4

HIERARCHICAL ARCHITECTURES FOR REASONING

R.C. Lacher and K.D. Nguyen

Department of Computer Science

Florida State University

Tallahassee, FL 32306-4019, USA

1 INTRODUCTION

This chapter has a threefold purpose: (1) to introduce a general framework for parallel/distributed computation, the *computational network*; (2) to expose in detail a symbolic example of a computational network, related to expert systems, called an *expert network*; and (3) to describe and investigate how an expert network can be realized as a neural network possessing a *hierarchical symbolic/sub-symbolic architectural organization.*

A computational network is essentially a directed graph in which each component (vertex or directed edge) has data processing functionality, further endowed with a concept of global network computation. Examples of computational entities that admit descriptions within the computational network model include biological neural networks, artificial neural networks, the parallel virtual machine model of loosely coupled MIMD computation, human collaborations such as committees, and expert networks. Many of the principles of neural network learning can be lifted to the level of computational networks. We present a re-examination of backpropagation learning in this context and derive the computational network backpropagation, or CNBP, learning algorithm.

An expert network is a computational network that can be obtained from an expert system. The architecture of the expert network is derived from the expert system: the network topology from the rule base, the local processing functionality of the vertices and edges from the system of inference, and the global computation concepts from the inference engine. The process of constructing an expert network from an expert system is reversible.

Expert network backpropagation, or ENBP, is a learning method for expert networks obtained as an instantiation of CNBP. ENBP has proven to be useful in knowledge refinement, allowing an expert system builder to make the transition from coarse knowledge, in the form of rough-draft rules, to fine knowledge, in the form of rules with subtlety represented by analog parameters such as certainty factors, using supervised learning and the historical record of expert behavior as a training set.

The symbolic-level nodes of an expert network can be represented by neural networks, which we view as computational networks of sub-symbolic processors. We investigate the optimal architectures for these representations, which provide a realization of an expert network as a neural network with a hierarchical topological organization: a sparsely interconnected collection of densely intraconnected neural nets. This hierarchical sparse/dense organization is analogous to biological neural organization. It captures two levels of knowledge: domain knowledge in the sparse superstructure and metaknowledge in the dense substructures. The hierarchical structural parameters are well within the connectivity constraints found in biology, making feasible the scaling up of neural-based expert networks to sizes comparable to those of living systems.

2 COMPUTATIONAL NETWORKS: A GENERAL SETTING FOR DISTRIBUTED COMPUTATIONS

A computational network is a general framework for parallel/distributed computation modeled on a directed graph in which the vertices and directed edges have computational functionality and for which there is some holistic notion of cooperative computation [32, 34]. Computational paradigms that fit within the computational network framework include biological neural networks; artificial neural networks; distributed computation on a loosely coupled collection of von-neuman machines connected to a digital communications network, as exemplified by Parallel Virtual Machine [52, 53]; human collaborative decision-making and problem-solving; and expert networks [31]. We return briefly to each of these examples after introducing computational network concepts.

2.1 Definitions and Notation

A *computational network* (CN) is a directed graph together with certain attributes and specifications. These may be *local* or *global*, the former referring to individual CN components (vertices or edges) and the latter to the CN itself. In particular, to compute with a CN one must specify the types of data allowed for (various parts of) the computation; the local functionalities associated with digraph components; a method of timekeeping or scheduling to keep the global computation organized; a method of aggregating the local computations into a global network computation; and how data is to be presented to, and retrieved from, the CN.

Data Types

The types of data with which the computational network is competent must be specified. Depending on the setting, allowed data types may be specific molecules, discrete or continuous numerical values, character data, or even sounds that represent either analog data or discrete symbolic information. Different components of the CN may require different data types, and the CN may operate internally with data types distinct from the I/O data types.

Local Functionalities

The components of a computational network must have computational attributes. Thus each vertex of a CN must have an associated ability to receive data at its incoming edges, process that data into an internal state, compute an output value, and make this output value available to each of its outgoing edges. Each directed edge must have an associated ability to receive data at its initial end, compute a value, and make this value available at its terminal end. We use the terms *node* and *connection* to mean, respectively, a vertex or a directed edge in a CN together with its associated functionality.

Node functionality is broken down into two stages, an input or combining stage and an output or firing stage. In the combining stage a node computes an internal state y from its input data values $x_1, \ldots, x_n$. We denote the function so implied by Γ and call it the *combining function* of the node (or associated with the vertex). After computing its internal state y, a node must compute an output value z. We denote this second function by φ and call it the *firing function* of the node.

The node combining functions of a computational network may be specified in a number of ways, either explicitly or implicitly. For example, if time is continuous Γ may be determined implicitly by a differential equation, whereas if time is discrete Γ may be given by an explicit formula.

Connection functionality transforms the data received by a connection into a transmitted signal value. The input to a connection is the output value z of the node at its initial end. The connection computes one of the input values x for the node at its terminal end. We denote the function making this computation by σ and call it the *synaptic function* of the connection.

Commonly encountered synaptic functions may be linear functions; linear threshold functions; sigmoidal functions; or simple conduits that transmit data unchanged except for a possible time delay.

Time

A notion of timekeeping or scheduling of the various component computations and data transactions is required in order to give meaning to whole-network activation and computation. The possibilities for timekeeping range intrinsic such as self-organization to extrinsic such as management by outside expertise.

Global Activation

The local components of a CN are activated by simply applying their functionality to whatever input they have at any given time. For global computation, however, these local activations must be orchestrated in some way to define a notion of global or network activation. Options include: *synchronous* activation, in which each network component is activated simultaneously; *asynchronous* activation, in which network components are activated randomly one at a time; *event-driven* activation, in which a network component activates when one of its input values changes; and *managed* activation, in which components activate on the command of a central scheduler or manager.

Network Computation

A computational network is intended to be used as a computer, and like a traditional computer the computations of the various parts must be orchestrated into a holistic whole-network computation in some way. In all cases, the global network computation is obtained by successive global activations. The cases

differ in how they determine when a network computation is completed. There are two basic choices: either activate for a certain length of time, or activate until the network has reached some kind of global equilibrium state.

Input and Output

A method must be prescribed whereby data values may be introduced into, and retrieved from, a CN from outside the network. For the purposes of this work we will assume that I/O is accomplished by specification of two subsets of nodes (possibly overlapping), "input" nodes and "output" nodes. Data is inserted into the CN to begin a computation by externally setting the states y of the input nodes to the input data values. After network computation is completed, data is retrieved from the CN by reading the outputs z of the output nodes.

2.2 Activation Dynamics

The attributes which collectively define a computational network are not independent. For example, the method of keeping time, the concept of holistic computation, and the meaning of I/O are all interelated, and some choices in one direction may preclude a possibility in another direction. A coherent set of attributes for timekeeping, global activation, network computation, and I/O together constitute the rules for *activation dynamics* of the CN. We consider briefly some of the most often used network computation strategies.

Centrally Managed Computation

Usually used with discrete time, although possible with continuous time. A central entity, such as an operating system or a manager, makes decisions as to timing of local computations and routing of data. Output is read at a time specified by the computation manager.

Synchronous Equilibrium Computation

Used with either discrete or continuous time. This method activates the local functionalities indefinitely at each clock tic (or continuously) until a dynamic equilibrium is reached. Output is read at equilibrium. Classically this equilibrium is assumed to be a fixed point in the space of node states, but more

general attractors are sometimes allowed [21, 10, 39, 49]. It may be quite challenging to decide whether appropriate equilibria are always attained in a given CN [19]. Virtually all continuous-time CNs use synchronous activation, and most use equilibrium dynamics to define network computation.

Fixed Time Delay Computation

Used with either discrete or continuous time. The network is activated as in synchronous activation above, but for a certain number of iterations (or length of time) after which output is retrieved. This is often used in place of an equilibrium rule. The time of activation is chosen so that the network will be close to equilibrium upon completion.

Asynchronous Equilibrium Computation

This makes sense only for discrete time. There are two variations, a global one in which a node is chosen at random and its incoming connections and the node itself are activated, and local one in which each component chooses to activate at random times. In either case the process continues until equilibrium is reached. When the probability of local activation is kept small, these produce equivalent equilibrium dynamics [18, 20].

Event Driven Computation

Again for discrete time only. Each component of the CN activates whenever it receives a new input value, until no values change. This is equivalent to synchronous equilibrium dynamics [32]. Expert networks and human collaborations typically use event-driven activation, and results are generally useful only when an equilibrium state is achieved.

2.3 Examples

In a biological neural network (BNN), the local functionalities are determined by the extraordinarily complex biochemical processes of synaptic transmission, membrane channels, and internal cell chemistry. The synaptic functions reflect the type and density of transmitter molecules, together with properties of the inbound membrane channels of the receiving cell. The combining and firing

functions reflect the internal cell biochemical accrual process and the sensitivity and other properties of the outbound membrane channels, respectively.

An artificial neural network (NN) is a mathematical analogy of a BNN. The synaptic and firing functions are usually specified explicitly. Quite typically, the synapses are simple linear functions. The firing functions may be of virtually any type, but most often are sigmoidals such as logistic or hyperbolic tangents, threshold functions (with discrete output), symmetric distributions such as the gaussian, or some combination of these types. In discrete time NNs, the combining function is usually given explicitly, with simple additive accrual being the most common, while in continuous time NNs Γ is more often given implicitly by constraints on its derivatives.

A typical use of parallel virtual machine (PVM) is to perform a computation by parceling out identifiable sub-computations to various computers on a high speed communications network. The synaptic functions are pure transmissions of data, with some small time delay. The node functionalities are quite complex and determined by user programs. Global activation is event-driven and network computation is centrally managed.

In human collaborations (HC) the synaptic functions transmitting human-human communication are again simple conduits, albeit of very complex data. The combining functions reflect the receiving and interpretation by one person of the information supplied by all the others in the collaboration. The firing functions reflect the formulation and transmission of personal information and conclusions out to other members of the collaboration. Activation dynamics can be a form of managed computation, for example when there is a strong leader such as a teacher or supervisor. Often more effective is the committee model, with event-driven activation. There is no guarantee of convergence; convergence is a goal of the collaboration.

An *expert network* (EN) is a computational network derived from a rule-based expert system (production system). The digraph topology is determined by the domain rule base; the local functionalities are determined by the inference system; and the timing mechanism is derived from the computational scheduling method of the expert system shell. ENs typically use discrete time, have acyclic topology, and process analog data. Expert networks are discussed further in following sections.

Table 1 Classification of example CNs.

	BNN	NN	PVM	HC	EN
DD/AD	1	x	0	x	1
DT/CT	1	x	0	0	0
AT/RT	1	x	1	1	0

2.4 CN Classification

There are three broad dichotomies that occur very naturally in the specification of a CN. The five examples discussed briefly above give evidence that the resulting categories are non-vacuous and interesting. These dichotomies, and some notation we will use for the resulting classification, are as follows:

- DD/AD: Discrete or Analog Data
- DT/CT: Discrete or Continuous Time
- AT/RT: Acyclic or Recurrent Topology

Only when compactness of notation is convenient, we use a 3-digit binary encoding to represent a set of choices in these three dichotomies, the left digit representing data type, the middle digit representing time type, and the right digit representing topology type. We also use 'x' as a don't-care or union of types. For example, A CN of type 101 computes with analog data using discrete time and a recurrent topology, while type 10x has the same data and time restrictions but does not specify whether the topology is acyclic or not.

Classification of the five examples discussed above is given in Table 1. Some of the classification choices are arguable, but most will agree that these choices indicate a legitimate point of view within which the paradigm may be studied and that some choice must be made in order to focus the study.

2.5 Discrete Time Computational Networks: Notation

We establish some notation for updating discrete time (or "type x0x") CNs. Similar notation is appropriate for continuous time CNs except that often some of the local functionalities are specified implicitly through differential equations.

A CN consists of nodes and connections organized into a directed graph structure. We will use an adjacency matrix notation system for the CN components based on 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" for the edge subscripts: 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. We also use z to denote node output (or activation value) and y to denote node internal state. If the ji synapse is linear, then $\sigma_{ji}(z_i) = w_{ji}z_i$, where w_{ji} is the weight of the connection. We assume in this discussion that the node labels constitute an enumeration $1, \ldots, n$.

The *internal state* of the CN at a particular time t consists of all the node states $y_j(t)$ usually collected into a vector $\mathbf{y}(t) = (y_1(t), \ldots, y_n(t))$. Similarly, the *activation state* of the CN at time t is the vector $\mathbf{z}(t) = (z_1(t), \ldots, z_n(t))$ of local activation values at time t. (It should be kept in mind that important properties of these state vectors are symmetric, that is, independent of the particular vertex ordering.) New states are calculated using the *update equations*

$$x_{ji} := \sigma_{ji}(z_i) \quad \text{for} \quad i = 1, \ldots, n \tag{4.1a}$$

$$y_j := \Gamma_j(x_{j1}, \ldots, x_{jn}) \tag{4.1b}$$

$$z_j := \varphi_j(y_j) \tag{4.1c}$$

during three time steps (or in one time step split into three sub-steps). How local updates are organized into network activation varies as discussed earlier. Activation dynamics is the study of the behavior of network states as they change over time.

One requirement not often made explicit for computational networks is *symmetry* of combining functions: Γ_j should give output that is independent of the labeling order of the nodes. Another system of notation that makes this requirement more obvious is based on predecessor/successor relations in the network topology. Define a *predecessor* of node j to be any node in the network that initiates a connection into j. The set of predecessors of j is defined and denoted as

$$\text{Pred}(j) = \{i | \text{there is an edge from } i \text{ to } j\}.$$

The update equations can be restated in terms of predecessors as follows. First compute post-synaptic input for node j:

$$x_{ji} := \sigma_{ji}(z_i)$$

for all $i \in \text{Pred}(j)$, where σ_{ji} is the synaptic function of the connection from i to j. Denote the vector of all post-synaptic input for node j by $\mathbf{x}_j$. This vector has dimension $|\text{Pred}(j)|$, one component x_{ji} for each $i \in \text{Pred}(j)$, but the order of components is not important. Next compute the internal state of node j:

$$y_j := \Gamma_j(\mathbf{x}_j)$$

where Γ_j is the combining function for node j. Γ_j is a symmetric function of $|\text{Pred}(j)|$ variables. Finally compute the activation value of node j:

$$z_j := \varphi_j(y_j)$$

where φ_j is the output function of node j.

We will generally stick to the simpler adjacency matrix notation of equations 4.1. This simplicity does blur certain subtleties, however, by making the tacit assumption that computation doesn't need to distinguish between no connection from i to j and a connection from i to j with $\sigma_{ji} \equiv 0$. Cases can arise where this distinction is important. In such cases the missing connectivity information can be maintained in a seperate adjacency matrix. If the network is sparse, it may be appropriate to use more compact representations such as adjacency lists that implement the predecessor/successor notation.

3 TYPE x00 COMPUTATIONAL NETWORKS

For the remainder of this chapter we restrict our attention to computational networks of type x00, that is, we assume discrete time and acyclic topology but allow either discrete or analog data types both internally and as I/O.

3.1 Activation Dynamics

Activation dynamics of acyclic, discrete-time computational networks may assume any of the forms discussed in Section 2.2. Synchronous, asynchronous, and event-driven activation are all equivalent to fixed time delay (if the delay is appropriately large) and all result in reaching a terminal activation state in finite time [32]. In other words, given a CN of type x00, we can activate using any of these methods for a globally fixed amount of time, after which activation will cease to produce changes in any of the internal states of the CN. This

activation may be component-parallel, component-distributed, or component-serial. The type x00 CN thus becomes a (parallel/distributed) computer with a fixed number of local computations required for I/O: insert input, compute a fixed number of local activations, then retrieve output. The local computations consist of applying the update assignment statements given by equations 4.1 until a steady activation state $(z_1, \ldots, z_n)$ is reached. We call this the *terminal activation state* of the network and refer to the component z_j as the terminal activation value of node j.

3.2 Influence and Error

Backpropagation is one of the most widely known and successfully used connectionist learning methods [55, 42]. Most often, backpropagation is applied to layered feedforward computational networks with the kind of simple processing functionality associated with low-level, sub-symbolic networks: linear synapses, additive combining functions, and sigmoidal or gaussian output functions. Many of the ingredients of backpropagation learning can be generalized for general computational networks. For CNs, the standard algorithm requires two changes: localize forward and backward activation to free the algorithm of the layer structure, and decouple the process of node error assignment from the weight correction step. The first is described previously and in [36]. The second uses the concept of influence factor, introduced in [32]. Influence factors are associated with connections and specific network input. The *influence factor* ε_{kj} of the connection from node j to node k is the rate of change of output of node k with respect to the output of node j, evaluated at the terminal activation state: $\varepsilon_{kj} = \partial z_k / \partial z_j(z_j)$. Expanding this derivative using the chain rule we obtain

$$\varepsilon_{kj} = \varphi_k'(y_k) \times \frac{\partial \Gamma_k}{\partial x_{kj}}(x_{k1}, \ldots, x_{kn}) \times \sigma_{kj}'(z_j). \tag{4.2}$$

Again we emphasize that 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. This reverse activation is in essence an activation of the reverse or dual of the CN. The dual network topology consists

of the vertices and edges of the original network, but with all edge orientations reversed. The nodes of the dual network use summation combining function and the identity activation function, i.e., the nodes are linear units. The synaptic functions are also linear with weight equal to the influence factor. Note that the dual network is also acyclic.

Error is assigned to each of the output nodes using equation 4.3a, where I is ideal output, and to all non-output nodes using equation 4.3b:

$$e_j := I_j - \zeta_j, \tag{4.3a}$$

$$e_j := \sum_k \varepsilon_{kj} e_k. \tag{4.3b}$$

Applying equation 4.3b recursively is an activation of the dual network with input given by 4.3a. The resulting terminal dual activation state is an error assignment throughout the network. The error assignment process works in any acyclic CN for which the derivatives of equation 4.2 are defined.

3.3 Local Gradient Descent

Once error has been distributed among the nodes in a computational network, we can apply gradient descent learning both *selectively* and *locally* to any node whose incoming synapses are linear. This decoupling of error assignment and learning means we can allow more complex synaptic functionality into certain nodes, we can have hard-wired connections into perhaps other selected nodes, and suppress learning at any selection of sites, while maintaining a global learning process. Local gradient descent amounts to applying the Widrow-Hoff delta rule using local error.

Calculating the gradient of squared local error at node j with respect to synaptic weights $w_{j1}, \ldots, w_{jn}$ and taking a step in the opposite direction yields the following learning rule:

$$\Delta w_{ji} = \eta e_j \varphi_j'(y_j) \frac{\partial \Gamma_j}{\partial x_{ji}}(x_{j1}, \ldots, x_{jn}) z_i + \mu \Delta w_{ji}^{prev}. \tag{4.4}$$

This equation defines one learning step with *learning rate* η and *momentum* μ in the direction of steepest descent of square local error.

3.4 CNBP

Putting all these components together results in a learning method called Computational Network BackPropagation, or CNBP. CNBP applies to type x00 CN at any nodes with linear incoming synapses. All that is required to complete an implementation of CNBP is calculation of the derivatives appearing in equations 4.2 and 4.4. CNBP is summarized as follows.

Assume given a set of training exemplars $(\xi^l, \mathbf{I}^l), l = 1, 2, \ldots$ consisting of input $\xi^l = (\xi^l_1, \ldots, \xi^l_m)$ and ideal output $\mathbf{I}^l = (I^l_1, \ldots, I^l_n)$. The basic learning process goes as follows:

Initialize
    present ξ^l to input nodes
Activate
    calculate terminal activation state z^l_j for each node
    calculate influence factors ε_{kj} for each connection
Initialize error
    present external error $e^l_k = I^l_k - z^l_k$ to output nodes
Reverse activate
    activate the dual network, assigning error e^l_j to each node
Learn
    change soft weights using local gradient descent

The learn step can be carried out after each exemplar presentation (on-line learning) or accumulated and carried out at the end of an epoch (batch learning). The entire procedure loops until error is reduced sufficiently.

4 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 [15]. 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.

Rule-based expert systems have become a mature advanced technology, with many successful software shells on the market, whether success is measured by

technical achievement or commercial viability. Three of these are particularly pertinent to the research, development, and production discussed here: M-4[1], CLIPS[2], and G2[3]. These products are all in significant use in a wide variety of application domains by a heterogeneous user community. Commercial users of M-4 have valuable (and proprietary) rule bases ranging in size from a few dozen to ten thousand rules.[4] CLIPS has an avid following despite its lack of user amenities, due in part to its low cost. G2 is the most elaborate (and costly) of the three, with commercial site licenses listing at $42,000. Many of the worlds largest corporations have signed with Gensym for developing their real-time expert system needs, including ASEA Brown Boveri, GE, Monsanto, Occidental Petroleum, Boeing, Du Pont, Texaco, Lafarge Coppee, and 3M [17].

These three shells each deal with uncertainty using a form of EMYCIN logical semantics. M.4 is discussed in some detail below; Hruska and coworkers have constructed a superset of CLIPS that uses essentially the same uncertainty semantics as M.4 [43]; and G2 uses a classical version of fuzzy inference.

4.1 EMYCIN

A seminal demonstration of the efficacy of rule-based systems was a medical diagnosis and treatment advisory system for infectious diseases called MYCIN[46, 1]. A natural consequence of the success of MYCIN was its abstraction to an "expert system shell" in order to apply the same reasoning automation in other domains. A shell is just an expert system with an empty knowledge base and a user interface system to facilitate the insertion and modification of rules. A shell that implements the MYCIN reasoning system is called EMYCIN (for "Empty MYCIN"). M.4 is a commercially available EMYCIN shell. The computational experiments discussed below are based on M.4. The features of EMYCIN inferencing that are important in what follows are the evidence accumulator and the various logical operations [47, 14].

A rule in EMYCIN has the form

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

where a and b are assertions and cf is a *certainty factor* or confidence factor associated with the rule. The certainty factor may take on any value in the range

[1] Product of Cimflex Teknowledge Corporation.
[2] Designed and Produced by NASA, distributed as shareware.
[3] Product of Gensym Corporation.
[4] Private communication from representative of Cimflex Teknowledge

$-1 \leq cf \leq 1$. We use the notation $cf_{b|a}$ to denote the certainty value of the implication IF a THEN b. Certainty factors are static numerical attributes of rules. They reside in the knowledge base and do not change during inferencing.

An assertion b may take on an *evidence value* (also sometimes called a certainty factor). The evidence value of an assertion is dynamically updated during inferencing, either through assignment when a query is made or through calculation in terms of evidence values of other assertions previously calculated or assigned during the inference. We denote the evidence value of assertion b by y_b. y_b may range in the interval $-1 \leq y \leq 1$. 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 (in this version of EMYCIN) is restricted to the range $0 \leq z \leq 1$.

Suppose that we have a current dynamic evidence value y_b for assertion b and subsequently encounter another assertion IF a THEN b (cf). Then EMYCIN adjusts y_b by adding an amount proportional to the firing value z_a for a, the certainty factor $cf = cf_{b|a}$ for the rule, and the proximity of y_b to its domain limits. (When the current evidence value y_b and the rule certainty factor $cf_{b|a}$ have opposite signs, a mediation process is used instead.) The output value z_b for assertion b is then updated by applying the firing criterion to y_b. The firing criterion may vary somewhat from one EMYCIN shell to another. M.4 uses the linear-threshold firing function with threshold value of 0.2.

This update process breaks naturally into three steps. First calculate the certainty-mediated input evidence:

$$x_{b|a} := cf_{b|a} \times z_a; \tag{4.5}$$

then update the evidence value:

$$y_b^{new} := \begin{cases} y_b + x_{b|a}(1 - y_b) \ , & \text{if both } y_b \text{ and } x_{b|a} \text{ are positive,} \\ y_b + x_{b|a}(1 + y_b) \ , & \text{if both } y_b \text{ and } x_{b|a} \text{ are negative,} \\ \frac{x_{b|a} + y_b}{1 - \min\{|y_b|, |x_{b|a}|\}} \ , & \text{otherwise;} \end{cases} \tag{4.6}$$

then recalculate the firing value:

$$z_b := \begin{cases} y_b \ , & \text{if } y_b \geq 0.2; \\ 0 \ , & \text{otherwise.} \end{cases} \tag{4.7}$$

This firing value is then used as input to other rules of the form IF b THEN c (cf), and so on, until all firing values are stabilized. 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.

The reader will probably have noticed the similarity between the equations above and equations 4.1 as well as a principal distinction: 4.6 represents an accumulation process over rules with b as consequent, while 4.1b represents the evaluation of a combining function over all incoming connections simultaneously. We give a closed form version of 4.6 in the next section.

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. 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.4 recognizes three logical operations explicitly: AND, NOT, and UNK. The UNK (for "unknown") operation is a version of NOR (NOT following OR).[5] For AND, M.4 uses the same firing function as for evidence combining, given above by 4.7. For NOT, M.4 uses a firing function that is a strict threshold, with threshold value 0.8, resulting in discrete values for NOT and NOR operations.

Each of the operations can be described in three functional steps analogous to 4.5, 4.6, and 4.7 above. These operations, along with the evidence accumulation process, provide functionality to the vertices and edges of an inference network model of the knowledge base, resulting in a computational network. We describe this network, along with explicit M.4 functionalities, in detail in the next section.

5 EXPERT NETWORKS

Expert Network learning technology, a process developed by a group at FSU[6] in partnership with the Florida High Technology and Industry Council, pro-

[5] M.4 does not recognize an explicit OR operation, hence the non-standard terminology. M.4 implicitly uses two different versions of OR – the DeMorgan dual of AND as well as the evidence accumulator.

[6] Lacher, Hruska, and Kuncicky

vides a means of automated knowledge refinement in rule-based expert systems. In settings where sufficient historical data exists, expert network learning can significantly improve both the development time and the ultimate level of expertise captured in an expert system project.

The expert network method, at the algorithm level, is a method for knowledge refinement in a rule-based expert system that uses uncertainty. The uncertainty theory can be that of EMYCIN certainty factors as in M-4, fuzzy logic as in G2, probability, Dempster-Shaffer theory, or any other theory that uses a continuously variable value or values to define a level or degree of certainty to implications and/or factual statements. In all such systems the role of uncertainty is to represent the subtle variations of knowledge that, once discovered and captured, complete the transition from coarse novice-level knowledge to refined expertise.

Expert networks allow these systems to make this passage from novice to expert through neural network style learning from data rather than from laborious human expert tinkering. The data required may be either historical records of correct inferences, in which case the learning methods are supervised, particularly Expert Network BackPropagation (ENBP); or the data may be in the form of critique of the expert system's conclusions by experts, in which case the learning methods are reinforcement methods such as Expert Network Temporal Difference (ENTD(λ)). The critical technology implementing both of these learning methods is that of influence factors.

The expert network, or ExNet, technology consists of two major components: Translation and Learning.

Translation

The rule base is translated into a directed graph. The vertices of this digraph represent atomic-level factual statements or actions; these are the antecedents and consequents of the rules. The directed edges represent implications.

The logical semantics, or rules of inference, of the expert system, including the rules dealing with uncertainty, are used to assign information processing functionality to the vertices and edges. Thus the digraph becomes a computational network. This is called the *expert network* associated with the original expert system.

After the expert network has been modified during the learning phase (described below), the modified expert network is translated back into expert system form, resulting in a new, or refined, set of rules that have optimized performance with respect to the training data. This step requires nothing more than applying the translation process in reverse.

Learning

Neural network learning methods are applied to the expert network. This learning process results in changes in the parameter values for the uncertainties in the rules, optimized for set of correct inference instances data set (i.e., history). There are several difficult problems to overcome to make this idea actually work, including how to assign a local error to the nodes and how to reduce this local error through gradient descent. We have worked out and implemented all details of this idea in the case of EMYCIN (M-4) and for fuzzy inference. The solutions are detailed in the papers [31, 32, 34, 37]. When the expert system uses EMYCIN certainty factors and/or fuzzy logic to capture uncertainty, ExNet has been completely derived, proved, tested, and covered with patents (pending). In the following treatment we restrict to the M.4 instantiation of EMYCIN.

5.1 Translation

The network topology is constructed in two stages. First an inference network is created from the rule base. Each vertex in this network represents an 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

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

where a and b are assertions and $cf \equiv cf_{b|a}$ is the certainty or confidence factor, defines a connection

$$a \xrightarrow{cf} b.$$

At this point we have constructed an inference net in the usual sense (see [15], page 237).

The evidence accumulation process (equations 4.5, 4.6, and 4.7) of the inference engine defines functionality for the vertices of this inference net, and the edges process initial to terminal value by multiplication by cf (defining linear synaptic functions). 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* or *evidence* nodes and denoted as REG nodes.

The second stage of construction is to expand each regular node that represents a compound antecedent statement into a subnetwork. A regular node antecedent such as in the connection

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

expands to the subnetwork

$$\begin{array}{lll} a_1 & \xrightarrow{1} & \mathrm{OP} \\ . & & . \quad . \\ a_k & \xrightarrow{1} & \mathrm{OP} \\ \mathrm{OP} & \xrightarrow{cf} & b. \end{array}$$

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 for an OP node performs the logical computation defined by the rules of inference used by the expert system. The output function for an OP node is the firing condition for the logical operation. The resulting computational network is the second order expert network defined by the expert system.

Note that there are two kinds of nodes in the second order expert network: REG nodes representing assertions and OP nodes representing logical operations. Note also that all synaptic functions are linear with weights as already described above: soft connections (incoming to REG nodes) have weight cf and hard connections (incoming to OP) have weight 1. Thus synaptic functionality is completely specified. We now give more detailed descriptions of the node functionalities in EMYCIN/M.4 expert networks.

REG *Nodes*

The EMYCIN evidence accumulator given by equation 4.6 can be written in closed form. Let b be a REG node, and suppose b has at least one predecessor in the expert network. (In the parlance of expert systems, b is an assertion that is consequent to at least one other assertion.) For each predecessor a of b let $x_{a|b}$ denote the corresponding post-synaptic input $cf_{b|a} \times z_a$.

The positive and negative evidence values for regular node b are given by

$$y_b^+ = +1 - \prod_{x_{b|a}>0} (1 - x_{b|a}) \quad \text{and} \tag{4.8a}$$

$$y_b^- = -1 + \prod_{x_{b|a}<0} (1 + x_{b|a}), \tag{4.8b}$$

respectively. Positive and negative evidence are then reconciled, yielding the internal state of the node as the value of the REG combining function:

$$y_b := \Gamma_{\text{REG}}(x_{b|1}, \ldots, x_{b|n}) \equiv \frac{y_b^+ + y_b^-}{1 - \min\{y_b^+, -y_b^-\}}. \tag{4.9}$$

Note that Γ_{REG} is a symmetric function. The only input variables which affect the values of Γ_{REG} are those labeled by predecessors of b, and we could use alternative notation (as described in section 2) to reflect this fact. The notation above assumes that $x_{b|a} = 0$ whenever a is not a predecessor of b.

The output function φ_{REG} for a regular node b is the firing function for assertions defined by equation 4.7:

$$z_b := \varphi_{\text{REG}}(y_b) \equiv \begin{cases} y_b\ , & \text{if } y_b \geq 0.2; \\ 0\ , & \text{otherwise.} \end{cases} \tag{4.10}$$

OP *Nodes*

Consider an AND node generated by the antecedent of the rule

$$\text{IF} \quad a_1 \text{ AND } a_2 \text{ AND } \ldots \text{ AND } a_k \quad \text{THEN} \quad b \quad (cf)$$

for some assertions (nodes) $a_1, \ldots, a_k$ in the expert net. Let a denote the compound antecedent $\text{AND}(a_1, \ldots, a_k)$. Thus a is an OP node in the second order network. To define the logical AND operation as a function of dynamic evidence values is to define the combining and firing functions of a.

The combining function for a is given by

$$y_a := \Gamma_{\text{AND}}(x_1, \ldots, x_k) \equiv \min_i \{x_i\} \tag{4.11}$$

where $x_i = z_{a_i}$ is post-synaptic input. The output function is the same threshold function used for REG nodes:

$$z_a := \varphi_{\text{AND}}(y_a) \equiv \begin{cases} y_a\ , & \text{if } y_a \geq 0.2; \\ 0\ , & \text{otherwise.} \end{cases} \tag{4.12}$$

A NOT node such as generated by the antecedent of

$$\text{IF} \quad \text{NOT } a \quad \text{THEN} \quad b \quad (cf)$$

has only a single incoming connection, from a. The combining function is given by

$$y := \Gamma_{\text{NOT}}(x) \equiv 1 - x \tag{4.13}$$

(where $x = z_a$) and the output function is

$$z := \varphi_{\text{NOT}}(y) \equiv \begin{cases} 1\ , & \text{if } y \geq 0.8; \\ 0\ , & \text{otherwise.} \end{cases} \tag{4.14}$$

An UNK node may be generated by the antecedent of a rule such as

$$\text{IF} \quad a_1 \text{ UNK } a_2 \text{ UNK } \ldots \text{ UNK } a_k \quad \text{THEN} \quad b \quad (cf)$$

for some assertions (nodes) $a_1, \ldots, a_k$ in the expert net. Let a denote the compound antecedent $\text{UNK}(a_1, \ldots, a_k)$. The combining function for a is given by

$$y_a := \Gamma_{\text{UNK}}(x_1, \ldots, x_k) \equiv 1 - \max_i \{x_i\} \tag{4.15}$$

where $x_i = z_{a_i}$ is post-synaptic input. The output function is the same as for NOT:

$$z_a := \varphi_{\text{UNK}}(y_a) \equiv \begin{cases} 1\ , & \text{if } y_a \geq 0.8; \\ 0\ , & \text{otherwise.} \end{cases} \tag{4.16}$$

Notwithstanding the fact that M.1 does not explicitly acknowledge an OR operation, we could define an OR node that might be generated by the antecedent of a rule such as

$$\text{IF} \quad a_1 \text{ OR } a_2 \text{ OR } \ldots \text{ OR } a_k \quad \text{THEN} \quad b \quad (cf)$$

for some assertions (nodes) $a_1, \ldots, a_k$ in the expert net. As usual letting a denote the compound antecedent $\mathrm{OR}(a_1, \ldots, a_k)$, we define the combining function for a to be

$$y_a := \Gamma_{\mathrm{OR}}(x_1, \ldots, x_k) \equiv \max_i \{x_i\} \tag{4.17}$$

where $x_i = z_{a_i}$ is post-synaptic input and the output function to be the same as for AND:

$$z_a := \varphi_{\mathrm{OR}}(y_a) \equiv \begin{cases} y_a\ , & \text{if } y_a \geq 0.2; \\ 0\ , & \text{otherwise.} \end{cases} \tag{4.18}$$

Given this definition of OR, it is easily verified that UNK = NOT(OR).

Logical Functions

Composing appropriate functions given above yields the following throughput functions (from input to firing value) for logical operations in M.4:

$$\mathrm{AND}(x_1, \ldots, x_k) := \begin{cases} \min\{x_i\}\ , & \text{if } \min\{x_i\} \geq 0.2, \\ 0\ , & \text{otherwise;} \end{cases} \tag{4.19a}$$

$$\mathrm{NOT}(x) := \begin{cases} 1\ , & \text{if } x \leq 0.2, \\ 0\ , & \text{otherwise;} \end{cases} \tag{4.19b}$$

$$\mathrm{UNK}(x_1, \ldots, x_k) := \begin{cases} 1\ , & \text{if } \max\{x_i\} \leq 0.2, \\ 0\ , & \text{otherwise;} \end{cases} \tag{4.19c}$$

where as usual x is interpreted as post-synaptic input for the node (or current evidence value during inference).

5.2 Learning

An EMYCIN expert network satisfies all the requirements for 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 equations 4.2 and 4.4, 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. If we can calculate (or "define"[7]) the partial derivatives of the node combining functions then we can implement CNBP in expert networks.

[7]The CNBP learning algorithm is sufficiently robust to accommodate approximations. Thus if an approximate derivative can be devised it may work as well as a real derivative.

For a REG node k the partial derivatives are given by

$$\frac{\partial \Gamma_{\text{REG}}}{\partial x_{kj}}(\mathbf{x}_k) = \begin{cases} \frac{1}{1-x_{k|j}} \frac{1-y_k^+}{1+y_k^-}, & \text{if} y_k^+ \geq |y_k^-| \text{ and } x_{kj} > 0; \\ \frac{1}{1-x_{k|j}} \frac{1+y_k^-}{1-y_k^+}, & \text{if } y_k^+ < |y_k^-| \text{ and } x_{kj} > 0; \\ \frac{1}{1+x_{k|j}} \frac{1-y_k^+}{1+y_k^-}, & \text{if } y_k^+ \geq |y_k^-| \text{ and } x_{kj} < 0; \\ \frac{1}{1+x_{k|j}} \frac{1+y_k^-}{1-y_k^+}, & \text{if } y_k^+ < |y_k^-| \text{ and } x_{kj} < 0 \end{cases} \tag{4.20}$$

provided $x_{k|j} \neq \pm 1$. Here $x_{k|j}$ is $cf_{k|j} \times z_j$, the post-synaptic input to node k from node j, and $y_k^{\pm}$ is given by equation 4.8. (See [36] for details.)

For AND nodes we have

$$\frac{\partial \Gamma_{\text{AND}}}{\partial x_{kj}} = \begin{cases} 1 & , \text{ if } x_{k|j} = \min_i \{x_{k|i}\} \\ 0 & , \text{ otherwise} \end{cases} \tag{4.21}$$

It is interesting to examine what this means for reverse error assignment: the AND node assigns error backward through node k acting as a demultiplexer switch to the line with lowest incoming value.

Similar results hold for NOT and UNK nodes.

5.3 ENBP

Having calculated the derivatives appearing in equations 4.2 and 4.4, we can apply CNBP in the context of expert networks. This instantiation of CNBP is called Expert Network BackPropagation, or ENBP.

ENBP has been tested on several M.4-based expert systems, including the Wine Advisor [9] and the Control Chart Selection Advisor of Dagli and Stacey [3, 23, 24]. A functioning expert system is used to define expert knowledge by generating specific examples of correct reasoning. In *ablation* testing, a set of soft connections is ablated by setting their connections weights to zero. In *refinement* testing, all of the soft connections are initialized to the neutral value 0.5. The object of the tests is to determine whether the network can recover the knowledge embodied in the connection weights.

In these tests, both learning and generalization have worked remarkably well. The algorithms converge the ablated system to a knowledge state that correctly inferences on the training set, and generalization is perfect: the new system

reasons correctly on all possible inputs. Moreover, the ratio of training set size to test set size is small. For example, as few as 22 correct inferences are required to move a 25-connection ablation of wine advisor (a 97 node expert network) to a system that inferences correctly on all 6,912 sensible input queries [34]. Refinement tests have yielded 95–100% generalization rates using training sets of 40 or more exemplars [9].

6 NEURAL NETWORKS

By a neural network (NN) we mean a discrete-time computational network with linear synapses, linear combining functions, and non-decreasing firing functions. An expert network is a symbolic computer. The individual nodes have externally assigned and understood meaning – either assertion or logical operation – and the dynamically computed and transported values also have external meaning – degrees of certainty in a conclusion. A neural net, in contrast, is a sub-symbolic computer. The individual nodes and values have external meaning only collectively and selectively. In most cases, an individual node in a NN has no identifiable meaning to an outside observer.

It has been argued that an expert network can be realized as a neural network by replacing each node in the EN with a small NN [34]. We present here some results on the practicality of that process.

6.1 Optimal Architectures

We are interested in finding the "optimal" sub-symbolic NN to replace a symbolic node in an expert net. We consider here two nodes types: REG and AND. We restrict our investigation to the class of layered feedforward networks with one hidden layer and sigmoidal output functions, and we use standard backpropagation to train these networks. (See [50, 51] for similar considerations.) Thus the only architectural variable is the number of units in the hidden layer. Our working definition of "optimal" is as follows.

For a given architecture we train the NN until generalization error reaches a minimum value. Generally the generalization, or test, error reaches a minimum and begins to increase due to the "overtraining effect". The state of the NN at this minimum generalization error is saved as the acceptable state for that NN. This test is repeated a number of times to obtain an average minimum

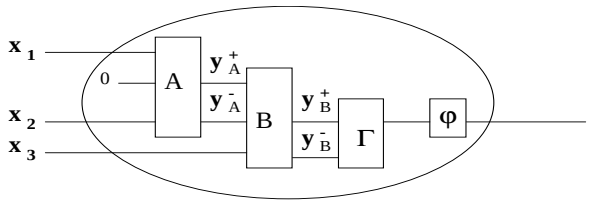

Figure 1 NN for REG node.

generalization error (MGE) for a given architecture. The MGE is then plotted as a function of the architecture. As the number of units increases, this plot can be expected to reach a minimum and begin to increase due to the "memorization effect". The architecture that attains this minimum MGE is our optimal architecture .

It is now well known that many functions can be approximated by neural networks (see [12, 22] for example). In particular, all of the node combining and output functions for EMYCIN/M.4, given section 5.1, can be approximated with NNs. We now present some experimental results on finding these approximations. All of the data discussed below was generated using 50 randomly generated training exemplars and 10 randomly generated test exemplars for each training run on each architecture. Five such training runs were made for each architecture, and the average training and testing error over all five runs was used to determine MGE for each architecture.

We are investigating two methods of constructing REG nodes. The first, shown in Figure 1, uses a parallel evidence network (labeled A) followed by a reconciler (Γ). To determine an "optimal" architecture for the evidence network we follow the process described in section 6.1 above. The results of averaging five training trials on a 4-4-1 architecture for a 4-input y^+ network are illustrated in Figure 2. The generalization curve attains a minimum at 41 epochs with MGE = 5.6×10^{-3}. The MGE for 4-n-1 architectures, n= $2, \ldots, 9$, are given in Figure 3. These computations show that 4-6-1 is the optimal architecture (in the class under consideration) for the 4-input y^+ network, with MGE = 3.9×10^{-3}.

The second architecture we are currently testing for REG nodes is a modular construction as illustrated in Figure 4. The modularity is based on the accumulation of evidence as given in equation 4.6. Modularity allows us to concentrate on solving the 3-input REG problem and then build more general REG nodes using extant components. The modules labeled A and B in Figure 4 are identical 3-n-2 NNs that take three evidence values as input and give the

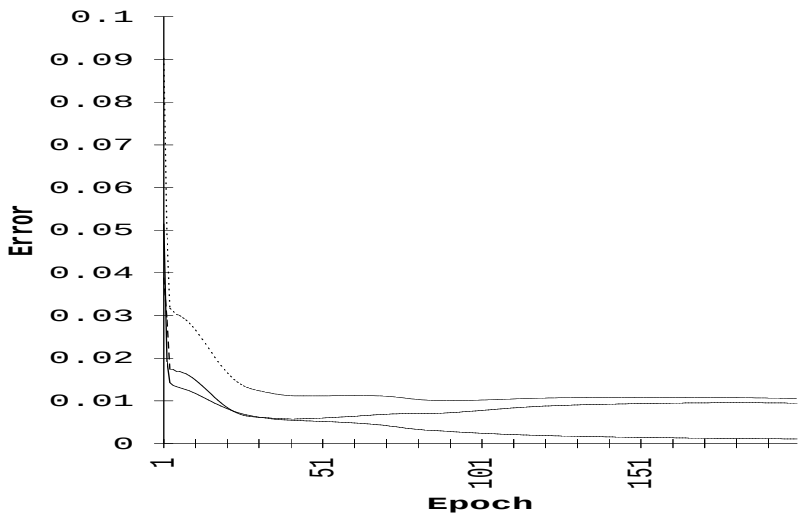

Figure 2 Training and generalization error for 4-4-1 y^+ network (average of five runs).

values y^+ and y^- as output. Once the optimal 3-n-2 module is trained it can be used in cascade fashion to build an evidence accumulator for any number of inputs. The savings in training effort is offset by loss of parallelism in the evidence computation. The MGE plot for 3-n-2 indicates that 3-6-2 is optimal.

We have tested the idea of replacing symbolic nodes with these subsymbolic networks and subsequently training the EN/NN with ENBP (as in section 5.3). For the small expert net we used for testing this experiment worked as one would expect: the EN/NN learned with about the same efficiency as the original EN.

We are carrying out exhaustive experiments to determine optimal architectures for y^+, y^- as well as "black box" REG nodes with n inputs, $n = 3, 4, \ldots$. These should give a good picture of how parallel REG NNs scale with the number of inputs. For models of human reasoning, however, this scaling may be irrelevant: it seems likely that the modular architecture approach more closely resembles human evidenciary techniques – we tend to weigh evidence a few components at a time and "build a case" rather than process many pieces of evidence in parallel. What even these preliminary results show is that symbolic-level

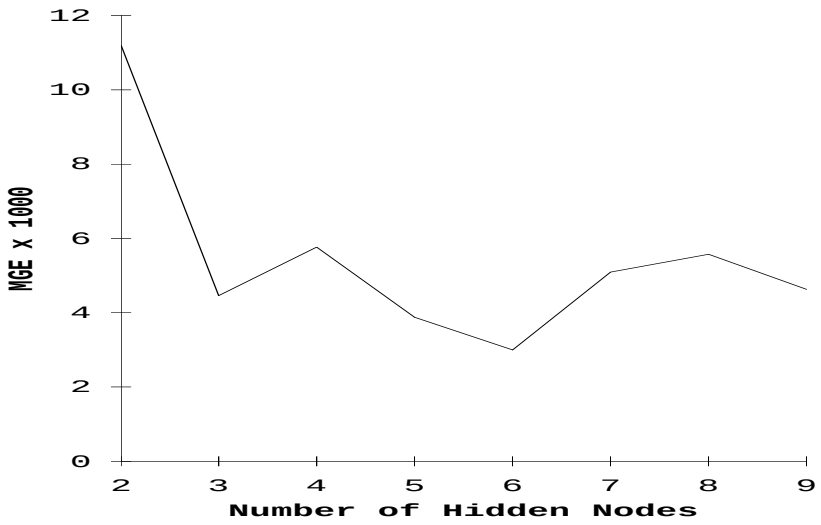

Figure 3 Minimum generalization error $\times 1000$ for 4-n-1 y^+ networks.

nodes in an expert network can be built with very simple sub-symbolic neural networks and standard training techniques.

7 SUMMARY

We have defined a general framework for parallel/distributed computation, the computational network, or CN. Examples of computational phenomena that admit descriptions within the CN model include biological neural networks, artificial neural networks, the parallel virtual machine model of loosely coupled MIMD computation, human collaborations such as committees, and expert networks. A CN is essentially a directed graph in which each component (vertex or directed edge) has data processing functionality, further endowed with a concept of global network computation.

A computational network can be classified according to whether it (1) processes discrete or analog data, (2) uses discrete or continuous time, and (3) has an acyclic or recurrent network topology. Expert networks are CNs of "type x00", according to this classification.

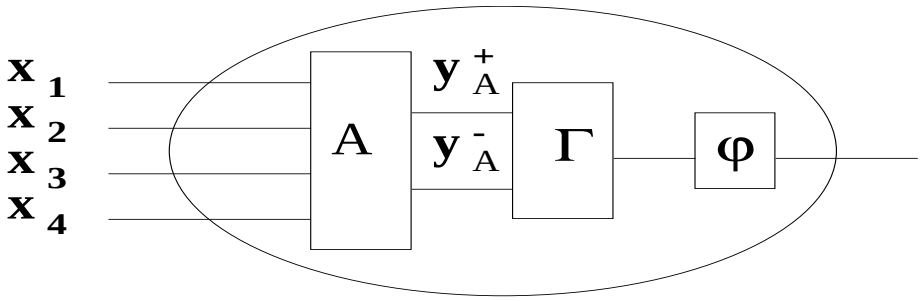

Figure 4 Modular NN for REG node.

The principles of backpropagation learning are re-examined in the context of computational networks, and a general learning method, computational network backpropagation, or CNBP, is derived.

Expert networks, or ENs, are the focus of the remainder of the chapter. An expert network is a symbolic-level computational network that can be derived from an expert system (ES). The network topology of the EN is derived from the rule base of the ES, the local processing functionality of the EN components from the rules of inference of the ES, and the global computation concepts of the EN from the inference engine of the ES. The process of constructing an EN from an ES is called *translation*. Translation, before or after learning, is a reversible process.

Learning methods for CNs can be instantiated for expert networks. In particular, CNBP specializes to expert network backpropagation, or ENBP, a learning method that has proven to be useful in knowledge acquisition and refinement. ENBP allows an expert system builder to make the transition from coarse knowledge, in the form of rough-draft rules, to fine knowledge, in the form of rules with subtlety represented by analog parameters such as certainty factors, using supervised learning and the historical record of expert behavior as a training set. This relieves the human expert whose knowledge is being captured from specifying any parameters such as probabilities or certainties.

We conclude with an investigation of how an EN can be given the structure of an artificial neural network. By a neural network, or NN, we mean a computational network consisting entirely of sub-symbolic processors such as linear/sigmoidal units. The nodes of an EN can, in principle, be represented by small NNs, and we investigate the practicality of this theory. We show in practice how such components can be constructed and determine optimal neural architectures

for such components. In this way an expert network is given a realization as a neural network with a hierarchical topological organization: a sparsely interconnected ($O(n)$) collection of densely intraconnected ($O(n^2)$) neural nets.

This hierarchical sparse/dense EN/NN organization is analogous to biological neural organization. It captures two levels of knowledge: domain knowledge in the sparse superstructure and metaknowledge in the dense substructures. The sparse/dense architecture also scales much more comfortably than the dense $O(n^2)$ connectivity of, for example, feedforward NNs. Memory stability is supported by constructive EN learning methods. Using conservative estimates of 10^{10} neurons and 10^{13} synapses in the human cerebral cortex, and assuming a sparse/dense topology with constant size dense subnetworks, an estimated subnetwork size is $1,000$ units. This is more than enough resource to train complex symbolic-level components.

Research continues in this area. Projects using expert networks as a tool in large expert system development are testing the limits of usefulness of EN technology. Other more fundamental work investigates how dual sparse/dense representations of expert networks may self-organize from random soup of neural networks and may shed light on questions of the role of early learning in cognitive development.

Computational networks are ubiquitous in the natural world and in the creations of humankind.

REFERENCES

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

[2] P. M. Churchland and P. S. Churchland. Could a machine think? *Scientific American*, 262:32–39, 1990.

[3] C. H. Dagli and R. Stacey. A prototype expert system for selecting control charts. *Int. J. Prod. Res.*, 26:987–996, 1988.

[4] S. P. Eberhardt, T. Daud, D. A. Kerns, T. X. Brown, and A. P. Thakoor. Competitive neural architecture for hardware solution to the assignment problem. *Neural Networks*, 4:431–442, 1991.

[5] S. P. Eberhardt, T. Duong, and A. P. Thakoor. Design of parallel hardware neural network systems from custom analog VLSI building block chips. In *Proceedings IJCNN 89 –Washington, DC*, volume 2, pages 183–190, Piscataway, NJ, 1989. IEEE.

[6] S. P. Eberhardt, T. Duong, and A. P. Thakoor. A VLSI building block chip for hardware neural network implementations. In *Proceedings Third Annual Parallel Processing Symposium*, volume 1, pages 257–267, Fullerton, CA, 1989. IEEE Orange County Computer Society.

[7] R. C. Eberhart and R. W. Dobbins. *Neural Network PC Tools*. Academic Press, San Diego, 1990.

[8] S. E. Fahlman and C. Lebiere. The cascade correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532. Morgan Kaufmann, New York, 1990.

[9] W.-Z. Fang, S. I. Hruska, and R. C. Lacher. Expert networks: An empirical study of expert network backpropagation learning. 1994. in preparation.

[10] W. J. Freeman. Simulation of chaotic EEG patterns with a dynamic model of the olfactory system. *Biological Cybernetics*, 56:139–150, 1987.

[11] L.-M. Fu and L.-C. Fu. Mapping rule-based systems into neural architecture. In *Knowledge Based Systems*, volume 3, pages 48–56. 1990.

[12] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[13] S. I. Gallant. Connectionist expert systems. *Communications of the Association for Computing Machinery*, 24:152–169, 1988.

[14] J. Giarratano and G. Riley. *Expert Systems: Principles and Practice*. PWS-KENT, Boston, 1989.

[15] J. Giarratano and G. Riley. *Expert Systems: Principles and Practice*. PWS-KENT, Boston, 1994. Second Edition.

[16] L. O. Hall and S. G. Romaniuk. Fuzznet: Toward a fuzzy connectionist expert system development tool. In *Proceedings IJCNN 90 – Washington, DC)*, volume 2, pages 483–486, 1990.

[17] P. Harmon. G2: Gensym's real-time expert system. *Intelligent Software Strategies*, 9:1–14, 1993.

[18] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, New York, 1991.

[19] M. W. Hirsch. Convergent activation dynamics in continuous time networks. *Neural Networks*, 2:331–349, 1989.

[20] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings National Academy of Sciences, USA*, 79:1554–2558, 1982.

[21] J. J. Hopfield. Neurons with graded responses have collective computational properties like those of two-state neurons. *Proceedings National Academy of Sciences, USA*, 81:3088–3092, 1984.

[22] K. Hornik, M. Stinchcomb, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[23] S. I. Hruska and D. C. Kuncicky. Application of two-stage learning to an expert network for control chart selection. In C. Dagli, S. Kumara, and Y. Shin, editors, *Intelligent Engineering Systems Through Artificial Neural Networks*, pages 915–920. ASME Press, New York, 1991.

[24] S. I. Hruska and D. C. Kuncicky. Automated knowledge refinement for control chart selection. *Heuristics*, 1992.

[25] S. I. Hruska, D. C. Kuncicky, and R. C. Lacher. Hybrid learning in expert networks. In *Proceedings IJCNN 91 – Seattle*, volume 2, pages 117–120. IEEE 91CH3049-4, July 1991.

[26] S. I. Hruska, D. C. Kuncicky, and R. C. Lacher. Resuscitation of certainty factors in expert networks. In *Proceedings IJCNN 91 – Singapore*, pages 1653–1657. IEEE 91CH3065-0, November 1991.

[27] B. Kosko. *Neural Networks and Fuzzy Systems.* Prentice Hall, Englewood Cliffs, NJ, 1992.

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

[29] D. C. Kuncicky. *Isomorphism of Reasoning Systems with Applications to Autonomous Knowledge Acquisition.* PhD thesis, Florida State University, Tallahassee, FL., 1991. R. C. Lacher, Major Professor.

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

[31] D. C. Kuncicky, S. I. Hruska, and R. C. Lacher. Hybrid systems: The equivalence of expert system and neural network inference. *International Journal of Expert Systems*, 4:281–297, 1992.

[32] R. C. Lacher. Node error assignment in expert networks. In A. Kandel and G. Langholz, editors, *Hybrid Architectures for Intelligent Systems*, pages 29–48. CRC Press, London, 1992.

[33] R. C. Lacher. The symbolic/sub-symbolic interface: Hierarchical network organizations for reasoning. In R. Sun, editor, *Integrating Neural and Symbolic Processes*. AAAI-92 Workshop, 1992.

[34] R. C. Lacher. Expert networks: Paradigmatic conflict, technological rapprochement. *Minds and Machines*, 3:53–71, 1993.

[35] R. C. Lacher, S. I. Hruska, and D. C. Kuncicky. Expert networks: A neural network connection to symbolic reasoning systems. In M. B. Fishman, editor, *Proceedings FLAIRS 91*, pages 12–16, St. Petersburg, FL, 1991. Florida AI Research Society.

[36] R. C. Lacher, S. I. Hruska, and D. C. Kuncicky. Backpropagation learning in expert networks. *IEEE Transactions on Neural Networks*, 3:62–72, 1992.

[37] K. Narita and R. C. Lacher. The FEN learning architecture. In *Proceedings IJCNN 93 – Nagoya*, pages 1901–1905, Washington, DC, 1993. Institute of Electrical and Electronic Engineers.

[38] K. D. Nguyen, K. S. Gibbs, R. C. Lacher, and S. I. Hruska. A connection machine based knowledge refinement tool. In M. B. Fishman, editor, *FLAIRS 92*, pages 283–286, St. Petersburg, 1992. Florida Artificial Intelligence Research Symposium.

[39] T. Oi. Chaos dynamics executes inductive inference. *Biological Cybernetics*, 57:47–56, 1987.

[40] R. Ratliff. Continuous vs discrete information processing: Modelling the accumulation of partial information. *Psychological Review*, 95:238–255, 1988.

[41] R. R. Rocker. An event-driven approach to artificial neural networks. Master's thesis, Florida State University, Tallahassee, FL., 1991. S. I. Hruska, Major Professor.

[42] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing.* MIT Press, Cambridge, MA., 1986.

[43] M. J. Salzgeber, J. L. Franke, and S. I. Hruska. Managing uncertainty in clips: A system level approach. In M. B. Fishman, editor, *FLAIRS 93*, pages 142–146, St. Petersburg, 1993. Florida Artificial Intelligence Research Symposium.

[44] J. R. Searle. Is the brain's mind a computer program? *Scientific American*, 262:26–31, 1990.

[45] T. J. Sejnowski and C. R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168, 1987.

[46] E. H. Shortliffe. *Computer-Based Medical Consultations: MYCIN.* Elsevier, New York, 1976.

[47] E. H. Shortliffe and B. G. Buchanan. A model of inexact reasoning in medicine. In *Rule-Based Expert Systems*, pages 233–262. Addison-Wesley, New York, 1985.

[48] P. K. Simpson. *Artificial Neural Systems.* Pergamon Press, New York, 1990.

[49] C. Skarda and W. J. Freeman. How brains make chaos in order to make sense of the world. *Behavioral and Brain Sciences*, 10:161–195, 1987.

[50] R. Sun. A discrete neural network model for conceptual representation and reasoning. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, Hillsdale, N.J., 1989. Erlbaum.

[51] R. Sun. On variable binding in connectionist networks. *Connection Science*, 4:93–124, 1992.

[52] V. S. Sunderam. Heterogeneous environments for network concurrent computing. *Journal of Future Generation Computer Systems*, 8:191–203, 1992.

[53] V. S. Sunderam and G. A. Geist. Network based concurrent computing on the pvm system. *Journal of Concurrency: Practice and Experience*, 4:293–311, 1992.

[54] G. G. Towell, J. W. Shavlik, and M. O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings AAAI-90*, pages 861–866, New York, 1990. Morgan Kaufmann.

[55] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, Cambridge, MA, 1974.

ABSTRACT

(Reprinted from Chapter 4 of *Computational Architectures for Integrating Neural and Symbolic Processes* (R. Sun and L. Bookman, eds.), **Kluwer Academic Publishers**, 1994, pp 117–150.)

We introduce a general framework for parallel/distributed computation, the computational network. Computational networks provide a context for the comparative study of many different kinds of computation, including biological neural networks, artificial neural networks, parallel virtual machine, human collaborations, and symbolic and sub-symbolic models of reasoning. Computational network backpropagation (CNBP) learning is derived.

We discuss the method of expert networks (a symbolic model) and how expert networks can be realized by organization in a neural network (a sub-symbolic model). The architecture of the expert network is derived from the expert system: the network topology from the rule base, the local processing functionality of the vertices and edges from the system of inference, and the global computation concepts from the inference engine. Expert network backpropagation (ENBP) learning is obtained as a special case of CNBP.

We show how both the symbolic and sub-symbolic systems process inferences and learn from incorrect inferencing. We illustrate how individual symbolic nodes can be realized by low-level neural networks and how dual super/sub organizational levels facilitate scaling expert networks up to very large size.