Foundations of Cognitive Science

edited by
Michael I. Posner

A Bradford Book
The MIT Press
Cambridge, Massachusetts
London, England

4

The Architecture of Mind: A Connectionist Approach

David E. Rumelhart

Cognitive science has a long-standing and important relationship to the computer. The computer has provided a tool whereby we have been able to express our theories of mental activity; it has been a valuable source of metaphors through which we have come to understand and appreciate how mental activities might arise out of the operations of simple-component processing elements.

I recall vividly a class I taught some fifteen years ago in which I outlined the then-current view of the cognitive system. A particularly skeptical student challenged my account with its reliance on concepts drawn from computer science and artificial intelligence with the question of whether I thought my theories would be different if it had happened that our computers were parallel instead of serial. My response, as I recall, was to concede that our theories might very well be different, but to argue that that wasn't a bad thing. I pointed out that the inspiration for our theories and our understanding of abstract phenomena always is based on our experience with the technology of the time. I pointed out that Aristotle had a wax tablet theory of memory, that Leibniz saw the universe as clockworks, that Freud used a hydraulic model of libido flowing through the system, and that the telephone-switchboard model of intelligence had played an important role as well. The theories posited by those of previous generations had, I suggested, been useful in spite of the fact that they were based on the metaphors of their time. Therefore, I argued, it was natural that in our generation—the generation of the serial computer—we should draw our insights from analogies with the most advanced technological developments of our time. I don't now remember whether my response satisfied the student, but I have no doubt that we in cognitive science have gained much of value through our use of concepts drawn from our experience with the computer.

In addition to its value as a source of metaphors, the computer differs from earlier technologies in another remarkable way. The computer can be made to *simulate* systems whose operations are very different from the computers on which these simulations run. In this way we can use the computer to simulate systems with which we *wish* to have experience and thereby provide a source of experience that can be drawn upon in giving us new metaphors and new insights into how mental operations might be accomplished. It is this use of the computer that the connectionists have employed. The architecture that we are exploring is not one based on the von Neumann architecture of our current generation of computers but rather an architecture based on considerations of how brains themselves might function. Our strategy has thus become one of offering a general and abstract model of the computational architecture of brains, to develop algorithms and procedures well suited to this architecture, to simulate these procedures and architecture on a computer, and to explore them as hypotheses about the nature of the human information-processing system. We say that such models are *neurally inspired,* and we call computation on such a system *brain-style computation*. Our goal in short is to replace the computer metaphor with the brain metaphor.

4.1 Why Brain-Style Computation?

Why should a brain-style computer be an especially interesting source of inspiration? Implicit in the adoption of the computer metaphor is an assumption about the appropriate level of explanation in cognitive science. The basic assumption is that we should seek explanation at the *program* or *functional* level rather than the implementational level. It is thus often pointed out that we can learn very little about what kind of program a particular computer may be running by looking at the electronics. In fact we don't care much about the details of the computer at all; all we care about is the particular program it is running. If we know the program, we know how the system will behave in any situation. It doesn't matter whether we use vacuum tubes or transistors, whether we use an IBM or an Apple, the essential characteristics are the same. This is a very misleading analogy. It is true for computers because they are all essentially the same. Whether we make them out of vacuum tubes or transistors, and whether we use an IBM or an Apple computer, we are using computers of the same general design. When we look at essentially different architecture, we see that the architecture makes a good deal of difference. It is the architecture that determines which kinds of algorithms are most easily carried out on the machine in question. It is the architecture of the machine that determines the essential nature of the program itself. It is thus reasonable that we should begin by asking what we know about the architecture of the brain and how it might shape the algorithms underlying biological intelligence and human mental life.

The basic strategy of the connectionist approach is to take as its fundamental processing unit something close to an abstract neuron. We imagine that computation is carried out through simple interactions among such processing units. Essentially the idea is that these processing elements communicate by sending numbers along the lines that connect the processing elements. This identification already provides some interesting constraints on the kinds of algorithms that might underlie human intelligence.

The operations in our models then can best be characterized as "neurally inspired." How does the replacement of the computer metaphor with the brain metaphor as model of mind affect our thinking? This change in orientation leads us to a number of considerations that further inform and constrain our model-building efforts. Perhaps the most crucial of these is time. Neurons are remarkably slow relative to components in modern computers. Neurons operate in the time scale of milliseconds, whereas computer components operate in the time scale of nanoseconds—a factor of 10^6 faster. This means that human processes that take on the order of a second or less can involve only a hundred or so time steps. Because most of the processes we have studied—perception, memory retrieval, speech processing, sentence comprehension, and the like—take about a second or so, it makes sense to impose what Feldman (1985) calls the "100-step program" constraint. That is, we seek explanations for these mental phenomena that do not require more than about a hundred elementary sequential operations. Given that the processes we seek to characterize are often quite complex and may involve consideration of large numbers of simultaneous constraints, our algorithms *must* involve considerable parallelism. Thus although a serial computer could be created out of the kinds of components represented by our units, such an implementation would surely violate the 100-step program constraint for any but the simplest processes. Some might argue that although parallelism is obviously present in much of human information processing, this fact alone need not greatly modify our world view. This is unlikely. The speed of components is a critical design constraint. Although the brain has *slow* components, it has *very many* of them. The human brain contains billions of such processing elements. Rather than organize computation with many, many serial steps, as we do with systems whose steps are very fast, the brain must deploy many, many processing elements cooperatively and in parallel to carry out its activities. These design characteristics, among others, lead, I believe, to a general organization of computing that is fundamentally different from what we are used to.

A further consideration differentiates our models from those inspired by the computer metaphor—that is, the constraint that all the knowledge is *in the connections*. From conventional programmable computers we are used to thinking of knowledge as being stored in the state of certain units in the system. In our systems we assume that only very short-term storage can occur in the states of units; long-term storage takes place in the connections among units. Indeed it is the connections—or perhaps the rules for forming them through experience—that primarily differentiate one model from another. This is a profound difference between our approach and other more conventional approaches, for it means that almost all knowledge is *implicit* in the structure of the device that carries out the task rather than *explicit* in the states of units themselves. Knowledge is not directly accessible to interpretation by some separate processor, but it is built into the processor itself and directly determines the course of processing. It is acquired through tuning of connections as these are used in processing, rather than formulated and stored as declarative facts.

These and other neurally inspired classes of working assumptions have been one important source of assumptions underlying the connectionist program of research. These have not been the only considerations. A second class of constraints arises from our beliefs about the nature of human information processing considered at a more abstract, computational level of analysis. We see the kinds of phenomena we have been studying as products of a kind of constraint-satisfaction procedure in which a very large number of constraints act simultaneously to produce the behavior. Thus we see most behavior not as the product of a single, separate component of the cognitive system but as the product of large set of interacting components, each mutually constraining the others and contributing in its own way to the globally observable behavior of the system. It is very difficult to use serial algorithms to implement such a conception but very natural to use highly parallel ones. These problems can often be characterized as *best-match* or *optimization* problems. As Minsky and Papert (1969) have pointed out, it is very difficult to solve best-match problems serially. This is precisely the kind of problem, however, that is readily implemented using highly parallel algorithms of the kind we have been studying.

The use of brain-style computational systems, then, offers not only a hope that we can characterize how brains actually carry out certain information-processing tasks but also solutions to computational problems that seem difficult to solve in more traditional computational frameworks. It is here where the ultimate value of connectionist systems must be evaluated.

In this chapter I begin with a somewhat more formal sketch of the computational framework of connectionist models. I then follow with a general discussion of the kinds of computational problems that connectionist models seem best suited for. Finally, I will briefly review the state of the art in connectionist modeling.

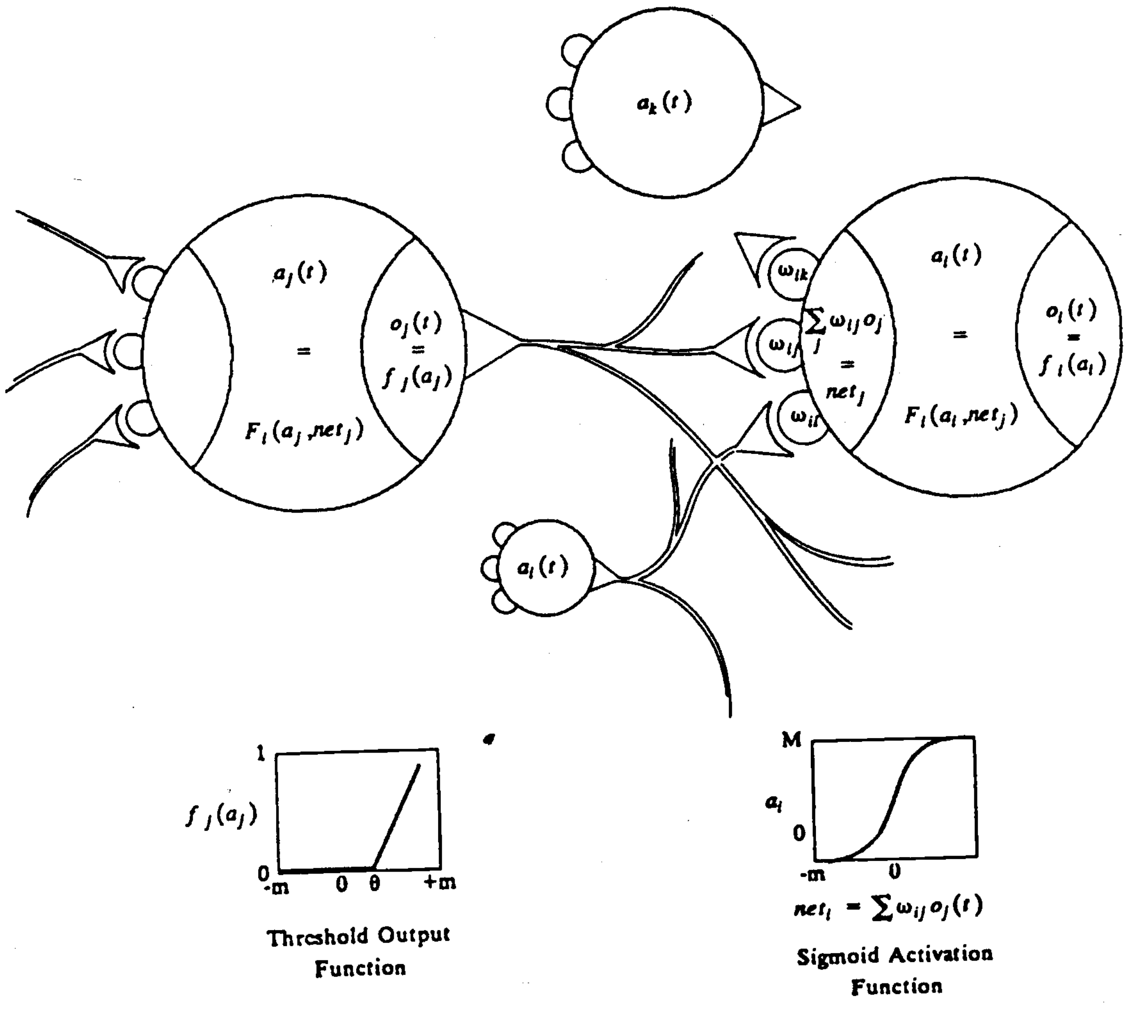
The Connectionist Framework

There are seven major components of any connectionist system:

- a *set of processing units;*
- a *state of activation* defined over the processing units;

- an *output function* for each unit that maps its state of activation into an output;
- a *pattern of connectivity* among units;
- an *activation rule* for combining the inputs impinging on a unit with its current state to produce a new level of activation for the unit;
- a *learning rule* whereby patterns of connectivity are modified by experience;
- an *environment* within which the system must operate.

Figure 4.1 illustrates the basic aspects of these systems. There is a set of processing units, generally indicated by circles in my diagrams; at each point in time each unit u_i has an activation value, denoted in the diagram as $a_i\ (t)$; this activation value is passed through a function f_i to produce an output value $o_i(t)$. This output value can be seen as passing through a set of unidirectional connections (indicated by lines or arrows in the diagrams) to other units in the system. There is associated with



$a_k(t)$
$a_j(t) = F_i(a_j, net_j)$
$o_j(t) = f_j(a_j)$
ω_{ik}
ω_{ij}
ω_{il}
$\sum_j \omega_{ij} o_j = net_j$
$a_i(t) = F_i(a_i, net_j)$
$o_i(t) = f_i(a_i)$
$a_l(t)$
1
$f_j(a_j)$
0
-m 0 θ +m
Threshold Output Function
M
a_i
0
-m 0
$net_i = \sum \omega_{ij} o_j(t)$
Sigmoid Activation Function

Figure 4.1 The basic components of a parallel distributed processing system

each connection a real number, usually called the *weight* or *strength* of the connection, designated w_{ij}, which determines the affect that the first unit has on the second. All of the inputs must then be combined, and the combined inputs to a unit (usually designated the *net input* to the unit) along with its current activation value determine its new activation value via a function F. These systems are viewed as being plastic in the sense that the pattern of interconnections is not fixed for all time; rather the weights can undergo modification as a function of experience. In this way the system can evolve. What a unit represents can change with experience, and the system can come to perform in substantially different ways.

A Set of Processing Units Any connectionist system begins with a set of processing units. Specifying the set of processing units and what they represent is typically the first stage of specifying a connectionist model. In some systems these units may represent particular conceptual objects such as features, letters, words, or concepts; in others they are simply abstract elements over which meaningful patterns can be defined. When we speak of a distributed representation, we mean one in which the units represent small, featurelike entities we call *microfeatures*. In this case it is the pattern as a whole that is the meaningful level of analysis. This should be contrasted to a *one-unit–one-concept* or *localist* representational system in which single units represent entire concepts or other large meaningful entities.

All of the processing of a connectionist system is carried out by these units. There is no executive or other overseer. There are only relatively simple units, each doing its own relatively simple job. A unit's job is simply to receive input from its neighbors and, as a function of the inputs it receives, to compute an output value, which it sends to its neighbors. The system is inherently parallel in that many units can carry out their computations as the same time.

Within any system we are modeling, it is useful to characterize three types of units: *input*, *output*, and *hidden* units. Input units receive inputs from sources external to the system under study. These inputs may be either sensory inputs or inputs from other parts of the processing system in which the model is embedded. The output units send signals out of the system. They may either directly affect motoric systems or simply influence other systems external to the ones we are modeling. The hidden units are those whose only inputs and outputs are within the system we are modeling. They are not "visible" to outside systems.

The State of Activation In addition to the set of units we need a representation of the state of the system at time t. This is primarily specified by a vector $\mathbf{a}(t)$, representing the pattern of activation over the set of processing units. Each element of the vector stands for the activation of one of the units. It is the pattern of activation over the set of

u_j inhibits unit u_i; and it is 0 if unit u_j has no direct connection to unit u_i. The absolute value of w_{ij} specifies the *strength of the connection.*

The pattern of connectivity is very important. It is this pattern that determines what each unit represents. One important issue that may determine both how much information can be stored and how much serial processing the network must perform is the *fan-in* and *fan-out* of a unit. The fan-in is the number of elements that either excite or inhibit a given unit. The fan-out of a unit is the number of units affected directly by a unit. It is useful to note that in brains these numbers are relatively large. Fan-in and fan-out range as high as 100,000 in some parts of the brain. It seems likely that this large fan-in and fan-out allows for a kind of operation that is less like a fixed circuit and more statistical in character.

Activation Rule We also need a rule whereby the inputs impinging on a particular unit are combined with one another and with the current state of the unit to produce a new state of activation. We need function **F**, which takes $\mathbf{a}(t)$ and the net inputs, $\text{net}_i = \Sigma_j\, w_{ij}o_j(t)$, and produces a new state of activation. In the simplest cases, when **F** is the identity function, we can write $\mathbf{a}(t + 1) = \mathbf{Wo}(t) = \mathbf{net}\ (t)$. Sometimes **F** is a threshold function so that the net input must exceed some value before contributing to the new state of activation. Often the new state of activation depends on the old one as well as the current input. The function **F** itself is what we call the activation rule. Usually the function is assumed to be deterministic. Thus, for example, if a threshold is involved it may be that $a_i(t) = 1$ if the total input exceeds some threshold value and equals 0 otherwise. Other times it is assumed that **F** is stochastic. Sometimes activations are assumed to decay slowly with time so that even with no external input the activation of a unit will simply decay and not go directly to zero. Whenever $a_i(t)$ is assumed to take on continuous values, it is common to assume that **F** is a kind of sigmoid function. In this case an individual unit can *saturate* and reach a minimum or maximum value of activation.

Modifying Patterns of Connectivity as a Function of Experience Changing the processing or knowledge structure in a connectionist system involves modifying the patterns of interconnectivity. In principle this can involve three kinds of modifications:

1. development of new connections;
2. loss of existing connections;
3. modification of the strengths of connections that already exist.

Very little work has been done on (1) and (2). To a first order of approximation, however, (1) and (2) can be considered a special case of (3). Whenever we change the strength of connection away from zero to some positive or negative value, it has the same effect as growing a new connection. Whenever we change the strength of a connection to zero, that has the same effect as losing an existing connection. Thus we have concentrated on rules whereby *strengths* of connections are modified through experience.

Virtually all learning rules for models of this type can be considered a variant of the *Hebbian* learning rule suggested by Hebb (1949) in his classic book *Organization of Behavior*. Hebb's basic idea is this: If a unit u_i receives a input from another unit u_j, then, if both are highly active, the weight w_{ij} from u_j to u_i should be *strengthened*. This idea has been extended and modified so that it can be more generally stated as

$$\delta w_{ij} = g\,(a_i(t), t_i(t))h(o_j(t), w_{ij}),$$

where $t_i(t)$ is a kind of *teaching* input to u_i. Simply stated, this equation says that the change in the connection from u_j to u_i is given by the product of a function $g()$ of the activation of u_i and its teaching input t_i and another function $h()$ of the output value of u_j and the connection strength w_{ij}. In the simplest versions of Hebbian learning, there is no teacher and the functions g and h are simply proportional to their first arguments. Thus we have

$$\delta w_{ij} = \epsilon a_i o_j,$$

where ϵ is the constant of proportionality representing the learning rate. Another common variation is a rule in which $h(o_j(t), w_{ij}) = o_j(t)$ and $g(a_i(t), t_i(t)) = \epsilon(t_i(t) - a_i(t))$. This is often called the *Widrow-Hoff*, because it was originally formulated by Widrow and Hoff (1960), or the *delta rule*, because the amount of learning is proportional to the *difference* (or delta) between the actual activation achieved and the target activation provided by a teacher. In this case we have

$$\delta w_{ij} = \epsilon(t_i(t) - a_i(t))o_j(t).$$

This is a generalization of the *perceptron* learning rule for which the famous *perception convergence theorem* has been proved. Still another variation has

$$\delta w_{ij} = \epsilon a_i(t)(o_i(t) - w_{ij}).$$

This is a rule employed by Grossberg (1976) and others in the study of *competitive learning*. In this case usually only the units with the strongest activation values are allowed to learn.

Representation of the environment It is crucial in the development of any model to have a clear representation of the environment in which this model is to exist. In connectionist models we represent the environment as a time-varying stochastic function over the space of input patterns. That is, we imagine that at any point in time there is some probability that any of the possible set of input patterns is impinging on the input units. This probability function may in general depend on the history of inputs to the system as well as outputs of the system. In practice most connectionist models involve a much simpler characterization of the environment. Typically the environment is characterized by a stable probability distribution over the set of possible input patterns independent of past inputs and past responses of the system. In this case we can imagine listing the set of possible inputs to the system and numbering them from 1 to M. The environment is then characterized by a set of probabilities p_i for $i = 1, \ldots, M$. Because each input pattern can be considered a vector, it is sometimes useful to characterize those patterns with nonzero probabilities as constituting *orthogonal* or *linearly independent* sets of vectors.

To summarize, the connectionist framework consists not only of a formal language but also a perspective on our models. Other qualitative and quantitative considerations arising from our understanding of brain processing and of human behavior combine with the formal system to form what might be viewed as an aesthetic for our model-building enterprises.

Computational Features of Connectionist Models

In addition to the fact that connectionist systems are capable of exploiting parallelism in computation and mimicking brain-style computation, connectionist systems are important because they provide good solutions to a number of very difficult computational problems that seem to arise often in models of cognition. In particular they are good at solving constraint-satisfaction problems, implementing content-addressable memory-storage systems, and implementing best match; they allow for the automatic implementation of similarity-based generalization; they exhibit graceful degradation with damage or information overload; and there are simple, general mechanisms for learning that allow connectionist systems to adapt to their environments.

Constraint Satisfaction Many cognitive-science problems are usefully conceptualized as constraint-satisfaction problems in which a solution is given through the satisfaction of a very large number of mutually interacting constraints. The problem is to devise a computational algorithm that is capable of efficiently implementing such a system. Connectionist systems are ideal for implementing such a constraint-satisfaction system, and the trick for getting connectionist networks to solve difficult problems is often to cast the problem as a constraint-satisfaction problem. In this case we conceptualize the connectionist network as a *constraint network* in which each unit represents a hypothesis of some sort (for example, that a certain semantic feature, visual feature, or acoustic feature is present in the input) and in which each connection represents constraints among the hypotheses. Thus, for example, if feature B is expected to be present whenever feature A is present, there should be a positive connection from the unit corresponding to the hypothesis that A is present to the unit representing the hypothesis that B is present. Similarly if there is a constraint that whenever A is present B is expected *not* to be present, there should be a negative connection from A to B. If the constraints are weak, the weights should be small. If the constraints are strong, then the weights should be large. Similarly the inputs to such a network can also be thought of as constraints. A positive input to a particular unit means that there is evidence from the outside that the relevant feature is present. A negative input means that there is evidence from the outside that the feature is not present. The stronger the input, the greater the evidence. If such a network is allowed to run, it will eventually *settle* into a locally optimal state in which as many as possible of the constraints are satisfied, with priority given to the strongest constraints. (Actually, these systems will find a *locally* best solution to this constraint satisfaction problem. *Global* optima are more difficult to find.) The procedure whereby such a system *settles* into such a state is called *relaxation*. We speak of the system *relaxing* to a solution. Thus a large class of connectionist models contains constraint satisfaction models that settle on locally optimal solutions through the process of relaxation.

Figure 4.2 shows an example of a simple 16-unit constraint network. Each unit in the network represents a hypothesis concerning a vertex in a line drawing of a Necker cube. The network consists of two interconnected subnetworks—one corresponding to each of the two global interpretations of the Necker cube. Each unit in each network is assumed to receive input from the region of the input figure—the cube—corresponding to its location in the network. Each unit in figure 4.2 is labeled with a three-letter sequence indicating whether its vertex is hypothesized to be front or back (F or B), upper or lower (U or L), and right or left (R or L). Thus, for example, the lower-left unit of each subnetwork is assumed to receive input from the lower-left vertex of the input figure. The unit in the left network represents the hypothesis that it is receiving input from a lower-left vertex in the front surface of the cube (and is thus labeled FLL), whereas the one in the right subnetwork represents the hypothesis that it is receiving input from a lower-left vertex in the back surface (BLL). Because there is a constraint that each vertex has a single interpretation, these two units are connected by a strong negative connection. Because the interpretation of any given vertex is constrained by the interpretations of its neighbors, each unit in a subnetwork is connected positively with each of its neighbors within the network. Finally there is the constraint that there can be only one vertex of a single kind (for example, there can be only one lower-left vertex in the front plane FLL). There is a strong negative connection between units representing the same label in each subnetwork. Thus each unit has three neighbors connected positively, two competitors connected negatively, and one positive input from the stim-

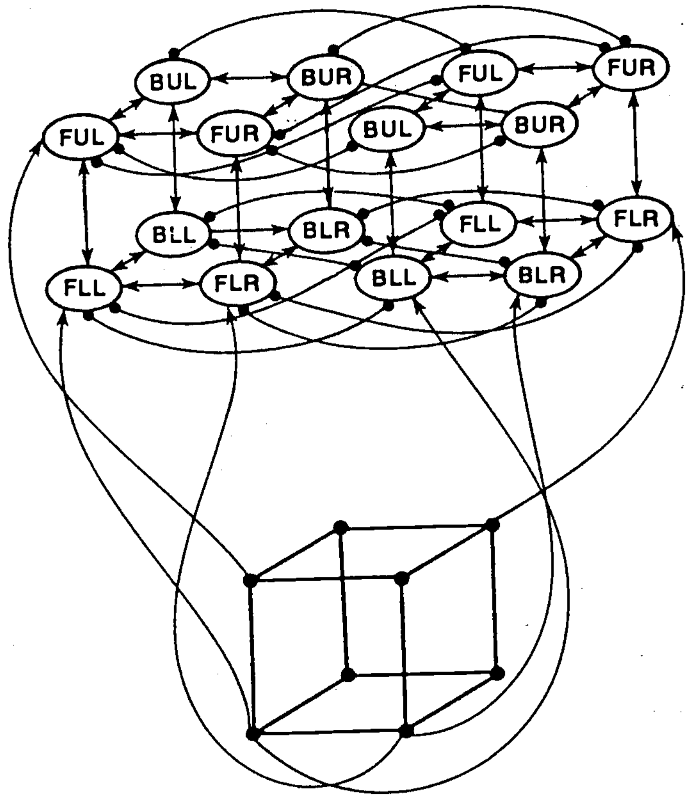


Figure 4.2 A simple network representing some constraints involved in perceiving a Necker cube

ulus. For purposes of this example the strengths of connections have been arranged so that two negative inputs exactly balance three positive inputs. Further it is assumed that each unit receives an excitatory input from the ambiguous stimulus pattern and that each of these excitatory influences is relatively small. Thus if all three of a unit's neighbors are on and both of its competitors are on, these effects would entirely cancel out one another; and if there were a small input from the outside, the unit would have a tendency to come on. On the other hand if fewer than three of its neighbors were on and both of its competitors were on, the unit would have a tendency to turn off, even with an excitatory input from the stimulus pattern.

In the preceding paragraph I focused on the individual units of the networks. It is often useful to focus not on the units, however, but on entire *states* of the network. In the case of binary (on-off or 0-1) units, there is a total of 2^{16} possible states in which this system could reside. That is, in principle each of the 16 units could have either value 0 or 1. In the case of continuous units, in which each unit can take on any value between 0 and 1, the system can in principle take on any of an infinite number of states. Yet because of the constraints built into the network, there are only a few of those states in which the system will settle. To see this, consider the case in which the units are updated asynchronously, one at a time. During each time slice one of the units is chosen to update. If its net input exceeds 0, its value will be pushed toward 1; otherwise its value will be pushed toward 0.

Imagine that the system starts with all units off. A unit is then chosen at random to be updated. Because it is receiving a slight positive input from the stimulus and no other inputs, it will be given a positive activation value. Then another unit is chosen to update. Unless it is in direct competition with the first unit, it too will be turned on. Eventually a coalition of neighboring units will be turned on. These units will tend to turn on more of their neighbors in the same subnetwork and turn off their competitors in the other subnetwork. The system will (almost always) end up in a situation in which all of the units in one subnetwork are fully activated and none of the units in the other subnetwork is activated. That is, the system will end up interpreting the Necker cube as either facing left or facing right. Whenever the system gets into a state and stays there, the state is called a *stable state* or a *fixed point* of the network. The constraints implicit in the pattern of connections among the units determine the set of possible stable states of the system and therefore the set of possible interpretations of the inputs.

Hopfield (1982) has shown that it is possible to give a general account of the behavior of systems such as this one (with symmetric weights and asynchronous updates). In particular Hopfield has shown that such systems can be conceptualized as minimizing a global measure, which he calls the *energy* of the system, through a method of *gradient descent* or, equivalently, maximizing the constraints satisfied through a method of *hill climbing*. In particular Hopfield has shown that the system operates in such a way as to always move from a state that satisfies fewer constraints to a state that satisfies more constraints, where the measure of constraint satisfaction is given by

$$G(t) = \sum_i \sum_j w_{ij} a_i(t) a_j(t) + \sum_i \text{input}_i(t) a_i(t).$$

Essentially the equation says that the overall goodness of fit is given by the sum of the degrees to which each pair of units contributes to the goodness plus the degree to which the units satisfy the input constraints. The contribution of a pair of units is given by the product of their activation values and the weights connecting them. Thus if the weight is positive, each unit wants to be as active as possible—that is, the activation values for these two units should be pushed toward 1. If the weight is negative, then at least one of the units should be 0 to maximize the pairwise goodness. Similarly if the input constraint for a given unit is positive, then its contribution to the total goodness of fit is maximized by being the activation of that unit toward its maximal value. If it is negative, the activation value should be decreased toward

0. Of course the constraints will generally not be totally consistent. Sometimes a given unit may have to be turned on to increase the function in some ways yet decrease it in other ways. The point is that it is the sum of all of these individual contributions that the system seeks to maximize. Thus for every state of the system—every possible pattern of activation over the units—the pattern of inputs and the connectivity matrix **W** determine a value of the goodness-of-fit function. The system processes its input by moving upward from state to adjacent state until it reaches a state of maximum goodness. When it reaches such a *stable state* or *fixed point*, it will stay in that state and it can be said to have "settled" on a solution to the constraint-satisfaction problem or alternatively, in our present case, "settled into an interpretation" of the input.

It is important to see then that entirely *local* computational operations, in which each unit adjusts its activation up or down on the basis of its net input, serve to allow the network to converge toward states that maximize a *global* measure of goodness or degree of constraint satisfaction. Hopfield's main contribution to the present analysis was to point out this basic fact about the behavior of networks with symmetrical connections and asynchronous update of activations.

To summarize, there is a large subset of connectionist models that can be considered constraint-satisfaction models. These networks can be described as carrying out their information processing by climbing into states of maximal satisfaction of the constraints implicit in the network. A very useful concept that arises from this way of viewing these networks is that we can describe the behavior of these networks not only in terms of the behavior of individual units but also in terms of properties of the network itself. A primary concept for understanding these network properties is the *goodness-of-fit landscape* over which the system moves. Once we have correctly described this landscape, we have described the operational properties of the system—it will process information by moving uphill toward goodness maxima. The particular maximum that the system will find is determined by where the system starts and by the distortions of the space induced by the input. One of the very important descriptors of a goodness landscape is the set of maxima that the system can find, the size of the region that feeds into each maximum, and the height of the maximum itself. The states themselves correspond to possible interpretations, the peaks in the space correspond to the best interpretations, the extent of the foothills or skirts surrounding a particular peak determines the likelihood of finding the peak, and the height of the peak corresponds to the degree to which the constraints of the network are actually met or alternatively to the goodness of the interpretation associated with the corresponding state.

Interactive Processing One of the difficult problems in cognitive science is to build systems that are capable of allowing a large number of knowledge sources to usefully interact in the solution of a problem. Thus in language processing we would want syntactic, phonological, semantic, and pragmatic knowledge sources all to interact in the construction of the meaning of an input. Reddy and his colleagues (1973) have had some success in the case of speech perception with the Hearsay system because they were working in the highly structured domain of language. Less structured domains have proved very difficult to organize. Connectionist models, conceptualized as constraint-satisfaction networks, are ideally suited for the blending of multiple-knowledge sources. Each knowledge type is simply another constraint, and the system will, in parallel, find those figurations of values that best satisfy all of the constraints from all of the knowledge sources. The uniformity of representation and the common currency of interaction (activation values) make connectionist systems especially powerful for this domain.

Rapid Pattern Matching, Best-Match Search, Content-Addressable Memory Rapid pattern matching, best-match search, and content-addressable memory are all variants on the general best-match problem (compare Minsky and Papert 1969). Best-match problems are especially difficult for serial computational algorithms (it involves exhaustive search), but as we have just indicated connectionist systems can readily be used to find the interpretation that best matches a set of constraints. It can similarly be used to find stored data that best match some target. In this case it is useful to imagine that the network consists of two classes of units, with one class, the *visible* units, corresponding to the content stored in the network, and the remaining, *hidden* units are used to help store the patterns. Each visible unit corresponds to the hypothesis that some particular feature was present in the stored pattern. Thus we think of the content of the stored data as consisting of collections of features. Each hidden unit corresponds to a hypothesis concerning the *configuration* of features present in a stored pattern. The hypothesis to which a particular hidden unit corresponds is determined by the exact *learning rule* used to store the input and the characteristics of the ensemble of stored patterns. Retrieval in such a network amounts to turning on some of the visible units (a retrieval probe) and letting the system settle to the best interpretation of the input. This is a kind of pattern completion. The details are not too important here because a variety of learning rules lead to networks with the following important properties:

- When a previously stored (that is, familiar) pattern enters the memory system, it is amplified, and the system responds with a stronger version of the input pattern. This is a kind of recognition response.
- When an unfamiliar pattern enters the memory system, it is dampened, and the activity of the memory system is shut down. This is a kind of unfamiliarity response.

- When part of a familiar pattern is presented, the system responds by "filling in" the missing parts. This is a kind of recall paradigm in which the part constitutes the retrieval cue, and the filling in is a kind of memory-reconstruction process. This is a content-addressable memory system.
- When a pattern similar to a stored pattern is presented, the system responds by distorting the input pattern toward the stored pattern. This is a kind of assimilation response in which similar inputs are assimilated to similar stored events.
- Finally, if a number of similar patterns have been stored, the system will respond strongly to the central tendency of the stored patterns, even though the central tendency itself was never stored. Thus this sort of memory system automatically responds to prototypes even when no prototype has been seen.

These properties correspond very closely to the characteristics of human memory and, I believe, are exactly the kind of properties we want in any theory of memory.

Automatic Generalization and Direct Representation of Similarity One of the major complaints against AI programs is their "fragility." The programs are usually very good at what they are programmed to do, but respond in unintelligent or odd ways when faced with novel situations. There seem to be at least two reasons for this fragility. In conventional symbol-processing systems similarity is indirectly represented and therefore are generally incapable of generalization, and most AI programs are not self-modifying and cannot adapt to their environment. In our connectionist systems on the other hand, the content is directly represented in the pattern and similar patterns have similar effects—therefore generalization is an automatic property of connectionist models. It should be noted that the degree of similarity between patterns is roughly given by the inner product of the vectors representing the patterns. Thus the dimensions of generalization are given by the dimensions of the representational space. Often this will lead to the right generalizations. There are situations in which this will lead to inappropriate generalizations. In such a case we must allow the system to *learn* its appropriate representation. In the next section I describe how the appropriate representation can be learned so that the correct generalizations are automatically made.

Learning A key advantage of the connectionist systems is the fact that simple yet powerful learning procedures can be defined that allow the systems to adapt to their environment. It was work on the learning aspect of these neurally inspired models that first led to an interest in them (compare Rosenblatt, 1962), and it was the demonstration that the learning procedures for complex networks could never be developed that contributed to the loss of interest (compare Minsky and Papert 1969). Although the *perceptron convergence procedure* and its variants have been around for some time, these learning procedures were limited to simple one-layer networks involving only input and output units. There were no hidden units in these cases and no internal representation. The coding provided by the external world had to suffice. Nevertheless these networks have proved useful in a wide variety of applications. Perhaps the essential character of such networks is that they map similar input patterns to similar output patterns. This is what allows these networks to make reasonable generalizations and perform reasonably on patterns that have never before been presented. The similarity of patterns in the connectionist system is determined by their overlap. The overlap in such networks is determined outside the learning system itself—by whatever produces the patterns.

The constraint that similar input patterns lead to similar outputs can lead to an inability of the system to learn certain mappings from input to output. Whenever the representation provided by the outside world is such that the similarity structure of the input and output patterns is very different, a network without internal representations (that is, a network without hidden units) will be unable to perform the necessary mappings. A classic example of this case is the exclusive-or (XOR) problem illustrated in table 4.1. Here we see that those patterns that overlap least are supposed to generate identical output values. This problem and many others like it cannot be performed by networks without hidden units with which to create their own internal representations of the input patterns. It is interesting to note that if the input patterns contained a third input taking the value 1 whenever the first two have value 1, as shown in table 4.2, a two-layer system would be able to solve the problem.

Table 4.1 XOR Problem

Input Patterns		Output Patterns
00	→	0
01	→	1
10	→	1
11	→	0

Table 4.2 XOR with Redundant Third Bit

Input Patterns		Output Patterns
000	→	0
010	→	1
100	→	1
111	→	0

Minsky and Papert (1969) have provided a careful analysis of conditions under which such systems are capable of carrying out the required mappings. They show that in many interesting cases networks of this kind are incapable of solving the problems. On the other hand, as Minsky and Papert also pointed out, if there is a layer of simple perceptronlike hidden units, as shown in figure 4.3, with which the original input pattern can be augmented, there is always a recoding (that is, an internal representation) of the input patterns in the hidden units in which the similarity of the patterns among the hidden units can support any required mapping from the input to the output units. Thus if we have the right connections from the input units to a large enough set of hidden units, we can always find a representation that will perform any mapping from input to output through these hidden units. In the case of the XOR problem, the addition of a feature that detects the conjunction of the input units changes the similarity structure of the patterns sufficiently to allow the solution to be learned. As illustrated in figure 4.4, this can be done with a single hidden unit. The numbers on the arrows represent the strengths of the connections among the units. The numbers written in the circles represent the thresholds of the units. The value of +1.5 for the threshold of the hidden unit ensures that it will be turned on only when both input units are on. The value 0.5 for the output unit ensures that it will turn on only when it receives

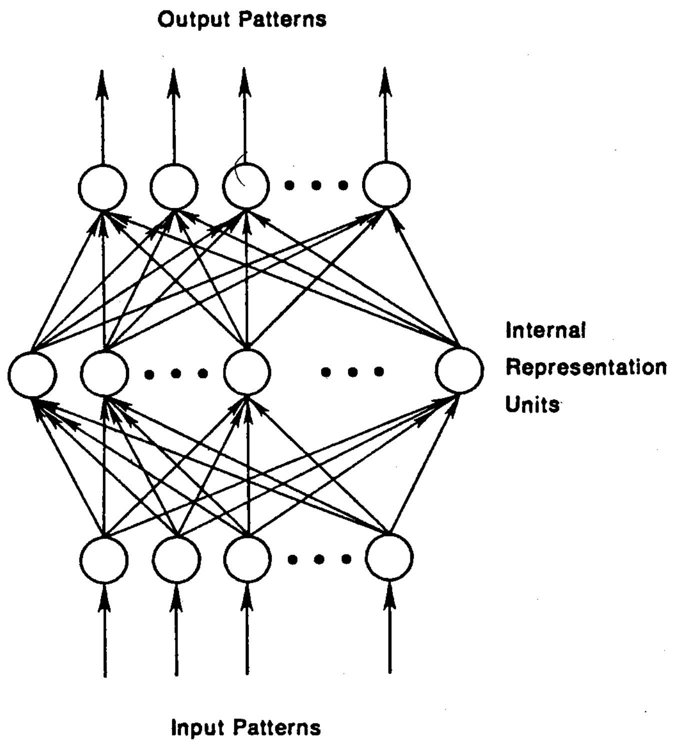


Figure 4.3 A multilayer network in which input patterns are *recoded* by internal representation units

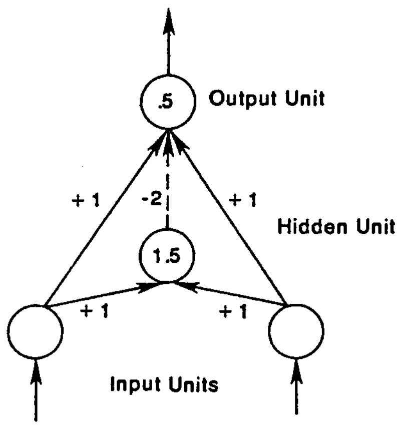


Figure 4.4 A simple XOR network with one hidden unit

a net positive input greater than 0.5. The weight of −2 from the hidden unit to the output unit ensures that the output unit will not come on when both input units are on. Note that from the point of view of the output unit the hidden unit is treated as simply another input unit. It is as if the input patterns consisted of three rather than two units.

The existence of networks such as this illustrates the potential power of hidden units and internal representations. The problem, as noted by Minsky and Papert, is that whereas there is a very simple guaranteed learning rule for all problems that can be solved without hidden units, namely, the perceptron convergence procedure (or the variation reported originally by Widrow and Hoff 1960), there has been no equally powerful rule for learning in multilayer networks.

It is clear that if we hope to use these connectionist networks for general computational purposes, we must have a learning scheme capable of learning its own internal representations. This is just what we (Rumelhart, Hinton, and Williams 1986) have done. We have developed a generalization of the perceptron learning procedure, called the *generalized delta rule,* which allows the system to learn to compute arbitrary functions. The constraints inherent in networks without self-modifying internal representations are no longer applicable. The basic learning procedure is a two-stage process. First, an input is applied to the network; then, after the system has processed for some time, certain units of the network are informed of the values they ought to have at this time. If they have attained the desired values, the weights are unchanged. If they differ from the target values, then the weights are changed according to the difference between the actual value the units have attained and the target for those units. This difference becomes an error signal. This error signal must then be sent back to those units that impinged on the output units. Each such unit receives an error measure that is equal to the error in all of the units to which it connects times the weight connecting it to the output unit. Then, based on the error, the weights into these "second-layer" units are modified, after which the error is passed back another layer. This process continues until the error signal reaches the input units or until it has been passed back for a fixed number of times. Then a new input pattern is presented and the process repeats. Although the procedure may sound difficult, it is actually quite simple and easy to implement within these nets. As shown in Rumelhart, Hinton, and Williams 1986, such a procedure will always change its weights in such a way as to reduce the difference between the actual output values and the desired output values. Moreover it can be shown that this system will work for any network whatsoever.

Minsky and Papert (1969, pp. 231–232), in their pessimistic discussion of perceptrons, discuss *multilayer machines*. They state that

> The perceptron has shown itself worthy of study despite (and even because of!) its severe limitations. It has many features that attract attention: its linearity; its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation. There is no reason to suppose that any of these virtues carry over to the many-layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgment that the extension is sterile. Perhaps some powerful convergence theorem will be discovered, or some profound reason for the failure to produce an interesting "learning theorem" for the multilayered machine will be found.

Although our learning results do not *guarantee* that we can find a solution for all solvable problems, our analyses and simulation results have shown that as a practical matter, this error propagation scheme leads to solutions in virtually every case. In short I believe that we have answered Minsky and Papert's challenge and *have* found a learning result sufficiently powerful to demonstrate that their pessimism about learning in multilayer machines was misplaced.

One way to view the procedure I have been describing is as a parallel computer that, having been shown the appropriate input/output exemplars specifying some function, programs itself to compute that function in general. Parallel computers are notoriously difficult to program. Here we have a mechanism whereby we do not actually have to know how to write the program to get the system to do it.

Graceful Degradation Finally connectionist models are interesting candidates for cognitive-science models because of their property of graceful degradation in the face of damage and information overload. The ability of our networks to learn leads to the promise of computers that can literally learn their way around faulty components because every unit participates in the storage of many patterns and because each pattern involves many different units, the loss of a few components will degrade the stored information, but will not lose it. Similarly such memories should not be conceptualized as having a certain fixed capacity. Rather there is simply more and more storage interference and blending of similar pieces of information as the memory is overloaded. This property of graceful degradation mimics the human response in many ways and is one of the reasons we find these models of human information processing plausible.

4.2 The State of the Art

Recent years have seen a virtual explosion of work in the connectionist area. This work has been singularly interdisciplinary, being carried out by psychologists, physicists, computer scientists, engineers, neuroscientists, and other cognitive scientists. A number of national and international conferences have been established and are being held each year. In such environment it is difficult to keep up with the rapidly developing field. Nevertheless a reading of recent papers indicates a few central themes to this activity. These themes include the study of learning and generalization (especially the use of the backpropagation learning procedure), applications to neuroscience, mathematical properties of networks—both in terms of learning and the question of the relationship among connectionist style computation and more conventional computational paradigms—and finally the development of an implementational base for physical realizations of connectionist computational devices, especially in the areas of optics and analog VLSI.

Although there are many other interesting and important developments, I conclude with a brief summary of the work with which I have been most involved over the past several years, namely, the study of learning and generalization within multilayer networks. Even this summary is necessarily selective, but it should give a sampling of much of the current work in the area.

Learning and Generalization

The backpropagation learning procedure has become possibly the single most popular method for training networks. The procedure has been used to train networks on problem domains including character recognition, speech recognition, sonar detection, mapping from spelling to sound, motor control, analysis of molecular structure, diagnosis of eye diseases, prediction of chaotic functions, playing backgammon, the parsing of simple sentences, and many, many more areas of application. Perhaps the major point of these examples is the enormous range of problems to which the backpropagation learning procedure can usefully be applied. In spite of the rather impressive breadth of topics and the success of some of these applications, there are a number of serious open problems. The theoretical issues of primary concern fall into three main areas: (1) The architecture problem—are there useful architectures beyond the standard three-layer network used in most of these areas that are appropriate for certain areas of application? (2) The scaling problem—how can we cut down on the substantial training time that seems to be involved for the more difficult and interesting problem application areas? (3) The generalization problem—how can we be certain that the network trained on a subset of the example set will generalize correctly to the entire set of exemplars?

Some Architecture

Although most applications have involved the simple three-layer back-propagation network with one input layer, one hidden layer, and one output layer of units, there have been a large number of interesting architectures proposed—each for the solution of some particular problem of interest. There are, for example, a number of "special" architectures that have been proposed for the modeling of such sequential phenomena as motor control. Perhaps the most important of these is the one proposed by Mike Jordan (1986) for producing sequences of phonemes. The basic structure of the network is illustrated in figure 4.5. It consists of four groups of units: *Plan units,* which tell the network which sequence it is producing, are fixed at the start of a sequence and are not changed. *Context units,* which keep track of where the system is in the sequence, receive input from the output units of the systems and from themselves, constituting a memory for the sequence produced thus far. *Hidden units* combine the information from the plan units with that from the context units to determine which output is to be produced next. *Output units* produce the desired output values. This basic structure, with numerous variations, has been used successfully in producing sequences of phonemes (Jordan 1986), sequences of movements (Jordan 1989), sequences of notes in a melody (Todd 1989), sequences of turns in a simulated ship (Miyata 1987), and for many other applications. An analogous network for *recognizing* sequences has been used by Elman (1988) for processing sentences one at a time, and another variation has been developed and studied by Mozer (1988). The architecture used by Elman is illustrated in figure 4.6. This network also

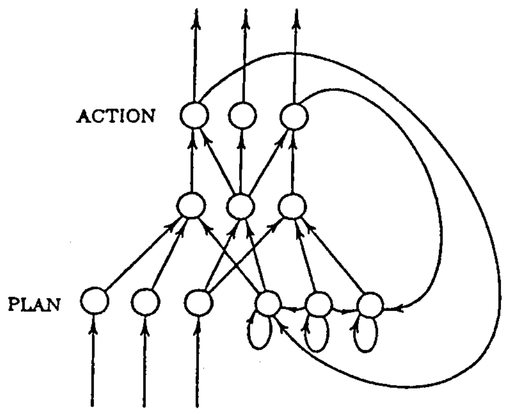


Figure 4.5 A recurrent network of the type developed by Jordan (1986) for learning to perform sequences

involves three sets of units: *input units,* in which the sequence to be recognized is presented one element at a time; a set of *context units* that receive inputs from and send inputs to the hidden units and thus constitute a memory for recent events; a set of *hidden units* that combine the current input with its memory of past inputs to either name the sequence, predict the next element of the sequence, or both.

Another kind of architecture that has received some attention has been suggested by Hinton and has been employed by Elman and Zipser (1987), Cottrell, Munro, and Zipser (1987), and many others. It has become part of the standard toolkit of backpropagation. This is the so-called method of autoencoding the pattern set. The basic architecture in this case consists of three layers of units as in the conventional case; however, the input and output layers are identical. The idea is to pass the input through a small number of hidden units and reproduce it over the output units. This requires the hidden units to do a kind of nonlinear-principle components analysis of the input patterns. In this case that corresponds to a kind of extraction of critical features. In many applications these features turn out to provide a useful compact description of the patterns. Many other architectures are being explored. The space of interesting and useful architecture is large and the exploration will continue for many years.

The Scaling Problem

The scaling problem has received somewhat less attention, although it has clearly emerged as a central problem with backpropagationlike learning procedures. The basic finding has been that difficult problems require many learning trials. For example, it is not unusual to require tens or even hundreds of thousands of pattern presentations to learn moderately difficult problems—that is, those whose solution requires tens of thousands to a few hundred thousand connections. Large and

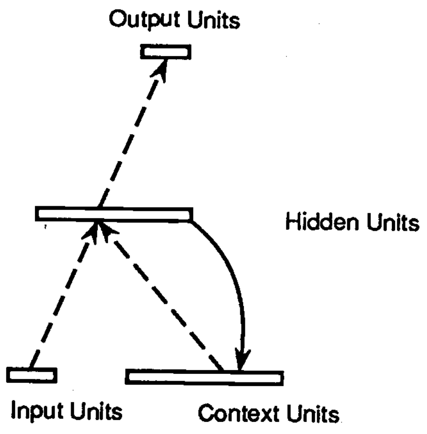


Figure 4.6 A recurrent network of the type employed by Elman (1988) for learning to recognize sequences

fast computers are required for such problems, and it is impractical for problems requiring more than a few hundred thousand connections. It is therefore a matter of concern to learn to speed up the learning so that it can learn more difficult problems in a more reasonable number of exposures. The proposed solutions fall into two basic categories. One line of attack is to improve the learning procedure either by optimizing the parameters dynamically (that is, change the learning rate systematically during learning) or by using more information in the weight-changing procedure (that is, the so-called second-order backpropagation in which the second derivatives are also computed). Although some improvements can be attained through the use of these methods, in certain problem domains the basic scaling problem still remains. It seems that the basic problem is that difficult problems require a large number of exemplars, however efficiently each exemplar is used. The other view grows from viewing *learning* and *evolution* as continuous with one another. On this view the fact that networks take a long time to learn is to be expected because we normally compare their behavior to organisms that have long evolutionary histories. On this view the solution is to start the system at places that are as appropriate as possible for the problem domain to be learned. Shepherd (1989) has argued that such an approach is critical for an appropriate understanding of the phenomena being modeled.

A final approach to the scale problem is through modularity. It is possible to break the problem into smaller subproblems and train subnetworks on these subproblems. Networks can then finally be assembled to solve the entire problem after all of the modules are trained. An advantage of the connectionist approach in this regard is that the original training needs to be only approximately right. A final round of training can be used to learn the interfaces among the modules.

The Generalization Problem

One final aspect of learning that has been looked at is the nature of generalization. It is clear that the most important aspect of networks is not that they learn a set of mappings but that they learn the function implicit in the exemplars under study in such a way that they respond properly to those cases not yet observed. Although there are many cases of successful generalization (compare the learning of spelling with phoneme mappings in Sejnowski and Rosenberg's Nettalk (1987), there are a number of cases in which the networks do not generalize correctly (compare Denker et al. 1987). One simple way to understand this is to note that for most problems there are enough degrees of freedom in the network that there are a large number of genuinely different solutions to the problems, and each solution constitutes a different way of generalizing to the unseen patterns. Clearly not all of these can be correct. I have proposed a hypothesis that shows some promise in promoting better generalization (Rumelhart 1988). The basic idea is this:

The problem of generalization is essentially the induction problem. Given a set of observations, what is the appropriate principle that applies to all cases? Note that the network at any point in time can be viewed as a specification of the inductive hypothesis. I have proposed that we follow a version of Occam's razor and select the *simplest, most robust* network that is consistent with the observations made. The assumption of robustness is simply an embodiment of a kind of continuity assumption that small variations in the input patterns should have little effect on the output and on the performance of the system. The simplicity assumption is simply—of all networks that correctly account for the input data—to choose that net with the fewest hidden units, fewest connections, most symmetries among the weights, and so on. I have formalized this procedure and modified the backpropagation learning procedure so that it prefers simple, robust networks and, all things being equal, will select those networks. In many cases it turns out that these are just the networks that do the best job generalizing.

References

Cottrell, G. W., Munro, P. W., and Zipser, D. 1987. Learning internal representations from grey-scale images: An example of extensional programming. In *Proceedings of the Ninth Annual Meeting of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Denker, J., Schwartz, D., Wittner, B., Solla, S., Hopfield, J., Howard, R., and Jackel, L. 1987. Automatic learning, rule extraction, and generalization. *Complex Systems* 1:877–922.

Elman, J. 1988. *Finding Structure in Time.* CRL Tech. Rep. 88-01, Center for Research in Language, University of California, San Diego.

Elman, J., and Zipser, D. 1987. *Learning the Hidden Structure of Speech.* Rep. no. 8701. Institute for Cognitive Science, University of California, San Diego.

Feldman, J. A. 1985. Connectionist models and their applications: Introduction. *Cognitive Science* 9:1–2.

Grossberg, S. 1976. Adaptive pattern classification and universal recoding: Part I. Parallel development and coding of neural feature detectors. *Biological Cybernetics* 23:121–134.

Hebb, D. O. 1949. *The Organization of Behavior.* New York: Wiley.

Hinton, G. E., and Sejnowski, T. 1986. Learning and relearning in Boltzmann machines. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.* Cambridge, MA: MIT Press, A Bradford Book.

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

Jordan, M. I. 1986. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Meeting of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Jordan, M. I. 1989. Supervised learning and systems with excess degrees of freedom. In D. Touretzky, G. Hinton, and T. Sejnowski, eds. *Connectionist Models.* San Mateo, CA: Morgan Kaufmann.

Minsky, M., and Papert, S. 1969. *Perceptrons.* Cambridge, MA: MIT Press.

Miyata, Y. 1987. *The Learning and Planning of Actions.* Ph.D. thesis, University of California, San Diego.

McClelland, J. L., Rumelhart D. E., and the PDP Research Group. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 2: Psychological and Biological Models.* Cambridge, MA: MIT Press, A Bradford Book.

Mozer, M. C. 1988. *A Focused Book-Propagation Algorithm for Temporal Pattern Recognition.* Rep. no. 88-3, Departments of Psychology and Computer Science, University of Toronto, Toronto, Ontario.

Reddy, D. R., Erman, L. D., Fennell, R. D., and Neely, R. B. 1973. The Hearsay speech understanding system: An example of the recognition process. In *Proceedings of the International Conference on Artificial Intelligence.* pp. 185–194.

Rosenblatt, F. 1962. *Principles of Neurodynamics.* New York: Spartan.

Rumelhart, D. E. 1988. *Generalization and the Learning of Minimal Networks by Backpropagation.* In preparation.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.* Cambridge, MA: MIT Press. A Bradford Book.

Rumelhart, D. E., McClelland, J. L., and the PDP Research Group 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.* Cambridge, MA: MIT Press. A Bradford Book.

Sejnowski, T., and Rosenberg, C. 1987. Parallel networks that learn to pronounce English text. *Complex Systems* 1:145–168.

Shepherd, R. N. 1989. Internal representation of universal regularities: A challenge for connectionism. In L. Nadel, L. A. Cooper, Calicover, P., and Harnish, R. M., eds. *Neural Connections, Mental Computation.* Cambridge, MA: MIT Press. A Bradford Book.

Smolensky, P. 1986. Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.* Cambridge, MA: MIT Press. A Bradford Book.

Todd, P. 1989. A sequential network design for musical applications. In D. Touretzky, G. Hinton, and T. Sejnowski, eds. *Connectionist Models.* San Mateo, CA: Morgan Kaufmann.

Widrow, G., and Hoff, M. E. 1960. Adaptive switching circuits. In *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part* 4. pp. 96–104.