Concise Papers

A Multilayer Perceptron Solution to the Match Phase Problem in Rule-Based Artificial Intelligence Systems

Michael A. Sartori, Kevin M. Passino, and Panos J. Antsaklis,

Abstract— **In rule-based artificial intelligence (AI) planning, expert, and learning systems, it is often the case that the left-hand-sides of the rules must be repeatedly compared to the contents of some "working memory." Normally, the intent is to determine which rules are relevant to the current situation (i.e., to find the "conflict set"). The traditional approach to solve such a "match phase problem" for production systems is to use the Rete Match Algorithm. Here, a new technique using a multilayer perceptron, a particular artificial neural network model, is presented to solve the match phase problem for rule-based AI systems. A syntax for premise formulas (i.e., the left-hand-sides of the rules) is defined, and working memory is specified. From this, it is shown how to construct a multilayer perceptron that finds all of the rules which can be executed for the current situation in working memory. The complexity of the constructed multilayer perceptron is derived in terms of the maximum number of nodes and the required number of layers. A method for reducing the number of layers to at most three is also presented.**

Index Terms—**Expert systems, multilayer perceptron, neural networks, production systems, Rete Match Algorithm.**

I. INTRODUCTION

IN rule-based artificial intelligence (AI) systems, the problem of finding which rules are executable from a given set of rules at different instances in time is often encountered. At each time instant, the left-hand-side of every rule of a given set of rules is compared to the dynamically changing "working memory" of the system. The rules whose left-hand-sides are satisfied by the current working memory form the "conflict set" at that particular time. The problem of determining the conflict set from the contents of working memory at a particular time is referred to here as the *match phase problem.* Some of the types of AI systems in which the match phase problem occurs include rule-based expert, planning, and learning systems. Typically, in such rule based systems, an "inference engine" or some other suitable algorithm finds the rules which are executable (in the "match phase"), chooses one (in the "select phase"), and executes it (in the "act phase"). This paper focuses on a novel technique to solve the match phase problem via a multilayer perceptron. The problem of choosing which rule to execute (e.g., via an inference engine using certainty factors or fuzzy logic) and the problem of deciding the manner in which the chosen rule is to be executed are not addressed here. Also, the actual implementation of such a system is not treated. The purpose of this paper is to present an alternative and new approach to performing the match phase for rule-based AI systems. A brief overview of where the match phase problem is found in rule-based AI systems is now given and is followed by a brief description of the Rete Match Algorithm, the conventional solution to the match phase problem.

Manuscript received October 25, 1989; revised June 4, 1990 and March 6, 1991. This work was supported in part by the Jet Propulsion Laboratory under Contract 957856.

M. A. Sartori is with the David Taylor Research Center, Bethesda, MD 20084-5000.

K. M. Passino is with the Department of Electrical Engineering, Ohio State University, Columbus, OH 43210.

P. J. Antsaklis is with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556.

IEEE Log Number 9106262.

A *rule-based expert system* uses *rules*, sometimes referred to as *productions* or production rules, to represent knowledge and uses an *inference engine* to perform the actions of the expert system. In general, a rule is of the form

$$\text{IF (antecedent) THEN (consequence)} \tag{1}$$

Customarily, the antecedent is referred to as the left-hand-side, and the consequence is referred to as the right-hand-side. The working memory (data memory) contains dynamic data that is compared to the left-hand-side of the rules. The individual elements of the working memory are referred to as the working memory elements. The inference engine performs the comparison of the working memory elements to the left-hand-sides of the rules, chooses which rules are executable for the given state of the expert system, chooses one of the executable rules, and executes it. Often the inference engine is viewed as having a three phase cycle [1], [2]:

1) *Match:* Compare the left-hand-side of all of the rules to the working memory elements. If the left-hand-side is satisfied, include the rule in the conflict set, the set of satisfied rules for the present working memory state.
2) *Select:* Choose one rule from the conflict set to execute.
3) *Act:* Execute the rule in accordance with the right-hand-side of the chosen rule.

The results obtained here are applicable to rule-based expert systems in which the match phase can be separated from the select and the act phases. It is assumed, without loss of generality, that forward chaining instead of backward chaining is used. Rule-based expert systems developed with OPS5, EMYCIN, ROSIE, and KEE as cited in [1] and Level 5 as described in [3] may benefit from a multilayer perceptron implementation of the match phase as described in this paper.

The match phase problem is also encountered in *rule-based planning systems* and *rule-based learning systems.* Often, such systems can be implemented using the same tools that are listed above for rule-based expert systems. In general, the match phase determines whether or not certain patterns match, which is equivalent to finding which formulas in a set of logical formulas are true. Such tasks are executed by most AI systems including planning and learning systems. Thus, the results here are generally applicable to the implementation of a wide variety of AI systems.

Currently, the match phase problem (particularly for rule-based expert systems) is often solved via the Rete Match Algorithm [4]. If a rule-based expert system's rules are of the form depicted in (1) and if the inference engine explicitly follows the three phase cycle, the rules are referred to as "productions" and the inference engine is referred to as the "production interpreter." Of the three phases, the match phase traditionally consumes the most time of the production interpreter. Using conventional approaches, a production interpreter can spend more than 90% of its time in the match phase of the cycle [4]. The Rete Match Algorithm, introduced in [4] and [5] and implemented in the OPS5 expert system building tool in [1], avoids the brute force approach by manipulating the rules and the working memory elements to form a software tree structure to increase the speed of the interpreter. In [5], it was reported that the processing

TABLE I
SUMMARIZED RESULTS OF SPECIAL HARDWARE IMPLEMENTATIONS.

Architecture	WME Changes/Sec	Production Cycles/Sec	Hardware Needed	Reference
CMU-PMS	9400	-	32 Ps	[10]
DADO	215	85	1023 PE's	[21], [10]
MAPPS	10000	-	128 PE's	[18]
NON-VON	2000	903	16032 PE's	[18]
Olfazer's	4500	-	512 Ps	[10]
PESA-1	25000	8000	32 PE's	[19]

WME = working memory element. - = not available. Ps = processors. PE's = processing elements

time for a production interpreter using the Rete Match Algorithm for the match phase is dependent on both the number of working memory elements and the number of rules. For a production system, if W is the number of working memory elements in the working memory and A is the number of atomic propositions per rule, the effect of the working memory size on the time for one firing using the Rete Match Algorithm is $O(1)$ for the best case and $O(W^{2A-1})$ for the worst case [5] (where the "O" notation denotes "on the order of" and is the standard one defined in [6]). If R is the number of rules in the production system, the effect of the number of rules on the time for one firing using the Rete Match Algorithm is $O(\log_2 R)$ for the best case and $O(R)$ for the worst case [5]. So, the time for one firing of the match phase using the Rete Match Algorithm is dependent on both the number of rules and the number of working memory elements. In the following, it will be shown that when the proposed multilayer perceptron solution to the match phase problem is used, the processing time is in fact independent of both the number of working memory elements and the number of rules.

Since the introduction of the original Rete Match Algorithm, other algorithms, some of which are based on the Rete Match Algorithm, have been introduced to increase the speed of the interpreter [7]–[14]. To accelerate the performance of the interpreter in the match phase, parallel hardware solutions have been developed [10], [15]–[23]. The Rete Match Algorithm has also been implemented on a multiprocessor machine, and an increase in speed was reported for specific rule-based systems [24]. It should be noted that some of these attempts are based on the Rete Match Algorithm. These architectures, as reported in the literature, strive to decrease the time required in the match phase by attempting to match as many rules as possible in parallel and by attempting to fire as many rules as possible in parallel. Using the YES/OPS production system language and advances in the Rete Match Algorithm, a drop in CPU time was reported in [7]. Using partitioning of the productions, a reduction in production cycles was reported in [11], [12], and [14]. The results from the literature of using special hardware are summarized in Table I. The current hardware implementations cited in Table I use either many processors or many processing elements.

The method proposed here for the match phase is quite different from the Rete Match Algorithm and the above mentioned modifications. The novel approach presented in this paper takes advantage of the inherent parallelism of the multilayer perceptron by simultaneously matching *all* of the rules to *all* of the working memory elements. By first defining "premise formulas" that represent the left-hand-sides of the rules, the match phase problem is defined as the determination of the truth of the premise formulas for the current working memory. The multilayer perceptron is the vehicle used to accomplish this. The input to the multilayer perceptron is the working memory, and the output is the conflict set. The specially designed multilayer perceptron simultaneously finds all of the rules which are executable at any particular time by matching all of the rules in parallel to the current working memory.

The work described in this paper is the first of its kind, and many future directions stem from this initial investigation. For instance, there are many different models of artificial neural networks that could be used for the match phase instead of the chosen multilayer perceptron. As an example, the multilayer perceptron trained with the Back Propagation Algorithm of [25] appears promising although there exist numerous difficulties with such an approach (e.g., convergence, the number of nodes and the number of layers to use, and a long training time). Another possible artificial neural network model to use is the Hopfield model [26]. If the minima of the energy surface can be chosen such that they correspond to the appropriate selection of rules given a set of working memory elements, the Hopfield model could replace the multilayer perceptron proposed here provided that the global stability of the net is ensured and that the number of spurious states is minimized. The proposed multilayer perceptron is also rigid and not fault tolerant. An investigation into alleviating this is a future direction. One possibility might be to use a Hopfield model instead of the multilayer perceptron for the match phase. As another future direction, the use of the multilayer perceptron, or any other artificial neural network model, to implement the other two phases (i.e., the select phase and the act phase) is a possibility. Perhaps an artificial neural network could even be trained to mimic an entire rule-based AI system. Also, the actual implementation of the solution proposed here for the match phase problem is an important future direction, as well as an investigation of the interfacing of the multilayer perceptron with other hardware. In addition, the automation of the design of the multilayer perceptron as discussed in Section III-B could be a valuable contribution. Next, the contents of the paper are summarized.

In Section II, premise formulas and the match phase problem are defined in precise mathematical terms. In Section III, a procedure for designing a multilayer perceptron for a single premise formula is described, extended to one for designing a multilayer perceptron for a set of given premise formulas, and related to its use as a solution to the match phase problem. The number of nodes and the number of layers needed to implement the multilayer perceptron are discussed. A method to reduce the number of layers in any multilayer perceptron to at most three and a formula for the number of nodes required to do so is also presented in this section. Finally in Section IV, concluding remarks are made including a citing of some of the potential limitations in using the multilayer perceptron solution for the match phase. This work is an extension of the research reported in [27] via the results in [28] and [29]. The artificial neural network used in the previous work was the model termed the "ProNet," which was obtained by modifying the Hamming net of [30]. The multilayer perceptron was used rather than the ProNet because the multilayer perceptron is better able to address a greater diversity of types of rules as well as rules which contain real numbers. A shorter version of this paper appears in [31], and an example of the forming of a multilayer perceptron for a particular rule-based expert system's rules (the "Monkey and Bananas Problem" [1]) is included in [32], but excluded here due to space limitations.

II. THE MATCH PHASE PROBLEM

In this section, the premise formulas for the left-hand-sides of the AI system's rules are defined, and the match phase problem is defined

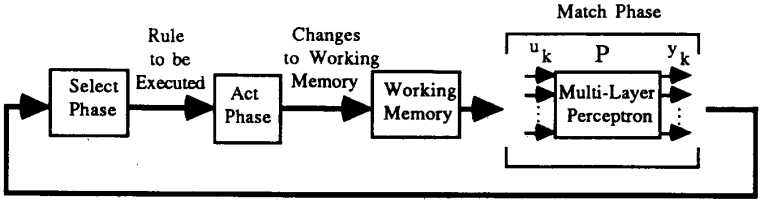


Fig. 1. The multilayer perceptron implementation of the match phase in a rule-based AI system.

in terms of the premise formulas and the working memory. Rule-based systems with left-hand-sides which can be described using the premise formulas described below can have the match phase of their inference engine performed by a multilayer perceptron as discused in the following sections.

The following describes the syntax of the premise formulas. Let $|\boldsymbol{S}|$ denote the number of elements in the set $\boldsymbol{S}$. Let $\boldsymbol{A} = \{a_1, a_2, \cdots, a_n\}$ be a nonempty finite set of *atomic propositions* a_i, where $|\boldsymbol{A}| = n$. Such propositions will represent facts that are stored in working memory. Let $X \subset \mathbb{R}^p$, where $\mathbb{R}$ denotes the set of real numbers. Each $\boldsymbol{x} = [x_1, x_2, \cdots, x_p]^t \in X$ (where "t" denotes transpose) represents numeric information in working memory. The standard Boolean connectives ($\neg$ (negation), $\wedge$ (disjunction), and $\vee$ (conjunction)) are used. The rules for forming the *premise formulas* are as follows:

1) A single atomic proposition $a \in \boldsymbol{A}$ is a premise formula.
2) If σ is a premise formula, then $\neg\sigma$ is a premise formula.
3) If σ and Ψ are premise formulas, then so are $(\sigma \wedge \Psi)$ and $(\sigma \vee \Psi)$.
4) If $\boldsymbol{x} \in \boldsymbol{X}$, $\boldsymbol{x} = [x_1, x_2, \cdots, x_p]^t$, and $r \in \mathbb{R}$, then $(x_i > r)$, $(x_i < r)$, $(x_i \geq r)$, and $(x_i \leq r)$ for any i such that $1 \leq i \leq p$ are premise formulas.
5) Nothing is a premise formula unless it can be obtained by finitely many applications of 1)–4) above.

If some left-hand-side of a rule has either the Boolean connective $\Rightarrow$ (implication) or $\Leftrightarrow$ (equivalence), then the following substitutions can be made. If σ and Ψ are premise formulas, then $(\sigma \Rightarrow \Psi)$ can be replaced by the equivalent premise formula $(\neg\sigma \vee \Psi)$, and $(\sigma \Leftrightarrow \Psi)$ by the equivalent premise formula $((\sigma \wedge \Psi) \vee (\neg\sigma \wedge \neg\Psi))$. If some left-hand-side of a rule has either the predicate symbol = (equal) or $\neq$ (not equal), then the following substitutions can be made. If $\boldsymbol{x} \in \boldsymbol{X}$, $\boldsymbol{x} = [x_1, x_2, \cdots, x_p]^t$, and $r \in \mathbb{R}$, then $(x_i = r)$ can be replaced by the equivalent premise formula $((x_i \geq r) \wedge (x_i \leq r))$, and $(x_i \neq r)$ by the equivalent premise formula $((x_i > r) \vee (x_i < r))$ for $1 \leq i \leq p$.

Assume that the working memory for the rule-based system is composed of atomic propositions $a \in \boldsymbol{A}$ and the current working memory information $\boldsymbol{x} \in \boldsymbol{X}$. The output of the working memory is specified next. Define the function

$$V : \boldsymbol{A} \rightarrow \{0, 1\} \tag{2}$$

where $V(a) = 1$ indicates "a is true" and $V(a) = 0$ indicates that "a is false." Let the n-vector $\boldsymbol{v_k} = [V(a_1), V(a_2), \cdots V(a_n)]^t$ with components $V(a_i)$ representing the truth values of all of the atomic propositions at step k. Let $\boldsymbol{x_k} \in \boldsymbol{X}$ denote the value of $\boldsymbol{x}$ at step k. At step k, the output of the working memory is defined to be the $(n+p)$-vector $\boldsymbol{u_k} = [\boldsymbol{v_k^t}, \boldsymbol{x_k^t}]^t$, which contains information about both the truth values of atomic propositions and the numeric values of the working memory elements. Let $\boldsymbol{U}$ denote the set of all possible working memory outputs.

Let $\Phi = \{\phi_1, \phi_2, \cdots, \phi_m\}$ denote a finite set of premise formulas. Let $\boldsymbol{Y} \subset \{0, 1\}^m$. Number the rules in the rule base from 1 to m. For the ith rule in the rule base form the premise formula ϕ_i and associate ϕ_i with a component y_i of $\boldsymbol{y_k} = [y_1, y_2, \cdots, y_m]^t$ for $1 \leq i \leq m$. If $y_i = 1$ for $1 \leq i \leq m$, then the premise formula ϕ_i is true, and the ith rule is executable and included in the conflict set. If $y_i = 0$, then the corresponding rule is not included in the conflict set. Hence, at step k, $\boldsymbol{y_k} \in \boldsymbol{Y}$ represents the conflict set.

The match phase problem can be solved by implementing the function

$$P : \boldsymbol{U} \times \Phi \rightarrow \boldsymbol{Y}. \tag{3}$$

Here, the focus is on the use of the multilayer perceptron to implement the function P of (3) (i.e., to perform the match phase in a rule-based AI system). This is illustrated in Fig. 1. The input to the multilayer perceptron, which describes the working memory, is the vector $\boldsymbol{u_k}$. The input vector is compared to the left-hand-sides of the entire set of rules described by the set Φ which is stored in the multilayer perceptron as its weights, biases, and interconnections. The output of the multilayer perceptron is the vector $\boldsymbol{y_k}$, which denotes the conflict set. Thus, the multilayer perceptron specifies the conflict set $\boldsymbol{y_k} \in \boldsymbol{Y}$ at each time instant k for all possible inputs $\boldsymbol{u_k} \in \boldsymbol{U}$ and premise formulas $\phi \in \Phi$, and hence implements the function P.

Following Fig. 1, from the output vector $\boldsymbol{y_k}$, the conflict set is placed in a form which is usable by the select phase. The select phase can be implemented in numerous ways; for an explanation of some of the possibilities for accomplishing this, see [1], [2], and [33]. Once the rule is chosen by the select phase, it is executed via the act phase. The execution of a rule causes changes to the working memory. The vector $\boldsymbol{v_{k+1}}$ is produced by altering the vector $\boldsymbol{v_k}$ such that some of the elements $V(a_i)$ are changed from one to zero (indicating that a_i becomes false) and others from zero to one (indicating that a_i becomes true). The vector $\boldsymbol{x_k}$ can also be changed according to the fired rule to produce the vector $\boldsymbol{x_{k+1}}$. So, the working memory is updated, and the input vector $\boldsymbol{u_k}$ is changed to $\boldsymbol{u_{k+1}}$. The new input vector $\boldsymbol{u_{k+1}}$ is used as an input to the multilayer perceptron which gives the output $\boldsymbol{y_{k+1}}$, and the process is repeated. Next, it is shown how to construct a multilayer perceptron which implements the function P and hence solves the match phase problem.

III. A Multilayer Perceptron Solution to the Match Phase Problem

Artificial neural networks are used to perform computations in a massively parallel fashion. They are processing models which derive their structure and functionality as an interconnected network of neurons from models of biological neurons. Each neuron of the artificial neural network has many inputs and one output. The output of each neuron is generally considered to be the weighted sum of its inputs passed through a nonlinear function. The output is then used as inputs to other neurons. Artificial neural networks are characterized by the type of neurons used, the way in which the weights are selected, and the types of interconnections that are allowed between neurons. Here, a specific artificial neural network called the *multilayer perceptron* is used as the model for solving the match phase problem.

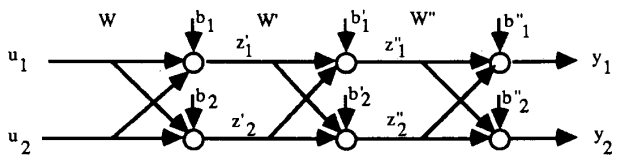


Fig. 2. The multilayer perceptron for two nodes at each layer.

The multilayer perceptron is a feedforward model which is restricted here so that it is partially connected and also so that it does not self-adjust its weights or learn.

As discussed at the end of Section I, instead of using the multilayer perceptron, other artificial neural network models may be used for the match phase problem, but these do not appear to be as favorable. For example, the Hopfield network may be considered [26], but because it uses feedback to determine its output, it requires time to converge to a minimum. The multilayer perceptron always determines its outputs immediately since it is a feedforward network with theoretically no significant delays. Also, if the weights for the Hopfield network are not chosen properly, spurious states may result. Artificial neural networks which use unsupervised training, such as the Kohonen network [34], may not be as useful as the multilayer perceptron since the input and output relationships for the artificial neural network are known *a priori*, and thus the clustering methodology of the unsupervised training is not needed. Artificial neural networks that use supervised training, such as the Back Propagation Training Algorithm applied to the multilayer perceptron [25], also may not be as useful since the weights for the network can be selected as explained in Section III-B, and so the training of the weights of the multilayer perceptron is not required. The method specified here to construct the multilayer perceptron takes advantage of the special nature of the match phase problem. In addition, problems that occur in learning algorithms that adjust the weights do not occur for the weight selection procedure proposed here. For example, in the Back Propagation Training Algorithm, the convergence of the weights does not always occur, and when they do, it is often after a long time [25]. In addition, there exist neither rules nor accurate guidelines for choosing the number of nodes and the number of layers needed to properly implement the Back Propagation Algorithm. For these reasons, the multilayer perceptron was judged to be the most suitable artificial neural network model for use in the match phase. This brief discussion about artificial neural networks and the various neural network models is not meant to be encompassing; the interested reader may examine [30] for further information on a number of different artificial neural networks used for other applications.

A. The Multilayer Perceptron

The multilayer perceptron is a feedforward artificial neural network considered here to contain at least one *hidden layer* between the *input* and the *output layers*. A multilayer perceptron with one hidden layer and with two nodes in each layer is illustrated in Fig. 2. The output of one layer is cascaded to the input of the next one. In general, the input layer feeds the first hidden layer, and the last hidden layer feeds the output layer. The input of the multilayer perceptron is applied to the input layer. The input to the multilayer perceptron considered here is the vector $\boldsymbol{u}$ with components u_i, which contains $(n+p)$ continuous real valued elements. The vector $\boldsymbol{z}'$ with components z'_i, which contains q binary elements, is the output of the input layer and the input to the hidden layer. The vector $\boldsymbol{z}''$, which contains r binary elements, is the output of the hidden layer and the input to the output layer. The vector $\boldsymbol{y}$ with components y_i, which contains m binary elements, is the output of the multilayer perceptron.

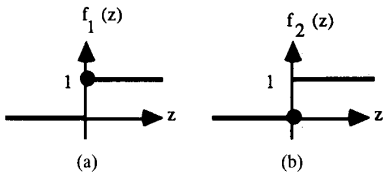


Fig. 3. Threshold nonlinearities for the multilayer perceptron.

In Fig. 2, the *nodes* are denoted with circles and the *biases* with arrows that point downward. The biases on the input layer are denoted by b_i, on the hidden layer by b'_i and on the output layer by b''_i. The *weights* are denoted by all of the other arrows (which are labeled with the weights) that are between $\boldsymbol{u}$ and the input nodal layer, between $\boldsymbol{z}'$ and the hidden nodal layer, and between $\boldsymbol{z}''$ and the output layer. Note that unlike the traditional three layer perceptron used with the Back Propagation Algorithm, the input layer can assume nonunity valued weights. The element w_{ij} of the $(n+p)\times q$ matrix W denotes the weight on the arc from u_i to the node with z'_j as its output. The $q\times r$ matrix W' denotes the weights on the arcs from each z'_i to the node with z''_j as its output. The $r\times m$ matrix W'' denotes the weights on the arcs from each z''_i to the node with y_j as its output. The weights on the arcs connecting the output nodal layer to the outputs y_i are unity. For convenience, if the weight of any arc is zero the arc will be omitted from the graphical representation of the perceptron, and if the weight of any arc is unity, the arc will be represented with no weight denoted.

Each node produces at its output a summation of its weighted inputs and its bias. This summation is passed through a *threshold nonlinearity*. The result is a binary output for each layer. The two threshold nonlinearities used in this paper are illustrated in Fig. 3, where $f_1(z)=1$ if $z\geq 0$ and 0 if $z<0$ and $f_2(z)=1$ if $z>0$ and 0 if $z\leq 0$. In this paper, nodes which use the threshold of Fig. 3(a) are unshaded and those which use the threshold of Fig. 3(b) are shaded.

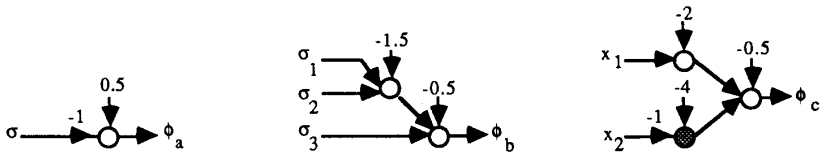
The input to the jth threshold nonlinearity of the input layer, denoted by $\tilde{z}j(k)$, is the weighted sum of the inputs added to the bias

$$\tilde{z}_j(k)=\left(\sum_{i=1}^{M} w_{ij}u_i(k)\right)+b_j. \tag{4}$$

If f_{tj} denotes a threshold nonlinearity of type t (where here $t=1$ or 2 following Fig. 3) for the jth node, then the output of the input nodal layer is given by

$$z'_j(k)=f_{tj}(\tilde{z}_j(k)). \tag{5}$$

The outputs of the hidden layers and the output layer are given by similar relations. With these equations the input–output relationship for the multilayer perceptron is specified. Using this description of the multilayer perceptron, the technique to determine the weights, biases, and number of nodes for the multilayer perceptron used in solving the match phase problem is given next.



(a) (b) (c)

Fig. 4. The multilayer perceptron for premise formulas (a) $\phi_a = \neg\sigma$, (b) $\phi_b = ((\sigma_1 \wedge \sigma_2) \wedge \sigma_3)$, and (c) $\phi_c = ((x_1 \geq 2) \vee (x_2 < -4))$.

B. Construction of the Multilayer Perceptron for the Match Phase

The construction of a multilayer perceptron for one premise formula is examined first. Given a single premise formula $\phi \in \Phi$, an artificial neural network is constructed to indicate whether or not ϕ is true for a given situation in working memory $\boldsymbol{u_k} \in \boldsymbol{U}$. The construction of the multilayer perceptron has two steps: forming the nodes and connecting the nodes. The nodes can be one of three types, each of which corresponds to a different premise formula type (i.e., 2, 3, and 4 in the rules for forming premise formulas).

Negation of Formulas: For a premise formula formed by rule 2, form a node with one input and one output. The weight is -1, the bias is 0.5, and the nonlinearity used is in Fig. 3(a). The network for the premise formula

$$\phi_a = \neg\sigma \tag{6}$$

is shown in Fig. 4(a), where σ is a premise formula.

Disjunction and Conjunction of Formulas: For a premise formula defined by rule 3, let σ_i be a premise formula for $i = 1, \cdots, j$, and consider premise formulas either of the type $(\sigma_1 \wedge \sigma_2 \wedge \cdots \wedge \sigma_j)$ or $(\sigma_1 \vee \sigma_2 \vee \cdots \vee \sigma_j)$. Form a node with j inputs and 1 output. The weights are unity and the nonlinearity used is in Fig. 3(a). If the Boolean connector is $\wedge$, then the bias of the node is $-(j - 0.5)$ where j is the number of premise formulas in the conjunction. If the Boolean connector is $\vee$, then the bias of the node is -0.5 no matter how many formulas are in the disjunction. The network formed for the premise formula

$$\phi_b = ((\sigma_1 \wedge \sigma_2) \vee \sigma_3) \tag{7}$$

is shown in Fig. 4(b), where σ_1, σ_2, and σ_3 are premise formulas.

Relational Formulas: For a premise formula formed by rule 4 of the syntax (e.g., $(x_i > r)$, $(x_i < r)$, $(x_i \geq r)$, and $(x_i \geq r)$ for $1 \leq i \leq p$, where $\boldsymbol{x} = [x_1, x_2, \cdots x_p]^t$ and $r \in \mathbb{R}$) form a node with one input and one output. Connect the node's input to the appropriate element of the vector $\boldsymbol{x}$ in the input vector $\boldsymbol{u}$ of the multilayer perceptron. If the predicate symbol is $>$ or $\geq$, then the weight is 1 and the bias is $-r$. If the predicate symbol is $<$ or $\leq$, the weight is -1 and the bias is r. If the predicate symbol is $\leq$ or $\geq$, use the nonlinearity of Fig. 3(a). If the predicate symbol is $<$ or $>$, use the nonlinearity of Fig. 3(b). The network for the premise formula

$$\phi_c = ((x_1 \geq 2) \vee (x_2 < -4)) \tag{8}$$

is shown in Fig. 4(c), where $\boldsymbol{x} \in \boldsymbol{X}$, $\boldsymbol{x} = [x_1, x_2]^t$.

Once the nodes of the multilayer perceptron are formed, the connections between them are specified by an inductive process. The output of the node describing the outermost parentheses of the premise formula is the output of the multilayer perceptron. Let this node be the first layer (i.e., the output layer). The connections between the first layer and the second layer (i.e., the last hidden layer) are made between the inputs of the first layer's node and the outputs of the nodes formed for the premise formulas of the second layer. If the first layer's node requires an element of the input vector as an input, the appropriate connection between the input vector and the node's input is made. Next, the connections between the first and second layer and the third layer (i.e., the second to the last hidden layer) are made. If the second layer's node requires an element of the input vector as an input, the appropriate connection between the input vector and the node's input is made. This process continues for each successive layer until all inputs for all nodes are connected. Notice that given any premise formula, the method used here to form the nodes and the interconnections of the multilayer perceptron can be mechanized. The entire process for constructing a multilayer perceptron for a premise formula is illustrated in Fig. 5 with the premise formula

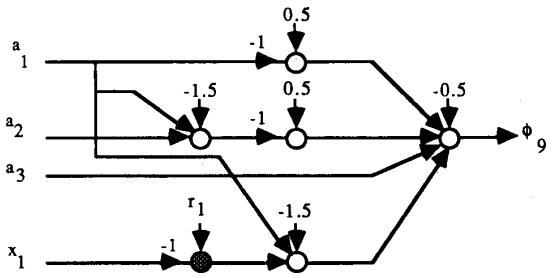


Fig. 5. The multi-layer perceptron for the premise formula of (9).

$$\phi_9 = ((a_1 \wedge (x_1 < r_1)) \vee \neg(a_1 \wedge a_2) \vee \neg a_1 \vee a_3) \tag{9}$$

where $\boldsymbol{A} = \{a_1, a_2, a_3\}$ and $\boldsymbol{x} = [x_1]$.

Using the steps detailed above to produce a neural network for an individual premise formula, the multi-layer perceptron for every premise formula $\phi \in \Phi$ is formed by repeating the above procedure for every premise formula. If two or more premise formulas share a similar premise formula, the node for that premise formula can be shared by the other premise formulas. For example, if premise formulas ϕ_9 and

$$\phi_{10} = (\neg(a_1 \wedge a_2) \vee a_3) \tag{10}$$

are given, the node formed for the premise formula $\neg(a_1 \wedge a_2)$ can be shared by the multilayer perceptrons constructed for both ϕ_9 and ϕ_{10}. Thus one combined multilayer perceptron implements all of the premise formulas in the set Φ. Notice that the possibility exists for automating the development of the multilayer perceptron once the set Φ of premise formulas is given, but this is left as a future research direction.

As stated in Section I, the time for the multilayer perceptron to process the match phase is independent of both the number of working memory elements and the number of rules. Using the multilayer perceptron solution to the match phase problem, the working memory size does not affect the time required to perform the match phase. This is the case because the working memory is processed in parallel as the input vector $\boldsymbol{u_k}$ of the multilayer perceptron, and thus the working memory size does not affect the time required to process the match phase. In addition, the number of rules (premise formulas) also

does not affect the time required to perform the match phase since the truth value of the rules with regard to the current working memory are represented as the parallel elements in the output vector $\boldsymbol{y_k}$ of the multilayer perceptron. If the multilayer perceptron is physically implemented, only the number of layers adversely affects the time needed for the multilayer perceptron to process. This problem is addressed in Section III-C.

Given any premise formula, the maximum number of nodes needed to implement the premise formula with a multilayer perceptron can be found. This is accomplished by using the function

$$N : \Phi \rightarrow \mathbb{R} \tag{11}$$

which is defined to represent the maximum number of nodes needed to implement the match phase for one $o \in \Phi$. Let ϕ, σ, and Ψ be premise formulas.

1) If $\phi = a$ where $a \in \boldsymbol{A}$, then $N(\phi) = 0$.
2) If $\phi = \neg\sigma$, then $N(\phi) = 1 + N(\sigma)$.
3) If $\phi = (\sigma \vee \Psi)$ or $\phi = (\sigma \wedge \Psi)$, then $N(\phi) = 1 + N(\sigma) + N(\Psi)$.
4) If $\phi = (x_i \triangle r)$ where $\boldsymbol{x} = [x_1, x_2, \cdots, x_p]^t$, $1 \leq i \leq p, r \in \mathbb{R}$, and $\triangle \in \{<, >, \leq, \geq\}$, then $N(\phi) = 1$.

The number $N(\phi)$ does not include subtracting the number of nodes shared by premise formulas. Hence, the number $N(\phi)$ is the maximum number of nodes needed to implement a multilayer perceptron for the premise formula ϕ. Using (9) as an example, $N(\phi_9) = 6$. Clearly, the maximum number of neurons needed to implement the function P for the entire match phase is given by

$$\sum_{\phi \in \Phi} N(\phi). \tag{12}$$

If $\Phi = \{\phi_9, \phi_{10}\}$, then

$$\sum_{i=9}^{10} N(\phi_i) = 10. \tag{13}$$

If the node for the premise formula $\neg(a_1 \wedge a_2)$ is shared by ϕ_9 and ϕ_{10}, then the actual number of nodes needed to implement the multilayer perceptron is 8, which is less than the maximum predicted in (13).

The match phase problem as formulated here appears to be implementable with standard logic gates. This is not true since premise formulas following rule 4 of the syntax are allowable. If premise formulas following rule 4 are not allowed, then clearly this solution to the match phase problem can be implemented using Boolean logic gates. The multilayer perceptron proposed here is also similar to the threshold logic circuits (networks) described in [35], except that in these circuits the theory only allows a finite number of discrete inputs. However, such threshold logic networks could be used to implement the match phase.

C. Reducing the Number of Layers in the Multilayer Perceptron

One problem with the multilayer perceptron solution is the potential for many hidden layers in the multilayer perceptron. For example, a multilayer perceptron which implements the premise formula

$$\phi_{14} = ((((\neg a \wedge b \wedge c \wedge (d \geq 5)) \vee e) \wedge f) \vee g) \tag{14}$$

where $\boldsymbol{A} = \{a, b, c, e, f, g\}$ and $\boldsymbol{x} = [d]$ would require five layers and six nodes. The more layers a multilayer perceptron has, the longer it will take the multilayer perceptron to process an output. If a fast processing time is an important design criterion for the finding of the conflict set, then any possible reduction in the number of layers needed to implement the multi-layer perceptron should be used (provided that it does not require the addition of too many more nodes).

One way to reduce the number of layers is to place all of the premise formulas in an equivalent disjunctive or conjunctive normal form. By doing this, the maximum number of layers for any premise formula is guaranteed to be at most three. This however may increase the number of nodes per layer. Regardless of this possible increase, the processing time for the multilayer perceptron will not increase beyond that of one with three layers and a single node per layer. Intuitively, converting premise formulas to a disjunctive or conjunctive normal form places the premise formulas in a form most suitable for exploring the parallel processing capabilities of the multilayer perceptron in finding the conflict set.

By using existing algorithms, a premise formula can (off-line) be placed in disjunctive or conjunctive normal form [36]. Note that the algorithm in [36] does not produce a unique disjunctive or conjunctive normal form. In using these algorithms, treat both a single proposition symbol $a \in \boldsymbol{A}$ as a proposition symbol and a premise formula $(x_i \triangle r)$ where $\boldsymbol{x} \in \boldsymbol{X}$, $\boldsymbol{x} = [x_1, x_2, \cdots, x_p]^t, r \in \mathbb{R}$, and $\triangle \in \{<, >, \geq, \leq\}$ for $1 \leq i \leq p$ is a proposition symbol. If $\beta_j = \neg(x_i \triangle r)$ is a resultant premise formula, then interchange $<$ with $\geq$; and $>$ with $\leq$; and remove the negation. For example, $(x_i \geq r)$ becomes $(x_i < r)$. Following the algorithm in [36], an equivalent disjunctive normal form for the premise formula in (12) is

$$\phi_{14d} = ((\neg a \wedge b \wedge c \wedge (d \geq 5) \wedge f) \vee (e \wedge f) \vee g), \tag{15}$$

and an equivalent conjunctive normal form is

$$\begin{aligned}\phi_{14c} =& ((\neg a \vee e \vee g) \wedge (b \vee e \vee g) \wedge (c \vee e \vee g) \\ & \wedge ((d \geq 5) \vee e \vee g) \wedge (f \vee g)).\end{aligned} \tag{16}$$

Finding the number of layers and the maximum number of nodes needed to implement the premise formulas of (15) and (16) with a multilayer perceptron, both equations require a three layer perceptron and, using the function N in (11), $N(\phi_{14d}) = 5$ and $N(\phi_{14c}) = 8$. For this example, the disjunctive normal form requires less layers and less nodes compared to the original form (14), while the conjunctive normal form needs less layers and more nodes. As another example, a disjunctive and a conjunctive normal form for the premise formula

$$\phi_{17} = ((((\neg a \vee b \vee c \vee (d \geq 5)) \wedge e) \vee f) \wedge g) \tag{17}$$

are

$$\begin{aligned}\phi_{17d} =& ((\neg a \wedge e \wedge g) \vee (b \wedge e \wedge g) \vee (c \wedge e \\ & \wedge g) \vee ((d \geq 5) \wedge e \wedge g) \vee (f \wedge g))\end{aligned} \tag{18}$$

and

$$\phi_{f17c} = ((\neg a \vee b \vee c \vee (d \geq 5) \vee f) \wedge (e \vee f) \wedge g), \tag{19}$$

respectively. If all three were implemented with multilayer perceptrons, ϕ_{17} would require five layers while ϕ_{17d} and ϕ_{17c} would require only three, and $N(\phi_{17}) = 6$ while $N(\phi_{17d}) = 8$ and $N(\phi_{17c}) = 5$. In this example, the disjunctive normal form requires less layers and more nodes compared to the original form, while the conjunctive normal form needs less layers and less nodes. These two examples illustrate that placing a formula in either a disjunctive or conjunctive normal form reduces the number of layers to a maximum of three but does not necessarily guarantee a reduction in the total number of nodes.

Next, for a premise formula already in the disjunctive or conjunctive normal form, the maximum number of layers required for the premise formula's multilayer perceptron is shown to be three, and the maximum number of nodes needed is explicitly stated. If a premise formula $\phi_d \in \Phi$ is in disjunctive normal form, then $\phi_d = (\gamma_1 \vee \gamma_2 \vee \cdots \vee \gamma_k), \gamma_i = (\beta_{i,1} \wedge \beta_{i,2} \wedge \cdots \wedge \beta_{i,hi})$ for $1 \leq i \leq k$, and $\beta_{i,j} = a$ or $\beta_{i,j} = \neg a$ where $a \in \boldsymbol{A}$ for $1 \leq j \leq h_i$

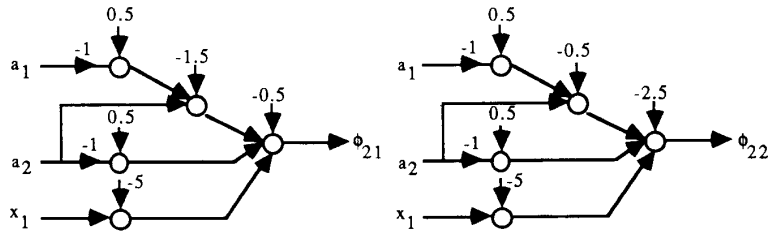


Fig. 6. The multilayer perceptrons for (21) and (22).

or $\beta_{i,j} = (x_\tau \triangle r)$ where $\boldsymbol{x} \in \boldsymbol{X}$, $\boldsymbol{x} = [x_1, x_2, \cdots, x_p]^t$, $r \in \mathbb{R}$, and $\triangle \in \{<, >, \leq, \geq\}$ for $1 \leq j \leq h_i$ and for $1 \leq \tau \leq p$. For further simplification, if $\beta_{i,j} = \neg(x_i \triangle r)$, interchange the appropriate predicate symbols and remove the negation as described above. Therefore, if the premise formula is in disjunctive normal form, a maximum of three layers are needed in the multilayer perceptron: the output layer for the disjunction of formulas, one hidden layer for the conjunction of formulas, and the input layer for the negation of formulas and relational formulas. The number three is a maximum since a premise formula is not required to have a conjunction, a disjunction, and either a negation or a relational formula in its structure.

Using (12), the maximum number of nodes needed by a premise formula in disjunctive normal form is derived. The output layer has one node with k inputs corresponding to the conjunction of the premise formula, so

$$N(\phi_d) = 1 + N(\gamma_1) + N(\gamma_2) + \cdots + N(\gamma_k).$$

The hidden layer has a maximum of k nodes with h_i inputs for $1 \leq i \leq k$ for each node corresponding to the disjunctions of premise formulas, so

$$\begin{aligned} N(\phi_d) =& 1 + [1 + N(\beta_{1,1}) + \ldots + N(\beta_{1,h_1})] \\ &+ [1 + N(\beta_{2,1}) + \cdots + N(\beta_{2,h_2})] \\ &+ \cdots + [1 + N(\beta_{k,1}) \\ &+ \cdots + N(\beta_{k,h_k})]. \end{aligned}$$

The input layer has a maximum of $(h_1 + h_2 + \cdots + h_k)$ one input nodes where each node either corresponds to a negation of a premise formula or a relational premise formula. Thus, the maximum number of nodes needed to implement a premise formula in disjunctive normal form is

$$\begin{aligned} N(\phi_d) &= 1 + [1 + h_1] + [1 + h_2] + \ldots + [1 + h_k] \\ &= 1 + k + \sum_{i=1}^{k} h_i. \end{aligned} \tag{20}$$

As an example, the formation of a multilayer perceptron for a premise formula in the disjunctive normal form is illustrated in Fig. 6(a) for the premise formula

$$\phi_{21} = ((\neg a_1 \wedge a_2) \vee \neg a_2 \vee (x_1 \geq 5)) \tag{21}$$

where $\boldsymbol{A} = \{a_1, a_2\}$ and $\boldsymbol{x} = [x_1]$.

If a premise formula $\phi_c \in \Phi$ is in conjunctive normal form, then $\phi_c = (\gamma_1 \wedge \gamma_2 \wedge \cdots \wedge \gamma_k)$, $\gamma_i = (\beta_{i,1} \vee \beta_{i,2} \vee \cdots \vee \beta_{i,h_i})$ for $1 \leq i \leq k$, and $\beta_{i,j}$ for $1 \leq j \leq h_i$ are the same as above for the disjunctive normal form. Clearly, as stated above for the disjunctive normal form, if the premise formula is in conjunctive normal form, the multilayer perceptron has a maximum of three layers: the output layer for the conjunction of formulas, one hidden layer for the disjunction of formulas, and the input layer for the negation of formulas and relational formulas. In addition, the maximum number of nodes which are needed to construct a multilayer perceptron for a conjunctive normal premise formula, $N(\phi_c)$, is given by (20). As an example, the formation of a multilayer perceptron for a conjunctive normal form premise formula is illustrated in Fig. 6(b) for

$$\phi_{22} = ((\neg a_1 \vee a_2) \wedge \neg a_2 \wedge (x_1 \geq 5)) \tag{22}$$

where $\boldsymbol{A} = \{a_1, a_2\}$ and $\boldsymbol{x} = [x_1]$.

So, given a premise formula, it has been shown that the number of layers in the multilayer perceptron implementation can be reduced to at most three if the premise formula is placed in a disjunctive or conjunctive normal form. In doing this, the processing time for the conflict set is inherently decreased if the number of layers in the original premise formula was greater than three. Although the number of layers can always be decreased to three or less, the possibility of increasing the number of nodes per layer may exist.

IV. CONCLUDING REMARKS

This paper presented the use of a multilayer perceptron to perform the match phase of rule-based AI systems (which include rule-based expert, planning, and learning systems). Using the proposed approach, the match phase is performed by matching in parallel all of the rules to all of the working memory elements. The result is the formation of the conflict set for the current situation of working memory. Given a set of rules with left-hand-sides following the prescribed syntax, the construction of a multilayer perceptron which implements the match phase was described. In addition, a formula for the maximum number of nodes needed to implement the multilayer perceptron and a procedure to reduce the number of layers of the multilayer perceptron were also given. The multilayer perceptron match phase approach was compared to a conventional match phase interpreter for production systems, the Rete Match Algorithm. As an example, the process for forming the multilayer perceptron is illustrated using a rule-based expert system in [32]. Next, some potential limitations to the multilayer perceptron approach are discussed.

Comparing the types of rules used by the multilayer perceptron match phase to the types of rules used by the Rete Match Algorithm match phase, the multilayer perceptron approach is able to represent many rules that can be described in OPS5. This approach can thus be used as the match phase for many production systems which have been described using OPS5. For a description of OPS5 rules, see [1]. To use the multilayer perceptron approach, an OPS5 rule's left-hand-side needs to be transformed into the appropriate premise formula. One limitation of this approach is that the left-hand-side of an OPS5 rule cannot contain certain variables (for example, a symbolic variable which takes on an infinite number of values). This limitation can be overcome, however, if these variables can be redefined so that

they take on only a finite number of values, and new rules formed per the finite values. Hence, the multilayer perceptron approach can implement the matching of variables for this case, but the number of new rules may become large, which does not decrease the speed of the multilayer perceptron but does increase its size.

Another potential limitation of the multilayer perceptron solution to the match phase problem, as well as another difference between the two match phase implementations (the multilayer perceptron and the Rete Match Algorithm), is the procedure for the adding of new working memory elements and new rules (sometimes referred to as "learning rules" or "automatic knowledge acquisition") while the inference process is executing. The Rete Match Algorithm uses extensive changes in its underlying software tree structure to represent added working memory elements and rules (as well as deleted rules). When a completely new working memory element or rule is added using the multilayer perceptron approach, new arcs, nodes, weights, and biases are augmented to the existing multilayer perceptron as required by the new working memory element or rule. (In a similar manner, old rules can be deleted by removing the appropriate parts of the multilayer perceptron.) When a new working memory element or rule is added, the time for the multilayer perceptron to process may not be affected because the depth of the network may not increase, but the size of the neural network will increase. If a rule needs to be changed, arcs, nodes, weights, and biases associated with the changed rule are affected, and the time and space required by the multilayer perceptron may increase. If the multilayer perceptron is implemented with hardware, as discussed in the future directions section of the Introduction, the adding, changing, or even deleting of a rule could be costly. For this reason, a rule-based AI system which requires a fast match phase should first be developed using software. Then once it is in its final form, the rule-based AI system can be implemented with a multilayer perceptron for greater processing efficiency.

REFERENCES

[1] L. Brownston, R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*. Reading, MA: Addison-Wesley, 1985.

[2] B. G. Buchanan and E. H. Shortliffe, *Rule-Based Expert Systems*. Reading, MA: Addison-Wesley, 1984.

[3] *Level 5 Expert System Software*, Information Builders Inc., New York, NY, 1989.

[4] C. L. Forgy, "On the efficient implementation of production systems," Ph.D. dissertation, Dep. Comput. Sci., Carnegie-Mellon Univ., Feb. 1979.

[5] ——, "Rete: A fast algorithm for the many pattern/many object pattern problem," *Artif. Intell.*, vol. 19, pp. 17–37, 1982.

[6] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[7] M. I. Schor, T. P. Daly, S. L. Ho, and B. R. Tibbitts, "Advances in Rete pattern matching," in *Proc. 1986 Nat. Conf. Artif. Intell.*, 1986, pp. 226–232.

[8] S. J. Stolfo, "Five parallel algorithms for production system execution on the DADO machine," in *Proc. Nat. Conf. Artif. Intell.*, 1984, pp. 300–307.

[9] D. P. Miranker, "Performance estimates for the DADO machine: A comparison of TREAT and RETE," in *Proc. Int. Conf. Fifth Generation Comput. Syst.*, 1984, pp. 449–457.

[10] A. Gupta, C. L. Forgy, A. Newell, and R. Wedig, "Parallel algorithms and architectures for rule-based systems," in *Proc. 13th Int. Symp. Comput. Architecture*, 1986, pp. 28–37.

[11] T. Ishida and S. Stolfo, "Toward the parallel execution of rules in production system programs," in *Proc. 1985 Int. Conf. Parallel Processing*, 1985, pp. 568–575.

[12] K. Oflazer, "Partitioning in parallel processing of production systems," in *Proc. 1984 Int. Conf. Parallel Processing*, 1984, pp. 92–100.

[13] D. I. Moldovan, "A model for parallel processing of production systems," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 1986, pp. 568–573.

[14] M. F. M. Tenorio and D. I. Moldovan, "Mapping production systems into multiprocessors," in *Proc. 1985 Int. Conf. Parallel Processing*, 1985, pp. 56–62.

[15] H. Won, "On parallel processing of universal rule based expert system," in *Proc. 1987 Western Conf. Expert Syst.*, 1987, pp. 136–143.

[16] C. Forgy, A. Gupta, A. Newell, and R. Wedig, "Initial assessment of architectures for production systems," in *Proc. 1984 Nat. Conf. Artif. Intell.*, 1984, pp. 116–120.

[17] C. Forgy and A. Goopta, "Preliminary architecture of the CMU production system machine," in *Proc. Nineteenth Annu. Hawaii Int. Conf. Syst. Sci.*, 1986, pp. 194–200.

[18] A. O. Oshisanwo and P. P. Dasiewicz, "A parallel model and architecture for production systems," in *Proc. 1987 Int. Conf. Parallel Processing*, 1987, pp. 147–153.

[19] F. Schreiner and G. Zimmermann, "PESA 1—A parallel architecture for production systems," in *Proc. 1987 Int. Conf. Parallel Processing*, 1987, pp. 166–169.

[20] R. Ramnarayan, G. Zimmermann, and S. Krolikoski, "PESA-1: A parallel architecture for OPS5 production system," in *Proc. Nineteenth Annu. Hawaii Int. Conf. Syst. Sci.*, 1986, pp. 201–205.

[21] S. J. Stolfo and D. P. Miranker, "DADO: A parallel processor for expert systems," in *Proc. 1984 Int. Conf. Parallel Processing*, 1984, pp. 74–82.

[22] A. Gupta, "Implementing OPS5 production systems on DADO," in *Proc. 1984 Int. Conf. Parallel Processing*, 1984, pp. 83–91.

[23] D. E. Shaw, "NON-VON: A parallel machine architecture for knowledge-based information processing," in *Proc. Seventh Int. Joint Conf. Artif. Intell.*, 1981, pp. 961–963.

[24] A. Gupta, C. L. Forgy, D. Kalp, A. Newell, and M. Tambe, "Parallel OPS5 on the Encore Multimax," in *Proc. 1988 Int. Conf. Parallel Processing*, vol. 1, 1988, pp. 271–280.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the MicroStructrure of Cognition. Vol. 1: Foundations*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986.

[26] J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Nat. Acad. Sci.*, pp. 3088–3092, May 1984.

[27] M. A. Sartori and P. J. Antsaklis, "Neural computing and production systems," in *Proc. Third Annu. Symp. Intell. Contr.*, Aug. 1988.

[28] K. M. Passino, M. A. Sartori, and P. J. Antsaklis, "Neural computing for numeric/symbolic conversion in control systems," *IEEE Contr. Syst. Mag.*, Apr. 1989, pp. 44–52.

[29] ——, "Neural computing for information extraction in control systems," in *Proc. Twenty-Sixth Annu. Allerton Conf. Commun., Contr., Comput.*, Univ. of Illinois at Urbana-Champaign, Sept. 1988, pp. 1172–1180.

[30] R. P. Lippmann, "An introduction to computing with neural networks," *IEEE ASSP Mag.*, Apr. 1987, pp. 4–22.

[31] M. A. Sartori, K. P. Passino, and P. J. Antsaklis, "Artificial neural networks in the match phase of rule-based expert systems," in *Proc. Twenty-Seventh Annu. Allerton Conf. Commun., Contr., Comput.*, Univ. of Illinois at Urbana-Champaign, Sept. 1989.

[32] ——, "An artificial neural network solution to the match phase problem of rule-based artificial intelligence systems," Tech. Rep. 89–09–02, Dep. Elec. Eng., Univ. of Notre Dame, Sept. 1989.

[33] J. McDermott and C. Forgy, "Production system conflict resolution strategies," in *Pattern-Directed Inference Systems*, D. A. Waterman, F. Hayes-Roth, Eds. New York: Academic, 1978.

[34] T. Kohonen, *"Self-Organization and Associative Memory*, 2nd ed. New York: Springer-Verlag, 1988.

[35] Z. Kohavi, *Switching and Finite Automata Theory*. New York: McGraw-Hill, 1978.

[36] E.J. Lemmon, *Beginning Logic*. Indianapolis, IN: Hackett, 1978.