1987

Connectionist Architectures for Artificial Intelligence

Scott E. Fahlman and Geoffrey E. Hinton
Carnegie-Mellon University

Massively parallel networks of simple neuron-like processing elements may hold the key to some important aspects of intelligence not captured by existing AI technology on serial machines.

Current AI technology can do a good job of emulating many of man's higher mental functions, but some of the most fundamental aspects of human intelligence have proven more elusive. AI can match the best human experts on certain narrow technical problems, but it cannot begin to approach the common sense and sensory abilities of a five-year-old child. Some important ingredients of intelligence seem to be missing, and our technology of symbolic representation and heuristic search, based on serial computers, does not seem to be closing the gap. Among the missing elements are the following:

- The human memory can store a huge quantity and variety of knowledge, and can find relevant items in this storehouse very quickly and without apparent effort. The phenomenon we call common sense is complex, but it derives in part from the ready availability of a large body of assorted knowledge about the world. Our serial machines can store large amounts of information, but it is very hard to make this knowledge an effective part of the machine's activities.
- In many domains, human recognition abilities far exceed what our machines can accomplish. Whether the domain is speech recognition, vision, or some higher-level task like medical diagnosis, the key operation seems to be an ability to locate, from among all known candidates, the one that best matches the sample to be identified. We humans can do this even with noisy, distorted input data and faulty expectations. For us, this is quick and seemingly effortless.
- In many cases, humans seem to handle information in some form other than the symbolic assertions of traditional AI. We can all recognize an elephant, but few of us can describe its appearance symbolically in an unambiguous way. We have difficulty coming up with formal symbolic descriptions for movements, shapes, sounds, and spatial relationships, and yet people work easily in all these domains. It can be argued that we humans use internal symbolic representations that we cannot access consciously, but it seems more plausible that some other kinds of representations are in use.

The traditional AI approach to knowledge and recognition problems is to use ever more complex and clever strategies to reduce the need for excessive search and computation. An alternative is to solve such problems with a less complicated but very cycle-intensive approach, using very large numbers (millions) of very simple processors to get the job done in a reasonable time. For example, one approach to interpreting visual input is to use clever reasoning to restrict the areas in which a computationally expensive edge-finder is applied. The alternative is to simply accept the cost of doing high-quality edge finding all over the image—even in places that will turn out not to be critical to later stages of interpretation.

A number of researchers have begun exploring the use of massively parallel architectures in an attempt to get around the limitations of conventional symbol processing. Many of these parallel architectures are *connectionist*: The system's collection of permanent knowledge is stored

0018-9162/87/0100-0100$01.00 © 1987 IEEE

as a pattern of connections or connection strengths among the processing elements, so the knowledge directly determines how the processing elements interact rather than sitting passively in a memory, waiting to be looked at by the CPU. Some connectionist schemes use formal, symbolic representations, while others use more analog approaches. Some even develop their own internal representations after seeing examples of the patterns they are to recognize or the relationships they are to store.

Connectionism is somewhat controversial in the AI community. It is new, still unproven in large-scale practical applications, and very different in style from the traditional AI approach. We have only begun to explore the behavior and potential of connectionist networks. In this article, we describe some of the central issues and ideas of connectionism, and also some of the unsolved problems facing this approach. Part of the motivation for connectionist research is the possible similarity in function between connectionist networks and the neural networks of the human cortex, but we concentrate here on connectionism's potential as a practical technology for building intelligent systems.

What is connectionism?

Jerry Feldman coined the term "connectionism" to refer to the study of a certain class of massively parallel architectures for artificial intelligence. A connectionist system uses a large number of simple processing elements or *units*, each connected to some number of other units in the system. The units have little information stored internally, typically only a few *marker bits* or a single scalar *activity-level*, used as a sort of short-term working memory. The long-term storage of information is accomplished by altering the pattern of interconnections among the units, or by modifying a quantity called the *weight* associated with each connection. This use of connections, rather than memory cells, as the principal means of storing information motivated the name connectionism.

The parallel processing units in a connectionist network do not follow individual programs. The units are capable of only a few simple actions such as accepting incoming signals, performing some Boolean or arithmetic processing on the data, and sending signals out over some or all of the connections. These operations may be completely autonomous—part of a unit's built-in behavior—or they may be controlled by commands broadcast by some external controller, perhaps a serial computer of the traditional kind.

Since all of the connections can carry signals simultaneously, and all of the processing elements can act in parallel to integrate their arriving data, a connectionist system can bring a large amount of knowledge to bear simultaneously when making a decision, and can weigh many choices at once.

In some connectionist systems, the parallelism is used to implement a sort of simultaneous brute-force search through units, each representing a single item in the knowledge base. In other systems, the parallelism is used to allow richer representations; the pattern of activity in a large group of units represents an item, and different items are represented by alternative patterns of activity. Given some inputs and an initial state of the network, one of these patterns will emerge. Many connections, representing many small pieces of knowledge, will play a role simultaneously in determining which alternative will win. Whichever strategy is used, this ability to bring a lot of knowledge into the game at once is a major reason for the growing interest in connectionism.

The interunit signals sent through the connections are typically single-bit markers or continuous scalar values. Thus, we speak of *marker-passing* or *value-passing* parallelism. A unit can receive many of these signals at once, each arriving over a different connection. These multiple signals are combined upon receipt: Multiple instances of the same marker bit are simply OR'ed into one, and multiple scalar values are usually just combined into a weighted sum as they arrive at the destination unit. Systems that send more complex symbolic messages from unit to unit, the so-called *message-passing* parallel systems, are not usually considered connectionist systems because they require much more complex processing units with a considerable amount of storage for the messages.

An important difference between the connectionist approach and the more conventional "modestly parallel" architectures for AI is that the connectionists are willing to assign a processing element to each tiny subtask (one element for every item of knowledge in the system, for example) and to postulate that there are enough of these simple processing elements to handle the task at hand. The more conventional approach assumes a fixed number of larger processors (typically between 2 and 1024) and tries to find ways of cutting the problem up into that many pieces, all of which can be worked on concurrently.

In a serial system, the time required to sift through a finite set of items in memory or to consider a finite set of hypotheses in a recognition task grows linearly with the size of the set. The modestly parallel approach attempts to approach N-fold speedup from N processors. The connectionist thinks in terms of performing these simple tasks in constant time, while the amount of hardware grows linearly with the number of memory items or the number of hypotheses to be considered at once. This view may seem less radical if we think of a connectionist unit not as a CPU but as a fancy kind of memory cell that stores knowledge in its connectivity or connection weights. We need enough of these memory cells to hold the system's knowledge.

Many kinds of connectionist architectures are being investigated by the small but growing community of researchers interested in this kind of parallelism. It is hard to say much more about these systems as a class, and in an article of this length it is impossible to mention all of the connectionist research going on. We will describe some of the key issues and ideas that seem important to us, and mention one or two pieces of work exemplifying each of these key ideas.

Distributed representations

The simplest way to represent things in a massively parallel network is to use *local representations*, in which each concept or feature is represented by a specific piece of hardware. For example, when the system wants to work with the concept of "elephant," it turns on the elephant unit. This kind of representation is easy to create and easy to understand. Unfortunately, if the elephant unit breaks, the system loses all of the knowledge tied to it. This creates some obvious reliability problems: In a system with millions of processing elements, not all of them will be working all the time. It also means that most of the units will be idle most of the time.

Many neuroscientists believe that the brain does not work this way, that instead it uses some sort of *distributed representa-*

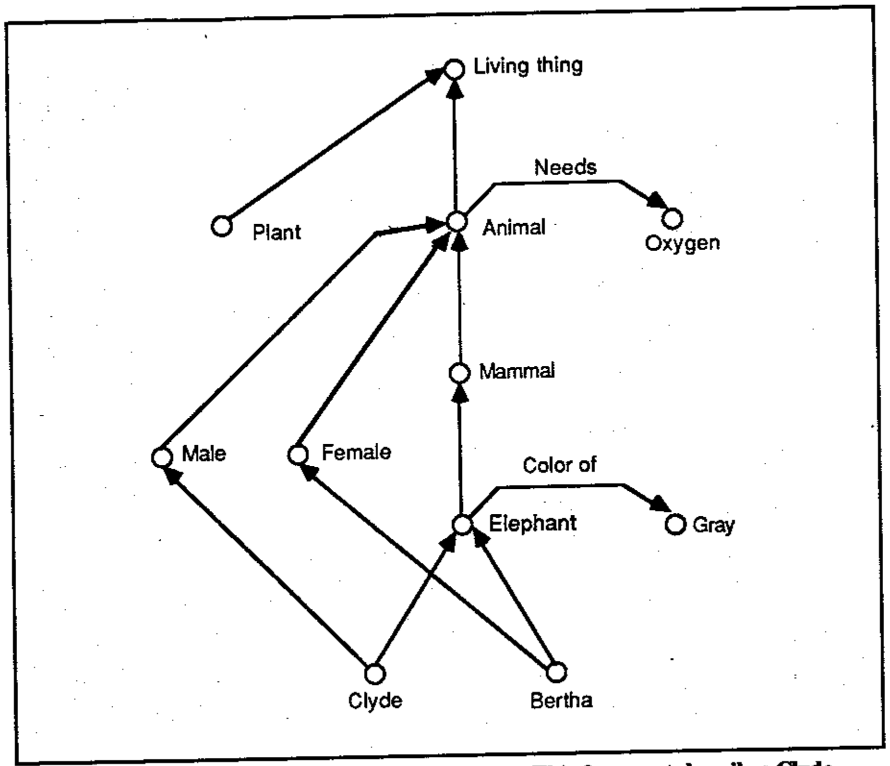

Figure 1. A portion of a NETL semantic network. This fragment describes Clyde the elephant and related information. (Source: *Artificial Intelligence,* Vol. 1, 1985. Used by the courtesy of MIT Press, publisher.)

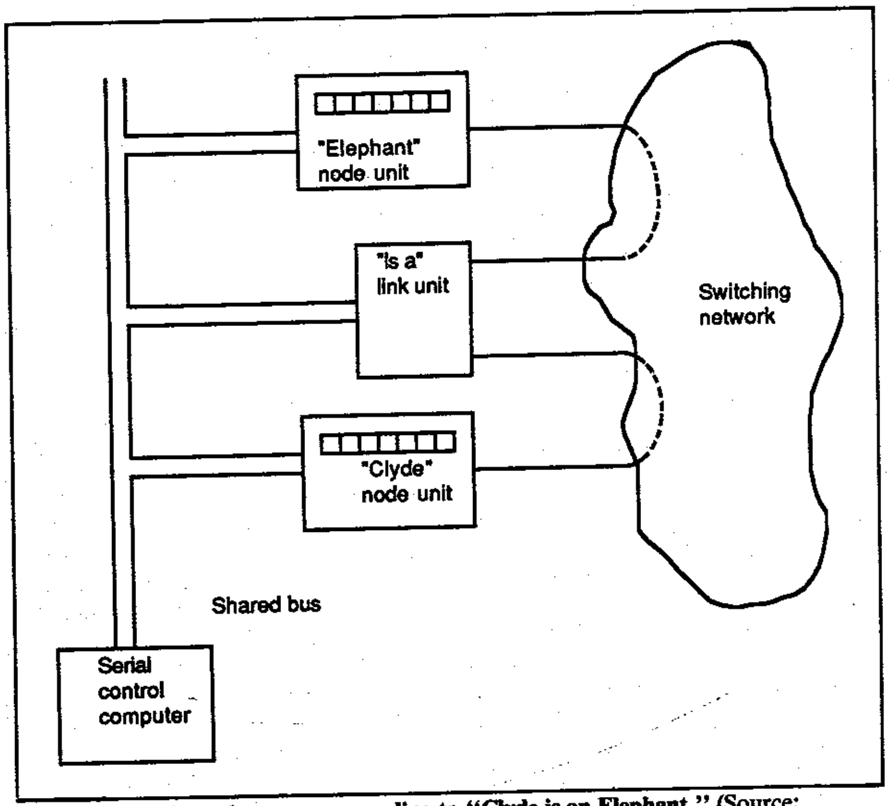

Figure 2. NETL hardware corresponding to "Clyde is an Elephant." (Source: *Artificial Intelligence,* Vol. 1, 1985. Used by the courtesy of MIT Press, publisher.)

tion: A concept like "elephant" is represented not by a single neuron, but by a pattern of activation over a large number of neurons. They often use the analogy of a hologram, in which each point of the image is constructed from information from all over the film; if some part of the hologram is destroyed, the total image is degraded slightly, but no part of the image is completely lost. A distributed representation in a massively parallel network (whether built from neurons or silicon) would have a similar kind of reliability: If a few of the units malfunction, the resulting pattern is imperfect but still usable. Each macroscopically important behavior of the network is implemented by many different microscopic units, so any small random subset can disappear without changing the macroscopic description of the network's behavior.

This kind of inherent fault-tolerance has important implications for the construction of large-scale parallel networks. For example, wafer-scale integration becomes more feasible, since a few malfunctioning units in the wafer would simply be ignored. It is almost certainly easier to build a billion-transistor system, in which only 95 percent of the circuit elements have to work, than to build a million-transistor system that has to be perfect. It can also be shown[1] that in many situations it is much more efficient, in terms of the number of units required, to represent a value using a *coarse-coded* distributed scheme than to assign a single unit to represent each small interval in the range. In the coarse-coded scheme, each unit is coarsely tuned to cover a considerable range of values, and a particular value is represented by activity in a number of overlapping units.

The main disadvantage of distributed representations is that they are hard for an outside observer to understand or to modify. To add a single new piece of macroscopic knowledge to a network that uses distributed representations, it is necessary to change the interactions between many microscopic units slightly so that their joint effects implement the new knowledge. For any problem of significant size, it is nearly impossible to do this by hand, so it follows that a network using a distributed representation must employ some sort of automatic learning scheme. In the absence of an automatic learning procedure that works efficiently in large networks, distributed representations are generally too awkward to be of much practical value.

NETL: A connectionist system for symbolic knowledge

The NETL system[2] is an example of a connectionist architecture that uses local representations. It was designed to store and access a large number of symbolic assertions, and to perform certain simple searches and deductions within this collection of knowledge. NETL is an implementation in hardware of a *semantic network*, a graph-like data structure in which the nodes represent noun-like concepts and the links represent relationships between these concepts. Such a network is illustrated in Figure 1; the NETL hardware corresponding to a part of this network is illustrated in Figure 2.

In the NETL system, each of the nodes in the network is represented by a simple processing unit, capable of storing a few single-bit markers and of performing simple Boolean operations on the markers. Each link is also a simple processing unit, wired up to two or more node units. Link units, too, can perform simple Boolean operations, which generally amount to passing a specific marker from one of the attached nodes to another. All of the nodes and links in the network can perform these operations simultaneously in response to commands broadcast by the system controller, a serial machine of the familiar sort.

Every time a new assertion is added to the system, new nodes and links must be wired into the network to represent that assertion. Since all the node-to-link connections must be capable of carrying signals simultaneously, they must be true private-line connections and not just addresses sent over a shared party-line bus. Of course, in a practical implementation, these new connections would be established in a switching system resembling a telephone exchange and not by stringing tiny wires from unit to unit.

A NETL network can perform searches and simple inferences that go beyond what can be done with a simple associative memory. For example, a very heavily used operation in most AI knowledge-based systems is *inheritance*: We want anything we know about the typical elephant to apply to the subclasses and individuals below elephant in the hierarchy of is-a links (unless that inherited information is specifically canceled for some individual). Because of branching in the is-a hierarchy, an individual node like clyde might inherit information from a large number of superior nodes, so we need some very fast way of scanning a large part of the network.

In NETL, inheritance is handled by marker propagation. If the problem is to find the color of Clyde the elephant, the controller sets marker 1 on the clyde node, then repeatedly orders any is-a link with marker 1 set on the node below it to pass it on to the node above. When the network settles, we have marker 1 on all of the nodes representing superior classes from which clyde is supposed to inherit properties. We then command every color-of link with marker 1 on its tail to put marker 2 on its nose. Finally, we ask every node marked with marker 2 to report its identity to the controller. In this case, we get a single winner, the gray node. In a few cycles, we have followed a chain of inference to locate this information, regardless of where in the network the color information may have been attached. (The actual sequence of operations is slightly more complex than this, due to the possible presence of exceptions in the network, but the idea is the same.)

Because of its ability to do set intersections in parallel, NETL can handle recognition tasks of a certain limited kind. Suppose we are confronted with a gray four-legged mammal and we want to quickly locate any stored description that exhibits the intersection of these features. By marker-propagation from the feature nodes, we can mark all of the gray things with one marker, four-legged things with a second marker, and mammals with a third. Then we simply broadcast a command that any unit with all of these markers should queue up to report its identity to the controller. The intersection step takes only a small, constant time, regardless of the size of the sets being intersected.

This approach to recognition is very strong in some ways, but weak in others. On the positive side, NETL examines all of the descriptions in memory at once, not depending on heuristics that might prematurely rule out the right answer. Any set of observed features that is sufficient to select one of the stored descriptions will suffice, and if not enough features are present, the system will still narrow the set of possibilities down as much as possible. When new descriptions are added to the network, the knowledge immediately becomes an effective part of the recognition process. We do not need to hire a programmer or knowledge engineer to specify exactly how and when each piece of knowledge is to be applied. On the negative side, NETL treats every feature as an atomic entity that is either present or absent and expects that the winning description will explain all of the observed features. Like traditional symbolic AI systems, NETL works best in clean domains, far from the noise and confusion of the low-level sensory inputs.

The NETL architecture has been simulated, but not yet implemented directly. The node and link units are so simple that several thousand of them could fit on a single chip. The hard problem is to design the switching network that implements the node-to-link interconnections. A design study by Fahlman[3] demonstrated that a NETL machine can be built for only a few times the cost of a conventional memory system able to hold the same amount of information. The Connection Machine[4] was designed in part to implement a NETL-like knowledge base, though the hardware is general enough to do other jobs as well. The current 64,000-processor version of this machine is large enough to handle a substantial body of knowledge, by the standards of current AI systems. The proposed million-processor version may be able to handle enough assorted knowledge to exhibit some degree of common sense.

Layered value-passing networks for recognition

As we mentioned earlier, a marker-passing system like NETL does not gracefully handle the messy recognition problems we encounter in the real world. Whether we are dealing with vision, speech understanding, medical diagnosis, or some other real-world recognition problem, we normally face a situation where no match is ever perfect and in which few of the incoming features are noise-free and certain. Some features provide strong evidence for particular hypotheses; other features are only suggestive. Some features are clearly present; others are borderline cases or the result of low-confidence observations. Instead of the all-or-none set operations of NETL, we need a system that can combine many observations of varying quality and find hypotheses that fit well, even if they do not match perfectly.

This kind of problem can be handled by a *value-passing* system, in which each connection has an associated scalar weight. Each unit computes a weighted sum of the incoming values and passes this sum through a nonlinear function, with the

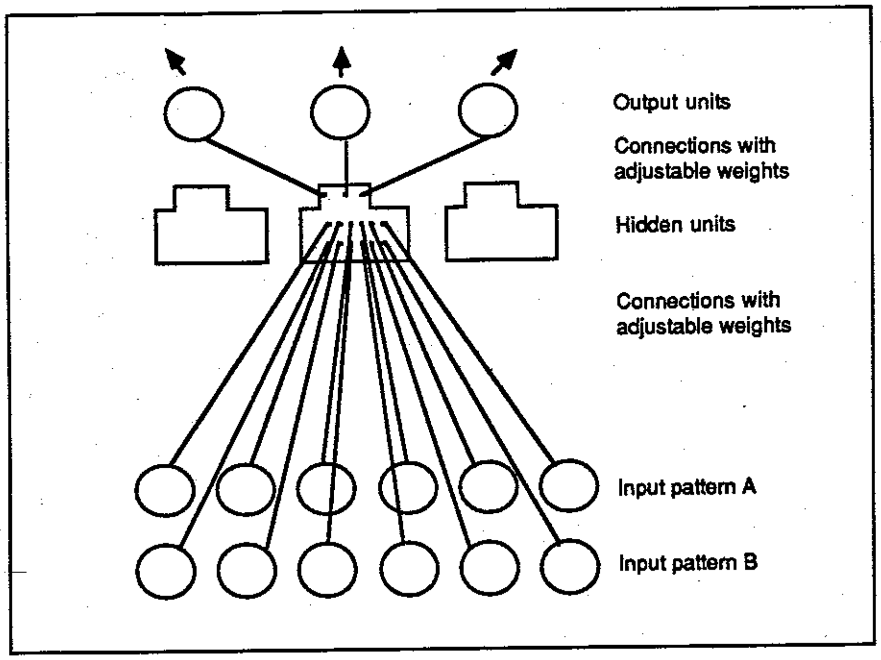
Output units
Connections with adjustable weights
Hidden units
Connections with adjustable weights
Input pattern A
Input pattern B

Figure 3. Value-passing network to compare two six-bit patterns. Input patterns A and B are identical, except that pattern A may be rotated one position to the left or right from pattern B. The three output units represent a left rotation, no rotation, and a right rotation, respectively. This task cannot be done without some hidden units. In this network we employ 12 hidden units, though only three appear in the diagram. An appropriate set of weights can be learned using the back-propagation method.

resulting value becoming the unit's output. A value-passing network may be implemented in either digital or analog hardware, but computationally it can be viewed as a sort of analog computer.

Using these value-passing units, we can set up a layered recognition system able to handle uncertain observations and varying degrees of evidence. Each directly observable feature is represented by an *input unit* whose value represents the probability that a discrete feature is present, the magnitude of a continuous quantity (like intensity), or the probability that a magnitude lies within a particular interval. Each of the possible hypotheses that we wish to evaluate is represented by an *output unit.*

In the simplest of these networks, the input units are connected directly to the output units by a set of connections with modifiable weights. In such networks, it is not necessary to set the connection weights by hand. Given a set of input vectors and the desired output for each, the perceptron convergence procedure[5] can be used to find a set of weights that will perform this mapping, if such a set exists.

Unfortunately, for most interesting recognition tasks there is no set of weights in a simple two-layer network that will do the job. In most cases, one cannot treat the lowest-level features as independent sources of evidence and simply add them up. It is necessary to introduce one or more intervening layers of hidden units, which combine the raw observations into higher-order features more useful in determining the output. Consider, for example, the problem of deciding whether an image contains a telephone. The fact that a single pixel in the image has high intensity cannot be used directly as evidence for or against a telephone, because telephones can be black or white. It is necessary to extract relationships between pixel intensities (such as edges) before trying to detect the telephone.

Figure 3 shows a simple task for which hidden units are essential. Pattern A is identical to pattern B, except that it may be rotated one step to the left or right. The task is to determine the shift for any pattern of input bits. This cannot be done without hidden units because each input bit, considered in isolation, provides no evidence whatsoever about the shift. All the information is in the joint behavior of combinations of input bits, so one or more layers of hidden units must be used to extract informative multi-unit combinations, which then can be combined into an overall answer.

Learning which of the exponentially many possible combinations are relevant for predicting the output is a hard problem. Techniques such as back-propagation (described below) can indeed discover a set of weights and hidden-unit assignments that, taken together, will solve the problem. Figure 4 shows some examples of hidden units and associated weights taken from one such solution.

Minsky and Papert[6] thoroughly analyzed which tasks require hidden units and demonstrated that an insightful way of categorizing tasks is in terms of how many input units must connect to each hidden unit in a three-layer network (this determines the order of the statistics that can be extracted). Unfortunately, they gave no procedure for learning appropriate hidden units, and suggested that there may not be any simple general procedure.

For many years after Minsky and Papert's book was published, AI researchers criticized the connectionist approach because the effective learning procedures were restricted to networks incapable of performing tasks like figure-ground segmentation or viewpoint-independent shape recognition. More recently, a deeper appreciation of the sheer quantity of computation required and a better understanding of the nature of the computations has led to renewed interest in massively parallel approaches to perception.[7]

Learning representations

Three-layered nets of the kind studied by Minsky and Papert do not have the freedom to choose their own representations because the weights between the input units and the feature detectors in the middle layer are predetermined and do not learn. If we allow these weights to learn, the network can decide for itself what the units in the middle layer will represent. Consequently, we get a much more interesting and powerful kind of learning, especially when extended to multiple hidden layers. However, finding learning procedures that choose good representations is difficult, and because the space of possible representations is very large, any procedure that explores the space in a random way will be very slow.

One promising new procedure, called back-propagation, has been discovered in-

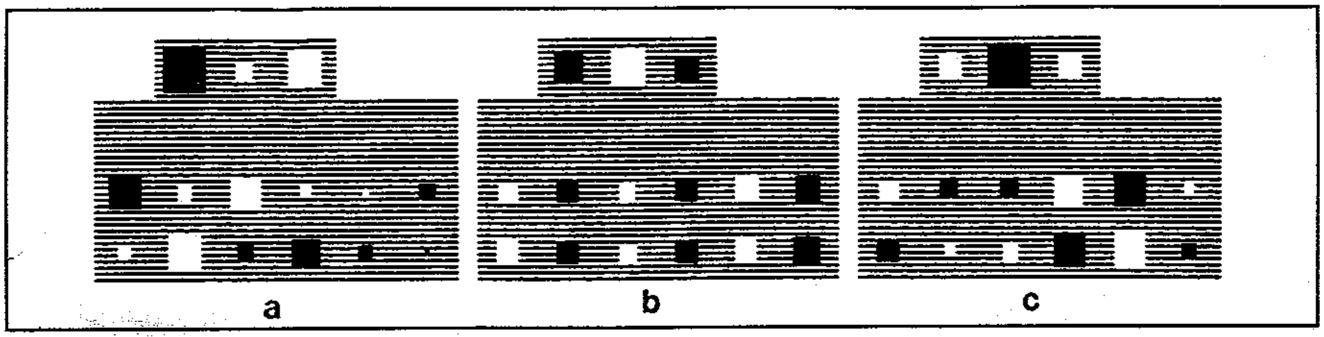

Figure 4. The weights learned by three of the twelve hidden units. The white squares are positive weights, the black squares are negative weights, and the area of a square is the magnitude of the weight. The bottom two rows in each hidden unit show the weights coming from the input units, and the top row shows the weights to the three output units that represent the three possible shifts. All the units shown learned positive thresholds (not shown here) so that they will not come on unless they receive a net positive input. Unit a detects some cases of right-shift; the two large negative weights prevent it from responding to left shifts. Unit b detects some no-shift cases. Unit c responds positively to shifts in either direction, but not to no-shift cases.

dependently by David Rumelhart, David Parker, and Yann Le Cun. It involves two passes each time an input vector is presented. In the forward pass, activity starts at the input units and passes through the layers to produce an output vector. In the backward pass, the derivative of the error (the difference between the actual output vector and the desired output vector) is back-propagated through the same connections but in the reverse direction. This allows the network to compute, for each weight, the gradient of the output error with respect to that weight. The weight is then changed in the direction that reduces the error. So learning works by gradient descent on an error surface in weight space.

Back-propagation has already been shown capable of learning many different kinds of interesting representation in the hidden units. It can learn optimal codes for squeezing information through narrow bandwidth channels, or sets of optimal filters for discriminating between very similar noisy signals.

Sejnowski and Rosenberg[8] showed that a back-propagation net can be trained to transform an input vector that encodes a sequence of letters into an output vector that encodes the phonemic features of the sequence; that phonemic output can then be used to drive a speech-synthesis device. From the input-output examples it sees during training, this system is able to extract both the regularities exhibited by the mapping, such as the effect of a terminal "e" in changing the sound of the preceding vowel, and specific exceptions, such as the odd pronunciation of a word like "women." This same task is performed in commercial speech-generation systems by conventional programs of considerable size and complexity.

Back-propagation can also be used to learn the semantic constraints that underlie a set of facts. One five-layer network[9] was trained on a set of 100 triples such as (Victoria has-father Christopher) or (Christopher has-wife Penelope) derived from two family trees involving people of two nationalities. The input vector represented the first two terms of a triple and the required output vector represented the third term. As far as the network was concerned, these input vectors were just arbitrary symbols, but after extensive training the network could generalize appropriately to triples on which it had not been trained, like (Victoria has-mother ??). By recording the set of triples for which each of the hidden units became active, it was possible to show that these units had learned to represent important properties like "Italian" or "old" that were never mentioned in the input and output.

Back-propagation does gradient descent in the space of possible representational schemes, and these semantic features happen to be a very good representation for the relationships the network is trained on. The hidden features and their interactions encode the underlying regularities of the domain. When viewed locally, the learning procedure just tunes the weight parameters, but the global effect is that the network does structural learning, creating new terms that allow it to express important regularities. This example also illuminates the "local versus distributed" issue. The network's internal representation of each person is a distributed pattern of active semantic features, but each of these local features captures an important underlying regularity in the domain.

Back-propagation has similarities to the Baum-Welch algorithm for tuning parameters in a stochastic, finite-state automaton. These trainable automata are widely used as generative models in practical speech-recognition systems[10] and there is some hope that procedures like back-propagation will be able to improve on these models by overcoming an important limitation. In a finite-state automaton, the system's knowledge of what it has produced so far is encoded as the current node in a graph of states and transitions. So if the automaton needs to remember 20 bits of information about the partial sequence to constrain the rest of the sequence, it requires at least 2^{20} nodes. In a connectionist network, many units can be active at once, so the number of units need not grow exponentially with the amount of information that must be carried along in generating a sequence.

Another interesting new learning procedure, called $A_{R\text{-}P}$, was described by Andy Barto.[11] He showed how layered networks of relatively simple stochastic units can learn to cooperate in order to maximize a global payoff signal that depends on the output vector and is received by every unit. Like back-propagation, this procedure learns interesting internal representations, but it is slower because it discovers the effect of changing a weight by sampling the effects of random variation instead of by explicit computation of the gradient. The $A_{R\text{-}P}$ model is exciting as a biological model because it does not need a separate pass in which to back-propagate detailed error information.

A major criticism of all the current multilayer learning procedures is that they are slow even for modest examples and they appear to scale poorly. The obvious problem of gradient descent—that it will get stuck in a poor local minimum—turns out to be only a minor problem in practice. The real difficulty is that simple gradient descent is very slow because we have information about only one point and no clear picture of how the surface may curve. In high-dimensional spaces the error surface usually contains ravines that are highly curved across the ravine and slope gently downwards along the ravine. Small steps take forever to meander down the ravine, and big steps cause divergent oscillations across the ravine. Even with a large speedup from faster implementation technology, these learning techniques are too slow to handle many problems of interest. Future progress may well depend on discovering ways of partitioning large networks into relatively small modules that can learn more or less independently of each other.

Constraint-satisfaction in iterative networks

The idea that perceptual interpretation consists of transforming an input vector through successive layers of units until it becomes a categorization is rather restrictive. Most real recognition tasks require that multiple layers of features be recognized all at once and that the result of perception be a coherent, articulated structure rather than a single category. In visual recognition, for example, we might have a scene, various objects in the scene, their parts, and subparts, all constraining one another and all relating in complex ways to the low-level stimuli the system is receiving. An African context provides evidence in favor of seeing an elephant. An elephant helps us to identify an African scene, and that in turn can help us to spot the giraffe. The trunk, tusks, and ears all are features that help us to recognize an elephant. Likewise, knowing that we're looking at an elephant helps us to recognize the elephant's parts.

In a situation like this, we want to start with a network that records the tangle of interlevel constraints and evidence relationships, put in our observations as a sort of boundary condition, and get the system to settle into the best possible solution (or set of solutions) spanning all the levels of description. Some constraints and expectations will be violated, but we want to find the best-scoring constellation of decisions overall.[7]

The task is similar to finding the solutions to a set of simultaneous equations. A value-passing network would seem a sensible parallel way of converging on a good solution—a sort of analog computer with links representing the constraints. Non-learning-constraint networks of this sort have been studied for a variety of problem domains, most notably at the University of Rochester.[12] In some simple cases, the network can be guaranteed to settle to the best final state no matter what values it starts with, but for more difficult problems, especially ones that involve discrete decisions, a value-passing network with loops and many nonlinear decision elements is generally not well-behaved. Such networks tend to oscillate or to get trapped in uninteresting states that do not represent good solutions to the problem.

Hopfield and Boltzmann networks for constraint satisfaction

One way to guarantee that a network will settle down is to show that there exists some cost function that decreases every time one of the values changes. Hummel and Zucker[13] showed that there exists such a function for networks that pass values and have symmetrical connections. At about the same time, Hopfield[14] identified a cost function (which he called "energy") for networks of symmetrically connected binary threshold units.

Hopfield's energy function can be interpreted in the following way: A connection between units *i* and *j* with a positive weight represents a constraint that if one of these units is on, the other one should be also. A negative connection weight says that if one of the units is on, the other should not be. The weight corresponds to the penalty to be applied if the constraint is violated. The energy of any state of the network is given by

$$E = -\sum_{i<j} s_i s_j w_{ij} + \sum_i s_i \Theta_i$$

where w_{ij} is the weight on the connection between units *i* and *j*, s_i is 1 if unit *i* is on and 0 otherwise, and Θ_i is a threshold for unit *i*.

Given Hopfield's quadratic definition of energy, each unit *i* can determine the difference between the global energy of the network when it is off and the global energy when it is on, given the current states of the other units. This energy gap is simply

$$E_{i\,off} - E_{i\,on} = \Delta E_i = \sum_j s_j w_{ij}$$

If the energy gap is positive, the unit should turn on (or stay on) to minimize the global energy. Otherwise it should turn off (or stay off). In other words, to minimize energy it should behave exactly like a binary threshold unit.

Hopfield originally proposed his model as a theory of memory. Each local minimum corresponds to a stored vector, and the memory is content-addressable because if it is started anywhere near one of the stored states it normally converges on that state. The same kind of network can also be used for perceptual interpretation by defining some of the units as input units and clamping their states on or off to represent the current perceptual input. The other units then settle into a low-energy global state consistent with these boundary conditions. This state represents a locally-optimal solution given this set of inputs, but with no guarantee that it is the global optimum.

The Boltzmann machine architecture[15,16] is essentially a Hopfield network (with hidden units) that uses a *simulated annealing* search to escape from local minima. This same general idea, with some differences of emphasis, was independently proposed by other research groups[17,18] at about the same time.

Simulated annealing is a search technique that has been applied to a number of optimization problems.[19] The idea is that we escape from high local minima by adding a random component to the decision process of each unit. In most cases, the unit still takes a step downhill, but occasionally it will take a step uphill instead. More precisely, each unit *i* computes the ΔE_i value as above, then assumes the 1 state with probability p_i given by the following formula:

$$p_i = \frac{1}{1 + e^{-\Delta E_i / T}}$$

The *T* term is a scaling factor that controls the amount of noise. It is analogous to a temperature: For large *T*, *p* is about .5 and the system assumes states at random, ignoring the constraints in the network; for *T* = *0*, the random element is eliminated, and the system behaves as in the pure Hopfield net, moving downhill into the nearest local minimum. At any given

T, once the system has reached thermal equilibrium, the relative probability of being in state A versus state B (see Figure 5) obeys a Boltzmann distribution:

$$\frac{P_A}{P_B} = e^{-(E_A - E_B)/T}$$

Thermal equilibrium does not mean that the system has settled into a particular stable state. It means that the probability distribution over states has settled down, even though the states are still changing. A deck of cards is at thermal equilibrium when it has been shuffled for long enough that the probability of finding any card in any position is 1/52, even though the cards keep moving around. The best way to reach thermal equilibrium at a given temperature is generally to start at a higher temperature, which makes it easy for the system to cross energy barriers but gives it little preference for the lower energy states. Then we gradually increase the preference for low energy states by reducing *T*. If *T* is reduced slowly enough, there is a high probability of ending up in the best global state, or if not there, in a state not much worse than the best. This process of slow cooling is analogous to slow annealing of a metal in order to crystallize it in its lowest energy state.

Boltzmann networks, designed by hand and running on a serial simulator, have been demonstrated successfully on a number of constraint-satisfaction problems, such as the separation of figure and ground in a two-dimensional image. Touretzky[20] is investigating ways of implementing symbolic processing, including a production system interpreter and a system that can manipulate lists and tree structures, using Boltzmann-type networks. But most of the interest in Boltzmann machines is due to a learning procedure, described later, that enables them to program constraints into their connections when shown examples of good solutions to a problem.

Hopfield and Tank[21] investigated a different method to avoid being trapped in local minima: They used real-valued, analog, nonlinear elements that obey the same equation as the units in a Boltzmann machine but send out a deterministic real value instead of a single stochastic bit. Reducing the temperature in a Boltzmann machine is equivalent to raising the gain (the nonlinearity) of their analog units, and they have shown that they can approximate simulated annealing without actually simulating a stochastic system directly.

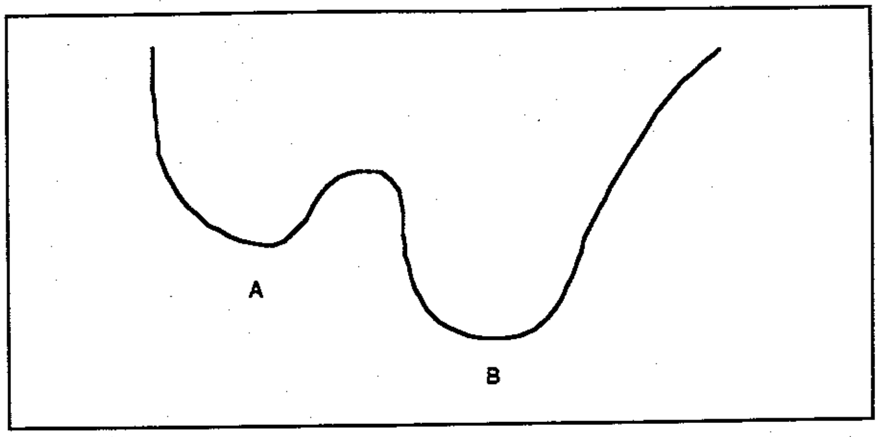

Figure 5. Energy surface with local minimum at A, global minimum at B.

As the gain is increased, single minima of the low-gain energy surface develop a fine structure of smaller minima.

Hopfield and Tank's approximation loses all information about the higher-order statistics. For example, they would represent a stochastic system that spent half its time in the state (1, 1) and half its time in the state (0, 0) by the mean values (0.5, 0.5). But the very same representation would be used for a system that spends half its time in the state (1, 0) and half in the state (0, 1). It is not yet clear how important this loss of higher-order statistical information is.

Networks of this type have been applied to a variety of problems, the most ambitious of which is the Traveling Salesman Problem, for which the network finds moderately good solutions very fast. Hopfield's group has shown a particular enthusiasm for implementing their networks in silicon, rather than simulating them, and a number of small-scale physical implementations have appeared. At the same time, Hopfield has been working with Tank to investigate the possible connection between these networks and real neurons.

Hopfield's group has chosen not to focus on the problem of learning what to do with extra, hidden, units whose required behavior is not specified by the task definition. These are not needed if the experimenter decides how to represent things, but they are crucial if the network is to be given the freedom to develop its own representations. There exists an iterative form of the back-propagation learning procedure that can learn how to use hidden units in a Hopfield and Tank network and can also learn an optimal schedule for varying the gain of the units during the search process. If the same learning procedure is applied to nets with asymmetric connections, they can be trained to produce sequences.[22] The learning procedure is infeasibly slow for large nets, so the research currently divides into studies of search in larger nets, built by hand, and learning in smaller nets.

The Boltzmann machine learning procedure

The Boltzmann learning scheme is surprisingly simple. It has two versions; we will describe the version in which there are input units and output units and the machine must learn to map input vectors into output vectors. We begin with a set of I/O patterns on which the network is to be trained. The goal is to adjust the weights of the network so that, when we clamp the input units into one of these patterns and anneal the network, the corresponding output pattern will appear on the output units. If we clamp the inputs into a pattern that the system has not seen before, we would like the system to generalize according to the underlying regularities in the I/O pairs that it has seen.

The learning cycle has two phases, positive and negative, followed by weight adjustment. During the positive phase, we cycle through the entire set of I/O pairs. Each of these, in turn, is clamped into the input and output units and the rest of the network is then annealed, starting with a high temperature and gradually cooling down to thermal equilibrium at a temperature of 1. Once the system is close to equilibrium, we keep running it for a few more cycles, during which time each connection keeps a record of how often the two units it joins are on at the same time. After all of the I/O pairs have been presented in this way, each connection will have recorded a value, p^+, which is the fraction of time during the positive phase in which the two connected nodes are on at the same time at thermal equilibrium.

The negative phase is identical, except that only the inputs are clamped; the output units are allowed to settle into whatever states they like. In a network that has learned perfectly, the output units will exhibit the same probability distribution over output vectors as they would if we were still clamping them along with the inputs. During the negative phase we again run for a few more cycles after reaching equilibrium. Each connection records a second value, p^-, the fraction of time in which its two units are both active during the negative phase. If the network is producing all the right answers, the output units will exhibit the same probability distribution whether or not we clamp them and p^+ will be the same as p^-. If p^+ and p^- differ for a given connection, the two probability distributions can be made to match better by changing that connection's weight. If we ignore sampling noise, the difference between p^+ and p^- is exactly equal to the gradient of an information theoretic measure of the difference between the system's output behavior (during the negative phase) and its desired behavior (during the positive phase).[16] So we can perform steepest descent in this difference measure by changing the weight by an amount proportional to $p^+ - p^-$.

Even though Boltzmann machines only contain pairwise connections, the learning procedure allows them to capture higher-order constraints by dedicating hidden units to represent higher-order features. The learning procedure is interesting because it decides what the hidden units should represent, and it typically chooses to use distributed representations.

Unfortunately, the Boltzmann machine learning procedure suffers from all the usual problems of gradient descent in large parameter spaces. On top of this, the estimated gradient is usually inaccurate. The proofs assume that we have a large, unbiased sample of the statistics at equilibrium in the positive and negative phases. In practice, we never quite reach equilibrium, so the statistics are systematically biased. Moreoever, unless we are very patient there is also sampling error caused by taking too few samples.

Despite these difficulties, Boltzmann machines have been used successfully for aspects of speech recognition.[23] The simplicity of the Boltzmann machine architecture makes it feasible to build VLSI chips that contain many units, and it is just possible that this will lead to sufficient speed to make this a practical scheme. Even so, the scaling properties are poor and very large networks will remain impractical unless clever scaling tricks are discovered.

Ten years ago there were no connectionist learning procedures powerful enough to build useful representations with multiple layers of hidden units. Now there are many. The current challenge is to develop faster learning schemes that can be scaled up to networks with millions of modifiable connections. □

Acknowledgments

Scott Fahlman's research was supported by the Defense Advanced Research Projects Agency (DOD), DARPA Order No. 4976, monitored by the Air Force Avionics Laboratory under contract F33615-84-K-1520. Geoffrey Hinton's research was supported by the Office of Naval Research under contract N00014-86-K-00167 and by the National Science Foundation, grant number IST-8520359. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsors or the US government.

Further reading

Parallel Models of Associative Memory, eds. J. A. Anderson and G. E. Hinton, Erlbaum, Hillsdale, NJ, 1981.

Parallel Distributed Processing: Explorations in the Microstructure of Cognition, eds. D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, MIT Press, Cambridge, MA, 1986.

Cognitive Science 9, special issue on connectionist models and their applications, ed. J. A. Feldman, 1985.

References

1. G. E. Hinton, J. L. McClelland, and D. E. Rumelhart, "Distributed Representation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, eds. D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, MIT Press, Cambridge, MA, 1986.
2. S. E. Fahlman, *NETL: A System for Representing and Using Real-World Knowledge*, MIT Press, Cambridge, MA, 1979.
3. S. E. Fahlman, "Design Sketch for a Million-Element NETL Machine," *Proc. Nat'l Conf. AI*, Am. Assoc. for AI, Menlo Park, CA, 1980.
4. W. D. Hillis, *The Connection Machine*, MIT Press, Cambridge, MA, 1985.
5. F. Rosenblatt, *Principles of Neurodynamics*, Spartan Books, New York, NY, 1962.
6. M. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge, MA, 1969.
7. D. H. Ballard, G. E. Hinton, and T. J. Sejnowski, "Parallel Visual Computation," *Nature*, Vol. 306, Nov. 1983, pp. 21-26.
8. T. J. Sejnowski and C. R. Rosenberg, "NETtalk: A Parallel Network that Learns to Read Aloud," tech. report JHU/EECS-86-01, The Johns Hopkins Univ., EE and CS tech. reports, 1986.
9. G. E. Hinton, "Learning Distributed Representations of Concepts," *Proc. 8th An. Conf. Cognitive Science Soc.*, Lawrence Erlbaum Assoc., Hillsdale, NJ, 1986.
10. L. R. Bahl, F. Jelinek, and R. L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-5, No. 2, Mar. 1983, pp. 179-190.
11. A. G. Barto, "Learning by Statistical Cooperation of Self-Interested Neuron-Like Computing Elements," *Human Neurobiology*, Vol. 4, No. 4, Springer-Verlag, New York, NY, 1985, pp. 229-256.
12. J. A. Feldman and D. H. Ballard, "Connectionist Models and their Properties," *Cognitive Science*, Vol. 6, No. 3, Ablex, Norwood, NJ, 1982, pp. 205-254.
13. R. A. Hummel and S. W. Zucker, "On the Foundations of Relaxation Labeling Processes," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-5, No. 3, May 1983, pp. 267-287.
14. J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. Nat'l Academy of Sciences USA*, Vol. 79, No. 8, Apr. 1982, Nat'l Academy of Sciences, Washington, DC, pp. 2554-2558.
15. S. E. Fahlman, G. E. Hinton, and T. J. Sejnowski, "Massively Parallel Architecture for AI: NETL, Thistle, and Boltzmann Machines," *Proc. Nat'l Conf. AI*, Am. Assoc. for AI, Menlo Park, CA, 1983. Dist. by William Kafman, Inc., Los Altos, CA.
16. D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines," *Cognitive Science*, Vol. 9, No. 1, Ablex, Norwood, NJ, 1985, pp. 147-169.
17. S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-6, No. 6, Nov. 1984, pp. 721-741.
18. P. Smolensky, "Schema Selection and Stochastic Inference in Modular Environments," *Proc. Nat'l Conf. AI, AAAI-83*, Am. Assoc. for AI, Menlo Park, CA, 1983, pp. 109-113. Dist. by William Kafman, Inc., Los Altos, CA.
19. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4589, Am. Assoc. for the Advancement of Science, Washington, DC, 1983, pp. 671-680.
20. D. S. Touretzky and G. E. Hinton, "Symbols Among the Neurons: Details of a Connectionist Inference Architecture," *IJCAI*, Vol. 9, Aug. 1985, Morgan Kauffman, Los Altos, CA, pp. 238-243.
21. J. J. Hopfield and D. W. Tank, "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, Vol. 52, No. 3, Springer-Verlag, New York, NY, 1985, pp. 141-152.
22. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations

by Error Propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, eds. D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, Bradford Books, Cambridge, MA, 1986.

23. R. Prager, T. D. Harrison, and F. Fallside, "Boltzmann Machines for Speech Recognition," *Computer Speech and Language*, Vol. 1, No. 1, Academic Press, London, UK, 1986, pp. 1-20.

Scott E. Fahlman is a senior research computer scientist in the computer science department of Carnegie-Mellon University. His primary research interest is in massively parallel computing architecture for such AI problems as recognition, knowledge representation, learning, and simple kinds of search and inference. He has also been active in the definition, implementation, and standardization of the Common Lisp language.

Fahlman is a founder of Expert Technologies Inc. and Lucid Inc. and consults for Siemens Central Research Laboratories in Munich. He received his PhD from MIT in 1977, where he also received his BS and MS.

Geoffrey E. Hinton is an associate professor in the computer science department at Carnegie-Mellon University. His main research interest is in models of computation in the brain.

Hinton received a PhD in AI from Edinburgh University in 1978. He has been a research fellow in AI at Sussex University, a visiting scholar in cognitive science at UC San Diego, and a research scientist at the Applied Psychology Unit in Cambridge, England.

Readers may write to Fahlman at the Computer Science Dept., Carnegie-Mellon University, Pittsburgh, PA 15213.

What's new? Ask us!

In the Advanced Technology directorate of ESL (a 2500 person company), our charter is to develop the most imaginative and necessarily unconventional solutions to deal with some of the hardest signal processing problems and missions confronting our Nation.

Our goals:

At ESL, we are only interested in developing Advanced Technology Solutions for missions vital to our National Defense.

Our technology:

From Multi-Gigaflop Advanced Digital Signal Processor Systems, to Full-Custom 1.25 micronCMOS VLSI microcircuits, to Wafer-Scale Integrated GigaflopSystolic Processors, to MIPS and 68020 Based, Tightly-Coupled Distributed Computing Systems, to High-Speed Data-Flow Systems, to Artificial Intelligence Systems.

Our missions:

From Acoustic Signal Processing, to ESM/ELINT Systems, to Tracking Missions, to Advanced COMINT, to other Tactical & Strategic Surveillance and Reconnaissance.

Our business:

We are currently enjoying record business growth due to the quality of our work and our ability to provide products which meet our customers' needs.

Our requirements:

We are only interested in self-motivated, innovative, responsible, and curious engineering professionals, from the disciplines of Electrical Engineering, to Computer Science/Engineering, to Mathematics, to Physics and other Physical Sciences.

Our needs:

Openings exist at all levels: Managers, Staff Engineers, Engineers, System Architects, and Designers. PhD or equivalent, with clear track record of doing something big.

Your move:

At ESL, you'll find excellent salaries and benefits complemented by a stimulating, health-oriented environment. Please send resume to Bijoy G. Chatterjee, Deputy Director, Advanced Technology, Dept. COM-781, ESL, 1345 Crossman Ave., P.O. Box 3510, Sunnyvale, CA 94088-3510 or call him directly at (408) 743-6254. An equal opportunity employer. U.S. CITIZENSHIP IS REQUIRED FOR ALL POSITIONS.

There's a formula for the future. And it's ESL.

ESL
A Subsidiary of TRW

TRW