Revision to appear in *Minds and Machines*

Expert Networks: Paradigmatic Conflict, Technological Rapprochement[†]

R. C. Lacher[‡]
Department of Computer Science Mail Code B-173
Florida State University
Tallahassee, FL 32306 USA
lacher@cs.fsu.edu

Abstract. A rule-based expert system is demonstrated to have both a symbolic computational network representation and a sub-symbolic connectionist representation. These alternate views enhance the usefulness of the original system by facilitating introduction of connectionist learning methods into the symbolic domain. The connectionist representation learns and stores metaknowledge in highly connected subnetworks and domain knowledge in a sparsely connected expert network superstructure. The total connectivity of the neural network representation approximates that of real neural systems and hence avoids scaling and memory stability problems associated with other connectionist models.

Keywords. symbolic AI, connectionist AI, connectionism, neural networks, learning, reasoning, expert networks, expert systems, symbolic models, sub-symbolic models.

[†] Paper given to the symposium *Approaches to Cognition*, the fifteenth annual Symposium in Philosophy held at the University of North Carolina, Greensboro, April 5-7, 1991.

[‡] Research partially supported by the US Office of Naval Research and the Florida High Technology and Industry Council.

Expert Networks: Paradigmatic Conflict, Technological Rapprochement[†]

R. C. Lacher[‡]
Florida State University

1. Introduction

Recently there has been considerable debate that could be characterized as "Connectionist vs symbolic approaches to AI". (See, for example, [Searle (1990)] and [Churchland and Churchland (1990)].) While this conflict continues at the theory level, pragmatic investigation is proving its futility. As an example, I present here a technological artifact that can be viewed as both a symbolic and connectionist system.

At first, the artifact appears to be a pure symbolic system, a garden variety rule-based expert system (production system). A change of representation discloses a striking resemblance to a connectionist system. Closer inspection reveals symbolic processing still taking place in a connectionist disguise. A second metamorphosis produces a true connectionist system of sub-symbolic neuronal processing. This last representation exactly reproduces the inferential dynamics of the original rule-based representation. Thus the same process can be viewed both as a symbolic rule-based system and as a sub-symbolic neural network-based system. These two views add to, rather than detract from, capability. The rule-based view makes it possible to analyze how conclusions are drawn, in effect providing the complete inference process for review. The neural network view introduces connectionist learning methods, allowing the system to learn from examples of correct inferences.

The connectionist representation learns and stores metaknowledge in highly connected subnetworks and domain knowledge in a sparsely connected expert network superstructure. The total connectivity of the neural network representation approximates that of real neural systems and hence avoids scaling and memory stability problems associated with other connectionist models.

[†] Paper given to the symposium *Approaches to Cognition*, the fifteenth annual Symposium in Philosophy held at the University of North Carolina, Greensboro, April 5-7, 1991.

[‡] Research partially supported by the US Office of Naval Research and the Florida High Technology and Industry Council.

2. Expert Systems

An expert system (ES) captures domain-specific knowledge and uses this knowledge to reason about problems in the domain. By far the most successful type of expert system so far has been the rule-based system [Giarratano and Riley (1989)]. A rule-based expert system consists of an inference engine that defines and executes the rules of inference and a rule base that comprises the domain-specific knowledge of the system.

One of the early successes in this field was MYCIN, a medical diagnosis and treatment advisory system in the domain of infectious diseases [Shortliffe (1976)], [Buchanan and Shortliffe (1984)]. The MYCIN system has been abstracted to an "expert system shell" in order to apply the same inference engine in other domains. A shell is just an expert system with an empty knowledge base, and usually some user interface system to facilitate the insertion and modification of rules. A shell that implements the MYCIN reasoning system is often called EMYCIN (for "Empty MYCIN"). One commercially available EMYCIN shell, M.1, is distributed by Teknowledge, Inc.[1] The computational experiments discussed below are based on M.1. The features of EMYCIN inferencing that are important in what follows are the evidence accumulator and the various logical operations [Shortliffe and Buchanan (1985)], [Giarratano and Riley (1989), pp. 533-536.].

A rule in EMYCIN has the form

$$\text{IF} \quad a \quad \text{THEN} \quad b \quad (cf) \tag{1}$$

where a and b are assertions and $cf \equiv cf_{b|a}$ is a certainty or confidence factor associated with the rule. The confidence factor is a static part of the rule and may have a value in the range $-1 \leq cf \leq 1$. During inferencing, assertions a, b may take on dynamically assigned and updated evidence values y_a, y_b in the range $-1 \leq y \leq 1$. The dynamically calculated evidence value of an assertion may be interpreted as a degree of confidence or correctness of the assertion. The evidence value y_b is then converted to a firing value z_b through the use of a threshold or other postprocessing criterion. The firing value is restricted to the range $0 \leq z \leq 1$.

Evidence accumulation

Suppose that we have a current dynamic evidence value for assertion b and subsequently encounter another assertion IF a THEN b (cf). Then EMYCIN adjusts the evidence value for b using the following assignment, where $x_{b|a} = cf_{b|a} \times z_a$:

$$y_b^{new} := \begin{cases} y_b + x_{b|a}(1 - y_b) \ , & \text{if both } y_b \text{ and } x_{b|a} \text{ are positive,} \\ y_b + x_{b|a}(1 + y_b) \ , & \text{if both } y_b \text{ and } x_{b|a} \text{ are negative,} \\ \frac{x_{b|a} + y_b}{1 - \min\{|y_b|, |x_{b|a}|\}} \ , & \text{otherwise.} \end{cases} \tag{2}$$

The output value z_b for assertion b is obtained by applying the firing criterion to y_b which may vary somewhat from one EMYCIN shell to another. M.1 uses the linear-threshold firing function

$$z_b := \begin{cases} y_b \ , & \text{if } y_b \geq 0.2; \\ 0 \ , & \text{otherwise.} \end{cases} \tag{3}$$

This firing value is then used as input to other rules of the form IF b THEN c (cf) using $x_{c|b} = cf_{c|b} \times z_b$. The inference process begins with external setting of the firing values of selected rule antecedents and spreads through the rule base under control of the inference engine. After the inference process terminates, the values of consequents with non-zero values constitute the conclusions of inference.

Logical Operations

EMYCIN shells differ somewhat in their treatment of logical operations, although they typically use minimum and maximum for AND and OR, respectively, and some kind of inversion for NOT:

$$\text{AND}(x_1, \ldots, x_k) := \min_i \{x_i\} \tag{4}$$

$$\text{OR}(x_1, \ldots, x_k) := \max_i \{x_i\} \tag{5}$$

$$\text{NOT}(x) := 1 - x. \tag{6}$$

The differences among shells appear in the way these values are thresholded (or otherwise postprocessed), after applying this common calculation, to determine whether the compound assertion fires. Generally, rules are allowed to have compound antecedents (using the defined logical operations) but compound consequents are discouraged.

M.1 recognizes three logical operations explicitly: AND, NOT, and UNK (for “unknown”). The operation UNK is essentially NOR (NOT following OR) using the definitions above and the M.1 firing functions.[2] For AND, M.1 uses the same firing function as for evidence combining, given above by (3). For NOT, M.1 uses a firing function that is a strict threshold, with threshold value 0.8, resulting in discrete values for NOT and UNK operations:

$$z_b := \begin{cases} 1\ , & \text{if } y_b \geq 0.8; \\ 0\ , & \text{otherwise.} \end{cases} \tag{7}$$

Composing appropriate functions from (3) – (7) yields the following throughput functions (from input to firing value) for logical operations in M.1:

$$\mathrm{AND}(x_1, \ldots, x_k) := \begin{cases} \min\{x_i\}\ , & \text{if } \min\{x_i\} \leq 0.2; \\ 0\ , & \text{otherwise;} \end{cases} \tag{8}$$

$$\mathrm{NOT}(x) := \begin{cases} 1\ , & \text{if } x \leq 0.2; \\ 0\ , & \text{otherwise;} \end{cases} \tag{9}$$

$$\mathrm{UNK}(x_1, \ldots, x_k) := \begin{cases} 1\ , & \text{if } \max\{x_i\} \leq 0.2; \\ 0\ , & \text{otherwise.} \end{cases} \tag{10}$$

There are many points to address in order to understand the inferential dynamics of EMYCIN systems, including order dependence of rules. The features discussed above are those most relevant to the expert network technology that is the focus of this paper.

3. Computational Networks

Networks that can compute provide a general framework within which both symbolic and sub-symbolic neural processing can be formulated. A (discrete) *computational network* (CN) is essentially a directed graph with information processing capability that is empowered by localized processing at its edges and vertices. Three distinct classes of local functionality are recognized. The first two are associated with vertices: (1) a combining function Γ integrates vertex input $x_1, \ldots, x_n$ into an internal state $y = \Gamma(x_1, \ldots, x_n)$, and (2) an output (or firing, or activation) function φ converts the internal state into an activation value $z = \varphi(y)$. Vertices so equipped are called computational nodes, or just *nodes.* The third class of functionality is associated with directed edges: (3) a synaptic function σ converts the output of the node at its initial end (pre-synaptic input) into an input value for the node at its terminal end (post-synaptic input). Edges so equipped are called *connections.* In many cases, the synaptic functions of a CN are defined as simple multiplication by a weight or connection strength. These are called *linear synapses.*

Thus a CN consists of nodes and connections organized into a directed graph structure. Given a labeling of the nodes, a single subscript indicates an association with the vertex so labeled, and a double subscript indicates association with a directed edge, with "assignment statement order": a subscript ji indicates association with the edge from vertex i to vertex j. In this notation, Γ_j and φ_j are the combining and firing function, respectively, of node j, and σ_{ji} is the synaptic function of the connection from node i to node j. If the ji synapse is linear, then $\sigma_{ji}(z_i) = w_{ji}z_i$, where w_{ji} is the weight of the connection. For simplicity in this discussion, it is assumed that the node labels constitute an enumeration $1, \ldots, n$.

In order to compute with a CN, we need to know how to initialize the network, perform the computation, and retrieve results. Initialization and result retrieval are accomplished through designated *input* and *output* nodes, respectively. Initialization consists of setting the internal states of all input nodes to values designated outside the CN and setting all other internal states to zero. Result retrieval consists of reading the activation values of all of the designated output nodes.

The computational process itself is more complex. First a method of keeping time must be specified. Discrete time has been built into the concept of CN used herein[3], but that still leaves several possibilities for deciding when to update a given local value in the network. The biologically and psychologically motivated event-driven method is preferred for expert networks, but other possibilities include synchronous (parallel update) and several variations of asynchronous (where updating occurs randomly over the network). (See [Hertz, Krogh, and Palmer (1991)] and [Lacher (1992a)] for more discussion of update timing methods.) Once the timing method is selected, network computation proceeds as described, applying the update assignment statements

$$x_{ji} := \sigma_{ji}(z_i) \quad \text{for} \quad i = 1, \ldots, n \tag{11}$$

$$y_j := \Gamma_j(x_{j1}, \ldots, x_{jn}) \tag{12}$$

$$z_j := \varphi_j(y_j) \tag{13}$$

repeatedly until some kind of stable activation state $(z_1, \ldots, z_n)$ is reached.[4] For most of the CNs considered herein, all of these methods of timekeeping are equivalent: If the network topology is acyclic, that is, if the underlying digraph has no directed cycles, then after finitely many applications of (11), (12), (13) the network reaches an activation state that remains unchanged by further updating. This *terminal activation state* is independent of the timing method used. (For further discussion and proof, see [Lacher (1992a)]). The terminal activation values associated with output nodes constitute the network output.

4. Expert Networks

Kuncicky et al. have shown how rule-based expert systems can be translated into "neural" networks [Kuncicky (1990)], [Kuncicky (1991)], [Kuncicky, Hruska, and Lacher (1992)]. The network topology is determined by the rule base, and the various node functionalities are determined by the inference engine. The resulting computational network is called an *expert network* (EN)[5]. The activation dynamics of the expert network are exactly the same as the inferential dynamics of the original expert system, provided no learning has occurred.

Regular Nodes

The expert network is constructed in two stages. First an inference network is created from the rule base. Each vertex in this network represents an assertion (antecedent or consequent of a rule) and each directed edge represents a rule. The certainty factor of the rule is placed on the edge as a weight. Thus a rule of the form (1) defines a connection

$$a \xrightarrow{cf} b \tag{14}$$

The evidence accumulation process of the inference engine defines functionality for these vertices, and the edges process initial to terminal value by multiplication by cf. The resulting computational network is the first order expert network defined by the expert system. Note that all of the nodes in this network represent assertions; they are called *regular* nodes.

If the original expert system shell is a version of EMYCIN, the EN is called an EMYCIN network. The EMYCIN evidence accumulator given by equation (2) can be written in closed form. The positive and negative evidence values for regular node b are given by

$$\begin{aligned} y_b^+ &= +1 - \prod_{x_{b|a}>0} (1 - x_{b|a}) \quad \text{and} \\ y_b^- &= -1 + \prod_{x_{b|a}<0} (1 + x_{b|a}), \end{aligned} \tag{15}$$

respectively; then positive and negative evidence are reconciled, yielding the internal state of the node:

$$y_b := \Gamma_b(x_{a|1}, \ldots, x_{b|n}) \equiv \frac{y_b^+ + y_b^-}{1 - \min\{y_b^+, -y_b^-\}} \tag{16}$$

(where it is assumed that node names range over $1, \ldots, n$). The output function φ_b for a regular node b is the firing function for assertions defined by equation (3):

$$z_b := \varphi_b(y_b) \equiv \begin{cases} y_b \,, & \text{if } y_b \geq 0.2; \\ 0 \,, & \text{otherwise.} \end{cases} \tag{17}$$

OP Nodes

The second order network is obtained by expanding the regular nodes representing compound antecedent statements into subnetworks. A regular node antecedent such as in the connection

$$\mathrm{OP}(a_1, \ldots, a_k) \xrightarrow{cf} b \tag{18}$$

expands to the subnetwork

$$\begin{aligned} a_1 &\xrightarrow{1} \mathrm{OP} \\ &\cdot \quad \cdot \quad \cdot \\ a_k &\xrightarrow{1} \mathrm{OP} \\ \mathrm{OP} &\xrightarrow{cf} b. \end{aligned} \tag{19}$$

Those a_i that are consequents of other rules are already represented by existing nodes. New nodes are created for the other a_i. A connection of weight 1 is added from each a_i to the new OP node, and a connection of weight cf added from the OP node to the consequent b replaces the original outgoing connection. All connections into OP nodes have fixed weight 1 and are called *hard* connections. Connections into REG nodes have weight originating as a certainty factor of a rule and are called *soft* connections.

The combining function Γ for an OP node performs the same logical computation defined by the inference rules. For EMYCIN these are given in equations (4), (5), and (6). The output function φ for an OP node is the same as the firing condition for the logical operation. For M.1 these are given by (3) for AND and (7) for NOT and UNK.

The WA Testbed

There are four generic node types in an M.1 network, REG nodes as described above and three OP node types corresponding to the three logical operations AND, NOT, and UNK recognized by M.1. Thus an M.1-based expert system transforms into an expert network with these four node types.

One well known EMYCIN-based expert system is designed to give advice on the choice of wine with a meal. This system has evolved in the public domain for more than a decade. A particularly refined version, WA, is distributed as a demonstration with M.1. WA has 44 rules, 23 input variables describing the proposed meal and related circumstances, and 13 outputs representing wine selections. WA captures a considerable amount of human expertise and intuition. Translating and expanding WA results in an expert network with 97 nodes and 175 connections, 64 of which are soft. Of the 2^{23} possible input values for

WA, at least $6,912$ represent valid queries. The WA expert network is illustrated in Figure 1.

5. Learning

Creating a rule base such as WA often requires significant quantities of expertise-intensive human labor, particularly when rule uncertainty must be quantified. Experts, by definition, can come to correct conclusions when presented with a set of inputs within their domain of expertise: they know *what* is correct. The problem with rule extraction is that it requires experts to explain *how* they reason, and to do so within an unfamiliar and possibly constraining system of inference rules. This costly and time-consuming task is referred as the *knowledge acquisition bottleneck* and is a major limitation to the applicability of expert systems.

The translation of an expert system into an expert network opens the possibility of applying connectionist learning methods to the expert system. The general connectionist supervised learning approach requires a pool of desired, or known, I/O data, and uses this pool to train the network to reproduce the correct I/O through network computation. In the case of expert networks, I/O is inferencing. Thus, to train an expert network requires a pool of correct inferences in the form of pairs (values for input variables, correct conclusions). For WA this would mean pairs (meal specification, wine selected). This kind of data is the result of an expert reasoning about the domain of expertise and is relatively easy to acquire, because that is precisely what experts do: given input, they come to conclusions about the input. Experts know *what* is correct but not necessarily *how* they know it.

We have developed two supervised learning methods particularly suited to expert networks and other high-level CNs with linear synapses. The first is a reinforcement method based loosely on the shaping technique of behavioral psychology, and the second is an adaptation of the well-known backpropagation method.

Goal-Directed Monte Carlo

Goal-directed Monte Carlo was introduced by Kuncicky (1991) using by-now standard concepts in reinforcement training of neural networks. The basic idea is to analogize with methods used to train or “shape” an animal’s behavior. One such training session might begin with a completely untrained pigeon and the goal of training the pigeon to peck a target for food. At first, the pigeon exhibits completely random behavior with respect to the target. The trainer reinforces the pigeon with a food pellet when the pigeon faces the target side of the cage. This tends to focus the pigeon's attention, resulting temporarily in smaller range of random behavior and higher probability that the pigeon will face the target side. As the behavior is modified, the goals for reinforcement become more stringent, until the pigeon must actually peck the target for reward.

The analogies are as follows. The pigeon corresponds to a computational network with linear synapses. Random behavior of the pigeon corresponds to random variation ("noise") in the synaptic weights of the CN about a base value. Decrease in attention, occurring in the pigeon as the time since last reinforcement increases, corresponds to an increase in the amplitude of noise in the synapses. Reinforcement corresponds to a resetting of the base values of synaptic weights to the particular noisy value that resulted in rewardable behavior. Attention is reset to high (zero noise level) at the time of reinforcement. Reinforcable behavior is determined by calculating error at output nodes. Reinforcability is based on a decreasing error criterion – the better the network is trained, the more is expected of it.

Mathematical specification of this analogy has been worked out. (See [Kuncicky (1991)], [Kuncicky, Hruska, and Lacher (1991)] for details.) A significant virtue of the method is its independence of node combining and output functions, which may be quite non-linear in some CNs.

Computational Network Back-Propagation

Introduced by Werbos (1974) and rediscovered and popularized by the PDP group in the late 80's [Rumelhart and McClelland (1986)], backpropagation is one of the most widely known and successfully used connectionist learning methods. Normally, however, backpropagation is applied to layered feedforward computational networks with very simple (low-level) processing functionality: linear synapses, additive combining functions, and sigmoidal or gaussian output functions. To generalize backpropagation to acyclic CNs, it is necessary to (1) localize forward and backward activation to free the algorithm of the layer structure, and (2) decouple the process of node error assignment from the weight correction step. Step (1) is easily accomplished and described previously and in [Lacher, Hruska, and Kuncicky (1992)]. Step (2) is accomplished through the concept of "influence". In a general CN, how does the output of node j influence the output of an immediate successor node k? The answer is the *influence factor* $\varepsilon_{kj} = \partial z_k / \partial z_j$ given by

$$\varepsilon_{kj} = \varphi_k'(y_k) \times \frac{\partial \Gamma_k}{\partial x_{kj}}(x_{k1}, \ldots, x_{kn}) \times \sigma_{kj}'(x_j). \qquad (20)$$

Influence factors are dependent on particular network input: The derivative of φ_k is evaluated at the terminal internal state of node k, the partial of Γ_k is evaluated at the terminal post-synaptic input to node k, and the derivative of σ_{kj} is evaluated at the terminal output of node j. Influence factors are associated with connections and are calculated during forward activation of the network.

Once influence factors have been calculated for all the connections of an acyclic CN during forward activation, error can be assigned to all of the nodes in the CN during a reverse activation. Suppose a presentation of input data ξ results in network output ζ and influence factors ε_{kj}. Then we assign error to each output node j using

$$e_j := I_j - \zeta_j, \tag{21}$$

where I is ideal output, and to all other nodes in the network using

$$e_j := \sum_k \varepsilon_{kj} e_k. \tag{22}$$

Applying this last equation recursively is in essence a reverse activation of the network (the reversed network is still acyclic) using influence factors as reverse connection strengths, addition as reverse combining functions, and identity as reverse output functions. The resulting terminal reverse activation state is an error assignment throughout the network. The error assignment process works in any acyclic CN.

Now assume error has been assigned to all the nodes as described. For any node with linear incoming synapses, *local* gradient descent can be applied *selectively* to that node. The gradient of square error at node j with respect to synaptic weights $w_{j1}, \ldots, w_{jn}$ is the vector of partial derivatives

$$\frac{\partial E}{\partial w_{ji}} = -2 e_j \varphi_j'(y_j) \frac{\partial \Gamma_j}{\partial x_{ji}}(x_{j1}, \ldots, x_{jn}) z_i. \tag{23}$$

A step with *learning rate* η and *momentum* μ in the direction of steepest descent of square error is given by

$$\Delta w_{ji} = \eta e_j \varphi_j'(y_j) \frac{\partial \Gamma_j}{\partial x_{ji}}(x_{j1}, \ldots, x_{jn}) z_i + \mu \Delta w_{ji}^{prev}. \tag{24}$$

The resulting learning method is called Computational Network Back-Propagation, or CNBP. It applies in any acyclic CN at any nodes with linear incoming synapses. All that is required to complete an implementation of CNBP is calculation of the various derivatives appearing in (20) and (24). Details of the theory of influence factors, acyclic activation, and CNBP may be found in [Lacher (1992a)].

Expert Network Back-Propagation

An EMYCIN expert network satisfies all the requirements for GDMC and CNBP learning: an acyclic CN with linear synapses. Learning can take place only at soft connections (connections into regular nodes), but of course all connections must be used in the error assignment process.

Of the derivatives appearing in (20) and (24), σ'_{kj} is just the weight w_{kj} of the kj connection, and φ'_j is easily calculated, but may vary because of choices of φ made during a particular implementation. Except in the case of REG nodes, the partials of the various combining functions are also easily computed from the definitions. For REG nodes these are given as follows:

$$\frac{\partial \Gamma_j}{\partial x_{ji}}(x_{j1}, \ldots, x_{jn}) = \begin{cases} \frac{1}{1-x_{ji}} \frac{1-y_j^+}{1+y_j^-}, & \text{if } y_j^+ \geq |y_j^-| \text{ and } x_{ji} > 0; \\ \frac{1}{1-x_{ji}} \frac{1+y_j^-}{1-y_j^+}, & \text{if } y_j^+ < |y_j^-| \text{ and } x_{ji} > 0; \\ \frac{1}{1+x_{ji}} \frac{1-y_j^+}{1+y_j^-}, & \text{if } y_j^+ \geq |y_j^-| \text{ and } x_{ji} < 0; \\ \frac{1}{1+x_{ji}} \frac{1+y_j^-}{1-y_j^+}, & \text{if } y_j^+ < |y_j^-| \text{ and } x_{ji} < 0 \end{cases} \tag{25}$$

provided $x_{ji} \neq \pm 1$. This calculation, along with the formula in the boundary cases, may be found in [Lacher, Hruska, and Kuncicky (1992)].

Another technical problem arises in the context of expert networks: the knowledge space (the set of possible weight vectors) is a hypercube rather than an open subset of euclidean space, so boundary knowledge states exist. As usual, calculus doesn't work well at the boundary, so more primitive means must be devised. My colleagues and I have proposed several ways of dealing with the boundary, including the shrink-and-test loop described in [Lacher, Hruska, and Kuncicky (1992)], "doubt" and "caution" parameters. The version of CNBP that implements influence factors and uses the shrink-and-test loop is the specific version used in the learning experiments discussed below. We refer to this as Expert Network Back-Propagation (ENBP).

Learning Experiments

The methods described have been tested on several M.1-based expert systems, including the wine advisor WA described at the end of Section 4 [Lacher, Hruska, and Kuncicky (1991)] and the Control Chart Selection Advisor of [Dagli and Stacey (1988)], [Hruska and Kuncicky (1991)]. The following experimental procedure was used.

For testing purposes we have a functioning expert system which is used to define expert knowledge. Using the test expert system, or the functionally identical expert network described in Section 4, we generate specific examples of correct reasoning. Then the EN is ablated by setting some of the soft connection weights to zero. Then supervised learning is applied to the ablated network. The object of the test is to determine whether the ablated network can recover the knowledge embodied in the connection weights.

Supervised learning proceeds in two stages. First GDMC is applied to the ablated network. When GDMC gets bogged down, usually after error has been reduced by about 90% on the training set, the resulting network is handed over to ENBP for further training. Use of these two learning methods in this order tends to take advantage of the strengths of each: GDMC is better at long traversals through knowledge space, where ENBP may find the going rough due to the presence of local minima; and ENBP is better at sharp convergence to a global minimum when given a start within the basin of attraction of that minimum. Thus GDMC in effect initializes ENBP, which then converges to a good knowledge state.

Figure 2 shows the learning curve resulting from one such experiment, performed recently by Susan Hruska, David Kuncicky and myself, in which 25 soft connections were set to zero in WA and the resulting ablated network was trained with 22 correct inferences. The graph shows epochs of training (an epoch is one cycle through the training set) versus orders of magnitude of total square error. In this particular run, GDMC reduced error one order of magnitude in 10,575 epochs, after which ENBP reduced error three more orders of magnitude in 183 epochs.[6,7] This level of error can be considered zero for M.1, since only two decimal place accuracy is reported from the system. Thus the training session resulted in a network that correctly inferences on the training set.

A somewhat surprising outcome of such experiments on WA has been that, not only does the process described above produce an expert network that correctly inferences on the original training set, but it recovers the original weights (which, recall, are the original certainty factors on rules). This means that after training, the network exhibits *perfect generalization*: It correctly reproduces all inferences that are possible for the original system to make. For the specific experiment discussed above, training on 22 examples of correct inferences resulted in a system that correctly processes all of the more than 6,900 valid inputs. Of course, in a real application, there would be no a priori "correct" values for rule certainty factors, and the training set would consist of examples of correct inferences from real experts. Nevertheless, this experimental outcome can be taken as evidence that expert networks have good generalization ability. A likely source of this phenomenon is the sparseness of the network. This subject is under investigation.

A full and exhaustive set of learning tests on WA is currently under way, including experimenting with parameter settings and efficient selection of training sets. Complete results will be reported elsewhere.

Expert Network Learning: Further Remarks

An expert network represents knowledge in its synaptic weights. Expert network learning, as discussed above, involves changing the values of these weights, changing the knowledge state of the network. The learning experiments actually begin with "ablation" – setting weights to zero. There is a hidden implication here of a distinction between zero weight on a connection and no connection at all. In the experiments, weights are allowed to grow from zero, but connections are not allowed to emerge from non-existence: the network topology is assumed as a substrate for learning.

The connection topology clearly also represents knowledge, however. When an expert network is derived from an expert system, as in Section 4, the network topology comes from the rule structure while the weights come from rule certainty factors. A new rule, including certainty factor, can be learned with the methods here, but only if the connection is made in some sense.

In an EMYCIN system, a rule consists of an implication and an associated confidence factor. Call a rule with zero weight a *virtual rule* (or, in an expert network, a *virtual connection*). What is the effect of adding or deleting a virtual rule in an ES (or a virtual connection in an EN)? A virtual rule has zero output value no matter what its input. Thus addition (or deletion) of a virtual rule would leave the inferential dynamics of the system unchanged: there would be no difference in functionality after insertion/deletion of a virtual rule. In some sense, then, a virtual rule does not represent knowledge.

On the other hand, if the certainty factor of a virtual rule is changed (through the learning process, as discussed above) then the virtual rule begins to affect the results of inferencing. The system could well learn to inference correctly with the new connection where such was impossible without it (or discover that proper inferencing can be learned without the deleted connection, thus removing redundancy from the rule base). Thus a virtual rule represents *potential* new knowledge that can only be realized through refinement: a potential for learning.

Virtual rules may be much easier to extract from experts than real rules, especially rules quantified with certainty factors. A virtual rule is essentially a statement like "a may have something to do with outcome b", which is more likely to be enunciated than a statement like "a implies b with certainty 0.6". Thus a virtual rule structure represents a kind of "first draft" of expertise. In some cases, virtual rules may be obvious even to a novice using a textbook. Once a set of virtual rules is found, the expert network can be trained as above.

At other times, a virtual rule structure may be inherent or generic to a class of inference problems that vary only in their fine knowledge structure. Examples of such situations include (1) medical advice systems for chronic diseases where treatment follows a well-known set of general guidelines but varies in subtle ways with individual patients and (2) control systems that must adapt to environments that are as yet unencountered. In the latter case, the virtual rule structure is derived from properties of the device, say an unmanned exploration vehicle, and general facts about the environment. In (1), the virtual rules come from the "textbook" treatment of the generic patient. My group is currently investigating the use expert network technology as adaptive advisors for treatment of diabetes.

Finally, (virtual) rules are the product of many classical machine learning techniques as well as some new connectionist learning mechanisms ([Fu and Fu (1990)], [Hall and Romaniuk (1990)]). In fact, in the context of *crisp*[8] rule-based expert systems, "learning" is synonymous with rule creation, modification, or deletion. And constructive connectionist methods such as Cascade Correlation [Fahlman and Lebiere (1990)] build a network topology concomitant to adjusting connection weights. Any or all of these methods can be brought to bear in building an original expert network topology on which to begin learning.

The situation is analagous to learning to drive an automobile. The first draft system is a set of rules obtained by reading a simplistic manual. Actually driving by following these rules is cumbersome, slow, and error prone. Only refinement of these rules through training (practice) produces driving expertise.

6. From Symbolic To Sub-Symbolic Processing

I have discussed expert systems and expert networks and shown how a single system can be viewed as either or both and how this view can facilitate the migration of ideas back and forth between symbolic and connectionist AI. The main example, discussed in detail, was connectionist learning applied to EMYCIN-based expert systems. But clearly there are others in both directions, and similar ideas will work with inference systems other than EMYCIN. The question remains, is this a complete transition from symbolic to connectionist models?

The answer is, not really. An expert network, and more generally a computational network, has the appearance of a connectionist system, complete with artificial "neurons" and "synapses". But the artificiality is perhaps too abstract. In particular, specific meaning can be assigned to the processing of an individual node in an expert network. And although an active concept in the form of an assertion does get distributed across the expert network during inferencing, some analysis shows that this distribution follows predictable pathways based on the syntax of the concept. Thus, though an expert network has some of the attributes of a connectionist system, it is still processing on a symbolic level: It is possible to assign meaning to the input, processing functionality, and output of a single node in the network.

Define an (artificial) *neural network* (NN) to be a computational network with particularly simple processing functionality: linear synapses; node combining functions that simply sum the post-synaptic input; and bounded, non-decreasing output functions. A model based on NN would be a genuine connectionist model. Can an expert network be realized as such a model?

The answer is, generally yes. Consider an EMYCIN network. The synapses are linear, so the symbolic-level node functionality is the main problem. Now, each node in an EMYCIN network can be realized as a self-contained neural network, where the input of the NN is the input for the node, the output of the NN is the output of the node, and the connections into and out of the NN are weighted exactly as into and out of the node. The NN replacing the node may be either a feedforward back-prop network or a recurrent network, at the choice of the builder.[9] The NN learns how to do its processing using standard NN supervised learning methods.

Thus, an expert network can be realized as a NN with two levels of organization: a lower level consisting of networks of sub-symbolic processors self-organized for general conceptual reasoning, all organized at a higher level by interconnections representing inference rules. Call the higher level organization the "EN superstructure" on the NN. This can be visualized for WA by imagining each node in Figure 1 replaced with a small neural net. With this realization, an expert system is transformed into a genuine sub-symbolic connectionist system whose activation dynamics are identical to the inferential dynamics of the ES. The ability to learn domain knowledge is retained in the EN superstructure of the NN.

Note that there are now two kinds of knowledge in the NN model. At the symbolic level is the domain knowledge embodied in the EN superstructure. At the sub-symbolic level is metaknowledge: the NN components have captured knowledge about how to reason. There are two learning processes also. The system learns how to reason by training its NN components and learns domain knowledge by training the EN superstructure. The EN/NN two-level organization of the connectionist expert systems discussed here is a key distinguishing feature from previous work such as [Gallant (1988)], [Fu and Fu (1990)], and [Towell, Shavlik, and Noordewier (1990)] where an emphasis is placed on regularity of the NN connection topology (e.g., a regular layer structure for backprop nets) either for convenience or because regularity is a normal requirement of standard learning mechanisms. Our approach uses knowledge to guide the architectural decisions – domain knowledge for the superstructure and metaknowledge for the substructure. The two levels of organization also allow learning to take place at two levels – domain and meta.

The EN/NN two-level organization discussed here is proven to exist. The proof is "top-down" constructive: First build the EN, and then use approximation theory to replace the symbolic nodes of the EN with densely connected NNs. The principal value of this approach is to show that symbolic processing can be done on a certain kind of sub-symbolic computer and that knowledge can be found (and updated) at distinct levels. The NN structure may also lead to efficient hardware implementations of expert networks using existing neural network hardware chip technology [Eberhardt et al. (1989a,b; 1991)].

A related question, of more interest to cognitive science, is whether a two-level symbolic/ sub-symbolic structure can *self-organize* from a disorganized (or homogeneously organized) NN: Can the EN/NN organization be realized by a "bottom-up" construction. This question is the focus of on-going work [Lacher (1992b)].

7. Conclusion

It is interesting to make some quantitative estimates based on human anatomy. First observe that the two-level organization of the connectionist realization of an expert system generally consists of densely intraconnected subnetworks at the sub-symbolic level and sparse connectivity at the symbolic level. It is a big NN organized as a sparsely interconnected collection of densely intraconnected subnetworks. Assume that the subnetworks each have about k nodes and the NN has a total of n subnetworks. Taking $O(n)$ as sparse and $O(k^2)$ as dense connectivity, the NN has $n \times k$ nodes and about $n \times k^2$ total connections. Taking a crude estimate of about 10^{10} neurons and 10^{13} connections for the human cerebral cortex, straightforward calculation yields $n = 10^7$ and $k = 10^3$: an organization consisting of about 10,000,000 sparsely interconnected subnetworks of about 1,000 densely connected neurons each. Initial investigation indicates that 1,000 sub-symbolic nodes is more than enough to realize any one of the various logical processing networks of an EMYCIN system, with room left over to add small connectionist associative memories that might be used for concept binding during inferencing under a model where the logical subnetworks are re-used across knowledge domains.

It is now well within scientific plausibility to design a complete connectionist reasoning system that learns how to reason and learns domain knowledge. A sparse symbolic superstructure over a dense sub-symbolic substructure may be a key component. Not only does this reflect the organization of real neural networks, it solves such problems as scaling and memory stability that have been associated with the more simple mono-level connectionist models. The system is chameleon-like – it can appear to be either symbolic or sub-symbolic, at the pleasure of the viewer. Perhaps most important, these ideas would remain inactivated had researchers confined themselves to debate on whether one "pure" theory is better than another.

Notes

1. *M.1 Reference Manual* (Software version 2.1), Teknowledge, Palo Alto, CA, 1986.

2. There is an implied conjunction of all the rules in an EMYCIN rule base. In some versions, notably M.1, replacement of a rule of the form 'IF a_1 OR a_2 THEN b (cf)' with the two rules 'IF a_1 THEN b (cf)' and 'IF a_2 THEN b (cf)' is allowed. This identity effectively defines 'OR' in terms of the evidence accumulation rule. Often, there is also an explicit or implicit definition of 'OR' as the maximum operator, giving two different concepts for disjunction of assertions. In any case, using the evidence definition, de Morgan's laws fail in the system. Such systems seem to be able to muddle through rather nicely, however, and almost never bother the human experts who are using them.

3. Continuous time computational networks are also useful. The various local functionalities are specified by differential equations, in the spirit of [Hirsh (1989)], [Kosko (1992)].

4. Without further assumptions, it cannot be concluded that the activation state of the CN will approach equilibrium. In such generality, the network output might be better defined as some appropriate description of the asymptotic dynamics.

5. Another use of *expert network* appears in [Eberhart and Dobbins (1990), Chapter 9 (by M. Caudill)], where the term means a standard neural network of sub-symbolic processors that is trained so as to replace or enhance the expert system.

6. Epochs required is not an accurate method of comparison of the two learning methods, since one epoch of GDMC requires far less computational work than one epoch of ENBP. We ran GDMC on a 386 PC, where several hours were required for this particular experiment. ENBP required about 20 minutes on a Sun sparkstation. There has been little attempt to optimize these programs for speed.

7. Several projects are currently underway to produce faster implementations of these learning algorithms, including parallel versions on the Thinking Machines Corp. CM-2 [Nguyen, et al. (1992)] and distributed versions running on Parallel Virtual Machine (PVM) and networked workstations.

8. That is, without a notion of uncertainty.

9. For backprop nets, these results follow from [Funahashi (1989)]. A complete investigation, including theory and computational experimentation, is under way at FSU.

References

Buchanan, B. G. and Shortliffe, E. H. (1984), *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, Reading, MA.

Churchland, P. M. and Churchland, P. S. (1990), 'Could a machine think?', *Scientific American* **262**, pp. 32-39.

Dagli, C. H. and Stacey, R. (1988), 'A prototype expert system for selecting control charts', *Int. J. Prod. Res.* **26**, pp. 987-996.

Eberhardt, S. P., Duong, T. and Thakoor, A. P. (1989a), 'A VLSI "building block" chip for hardware neural network implementations', *Proceedings Third Annual Parallel Processing Symposium*, vol. I, IEEE Orange County Computer Society, Fullerton, CA, pp. 257-267.

Eberhardt, S. P., Duong, T. and Thakoor, A. P. (1989b), 'Design of parallel hardware neural network systems from custom analog VLSI "building block" chips', *Proceedings IJCNN 89* (Washington, DC), vol. II, IEEE, Piscataway, NJ, pp. 183-190.

Eberhardt, S. P., Daud, T., Kerns, D. A., Brown, T. X. and Thakoor, A. P. (1991), 'Competitive neural architecture for hardware solution to the assignment problem', *Neural Networks* **4**, pp. 431-442.

Eberhart, R. C. and Dobbins, R. W. (1990), *Neural Network PC Tools*, Academic Press, San Diego.

Fahlman, S. E. and Lebiere, C. (1990), 'The Cascade Correlation learning architecture', *Advances in Neural Information Processing Systems* **2** (D. S. Touretzky, ed.), Morgan Kaufmann, New York, pp. 524-532.

Fu, L.-M. and Fu, L.-C. (1990), 'Mapping rule-based systems into neural architecture', *Knowledge Based Systems* **3**, pp. 48-56.

Funahashi, K.-I. (1989), 'On the approximate realization of continuous mappings by neural networks', *Neural Networks* **2**, pp. 183-192.

Gallant, S. I. (1988), 'Connectionist expert systems', *Communications of the Association for Computing Machinery* **24**, pp. 152-169.

Giarratano, J. and Riley, G. (1989), *Expert Systems: Principles and Practice*, PWS-KENT, Boston.

Hall, L. O. and Romaniuk, S. G. (1990), 'FUZZNET: Toward a fuzzy connectionist expert system development tool', *Proceedings IJCNN 90* (Washington, DC), vol. II, pp. 483-486.

Hertz, Krogh, A. and Palmer, R. G. (1991), *Introduction to the Theory of Neural Computation*, Addison-Wesley, New York.

Hirsch, M. W. (1989), 'Convergent activation dynamics in continuous time networks', *Neural Networks* **2**, pp. 331-349.

Hruska, S. I. and Kuncicky, D. C. (1991), 'Application of two-stage learning to an expert network for control chart selection', *Intelligent Engineering Systems Through Artificial Neural Networks* (C. Dagli, S. Kumara, and Y. Shin (eds.), ASME Press, New York, pp. 915-920.

Kosko, B. (1992), *Neural Networks and Fuzzy Systems*, Prentice Hall, Englewood Cliffs, NJ.

Kuncicky, D. C. (1990), 'The transmission of knowledge between neural networks and expert systems', *WNN-AIND 91* (Proceedings of the First Workshop on Neural Networks, Auburn University), pp. 311-319.

Kuncicky, D. C. (1991), *Isomorphism of Reasoning Systems with Applications to Autonomous Knowledge Acquisition.* PhD Dissertation (R. C. Lacher, Major Professor), Florida State University, Tallahassee, FL.

Kuncicky, D. C., Hruska, S. I., and Lacher, R. C. (1991), 'Shaping the behavior of neural networks', *WNN-AIND 91* (Proceedings of the Second Workshop on Neural Networks, Auburn University), SPIE Volume 1515, pp. 173-180.

Kuncicky, D. C., Hruska, S. I., and Lacher, R. C. (1992), 'Hybrid Systems: The equivalence of expert system and neural network inference', *International Journal of Expert Systems*, to appear.

Lacher, R. C., Hruska, S. I., and Kuncicky, D. C. (1991), 'Expert networks: a neural network connection to symbolic reasoning systems', *Proceedings FLAIRS 91* (M. B. Fishman, ed.), Florida AI Research Society, St. Petersburg, FL, pp. 12-16.

Lacher, R. C., Hruska, S. I., and Kuncicky, D. C. (1992), 'Backpropagation learning in expert networks', *IEEE Transactions on Neural Networks* **3**, pp. 62-72.

Lacher, R. C. (1992a), 'Node error assignment in expert networks', *Hybrid Architectures for Intelligent Systems* (A. Kandel and G. Langholz, ed), CRC Press, London, pp. 29-48.

Lacher, R. C. (1992b), 'The symbolic/sub-symbolic interface: Hierarchical network organizations for reasoning', *Integrating Neural and Symbolic Processes* (AAAI-92 Workshop, R. Sun, ed.), to appear.

Nguyen, K. D., Gibbs, K. S., Lacher, R. C., and Hruska, S. I. (1992), 'A connection machine based knowledge refinement tool', *FLAIRS 92* (M. B. Fishman, ed.), Florida Artificial Intelligence Research Symposium, St. Petersburg, pp. 283-286.

Rocker, R. R. (1991), *An event-driven approach to artificial neural networks*, Masters Thesis (S. I. Hruska, Major Professor), Florida State University, Tallahassee, FL.

Rumelhart, D. E. and McClelland, J. L. (1986), *Parallel Distributed Processing*, MIT Press, Cambridge, MA.

Searle, J. R. (1990), 'Is the brain's mind a computer program?', *Scientific American* **262**, pp. 26-31.

Shortliffe, E. H. (1976), *Computer-Based Medical Consultations: MYCIN*, Elsevier, New York.

Shortliffe, E. H. and Buchanan, B. G. (1985), 'A model of inexact reasoning in medicine', *Rule-Based Expert Systems*, Addison-Wesley, New York, pp. 233-262.

Towell, G. G., Shavlik, J. W., and Noordewier, M. O. (1990), 'Refinement of approximate domain theories by knowledge-based neural networks', *Proceedings AAAI-90*, Morgan Kaufmann, New York, pp. 861-866.

Werbos, P. (1974), *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD Thesis, Harvard University, Cambridge, MA.

Figure Captions

Figure 1. The wine advisor expert network. Node type is coded as follows: plain fill: REG nodes; circled circles: AND nodes; triangles: UNK nodes; squares: NOT nodes. Created with TONGS software [Rocker (1991)].

Figure 2. Learning curve for a 25-connection ablation of WA using 22 training examples. The scales represent epochs v orders of magnitude of reduction of mean absolute error.

Figure 1. The wine advisor expert network. Node type is coded as follows: plain fill: REG nodes; circled circles: AND nodes; triangles: UNK nodes; squares: NOT nodes. Created with TONGS software [Rocker (1991)].

Figure 2. Learning curve for a 25-connection ablation of WA using 22 training examples. The scales represent epochs v orders of magnitude of reduction of mean absolute error.