CHAPTER 12

Resource Requirements of Standard and Programmable Nets

J. L. McCLELLAND

In several places in this book we have examined the capabilities of various models of parallel distributed processing. We have considered models that are guaranteed to do one thing or another—to learn, say, up to some criterion of optimality or to settle into global states with probabilities proportional to the goodness of the states. In later chapters, we describe various models of psychological or neurophysiological processes and consider how well they account for the data. The models, then, are held up against various criteria of computational, psychological, and sometimes physiological adequacy.

In this chapter I raise another question about PDP models. I consider the resources they require, in terms of units and connections, to carry out a particular amount of work. This issue is touched on in various other places in the book, particularly Chapter 3. There we showed that a distributed model can often perform even an arbitrary mapping with less hardware than a local model would require to do the same task.

In this chapter I continue this line of thinking and extend it in various ways, drawing on the work of several other researchers, particularly Willshaw (1971, 1981). The analysis is far from exhaustive, but it focuses on several fairly central questions about the resource requirements of PDP networks. In the first part of the chapter, I consider the resource requirements of a simple pattern associator. I review the analysis offered by Willshaw (1981) and extend it in one or two small ways, and I consider how it might be possible to overcome some limitations that arise in networks consisting of units with limited connectivity. In the second part of the chapter, I consider the resource requirements of a distributed version of the dynamically programmable networks described in Chapter 16.

THE STANDARD PATTERN ASSOCIATOR

In this section, we will consider pattern associator models similar to the models studied by J. A. Anderson (e.g., Anderson, 1983) and Kohonen (1977, 1984), and to the past-tense learning model described in Chapter 18. A small pattern associator is illustrated in Figure 1. A pattern associator consists of two sets of units, called *input* and *output* units, and a connection from each input unit to each output unit. The associator takes as input a pattern of activation on its input units and produces in response a pattern on the output units based on the connections between the input and output units.

Different pattern associators make slightly different assumptions about the processing characteristics of the units. We will follow Willshaw's (1981) analysis of a particular, simple case; he used binary units and binary connections between units. Thus, units could take on activation values of 0 or 1. Similarly, the connections between the units could take on only binary values of 0 and 1.

In Willshaw nets, processing is an extremely simple matter. A pattern of activation is imposed on the input units, turning each one either on or off. Each active input unit then sends a quantum of activation to each of the output units it has a switched-on connection to. Output units go on if the number of quanta they receive exceeds a threshold; otherwise they stay off.

The learning rule Willshaw studied is equally simple. Training amounts to presenting each input pattern paired with the corresponding output pattern, and turning on the connection from each active input unit to each active output unit. This is, of course, a simple variant of Hebbian learning. Given this learning rule, it follows that when the input pattern of a known association is presented to the network, each of the activated input units will send one quantum of activation to all of the correct output units. This means that the number of quanta of activation each correct output unit will receive will be equal to the number of active input units.

In examining the learning capacity of this network, Willshaw made several further assumptions. First, he assumed that all of the associations (or pairs of patterns) to be learned have the same number of active input units and the same number of active output units. Second,

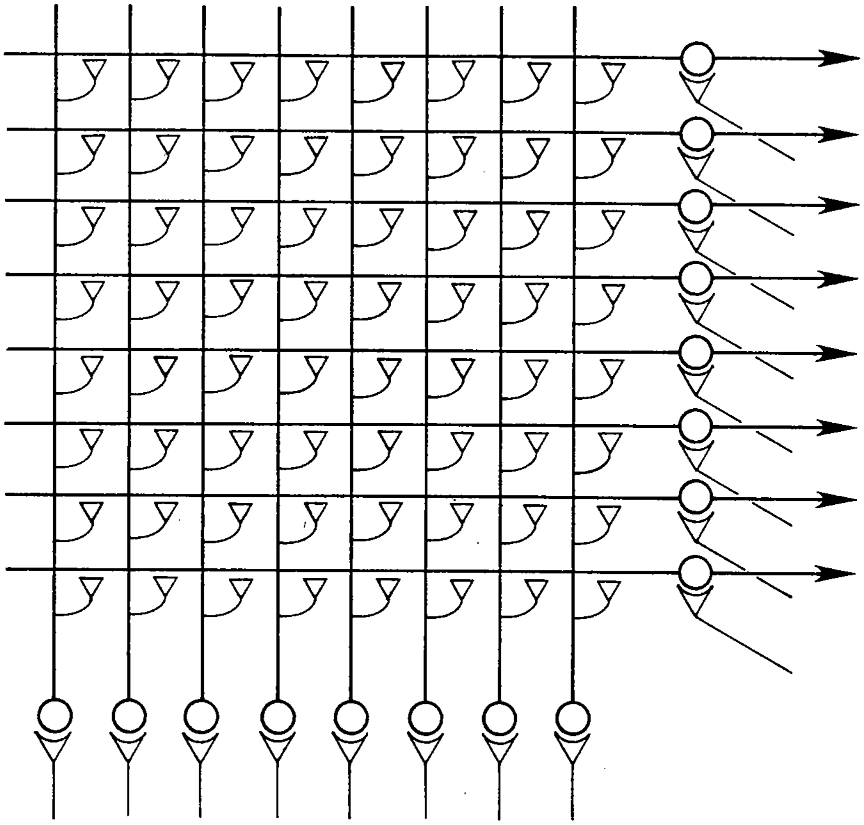

FIGURE 1. A pattern associator consisting of a set of input units (across the bottom) and output units (along the right side), with a connection from each input unit to each output unit.

he assumed that the threshold of each output unit is set equal to the number of active input units. Given this assumption, only those output units with switched-on connections from *all* of the active input units will reach threshold.

Now we can begin to examine the capacity of these networks. In particular, we can ask questions like the following. How many input units (n_i) and output units (n_o) would be needed to allow retrieval of the correct mate of each of r different input patterns?

The answer to such a question depends on the criterion of correct retrieval used. For present purposes, we can adopt the following criterion: All of the correct output units should be turned on, and, on the average, no more than one output unit should be turned on spuriously.

Since the assumptions of the model guarantee that all the correct output units will be turned on when the correct input is shown, the analysis focuses on the number of units needed to store r patterns without exceeding the acceptable number of spurious activations.

The answer to our question also depends on the number of units active in the input and output patterns in each pattern pair and on the similarity relations among the patterns. A very useful case that Willshaw considered is the case in which each of the r associations involves a random selection of m_i input units and m_o output units. From the assumption of randomness, it is easy to compute the probability that any given junction will be turned on after learning all r associations. From this it is easy to compute the average number of spurious activations. We will now go through these computations.

First we consider the probability p_{on} that any given junction will end up being turned on, for a particular choice of the parameters n_i, n_o, m_i, m_o, and r. Imagine that the r patterns are stored, one after the other, in the $n_i n_o$ connections between the n_i input units and the n_o output units. As each pattern is stored, it turns on $m_i m_o$ of the $n_i n_o$ connections, so each junction in the network is turned on with probability $m_i m_o / n_i n_o$. The probability that a junction is not turned on by a single association is just 1 minus this quantity. Since each of the r associations is a new random sample of m_i of the n_i input units and m_o of the n_o output units, the probability that a junction has not been turned on—or 1 minus the probability that it has been turned on—after r patterns have been stored is

$$1 - p_{on} = \left(1 - \frac{m_i m_o}{n_i n_o}\right)^r .$$

Rearranging to solve for p_{on} we obtain

$$p_{on} = 1 - \left(1 - \frac{m_i m_o}{n_i n_o}\right)^r .$$

Now that we know p_{on}, it is easy to calculate the number of spurious activations of output units. First, any output unit that should not be activated will be turned on if and only if all of its junctions from the m_i active input units happen to be on. Given the assumption of randomness, this will occur with probability $p_{on}^{m_i}$, since each junction is on with probability p_{on}. Since there are $n_o - m_o$ output units that are candidates for spurious activation, the average number of spuriously activated units is

$$(n_o - m_o) p_{on}^{m_i}.$$

We want to keep this number less than 1. Adopting a slightly more stringent criterion to simplify the calculations, we can set

$$1 \geqslant n_o p_{on}{}^{m_i} ,$$

or

$$\left(\frac{1}{n_o}\right)^{\frac{1}{m_i}} \geqslant p_{on}$$

$$\geqslant 1 - \left(1 - \frac{m_i m_o}{n_i n_o}\right)^r .$$

Rearranging, we get

$$1-\left(\frac{1}{n_o}\right)^{\frac{1}{m_i}} \leqslant \left(1 - \frac{m_i m_o}{n_i n_o}\right)^r .$$

For small positive x, $\log(1-x) = -x$. If we restrict ourselves to cases where $m_i m_o / n_i n_o < .1$—that is, reasonably sparse patterns in the sense that $m < n/\sqrt{10}$—the approximation will hold for the right-hand side of the equation, so that taking logs we get

$$\log\left[1-\left(\frac{1}{n}\right)^{\frac{1}{m_i}}\right] \leqslant - r \frac{m_i m_o}{n_i n_o}.$$

We can solve this for r, the number of patterns, to obtain

$$r \leqslant -\frac{n_i n_o}{m_i m_o}\log\left[1-\left(\frac{1}{n_o}\right)^{\frac{1}{m_i}}\right] . \tag{1}$$

Now, $-\log\left[1-\left(\frac{1}{n_o}\right)^{\frac{1}{m_i}}\right]$ ranges upward from .69 for very sparse patterns where $m_i = \log_2 n_o$. Using .69 as a lower bound, we are safe if we say:

$$r \leqslant .69\frac{n_i n_o}{m_i m_o}$$

or

$$n_i n_o \geqslant 1.45 r m_i m_o .$$

This result tells us that the number of storage elements (that is, connections, $n_i n_o$) that we need is proportional to the number of associations we wish to store times the number of connections $(m_i m_o)$ activated in storing each association. This seems about right, intuitively. In fact, this is an upper bound rather greater than the true number of storage elements required for less sparse patterns, as can be seen by plugging values of m_i greater than $\log_2 n_o$ into Equation 1.

It is interesting to compare Willshaw nets to various kinds of local representation. One very simple local representation would associate a single, active input unit with one or more active output units. Obviously, such a network would have a capacity of only n_i patterns. We can use the connections of a Willshaw net more effectively with a distributed input if the input and output patterns are reasonably sparse. For instance, in a square net with the same number n of input and output units and the same number m of active elements in each, if $n = 1000$ and $m = 10$, we find that we can store about 7,000 associations instead of the 1,000 we could store using local representation over the input units.

Another scheme to compare to the Willshaw scheme would be one that encodes each pattern to be learned with a single hidden unit between the input and output layers. Obviously a net that behaved perfectly in performing r associations between m_i active input elements and m_o active output units could be handcrafted using r hidden units, each having m_i input connections and m_o output connections. Such a network can be economical once it is wired up exactly right: It only needs $r(m_i+m_o)$ connections. However, there are two points to note. First, it is not obvious how to provide enough hardware in advance to handle an arbitrary r patterns of m_i active input units and m_o active output units. The number of such patterns possible is approximately $(n_i^{m_i}/m_i!)(n_o^{m_o}/m_o!)$, and if we had to provide a unit in advance for each of these our hardware cost would get out of hand very fast. Second, the economy of the scheme is not due to the use of local representation, but to the use of hidden units. In many cases even more economical representation can be achieved with coarse-coded hidden units (see Chapter 3 and Kanerva, 1984).

Randomly Connected Nets

Returning to the standard Willshaw net, there are several minor difficulties with Willshaw's scheme. First, it assumes that each input unit sends one and only one connection to each output unit. In a neural network, we might assume each input unit sends out a randomly distributed array of connections to the set of output units without any guarantee that each output unit actually receives a connection. Second, the analysis depends on a rather strict and sharp threshold for output unit activation. In a random net rather than a fully connected net, we could not actually guarantee that a given output unit would in fact receive m_i inputs; and in realistic nets, we would expect there to be some inherent variability in the activations of the units. Thus, we would not be able to guarantee that all correct units would exceed the sharp threshold, nor that all incorrect units would fall below it.

However, it turns out that we can reformulate the problem just slightly and get a handle on networks that have these properties. Assume that we have a square network of n input and n output units and that we wish to store associations between m active input units and m active output units. Suppose each input unit has f output connections which fall where they may among the n output units so that the output units have an average of f inputs each. Note again that the connections are randomly distributed without restriction so that there is no guarantee that input unit i projects to output unit j.

To study the performance of this net, imagine storing some number r of patterns using the Willshaw learning scheme. During testing, we will examine the number of active inputs each output unit that should be turned on will receive and the number of active inputs each unit that should not be turned on will receive, and we will then calculate the signal-detection measure of sensitivity d' (Green & Swets, 1966) as an index of the ability of inputs reaching each output unit to distinguish between units that should be on and units that should not be on. Since d' is independent of the threshold, this measure allows us to bypass the question of the threshold itself.

Let us first consider what happens in our random network as we train it with pairs of patterns using Willshaw's scheme. Pick an arbitrary connection in our net between an arbitrary input unit and an arbitrary output unit. Now, consider learning an arbitrary pattern. The probability that a particular input unit will be on is m/n. Similarly, the probability that a particular output unit will be on is m/n. The probability that the units joined by the particular connection we are considering will be one of the ones turned on in learning a particular pattern, then, is m^2/n^2 just as before. The rest of the earlier analysis still applies, and we get

$$p_{on} = 1 - \left(1 - \frac{m^2}{n^2} \right)^r .$$

This is exactly the same value that we had before in the original Willshaw model, and it is independent of f, the number of connections each unit makes. This factor will become important soon, but it does not affect the probability that a particular connection will be on after learning r patterns.

Now consider what happens during the testing of a particular learned association. We activate the correct m input units and examine the mean number of quanta of activation that each output unit that should be on will receive. The m active input units each have f outputs, so there are mf total "active" connections. A particular one of these connections reaches a particular output unit with probability $1/n$, since each connection is assumed to fall at random among the n output units. Thus, the average number of active connections each output unit receives will simply be mf/n. For output units that should be on, each of these connections will have been turned on during learning, so mf/n is the average number of quanta that unit will receive. Assuming that n is reasonably large, the distribution of this quantity is approximately Poisson, so its variance is also given by mf/n.

Units that should not be on also receive an arbitrary connection from an active input unit with probability $1/n$, but each such connection is only on with probability p_{on}. Thus, the average number of quanta such units receive is $(mf/n)p_{on}$. This quantity is also approximately Poisson, so its variance is also equal to its mean.

Our measure of sensitivity, d', is the difference between these means divided by the square root of the average of the variances. That is,

$$d' = \frac{mf/n\,(1-p_{on})}{\sqrt{(mf/n)(1+p_{on})/2}}.$$

Simplifying, this becomes

$$d' = \sqrt{mf/n}\,\frac{1-p_{on}}{\sqrt{(1+p_{on})/2}}. \qquad (2)$$

We can get bounds on the true value of d' by noting that the denominator above cannot be greater than 1 or less than $\sqrt{½}$. The largest value of the denominator sets a lower bound on d', so we find that

$$d' \geqslant \sqrt{mf/n}\,(1-p_{on}).$$

Substituting for $1-p_{on}$, we obtain

$$d' \geqslant \sqrt{mf/n}\left[1-\frac{m^2}{n^2}\right]^r. \qquad (3)$$

Taking logs of both sides, invoking the $\log(1-x) = -x$ approximation, and solving for r, we obtain

$$r \leqslant .5\frac{n^2}{m^2}\Big[\log(mf/n) - 2\log(d')\Big].$$

One of the first things to note from this expression is its similarity to the expression we had for the case of Willshaw's fully connected net. In particular, if we let $f = n$, so that each unit sends an average of one connection to reach another unit, we get

$$r \leqslant .5\frac{n^2}{m^2}\Big[\log(m) - 2\log(d')\Big]. \tag{4}$$

The expression in brackets on the right expresses the fact that the capacity of the net goes down as the sensitivity we want to achieve goes up and captures the fact that there is a slight benefit as we increase m, independent of its effect on the ratio of total connections to connections activated per association. This is due to the fact that the distributions of activations of correct and spurious units pull apart as m gets larger. These are relatively small factors for moderate values of m and d'. More important, as before, is the ratio of the number of connections (n^2) relative to the average number of connections each pattern takes up (m^2).

Effects of Limited Fan-Out

A new result emerges when we consider other possible values for f: Equation 3 indicates that d' is directly proportional to the square root of f. Thus, we can achieve any degree of fidelity we require by increasing f though returns diminish as f gets bigger and bigger. Alternatively, the performance of our network will degrade gracefully as fan-out is reduced.

We can also see that increases in n are no longer uniformly beneficial. The term $\log(mf/n)$ decreases a n increases; we can no longer increase the capacity indefinitely simply by increasing n.

Figure 2 indicates a discovery of Mitchison (personal communication, 1984) concerning the capacity r of a network as a function of n for several values of m and f, with $d' = 5$. Capacity depends roughly on n^2/m^2 and is relatively insensitive to f as long as $\sqrt{mf/n} >> d'$. However, as n increases we reach a point where $\sqrt{mf/n}$ approaches d';

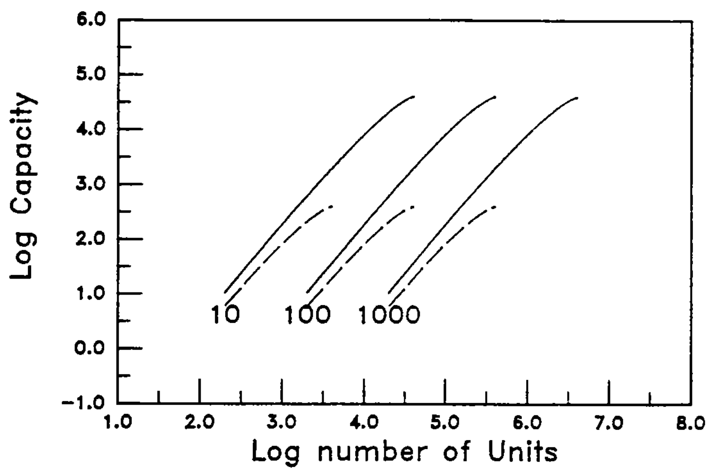

FIGURE 2. Effects of limited fan-out in a randomly connected associative net as a function of the log of n, the number of input and output units; for different values of m, the number of active units in each input and output pattern indicated below each pair of curves. The upper member of each pair of curves is for fan-out of 10,000; the lower member is for fan-out of 1,000. In all cases, the y-axis reflects the log of the maximum number of patterns that can be stored while maintaining a d' of 5. Calculations are based on Equation 2 (Equation 3 gives misleading results for $\sqrt{mf/n}$ near d').

here capacity levels off and further increases in n result in no further increase in sensitivity. The maximal capacity achievable by increasing n is invariant, regardless of the value of m, and depends only on d' and f. Thus, if we were to pick a fixed value of d', we would find that the maximum number of patterns we could store would be strictly limited by f.

Biological limits on storage capacity of neural nets. With these analyses in mind, we can now consider what limits biological hardware might place on the storage capacity of a neural net. Of course, we must be clear on the fact that we are considering a very restricted class of distributed models and there is no guarantee that our results will generalize. Nevertheless, it is reasonably interesting to consider what it would take to store a large body of information, say, a million different pairs of patterns, with each pattern consisting of a 1,000 active input units and 1,000 active output units.

To be on the safe side, let's adopt a d' of 5. With this value, if the units have an unbiased threshold, the network will miss less than 1% of the units that should be on and false alarm to less than 1% of the units that should be off.[1]

How big would a net have to be to meet these specifications? Assuming a fully connected net, and consulting Equation 4, we find that we need to set n equal to a value near about 10^6 to get r large enough.

This number of units is not a serious problem since estimates of the number of units in the brain generally range upward from 10^{10} (see Chapter 20). However, individual units are not generally assumed to have enough connections for this scheme to work as stated. If there are 1,000 to 10,000 connections per unit, as suggested in Chapter 20, we are off by two to three orders of magnitude in the number of connections per unit.

Given this limitation on fan-out, we had better consult Figure 2. The figure indicates that the maximum capacity of a net with a fan-out of 1,000 and a d' of 5 is only about 150 patterns. With $f = 10{,}000$ we get up to a capacity of about 15,000 patterns, but we are still well short of the mark. It seems, then, that the fan-out of neurons drastically limits the capacity of a distributed network.

A simple method for overcoming the fan-out limitation. But all is not completely lost. It turns out that it is a relatively simple matter to overcome the fan-out limitation. The trick is simply to use multiple layers of units. Let each input unit activate a set of what we might call *dispersion* units, and let each output unit receive input from a set of *collection* units. Let the f outgoing connections of each of the dispersion units be randomly distributed among the "dendrites" of the collection units. A miniature version of this scheme is illustrated in Figure 3. Note that it is assumed that each dispersion unit is driven by a *single* input unit, and each collection unit projects to a *single* output unit. Collection units are assumed to be perfectly linear so that the net input to each output unit is just the sum of the net inputs to the collector units that project to it. Assuming each input unit and each dispersion unit has a fan-out of f, the effective fan-out of the input and dispersion layers together becomes f^2. Similarly, the set of collection units feeding into each output unit collect an average of f^2 connections. To construct an associator of 1 million input units by 1 million output units assuming each unit has a fanout of 1,000, we will need 1 billion dispersion units and 1 billion collection units. The number of connections between the dispersion units and the collection units would be on the order of 10^{12}, or 1 trillion connections.

[1] It should be pointed out that any intrinsic noise in the units would reduce the actual observed sensitivity.

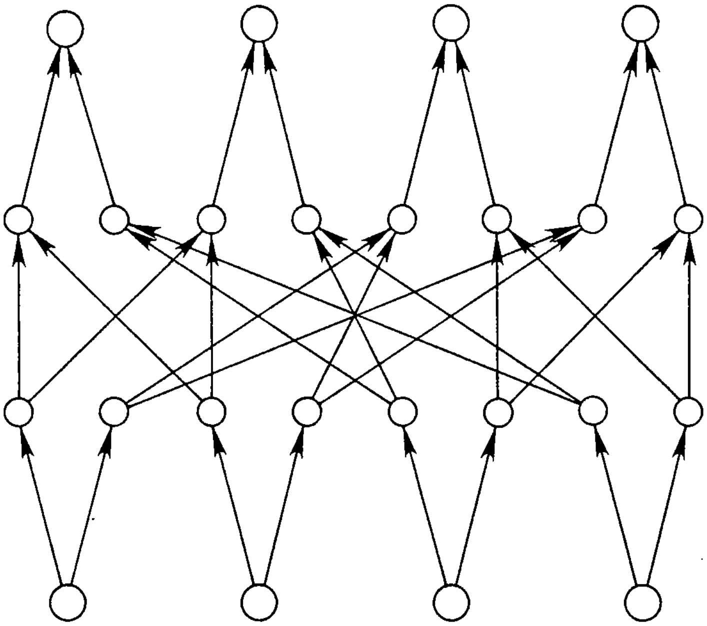

FIGURE 3. A diagram of a multilayer network consisting of an input layer, a dispersion layer, a collection layer, and an output layer. The network serves to square the effective fan-out of each input unit, relative to the simple two-layer case.

The network would require about 20% of human cortex, based on the estimate that there are 10^5 neurons under each square millimeter of the brain and that there are about 10^5 square millimeters of cortical surface. This might be a little tight, but if the fan-out were 10,000, the network would fit handily. In that case, it would only require about 2 percent of the 10^{10} units.

There are, of course, a lot of reasons to doubt that these figures represent anything more than a first-order estimate of the capacity of real associative networks. There are several oversimplifications, including for example the assumption that the dispersion units are each driven by a single connection. We must also note that we have assumed a two-layer net along with an extremely simple learning rule. The intermediate layers postulated here merely serve to provide a way of overcoming the fan-out limits of individual units. However, as has been pointed out in Chapter 7, a multilayer net can often learn to construct its own coding schemes that are much more efficient than the random coding schemes used here. Even simple two-layer nets can profit if there are some regularities in the network and if they use a sensible learning rule, as shown in Chapter 18. Thus random nets like the ones that have been analyzed in this section probably represent a lower limit on efficiency that we can use as a benchmark against which to measure "smarter" PDP mechanisms.

Effects of Degradation and the Benefits of Redundancy

One virtue of distributed models is their ability to handle degradation, either of the input pattern or of the network itself. The d' analysis allows us to tell a very simple story about the effects of degradation. In this section I will just consider the effects of degradation by removal, either of a random fraction of the pattern or of a random fraction of the connections in the network; effects of added noise will be considered later on. In the case of removal, we can think of it either in terms of presenting an incomplete pattern or actually destroying some of the input units so that parts of the pattern are simply no longer represented. Consider the case of a network that has already been trained with some number of patterns so that p_{on} can be treated as a constant. Then we can write the equation relating d' to m, f, and n as

$$d' \geqslant k\sqrt{mf/n}.$$

Now, suppose that during testing we turn on only some proportion p_t of the m units representing a pattern. The m in the above equation becomes mp_t, so we see that the sensitivity of the network as indexed by d' falls off as the *square root* of the fraction of the probe that is presented. Similarly, suppose some of the connections leading out of each unit are destroyed, leaving a random intact proportion p_i of the mf active connections. Again, the sensitivity of the network will be proportional to the square root of the number of remaining connections. Thus, performance degrades gracefully under both kinds of damage.

Another frequently noted virtue of distributed memories is the redundancy they tend naturally to provide. The ability of simple distributed memories to cope with degraded input patterns is really just a matter of their redundancy, as Willshaw (1981) pointed out. For, if a network is fully loaded, in the sense that it can hold no more associations and still meet some predetermined standard of accuracy with complete patterns, it will not be able to meet that same criterion with degradation. The only way to guard against this problem is to load the network lightly enough so that the criterion can still be met after subjecting the network or the inputs to the specified degree of degradation.

PROGRAMMABLE PATTERN ASSOCIATORS

In this section, I extend the sort of analysis we have performed on simple associator models to the resource requirements of connection information distribution (CID) networks of the type discussed in Chapter 16.

The mechanism shown in Figure 4 is a distributed CID mechanism. The purpose of this network is to allow connection information stored in a central associative network to be used to set connections in several *local* or *programmable* networks in the course of processing so that more

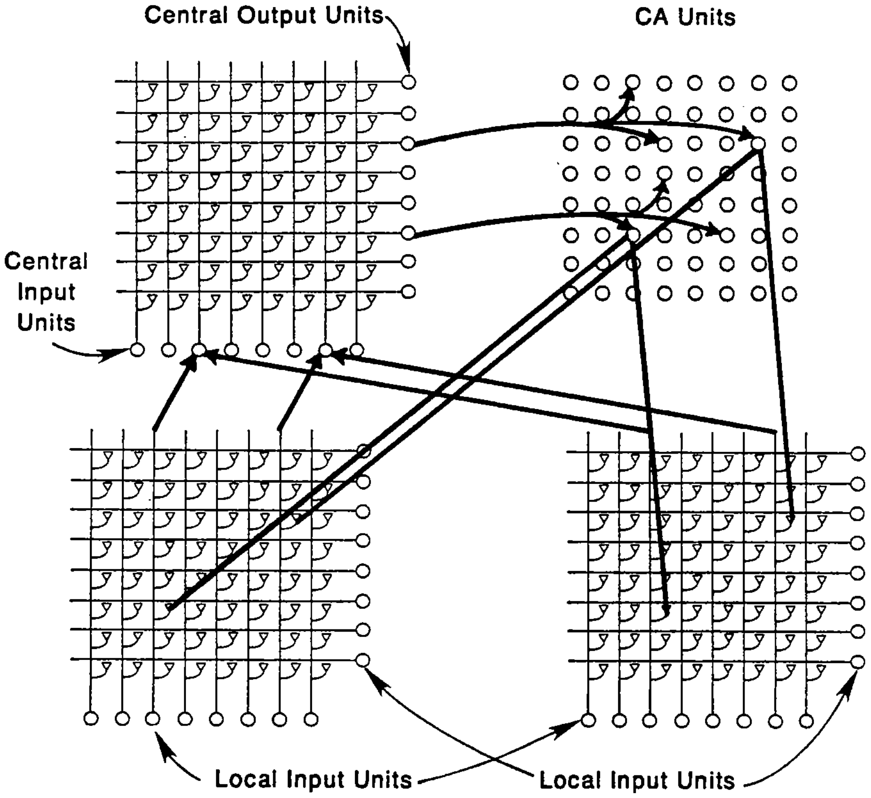

FIGURE 4. A connection information distribution (CID) network consisting of two local, programmable networks; a central, standard network; and a set of connection activation (CA) units. Each local input unit projects to the corresponding central input unit, and each CA unit projects to the corresponding connection in both local networks. Central output units turn on CA units relevant to processing the patterns they program the local modules to process. The connections drawn in are a few examples of each type.

than one input pattern can be processed at one time. The mechanism works as follows: One or more patterns to be processed are presented as inputs, with each pattern going to the input units in a different programmable network. The input pattern to each local net is also transmitted to the input units of the central associative network. When more than one pattern is presented at a time, the input to the central network is just the pattern that results from superimposing all of the input patterns. This pattern, via the connections in the central associative network, causes a pattern of activation over the central output units. The central output pattern, of course, is a composite representation of all of the input patterns. It is not itself the desired output of the system, but is the pattern that serves as the basis for programming (or turning on connections) in the local, programmable networks. The local networks are programmed via a set of units called the connection activation (CA) units. The CA units act essentially as switches that turn on connections in the programmable networks. In the version of the model we will start with, each CA unit projects to the one specific connection it corresponds to in each programmable network, so there are as many CA units as there are connections in a single programmable net. In the figure, the CA units are laid out so that the location of each one corresponds to the location of the connection it commands in each of the programmable networks.

To program the local networks, then, central output units activate the CA units corresponding to the connections needed to process the patterns represented on the central output units. The CA units turn on the corresponding connections. This does not mean that the CA units actually cause activation to pass to the local output units. Rather, they simply enable connections in the programmable nets. Each active local input unit sends a quantum of activation to a given local output unit if the connection between them is turned on.

The question we will be concerned with first is the number of CA units required to make the mechanism work properly. In a later section, we will consider the effect of processing multiple items simultaneously on the resource requirements of the central network.

Connection Activation Unit Requirements

Consider a CID mechanism containing programmable networks of n_i by n_o units in which we wish to be able to associate each of s different output patterns with each of s different input patterns arising at the same time in different local networks. Input and output patterns consist of m_i by m_o active units, respectively. Following the assumptions for Willshaw nets, we assume binary units and connections, and we assume that output units are turned on only if they receive m quanta of activation.

Now, let us consider how many CA units are needed to implement this mechanism. For now we bypass the bottom-up activation of CA units and assume instead that we know in advance which connections need to be turned on. If each local network must be as complex as a standard network capable of processing r different patterns, we are in serious trouble. In the previous analysis of Willshaw networks, we found that the number of connections we needed to process r associations of m by m active units was

$$n_c = n_i n_o = 1.45 r m_i m_o .$$

It looks as though the number of connections required in each local network grows linearly with the number of known patterns times the content of each. If we had one CA unit for each programmable connection, a programmable version of our square 1-million-pattern associator would require 10^{12} CA units, a figure which is one or two orders of magnitude larger than conventional estimates of the number of units in the brain. Just putting the matter in terms of the cost we must bear to use programmable connections, it appears that we need n^2 CA units just to specify the connections needed for a standard net that could do the same work with just the connections between n input and n output units.[2]

However, things are not nearly as bad as this argument suggests. The computation I just gave misses the very important fact that it is generally not necessary to pinpoint only those connections that are relevant to a particular association. We can do very well if we allow each CA unit to activate a whole *cohort* of connections, as long as (a) we activate all the connections that we need to process any particular pattern of interest, and (b) we do not activate so many that we give rise to an inordinate number of spurious activations of output units.

The idea of using one CA unit for a whole cohort of programmable connections is a kind of coarse coding. In this case, we will see that we can reap a considerable benefit from coarse coding, compared to using one CA unit per connection. A simple illustration of the idea is shown in Figure 5. The figure illustrates CA units projecting to a single one

[2] Many readers will observe that the CA units are not strictly necessary. However, the specificity of their connections to connections in local networks is an issue whether CA units are used as intermediaries or not. Thus, even if the CA units were eliminated, it would not change the relevance of the following results. In a later section, the CA units and central output units will be collapsed into one set of units; in that case , this analysis will apply directly to the number of such units that will be required.

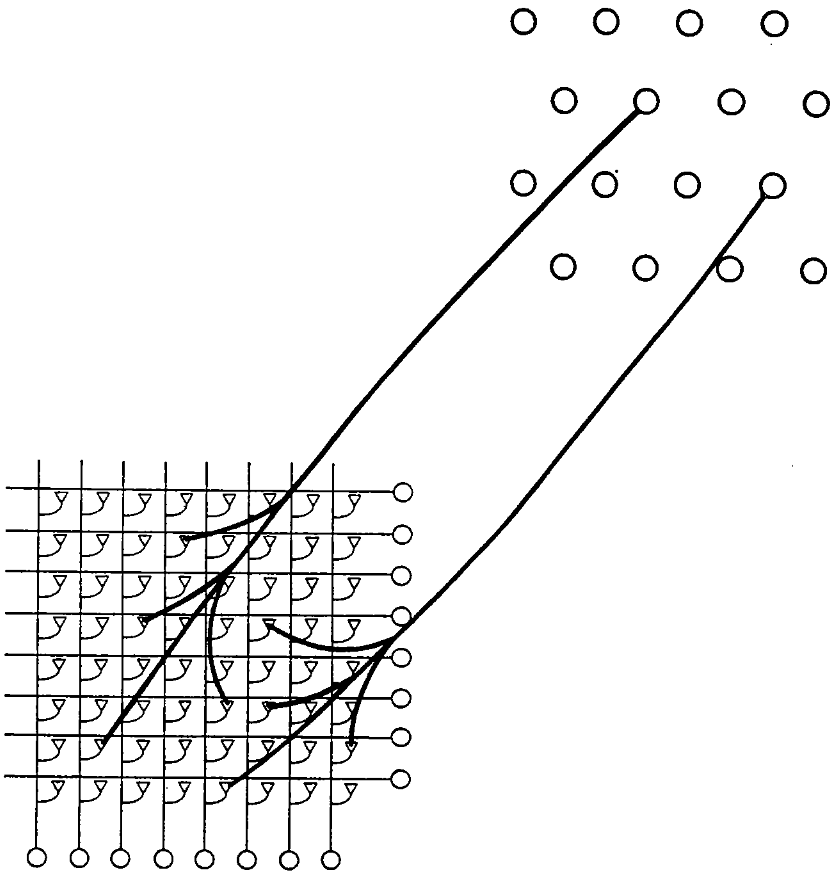

FIGURE 5. A programmable network with 8 input units and 8 output units and 64 programmable connections. Each of the 16 connection activation units is assumed to project to a random set of 4 programmable connections. These connections are only drawn in for two of the CA units. The sets of connections are chosen without replacement so that each connection is programmed by one and only one CA unit. Whenever a CA unit is on it turns on all of the connections it projects to.

of two programmable networks. Note that a given CA unit must activate the same connections in each programmable net when there is more than one.

One Pattern at a Time

To see how much this scheme can buy us, I will start by considering the case in which we want to program some local nets to process a single pattern. We ask, how small a number n_{ca} of CA units can we get by with, assuming that each one activates a distinct, randomly selected set of $n_i n_o / n_{ca}$ connections?

First of all, the number of CA units that must be activated may have to be as large as $m_i m_o$, in case each of the different connections required to process the pattern is a member of a distinct cohort. Second, for comparability to our analysis of the standard network, we want the total fraction of connections turned on to allow no more than an average of 1 output unit to be spuriously activated. As before, this constraint is represented by

$$p_{on} \leqslant \left[\frac{1}{n_i} \right]^{\frac{1}{m_i}}.$$

As long as $m_i \geqslant \log_2 n_i$, .5 will be less than the right-hand side of the expression, so we will be safe if we keep p_{on} less than or equal to .5. Since we may have to activate $m_i m_o$ CA units to activate all the right connections and since we do not want to activate more than half of the connections in all, we conclude that

$$n_{ca} \geqslant 2 m_i m_o.$$

From this result we discover that the number of CA units required *does not depend at all* on the number of connections in each programmable network. Nor in fact does it depend on the number of different known patterns. The number of known patterns does of course influence the complexity of the central network, but it does not affect the number of CA units. The number of CA units depends on $m_i m_o$, the number of connections that need to be turned on per pattern. Obviously, this places a premium on the sparseness of the patterns. Regardless of this, we are much better off than before.

Several Patterns at a Time

So far we have considered the case in which only one item is presented for processing at a time. However, the whole point of the connection information distribution scheme is that it permits simultaneous processing of several different patterns. There is, however, a cost associated with simultaneous processing, since for each pattern we need to turn on all the connections needed to process it. In this situation, we will need to increase the total number of CA units to increase the specificity of the set of connections each association requires if we are to keep the total fraction of connections that have been turned on below .5. Formally, assume that we know which s patterns we want to process. Each one will need to turn on its own set of $m_i m_o$ CA units out of the total number n_{ca} of CA units. The proportion of connections turned on will then be

$$p_{on} = 1 - \left(1 - \frac{m_i m_o}{n_{ca}} \right)^s .$$

This formula is, of course, the same as the one we saw before for the number of connections activated in the standard net with s, the number of different patterns to be processed simultaneously, replacing r, the number of patterns stored in the memory, and with n_{ca}, the number of connection activation units, replacing $n_i n_o$, the total number of connections. Using $p_{on} = .5$ and taking the log of both sides we get

$$-.69 = s \log \left(1 - \frac{m_i m_o}{n_{ca}} \right).$$

Invoking the $\log(1 - x) = -x$ approximation, we obtain

$$n_{ca} \geqslant 1.45sm^2.$$

This formula underestimates n_{ca} slightly for $s < 3$. With this caveat, the number of CA units required is roughly proportional to the number of patterns to be processed at one time, times the number of connections needed to process each pattern.

Overlapping the Programmable Networks

In Chapter 16, the CID scheme we have been considering thus far was generalized to the case where the programmable networks overlapped with each other. This allowed strings of letters starting in any of a large number of input locations to correctly activate units for the corresponding word at the appropriate location at the next higher level. Here I will consider a more general overlapping scheme using distributed representations in the overlapping local networks. A set of three overlapping local networks is illustrated in Figure 6. In this scheme, both the input and the output units can play different roles depending on the alignment of the input pattern with the input units. In consequence, some of the connections also play more than one role. These connections are assumed to be programmable by a number of different CA units, one for each of the connection's different roles. Obviously, this will tend to increase the probability that a connection will be turned

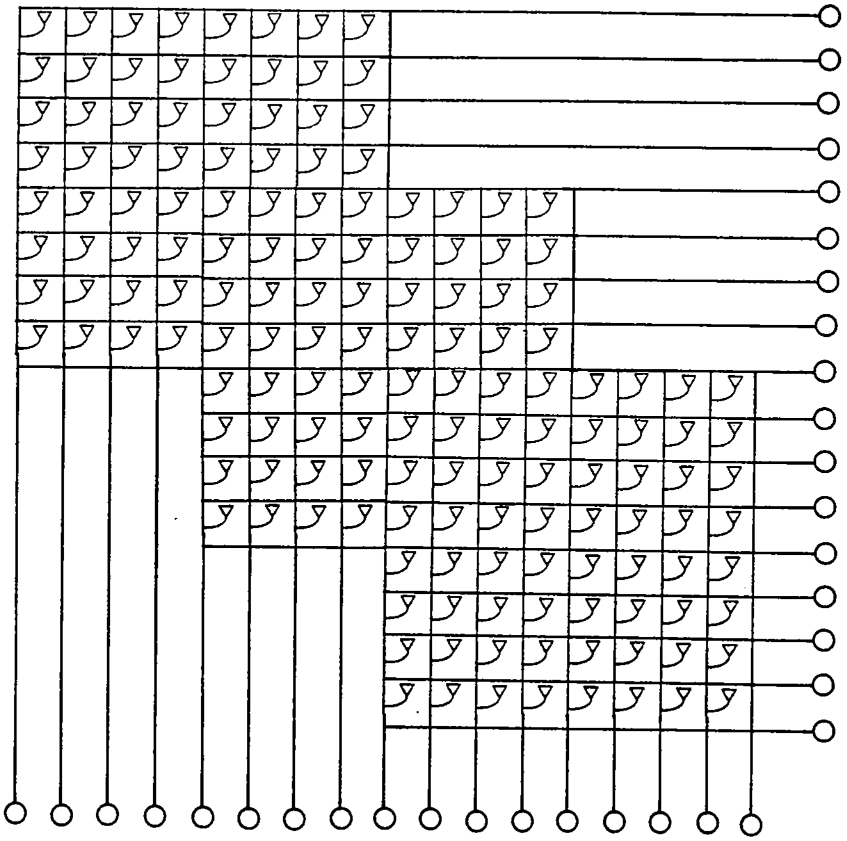

FIGURE 6. Three overlapping programmable networks of 8×8 units each. The networks overlap every four units, so the input and output units can participate in two different, partially overlapping networks.

on, and therefore will require a further revision of our estimate of the number of CA units required.

Unfortunately, an exact mathematical analysis is a bit tricky due to the fact that different junctions have different numbers of opportunities to be turned on. In addition, input patterns in adjacent locations will tend to cross-activate each other's output units. If the patterns to be processed are well separated, this will not be a problem. Restricting our attention to the well-separated case, we can get an upper bound on the cost in CA units of allowing overlapping modules by considering the case where *all* of the connections are assumed to play the maximum number of roles. This number is equivalent to the step size or grain, g, of the overlap, relative to the size of the pattern as a whole. For example, for four-letter words, if the increments in starting places of successive overlapping networks were one letter wide, g would be 4. Assuming that the connections turned on for each slice of a pattern are independent of those turned on by each other slice, it is easy to show that the formula for p_{on} becomes

$$p_{on} \leqslant 1 - \left(1 - \frac{m_i m_o}{n_{ca}} \right)^{sg},$$

and the number of CA units required to keep p_{on} less than .5 is approximated by

$$n_{ca} \geqslant 1.45\, s\, g\, m_i m_o .$$

The cost goes up with the number of patterns to be processed simultaneously times the grain of the overlap.

Summary of CA Unit Requirements

In summary, the number of CA units required to program a programmable network depends on different variables than the number of connections required in a standard associator. We can unify the two analyses by noting that both depend on the number of patterns the net must be ready to process at any given time. For the standard associator, the number is r, the number of known patterns; for the programmable net, the number is sg, the number of patterns the net is programmed for times the grain of overlap allowed in the starting locations of input patterns.

This analysis greatly increases the plausibility of the CID scheme. For we find that the "initial investment" in CA units needed to program a set of networks to process a single association is related to the content of the association or the number of connections required to allow each of the active input elements to send a quantum of activation to each of the active output elements. Incorporating a provision for overlapping networks, we find that the investment required for processing one association is related to the content of the association times the grain of the overlap. This cost is far more reasonable than it looked like it might be at first, and, most importantly, it does not depend on the number of patterns known.

An additional important result is that the cost of programming a set of networks grows with the number of patterns we wish to program for at one time. This cost seems commensurate with the linear speedup we would get by being able to process several patterns simultaneously.

The somewhat intangible benefit to be derived from mutual constraint among the patterns would come over and above the simple linear throughput effect. However, this benefit, as we shall see in the next section, is balanced by the extra cost associated with the possibility that there might be spurious patterns in the intersection of input elements of the presented patterns.

The Cost of Simultaneous Access

So far, we have proceeded as though we already knew what patterns to prepare each local module for. However, the CID mechanism was intended to allow several inputs to access the central network simultaneously and thereby program the local networks in the course of processing. This simultaneous access costs something; in this section we consider how much. The discussion here is relevant to the general issue of the costs of simultaneous access to a PDP network, as well as to the specific question of the capacity requirements of CID.

For simplicity I will begin by considering local representations at the central output level. That is, I will assume that each central output unit represents a different pattern and that it is switched on only when all of the central input units corresponding to its pattern are active.

Now, recall that a central input unit is switched on if the corresponding unit is active in any of the programmable nets. Thus, what the central output units actually see is the pattern of activation that results from the superimposition of the input patterns presented for simultaneous processing. The effect of this is that there is some possibility that ghosts of patterns not actually presented will show up in the result. This is just the kind of situation that is described in Chapter 16 when similar words such as *SAND* and *LANE* are presented to each of two programmable networks for simultaneous processing. When the activation patterns of the two words are superimposed, the central word units for *LAND* and *SANE* get turned on just as strongly as the central word units for *SAND* and *LANE*. Thus, the programmable networks end up being programmed to process any one of these four words, rather than just any one of the two actually presented.

Is there anything that can be done to control the number of different patterns that show up when several patterns are superimposed? In fact, there is. If we increase the number of input units in each programmable network or if we reduce the number of input units active in each pattern, we will reduce the possibility of spurious patterns showing up in the superposition.

To get a quantitative grip on this matter, assume that the input patterns are random selections of m out of the n input units as we have been assuming throughout. The probability that a spurious pattern is present in the superposition of s patterns can now be easily calculated. First, we calculate the probability that a randomly selected unit will be on; this is just

$$1 - (1 - m/n)^s.$$

The probability that a particular spurious pattern is fully represented in the set of units activated by the s patterns is just this number to the power m, and the average number of such patterns out of r known patterns is just this probability times $r - s$. Thus, the average number of spurious patterns present in the superposition is

$$(r - s)[1 - (1 - m/n)^s]^m.$$

Assuming $r >> s$, we can simplify by replacing $r - s$ with r. If we take acceptable performance to be an average of one or fewer spurious patterns present and therefore of spurious CP units active, we get

$$1 = r[1 - (1 - m/n)^s]^m.$$

Rearranging and taking logs,

$$\log\left[1 - (\frac{1}{r})^{\frac{1}{m}}\right] = s\log(1 - m/n).$$

Several things are apparent from this equation. First, the number of patterns that can be processed at one time increases with the number of input units. The effect is approximately linear as long as $m/n \leqslant .1$. Second, though it is not quite as straightforward, s tends to increase with a decrease in m. For example, suppose $n = 5{,}000$ and $r = 10{,}000$. In this case, when m drops from 1,000 to 500, s increases from 21 to about 37; if m drops to 100, s goes up to about 120. Third, for a fixed m and n, especially for large m, we can make very large changes in r with only minimal impact on s. Thus, if we have, say, $n = 10{,}000$ and $m = 1{,}000$ with $r = 10^6$, we get $s = 43$; if we reduce r to 10^5, we only get an increase of 2 in s, to 45.

If we allow overlapping local networks, and we assume that the patterns are random with independent subparts, we need only replace s in the preceeding equation with sg. While this is a fairly steep cost, it is still the case that reasonably moderate values of n (about 2.5×10^5) would be sufficient to process 10 out of 10^6 known patterns of size 1,000 simultaneously with a grain of 100.

Simultaneous Access to Distributed Representations

The results just described, it must be remembered, depend on the use of *local* representations at the central output level. What happens if we consider simultaneously accessing distributed representations instead? Obviously this question remains relevant to general questions about simultaneous access, as well as to the situation that would arise using distributed central output units in CID. Furthermore, we should note that the central output units in Figure 4 simply mediate a mapping from one distributed representation—on the central input units—to another—on the CA units. The present analysis describes what would happen if we simply collapsed these two sets of units into one, activating the connections directly from the central output units.

We consider a case exactly like the one we were just considering, except that now the output representation is not a single unit per pattern, but m_o active units on out of n_o central output units. We consider two somewhat separate questions. First, if we superimpose several input patterns, what effect does this have on d' at the central output level, relative to the case where only a single pattern is shown? Second, what is the probability that ghosts of whole patterns not presented will show up in the *output* of the central network?

To begin our analysis of the first question, recall from Equation 2 the expression for d' in random nets with full fan-out ($n = f$):

$$d' = \sqrt{m_i}\,\frac{1 - p_{on}}{\sqrt{(1 + p_{on})/2}}.$$

We first ask, what is the effect on d' of turning on spurious input units with probability ρ, in addition to the m units representing a particular pattern to be processed? The number M_i of input units that will then be on is

$$M_i = m_i + (n_i - m_i)\rho.$$

Consider first, output units that should not be on. These will receive M_i active inputs, and each of these connections will be on with probability p_{on}. The output units that should be on will receive m_i inputs on the input lines whose connections were turned on in learning the presented pattern plus $(n_i - m_i)\rho$ inputs to connections that will have been turned on in learning other patterns with probability p_{on}. The numerator for our revised expression will then simply reduce to its old value, with the $(n_i - m_i)\rho$ term canceling out. However, there will be an increase in variance, and hence a decrease in d'. The denominator is as before the square root of the average of the variances of the two means, which are, also as before, equal to the means. The expression for d' therefore becomes

$$d' = m \frac{(1 - p_{on})}{\sqrt{[m_i + m_i p_{on} + 2(n_i - m_i)\rho]/2}}.$$

We get a simpler expression if we approximate by replacing p_{on} in the denominator with its maximum value of 1; this gives us a slight overestimate of the variance and therefore a slight underestimate of d':

$$d' \geqslant m \frac{(1 - p_{on})}{\sqrt{m_i + (n_i - m_i)\rho}}.$$

The variance goes up with the mean number of spuriously activated units, and d' goes down with the effect of this on the square root of the variance.

To determine the effect of presenting several patterns on d', we note that from the point of view of the units that belong to one of the patterns, all the units activated by the other patterns are spurious. The number of such units is

$$1 - (1 - m_i/n_i)^{s-1}$$

Inserting this for ρ in the previous equation gives

$$d' = m \frac{(1 - p_{on})}{\sqrt{m_i + (n_i - m_i)[1 - (1 - m_i/n_i)^{s-1}]}}.$$

Using this equation we can examine the effects of increasing s on the value of d'. Not too surprisingly, d' does go down as s goes up, but the effect is relatively benign. For example, with $n = 10^6$, $r = 10^6$, $m = 1{,}000$, and $s = 1$, d' is about 11.6. It drops to half that value at $s = 4$, and drops much more gradually thereafter. With $n = 2 \times 10^6$ units and the same values of r and m, we can get an acceptable value of d' ($\geqslant 6$) with s as high as 16.

The final issue we will consider is the possibility that new spurious output patterns have been introduced in the superposition of the s output patterns simultaneously activated in processing the s input patterns. For simplicity, we will just consider the probability of a "ghost," given that all and only the correct m_o units are active for each of the s patterns. The analysis is entirely the same as the one we gave before for the probability of ghosts showing up in the input patterns. We get an average of one ghost when

$$1 \geqslant r\,[1 - (1 - m/n)^s]^m.$$

As before, the number of simultaneous patterns we can tolerate increases with n and decreases with m and is relatively insensitive to the value of r.

In general, it appears that the probability of ghosts occurring can be kept small with sufficiently large dedication of resources, but these trade off approximately linearly with s. With fixed n, we must simply make the patterns sparser or tolerate some loss of.sensitivity.

Discussion

This analysis of the resource requirements of networks like the CID model has discovered a number of basic results. In essence, the picture is really a very simple one. The resource requirements of CID depend on the number of programmable networks one wants to program for at once. The number of connections needed in each local network depends on the number of patterns to be programmed for and is independent of r, the number of known patterns. In the central network, the number of units required to keep spurious activations under control grows with s, as does the number of units required to keep ghosts from emerging in the input and output patterns. It is worth noting, also, that the probability of ghosts *increases* as we increase m.

The fact that the resource requirements of the local networks are independent of the number of patterns known is obviously important. Relative to the central network, it means that the local networks are very cheap. The number of distinct inputs that are needed to program them is quite reasonable, and, as I will explain, we can even get by with far fewer units in the local networks than we need at the central level.

On the other hand, the results concerning the costs of simultaneous access to the central network are much less encouraging for the CID scheme. Using local or distributed representations in the central module, the unit requirements grow with the product of s and r—a very expensive proposition since the number of central connections will then grow as sr^2.

However, there are several important further observations. One is that, at fixed numbers of units and patterns known, the degradation of sensitivity as a function of s is rather gradual. And, given a lightly loaded network, one can take s up to reasonable values without catastrophe. Simultaneous access by multiple patterns is very much like degradation: a network can handle it without a noticeable decrement of function if it is lightly loaded. A second observation concerns the limits of coarse coding. For fixed n, the choice of m essentially amounts to a question of how coarse the code is: Large *m* corresponds to very coarse coding, and small *m* corresponds to very fine coding. As we saw in Chapter 3, the ability to represent several patterns at a time goes down as the coding gets coarser. For simultaneous processing we need sparse patterns, with each unit serving as a rather sharply tuned conjunctive detector.

The final observation is that large costs are associated with simultaneous access to the central network. This fact has lead me to the view that it is probably most reasonable to imagine that we must probably restrict simultaneous access, except perhaps in the case of small, compact and well-differentiated subpatterns like letters. I incorporated this idea of restricted access in the programmable blackboard model of reading by assuming that we program successive parts of the programmable blackboard sequentially, using only the contents of the spotlight of attention to access the central network; but that the local networks so programmed continue to process and hold patterns of activation and to allow those patterns to interact with one and other after the spotlight of attention has moved on. In this way we get the best of both worlds: sequential access to central knowledge, combined with interactive parallel processing of several stimuli in the programmable nets. Another point is that it may be a good idea to dissociate the inputs to the local networks and the inputs to the central networks. Throughout this chapter and Chapter 16, I have assumed that the units in each local network would be isomorphic to units in the central network. However, there is no reason for them to be. The central network needs much higher "resolution" (n proportional to r) than the local networks (n proportional to s). Thus, the units in the programmable modules need only provide a few primitive clues to which of the s patterns are to be represented in their outputs, while the units in the central network would require a much higher-resolution representation.

CONCLUSION

This chapter has indicated how Willshaw's fruitful analysis of simple pattern associator models can be extended in several directions. These extensions have lead to several interesting observations, particularly into the effects of limited connectivity (Mitchison, personal communication, 1984) and into the capacity requirements of programmable networks. A large number of issues remain to be explored. I hope that this discussion and elaboration of Willshaw's analysis will aid in this continuing exploration.

ACKNOWLEDGMENTS

This work was supported by Contract N-00014-82-C-0374, NR 667-483 with the Personnel and Training Research Programs of the Office of Naval Research, by a grant from the System Development Foundation to the Institute for Cognitive Science at UCSD, and by an NIMH Research Scientist Career Development Award (MH-00385). This chapter was developed in response to a number of questions raised by Geoff Hinton and Scott Fahlman about the resource requirements of programmable nets. I thank Dave Rumelhart for several useful discussions and for encouraging me to pursue the issues described herein. The material described in the section entitled "Randomly Connected Nets" was developed in collaboration with Dave, and the application of the *d'* analysis to the problem of simultaneous access to a distributed memory network was Dave's suggestion.