or an approximate sample from, $P(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$. This recognition network essentially replaces the variational optimization needed for variational inference. It is called a "recognition" network because it is meant to recognize the hidden variable values, or "causes", that are responsible for the values of the visible variables.

As described above, the job of the recognition network is to quickly produce an ap-

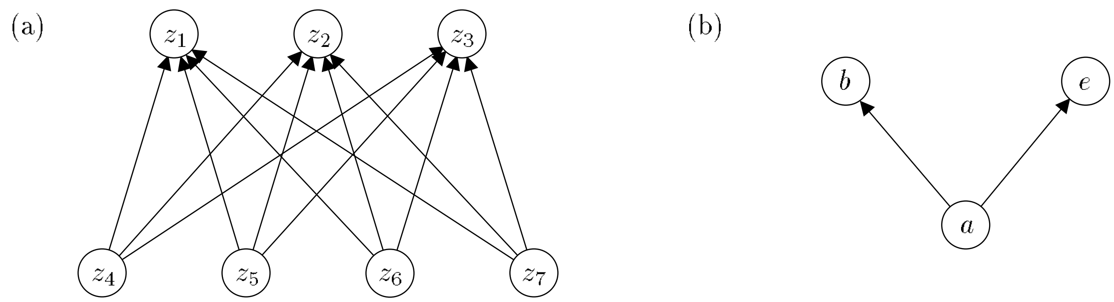

Figure 1.15: (a) An example of a factorial recognition network. (b) A factorial recognition network for the burglar alarm problem.

proximation to $P(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$. Obviously, the recognition network must be different from the generative network, or the inference could be done directly on the generative network. I will highlight this difference by labeling the recognition distribution with Q. So, the recognition network is used to compute $Q(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$, which is an approximation to $P(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$ as given by the generative network. Various types of recognition network are described below, but they all share a common property. Since the recognition network is a Bayesian network, we cannot expect to be able to quickly compute $Q(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$ for arbitrary sets $\mathbf{z}_\mathrm{H}$ and $\mathbf{z}_\mathrm{V}$. In fact, I will usually assume that the set of visible variables is the same for each case, although, of course, the values for the visible variables may differ from case to case. This restriction is the main disadvantage of the Helmholtz machine compared to Monte Carlo inference and variational inference, which usually place no restrictions on which variables are visible.

1.6.1 Factorial recognition networks

To ensure that the inference process is fast, we ought to design the recognition network so that the computation of $Q(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$ can be carried out efficiently. The simplest recognition network in this sense is one for which each variable in $\mathbf{z}_\mathrm{H}$ is dependency-separated from each other variable in $\mathbf{z}_\mathrm{H}$ by the visible variables $\mathbf{z}_\mathrm{V}$. In other words, given the visible variables, the hidden variables are independent. I will refer to such a network as a *factorial* recognition network, since given the visible variables, the distribution over the hidden variables can be factored into a product of probabilities:

$$Q(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V}) = \prod_{z_k \in \mathbf{z}_\mathrm{H}} Q(z_k|\mathbf{z}_\mathrm{V}). \tag{1.46}$$

A factorial recognition network with $\mathbf{z}_\mathrm{H} = (z_1, z_2, z_3)$ is shown in Figure 1.15a. Note that by condition 2 in Section 1.2.7, variables in $\mathbf{z}_\mathrm{H}$ are dependency-separated by $\mathbf{z}_\mathrm{V} =$

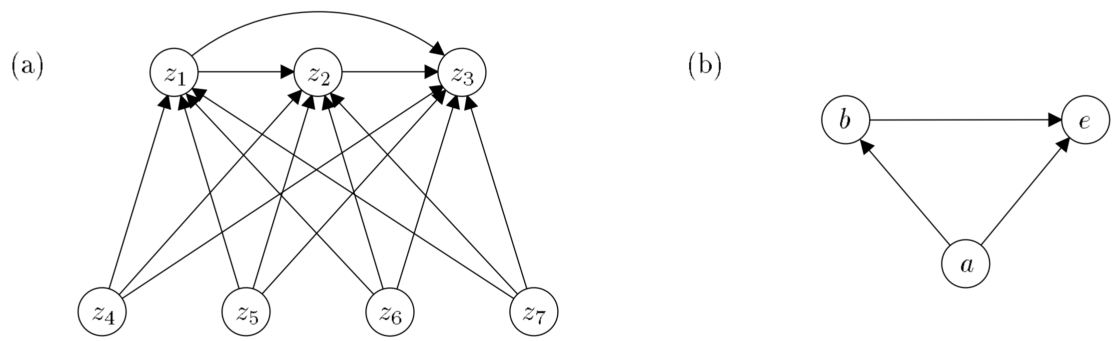

Figure 1.16: (a) An example of a nonfactorial recognition network. (b) A nonfactorial recognition network for the burglar alarm problem.

(z_4, z_5, z_6, z_7).

In many cases, the product form approximation given in (1.46) is not very close to $P(\mathbf{z}_\mathrm{H}|\mathbf{z}_\mathrm{V})$. However, it is the easiest network to design or estimate, and because the hidden variables are independent given the visible variables, it is computationally efficient for inference.

Figure 1.15b shows a factorial recognition network for the burglar alarm problem. The recognition distribution is given by $Q(b, e|a) = Q(b|a)Q(e|a)$, and so is limited to the inference estimates produced by probability propagation and the variational technique. Can we design a recognition network that can give better estimates? The answer is "yes", by using a nonfactorial recognition network.

1.6.2 Nonfactorial recognition networks

Although it is easy to imagine situations where a factorial recognition network will suffice, for the burglar alarm problem discussed above, we saw that a factorial recognition network could not capture the covariance between the two causes of the alarm. In this section, I describe nonfactorial recognition networks that are more powerful than factorial ones.

A *nonfactorial* recognition network can represent a distribution where at least one variable in $\mathbf{z}_\mathrm{H}$ is *not* dependency-separated from at least one other variable in $\mathbf{z}_\mathrm{H}$ by the visible variables $\mathbf{z}_\mathrm{V}$. Of course, there are many ways to make a network nonfactorial. For example, a nonfactorial recognition network is obtained by making some hidden variables depend on other ones in addition to the visible variables. Figure 1.16a shows a fully-connected nonfactorial recognition network, which can be contrasted with the factorial network in Figure 1.15a.

Another way to produce a nonfactorial recognition network is through the use of *auxil-*

iary variables or *dangling units* [Dayan and Hinton]. These variables are not part of the generative network, but help facilitate the recognition network for inference. For example, an auxiliary variable in the recognition network can be used to choose between two or more modes.

1.6.3 The stochastic Helmholtz machine

Suppose we are interested in only one of the hidden variables, and we'd like to obtain its distribution given the visible variables, after marginalizing out the other hidden variables. For a factorial recognition network, each hidden variable is independent given the visible variables. So, the marginal distribution is obtained simply by ignoring the other hidden variables. In fact, the marginal distribution for $z_k \in \mathbf{z}_{\mathrm{H}}$ in this case is $Q(z_k|\mathbf{z}_{\mathrm{V}})$, which is part of the recognition network specification.

Such a simple procedure for marginalization is not in general available for nonfactorial recognition networks. In these networks, the hidden variables are not independent given the visible variables. However, Monte Carlo provides an easy way to estimate marginal statistics. If we can obtain a sufficiently large sample from the recognition network, the distribution for z_k can be approximated by constructing a histogram for z_k alone. Of course, we could directly apply Monte Carlo methods such as Gibbs sampling (Section 1.4.2) to the generative network. However, the hope is that we can carefully design the nonfactorial recognition network so that it is better suited to Monte Carlo than the generative network. In fact, we can avoid complicated Markov chain Monte Carlo by using a recognition network for which ancestral simulation (see Section 1.2.3) can be used.

In general, recognition networks can be either factorial or nonfactorial and stochastic or nonstochastic. Here, "nonstochastic" refers to the way the recognition network is used, not to what the network represents. All Bayesian networks represent a stochastic phenomena, but not all networks are used with Monte Carlo. A factorial recognition network can easily be operated stochastically, simply by choosing each hidden variable z_k from its distribution $Q(z_k|\mathbf{z}_{\mathrm{H}})$. A nonfactorial recognition network is operated stochastically using a Monte Carlo method (preferably ancestral simulation). A factorial recognition network can easily be operated nonstochastically, since the joint distribution over the hidden variables factors and the marginal distribution for each hidden variable is readily available. However, a nonfactorial recognition network *usually* cannot be operated nonstochastically. As described above, the dependencies between the hidden variables makes this difficult. However, there are special cases where nonfactorial recognition networks can be operated nonstochastically. In particular, recognition networks that can be viewed as a mixture of factorial networks can be operated nonstochastically with relative ease.

1.6.4 A nonfactorial recognition network for the burglar alarm problem

In many cases, a simple nonfactorial recognition network can be used to represent covariances between hidden variables. A nonfactorial recognition network for the burglar alarm problem is shown in Figure 1.16b. The difference between this network and the factorial one in Figure 1.15b, is that e now depends on b as well as a. The conditional distributions for a recognition network that performs exact inference are

$$Q(b\!=\!1|a\!=\!1) = 0.751, \qquad Q(e|b, a\!=\!1) = \begin{cases} 0.154 & \text{if } b = 1, \\ 0.936 & \text{if } b = 0. \end{cases} \tag{1.47}$$

Sampling hidden variables using ancestral simulation in this network is actually more efficient than using Gibbs sampling in the generative network, as described in Section 1.4.3. (The computational savings are quite low in this case, because there are only two hidden variables.)

The joint distribution over the hidden variables given $a = 1$ can be computed from $Q(b, e|a\!=\!1) = Q(e|b, a\!=\!1)Q(b|a\!=\!1)$:

$$\begin{aligned} Q(b\!=\!0, e\!=\!0|a\!=\!1) &= 0.016, \\ Q(b\!=\!1, e\!=\!0|a\!=\!1) &= 0.635, \\ Q(b\!=\!0, e\!=\!1|a\!=\!1) &= 0.233, \\ Q(b\!=\!1, e\!=\!1|a\!=\!1) &= 0.116. \end{aligned} \tag{1.48}$$

These probabilities are identical to the probabilities in (1.35) for exact inference.