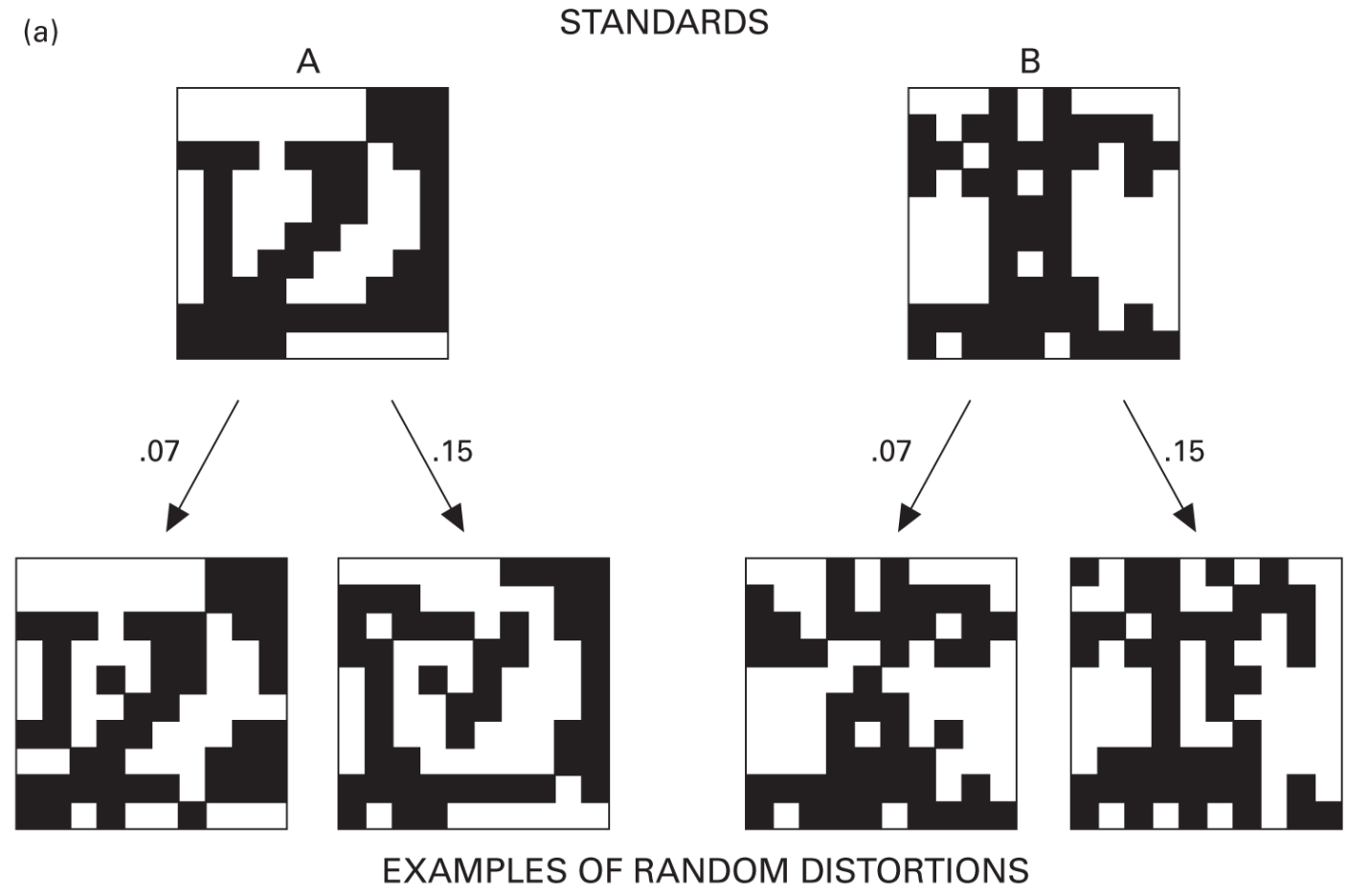


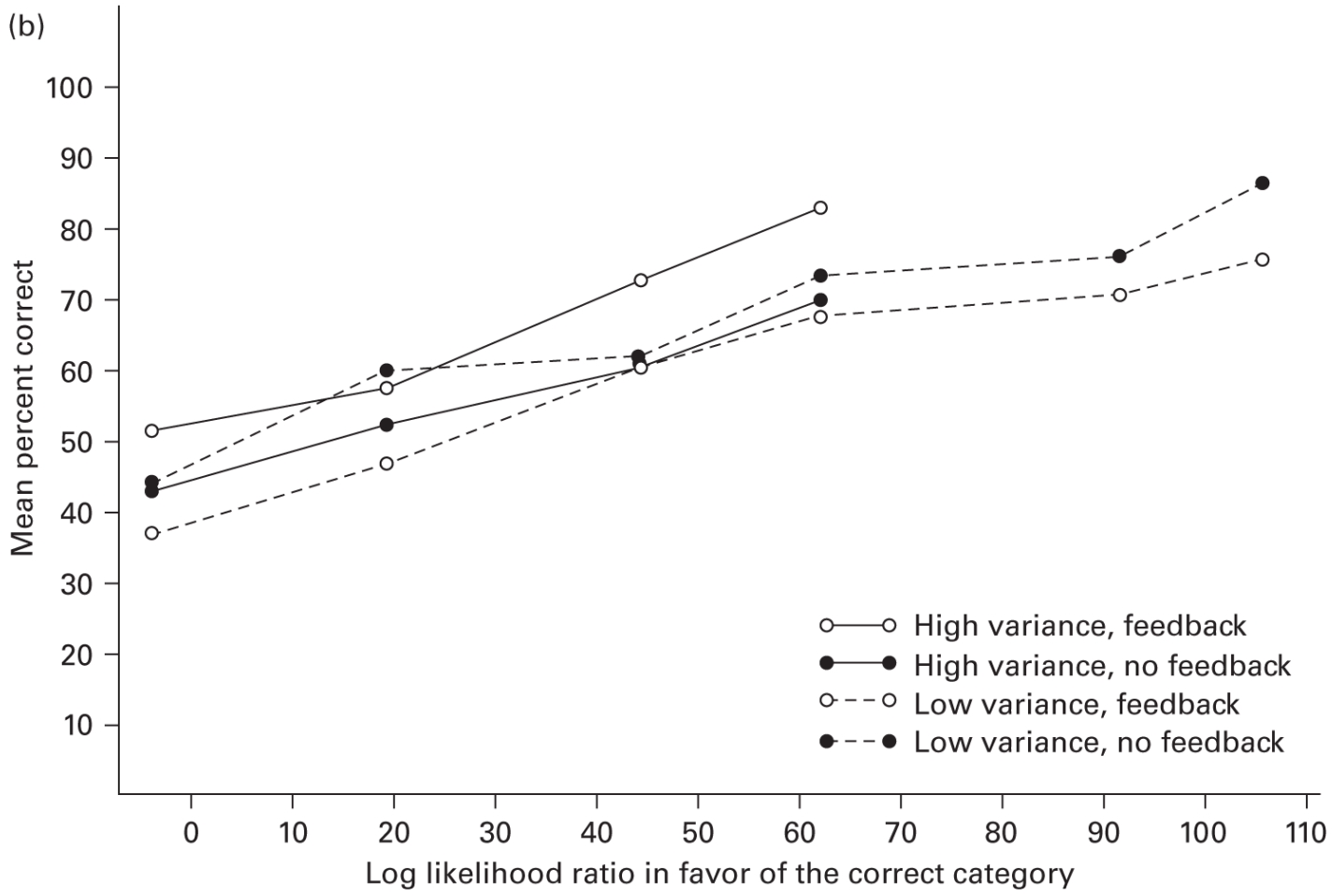


Figure 5.6
Fried and Holyoak (1984) examined whether people can learn categories without labels. (a) Their stimuli were binary arrays described as designs generated by different artists. Each category was defined by choosing a "standard" array (a prototype) and then randomly flipping bits (i.e., changing a black square to white or vice versa). The probability of flipping a bit was either.07 or.15, resulting in "low variability" and "high variability" categories, respectively. (b) People were able to learn the categories both with and without feedback on their categorization decisions. The results here show that people were increasingly correct in identifying the category membership of an array as that array provided more evidence of belonging to one of the categories, as reflected in the log-likelihood ratio. While the best performance was seen in the high variance categories with feedback, participants were still able to learn the categories without labels. Figure adapted from Fried and Holyoak (1984).

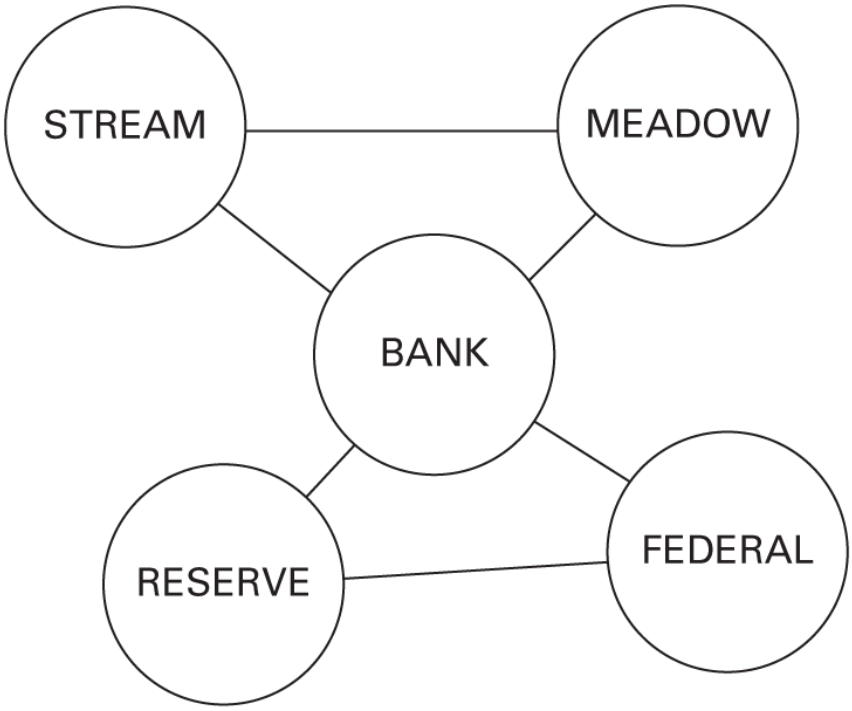
Fried and Holyoak (1984) found that people were able to learn the categories either with or without labels, as shown in figure 5.6b. The best performance was seen in the condition where participants received feedback on their classification decisions, but in all conditions people were able to learn to generalize to new instances of the categories. This behavior was consistent with the predictions of a model that Fried and Holyoak (1984) proposed that is very similar to the EM algorithm for mixture models, updating estimates of the parameters of categories using a fractional allocation of observations based on the posterior probability of belonging to that category.

5.4 Topic Models

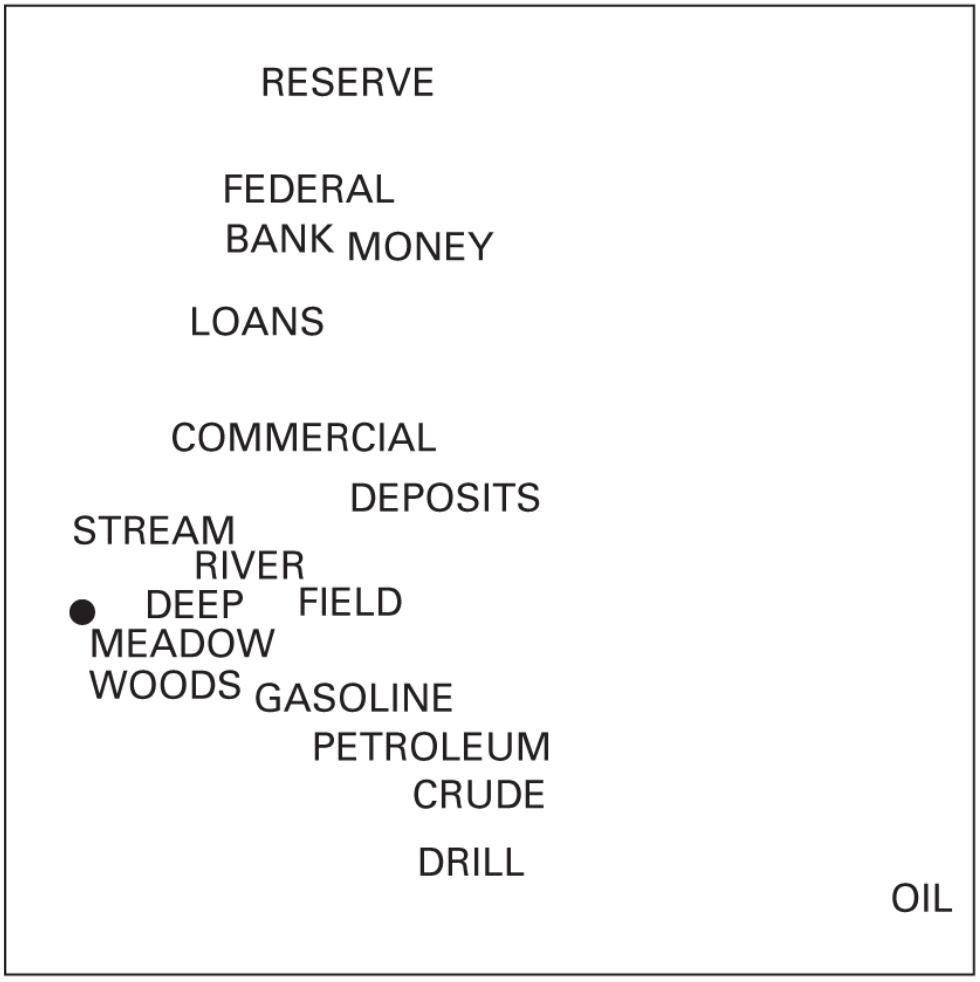
The basic idea behind mixture models can be used to define models that capture people's inferences in other settings. We now turn to *topic models*, which add a simple twist to mixture models to capture an important aspect of language. These models have also been used to account for aspects of human semantic memory, tracking the abstract structure behind the associations that we perceive between words.

Semantic Memory Several computational models have been proposed to account for the large-scale structure of semantic memory, including semantic networks (e.g., Collins & Loftus, 1975; Collins & Quillian, 1969) and semantic spaces (e.g., Landauer & Dumais, 1997; Lund & Burgess, 1996; Mikolov et al., 2013; Pennington, Socher, & Manning, 2014). These approaches embody different assumptions about the way that words are represented. In semantic networks, words are nodes in a graph where edges indicate semantic relationships, as shown in figure 5.7a. In semantic space models, words are represented as points in high-dimensional space, where the distance between two words reflects the extent to which they are semantically related, as shown in figure 5.7b.

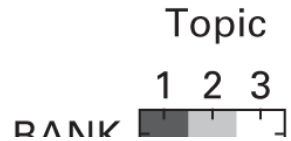
(a)



(b)



(c)



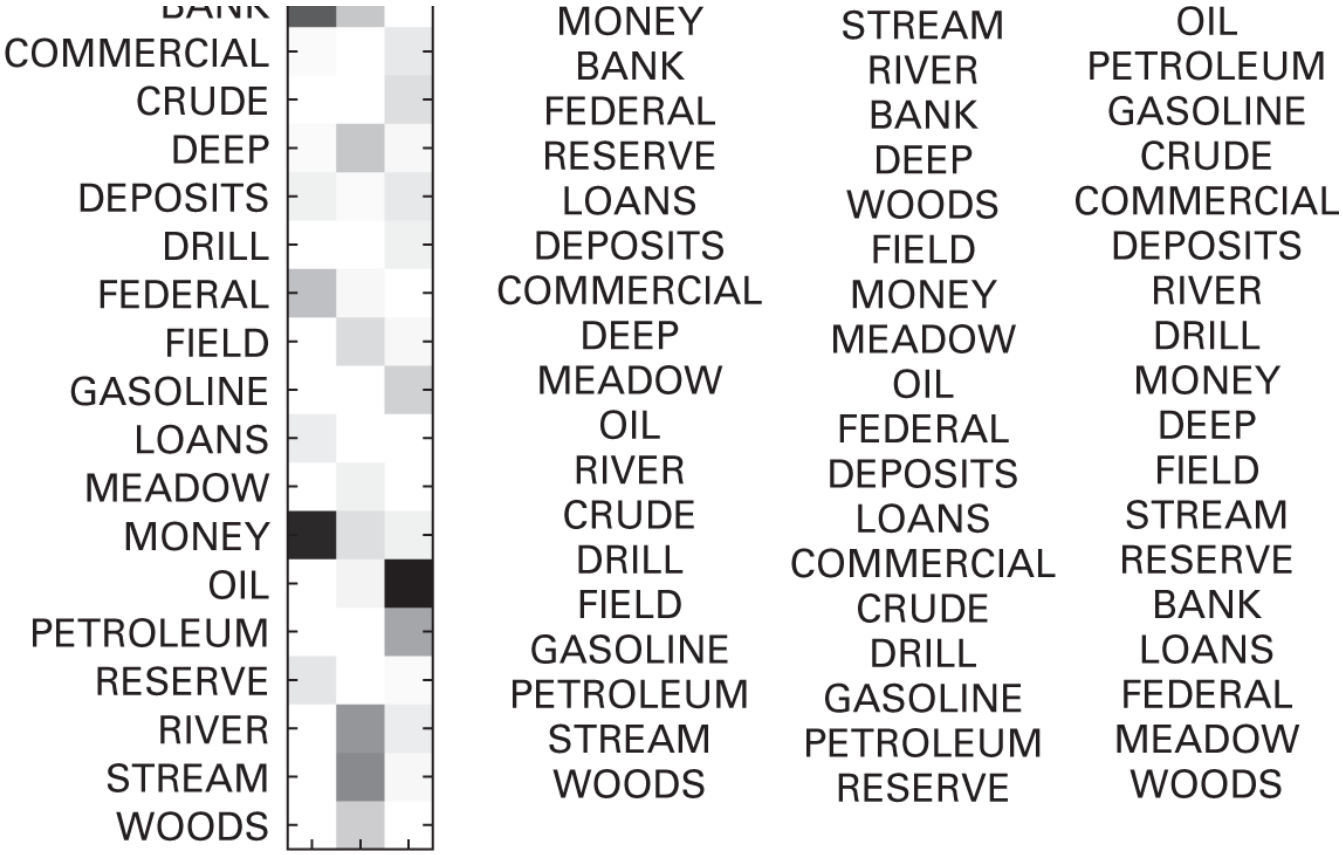


Figure 5.7

Approaches to semantic representation. (a) In a semantic network, words are represented as nodes, and edges indicate semantic relationships. (b) In a semantic space, words are represented as points, and proximity indicates semantic association. These are the first two dimensions of a solution produced by Latent Semantic Analysis (Landauer & Dumais, 1997). The black dot is the origin. (c) In the topic model, words are represented as belonging to a set of probabilistic topics. The matrix shown on the left indicates the probability of each word under each of three topics. The three columns on the right show the words that appear in those topics, ordered from highest to lowest probability. Figure reproduced with permission from Griffiths et al. (2007).

Words and Topics Probabilistic models provide an opportunity to explore alternative representations for the meaning of words. Topic models represent words in terms of the set of topics to which they belong (Hofmann, 1999; Blei, Ng, & Jordan, 2003; Griffiths & Steyvers, 2004). Each topic is a probability distribution over words, and the content of the topic is reflected in the words to which it assigns high probability. For example, high probabilities for WOODS and STREAM would suggest that a topic refers to the countryside, while high probabilities for FEDERAL and RESERVE would suggest that a topic refers to finance. Each word will have a probability under each of these topics, as shown in figure 5.7c.

For example, MEADOW has a relatively high probability under the countryside topic, but a low probability under the finance topic, similar to WOODS and STREAM.

Representing word meanings using probabilistic topics makes it possible to use Bayesian inference to answer some of the critical problems that arise in processing language. In particular, we can make inferences about which semantically related concepts are likely to arise in the context of an observed set of words or sentences, in order to facilitate subsequent processing.

Let z denote the dominant topic in a particular context, and w_1 and w_2 be two words that arise in that context. The semantic content of these words is encoded through a set of probability distributions that identify their probability under different topics:

if there are k topics, then these are the distributions $P(w|z)$ for $z = \{1, \ldots, k\}$. Given w_1, we can infer which topic z was likely to have produced it by using Bayes' rule:

$$P(z|w_1) = \frac{P(w_1|z)P(z)}{\sum_{j=1}^{k} P(w_1|z' = j)P(z' = j)}, \tag{5.27}$$

where $P(z)$ is a prior distribution over topics. Having computed this distribution over topics, we can make a prediction about future words by summing over the possible topics:

$$P(w_2|w_1) = \sum_{j=1}^{k} P(w_2|z = j)P(z = j|w_1). \tag{5.28}$$

A topic-based representation can also be used to disambiguate words: if BANK occurs in the context of STREAM, it is more likely that it was generated from the bucolic topic than the financial topic.

Probabilistic topic models are an interesting alternative to traditional approaches to semantic representation, and in many cases, they actually provide better predictions of human behavior (Griffiths & Steyvers, 2003; Griffiths et al., 2007; Nematzadeh, Meylan, & Griffiths, 2017). However, one critical question in using this kind of representation is that of which topics should be used. Fortunately, work in machine learning and information retrieval has provided an answer to this question. As with popular semantic space models (Landauer & Dumais, 1997; Lund & Burgess, 1996), the representation of a set of words in terms of topics can be inferred automatically from the text contained in large document collections. The key to this process is viewing topic models as generative models for documents, making it possible to identify a set of topics that are likely to have generated an observed collection of documents. This can be done using the EM algorithm (Hofmann, 1999; Blei et al., 2003) or Monte Carlo methods, which we will introduce in chapter 6 (Griffiths & Steyvers, 2004). Figure 5.8 shows a sample of topics inferred from the TASA corpus (Landauer & Dumais, 1997), a collection of passages excerpted from educational texts used in curricula from the first year of school to the first year of college.

PRINTING	**PLAY**	TEAM	JUDGE	HYPOTHESIS	STUDY	**CLASS**	ENGINE
PAPER	PLAYS	GAME	TRIAL	EXPERIMENT	**TEST**	MARX	FUEL
PRINT	STAGE	BASKETBALL	**COURT**	SCIENTIFIC	STUDYING	ECONOMIC	ENGINES
PRINTED	AUDIENCE	PLAYERS	CASE	OBSERVATIONS	HOMEWORK	CAPITALISM	STEAM
TYPE	THEATER	PLAYER	JURY	SCIENTISTS	NEED	CAPITALIST	GASOLINE
PROCESS	ACTORS	**PLAY**	ACCUSED	EXPERIMENTS	**CLASS**	SOCIALIST	AIR
INK	DRAMA	PLAYING	GUILTY	SCIENTIST	MATH	SOCIETY	**POWER**
PRESS	SHAKESPEARE	SOCCER	DEFENDANT	EXPERIMENTAL	TRY	SYSTEM	COMBUSTION
IMAGE	ACTOR	PLAYED	JUSTICE	**TEST**	TEACHER	**POWER**	DIESEL
PRINTER	THEATRE	BALL	**EVIDENCE**	METHOD	WRITE	RULING	EXHAUST
PRINTS	PLAYWRIGHT	TEAMS	WITNESSES	HYPOTHESES	PLAN	SOCIALISM	MIXTURE
PRINTERS	PERFORMANCE	BASKET	CRIME	TESTED	ARITHMETIC	HISTORY	GASES
COPY	DRAMATIC	FOOTBALL	LAWYER	**EVIDENCE**	ASSIGNMENT	POLITICAL	CARBURETOR
COPIES	COSTUMES	SCORE	WITNESS	BASED	PLACE	SOCIAL	GAS
FORM	COMEDY	**COURT**	ATTORNEY	OBSERVATION	STUDIED	STRUGGLE	COMPRESSION
OFFSET	TRAGEDY	GAMES	HEARING	SCIENCE	CAREFULLY	REVOLUTION	JET
GRAPHIC	**CHARACTERS**	TRY	INNOCENT	FACTS	DECIDE	WORKING	BURNING
SURFACE	SCENES	COACH	DEFENSE	DATA	IMPORTANT	PRODUCTION	AUTOMOBILE
PRODUCED	OPERA	GYM	CHARGE	RESULTS	NOTEBOOK	CLASSES	STROKE
CHARACTERS	PERFORMED	SHOT	CRIMINAL	EXPLANATION	REVIEW	BOURGEOIS	INTERNAL

Figure 5.8

A sample of topics from a 1,700-topic solution derived from the TASA corpus. Each column contains the 20 highest-probability words in a single topic, as indicated by $P(w|z)$. Words in boldface occur in different senses in neighboring topics, illustrating how the model deals with polysemy and homonymy. These topics were discovered in a completely unsupervised fashion, using just word-document cooccurrence frequencies. Figure reproduced with permission from Griffiths et al. (2007).

A Generative Model for Documents We can specify a generative model for documents by assuming that each document is a mixture of topics, with each word in that document being drawn from a particular topic and the topics varying in probability across documents. For any particular document, we write the probability of a word w in that document as

$$P(w) = \sum_{j=1}^{k} P(w|z = j)P(z = j), \tag{5.29}$$

where $P(w|z)$ is the probability of word w under topic z, which remains constant across all documents, and $P(z = j)$ is the

probability of topic j in this document. We can summarize these probabilities with two sets of parameters, taking $\phi_w^{(j)}$ to indicate $P(w|z = j)$, and $\theta_z^{(d)}$ to indicate $P(z)$ in a particular document d. The procedure for generating a collection of documents is then straightforward. First, we generate a set of topics, sampling $^{(j)}$ from some prior distribution $p()$. Then for each document d, we generate the weights of those topics, sampling $^{(d)}$ from a distribution $p()$. Assuming that we know in advance how many words will appear in the document, we then generate those words in turn. A topic z is chosen for each word that will be in the document by sampling from the distribution over topics implied by $^{(z)}$. Finally, the identity of the word w is determined by sampling from the distribution over words associated with that topic.

To complete the specification of our generative model, we need to specify distributions for and so we can make inferences about these parameters from a corpus of documents. As in the case of coin-flipping, calculations can be simplified by using a conjugate prior. Both and are arbitrary distributions over a finite set of outcomes—multinomial distributions when we care about the counts of events, otherwise discrete distributions—and the conjugate prior for these distributions is the Dirichlet distribution. Just as the discrete distribution is a multivariate generalization of the Bernoulli distribution used in the coin-flipping example, the Dirichlet distribution is a multivariate generalization of the beta distribution. We assume that the number of "virtual examples" of instances of each topic appearing in each document is set by parameter , and likewise use parameter to represent the number of instances of each word in each topic. Figure 5.9 shows a graphical model depicting the dependencies among these variables. This model, known as *latent Dirichlet allocation*, was introduced in machine learning by Blei et al. (2003).

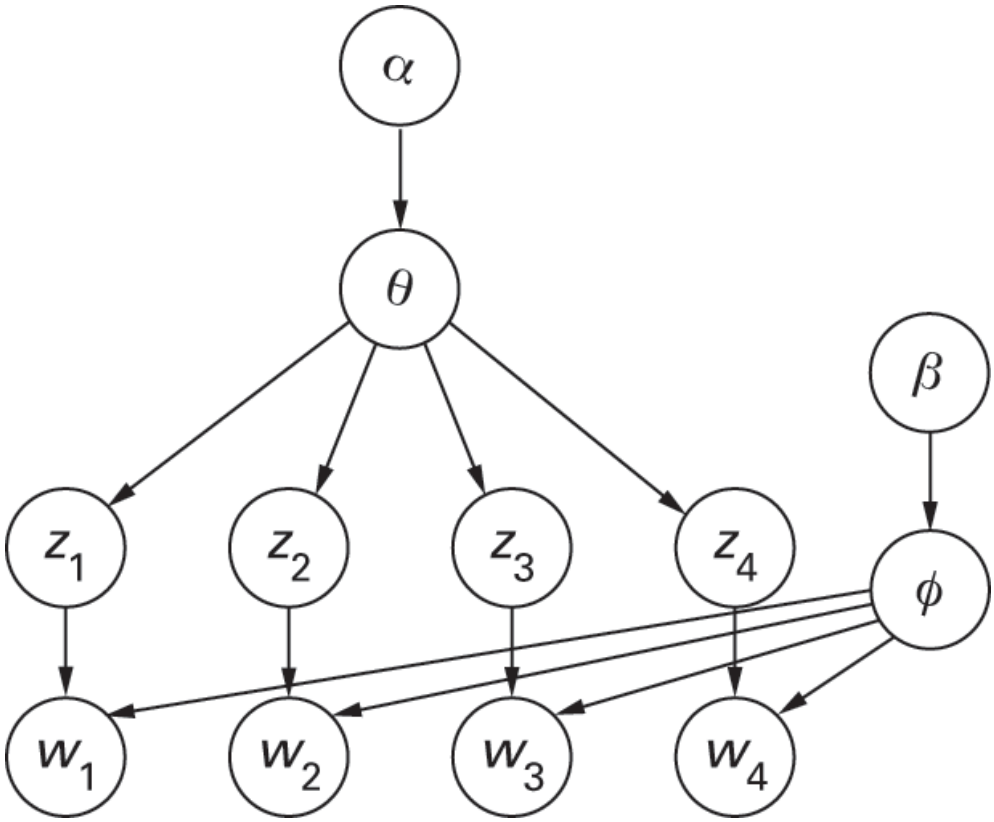


Figure 5.9

Graphical model for Latent Dirichlet Allocation (Blei, Ng, & Jordan, 2003). The distribution over words given topics, , and the distribution over topics in a document, , are generated from Dirichlet distributions with parameters and respectively. Each word in the document is generated by $_i$ $^{(z)}$ first choosing a topic z from , and then choosing a word according to .

Modeling Human Semantic Associations Griffiths and Steyvers (2002, 2003) suggested that topic models might provide an alternative to traditional approaches to semantic representation, and showed that they can provide better predictions of human word association data than latent semantic analysis (LSA) (Landauer & Dumais, 1997). Topic models can also be applied to a range of other tasks that draw on semantic association, such as semantic priming and sentence comprehension (Griffiths et al., 2007).

The key advantage that topic models have over semantic space models is postulating a more structured representation—different topics can capture different senses of words, allowing the model to deal with polysemy and homonymy in a way that is automatic and transparent. For instance, similarity in semantic space models must obey a version of the triangle inequality for distances: if there is high similarity between words w_1 and w_2, and between words w_2 and w_3, then w_1 and w_3 must be at least fairly similar. But word associations often violate this rule. For instance, ASTEROID is highly associated with BELT and BELT is highly associated with BUCKLE, but ASTEROID and BUCKLE have little association. LSA thus has trouble representing these associations. Out of approximately 4,500 words in a large-scale set of word association norms (Nelson, McEvoy, & Schreiber, 1998), LSA judges that BELT is the 13th most similar word to ASTEROID, that BUCKLE is the 2nd most similar word to BELT, and consequently BUCKLE is the 41st most similar word to ASTEROID–more similar than TAIL, IMPACT, or SHOWER. In contrast, using topics makes it possible to represent these associations faithfully because BELT belongs to multiple topics, one highly associated with ASTEROID but not BUCKLE, and another highly associated with BUCKLE but not ASTEROID. More recent semantic space models such as word2vec (Mikolov et al., 2013) or GloVe (Pennington et al., 2014), which can be trained on larger data sets than topic models, produce better performance in predicting human semantic associations but still have difficulty handling phenomena like the triangle inequality due to their representational assumptions (Nematzadeh et al., 2017).

5.5 Hidden Markov Models

So far, we have discussed some very simple models for sequential structure in data—Markov models. However, when we come to model inferences that involve reasoning about the dynamics of the environment or aspects of language, richer models of sequential dependency can be useful. The simplest of these models is the *hidden Markov model (HMM)*. We will start by reviewing Markov models and explaining how HMMs build on them, and then consider some of the basic ideas behind computing with HMMs. While these models can be applied to a range of problems, we will focus on the linguistic case in our description.

Simple Models of Language The simplest model of language assumes that each word is generated independently at random from a single fixed distribution over words. This model assumes no dependencies among words, predicting new words using the same distribution, $P(w)$, regardless of the other words that might appear in a sentence. In this model, the probability of a collection of words $\mathbf{w}$ depends only upon the frequency of each individual word in $\mathbf{w}$. If we define the distribution $P(w)$ using a vector of parameters θ_w such that $P(w) = \theta_w$, then we can write this model as

$$w_i \mid \theta \sim \text{Discrete}(\theta),$$

where $\sim$ should be read as "distributed as," and Discrete() is the multivariate analog of the Bernoulli distribution. The dependency structure assumed by this model is illustrated in figure 5.10a, and the unknown parameters that have to be estimated from data are just those of the distribution over words, θ.

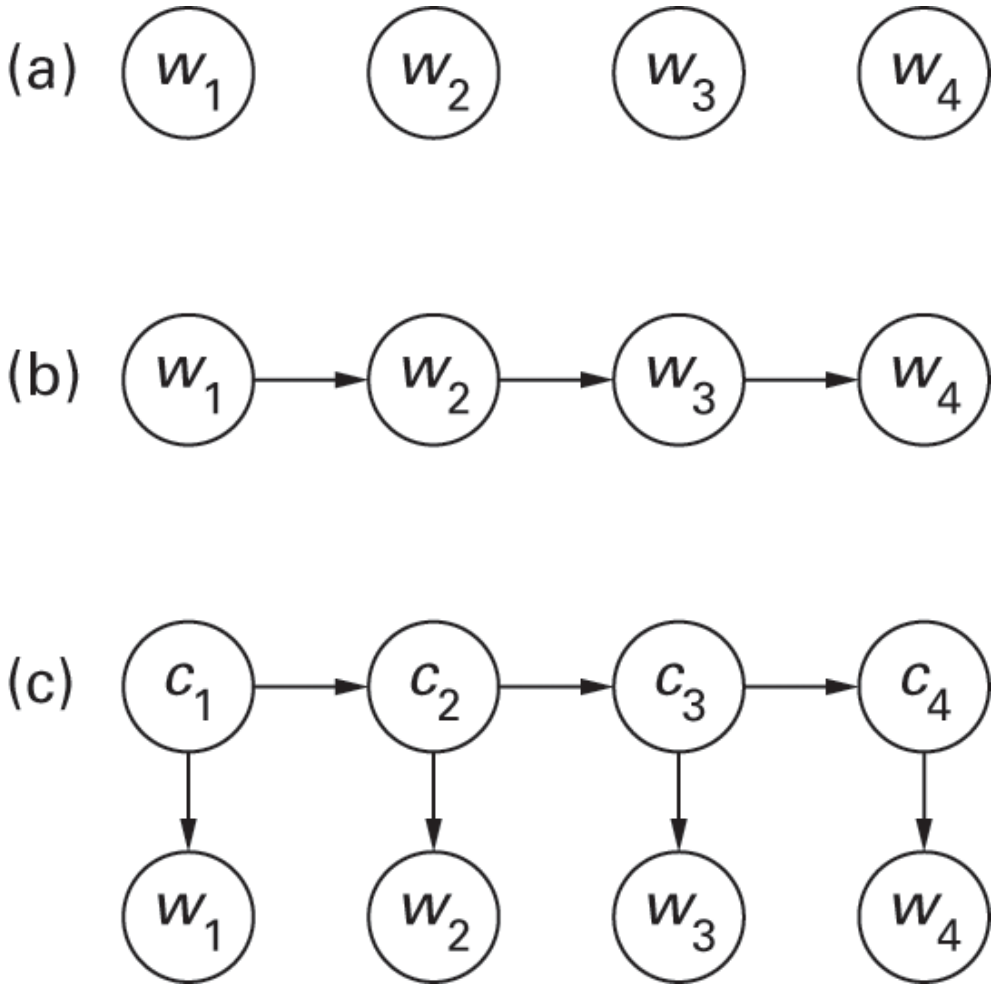


Figure 5.10

Graphical models illustrating the dependencies among variables in statistical models of language. (a) The simplest model is a unigram model, in which words are independently drawn from a common distribution. (b) A Markov model (in this case, a bigram model) adds dependencies between successive words. (c) In a hidden Markov model, the dependencies are among latent classes, with words being generated conditioned on the class. The variable c_i is the class associated with word w_i.

A slightly more complex model, a *Markov model*, allows dependencies among words. The dependency structure of one such model, a first-order Markov model, is shown in figure 5.10b. In this model, each word after the first is chosen from a distribution that is conditioned on the previous word: the ith word, w_i, is chosen from $P(w_i|w_{i-1})$, which specifies the transition probabilities between words. The probability of a collection of words $\mathbf{w}$ depends only upon the frequency with which one word precedes another. Associating $P(w_i|w_{i-1})$ with a vector of parameters $\theta^{(w_{i-1})}$, we can write

$$w_i \mid w_{i-1}, \theta \sim \text{Discrete}(\theta^{(w_{i-1})}).$$

This focus on pairs of words is the central innovation of this model over that shown in figure 5.10a, and this leads to the name *bigram model*. In general, a model in which each word is generated from a distribution that depends on the preceding n 1 words

(a Markov model of order n 1) is referred to as an *n-gram model*, with the model shown in figure 5.10a being a *unigram model*.

In any *n*-gram model, the parameters characterizing the distribution over words given the preceding words (for a bigram model, $P(w_i \mid w_{i-1})$ for all words *w*) need to be estimated from data.

The application of Markov models to language has a long history. Language modeling was one of the early examples explored by Markov (1913), and the approach was popularized by Shannon (1948). Research in language development has explored the sensitivity of learners to the transition probabilities of artificial languages, showing that infants can learn the kind of patterns captured by a Markov model (e.g., Saffran, Aslin, & Newport, 1996). However, the syntactic capabilities of Markov models have been roundly criticized, as they give low probability to grammatical sentences in which transitions between particular words are rare. Chomsky (1957) famously used this property of Markov models to argue against their adequacy as a model of language because people are able to judge sentences such as "colorless green ideas sleep furiously" to be grammatical even though no two successive words in that sentence are likely to have been heard together before.

Adding Latent Structure Some of the weaknesses of Markov models can be addressed by introducing latent structure into the model, associating each word w_i with an unobserved class c_i. Defining the transition probabilities in terms of the classes instead of the words makes it possible to give high probability to sentences that display appropriate transitions among classes, even though particular pairs of words may never have been seen before. Consequently, Chomsky's "colorless" sentence is not problematic since the transitions among word classes (nouns, adjectives, verbs, and adverbs) are consistent with the statistics of English (Pereira, 2000). This generative model is known as a hidden Markov model, since the classes of the words are "hidden" and has the dependency structure shown in figure 5.10c. In an HMM, each word is generated by choosing a class c_i from a distribution specified by $P(c_i \mid c_{i-1})$, then choosing a word w_i from the distribution associated with that class, $P(w_i \mid c_i)$. If we let $\theta^{(c)}$ be the parameters of the distribution over words associated with class *c*, and $\theta^{(c_{i-1})}$ be the distribution over the next class selected from class *c*, we can write an HMM as

$$w_i \mid c_i, \phi \sim \text{Discrete}(\phi^{(c_i)})$$

$$c_i \mid c_{i-1}, \theta \sim \text{Discrete}(\theta^{(c_{i-1})}),$$

where ϕ and θ need to be estimated from data.

Hidden Markov models were first described by Baum and Petrie (1966). HMMs have been used for processing text and speech (e.g., Charniak, 1993; Jurafsky & Martin, 2000). One of their most successful applications was part-of-speech tagging (e.g., Charniak, Hendrickson, Jacobson, & Perkowitz, 1993), in which the word class **c** are trained to match the syntactic categories, such as nouns and verbs, that comprise the most basic level of linguistic structure. When these syntactic categories are learned from scratch (e.g., Goldwater & Griffiths, 2007), they are inferred based on the extent to which words tend to be similar in the distribution of neighboring words. These models thus provide a way to capture the idea of *distributional clustering* that cognitive scientists have claimed may play a role in the acquisition of syntactic categories by children (e.g., Redington, Chater, & Finch, 1998). We discuss other models of language in chapter 16.

5.5.1 Computing with Hidden Markov Models

Part of the attraction of HMMs is that they can capture latent structure but still remain computationally tractable. To a large extent, this tractability is the consequence of the simple dependency structure between the latent variables in the model. This dependency structure admits efficient procedures (based on *dynamic programming*) for computing probabilities of interest. Now we will consider a few of the questions that one might want to answer using HMMs and show how the resulting computations simplify.

What Is the Probability of a Sentence? A basic question that we might want to answer is how likely it is that a sentence **w**

1 n

= (w_1, …, w_n) would be generated from a particular HMM with known parameters . This is a marginal probability, calculated by summing over the latent variables **c**,

$$P(\mathbf{w}) = \sum_{\mathbf{c}} P(\mathbf{w}, \mathbf{c}), \tag{5.30}$$

n

which will require summing over k values of **c** for an HMM with k states (since each of the n latent variables can take on k values). Fortunately, this can be simplified significantly because of the dependency structure of the variables involved.

The key is to define a simple recursive computation that allows us to make use of previously stored results. First, from the graphical model shown in figure 5.10c, we observe that $P(\mathbf{w})$ can also be written as

$$P(\mathbf{w}) = \sum_{c_n} P(w_n|c_n) P(c_n, \mathbf{w}_{n-1}), \tag{5.31}$$

n1 1 n1 n n1

where $\mathbf{w}_{n-1}$ = (w_1, …, w_{n-1}). This is easy to compute, provided that we can calculate $P(c_{n-1}, \mathbf{w}_{n-1})$. We can then observe that

$$P(c_n, \mathbf{w}_{n-1}) = \sum_{c_{n-1}} P(c_n|c_{n-1}) P(w_{n-1}|c_{n-1}) P(c_{n-1}, \mathbf{w}_{n-2}), \tag{5.32}$$

n1 n2 1 1

which is easy to compute if we know $P(c_{n-1}, \mathbf{w}_{n-2})$. This sets up a recursion that terminates at $P(c_1)$. Since we know $P(c_1)$, which is

i i1

just the probability distribution over the first state, we can start there and compute $P(c_i, \mathbf{w}_{i-1})$ for i = 2, …, n, and then apply equation (5.31). The resulting algorithm requires only n sums over k states, and it is thus much more efficient than summing over all states.

i i1

This procedure for computing $P(c_i, \mathbf{w}_{i-1})$ is known as the *forward procedure*. There is a corresponding procedure for

i n i

computing $P(w_i, \ldots, w_n|c_i)$, which is known as the *backward procedure*. The basic idea is that

$$P(w_i, \ldots, w_n|c_i) = P(w_i|c_i) \sum_{c_{i+1}} P(w_{i+1}, \ldots, w_n|c_{i+1}) P(c_{i+1}|c_i), \tag{5.33}$$

n n

which establishes a similar kind of recursion. Here, the recursion grounds out at $P(w_n|c_n)$, which is also known. The backward procedure can also be used to calculate $P(\mathbf{w})$ since we can observe that $P(\mathbf{w}) = \sum_{c_1} P(\mathbf{w}|c_1) P(c_1)$.

Inferring a State Sequence Assume that we want to find the state sequence **c** that was used in generating a sentence **w**. A natural way to formulate this problem is in terms of maximizing the posterior probability $P(\mathbf{c}|\mathbf{w})$. Naively, finding this sequence

n

would require considering all k possibilities. However, the same kind of approach can yield an efficient procedure for

i

identifying state sequences as well. If we want to find the posterior probability of a single state, c_i, given sentence **w**, we can

observe that $P(c_i|\mathbf{w}) \propto P(w_1, \ldots, w_{i-1}, c_i)P(w_i, \ldots, w_n|c_i)$, and use the forward and backward procedures to compute the probabilities that appear on the right side. However, finding the most likely sequence is a little more involved.

First, we observe that the $\mathbf{c}$ that maximizes $P(\mathbf{c}|\mathbf{w})$ also maximizes $P(\mathbf{w}, \mathbf{c})$. We thus need only maximize the latter. When computing $P(\mathbf{w})$, we needed to sum over $\mathbf{c}_{n-1}$, but now we want to maximize over $\mathbf{c}_{n-1}$. Fortunately, a similar recursion applies. We might hope that if we knew what $\mathbf{c}_{n-1}$ maximized $P(\mathbf{w}_{n-1}, \mathbf{c}_{n-1})$, we could use that information to find the $\mathbf{c}_n$ that maximizes $P(\mathbf{w}, \mathbf{c})$. However, it is a little more complicated than that. If we actually write the joint probability using this recursion, we see

$$P(\mathbf{w}, \mathbf{c}) = P(w_n|c_n)P(c_n|c_{n-1})P(\mathbf{w}_{n-1}, \mathbf{c}_{n-1}), \tag{5.34}$$

which makes it clear that the most likely $\mathbf{c}_{n-1}$ for $\mathbf{w}_{n-1}$ could have lower $P(c_n|c_{n-1})$ and $P(w_n|c_n)$ than another $\mathbf{c}_{n-1}$ and thus not maximize $P(\mathbf{w}_{n-1}, \mathbf{c}_{n-1})$. However, since the only part of $\mathbf{c}_{n-1}$ that is relevant to these new terms is c_{n-1}, this does suggest an algorithm: for every c_{n-1}, find the $\mathbf{c}_{n-1}$ that maximizes $P(\mathbf{w}_{n-1}, \mathbf{c}_{n-1})$, giving the most likely sequence of states that ends in c_{n-1}. The most likely sequence of states that ends in c_n must use one of these sequences of states. The algorithm that uses this observation is known as the *Viterbi algorithm* (Viterbi, 1967).

5.5.2 Estimating Parameters

While the preceding analysis assumes that the parameters of the HMM are known, it is more common that these parameters need to be estimated from data. This is often a key part of applying HMMs to problems in cognitive science and computational linguistics. Fortunately, the fact that probabilistic inference is straightforward in these models means that parameter estimation is also relatively easy.

Since the HMM involves latent variables, maximum-likelihood (or MAP) estimation can be done using the EM algorithm. The E-step takes the expectation of the complete log-likelihood over the posterior distribution on the latent classes. The distributions involved in this HMM are all discrete distributions and are thus estimated from the frequency with which events involving $\mathbf{c}$ and $\mathbf{w}$ occur. The transition probabilities are estimated from the frequency of transitions between states, and the emission probabilities are estimated from the frequency with which words are produced from a given state. When the expectation over the posterior distribution on the latent classes is taken, these frequencies are replaced by the expected number of transitions and the expected number of emissions, respectively. Consequently, the only challenge in estimating the parameters of the model is computing these expectations.

The posterior probability that c_i is one state and c_{i+1} is another is given by

$$P(c_i, c_{i+1}|\mathbf{w}) \propto P(c_i, w_1, \ldots, w_{i-1})P(w_i|c_i)P(c_{i+1}|c_i)P(w_{i+1}, \ldots, w_n|c_{i+1}), \tag{5.35}$$

where the first and last terms on the right side are given by the forward and backward procedures, respectively. The posterior probability that c_i is a particular state, needed to calculate the expected number of emissions, is computed directly from the forward and backward probabilities, as noted previously. Summing these quantities over all i yields the expected number of transitions and emissions required for either maximum-likelihood or MAP estimation of the parameters of the model. The resulting algorithm is known as the *Baum-Welch* algorithm or the *forward-backward* algorithm (for a more detailed tutorial, see Rabiner, 1989).

5.5.3 An Example: Changepoint Detection

While language is one of the prime applications of hidden Markov models, they also provide a simple tool for modeling other

kinds of events that unfold over time. If the dynamics of a sequence of events can be captured by assuming that there are a fixed set of possible latent states and transitions take place between those states, then an HMM is an appropriate model. A simple example of a scenario fitting this description is a problem known in statistics as *changepoint detection*.

The most basic assumption used in defining a statistical model is that events are independent and identically distributed—that each event is drawn independently from the same distribution, with the distribution undergoing no change over time. However, this is a poor description of many real-world phenomena. In particular, it is possible for sudden changes to take place that completely change the distribution from which events are drawn. For example, the amount of road traffic increases significantly between 4 and 7p.m., so assuming that traffic levels are identically distributed throughout the day could cause significant problems for planning your evening. Consequently, a better assumption might be that the distribution from which events are drawn can change over time, perhaps moving between a few discrete states. This is exactly the assumption that is made in an HMM.

Brown and Steyvers (2009) explored the human capacity to detect and predict changes using a simple task in which people observed events taking place in a factory that produced cans of tomatoes. Tomato cans could fall from one of four chutes, and they tended to repeatedly fall from the same chute but sometimes switched chutes (see figure 5.11). Cans falling from each chute followed a Gaussian distribution. The task of inferring which chute produced a can and predicting which chute would be used for the next can can be optimally solved using an HMM, illustrating that these models can be applied to continuous observations (can locations), as well as discrete observations (words).

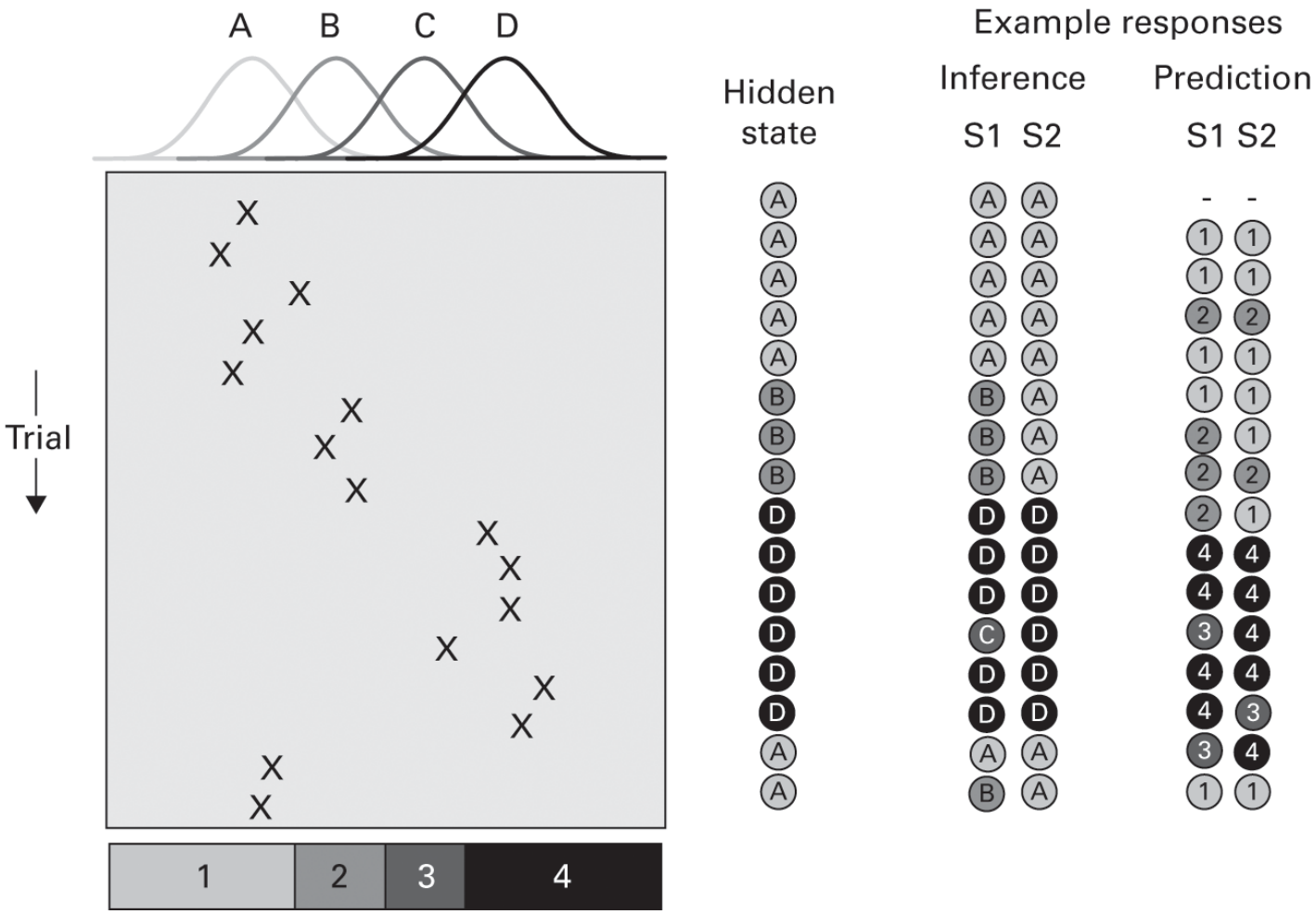


Figure 5.11

A changepoint detection task used by Brown and Steyvers (2009). In this experiment, people (here represented by two participants, S1 and S2) had to infer which of four chutes was releasing tomato cans onto a conveyer belt based on the past pattern of the locations of tomato cans ("Inference" responses) and predict which chute the next can would come from ("Prediction" responses). Here, the can locations are shown with X marks, and the big rectangle represents the conveyer belt. Each chute was associated with a Gaussian distribution over locations, and there was a small probability that there would be a change in chutes between cans. The dynamics of this system are well described by an HMM, which can be used to make optimal inferences and predictions in this task. Figure reproduced with permission from Brown and Steyvers (2009).

5.6 The Bayes-Kalman Filter and Linear Dynamical Systems

So far, we have focused on dynamical models for discrete random variables. However, the same approach can be applied to continuous variables. Defining dynamical models for continuous variables can be challenging because there are many ways in which continuous dynamics can be expressed. Perhaps the simplest assumption is that of linear dynamics—an assumption that leads to the model known as the *Bayes-Kalman filter* (or just *Kalman filter*).

A Generative Model for Dynamic Sequences The Bayes-Kalman filter was originally developed in the 1950s and early 1960s for tracking aircraft and spacecraft (Kalman, 1960). It is a technique commonly used in prediction and control. But it has also been applied to many problems in cognitive psychology, including classical conditioning (Daw, Courville, & Dayan, 2008) and causal learning (Lu et al., 2016). It was applied to vision to model human perception of a moving target dot over time in the presence of randomly moving background dots (Yuille, Burgi, & Grzywacz, 1998). It is applicable to any problem where you want to estimate a state z_t that changes over n discrete time steps t and you have some noisy observations x_t.

For the sake of concreteness, we will focus on the traditional task of estimating the position of an object over time, but the formulation presented here generalizes naturally to these other cases. Suppose that you want to estimate the object's position z_1 and you have observations $x_1, \ldots, x_n$ drawn from a distribution $p(x|z)$. If these observations are independent, and the object is static, then you can estimate z from them by maximum likelihood $z^*_{ML} = \arg\max_z \prod_{t=1}^{n} p(x_t|z)$. If you have prior knowledge $p(z)$ about the position of the object, then a better estimate is the MAP value $z^*_{MAP} = \arg\max_z p(z) \prod_{t=1}^{n} p(x_t|z)$. But what happens if the object is moving? Suppose that the position of z_1 changes with time, so it is a sequence $z_1, \ldots, z_n$. In this case, you have to provide a model $p(z_{t+1}|z_t)$ for how the object is moving.

This model is illustrated graphically in figure 5.12. It makes conditional independence assumptions that state z_t depends only on the previous time state z_{t-1}.

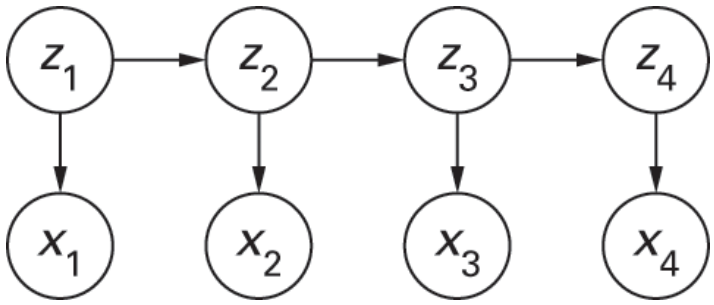


Figure 5.12

The graphical model assumed by the Bayes-Kalman filter. An object has changing position $z_1, \ldots, z_n$, generated by a distribution $p(z_{t+1}|z_t)$, and we have observations x_t sampled from $p(x_t|z_t)$.

Computing the Updates The Bayes-Kalman filter can be formulated in terms of an update of the distribution $p(z_{t+1}|\mathbf{x}_{t+1})$, where $\mathbf{x}_t = (x_1, \ldots, x_t)$ is all the observations taken up to time t. We update this distribution to $p(z_{t+1}|\mathbf{x}_{t+1})$ in two stages. First, we calculate

the probability $p(z_{t+1}|\mathbf{x}_t)$ of the position z_{t+1} of the object at time $t + 1$ (i.e., where the object will be in the future). This is given by

$$p(z_{t+1}|\mathbf{x}_t) = \int p(z_{t+1}|z_t)p(z_t|\mathbf{x}_t)\, dz_t. \tag{5.36}$$

Next, we must take into account the new measurement x_{t+1} using $p(z_{t+1}|\mathbf{x}_t)$ as the new "prior." This uses Bayes' rule (and the fact that $\mathbf{x}_{t+1} = (\mathbf{x}_t, x_{t+1})$) to obtain

$$p(z_{t+1}|\mathbf{x}_{t+1}) = \frac{p(x_{t+1}|z_{t+1})p(z_{t+1}|\mathbf{x}_t)}{p(x_{t+1}|\mathbf{x}_t)}. \tag{5.37}$$

The Bayes-Kalman filter is characterized by equations (5.36) and (5.37), which *predict* z_{t+1} and then *correct* this prediction using the new measurement x_{t+1}, respectively. The filter is initialized by setting $p(z_1|x_1) = p(x_1|z_1)p(z_1)/p(x_1)$, where $p(z_1)$ is the prior for the original position of the object at the start of the sequence.

Observe that the Bayes-Kalman filter is really a procedure for updating the probability distribution $p(z_t|\mathbf{x}_t)$ representing the current beliefs about the location of the object. To estimate an exact position for the object, we need to estimate z_{t+1} from $p(z_{t+1}|\mathbf{x}_t)$. In the Bayes-Kalman filter, this can be done analytically because the prior $p(z_t)$, the dynamics $p(z_{t+1}|z_t)$, and the observation model $p(x_t|z_t)$ are all Gaussian distributions. In this case, $p(z_t|x_t)$, $p(z_{t+1}|\mathbf{x}_t)$, and $p(z_{t+1}|\mathbf{x}_{t+1})$ are also all Gaussian distributions (this shouldn't be a surprise—it happens because the priors and likelihoods for all the inferences that we need to make are conjugate).

The One-Dimensional Case Using (μ, σ^2) to denote the Gaussian distribution with mean μ and standard deviation σ, we can write the simplest 1D model as

$$p(x_t|z_t) = \mathcal{N}(x_t, \sigma_m^2),\ p(z_{t+1}|z_t) = \mathcal{N}(z_t + \nu, \sigma_p^2),\ p(z_1) = \mathcal{N}(\mu_1, \sigma_1^2), \tag{5.38}$$

where ν_t is the mean distance traveled by the object from t to $t + 1$. If we write the distribution $p(z_t|\mathbf{x}_t)$ as a Gaussian with mean μ_t and standard deviation σ_t (i.e., $\mathcal{N}(\mu_t, \sigma_t^2)$) then we can reexpress the prediction and correction update equations (equations (5.36) and (5.37)) as

$$p(z_{t+1}|\mathbf{x}_t) = \mathcal{N}(\mu_t + \nu, \sigma_p^2 + \sigma_t^2) \tag{5.39}$$

$$p(z_{t+1}|\mathbf{x}_{t+1}) = \mathcal{N}(\mu_{t+1}, \sigma_{t+1}^2), \tag{5.40}$$

where

$$\mu_{t+1} = \mu_t + \nu - \frac{(\sigma_p^2 + \sigma_t^2)[(\mu_t + \nu) - x_{t+1}]}{\sigma_m^2 + (\sigma_p^2 + \sigma_t^2)},$$

$$\sigma_{t+1}^2 = \frac{\sigma_m^2(\sigma_p^2 + \sigma_t^2)}{\sigma_m^2 + (\sigma_p^2 + \sigma_t^2)}. \tag{5.41}$$

$t+1$ t

The update for includes a prediction part (+) and a correction part (the rest). Observe that if the object is at the mean $t+1$ t

predicted position (i.e., x = +), then the correction part disappears. Also, note that the update combines the various sources of $t+1$ t

information—the observation x and the mean estimated position + by a linear weighted average similar to that seen for other Gaussian models that we have considered.

We can get a better understanding of the Bayes-Kalman filter by considering special cases. If the observations are noiseless $t+1$ $t+1$

(i.e., $\sigma_m^2 = 0$), then it follows that = x , so we should forget the history and just use the current observation as our estimate $t+1$

of z . If $\sigma_p^2 = 0$, then we have perfect prediction, and so $\mu_{t+1} = \frac{\sigma_t^2}{\sigma_m^2+\sigma_t^2} x_{t+1} + \frac{\sigma_m^2}{\sigma_m^2+\sigma_t^2}(\mu_t + \nu)$ with $\sigma_{t+1}^2 = \frac{\sigma_m^2\sigma_t^2}{\sigma_m^2+\sigma_t^2}$, $t+1$ t

which corresponds to taking the weighted average of x with + . If we also require that = 0 (i.e., the object does not move), then we obtain $\mu_{t+1} = \frac{\sigma_t^2}{\sigma_m^2+\sigma_t^2} x_{t+1} + \frac{\sigma_m^2}{\sigma_m^2+\sigma_t^2}\mu_t$, which is simply an online method for computing the MAP estimate of a static object at position x (as described in our first model earlier in this chapter).

5.6.1 An Example: Multiple Object Tracking

A common paradigm that is used in the psychophysical literature on visual attention is the Multiple Object Tracking (MOT) task (see figure 5.13). In this task, people are shown an array of objects (typically represented as geometric shapes such as circles) and a subset of the objects are indicated as the ones to pay attention to, either by flashing or being highlighted with a color. Then all the objects begin to move, and the participant has to try to keep track of which were the indicated objects. After a period of motion, the participant is asked to click on the objects that they think they were supposed to be tracking.

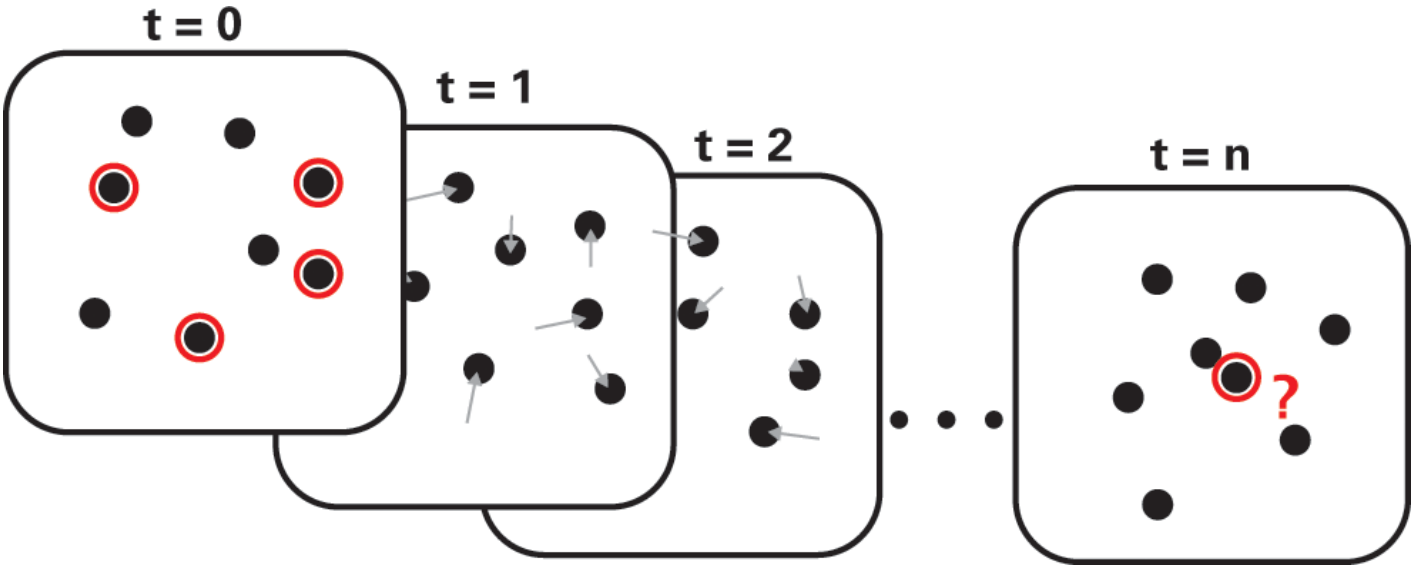


Figure 5.13
The Multiple Object Tracking (MOT) task. A set of objects are shown on the screen, and a subset are highlighted as targets to be tracked. After the objects move randomly for a while, participants have to indicate the ones that they were supposed to be tracking. Figure reproduced with permission from Vul et al. (2009).

Research using the MOT task has revealed a number of interesting properties of human performance: people can track only a limited number of objects, but the exact number depends on other aspects of the task, such as the speed and spacing of the objects. A natural question to ask is whether these limitations are an intrinsic consequence of the structure of the computational problem being solved, or whether they reflect a fundamental limitation of the human visual system.

Vul, Alvarez, Tenenbaum, and Black (2009) set out to answer this question, formulating a probabilistic model of the MOT task that can be used to evaluate optimal performance. If the goal is to model the motion of a single object, then the problem can be formulated in the terms outlined here—a linear dynamical system, which can be solved using the Bayes-Kalman filter. But with multiple objects, there is a new challenge: if two objects get close to one another, it is plausible that they could be confused for one another. To address this, Vul et al. (2009) introduced another variable into their model that identifies which object is which. Successfully solving the MOT problem requires maintaining correct assignments for the objects that are supposed to be tracked. The motion of the objects is modeled using a separate linear dynamical system for each object, and the assignment variable links these linear dynamical systems to the corresponding objects on the screen. If the assignment of objects to linear dynamical systems is known, inference in each system simply reduces to the Bayes-Kalman filter. Exploiting this, Vul et al. (2009) used a sampling algorithm to maintain estimates of the assignments, applying the Bayes-Kalman filter separately with each sample (for more details of this kind of approach, see the next chapter).

Vul et al. (2009) found that some of the basic results from the MOT literature could be captured by this model: easier tracking of objects moving at lower speeds or with greater separation is just a consequence of the statistical structure of the problem. However, this approach doesn't explain why people find it harder to track more objects. For example, people find it easier to track 4 out of 16 objects than 8 out of 16 objects, even though from the model's perspective both cases require modeling the dynamics of 16 objects. Vul et al. (2009) suggested that this phenomenon must reflect an additional constraint on the visual system, which they were able to capture effectively by assuming that people experience more uncertainty about the location of objects when they are asked to track more objects.

5.7 Combining Probabilistic Models

One of the strengths of probabilistic generative models is that they are easy to combine. Since a generative model is just a procedure for generating data, we can make one step in that procedure another generative model. For example, we can define a Markov model whose emission probabilities are mixture distributions, or a mixture model whose component distributions are Markov processes. This ability to compose generative models in various ways creates the capacity to define richly structured probabilistic models that can be applied to increasingly complex problems.

5.7.1 An Example: Combining Syntax and Semantics

We have discussed two simple models of language in this chapter: HMMs, which capture a rudimentary form of syntax, and

topic models, which capture a rudimentary form of semantics. Since both are expressed as generative models, we can combine them into a single model that incorporates both “syntax” and “semantics” and makes it possible to identify the different roles that words play in documents.

Griffiths, Steyvers, Blei, and Tenenbaum (2005) defined a probabilistic generative model for text that combines latent Dirichlet allocation and a hidden Markov model—appropriately called the LDA-HMM (see figure 5.14). This *composite model* assumes that a sequence of words is generated by an HMM, but that one state in that HMM is a distribution over words that depends on the particular document. The emission probabilities for that state are determined by a topic model, where the weight given to each topic depends on the document in exactly the way specified in the latent Dirichlet allocation model introduced previously.

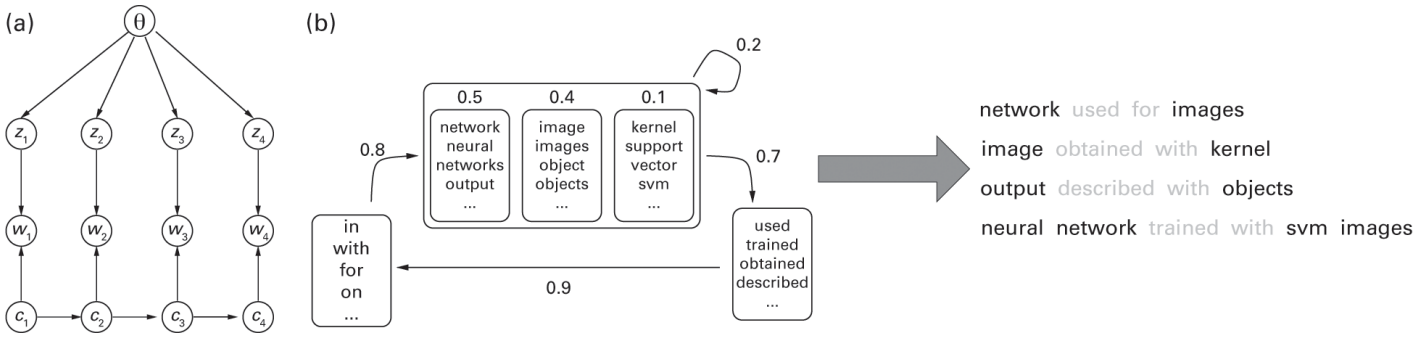


Figure 5.14

The LDA-HMM model combines an HMM with a topic model. (a) Graphical model, with θ_i being the mixture weights of different topics in a document, z_i being a topic, c_i a syntactic class, and w_i a word. (b) The distribution over words given classes and topics is a standard HMM, with one special state in which the distribution over words comes from a document-specific mixture of topics. As a result, words are naturally factorized into two kinds—those generated by the HMM “syntax” and those generated by the topic model “semantics.” Figure reproduced with permission from Griffiths et al. (2004).

The LDA-HMM makes it possible to capture the two kinds of roles that words can play in documents—function words that make up part of the standard structure of sentences, regardless of the documents in which they appear, and content words that vary across documents. In the model, each word that appears in the document appears in one of these roles—either it comes from the general HMM states or from the special document-dependent state. In practice, this model “factorizes” the words that appear in sentences into these two parts, and makes it possible to simultaneously estimate both semantic representations and syntactic classes within a single model (see figure 5.15).

BLOOD	FOREST	FARMERS	GOVERNMENT	LIGHT	WATER	STORY	DRUGS	BALL
HEART	TREES	LAND	STATE	EYE	MATTER	STORIES	DRUG	GAME
PRESSURE	FORESTS	CROPS	FEDERAL	LENS	MOLECULES	POEM	ALCOHOL	TEAM
BODY	LAND	FARM	PUBLIC	IMAGE	LIQUID	CHARACTERS	PEOPLE	*
LUNGS	SOIL	FOOD	LOCAL	MIRROR	PARTICLES	POETRY	DRINKING	BASEBALL
OXYGEN	AREAS	PEOPLE	ACT	EYES	GAS	CHARACTER	PERSON	PLAYERS
VESSELS	PARK	FARMING	STATES	GLASS	SOLID	AUTHOR	EFFECTS	FOOTBALL
ARTERIES	WILDLIFE	WHEAT	NATIONAL	OBJECT	SUBSTANCE	POEMS	MARIJUANA	PLAYER
*	AREA	FARMS	LAWS	OBJECTS	TEMPERATURE	LIFE	BODY	FIELD
BREATHING	RAIN	CORN	DEPARTMENT	LENSES	CHANGES	POET	USE	BASKETBALL
THE	IN	HE	*	BE	SAID	CAN	TIME	,
A	FOR	IT	NEW	HAVE	MADE	WOULD	WAY	;
HIS	TO	YOU	OTHER	SEE	USED	WILL	YEARS	(
THIS	ON	THEY	FIRST	MAKE	CAME	COULD	DAY	:
THEIR	WITH	I	SAME	DO	WENT	MAY	PART	)
THESE	AT	SHE	GREAT	KNOW	FOUND	HAD	NUMBER	
YOUR	BY	WE	GOOD	GET	CALLED	MUST	KIND	
HER	FROM	THERE	SMALL	GO		DO	PLACE	
MY	AS	THIS	LITTLE	TAKE		HAVE		
SOME	INTO	WHO	OLD	FIND		DID		

Figure 5.15

Results of applying the LDA-HMM to the TASA corpus. The top part of the table shows topics from the LDA model, comprised of words that were inferred to be content words in the various documents. The bottom part shows syntactic classes corresponding to the states of the HMM, comprised of words that were inferred to be function words. Each column shows a separate distribution over words, ordered in decreasing probability. Figure reproduced with permission from Griffiths et al. (2004).

5.8 Summary

In chapter 4, we showed how graphical models provide a way to express the independence assumptions that make probabilistic inference tractable. In this chapter, we have explored how we can define increasingly complex probabilistic models while continuing to exploit conditional independence to make it possible to estimate parameters and perform inference. In particular, by postulating latent variables that play a role in generating the observed data, we can define models that begin to capture the richer structure that underlies the observable world. By making careful choices about how those latent variables relate to one another and to observed data, we can define expressive models that nonetheless remain tractable. Mixture models, topic models, HMMs, and the Bayes-Kalman filter all extend the kinds of observed data that we can hope to understand via Bayesian inference. However, even with these models—and increasingly as we try to go beyond them—we can run into challenges that make exact probabilistic inference difficult. In the next chapter, we consider methods of addressing these challenges.

i i j

1. The constant of proportionality is determined by $f(x, x)\,dx$, being $\frac{1}{n_c}$ if $f(x, x)\,dx = 1$ for all i, and is absorbed into to produce direct equivalence to equation (5.7)).

6

Approximate Probabilistic Inference

Thomas L. Griffiths and Adam N. Sanborn

The probability distributions that we need to evaluate when applying Bayesian inference can quickly become very complicated. Graphical models provide some tools for speeding up probabilistic inference, but these tools tend to work best when most variables are directly dependent on a relatively small number of other variables. There are still many cases where calculating posterior distributions or conditional probabilities is a computational challenge. For example, how can we perform Bayesian inference over continuous variables when we cannot evaluate the integral required to calculate the denominator of Bayes' rule? Alternatively, how can we compute posterior distributions over large discrete hypothesis spaces, such as the space of all directed graphs defined on a set of variables?

The computational complexity of probabilistic inference leads to two kinds of problems. The first is an engineering problem: how can we as modelers calculate the predictions of our probabilistic models? The second is a reverse-engineering problem: how could human behavior be consistent with the predictions of these models if probabilistic inference is so hard to do? In this chapter, we will present methods for approximating probabilistic computations that are relevant to both of these problems, but we will focus on the engineering problem. We will examine to the reverse-engineering problem in detail in chapter 11.

There are various approaches to approximating probabilistic inference, including generic methods for performing numerical integration or optimization and methods developed specifically for probabilistic inference, such as loopy belief propagation (Pearl, 1988) and variational inference (Jordan, Ghahramani, Jaakkola, & Saul, 1999). All these methods are relevant to solving the engineering problem of working with probabilistic models, and many of them are likely to shed light on how human minds and brains might perform probabilistic inference. Our focus in this chapter will be on *Monte Carlo methods*, in which probabilistic computations are approximated by replacing probability distributions with samples from those distributions. Monte Carlo methods are one of the basic tools for approximate inference, and they can be adapted to work with a wide range of probabilistic models. They can also provide a clear source of hypotheses about how human minds and brains might deal with the severe computational challenges involved in probabilistic inference. At the end of the chapter, we will briefly discuss variational inference, which has become increasingly popular in the machine learning community with the development of automated tools for optimization. We recommend Neal (1993), Mackay (1998), and Robert and Casella (1999) for more details on Monte Carlo methods, and Blei, Kucukelbir, and McAuliffe (2017) as an introduction to variational inference.

6.1 Simple Monte Carlo

The basic idea behind Monte Carlo methods is to represent a probability distribution by a set of samples from that distribution. Those samples provide an indication of which values have high probability (since high probability values are more likely to be produced as samples), and they can be used in place of the distribution itself when performing various computations. When working with probabilistic models of cognition, we are typically interested in understanding the posterior distribution over a parameterized model—such as a causal network with its causal strength parameters—or over a class of models—such as the space of all causal network structures on a set of variables or all taxonomic tree structures on a set of objects. Samples from the posterior distribution can be useful in discovering the best parameter values for a model or the best models in a model class, and for estimating how concentrated the posterior is on those best hypotheses (i.e., how confident a learner should be in those hypotheses).

Sampling can also be used to approximate averages over the posterior distribution. For example, in computing the posterior probability of a parameterized model given data, it is necessary to compute the model's marginal likelihood, or the average probability of the data over all parameter settings of the model. Averaging over all parameter settings is also necessary for ideal

Bayesian prediction about future data points (as in computing the posterior predictive distribution for a weighted coin). Finally, we could be interested in averaging over a space of model structures, making predictions about model features that are likely to hold regardless of which structure is correct. For example, we could estimate how likely it is that one variable *A* causes another variable *B* in a complex causal network of unknown structure by computing the probability that a link *A* → *B* exists in a high-probability sample from the posterior over network structures (e.g., Friedman & Koller, 2000).

Monte Carlo methods were originally developed primarily for approximating these sophisticated averages—that is, approximating a sum over all of the values taken on by a random variable with a sum over a random sample of those values. Assume that we want to evaluate the average (also called the *expected value*) of a function $f(\mathbf{x})$ over a probability distribution $p(\mathbf{x})$ defined on a set of k random variables taking on values $\mathbf{x} = (x_1, x_2, \ldots, x_k)$. This can be done by taking the integral of $f(\mathbf{x})$ over all values of $\mathbf{x}$, weighted by their probability $p(\mathbf{x})$, with

$$E_{p(\mathbf{x})}[f(\mathbf{x})] = \int f(\mathbf{x})p(\mathbf{x})\, d\mathbf{x}, \tag{6.1}$$

where $E[\cdot]$ denotes the expectation. For convenience, we will denote this expected value , with $\mu = E_{p(\mathbf{x})}[f(\mathbf{x})]$.

Monte Carlo provides an alternative to explicitly evaluating an expectation. The *law of large numbers*, a standard result in probability theory, indicates that the average of a set of samples from a probability distribution will approximate the expectation with respect to that distribution increasingly well as the number of samples increases. This is exactly what we do in Monte Carlo, giving us the approximation

$$\int f(\mathbf{x})p(\mathbf{x})\, d\mathbf{x} \approx \frac{1}{m}\sum_{i=1}^{m} f(\mathbf{x}^{(i)}) = \hat{\mu}_{MC}, \tag{6.2}$$

where the $\mathbf{x}^{(i)}$ are a set of m samples from the distribution $p(\mathbf{x})$. This procedure—replacing $p(\mathbf{x})$ with samples from $p(\mathbf{x})$—is called *simple Monte Carlo*.

The estimator $\hat{\mu}_{MC}$ has several desirable properties. Under weak conditions on the functions $f(\mathbf{x})$ and $p(\mathbf{x})$, it is *consistent*, with $(\hat{\mu}_{MC}) \to 0$ almost surely as $m \to \infty$. It is also *unbiased*, with $E[\hat{\mu}_{MC}] = \mu$. Finally, it is *asymptotically normal*, with

$$\sqrt{m}(\hat{\mu}_{MC} - \mu) \to \mathcal{N}(0, \sigma^2_{MC}), \tag{6.3}$$

in distribution, where $\mathcal{N}(0, \sigma^2_{MC})$ is the Gaussian with mean 0 and variance $\sigma^2_{MC} = E_{p(\mathbf{x})}\left[(f(\mathbf{x}) - \mu)^2\right]$. These properties make simple Monte Carlo the method of choice in cases where it is straightforward to sample from $p(\mathbf{x})$ and the fluctuations in $f(\mathbf{x})$ are not correlated with $p(\mathbf{x})$.

6.1.1 An Example: Computing Causal Support

To show how the Monte Carlo approach to approximate numerical integration is useful for evaluating Bayesian models, recall the model of causal structure-learning introduced in chapter 4, causal support (Griffiths & Tenenbaum, 2005). To compute the evidence that a set of contingencies d provides in favor of a causal relationship, we needed to evaluate the integral

$$P(d|\text{Graph 1}) = \int_0^1 \int_0^1 P_1(d|w_0, w_1, \text{Graph 1})\, P(w_0, w_1|\text{Graph 1})\, dw_0\, dw_1, \tag{6.4}$$

where $P(w_0, w_1|Graph\ 1)$ is assumed to be uniform over all values of w_0 and w_1 between 0 and 1 and $P(d|w_0, w_1, Graph\ 1)$ is

derived from the noisy-OR parameterization

$$P_1(e^+|b,c;w_0,w_1) = 1-(1-w_0)^b(1-w_1)^c, \tag{6.5}$$

with e^+ being the occurrence of the effect, and b and c indicating the presence (1) or absence (0) of the background b and cause c on a given trial. If we view $P(d|w_0, w_1, \textit{Graph 1})$ simply as a function of w_0 and w_1, it is clear that we can approximate this integral using Monte Carlo. The analog of equation (6.2) is

$$P(d|\text{Graph 1}) \approx \frac{1}{m}\sum_{i=1}^{m} P_1(d|w_0^{(i)}, w_1^{(i)}, \text{Graph 1}), \tag{6.6}$$

where the $w_0^{(i)}$ and $w_1^{(i)}$ are m samples from the distribution $P(w_0, w_1|\textit{Graph 1})$. This is the approach to computing causal support taken in Griffiths and Tenenbaum (2005).

6.2 When Does Simple Monte Carlo Fail?

One limitation of the simple Monte Carlo method is that it is not easy to automatically generate samples from most probability distributions. Using the Monte Carlo method requires that we are able to generate samples from distribution $p(\mathbf{x})$. However, this is often not the case. For distributions like the uniform, Gaussian, exponential, Poisson, gamma, and beta, there are straightforward algorithms for generating random samples, but these are particularly well behaved distributions (in fact, they are all exponential family distributions, as introduced in chapter 3). Other distributions can require ingenuity to sample from, and developing a method for sampling from such distributions can be a research topic in its own right. When we want to use Monte Carlo to sample from posterior distributions computed by applying Bayes' rule, we often run into distributions with a very specific problem that makes it hard to apply simple Monte Carlo methods.

When dealing with posterior distributions, we can end up in a situation where we don't actually know the probability of a particular hypothesis—we just know the value of that probability up to a constant. If we have a prior distribution $P(h)$, a likelihood function $p(d|h)$, and an observed data point d, the posterior distribution is given by

$$P(h|d) = \frac{p(d|h)P(h)}{\sum_{h'} p(d|h')P(h')}, \tag{6.7}$$

where the sum in the denominator ranges over all hypotheses. Since the denominator is what ensures that $P(h|d)$ is "normalized"—that it is a well-defined probability distribution, with the probabilities of all the hypotheses summing to 1—this sum is often referred to as the *normalizing constant*. With a large discrete hypothesis space—say, all partitions of a set of n objects or all directed graphs on n variables—calculating the normalizing constant can quickly become intractable. If it's easy to compute the numerator, we end up in a situation where we can calculate something that is *proportional* to $P(h|d)$, and we can compare the relative values of these probabilities for different hypotheses, but we know their actual probability only up to a constant.

The same issue arises when we apply Bayes' rule to continuous variables. In this case, we have

$$p(\theta|d) = \frac{p(d|\theta)p(\theta)}{\int p(d|\theta)(\theta)\,d\theta}, \tag{6.8}$$

where the integral in the denominator may not have an analytic solution. In this case, we can try approximating that integral numerically—something that becomes increasingly difficult as the dimensionality of θ increases—or we can rely on methods that require us to know $p(\theta|d)$ only up to a constant.

In these situations, simple Monte Carlo is not a viable solution—we don't even know the probability distribution we want to

sample from, let alone how to sample from it. However, there are a number of more sophisticated Monte Carlo methods that can be applied even when we know a probability distribution only up to a constant.

6.3 Rejection Sampling

Assume that we have a distribution $p(\mathbf{x})$ that is difficult to sample from—perhaps because we know its value only up to a constant. If another distribution is close to $p(\mathbf{x})$ but easy to sample from, *rejection sampling* can often be used to generate samples from $p(\mathbf{x})$.

Changing the Sampling Problem Say that there is a distribution $q(\mathbf{x})$ that we can sample from easily. We will call this the *proposal distribution*. Further, assume that we can compute the probabilities $p(\mathbf{x})$ up to a constant; that is, we can compute $p^*(\mathbf{x})$ $p(\mathbf{x})$. Imagine, in addition, that we know a constant c such that $c\,q(\mathbf{x}) \geq p^*(\mathbf{x})$ for all $\mathbf{x}$. We can then generate a sample by performing the following steps:

1. Sample $\mathbf{x} \sim q(\mathbf{x})$,
2. Generate a random real number r uniformly drawn from the interval $[0, c\,q(\mathbf{x})]$,
3. If $r < p^*(\mathbf{x})$, then keep $\mathbf{x}$; otherwise, reject and start again at Step 1.

The result is a sample from $p(\mathbf{x})$.

Why does this work? It is easy to see this intuitively when $p(x)$ is a one-dimensional (1D) distribution (see figure 6.1, where $q(x)$ is taken to be the uniform distribution). The two random values, x and r, give the coordinates of a point in a two-dimensional (2D) plane. This point is drawn uniformly at random from a region that includes the $p^*(x)$ curve (in fact, it is the region under $cq(x)$). If we only keep points under the $p^*(x)$ curve, then we sample each value x with probability proportional to $p^*(x)$. Since $p^*(x) \propto p(x)$, this is equivalent to sampling from the distribution $p(x)$.

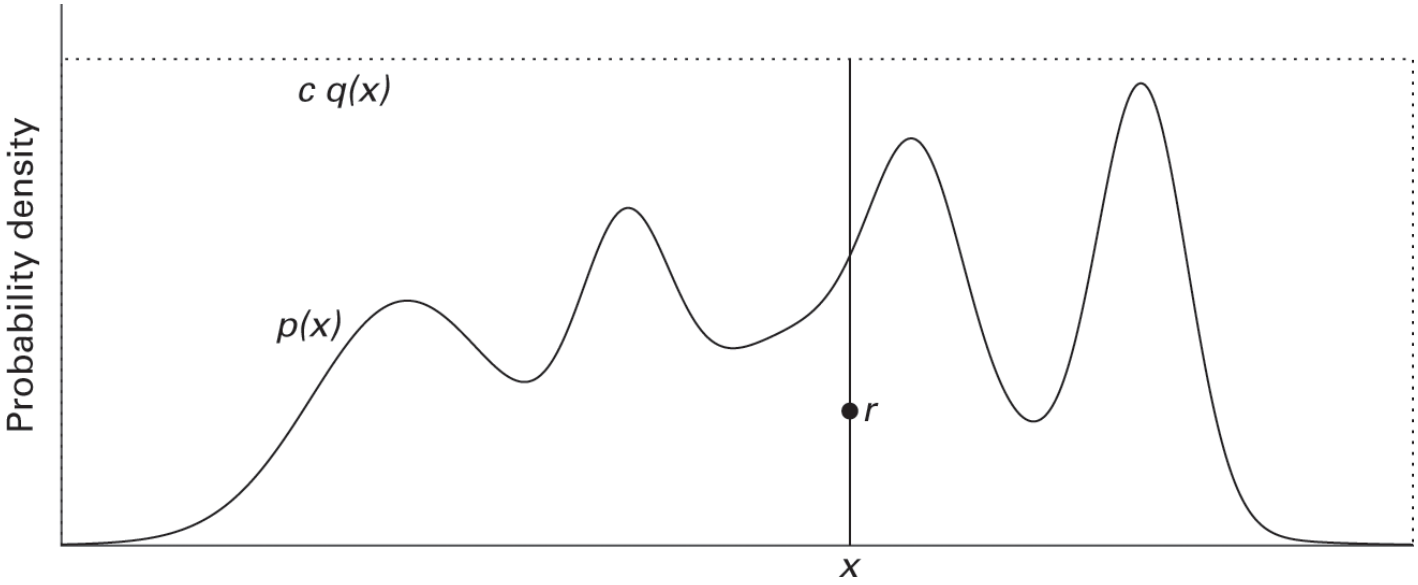


Figure 6.1

Rejection sampling. Here, our target distribution is $p(x)$ and we take the uniform distribution over the range of $p(x)$ as the distribution $q(x)$ from which we generate our initial samples x. We set c such that $p(x) < cq(x)$ for all x. We sample x from $q(x)$ and then r uniformly from $[0, c\,q(x)]$. If r $p(x)$, as in this case, we keep x. Conditioned on keeping x, the probability of sampling a given x is just $p(x)$. The vertical axis here is arbitrary, so this procedure can be followed even if we know $p(x)$ only up to a constant.

Bayesian Inference by Rejection Sampling Rejection sampling can be used to sample from posterior distributions, and hence to approximate Bayesian inference. We will start with the case where the data and hypotheses are discrete, so we want to sample from $P(h|d)$.

The first step is to construct a distribution to sample from. We can write the posterior as a joint distribution on hypothesis h *

and data d, $P(h, d) = P(h)P(d|h)(d, d)$, where $(\cdot, \cdot)$ is 1 when its arguments match and 0 otherwise. This looks a little odd, but it amounts to the assumption that we are sampling hypothesis h from the prior, then sampling data d from the likelihood $P(d|h)$, but only keeping data that match the observed data d. This defines a distribution that puts all its mass on the observed data, while technically being a joint distribution on h and d.

*

If we use the proposal distribution $Q(h, d) = P(h)P(d|h)$, then it is clear that $P(h, d) \quad Q(h, d)$ because $(\cdot, \cdot)$ only takes on the * *

values 0 and 1. Consequently, $c = 1$. Furthermore, since $P(h, d)$ is equal to $Q(h, d)$ when $d = d$ and is zero otherwise, we will reject a sample if and only if $d \quad d$.

This provides a simple way to draw from the posterior distribution via rejection sampling: sample hypothesis h from the prior, then sample data d from the likelihood $P(d|h)$ associated with this hypothesis and reject the sampled hypothesis if d does not match the observed data d. The probability of rejecting a hypothesis h is thus $P(d|h)$, and the hypotheses that we keep will be samples from the posterior distribution.

Extensions and Limitations This approach generalizes naturally to continuous hypotheses: we sample from the prior $p()$ and retain those samples with probability $P(d|)$. However, it cannot be used when the data are continuous because the probability of those data becomes infinitesimally small. In this situation, researchers have used a related method known as *approximate Bayesian computation* (ABC; Beaumont, Zhang, & Balding, 2002), where a second function is used to define a measure of proximity among data sets. Samples are then drawn from the prior, used to simulate data, and retained if the simulated data are close to the observed data.

While rejection sampling is simple, it is often not efficient. Since we only retain each sample with probability $P(d|h)$, the proportion of the time that we retain samples is given by the expectation of $P(d|h)$ with respect to $P(h)$, which is $\sum_h P(d|h)P(h) = P(d)$. This approach will thus only be efficient when the probability of the observed data is

high—typically meaning that the amount of observed data is small. However, rejection sampling is a good source of intuitions about approximating Bayesian inference and can be a starting point for defining a strategy for performing Bayesian inference in complex models.

6.4 Importance Sampling

Rejection sampling exactly generates samples from the target distribution $p(\mathbf{x})$, but it can be very inefficient. An alternative approach that starts from the same idea of generating samples from the wrong distribution and then correcting for this is known as *importance sampling*.

Changing the Sampling Problem Assume that we want to evaluate an expectation of a function $f(\mathbf{x})$ with respect to a probability distribution $p(\mathbf{x})$ but have another distribution $q(\mathbf{x})$ that is easier to sample from. Further, assume that $q(\mathbf{x}) > 0$ whenever $p(\mathbf{x}) > 0$.

We can introduce $q(\mathbf{x})$ into our expectation by multiplying and dividing by the same term, giving

$$E_{p(\mathbf{x})}[f(\mathbf{x})] = \int f(\mathbf{x})p(\mathbf{x})\,d\mathbf{x} = \frac{\int f(\mathbf{x})p(\mathbf{x})\,d\mathbf{x}}{\int p(\mathbf{x})\,d\mathbf{x}} = \frac{\int f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x})\,d\mathbf{x}}{\int \frac{p(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x})\,d\mathbf{x}}. \tag{6.9}$$

The numerator and denominator of the final expression are each expectations with respect to $q(\mathbf{x})$. Applying simple Monte Carlo to approximate these expectations, we obtain

$$E_{p(\mathbf{x})}[f(\mathbf{x})] \approx \frac{\frac{1}{m}\sum_{j=1}^{m} f(\mathbf{x}_j)\frac{p(\mathbf{x}_j)}{q(\mathbf{x}_j)}}{\frac{1}{m}\sum_{j=1}^{m}\frac{p(\mathbf{x}_j)}{q(\mathbf{x}_j)}}, \tag{6.10}$$

where each sample $\mathbf{x}_j$ is drawn from $q(\mathbf{x})$.

The ratios $\frac{p(\mathbf{x}_j)}{q(\mathbf{x}_j)}$ can be interpreted as "weights" on the sample $\mathbf{x}_j$, correcting for the fact that they were drawn from $q(\mathbf{x})$ rather than $p(\mathbf{x})$ (see figure 6.2). Samples that have higher probability under $p(\mathbf{x})$ than $q(\mathbf{x})$, and thus occur less often than they would if we were sampling from $p(\mathbf{x})$, receive greater weight. More formally, define the weight w_j to be

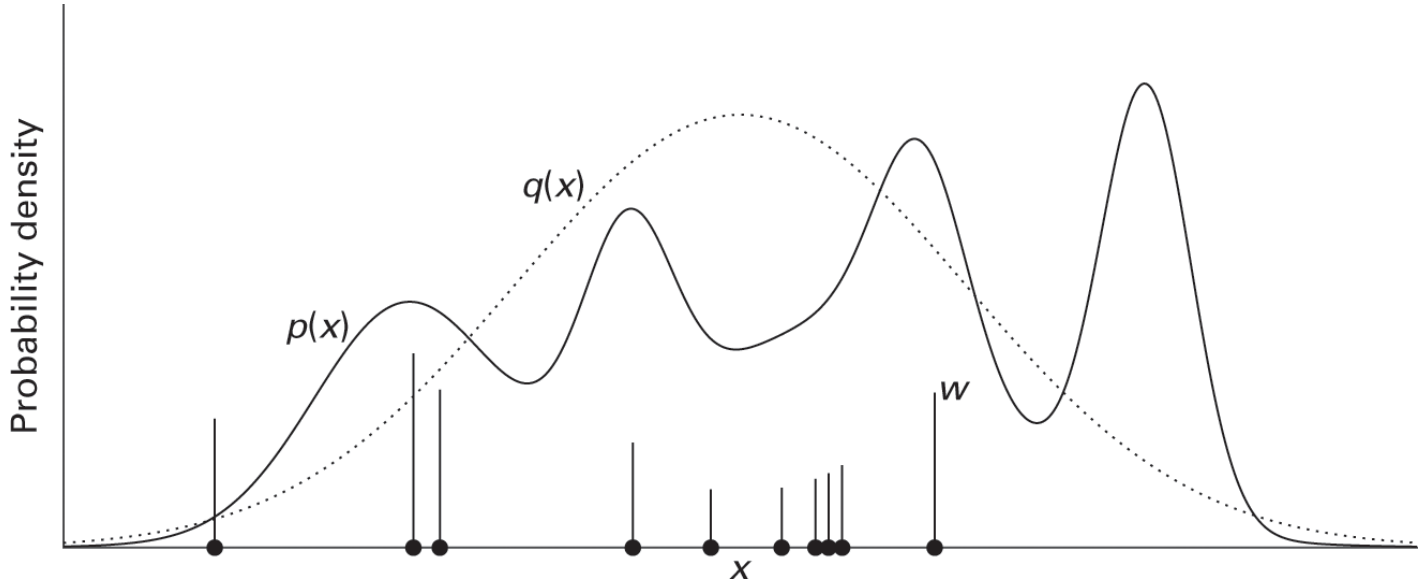


Figure 6.2
Importance sampling. We want to sample values of x from $p(x)$. Instead, we sample values of x from $q(x)$ and correct for the fact that we sampled from the wrong distribution by assigning each sampled x a weight w that is proportional to $\frac{p(x)}{q(x)}$. These weights are higher when $p(x) > q(x)$, corresponding to regions where we undersampled, and lower when $p(x) < q(x)$, corresponding to regions where we oversampled.

$$w_j = \frac{\frac{p(\mathbf{x}_j)}{q(\mathbf{x}_j)}}{\sum_{j=1}^{m} \frac{p(\mathbf{x}_j)}{q(\mathbf{x}_j)}}. \tag{6.11}$$

We can then write

$$E_{p(\mathbf{x})}[f(\mathbf{x})] \approx \sum_{j=1}^{m} f(\mathbf{x}_j) w_j = \hat{\mu}_{IS}. \tag{6.12}$$

j

Since the only role of $p(\mathbf{x})$ is in defining the weight w, importance sampling can still be used even if we know $p(\mathbf{x})$ only up to a constant. Because the weights are normalized, the unknown constant in $p(\mathbf{x})$ appears in both the numerator and the denominator and cancels out.

IS

Analysis of Importance Sampling It is instructive to compare the properties of the the importance sampling estimator $\hat{\mu}$ with that of simple Monte Carlo. Under similar assumptions about $f(\mathbf{x})$ and $p(\mathbf{x})$ (with the significant addition that $q(\mathbf{x}) > 0$ for all

IS

$\mathbf{x}$ such that $p(\mathbf{x}) > 0$), it is a consistent estimator of , with ($\hat{\mu}$) 0 almost surely as m . It is also asymptotically normal, with

$$\sqrt{m}(\hat{\mu}_{IS} - \mu) \rightarrow N(0, \sigma_{IS}^2) \tag{6.13}$$

IS

in distribution, where $\sigma_{IS}^2 = E_{p(\mathbf{x})}\left[\left(f(\mathbf{x}) - E_{p(\mathbf{x})}[f(\mathbf{x})]\right)^2 \frac{p(\mathbf{x})}{q(\mathbf{x})}\right]$. However, unlike simple Monte Carlo, $\hat{\mu}$ is biased, with

$$\hat{\mu}_{IS} - \mu = \frac{1}{m}\left(E_{p(\mathbf{x})}\left[f(\mathbf{x})\right]E_{p(\mathbf{x})}\left[\frac{p(\mathbf{x})}{q(\mathbf{x})}\right] - E_{p(\mathbf{x})}\left[f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}\right]\right), \tag{6.14}$$

which goes to zero as m , but it can be substantial for smaller values of m.

Despite being biased, there are several reasons why using an importance sampler can make sense even in cases where simple Monte Carlo can be applied. First, it allows a single set of samples to be used to evaluate expectations with respect to a range of distributions, through the use of different weights for each distribution. Second, the estimate produced by the importance sampler can have lower variance than the estimate produced by simple Monte Carlo. If the function $f(\mathbf{x})$ takes on its most extreme values in regions where $p(\mathbf{x})$ is small, the variance of the simple Monte Carlo estimate can be large. An importance sampler can have lower variance than simple Monte Carlo if $q(\mathbf{x})$ is chosen to be complementary to $f(\mathbf{x})$. In particular, the asymptotic variance of the sampler is minimized by specifying $q(\mathbf{x})$ as

$$q(\mathbf{x}) \propto |f(\mathbf{x}) - E_{p(\mathbf{x})}\left[f(\mathbf{x})\right]|p(\mathbf{x}). \tag{6.15}$$

$p(\mathbf{x})$

This is not a practical procedure since finding this distribution requires computing E $[f(\mathrm{x})]$, but the fact that the minimum variance sampler need not be $p(\mathbf{x})$ illustrates that importance sampling can sometimes provide a lower variance estimate of an expectation than simple Monte Carlo.

Bayesian Inference by Importance Sampling We can use importance sampling to perform Bayesian inference. The distribution that we want to approximate is the posterior, $p(h|d)$. We can do this by choosing another distribution on hypotheses $q(h)$ from which to generate samples, assigning to the resulting samples weights that are proportional to $p(h|d)/q(h)$. One attractive property of this approach is that since the weights are simply proportional to this ratio, we can equivalently assign weights that are proportional to $p(d|h)p(h)/q(h)$. This gives us a method for generating samples from the posterior distribution without having to calculate the normalizing constant—a significant reduction in the cost of computation.

The analysis of the bias and variance of importance sampling presented here makes it clear that this method is most effective when $q(x)$ and $p(x)$ are similar enough that the weights do not vary too much. Bayesian inference using importance sampling will thus be most effective if we can find a proposal distribution $q(h)$ that is close to the posterior distribution $p(h|d)$. In some cases—when the observed data have a small effect on an agent's beliefs, corresponding to a likelihood function $p(d|h)$ that does not take on extreme values—we can simply use the prior distribution $p(h)$ as our proposal distribution. In this case, the weights assigned to each sample are simply $p(d|h)p(h)/p(h) = p(d|h)$, the likelihood. The resulting algorithm for approximating Bayesian inference is extremely simple: generate samples from the prior $p(h)$, and assign those samples weights proportional to the likelihood $p(d|h)$. For obvious reasons, this algorithm is known as *likelihood weighting*.

6.4.1 An Example: Computing Spatial Coincidences

The Bayesian model of spatial coincidences proposed by Griffiths and Tenenbaum (2007a), discussed in chapter 3, requires computing a complicated integral. In general, this is the challenge of applying the Bayesian Occam's razor—calculating the marginal probability of all observed data for a particular model requires integrating over all parameters of that model. In the case of spatial coincidences, this means integrating over the parameters of a mixture model.

0

The model assumed in the spatial coincidences scenario is relatively simple: either bombs are falling at random (h) or some

1 B

of them are falling around a target (h). If n bombs fall within some region at random, then the probability of any set of bomb locations $\mathbf{x}$ is just $p(\mathbf{x}|h_0) = \frac{1}{|\mathcal{R}|^{n_B}}$. If all the bombs were falling around a target, that would also be relatively easy to assess—we would want to calculate the marginal probability of $\mathbf{x}$ under a Gaussian distribution, integrating over the mean and

1

covariance of that Gaussian, $p(\mathbf{x}|h$) = $p(\mathbf{x}|,$)$p()p()$ d d. Even though this sounds intimidating, it is actually straightforward to do if we use conjugate priors for and —in this case, a normal distribution on (or even a uniform distribution) and an inverse-Wishart prior on (see Minka, 2001, for details).

The tricky part arises when *some* of the bombs are targeted and others fall at random. Then we have a mixture model, where

one component is uniform over and the other is a Gaussian with unknown mean and variance. If we know which bombs were targeted, then the problem is still easy—we integrate the probability of the bombs that were targeted over and , and the others follow a uniform distribution over . Letting **z** denote the vector of assignments of bombs to mixture components, we can thus easily compute $p(\mathbf{x}|\mathbf{z})$. The problem is that we don't know **z**, so we need to sum over all the possible values it could take on.

Writing this out, we have

$$p(\mathbf{x}|h_1) = \sum_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})P(\mathbf{z}), \tag{6.16}$$

where $P(\mathbf{z})$ is the prior distribution on assignments. The problem is that if there are n bombs, there are 2^B possible values for **z**—a number that becomes large quickly. Using simple Monte Carlo is a tempting solution—we could just sample values of **z** from $P(\mathbf{z})$ and then average together the resulting values of $p(\mathbf{x}|\mathbf{z})$. Unfortunately this doesn't work well in cases where some of the bombs really are targeted, because most choices of **z** won't pick out the targeted bombs and consequently have very small values for $p(\mathbf{x}|\mathbf{z})$, while others will get close and have very large values. The answer that simple Monte Carlo produces is going to depend more on how lucky the choice of samples was than on what the actual answer is.

Computing marginal probabilities for complex models such as mixture models is one of the places where importance sampling is extremely effective. The trick is to choose a proposal distribution $Q(\mathbf{z})$ that assigns higher probabilities than $P(\mathbf{z})$ to those values of **z** for which $p(\mathbf{x}|\mathbf{z})$ is large. One method that has been proposed for doing this is using an inference algorithm such as the Expectation-Maximization (EM) algorithm to estimate the parameters of the mixture model, and then using those parameters to define $Q(\mathbf{z})$. In this case, we could estimate and by trying to find values that seemed to correspond to a good location for the target in each bombing scene, and then calculate $P(\mathbf{z}|\mathbf{x}, ,)$ and take this as $Q(\mathbf{z})$. There are some perils involved in this—it will pick out one possible target, but there could be others that are equally plausible—so the standard practice is to take $Q(\mathbf{z})$ as a mixture of this and the prior $P(\mathbf{z})$. This is the method that was used to compute the probabilities that were compared to human behavior (figuring out how to do this and implementing the resulting algorithm took longer than running the experiment and analyzing the data!).

6.5 Sequential Monte Carlo

A basic problem that arises in probabilistic models of cognition is explaining how learners can deal with the severe computational challenges posed by probabilistic inference. We have discussed how some of these problems can be addressed using Monte Carlo methods in a static setting, where all the data are available and the learner simply has to evaluate a set of hypotheses. However, the challenges seem to become even more severe in contexts where dynamic probabilistic inference is required, either because of intrinsic dynamics of a model, such as the assumption that the state of the world can change over time, or because more data gradually becomes available. These problems can be solved using *sequential Monte Carlo*.

6.5.1 Dealing with a Dynamic World

Sequential Monte Carlo methods are schemes for generating samples from a sequence of probability distributions, typically using the relationship between successive distributions to use samples from one distribution to generate samples from the next. The standard example of such a method is the *particle filter*, which is a sequential version of importance sampling. The particle filter was originally developed for making inferences about variables in a dynamic environment—the problem of "filtering" is inferring the current state of the world given a sequence of observations.

Assume that we have a probabilistic model with a structure similar to a hidden Markov model (HMM), where a sequence of latent variables $z_1, \ldots, z_n$ are responsible for generating a sequence of observed variables $x_1, \ldots, x_n$. In the simplest case, each z_i is generated from a distribution that depends only on z_{i-1}, and each x_i is generated from a distribution that depends only on z_i. Assume that we want to compute $p(z_n|x_1, \ldots, x_n)$. While efficient algorithms exist for computing this probability when the z_i are discrete (such as the forward procedure for HMMs), or when z_i is continuous and $p(z_i|z_{i-1})$ is a Gaussian distribution whose mean

$i1$

is a linear function of z (the Kalman filter discussed in chapter 5), the particle filter provides a way to perform inference that generalizes beyond these cases.

$n\ 1$ n

The basic idea behind the particle filter is that we are going to approximate $p(z|x, \ldots, x)$ using importance sampling. In

n $n\ 1$ $n1$

particular, we can perform importance sampling where we use as our proposal the "prior" on z, $p(z|x, \ldots, x)$. If we can

$n\ 1$ $n1$

generate samples from $p(z|x, \ldots, x)$, then we can construct an importance sampler by giving each sample weight proportional

$n\ n$ $n\ 1$ $n1$

to $p(x|z)$. How can we generate samples from $p(z|x, \ldots, x)$? Well, we know that

$$p(z_n|x_1, \ldots, x_{n-1}) = \sum_{z_{n-1}} p(z_n|z_{n-1})p(z_{n-1}|x_1, \ldots, x_{n-1}), \tag{6.17}$$

$n1\ 1$ $n1$ j

so if we can generate samples $z_{n-1}^{(j)}$ from $p(z|x, \ldots, x)$ with weights w, we can just sample from the distribution

$$p(z_n|x_1, \ldots, x_{n-1}) \approx \sum_j p(z_n|z_{n-1}^{(j)})w_j, \tag{6.18}$$

which is a simple mixture distribution. Alternatively, we could stratify our sampling, making sure that the samples are more diverse, by generating one sample from $p(z_n|z_{n-1}^{(j)})$ for each $z_{n-1}^{(j)}$.

$n\ 1$

The procedure outlined in the previous paragraph identifies an interesting recursion—it gives us a way to sample from $p(z|x, \ldots, x)$ so long as we can sample from $p(z|x, \ldots, x)$. This sets us up to introduce the sequential Monte Carlo scheme that

n $n1\ 1$ $n1$

$1\ 1$

underlies the particle filter (see figure 6.3). First, we generate samples from $p(z|x)$. Then, for each sample $z_1^{(j)}$ we generate $z_2^{(j)}$

j

from $p(z_2|z_1^{(j)})$ and assign each resulting sample a weight w proportional to $p(x_2|z_2^{(j)})$. This step is equivalent to the likelihood weighting procedure introduced previously. We repeat this process for all $i = 3, \ldots, n$, multiplying the old weight of each sample $z_i^{(j)}$ (now called a "particle") by $p(x_i|z_i^{(j)})$. There is no need to normalize the particle weights at each step—this can be done at the end of the process.

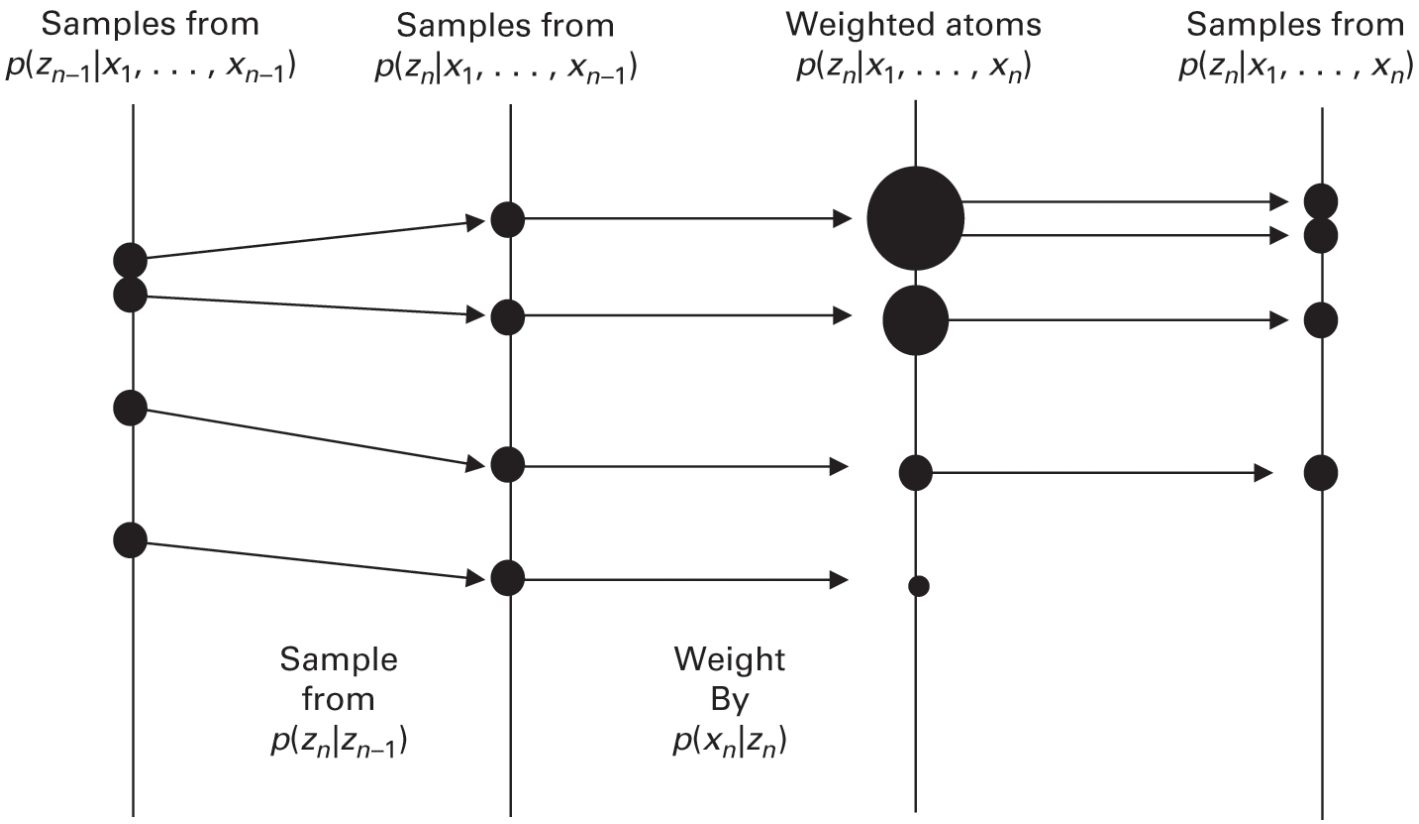


Figure 6.3

The particle filter. We start with samples from $p(z_{n-1}|x_1, \ldots, x_{n-1})$. We then obtain samples from $p(z_n|x_1, \ldots, x_n)$ by sampling from $p(z_n|z_{n-1})$ for each sample of z_{n-1}. We calculate the likelihood $p(x_n|z_n)$ for each of these z_n, using them to assign weights to the samples. If we sample from this weighted distribution, we obtain samples from $p(z_n|x_1, \ldots, x_n)$.

This simple recursive scheme is known as *sequential importance sampling*. However, it has one big problem: over time, the weights of the particles can diverge significantly since some of the particles are likely to have ended up with a sequence of z_i values that are very unlikely. In some ways, this is a waste of computation, since those particles with very small weights will make little contribution to the final representation of the distribution of z_n. To address this problem, we can use an alternative approach known as *sequential importance resampling*. Under this approach, we monitor the variance of the normalized particle weights. When this variance exceeds a preset threshold, we sample a new set of particles from a probability distribution corresponding to the normalized weights. This increases the number of particles that are in a good part of the space. The weights are then reset to be uniform over particles, and the process continues. Resampling decreases the diversity in the set of particles, so if the intrinsic dynamics determined by $p(z_i|z_{i-1})$ do not result in a lot of variability, other schemes for introducing diversity (such as perturbing the values of the particles) may need to be used.

While the standard particle-filtering algorithm follows this schema, this only scratches the surface of possible sequential Monte Carlo techniques (for a comprehensive review, see Doucet, de Freitas, & Gordon, 2001; Chopin & Papaspiliopoulos, 2020). Even within particle filtering, many options can be used to improve performance on certain problems. For example, in some cases, it is possible to compute $p(z_i|x_1, \ldots, x_{i-1})$ directly, and it is better to sample from this distribution. In other cases, we

can take into account the value of the observation x_i in proposing a value for z_i, increasing the accuracy of our proposals and decreasing the variance of the weights. As with other Monte Carlo methods, tailoring the algorithm to fit a particular probabilistic model is something of an art.

6.5.2 Accumulating Evidence

While working with dynamic models is important, a far more basic context in which we have to perform a challenging probabilistic computation is when we want to update our posterior distribution over hypotheses in light of a new piece of evidence. This requires recomputing the probability of every single hypothesis, which can be extremely demanding when we have a large number of hypotheses. However, sequential Monte Carlo can be useful in this context as well.

Imagine that we have a set of hypotheses h and a sequence of observations $d_1, \ldots, d_n$. We want to define an efficient scheme for updating our beliefs, allowing us to incrementally compute $p(h|d_1, \ldots, d_i)$ (i.e., giving us the posterior after every observation d_i) without requiring us to sum over all $|h|$ hypotheses each time we obtain a new observation. Again, imagine performing importance sampling, where we use $p(h|d_1, \ldots, d_{n-1})$ as our proposal. If the d_i are independent given h, then we can just give the resulting samples of h weights proportional to $p(d_n|h)$. So, as before, we have a way to obtain samples from $p(h|d_1, \ldots, d_{n-1})$, provided that we can obtain samples from $p(h|d_1, \ldots, d_{n-2})$. This allows us to set up the kind of recursion used in the particle filter.

The simplest sequential Monte Carlo scheme is as follows. First, generate samples from the prior $p(h)$. Then, as each observation d_i comes in, reweight those samples by $p(d_i|h)$. When we need to compute a posterior distribution, we can normalize the weights. This potentially gives us a set of weighted hypotheses as the approximation to the posterior distribution after each d_i.

While attractive in its simplicity, this scheme has a basic problem: the hypotheses we use in our approximation will be the same regardless of the data we observe, corresponding to the set we sampled from the prior. If the posterior after $d_1, \ldots, d_n$ is very different from the prior, this will result in a poor approximation. As a consequence, we need to introduce techniques such as perturbation of the particles to generate distinct hypotheses at each iteration. When perturbing the particles, our goal is to not change the distribution from which they are drawn—if they are samples from $p(h|d_1, \ldots, d_i)$ before perturbation, they should be samples from that distribution after perturbation. One way to achieve this is to perturb the particles by applying one or more iterations of a Markov chain Monte Carlo (MCMC) algorithm for the distribution $p(h|d_1, \ldots, d_i)$ (for details, see Fearnhead, 2002). We will discuss these algorithms in more detail after considering an example of sequential Monte Carlo in practice.

6.5.3 An Example: Inference for Multiple Object Tracking

In the model of multiple object tracking presented by Vul et al. (2008) and summarized in chapter 5, the presence of multiple objects creates a problem: even though each object follows linear dynamics with Gaussian noise, and can hence be tracked using a particle filter, when objects occupy similar locations they can become confused with one another. Vul et al. (2008) modeled this by introducing an additional variable, τ_t, which is a permutation of the indices of the objects at time t. With eight objects, say,

$_1$

the initial assignment of would be the vector (1, 2, 3, 4, 5, 6, 7, 8). However, if objects 1 and 2 crossed paths and became $_2$

confused with one another at the next time step, would become (2, 1, 3, 4, 5, 6, 7, 8). Part of multiple object tracking is thus not $_t$

just calculating a posterior distribution on the location of the objects, but calculating a posterior distribution on .

$_t$

Because of the way that the model is defined, if is known, then the objects just evolve according to their own $_t$ $_{t+1}$

linear-Gaussian dynamics. This means that the inference problem reduces to one of keeping track of . For each value of , it is $_t$

straightforward to compute the probability of the location of the objects at time $t + 1$ based on and their locations at previous times. That means that we have a perfect setup for a particle filter.

$_t$

Vul et al. (2008) assumed that each was independent—that the indices of the objects (and hence the potential confusions of $_t$

different objects) are drawn from the same distribution $P(\ )$ at each time t and are unrelated to those at the previous time t 1. The $_t$

data consist of a set of observations of the "motion" of each object **m** containing both the location and velocity for each object at $_1$ $_t$ $_1$

time t. The particle filter takes a set of samples of , …, that capture the posterior distribution on assignments based on **m** , …, $_t$ $_{t+1}$ $_{t+1}$

m , and updates these samples to reflect the posterior on after observing **m** . This can be done by augmenting each sample $_{t+1}$ $_{t+1}$ $_{t+1}$ $_1$

with a value of sampled from the prior $P(\ )$, and then assigning each resulting sample a weight proportional to p(**m** |**m** , …, $_t$ $_1$ $_{t+1}$

m , , …,), which can be computed by applying the linear dynamics and Gaussian noise assumed by the model to the objects $_{t+1}$ $_{t+1}$

indicated by . Conditioned on , each object's position evolves as in the Kalman filter discussed in the previous chapter.

6.6 Markov Chain Monte Carlo

One of the most flexible methods for generating samples from a probability distribution is *Markov chain Monte Carlo (MCMC)*, which can be used to construct samplers for arbitrary probability distributions even if the normalizing constants of those distributions are unknown. MCMC algorithms were originally developed to solve problems in statistical physics (Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953), and are now widely used across physics, statistics, machine learning, and related fields (for more details, see Newman & Barkema, 1999; Gilks, Richardson, & Spiegelhalter, 1996; Mackay, 2003; Neal, 1993). As with other Monte Carlo methods, MCMC algorithms provide tools for working with probability distributions and hypotheses about processes by which people could approximate Bayesian inference. In chapter 10, we also explore the idea that MCMC algorithms can be used to design behavioral experiments that allow us to sample from people's subjective probability distributions.

6.6.1 Markov Chains

As the name suggests, Markov chain Monte Carlo is based upon the theory of Markov chains—sequences of random variables in which each variable is conditionally independent of all previous variables given its immediate predecessor. The probability that a variable in a Markov chain takes on a particular value conditioned on the value of the preceding variable is determined by the

$(i+1)$ (i) (i)

transition kernel for that Markov chain. The transition kernel $K(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)})$ indicates the probability that a chain in state $\mathbf{x}^{(i)}$ at step i changes to state $\mathbf{x}^{(i+1)}$ at step $i + 1$. When the states are discrete, this information can be encoded in a matrix where the rows correspond to the values at the next iteration, and the columns correspond to the distributions specified by the kernel. This is called a *transition matrix*.

One well-known property of Markov chains is their tendency to converge to a *stationary distribution*: as the length of a Markov chain increases, the probability that a variable in that chain takes on a particular value converges to a fixed quantity determined by the choice of transition kernel. The stationary distribution is defined to be the distribution $\pi(\mathbf{x})$ that satisfies

$$\pi(\mathbf{x}) = \int K(\mathbf{x}|\mathbf{x}')\pi(\mathbf{x}')d\mathbf{x}, \tag{6.19}$$

meaning that if we sample a state $\mathbf{x}$ from $\pi(\mathbf{x})$ and then choose the next state $\mathbf{x}$ by sampling from the distribution $K(\mathbf{x}|\mathbf{x})$, the probability distribution over states remains $\pi(\mathbf{x})$. This is the sense in which this distribution is "stationary": once the probability distribution over states reaches the stationary distribution, it never changes. Convergence to the stationary distribution can be (m) (1) (m)

described formally by defining $p(\mathbf{x}^{(m)}|\mathbf{x}^{(1)})$ to be the probability of the Markov chain being in state $\mathbf{x}^{(m)}$ after m iterations, having started in state $\mathbf{x}^{(1)}$. Provided the Markov chain satisfies conditions of *ergodicity* (for details, see Norris, 1997), this probability will converge to the stationary distribution as m becomes large. That is,

$$\lim_{m\to\infty} p(\mathbf{x}^{(m)}|\mathbf{x}^{(1)}) = \pi(\mathbf{x}^{(m)}) \tag{6.20}$$

(1) (m)

for all $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(m)}$. Consequently, if we sample from the Markov chain by picking some initial value and then repeatedly sampling from the distribution specified by the transition kernel, we will ultimately generate samples from the stationary (m)

distribution. For the same reason, averaging a function $f(\mathbf{x})$ over the values of $\mathbf{x}^{(m)}$ will approximate the average of that function [1]

over the probability distribution $\pi(\mathbf{x})$ as m becomes large.

6.6.2 The Metropolis-Hastings Algorithm

In MCMC, a Markov chain is constructed such that its stationary distribution is the distribution from which we want to generate $(i+1)$ (i)

samples. If the target distribution is $p(\mathbf{x})$, then the transition kernel $K(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)})$ would be chosen to yield $\pi(\mathbf{x}) = p(\mathbf{x})$ as the stationary distribution. Fortunately, there is a simple procedure that can be used to construct a transition kernel that will have that property for any choice of $p(\mathbf{x})$. This procedure is known as the *Metropolis-Hastings algorithm* (Hastings, 1970; Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller, 1953).

$(i+1)$ (i)

The Metropolis-Hastings algorithm defines a kernel $K(\mathbf{x}^{*}|\mathbf{x}^{(i)})$ as the result of two probabilistic steps. The first uses an $*$ $(i+1)$

arbitrary *proposal distribution*, $q(\mathbf{x}^{*}|\mathbf{x}^{(i)})$, to generate a proposed value $\mathbf{x}^{*}$ for $\mathbf{x}^{(i+1)}$. The second is to decide whether to accept this proposal, using information about how the proposal was generated and its probability under the target distribution. The key to making sure that the resulting Markov chain converges to the target distribution is in how the *acceptance function* is defined.

The acceptance function used in the Metropolis-Hastings algorithm is

$$A(\mathbf{x}^*, \mathbf{x}^{(i)}) = \min\left[\frac{p(\mathbf{x}^*)q(\mathbf{x}^{(i)}|\mathbf{x}^*)}{p(\mathbf{x}^{(i)})q(\mathbf{x}^*|\mathbf{x}^{(i)})}, 1\right], \tag{6.21}$$

where $p(\mathbf{x})$ is our target distribution. If a random number generated from a uniform distribution over [0, 1] is less than $A(\mathbf{x}^*, \mathbf{x}^{(i)})$, the proposed value $\mathbf{x}^*$ is accepted as the value of $\mathbf{x}^{(i+1)}$. Otherwise, the Markov chain remains at its previous value, and $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)}$. Intuitively, equation (6.21) says that we will be more likely to accept proposals if they have a high probability under our target distribution, and less likely to accept them if the proposal distribution strongly favors the proposed value $\mathbf{x}^*$ over the current value $\mathbf{x}^{(i)}$. These two considerations mean that our Markov chain spends more time in regions where $p(\mathbf{x})$ is high and is not unduly affected by the vagaries of the proposal distribution. If you're curious where equation (6.21) comes from, we will explain that in detail in the next section.

So, to summarize the algorithm: Start with an initial value $\mathbf{x}^{(1)}$. Define a proposal distribution $q(\mathbf{x}^*|\mathbf{x}^{(1)})$, and sample $\mathbf{x}^*$ given $\mathbf{x}^{(2)}$. Decide whether to accept $\mathbf{x}^*$ by applying equation (6.21), substituting in the target distribution $p(\mathbf{x})$. If accepted, $\mathbf{x}^{(2)} = \mathbf{x}^*$; if not, $\mathbf{x}^{(2)} = \mathbf{x}^{(1)}$. Repeat many times until the Markov chain has plausibly converged to its stationary distribution, and an average over the resulting $\mathbf{x}^{(m)}$ will approximate the expectation of the same quantity over $p(\mathbf{x})$.

To illustrate the use of the Metropolis-Hastings algorithm, we can use it to generate samples from a Gaussian distribution. For simplicity, we will assume that the Gaussian has zero mean and unit variance, but the same procedure could be used in more interesting cases, such as for sampling from the posterior distribution on the mean of a Gaussian. It is easy to sample from Gaussians in general, so we have no need to use MCMC, but it provides a way to demonstrate the steps that are involved. The state space of our Markov chain is a single dimension that ranges from to . We need to choose a proposal distribution to take us from state to state. In general, it is good to choose a proposal that is neither too big (meaning that the vast majority of proposals will get rejected) nor too small (meaning that proposals will get accepted but not result in any real movement around the state space). We will use a Gaussian for our proposal, with a mean corresponding to the previous value and a variance 0.2 times the variance of the target distribution. Repeating the process of generating a proposal and deciding whether to accept it for 500 steps with a total of three different starting points produces the results shown in figure 6.4.

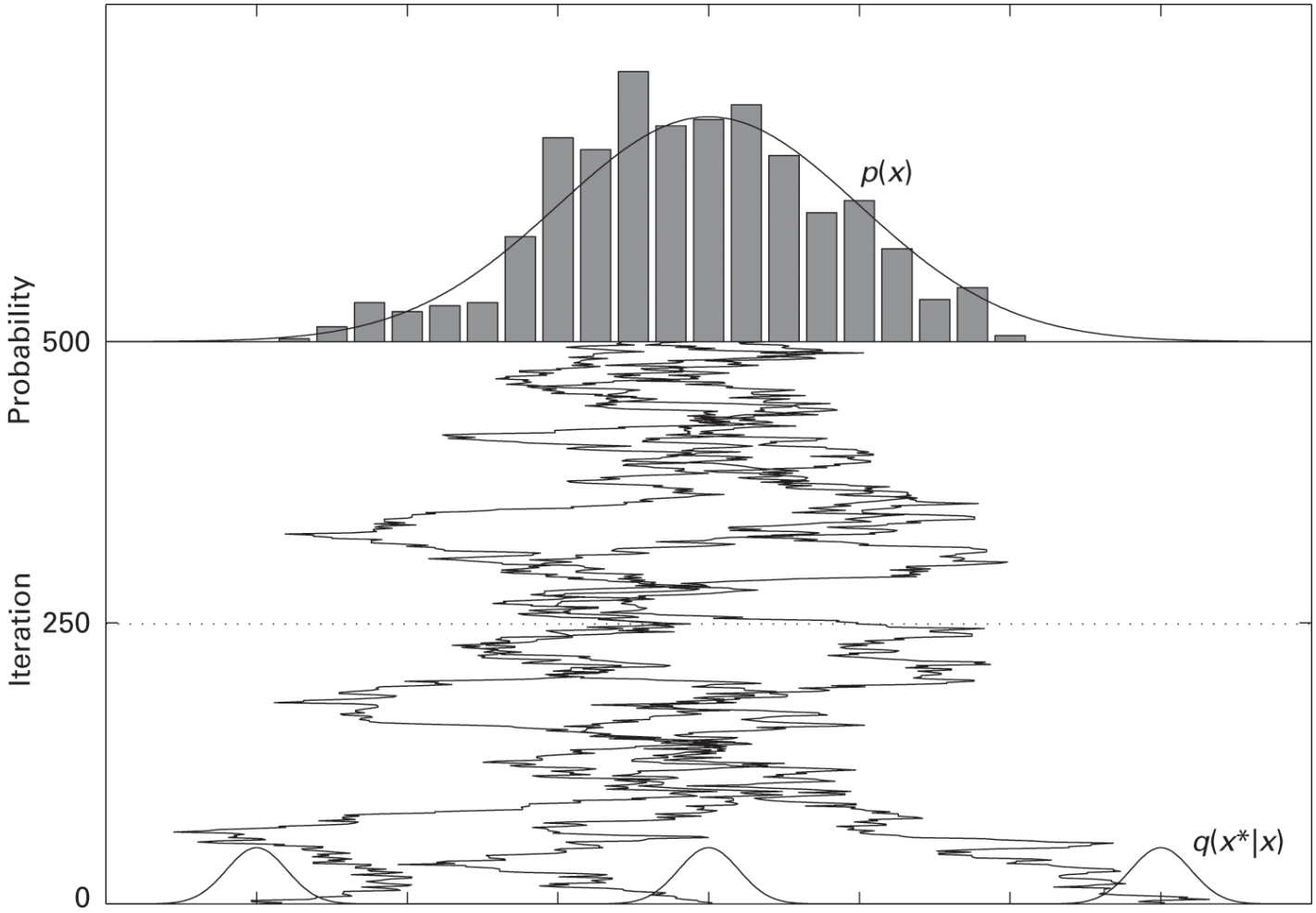


Figure 6.4

The Metropolis-Hastings algorithm. The solid lines shown in the bottom part of the figure are three sequences of values sampled from a Markov chain. Each chain began at a different location in the space, but used the same transition kernel. The transition kernel was constructed using the procedure described in the text for the Metropolis-Hastings algorithm: the proposal distribution, $q(x^*|x)$, was a Gaussian distribution with mean x and standard deviation 0.2 (shown centered on the starting value for each chain at the bottom of the figure), and the acceptance probabilities were computed by taking $p(x)$ to be Gaussian with mean 0 and standard deviation 1 (plotted with a solid line in the top part of the figure). This guarantees that the stationary distribution associated with the transition kernel is $p(x)$. Thus, regardless of the initial value of each chain, the probability that the chain takes on a particular value will converge to $p(x)$ as the number of iterations increases. In this case, all three chains move to explore a similar part of the space after around 100 iterations. The histogram in the top part of the figure shows the proportion of time that the three chains spend visiting each part in the space after 250 iterations (marked with the dotted line), which closely approximates $p(x)$. Samples from the Markov chains can thus be used similarly to samples from $p(x)$. Figure reproduced with permission from Griffiths, Kemp, and Tenenbaum (2008c).

One advantage of the Metropolis-Hastings algorithm is that it requires only limited knowledge of the probability distribution $p(\mathbf{x})$. Inspection of equation (6.21) reveals that, in fact, the Metropolis-Hastings algorithm can be applied even if we know the value of $p(\mathbf{x})$ only up to a constant, since only the ratio of these quantities affects the algorithm. This can be extremely useful when we are performing Bayesian inference, as we have previously demonstrated that the hard part of applying Bayes' rule is usually computing the normalizing constant. Using Metropolis-Hastings, we do not need to compute this normalizing constant to generate samples from the posterior—we only need to know the likelihood and prior. This makes it far easier to work with probabilistic models since defining the likelihood and prior simply requires thinking about the generative process and inductive biases that are relevant, rather than taking into account concerns about analytic tractability. As a consequence, we can go beyond the restrictive world of conjugate priors and begin to explore richer and more expressive models.

6.6.3 Analyzing the Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm seems a little like magic—it lets us define a Markov chain that converges to whatever

stationary distribution we like. How is this possible? The key is the acceptance function given by equation (6.21). We can derive this acceptance function, and explain why it works, using another property of Markov chains.

Assume that we have a Markov chain that has a transition kernel $K(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)})$ that satisfies the equation

$$\pi(\mathbf{x})K(\mathbf{x}'|\mathbf{x}) = \pi(\mathbf{x}')K(\mathbf{x}|\mathbf{x}') \tag{6.22}$$

for all $\mathbf{x}$ and $\mathbf{x}'$, where $\pi(\mathbf{x})$ is the stationary distribution of the Markov chain. This property is called *detailed balance*—intuitively, it means that once the Markov chain reaches its stationary distribution, the probability of observing a transition from $\mathbf{x}$ and $\mathbf{x}'$ or $\mathbf{x}'$ and $\mathbf{x}$ is the same (Markov chains that have this property are described as being *reversible*). Detailed balance is a stronger condition than the existence of a stationary distribution—you can check that you can obtain equation (6.19) just by integrating over $\mathbf{x}^{(i)}$ on both sides of equation (6.22).

The acceptance function used in the Metropolis-Hastings algorithm is derived using detailed balance. As before, let $A(\mathbf{x}^*, \mathbf{x}^{(i)})$ be the probability of accepting $\mathbf{x}^*$ in state $\mathbf{x}^{(i)}$. Each time a proposal is rejected, it adds a little bit of probability mass to $\mathbf{x}^{(i)}$, so the probability of no proposal being accepted, which we will denote as $A = 0$, is

$$P(A=0|\mathbf{x}^{(i)}) = \int \left[1 - A(\mathbf{x}^*, \mathbf{x}^{(i)})\right] q(\mathbf{x}^*|\mathbf{x}^{(i)})\, d\mathbf{x}^* = 1 - \int A(\mathbf{x}^*, \mathbf{x}^{(i)}) q(\mathbf{x}^*|\mathbf{x}^{(i)})\, d\mathbf{x}^*. \tag{6.23}$$

We can thus write the transition kernel as

$$K(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)}) = q(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)})A(\mathbf{x}^{(i+1)}, \mathbf{x}^{(i)}) + \delta(\mathbf{x}^{(i+1)}, \mathbf{x}^{(i)})P(A=0|\mathbf{x}^{(i)}), \tag{6.24}$$

where $\delta(\mathbf{x}^{(i+1)}, \mathbf{x}^{(i)})$ is the delta function picking out $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)}$. Now we just need to specify $A(\mathbf{x}^*, \mathbf{x}^{(i)})$ so the stationary distribution associated with this kernel is $p(\mathbf{x})$.

We can rewrite detailed balance (equation 6.22) as

$$\frac{\pi(\mathbf{x})}{\pi(\mathbf{x}')} = \frac{K(\mathbf{x}|\mathbf{x}')}{K(\mathbf{x}'|\mathbf{x})}. \tag{6.25}$$

This is trivially satisfied when $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)}$, so we can focus on $\mathbf{x}^{(i+1)} \neq \mathbf{x}^{(i)}$, in which case our kernel $K(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)})$ is just $q(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)})A(\mathbf{x}^{(i+1)}, \mathbf{x}^{(i)})$. Putting this into the detailed balance equation, we get

$$\frac{\pi(\mathbf{x})}{\pi(\mathbf{x}')} = \frac{q(\mathbf{x}|\mathbf{x}')A(\mathbf{x}, \mathbf{x}')}{q(\mathbf{x}'|\mathbf{x})A(\mathbf{x}', \mathbf{x})}, \tag{6.26}$$

which we can rearrange to

$$\frac{A(\mathbf{x}', \mathbf{x})}{A(\mathbf{x}, \mathbf{x}')} = \frac{q(\mathbf{x}|\mathbf{x}')\pi(\mathbf{x}')}{q(\mathbf{x}'|\mathbf{x})\pi(\mathbf{x})}. \tag{6.27}$$

If we can define an acceptance function that satisfies this relationship, we will have defined a Markov chain that has $p(\mathbf{x})$ as its stationary distribution. Any acceptance function $A(\mathbf{x}^{(i)}, \mathbf{x}^*)$ that is proportional to $q(\mathbf{x}^{(i)}, \mathbf{x}^*)p(\mathbf{x}^*)$ will thus result in a Markov chain with $p(\mathbf{x})$ as its stationary distribution. It is straightforward to check that this is the case for the acceptance function given in equation (6.21), which is used in the Metropolis-Hastings algorithm because it maximizes the chance of proposals being accepted (and thus minimizes $P(A = 0|\mathbf{x}^{(i)})$). However, we will have the opportunity to explore some other valid acceptance functions in chapter 10.

6.6.4 Gibbs Sampling

If we can sample from distributions related to $p(\mathbf{x})$, we can use other MCMC methods. In particular, if we are able to sample from the conditional probability distribution for each variable in a set given the remaining variables, $p(x_j|x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_d)$, we can use another popular algorithm, *Gibbs sampling* (Geman & Geman, 1984; Gilks et al., 1996), which is known in statistical physics as the *heatbath* algorithm (Newman & Barkema, 1999). The Gibbs sampler for a target distribution $p(\mathbf{x})$ is the Markov chain defined by drawing each x_j from the conditional distribution $p(x_j|x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_d)$. You can actually derive the Gibbs sampler from the Metropolis-Hastings algorithm: if you propose to sample just one element of $\mathbf{x}$ from its conditional distribution given the values of the other elements, the proposal is always accepted. This is exactly what is done in Gibbs sampling. You can either cycle through the x_j in turn, or choose one at random to sample at each iteration.

Gibbs sampling has a lot in common with the EM algorithm, but it is typically used to generate samples from a posterior distribution, while the EM algorithm produces a point estimate. The commonalities between the algorithms can be illustrated by considering how to apply Gibbs sampling to the Gaussian mixture model where the components have known variances but unknown means considered in the previous chapter (the graphical model is reproduced in figure 6.5 for convenience). In this case, we want to compute the posterior distribution over the assignments of observations to cluster $\mathbf{z}$ and the component means $\mu^{(1)}$ and $\mu^{(2)}$ and mixing proportions π, given the observations $\mathbf{x}$ (here simplifying things by assuming that our Gaussians are just 1D). Starting from some initial values for $\mathbf{z}$, $\mu^{(1)}$, and $\mu^{(2)}$, the Gibbs sampler cycles through these variables, drawing each from its distribution given all other variables. This means that the algorithm effectively alternates between sampling the assignments of observations to components and sampling the values of the parameters. These two parts of the algorithm are similar to the E- and M-steps of the EM algorithm, although sampling is used for both latent variables and parameters, while EM uses one process (expectation) to deal with latent variables and another (maximization) to deal with parameters.

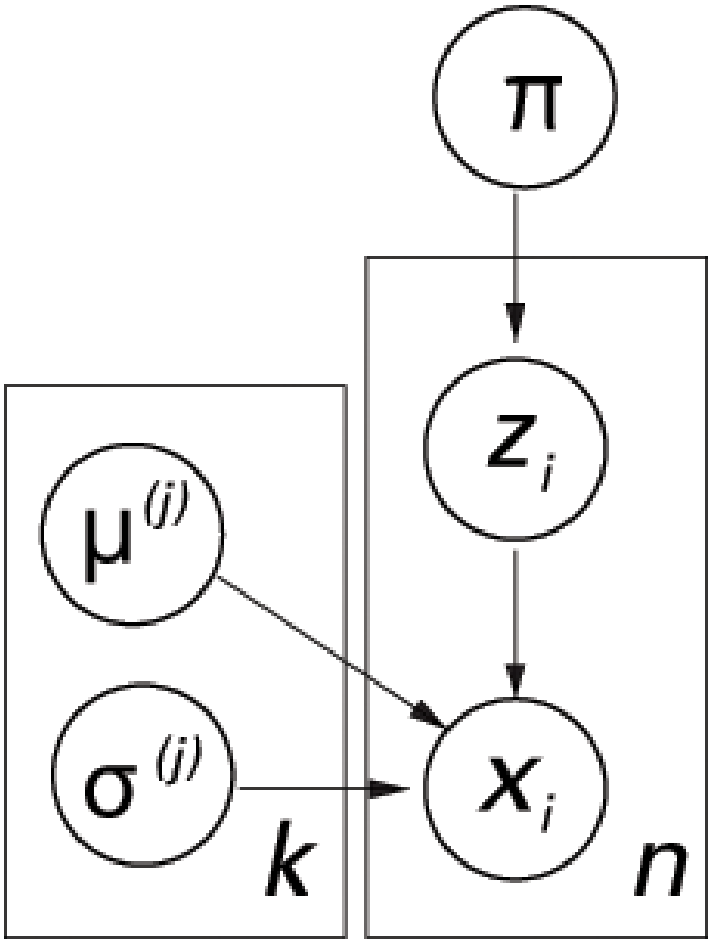


Figure 6.5

Reminder of the graphical model for clustering with a mixture of Gaussians. Here, x_i is the ith data point, z_i is its cluster assignment, k are the parameters of the Discrete distribution on z_i with $P(z_i = k) = (k)$, and (k) and (k) are the parameters of the Gaussian corresponding to the kth cluster. The distribution $p(x_i|z_i = k, ,)$ is Gaussian(,). The boxes around the variables are plate indicating that this structure is replicated across n observations and k clusters. In the example considered in the text, $k = 2$, so i becomes the parameter of a Bernoulli distribution indicating the probability of choosing the first cluster, $z_i = 1$.

Applying the Gibbs sampler in this mixture model is very simple. In each iteration, we cycle through the z_i drawing each from the conditional distribution

$$P(z_i|\mathbf{z}_{-i}, \mathbf{x}, \mu^{(1)}, \mu^{(2)}, \pi) \propto p(x_i|z_i, \mu^{(z_i)})P(z_i|\pi) \tag{6.28}$$

$$= \exp\{-\frac{(x_i - \mu^{(z_i)})^2}{2\sigma^2}\}\pi^{I(z_i=1)}(1-\pi)^{I(z_i=2)}, \tag{6.29}$$

i 1 $i1$ $i+1$ n

where $\mathbf{z}$ indicates (z_1, …, z_{i-1}, z_{i+1}, …, z_n), $I(\cdot)$ is the indicator function taking value 1 when its argument is true and 0 otherwise,

and we exploit the conditional independence assumptions incorporated into this model. We then draw $\mu_1^{(j)}$, $\mu_2^{(j)}$, and π_j from their posterior distributions given these values of **z**. Since conditioning on **z** is like observing it, this is extremely easy: we need only compute the posterior distributions of these parameters, assuming that each observation x_i is assigned to the component indicated in **z**. The posterior distribution on $\mu_1^{(j)}$ is just the posterior distribution given all x_i such that $z_i = j$ (which we showed is Gaussian in our earlier analyses), and the posterior distribution on π_j is Beta($n_j + 1$, $n_{\neg j} + 1$), assuming a uniform prior on π_j. It is easy to generate from both of these distributions. The Gibbs sampler then iterates back and forth between these two steps, sampling assignments, then parameters, then assignments, and so forth. Figure 6.6 shows an example of using this sampler in a 2D Gaussian mixture.

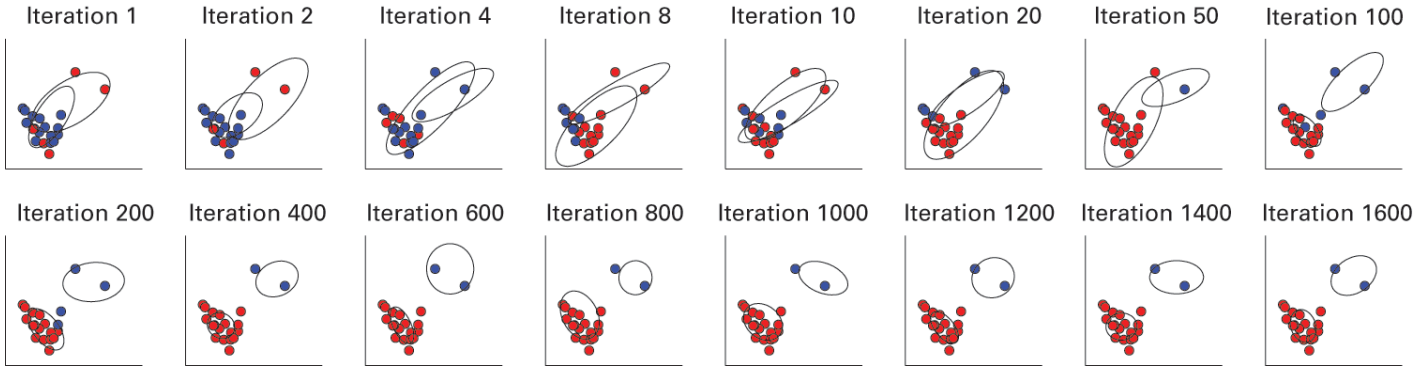


Figure 6.6
Gibbs sampling for a mixture of two Gaussians in two dimensions. The color of the points indicate their assignments, while density ellipses denote the Gaussians. After taking a number of iterations to find a reasonable solution, the algorithm is fairly stable in its assignment of observations to clusters.

In deriving the Gibbs sampler, we need to compute the conditional distribution for each variable conditioned on all others. Graphical models can be extremely useful in this context. Remember that one of the things that graphical models encode is the pattern of dependencies that exists among a set of variables. In particular, when we condition on all other variables, we know that each variable will be independent of any other variable that is not its parent in the graph, its child, or the parent of a common child. This set of variables is known as the *Markov blanket* of the variable, as it shields the variable from other dependencies. Thus, when we compute the conditional distributions used in Gibbs sampling, we need only consider the variables that appear in the Markov blanket of the target variable. You can check that this is the case for the mixture of Gaussians given in this discussion by generating the Markov blanket for each variable using the graphical model shown in figure 6.5.

6.6.5 An Example: Gibbs Sampling for Topic Models

Using the topic models introduced in chapter 5 requires solving a difficult probabilistic inference problem—finding the topics that best account for the content of a set of documents. As a reminder, each word w_i is associated with a topic z_i, each topic is a probability distribution over words expressed as a multinomial distribution with parameters ϕ_i, and the probability distribution over topics within a document is a multinomial distribution with parameters θ. We define a Dirichlet prior (the multivariate generalization of the beta distribution) on both θ and ϕ, with θ generated from a symmetric Dirichlet with parameter α and ϕ generated from a symmetric Dirichlet with parameter β.

We can extract a set of t topics from a collection of documents in a completely unsupervised fashion using Bayesian inference. Since Dirichlet priors are conjugate to multinomial distributions, we can compute the joint distribution $P(\mathbf{w}, \mathbf{z})$ by integrating out θ and ϕ, just as we did in the model selection examples in chapter 3. We can then ask questions about the posterior distribution over **z** given **w**, given by Bayes' rule:

$$P(\mathbf{z}|\mathbf{w}) = \frac{P(\mathbf{w}, \mathbf{z})}{\sum_{\mathbf{z}} P(\mathbf{w}, \mathbf{z})}. \quad (6.30)$$

Since the sum in the denominator is intractable, having t^n terms, we are forced to evaluate this posterior using MCMC. In this case, we use Gibbs sampling to investigate the posterior distribution over assignments of words to topics, **z**.

The Gibbs sampling algorithm consists of choosing an initial assignment of words to topics (e.g., choosing a topic uniformly at random for each word), and then sampling the assignment of each word z_i from the conditional distribution $P(z_i|\mathbf{z}_{-i}, \mathbf{w})$. Each iteration of the algorithm is thus a probabilistic shuing of the assignments of words to topics. This procedure is illustrated in figure 6.7, which shows the results of applying the algorithm (using just two topics) to a small portion of the TASA corpus. This portion features 30 documents that use the word MONEY, 30 documents that use the word OIL, and 30 documents that use the word RIVER. The vocabulary is restricted to 18 words, and the entries indicate the frequency with which the 731 tokens (i.e., instances) of those words appeared in the 90 documents. Each word token in the corpus, w_i, has a topic assignment, z_i, at each iteration of the sampling procedure. In the figure, we focus on the tokens of three words: MONEY, BANK, and STREAM. Each word token is initially assigned a topic at random, and each iteration of MCMC results in a new set of assignments of tokens to topics. After a few iterations, the topic assignments begin to reflect the different usage patterns of MONEY and STREAM, with tokens of these words ending up in different topics, and the multiple senses of BANK.

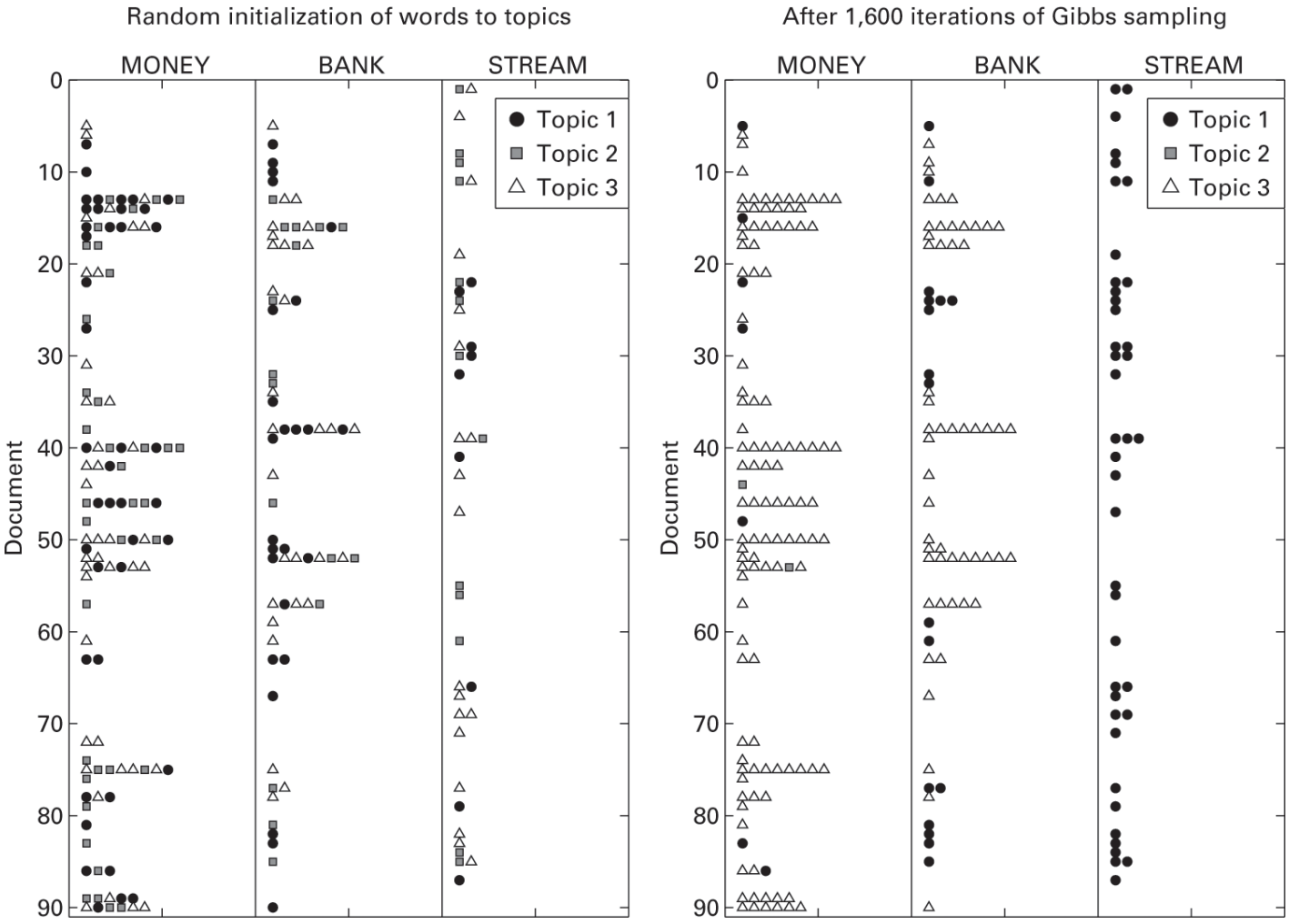


Figure 6.7

Illustration of the Gibbs sampling algorithm for learning topics. Each word token w appearing in the corpus has a topic assignment, z. The figure shows the assignments of all tokens of three types— money, bank, and stream—before and after running the algorithm. Each marker corresponds to a single token appearing in a particular document, and shape and color indicates assignment: topic 1 is a black circle, topic 2 is a gray square, and topic 3 is a white triangle. Before running the algorithm, assignments are relatively random, as shown in the left panel. After running the algorithm, tokens of money are almost exclusively assigned to topic 3, tokens of stream are almost exclusively assigned to topic 1, and tokens of bank are assigned to whichever of topic 1 and topic 3 seems to dominate a given document. The algorithm consists of iteratively choosing an assignment for each token, using a probability distribution over tokens that guarantees convergence to the posterior distribution over assignments. Figure reproduced with permission from Griffiths, Kemp, and Tenenbaum (2008c).

The details behind this particular Gibbs sampling algorithm are given in Griffiths and Steyvers (2004), where the algorithm is

used to analyze the topics that appear in a large database of scientific documents. The conditional distribution for z that is used in the algorithm can be derived using an argument similar to our derivation of the posterior predictive distribution in coin-flipping, giving

$$P(z_i|\mathbf{z}_{-i},\mathbf{w}) \propto \frac{n_{-i,z_i}^{(w_i)}+\beta}{n_{-i,z_i}^{(\cdot)}+v\beta}\,\frac{n_{-i,z_i}^{(d_i)}+\alpha}{n_{-i,\cdot}^{(d_i)}+t\alpha}, \tag{6.31}$$

where $\mathbf{z}$ is the assignment of all z such that k i, and $n_{-i,z_i}^{(w_i)}$ is the number of words assigned to topic z that are the same as w;

$n^{(\cdot)}_{-i,z_i}$ is the total number of words assigned to topic z; $n^{(d_i)}_{-i,z_i}$ is the number of words from document d_i assigned to topic z; v is the number of words in the vocabulary of the model; and $n^{(d_i)}_{-i,\cdot}$ is the total number of words in document d_i; all of which do not count the assignment of the current word w_i. The two terms in this expression have intuitive interpretations, being the posterior predictive distributions on words within a topic and topics within a document given the current assignments $\mathbf{z}$ respectively. The result of the MCMC algorithm is a set of samples from $P(\mathbf{z}|\mathbf{w})$, reflecting the posterior distribution over topic assignments given a collection of documents. A single sample can be used to evaluate the topics that appear in a corpus or to reveal the assignments of words to topics, as shown in figure 6.7. We can also compute quantities such as the strength of association between words using the conditional probability

$$P(w_2|w_1) = \sum_{j=1}^{k} P(w_2|z=j)P(z=j|w_1) \tag{6.32}$$

[2]

by averaging over many samples.

While other inference algorithms exist that can be used with this generative model (e.g., Blei et al., 2003; Minka & Lafferty, 2002), the Gibbs sampler is an extremely simple (and reasonably efficient) way to investigate the consequences of using topics to represent semantic relationships between words. Griffiths and Steyvers (2002, 2003) suggested that topic models might provide an alternative to traditional approaches to semantic representation, and they showed that they can provide better predictions of human word association data than latent semantic analysis (LSA) (Landauer & Dumais, 1997). Figure 6.8 is a contemporaneous record of an early presentation of this work. Topic models can also be applied to a range of other tasks that draw on semantic association, such as semantic priming and sentence comprehension (Griffiths et al., 2007).