CONNECTIONIST EXPERT SYSTEMS

Connectionist networks can be used as expert system knowledge bases. Furthermore, such networks can be constructed from training examples by machine learning techniques. This gives a way to automate the generation of expert systems for classification problems.

STEPHEN I. GALLANT

Connectionist (or neural network) models are drawing increasing interest as useful tools for mainstream artificial intelligence (AI) tasks. Here we give an introduction to these models and examine expert systems that have connectionist networks for their knowledge bases. We call such systems *connectionist expert systems*.

There are powerful machine learning techniques for generating connectionist networks from training examples. These methods allow us to automate the construction of network knowledge bases for certain types of applications, thereby reducing the effort required for expert system development.

We have implemented a two-program package for constructing connectionist expert systems from training examples. The first program is a network knowledge-base generator that uses several connectionist learning techniques, and the second (MACIE) is a stand-alone expert-system inference engine that interprets such knowledge bases. The system has been tested on a large number of artificial problems and on several medical and economic problems where data were readily available.

Part of the work presented here is also described in [10], [11], and [14].

This work is partially supported by the National Science Foundation under Grant IRI-8611596.

CONNECTIONIST MODELS

The popularity of connectionist models in AI has taken wide swings, ranging from extreme enthusiasm in the 1960s to utter anathema in the 1970s. Currently there is an explosion of interest in these approaches; we estimate that research in this area has increased by two orders of magnitude over the last five years.

Theoretical developments triggered these shifts in interest. Early success with learning by neuron-like models called perceptrons [33] excited many researchers. Then in 1969 Minsky and Papert [27] pointed out key limitations of perceptrons that led to mass abandonment of this line of research. Recent advances have managed to sidestep or overcome many of the deficiencies with respect to learning, thereby bringing about the current flourishing of connectionist activity. Psychologists are now interested in connectionist models because of their structural and behavioral resemblance to systems of neurons [22], while AI researchers seek the ability to develop algorithms for machine learning, one of the key problems in the field of AI.

Connectionist models can be described according to their network, cell, and dynamic properties as follows:

(1) *Network properties.* A connectionist model consists of a network of (more or less) autonomous processing units called *cells* that are joined by directed arcs as in Figure 1. Each arc ("connection") has a numerical

weight $w_{i,j}$ that roughly corresponds to the influence of cell u_j on cell u_i. Positive weights indicate reinforcement; negative weights represent inhibition. The weights determine the behavior of the network, playing somewhat the same role as a conventional program. It has been known for a long time that any computer program could be simulated by an appropriate network [23].

In some models a subset of cells $u_1, \ldots, u_p$ are considered as *network inputs* that are set externally and that do not recompute their outputs. These cells have no entering arcs and are represented by squares in Figure 1. Other *output cells* have outputs that are taken to be the outputs of the network as a whole. In Figure 1, u_{11} and u_{12} are the output cells and are pictured with heavier exit connections. Cells that are neither input cells nor output cells are called *intermediate cells* or "hidden units". Intermediate cells are necessary for a network to compute difficult functions known as nonseparable functions; more on these functions later. We classify networks as either *feedforward networks* if they do not contain directed cycles or *feedback networks* if they do contain such cycles. For example, the network in Figure 1 is a feedforward network.

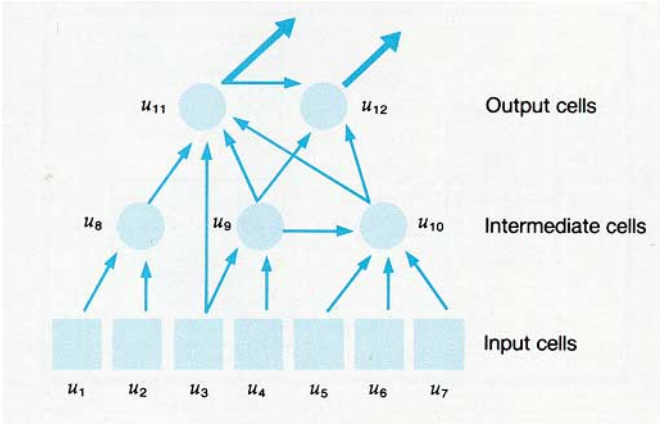


FIGURE 1. Connectionist Network

(2) *Cell properties.* Each cell, u_i, computes a single numerical cell output or *activation*. For example, the output of u_{11} is both an output of the network as a whole and an input for cell u_{12} to be used in the computation of its activation. Typically, every cell uses the same algorithm for computing its activation. The activation for a cell must be computed from the activations of cells directly connected to it and the corresponding weights for these connections. Thus, in Figure 1, when u_{11} is reevaluated its activation is determined by the activations of u_3, u_8, u_9, and u_{10} and the weights $w_{11,3}$, $w_{11,8}$, $w_{11,9}$, and $w_{11,10}$. We refer to the weights $w_{i,*}$ as the *weights of cell* u_i.

Cell inputs and activations may be discrete, taking on values {0, 1} or {−1, 0, 1}; alternatively, they may be continuous, assuming values in the interval [0, 1] or [−1, +1]. Where there is no confusion, we use u_i to refer to both the cell and the numerical activation of that cell. Every cell u_i (except for input cells) computes its new activation u_i' as a function of the weighted sum of the inputs to cell u_i from directly connected cells.

$$S_i = \sum_{j=0}^{n} w_{i,j} u_j$$
$$u_i' = f(S_i) \qquad (1)$$

If u_j is not connected to u_i, then $w_{i,j} = 0$. By convention there is a cell u_0 whose output is always +1 that is connected to every other cell u_i (except for input cells). The corresponding weights ($w_{i,0}$) are called *biases*. The biases are merely a constant term added to the sum in eq. (1).

(3) *Dynamic properties.* A connectionist model must specify when a cell computes a new activation value and when the change to that cell's output actually takes place. In some models, cells are visited in a fixed order, each cell reevaluating and changing its activation before the next one is visited. In other models, all cells compute their new activations simultaneously and *then* make changes to the cell outputs simultaneously. Still other models pick a cell at random, compute its new activation, and then change the output immediately before any other cell computes its new activation.

Machine learning is an additional part of many connectionist models. It deals with finding weights (and sometimes network topology) that will produce desired behavior for the model in question. Usually the learning algorithms work from training examples of desired behavior to generate appropriate weights. Each training example consists of correct values for a set of output cells in the network when input cells are given certain settings. (See Figure 1.)

In some paradigms ("easy learning"), the training examples specify values for intermediate cells;[1] otherwise the learning algorithm must also find appropriate intermediate cell values for the training examples ("hard learning"). Algorithms for easy learning are usually faster than those for hard learning and are more likely to produce one of the optimal sets of weights.

Machine learning is one of the most interesting and important research areas in the study of connectionist models. We will give only the briefest introduction to this work, focusing instead on the structure and workings of connectionist expert systems.

Clearly, there are a great variety of possible connectionist models, but a review of the major ones would be beyond the scope of this article. Some of the interesting properties of these systems include the following:

(1) *Learning.* There are powerful learning algorithms for determining connection weights, $w_{i,j}$, from training examples [17, 34].

[1] For "easy learning" the intermediate cells are treated as output cells during training.

(2) *Knowledge representation.* Connectionist models facilitate robust representations for certain types of problems [12], particularly problems involving recognition processes.

(3) *System integration.* Networks provide uniform representations of inputs from diverse sources. For example, a cell's input may come from p other cells concerned with vision, touch, and hearing. Connectionist models can automatically adjust to the cases where information is available from one of the 2^p possible subsets of the inputs. This is often important for real-world problems.

(4) *Parallelism.* Many models are well suited for parallel hardware implementation. We will not concern ourselves with hardware here since conventional computers are quite adequate for the algorithms being discussed.[2]

(5) *Error resistance.* Connectionist networks tend to be tolerant of errors in cell weights or activations. Network behavior gradually decays ("gracefully degrades") as the number of errors increases [20]. By contrast, if a single bit of a standard computer program is altered, that program might well become totally inoperative.

To learn more about connectionist research, readers should consult [13], [18], [22], [34], the January 1985 (Vol. 9, no. 1) issue of *Cognitive Science*, the proceedings of the annual Cognitive Science Society conferences, and the proceedings of the International Conference on Neural Networks.

EXPERT SYSTEMS

Although there is not general agreement as to what defines an expert system, we will be interested in the following features of a program designed to solve (or help solve) problems in a particular narrow domain:

(1) *Inferencing.* The program should be able to draw conclusions without seeing all possible information. For example, a medical system cannot require all potentially available tests prior to reaching a conclusion.

(2) *Interactive acquisition of data.* The program should direct the acquisition of new information in an efficient manner. For a medical system, this means seeking out highly relevant information through questions and ignoring possible tests that would not be relevant to the case at hand.

(3) *Modular structure.* An expert system usually contains three major parts (Figure 2):

(a) The *knowledge base* is a problem-specific module containing information that controls inferencing. Traditional expert systems generally employ `if-then` rules to represent this information, while connectionist expert systems use a network model for this purpose.

(b) The *inference engine* is the driver program for the expert system. Ideally it is problem independent so that it does not vary from one expert system to another.

(c) The *user interface* links the inference engine to the external environment using standard programming techniques.

(4) *Justification of conclusions.* The expert system should be able to tell how a conclusion was reached. This provides a check on the internal logic of the system and might give a novice user insights into the problem at hand.

These four features will serve as a definition of *expert system* for the purposes of this article. Readers interested in a more thorough overview should consult [7] or one of the many texts on the subject.

Classification Expert Systems

We restrict our consideration of expert systems to classification expert systems where the outputs from the system can be represented by variables, each of which assumes one of several possible values. The simplest case of a classification system is where there is only one output variable and it assumes one of two possible values; in other words a Boolean dichotomy.

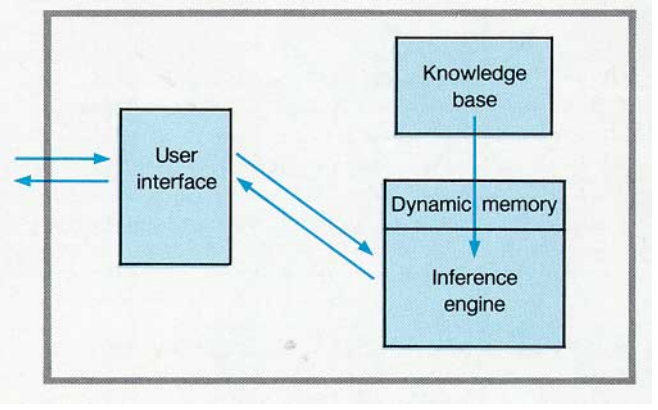


FIGURE 2. Expert System Structure

Limiting ourselves to classification systems is not as severe a restriction at it may appear at first, since many applications can be recast as classification problems. For example, a particular output variable x that takes on continuous values in the range [0, 1] (such as a probability) could be approximated by several Boolean choice variables, x_1', x_2', x_3', where each x_i' corresponds to ranges of values:

$$\begin{aligned} &\text{if } x \geq \tfrac{1}{4}, \quad \text{then } x_1' = 1; \quad \text{else } x_1' = -1; \\ &\text{if } x \geq \tfrac{1}{2}, \quad \text{then } x_2' = 1; \quad \text{else } x_2' = -1; \\ &\text{if } x \geq \tfrac{3}{4}, \quad \text{then } x_3' = 1; \quad \text{else } x_3' = -1; \end{aligned} \tag{2}$$

so that $x = .6$ would be represented by

$$x_1' = x_2' = +1, \qquad x_3' = -1.$$

Classification systems can approximate continuous output variables to arbitrary precision using this technique.

[2] Connectionist models should not be confused with the Connection Machine® trademark of Thinking Machines Corporation, one of several highly parallel hardware configurations that are suitable for connectionist models.

Programs for diagnosis, fault detection, and pattern recognition are examples of applications that can be represented as classification problems. Other tasks that are irregularly structured or procedural in nature would not be well suited for representation as classification problems, and therefore could not be handled conveniently by the methods presented here. An example of such a nonclassificatory task would be configuring computers as is done by the expert system R1/XCON [24].

The Problem of Knowledge Base Construction

The most difficult, time-consuming, and expensive task in building an expert system is constructing and debugging its knowledge base. In light of the proliferation of commercially available expert system environments (or shells), one could even argue that knowledge base construction is the *only* real task in building an expert system.

Several approaches have been explored for easing the knowledge-acquisition bottleneck for expert systems. Davis [6] developed an interactive system to aid in extracting knowledge from experts that does some clever consistency checks against the already existing knowledge base.

Work in symbolic machine learning also bears upon this problem [25, 26]. The former reference contains Quinlan's ID3 algorithm [32], a decision-tree-based method that is the basis for several commercially available packages. See [3] for a comparison of these methods to each other and [12] and [16] for comparisons with the connectionist expert system approach. Another approach focuses on probabilistic aspects, including belief networks. See [4], [5], [21], [29], and [30] for some of this work.

CONNECTIONIST EXPERT SYSTEMS

Can the learning power of connectionist models be harnessed for expert system construction? Alternatively, can connectionist models be used for inferencing in an interactive fashion, and can they be made to give reasonable justifications for their conclusions? Our aim is to answer both questions affirmatively by showing how to construct such connectionist expert systems.

We fear that our functional definition of the term *expert system* might offend those who consider the task of extracting rules from a human expert to be a prerequisite for a program to be called an expert system. These readers might prefer the term *interactive classification system with justification capability* to our use of *expert system*. Nevertheless, our goals have been to ease or eliminate the knowledge-acquisition bottleneck for expert system creation in data rich situations and to make a connectionist model behave as much as possible like an expert system.

Early research on connectionist inferencing was done by James A. Anderson [19] using an autoassociative model, where input cells and output cells are in correspondence. The system is trained so that when one of a fixed set of patterns is presented to the input cells then that same pattern is reproduced by the output cell activations. Now, when a partial (or perturbed) pattern is presented to the input cells, the system will often reconstruct the entire pattern of the closest training example in the activations of its output cells. This may take several iterations during which the output activations from the previous iteration are copied to the input cells for the next iteration.

Anderson's autoassociative model was not intended to direct the acquisition of data (backward chain), to provide justifications for inferences, or to determine when enough information was known to commit to a final conclusion. The question "What is the most likely answer?" is a separate question from "Is there sufficient evidence to adopt that conclusion or is more evidence needed?" Both tasks are important for inferencing from partial information. In the following sections we will specify a particular connectionist model for use with expert systems, express a medical-like problem in this framework and describe the generation of a connectionist expert system knowledge base from training examples. We will then detail MACIE, a general purpose connectionist inference engine, and look at some applications.

CONNECTIONIST NETWORK KNOWLEDGE BASES

A Simple Connectionist Model: Linear Discriminant Networks

We will be using connectionist models with the following properties:

(1) *Network properties.* Networks are feedforward models (no directed cycles allowed). Weights $w_{i,j}$ are positive or negative integers. Since there are no directed cycles, cells may be indexed so that if a directed arc points from cell u_j to u_i then $i > j$ (as in Figure 1). We henceforth assume a cell numbering consistent with this property.

It is convenient to express the network by a *weight matrix* W, where $w_{i,j} = 0$ if no arc connects cell u_j to u_i. As usual, $w_{i,0}$ are the bias values. Our cell numbering scheme for feedforward networks implies that $w_{i,j} = 0$ for $j \geq i$.

(2) *Cell properties.* Cell activations are discrete, taking on values +1, −1, or 0. These values correspond to logical values of *True*, *False*, or *Unknown*, respectively. Cell u_i computes its new activation u_i' as a linear discriminant function [8]:

$$S_i = \sum_{j \geq 0} w_{i,j} u_j$$

$$u_i' = \begin{cases} +1 \text{ or } \textit{True} & \text{if } S_i > 0 \\ -1 \text{ or } \textit{False} & \text{if } S_i < 0 \\ 0 \text{ or } \textit{Unknown} & \text{if } S_i = 0. \end{cases}$$

Such cells are also called threshold logic units or McCulloch–Pitts cells [23], or perceptrons [33]. By convention, cell u_0 always has activation +1, and cells $u_1, \ldots, u_p$ are set externally as inputs to the network.

(3) *Dynamic properties.* An iteration of the network consists of reevaluating each cell in index order and changing its activation before the next cell is reevaluated. One iteration suffices to bring the network to

steady state because of the lack of directed cycles and because of our indexing scheme.

We call this network model a *linear discriminant network.*[3] It is one of the simplest possible connectionist models, since there is no feedback and all computations can be performed using integer arithmetic.

A Sample Problem: Diagnosis and Treatment of Acute Sarcophagal Disease

To illustrate the workings of a connectionist expert system, we consider the following example that deals with acute theoretical diseases of the sarcophagus.[4]

Our model consists of symptoms, diseases, and treatments. There are six symptoms, two diseases whose diagnoses are based on (subsets of) the symptoms, and three possible treatments. Each training example is a patient's case history that lists

- symptoms present, absent, and for which there was no information;
- diseases present, absent, and for which there was no information; and
- treatments performed and not performed.

This information is used to generate a linear discriminant network as in Figure 3. This network then serves as a knowledge base for a connectionist expert system.

Figure 3 shows all connections and their associated weights $w_{i,j}$. The numbers within the nodes are bias values, $w_{i,0}$. By giving names to nodes of the network, we can specify a semantic interpretation for the activation of any cell. Thus, if we set each input cell to either *True* (+1), *False* (−1), or *Unknown* (0) and make one iteration over the intermediate and output cells, the network will compute which diseases are present and which drugs to prescribe from corresponding cell activations. For example, if the patient has swollen feet ($u_1 = +1$), but neither red ears ($u_2 = -1$) nor hair loss ($u_3 = -1$), then we can conclude that supercilliosis is present ($u_7 = +1$) because

$$0 + (2)(1) + (-2)(-1) + (3)(-1) > 0.$$

If other symptoms are *False* ($u_4, u_5, u_6 = -1$), then we can similarly conclude namastosis is absent ($u_8 = -1$), placibin should not be prescribed ($u_9 = -1$), and biramibio should be prescribed ($u_{10} = +1$). Cells u_A, u_B, and u_C are intermediate cells added to help with the computation of u_{11} (posiboost). Their activations are seen to be $u_A = +1$, $u_B = +1$, and $u_C = -1$. The addition of intermediate cells is necessary because without such cells no assignment of weights to u_{11} would work for all training examples. Finally, we compute that posiboost should also be prescribed ($u_{11} = +1$).

This example illustrates a very weak type of inference since it requires information on all input variables. Later we will see how to make more useful deductions based upon partial information.

The network with weights in Figure 3 serves as the knowledge base for a connectionist expert system that uses a special inference engine as in Figure 4.

[3] Such networks are also known as Gamba perceptrons [27].

[4] Although this is a simple example, it nevertheless captures over 85 percent of what is currently known in this highly specialized domain.

Network Generation: Inputs

To generate the connectionist knowledge base, we must specify the following information (see Figure 5):

(1) *The name of each cell corresponding to variables of interest* (symptoms, diseases, treatments). Each variable will correspond to a cell u_i. For Figure 3 the correspondence is as follows:

Symptoms
- u_1: Swollen feet
- u_2: Red ears
- u_3: Hair loss
- u_4: Dizziness
- u_5: Sensitive aretha
- u_6: Placibin allergy

Diseases
- u_7: Supercilliosis
- u_8: Namastosis

Treatments
- u_9 : Placibin
- u_{10}: Biramibio
- u_{11}: Posiboost

(2) *A question for each input variable, to elicit the value of that variable from the user* ("Does the patient have swollen feet?").

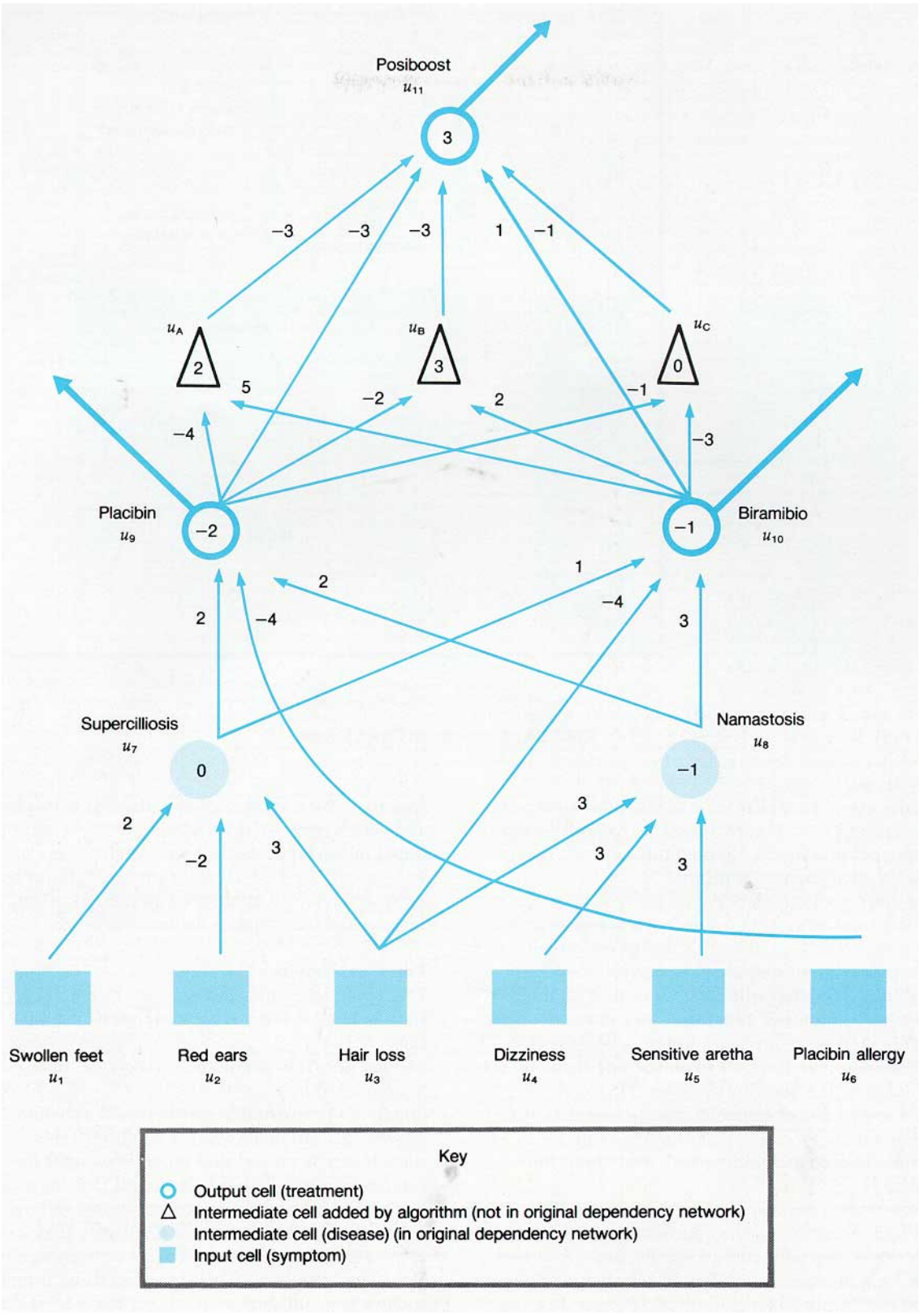
(3) *Dependency information for intermediate variables* (diseases) *and output variables* (treatments). Each of these variables has a list of other variables whose values suffice for computing it. For example, when deciding whether to prescribe placibin, it suffices to know which diseases the patient has and whether the patient is allergic to this drug. Symptoms such as "swollen feet" are *not* included in the placibin dependency list even though they indirectly influence whether placibin is prescribed. It is much easier to extract such qualitative information about "immediate causes" from a domain expert than it is to extract a specific function relating inputs to outputs. (See [29] and [31] for a discussion of this point.)

The dependency information is optional because every output cell may be specified as dependent on every input cell, as in Figure 6. This figure shows a default dependency that is useful for some applications. If more precise dependency information is available, however, then dependency lists improve network generation algorithms since accidental correlations between unrelated variables are prevented from influencing the final network weights.

For the sarcophagal problem, suppose the dependency information is as follows:

u_7 directly depends on u_1, u_2, u_3.
u_8 directly depends on u_3, u_4, u_5.
u_9 directly depends on u_7, u_8, u_6.
u_{10} directly depends on u_7, u_8.
u_{11} directly depends on u_9, u_{10}.

The dependency information specifies a dependency network consisting of an arc from u_j to u_i for every node



Biases are pictured within cells. The triangular cells were added to the original dependency network by the learning algorithm.

FIGURE 3. Final Linear Discriminant Network for Sarcophagal Disease

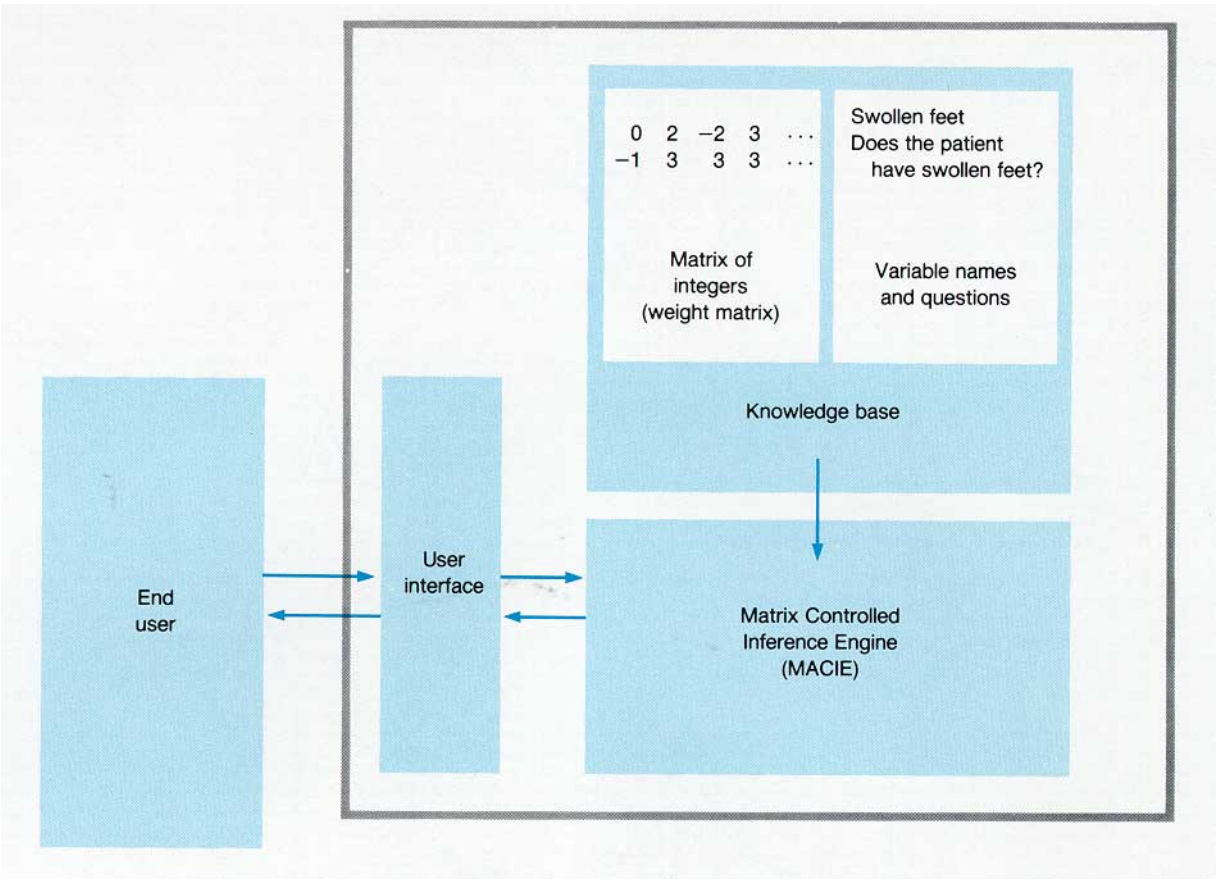


FIGURE 4. Connectionist Expert System

u_j on the dependency list for u_i. This is the same network topology as in Figure 3, except triangular-shaped cells are not present. In Figure 5 this network is represented as an adjacency matrix.

Another way to think of dependency networks is from the point of view of connections *not* present. If u_j is not connected to u_i, then this means we can always compute u_i without directly considering u_j, even though u_j might affect other variables that we do look at for u_i's computation. Eliminating a connection from u_j makes it easier to learn u_i's function because it reduces the number of inputs to u_i, thereby reducing the complexity of the learning problem. Therefore, we should expect better generalization to new data from our learned model of u_i, given that the same set of examples is used for training with or without the connection.

Some caution may be required to prevent directed loops from occurring in the dependency network when, for example, two cells could logically depend on each other. This situation is handled by eliminating arcs and letting the training examples implicitly specify any mutual dependency or by the use of choice variables (see the sidebar on p. 160).

(4) The final information supplied to the learning program is the set of training examples. For the sarcophagal problem, each example is a particular case and specifies which symptoms and diseases were present and which treatments were appropriate. This is illustrated in the input data where variables take on values +1, −1, or 0 for *True*, *False*, or *Unknown*, respectively. As was previously mentioned, non-Boolean data can be represented by groups of Boolean variables.

The Final Network

The dependency network may not be capable of perfectly modeling a given set of training examples because not every Boolean function can be represented as a single linear discriminant. If a Boolean function can be computed by a single cell, it is called a *separable* function; otherwise it is a *nonseparable* function. The sarcophagal problem involves a nonseparable function since it can be proved that no set of weights for cell u_{11} can produce correct behavior for all training examples. It is possible, however, to add additional intermediate (triangular) cells to form a final network that is capable of modeling any self-consistent set of training examples.[5] One way to do this is to add cells with random weights (e.g., integers between −5 and +5), as described in [17]. Such random cells were added to produce the final network pictured in Figure 3.

Because the added cells have fixed input weights that

[5] We do not require the network to compute intermediate or output variables with values of *Unknown* (0).

```
6 2 3
NAMES AND QUESTIONS:
 swollen feet
 Does the patient have swollen feet?
 red ears
 Does the patient have red ears?
 hair loss
 Is the patient suffering from hair loss?
 dizziness
 Is the patient dizzy?
 sensitive aretha
 Is the aretha sensitive?
 placibin allergy
 Is the patient allergic to placibin?
 Supercilliosis
 Namastosis
 Placibin
 Biramibio
 Posiboost
DEPENDENCY:   U1   U2   U3   U4   U5   U6        U7   U8        U9   U10  U11
               1    1    1    0    0    0         0    0         0    0    0 -> U7
               0    0    1    1    1    0         0    0         0    0    0 -> U8
               0    0    0    0    0    1         1    1         0    0    0 -> U9
               0    0    1    0    0    0         1    1         0    0    0 -> U10
               0    0    0    0    0    0         0    0         1    1    0 -> U11
EXAMPLES:
               1    1    1   -1    0   -1         1   -1         1   -1    1   TE#1
              -1   -1   -1    1    1   -1        -1    1         1    1   -1   TE#2
              -1   -1    1    1   -1    1         1    1        -1   -1   -1   TE#3
               1    1   -1   -1    1   -1        -1   -1        -1   -1   -1   TE#4
               1   -1    0    1    1    1         1    1        -1    1    1   TE#5
               1   -1   -1    1    1   -1         1    1         1    1   -1   TE#6
               1    1    1   -1   -1    1         1   -1        -1   -1   -1   TE#7
              -1    1    1   -1    1    1        -1    1        -1   -1   -1   TE#8
```

FIGURE 5. Input to the Learning Program

are chosen at random, adding cells is a very easy task indeed. In fact we did not really need to add all three cells in Figure 3; either one of the first two cells (u_A or u_B) would have permitted us to compute weights for u_{11} that worked for all examples. The third randomly generated cell, u_C, duplicates an input and is therefore of no help whatsoever; it could have been eliminated, and different but equivalent weights computed for u_{11}. We do not bother to test for and eliminate such useless cells, since the increase in speed would be negligible.

In some cases the original dependency network can be used unchanged for the final network if it can capture the behavior specified by the training examples. Also it is sometimes better to dispense with the addition of extra cells, even though original cells cannot correctly model all of the examples (i.e., the training examples are nonseparable). Forgoing extra cells can prevent "overfitting the data" with a model that is accurate on the training examples, but does not properly generalize to new inputs.

For simplicity, the sarcophagal examples were chosen so that additional cells were needed only for cell u_{11}. A more typical example would generate a layer of new intermediate cells just above the input cells for use by the rest of the network.

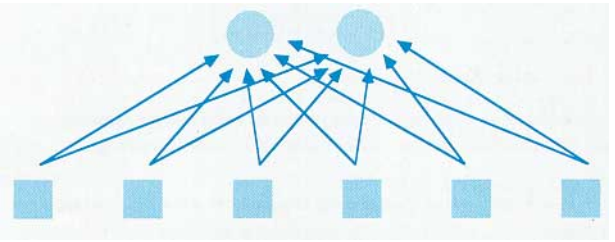


FIGURE 6. Default Network with No Dependency Information

The Pocket Algorithm: A Procedure that Generates Weights for Discrete Networks

Although a comprehensive treatment of learning in connectionist models lies beyond the scope of this article, we present a quick overview of the learning algorithms used in our system.

It is important that the training examples specify the desired activations for intermediate and output cells in the network (easy learning). This allows us to decompose the problem and consider each cell independently in terms of training example inputs, desired cell activations, and weights to be generated. Therefore, we can drop the subscript i and refer only to u and w_* to state the learning algorithm for a single cell. This makes weight computations much simpler.

To compute the vector of weights w_* for intermediate or output cell u, we first set $w_j = 0$ for every connection from a cell that is not in u's dependency list. We ignore these weights for the remainder of the computation. For connections from the remaining cells (that are in u's dependency list), we use the following procedure to compute the rest of w_*: For cell u let $\{E^k\}$ be the set of training example activations, and $\{C^k\}$ the corresponding correct activations for u. For simplicity we assume Boolean activations so that each C^k takes on values $\{+1, -1\}$ for $\{True, False\}$, respectively, while each component of E^k can take on values of $\{+1, -1, 0\}$ for $\{True, False, Unknown\}$. $E_0^k \equiv +1$ so that w_0 will be the computed bias for the cell in question.

The basic learning algorithm for generating weights is a modification of perceptron learning [33] called the pocket algorithm [13]. It computes perceptron weight vectors, P, which occasionally replace pocket weight vectors, w_*, as follows:

(1) Set P to the 0 vector.

(2) Let P be the current perceptron weights. Randomly pick a training example E^k (with corresponding classification C^k).

(3a) If P correctly classifies E^k, that is,

$$\{P \cdot E^k > 0 \quad \text{and} \quad C^k = +1\}$$

or

$$\{P \cdot E^k < 0 \quad \text{and} \quad C^k = -1\},$$

then,

(3aa) if the current run of correct classifications with P is longer than the run of correct classifications for the weight vector w_* in your pocket,

(3aaa) replace the pocket weights w_* by P, and remember the length of its correct run.[6]

(3b) Otherwise, form a new set of weights P' as follows:

$$P' = P + C^k E^k.$$

(4) Go to (2).

Table I illustrates the algorithm for several iteration steps.

A drawback to the pocket algorithm is that there is no known bound on the number of iterations required to achieve a fixed probability that the pocketed weights are the best possible. Nevertheless, if there are not too many training examples ($<10^5$), it is relatively easy to periodically check the pocketed weights against the set of all training examples in order to evaluate their performance.

An important advantage of the pocket algorithm over perceptron learning is that it works well with nonseparable or even contradictory training examples.

RATCHETS

Whenever there is a fixed set of training examples, we have found it very useful to modify (3aa) above to include a *ratchet*:

(3aa)′ If the current run of correct classifications with P is longer than the run of correct classifications for the weight vector w_* in your pocket *and P correctly classifies more training examples than w_*, ...*

Thus, we check a potential new w_* to see if it is really an improvement before making it the pocketed weights. This guarantees the new w_* correctly classifies a greater number of training examples than the previous w_*. Note that (3aa)′ is not possible when training examples are generated dynamically as described in the "Extensions" section since there are too many potential training examples to examine.

RULES

Another important modification allows rules to be specified in addition to the training examples. Here we define a *rule* as an example E' with corresponding classification C' that *must* be satisfied by the resulting weights w_*. Normal training examples, on the other hand, need not be satisfied by w_* if they are noisy or contradictory, or if no w_* exists that can simultaneously satisfy all training examples (i.e., nonseparable training examples). Thus, we now seek w_* that

(1) satisfies all rules, and
(2) satisfies as many training examples as possible without violating (1).

To meet these conditions, we first must modify the initial dependency network to form a final network where the rules are separable. Note that directly contradictory rules ($E' = E^s$, but $C' \neq C^s$) are not allowed. (Training examples, however, may be specified arbitrarily.)

We now modify (3b) as follows:

(3b)′ Otherwise, form a new set of weights P' as follows:

$$P' = P + C^k E^k, \tag{3}$$

and while P' violates any rule E' (with classification C') repeat eq. (3) using E' and C'.

One final modification to the basic algorithm involves a *choice group* of cells. Here we stipulate that exactly one cell from a group of cells should be *True* for any input presented to the group; in other words, we make a single choice from the group for any input. (Nilsson [28] refers to such groups as *linear machines*.) See [14] for more details.

[6] It might be said that these weights fit handily in *pocket* or *perceptron*.

TABLE I. Computation for Cell u_{10} Over Several Iterations

Perceptron weights				Perceptron run length	Pocket weights				Pocket run length	Example selected	Desired response	Correct response	Action
P_0	P_3	P_7	P_8		w_0	w_3	w_7	w_8					
	⋮			⋮		⋮			⋮		⋮		⋮
0	−3	2	4	3	1	1	1	−1	3	TE 6	1	Yes	3aaa
0	−3	2	4	4	0	−3	2	4	4	TE 3	−1	No	3b
−1	−4	1	3	0	0	−3	2	4	4	TE 2	1	Yes	—
−1	−4	1	3	1	0	−3	2	4	4	TE 4	−1	Yes	—

Weights

Once the shape of the final network has been determined by the dependency network (with the possible addition of random cells), we use connectionist learning methods to generate the corresponding weights and biases. There are a variety of algorithms that can be used for this purpose. The Pocket Algorithm (see sidebar) presents the basic method we use, but other methods are possible such as Back Propagation [34, 35], Associative Reward-Penalty cells [2], and Boltzmann machines [34].

The network, weights, and names and questions for variables constitute the knowledge base for a connectionist expert system as diagramed in Figure 4. The actual knowledge base for the sacrophagal example is given in Figure 7. In this file the connectionist network of Figure 3 is represented in matrix form, with one row being used for each intermediate or output cell in the final network. There is one column for each cell, including input cells and the bias terms.

MACIE: A CONNECTIONIST EXPERT SYSTEM INFERENCE ENGINE

We now present the details of an expert system inference engine that uses a connectionist network knowledge base. The connectionist network is represented

```
        6         5         3
swollen feet
Does the patient have swollen feet?
red ears
Does the patient have red ears?
hair loss
Is the patient suffering from hair loss?
dizziness
Is the patient dizzy?
sensitive aretha
Is the aretha sensitive?
placibin allergy
Is the patient allergic to placibin?
I/ Supercilliosis
   0    2   -2    3    0    0    0    0    0    0    0    0    0    0    0
I/ Namastosis
  -1    0    0    3    3    3    0    0    0    0    0    0    0    0    0
O/ Placibin
  -2    0    0    0    0    0   -4    2    2    0    0    0    0    0    0
O/ Biramibio
  -1    0    0   -4    0    0    0    1    3    0    0    0    0    0    0
I/ Intermediate Var. 1 for Posiboost
   2    0    0    0    0    0    0    0    0   -4    5    0    0    0    0
I/ Intermediate Var. 2 for Posiboost
   3    0    0    0    0    0    0    0    0   -2    2    0    0    0    0
I/ Intermediate Var. 3 for Posiboost
   0    0    0    0    0    0    0    0    0   -1   -3    0    0    0    0
O/ Posiboost
   3    0    0    0    0    0    0    0    0   -3    1   -3   -3   -1    0
  BIAS swol red  hair dizz sens plac Supe Nama Plac Bira Int1 Int2 Int3 Posi
```

FIGURE 7. Knowledge Base for the Sarcophagal Example (edited)

internally by a weight matrix suggesting the acronym MACIE for *Matrix Controlled Inference Engine.*

MACIE must use the connectionist network for the several tasks:

- inferencing based on partial information,
- finding unknown input variables that are key for reaching additional inferences, and
- producing justifications for inferences.

Expert System Algorithms: Initial Information

The program starts by listing for the user all variables and allowing any input variable to be initialized to *True* or *False*. Initialization is important since it focuses the subsequent problem solving. By contrast, systems based on decision trees cannot take full advantage of initial information because such information does not change the order in which nodes of the tree are examined [12].

When prompting for initial information and whenever else names must be given to the user, the system uses the appropriate character strings placed in the knowledge base (see Figure 4).

Inferencing/Forward Chaining

It is usually possible to deduce the activation for a cell u_i without knowing the values of all of its inputs; in other words, inferencing is possible from partial information.

For example, in Figure 3, if we know that the patient has swollen feet ($u_1 = +1$) and suffers from hair loss ($u_3 = +1$) then we can conclude the patient has supercilliosis ($u_7 = +1$) regardless of whether he or she has red ears. This is because the unknown variable cannot force the discriminant sum to change to negative.

More generally, for cell u_i we compute $KNOWN_i$, the partial weighted sum for cell u_i, and $MAX\_UNKNOWN_i$, the most that this weighted sum can change when we find values for all currently unknown variables:

$$KNOWN_i = \sum_{j:u_j \text{ known}} w_{i,j}u_j$$

$$MAX\_UNKNOWN_i = \sum_{k:u_k \text{ unknown}} |w_{i,k}|.$$

Now, whenever

$$|KNOWN_i| > MAX\_UNKNOWN_i \tag{4}$$

any additional information will not change the sign of the discriminant for u_i so that we can conclude

$$u_i = \begin{cases} +1 & \text{if } KNOWN_i > 0 \\ -1 & \text{if } KNOWN_i < 0. \end{cases} \tag{5}$$

It should be noted that this simple procedure addresses the question of *when* an inference is valid (eq. (4)) as well as *what* that inference should be (eq. (5)).

A newly changed activation of u_i can propagate up the network triggering further inferences. Thus, in Figure 3, if we infer a value for cell u_7 then this might provide enough additional information to conclude values for u_9 or u_{10} and so on. Each allowable inference can be made in one bottom-up pass through the cells due to the indexing scheme previously mentioned.

This inferencing technique works well in practice to allow the expert system to reach conclusions when only a fraction of the input values are known.

Confidence Estimation

At any time during a run, we can compute $\text{Conf}(u_i)$, an estimate of the likelihood that an unknown variable u_i will eventually be deduced to be *True* or *False*. $\text{Conf}(u_i)$ is useful for comparing two unknown variables (but cannot be interpreted as the probability that u_i will eventually be found true).

Several heuristics are available for computing $\text{Conf}(u_i)$. One of the simplest is the following:

- For a known cell,

$$\text{Conf}(u_i) = u_i.$$

- For an unknown input cell,

$$\text{Conf}(u_i) = 0.$$

- For other unknown cells, we compute $\text{Conf}(u_i)$ in index order by

$$\text{Conf}(u_i) = \frac{\sum_{j=0}^{i-1} w_{i,j}\,\text{Conf}(u_j)}{\sum_{j:u_j \text{ unknown}} |w_{i,j}|}.$$

We can compare likelihoods for all cells with one bottom-up pass through the network. It is easy to check that $-1 \le \text{Conf}(u_i) \le +1$.

Question Generation/Backward Chaining

If the system has not yet reached conclusions for enough of the output cells to complete the session, it must find an input cell with unknown activation and ask the user for its value. Again there are several possible heuristics for this task. Perhaps the simplest is the following:

(1) Select the unknown output variable u_i such that $|\text{Conf}(u_i)|$ is maximum. (This strategy starts with an output cell close to having its value set.) We say u_i is the cell being *pursued.*

(2) If pursuing cell u_i, find the unknown cell u_j with the greatest absolute influence on u_i. In other words, find a j yielding

$$\max_j |w_{i,j}| : u_j \text{ unknown}.$$

If u_j is an input variable, ask the user for its value (employing the character string question for u_j in the knowledge base). Otherwise pursue u_j and repeat (2).

Since we have been careful to prevent directed loops in the connectionist network, no variable can be pursued more than once without a question being asked. Therefore, this method of backward chaining quickly chooses an unknown variable to ask the user with no need for backtracking.

Other heuristics are also possible. For example, in step (1) we might look for max $\text{Conf}(u_i)$ rather than max $|\text{Conf}(u_i)|$ to emphasize output variables with values of *True*. Or for step (2) we might choose u_j to maximize

$$|w_{i,j}|\left(|\text{Conf}(u_j)| + \frac{1}{\text{level}(u_j)}\right)$$

to take into account confidence estimates and how far removed nodes are from the input nodes. Here we define

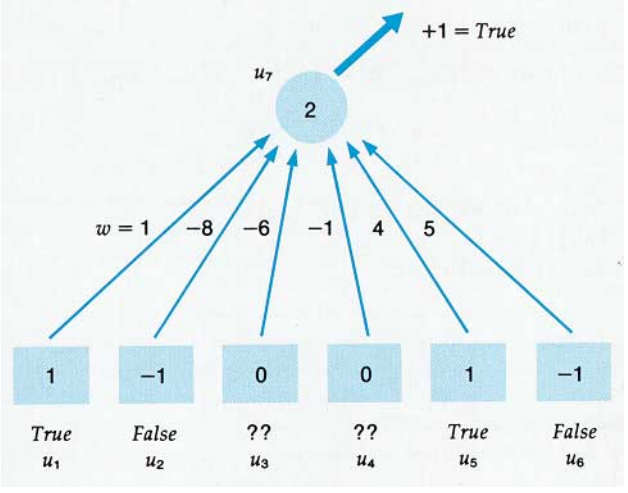
$$\text{level}(u_j) \equiv 1 + \text{distance to closest input cell.}$$

For confidence estimates and backward chaining, the choice of heuristics does not appear to affect the practical performance of the inference engine very much.

When the user is queried to obtain the value of an unknown variable, he or she can respond with *True*, *False*, or *Unobtainable*. Unobtainable means that the value of that variable will not be available for the remainder of the session. The inferencing mechanism treats such variables as known but with activations of 0. It should be noted the distinction between unknown and unobtainable variables: An unobtainable variable has a known final value of 0; an unknown variable has a temporary value of 0 that might be changed later in the session. In particular, a response of unobtainable might allow a cell's activation to be inferred because it reduces the value of *MAX_UNKNOWN* in eq. (4).

Explaining Conclusions by `if-then` Rules

The user can ask the system why it concluded a particular cell was *True* or *False* (see the sidebar on p. 164–65). The system will answer with an **`if-then`** rule applicable to the case at hand. It is amusing to note that these **`if-then`** rules are not represented explicitly in the knowledge base; they are generated by the inference engine as needed for explanations. Figure 8 gives an example to illustrate rule generation.



Rule:
If

U_2 = *False* and U_5 = *True*,

Then, conclude

U_7 is *True*.

FIGURE 8. Explanations by `if-then` Rules

The basic idea is that the system has already inferred that cell u_7 has value *True*, so we can take a minimal subset of the currently known information that is sufficient to make this inference. More explicitly, we perform the following calculation:

(1) List all inputs that are known and contributed to the ultimate positivity of the discriminant for u_7. In Figure 8 this gives u_1, u_2, and u_5. We omit u_6 since $w_6 u_6 < 0$ and since u_7 was concluded to be *True*.

(2) Arrange the list by decreasing absolute value of the weights. This gives u_2, u_5, u_1.

(3) Generate clauses for an **`if-then`** rule from this ordered list until

$$\left\{\sum_{u_i \text{ used for clause}} |w_i|\right\} > \left\{\sum_{\text{remaining inputs to } u_i} |w_i|\right\}$$

is satisfied. This produces the rule

```
if u2 is False and
   u5 is True
then Conclude that u7 is True.
```

This rule always holds by eq. (4) and justifies the inference made for the current problem. It should be noted that any rule for cell u_i will only involve cells connected to u_i in the final network.

In principle it would be possible to examine a connectionist network and produce every such **`if-then`** rule. These rules could then form the knowledge base of a traditional expert system.

However, this procedure would work only if the connectionist network was very small. Even for a network consisting of a single cell, the number of implicitly encoded **`if-then`** rules can grow exponentially with the number of cell inputs.[7]

A second type of explanation is available to the user. The user can request an explanation of why the current question is being asked. In response, the system can list the backward-chaining logic used to produce that particular question. See p. 164–65 for an example of this type of explanation.

MAINLINE FOR MACIE

We can now construct an inference engine based on the above capabilities. Here is the mainline procedure:

(1) Obtain initial information (under "Expert-System Algorithms: Initial Information," p. 162).
(2) Forward chain to make inferences ("Inferencing/Forward Chaining," p. 162).
(3) Quit if enough variables have been deduced.
(4) Backward chain to find an unknown variable whose value, if known, would likely be useful for inferencing ("Question Generation/Backward

[7] But not conversely. Any disjunctive normal form expression can be computed by a network model of approximately the same size [12]. In this sense connectionist models are more efficient for knowledge representation than disjunctive normal form expressions.

Sample Run of MACIE (edited)

```
MACIE Version 2.4

Numbers and names of variables:
             1:  swollen feet
             2:  red ears
             3:  hair loss
             4:  dizziness
             5:  sensitive aretha
             6:  placibin allergy
             7:  Supercilliosis
             8:  Namastosis
             9:  Placibin
            10:  Biramibio
            11:  Intermediate Var. 1 for Posiboost
            12:  Intermediate Var. 2 for Posiboost
            13:  Intermediate Var. 3 for Posiboost
            14:  Posiboost

Enter initial values for Input variables.

Format:  Variable number, value, ...
 2t  4f

PURSUING Posiboost
PURSUING Intermediate Var. 2 for Posiboost
PURSUING Placibin

 Is the patient allergic to placibin?
 --> y)es, n)o, u)nknown, ?)explain, i)nformation on vars.

n

PURSUING Posiboost
PURSUING Intermediate Var. 2 for Posiboost
PURSUING Biramibio

 Is the patient suffering from hair loss?
 --> y)es, n)o, u)nknown, ?)explain, i)nformation on vars.

?

Enter '0' for explanation of the last question
      '#' for explanation of variable number # (E.g. '3')

0

 hair loss is a factor
   which gives negative support for Biramibio
   which gives positive support for Intermediate Var. 2 for Posiboost
   which gives negative support for Posiboost
   which is the Goal Variable with highest Likelihood.

PURSUING Posiboost
PURSUING Intermediate Var. 2 for Posiboost
PURSUING Biramibio

 Is the patient suffering from hair loss?
 --> y)es, n)o, u)nknown, ?)explain, i)nformation on vars.

y

CONCLUDE:  (10) Biramibio FALSE.

CONCLUDE:  (13) Intermediate Var. 3 for Posiboost TRUE.

PURSUING Placibin
PURSUING Supercilliosis
```

```
 Does the patient have swollen feet?
 --> y)es, n)o, u)nknown, ?)explain, i)nformation on vars.

?

Enter '0' for explanation of the last question
      '#' for explanation of variable number # (E.g. '3')

10

Biramibio is  FALSE  due to the following rule:

     IF hair loss is  TRUE

     THEN CONCLUDE Biramibio is  FALSE


 Does the patient have swollen feet?
 --> y)es, n)o, u)nknown, ?)explain, i)nformation on vars.

y

CONCLUDE:  (7) Supercilliosis TRUE.

CONCLUDE:  (9) Placibin TRUE.

CONCLUDE:  (11) Intermediate Var. 1 for Posiboost FALSE.

CONCLUDE:  (12) Intermediate Var. 2 for Posiboost FALSE.

CONCLUDE:  (14) Posiboost TRUE.


Would you like to try a new case?
 --> y)es, n)o, ?)explain, i)nformation on vars.

?

      '#' for explanation of variable number # (E.g. '3')

9

Placibin is  TRUE   due to the following rule:

     IF placibin allergy is  FALSE
          AND Supercilliosis is  TRUE

     THEN CONCLUDE Placibin is  TRUE


Would you like to try a new case?
 --> y)es, n)o, ?)explain, i)nformation on vars.

?

      '#' for explanation of variable number # (E.g. '3')

14

Posiboost is  TRUE   due to the following rule:

     IF Intermediate Var. 1 for Posiboost is  FALSE
          AND Intermediate Var. 2 for Posiboost is  FALSE

     THEN CONCLUDE Posiboost is  TRUE


Would you like to try a new case?
 --> y)es, n)o, ?)explain, i)nformation on vars.

n
```

Chaining," p. 162). Ask the user for the value of this variable, and respond to any user request for explanations ("Explaining Conclusions by `if-then` Rules," p. 163).

(5) Go to (2).

Step (3) is problem specific. In some cases we seek the values of all output variables, whereas for other problems we might want to quit as soon as the first output variable was inferred to be *True*.

EXTENSIONS

Combining Learning with Expert System Operation (on-line learning)

So far we have separated the production of the connectionist network from the operation of the expert system, but this need not be the case. We can integrate the learning algorithm with the expert system according to the following time sequencing:

(1) Receive inputs at time t.
(2) Compute outputs from the expert system using the best (pocket) weights.
(3) Obtain the correct output for previous inputs presented at time $t - d$, where d is a nonnegative delay between the time inputs are received and the time the correct response is known.
(4) Perform one iteration of the learning algorithm with a training example consisting of the original input at time $t - d$ and its correct output. This may change perceptron weights or, less frequently, the pocket weights.

The input from time $t - d$ must be retained by the system for step (4) to take place, but the delay d can vary for inputs at different times. In other words the correct outputs need not arrive in temporal order.

The pocket algorithm (see p. 160) is appropriate for such dynamic systems because of two useful properties: First, the algorithm does not require that training examples be stored if there is a way of obtaining them as needed. Thus, an on-line learning system would need to retain only the last d inputs. Second, the pocket algorithm tracks changes to desired behavior in accordance with the training examples. Thus, if the training examples indicated a change in correct performance, the dynamically changing weights would eventually conform.

Noisy Data

The ability to handle dynamically generated training examples is important for constructing expert systems for problems involving noisy data. It is sometimes easy to list noise-free training examples and make a model of the noise involved, thereby allowing generation of noisy training examples.

To illustrate, suppose we have a problem involving 20 input variables where only 100 basic input patterns exist in the absence of noise. Due to noise, however, suppose that any input might be corrupted independently with some probability P. One way to handle this problem would be to create a set of noisy training examples containing copies of examples according to the effects of noise. Unfortunately this would require around 2^{20} training examples since any set of feature patterns would be possible due to the noise.

Figure 9 shows an easier way to handle this situation. When selecting training examples (step (2) in the pocket algorithms, p. 160), we use dynamic learning properties as follows:

(1) Select one of the 100 basic input patterns.
(2) Add noise to the 20 features using the noise model by reversing the sign for each feature with probability P. Do not alter the desired output.
(3) Use this training example for one iteration of the pocket algorithm.

See [14], and [15] for an example of this process applied to a fault detection system for a noisy environment.

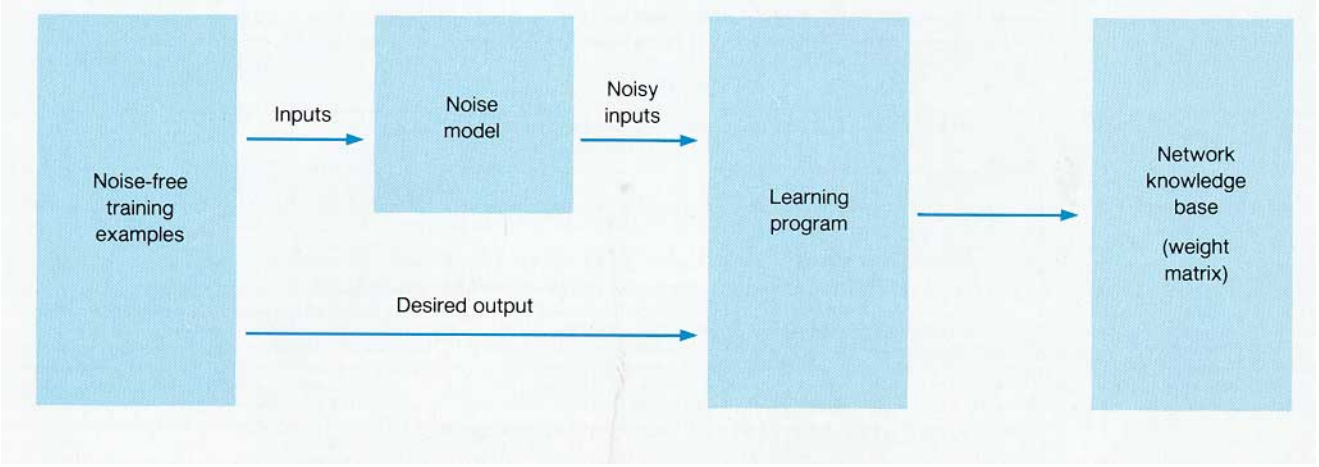


FIGURE 9. Dynamic Generation of Noisy Training Examples

APPLICATIONS

We have programmed the network generation algorithms on pages 160 and 164–65 and have produced connectionist expert systems from actual data for a variety of applications. Generally the results have been fairly good, but these problems were chosen as demonstrations because data were readily available and not because a particular expert system was actually planned for commercial or other reasons.

One early system was for diagnosis of causes of infantile diarrhea where the choices were *salmonella*, *shigella*, *rotovirus*, or *nonspecific*. Diagnosis prior to receiving laboratory results is difficult for experts, and their opinions are correct roughly 70 percent of the time. A doctor at Children's Hospital in Boston supplied about 70 cases from a study she had performed, and a connectionist expert system was constructed with little difficulty. Before demonstrating the system to the doctor, the connectionist network was examined, and simple rules involving two or fewer factors were written down.[8] Independently, the doctor wrote down rules she relied on, and the lists were compared. Almost all the induced rules were judged reasonable, and conversely, the human rules were not at great variance from the connectionist network generated from data. The system performance was interesting enough that new cases were brought in and tested. The results were about 70 percent correct diagnoses on these previously unseen cases, roughly the same performance as by doctors.

Another demonstration of these techniques involved the management decision of whether to continue with development of a new product or cancel development [16]. Here a wrong choice in either direction might mean a significant loss for the business concerned. We relied on data consisting of about 40 input features and over 100 actual cases that a colleague in the Northeastern College of Business Administration had collected previously. The resulting system behaved reasonably well, but would have required extensive interface work to bring it to market as a commercial product. Other demonstration projects now in progress involve medical and economic systems, and a chemical process fault diagnosis model that includes temporal information.

DISCUSSION

Connectionist Expert Systems (not) as Psychological Models

Connectionist researchers interested in psychological modeling would probably argue that MACIE is an implausible model for human reasoning. We would agree. We would also argue that expert systems in general do not reason like humans do (despite some claims to the contrary).

It is important to distinguish between the process humans use for arriving at a conclusion and the process they use for communicating (sequentially, by language) a justification for that conclusion [12]. We believe humans seldom follow logical rules in daily life, yet they may construct such rules to justify or explain those conclusions to other humans. Therefore, requiring a system to be both a good model of human reasoning and a close kin of a rule-driven expert system is asking for the impossible. Our desire for a practical system led us to favor performance over modeling and has resulted in a system much closer to conventional expert systems than to psychological models.

The Problem of Generalization

A key question for any inductive learning method is how well that method handles new input data after the system has been trained. This is a difficult question to answer in general because of the wide variety of problem types. We would face similar difficulties in trying to categorize how well conventional expert systems work in practice. About the best we can do is to collect individual cases of connectionist expert system implementations and compare them, whenever possible, with corresponding implementations by conventional methods. We plan to do this over the next several years.

Currently we are examining a relatively large family of noisy fault detection (or pattern recognition) problems where it will be possible to get overall quantitative results. Our tests so far have been quite promising with this class of problems [14, 15].

There is a related notion of generalization that involves recognizing spatial or temporal deformations of the input data. For example, if each input cell corresponds to a pixel on a rectangular grid we might want the system to recognize certain shapes regardless of their position on the grid [9]. This is an important area for connectionist research.

Discrete versus Continuous Models

So far we have worked with discrete rather than continuous connectionist models. When generating expert systems the functionality of the final network is the primary consideration, not the topological structure of the network used to achieve that functionality. This allows us to pick whatever shape network we want and employ faster learning algorithms that use only integer calculations.

On the other hand, it is possible that learning methods such as back propagation [34] that use continuous variables might generalize better to unseen examples, thereby creating more robust systems. This is an open question for future investigation.

Bayesian Models

It is interesting to compare connectionist expert systems with Bayesian methods. The former are geared more toward making decisions; the latter toward computing probabilities. Of course, the probabilities computed by a Bayesian model can aid decisions by a human (or computer) decision maker. Similarly, the connectionist model can compute approximate ranges of

[8] This `if-then` rule generation can be automated to produce lists of short rules.

probabilities for its outputs (as in eq. (2) under "Classification Expert Systems").

The Bayesian methods can give optimal accuracy provided that good underlying models are selected. In this sense connectionist models might be viewed as easy-to-compute approximations of probabilistic models. We are currently analyzing the fault detection problems mentioned earlier to determine how closely these connectionist models approach the Bayesian optimal decision rules.

Bayesian models seem not well suited for deciding *when* to commit to a value, as discussed under "Connectionist Expert Systems." We must either calculate over a large number of cases (corresponding to all possible values of unknown variables) or make some simplifying assumption on the model. Simplifying the model, however, raises the question of whether the result continues to be more faithful probabilistically than a corresponding connectionist network.

One often cited drawback of Bayesian models is a requirement for large amounts of data and computations. Recently Pearl [30], Kim [21], and others have shown that under certain conditions we can use network models to avoid such problems. Their *belief networks* (also called causal or Bayesian networks) have underlying structures that are identical to what we have referred to as dependency networks in the "Network Generation: Inputs" section (except that arrows point in the opposite directions for the two models).

In some ways belief networks are more flexible. They automatically allow probability inferences in all directions, such as from diseases to likely symptoms. For connectionist models this type of inference would require reversing the dependency network connections and regenerating the final network (using the same training examples). In other ways belief networks are less flexible because efficient propagation algorithms are limited to singly connected models (i.e., networks where two cells are joined by at most one undirected path in the dependency network). Both the dependency network and the final network for the sarcophagal problem would violate this restriction.

A Tool, Not a Replacement, for Knowledge Engineers

When we originally began this work, we thought automated methods might eliminate the need for knowledge engineers in many cases. We now believe, to the contrary, that these techniques best serve as additional tools that a knowledge engineer can employ for constructing all or part of an expert system. Furthermore, an understanding of the theory involved appears necessary for avoiding "garbage in, garbage out," the bane of automated tools.

CONCLUSION

We have examined how to construct a connectionist network from training examples and how to use it as the knowledge base for an expert system. The principal reason for doing this was to harness connectionist learning techniques for generating expert systems.

We believe connectionist expert systems present a promising approach to the knowledge-acquisition problem for expert systems. These methods are most appropriate for classification problems in environments where data are abundant and noisy, and where, conversely, humans tend to generate brittle and perhaps contradictory `if-then` rules. Moreover, the resulting systems run very fast, even on standard computers. This makes connectionist expert systems especially well suited for real-time applications such as process control.

The next several years should give us a better understanding of how useful this approach can be for producing commercial expert systems.

Acknowledgments. The author wishes to thank Gene Cooperman, Emmanouil Kalfaoglu, Joanna DeRiso, and the referees for their helpful suggestions.

REFERENCES

1. Anderson, J.A., and Rosenfeld, E., Eds. *Neurocomputing, A Reader.* MIT Press, Cambridge, Mass., 1988.
2. Barto, A.G., and Anandan, P. Pattern recognizing stochastic learning automata. *IEEE Trans. Syst. Man Cybern. 15,* 1985, 360–375.
3. Bundy, A., Silver, B., and Plummer, D. An analytical comparison of some rule-learning programs. *Artif. Intell. 27,* 2 (November 1985), 137–181.
4. Cheeseman, P.C. A method of computing generalized Bayesian probability values for expert systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence* (Karlsruhe, W. Germany, Aug. 8–12). 1983, pp. 198–202.
5. Cheeseman, P.C. Learning of expert system data. In *Proceedings of the IEEE Workshop on Principles of Knowledge Based Systems* (Denver, Colo., Dec. 3–4). IEEE Press, New York, 1984, pp. 115–122.
6. Davis, R., and Lenat, D.B. *Knowledge-Based Systems in Artificial Intelligence.* McGraw-Hill, New York, 1980.
7. Duda, R.O., and Shortliffe, E.H. Expert systems research. *Science 220,* 4594 (Apr. 15, 1983), 261–268.
8. Fisher, R.A. The use of multiple measurements in taxonomic problems. *Ann. Eugen. 7,* (1936) Part II, 179–188. (Also in *Contributions to Mathematical Statistics,* Wiley, New York, 1950.)
9. Fukushima, K., Miyake, S., and Ito, T. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Trans. Syst. Man Cybern. SMC-13,* 5 (Sept.–Oct. 1983), 826–834.
10. Gallant, S.I. Automatic generation of expert systems from examples. In *Proceedings of the 2nd International Conference on Artificial Intelligence Applications* (Miami Beach, Fl., Dec. 11–13). IEEE Press, New York, 1985, pp. 313–319.
11. Gallant, S.I. Matrix controlled expert system producible from examples. U.S. Patent Pending 707,458, 1985.
12. Gallant, S.I. Brittleness and machine learning. In *International Meeting on Advances in Learning* sponsored by Association Française pour l'Apprentissage Symbolique Automatique, CNRS, Paris, France. (Les Arcs, France, July 28–Aug. 1, 1986).
13. Gallant, S.I. Optimal linear discriminants. In *Proceedings of the 8th International Conference on Pattern Recognition* (Paris, France, Oct. 28–31). IEEE Press, New York, 1986, pp. 849–852.
14. Gallant, S.I. Automated generation of expert systems for problems involving noise and redundancy. In *AAAI Workshop on Uncertainty in Artificial Intelligence* sponsored by AAAI. (Seattle, Wash., July 10–12, 1987), pp. 212–221.
15. Gallant, S.I. Bayesian assessment of a connectionist model for fault detection. Tech. Rep. NU-CCS-87-25, College of Computer Science, Northeastern Univ., Boston, Mass., 1987.
16. Gallant, S.I., and Balachandra, R. Using automated techniques to generate an expert system for R/D project monitoring. In *International Conference on Economics and Artificial Intelligence* sponsored by AFCET, Paris, France. (Aix-en-Provence, France, Sept. 2–4, 1986), pp. 87–92.
17. Gallant, S.I., and Smith, D. Random cells: An idea whose time has come and gone . . . and come again? In *IEEE International Conference on Neural Networks* (San Diego, Calif., June). IEEE Press, New York, 1987, pp. 21–24.
18. Grossberg, S. *Studies of Mind and Brain.* Reidel, Hingham, Mass. (1982).

19. Hinton, G.E., and Anderson, J.A., Eds. *Parallel Models of Associative Memory*. Erlbaum, Hillsdale, N.J., 1981.
20. Hopfield, J.J. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences USA*. National Academy of Sciences, Washington, D.C., 1982, vol. 79, 2554–2558.
21. Kim, J.H., and Pearl, J. CONVINCE: A conversational inference consolidation engine. *IEEE Trans. Syst. Man Cybern. SMC-17*, 2 (Mar.–Apr. 1987), 120–132.
22. McClelland, J.L., and Rumelhart, D.E., Eds. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*. Vol. 2. MIT Press, Cambridge, Mass. (1986).
23. McCulloch, W.S., and Pitts, W.H. A logical calculus of the ideas imminent in nervous activity. *Bull. Math. Biophys.* 5 (1943), 115–133. (Reprinted: McCulloch, W.S. *Embodiments of Mind*. MIT Press, Cambridge, Mass., 1965.)
24. McDermott, J. R1: The formative years. *AI Mag.* 2, 2 (1981), 21–29.
25. Michalski, R.S., Carbonell, J.G., and Mitchell, T.M. *Machine Learning*. Tioga, Palo Alto, Calif., 1983.
26. Michalski, R.S., Carbonell, J.G., Mitchell, T.M. *Machine Learning*. Vol. 2. Kaufmann, Los Altos, Calif., 1986.
27. Minsky, M., and Papert, S. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, Mass., 1969.
28. Nilsson, N.J. *Learning Machines*. McGraw-Hill, New York, 1965.
29. Pearl, J. How to do with probabilities what people say you can't. In *Proceedings of the 2nd International Conference on Artificial Intelligence Applications* (Miami Beach, Fla., Dec. 11–13). IEEE Press, New York, 1985, pp. 6–12.
30. Pearl, J. Fusion, propagation, and structuring in belief networks. *Artif. Intell.* 29, 3 (Sept. 1986), 241–288.
31. Pearl, J. The logic of representing dependencies by directed graphs. In *AAAI-87* sponsored by AAAI, Menlo Park, Calif. (Seattle, Wash. July 13–17, 1987), pp. 374–379.
32. Quinlan, J.R. Learning efficient classification procedures and their application to chess end games. In *Machine Learning*, Eds. R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, Tioga, Palo Alto, Calif., 1983.
33. Rosenblatt, F. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan, Washington, D.C., 1961.
34. Rumelhart, D.E., and McClelland, J.L., Eds. *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*. Vol. 1. MIT Press, Cambridge, Mass.
35. Werbos, P.J. Beyond regression: New tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Dept. of Applied Mathematics, Harvard Univ., Cambridge, Mass., 1974.

CR Categories and Subject Descriptors: H.4.2 [**Types of Systems**]: Decision support; I.2.1 [**Applications and Expert Systems**]: Industrial Automation—*medicine and science*; I.2.2 [**Automatic Programming**]: Program synthesis; I.2.3 [**Deduction and Theorem Proving**]: Answer/reason extraction—*deduction*; I.2.4 [**Knowledge Representation Formalisms and Methods**]; I.2.5 [**Programming Languages and Software**]: Expert System Tools and Techniques—*MACIE*; I.2.6 [**Learning**]: Concept Learning, Induction, Knowledge Acquisition; J.7 [**Computers in Other Systems**]: Process Control

General Terms: Algorithms

Additional Key Words and Phrases: Connectionist expert systems, machine learning, MACIE, neural networks

Author's Present Address: Stephen I. Gallant, College of Computer Science, 221 Cullinane Hall, Northeastern University, Boston, MA 02115.

Announcement (*continued from p. 108*)

which both manuscript and illustrations can be delivered (machine-readable, camera-ready, etc.).

Subsequent to positive review, contracts will be offered by representatives of the ACM in consultation with Addison-Wesley. Terms will be competitive with those of other publishers. The combination of the editorial support described above, the marketing expertise of Addison-Wesley, and the advertising exposure and professional resources of the ACM should prove attractive to many authors.

Cast of Characters

ACM Press Books is a cooperative effort between ACM volunteers, ACM Headquarters, and Addison-Wesley. Inquiries should be addressed to the following representatives of these three constituencies:

Peter Wegner, Box 1910, Brown University, Providence, RI 02912. Tel:401-863-3311, E-Mail: pw@cs.brown.edu

Janet G. Benton, ACM, 11 West 42 St, New York, NY 10036. Tel:212-869-7440, E-Mail: janetb@acmvm.bitnet

Peter S. Gordon, Addison-Wesley, Route 128, Reading, MA 01867. Tel:617-944-3700

Series and area editors are in the process of being selected. An announcement of editors and of members of the advisory board will be made at a later time.

Many people have helped in bringing ACM Press Books to its present stage of development, including Paul Abrahams, Robert Ashenhurst, Lorraine Borman, Peter Denning, Adele Goldberg, Richard Hespos, Marsha Hopwood, Michael Mahoney, Mark Mandelbaum, Jim Maurer, Bernard Rous, Helen Takacs, Keith Wollman, and Steve Zilles.

We are keenly aware that building a reputation for excellence and relevance will be difficult and will depend critically on authors and volunteers. Your comments and suggestions, particularly at this formative stage, will be extremely useful. If you wish to explore the possibility of being an author or volunteer, or have suggestions and comments, please contact whichever person above you deem most appropriate.