

10.4 In this problem we explore the use of relative entropy or the Kullback–Leibler divergence to derive a supervised learning algorithm for multilayer perceptrons (Hopfield, 1987b; Baum and Wilczek, 1988). To be specific, consider a multilayer perceptron consisting of an input layer, a hidden layer, and an output layer. Given a case or example α presented to the input, the output of neuron k in the output layer is assigned a probabilistic interpretation:

$$y_{k|\alpha} = p_{k|\alpha}$$

Correspondingly, let $q_{k|\alpha}$ denote the actual (true) value of the conditional probability that the proposition k is true, given the input case α . The relative entropy for the multilayer perceptron is defined by

$$D_{p\|q} = \sum_{\alpha} p_{\alpha} \sum_k \left(q_{k|\alpha} \log \left(\frac{q_{k|\alpha}}{p_{k|\alpha}} \right) + (1 - q_{k|\alpha}) \log \left(\frac{1 - q_{k|\alpha}}{1 - p_{k|\alpha}} \right) \right)$$

where p_{α} is the *a priori* probability of occurrence of case α .

Using $D_{p\|q}$ as the cost function to be optimized, derive a learning rule for training the multilayer perceptron.

Infomax Principle

10.5 Consider two channels whose outputs are represented by the random variables X and Y . The requirement is to maximize the mutual information between X and Y . Show that this requirement is achieved by satisfying two conditions:

- (a) The probability of occurrence of X or that of Y is 0.5.
- (b) The joint probability distribution of X and Y is concentrated in a small region of the probability space.

10.6 Consider the noise model of Fig. P10.6, which shows m source nodes in the input layer of a two-neuron network. Both neurons are linear. The inputs are denoted by $X_1, X_2, \dots, X_m$, and the resulting outputs are denoted by Y_1 and Y_2 . You may make the following assumptions:

- The additive noise components N_1 and N_2 at the outputs of the network are Gaussian distributed, with zero mean and common variance σ_N^2 . They are also uncorrelated with each other.
- Each noise source is uncorrelated with the input signals.
- The output signals Y_1 and Y_2 are both Gaussian random variables with zero mean.

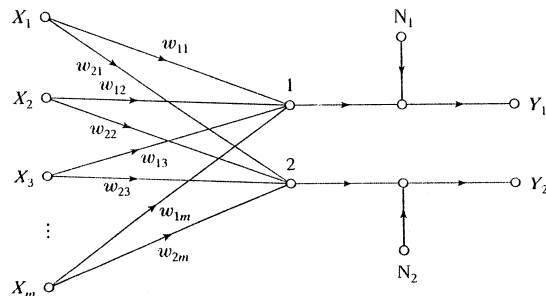


FIGURE P10.6

- (a) Determine the mutual information $I(\mathbf{Y}; \mathbf{X})$ between the output vector $\mathbf{Y} = [Y_1, Y_2]^T$ and the input vector $\mathbf{X} = [X_1, X_2, \dots, X_m]^T$.
- (b) Using the result derived in part (a), investigate the redundancy/diversity tradeoff under the following conditions (Linsker, 1988a):
 - (i) Large noise variance, represented by σ_N^2 being large compared to the variances of Y_1 and Y_2 .
 - (ii) Low noise variance, represented by σ_N^2 being small compared to the variances of Y_1 and Y_2 .

10.7 In the variant of the Infomax principle described in Section 10.9, due to Becker and Hinton (1992), the objective is to maximize the mutual information $I(Y_a; Y_b)$ between the outputs Y_a and Y_b of a noisy neural system due to the input vectors $\mathbf{X}_a$ and $\mathbf{X}_b$. In another approach discussed in Becker and Hinton (1992), a different objective is set: maximize the mutual information $I\left(\frac{Y_a + Y_b}{2}, S\right)$ between the average of the outputs Y_a and Y_b and the underlying signal component S common to these two outputs.

Using the noisy model described in Eqs. (10.59) and (10.60), do the following:

- (a) Show that

$$I\left(\frac{Y_a + Y_b}{2}, S\right) = \frac{\text{var}[Y_a + Y_b]}{\text{var}[N_a + N_b]}$$

where N_a and N_b are the noise components in Y_a and Y_b , respectively.

- (b) Demonstrate the interpretation of this mutual information as a signal-plus-noise to noise ratio.

Independent Components Analysis

10.8 Make a detailed comparison between principal components analysis (discussed in Chapter 8) and independent components analysis (discussed in this chapter).

10.9 Independent components analysis may be used as a preprocessing step for approximate data analysis before detection and classification (Comon, 1994). Discuss the property of independent components analysis that can be exploited for this application.

10.10 *Darmois' theorem* states that the sum of independent variables can be Gaussian distributed only if these variables are themselves Gaussian distributed (Darmois, 1953). Use independent components analysis to prove this theorem.

10.11 In practice, an algorithmic implementation of independent components analysis can only go for "as statistically independent as possible." Contrast the solution to the blind source separation problem using such an algorithm to the solution obtained using a decorrelation method. Assume that the covariance matrix of the observation vector is nonsingular.

10.12 Referring to the scheme described in Fig. 10.9, show that minimizing the mutual information between any two components of the demixer output $\mathbf{Y}$ is equivalent to minimizing the Kullback–Leibler divergence between the parameterized probability density function $f_Y(\mathbf{y}, \mathbf{W})$ and the corresponding factorial distribution $\tilde{f}_Y(\mathbf{y}, \mathbf{W})$.

10.13 The adaptive algorithm for blind source separation described in Eq. (10.104) has two important properties: (1) the equivariant property, and (2) the property that the weight matrix $\mathbf{W}$ is maintained nonsingular. Property (1) is discussed in some detail in the latter part of Section 10.11. In this problem we consider the second property.

Provided that the initial value $\mathbf{W}(0)$ used in starting the algorithm of Eq. (10.104) satisfies the condition $|\det(\mathbf{W}(0))| \neq 0$, show that

$$|\det(\mathbf{W}(n))| \neq 0 \quad \text{for all } n$$

This is the necessary and sufficient condition for ensuring that $\mathbf{W}(n)$ is nonsingular for all n .

10.14 In this problem we formulate the batch version of the blind source-separation algorithm described in Eq. (10.104). Specifically we write

$$\Delta \mathbf{W} = \eta \left(\mathbf{I} - \frac{1}{N} \Phi(\mathbf{Y}) \mathbf{Y}^T \right) \mathbf{W}$$

where

$$\mathbf{Y} = \begin{bmatrix} y_1(1) & y_1(2) & \cdots & y_1(N) \\ y_2(1) & y_2(2) & \cdots & y_2(N) \\ \vdots & \vdots & \ddots & \vdots \\ y_m(1) & y_m(2) & \cdots & y_m(N) \end{bmatrix}$$

and

$$\Phi(\mathbf{Y}) = \begin{bmatrix} \varphi(y_1(1)) & \varphi(y_1(2)) & \cdots & \varphi(y_1(N)) \\ \varphi(y_2(1)) & \varphi(y_2(2)) & \cdots & \varphi(y_2(N)) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi(y_m(1)) & \varphi(y_m(2)) & \cdots & \varphi(y_m(N)) \end{bmatrix}$$

where N is the number of available data points. Justify the formulation of the adjustment $\Delta \mathbf{W}$ applied to the weight matrix $\mathbf{W}$ as described.

Maximum Entropy Method

10.15 Consider Fig. 10.15 in which we have

$$\mathbf{Y} = \mathbf{W} \mathbf{X}$$

where

$$\mathbf{Y} = [Y_1, Y_2, \dots, Y_m]^T$$

$$\mathbf{X} = [X_1, X_2, \dots, X_m]^T$$

and $\mathbf{W}$ is an m -by- m weight matrix. Let

$$\mathbf{Z} = [Z_1, Z_2, \dots, Z_m]^T$$

where

$$Z_k = \varphi(Y_k), \quad k = 1, 2, \dots, m$$

(a) Show that the joint entropy of $\mathbf{Z}$ is related to the Kullback–Leibler divergence $D_{f\|g}$ as follows:

$$h(\mathbf{Z}) = -D_{f\|g} + D_{f\|q}$$

where $D_{f\|q}$ is the Kullback–Leibler divergence between (a) the probability density function of the statistically independent (i.e., factorized) version of the output vector $\mathbf{Y}$ and (b) a “probability density function” defined by $\prod_{i=1}^m q(y_i)$.

(b) How is the formula for $h(\mathbf{Z})$ modified for the case when $q(Y_i)$ is equal to the probability density function of the original source output U_i for all i ?

10.16 (a) Starting with Eq. (10.134), derive the result given in Eq. (10.135).

(b) For the logistic function described in Eq. (10.136), show that the use of Eq. (10.135) yields the formula given in Eq. (10.137).

Stochastic Machines and Their Approximates Rooted in Statistical Mechanics

11.1 INTRODUCTION

For our last class of unsupervised (self-organized) learning systems, we turn to statistical mechanics as the source of ideas. The subject of *statistical mechanics* encompasses the formal study of macroscopic equilibrium properties of large systems of elements that are subject to the microscopic laws of mechanics. The main aim of statistical mechanics is to derive the thermodynamic properties of macroscopic bodies starting from the motion of microscopic elements such as atoms and electrons (Landau and Lifshitz, 1980; Parisi, 1988). The number of degrees of freedom encountered here is enormous, making the use of probabilistic methods mandatory. As with Shannon’s information theory, the concept of entropy plays a vital role in the study of statistical mechanics: The more ordered the system, or the more concentrated the underlying probability distribution, the smaller the entropy. By the same token, we can say that the more disordered the system, or the more uniform the underlying probability distribution, the larger the entropy. In 1957, Jaynes showed that entropy can be used not only as the starting point of formulating statistical inference as described in the previous chapter, but also for generating the Gibbs distribution that is basic to the study of statistical mechanics.

Interest in the use of statistical mechanics as a basis for the study of neural networks goes back to the early works of Cragg and Temperley (1954) and Cowan (1968). The *Boltzmann machine* (Hinton and Sejnowski, 1983, 1986; Ackley et al., 1985) is perhaps the first multilayer learning machine inspired by statistical mechanics. The machine is named in recognition of the formal equivalence between Boltzmann’s original work on statistical thermodynamics and the network’s own dynamic behavior. Basically, the Boltzmann machine is a device for modeling the underlying probability distribution of a given data set, from which conditional distributions for use in tasks such as pattern completion and pattern classification can be derived. Unfortunately,

the learning process in the Boltzmann machine is painfully slow. This shortcoming has motivated modifications to the Boltzmann machine and inspired the formulation of new stochastic machines. These issues constitute the bulk of the material presented in this chapter.

Organization of the Chapter

The chapter is organized in three parts. The first part consists of Sections 11.2 to 11.6. Section 11.2 presents a brief review of statistical mechanics. This is followed by Section 11.3, a brief review of a special kind of stochastic process known as the Markov chain that is commonly encountered in the study of statistical mechanics. Sections 11.4, 11.5, and 11.6 describe three stochastic simulation techniques: the Metropolis algorithm, simulated annealing, and Gibbs sampling.

The second part of the chapter, made up of Sections 11.7 to 11.9, discusses three types of stochastic machines. Section 11.7 describes the Boltzmann machine. Section 11.8 describes sigmoid belief networks. Section 11.9 describes another novel stochastic machine known as the Helmholtz machine.

The last part of the chapter, made up of Sections 11.10 to 11.13, discusses approximations to stochastic machines. The approximations are based on the idea of mean-field theory in statistical mechanics. Section 11.10 discusses mean-field theory in general terms. Section 11.11 discusses a naive mean-field theory of the Boltzmann machine, followed by a more principled approach to the mean-field theory of sigmoid belief networks in Section 11.12. Section 11.13 describes deterministic annealing, which is an approximation to simulated annealing.

The chapter concludes with some final remarks in Section 11.14.

11.2 STATISTICAL MECHANICS

Consider a physical system with many degrees of freedom, that can reside in any one of a large number of possible states. Let p_i denote the probability of occurrence of state i , for example, with the following properties:

$$p_i \geq 0 \quad \text{for all } i \quad (11.1)$$

and

$$\sum_i p_i = 1 \quad (11.2)$$

Let E_i denote the *energy* of the system when it is in state i . A fundamental result from statistical mechanics tells us that when the system is in thermal equilibrium with its surrounding environment, state i occurs with a probability defined by

$$p_i = \frac{1}{Z} \exp\left(-\frac{E_i}{k_B T}\right) \quad (11.3)$$

where T is the *absolute temperature* in kelvins, k_B is *Boltzmann's constant*, and Z is a constant that is independent of all states. One degree kelvin corresponds to -273° on the Celsius scale, and $k_B = 1.38 \times 10^{-23}$ joules/kelvin.

Equation (11.2) defines the condition for the normalization of probabilities. Imposing this condition on Eq. (11.3), we get

$$Z = \sum_i \exp\left(-\frac{E_i}{k_B T}\right) \quad (11.4)$$

The normalizing quantity Z is called the *sum over states* or the *partition function*. (The symbol Z is commonly used because the German name for this term is *Zustadsumme*. The probability distribution of Eq. (11.3) is called the *canonical distribution* or *Gibbs distribution*;¹ the exponential factor $\exp(-E_i/k_B T)$ is called the *Boltzmann factor*.)

The following two points are noteworthy from the Gibbs distribution:

1. States of low energy have a higher probability of occurrence than states of high energy.
2. As the temperature T is reduced, the probability is concentrated on a smaller subset of low-energy states.

In the context of neural networks, which is our primary concern, the parameter T may be viewed as a *pseudotemperature* that controls thermal fluctuations representing the effect of "synaptic noise" in a neuron. Its precise scale is therefore irrelevant. Accordingly, we may choose to measure it by setting the constant k_B equal to unity, and thereby redefine the probability p_i and partition function Z as follows, respectively:

$$p_i = \frac{1}{Z} \exp\left(-\frac{E_i}{T}\right) \quad (11.5)$$

and

$$Z = \sum_i \exp\left(-\frac{E_i}{T}\right) \quad (11.6)$$

Henceforth our treatment of statistical mechanics is based on these two definitions, where T is referred to simply as the *temperature of the system*. From Eq. (11.5) we note that $-\log p_i$ may be viewed as a form of "energy" measured at unit temperature.

Free Energy and Entropy

The Helmholtz *free energy* of a physical system, denoted by F , is defined in terms of the partition function Z as follows:

$$F = -T \log Z \quad (11.7)$$

The *average energy* of the system is defined by

$$\langle E \rangle = \sum_i p_i E_i \quad (11.8)$$

where $\langle \cdot \rangle$ denotes the ensemble averaging operation. Thus, using Eqs. (11.5) to (11.8), we see that the difference between the average energy and free energy is given by

$$\langle E \rangle - F = -T \sum_i p_i \log p_i \quad (11.9)$$

The quantity on the right-hand side of Eq. (11.9), except for the temperature T , is recognized as the *entropy* of the system, as shown by

$$H = - \sum_i p_i \log p_i \quad (11.10)$$

We may therefore rewrite Eq. (11.9) in the form

$$\langle E \rangle - F = TH$$

or, equivalently,

$$F = \langle E \rangle - TH \quad (11.11)$$

Consider two systems, A and A' , placed in thermal contact with each other. Suppose that system A is small compared to system A' , so that A' acts as a heat reservoir at some constant temperature, T . The total entropy of the two systems tends to increase in accordance with the relation (Reif, 1967)

$$\Delta H + \Delta H' \geq 0$$

where ΔH and $\Delta H'$ denote the entropy changes of systems A and A' , respectively. The implication of this relation, in light of Eq. (11.11), is that the free energy of the system, F , tends to decrease and become a minimum in equilibrium situation. From statistical mechanics we find that the resulting probability distribution is defined by the Gibbs distribution. We thus have an important principle called the *principle of minimal free energy*, which may be stated as follows (Landau and Lifshitz, 1980; Parisi, 1988):

The minimum of the free energy of a stochastic system with respect to variables of the system is achieved at thermal equilibrium, at which point the system is governed by the Gibbs distribution.

Nature likes to find a physical system with minimum free energy.

11.3 MARKOV CHAINS

Consider a system whose evolution is described by a stochastic process $\{X_n, n = 1, 2, \dots\}$, consisting of a family of random variables. The value x_n assumed by the random variable X_n at discrete time n is called the *state* of the system at that time instant. The space of all possible values that the random variables can assume is called the *state space* of the system. If the structure of the stochastic process $\{X_n, n = 1, 2, \dots\}$ is such that the conditional probability distribution of X_{n+1} depends only on the value of X_n and is independent of all previous values, we say that the process is a *Markov chain* (Feller, 1950; Ash, 1965). More precisely, we have

$$P(X_{n+1} = x_{n+1} | X_n = x_n, \dots, X_1 = x_1) = P(X_{n+1} = x_{n+1} | X_n = x_n) \quad (11.12)$$

which is called the *Markov property*. In words, a sequence of random variables $X_1, X_2, \dots, X_n, X_{n+1}$ forms a Markov chain if the probability that the system is in state x_{n+1} at time $n+1$ depends exclusively on the probability that the system is in state x_n at time n .

We may therefore think of the Markov chain as a *generative model*, consisting of a number of states linked together (on a pair-wise basis) by possible transitions.

Each time a particular state is visited, the model outputs the symbol associated with that state.

Transition Probabilities

In a Markov chain, the transition from one state to another is *probabilistic*, but the production of an output symbol is deterministic. Let

$$p_{ij} = P(X_{n+1} = j | X_n = i) \quad (11.13)$$

denote the *transition probability* from state i at time n to state j at time $n+1$. Since the p_{ij} are conditional probabilities, all transition probabilities must satisfy two conditions:

$$p_{ij} \geq 0 \quad \text{for all } (i, j) \quad (11.14)$$

$$\sum_j p_{ij} = 1 \quad \text{for all } i \quad (11.15)$$

We will assume that the transition probabilities are fixed and do not change with time; that is, Eq. (11.13) is satisfied for all time n . In such a case the Markov chain is said to be *homogeneous* in time.

In the case of a system with a finite number of possible states K , for example, the transition probabilities constitute a K -by- K matrix:

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & \dots & p_{1K} \\ p_{21} & p_{22} & \dots & p_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ p_{K1} & p_{K2} & \dots & p_{KK} \end{bmatrix} \quad (11.16)$$

whose individual elements satisfy the conditions described in Eqs. (11.14) and (11.15); the latter condition says that each row of $\mathbf{P}$ must add to one. A matrix of this type is called a *stochastic matrix*. Any stochastic matrix can serve as a matrix of transition probabilities.

The definition of one-step transition probability given in Eq. (11.13) may be generalized to cases where the transition from one state to another takes place in some fixed number of steps. Let $p_{ij}^{(m)}$ denote the *m-step transition probability* from state i to state j :

$$p_{ij}^{(m)} = P(X_{n+m} = x_j | X_n = x_i), \quad m = 1, 2, \dots \quad (11.17)$$

We may view $p_{ij}^{(m)}$ as the sum over all intermediate states k through which the system passes in its transition from state i to state j . Specifically, $p_{ij}^{(m+1)}$ is related to $p_{ij}^{(m)}$ by the recursive relation:

$$p_{ij}^{(m+1)} = \sum_k p_{ik}^{(m)} p_{kj}, \quad m = 1, 2, \dots \quad (11.18)$$

with

$$p_{ik}^{(1)} = p_{ik}$$

Equation (11.18) may be generalized as follows:

$$p_{ij}^{(m+n)} = \sum_k p_{ik}^{(m)} p_{kj}^{(n)}, \quad (m, n) = 1, 2, \dots \quad (11.19)$$

which is a special case of the *Chapman-Kolmogorov identity* (Feller, 1950).

When a state of the chain can only reoccur at time intervals that are multiples of d , where d is the largest such integer, we say that the state has *period* d . A Markov chain is called *aperiodic* if all of its states have period 1.

Recurrent Properties

Suppose a Markov chain starts in state i . The state i is said to be a *recurrent* state if the Markov chain returns to state i with probability 1; that is,

$$f_i = P(\text{ever returning to state } i) = 1$$

If the probability f_i is less than 1, state i is said to be a *transient* state (Leon-Garcia, 1994).

If the Markov chain starts in a recurrent state, that state reoccurs an infinite number of times. If it starts in a transient state, that state reoccurs only a finite number of times, which may be explained as follows. We may view the reoccurrence of state i as a Bernoulli trial with a probability of success equal to f_i . The number of returns is thus a geometric random variable with a mean of $(1 - f_i)^{-1}$. If $f_i < 1$, it follows that the number of an infinite number of successes is zero. Therefore, a transient state does not reoccur after some finite number of returns (Leon-Garcia, 1994).

If a Markov chain has some transient states and some recurrent states, then the process will eventually move only among the recurrent states.

Irreducible Markov Chains

The state j of a Markov chain is said to be *accessible* from state i if there is a finite sequence of transitions from i to j with positive probability. If the states i and j are accessible to each other, the states i and j of the Markov chain are said to *communicate* with each other. This communication is described by writing $i \leftrightarrow j$. Clearly, if state i communicates with state j and state j communicates with state k , that is, $i \leftrightarrow j$ and $j \leftrightarrow k$, then state i communicates with state k , that is, $i \leftrightarrow k$.

If two states of a Markov chain communicate with each other, they are said to belong to the same *class*. In general, the states of a Markov chain consist of one or more disjoint classes. If, however, all the states consist of a single class, the Markov chain is said to be *indecomposable* or *irreducible*. In other words, by starting at any state of an irreducible Markov chain we can reach any other state with positive probability. Reducible chains are of little practical interest in most areas of application. Accordingly, we restrict our attention to irreducible chains.

Consider an irreducible Markov chain that starts in a recurrent state i at time $n = 0$. Let $T_i(k)$ denote the time that elapses between the $(k-1)$ th and k th returns to

state i . The *mean recurrence time* of state i is defined as the expectation of $T_i(k)$ over the returns k . The *steady-state probability* of state i , denoted by π_i , is equal to the reciprocal of the mean recurrence time $E[T_i(k)]$, as shown by

$$\pi_i = \frac{1}{E[T_i(k)]}$$

If $E[T_i(k)] < \infty$, that is, $\pi_i > 0$, the state i is said to be a *positive recurrent* (*persistent*) state. If $E[T_i(k)] = \infty$, that is, $\pi_i = 0$, the state i is said to be a *null recurrent* (*transient*) state. The implication of $\pi_i = 0$ is that the Markov chain will eventually reach a point where a return to state i is impossible. Positive recurrence and null recurrence are *different* class properties, which means that a Markov chain with positive recurrent and null recurrent states is reducible.

Ergodic Markov Chains

In principle, *ergodicity* means that we may substitute time averages for ensemble averages. In the context of a Markov chain, ergodicity means that the long-term proportion of time spent by the chain in state i corresponds to the steady-state probability π_i , which may be justified as follows. The *proportion of time spent in state i after k returns*, denoted by $v_i(k)$, is defined by

$$v_i(k) = \frac{k}{\sum_{l=1}^k T_i(l)}$$

The return times $T_i(l)$ form a sequence of independently and identically distributed random variables since, by definition, each return time is statistically independent of all previous return times. Moreover, in the case of a recurrent state i the chain returns to state i an infinite number of times. Hence, as the number of returns k approaches infinity, the *law of large numbers* states that the proportion of time spent in state i approaches the steady-state probability, as shown by

$$\lim_{k \rightarrow \infty} v_i(k) = \pi_i \quad \text{for } i = 1, 2, \dots, K \quad (11.20)$$

A sufficient, but not necessary condition, for a Markov chain to be *ergodic* is for it to be both irreducible and aperiodic.

Convergence to Stationary Distributions

Consider an ergodic Markov chain characterized by a stochastic matrix $\mathbf{P}$. Let the row vector $\pi^{(n-1)}$ denote the *state distribution vector* of the chain at time $n-1$; the j th element of $\pi^{(n-1)}$ is the probability that the chain is in state x_j at time $n-1$. The state distribution vector at time n is defined by

$$\pi^{(n)} = \pi^{(n-1)} \mathbf{P} \quad (11.21)$$

By iteration of Eq. (11.21), we obtain

$$\pi^{(n)} = \pi^{(n-1)}\mathbf{P} = \pi^{(n-2)}\mathbf{P}^2 = \pi^{(n-3)}\mathbf{P}^3 = \dots$$

and finally we may write

$$\pi^{(n)} = \pi^{(0)}\mathbf{P}^n \quad (11.22)$$

where $\pi^{(0)}$ is the *initial value* of the state distribution vector. In words, the state distribution vector of the Markov chain at time n is the product of the initial state distribution vector $\pi^{(0)}$ and the n th power of the stochastic matrix $\mathbf{P}$.

Let $p_{ij}^{(n)}$ denote the ij -th element of $\mathbf{P}^n$. Suppose that as time n approaches infinity, $p_{ij}^{(n)}$ tends to π_j independent of i , where π_j is the steady-state probability of state j . Correspondingly, for large n , the matrix $\mathbf{P}^n$ approaches the limiting form of a square matrix with identical rows as shown by

$$\lim_{n \rightarrow \infty} \mathbf{P}^n = \begin{bmatrix} \pi_1 & \pi_2 & \dots & \pi_K \\ \pi_1 & \pi_2 & \dots & \pi_K \\ \vdots & \vdots & & \vdots \\ \pi_1 & \pi_2 & \dots & \pi_K \end{bmatrix} \quad (11.23)$$

$$= \begin{bmatrix} \pi \\ \pi \\ \vdots \\ \pi \end{bmatrix}$$

where π is a row vector consisting of $\pi_1, \pi_2, \dots, \pi_K$. We then find from Eq. (11.22) that (after rearranging terms)

$$\left[\sum_{j=1}^K \pi_j^{(0)} - 1 \right] \pi = 0$$

Since, by definition $\sum_{j=1}^K \pi_j^{(0)} = 1$, this condition is satisfied by the vector π independent of the initial distribution.

We may now state the *ergodicity theorem* for Markov chains as follows (Feller, 1950; Ash, 1965):

Let an ergodic Markov chain with states $x_1, x_2, \dots, x_K$ and stochastic matrix $\mathbf{P} = \{p_{ij}\}$ be irreducible. The chain then has a unique stationary distribution to which it converges from any initial state; that is, there is a unique set of numbers $\{\pi_j\}_{j=1}^K$ such that

$$1. \lim_{n \rightarrow \infty} p_{ij}^{(n)} = \pi_j \text{ for all } i \quad (11.24)$$

$$2. \pi_j > 0 \text{ for all } j \quad (11.25)$$

$$3. \sum_{j=1}^K \pi_j = 1 \quad (11.26)$$

$$4. \pi_j = \sum_{i=1}^K \pi_i p_{ij} \text{ for } j = 1, 2, \dots, K \quad (11.27)$$

Conversely, suppose that the Markov chain is irreducible and aperiodic, and that there exist numbers $\{\pi_j\}_{j=1}^K$ satisfying Eqs. (11.25) through (11.27). Then the chain is ergodic, the π_j are given by Eq. (11.24), and the mean recurrence time of state j is $1/\pi_j$.

The probability distribution $\{\pi_j\}_{j=1}^K$ is called an *invariant* or *stationary distribution*. It is so called because it persists forever once it is established. In light of the ergodicity theorem, we may thus say the following:

- Starting from an arbitrary initial distribution, the transition probabilities of a Markov chain will converge to a stationary distribution provided that such a distribution exists.
- The stationary distribution of the Markov chain is completely independent of the initial distribution if the chain is ergodic.

Example 11.1

Consider a Markov chain whose *state-transition diagram* is depicted in Fig. 11.1. The chain has two states x_1 and x_2 . The stochastic matrix of the chain is

$$\mathbf{P} = \begin{bmatrix} \frac{1}{4} & \frac{3}{4} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

which satisfies the conditions of Eqs. (11.14) and (11.15).

Suppose the initial condition is

$$\pi^{(0)} = \begin{bmatrix} \frac{1}{6} & \frac{5}{6} \end{bmatrix}$$

From Eq. (11.21) we find that the state distribution vector at time $n = 1$ is

$$\begin{aligned} \pi^{(1)} &= \pi^{(0)}\mathbf{P} \\ &= \begin{bmatrix} \frac{1}{6} & \frac{5}{6} \end{bmatrix} \begin{bmatrix} \frac{1}{4} & \frac{3}{4} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \\ &= \begin{bmatrix} \frac{11}{24} & \frac{13}{24} \end{bmatrix} \end{aligned}$$

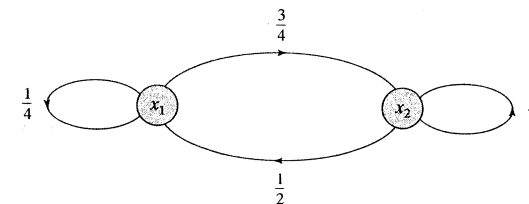


FIGURE 11.1 State-transition diagram of Markov chain for Example 11.1.

Raising the stochastic matrix $\mathbf{P}$ to power $n = 2, 3, 4$, we have

$$\mathbf{P}^2 = \begin{bmatrix} 0.4375 & 0.5625 \\ 0.3750 & 0.6250 \end{bmatrix}$$

$$\mathbf{P}^3 = \begin{bmatrix} 0.4001 & 0.5999 \\ 0.3999 & 0.6001 \end{bmatrix}$$

$$\mathbf{P}^4 = \begin{bmatrix} 0.4000 & 0.6000 \\ 0.4000 & 0.6000 \end{bmatrix}$$

Thus, $\pi_1 = 0.4000$ and $\pi_2 = 0.6000$. In this example, convergence to the stationary distribution is accomplished essentially in $n = 4$ iterations. With both π_1 and π_2 being greater than zero, both states are positive recurrent, and the chain is therefore irreducible. Note also that the chain is aperiodic since the greatest common divisor of all integers $n \geq 1$, such that $(\mathbf{P}^n)_{ij} > 0$, is equal to 1. We therefore conclude that the Markov chain of Fig. 11.1 is ergodic.

Example 11.2

Consider a Markov chain with a stochastic matrix some of whose elements are zero:

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 1 \\ \frac{1}{3} & \frac{1}{6} & \frac{1}{2} \\ \frac{3}{4} & \frac{1}{4} & 0 \end{bmatrix}$$

The state transition diagram of the chain is depicted in Fig. 11.2.

By applying Eq. (11.27) we obtain the following set of simultaneous equations:

$$\pi_1 = \frac{1}{3}\pi_2 + \frac{3}{4}\pi_3$$

$$\pi_2 = \frac{1}{6}\pi_2 + \frac{1}{4}\pi_3$$

$$\pi_3 = \pi_1 + \frac{1}{2}\pi_2$$

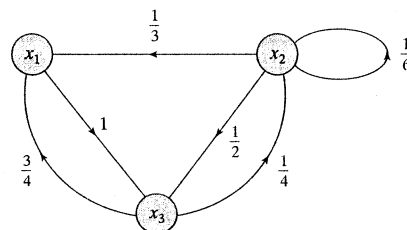


FIGURE 11.2 State-transition diagram of Markov chain for Example 11.2.

By solving these equations for π_1 , π_2 , and π_3 we get

$$\pi_1 = 0.3953$$

$$\pi_2 = 0.1395$$

$$\pi_3 = 0.4652$$

The given Markov chain is ergodic with its stationary distribution defined by π_1 , π_2 and π_3 .

Classification of States

On the basis of the material presented here, we may develop a summary of the classes to which a state can belong as shown in Fig. 11.3 (Feller, 1950; Leon-Garcia, 1994). This figure also includes the associated long-term behavior of the state.

Principle of Detailed Balance

Equations (11.25) and (11.26) merely emphasize the fact that the numbers π_j are probabilities. Equation (11.27) is the critical one because it also has to be satisfied for the Markov chain to be irreducible and therefore, for a stationary distribution to exist. This latter equation is a restatement of the principle of detailed balance that arises in first-order reaction kinetics. The *principle of detailed balance* states that at thermal equilibrium, the rate of occurrence of any transition equals the corresponding rate of occurrence of the inverse transition, as shown by (Reif, 1965):

$$\pi_i p_{ij} = \pi_j p_{ji} \quad (11.28)$$

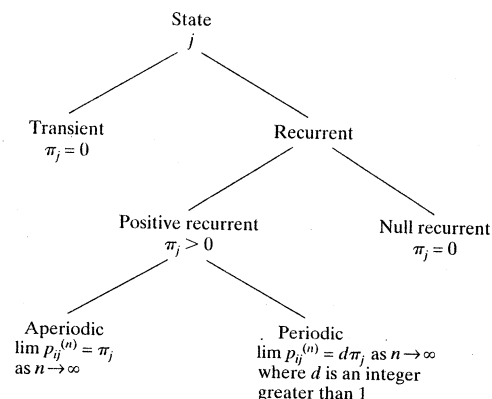


FIGURE 11.3 Classification of the states of a Markov chain and their associated long-term behavior.

To derive the relation of Eq. (11.27), we may manipulate the summation on the right-hand side of this equation as follows:

$$\begin{aligned}\sum_{i=1}^K \pi_i p_{ij} &= \sum_{i=1}^K \left(\frac{\pi_i}{\pi_j} p_{ij} \right) \pi_j \\ &= \sum_{i=1}^K (p_{ji}) \pi_j \\ &= \pi_j\end{aligned}$$

In the second line of this expression we made use of the principle of detailed balance, and in the last line we made use of the fact that the transition probabilities of a Markov chain satisfy the condition (see Eq. (11.15) with the roles of i and j interchanged):

$$\sum_{i=1}^K p_{ji} = 1 \quad \text{for all } j$$

Note that the principle of detailed balance implies that the distribution $\{\pi_j\}$ is a stationary distribution.

11.4 METROPOLIS ALGORITHM

Now that we understand the composition of a Markov chain, we will use it to formulate a stochastic algorithm for simulating the evolution of a physical system to thermal equilibrium. The algorithm is called the *Metropolis algorithm* (Metropolis et al., 1953). It is a modified Monte Carlo method, introduced in the early days of scientific computation for the stochastic simulation of a collection of atoms in equilibrium at a given temperature.

Suppose that the random variable X_n representing an arbitrary Markov chain is in state x_i at time n . We randomly generate a new state x_j , representing a realization of another random variable Y_n . It is assumed that the generation of this new state satisfies the symmetry condition:

$$P(Y_n = x_j | X_n = x_i) = P(Y_n = x_i | X_n = x_j)$$

Let ΔE denote the energy difference resulting from the transition of the system from state $X_n = x_i$ to state $Y_n = x_j$. If the energy difference ΔE is negative, the transition leads to a state with lower energy and the transition is accepted. The new state is then accepted as the starting point for the next step of the algorithm, that is, we put $X_{n+1} = Y_n$. If, on the other hand, the energy difference ΔE is positive, the algorithm proceeds in a probabilistic manner at that point. First, we select a random number ξ uniformly distributed in the range $[0, 1]$. If $\xi < \exp(-\Delta E/T)$, where T is the operating temperature, the transition is accepted and we put $X_{n+1} = Y_n$. Otherwise, the transition is rejected and we put $X_{n+1} = X_n$; that is, the old configuration is reused for the next step of the algorithm.

Choice of Transition Probabilities

Let the arbitrary Markov chain have *a priori* transition probabilities denoted by τ_{ij} , which satisfy three conditions:

1. *Nonnegativity*:

$$\tau_{ij} \geq 0 \quad \text{for all } (i, j)$$

2. *Normalization*:

$$\sum_j \tau_{ij} = 1 \quad \text{for all } i$$

3. *Symmetry*:

$$\tau_{ij} = \tau_{ji} \quad \text{for all } (i, j)$$

Let π_i denote the steady state probability that the Markov chain is in state x_i , $i = 1, 2, \dots, K$. We may then use the symmetric τ_{ij} s and the probability distribution ratio π_j/π_i , to be defined, to formulate the desired set of transition probabilities as (Beckerman, 1997):

$$p_{ij} = \begin{cases} \tau_{ij} \left(\frac{\pi_j}{\pi_i} \right) & \text{for } \frac{\pi_j}{\pi_i} < 1 \\ \tau_{ij} & \text{for } \frac{\pi_j}{\pi_i} \geq 1 \end{cases} \quad (11.29)$$

To ensure that the transition probabilities are normalized to unity, we introduce this additional definition for the probability of no transition:

$$\begin{aligned}p_{ii} &= \tau_{ii} + \sum_{j \neq i} \tau_{ij} \left(1 - \frac{\pi_j}{\pi_i} \right) \\ &= 1 - \sum_{j \neq i} \alpha_{ij} \tau_{ij}\end{aligned} \quad (11.30)$$

where α_{ij} is the moving probability defined by

$$\alpha_{ij} = \min \left(1, \frac{\pi_j}{\pi_i} \right) \quad (11.31)$$

The only outstanding requirement is how to choose the ratio π_j/π_i . To cater to this requirement, we choose the probability distribution that we want the Markov chain to converge to be a Gibbs distribution, as shown by

$$\pi_j = \frac{1}{Z} \exp \left(-\frac{E_j}{T} \right)$$

in which case the probability distribution ratio π_j/π_i takes the simple form

$$\frac{\pi_j}{\pi_i} = \exp \left(-\frac{\Delta E}{T} \right) \quad (11.32)$$

where

$$\Delta E = E_j - E_i \quad (11.33)$$

By using the ratio of probability distributions we have eliminated dependence on the partition function Z .

By construction, the transition probabilities are all nonnegative and normalized to unity, as required by Eqs. (11.14) and (11.15). Moreover, they satisfy the principle of detailed balance defined by Eq. (11.28). This principle is a sufficient condition for thermal equilibrium. To demonstrate that the principle of detailed balance is satisfied, we offer the following considerations:

Case 1: $\Delta E < 0$. Suppose that in going from state x_i to state x_j , the energy change ΔE is negative. From Eq. (11.32) we find that $(\pi_i/\pi_j) > 1$, so the use of Eq. (11.29) yields

$$\pi_i p_{ij} = \pi_i \tau_{ij} = \pi_j \tau_{ji}$$

and

$$\pi_j p_{ji} = \pi_j \left(\frac{\pi_i}{\pi_j} \tau_{ji} \right) = \pi_i \tau_{ji}$$

Hence, the principle of detailed balance is satisfied for $\Delta E < 0$.

Case 2: $\Delta E > 0$. Suppose next that the energy change ΔE in going from state x_i to state x_j is positive. In this case we find that $(\pi_j/\pi_i) < 1$, and the use of Eq. (11.29) yields

$$\pi_i p_{ij} = \pi_i \left(\frac{\pi_j}{\pi_i} \tau_{ij} \right) = \pi_j \tau_{ij} = \pi_j \tau_{ji}$$

and

$$\pi_j p_{ji} = \pi_j \tau_{ji}$$

Here again we see that the principle of detailed balance is satisfied.

To complete the picture, we need to clarify the use of the *a priori* transition probabilities denoted by τ_{ij} . These transition probabilities are in fact the probabilistic model of the random step in the Metropolis algorithm. From the description of the algorithm presented earlier, we recall that the random step is followed by a random decision. We may therefore conclude that the transition probabilities p_{ij} defined in Eqs. (11.29) and (11.30) in terms of the *a priori* transition probabilities, τ_{ij} , and the steady state probabilities, π_j , are indeed the correct choice for the Metropolis algorithm.

It is noteworthy that the stationary distribution generated by the Metropolis algorithm does not uniquely determine the Markov chain. The Gibbs distribution at equilibrium may be generated by using an update rule other than the Monte Carlo rule applied in the Metropolis algorithm. For example, it may be generated using the Boltzmann learning rule due to Ackley et al. (1986); this latter rule is discussed in Section 11.7.

11.5 SIMULATED ANNEALING

Consider the problem of finding a low energy system whose states are ordered in a Markov chain. From Eq. (11.11) we observe that as the temperature T approaches zero, the free energy F of the system approaches the average energy $\langle E \rangle$. With $F \rightarrow \langle E \rangle$, we next observe from the principle of minimal free energy that the Gibbs distribution, which is the stationary distribution of the Markov chain, collapses on the global minima

of the average energy $\langle E \rangle$ as $T \rightarrow 0$. In other words, low energy ordered states are strongly favored at low temperatures. These observations prompt us to raise the question: Why not simply apply the Metropolis algorithm for generating a population of configurations representative of the stochastic system at very low temperatures? We do not advocate the use of such a strategy because the rate of convergence of the Markov chain to thermal equilibrium is extremely slow at very low temperatures. Rather, the preferred method for improved computational efficiency is to operate the stochastic system at a high temperature where convergence to equilibrium is fast, and then maintain the system at equilibrium as the temperature is carefully lowered. That is, we use a combination of two related ingredients:

- A schedule that determines the rate at which the temperature is lowered.
- An algorithm—exemplified by the Metropolis algorithm—that iteratively finds the equilibrium distribution at each new temperature in the schedule by using the final state of the system at the previous temperature as the starting point for the new temperature.

The twofold scheme that we have just described is the essence of a widely used stochastic relaxation technique known as *simulated annealing*² (Kirkpatrick et al., 1983). The technique derives its name from analogy with an annealing process in physics/chemistry where we start the process at high temperature and then lower the temperature slowly while maintaining thermal equilibrium.

The primary objective of simulated annealing is to find the global minimum of a cost function that characterizes large and complex systems.³ As such, it provides a powerful tool for solving nonconvex optimization problems, motivated by the following simple idea:

When optimizing a very large and complex system (i.e., a system with many degrees of freedom), instead of always going downhill, try to go downhill most of the time.

Simulated annealing differs from conventional iterative optimization algorithms in two important respects:

- The algorithm need not get stuck, since transition out of a local minimum is always possible when the system operates at a nonzero temperature.
- Simulated annealing is *adaptive* in that gross features of the final state of the system are seen at higher temperatures, while fine details of the state appear at lower temperatures.

Annealing Schedule

As already mentioned, the Metropolis algorithm is the basis for the simulated annealing process, in the course of which the temperature T is decreased slowly. That is, the temperature T plays the role of a *control* parameter. The simulated annealing process will converge to a configuration of minimal energy provided that the temperature is decreased no faster than logarithmically. Unfortunately, such an annealing schedule is extremely slow—too slow to be of practical use. In practice, we must resort to a *finite-time approximation* of the asymptotic convergence of the algorithm. The price paid for

the approximation is that the algorithm is no longer guaranteed to find a global minimum with probability 1. Nevertheless, the resulting approximate form of the algorithm is capable of producing near optimum solutions for many practical applications.

To implement a finite-time approximation of the simulated annealing algorithm, we must specify a set of parameters governing the convergence of the algorithm. These parameters are combined in a so-called *annealing schedule* or *cooling schedule*. The annealing schedule specifies a finite sequence of values of the temperature and a finite number of transitions attempted at each value of the temperature. The annealing schedule due to Kirkpatrick et al. (1983) specifies the parameters of interest as follows:⁴

- *Initial Value of the Temperature.* The initial value T_0 of the temperature is chosen high enough to ensure that virtually all proposed transitions are accepted by the simulated annealing algorithm
- *Decrement of the Temperature.* Ordinarily, the cooling is performed *exponentially*, and the changes made in the value of the temperature are small. In particular, the *decrement function* is defined by

$$T_k = \alpha T_{k-1}, \quad k = 1, 2, \dots \quad (11.34)$$

where α is a constant smaller than, but close to, unity. Typical values of α lie between 0.8 and 0.99. At each temperature, enough transitions are attempted so that there are 10 *accepted* transitions per experiment on the average.

- *Final Value of the Temperature.* The system is frozen and annealing stops if the desired number of acceptances is not achieved at three successive temperatures.

The latter criterion may be refined by requiring that the *acceptance ratio*, defined as the number of accepted transitions divided by the number of proposed transitions, is smaller than a prescribed value (Johnson et al., 1989).

Simulated Annealing for Combinatorial Optimization

Simulated annealing is particularly well suited for solving combinatorial optimization problems. The objective of *combinatorial optimization* is to minimize the cost function of a finite, discrete system characterized by a large number of possible solutions. Essentially, simulated annealing uses the Metropolis algorithm to generate a sequence of solutions by invoking an analogy between a physical many-particle system and a combinatorial optimization problem.

In simulated annealing, we interpret the energy E_i in the Gibbs distribution of Eq. (11.5) as a numerical cost and the temperature T as a control parameter. The numerical cost assigns to each configuration in the combinatorial optimization problem a scalar value that describes how desirable that particular configuration is to the solution. The next issue in the simulated annealing procedure to be considered is how to identify configurations and generate new configurations from previous ones in a local manner. This is where the Metropolis algorithm performs its role. We may thus summarize the correspondence between the terminology of statistical physics and that of combinatorial optimization as shown in Table 11.1 (Beckerman, 1997).

11.6 GIBBS SAMPLING

Like the Metropolis algorithm, the *Gibbs sampler*⁵ generates a Markov chain with the Gibbs distribution as the equilibrium distribution. However, the transition probabilities associated with the Gibbs sampler are nonstationary (Geman and Geman, 1984). In the final analysis, the choice between the Gibbs sampler and the Metropolis algorithm is based on technical details of the problem at hand.

To proceed with a description of this sampling scheme, consider a K -dimensional random vector $\mathbf{X}$ made up of the components $X_1, X_2, \dots, X_K$. Suppose that we have knowledge of the conditional distribution of X_k , given values of all the other components of $\mathbf{X}$ for $k = 1, 2, \dots, K$. The problem we wish to address is how to obtain a numerical estimate of the marginal density of the random variable X_k for each k . The Gibbs sampler proceeds by generating a value for the conditional distribution for each component of the random vector $\mathbf{X}$, given the values of all other components of $\mathbf{X}$. Specifically, starting from an arbitrary configuration $\{x_1(0), x_2(0), \dots, x_K(0)\}$, we make the following drawings on the first iteration of Gibbs sampling:

$x_1(1)$ is drawn from the distribution of X_1 , given $x_2(0), x_3(0), \dots, x_K(0)$.

$x_2(1)$ is drawn from the distribution of X_2 , given $x_1(1), x_3(0), \dots, x_K(0)$.

$\vdots$

$x_k(1)$ is drawn from the distribution of X_k , given $x_1(1), \dots, x_{k-1}(1), x_{k+1}(0), \dots, x_K(0)$.

$\vdots$

$x_K(1)$ is drawn from the distribution of X_K , given $x_1(1), x_2(1), \dots, x_{K-1}(1)$.

We proceed in this same manner on the second iteration and every other iteration of the sampling scheme. The following two points should be carefully noted:

1. Each component of the random vector $\mathbf{X}$ is "visited" in the natural order, with the result that a total of K new variates are generated on each iteration.
2. The new value of component X_{k-1} is used immediately when a new value of X_k is drawn for $k = 2, 3, \dots, K$.

From this discussion we see that the Gibbs sampler is an *iterative adaptive* scheme. After n iterations of its use, we arrive at the K variates: $X_1(n), X_2(n), \dots, X_K(n)$.

TABLE 11.1 Correspondence between Statistical Physics and Combinatorial Optimization

Statistical physics	Combinatorial optimization
Sample	Problem instance
State (configuration)	Configuration
Energy	Cost function
Temperature	Control parameter
Ground-state energy	Minimum cost
Ground-state configuration	Optimal configuration

Under mild conditions, the following three theorems hold for Gibbs sampling (Geman and Geman, 1984; Gelfand and Smith, 1990):

1. *Convergence theorem.* The random variable $X_k(n)$ converges in distribution to the true probability distributions of X_k for $k = 1, 2, \dots, K$ as n approaches infinity; that is,

$$\lim_{n \rightarrow \infty} P(X_k^{(n)} \leq x | x_k(0)) = F_{X_k}(x) \quad \text{for } k = 1, 2, \dots, K \quad (11.35)$$

where $F_{X_k}(x)$ is the marginal probability distribution function of X_k .

In fact, a stronger result is proven in Geman and Geman (1984). Specifically, rather than requiring that each component of the random vector $\mathbf{X}$ be visited in repetitions of the natural order, convergence of Gibbs sampling still holds under an arbitrary visiting scheme provided that this scheme does not depend on the values of the variables and that each component of $\mathbf{X}$ is visited on an “infinitely often” basis.

2. *Rate of convergence theorem.* The joint probability distribution of the random variables $X_1(n), X_2(n), \dots, X_K(n)$ converges to the true joint probability distribution of $X_1, X_2, \dots, X_K$ at a geometric rate in n .

This theorem assumes that the components of $\mathbf{X}$ are visited in the natural order. When, however, an arbitrary but infinitely often visiting approach is used, then a minor adjustment to the rate of convergence is required.

3. *Ergodic theorem.* For any measurable function g , for example, of the random variables $X_1, X_2, \dots, X_K$ whose expectation exists, we have

$$\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=1}^n g(X_1(i), X_2(i), \dots, X_K(i)) \rightarrow E[g(X_1, X_2, \dots, X_K)] \quad (11.36)$$

with probability 1 (i.e., almost surely).

The ergodic theorem tells us how to use the output of the Gibbs sampler to obtain numerical estimations of the desired marginal densities.

Gibbs sampling is used in the Boltzmann machine to sample from distributions over hidden neurons; this stochastic machine is discussed in the next section. In the context of a stochastic machine using binary units (e.g., Boltzmann machine), it is noteworthy that the Gibbs sampler is exactly the same as a variant of the Metropolis algorithm. In the standard form of the Metropolis algorithm, we go downhill with probability 1. In contrast, in the alternative form of the Metropolis algorithm, we go downhill with a probability equal to 1 minus the exponential of the energy gap (i.e., the complement of the uphill rule). In other words, if a change lowers the energy E or leaves it unchanged, that change is accepted; if the change increases the energy, it is accepted with probability $\exp(-\Delta E)$ and is rejected otherwise, with the old state then being repeated (Neal, 1993).

11.7 BOLTZMANN MACHINE

The *Boltzmann machine* is a stochastic machine whose composition consists of stochastic neurons. A *stochastic neuron* resides in one of two possible states in a probabilistic manner, as discussed in Chapter 1. These two states may be designated as +1

for the “on” state and -1 for the “off” state, or 1 and 0, respectively. We will adopt the former designation. Another distinguishing feature of the Boltzmann machine is the use of *symmetric synaptic connections* between its neurons. The use of this form of synaptic connections is also motivated by statistical physics considerations.

The stochastic neurons of the Boltzmann machine partition into two functional groups: *visible* and *hidden*, as depicted in Fig. 11.4. The visible neurons⁶ provide an interface between the network and the environment in which it operates. During the training phase of the network, the visible neurons are all *clamped* onto specific states determined by the environment. The hidden neurons, on the other hand, always operate freely; they are used to explain underlying constraints contained in the environmental input vectors. The hidden neurons accomplish this task by capturing higher-order statistical correlations in the clamping vectors. The network described here represents a special case of the Boltzmann machine. It may be viewed as an unsupervised learning procedure for modeling a probability distribution that is specified by clamping patterns onto the visible neurons with appropriate probabilities. By so doing, the network can perform *pattern completion*. Specifically, when a partial information-bearing vector is clamped onto a subset of the visible neurons, the network performs completion on the remaining visible neurons, provided that it has learned the training distribution properly (Hinton, 1989).

The primary goal of Boltzmann learning is to produce a neural network that correctly models input patterns according to a Boltzmann distribution. In applying this form of learning, two assumptions are made:

- Each environmental input vector (pattern) persists long enough to permit the network to reach *thermal equilibrium*.
- There is *no* structure in the sequential order in which the environmental vectors are clamped onto the visible units of the network.

A particular set of synaptic weights is said to constitute a perfect model of the environmental structure if it leads to exactly the same probability distribution of the states of

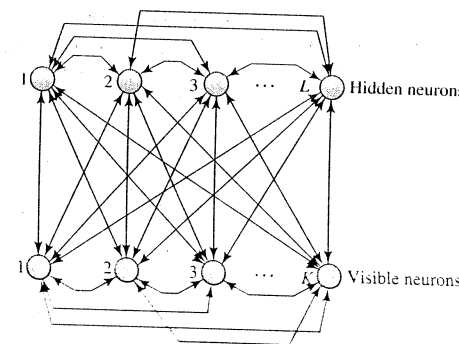


FIGURE 11.4 Architectural graph of Boltzmann machine; K is the number of visible neurons and L is the number of hidden neurons.

the visible units (when the network is running freely) as when these units are clamped by the environmental input vectors. In general, unless the number of hidden units is exponentially large compared to the number of visible units, it is impossible to achieve such a perfect model. If, however, the environment has a regular structure, and the network uses its hidden units to capture these regularities, it may achieve a good match to the environment with a manageable number of hidden units.

Gibbs Sampling and Simulated Annealing for the Boltzmann Machine

Let $\mathbf{x}$ denote the state vector of the Boltzmann machine, with its component x_i denoting the state of neuron i . The state $\mathbf{x}$ represents a realization of the random vector $\mathbf{X}$. The synaptic connection from neuron i to neuron j is denoted by w_{ji} , with

$$w_{ji} = w_{ij} \quad \text{for all } (i, j) \quad (11.37)$$

and

$$w_{ii} = 0 \quad \text{for all } i \quad (11.38)$$

Equation (11.37) describes symmetry and Eq. (11.38) emphasizes the absence of self-feedback. The use of a bias is permitted by using the weight w_{j0} from a fictitious node maintained at +1 and by connecting it to neuron j for all j .

From an analogy with thermodynamics, the energy of the Boltzmann machine is defined by⁷

$$E(\mathbf{x}) = -\frac{1}{2} \sum_i \sum_{j \neq i} w_{ji} x_i x_j \quad (11.39)$$

Invoking the Gibbs distribution of Eq. (11.5), we may define the probability that the network (assumed to be in equilibrium at temperature T) is in state $\mathbf{x}$ as follows:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \quad (11.40)$$

where Z is the partition function.

To simplify the presentation, define the single event A and joint events B and C as follows:

$$\begin{aligned} A: & X_j = x_j \\ B: & \{X_i = x_i\}_{i=1, i \neq j}^K \\ C: & \{X_i = x_i\}_{i=1}^K \end{aligned}$$

In effect, the joint event B excludes A , and the joint event C includes both A and B . The probability of B is the marginal probability of C with respect to A . Hence, using Eqs. (11.39) and (11.40), we may write

$$\begin{aligned} P(C) &= P(A, B) \\ &= \frac{1}{Z} \exp\left(\frac{1}{2T} \sum_i \sum_{j \neq i} w_{ji} x_i x_j\right) \end{aligned} \quad (11.41)$$

and

$$\begin{aligned} P(B) &= \sum_A P(A, B) \\ &= \frac{1}{Z} \sum_{x_j} \exp\left(\frac{1}{2T} \sum_i \sum_{j \neq i} w_{ji} x_i x_j\right) \end{aligned} \quad (11.42)$$

The exponent in Eqs. (11.41) and (11.42) may be expressed as the sum of two components, one involving x_j and the other being independent of x_j . The component involving x_j is given by

$$\frac{x_j}{2T} \sum_{i \neq j} w_{ji} x_i$$

Accordingly, by setting $x_j = x = \pm 1$, we may express the conditional probability of A given B , as follows:

$$\begin{aligned} P(A|B) &= \frac{P(A, B)}{P(B)} \\ &= \frac{1}{1 + \exp\left(-\frac{x_j}{T} \sum_{i \neq j} w_{ji} x_i\right)} \end{aligned}$$

That is, we may write

$$P(X_j = x | \{X_i = x_i\}_{i=1, i \neq j}^K) = \phi\left(\frac{x}{T} \sum_{i \neq j} w_{ji} x_i\right) \quad (11.43)$$

where $\phi(\cdot)$ is a sigmoid function of its argument, as shown by

$$\phi(v) = \frac{1}{1 + \exp(-v)} \quad (11.44)$$

Note that although x varies between -1 and $+1$, the whole argument $v = \frac{x}{T} \sum_{i \neq j} w_{ji} x_i$ for large N may vary between $-\infty$ and $+\infty$, as depicted in Fig. 11.5. Note also, in deriving

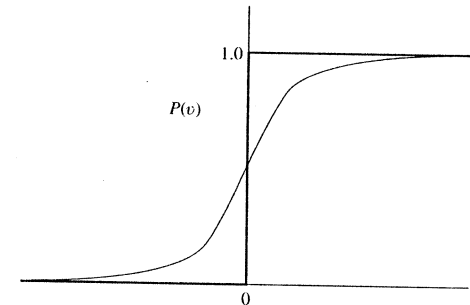


FIGURE 11.5 Sigmoid-shaped function $\phi(v)$.

Eq. (11.43), the need for the partition function Z has been eliminated. This is highly desirable since a direct computation of Z is infeasible for a network of large complexity.

The use of Gibbs sampling exhibits the joint distribution $P(A, B)$. Basically, as explained in Section 11.6, this stochastic simulation starts with the network assigned an arbitrary state, and the neurons are all repeatedly visited in their natural order. On each visit, a new value for the state of each neuron is chosen in accordance with the probability distribution for that neuron, conditional on the values for the states of all other neurons in the network. Provided that the stochastic simulation is performed long enough, the network will reach thermal equilibrium at temperature T .

Unfortunately, the time taken to reach thermal equilibrium can be much too long. To overcome this difficulty, simulated annealing for a finite sequence of temperatures $T_0, T_1, \dots, T_{\text{final}}$ is used, as explained in Section 11.5. Specifically, the temperature is initially set to the high value T_0 , thereby permitting thermal equilibrium to be reached fast. Thereafter, the temperature T is gradually reduced to the final value T_{final} , at which point the neuronal states will have (hopefully) reached their desired marginal distributions.

Boltzmann Learning Rule

Because the Boltzmann machine is a stochastic machine, it is natural to look to probability theory for an appropriate index of performance. One such criterion is the *likelihood function*.⁸ On this basis, the goal of Boltzmann learning is to maximize the likelihood function or, equivalently, the log-likelihood function, in accordance with the *maximum-likelihood principle*.

Let $\mathcal{T}$ denote the set of training examples drawn from the probability distribution of interest. It is assumed that the examples are all two-valued. Repetition of training examples is permitted in proportion to how common certain cases are known to occur. Let a subset of the state vector $\mathbf{x}$, say $\mathbf{x}_\alpha$, denote the state of the visible neurons. The remaining part of the state vector $\mathbf{x}$, say $\mathbf{x}_\beta$, represents the state of the hidden neurons. The state vectors $\mathbf{x}, \mathbf{x}_\alpha$ and $\mathbf{x}_\beta$ are realizations of the random vectors $\mathbf{X}, \mathbf{X}_\alpha$, and $\mathbf{X}_\beta$, respectively. There are two phases to the operation of the Boltzmann machine:

- *Positive phase.* In this phase the network operates in its clamped condition (i.e., under the direct influence of the training set $\mathcal{T}$).
- *Negative phase.* In this second phase, the network is allowed to run freely, and therefore with no environmental input.

Given the synaptic weight vector $\mathbf{w}$ for the whole network, the probability that the visible neurons are in state $\mathbf{x}_\alpha$ is $P(\mathbf{X}_\alpha = \mathbf{x}_\alpha)$. With the many possible values of $\mathbf{x}_\alpha$ contained in the training set $\mathcal{T}$, assumed to be statistically independent, the overall probability distribution is the factorial distribution $\prod_{\mathbf{x}_\alpha \in \mathcal{T}} P(\mathbf{X}_\alpha = \mathbf{x}_\alpha)$. To formulate the log-likelihood function $L(\mathbf{w})$, take the logarithm of this factorial distribution and treat $\mathbf{w}$ as the unknown parameter vector. We may thus write

$$\begin{aligned} L(\mathbf{w}) &= \log \prod_{\mathbf{x}_\alpha \in \mathcal{T}} P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) \\ &= \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \log P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) \end{aligned} \quad (11.45)$$

To formulate the expression for the marginal probability $P(\mathbf{X}_\alpha = \mathbf{x}_\alpha)$ in terms of the energy function $E(\mathbf{x})$, we use the following:

- The probability $P(\mathbf{X} = \mathbf{x})$ is equal to $\frac{1}{Z} \exp(-E(\mathbf{x})/T)$ from Eq. (11.40).
- By definition, the state vector $\mathbf{x}$ is the joint combination of $\mathbf{x}_\alpha$ pertaining to the visible neurons and $\mathbf{x}_\beta$ pertaining to the hidden neurons. Hence, the probability of finding the visible neurons in state $\mathbf{x}_\alpha$ with any $\mathbf{x}_\beta$ is given by

$$P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) = \frac{1}{Z} \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \quad (11.46)$$

where the random vector $\mathbf{X}_\alpha$ is a subset of $\mathbf{X}$. The partition function Z is itself defined by (see Eq. (11.6)):

$$Z = \sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \quad (11.47)$$

Thus, substituting Eqs. (11.46) and (11.47) in (11.45), we obtain the desired expression for the log-likelihood function:

$$L(\mathbf{w}) = \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \left(\log \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) - \log \sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \right) \quad (11.48)$$

The dependence on $\mathbf{w}$ is contained in the energy function $E(\mathbf{x})$, as shown in Eq. (11.39).

Differentiating $L(\mathbf{w})$ with respect to w_{ji} in light of Eq. (11.39), we obtain the following result after some manipulation of terms (see Problem 11.8):

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \frac{1}{T} \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \left(\sum_{\mathbf{x}_\beta} P(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) x_j x_i - \sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}) x_j x_i \right) \quad (11.49)$$

To simplify matters, we introduce two definitions:

$$\begin{aligned} \rho_{ji}^+ &= \langle x_j x_i \rangle^+ \\ &= \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \sum_{\mathbf{x}_\beta} P(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) x_j x_i \end{aligned} \quad (11.50)$$

and

$$\begin{aligned} \rho_{ji}^- &= \langle x_j x_i \rangle^- \\ &= \sum_{\mathbf{x} \in \mathcal{T}} \sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}) x_j x_i \end{aligned} \quad (11.51)$$

In a loose sense, we may view the first average, ρ_{ji}^+ , as the mean firing rate or *correlation* between the states of neurons i and j with the network operating in its clamped or positive phase, and similarly view the second average, ρ_{ji}^- , as the *correlation* between

the states of neurons i and j with the network operating in its free-running or negative phase. With these definitions we may simplify Eq. (11.49) to

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \frac{1}{T} (\rho_{ji}^+ - \rho_{ji}^-) \quad (11.52)$$

The goal of Boltzmann learning is to maximize the log-likelihood function $L(\mathbf{w})$. We may use *gradient ascent* to achieve that goal by writing

$$\begin{aligned} \Delta w_{ji} &= \epsilon \frac{\partial L(\mathbf{w})}{\partial w_{ji}} \\ &= \eta (\rho_{ji}^+ - \rho_{ji}^-) \end{aligned} \quad (11.53)$$

where η is a *learning-rate parameter*; it is defined in terms of ϵ and the operating temperature T as:

$$\eta = \frac{\epsilon}{T} \quad (11.54)$$

The gradient ascent rule of Eq. (11.53) is called the *Boltzmann learning rule*. The learning described here is performed in batch; that is, changes to the synaptic weights are made on the presentation of the entire set of training examples.

According to this learning rule, the synaptic weights of a Boltzmann machine are adjusted by using only locally available observations under two different conditions: (1) clamped, and (2) free running. This important feature of Boltzmann learning greatly simplifies the network architecture, particularly when dealing with large networks. Another useful feature of Boltzmann learning, which may come as a surprise, is that the rule for adjusting the synaptic weight from neuron i to neuron j is independent of whether these two neurons are both visible, both hidden, or one of each. All of these nice features of Boltzmann learning result from a key insight by Hinton and Sejnowski (1983, 1986), which ties the abstract mathematical model of the Boltzmann machine to neural networks by using a combination of two things:

- The Gibbs distribution for describing the stochasticity of a neuron.
- The statistical physics-based energy function of Eq. (11.39) for defining the Gibbs distribution.

From a learning point of view, the two terms that constitute the Boltzmann learning rule of Eq. (11.53) have opposite meaning. We may view the first term, corresponding to the clamped condition of the network, as essentially a Hebbian *learning* rule; and view the second term, corresponding to the free-running condition of the network, as an *unlearning* or *forgetting* term. Indeed, the Boltzmann learning rule represents a *generalization of the repeated forgetting and relearning rule* described by Pöppel and Krey (1987) for the case of symmetric networks with no hidden neurons.

It is also of interest that since the Boltzmann machine learning algorithm requires that hidden neurons know the difference between stimulated and free-running activities, and given that there is a (hidden) external network that signals to hidden neurons that the machine is being stimulated, we have a primitive form of an *attention* mechanism (Cowan and Sharp, 1988).

Need for the Negative Phase and its Implications

The combined use of a positive and a negative phase stabilizes the distribution of synaptic weights in the Boltzmann machine. This need may be justified in another way. Intuitively, we may say that the need for a negative as well as a positive phase in Boltzmann learning arises due to the presence of the partition function, Z , in the expression for the probability of a neuron's state vector. The implication of this statement is that the direction of steepest descent in energy space is *not* the same as the direction of steepest ascent in probability space. In effect, the negative phase in the learning procedure is needed to account for this discrepancy (Neal, 1992).

The use of a negative phase in Boltzmann learning has two major disadvantages:

1. *Increased computation time.* In the positive phase some of the neurons are clamped to the external environment, whereas in the negative phase all the neurons are free running. Accordingly, the time taken for stochastic simulation of a Boltzmann machine is increased.
2. *Sensitivity to statistical errors.* The Boltzmann learning rule involves the *difference* between two average correlations, one computed for the positive phase and the other computed for the negative phase. When these two correlations are similar, the presence of sampling noise makes the difference between them even more noisy.

We may eliminate these shortcomings of the Boltzmann machine by using a sigmoid belief network. In this new class of stochastic machines, control over the learning procedure is exercised by means other than a negative phase.

11.8 SIGMOID BELIEF NETWORKS

Sigmoid belief networks or *logistic belief nets* were developed by Neal in 1992 in an effort to find a stochastic machine that would share with the Boltzmann machine the capacity to learn arbitrary probability distributions over binary vectors, but would not need the negative phase of the Boltzmann machine learning procedure. This objective was achieved by replacing the symmetric connections of the Boltzmann machine with *directed connections that form an acyclic graph*. Specifically, a sigmoid belief network consists of a multilayer architecture with binary stochastic neurons, as illustrated in Fig. 11.6. The acyclic nature of the machine makes it easy to perform probabilistic calculations. In particular, the network uses the sigmoid function of Eq. (11.43), in analogy with the Boltzmann machine, to calculate the conditional probability of a neuron being activated in response to its own induced local field.

Fundamental Properties of Sigmoid Belief Networks

Let the vector $\mathbf{X}$, consisting of the two-valued random variables $X_1, X_2, \dots, X_N$, define a sigmoid belief network composed of N stochastic neurons. The *parents* of element X_j in $\mathbf{X}$ are denoted by

$$\text{pa}(X_j) \subseteq \{X_1, X_2, \dots, X_{j-1}\} \quad (11.55)$$

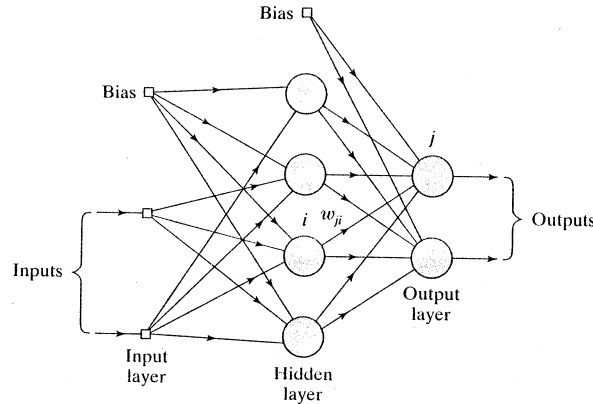


FIGURE 11.6 Architectural graph of sigmoid belief network.

In words, $\text{pa}(X_j)$ is the smallest subset of random vector $\mathbf{X}$ for which we have

$$P(X_j = x_j | X_1 = x_1, \dots, X_{j-1} = x_{j-1}) = P(X_j = x_j | \text{pa}(X_j)) \quad (11.56)$$

An important virtue of sigmoid belief networks is their ability to clearly exhibit the conditional dependencies of the underlying probabilistic model of the input data. In particular, the probability that the i th neuron is activated is defined by the sigmoid function (see Eq. (11.43))

$$P(X_i = x_i | \text{pa}(X_i)) = \phi\left(\frac{x_i}{T} \sum_{i < j} w_{ji} x_j\right) \quad (11.57)$$

where w_{ji} is the synaptic weight from neuron i to neuron j , as shown in Fig. 11.6. That is, the conditional probability $P(X_j = x_j | \text{pa}(X_j))$ depends on $\text{pa}(X_j)$ solely through a sum of weighted inputs. Thus, Eq. (11.57) provides the basis for the propagation of beliefs through the network.

In performing probability calculations on sigmoid belief networks, the following two points are noteworthy:

1. $w_{ji} = 0$ for all X_i not belonging to $\text{pa}(X_j)$
2. $w_{ji} = 0$ for all $i \geq j$

The first point follows from the definition of parents. The second point follows from the fact that a sigmoid belief network is a directed acyclic graph.

As the name implies, sigmoid belief networks belong to the general class of *belief networks*⁹ studied extensively in the literature (Pearl, 1988). The stochastic operation of sigmoid belief networks is somewhat more complex than the Boltzmann machine. Nevertheless, they do lend themselves to the use of gradient-ascent learning in probability space, based on locally available information.

Learning in Sigmoid Belief Networks

Let $\mathcal{T}$ denote a set of training examples drawn from the probability distribution of interest. It is assumed that each example is two-valued, representing certain attributes. Repetition of training examples is permitted, in proportion to how commonly a particular combination of attributes is known to occur. To model the distribution from which $\mathcal{T}$ is drawn, we proceed as follows:

1. Some size for a state vector, $\mathbf{x}$, is decided for the network.
2. A subset of the state vector, say $\mathbf{x}_\alpha$, is selected to represent the attributes in the training cases; that is, $\mathbf{x}_\alpha$ represents the state vector of the visible neurons (i.e., evidence nodes).
3. The remaining part of the state vector $\mathbf{x}$, denoted by $\mathbf{x}_\beta$, defines the state vector of the hidden neurons (i.e., those computational nodes for which we do not have instantiated values).

The design of a sigmoid belief network is highly dependent on the way in which, for a given state vector $\mathbf{x}$, the visible and hidden units are arranged. Therefore, different arrangements of visible and hidden neurons may result in different configurations.

As with the Boltzmann machine, we derive the desired learning rule for a sigmoid belief network by maximizing the log-likelihood function, computed from the training set $\mathcal{T}$. The log-likelihood function, $L(\mathbf{w})$, is defined by Eq. (11.45), reproduced here for convenience of presentation:

$$L(\mathbf{w}) = \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \log P(\mathbf{X}_\alpha = \mathbf{x}_\alpha)$$

where $\mathbf{w}$ is the synaptic weight vector of the network, treated as unknown. The state vector $\mathbf{x}_\alpha$, pertaining to the visible neurons, is a realization of the random vector $\mathbf{X}_\alpha$. Let w_{ji} denote the ji -th element of $\mathbf{w}$ (i.e., synaptic weight from neuron i to neuron j). Differentiating $L(\mathbf{w})$ with respect to w_{ji} , we obtain

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \frac{1}{P(\mathbf{X}_\alpha = \mathbf{x}_\alpha)} \frac{\partial P(\mathbf{X}_\alpha = \mathbf{x}_\alpha)}{\partial w_{ji}}$$

Next we note the following two probabilistic relations:

$$\begin{aligned} P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) &= \sum_{\mathbf{x}_\beta} P(\mathbf{X} = (\mathbf{x}_\alpha, \mathbf{x}_\beta)) \\ &= \sum_{\mathbf{x}_\beta} P(\mathbf{X} = \mathbf{x}) \end{aligned} \quad (11.58)$$

where the random vector $\mathbf{X}$ pertains to the whole network and the state vector $\mathbf{x} = (\mathbf{x}_\alpha, \mathbf{x}_\beta)$ is a realization of it, and

$$P(\mathbf{X} = \mathbf{x}) = P(\mathbf{X} = \mathbf{x} | \mathbf{X}_\alpha = \mathbf{x}_\alpha) P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) \quad (11.59)$$

which defines the probability of the joint event $\mathbf{X} = \mathbf{x} = (\mathbf{x}_\alpha, \mathbf{x}_\beta)$.

In light of these two relations, we may redefine the partial derivative $\partial L(\mathbf{w})/\partial w_{ji}$ in the equivalent form:

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \sum_{\mathbf{x}_\beta} \frac{P(\mathbf{X} = \mathbf{x} | \mathbf{X}_\alpha = \mathbf{x}_\alpha)}{P(\mathbf{X} = \mathbf{x})} \frac{\partial P(\mathbf{X} = \mathbf{x})}{\partial w_{ji}} \quad (11.60)$$

In light of Eq. (11.43), we may write

$$P(\mathbf{X} = \mathbf{x}) = \prod_j \varphi\left(\frac{x_j}{T} \sum_{i < j} w_{ji} x_i\right) \quad (11.61)$$

where $\varphi(\cdot)$ is a sigmoid function. We may therefore write

$$\begin{aligned} \frac{1}{P(\mathbf{X} = \mathbf{x})} \frac{\partial P(\mathbf{X} = \mathbf{x})}{\partial w_{ji}} &= \frac{\partial}{\partial w_{ji}} \log P(\mathbf{X} = \mathbf{x}) \\ &= \frac{\partial}{\partial w_{ji}} \sum_j \log \varphi\left(\frac{x_j}{T} \sum_{i < j} w_{ji} x_i\right) \\ &= \frac{1}{T} \sum_j \frac{1}{\varphi\left(\frac{x_j}{T} \sum_{i < j} w_{ji} x_i\right)} \varphi'\left(\frac{x_j}{T} \sum_{i < j} w_{ji} x_i\right) x_j x_i \end{aligned}$$

where $\varphi'(\cdot)$ is the first derivative of the sigmoid function $\varphi(\cdot)$ with respect to its argument. But, from the definition of $\varphi(\cdot)$ given in Eq. (11.44) we readily find that

$$\varphi'(v) = \varphi(v)\varphi(-v) \quad (11.62)$$

where $\varphi(-v)$ is obtained from $\varphi(v)$ by replacing v with $-v$. Hence, we may write

$$\frac{1}{P(\mathbf{X} = \mathbf{x})} \frac{\partial P(\mathbf{X} = \mathbf{x})}{\partial w_{ji}} = \frac{1}{T} \sum_j \varphi\left(-\frac{x_j}{T} \sum_{i < j} w_{ji} x_i\right) x_j x_i \quad (11.63)$$

Accordingly, substituting Eq. (11.63) in (11.60), we obtain

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \frac{1}{T} \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \sum_{\mathbf{x}_\beta} P(\mathbf{X} = \mathbf{x} | \mathbf{X}_\alpha = \mathbf{x}_\alpha) \varphi\left(-\frac{x_j}{T} \sum_{i < j} w_{ji} x_i\right) x_j x_i \quad (11.64)$$

To simplify matters, we define the ensemble average

$$\begin{aligned} \rho_{ji} &= \langle \varphi\left(-x_j \sum_{i < j} w_{ji} x_i\right) x_j x_i \rangle \\ &= \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \sum_{\mathbf{x}_\beta} P(\mathbf{X} = \mathbf{x} | \mathbf{X}_\alpha = \mathbf{x}_\alpha) \varphi\left(-\frac{x_j}{T} \sum_{i < j} w_{ji} x_i\right) x_j x_i \end{aligned} \quad (11.65)$$

which represents an *average correlation* between the states of neurons i and j , weighted by the factor $\varphi(-\frac{x_j}{T} \sum_{i < j} w_{ji} x_i)$. This average is taken over all possible values of $\mathbf{x}_\alpha$ (drawn from the training set $\mathcal{T}$) as well as all possible values of $\mathbf{x}_\beta$, with $\mathbf{x}_\alpha$ referring to the visible neurons and $\mathbf{x}_\beta$ referring to the hidden neurons.

Gradient ascent in probability space is accomplished by defining the incremental change in synaptic weight w_{ji} as

$$\begin{aligned} \Delta w_{ji} &= \epsilon \frac{\partial L(\mathbf{w})}{\partial w_{ji}} \\ &= \eta \rho_{ji} \end{aligned} \quad (11.66)$$

where $\eta = \epsilon/T$ is a learning-rate parameter and ρ_{ji} is itself defined by Eq. (11.65). Equation (11.66) is the *learning rule for a sigmoid belief network*.

A summary of the sigmoid belief network learning procedure is presented in Table 11.2, where learning is performed in the batch mode; that is, the changes to the synaptic weights of the network are made on the basis of the entire set of training cases. The summary presented in Table 11.2 does not include the use of simulated annealing, which is why we have set the temperature T to 1 therein. However, as with the Boltzmann machine, simulated annealing can be incorporated into the sigmoid belief network learning procedure to reach thermal equilibrium faster, if so desired.

TABLE 11.2 Summary of the Sigmoid Belief Network Learning Procedure

Initialization. Initialize the network by setting the weights w_{ji} of the network to random values uniformly distributed in the range $[-a, a]$; a typical value for a is 0.5.

1. Given a set of training cases $\mathcal{T}$, clamp the visible neurons of the network to $\mathbf{x}_\alpha$, where $\mathbf{x}_\alpha \in \mathcal{T}$.
2. For each $\mathbf{x}_\alpha$, perform a separate Gibbs sampling simulation of the network at some operating temperature T , and observe the resulting state vector $\mathbf{x}$ of the whole network. Provided that the simulation is performed long enough, the values of $\mathbf{x}$ for the different cases contained in the training set $\mathcal{T}$ should come from the conditional distribution of the corresponding random vector $\mathbf{X}$, given that particular training set.
3. Compute the ensemble average

$$\rho_{ji} = \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \sum_{\mathbf{x}_\beta} P(\mathbf{X} = \mathbf{x} | \mathbf{X}_\alpha = \mathbf{x}_\alpha) x_j x_i \varphi\left(-x_j \sum_{i < j} w_{ji} x_i\right)$$

where the random vector $\mathbf{X}_\alpha$ is a subset of $\mathbf{X}$, and $\mathbf{x} = (\mathbf{x}_\alpha, \mathbf{x}_\beta)$ with $\mathbf{x}_\alpha$ referring to the visible neurons and $\mathbf{x}_\beta$ referring to the hidden neurons; x_j is the j th element of state vector $\mathbf{x}$ (i.e., state of neuron j), and w_{ji} is the synaptic weight from neuron i to neuron j . The sigmoid function $\varphi(\cdot)$ is defined by

$$\varphi(v) = \frac{1}{1 + \exp(-v)}$$

4. Increment each synaptic weight w_{ji} of the network by the amount

$$\Delta w_{ji} = \eta \rho_{ji}$$

where η is the learning-rate parameter. This adjustment should move the synaptic weights of the network along the gradient toward a local maximum of the log-likelihood function $L(\mathbf{w})$ in accordance with the maximum likelihood principle.

Unlike the Boltzmann machine, only a single phase is needed for learning in a sigmoid belief network. The reason for this simplification is that normalization of the probability distribution over state vectors is accomplished at the local level of each neuron via the sigmoid function $\varphi(\cdot)$, rather than globally via the difficult to compute partition function Z that involves all possible configurations of states. Once the conditional distribution of the random vector $\mathbf{X}$, given the values of $\mathbf{x}_a$ drawn from the training set $\mathcal{T}$, has been correctly modeled via Gibbs sampling, the role of the negative phase in the Boltzmann machine learning procedure is taken over by the weighting factor $\varphi(-\sum_i w_{ji}x_i)$ involved in computing the ensemble-averaged correlation ρ_{ji} between the states of neurons i and j . When the local minimum of the log-likelihood function $L(\mathbf{w})$ is reached, this weighting factor becomes zero when the network learns a deterministic mapping; otherwise, its average effect comes out to zero.

In Neal (1992), experimental results are presented that show (1) sigmoid belief networks are capable of learning to model nontrivial distributions, (2) these networks can learn at a faster rate than the Boltzmann machine, and (3) this advantage of a sigmoid belief network over the Boltzmann machine is due to the elimination of the negative phase from the learning procedure.

9 HELMHOLTZ MACHINE

Sigmoid belief networks provide a powerful multilayer framework for representing and learning higher-order statistical relationships among sensory inputs of interest in an unsupervised manner. The *Helmholtz machine*,¹⁰ first described in Dayan et al. (1995) and Hinton et al. (1995), provides another ingenious multilayer framework for achieving a similar objective without the use of Gibbs sampling.

The Helmholtz machine uses two entirely different sets of synaptic connections, as illustrated in Fig. 11.7 for the case of a network with two layers of two-valued, stochastic neurons. The forward connections, shown as solid lines in Fig. 11.7, constitute

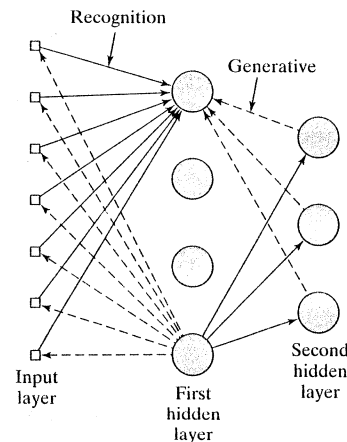


FIGURE 11.7 Architectural graph of Helmholtz machine consisting of interconnected neurons with recognition (solid lines) and generative (dashed lines) connections.

the *recognition model*. The purpose of this model is to *infer* a probability distribution over the underlying causes of the input vector. The backward connections, shown as dashed lines in Fig. 11.7, constitute the *generative model*. The purpose of this second model is to *reconstruct* an approximation to the original input vector from the underlying representations captured by the hidden layers of the network, thereby enabling it to operate in a *self-supervised manner*. Both the recognition and generative models operate in a strictly feedforward fashion, with no feedback; they interact with each other only via the learning procedure.

Hinton et al. (1995) describe a stochastic algorithm, called the *wake-sleep algorithm*, for calculating the recognition and generative weights of the Helmholtz machine. As the name implies, there are two phases to the algorithm: a “wake” phase and a “sleep” phase. In the “wake” phase, the network is driven in the forward direction by the recognition weights. A representation of the input vector is thereby produced in the first hidden layer. This is, in turn, followed by a second representation of that first representation, which is produced in the second hidden layer, and so on for the other hidden layers of the network. The set of representations so produced in the different hidden layers of the network provides a total representation of the input vector by the network. Although the neurons are driven by the recognition weights, it is only the generative weights that are actually learned during the “wake” phase using locally available information. In effect, this phase of the learning process makes each layer of the total representation better at reconstructing the activities formed in the preceding layer.

In the “sleep” phase of the algorithm, the recognition weights are turned off. The network is driven in the backward direction by the generative weights, starting at the outermost hidden layer and working backwards, layer by layer, all the way to the input layer. Because of the fact that the neurons are stochastic, repeating this process would typically give rise to many different “fantasy” vectors on the input layer. These fantasies supply an unbiased sample of the network’s generative model of the world. Having produced a fantasy, the simple delta rule (described in Chapter 3) is used to adjust the recognition weights so as to maximize the logarithm of the probability of recovering the hidden activities that actually caused the fantasy. Like the “wake” phase, the “sleep” phase only uses locally available information.

The learning rule for the generative weights (i.e., backward connections) also uses the simple delta rule. However, instead of following the gradient of the log-likelihood function, this rule follows the gradient of a *penalized* log-likelihood function. The penalty term is the Kullback–Leibler divergence between the true *a posteriori* distribution and the actual distribution produced by the recognition model (Hinton et al., 1995); the Kullback–Leibler divergence or relative entropy is discussed in the preceding chapter. In effect, the penalized log-likelihood function acts as a lower bound on the log-likelihood function of the input data, with the lower bound being improved through the learning process. In particular, the learning process tries to adjust the generative weights to bring the true *a posteriori* distribution as close as possible to the distribution actually computed by the recognition model. Unfortunately, learning the recognition model’s weights does not precisely correspond to the penalized likelihood function. The wake-sleep learning procedure is not guaranteed to work in all practical situations; it does fail sometimes.

10 MEAN-FIELD THEORY

The learning machines considered in the preceding three sections share a common feature: they all use stochastic neurons and may therefore suffer from a slow learning process. In the third and final part of the chapter we study the use of mean-field theory as the mathematical basis for deriving *deterministic approximations* to these stochastic machines to speed up learning. Because the stochastic machines considered herein have different architectures, the theory is applied in correspondingly different ways. In particular, we may identify two specific approaches that have been pursued in literature:

1. Correlations are replaced by their mean-field approximations.
2. An intractable model is replaced by a tractable model via a variational principle.

Approach 2 is highly principled and therefore very appealing. It lends itself for application to the sigmoid belief network (Saul et al., 1996) and the Helmholtz machine (Dayan et al., 1995). However, in the case of the Boltzmann machine, the application of approach 2 is complicated by the need for an upper bound on the partition function Z . For this reason, in Peterson and Anderson (1987), the first approach is used to accelerate the Boltzmann learning rule. In this section we provide a rationale for the first approach. The second approach is considered later in the chapter.

The idea of mean-field approximation is well known in statistical physics (Glauber, 1963). While it cannot be denied that in the context of stochastic machines it is desirable to know the states of all the neurons in the network at all times, we must nevertheless recognize that, in the case of a network with a large number of neurons, the neural states contain vastly more information than we usually require in practice. In fact, to answer the most familiar physical questions about the stochastic behavior of the network, we need only know the average values of neural states or the average products of pairs of neural states.

In a stochastic neuron, the firing mechanism is described by a probabilistic rule. In such a situation, it is rational for us to enquire about the *average* of the state x_j of neuron j . To be precise, we should speak of the average as a "thermal" average, since the synaptic noise is usually modeled in terms of thermal fluctuations. In any event, let $\langle x_j \rangle$ denote the average (mean) of x_j . The state of neuron j is described by the probabilistic rule:

$$x_j = \begin{cases} +1 & \text{with probability } P(v_j) \\ -1 & \text{with probability } 1 - P(v_j) \end{cases} \quad (11.67)$$

where

$$P(v_j) = \frac{1}{1 + \exp(-v_j/T)} \quad (11.68)$$

where T is the operating temperature. Hence we may express the average $\langle x_j \rangle$ for some *specified* value of induced local field v_j as follows:

$$\begin{aligned} \langle x_j \rangle &= (+1)P(v_j) + (-1)[1 - P(v_j)] \\ &= 2P(v_j) - 1 \\ &= \tanh(v_j/2T) \end{aligned} \quad (11.69)$$

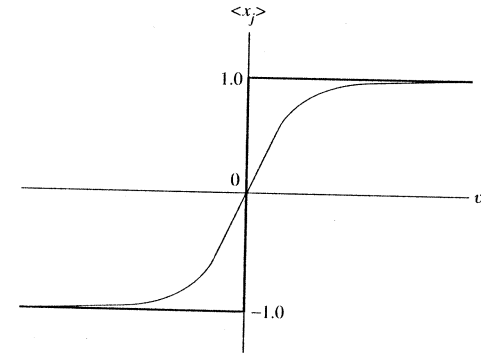


FIGURE 11.8 Graph of the thermal average $\langle x_j \rangle$ versus the induced local field v_j ; heavy solid curve corresponds to normal operation of the McCulloch-Pitts neuron.

where $\tanh(v_j/2T)$ is the hyperbolic tangent of $v_j/2T$. Figure 11.8 shows two plots of the average $\langle x_j \rangle$ versus the induced local field v_j . The continuous curve is for some temperature T greater than zero, and the plot shown in heavy solid lines is for the limiting case of $T = 0$. In the latter case, Eq. (11.69) takes the limiting form

$$\langle x_j \rangle \rightarrow \text{sgn}(v_j) \quad \text{as } T \rightarrow 0 \quad (11.70)$$

which corresponds to the activation function of the McCulloch-Pitts neuron.

The discussion so far has focused on the simple case of a single stochastic neuron. In the more general case of a stochastic machine composed of a large assembly of neurons, we have a much more difficult task on our hands. The difficulty arises because of the combination of two factors:

- The probability $P(v_j)$ that neuron j is on is a nonlinear function of the induced local field v_j .
- The induced local field v_j is a random variable, being influenced by the stochastic action of other neurons connected to the inputs of neuron j .

It is generally safe to say that there is no mathematical method that we can use to evaluate the behavior of a stochastic machine in exact terms. But there is an approximation known as the *mean-field approximation* that we can use, which often yields good results. The basic idea of mean-field approximation is to replace the actual fluctuating induced local field v_j for each neuron j in the network by its average $\langle v_j \rangle$, as shown by

$$v_j^{\text{approx}} = \langle v_j \rangle = \left\langle \sum_i w_{ji} x_i \right\rangle = \sum_i w_{ji} \langle x_i \rangle \quad (11.71)$$

Accordingly, we may compute the average state $\langle x_j \rangle$ for neuron j embedded in a stochastic machine made up of a total of N neurons, just as we did in Eq. (11.69) for a single stochastic neuron, by writing

$$\langle x_j \rangle = \tanh\left(\frac{1}{2T} v_j\right) \stackrel{\text{approx}}{=} \tanh\left(\frac{1}{2T} \langle v_j \rangle\right) = \tanh\left(\frac{1}{2T} \sum_i w_{ji} \langle x_i \rangle\right) \quad (11.72)$$

In light of Eq. (11.72), we may formally state the mean-field approximation as:

The average of some function of a random variable is approximated as the function of the average of that random variable.

For $j = 1, 2, \dots, N$, Eq. (11.72) represents a set of nonlinear equations with N unknowns $\langle x_j \rangle$. The solution of this set of nonlinear equations is now a manageable proposition because the unknowns are all *deterministic* rather than stochastic variables, which they are in the original network.

11.11 DETERMINISTIC BOLTZMANN MACHINE

Learning in the Boltzmann machine is *exponential* in the number of neurons because the Boltzmann learning rule requires the computation of correlations between every pair of neurons in the network. Boltzmann learning therefore requires exponential time. Peterson and Anderson (1987) proposed a method for accelerating the Boltzmann learning process. The method involves replacing the correlations in the Boltzmann learning rule of Eq. (11.53) by a mean-field approximation as shown here:

$$\langle x_j x_i \rangle \stackrel{\text{approx}}{=} \langle x_j \rangle \langle x_i \rangle, \quad (i, j) = 1, 2, \dots, K \quad (11.73)$$

where the mean quantity $\langle x_j \rangle$ is itself computed using the mean-field equation (11.72).

The form of Boltzmann learning in which the computation of the correlations is approximated in the manner just described is called the *deterministic Boltzmann learning rule*. Specifically, the standard Boltzmann learning rule of Eq. (11.53) is approximated as:

$$\Delta w_{ji} = \eta(U_j^+ U_i^+ - U_j^- U_i^-) \quad (11.74)$$

where U_j^+ and U_j^- are the average outputs of visible neuron j (on a single pattern) in the clamped and free-running conditions, respectively, and η is the learning-rate parameter. Whereas the Boltzmann machine uses binary stochastic neurons, its deterministic counterpart uses analog deterministic neurons.

The deterministic Boltzmann machine provides a substantial increase in speed over the standard Boltzmann machine by one or two orders of magnitude (Peterson and Anderson, 1987). However, there are some cautionary notes on its practical use:

1. The deterministic Boltzmann learning rule only works in the supervised case, that is, when some of the visible neurons are assigned the role of output neurons. Unsupervised learning does not work at all in the mean-field regime because the mean state is a very impoverished representation of the free-running probability distribution.

2. In supervised learning, the use of deterministic Boltzmann learning is restricted to neural networks with a single hidden layer (Galland, 1993). In theory, there is no reason not to load multiple hidden layers. In practice, however, the use of more than one hidden layer results in the same problem as for the unsupervised case mentioned under point 1.

The deterministic Boltzmann learning rule in Eq. (11.74) has a simple and local form which makes it a good candidate for implementation in very large scale integration (VLSI) hardware (Alspector et al., 1991; Schneider and Card, 1993). However, in Schneider and Card (1998) it is reported that in the case of continuous learning of capacitive weights, the deterministic Boltzmann machine cannot tolerate weight storage capacitor change decay and offsets in the learning circuitry. The reason is that these internal problems cause the synaptic weight values to drift, resulting in oscillation, which is clearly unacceptable.

11.12 DETERMINISTIC SIGMOID BELIEF NETWORKS

The essence of mean-field approximation described in Section 11.10 is that the average of some function of a random variable can be approximated by the function of the average of that random variable. This viewpoint of mean-field theory works in a limited fashion for the approximation of the Boltzmann machine as discussed in the previous section. In this section we describe another viewpoint of mean-field theory that is well suited to the approximation of a sigmoid belief network. Basically, here we find that an intractable model is approximated by a tractable one via a variational principle (Saul et al., 1996; Jordan et al., 1998). Generally speaking, the tractable model is characterized by a decoupling of the degrees of freedom that make the original model intractable. The decoupling is accomplished by expanding the intractable model to include additional parameters known as *variational parameters* that are designed to fit the problem at hand. The terminology comes from the use of techniques rooted in the calculus of variations (Parisi, 1988).

Lower Bound on the Log-Likelihood Function

The starting point of our discussion is the probabilistic relation of Eq. (11.58), reproduced here in the logarithmic form

$$\log P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) = \log \sum_{\mathbf{x}_\beta} P(\mathbf{X} = \mathbf{x}) \quad (11.75)$$

As in Section 11.8, we partition the random vector $\mathbf{X}$ into $\mathbf{X}_\alpha$ and $\mathbf{X}_\beta$, with $\mathbf{X}_\alpha$ pertaining to the visible neurons and $\mathbf{X}_\beta$ pertaining to the hidden neurons. Realizations of the random vectors $\mathbf{X}$, $\mathbf{X}_\alpha$, and $\mathbf{X}_\beta$ are denoted by the state vectors $\mathbf{x}$, $\mathbf{x}_\alpha$, and $\mathbf{x}_\beta$, respectively. Now, the logarithm of a sum of probabilities required in Eq. (11.75) is difficult to handle. We overcome this difficulty by noting that for any conditional distribution $Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha)$, we may rewrite Eq. (11.75) in a different but equivalent form:

$$\log P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) = \log \sum_{\mathbf{x}_\beta} Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) \left[\frac{P(\mathbf{X} = \mathbf{x})}{Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha)} \right] \quad (11.76)$$

This equation is formulated in a way to prepare it for the application of Jensen's inequality, which is discussed in the previous chapter. On this application we obtain the lower bound:

$$\log P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) \geq \sum_{\mathbf{x}_\beta} Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) \log \left[\frac{P(\mathbf{X} = \mathbf{x})}{Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha)} \right] \quad (11.77)$$

With mean-field theory in mind, henceforth we will refer to the approximation distribution $Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha)$ as the *mean-field distribution*.

What we are interested in is a formula for the log-likelihood function. In the case of a sigmoid belief network, the log-likelihood function $L(\mathbf{w})$ is defined where the summation is over all $\mathbf{x}_\alpha$ (determined by the training set $\mathcal{T}$), hence the use of a batch algorithm for the network. We are going to follow a different strategy for the mean-field approximation to the sigmoid belief network. Specifically, we will adopt a sequential mode of operation, where the log-likelihood function is computed on an *example-by-example basis*, as shown by

$$\mathcal{L}(\mathbf{w}) = \log P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) \quad (11.78)$$

where $\mathbf{w}$ is the weight vector of the network. In the case of identically and independently distributed (*iid*) data, the actual log-likelihood function $\mathcal{L}(\mathbf{w})$ is the sum of $L(\mathbf{w})$ terms, one for each data point. In such a situation, the definitions of $L(\mathbf{w})$ and $\mathcal{L}(\mathbf{w})$ are basically equivalent. In general, the use of $\mathcal{L}(\mathbf{w})$ provides an approximation to $L(\mathbf{w})$.

The sequential or on-line approach to learning has become the standard approach in the design of neural networks, largely because of its simplicity in implementation. Thus, in light of Eq. (11.78), we may write

$$\mathcal{L}(\mathbf{w}) \geq \sum_{\mathbf{x}_\beta} Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) \log \left[\frac{P(\mathbf{X} = \mathbf{x})}{Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha)} \right]$$

or equivalently,

$$\begin{aligned} \mathcal{L}(\mathbf{w}) \geq & - \sum_{\mathbf{x}_\beta} Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) \log Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) \\ & + \sum_{\mathbf{x}_\beta} Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) \log P(\mathbf{X} = \mathbf{x}) \end{aligned} \quad (11.79)$$

The first term on the right-hand side of Eq. (11.79) is the entropy of the mean-field distribution $Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha)$; this is *not* to be confused with a condition entropy. The second term is the average of $\log P(\mathbf{X} = \mathbf{x})$ over all possible states of the hidden neurons. At unit temperature, we note from the discussion on the Gibbs distribution presented in Section 11.2 that the energy of a sigmoid belief network is $-\log P(\mathbf{X} = \mathbf{x})$. Since from Eq. (11.61) we have (for $T = 1$)

$$P(\mathbf{X} = \mathbf{x}) = \prod_j \varphi \left(x_j \sum_{i < j} w_{ji} x_i \right)$$

it follows that

$$\begin{aligned} E &= -\log P(\mathbf{X} = \mathbf{x}) \\ &= -\sum_j \log \varphi \left(x_j \sum_{i < j} w_{ji} x_i \right) \end{aligned} \quad (11.80)$$

Using the definition of the sigmoid function

$$\begin{aligned} \varphi(v) &= \frac{1}{1 + \exp(-v)} \\ &= \frac{\exp(v)}{1 + \exp(v)} \end{aligned}$$

we may thus formally express the energy function of a sigmoid belief network as:

$$E = -\sum_{i < j} \sum_j w_{ji} x_i x_j + \sum_j \log \left(1 + x_j \sum_{i < j} w_{ji} x_i \right) \quad (11.81)$$

Except for a multiplying factor of 1/2, the first term on the right-hand side of Eq. (11.81) is recognized as the energy function of a Markovian system (e.g., Boltzmann machine). However, the second term is unique to sigmoid belief networks.

The lower bound of Eq. (11.79) is valid for any mean-field distribution $Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha)$. To put it to good use, however, we must choose a distribution that enables us to evaluate this bound. We may do so by choosing the *factorial distribution* (Saul et al., 1996)

$$Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) = \prod_{j \in \mathcal{H}} \mu_j^{x_j} (1 - \mu_j)^{1-x_j} \quad (11.82)$$

where $\mathcal{H}$ denotes the set of all hidden neurons, and the states of the hidden neurons appear as independent *Bernoulli variables* with adjustable means μ_j . (A Bernoulli (0) is defined as a binary random variable that takes the value 1 with probability 0.) Thus, substituting Eq. (11.82) in (11.79), we obtain (after simplifications)

$$\begin{aligned} \mathcal{L}(\mathbf{w}) \geq & - \sum_{j \in \mathcal{H}} [\mu_j \log \mu_j + (1 - \mu_j) \log (1 - \mu_j)] \\ & + \sum_{i < j} \sum_{j \in \mathcal{H}} w_{ji} \mu_i \mu_j - \sum_{j \in \mathcal{H}} \langle \log [1 + \exp \left(\sum_{i < j} w_{ji} x_i \right)] \rangle > \end{aligned} \quad (11.83)$$

where the use of $\langle \cdot \rangle$ signifies an ensemble average over the mean-field distribution and $j \in \mathcal{H}$ signifies that j refers to a hidden neuron. The first term on the right-hand side of Eq. (11.83) is the mean-field entropy, and the second term is the mean-field energy. Both of these terms pertain to the factorial distribution of Eq. (11.82).

Unfortunately, we still have an intractable problem: It is not possible to compute an average of the form $\langle \log [1 + \exp(z_j)] \rangle$ exactly. This term arises in Eq. (11.83) with

$$z_j = \sum_{i < j} w_{ji} x_i \quad (11.84)$$

To overcome this difficulty, we again resort to the use of Jensen's inequality for a bound. First, for any variable z_j and any real number ξ_j , we express $\langle \log[1 + \exp(z_j)] \rangle$ in a different but equivalent form as follows:

$$\begin{aligned} \langle \log(1 + e^{z_j}) \rangle &= \langle \log[e^{\xi_j z_j} e^{-\xi_j z_j} (1 + e^{z_j})] \rangle \\ &= \xi_j \langle z_j \rangle + \langle \log[e^{-\xi_j z_j} + e^{(1-\xi_j)z_j}] \rangle \end{aligned} \quad (11.85)$$

where $\langle z_j \rangle$ is the ensemble average of z_j . Next, we apply Jensen's inequality in the opposite direction to what we did before, so as to upper bound the average on the right-hand side of Eq. (11.85), thereby obtaining

$$1 - \langle \log(1 + e^{z_j}) \rangle \leq \xi_j \langle z_j \rangle + \log \langle e^{-\xi_j z_j} + e^{(1-\xi_j)z_j} \rangle \quad (11.86)$$

Putting $\xi_j = 0$ in Eq. (11.86), we obtain the standard bound:

$$\langle \log(1 + e^{z_j}) \rangle \leq \log \langle 1 + e^{z_j} \rangle$$

By permitting the use of nonzero values of ξ_j in Eq. (11.86), we get a tighter bound on the average $\langle \log(1 + e^{z_j}) \rangle$ than would be possible with the standard bound (Seung, 1995), as illustrated in the following example.

Example 11.3 Gaussian-Distributed Variable

To illustrate the utility of the bound described in Eq. (11.86), consider a Gaussian-distributed variable with zero mean and unit variance. For this special case, the exact value of $\langle \log(1 + e^{z_j}) \rangle$ is 0.806. The bound described in Eq. (11.86) yields $[e^{0.5\xi_j^2} + e^{0.5(1-\xi_j)^2}]$, which attains its minimum value of 0.818 at $\xi_j = 0.5$. This bound is a great deal closer to the actual result than the value of 0.974 obtained from the standard bound at $\xi_j = 0$ (Saul et al., 1996).

Returning to the issue at hand, substituting Eqs. (11.85) and (11.86) in (11.83) yields the lower bound on the instantaneous log-likelihood of the evidence $\mathbf{X}_\alpha = \mathbf{x}_\alpha$ as:

$$\begin{aligned} \mathcal{L}(\mathbf{w}) \geq & - \sum_{j \in \mathcal{H}} [\mu_j \log \mu_j + (1 - \mu_j) \log(1 - \mu_j)] \\ & + \sum_{j \in \mathcal{H}} \sum_{i < j} w_{ji} \mu_i (\mu_j - \xi_j) - \sum_{j \in \mathcal{H}} \log \langle \exp(-\xi_j z_j) + \exp((1 - \xi_j)z_j) \rangle \end{aligned} \quad (11.87)$$

where z_i is itself defined by Eq. (11.84). This is the desired bound on the log-likelihood function $\mathcal{L}(\mathbf{w})$ computed on an example-by-example basis.

Learning Procedure for the Mean-Field Approximate to a Sigmoid Belief Network

In deriving the bound of Eq. (11.87), we introduced two sets of *variational parameters*: μ_j for $j \in \mathcal{H}$ and ξ_j for all j , without actually specifying them. These are adjustable parameters. Since the objective is to maximize the log-likelihood function $\mathcal{L}(\mathbf{w})$, it is natural for us to seek those values of μ_j and ξ_j that maximize the expression on the

right-hand side of Eq. (11.87). To achieve this objective, we use a two-step iterative procedure as described in Saul et al. (1996).

Consider first the situation where the mean values μ_j are fixed and the requirement is to find the parameters ξ_j that yield the tightest bound on the log-likelihood function $\mathcal{L}(\mathbf{w})$. Here we note that the expression on the right-hand side of Eq. (11.87) does *not* couple those terms with ξ_j that belong to different neurons in the network. Hence, minimization of this expression with respect to the ξ_j reduces to N independent minimizations over the interval $[0, 1]$, where N is the total number of neurons in the network.

Consider next the situation where the values of ξ_j are fixed and the requirement is to find the mean values μ_j that yield the tightest bound on the log-likelihood function $\mathcal{L}(\mathbf{w})$. For this purpose we introduce the following definitions:

$$K_{ji} = - \frac{\partial}{\partial \mu_i} \log \langle \exp(-\xi_j z_j) + \exp((1 - \xi_j)z_j) \rangle \quad (11.88)$$

where the random variable z_j is itself defined by Eq. (11.84). The partial derivative K_{ji} provides a measure of the parental influence of the state x_i of neuron i on the state x_j of neuron j , given the evidence (example) $\mathbf{x}_\alpha \in \mathcal{T}$. As with the synaptic weight of a sigmoid belief network, K_{ji} is nonzero only when the state x_i is a parent to state x_j . Using the factorial distribution of Eq. (11.82), we may evaluate the ensemble averages of $\exp(-\xi_j z_j)$ and $\exp((1 - \xi_j)z_j)$ and then evaluate the partial derivative K_{ji} , where the formula for computing K_{ji} is given in Table 11.3. With the value of K_{ji} at hand, we may now resume the task of finding the value of parameter μ_j that maximizes the log-likelihood function $\mathcal{L}(\mathbf{w})$ for a fixed ξ_j . In particular, differentiating Eq. (11.87) with respect to μ_j , setting the result equal to zero, and rearranging terms, we obtain

$$\log \left(\frac{\mu_j}{1 - \mu_j} \right) = \sum_{i < j} [w_{ji} \mu_i + w_{ij} (\mu_i - \xi_i) + K_{ij}]$$

Equivalently, we may write

$$\mu_j = \varphi \left(\sum_{i < j} [w_{ji} \mu_i + w_{ij} (\mu_i - \xi_i) + K_{ij}] \right) \quad \text{for } j \in \mathcal{H} \quad (11.89)$$

where $\varphi(\cdot)$ is the sigmoid function. Equation (11.89) is called the *mean-field equation* for a sigmoid belief network. The argument of the sigmoid function in this equation constitutes the so-called *Markov blanket* of neuron j , which is made up as follows:

- The parents and children of neuron j , represented by the terms $w_{ji} \mu_i$ and $w_{ij} \mu_{ij}$, respectively.
- Other parents of the children of neuron j , inherited through the partial derivative K_{ij} .

The Markov blanket of neuron j is illustrated in Fig. 11.9. The notion of a "Markov blanket" was originated by Pearl (1988); it states that the effective input to neuron j , for example, is composed of terms due to its parents, children and their parents.

While it is granted that the choice of the factorial distribution described in Eq. (11.82) as an approximation to the true *a posteriori* distribution $P(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha)$ is not exact, the mean-field equations (11.89) set the parameters $\{\mu_j\}_{j \in \mathcal{H}}$ to optimum

TABLE 11.3 Learning Procedure for the Mean-Field Approximate to a Sigmoid Belief Network

Initialization. Initialize the network by setting the weights w_{ji} of the network to random values uniformly distributed in the range $[-a, a]$; a typical value for a is 0.5.

Computation. For example $\mathbf{x}_a$ drawn from the training set $\mathcal{T}$, perform the following computations:

1. *Updating of $\{\xi_j\}$ for fixed $\{\mu_j\}$.*

Fix the mean values $\{\mu_j\}_{j \in \mathcal{K}}$ pertaining to the factorial approximation to the *a posteriori* distribution $P(\mathbf{X}_\mu = \mathbf{x}_\mu | \mathbf{X}_a = \mathbf{x}_a)$, and minimize the following bound on the log-likelihood function:

$$B(\mathbf{w}) = -\sum_{j \in \mathcal{K}} [\mu_j \log \mu_j + (1 - \mu_j) \log (1 - \mu_j)] + \sum_{i < j} \sum_{j \in \mathcal{K}} w_{ji} \mu_i \mu_j - \sum_{i < j} \sum_{j \in \mathcal{K}} w_{ji} \mu_i \xi_j - \sum_{j \in \mathcal{K}} \log \langle \exp(-\xi_j z_j) + \exp((1 - \xi_j) z_j) \rangle$$

where

$$z_j = \sum_{i < j} w_{ji} x_i$$

The minimization of $B(\mathbf{w})$ reduces to N independent minimizations over the interval $[0, 1]$.

2. *Updating of $\{\mu_j\}$ for fixed $\{\xi_j\}$.*

For fixed values of the parameters $\{\xi_j\}$, iterate the mean-field equations:

$$\mu_j = \varphi \left(\sum_{i < j} [w_{ji} \mu_i + w_{ij} (\mu_i - \xi_i) + K_{ij}] \right)$$

where

$$\begin{aligned} K_{ji} &= -\frac{\partial}{\partial \mu_i} \log \langle \exp(-\xi_j z_j) + \exp((1 - \xi_j) z_j) \rangle \\ &= \frac{(1 - \theta_j)(1 - \exp(-\xi_j w_{ji}))}{1 - \mu_i + \mu_i \exp(-\xi_j w_{ji})} + \frac{\theta_j(1 - \exp((1 - \xi_j) w_{ji}))}{1 - \mu_i + \mu_i \exp((1 - \xi_j) w_{ji})} \\ \theta_j &= \frac{\langle \exp((1 - \xi_j) z_j) \rangle}{\langle \exp(-\xi_j z_j) + \exp((1 - \xi_j) z_j) \rangle} \\ z_j &= \sum_{i < j} w_{ji} x_i \end{aligned}$$

The function $\varphi(\cdot)$ is the sigmoid function:

$$\varphi(v) = \frac{1}{1 + \exp(-v)}$$

TABLE 11.3 (continued)

3. *Correction to synaptic weights.*

For the updated values of parameters $\{\xi_j\}$ and $\{\mu_j\}$, compute the correction Δw_{ji} to synaptic weight w_{ji} :

$$\Delta w_{ji} = \eta \frac{\partial B(\mathbf{w})}{\partial w_{ji}}$$

where η is the learning-rate parameter and

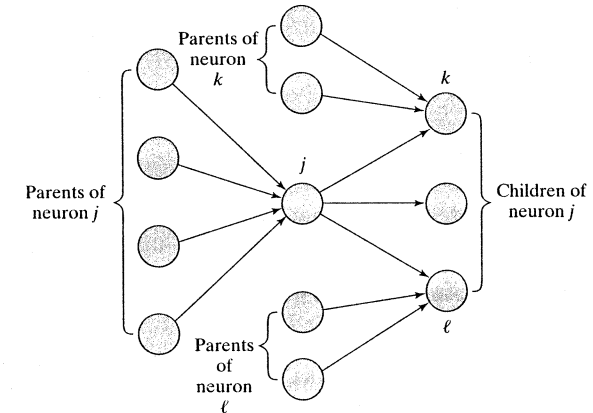
$$\frac{\partial B(\mathbf{w})}{\partial w_{ji}} = -(\xi_j - \mu_j) \mu_i + \frac{(1 - \theta_j) \xi_j \mu_i \exp(-\xi_j w_{ji})}{1 - \mu_i + \mu_i \exp(-\xi_j w_{ji})} - \frac{\theta_j (1 - \xi_j) \mu_i \exp((1 - \xi_j) w_{ji})}{1 - \mu_i + \mu_i \exp((1 - \xi_j) w_{ji})}$$

where θ_j is already defined. Update the synaptic weights:

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

4. *Cycling through the training set $\mathcal{T}$.*

Cycle through all the training examples contained in the training set $\mathcal{T}$, thereby maximizing their likelihood for a fixed number of iterations, or until the onset of overfitting is detected through the use of cross-validation, for example.

**FIGURE 11.9** Illustrating a Markov blanket.

values that make the approximation as accurate as possible. This in turn translates into the tightest mean-field bound on the log-likelihood function $\mathcal{L}(\mathbf{w})$ computed on an example-by-example basis (Saul et al., 1996).

After computing the updated values of parameters $\{\xi_i\}$ and $\{\mu_j\}$, we go on to compute the correction to synaptic weight w_{ji} using the formula:

$$\Delta w_{ji} = \eta \frac{\partial B(\mathbf{w})}{\partial w_{ji}} \quad (11.90)$$

where η is the learning-rate parameter, and $B(\mathbf{w})$ is the lower bound on the log-likelihood function $\mathcal{L}(\mathbf{w})$; that is, $B(\mathbf{w})$ is the expression on the right-hand side of Eq. (11.83). Using this expression, it is a straightforward matter to evaluate the partial derivative $\partial B(\mathbf{w})/\partial w_{ji}$.

A summary of the learning procedure for the mean-field approximate to a sigmoid belief network is presented in Table 11.3. This table includes the formulas for evaluating the partial derivatives K_{ji} and $\partial B(\mathbf{w})/\partial w_{ji}$.

11.3 DETERMINISTIC ANNEALING

We now come to the final topic of the chapter, deterministic annealing. In Section 11.5 we discuss simulated annealing, a stochastic relaxation technique that provides a powerful method for solving nonconvex optimization problems. However, care must be exercised in choosing the annealing schedule. In particular, a global minimum is achieved only if the temperature is decreased at a rate no faster than logarithmically. This requirement makes the use of simulated annealing impractical in many applications. Simulated annealing operates by making random moves on the energy surface (landscape). By contrast, in *deterministic annealing* some form of randomness is incorporated into the energy or cost function itself, which is then deterministically optimized at a sequence of decreasing temperatures (Rose et al., 1990; Rose, 1998); deterministic annealing is *not* to be confused with mean-field annealing (a term that is sometimes used to refer to the deterministic Boltzmann machine).

In what follows we describe the idea of deterministic annealing in the context of an unsupervised learning task: clustering.¹¹

Clustering via Deterministic Annealing

Clustering is defined as the partitioning of a given set of data points into subgroups, each of which should be as homogeneous as possible. Clustering is typically a nonconvex optimization problem since virtually all distortion functions used in clustering are nonconvex functions of the input data. Moreover, a plot of the distortion function versus the input is riddled with local minima, making the task of finding the global minimum even more difficult.

In Rose (1991, 1998), a probabilistic framework is described for clustering by *randomization of the partition*, or equivalently, *randomization of the encoding rule*. The main principle used here is that each data point is *associated in probability* with a particular cluster (subgroup). To be specific, let the random vector $\mathbf{X}$ denote a *source*

(input) vector, and let the random vector $\mathbf{Y}$ denote the best *reconstruction (output) vector* from a codebook of interest. Individual realizations of these two vectors are denoted by vectors $\mathbf{x}$ and $\mathbf{y}$, respectively.

For clustering we need a *distortion measure*, which is denoted by $d(\mathbf{x}, \mathbf{y})$. It is assumed that $d(\mathbf{x}, \mathbf{y})$ satisfies two desirable properties: (1) it is a convex function of $\mathbf{y}$ for all $\mathbf{x}$, and (2) it is finite whenever its arguments are finite. These mild assumptions are satisfied, for example, by the Euclidean distortion measure

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\| \quad (11.91)$$

The *expected distortion* for the randomized pattern is defined by

$$\begin{aligned} D &= \sum_{\mathbf{x}} \sum_{\mathbf{y}} P(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y}) d(\mathbf{x}, \mathbf{y}) \\ &= \sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}) \sum_{\mathbf{y}} P(\mathbf{Y} = \mathbf{y} | \mathbf{X} = \mathbf{x}) d(\mathbf{x}, \mathbf{y}) \end{aligned} \quad (11.92)$$

where $P(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y})$ is the probability of the joint event $\mathbf{X} = \mathbf{x}$ and $\mathbf{Y} = \mathbf{y}$. In the second line of Eq. (11.92) we have used the formula for the probability of a joint event:

$$P(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y}) = P(\mathbf{Y} = \mathbf{y} | \mathbf{X} = \mathbf{x}) P(\mathbf{X} = \mathbf{x}) \quad (11.93)$$

The *conditional probability* $P(\mathbf{Y} = \mathbf{y} | \mathbf{X} = \mathbf{x})$ is referred to as the *association probability*, that is, the probability of associating the code vector $\mathbf{y}$ with the source vector $\mathbf{x}$.

The expected distortion D is traditionally minimized with respect to the free parameters of the clustering model: the reconstruction vector $\mathbf{y}$ and the association probability $P(\mathbf{Y} = \mathbf{y} | \mathbf{X} = \mathbf{x})$. This form of minimization produces a “hard” clustering solution, hard in the sense that a source vector $\mathbf{x}$ is assigned to the nearest code vector $\mathbf{y}$. In deterministic annealing, on the other hand, the optimization problem is reformulated as that of seeking the probability distribution that minimizes the expected distortion *subject to a specified level of randomness*. For a principled measure of the level of randomness, we use the Shannon entropy defined by (see Section 10.4)

$$H(\mathbf{X}, \mathbf{Y}) = \sum_{\mathbf{x}} \sum_{\mathbf{y}} P(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y}) \log P(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y}) \quad (11.94)$$

The constrained optimization of the expected distortion is then expressed as the minimization of the Lagrangian:

$$F = D - TH \quad (11.95)$$

where T is the Lagrange multiplier. From Eq. (11.95) we observe the following:

- For large values of T the entropy H is maximized.
- For small values of T the expected distortion D is minimized, resulting in a hard (nonrandom) clustering solution.
- For intermediate values of T the minimization of F provides a tradeoff between an increase in the entropy H and a reduction in the expected distortion D .

Most importantly, comparing Eq. (11.95) with (11.11), we may identify the correspondence between the constrained clustering optimization problem and statistical

TABLE 11.4 Correspondence between Constrained Clustering and Statistical Physics

Constrained clustering optimization	Statistical physics
Lagrangian, F	Free energy, F
Expected distortion, D	Average energy, $\langle E \rangle$
Shannon entropy, H	Entropy, H
Lagrange multiplier, T	Temperature, T

mechanics listed in Table 11.4. In light of this analogy, henceforth we refer to T as the temperature.

To develop further insight into the Lagrangian F , we note that the joint entropy $H(\mathbf{X}, \mathbf{Y})$ may be decomposed into two terms as follows (see Eq. (10.25)):

$$H(\mathbf{X}, \mathbf{Y}) = H(\mathbf{X}) + H(\mathbf{Y}|\mathbf{X})$$

where $H(\mathbf{X})$ is the source entropy and $H(\mathbf{Y}|\mathbf{X})$ is the conditional entropy of the reconstruction vector $\mathbf{Y}$ given the source vector $\mathbf{X}$. The source entropy $H(\mathbf{X})$ is independent of clustering. Accordingly, we may drop the source entropy $H(\mathbf{X})$ from the definition of the Lagrangian F , and thereby focus on the conditional entropy

$$H(\mathbf{Y}|\mathbf{X}) = - \sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}) \sum_{\mathbf{y}} P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x}) \log P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x}) \quad (11.96)$$

which highlights the role of the association probability $P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x})$. Hence, keeping in mind the correspondence between the constrained clustering optimization problem and statistical physics, and invoking the principle of minimal free energy described in Section 11.2, we find that minimizing the Lagrangian F with respect to the association probabilities results in the Gibbs distribution

$$P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x}) = \frac{1}{Z_{\mathbf{x}}} \exp\left(-\frac{d(\mathbf{x}, \mathbf{y})}{T}\right) \quad (11.97)$$

where $Z_{\mathbf{x}}$ is the partition function for the problem at hand. It is defined by

$$Z_{\mathbf{x}} = \sum_{\mathbf{y}} \exp\left(-\frac{d(\mathbf{x}, \mathbf{y})}{T}\right) \quad (11.98)$$

As the temperature T approaches infinity, we find from Eq. (11.97) that the association probability approaches a uniform distribution. The implication of this statement is that at very high temperatures, each input vector is equally associated with all clusters. Such associations may be viewed as “extremely fuzzy.” At the other extreme, as the temperature T approaches zero, the association probability approaches a delta function. Accordingly, at very low temperatures the classification is hard, with each input sample being assigned to the nearest code vector with probability 1.

To find the minimum value of the Lagrangian F , we substitute the Gibbs distribution of Eq. (11.97) in (11.92) and (11.96) and then use the resulting expressions in the

formula for the Lagrangian F in Eq. (11.95). The result obtained by so doing is (see Problem 11.22):

$$\begin{aligned} F^* &= \min_{P(\mathbf{Y}=\mathbf{y}|\mathbf{X}=\mathbf{x})} F \\ &= -T \sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}) \log Z_{\mathbf{x}} \end{aligned} \quad (11.99)$$

To minimize the Lagrangian with respect to the remaining free parameters, namely the code vectors $\mathbf{y}$, we set the gradients of F^* with respect to $\mathbf{y}$ to zero. Hence, we obtain the condition

$$\sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y}) \frac{\partial}{\partial \mathbf{y}} d(\mathbf{x}, \mathbf{y}) = \mathbf{0} \quad \text{for all } \mathbf{y} \in \mathcal{Y} \quad (11.100)$$

where $\mathcal{Y}$ is the set of all code vectors. Using the formula of Eq. (11.93) and normalizing with respect to $P(\mathbf{X} = \mathbf{x})$, we may redefine this minimizing condition as

$$\frac{1}{N} \sum_{\mathbf{x}} P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x}) \frac{\partial}{\partial \mathbf{y}} d(\mathbf{x}, \mathbf{y}) = \mathbf{0} \quad \text{for all } \mathbf{y} \in \mathcal{Y} \quad (11.101)$$

where the association probability $P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x})$ is itself defined by the Gibbs distribution of Eq. (11.97). In Eq. (11.101) we have included the scaling factor $1/N$ merely for completeness, where N is the number of available examples.

We may now describe the deterministic annealing algorithm for clustering (Rose, 1998). Basically, the algorithm consists of minimizing the Lagrangian F^* with respect to the code vectors at a high value of temperature T , and then tracking the minimum while the temperature T is lowered. In other words, deterministic annealing operates with a specific annealing schedule where the temperature is lowered in an orderly fashion. At each value of the temperature T , a two-step iteration central to the algorithm is performed, as described here:

1. The code vectors are fixed, and the Gibbs distribution of Eq. (11.97) for a specific distortion measure $d(\mathbf{x}, \mathbf{y})$ is used to calculate the association probabilities.
2. The associations are fixed, and Eq. (11.101) is used to optimize the distortion measure $d(\mathbf{x}, \mathbf{y})$ with respect to the code vectors $\mathbf{y}$.

This two-step iterative procedure is monotonically nonincreasing in F^* , and is therefore assured of converging to a minimum. At high values of temperature T , the Lagrangian F^* is fairly smooth, and is a convex function of $\mathbf{y}$ under the mild assumptions previously made on the distortion measure $d(\mathbf{x}, \mathbf{y})$. A global minimum of F^* can be found at high temperatures. As the temperature T is lowered, the association probabilities become hard, resulting in a hard clustering solution.

As the temperature T is lowered in the course of going through the annealing schedule, the system undergoes a sequence of phase transitions, which consists of natural cluster splits where the clustering model grows in size (i.e., number of clusters) (Rose et al., 1990; Rose, 1991). This phenomenon is significant for the following reasons:

- It provides a useful tool for *controlling* the size of the clustering model.
- As in ordinary physical annealing, the phase transitions are the *critical points* of the deterministic annealing process where care has to be exercised with the annealing.

- The critical points are *computable*, thereby providing information that can be used to accelerate the algorithm in between phase transitions.
- An *optimum model size* may be identified by coupling a validation procedure with the sequence of solutions produced at various phases, which represent solutions of increasing model size.

Example 11.4

Figures 11.10 and 11.11 illustrate the evolution of the clustering solution via deterministic annealing at various phases as the temperature T is decreased or the reciprocal of temperature, $B = 1/T$, is increased (Rose 1991). The data set used to generate these figures is a mixture of six Gaussian distributions whose centers are marked with X. The centers of the computed clusters are marked with o. Since the clustering solutions at non-zero temperatures are not hard, this random partition is depicted by contours of equal probability—for example, probability 1/3 of belonging to a particular cluster. This process starts with one natural cluster containing the training set (Fig. 11.10a). At the first phase transition, it splits into two clusters (Fig. 11.10b), and then passes through a sequence of phase transitions until it reaches the “natural” set of six clusters. The next phase transition results in an “explosion” when all clusters split. Figure 11.11 shows the phase diagram, displaying the behavior of the average distortion throughout the annealing process and the number of natural clusters at each phase. In this figure the average distortion (normalized with respect to its minimum value) is plotted versus the reciprocal of temperature, namely B (normalized with respect to its minimum value). Both axes are labeled in their relative logarithmic forms.

FIGURE 11.10 Clustering at various phases. The lines are equiprobability contours, $p = 1/2$ in (b), and $p = 1/3$ elsewhere:

- (a) 1 cluster ($B = 0$),
 (b) 2 clusters ($B = 0.0049$),
 (c) 3 clusters ($B = 0.0056$),
 (d) 4 clusters ($B = 0.0100$),
 (e) 5 clusters ($B = 0.0156$),
 (f) 6 clusters ($B = 0.0347$), and
 (g) 19 clusters ($B = 0.0605$).

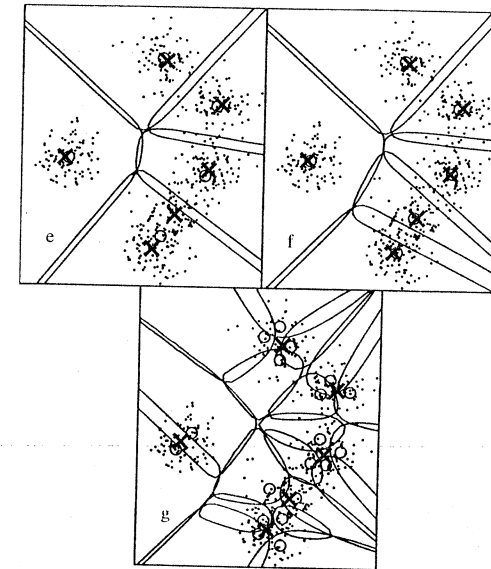
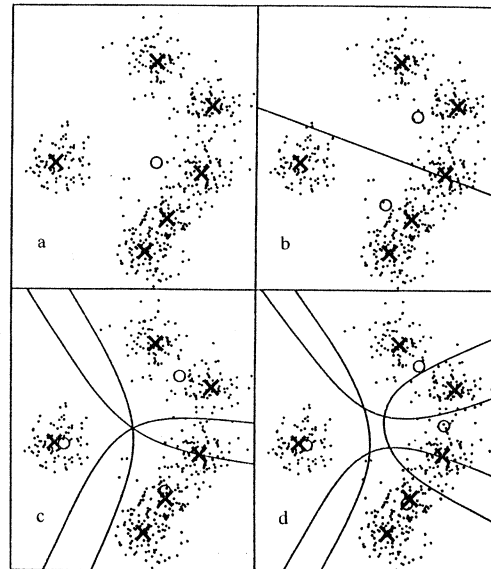


FIGURE 11.10 (continued)

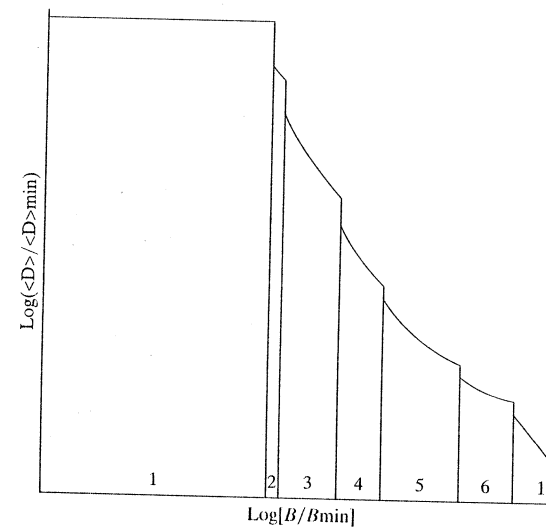


FIGURE 11.11 Phase diagram for the mixture of Gaussian example. The number of effective clusters is shown for each phase.

Analogy with the EM Algorithm

For another important aspect of the deterministic annealing algorithm, suppose we view the association probability $P(\mathbf{Y} = \mathbf{y} | \mathbf{X} = \mathbf{x})$ as the expected value of a random binary variable $V_{\mathbf{xy}}$ defined as:

$$V_{\mathbf{xy}} = \begin{cases} 1 & \text{if the source vector } \mathbf{x} \text{ is assigned to code vector } \mathbf{y} \\ 0 & \text{otherwise} \end{cases} \quad (11.102)$$

Then from such a perspective we recognize the two-step iteration of the deterministic annealing algorithm to be a form of the Expectation-Maximization (EM) algorithm described in Chapter 7 for maximum likelihood estimation. In particular, in step 1 that computes the association probabilities, we have the equivalent of the expectation step. In step 2 that minimizes the Lagrangian F^* , we have the equivalent of the maximization step.

In making this analogy, however, note that deterministic annealing is more general than maximum likelihood estimation. We say so because, unlike maximum likelihood estimation, deterministic annealing does *not* make any assumption on the underlying probability distribution of the data. The association probabilities are, in fact, derived from the Lagrangian F^* to be minimized.

11.14 SUMMARY AND DISCUSSION

In this chapter we discuss the use of ideas rooted in statistical mechanics as the mathematical basis for the formulation of optimization techniques and learning machines. The learning machines considered here may be categorized as follows:

- *Stochastic machines*, exemplified by the Boltzmann machine, sigmoid belief networks, and the Helmholtz machine.
- *Deterministic machines*, derived from the Boltzmann machine and sigmoid belief networks by invoking mean-field approximations.

The Boltzmann machine uses hidden and visible neurons that are in the form of stochastic, binary-state units. It cleverly exploits the beautiful properties of the Gibbs distribution, thereby offering some appealing features:

- Through training, the probability distribution exhibited by the neurons is matched to that of the environment.
- The network offers a generalized approach that is applicable to the basic issues of search, representation, and learning (Hinton, 1989).
- The network is guaranteed to find the global minimum of the energy surface with respect to the *states*, provided that the annealing schedule in the learning process is performed slowly enough (Geman and Geman, 1984).

Unfortunately, the annealing schedule is much too slow to be of practical value. However, the learning process can be accelerated for specific classes of Boltzmann machines, for which we do not have to run a sampling algorithm or apply a mean-field approximation. In particular, in Boltzmann machines in which the hidden neurons are

in the form of a chain, a tree, or a coupled pair of chains or trees, learning can be performed exactly in polynomial time. This is achieved by using an algorithm from statistical mechanics known as “decimation,” which is a simple and elegant procedure that recursively removes links and nodes from the graph, much like solving a resistance inductance capacitance (RLC) circuit (Saul and Jordan, 1995, 1996).

Sigmoid belief networks offer a significant improvement over the Boltzmann machine by eliminating the need for the negative (free-running) phase. They do so by replacing the symmetric connections of the Boltzmann machine with directed acyclic connections. That is, whereas the Boltzmann machine is a recurrent network with an abundance of feedback, sigmoid belief networks have a multilayer architecture with *no* feedback. As the name implies, sigmoid belief networks are closely related to classical belief networks pioneered by Pearl (1988), thereby linking the subject of neural networks with that of *probabilistic reasoning and graphical models* (Jordan, 1998; Jordan et al., 1998).

The Helmholtz machine is different again. Its development is motivated by the idea that vision is inverse graphics (Horn, 1977; Hinton and Ghahramani, 1997). In particular it uses a stochastic, generative model, operating in the backward direction, to convert an abstract representation of a scene into an intensity image. The abstract representation of the scene (i.e., the network’s own visual knowledge of the world) is itself learned by a stochastic, recognition model, operating in the forward direction. Through a clever integration of the recognition and generative models (i.e., forward/backward projections), the Helmholtz machine assumes the role of a self-supervising machine, thereby eliminating the need for a teacher.

Turning next to the class of deterministic machines, the deterministic Boltzmann machine is derived from the Boltzmann machine by applying a naive form of mean-field approximation, where the correlation between two random variables is replaced by the product of their mean values. The net result is that the deterministic Boltzmann machine can be considerably faster than the standard stochastic Boltzmann machine. Unfortunately, in practice its use is restricted to a single layer of hidden neurons. In Kappen and Rodriguez (1998), it is argued that, in the correct treatment of mean-field theory for the Boltzmann machine, the correlations need to be computed using the linear response theorem. The essence of this theorem is to replace the clamped and free-running correlations in the Boltzmann learning rule of Eq. (11.53) by their linear response approximations. According to Kappen and Rodriguez, the new learning procedure is applicable to networks with or without hidden neurons.

The deterministic form of sigmoid belief networks is derived by applying another form of mean-field theory, where a rigorous lower bound on the log-likelihood function is derived through the use of Jensen’s inequality. Moreover, the theory exploits the virtues of a tractable substructure in a principled manner, making this class of neural networks into an important addition to belief networks.

In this chapter we also discuss two optimization techniques: simulated annealing and deterministic annealing. Simulated annealing distinguishes itself by performing random moves on the energy surface, which can make the annealing schedule very slow, with the result that its use is unrealistic for many applications. By contrast, deterministic annealing incorporates randomness into the cost function, which is then deterministically optimized at each temperature sequentially, starting at a high temperature and then

going down. However, note that simulated annealing is guaranteed to reach a global minimum, whereas no such guarantee has yet been found for deterministic annealing.

Although, in this chapter, we emphasize the use of optimization techniques and stochastic machines for solving unsupervised learning tasks, they can all also be used for supervised learning tasks if so desired.

NOTES AND REFERENCES

1. The term "canonical distribution" as a description of Eq. (11.3) was coined by J. Willard Gibbs (1902). On page 33 of Part One (Elementary Principles in Statistical Mechanics) of his collected works, he wrote

"The distribution represented by...

$$P = \exp\left(\frac{\psi - \epsilon}{H}\right)$$

where H and ψ are constants, and H positive, seems to represent the most simple case conceivable, since it has the property that when the system consists of parts with separate energies, the laws of the distribution in phase of the separate parts are of the same nature—a property which enormously simplifies the discussion, and is the foundation of extremely important relations to thermodynamics....

When an ensemble of systems is distributed in phase in the manner described, i.e., when the index of probability (P) is a linear function of the energy (ϵ), we shall say that the ensemble is *canonically distributed*, and shall call the divisor of the energy (H) the modulus of distribution."

In the physics literature, Eq. (11.3) is commonly referred to as the canonical distribution (Reif, 1965) or Gibbs distribution (Landau and Lifschitz, 1980). In the neural network literature it has been referred to as the Gibbs distribution, Boltzmann distribution, and Boltzmann-Gibbs distribution.

2. The idea of introducing temperature and simulating annealing into combinatorial optimization problems is due to Kirkpatrick, Gelatt, and Vacchi (1983) and independently to Černý (1985).

In a physical context, annealing is a delicate process by nature. In their 1983 paper, Kirkpatrick et al. discuss the notion of "melting" a solid, which involves raising the temperature to a maximum value at which all particles of the solid arrange themselves "randomly" in the liquid phase. Then the temperature is lowered, permitting all particles to arrange themselves in the low-energy ground state of a corresponding lattice. If the cooling is too rapid—that is, the solid is not allowed enough time to reach thermal equilibrium at each temperature value—the resulting crystal will have many defects, or the substance may form a glass with no crystalline order and only metastable locally optimal structures.

The notion of "melting" may be the right way of thinking about glasses, and perhaps combinatorial optimization problems in a corresponding computational context. However, it is misleading when discussing many other application domains (Beckerman, 1997). For example, in image processing, if we raise the "temperature" so that all particles arrange themselves randomly we have lost the image—it becomes uniformly gray. In a corresponding metallurgical sense, when we anneal either iron or copper we must keep the annealing temperature below the melting point; otherwise we ruin the sample.

There are several important parameters that govern metallurgical annealing:

- *Annealing temperature*, which specifies the temperature to which the metal or alloy is heated.
- *Annealing time*, which specifies the duration of time for which the elevated temperature is maintained.
- *Cooling schedule*, which specifies the rate at which the temperature is lowered.

These parameters have their counterparts in simulated annealing as described in the subsection on annealing schedule.

3. The *Langevin equation* (with time-dependent temperature) provides the basis for another global optimization algorithm that was proposed by Grenander (1983), and subsequently analyzed by Gidas (1985). The Langevin equation is a stochastic differential equation described as (Reif, 1965):

$$\frac{dv(t)}{dt} = -\gamma v(t) + \Gamma(t)$$

where $v(t)$ is the velocity of a particle of mass m immersed in a viscous fluid, γ is a constant equal to the ratio of the coefficient of friction to the mass m , and $\Gamma(t)$ is a fluctuating force per unit mass. The Langevin equation was the first mathematical equation to describe nonequilibrium thermodynamics.

4. For more elaborate and theoretically oriented annealing schedules, see the books by Aarts and Korst (1989, pp. 60–75) and by van Laarhoven and Aarts (1988, pp. 62–71).
5. Gibbs sampling is referred to in statistical physics as a "heat bath" version of the Metropolis algorithm. It is widely used in image processing, neural networks, and statistics, following its formal exposition in the literature by Geman and Geman (1984) and Gelfand and Smith (1990). The latter paper also discusses other approaches to sampling (or Monte Carlo) based approaches to the numerical calculation of estimates of marginal probability distributions. Hastings (1970) presented a generalization of the Metropolis algorithm, of which Gibbs sampling is a special case; its potential for solving numerical problems in statistics was mentioned.
6. The visible neurons of a Boltzmann machine may also be subdivided into input and output neurons. In this second configuration, the Boltzmann machine performs *association* under the supervision of a teacher. The input neurons receive information from the environment and the output neurons report the outcome of the computation to an end user.
7. The formula of Eq. (11.39) applies to a Boltzmann machine whose "on" and "off" states are denoted by +1 and -1, respectively. In the case of a machine using 1 and 0 to denote its "on" and "off" states, respectively, we have

$$E(\mathbf{x}) = - \sum_i \sum_{j, i \neq j} w_{ji} x_i x_j$$

8. Traditionally, the relative entropy or Kullback-Leibler distance has been used as the index of performance for the Boltzmann machine (Ackley et al., 1985; Hinton and Sejnowski, 1986). This criterion provides a measure of the discrepancy between the environment and the network's internal model. It is defined by

$$D_p \|_p = \sum_{\alpha} p_{\alpha}^{+} \log \left(\frac{p_{\alpha}^{+}}{p_{\alpha}^{-}} \right)$$

where p_{α}^{+} is the probability that the visible neurons are in state α when the network is in its clamped condition, and p_{α}^{-} is the probability that the same neurons are in state α when

the network is in its free-running condition. The synaptic weights of the network are adjusted to minimize $D_{p_{\text{all}}||p_0}$; see Problem 11.10.

The principles of minimum Kullback–Leibler divergence and maximum likelihood are basically equivalent when applied to a training set. To see the equivalence, we note that the Kullback–Leibler divergence between two distributions f and g is given by

$$D_{f||g} = -H(f) - \sum f \log(g)$$

If the distribution f is specified by a training set, and a model for g is given for optimization, the first term is constant, and the second term is the negative of log-likelihood. Hence, minimum Kullback–Leibler divergence is equivalent to maximum likelihood.

9. Belief networks were originally introduced for the purpose of representing probabilistic knowledge in expert systems (Pearl, 1988). They are also referred to in the literature as *Bayesian networks* or *Bayesian nets*.
10. The Helmholtz machine belongs to a class of neural networks characterized by *forward-backward projections*. The idea of forward-backward projections was originated by Grossberg (1980) in his studies of *adaptive resonance theory*; see also Carpenter and Grossberg (1987). In this model, forward adaptive filtering is combined with backward template matching so that adaptive resonance (i.e., amplification and prolongation of neural activity) takes place. In contrast to Grossberg's adaptive resonance theory, the Helmholtz machine uses a statistical approach to treat self-supervised learning as one of ascertaining a generative model that tries to accurately capture the underlying structure of the input data.

Another closely related work is that of Luttrell (1994, 1997). In Luttrell (1994), the idea of a *folded Markov chain* (FMC) is developed. Specifically, the forward transitions through a Markov chain are followed by inverse transitions (using Bayes' theorem) in a backward direction through a copy of the same chain. In Luttrell (1997), the relationship between the FMC and Helmholtz machine is discussed.

Other related works include those of Kawato et al., (1993) where forward (recognition) and backward (generative) models are considered in a similar manner to the Helmholtz machine but without a probabilistic perspective, and the proposals by Mumford (1994) for mapping Grenander's generative model onto the brain.

In Dayan and Hinton (1996), a number of different varieties of Helmholtz machine, including a supervised scheme, are suggested.

11. Deterministic annealing has been successfully applied to many learning tasks:

- Vector quantization (Rose et al., 1992; Miller and Rose, 1994)
- Statistical classifier design (Miller et al., 1996)
- Nonlinear regression using mixture of experts (Rao et al., 1997a)
- Hidden Markov models for speech recognition (Rao, et al., 1997b)

A *hidden Markov model* is similar to a Markov chain because in both cases the transition from one state to another is probabilistic. However, they differ from each other in one fundamental respect. In a Markov chain, the production of an output symbol is deterministic. In a hidden Markov model, on the other hand, the output symbols are probabilistic as well, with the result that all symbols are possible at each state. Thus, with each state of a hidden Markov model, we have a probability distribution of all the output symbols. Hidden Markov models are discussed in Rabiner (1989), Rabiner and Juang (1986), and Jelinek (1997).

PROBLEMS

Markov Chains

- 11.1 The n -step transition probability from state i to state j is denoted by $p_{ij}^{(n)}$. Using the method of induction, show that

$$p_{ij}^{(1+n)} = \sum_k p_{ik} p_{kj}^{(n)}$$

- 11.2 Figure P11.2 shows the state transition diagram for the *random walk* process, where the transition probability p is greater than zero. Is the infinitely long Markov chain depicted here irreducible? Justify your answer.
- 11.3 Consider the Markov chain depicted in Fig. P11.3, which is reducible. Identify the classes of states contained in this state transition diagram.
- 11.4 Calculate the steady-state probabilities of the Markov chain shown in Fig. P11.4.

Simulation techniques

- 11.5 The Metropolis algorithm and the Gibbs sampler represent two alternative techniques for simulating a large-scale problem of interest. Discuss the basic similarities and differences between them.
- 11.6 In this problem we consider the use of simulated annealing for solving the *traveling salesman problem* (TSP). You are given the following:

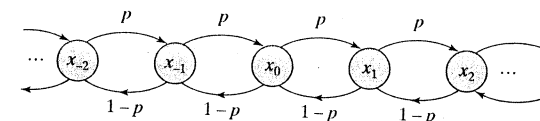


FIGURE P11.2

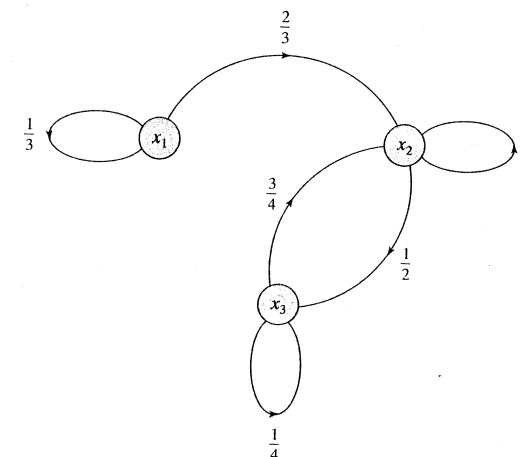


FIGURE P11.3

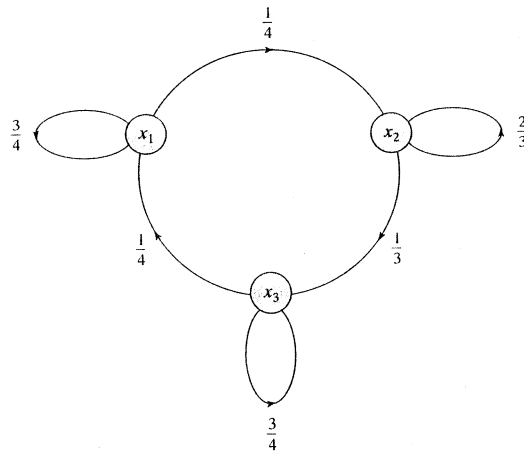


FIGURE P11.4

- N cities
- The distance between each pair of cities, d
- A tour represented by a closed path visiting each city once and only once.

The objective is to find a tour (i.e., permutation of the order in which the cities are visited) that is of minimal total length L . In this problem the different possible tours are the configurations, and the total length of a tour is the cost function to be minimized.

(a) Devise an iterative method of generating valid configurations.

(b) The total length of a tour is defined by

$$L_P = \sum_{i=1}^N d_{P(i)P(i+1)}$$

where P denotes a permutation with $P(N+1) = P(1)$. Correspondingly, the partition function is

$$Z = \sum_P e^{-L_P/T}$$

where T is a control parameter. Set up a simulated annealing algorithm for the TSP.

Boltzmann machine

11.7 Consider a stochastic, two-state neuron j operating at temperature T . This neuron flips from state x_j to state $-x_j$ with probability

$$P(x_j \rightarrow -x_j) = \frac{1}{1 + \exp(-\Delta E_j/T)}$$

where ΔE_j is the energy change resulting from such a flip. The total energy of the Boltzmann machine is defined by

$$E = -\frac{1}{2} \sum_i \sum_{j \neq i} w_{ij} x_i x_j$$

where w_{ij} is the synaptic weight from neuron i to neuron j , with $w_{ii} = w_{ji}$ and $w_{ii} = 0$.

(a) Show that

$$\Delta E_j = -2x_j v_j$$

where v_j is the induced local field of neuron j .

(b) Hence, show that for an initial state $x_j = -1$, the probability that neuron j is flipped into state $+1$ is $1/(1 + \exp(-2v_j/T))$.

(c) Show that the same formula in part (b) holds for neuron j flipping into state -1 when it is initially in state $+1$.

11.8 Derive the formula given in Eq. (11.49) that defines the derivative of the log-likelihood function $L(\mathbf{w})$ with respect to the synaptic weight w_{ij} for the Boltzmann machine.

11.9 The Gibbs distribution may be derived using a self-contained mathematical approach that does not rely on concepts from statistical physics. In particular, a *two-step Markov chain model* of a stochastic machine may be used to formalize the assumptions that yield the unique properties of the Boltzmann machine (Mazaika, 1987). This should not come as a surprise since the simulated annealing, basic to the operation of the Boltzmann machine is known to have a Markov property of its own (van Laarhoven and Aarts, 1988).

Consider then a transition model between states of a neuron in a stochastic machine that is composed of two random processes:

- The first process decides which state transition should be attempted.
- The second process decides if the transition succeeds.

(a) Expressing the transition probability p_{ji} as the product of two factors, that is,

$$p_{ji} = \tau_{ji} q_{ji} \quad \text{for } j \neq i$$

show that

$$p_{ii} = 1 - \sum_{j \neq i} \tau_{ji} q_{ji}$$

(b) Assume that the attempt rate matrix is symmetric:

$$\tau_{ji} = \tau_{ij}$$

Also assume that the probability of a successful attempt satisfies the property of complementary conditional transition probability:

$$q_{ji} = 1 - q_{ij}$$

By invoking these two assumptions, show that

$$\sum_j \tau_{ji} (q_{ji} \pi_j + q_{ij} \pi_i - \pi_j) = 0$$

(c) Given that $\tau_{ji} \neq 0$, use the result of a part (a) of the problem to show that

$$q_{ji} = \frac{1}{1 + (\pi_i/\pi_j)}$$

(d) Finally, make a change of variables:

$$E_i = -T \log \pi_i + T^*$$

where T and T^* are arbitrary constants. Hence, derive the following results:

$$(i) \quad \pi_i = \frac{1}{Z} \exp\left(-\frac{E_i}{T}\right)$$

$$(ii) Z = \sum_j \exp\left(-\frac{E_j}{T}\right)$$

$$(iii) q_i = \frac{1}{1 + \exp(-\Delta E/T)}$$

where $\Delta E = E_j - E_i$.

(e) What conclusions can you draw from these results?

- 11.10 In Section 11.7 we use maximum likelihood as the criterion for deriving the Boltzmann learning rule, described in Eq. (11.53). In this problem we revisit this learning rule using another criterion. From the discussion presented in Chapter 10, the Kullback–Leibler divergence between two probability distributions p_α^+ and p_α^- is defined by

$$D_{p^+ \| p^-} = \sum_\alpha p_\alpha^+ \log\left(\frac{p_\alpha^+}{p_\alpha^-}\right)$$

where the summation is over all possible states α . The probability p_α^+ denotes the probability that the visible neurons are in state α when the network is in its clamped (positive) condition, and the probability p_α^- denotes the probability that the same neurons are in a state α when the network is in its free-running (negative) condition. Using $D_{p^+ \| p^-}$, rederive the Boltzmann learning rule.

- 11.11 Consider a Boltzmann machine whose visible neurons are divided into input neurons and output neurons. The states of these neurons are denoted by α and γ , respectively. The state of the hidden neurons is denoted by β . The Kullback–Leibler divergence for this machine is defined by

$$D_{p^+ \| p^-} = \sum_\alpha p_\alpha^+ \sum_\gamma p_{\gamma|\alpha}^+ \log\left(\frac{p_{\gamma|\alpha}^+}{p_{\gamma|\alpha}^-}\right)$$

where p_α^+ is the probability of state α over the input neurons; $p_{\gamma|\alpha}^+$ is the conditional probability that the output neurons are clamped in state α given an input state α ; and $p_{\gamma|\alpha}^-$ is the conditional probability that the output neurons are in thermal equilibrium in state γ given that only the input neurons are clamped in state α . As before, the plus and minus superscripts denote the positive (clamped) and negative (free-running) conditions, respectively.

- (a) Derive the formula $D_{p^+ \| p^-}$ for a Boltzmann machine that includes input, hidden, and output neurons.
 (b) Show that the Boltzmann learning rule for adjusting the synaptic weight w_{ji} in this network configuration may still be expressed in the same form as that described in Eq. (11.53), with new interpretations for the correlations p_{ji}^+ and p_{ji}^- .

Sigmoid Belief Networks

- 11.12 Summarize the similarities and differences between the Boltzmann machine and a sigmoid belief network.
 11.13 In Problem 11.9 we demonstrated that the Boltzmann machine is described by a two-step Markov chain model. Does a sigmoid belief network admit a Markov chain model? Justify your answer.
 11.14 Let w_{ji} denote the synaptic weight from neuron i to neuron j in a sigmoid belief network that uses +1 for the on state of a neuron and -1 for the off state. Let w'_{ji} denote the corresponding synaptic weight of a sigmoid belief network that uses 1 for the on state of a neuron and 0 for the off state. Show that w_{ji} can be converted to w'_{ji} by using the transformation:

$$w'_{ji} = \frac{w_{ji}}{2} \quad \text{for } 0 < i < j$$

$$w'_{j0} = w_{j0} = \frac{1}{2} \sum_{0 < i < j} w_{ji}$$

The last line pertains to bias applied to neuron j .

- 11.15 In a sigmoid belief network we identify the probability $P(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha)$ as a Gibbs distribution, and the probability $P(\mathbf{X}_\alpha = \mathbf{x}_\alpha)$ as the corresponding partition function. Justify the validity of this twofold identification.

Helmholtz Machine

- 11.16 The Helmholtz machine is free of feedback in both its recognition and generative models. What would happen to the operation of this network if the use of feedback is permitted in either of these two models?

Deterministic Boltzmann Machine

- 11.17 The Boltzmann machine performs gradient descent (in weight space) on the probability space, as discussed in Problem 11.10. On what function does the deterministic Boltzmann machine perform its gradient descent? You may refer to Hinton (1989) for a discussion of this issue.
 11.18 Consider a recurrent network that is asymmetric in that $w_{ji} \neq w_{ij}$. Show that the deterministic Boltzmann learning algorithm will automatically symmetrize the network provided that, after each weight update, each weight is decayed toward zero by a small amount proportional to its magnitude (Hinton, 1989).

Deterministic Sigmoid Belief Network

- 11.19 Show that the difference between the expressions on the left- and right-hand sides of Eq. (11.77) is equal to the Kullback–Leibler divergence between the distributions $Q(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha)$ and $P(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha)$.
 11.20 The argument of the sigmoid function in Eq. (11.89) defines the induced local field v_j of neuron j in the deterministic sigmoid belief network. In what ways does v_j differ from the corresponding induced local field of a neuron in a multilayer perceptron trained with the back-propagation algorithm?

Deterministic Annealing

- 11.21 In Section 11.13 we developed the idea of deterministic annealing using an information-theoretic approach. The idea of deterministic annealing may also be developed in a principled manner using the maximum entropy principle that is discussed in Chapter 10. Follow through the rationale of this second approach (Rose, 1998).
 11.22 (a) Using Eqs. (11.97) and (11.98), derive the result given in Eq. (11.99) that defines the Lagrangian F^* that results from use of the Gibbs distribution for the association probability.
 (b) Using the result from part a of this problem, derive the condition given in Eq. (11.101) for the minimum of F^* with respect to the code vectors $\mathbf{y}$.
 (c) Apply the minimizing condition of Eq. (11.101) to the squared distortion measure of Eq. (11.91) and comment on your result.
 11.23 Consider a data set that is a mixture of Gaussian distributions. In what way does the use of deterministic annealing offer an advantage over maximum likelihood estimation in such a situation?

11.24 In this problem we explore the use of deterministic annealing for pattern classification using a neural network (Miller et al., 1996). The output of neuron j in the output layer is denoted by $F_j(\mathbf{x})$ where $\mathbf{x}$ is the input vector. The classification decision is based on the maximum discriminant $F_j(\mathbf{x})$.

(a) For a probabilistic objective function, consider the following:

$$F = \frac{1}{N} \sum_{(\mathbf{x}, c) \in \mathcal{T}} \sum_j P(\mathbf{x} \in \mathcal{R}_j) F_j(\mathbf{x})$$

where $\mathcal{T}$ is a training set of labeled vectors with $\mathbf{x}$ denoting an input vector and c its class label, and $P(\mathbf{x} \in \mathcal{R}_j)$ is the probability of association between input vector $\mathbf{x}$ and class region $\mathcal{R}_j$. Using the maximum entropy principle that is discussed in Chapter 10, formulate the Gibbs distribution for $P(\mathbf{x} \in \mathcal{R}_j)$.

(b) Let $\langle P_e \rangle$ denote the average misclassification cost. Formulate the Lagrangian for minimization of $\langle P_e \rangle$ subject to the constraint that the entropy corresponding to the association probabilities $P(\mathbf{x} \in \mathcal{R}_j)$ is equal to some constant value H .

CHAPTER 12

Neurodynamic Programming

12.1 INTRODUCTION

In Chapter 2 we identify two main paradigms of learning: learning with a teacher, and learning without a teacher. The paradigm of learning without a teacher is subdivided into self-organized (unsupervised) learning and reinforcement learning. Different forms of learning with a teacher or supervised learning are covered in Chapters 4 through 7, and different forms of unsupervised learning are discussed in Chapters 8 through 11. In this chapter we discuss *reinforcement learning*.

Supervised learning is a “cognitive” learning problem performed under the tutelage of a teacher: It relies on the availability of an adequate set of input–output examples that are representative of the operating environment. In contrast, reinforcement learning is a “behavioral” learning problem: It is performed through *interaction* between the learning system and its environment, in which the system seeks to achieve a specific goal despite the presence of uncertainties (Barto et al., 1983; Sutton and Barto, 1998). The fact that this interaction is performed without a teacher makes reinforcement learning particularly attractive for dynamic situations where it is costly or difficult (if not impossible) to gather a satisfactory set of input–output examples.

There are two approaches to the study of reinforcement learning,¹ summarized as follows:

1. *Classical approach*, in which learning takes place through a process of punishment and reward with the goal of achieving a *highly skilled behavior*.
2. *Modern approach*, which builds on a mathematical technique known as dynamic programming to decide on a course of action by considering possible future stages without actually experiencing them; the emphasis here is on *planning*.

Our discussion focuses on modern reinforcement learning.

*Dynamic programming*² is a technique that deals with situations where decisions are made in stages, with the outcome of each decision being predictable to some extent before the next decision is made. A key aspect of such situations is that decisions cannot be made in isolation. Rather, the desire for a low cost at the present must be balanced against the undesirability of high costs in the future. This is a *credit assignment problem* because credit or blame must be assigned to each one of a set of interacting

decisions. For optimal planning it is necessary to have an efficient tradeoff between immediate and future costs. Such a tradeoff is indeed captured by the formalism of dynamic programming. In particular, dynamic programming addresses the question: How can a system learn to improve long-term performance when this may require sacrificing short-term performance?

Following Bertsekas and Tsitsiklis (1996), we refer to the modern approach to reinforcement learning as *neurodynamic programming*. We do so primarily for two reasons:

- The theoretical foundation is provided by dynamic programming.
- The learning capability is provided by neural networks.

A succinct definition of neurodynamic programming is (Bertsekas and Tsitsiklis, 1996):

Neurodynamic programming enables a system to learn how to make good decisions by observing its own behavior, and to improve its actions by using a built-in mechanism through reinforcement.

The observation of behavior is attained through the use of Monte Carlo simulation in an off-line manner. The improvement of actions through reinforcement is attained through the use of an iterative optimization scheme.

Organization of the Chapter

Dynamic programming has two main features: an underlying discrete-time dynamic system, and a cost function that is additive over time. These two features are discussed in Section 12.2. This is followed by a formulation of Bellman's optimality equation in Section 12.3, which plays an important role in dynamic programming. In Sections 12.4 and 12.5 we discuss two different methods for computing an optimal policy for dynamic programming, namely policy iteration and value iteration.

In Section 12.6 we present an overview of the issues involved in neurodynamic programming. This overview leads to the discussion of approximate policy iteration and Q-learning, which lend themselves to the use of neural networks for function approximation. These two algorithms are discussed in Sections 12.7 and 12.8, respectively. Section 12.9 presents a computer experiment on the use of Q-learning.

The chapter concludes with some final remarks in Section 12.10.

12.2 MARKOVIAN DECISION PROCESS

Consider a *learning system* or *agent* that interacts with its environment in the manner illustrated in Fig. 12.1. The system operates in accordance with a *finite, discrete-time Markovian decision process* that is characterized as follows:

- The environment evolves probabilistically occupying a finite set of discrete states. Note, however, that the state does *not* contain past statistics, even though these statistics could be useful to the learning system.
- For each environmental state there is a finite set of possible actions that may be taken by the learning system.

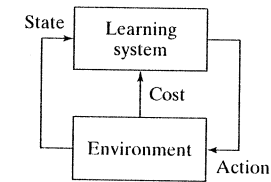


FIGURE 12.1 Block diagram of a learning system interacting with its environment.

- Every time the learning system takes an action, a certain cost is incurred.
- States are observed, actions are taken, and costs are incurred at discrete times.

In the context of our present discussion, the *state* of the environment is defined as: *a summary of the entire past experience of the learning system gained from its interaction with the environment, such that the information necessary for the learning system to predict future behavior of the environment is contained in that summary*. The random variable denoting the state at time step n is X_n , and the actual state at time step n is denoted by $x(n)$. The finite set of states is denoted by $\mathcal{X}$. A surprising aspect of dynamic programming is that its applicability depends very little on the nature of the state. We may therefore proceed without any assumption on the structure of the state space.

For state i , for example, the available set of *actions* (i.e., inputs applied to the environment by the learning system) is denoted by $\mathcal{A}_i = \{a_{ik}\}$, where the second subscript k in action a_{ik} taken by the learning system merely indicates the availability of more than one possible action when the environment is in state i . The transition of the environment from the state i to the new state j , for example, due to action a_{ik} is probabilistic in nature. Most importantly, however, *the transition probability from state i to state j depends entirely on the current state i and the corresponding action a_{ik}* . This is the *Markov property*, which is discussed in Chapter 11. This property is crucial because it means that the current state of the environment provides the necessary information for the learning system to decide what action to take.

The random variable denoting the action taken by the learning system at time step n is denoted by A_n . Let $p_{ij}(a)$ denote the transition probability from state i to state j due to action taken at time step n , where $A_n = a$. By virtue of the Markov property we have

$$p_{ij}(a) = P(X_{n+1} = j \mid X_n = i, A_n = a) \quad (12.1)$$

The transition probability $p_{ij}(a)$ satisfies two conditions that are imposed on it by probability theory:

$$1. \quad p_{ij}(a) \geq 0 \quad \text{for all } i \text{ and } j \quad (12.2)$$

$$2. \quad \sum_j p_{ij}(a) = 1 \quad \text{for all } i \quad (12.3)$$

For a given number of states and given transition probabilities the sequence of environmental states resulting from the actions taken by the learning system over time forms a *Markov chain*. Markov chains are discussed in Chapter 11.

At each transition from one state to another, a *cost* is incurred by the learning system. Thus, at the n th transition from state i to state j under action a_{ik} , the learning