Contents

Chapter 7

Stochastic Methods

7.1 Introduction

Learning plays a central role in the construction of pattern classifiers. As we have seen, the general approach is to specify a model having one or more parameters and then estimate their values from training data. When the models are fairly simple and of low dimension, we can use analytic methods such as computing derivatives and performing gradient descent to find optimal model parameters. If the models are somewhat more complicated, we may calculate local derivatives and use gradient methods, as in neural networks and some maximum-likelihood problems. In most high-dimensional and complicated models, there are multiple maxima and we must use a variety of tricks — such as performing the search multiple times from different starting conditions — to have any confidence that an acceptable local maximum has been found.

These methods become increasingly unsatisfactory as the models become more complex. A naive approach — exhaustive search through solution space — rapidly gets out of hand and is completely impractical for real-world problems. The more complicated the model, the less the prior knowledge, and the less the training data, the more we must rely on sophisticated search for finding acceptable model parameters. In this chapter we consider stochastic methods for finding parameters, where randomness plays a crucial role in search and learning. The general approach is to bias the search toward regions where we expect the solution to be and allow randomness — somehow — to help find good parameters, even in very complicated models.

We shall consider two general classes of such methods. The first, exemplified by Boltzmann learning, is based on concepts and techniques from physics, specifically statistical mechanics. The second, exemplified by genetic algorithms, is based on concepts from biology, specifically the mathematical theory of evolution. The former class has a highly developed and rigorous theory and many successes in pattern recognition; hence it will command most of our effort. The latter class is more heuristic yet affords flexibility and can be attractive when adequate computational resources are available. We shall generally illustrate these techniques in cases that are simple, and which might also be addressed with standard gradient procedures; nevertheless we emphasize that these stochastic methods may be preferable in complex problems.

The methods have high computational burden, and would be of little use without computers.

7.2 Stochastic search

We begin by discussing an important and general quadratic optimization problem. Analytic approaches do not scale well to large problems, however, and thus we focus here on methods of *search* through different candidate solutions. We then consider a form of stochastic search that finds use in learning for pattern recognition.

Suppose we have a large number of variables s_i, $i = 1, \ldots, N$ where each variable can take one of two discrete values, for simplicity chosen to be ± 1. The optimization problem is this: find the values of the s_i so as to minimize the cost or *energy*

ENERGY

$$E = -\frac{1}{2} \sum_{i,j=1}^{N} w_{ij} s_i s_j, \tag{1}$$

where the w_{ij} can be positive or negative and are problem dependent. We require the self-feedback terms to vanish, i.e., $w_{ii} = 0$, since non-zero w_{ii} merely add an unimportant constant to E, independent of the s_i. This optimization problem can be visualized in terms of a network of nodes, where bi-directional links or interconnections correspond to the weights $w_{ij} = w_{ji}$. (It is very simple to prove that we can always replace a non-symmetric connection matrix by its symmetric part, as asked in Problem 2. We avoid non-symmetric matrices because they unnecessarily complicate the dynamics described in Sect. 7.2.1.) Figure 7.1 shows such a network, where nodes are labeled input, output, and hidden, though for the moment we shall ignore such distinctions.

This network suggests a physical analogy which in turn will guide our choice of solution method. Imagine the network represents N physical magnets, each of which can have its north pole pointing up ($s_i = +1$) or pointing down ($s_i = -1$). The w_{ij} are functions of the physical separations between the magnets. Each pair of magnets has an associated interaction energy which depends upon their state, separation and other physical properties: $E_{ij} = -1/2\ w_{ij} s_i s_j$. The energy of the full system is the sum of all these interaction energies, as given in Eq. 1. The optimization task is to find the configuration of states of the magnets with the most stable configuration, the one with lowest energy. This general optimization problem appears in a wide range of applications, in many of which the weights do not have a physical interpretation.* As mentioned, we shall be particularly interested in its application to learning methods.

Except for very small problems or few connections, it is infeasible to solve directly for the N values s_i that give the minimum energy — the space has 2^N possible configurations (Problem 4). It is tempting to propose a greedy algorithm to search for the optimum configuration: Begin by randomly assigning a state to each node. Next consider each node in turn and calculate the energy with it in the $s_i = +1$ state and then in the $s_i = -1$ state, and choose the one giving the lower energy. (Naturally, this decision needs to be based on only those nodes connected to node i with non-zero weight w_{ij}.) Alas, this greedy search is rarely successful since the system usually gets caught in local energy minima or never converges (Computer exercise 1).

Another method is required.

* Similar generalized energy functions, called *Lyapunov functions* or *objective functions*, can be used for finding optimum states in other problem domains as well.

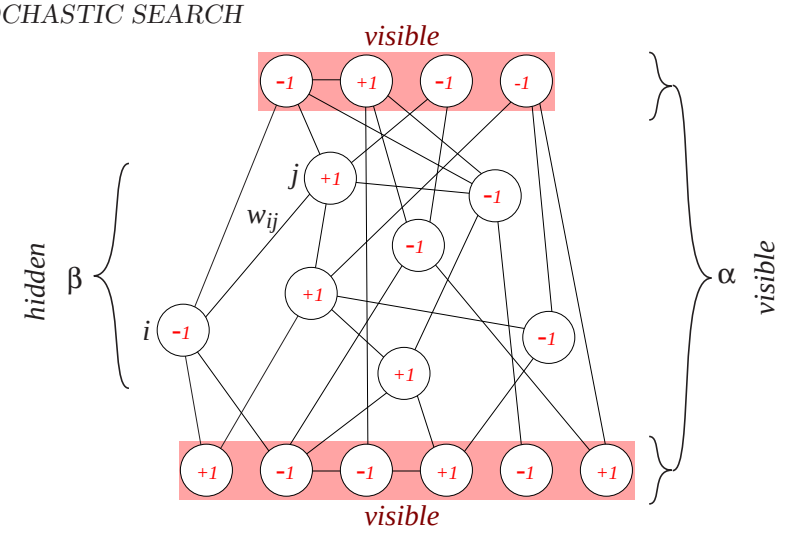

Figure 7.1: The class of optimization problems of Eq. 1 can be viewed in terms of a network of nodes or units, each of which can be in the $s_i = +1$ or $s_i = -1$ state. Every pair of nodes i and j is connected by bi-directional weights w_{ij}; if a weight between two nodes is zero then no connection is drawn. (Because the networks we shall discuss can have an arbitrary interconnection, there is no notion of layers as in multilayer neural networks.) The optimization problem is to find a configuration (i.e., assignment of all s_i) that minimizes the energy described by Eq. 1. The state of the full network is indexed by an integer γ, and since here there are 17 binary nodes, γ is bounded $0 \leq \gamma < 2^{17}$. The state of the visible nodes and hidden nodes are indexed by α and β, respectively and in this case are bounded $0 \leq \alpha \leq 2^{10}$ and $0 \leq \beta < 2^7$.

7.2.1 Simulated annealing

In physics, the method for allowing a system such as many magnets or atoms in an alloy to find a low-energy configuration is based on *annealing*. In physical annealing the system is heated, thereby conferring randomness to each component (magnet). As a result, each variable can temporarily assume a value that is energetically *un*favorable and the full system explores configurations that have high energy. Annealing proceeds by gradually lowering the temperature of the system — ultimately toward zero and thus no randomness — so as to allow the system to relax into a low-energy configuration. Such annealing is effective because even at moderately high temperatures, the system slightly favors regions in the configuration space that are overall lower in energy, and hence are more likely to contain the global minimum. As the temperature is lowered, the system has increased probability of finding the optimum configuration. This method is successful in a wide range of energy functions or energy "landscapes," though there are pathological cases such as the "golf course" landscape in Fig. 7.2 where it is unlikely to succeed. Fortunately, the problems in learning we shall consider rarely involve such pathological functions.

ANNEALING

7.2.2 The Boltzmann factor

The statistical properties of large number of interacting physical components at a temperature T, such as molecules in a gas or magnetic atoms in a solid, have been thoroughly analyzed. A key result, which relies on a few very natural assumptions, is

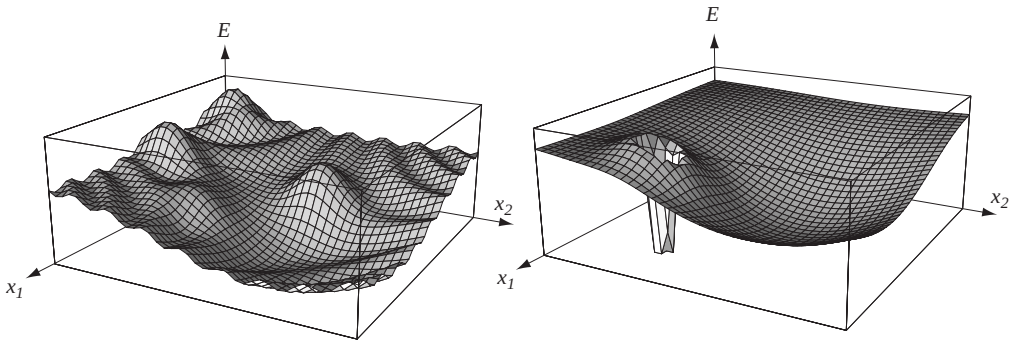

Figure 7.2: The energy function or energy "landscape" on the left is meant to suggest the types of optimization problems addressed by simulated annealing. The method uses randomness, governed by a control parameter or "temperature" T to avoid getting stuck in local energy minima and thus to find the global minimum, like a small ball rolling in the landscape as it is shaken. The pathological "golf course" landscape at the right is, generally speaking, not amenable to solution via simulated annealing because the region of lowest energy is so small and is surrounded by energetically unfavorable configurations. The configuration space of the problems we shall address are discrete and thus the continuous $x_1 - x_2$ space shown here is a bit misleading.

the following: the probability the system is in a (discrete) configuration indexed by γ having energy E_γ is given by

$$P(\gamma) = \frac{e^{-E_\gamma/T}}{Z(T)}, \tag{2}$$

BOLTZMANN FACTOR

where Z is a normalization constant. The numerator is the *Boltzmann factor* and the denominator the *partition function*, the sum over all possible configurations

PARTITION FUNCTION

$$Z(T) = \sum_{\gamma'} e^{-E_{\gamma'}/T}, \tag{3}$$

which guarantees Eq. 2 represents a true probability.* The number of configurations is very high, 2^N, and in physical systems Z can be calculated only in simple cases. Fortunately, we need not calculate the partition function, as we shall see.

Because of the fundamental importance of the Boltzmann factor in our discussions, it pays to take a slight detour to understand it, at least in an informal way. Consider a different, but nontheless related system: one consisting of a large number of *non-interacting* magnets, that is, without interconnecting weights, in a uniform external magnetic field. If a magnet is pointing up, $s_i = +1$ (in the same direction as the field), it contributes a small positive energy to the total system; if the magnet is pointing down, a small negative energy. The total energy of the collection is thus proportional to the total number of magnets pointing up. The *probability* the system has a particular total energy is related to the number of configurations that have that energy. Consider the highest energy configuration, with all magnets pointing up. There is only $\binom{N}{N} = 1$ configuration that has this energy. The next to highest

* In the Boltzmann factor for physical systems there is a "Boltzmann constant" which converts a temperature into an energy; we can ignore this factor by scaling the temperature in our simulations.

energy comes with just a single magnet pointing down; there are $\binom{N}{1} = N$ such configurations. The next lower energy configurations have two magnets pointing down; there are $\binom{N}{2} = N(N-1)/2$ of these configurations, and so on. The number of states declines exponentially with increasing energy. Because of the statistical independence of the magnets, for large N the probability of finding the state in energy E also decays exponentially (Problem 7). In sum, then, the exponential form of the Boltzmann factor in Eq. 2 is due to the exponential decrease in the number of accessible configurations with increasing energy. Further, at high temperature there is, roughly speaking, more energy available and thus an increased probability of higher-energy states. This describes qualitatively the dependence of the probability upon T in the Boltzmann factor — at high T, the probability is distributed roughly evenly among all configurations while at low T, it is concentrated at the lowest-energy configurations.

If we move from the collection of independent magnets to the case of magnets interconnected by weights, the situation is a bit more complicated. Now the energy associated with a magnet pointing up or down depends upon the state of others. Nonetheless, in the case of large N, the number of configurations decays exponentially with the energy of the configuration, as described by the Boltzmann factor of Eq. 2.

Simulated annealing algorithm

The above discussion and the physical analogy suggest the following *simulated annealing* method for finding the optimum configuration to our general optimization problem. Start with randomized states throughout the network, $s_i(1)$, and select a high initial "temperature" $T(1)$. (Of course in the simulation T is merely a control parameter which will control the randomness; it is not a true physical temperature.) Next, choose a node i randomly. Suppose its state is $s_i = +1$. Calculate the system energy in this configuration, E_a; next recalculate the energy, E_b, for a candidate new state $s_i = -1$. If this candidate state has a lower energy, accept this change in state. If however the energy is *higher*, accept this change with a probability equal to

$$e^{-\Delta E_{ab}/T}, \tag{4}$$

where $\Delta E_{ab} = E_b - E_a$. This occasional acceptance of a state that is energetically less favorable is crucial to the success of simulated annealing, and is in marked distinction to naive gradient descent and the greedy approach mentioned above. The key benefit is that it allows the system to jump out of unacceptable local energy minima. For example, at very high temperatures, every configuration has a Boltzmann factor $e^{-E/T} \approx e^0$ roughly equal. After normalization by the partition function, then, every configuration is roughly equally likely. This implies every node is equally likely to be in either of its two states (Problem 6).

The algorithm continues *polling* (selecting and testing) the nodes randomly several times and setting their states in this way. Next lower the temperature and repeat the polling. Now, according to Eq. 4, there will be a slightly smaller probability that a candidate higher energy state will be accepted. Next the algorithm polls all the nodes until each node has been visited several times. Then the temperature is lowered further, the polling repeated, and so forth. At very low temperatures, the probability that an energetically less favorable state will be accepted is small, and thus the search becomes more like a greedy algorithm. Simulated annealing terminates when the temperature is very low (near zero). If this cooling has been sufficiently slow, the system then has a high probability of being in a low energy state — hopefully the global energy minimum.

POLLING

Because it is the *difference* in energies between the two states that determines the acceptance probabilities, we need only consider nodes connected to the one being polled — all the units *not* connected to the polled unit are in the same state and contribute the same total amount to the full energy. We let $\mathcal{N}_i$ denote the set of nodes connected with non-zero weights to node i; in a fully connected net would include the complete set of $N-1$ remaining nodes. Further, we let Rand$[0,1)$ denote a randomly selected positive real number less than 1. With this notation, then, the randomized or stochastic simulated annealing algorithm is:

Algorithm 1 (Stochastic simulated annealing)

```
1  begin initialize T(k), k_max, s_i(1), w_ij for i, j = 1, ..., N
2        k ← 0
3        do  k ← k + 1
4            do  select node i randomly; suppose its state is s_i
5                E_a ← −1/2 Σ_j^{N_i} w_ij s_i s_j
6                E_b ← −E_a
7                if  E_b < E_a
8                  then  s_i ← −s_i
9                  else  if  e^{−(E_b−E_a)/T(k)} > Rand[0, 1)
10                           then  s_i ← −s_i
11                 until all nodes polled several times
12           until k = k_max or stopping criterion met
13           return E, s_i, for i = 1, ..., N
14       end
```

Because units are polled one at a time, the algorithm is occasionally called sequential simulated annealing. Note that in line 5, we define E_a based only on those units connected to the polled one — a slightly different convention than in Eq. 1. Changing the usage in this way has no effect, since in line 9 it is the *difference* in energies that determines transition probabilities.

There are several aspects of the algorithm that must be considered carefully, in particular the starting temperature, ending temperature, the rate at which the temperature is decreased and the stopping criterion. This function is called the *cooling schedule* or more frequently the *annealing schedule*, $T(k)$, where k is an iteration index. We demand $T(1)$ to be sufficiently high that all configurations have roughly equal probability. This demands the temperature be larger than the maximum difference in energy between any configurations. Such a high temperature allows the system to move to any configuration which may be needed, since the random initial configuration may be far from the optimal. The decrease in temperature must be both gradual and slow enough that the system can move to any part of the state space before being trapped in an unacceptable local minimum, points we shall consider below. At the very least, annealing must allow $N/2$ transitions, since a global optimum never differs from any configuration by more than this number of steps. (In practice, annealing can require polling several orders of magnitude more times than this number.) The final temperature must be low enough (or equivalently k_{max} must be large enough or a stopping criterion must be good enough) that there is a negligible probability that if the system is in a global minimum it will move out.

ANNEALING SCHEDULE

Figure 7.3 shows that early in the annealling process when the temperature is high, the system explores a wide range of configurations. Later, as the temperature

is lowered, only states "close" to the global minimum are tested. Throughout the process, each transition corresponds to the change in state of a single unit.

A typical choice of annealing schedule is $T(k+1) = cT(k)$ with $0 < c < 1$. If computational resources are of no concern, a high initial temperature, large $c < 1$, and large k_{max} are most desirable. Values in the range $0.8 < c < 0.99$ have been found to work well in many real-world problems. In practice the algorithm is slow, requiring many iterations and many passes through all the nodes, though for all but the smallest problems it is still faster than exhaustive search (Problem 5). We shall revisit the issue of parameter setting in the context of learning in Sect. 7.3.4.

While Fig. 7.3 displayed a single trajectory through the configuration space, a more relevant property is the *probability* of being in a configuration as the system is annealed gradually. Figure 7.4 shows such probability distributions at four temperatures. Note especially that at the final, low temperature the probability is concentrated at the global minima, as desired. While this figure shows that for positive temperature all states have a non-zero probability of being visited, we must recognize that only a small fraction of configurations are in fact visited in any anneal. In short, in the vast majority of large problems, annealing does not require that all configurations be explored, and hence it is more efficient than exhaustive search.

7.2.3 Deterministic simulated annealing

Stochastic simulated annealing is slow, in part because of the discrete nature of the search through the space of all configurations, i.e., an N-dimensional hypercube. Each trajectory is along a *single* edge, thereby missing full gradient information that would be provided by analog state values in the "interior" of the hypercube. An alternate, faster method is to allow each node to take on *analog* values during search; at the end of the search the values are forced to be $s_i = \pm 1$, as required by the optimization problem. Such a *deterministic* simulated annealing algorithm also follows from the physical analogy. Consider a single node (magnet) i connected to several others; each exerts a force tending to point node i up or down. In deterministic annealing we sum the forces and give a continuous value for s_i. If there is a large "positive" force, then $s_i \approx +1$; if a large negative force, then $s_i \approx -1$. In the general case s_i will lie between these limits.

The value of s_i also depends upon the temperature. At high T (large randomness) even a large upward force will not be enough to insure $s_i = +1$, whereas at low temperature it will. We let $l_i = \sum_j w_{ij} s_j$ be the force exerted on node i, the updated value is:

$$s_i = f(l_i, T) = \tanh[l_i/T], \tag{5}$$

where there is an implied scaling of the force and temperature in the *response function* $f(\cdot,\cdot)$ (Fig. 7.5). In broad overview, deterministic annealing consists in setting an annealing schedule and then at each temperature finding an equilibrium analog value for every s_i. This analog value is merely the expected value of the discrete s_i in a system at temperature T (Problem 8). At low temperatures (i.e., at the end of the anneal), each variable will assume an extreme value ± 1, as can be seen in the low-T curve in Fig. 7.5.

RESPONSE FUNCTION

It is instructive to consider the energy landscape for the continuous case. Differentiation of Eq. 1 shows that the energy is linear in each variable when others held fixed, as can be seen in Fig. 7.6 — there are no local minima along any "cut" parallel

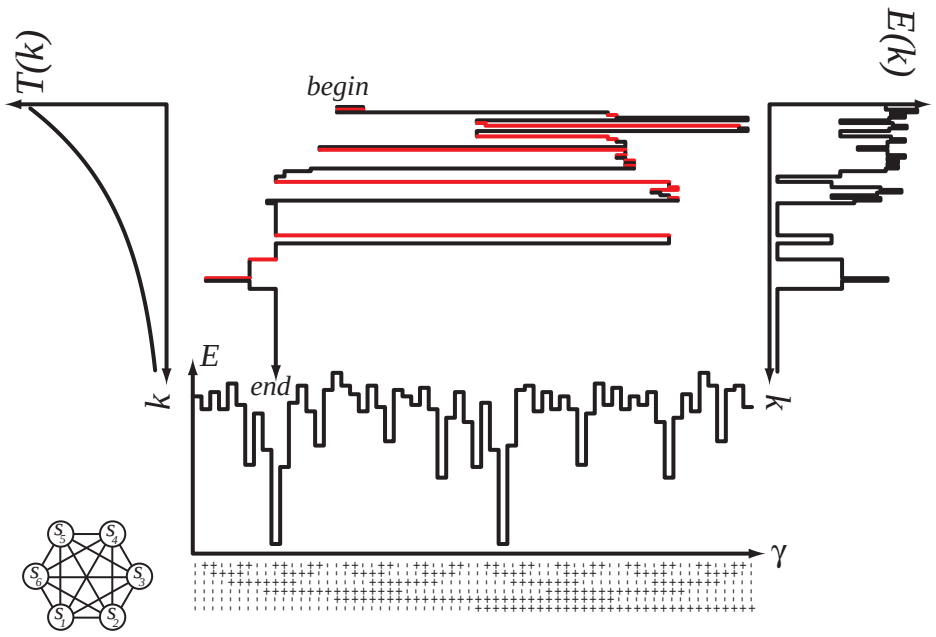

Figure 7.3: Stochastic simulated annealing (Algorithm 1) uses randomness, governed by a control parameter or "temperature" $T(k)$ to search through a discrete space for a minimum of an energy function. In this example there are $N = 6$ variables; the $2^6 = 64$ configurations are shown at the bottom along as a column of + and -. The plot of the associated energy of each configuration given by Eq. 1 for randomly chosen weights. Every transition corresponds to the change of just a single s_i. (The configurations have been arranged so that adjacent ones differ by the state of just a single node; nevertheless most transitions corresponding to a single node appear far apart in this ordering.) Because the system energy is invariant with respect to a global interchange $s_i \leftrightarrow -s_i$, there are two "global" minima. The graph at the upper left shows the annealing schedule — the decreasing temperature versus iteration number k. The middle portion shows the configuration versus iteration number generated by Algorithm 1. The trajectory through the configuration space is colored red for transitions that increase the energy; late in the annealing such energetically unfavorable (red) transitions are rarer. The graph at the right shows the full energy $E(k)$, which decreases to the global minimum.

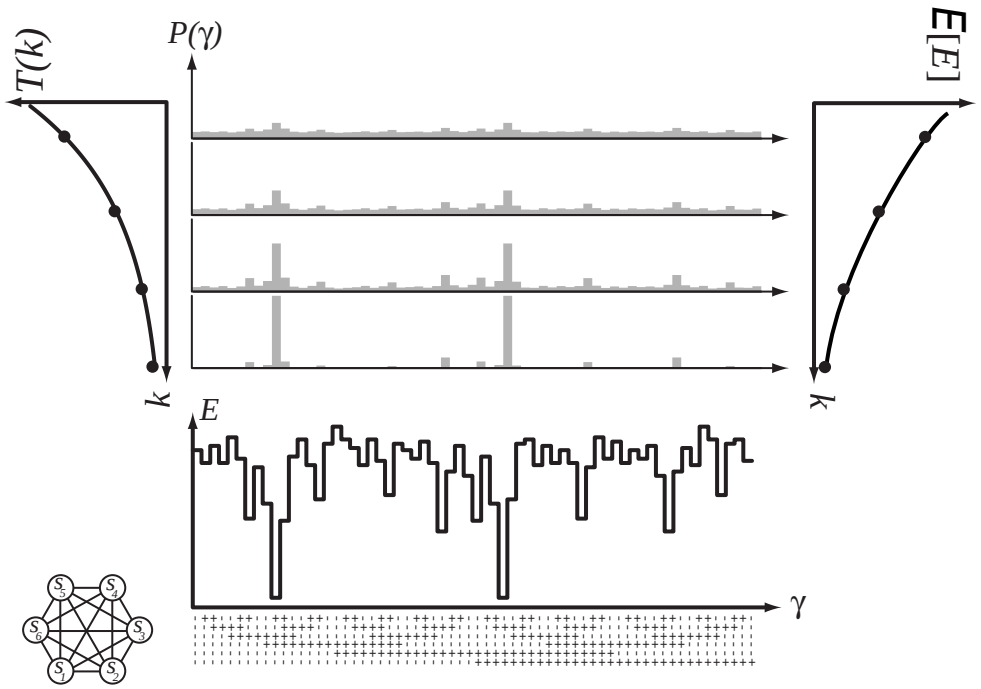

Figure 7.4: An estimate of the probability $P(\gamma)$ of being in a configuration denoted by γ is shown for four temperatures during a slow anneal. (These estimates, based on a large number of runs, are nearly the theoretical values e^{-E_γ}/T) Early, at high T, each configuration is roughly equal in probability while late, at low T, the probability is strongly concentrated at the global minima. The expected value of the energy, $\mathcal{E}[E]$ (i.e., averaged at temperature T), decreases gradually during the anneal.

to any axis. Note too that there are no stable local energy minima within the volume of the space; the energy minima always occur at the "corners," i.e., extreme $s_i = \pm 1$ for all i, as required by the optimization problem.

This search method is sometimes called *mean-field* annealing because each node responds to the average or mean of the forces (fields) due to the nodes connected to it. In essence the method approximates the effects of all other magnets while ignoring their mutual interactions and their response to the magnet in question, node i. Such annealing is also called *deterministic* because in principle we could deterministically solve the simultaneous equations governing the s_i as the temperature is lowered. The algorithm has a natural parallel mode of implementation, for instance where each value s_i is updated simultaneously and deterministically as the temperature is lowered. In and inherently serial simulation, however, the nodes are updated one at a time. Even though the nodes might be polled pseudo randomly, the algorithm is in principle deterministic — there need be no inherent randomness in the searchn. If we let $s_i(1)$ denote the initial state of unit i, the algorithm is:

Algorithm 2 (Deterministic simulated annealing)

1 **begin** **initialize** $T(k), w_{ij}, s_i(1), i, j = 1, \ldots N$
2 $k \leftarrow 0$
3 **do** $k \leftarrow k + 1$

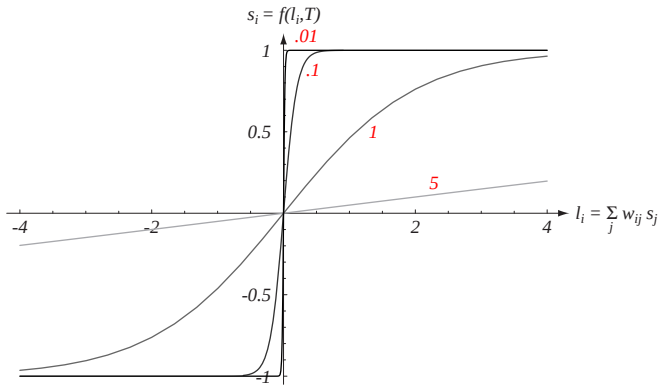

Figure 7.5: In deterministic annealing, each node can take on a continuous value $-1 \leq s_i \leq +1$, which equals the expected value of a binary node in the system at temperature T. In other words, the analog value s_i replaces the expectation of the discrete variable, $\mathcal{E}[s_i]$. We let l_i denote a force exerted by the nodes connected to s_i. The larger this force, the closer the analog s_i is to $+1$; the more negative this force, the closer to -1. The temperature T (marked in red) also affects s_i. If T is large, there is a great deal of randomness and even a large force will not insure $s_i \approx +1$. At low temperature, there is little or no randomness and even a small positive force insures that $s_i = +1$. Thus at the end of an anneal, each node has value $s_i = +1$ or $s_i = -1$.

```
4          Select node i randomly
5          l_i ← Σ_j^{N_i} w_ij s_j
6          s_i ← f(l_i, T(k))
7        until k = k_max or convergence criterion met
8    return E, s_i, i = 1, ..., N
9  end
```

In practice, deterministic and stochastic annealing give very similar solutions. In large real-world problems deterministic annealing is faster, sometimes by two or three orders of magnitude.

Simulated annealing can also be applied to other classes of optimization problem, for instance, finding the minimum in $\sum_{ijk} w_{ijk} s_i s_j s_k$. We will not consider such higher-order problems, though they can be the basis of learning methods as well.

7.3 Boltzmann learning

For pattern recognition, we will use a network such as that in Fig. 7.1, where the input units accept binary feature information and the output units represent the categories, generally in the familiar 1-of-c representation (Fig. 7.7). During classification the input units are held fixed or *clamped* to the feature values of the input pattern; the remaining units are annealed to find the lowest energy, most probable configuration. The category information is then read from the final values of the output units. Of course, accurate recognition requires proper weights, and thus we now turn to a

CLAMP

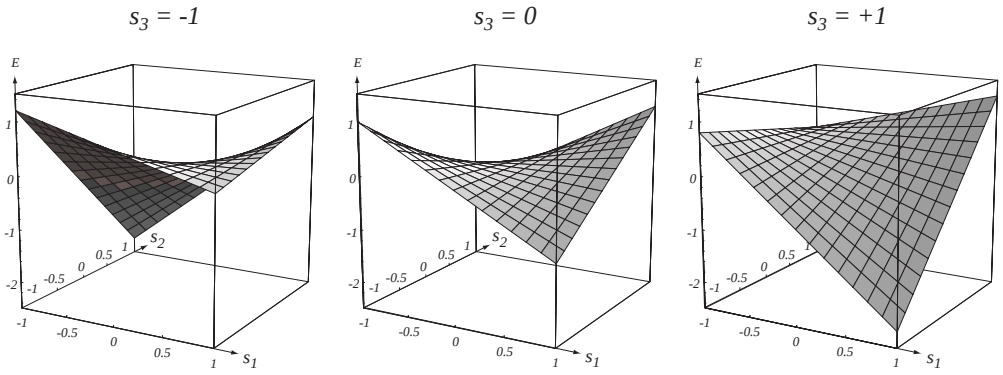

Figure 7.6: If the state variables s_i can assume analog values (as in mean-field annealing), the energy in Eq. 1 is a general quadratic form having minima at the extreme values $s_i = \pm 1$. In this case $N = 3$ nodes are fully interconnected with arbitrary weights w_{ij}. While the total energy function is three-dimensional, we show two-dimensional surfaces for each of three values of s_3. The energy is linear in each variable so long as the other variables are held fixed. Further, the energy is invariant with respect to the interchange of all variables $s_i \leftrightarrow -s_i$. In particular, here the global minimum occurs as $s_1 = -1$, $s_2 = +1$ and $s_3 = -1$ as well as the symmetric configuration $s_1 = +1$, $s_2 = -1$ and $s_3 = +1$.

method for learning weights from training patterns. There are two closely related approaches to such learning, one based on stochastic and the other on deterministic simulated annealing.

7.3.1 Stochastic Boltzmann learning of visible states

Before we turn to our central concern — learning categories from training patterns — consider an alternate learning problem where we have a set of desired probabilities for *all* the visible units, $Q(\alpha)$ (given by a training set), and seek weights so that the actual probability $P(\alpha)$, achieved in random simulations, matches these probabilities over all patterns as closely as possible. In this alternative learning problem the desired probabilities would be derived from training patterns containing both input (feature) and output (category) information. The actual probability describes the states of a network annealed with neither input nor output variables clamped.

We now make use of the distinction between configurations of "visible" units (the input and output, denoted α), and the hidden states, denoted β, shown in Fig. 7.1. For instance, whereas a and b (c.f., Eq. 4) refered to different configurations of the full system, α and β sill specify visible and hidden configurations.

The probability of a visible configuration is the sum over all possible hidden configurations:

$$\begin{aligned} P(\alpha) &= \sum_{\beta} P(\alpha, \beta) \\ &= \frac{\sum_{\beta} e^{-E_{\alpha\beta}/T}}{Z} \end{aligned} \tag{6}$$

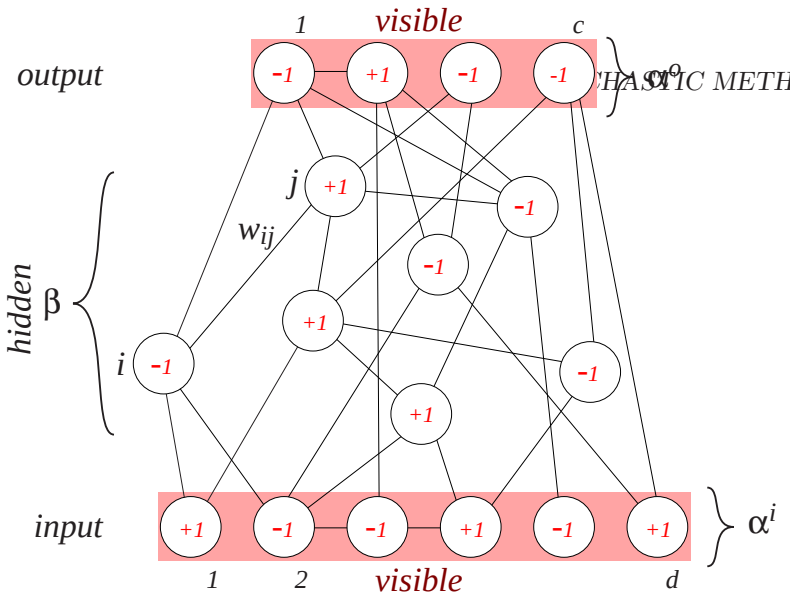

Figure 7.7: When a network such as shown in Fig. 7.1 is used for learning, it is important to distinguish between two types of visible units — the d input units and c output units, which receive external feature and category information — as well as the remaining, hidden units. The state of the full network is indexed by an integer γ, and since here there are 17 binary nodes, γ is bounded $0 \leq \gamma < 2^{17}$. The state of the visible nodes is described by α; moreover, α^i describes the input and α^o the output (the superscripts are not indexes, but merely refer to the input and output, respectively). The state of the hidden nodes is indexed by β.

where $E_{\alpha\beta}$ is the system energy in the configuration defined by the visible and hidden parts, and Z is again the full partition function. Equation 6 is based on Eq. 3 and states simply that to find the probability of a given visible state α, we sum over all possible hidden states. A natural measure of the difference between the actual and the desired probability distributions is the relative entropy, Kullback-Leibler distance or Kullback-Leibler divergence,

$$D_{KL}(Q(\alpha), P(\alpha)) = \sum_{\alpha} Q(\alpha) \log \frac{Q(\alpha)}{P(\alpha)}. \tag{7}$$

Naturally, D_{KL} is non-negative and can be zero if and only if $P(\alpha) = Q(\alpha)$ for all α (Appendix **??**). Note that Eq. 6 depends solely upon the visible units, not the hidden units.

Learning is based on gradient descent in the relative entropy. A set of training patterns defines $Q(\alpha)$, and we seek weights so that at some temperature T the actual distribution $P(\alpha)$ matches $Q(\alpha)$ as closely as possible. Thus we take an untrained network and update each weight according to:

$$\Delta w_{ij} = -\eta \frac{\partial D_{KL}}{\partial w_{ij}} = \eta \sum_{\alpha} \frac{Q(\alpha)}{P(\alpha)} \frac{\partial P(\alpha)}{\partial w_{ij}}, \tag{8}$$

where η is a learning rate. While P depends on the weights, Q does not, and thus we used $\partial Q(\alpha)/\partial w_{ij} = 0$. We take the derivative in Eq. 6 and find:

$$\frac{\partial P(\alpha)}{\partial w_{ij}} = \frac{\sum_\beta e^{-E_{\alpha\beta}} s_i(\alpha\beta) s_j(\alpha\beta)/T}{TZ} - \frac{\left(\sum_\beta e^{-E_{\alpha\beta}/T}\right) \sum_{\lambda\mu} e^{E_{\lambda\mu}} s_i(\lambda\mu) s_j(\lambda\mu)}{TZ^2}$$

$$= \frac{1}{T}\left[\sum_\beta s_i(\alpha\beta) s_j(\alpha\beta) P(\alpha,\beta) - P(\alpha)\mathcal{E}[s_i s_j]\right]. \tag{9}$$

Here $s_i(\alpha\beta)$ is the state of node i in the full configuration specified by α and β. Of course, if node i is a visible one, then only the value of α is relevant; if the node is a hidden one, then only the value of β is relevant. (Our notation unifies these two cases.) The expectation value $\mathcal{E}[s_i s_j]$ is taken at temperature T. We gather terms and find from Eqs. 8 & 9

$$\Delta w_{ij} = \frac{\eta}{T}\left[\sum_\alpha \frac{Q(\alpha)}{P(\alpha)} \sum_\beta s_i(\alpha\beta) s_j(\alpha\beta) P(\alpha,\beta) - \sum_\alpha Q(\alpha)\mathcal{E}[s_i s_j]\right]$$

$$= \frac{\eta}{T}\left[\sum_{\alpha\beta} Q(\alpha) P(\beta|\alpha) s_i(\alpha\beta) s_j(\alpha\beta) - \mathcal{E}[s_i s_j]\right]$$

$$= \frac{\eta}{T}\left[\underbrace{\mathcal{E}_Q[s_i s_j]_{\alpha\ clamped}}_{learning} - \underbrace{\mathcal{E}[s_i s_j]_{free}}_{unlearning}\right] \tag{10}$$

where $P(\alpha,\beta) = P(\beta|\alpha)P(\alpha)$. We have defined

$$\mathcal{E}_Q[s_i s_j]_{\alpha\ clamped} = \sum_{\alpha\beta} Q(\alpha) P(\beta|\alpha) s_i(\alpha\beta) s_j(\alpha\beta) \tag{11}$$

to be the correlation of the variables s_i and s_j when the visible units are held fixed — clamped — in visible configuration α, averaged according to the probabilities of the training patterns, $Q(\alpha)$.

The first term on the right of Eq. 10 is informally referred to as the *learning component* or teacher component (as the visible units are held to values given by the teacher), and the second term the *unlearning* or student component (where the variables are free to vary). If $\mathcal{E}_Q[s_i s_j]_{\alpha\ clamped} = \mathcal{E}[s_i s_j]_{free}$, then $\Delta w_{ij} = 0$ and we have achieved the desired weights. The unlearning component reduces spurious correlations between units — spurious in that they are not due to the training patterns. A learning algorithm based on the above derivation would present each pattern in the full training set several times and adjust the weights by Eq. 10, just as we saw in numerous other training methods such as backpropagation (Chap. **??**).

LEARNING COMPONENT

UNLEARNING COMPONENT

Stochastic Learning of input-output associations

Now consider the problem of learning mappings from input to output — our real interest in pattern recognition. Here we want the network to learn associations between the (visible) states on the input units, denoted α^i, and states on the output units, denoted α^o, as shown in Fig. 7.1. Formally, we want $P(\alpha^o|\alpha^i)$ to match $Q(\alpha^o|\alpha^i)$

as closely as possible. The appropriate cost function here is the Kullback-Leibler divergence weighted by the probability of each input pattern:

$$\bar{D}_{KL}(Q(\alpha^o|\alpha^i), P(\alpha^o|\alpha^i)) = \sum_{\alpha^i} P(\alpha^i) \sum_{\alpha^o} Q(\alpha^o|\alpha^i) \log \frac{Q(\alpha^o|\alpha^i)}{P(\alpha^o|\alpha^i)}. \tag{12}$$

Just as in Eq. 8, learning involves changing weights to reduce this weighted distance, i.e.,

$$\Delta w_{ij} = -\eta \frac{\partial \bar{D}_{KL}}{\partial w_{ij}}. \tag{13}$$

The derivation of the full learning rule follows closely that leading to Eq. 11; the only difference is that the input units are clamped in both the learning and unlearning components (Problem 11). The result is that the weight update is

$$\Delta w_{ij} = \frac{\eta}{T} \left[\underbrace{\mathcal{E}_Q[s_i s_j]_{\alpha^i \alpha^o \ clamped}}_{learning} - \underbrace{\mathcal{E}[s_i s_j]_{\alpha^i \ clamped}}_{unlearning} \right]. \tag{14}$$

In Sect. 7.3.3 we shall present pseudocode for implementing the preferred, deterministic version of Boltzmann learning, but first we can gain intuition into the general method by considering the learning of a single pattern according to Eq. 14. Figure 7.8 shows a seven-unit network being trained with the input pattern $s_1 = +1$, $s_2 = +1$ and the output pattern $s_6 = -1$, $s_7 = +1$. In a typical 1-of-c representation, this desired output signal would represent category ω_2. Since during both training and classification, the input units s_1 and s_2 are clamped at the value $+1$, we have shown only the associated $2^5 = 32$ configurations at the right. The energy before learning (Eq. 1), corresponding to randomly chosen weights, is shown in black. After the weights are trained by Eq. 14 using the pattern shown, the energy is changed (shown in red). Note particularly that all states having the desired output pattern have their energies lowered through training, just as we need. Thus when these input states are clamped and the remaining networked annealed, the desired output is more likely to be found.

Equation 14 appears a bit different from those we have encountered in pattern recognition, and it is worthwhile explaining it carefully. Figure 7.9 illustrates in greater detail the learning of the single training pattern in Fig. 7.8. Because s_1 and s_2 are clamped throughout, $\mathcal{E}_Q[s_1 s_2]_{\alpha^i \alpha^o clamped} = 1 = \mathcal{E}[s_1 s_2]_{\alpha^i clamped}$, and thus the weight w_{12} is not changed, as indeed given by Eq. 14. Consider a more general case, involving s_1 and s_7. During the learning phase both units are clamped at $+1$ and thus the correlation is $\mathcal{E}_Q[s_1 s_7] = +1$. During the unlearning phase, the output s_7 is free to vary and the correlation is lower; in fact it happens to be negative. Thus, the learning rule seeks to increase the magnitude of w_{17} so that the input $s_1 = +1$ leads to $s_7 = +1$, as can be seen in the matrix on the right. Because hidden units are only weakly correlated (or anticorrelated), the weights linking hidden units are changed only slightly.

In learning a training set of many patterns, each pattern is presented in turn, and the weights updated as just described. Learning ends when the actual output matches the desired output for all patterns (cf. Sect. 7.3.4).

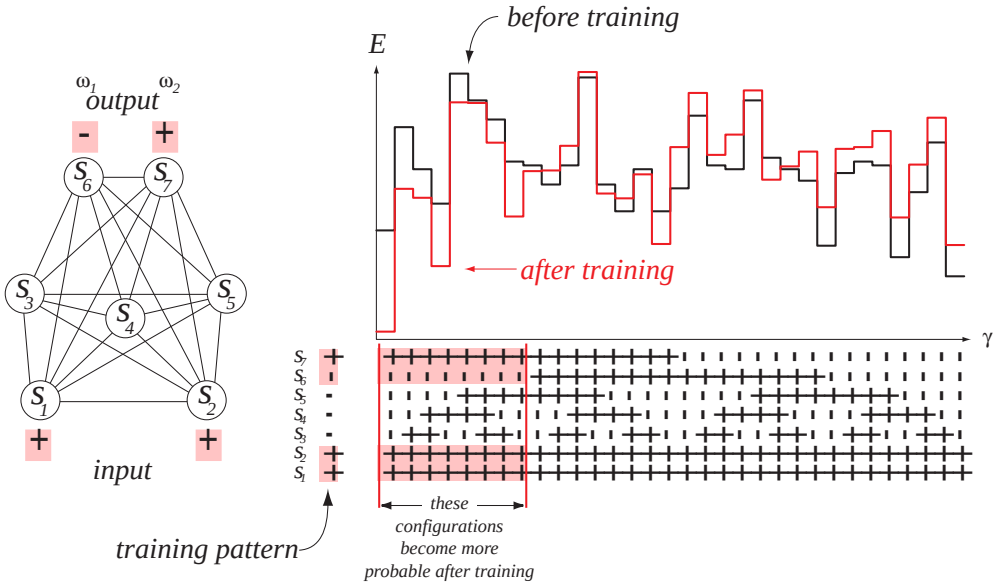

Figure 7.8: The fully connected seven-unit network at the left is being trained via the Boltzmann learning algorithm with the input pattern $s_1 = +1$, $s_2 = +1$, and the output values $s_6 = -1$ and $s_7 = +1$, representing categories ω_1 and ω_2, respectively. All $2^5 = 32$ configurations with $s_1 = +1$, $s_2 = +1$ are shown at the right, along with their energy (Eq. 1). The black curve shows the energy before training; the red curve shows the energy after training. Note particularly that after training all configurations that represent the full training pattern have been lowered in energy, i.e., have become more probable. Consequently, patterns that do not represent the training pattern become *less* probable after training. Thus, after training, if the input pattern $s_1 = +1$, $s_2 = +1$ is presented and the remaining network annealed, there is an increased chance of yielding $s_6 = -1$, $s_7 = +1$, as desired.

7.3.2 Missing features and category constraints

A key benefit of Boltzmann training (including its preferred implementation, described in Sect. 7.3.3, below) is its ability to deal with missing features, both during training and during classification. If a deficient binary pattern is used for training, input units corresponding to missing features are allowed to vary — they are temporarily treated as (unclamped) hidden units rather than clamped input units. As a result, during annealing such units assume values most consistent with the rest of the input pattern and the current state of the network (Problem 14). Likewise, when a deficient pattern is to be classified, any units corresponding to missing input features are not clamped, and are allowed to assume any value.

Some subsidiary knowledge or constraints can be incorporated into a Boltzmann network during classification. Suppose in a five-category problem it is somehow known that a test pattern is neither in category ω_1 nor ω_4. (Such constraints could come from context or stages subsequent to the classifier itself.) During classification, then, the output units corresponding to ω_1 and ω_4 are clamped at $s_i = -1$ during the anneal, and the final category read as usual. Of course in this example the possible categories are then limited to the unclamped output units, for ω_2, ω_3 and ω_5. Such constraint imposition may lead to an improved classification rate (Problem 15).

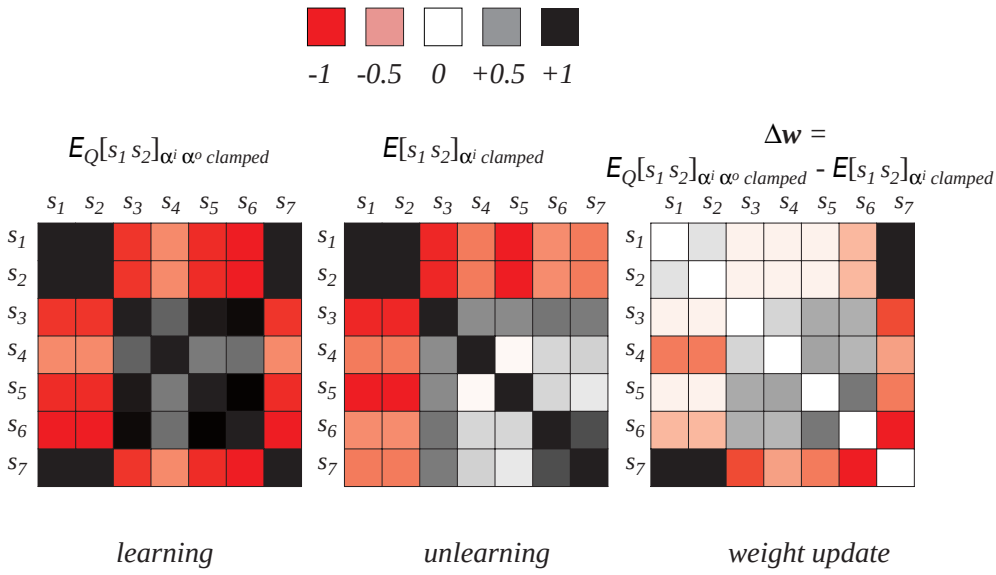

Figure 7.9: Boltzmann learning of a single pattern is illustrated for the seven-node network of Fig. 7.8. The (symmetric) matrix on the left shows the correlation of units for the learning component, where the input units are clamped to $s_1 = +1$, $s_2 = +1$, and the outputs to $s_6 = -1$, $s_7 = +1$. The middle matrix shows the unlearning component, where the inputs are clamped but outputs are free to vary. The difference between those matrices is shown on the right, and is proportional to the weight update (Eq. 14). Notice, for instance, that because the correlation between s_1 and s_2 is large in both the learning and unlearning components (because those variables are clamped), there is no associated weight change, i.e., $\Delta w_{12} = 0$. However, strong correlations between s_1 and s_7 in the learning but not in the unlearning component implies that the weight w_{17} should be increased, as can be seen in the weight update matrix.

Pattern completion

The problem of pattern completion is to estimate the full pattern given just a part of that pattern; as such, it is related to the problem of classification with missing features. Pattern completion is naturally addressed in Boltzmann networks. A fully interconnected network, with or without hidden units, is trained with a set of representative patterns; as before, the visible units correspond to the feature components. When a deficient pattern is presented, a subset of the visible units are clamped to the components of a partial pattern, and the network annealed. The estimate of the unknown features appears on the remaining visible units, as illustrated in Fig. 7.10 (Computer exercise 3). Such pattern completion in Boltzmann networks can be more accurate when known category information is imposed at the output units.

HOPFIELD NETWORK

Boltzmann networks without hidden or category units are related to so-called *Hopfield networks* or Hopfield auto-association networks (Problem 12). Such networks store patterns but not their category labels. The learning rule for such networks does not require the full Boltzmann learning of Eq. 14. Instead, weights are set to be proportional to the correlation of the feature vectors, averaged over the training set,

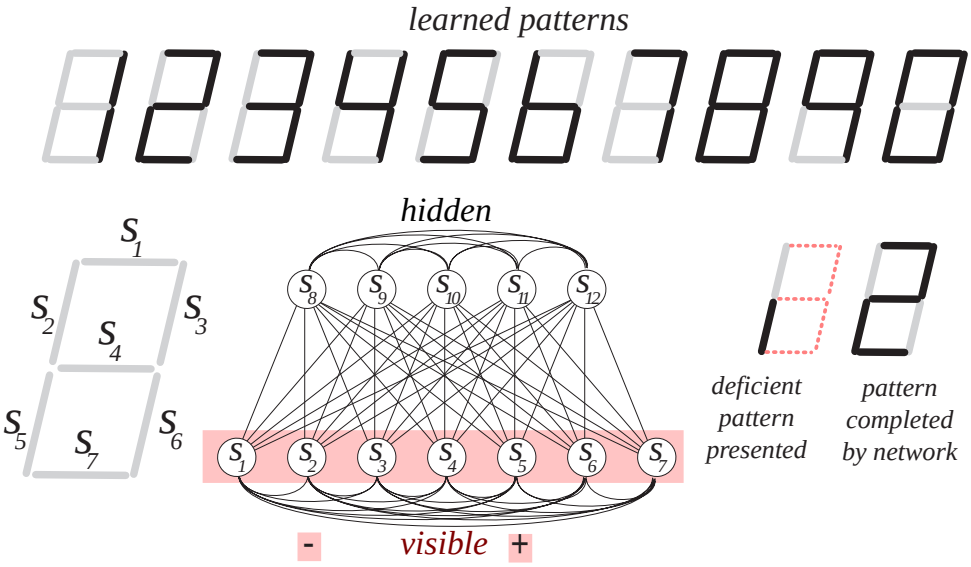

Figure 7.10: A Boltzmann network can be used for pattern completion, i.e., filling in unknown features of a deficient pattern. Here, a twelve-unit network with five hidden units has been trained with the 10 numeral patterns of a seven-segment digital display. The diagram at the lower left shows the correspondence between the display segments and nodes of the network; a black segment is represented by a $+1$ and a light gray segment as a -1. Consider the deficient pattern consisting of $s_2 = -1$, $s_5 = +1$. If these units are clamped and the full network annealed, the remaining five visible units will assume values most probable given the clamped ones, as shown at the right.

$$w_{ij} \propto \mathcal{E}_Q[s_i s_j], \tag{15}$$

with $w_{ii} = 0$; further, there is no need to consider temperature. Such learning is of course much faster than true Boltzmann learning using annealing. If a network fully trained by Eq. 15 is nevertheless annealed, as in full Boltzmann learning, there is no guarantee that the equilibrium correlations in the learning and unlearning phases are equal, i.e., that $\Delta w_{ij} = 0$ (Problem 13).

The successes of such Hopfield networks in true pattern recognition have been modest, partly because the basic Hopfield network does not have as natural an output representation for categorization problems. Occassionally, though they can be used in simple low-dimensional pattern completion or auto-association problems. One of their primary drawbacks is their limited capacity, analogous to the fact that a two-layer network cannot implement arbitrary decision boundaries as can a three-layer net. In particular, it has been shown that the number of d-dimensional random patterns that can be stored is roughly $0.14d$ — very limited indeed. In a Boltzmann with hidden units such as we have discussed, however, the number of hidden units can be increased in order to allow more patterns to be stored.

Because Boltzmann networks include loops and feedback connections, the internal representations learned at the hidden units are often difficult to interpret. Occasionally, though, the pattern of weights from the input units suggests feature groupings that are important for the classification task.

7.3.3 Deterministic Boltzmann learning

The computational complexity of stochastic Boltzmann learning in a network with hidden units is very high. Each pattern must be presented several times, and every anneal requires each unit to be polled several times. Just as mean-field annealing is usually preferable to stochastic annealing, so too a mean-field version of Boltzmann learning is preferable to the stochastic version. The basic approach in deterministic Boltzmann learning is to use Eq. 14 with mean-field annealing and analog values for the s_i. Recall, at the end of deterministic simulated annealing, the values of s_i are ± 1, as required by the problem.

Specifically, if we let $\mathcal{D}$ be the set of training patterns $\mathbf{x}$ containing feature and category information, the algorithm is:

Algorithm 3 (Deterministic Boltzmann learning)

1 **begin initialize** $\mathcal{D}, \eta, T(k), w_{ij}\ i, j = 1, \ldots, N$
2 **do** Randomly select training pattern $\mathbf{x}$
3 Randomize states s_i
4 Anneal network with input and output clamped
5 At final, low T, calculate $[s_i s_j]_{\alpha^i \alpha^o clamped}$
6 Randomize states s_i
7 Anneal network with input clamped but output free
8 At final, low T, calculate $[s_i s_j]_{\alpha^i clamped}$
9 $w_{ij} \leftarrow w_{ij} + \eta/T \left[[s_i s_j]_{\alpha^i \alpha^o clamped} - [s_i s_j]_{\alpha^i clamped} \right]$
10 **until** $k = k_{max}$ or convergence criterion met
11 **return** w_{ij}
12 **end**

Using mean-field theory, it is possible to efficiently calculate *approximations* of the mean of correlations entering the gradient. The analog state s_i of each unit replaces its average value $\mathcal{E}[s_i]$ and could in theory be calculated by iteratively solving a set of nonlinear equations. The mean of correlations is then calculated by making the approximation $\mathcal{E}[s_i s_j] \approx \mathcal{E}[s_i]\mathcal{E}[s_j] \approx s_i s_j$, as shown in lines 5 & 8.

7.3.4 Initialization and setting parameters

As with virtually every classifier, there are several interrelated parameters that must be set in a Boltzmann network. The first are the network topology and number of hidden units. The number of visible units (input and output) is determined by the dimensions of the binary feature vectors and number of categories. In the absence of detailed information about the problem, we assume the network is fully interconnected, and thus merely the number of hidden units must be set. A popular alternate topology is obtained by eliminating interconnections among input units, as well as among output units. (Such a network is faster to train but will be somewhat less effective at pattern completion or classifying deficient patterns.) Of course, generally speaking the harder the classification problem the more hidden units will be needed. The question is then, how many hidden units should be used?

Suppose the training set $\mathcal{D}$ has n distinct patterns of input-output pairs. An *upper bound* on the minimum number of hidden units is n — one for each pattern — where for each pattern there is a corresponding unique hidden unit having value $s_i = +1$ while all others are -1. This internal representation can be insured in the

following way: for the particular hidden unit i, set w_{ij} to be positive for each input unit j corresponding to a $+1$ feature in its associated pattern; further set w_{ij} to be negative for input units corresponding to a -1 feature. For the remaining hidden units, the sign of the corresponding weights should be inverted. Next, the connection from hidden unit i to the output unit corresponding to the known category should be positive, and negative to all other output units. The resulting internal representation is closely related to that in the probabilistic neural network implementation of Parzen windows (Chap. **??**). Naturally, this representation is undesirable as the number of weights grows exponentially with the number of patterns. Training becomes slow; furthermore generalization tends to be poor.

Since the states of the hidden units are binary valued, and since it takes $\lceil \log_2 n \rceil$ bits to specify n different items, there must be at least $\lceil \log_2 n \rceil$ hidden units if there is to be a distinct hidden configuration for each of the n patterns. Thus a *lower bound* on the number of hidden units is $\lceil \log_2 n \rceil$, which is necessary for a distinct hidden configuration for each pattern. Nevertheless, this bound need not be tight, as there may be no set of weights insuring a unique representation (Problem 16). Aside from these bounds, it is hard to make firm statements about the number of hidden units needed — this number depends upon the inherent difficulty of the classification problem. It is traditional, then, to start with a somewhat large net and use weight decay. Much as we saw in backpropagation (Chap. **??**), a Boltzmann network with "too many" hidden units and weights can be improved by means of weight decay. During training, a small increment ϵ is added to w_{ij} when s_i and s_j are both positive or both negative during learning phase, but subtracted in the unlearning phase. It is traditional to decrease ϵ throughout training. Such a version of weight decay tends to reduce the effects on the weights due to spurious random correlations in units and to eliminate unneeded weights, thereby improving generalization.

One of the benefits of Boltzmann networks over backpropagation networks is that "too many" hidden units in a backpropagation network tend to degrade performance more than "too many" in a Boltzmann network. This is because during learning, there is stochastic averaging over states in a Boltzmann network which tends to smooth decision boundaries; backpropagation networks have no such equivalent averaging. Of course, this averaging comes at a higher computational burden for Boltzmann networks.

The next matter to consider is weight initialization. Initializing all weights to zero is acceptable, but leads to unnecessarily slow learning. In the absence of information otherwise, we can expect that roughly half the weights will be positive and half negative. In a network with fully interconnected hidden units there is nothing to differentiate the individual hidden units; thus we can arbitrarily initialize roughly half of the weights to have positive values and the rest negative. Learning speed is increased if weights are initialized with random values within a proper range. Assume a fully interconnected network having N units (and thus $N - 1 \approx N$ connections to each unit). Assume further that at any instant each unit has an equal chance of being in state $s_i = +1$ or $s_i = -1$. We seek initial weights that will make the net force on each unit a random variable with variance 1.0, roughly the useful range shown in Fig. 7.5. This implies weights should be initialized randomly throughout the range $-\sqrt{3/N} < w_{ij} < +1\sqrt{3/N}$ (Problem 17).

As mentioned, annealing schedules of the form $T(k+1) = cT(k)$ for $0 < c < 1$ are generally used, with $0.8 < c < 0.99$.

If a very large number of iterations — several thousand — are needed, even $c = 0.99$ may be too small. In that case we can write $c = e^{-1/k_0}$, and thus

$T(k) = T(1)e^{-k/k_0}$, and k_0 can be interpreted as a decay constant. The initial temperature $T(1)$ should be set high enough that virtually all candidate state transitions are accepted. While this condition can be insured by choosing $T(1)$ extremely high, in order to reduce training time we seek the lowest adequate value of $T(1)$. A lower bound on the acceptable initial temperature depends upon the problem, but can be set empirically by monitoring state transitions in short simulations at candidate temperatures. Let m_1 be the number of energy-decreasing transitions (these are always accepted), and m_2 the number of energy-*in*creasing queries according to the annealing algorithm; let $\mathcal{E}_+[\Delta E]$ denote the average increase in energy over such transitions. Then, from Eq. 4 we find that the acceptance ratio is

$$R = \frac{\text{number of accepted transitions}}{\text{number of proposed transitions}} \approx \frac{m_1 + m_2 \cdot \exp[-\mathcal{E}_+[\Delta E]/T(1)]}{m_1 + m_2}. \tag{16}$$

Rearranging terms we see that the initial temperature obeys

$$T(1) = \frac{\mathcal{E}_+[\Delta E]}{\ln[m_2] - \ln[m_2 R - m_1(1-R)]}. \tag{17}$$

For any initial temperature set by the designer, the acceptance ratio may or may not be nearly the desired 1.0; nevertheless Eq. 17 will be obeyed. The appropriate value for $T(1)$ is found through a simple iterative procedure. First, set $T(1)$ to zero and perform a sequence of m_0 trials (pollings of units); count empirically the number of energetically favorable (m_1) and energetically unfavorable (m_2) transitions. In general, $m_1 + m_2 < m_0$ because many candidate energy increasing transitions are rejected, according to Eq. 4. Next, use Eq. 17 to calculate a new, improved value of $T(1)$ from the observed m_1 and m_2. Perform another sequence of m_0 trials, observe new values for m_1 and m_2, recalculate $T(1)$, and so on. Repeat this procedure until $m_1 + m_2 \approx m_0$. The associated $T(1)$ gives an acceptance ratio $R \approx 1$, and is thus to be used. In practice this method quickly yields a good starting temperature.

The next important parameter is the learning rate η in Eq. 14. Recall that the learning is based on gradient descent in the weighted Kullback-Leibler divergence between the actual and the desired distributions on the visible units. In Chap. **??** we derived bounds on the learning rate for multilayer neural networks by calculating the curvature of the error, and finding the maximum value of the learning rate that insured stability. This curvature was based on a Hessian matrix, the matrix of second-order derivatives of the error with respect to the weights. In the case of an N-unit, fully connected Boltzmann network, whose $N(N-1)/2$ weights are described by a vector $\mathbf{w}$, this curvature is proportional to $\mathbf{w}^t\mathbf{H}\mathbf{w}$, where

$$\mathbf{H} = \frac{\partial^2 \bar{D}_{KL}}{\partial \mathbf{w}^2} \tag{18}$$

is the appropriate Hessian matrix and the Kullback-Liebler divergence is given by Eq. 12. Given weak assumptions about the classification problem we can estimate this Hessian matrix; the stability requirement is then simply $\eta \leq T^2/N^2$ (Problem 18). Note that at large temperature T, a large learning rate is acceptable since the effective error surface is smoothed by high randomness.

While not technically parameter setting, one heuristic that provides modest computational speedup is to propose changing the states of *several* nodes simultaneously early in an anneal. The change in energy and acceptance probability are calculated

as before. At the end of annealing, however, polling should be of single units in order to accurately find the optimum configuration.

A method which occasionally improves the final solution is to update and store the current best configuration during an anneal. If the basic annealing converges to a local minimum that is worse than this stored configuration, this current optimal should be used. This is a variant of the *pocket algorithm* which finds broad use in methods that do not converge monotonically or can get caught in local minima (Chap **??**). POCKET ALGORITHM

There are two stopping criteria associated with Boltzmann learning. The first determines when to stop a single anneal (associated with either the learning or the unlearning components). Here, the final temperature should be so low that no energetically unfavorable transitions are accepted. Such information is readily apparent in the graph of the energy versus iteration number, such as shown at the right of Fig. 7.3. All N variables should be polled individually at the end of the anneal, to insure that the final configuration is indeed a local (though perhaps not global) energy minimum.

The second stopping criterion controls the number of times each training pattern is presented to the network. Of course the proper criterion depends upon the inherent difficulty of the classification problem. In general, overtraining is less of a concern in Boltzmann networks than it is in multilayer neural networks trained via gradient descent. This is because the averaging over states in Boltzmann networks tends to smooth decision boundaries while overtraining in multilayer neural networks tunes the decision boundaries to the particular training set. A reasonable stopping criterion for Boltzmann networks is to monitor the error on a validation set (Chap. **??**), and stop learning when this error no longer changes significantly.

7.4 *Boltzmann networks and graphical models

While we have considered fully interconnected Boltzmann networks, the learning algorithm (Algorithm 3) applies equally well to networks with arbitrary connection topologies. Furthermore, it is easy to modify Boltzmann learning in order to impose constraints such as weight sharing. As a consequence, several popular recognition architectures — so-called graphical models such as Bayesian belief networks and Hidden Markov Models — have counterparts in structured Boltzmann networks, and this leads to new methods for training them.

Recall from Chap. **??** that Hidden Markov Models consist of several discrete hidden and visible states; at each discrete time step t, the system is in a single hidden state and emits a single visible state, denoted $\omega(t)$ and $v(t)$, respectively. The transition probabilities between hidden states at successive time steps are

$$a_{ij} = P(\omega_j(t+1)|\omega_i(t)) \tag{19}$$

and between hidden and visible states at a given time are

$$b_{jk} = P(v_k(t)|\omega_j(t)). \tag{20}$$

The Forward-Backward or Baum-Welch algorithm (Chap. **??**, Algorithm **??**) is traditionally used for learning these parameters from a pattern of T_f visible states* $\mathbf{V}^{T_f} = \{v(1), v(2), \ldots, v(T_f)\}$.

* Here we use T_f to count the number of discrete time steps in order to avoid confusion with the temperature T in Boltzmann simulations.

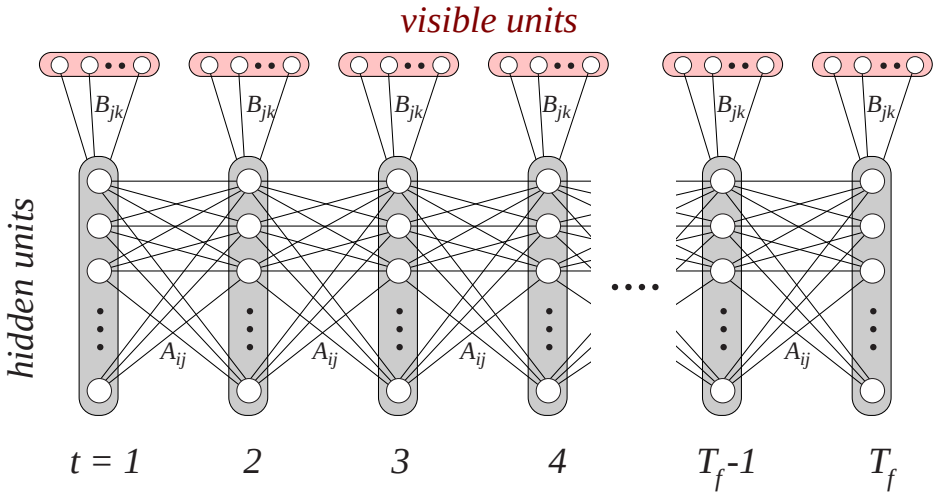

Figure 7.11: A Hidden Markov Model can be "unfolded" in time to show a trellis, which can be represented as a Boltzmann chain, as shown. The discrete hidden states are grouped into vertical sets, fully interconnected by weights A_{ij} (related to the HMM transition probabilities a_{ij}). The discrete visible states are grouped into horizontal sets, and are fully interconnected with the hidden states by weights B_{jk} (related to transition probabilities b_{jk}). Training the net with a single pattern, or list of T_f visible states, consists of clamping the visible states and performing Boltzmann learning throughout the full network, with the constraint that each of the time shifted weights labeled by a particular A_{ij} have the same numerical value.

BOLTZMANN CHAIN

Recall that a Hidden Markov model can be "unfolded" in time to yield a trellis (Chap. **??**, Fig. **??**). A structured Boltzmann network with the same trellis topology — a *Boltzmann chain* — can be used to implement the same classification as the corresponding Hidden Markov Model (Fig. 7.11). Although it is often simpler to work in a representation where discrete states have multiple values, we temporarily work in a representation where the binary nodes take value $s_i = 0$ or $+1$, rather than ± 1 as in previous discussions. In this representation, a special case of the general energy (Eq. 1) includes terms for a particular sequence of visible, $\mathbf{V}^{T_f}$, and hidden states $\boldsymbol{\omega}^{T_f} = \{\omega(1), \omega(2), \ldots, \omega(T_f)\}$ and can be written as

$$E_{\boldsymbol{\omega}\mathbf{V}} = E[\boldsymbol{\omega}^{T_f}, \mathbf{V}^{T_f}] = -\sum_{t=1}^{T_f - 1} A_{ij} - \sum_{t=1}^{T_f} B_{jk} \tag{21}$$

where the particular values of A_{ij} and B_{jk} terms depend implicitly upon the sequence. The choice of binary state representation implies that only the weights linking nodes that both have $s_i = +1$ appear in the energy. Each "legal" configuration — consisting of a single visible unit and a single hidden unit at each time — implies a set of A_{ij} and B_{jk} (Problem 20). The partition function is the sum over all legal states,

$$Z = \sum_{\boldsymbol{\omega}\mathbf{V}} e^{-E_{\boldsymbol{\omega}\mathbf{V}}/T}, \tag{22}$$

which insures normalization. The correspondence between the Boltzmann chain at temperature T and the unfolded Hidden Markov model (trellis) implies

$$A_{ij} = T\ln\, a_{ij} \qquad \text{and} \qquad B_{jk} = T\ln\, b_{jk}. \tag{23}$$

(As in our discussion of Hidden Markov Models, we assume the initial hidden state is known and thus there is no need to consider the correspondence of prior probabilities in the two approaches.) While the $0-1$ binary representation of states in the structured network clarifies the relationship to Hidden Markov Models through Eq. 21, the more familiar representation $s_i = \pm 1$ works as well. Weights in the structured Boltzmann network are trained according to the method of Sect. 7.3, though the relation to transition probabilities in a Hidden Markov Model is no longer simple (Problem 21).

Other graphical models

In addition to Hidden Markov Models, a number of graphical models have analogs in structured Boltzmann networks. One of the most general includes Bayesian belief nets, directed acyclic graphs in which each node can be in one of a number of discrete states, and nodes are interconnected with conditional probabilities (Chap. **??**). As in the case of Hidden Markov Models, the correspondence with Boltzmann networks is clearest if the discrete states in the belief net are binary states; nevertheless in practice multistate representations more naturally enforce the constraints and are generally preferred (Computer exercise **??**).

A particularly intriguing recognition problem arises when a temporal signal has two inherent time scales, for instance the rapid daily behavior in a financial market superimposed on slow seasonal variations. A standard Hidden Markov Model typically has a *single* inherent time scale and hence is poorly suited to such problems. We might seek to use two interconnected HMMs, possibly with different numbers of hidden states. Alas, the Forward-Backward algorithm generally does not converge when applied to a model having closed loops, as when two Hidden Markov Models have cross connections.

Here the correspondence with Boltzmann networks is particularly helpful. We can link two Boltzmann chains with cross connections, as shown in Fig. 7.12, to form a *Boltzmann zipper*. The particular benefit of such an architecture is that it can learn both short-time structure (through the "fast" component chain) as well as long-time structure (through the "slow" chain). The cross connections, labeled by weight matrix $\mathbf{E}$ in the figure, learn correlations between the "fast" and "slow" internal representations. Unlike the case in Eq. 23, the $\mathbf{E}$ weights are not simply related to transition probabilities, however (Problem **??**).

BOLTZMANN ZIPPER

Boltzmann zippers can address problems such as acoustic speech recognition, where the fast chain learns the rapid transitions and structure of individual phonemes while the slow component chain learns larger structure associated with prosody and stress throughout a word or a full phrase. Related applications include speechreading (lipreading), where the fast chain learns the acoustic transitions and the slow chain the much slower transitions associated with the (visible) image of the talker's lips, jaw and tongue and body gestures, where fast hand motions are coupled to slower large-scale motions of the arms and torso.

7.5 *Evolutionary methods

Inspired by the process of biological evolution, evolutionary methods of classifier design employ stochastic search for an optimal classifier. These admit a natural imple-

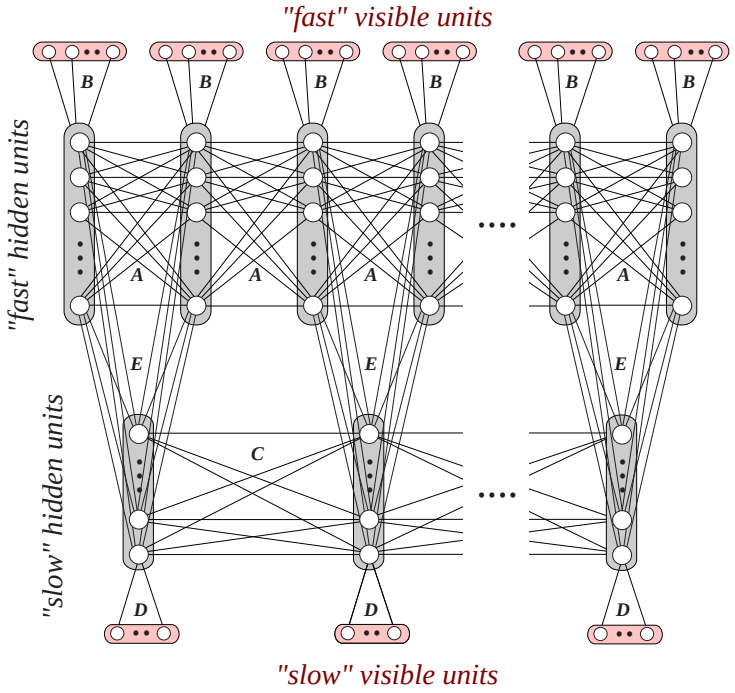

Figure 7.12: A Boltzmann zipper consists of two Boltzmann chains (cf. Fig. 7.11), whose hidden units are interconnected. The component chains differ in the rate at which visible features are sampled, and thus they capture structure at different temporal scales. Correlations are learned by the weights linking the hidden units, here labeled **E**. It is somewhat more difficult to train linked Hidden Markov Models to learn structure at different time scales.

POPULATION

SCORE

FITNESS

SURVIVAL OF THE FITTEST

OFFSPRING

PARENT

mentation on massively parallel computers. In broad overview, such methods proceed as follows. First, we create several classifiers — a *population* — each varying somewhat from the other. Next, we judge or *score* each classifier on a representative version of the classification task, such as accuracy on a set of labeled examples. In keeping with the analogy with biological evolution, the resulting (scalar) score is sometimes called the *fitness*. Then we rank these classifiers according to their score and retain the best classifiers, some portion of the total population. Again, in keeping with biological terminology, this is called *survival of the fittest*.

We now stochastically alter the classifiers to produce the next generation — the children or *offspring*. Some offspring classifiers will have higher scores than their *parents* in the previous generation, some will have lower scores. The overall process is then repeated for subsequent generation: the classifiers are scored, the best ones retained, randomly altered to give yet another generation, and so on. In part because of the ranking, each generation has, on average, a slightly higher score than the previous one. The process is halted when the single best classifier in a generation has a score that exceeds a desired criterion value.

The method employs stochastic variations, and these in turn depend upon the fundamental representation of each classifier. There are two primary representations we shall consider: a string of binary bits (in basic genetic algorithms), and snippets of computer code (in genetic programming). In both cases, a key property is that

occasionally very large changes in classifier are introduced. The presence of such large changes and random variations implies that evolutionary methods can find good classifiers even in extremely complex discontinuous spaces or "fitness landscapes" that are hard to address by techniques such as gradient descent.

7.5.1 Genetic Algorithms

In basic genetic algorithms, the fundamental representation of each classifier is a binary string, called a *chromosome*. The mapping from the chromosome to the features and other aspects of the classifier depends upon the problem domain, and the designer has great latitude in specifying this mapping. In pattern classification, the score is usually chosen to be some monotonic function of the accuracy on a data set, possibly with penalty term to avoid overfitting. We use a desired fitness, θ, as the stopping criterion. Before we discuss these points in more depth, we first consider more specifically the structure of the basic genetic algorithm, and then turn to the key notion of genetic operators, used in the algorithm.

CHROMOSOME

Algorithm 4 (Basic Genetic algorithm)

1 **begin initialize** $\theta, P_{co}, P_{mut}, L$ N-bit chromosomes
2 **do** Determine fitness of each chromosome, $f_i, i = 1, \ldots, L$
3 Rank the chromosomes
4 **do** Select two chromosomes with highest score
5 **if** $Rand[0,1) < P_{co}$ **then** crossover the pair at a randomly chosen bit
6 **else** change each bit with probability P_{mut}
7 Remove the parent chromosomes
8 **until** N offspring have been created
9 **until** Any chromosome's score f exceeds θ
10 **return** Highest fitness chromosome (best classifier)
11 **end**

Figure 7.13 shows schematically the evolution of a population of classifiers given by Algorithm 4.

Genetic operators

There are three primary genetic operators that govern reproduction, i.e., producing offspring in the next generation described in lines 5 & 6 of Algorithm 4. The last two of these introduce variation into the chromosomes (Fig. 7.14):

Replication: A chromosome is merely reproduced, unchanged.

Crossover: Crossover involves the mixing — "mating" — of two chromosomes. A split point is chosen randomly along the length of either chromosome. The first part of chromosome A is spliced to the last part of chromosome B, and vice versa, thereby yielding two new chromosomes. The probability a given pair of chromosomes will undergo crossover is given by P_{co} in Algorithm 4.

MATING

Mutation: Each bit in a single chromosome is given a small chance, P_{mut}, of being changed from a 1 to a 0 or vice versa.

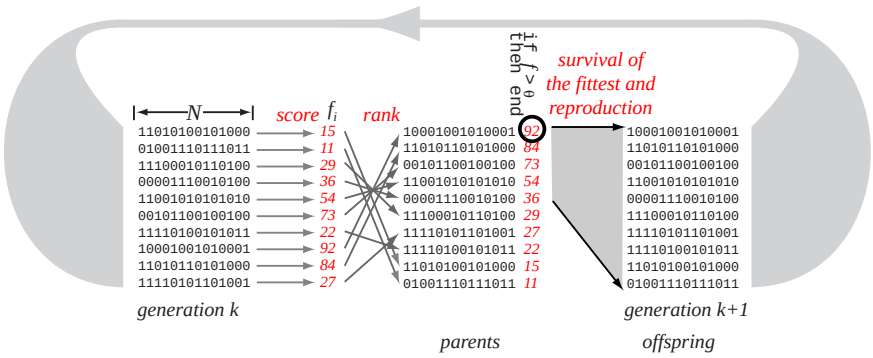

Figure 7.13: A basic genetic algorithm is a stochastic iterative search method. Each of the L classifiers in the population in generation k is represented by a string of bits of length N, called a chromosome (on the left). Each classifier is judged or scored according its performance on a classification task, giving L scalar values f_i. The chromosomes are then ranked according to these scores. The chromosomes are considered in descending order of score, and operated upon by the genetic operators of replication, crossover and mutation to form the next generation of chromosomes — the offspring. The cycle repeats until a classifier exceeds the criterion score θ.

Other genetic operators may be employed, for instance *inversion* — where the chromosome is reversed front to back. This operator is used only rarely since inverting a chromosome with a high score nearly always leads to one with very low score. Below we shall briefly consider another operator, *insertions*.

Representation

When designing a classifier by means of genetic algorithms we must specify the mapping from a chromosome to properties of the classifier itself. Such mapping will depend upon the form of classifier and problem domain, of course. One of the earliest and simplest approaches is to let the bits specify features (such as pixels in a character recognition problem) in a two-layer Perceptron with fixed weights (Chap. **??**). The primary benefit of this particular mapping is that different segments of the chromosome, which generally remain undisturbed under the crossover operator, may evolve to recognize different *portions* of the input space such as the descender (lower) or the ascender (upper) portions of typed characters. As a result, occasionally the crossover operation will append a good segment for the ascender region in one chromosome to a good segment for the descender region in another, thereby yielding an excellent overall classifier.

Another mapping is to let different segments of the chromosome represent the weights in a multilayer neural net with a fixed topology. Likewise, a chromosome could represent a network topology itself, the presence of an individual bit implying two particular neurons are interconnected. One of the most natural representations is for the bits to specify properties of a decision tree classifier (Chap. **??**), as shown in Fig. 7.15.

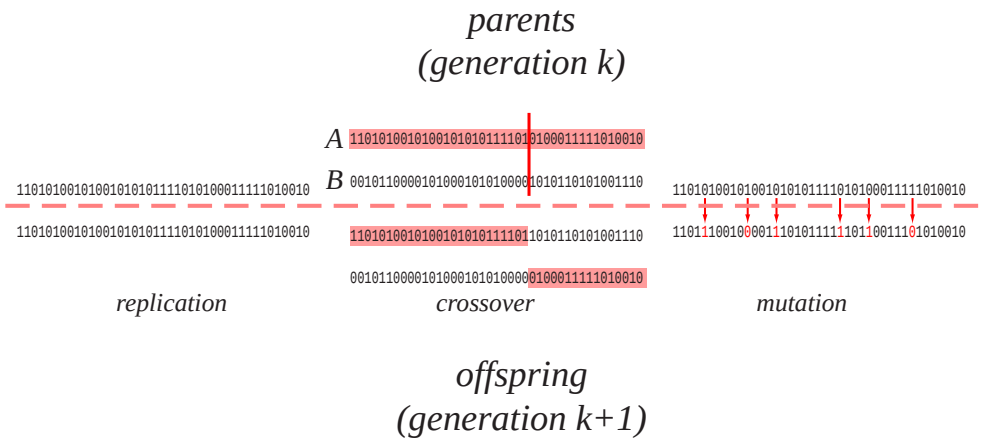

Figure 7.14: Three basic genetic operations are used to transform a population of chromosomes at one generation to form a new generation. In replication, the chromosome is unchanged. Crossover involves the mixing or "mating" of two chromosomes to yield two new chromosomes. A position along the chromosomes is chosen randomly (red vertical line); then the first part of chromosome A is linked with the last part of chromosome B, and vice versa. In mutation, each bit is given a small chance of being changed from a 1 to a 0 or vice versa.

Scoring

For a c-category classification problem, it is generally most convenient to evolve c dichotomizers, each to distinguish a different ω_i from all other ω_j for $j \neq i$. During classification, the test pattern is presented to each of the c dichotomizers and assigned the label accordingly. The goal of classifier design is accuracy on future patterns, or if decisions have associated costs, then low expected cost. Such goals should be reflected in the method of scoring and selection in a genetic algorithm. Given sample patterns representative version of the target classification task, it is natural to base the score on the classification accuracy measured on the data set. As we have seen numerous times, there is a danger that the classifier becomes "tuned" to the properties of the particular data set, however. (We can informally broaden our usage of the term "overfitting" from generic learning to apply to this search-based case as well.) One method for avoiding such overfitting is penalizing classifier complexity, and thus the score should have a term that penalizes overly large networks. Another method is to adjusting the stopping criterion. Since the appropriate measure of classifier complexity and the stopping criterion depend strongly on the problem, it is hard to make specific guidelines in setting these parameters. Nevertheless, designers should be prepared to explore these parameters in any practical application.

Selection

The process of selection specifies which chromosomes from one generation will be sources for chromosomes in the next generation. Up to here, we have assumed that the chromosomes would be ranked and selected in order of decreasing fitness until the next generation is complete. This has the benefit of generally pushing the population toward higher and higher scores. Nevertheless, the average improvement from one generation to the next depends upon the variance in the scores at a given generation,

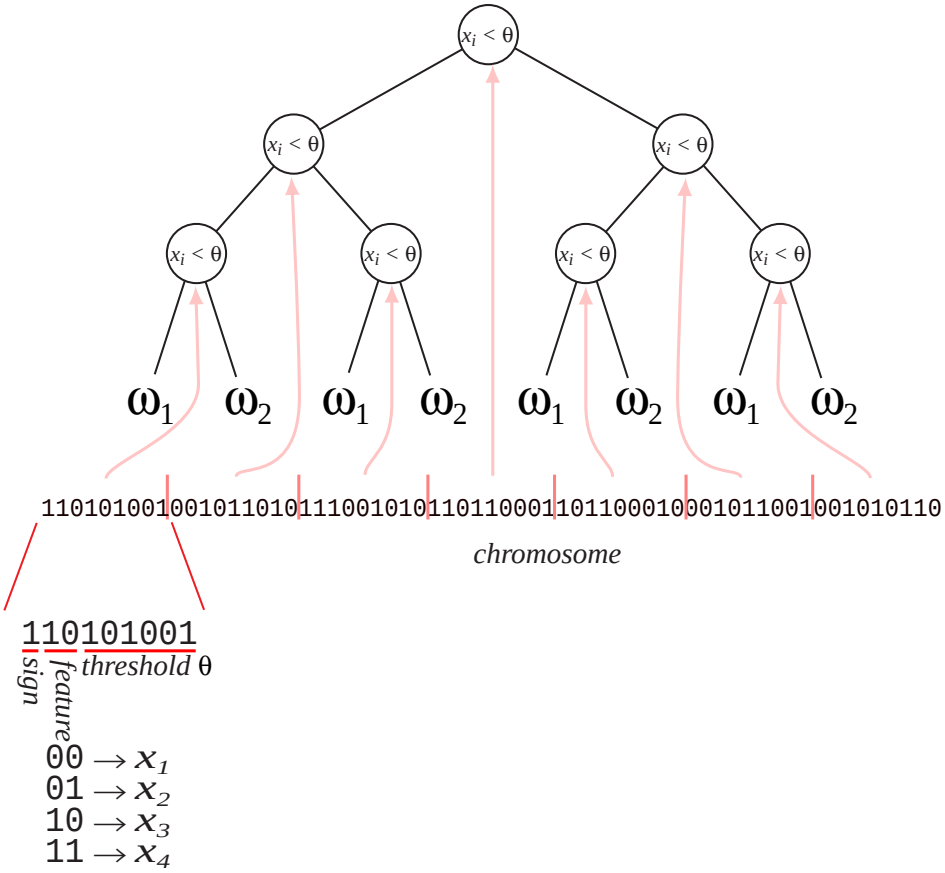

Figure 7.15: One natural mapping is from a binary chromosome to a binary tree classifier, illustrated here for a four-feature, monothetic tree dichotomizer. In this example, each of the nodes computes a query of the form $\pm x_i < \theta$? and is governed by nine bits in the chromosome. The first bit specifies a sign, the next two bits specify the feature queried. The remaining six bits are a binary representation of the threshold θ. For instance, the left-most node encodes the rule $+x_3 < 41$? (In practice, larger trees would be used for problems with four features.)

and because this standard fitness-based selection need not give high variance, other selection methods may prove superior.

FITNESS-PROPORTIONAL SELECTION

The principle alternative selection scheme is *fitness-proportional selection*, or fitness-proportional reproduction, in which the probability that each chromosome is selected is proportional to its fitness. While high-fitness chromosomes are preferentially selected, occasionally low-fitness chromosomes are selected, and this may preserve diversity and increase variance of the population.

A minor modification of this method is to make the probability of selection proportional to some monotonically increasing function of the fitness. If the function instead has a positive second derivative, the probability that high-fitness chromosomes is enhanced. One version of this heuristic is inspired by the Boltzmann factor of Eq. 2; the probability that chromosome i with fitness f_i will be selected is

$$P(i) = \frac{e^{f_i/T}}{\mathcal{E}[e^{f_i/T}]}, \tag{24}$$

where the expectation is over the current generation and T is a control parameter loosely referred to as a temperature. Early in the evolution the temperature is set high, giving all chromosomes roughly equal probability of being selected. Late in the evolution the temperature is set lower so as to find the chromosomes in the region of the optimal classifier. We can express such search by analogy to biology: early in the search the population remains diverse and *explores* the fitness landscape in search of promising areas; later the population *exploits* the specific fitness opportunities in a small region of the space of possible classifiers.

7.5.2 Further heuristics

There are many additional heuristics that can occasionally be of use. One concerns the adaptation of the crossover and mutation rates, P_{co} and P_{mut}. If these rates are too low, the average improvement from one generation to the next will be small, and the search unacceptably long. Conversely, if these rates are too high, the evolution is undirected and similar to a highly inefficient random search. We can monitor the average improvement in fitness of each generation and the mutation and crossover rates as long as such improvement is rapid. In practice, this is done by encoding the rates in the chromosomes themselves and allowing the genetic algorithm to select the proper values.

Another heuristic is to use a ternary, or n-ary chromosomes rather than the traditional binary ones. These representations provide little or no benefit at the algorithmic level, but may make the mapping to the classifier itself more natural and easier to compute. For instance, a ternary chromosome might be most appropriate if the classifier is a decision tree with three-way splits.

Occasionally the mapping to the classifier will work for chromosomes of different length. For example, if the bits in the chromosome specify weights in a neural network, then longer chromosomes would describe networks with a larger number of hidden units. In such a case we allow the *insertion* operator, which with a small probability inserts bits into the chromosome at a randomly chosen position. This so-called "messy" genetic algorithm method has a more appropriate counterpart in genetic programming, as we shall see in Sect. 7.6.

INSERTION

7.5.3 Why do they work?

Because there are many heuristics to choose as well as parameters to set, it is hard to make firm theoretical statements about building classifiers by means of evolutionary methods. The performance and search time depend upon the number of bits, the size of a population, the mutation and crossover rates, choice of features and mapping from chromosomes to the classifier itself, the inherent difficulty of the problem and possibly parameters associated with other heuristics.

A genetic algorithm restricted to mere replication and mutation is, at base, a version of stochastic random search. The incorporation of the crossover operator, which mates two chromosomes, provides a qualitatively different search, one that has no counterpart in stochastic grammars (Chap. **??**). Crossover works by finding, rewarding and recombining "good" segments of chromosomes, and the more faithfully the segments of the chromosomes represent such functional building blocks, the better

we can expect genetic algorithms to perform. The only way to insure this is with prior knowledge of the problem domain and the desired form of classifier.

7.6 *Genetic Programming

Genetic programming shares the same algorithmic structure of basic genetic algorithms, but differs in the representation of each classifier. Instead of chromosomes consisting of strings of bits, genetic programming uses snippets of computer programs made up of mathematical operators and variables. As a result, the genetic operators are somewhat different; moreover a new operator plays a significant role in genetic programming.

The four principal operators in genetic programming are (Fig. 7.16):

Replication: A snippet is merely reproduced, unchanged.

MATING **Crossover:** Crossover involves the mixing — "mating" — of two snippets. A split point is chosen from allowable locations in snippet A as well as from snippet B. The first part of snippet A is spliced to the back part of chromosome B, and vice versa, thereby yielding two new snippets.

Mutation: Each bit in a single snippet is given a small chance of being changed to a different value. Such a change must be compatible with the syntax of the total snippet. For instance, a number can be replaced by another number; a mathematical operator that takes a single argument can be replaced by another such operator, and so forth.

INSERTION **Insertion:** Insertion consists in replacing a single element in the snippet with another (short) snippet randomly chosen from a set.

In the c-category problem, it is simplest to form c dichotomizers just as in genetic algorithms. If the output of the classifier is positive, the test pattern belongs to category ω_i, if negative, then it is NOT in ω_i.

Representation

A program must be expressed in some language, and the choice affects the complexity of the procedure. Syntactically rich languages such as *C* or *C++* are complex and somwhat difficult to work with. Here the syntactic simplicity of a language such as*Lisp* is advantageous. Many *Lisp* expressions can be written in the form `(<operator> <operand> <operand>)`, where an `<operand>` can be a constant, a variable or another parenthesized expression. For example, `(+ X 2)` and `(* 3 (+ Y 5))` are valid *Lisp* expressions for the arithmetic expressions $x + 2$ and $3(y + 5)$, respectively. These expressions are easily represented by a binary tree, with the operator being specified at the node and the operands appearing as the children (Fig. 7.17).

Whatever language is used, genetic programming operators used for mutation should replace variables and constants with variables and constants, and operators with functionally compatible operators. They should aslo be required to produce syntactically valid results. Nevertheless, occassionally an ungrammatical code snippet
WRAPPER may be produced. For that reason, it is traditional to employ a *wrapper* — a routine that decides whether the classifier is meaningful, and eliminates them if not.

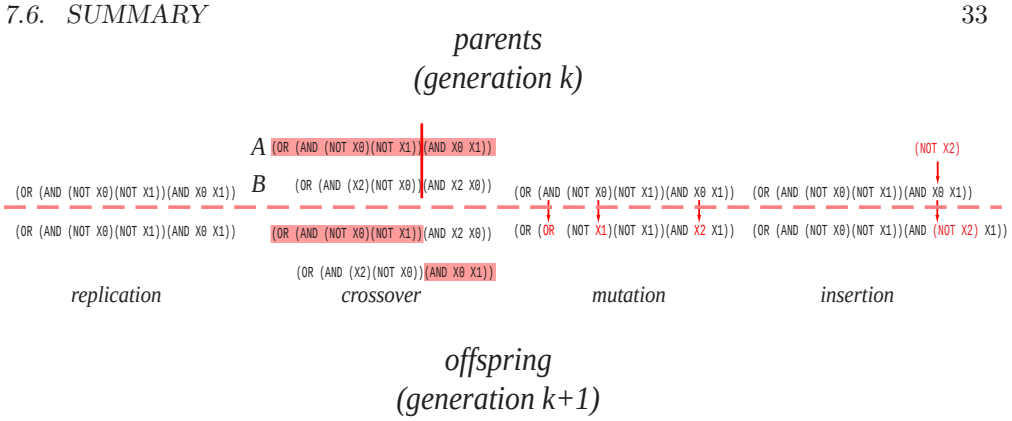

Figure 7.16: Four basic genetic operations are used to transform a population of snippets of code at one generation to form a new generation. In replication, the snippet is unchanged. Crossover involves the mixing or "mating" of two snippets to yield two new snippets. A position along the snippet A is randomly chosen from the allowable locations (red vertical line); likewise one is chosen for snippet B. Then the front portion of A is spliced to the back portion of B and vice versa. In mutation, each element is given a small chance of being changed. There are several different types of elements, and replacements must be of the same type. For instance, only a number can replace another number; only a numerical operator that takes a single argument can replace a similar operator, and so on. In insertion, a randomly selected element is replaced by a compatible snippet, keeping the entire snippet grammatically well formed and meaningful.

It is nearly impossible to make sound theoretical statements about genetic programming and even the rules of thumb learned from simulations in one domain, such as control or function optimization are of little value in another domain, such as classification problems. Of course, the method works best in problems that are matched by the classifier representation, as simple operations such as multiplication, division, square roots, logical NOT, and so on.

Nevertheless, we can state that as computation continues to decrease in cost, more of the burden of solving classification problems will be assumed by computation rather than careful analysis, and here techniques such as evolutionary ones will be of use in classification research.

Summary

When a pattern recognition problem involves a model that is discrete or of such high complexity that analytic or gradient descent methods are unlikely to work, we may employ stochastic techniques — ones that at some level rely on randomness to find model parameters. Simulated annealing, based on physical annealing of metals, consists in randomly perturbing the system, and gradually decreasing the randomness to a low final level, in order to find an optimal solution. Boltzmann learning trains the weights in a network so that the probability of a desired final output is increased. Such learning is based on gradient descent in the Kullback-Liebler divergence between two distributions of visible states at the output units: one distribution describes these units when clamped at the known category information, and the other when they are free to assume values based on the activations throughout the network. Some

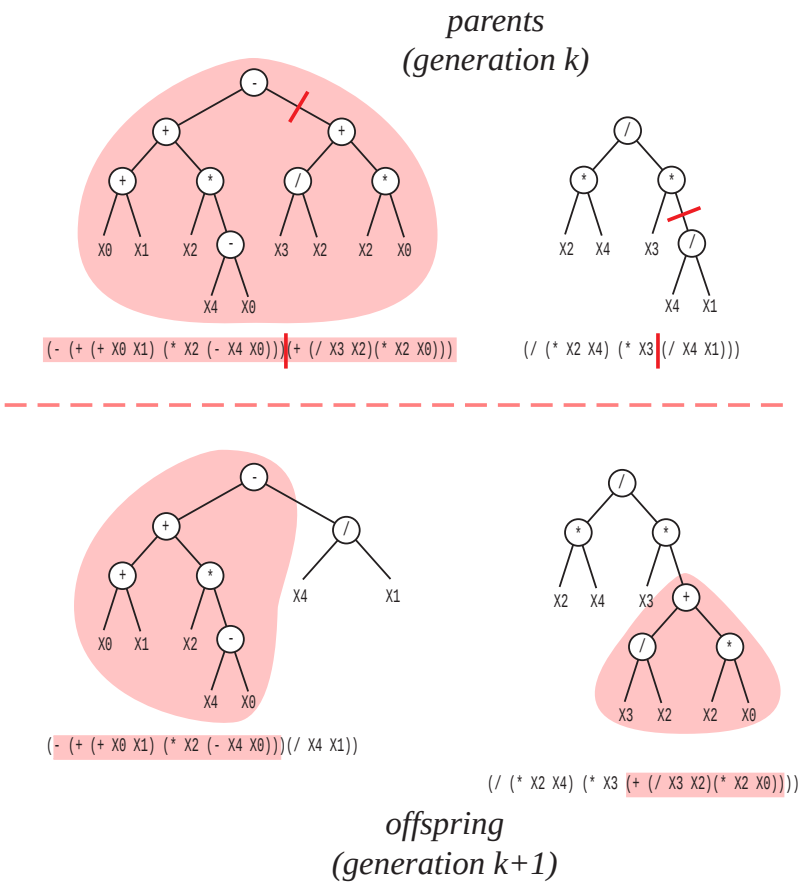

Figure 7.17: Unlike the decision trees of Fig. 7.15 and Chap. **??**, the trees shown here are merely a representation using the syntax of *Lisp* that implements a single function. For instance, the upper-right (parent) tree implements $\frac{x_2 x_4}{x_3(x_4/x_1)}$. Such functions are used with an implied threshold or sign function when used for classification. Thus the function will operate on the features of a test pattern and emit category ω_i if the function is positive, and NOT ω_i otherwise.

graphical models, such as hidden Markov models and Bayes belief networks, have counterparts in structured Boltzmann networks, and this leads to new applications of Boltzmann learning.

Search methods based on evolution — genetic algorithms and genetic programming — perform highly parallel stochastic searches in a space set by the designer. The fundamental representation used in genetic algorithms is a string of bits, or chromosome; the representation in genetic programming is a snippet of computer code. Variation is introduced by means of crossover, mutation and insertion. As with all classification methods, the better the features, the better the solution. There are many heuristics that can be employed and parameters that must be set. As the cost of computation contiues to decline, computationally intensive methods, such as Boltzmann networks and evolutionary methods, should become increasingly popular.

Bibliographical and Historical Remarks

The general problem of search is of central interest in computer science and artificial intelligence, and is far to expansive to treat here. Nevertheless, techniques such as depth first, breadth first, branch-and-bound, A* [19], occassionally find use in fields touching upon pattern recognition, and practitioners should have at least a passing knowledge of them. Good overviews can be found in [33] and a number of textbooks on artificial intelligence, such as [46, 67, 55]. For rigor and completeness, Knuth's book on the subject is without peer [32].

The infinite monkey theorem, attributed to Sir Arthur Eddington, states that if there is a sufficiently large number of monkeys typing at typewriters, eventually one will bang out the script to *Hamlet*. It reflects one extreme of the tradeoff between prior knowledge about the location of a solution on the one hand and the effort of search required to fit it on the other. Computers made available in the early 1950s permitted the first automated attempts at highly stochastic search, most notably the pioneering work of Metropolis and colleagues for simulating chemical processes [40]. One of the earliest and most influential applications of stochastic methods for pattern recognition was the Pandemonium learning method due to Selfridge, which used stochastic search for input weights in a feed-forward network model [57]. Kirkpatrick, Gelatt and Vecchi [30], and independently Černý [64], introduced the Boltzmann factor to general stochastic search methods, the first example of simulated annealing. The statistical physics foundations of Boltzmann factors, at the present level of mathematical sophistication, can be found in [31]. The physical model of stochastic binary components was introduced by Wilhemlm Lenz in 1920, but became associated with his doctoral student Ernst Ising several years thereafter, and first called the "Ising model" in a paper by R. Peierls [50]. It has spawned a great deal of theoretical and simulation research [20].

The use of simulated annealing for learning was proposed by Ackley, Hinton and Sejnowski [2], a good book on the method is [1], which described the procedure for initializing the temperature in simulated annealing and was the inspiration for Fig. 7.10. Peterson and Anderson introduced deterministic annealing and mean-field Boltzmann learning and described some of the (rare) conditions when the mean-field approximation might lead to non-optimal solutions [51]. Hinton showed that the Boltzmann learning rule performs steepest descent in weight space for deterministic algorithm [21].

A number of papers explore structured Boltzmann networks, including Hopfield's influential paper on networks for pattern completion or auto-association [25]. The linear storage capacity of Hopfield networks quoted in the text, and $n \log n$ relationships for partial storage, are derived in [66, 39, 65]. The learning rule described in that work has roots in the *Learning matrix* of [59, 60]. Harmonium [58, 14], another two-layer variant of a Boltzmann network is primarily of historical interest. The relation of Boltzmann networks to graphical models such as Hidden Markov models has been explored in [27, 37] and [56], which was the source for our discussion in Sect. 7.4. Implementation of constraints for Boltzmann machines was introduced in [42] and a second-order pruning algorithm was described in [49].

Boltzmann learning has been applied to a number of real-world pattern recognition problems, most notably speech recognition [8, 52] and stochastic restoration of images or pattern completion [16]. Because Boltzmann learning has high computational burden yet a natural VLSI implementation, a number of special-purpose chips have been fabricated [23, 43, 44]. The ordering of configurations in Fig. 7.3, in which

neighboring configurations differe in just one bit, is a version of a Gray code; an elegant method for constructing such codes is described in [18, Sect. 5.16 – 5.17].

Some of the earliest work inspired by evolution was described in [12, 13], but the computational power available was insufficient for anything but toy problems. Later, Rechenberg's "evolution strategies" were applied to optimization in aeronautical design problems [53]. His earliest work did not employ full *populations* of candidate solutions, nor the key operation of crossover. Evolutionary programming saves good parents while evolutionary strategies generally does not. Neither employ mating, i.e., crossover. Holland introduced genetic algorithms in 1975 [24], and like the algorithm itself, researchers have explored a very wide range of problems in search, optimization and pattern recognition. A review appears in [6], and there is an increasing number of textbooks [17, 41], the latter with a more rigorous approach to the mathematics. Koza's extensive books on Genetic Programming provide a good introduction, and include several illustrative simulations [34, 35], though relatively little on pattern recognition. There are several collections of papers on evolutionary techniques in pattern recognition, including [48]. An intriguing effect due to the interaction of learning and evolution is the Baldwin effect, where learning can influence the rate of evolution [22]; it has been shown that too much learning (as well as too little learning) leads to slower evolution [28]. Evolutionary methods can lead to "non-optimal" or inelegant solutions, and there is computational evidence that this occurs in nature [61, 62].

Problems

⨁ *Section 7.1*

1. One version of the infinite monkey theorem states that a single (immortal) monkey typing randomly will ultimately reproduce the script of *Hamlet*. Estimate the time needed for this, assuming the monkey can type two characters per second, that the play has 50 pages, each containing roughly 80 lines, and 40 characters per line. Assume there are 30 possible characters (`a` through `z`), space, period, exclamation point and carriage return. Compare this time to the estimated age of the universe, 10^{10} years.

⨁ *Section 7.2*

2. Prove that for any optimization problem of the form of Eq. 1 having a non-symmetric connection matrix, there is an equivalent optimization problem in which the matrix is replaced by its symmetric part.

3. The complicated energy landscape in the left of Fig. 7.2 is misleading for a number of reasons.

(a) Discuss the difference between the continuous space shown in that figure with the discrete space for the true optimization problem.

(b) The figure shows a local minimum near the middle of the space. Given the nature of the discrete space, are any states closer to any "middle"?

(c) Suppose the axes referred to *continuous* variables s_i (as in mean-field annealing). If each s_i obeyed a sigmoid (Fig. 7.5), could the energy landscape be non-monotonic, as is shown in Fig. 7.2?

4. Consider exhaustive search for the minimum of the energy given in Eq. 1 for binary units and arbitrary connections w_{ij}. Suppose that on a uniprocessor it takes

10^{-8} seconds to calculate the energy for each configuration. How long will it take to exhaustively search the space for $N = 100$ units? How long for $N = 1000$ units?

5. Suppose it takes a uniprocessor 10^{-10} seconds to perform a single multiply-accumulate, $w_{ij}s_is_j$, in the calculation of the energy $E = -1/2 \sum_{ij} w_{ij}s_is_j$ given in Eq. 1.

(a) Make some simplifying assumptions and write a formula for the total time required to search exhaustively for the minimum energy in a fully connected network of N nodes.

(b) Plot your function using a log-log scale for $N = 1, \ldots, 10^5$.

(c) What size network, N, could be searched exhaustively in a day? A year? A century?

6. Make and justify any necessary mathematical assumptions and show analytically that at high temperature, every configuration in a network of N units interconnected by weights is equally likely (cf. Fig. 7.1).

7. Derive the exponential form of the Boltzmann factor in the following way. Consider an isolated set of $M + N$ independent magnets, each of which can be in an $s_i = +1$ or $s_i = -1$ state. There is a uniform magnetic field applied and this means that the energy of the $s_i = +1$ state has some positive energy, which we can arbitrarily set to 1; the $s_i = -1$ state has energy -1. The total energy of the system is therefore the sum of the number pointing up, k_u, minus the number pointing down, k_d; that is, $E_T = k_u - k_d$. (Of course, $k_u + k_d = M + N$ regardless of the total energy.)

The fundamental statistical assumptions describing this system are that the magnets are independent, and that the probability a subsystem (viz., the N magnets), has a particular energy is proportional to the *number of configurations* that have this energy.

(a) Consider the subsystem of N magnets, which has energy E_N. Write an expression for the number of configurations $K(N, E_N)$ that have energy E_N.

(b) As in part (a), write a general expression for the number of configurations in the subsystem M magnets at energy E_M, i.e., $K(M, E_M)$.

(c) Since the two subsystems consist of independent magnets, total number of ways the full system can have total energy $E_T = E_N + E_M$ is the product $K(N, E_N)K(M, E_M)$. Write an analytic expression for this total number.

(d) In statistical physics, if $M \gg N$, the M-magnet subsystem is called the heat reservoire or heat bath. Assume that $M \gg N$, and write a series expansion for your answer to part (c).

(e) Use your answer in part (d) to show that the probability the N-unit system has energgy E_N has the form of a Boltzmann factor, e^{-E_N}.

8. Prove that the analog value of s_i given by Eq. 5 is the expected value of a binary variable in temperature T in the following simple case. Consider a single binary magnet whose $s = +1$ state has energy $+E_0$ and $s = -1$ state has energy $-E_0$, as would occur if an external magnetic field has been applied.

(a) Construct the partition function Z by summing over the two possible states $\gamma' = 0$ and $\gamma' = 1$ according to Eq. 3.

(b) Recall that the probability of finding the system in state $s = +1$ is given by a Boltzmann factor divided by the partition function (Eq. 2). Define the (analog) expected value of the state to be

$$s = \mathcal{E}[s] = P(s = +1)(+1) + P(s = -1)(-1).$$

Show that this implies the analog state of a single magnet obeys Eq. 5.

(c) Argue that if the $N - 1$ other magnets in a large system can be assumed to give an average field (this is the mean-field approximation), then the analog value of a single magnet will obey a function of the form given in Eq. 5.

9. Consider Boltzmann networks applied to the exclusive-OR problem.

(a) A fully connected network consisting solely of two input units and a single output unit, whose sign gives the class, cannot solve the exclusive-OR problem. Prove this by writing a set of inequalities for the weights and show that they are inconsistent.

(b) As in part (a), prove that a fully connected Boltzmann network consisting solely of two input units and two output units representing the two categories cannot solve the exclusive-OR problem.

(c) Prove that a Boltzmann network of part (b) with a single hidden unit *can* implement the exclusive-OR problem.

10. Consider a fully-connected Boltzmann network with two input units, a single hidden unit and a single (category) output unit. Construct by hand a set of weights w_{ij} for $i, j = 1, 2, 3, 4$ which allows the net to solve the exclusive-OR problem for a representation in which $s_i = \pm 1$.

⊕ *Section 7.3*

11. Show all intermediate steps in the derivation of Eq. 14 from Eq. 12. Be sure your notation distinguishes this case from that leading to Eq. 10.

12. Train a six-unit Hopfield network with the following three patterns using the learning rule of Eq. 15.

$$\begin{aligned} \mathbf{x}^1 &= \{+1, +1, +1, -1, -1, -1\} \\ \mathbf{x}^2 &= \{+1, -1, +1, -1, +1, -1\} \\ \mathbf{x}^3 &= \{-1, +1, -1, -1, +1, +1\} \end{aligned}$$

(a) Verify that each of the patterns gives a local minium in energy by perturbing each of the six units individually and monitoring the energy.

(b) Verify that the symmetric state $s_i \rightarrow -s_i$ for $i = 1, \ldots, 6$ also gives a local energy minimum of the same energy.

13. Repeat Problem 12 but with the eight-unit network and the following patterns:

$$\begin{aligned} \mathbf{x}^1 &= \{+1, +1, +1, -1, -1, -1, -1, +1\} \\ \mathbf{x}^2 &= \{+1, -1, +1, +1, +1, -1, +1, -1\} \\ \mathbf{x}^3 &= \{-1, +1, -1, -1, +1, +1, -1, +1\} \end{aligned}$$

14. show that a missing feature assumes the appropriate value when training a deficient pattern in a Boltzmann network.

15. show how if constraints that a pattern is not in a set of categories improves the recognition for the others.

16. The text states a lower bound on the number of hidden units needed in a Boltzmann network trained with n patterns is $\lceil \log_2 n \rceil$. This is, of course, the number of hiddens needed to insure a distinct hidden representation for each pattern. Show that this lower bound is not tight, as there may not be weights to insure such a representation. Do this by considering a Boltzmann network with three input units, three hiddens and a single output, addressing the 3-bit parity problem.

(a) Argue that the hidden representation must be equivalent to the input representation.

(b) Argue that there is no two-layer Boltzmann network (here, hidden to output) that can solve the three-bit parity problem. Explain why this implies that the $\lceil \log_2 n \rceil$ bound is not tight.

17. Consider the problem of initializing the N weights in a fully connected Boltzmann network. Let there be $N - 1 \approx N$ weights connected to each unit. Suppose too that the chance that any particulat units will be in the $s_i = +1$ state is 0.5, and likewise for the $s_i = -1$ state. We seek weights such that the variance of the net activation of each unit is roughly 1.0, a reasonable measure of the end of the linear range of the sigmoid nonlinearity. The variance of l_i is

$$\text{VAR}[\text{l}_i] = \sum_{j=1}^{N} VAR[w_{ij}s_j] = NVAR[w_{ij}]VAR[s_j]..$$

Set $VAR[l_i] = 1$, and solve for $VAR[w_{ij}]$ and thereby show that weights should be initialized randomly in the range $-1\sqrt{3/N} < w_{ij} < +\sqrt{3/N}$.

18. Show that under reasonable conditions, the learning rate η in Eq. 14 for a Boltzmann network of N units should be bounded $\eta \leq T^2/N$ to insure stability as follows:

(a) Take the derivative of Eq. 14 to prove that the Hessian is

$$\begin{aligned} \mathbf{H} &= \frac{\partial^2 \bar{D}_{KL}}{\partial \mathbf{w}^2} = \frac{\partial^2 \bar{D}_{KL}}{\partial w_{ij} \partial w_{uv}} \\ &= \frac{1}{T^2} \left[\mathcal{E}[s_i s_j s_u s_v] - \mathcal{E}[s_i s_j]\mathcal{E}[s_u s_v] \right]. \end{aligned}$$

(b) Use this to show that

$$\mathbf{w}^t \mathbf{H} \mathbf{w} \leq \frac{1}{T^2} \mathcal{E} \left[\left(\sum_{ij} |w_{ij}| \right)^2 \right].$$

(c) Suppose we normalize weights such that $\|\mathbf{w}\| = 1$ and thus

$$\sum_{ij} w_{ij} \leq \sqrt{N}.$$

Use this fact together with your answer to part (b) to show that the curvature of the $\bar{D}_{KL}$ obeys

$$\mathbf{w}^t \mathbf{H} \mathbf{w} \leq \frac{1}{T^2} \mathcal{E} \left[\left(\sqrt{N} \right)^2 \right] = \frac{N}{T^2}.$$

(d) Use the fact that stability demands the learning rate to be the inverse of the curvature, along with your answer in (c), to show that the learning rate should be bounded $\eta \leq T^2/N$.

⊕ *Section 7.4*

19. For any HMM, there exists a Boltzmann chain that implements the equivalent probability model. Show the converse is not true, that is, for every chain, there exists an HMM. Use the fact that weights in a Boltzmann chain are bounded $-\infty < A_{ij}, B_{jk} < +\infty$, but probabilities in an HMM are positive and sum to 1.

20. For a Boltzmann chain with T_f steps, c hidden units and xx visible units, how many legal paths are there (cf. Fig. 7.11).

21. The discussion of the relation between Boltzmann chains and hidden Markov models in the text assumed the initial hidden state was known. Show that if this hidden state is not known, the energy of Eq. 21 has another term which describes the prior probability the system is in a particular hidden state.

⊕ *Section 7.5*

22. Consider the populations of size L of N-bit chromosomes.

(a) Show the number of different populations is $\binom{L+2^N-1}{2^N-1}$.

(b) Assume some number $1 \leq L_s \leq L$ are selected for reproduction in a given generation. Use your answer to part (a) to write an expression for the number of possible sets of parents as a function of L and L_a. (It is just the set, not their order that is relevant.)

(c) Show that your answer to part (b) reduces to that in part (a) for the case $L_a = L$.

(d) Show that your answer to part (b) gives L in the case $L_a = 1$.

⊕ *Section 7.6*

23. For each of the below snippets, mark suitable positions for breaks for the crossover operator.

(a) `(* (X0 (+ x4 x8)) x5 (SQRT 5))`

(b) `(SQRT ( X0 (+ x4 x8)))`

(c) `(* (- (SIN X0) (* (TAN 3.4) (SQRT X4)))`

(d) `(* (X0 (+ x4 x8)) x5 (SQRT 5))`

(e) Separate the following Lisp symbols into groups such that any member in a group can be replaced by another through the mutation operator in genetic programming:
`{+, X3, NOR, *, X0, 5.5, SQRT, /, X5, SIN, -, -4.5, NOT, OR, 2.7, TAN}`

Computer exercises

Several of the exercises use the data in the following table.

ω_1	ω_2
xxxxx	xxxxx
xxxxx	xxxxx
xxxxx	xxxxx
xxxxx	xxxxx
xxxxx	xxxxx
xxxxx	xxxxx
xxxxx	xxxxx
xxxxx	xxxxx
xxxxx	xxxxx
xxxxx	xxxxx

⨁ *Section 7.2*

1. Consider the problem of searching for a global minimum of the energy given in Eq. 1 for a system of N units, fully interconnected by weights randomly chosen in the range $-1/\sqrt{N} < w_{ij} < +1/\sqrt{N}$. Let $N = 10$.

(a) Write a program to search through all 2^N configurations to find global minima, and apply it to your network. Verify that there are two "global" minima.

(b) Write a program to perform the following version of gradient descent. Let the units be numbered and ordered $i = 1, \ldots, N$ for bookkeeping. For each configuration, find the unit with the lowest index i which can be changed to lower the total energy. Iteratively make this change until the system converges, or it is clear that it will not converge.

(c) Perform a search as in part (b) but with *random* polling of units.

(d) Repeat parts (a – c) for $N = 100$ and $N = 1000$.

(e) Discuss your results, paying particular attention to convergence and the problem of local minima.

2. Algorithm 1

⨁ *Section 7.3*

3. Train a Boltzmann network consisting of eight input units and ten category units with the characters of a seven-segment display shown in Fig. 7.10.

(a) Use the network to classify each of the ten patterns, and thus verify that all have been learned.

(b) Explore pattern completion in your network the following way. For each of the 2^8 possible patterns do pattern completion for several characters. Add hidden units and show that better performance results for ambiguous characters

4. ;laskjdf

⊕ *Section 7.4*

5. lskjdf

⊕ *Section 7.5*

6. lksdfj

⊕ *Section 7.6*

7. Consider a two-category problem with four features bounded region $-1 \leq x_i \leq +1$ for $i = 1, 2, 3, 4$.

(a) Generate training points in each of two categories defined by

$$\begin{aligned} \omega_1 : &\quad x_1 + 0.5x_2 - 0.3x_3 - 0.1x_4 < 0.5 \\ \omega_2 : &\quad -x_1 + 0.2x_2 + x_3 - 0.6x_4 < 0.2 \end{aligned}$$

by randomly selecting a point in the four-dimensional space. If it satisfies neither of the two inequalities, delete the point. If it satisfies just one of the inequalities, label its category accordingly. If it satisfies both inequalities, randomly choose a label with probability 0.5. If it satisfies neither of the inequalities, discard the point. Continue in this way until you have 50 points for each category.

(b) GP

Bibliography

[1] Emile Aarts and Jan Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. Wiley, New York, NY, 1989.

[2] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

[3] Rudolf Ahlswede and Ingo Wegener. *Search Problems*. Wiley, New York, NY, 1987.

[4] Frantzisko Xabier Albizuri, Alicia d'Anjou, Manuel Graña, Francisco Javier Torrealdea, and Mari Carmen Hernandez. The high-order Boltzmann machine: Learned distribution and topology. *IEEE Transactions on Neural Networks*, TNN-6(3):767–770, 1995.

[5] David Andre, Forrest H. Bennett III, and John R. Koza. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 3–11, Cambridge, MA, 1996. MIT Press.

[6] Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In Rik K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, San Mateo, CA, 1991.

[7] J. Mark Baldwin. A new factor in evolution. *American Naturalist*, 30:441–451, 536–553, 1896.

[8] John S. Bridle and Roger K. Moore. Boltzmann machines for speech pattern processing. *Proceedings of the Institute of Acoustics*, 6(4):315–322, 1984.

[9] Lawrence Davis, editor. *Genetic Algorithms and Simulated Annealing*. Research Notes in Artificial Intelligence. Morgan Kaufmann, Los Altos, CA, 1987.

[10] Lawrence Davis, editor. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, NY, 1991.

[11] Michael de la Maza and Bruce Tidor. An analysis of selection procedures with particular attention paid to proportional and Boltzmann selection. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 124–131, San Mateo, CA, 1993. Morgan Kaufmann.

[12] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, NY, 1966.

[13] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Intelligence through Simulated Evolution*. Wiley, New York, NY, updated and expanded edition, 1999.

[14] Yoav Freund and David Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. In John E. Moody, Stephen J. Hanson, and Richard P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 912–919, San Mateo, CA, 1992. Morgan Kaufmann.

[15] Conrad C. Galland. The limitations of deterministic Boltzmann machine learning. *Network*, 4(3):355–379, 1993.

[16] Stewart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, 1984.

[17] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[18] Richard W. Hamming. *Coding and Information Theory*. Prentice-Hall, Englewood Cliffs, NJ, second edition, 1986.

[19] Peter E. Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.

[20] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, Redwood City, CA, 1991.

[21] Geoffrey E. Hinton. Deterministic Boltzmann learning performs steepest descent in weight space. *Neural Computation*, 1(1):143–150, 1989.

[22] Geoffrey E. Hinton and Stephen J. Nowlan. How learning can guide evolution. *Complex Systems*, 1(1):495–502, 1987.

[23] Yuzo Hirai. Hardware implementation of neural networks in Japan. *Neurocomputing*, 5(1):3–16, 1993.

[24] John H. Holland. *Adaptation in Natural and Artificial Systems: An introductory analysis with applications to biology, control and artificial intelligence*. MIT Press, Cambridge, MA, second edition, 1992.

[25] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the USA*, 79(8):2554–2558, 1982.

[26] Hugo Van Hove and Alain Verschoren. Genetic algorithms and trees I: Recognition trees (the fixed width case). *Computers and Artificial Intelligence*, 13(5):453–476, 1994.

[27] Michael I. Jordan, editor. *Learning in Graphical Models*. MIT Press, Cambridge, MA, 1999.

[28] Ron Keesing and David G. Stork. Evolution and learning in neural networks: The number and distribution of learning trials affect the rate of evolution. In Richard P. Lippmann, John E. Moody, and David S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 804–810, San Mateo, CA, 1991. Morgan Kaufmann.

[29] James D. Kelly, Jr. and Lawrence Davis. A hybrid genetic algorithm for classification. In Raymond Reiter and John Myopoulos, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 645–650, San Mateo, CA, 1991. Morgan Kaufmann.

[30] Scott Kirkpatrick, C. Daniel Gelatt, Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[31] Charles Kittel and Herbert Kroemer. *Thermal Physics*. Freeman, San Francisco, CA, second edition, 1980.

[32] Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, MA, 1 edition, 1973.

[33] Richard E. Korf. Optimal path finding algorithms. In Laveen N. Kanal and Vipin Kumar, editors, *Search in Artificial Intelligence*, chapter 7, pages 223–267. Springer-Verlag, Berlin, Germany, 1988.

[34] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[35] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA, 1994.

[36] Vipin Kumar and Laveen N. Kanal. The cdp: A unifying formulation for heuristic search, dynamic programming, and branch bound procedures. In Laveen N. Kanal and Vipin Kumar, editors, *Search in Artificial Intelligence*, pages 1–27. Springer-Verlag, New York, NY, 1988.

[37] David J. C. MacKay. Equivalence of Boltzmann chains and hidden Markov models. *Neural Computation*, 8(1):178–181, 1996.

[38] Robert E. Marmelstein and Gary B. Lamont. Pattern classification using a hybrid genetic program decision tree approach. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 223–231, San Mateo, CA, 1998. Morgan Kaufmann.

[39] Robert J. McEliece, Edward C. Posner, Eugene R. Rodemich, and Santosh S. Venkatesh. The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory*, IT-33(4):461–482, 1985.

[40] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[41] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1996.

[42] John Moussouris. Gibbs and Markov random systems with constraints. *Journal of Statistical Physics*, 10(1):11–33, 1974.

[43] Michael Murray, James B. Burr, David G. Stork, Ming-Tak Leung, Kan Boonyanit, Gregory J. Wolff, and Allen M. Peterson. Deterministic Boltzmann machine VLSI can be scaled using multi-chip modules. In Jose Fortes, Edward Lee, and Teresa Meng, editors, *Proceedings of the International Conference on Application-Specific Array Processors ASAP-92*, volume 9, pages 206–217, Los Alamitos, CA, 1992. IEEE Press.

[44] Michael Murray, Ming-Tak Leung, Kan Boonyanit, Kong Kritayakirana, James B. Burr, Greg Wolff, David G. Stork, Takahiro Watanabe, Ed Schwartz, and Allen M. Peterson. Digital Boltzmann VLSI for constraint satisfaction and learning. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 896–903, Cambridge, MA, 1994. MIT Press.

[45] Dana S. Nau, Vipin Kumar, and Laveen N. Kanal. General branch and bound, and its relation to A* and AO*. *Artificial Intelligence*, 23(1):29–58, 1984.

[46] Nils J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, San Mateo, CA, 1998.

[47] Mattias Ohlsson, Carsten Peterson, and Bo Söderberg. Neural networks for optimization problems with inequality constraints: The knapsack problem. *Neural Computation*, 5(2):331–339, 1993.

[48] Sankar K. Pal and Paul P. Wang, editors. *Genetic Algorithms for Pattern Recognition*. CRC Press, Boca Raton, FL, 1996.

[49] Morton With Pederson and David G. Stork. Pruning Boltzmann networks and Hidden Markov Models. In *Thirteenth Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 258–261, New York, NY, 1997. IEEE Press.

[50] Rudolf Peierls. On Ising's model of ferromagnetism. *xxxx*, 1936.

[51] Carsten Peterson and James R. Anderson. A mean-field theory learning algorithm for neural networks. *Complex Systems*, 1(5):995–1019, 1987.

[52] Richard W. Prager, Tim D. Harrison, and Frank Fallside. Boltzmann machines for speech recognition. *Computer Speech and Language*, 1(1):3–27, 1986.

[53] Ingo Rechenberg. Bionik, evolution und optimierung (in German). *Naturwissenschaftliche Rundschau*, 26(11):465–472, 1973.

[54] Ingo Rechenberg. Evolutionsstrategie – optimierung nach prinzipien der biologischen evolution (in German). In Jörg Albertz, editor, *Evolution und Evoultionsstrategien in Biologie, Technik und Gesellschaft*, pages 25–72. Freie Akademie, 1989.

[55] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall Series in Artificial Intelligence. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[56] Lawrence K. Saul and Michael I. Jordan. Boltzmann chains and Hidden Markov Models. In Gerald Tesauro, David S. Touretzky, and Todd K. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 435–442, Cambridge, MA, 1995. MIT Press.

[57] Oliver G. Selfridge. Pandemonium: a paradigm for learning. In *Mechanisation of Thought Processes: Proceedings of a Symposium held at the National Physical Laboratory*, pages 513–526, London, UK, 1958. HMSO.

[58] Paul Smolensky. Information processing in dynamical systems: Foundations of Harmony theory. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing*, volume 1, chapter 6, pages 194–281. MIT Press, Cambridge, MA, 1986.

[59] Karl Steinbuch. Die lernmatrix (in German). *Kybernetik (Biological Cybernetics)*, 1(1):36–45, 1961.

[60] Karl Steinbuch. *Automat und Mensch (in German)*. Springer, New York, NY, 1971.

[61] David G. Stork, Bernie Jackson, and Scott Walker. Non-optimality via preadaptation in simple neural systems. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, pages 409–429. Addison Wesley, Reading, MA, 1992.

[62] David G. Stork, Bernie Jackson, and Scott Walker. Nonoptimality in a neurobiological system. In Daniel S. Levine and Wesley R. Elsberry, editors, *Optimality in Biological and Artificial Networks?*, pages 57–75. Lawrence Erlbaum Associates, Mahwah, NJ, 1997.

[63] Harold Szu. Fast simulated annealing. In John S. Denker, editor, *Neural Networks for Computing*, pages 420–425, New York, NY, 1986. American Institute of Physics.

[64] Vladimír Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[65] Santosh S. Venkatesh and Demetri Psaltis. Linear and logarithmic capacities in associative neural networks. *IEEE Transactions on Information Theory*, IT-35(3):558–568, 1989.

[66] Gérard Weisbuch and Françoise Fogelman-Soulié. Scaling laws for the attractors of Hopfield networks. *Journal of Physics Letters*, 46(14):623–630, 1985.

[67] Patrick Henry Winston. *Artificial Intelligence*. Addison-Wesley, Reading, MA, third edition, 1992.

Index