the known types of networks would actually generalize to the full XOR. Nevertheless neural networks commonly make very useful generalizations that would be judged sensible in human terms.

Programming

Like most of the work done in neural networks, much of this book is concerned with the problem of programming or learning: how do we choose the connection weights so the network can do a specific task?

We will encounter some examples where we can choose the weights *a priori* if we are a little clever. This **embeds** some information into the network by design. But such problems are the exception rather than the rule. In other cases we can often "teach" the network to perform the desired computation by iterative adjustments of the w_{ij} strengths. This may be done in two main ways:

- **Supervised learning.** Here the learning is done on the basis of direct comparison of the output of the network with known correct answers. This is sometimes called **learning with a teacher**. It includes the special case of **reinforcement learning**, where the only feedback is whether each output is correct or incorrect, not what the correct answer is.
- **Unsupervised learning.** Sometimes the learning goal is not defined at all in terms of specific correct examples. The only available information is in the correlations of the input data or signals. The network is expected to create categories from these correlations, and to produce output signals corresponding to the input category.

There are many exciting implications of the possibility of *training* a network to do a computation. Instead of having to specify every detail of a calculation, we simply have to compile a training set of representative examples. This means that we can hope to treat problems where appropriate rules are very hard to know in advance, as in expert systems and robotics. It may also spare us a lot of tedious (and expensive) software design and programming even when we do have explicit rules. John Denker has remarked that "neural networks are the second best way of doing just about anything." The *best* way is to find and use the right rules or the optimum algorithm for each particular problem, but this can be inordinately expensive and time consuming. There is plenty of scope for a second best approach based on learning by example.

TWO
The Hopfield Model

2.1 The Associative Memory Problem

Associative memory is the "fruit fly" or "Bohr atom" problem of this field. It illustrates in about the simplest possible manner the way that collective computation can work. The basic problem is this:

> Store a set of p patterns ξ_i^μ in such a way that when presented with a new pattern ζ_i, the network responds by producing whichever one of the stored patterns most closely resembles ζ_i.

The patterns are labelled by $\mu = 1, 2, \ldots, p$, while the units in the network are labelled by $i = 1, 2, \ldots, N$. Both the stored patterns ξ_i^μ and the test patterns ζ_i can be taken to be either 0 or 1 on each site i, though we will adopt a different convention shortly.

We could of course do this serially in a conventional computer simply by storing a list of the patterns ξ_i^μ, writing a program which computed the Hamming distance[1]

$$\sum_i [\xi_i^\mu(1-\zeta_i) + (1-\xi_i^\mu)\zeta_i] \tag{2.1}$$

between the test pattern ζ_i and each of the stored patterns, finding which of them was smallest, and printing the corresponding stored pattern out.

Here we want to see how to get a McCulloch-Pitts network to do it. That is, if we *start* in the configuration $n_i = \zeta_i$, we want to know what (if any) set of w_{ij}'s

[1]The Hamming distance between two binary numbers means the number of bits that are different in the two numbers.

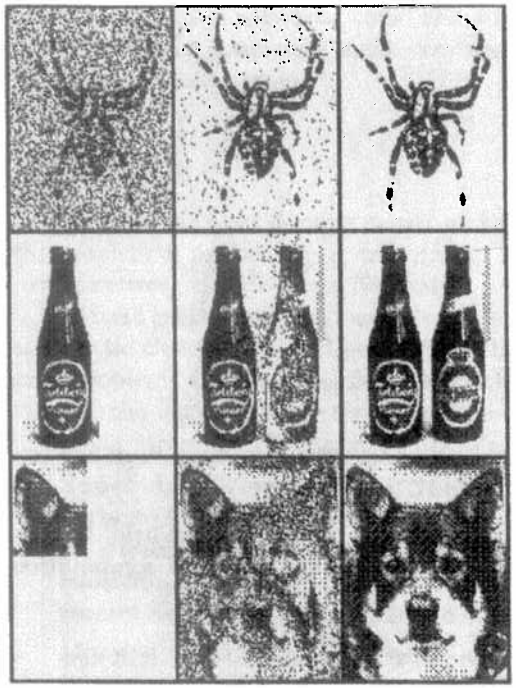

FIGURE 2.1 Example of how an associative memory can reconstruct images. These are binary images with 130 x 180 pixels. The images on the right were recalled by the memory after presentation of the corrupted images shown on the left. The middle column shows some intermediate states. A sparsely connected Hopfield network with seven stored images was used.

will make the network go to the state with $n_i = \xi_i^{\mu_0}$, where it is pattern number μ_0 that is the smallest distance (2.1) from ζ_i. Thus we want the memory to be **content-addressable** and insensitive to small errors in the input pattern.

A content-addressable memory can be quite powerful. Suppose, for example, we store coded information about many famous scientists in a network. Then the starting pattern "evolution" should be sufficient to recall everything about Darwin, and "$E = mc^3$" should recall Einstein, despite the error in the input pattern. Note that *some* pattern will always be retrieved for any clue (unless we invent a "don't know" pattern); the network will never retrieve a linear combination of, say, Darwin and Wallace in response to "evolution" but will pick the best match according to what has been stored. This depends on the nonlinearity of the network, and obviously has advantages for many practical applications.

Other common examples of applications for an associative memory are recognition and reconstruction of images (see Fig. 2.1), and retrieval of bibliographic information from partial references (such as from an incomplete title of a paper).

Figure 2.2 shows schematically the function of the dynamic associative (or content-addressable) memories that we construct in this chapter. The space of all possible states of the network—the **configuration space**—is represented by the region drawn. Within that space the stored patterns ξ_i^μ are **attractors**. The dy-

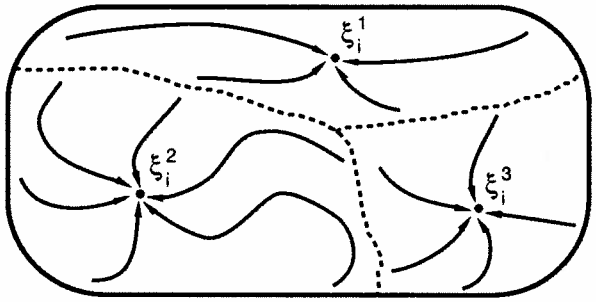

FIGURE 2.2 Schematic configuration space of a model with three attractors.

basins of attraction of the different attractors. This picture is very idealized, and in particular the space should really be a discrete set of points (on a hypercube), not a continuous region. But it is nevertheless a very useful image to keep in mind.

In the next section we treat the basic Hopfield [1982] model of associative memory. In Section 2.3 we turn to statistical mechanics, studying some magnetic systems that are analogous to our networks. Then in Section 2.4 we define a stochastic version of the original model, and analyze it using statistical mechanics methods. Finally, Section 2.5 presents a heuristic derivation of the famous $0.138N$ capacity of the Hopfield model. Various embellishments and generalizations of the basic model are discussed in the next chapter.

2.2 The Model

For mathematical convenience we now transform to a formulation where the activation values of the units are +1 (firing) and −1 (not firing) instead of 1 and 0. We denote[2] them by S_i rather than n_i. Conversion to and from the $n_i = 0$ or 1 notation is easy via $S_i = 2n_i - 1$. The dynamics of the network corresponding to (1.1) or (1.3) now reads

$$S_i := \operatorname{sgn}\Big(\sum_j w_{ij} S_j - \theta_i\Big) \tag{2.2}$$

where we take the sign function $\operatorname{sgn}(x)$ (illustrated in Fig. 2.3) to be

$$\operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0; \\ -1 & \text{if } x < 0; \end{cases} \tag{2.3}$$

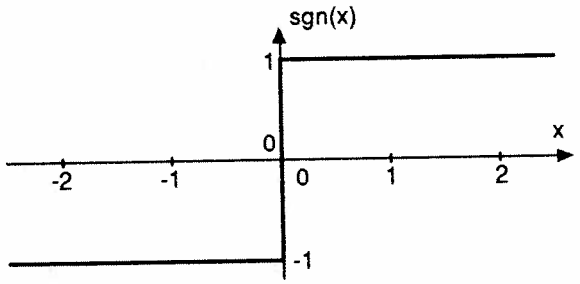

FIGURE 2.3 The function $\mathrm{sgn}(x)$.

and the threshold θ_i is related to the μ_i in (1.1) by $\theta_i = 2\mu_i - \sum_j w_{ij}$. In the rest of this chapter we drop these threshold terms, taking $\theta_i = 0$, because they are not useful with the random patterns that we will consider. Thus we use

$$S_i := \mathrm{sgn}\Big(\sum_j w_{ij} S_j\Big) . \tag{2.4}$$

There are at least two ways in which we might carry out the updating specified by (2.4). We could do it *synchronously*, updating all units simultaneously at each time step. Or we could do it *asynchronously*, updating them one at a time. Both kinds of models are interesting, but the asynchronous choice is more natural for both brains and artificial networks. The synchronous choice requires a central clock or pacemaker, and is potentially sensitive to timing errors. In the asynchronous case, which we adopt henceforth, we can proceed in either of two ways:

- At each time step, select at random a unit i to be updated, and apply the rule (2.4).
- Let each unit independently choose to update itself according to (2.4), with some constant probability per unit time.

These choices are equivalent (except for the distribution of update intervals) because the second gives a random sequence; there is vanishingly small probability of two units choosing to update at exactly the same moment. The first choice is appropriate for simulation, with central control, while the second is appropriate for autonomous hardware units.

We also have to specify for how long (for how many updatings) we will allow the network to evolve before demanding that its units' values give the desired stored pattern. One possibility in the case of *synchronous* updating is to require that the network go to the correct memorized pattern right away on the first iteration. In the present discussion (using asynchronous updating) we demand only that the network settle eventually into a stable configuration—one for which no S_i changes any more.

Rather than study a specific problem such as memorizing a particular set of pictures, we examine the more generic problem of a *random* set of patterns drawn

up of independent bits ξ_i which can each take on the values $+1$ and -1 with equal probability. More general situations are discussed in Section 3.2.

Our procedure for testing whether a proposed form of w_{ij} is acceptable is first to see whether the patterns to be memorized are themselves stable, and then to check whether small deviations from these patterns are corrected as the network evolves.

One Pattern

To motivate our choice for the connection weights, we consider first the simple case where there is just one pattern ξ_i that we want to memorize. The condition for this pattern to be stable is just

$$\mathrm{sgn}\Big(\sum_j w_{ij}\xi_j\Big) = \xi_i \qquad \text{(for all } i\text{)} \tag{2.5}$$

because then the rule (2.4) produces no changes. It is easy to see that this is true if we take

$$w_{ij} \propto \xi_i \xi_j \tag{2.6}$$

since $\xi_j^2 = 1$. For later convenience we take the constant of proportionality to be $1/N$, where N is the number of units in the network, giving

$$w_{ij} = \frac{1}{N}\xi_i\xi_j . \tag{2.7}$$

Furthermore, it is also obvious that even if a number (fewer than half) of the bits of the starting pattern S_i are wrong (i.e., not equal to ξ_i), they will be overwhelmed in the sum for the net input

$$h_i = \sum_j w_{ij} S_j \tag{2.8}$$

by the majority that are right, and $\mathrm{sgn}(h_i)$ will still give ξ_i. An initial configuration near (in Hamming distance) to ξ_i will therefore quickly relax to ξ_i. This means that the network will correct errors as desired, and we can say that the pattern ξ_i is an **attractor**.

Actually there are two attractors in this simple case; the other one is at $-\xi_i$. This is called a **reversed state**. All starting configurations with *more* than half the bits different from the original pattern will end up in the reversed state. The configuration space is symmetrically divided into two basins of attraction, as shown in Fig. 2.4.

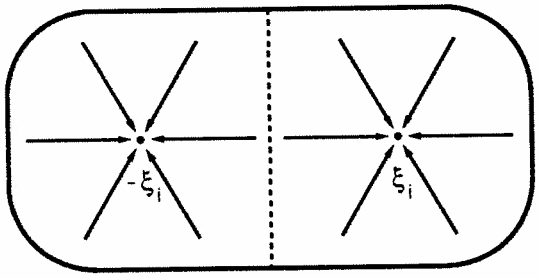

FIGURE 2.4 Schematic configuration space for the one pattern case, including the reversed state.

Many Patterns

This is fine for one pattern, but how do we get the system to recall the most similar of many patterns? The simplest answer is just to make w_{ij} a superposition of terms like (2.7), one for each pattern:

$$w_{ij} = \frac{1}{N}\sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu \,. \tag{2.9}$$

Here p is the total number of stored patterns labelled by μ.

This is usually called the "Hebb rule" or the "generalized Hebb rule" because of the similarity between (2.9) and a hypothesis made by Hebb [1949] about the way in which synaptic strengths in the brain change in response to experience: Hebb suggested changes proportional to the correlation between the firing of the pre- and post-synaptic neurons. If we apply our set of patterns ξ_i^μ to the network during a **training phase**, and adjust the w_{ij} strengths according to such pre/post correlations, we arrive directly at (2.9). Technically, however, (2.9) goes beyond Hebb's original hypothesis because it changes the weights positively when neither of the units is firing ($\xi_i^\mu = \xi_j^\mu = -1$). This is probably not physiologically reasonable. Equation (2.9) can even cause a particular connection to change from excitatory to inhibitory or vice versa as more patterns are added, which is never believed to occur at real synapses. It is possible to modify the equation in various ways to remedy these defects [Toulouse et al., 1986], but here we use the simple form (2.9) unchanged.

An associative memory model using the Hebb rule (2.9) for all possible pairs ij, with binary units and asynchronous updating, is usually called a **Hopfield model**. The term is also applied to various generalizations discussed in the next chapter. Although most of the ingredients of the model were known earlier, Hopfield's influential paper [Hopfield, 1982] brought them together, introduced an energy function, and emphasized the idea of stored memories as dynamical attractors. Earlier related models, often also using the Hebb rule, are reviewed by Cowan and Sharp [1988a, b]. Particularly important is the Little model [Little, 1974; Little and Shaw, 1975, 1978], which is based however on *synchronous* updating.

Let us examine the stability of a particular pattern ξ_i^ν. The stability condition (2.5) generalizes to

$$\mathrm{sgn}(h_i^\nu) = \xi_i^\nu \qquad \text{(for all } i\text{)} \tag{2.10}$$

where the net input h_i^ν to unit i in pattern ν is

$$h_i^\nu \equiv \sum_j w_{ij}\xi_j^\nu = \frac{1}{N}\sum_j\sum_\mu \xi_i^\mu \xi_j^\mu \xi_j^\nu \,. \tag{2.11}$$

We now separate the sum on μ into the special term $\mu = \nu$ and all the rest:

$$h_i^\nu = \xi_i^\nu + \frac{1}{N}\sum_j\sum_{\mu\neq\nu} \xi_i^\mu \xi_j^\mu \xi_j^\nu \,. \tag{2.12}$$

If the second term were zero, we could immediately conclude that pattern number ν was stable according to (2.10). This is still true if the second term is small enough: *if its magnitude is smaller than 1 it cannot change the sign of* h_i^ν, *and* (2.10) *will still be satisfied.*

It turns out that the second term, which we call the **crosstalk term**, *is* less than 1 in many cases of interest if p (the number of patterns) is small enough. We will discuss the details shortly; let us assume for now that the crosstalk term is small enough for all i and ν. Then the stored patterns are all stable—if we start the system from one of them it will stay there. Furthermore, a small fraction of bits different from a stored pattern will be corrected in the same way as in the single-pattern case; they are overwhelmed in the sum $\sum_j w_{ij}S_j$ by the vast majority of correct bits. A configuration near (in Hamming distance) to ξ_i^ν thus relaxes to ξ_i^ν. This shows that the chosen patterns are truly attractors of the system, as already anticipated in Fig. 2.2. The system works as expected as a content-addressable memory.

Storage Capacity

Consider the quantity

$$C_i^\nu \equiv -\xi_i^\nu \frac{1}{N}\sum_j\sum_{\mu\neq\nu} \xi_i^\mu \xi_j^\mu \xi_j^\nu \,. \tag{2.13}$$

This is just $-\xi_i^\nu$ times the crosstalk term in (2.12). If C_i^ν is negative the crosstalk term has the same sign as the desired ξ_i^ν term, and thus does no harm. But if C_i^ν is positive and larger than 1, it changes the sign of h_i^ν and makes bit (or unit) i of pattern ν unstable; if we start the system in the desired memory state ξ_i^ν, it will *not* stay there.

The C_i^ν's just depend on the patterns ξ_j^μ that we attempt to store. For now we consider purely *random* patterns, with equal probability for $\xi_j^\mu = +1$ and for

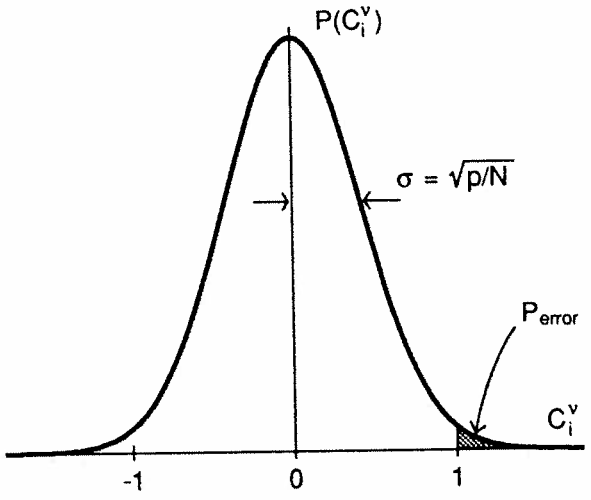

FIGURE 2.5 The distribution of values for the crosstalk C_i^ν given by (2.13). For p random patterns and N units this is a Gaussian with variance $\sigma^2 = p/N$. The shaded area is P_{error}, the probability of error per bit.

$\xi_j^\mu = -1$, independently for each j and μ. Then we can estimate the probability P_{error} that any chosen bit is unstable:

$$P_{\text{error}} = \text{Prob}(C_i^\nu > 1)\,. \tag{2.14}$$

Clearly P_{error} increases as we increase the number p of patterns that we try to store. Choosing a criterion for acceptable performance (e.g., $P_{\text{error}} < 0.01$) lets us find the storage **capacity** $p_{\max}$ of the network: the maximum number of patterns that can be stored without unacceptable errors. As we will see, there are actually several different expressions for $p_{\max}$, depending on the type of criterion we use for P_{error}.

Let us first calculate P_{error}. It depends on the number of units N and the number of patterns p. We assume that both N and p are large compared to 1, because this is typically the case and because it makes the mathematics easier. Now C_i^ν is $1/N$ times the sum of about Np independent random numbers,[3] each of which is $+1$ or -1. From the theory of random coin tosses [Feller, 1968] it has therefore a **binomial distribution** with mean zero and variance $\sigma^2 = p/N$. But since Np is assumed large this can be approximated by a Gaussian distribution with the same mean and variance, as shown in Fig. 2.5.

The probability P_{error} that C_i^ν exceeds 1 is just the shaded area in Fig. 2.5. Thus

$$\begin{aligned} P_{\text{error}} &= \frac{1}{\sqrt{2\pi}\sigma}\int_1^\infty e^{-x^2/2\sigma^2}\,dx \\ &= \tfrac{1}{2}\left[1-\text{erf}\left(1/\sqrt{2\sigma^2}\right)\right] = \tfrac{1}{2}\left[1-\text{erf}\left(\sqrt{N/2p}\right)\right] \end{aligned} \tag{2.15}$$

TABLE 2.1 Capacities

P_{error}	$p_{\max}/N$
0.001	0.105
0.0036	0.138
0.01	0.185
0.05	0.37
0.1	0.61

where the **error function** $\text{erf}(x)$ is defined by

$$\text{erf}(x) \equiv \frac{2}{\sqrt{\pi}}\int_0^x \exp(-u^2)\,du\,. \tag{2.16}$$

Table 2.1 shows the values of p/N required to obtain various values of P_{error}. Thus if we choose the criterion $P_{\text{error}} < 0.01$ for example, we arrive at $p_{\max} = 0.15N$.

This calculation only tells us about the *initial* stability of the patterns. If we choose $p < 0.185N$ for example, it tells us that no more than 1% of the pattern bits will be unstable initially. But if we start the system in a particular pattern ξ_i^ν and about 1% of the bits flip, what happens next? It may be that the first few flips will cause more bits to flip. In the worst case there could be an avalanche phenomenon in which more and more bits flip until the final state bears little or no resemblance to the original pattern. So our estimates for $p_{\max}$ are upper bounds; smaller values may be required to keep the final attractors close to the desired patterns. The more sophisticated calculation given in Section 2.5 deals with this problem, and shows that an avalanche occurs if $p > 0.138N$, making the whole memory useless. Thus $p_{\max} = 0.138N$ if we are willing to accept the errors that occur up to that point. At $p = 0.138N$ table 2.1 shows that only 0.37% of the bits will be unstable initially, though it turns out that about 1.6% of them flip before a stable attractor is reached.

An alternative definition of the capacity insists that most of the memories be recalled *perfectly*. Since each pattern contains N bits, we need $P_{\text{error}} < 0.01/N$ to get all N bits right with 99% probability.[4] This clearly implies $p/N \to 0$ as $N \to \infty$, so we can use the asymptotic expansion of the error function

$$1 - \text{erf}(x) \;\to\; e^{-x^2}/\sqrt{\pi}x \qquad (\text{as } x \to \infty) \tag{2.17}$$

to obtain

$$\log(P_{\text{error}}) \approx -\log 2 - N/2p - \tfrac{1}{2}\log\pi - \tfrac{1}{2}\log(N/2p)\,. \tag{2.18}$$

This turns the condition $P_{\text{error}} < 0.01/N$ into

$$-\log 2 - N/2p - \tfrac{1}{2}\log\pi - \tfrac{1}{2}\log(N/2p) < \log 0.01 - \log N \tag{2.19}$$

[4] Strictly speaking we should write $(1 - P_{\text{error}})^N > 0.99$ here, but $P_{\text{error}} < 0.01/N$ is a good

or, taking only the leading terms for large N,

$$N/2p > \log N \tag{2.20}$$

giving the capacity $p_{\max} = N/2 \log N$ for this case.

Even more stringently, we could ask that *all* the patterns be recalled perfectly. This requires us to get Np bits right with, say, 99% probability, and so needs $P_{\text{error}} < 0.01/pN$. It is easy to see that this changes (2.20) to

$$N/2p > \log(Np) \tag{2.21}$$

which gives $p_{\max} = N/4 \log N$ because $\log(Np) \sim \log N^2 = 2 \log N$ in leading order.

Note that we have assumed in the perfect recall cases that the C_i^ν's are independent of one another. Closer examination shows that this is justified. More detailed derivations of the $N/\log N$ results are available in Weisbuch and Fogelman-Soulié [1985] and McEliece et al. [1987].

In summary, the capacity $p_{\max}$ is proportional to N (but never higher than $0.138N$) if we are willing to accept a small percentage of errors in each pattern, but is proportional to $N/\log N$ if we insist that most or all patterns be recalled perfectly.

Realistic patterns will *not* in general be random, though some precoding can make them more so. The Hopfield model is usually studied with random patterns for mathematical convenience, though the effect of correlated patterns has also been examined (see Section 3.2). At the other extreme, if the different patterns are strictly *orthogonal*, i.e.,

$$\sum_j \xi_j^\mu \xi_j^\nu = 0 \qquad \text{for all } \mu \neq \nu \tag{2.22}$$

then there is no crosstalk at all; $C_i^\nu = 0$ for all i and ν.

In this orthogonal case the memory capacity $p_{\max}$ is apparently N patterns, because at most N mutually orthogonal bit strings of length N can be constructed. But the *useful* capacity is somewhat smaller. Trying to embed N orthogonal patterns with the Hebb rule actually makes *all* states stable; the system stays wherever it starts, and is useless as a memory. This occurs because the orthogonality conditions (2.22) lead necessarily to[5]

$$w_{ij} = \begin{cases} 1 & \text{if } i = j\,; \\ 0 & \text{otherwise.} \end{cases} \tag{2.23}$$

so each unit is connected only to itself. To define a useful measure of capacity for such a case it is clearly necessary to insist on a finite basin of attraction around each desired pattern. This leads to a useful capacity slightly less than N.

[5]Consider the matrix X with components $X_{\mu i} = \xi_i^\mu$. Equation (2.22) implies $XX^T = N1$, where 1 is

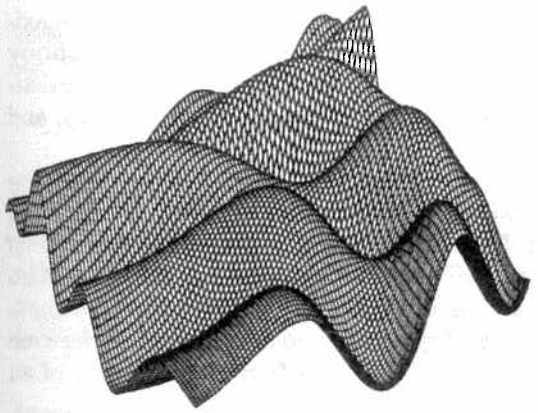

FIGURE 2.6 It is often useful (but sometimes dangerous) to think of the energy as something like this landscape. The z-axis is the energy and the 2^N corners of the hypercube (the possible states of the system) are formally represented by the x–y plane.

The Energy Function

One of the most important contributions of the Hopfield [1982] paper was to introduce the idea of an *energy function* into neural network theory. For the networks we are considering, the energy function H is

$$H = -\frac{1}{2} \sum_{ij} w_{ij} S_i S_j \,. \tag{2.24}$$

The double sum is over all i and all j. The $i = j$ terms are of no consequence because $S_i^2 = 1$; they just contribute a constant to H, and in any case we could choose $w_{ii} = 0$. The energy function is a function of the configuration $\{S_i\}$ of the system, where $\{S_i\}$ means the set of all the S_i's. We can thus imagine an **energy landscape** "above" the configuration space of Fig. 2.2. Typically this surface is quite hilly. Figure 2.6 illustrates the idea.

The central property of an energy function is that *it always decreases (or remains constant) as the system evolves according to its dynamical rule.* We will show this in a moment for (2.24). Thus the attractors (memorized patterns) in Fig. 2.2 are at local minima of the energy surface. The dynamics can be thought of as similar to the motion of a particle on the energy surface under the influence of gravity (pulling it down) and friction (so that it does not overshoot). From any starting point the particle (representing the whole state $\{S_i\}$ of the system) slides downhill until it comes to rest at one of these local minima—at one of the attractors. The basins of attraction correspond to the valleys or catchment areas around each minimum. Starting the system in a particular valley leads to the lowest point of that valley.

The term **energy function** comes from a physical analogy to magnetic systems that we will discuss in the next section. But the concept is of much wider applicability; in many fields there is a state function that always decreases during

In some fields the convention is reversed; the function increases or must be maximized. The most general name, from the theory of dynamical systems, is **Lyapunov function** [Cohen and Grossberg, 1983]. Other terms are **Hamiltonian** in statistical mechanics, **cost function** or **objective function** in optimization theory, and **fitness function** in evolutionary biology.

For neural networks in general an energy function exists if the connection strengths are *symmetric*, i.e., $w_{ij} = w_{ji}$. In real networks of neurons this is an unreasonable assumption, but it is useful to study the symmetric case because of the extra insight that the existence of an energy function affords us. The Hebb prescription (2.9) which we are now studying automatically yields symmetric w_{ij}'s. Gérard Toulouse has called Hopfield's use of symmetric connections a "clever step backwards from biological realism." The cleverness arises from the existence of an energy function.

For symmetric connections we can write (2.24) in the alternative form

$$H = C - \sum_{(ij)} w_{ij} S_i S_j \tag{2.25}$$

where (ij) means all the distinct pairs ij, counting for example 12 as the same pair as 21. We exclude the ii terms from (ij); they give the constant C.

It now is easy to show that the dynamical rule (2.4) can only decrease the energy. Let S_i' be the new value of S_i given by (2.4) for some particular unit i:

$$S_i' = \mathrm{sgn}\Big(\sum_j w_{ij} S_j\Big) . \tag{2.26}$$

Obviously if $S_i' = S_i$ the energy is unchanged. In the other case $S_i' = -S_i$ so, picking out the terms that involve S_i,

$$\begin{aligned} H' - H &= -\sum_{j\neq i} w_{ij} S_i' S_j + \sum_{j\neq i} w_{ij} S_i S_j \\ &= 2S_i \sum_{j\neq i} w_{ij} S_j \\ &= 2S_i \sum_j w_{ij} S_j - 2w_{ii} . \end{aligned} \tag{2.27}$$

Now the first term is negative from (2.26), and the second term is negative because the Hebb rule (2.9) gives $w_{ii} = p/N$ for all i. Thus the energy decreases every time an S_i changes, as claimed.

The **self-coupling terms** w_{ii} may actually be omitted altogether, both from the Hebb rule (where we can simply define $w_{ii} = 0$) and from the energy function. It is straightforward to check that they make no appreciable difference to the stability

Sompolinsky, 1987]. We can see why simply by separating the self-coupling term out of the dynamical rule (2.4):

$$S_i := \mathrm{sgn}\Big(w_{ii} S_i + \sum_{j\neq i} w_{ij} S_j\Big) . \tag{2.28}$$

If w_{ii} were larger than $\sum_{j\neq i} w_{ij} S_j$ in some state, then $S_i = +1$ and $S_i = -1$ could *both* be stable.[6] This can produce additional stable **spurious states** in the neighborhood of a desired attractor, reducing the size of the basin of attraction. If $w_{ii} = 0$, then this problem does not arise; for a given configuration of the other spins S_i will always pick one of its states over the other.

Starting from an Energy Function

The idea of the energy function as something to be minimized in the stable states gives us an alternate way to derive the Hebb prescription (2.9). Let us start again with the single-pattern case. We want the energy to be minimized when the overlap between the network configuration and the stored pattern ξ_i is largest. So we choose

$$H = -\frac{1}{2N}\Big(\sum_i S_i \xi_i\Big)^2 \tag{2.29}$$

where the factor $1/2N$ is the product of inspired hindsight. For the many-pattern case, we can try to make each of the ξ_i^μ into local minima of H just by summing (2.29) over all the patterns:

$$H = -\frac{1}{2N}\sum_{\mu=1}^{p}\Big(\sum_i S_i \xi_i^\mu\Big)^2 . \tag{2.30}$$

Multiplying this out gives

$$H = -\frac{1}{2N}\sum_{\mu=1}^{p}\Big(\sum_i S_i \xi_i^\mu\Big)\Big(\sum_j S_j \xi_j^\mu\Big) = -\frac{1}{2}\sum_{ij}\Big(\frac{1}{N}\sum_{\mu=1}^{p}\xi_i^\mu \xi_j^\mu\Big) S_i S_j \tag{2.31}$$

which is exactly the same as our original energy function (2.24) if w_{ij} is given by the Hebb rule (2.9).

This approach to finding appropriate w_{ij}'s is generally useful. If we can write down an energy function whose minimum satisfies a problem of interest, then we can multiply it out and identify the appropriate connection strength w_{ij} from the coefficient of $S_i S_j$. We will encounter several applications in Chapter 4. Of course we may find other terms, not of the $S_i S_j$ form. Constants are no problem, and terms linear in a single S_i can be represented by thresholds or by a connection to a clamped S_0 unit. But terms like $S_i S_j S_k$ take us outside the present framework of pairwise connections.

Spurious States

We have shown that the Hebb prescription (2.9) gives us (for small enough p) a dynamical system that has attractors—local minima of the energy function—at the desired points ξ_i^μ. These are sometimes called the **retrieval states**. But we have *not* shown that these are the only attractors. And indeed there are others.

First of all, the reversed states $-\xi_i^\mu$ are minima and have the same energy as the original patterns. The dynamics and the energy function both have a perfect symmetry, $S_i \leftrightarrow -S_i$ for all i. This is not too troublesome for the retrieved patterns; we could agree to reverse all the remaining bits when a particular "sign bit" is -1 for example.

Second, there are stable **mixture states** ξ_i^{mix}, which are not equal to any single pattern, but instead correspond to linear combinations of an odd number of patterns [Amit et al., 1985a]. The simplest of these are symmetric combinations of three stored patterns:

$$\xi_i^{\text{mix}} = \text{sgn}(\pm \xi_i^{\mu_1} \pm \xi_i^{\mu_2} \pm \xi_i^{\mu_3}) . \tag{2.32}$$

All eight sign combinations are possible, but we consider for definiteness the case where all the signs are chosen as +'s. The other cases are similar. Observe that on average ξ_i^{mix} has the same sign as $\xi_i^{\mu_1}$ three times out of four; only if $\xi_i^{\mu_2}$ and $\xi_i^{\mu_3}$ both have the opposite sign can the overall sign be reversed. So ξ_i^{mix} is Hamming distance $N/4$ from $\xi_i^{\mu_1}$, and of course from $\xi_i^{\mu_2}$ and $\xi_i^{\mu_3}$ too; the mixture states lie at points equidistant from their components. This also implies that $\sum_i \xi_i^{\mu_1}\xi_i^{\text{mix}} = N/2$ on average. Now to check the stability of (2.32), still with all + signs, we can repeat the calculation of (2.11) and (2.12), but this time pick out the three special μ's:

$$h_i^{\text{mix}} = \frac{1}{N}\sum_{j\mu} \xi_i^\mu \xi_j^\mu \xi_j^{\text{mix}} = \frac{1}{2}\xi_i^{\mu_1} + \frac{1}{2}\xi_i^{\mu_2} + \frac{1}{2}\xi_i^{\mu_3} + \text{cross-terms} . \tag{2.33}$$

Thus the stability condition (2.10) is indeed satisfied for the mixture state (2.32). Similarly 5, 7, ... patterns may be combined. The system does not choose an *even* number of patterns because they can add up to zero on some sites, whereas the units have to be ± 1.

Third, for large p there are local minima that are not correlated with any finite number of the original patterns ξ_i^μ [Amit et al., 1985b]. These are sometimes called **spin glass states** because of a close correspondence to spin glass models in statistical mechanics. We will meet them again in Section 2.5.

So the memory does not work perfectly; there are all these additional minima in addition to the ones we want. The second and third classes are generally called **spurious minima**. Of course we only fall into one of them if we start close to it, and they tend to have rather small basins of attraction compared to the retrieval states. There are also various tricks that we will consider later, including finite

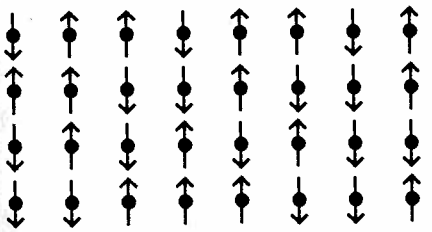

FIGURE 2.7 A very simplified picture of a magnetic material described by an Ising model.

2.3 Statistical Mechanics of Magnetic Systems

There is a close analogy between Hopfield networks and some simple models of **magnetic materials** in statistical physics. The analogy becomes particularly useful when we generalize the networks to use **stochastic units**, which brings the idea of **temperature** into network theory. We will explore this development in the next section, after providing here the necessary background in the statistical mechanics of magnetic systems. The Appendix goes further into statistical mechanics.

A simple description of a magnetic material consists of a set of atomic magnets arranged on a regular lattice that represents the crystal structure of the material (metals are crystals in this sense). We will use the term **spins** for these atomic magnets. The name comes from the quantum mechanical origin of the magnetic moments. The spins can each point in various directions, the number of possibilities depending on the type of atom considered. Particularly simple is the case of "spin 1/2" atoms, in which only two distinct directions are possible. This is represented in an **Ising model** by a variable S_i for each lattice site i, with allowed values ± 1. The spin is oriented "up" if $S_i = +1$ and "down" if $S_i = -1$. Figure 2.7 illustrates a possible configuration with spins shown by arrows pointing up or down.

The analogy of the Ising model spins to the activation of units in a neural network is obvious, and indeed we have used the same symbol S_i for both. An active unit (firing cell) in the network corresponds to "spin up" in the magnet and an inactive one to "spin down". Ising models are in fact used widely [e.g., Ma, 1985; Huang, 1987], not only for spin 1/2 magnetic materials, but also for many physical systems which can be described by binary (i.e., two-valued) variables. In many cases the description is very idealized. One might for instance simplify a continuous variable to an Ising one (as we are doing with our McCulloch-Pitts assumption), or one might describe a gas by specifying an Ising variable (for *filled* or *empty*) in each of a fine grid of cells covering the system. There is a great deal of accumulated knowledge about Ising models, some of which can be applied to neural networks.

An Ising model is not fully specified until the interactions and dynamics of the spins are given. In a magnetic material each of the spins is influenced by the **magnetic field** h at its location. This magnetic field consists of any **external field** h^{ext} applied by the experimenter, plus an **internal field** produced by the other spins. The contribution of each atom to the internal field at a given location