Proceedings of the 27th Annual
Allerton Conference on
Communication, Control, and Computing,
Univ. of Illinois at Urbana-Champaign,
September 27-29, 1989

ARTIFICIAL NEURAL NETWORKS IN THE MATCH PHASE OF RULE-BASED EXPERT SYSTEMS

MICHAEL A. SARTORI, KEVIN M. PASSINO, and PANOS J. ANTSAKLIS
Department of Electrical and Computer Engineering
University of Notre Dame
Notre Dame, IN 46556

Abstract

In rule-based expert 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, an artificial neural network is used to solve the match phase problem for rule-based expert 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 form an artificial neural network (here, the multi-layer perceptron) that finds all of the rules which can be executed for the current situation in working memory. The conventional Rete Match Algorithm approach is compared to the novel approach presented here.

1 INTRODUCTION

In rule-based expert 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*. 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 solving the match phase problem via an artificial neural network. 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. A brief overview of where the match phase problem is found in rule-based expert systems is now given and is followed by a brief description of the Rete Match Algorithm, the conventional solution to the match phase problem.

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:

IF (antecedent) THEN (consequence) (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*) is a dynamically changing memory which contains the 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]:

(i) *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.
(ii) *Select*. Choose one rule from the conflict set to execute.
(iii) *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 an artificial neural network implementation of the match phase as described in this paper.

The conventional solution to the match phase problem (particularly for rule-based expert systems) utilizes the Rete Match Algorithm [4]. If the expert system's rules are explicitly of the form depicted in equation (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 [4,5] 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. The time needed to perform the match phase using the Rete Match Algorithm can be quantified. 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 artificial neural network solution to the match phase problem is used, the processing time is independent of both the number of rules and the number of working memory elements.

Since the Rete Match Algorithm's introduction, other Rete based algorithms which attempt to increase the speed of the interpreter have been introduced [7-14]. To further reduce the amount of time consumed by the interpreter in the match phase, special hardwares have been developed using parallelism and multiprocessor architectures [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]. Almost all of these attempts are based, at least in part, on the Rete Match Algorithm, which is assumed to yield the most efficient match phase for a production interpreter. 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 [11,12,14]. 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.

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 PEs	[21,10]
MAPPS	10,000	-	128 PEs	[18]
NON-VON	2000	903	16032 PEs	[18]
Olfazer's	4500	-	512 Ps	[10]
PESA-1	25,000	8000	32 PEs	[19]

WME = working memory element
- = not available
Ps = processors
PEs = processing elements

The method proposed here for the match phase is quite different from the Rete Match Algorithm and the above mentioned modifications; the approach described in this paper simultaneously matches <u>all</u> of the rules to <u>all</u> of the working memory elements in parallel via an artificial neural network and, as a result, produces the conflict set. Artificial neural networks are chosen here because of

their ability to perform fast computations in a massively parallel fashion. The approach to solving the match phase problem with an artificial neural network begins by first defining "premise formulas" which represent the left-hand-sides of the rules. The match phase problem is defined in Section 2 as the determination of the truth of the premise formulas for the current working memory. The artificial neural network that is used here to implement the match phase is the multi-layer perceptron. In Section 3, a procedure for designing a multi-layer perceptron for a single premise formula is described and extended to one for designing a multi-layer perceptron for a set of given premise formulas. The input to the multi-layer perceptron is the working memory, and the output is the conflict set. The specially designed multi-layer 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 use of an artificial neural network as the match phase in the three phase cycle of a rule-based expert system is further explained in Section 3. Finally in Section 4, concluding remarks are made including a citing of some of the potential limitations in using the artificial neural network solution for the match phase. Note that this work is an extension of the research reported in [25] via the results in [26,27]. The artificial neural network used in the previous work was the model termed the "ProNet", which was obtained by modifying the Hamming net of [28]. The multi-layer perceptron was used rather than the ProNet because the multi-layer perceptron is better able to address a greater diversity of types of rules as well as rules which contain real numbers.

2 THE MATCH PHASE PROBLEM

In this section, the premise formulas for the left-hand-sides of the expert system's rules are defined, and the match phase phase problem is defined in terms of the premise formulas and the working memory. The premise formulas correspond to the left-hand-sides of the expert system's rules. Rule-based expert 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 an artificial neural network as discused in the following sections.

The syntax defining the premise formulas is now described. Let $|S|$ denote the number of elements in the set S. Let $A = \{a_1, a_2, ..., a_n\}$ be a nonempty finite set of *atomic propositions* a_i, where $|A| = n$. 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 $x \in X$ represents numeric information in working memory as discussed below. The *premise formulas* are composed of the following symbols:

(a) Atomic propositions $a \in A$,
(b) Real numbers x_i for $1 \le i \le p$, where $x \in X$ and $x = [x_1, x_2, ..., x_p]^t$ (and "t" denotes transpose),
(c) The Boolean connectives: $\neg$ (negation), $\wedge$ (disjunction), and $\vee$ (conjunction),
(d) The set $\mathbb{R}$ of real numbers,
(e) The predicate symbols: $>$ (greater than), $<$ (less than), $\ge$ (greater than or equal to), and $\le$ (less than or equal to), and
(f) The parentheses: (and).

The rules for forming premise formulas are as follows:

(1) A single atomic proposition $a \in A$ 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 $x \in X$, $x = [x_1, x_2, ..., x_p]^t$, and $r \in \mathbb{R}$, then $(x_i > r)$, $(x_i < r)$, $(x_i \ge r)$, and $(x_i \le r)$ for any i such that $1 \le i \le p$ 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 $x \in X$, $x = [x_1, x_2, ..., x_p]^t$, and $r \in \mathbb{R}$, then $(x_i = r)$ can be replaced by the equivalent premise formula $((x_i \ge r) \wedge (x_i \le r))$, and $(x_i \ne r)$ by the equivalent premise formula $((x_i > r) \vee (x_i < r))$ for $1 \le i \le p$.

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

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

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

Let $\Phi = \{\phi_1, \phi_2, .., \phi_m\}$ denote a finite set of premise formulas. Let $Y \subset \{0,1\}^m$. 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 $\mathbf{y_k} = [y_1, y_2, ..., y_m]^t$ for $1 \le i \le m$. If $y_i = 1$ for $1 \le i \le m$, 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, $\mathbf{y_k} \in Y$ represents the conflict set.

The match phase problem can be solved by implementing the function

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

with a multi-layer perceptron. The utilization of the multi-layer perceptron to perform the match phase in a rule-based expert system is illustrated in Figure 1. The input to the multi-layer perceptron, which describes the working memory, is the vector $\mathbf{u_k}$. 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 multi-layer perceptron as its weights, biases, and interconnections. The output of the multi-layer perceptron is the vector $\mathbf{y_k}$, which denotes the conflict set. Thus, the multi-layer perceptron specifies the conflict set $\mathbf{y_k} \in Y$ at each time instant k for all possible inputs $\mathbf{u_k} \in U$ and premise formulas $\phi \in \Phi$, and hence implements the function P.

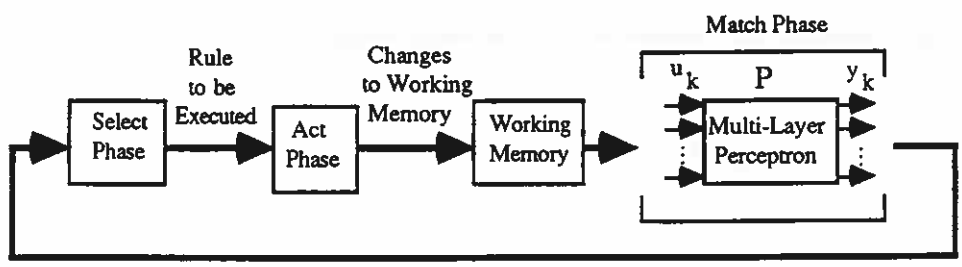

Figure 1 The multi-layer perceptron implementation of the match phase in a rule-based expert system.

Following Figure 1, from the output vector $\mathbf{y_k}$, 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 [1,2,29]. 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 $\mathbf{v_{k+1}}$ is produced by altering the vector $\mathbf{v_k}$ such that some of the elements $V(a_i)$ are changed form one to zero (indicating that a_i becomes false) and others from zero to one (indicating that a_i becomes true). The vector $\mathbf{x_k}$ can also be changed according to the fired rule to produce the vector $\mathbf{x_{k+1}}$. So, the working memory is updated, and the input vector $\mathbf{u_k}$ is changed to $\mathbf{u_{k+1}}$. The new input vector $\mathbf{u_{k+1}}$ is used as an input to the multi-layer perceptron which gives the output $\mathbf{y_{k+1}}$, and the process is repeated. Next, it is shown how to construct a multi-layer perceptron which implements the function P and hence solves the match phase problem.

3 AN ARTIFICIAL NEURAL NETWORK 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 *multi-layer perceptron* is used as the model for solving the match phase problem. The multi-layer perceptron is a feed forward model which is restricted here so that it is partially connected and also so that it does not self-adjust its weights or learn.

Instead of using the multi-layer perceptron, other 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 [30], but because it uses feedback to determine its output, it requires time to converge to a minimum; the multi-layer perceptron always determines its outputs immediately since it is a feed forward 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 [31], may not be as useful as the multi-layer 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 which use supervised training, such as the Back Propagation Training Algorithm applied to the multi-layer perceptron [32], also may not be as useful since the weights for the network can be selected as explained in Section 3.1., and so the training of the weights of the multi-layer perceptron is not required. In addition, problems which occur in learning algorithms which 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 [30]. For these reasons, the multi-layer perceptron was judged as 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.

3.1 The Multi-Layer Perceptron

The multi-layer perceptron is a feed forward artificial neural network considered here to contain at least one *hidden layer* between the *input* and the *output layers*. A multi-layer perceptron with one hidden layer and with two nodes in each layer is illustrated in Figure 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 multi-layer perceptron is applied to the input layer. The input to the multi-layer perceptron considered here is the vector **u** with components u_i, which contains (n+p) continuous real valued elements. The vector **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 **z''**, which contains r binary elements, is the output of the hidden layer and the input to the output layer. The vector **y** with components y_i, which contains m binary elements, is the output of the multi-layer perceptron.

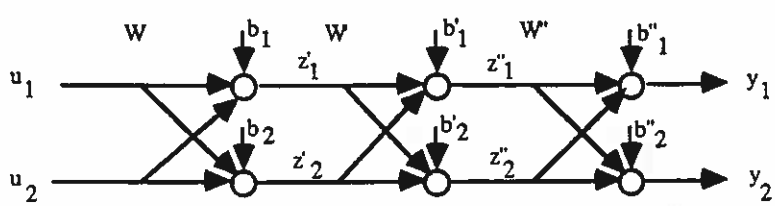

Figure 2 The multi-layer perceptron for two nodes at each layer.

In Figure 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 labelled with the weights) that are

between **u** and the input nodal layer, between **z'** and the hidden nodal layer, and between **z''** and the output layer. Note that unlike the traditional three layer perceptron used with the Back Propagation Algorithm, the input layer can assume non-unity 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 which is passed through a threshold nonlinearity. The result is a binary output for each layer. Two typical threshold nonlinearities are illustrated in Figure 3. Note that in this paper, nodes which use the threshold of Figure 3(a) are unshaded and those which use the threshold of Figure 3(b) are shaded.

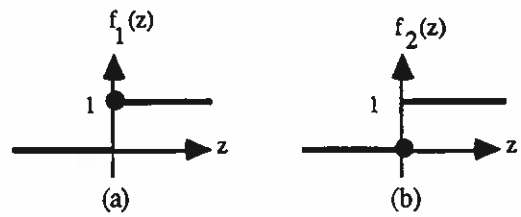

Figure 3 Threshold nonlinearities for the multi-layer perceptron.

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 Figure 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 multi-layer perceptron is specified. Using this description of the multi-layer perceptron, the technique to determine the weights, biases, and number of nodes for the multi-layer perceptron used in solving the match phase problem is given next.

3.2 Construction of the Multi-Layer Perceptron

The construction of a multi-layer perceptron for one premise formula is examined first. Given a single premise formula $\phi \in \Phi$, an artificial neural network is constructed which indicates whether or not ϕ is true for any situation in working memory $\mathbf{u_k} \in U$. The construction of the multi-layer 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).

<u>Negation of Formulas:</u>

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 Figure 3(a). The network for the premise formula

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

is shown in Figure 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, ..., j, and consider premise formulas either of the type $(\sigma_1 \wedge \sigma_2 \wedge \ldots \wedge \sigma_j)$ or $(\sigma_1 \vee \sigma_2 \vee \ldots \vee \sigma_j)$. Form a node with j inputs and 1 output. The weights are unity and the nonlinearity used is in Figure 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 Figure 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 $\mathbf{x} = [x_1, x_2, \ldots, x_p]^t$ 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 **x** in the input vector **u** of the multi-layer perceptron. If the predicate symbol is > or ≥, then the weight is 1 and the bias is -r. If the predicate symbol is < or ≤, then the weight is -1 and the bias is r. If the predicate symbol is ≤ or ≥, use the nonlinearity of Figure 3(a). If the predicate symbol is < or >, use the nonlinearity of Figure 3(b). The network for the premise formula

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

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

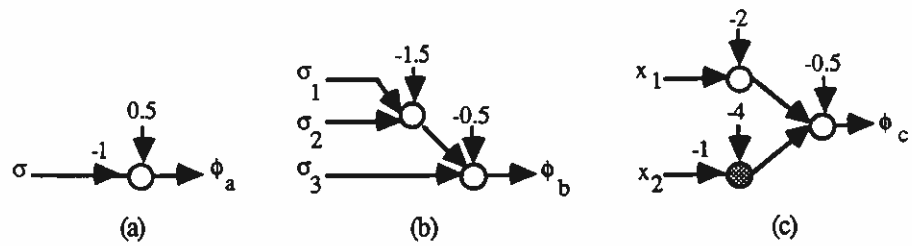

Figure 4 The multi-layer perceptron for premise formulas (a) $\phi_a = \neg\sigma$, (b) $\phi_b = ((\sigma_1 \wedge \sigma_2) \vee \sigma_3)$, and (c) $\phi_c = ((x_1 \geq 2) \vee (x_2 < -4))$.

Once the nodes of the multi-layer 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 multi-layer 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 multi-layer perceptron can be mechanized. The entire process for constructing a multi-layer perceptron for a premise formula is illustrated in Figure 5 with the premise formula

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

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

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. Thus, one combined multi-layer perceptron implements all of the premise formulas in the set Φ. Notice that the possibility exists for automating the development of the multi-layer perceptron once the set Φ of premise formulas is given.

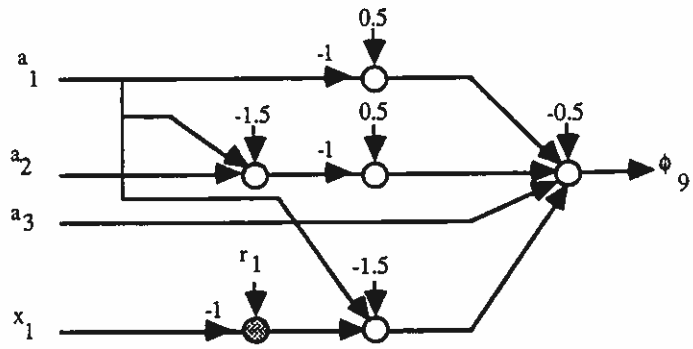

Figure 5 The multi-layer perceptron for the premise formula of equation (9).

As stated in Section 1, the time for the multi-layer perceptron to process the match phase is independent of both the number of working memory elements and the number of rules. Using the artificial neural network 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 $\mathbf{u_k}$ of to the multi-layer perceptron, and thus the working memory size is not detrimental to 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 $\mathbf{y_k}$ of the multi-layer perceptron. If the multi-layer perceptron is physically implemented, only the number of layers adversely affects the time needed for the multi-layer perceptron to process.

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 multi-layer perceptron proposed here is also similar to the threshold logic circuits (networks) described in [33], except that in these circuits the theory only allows a finite number of discrete inputs. Such threshold logic networks could however be used to implement the match phase.

4 CONCLUDING REMARKS

In this paper, it was proposed to use an artificial neural network to perform the match phase of rule-based expert 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 of doing this 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 multi-layer perceptron which implements the match phase was described. The multi-layer perceptron match phase approach was compared to a conventional match phase interpreter for production systems, the Rete Match Algorithm. Note that an extended version of the work in this paper appears in [34]. These extensions include a formula for the maximum number of nodes needed to implement any premise formula with a multi-layer perceptron, a procedure for reducing the number of layers in a multi-layer perceptron implementing any premise formula to at most three, and as an example of the proposed match phase approach, the construction of a multi-layer perceptron for the rules of a rule-based expert system in [1]. Next, some potential limitations to the artificial neural network approach are discussed.

Comparing the types of rules used by the artificial neural network match phase to the types of rules used by the Rete Match Algorithm match phase, the artificial neural network approach is able to represent many rules which can be described in OPS5, and thus can be used as the match phase for many production systems which have been described using OPS5. For a description of OPS5 rules, see [1]. The transformation (which is often immediate in most cases) of an OPS5 rule's left-hand-side into the appropriate premise formula is the step required in order to use the artificial neural network approach. One limitation of this approach is that the left-hand-side of an OPS5 rule can not 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.

Another potential limitation of the multi-layer perceptron implementation of the match phase as well as another difference between the two match phase implementations (the multi-layer perceptron and the Rete Match Algorithm) is the procedure for the forming of new rules while the inference process is executing (sometimes referred to as "learning rules" or "automatic knowledge acquisition"). The Rete Match Algorithm uses changes in its software tree structure to represent added rules (and deleted rules). When a new rule is added using the artificial neural networks approach, new arcs, nodes, weights, and biases are augmented to the already existing multi-layer perceptron as cited by the new rule. (In a similar manner, old rules can be deleted by removing the appropriate parts of the multi-layer perceptron.) When a rule is added, the time for the multi-layer perceptron to process may not be affected because the depth of the network may not increase, but the size of the neural network will undoubtedly 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 multi-layer perceptron may increase. If the multi-layer perceptron is implemented with hardware, the adding, changing, or even deleting of a rule could be costly. For this reason, a rule-based expert system which requires a fast match phase should first be developed using software. Then once it is in its final form, the rule-based expert system can be implemented with a multi-layer perceptron to more efficiently utilize the proposed artificial neural network approach.

5 REFERENCES

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

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

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

[4] Forgy C.L., *On the Efficient Implementation of Production Systems*, Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, February 1979.

[5] Forgy C.L., "Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Problem", *Artificial Intelligence*, Vol. 19, pp. 17-37, 1982.

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

[7] Schor M.I., Daly T.P., Ho S.L., Tibbitts B.R., "Advances in Rete Pattern Matching", *Proceedings of the 1986 National Conference on Artificial Intelligence*, pp. 226-232, 1986.

[8] Stolfo S.J., "Five Parallel Algorithms for Production System Execution on the DADO Machine", *Proceedings of the National Conference on Artificial Intelligence*, pp. 300-307, 1984.

[9] Miranker D.P., "Performance Estimates for the DADO Machine: A Comparison of TREAT and RETE", *Proceedings of the International Conference on Fifth Generation Computer Systems*, pp. 449-457, 1984.

[10] Gupta A., Forgy C.L., Newell A., Wedig R., "Parallel Algorithms and Architectures for Rule-Based Systems", *Proceedings of the 13th International Symposium on Computer Architecture*, pp. 28-37, 1986.

[11] Ishida T., Stolfo S., "Towards the Parallel Execution of Rules in Production System Programs", *Proceedings of the 1985 International Conference on Parallel Processing*, pp. 568-575, 1985.

[12] Oflazer K., "Partitioning in Parallel Processing of Production Systems", *Proceedings of the 1984 International Conference on Parallel Processing*, pp. 92-100, 1984.

[13] Moldovan D.I., "A Model for Parallel Processing of Production Systems", *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pp. 568-573, 1986.

[14] Tenorio M.F.M., Moldovan D.I., "Mapping Production Systems into Multiprocessors", *Proceedings of the 1985 International Conference on Parallel Processing*, pp. 56-62, 1985.

[15] Won H., "On Parallel Processing of Universal Rule Based Expert System", *Proceedings of the 1987 Western Conference on Expert Systems*, pp. 136-143, 1987.

[16] Forgy C., Gupta A., Newell A., Wedig R., "Initial Assessment of Architectures for Production Systems", *Proceedings of the 1984 National Conference on Artificial Intelligence*, pp. 116-120, 1984.

[17] Forgy C., Goopta A., "Preliminary Architecture of the CMU Production System Machine", *Proceedings of the Nineteenth Annual Hawaii International Conference on System Sciences*, 1986, pp. 194-200, 1986.

[18] Oshisanwo A.O., Dasiewicz P.P., "A Parallel Model and Architecture for Production Systems", *Proceedings of the 1987 International Conference on Parallel Processing*, pp. 147-153, 1987.

[19] Schreiner F., Zimmermann G., "PESA 1 - A Parallel Architecture for Production Systems", *Proceedings of the 1987 International Conference on Parallel Processing*, pp. 166-169, 1987.

[20] Ramnarayan R., Zimmermann G., Krolikoski S., "PESA-1: A Parallel Architecture for OPS5 Production System", *Proceedings of the Nineteenth Annual Hawaii International Conference on System Sciences*, pp. 201-205, 1986.

[21] Stolfo S.J., Miranker D.P., "DADO: A Parallel Processor for Expert Systems", *Proceedings of the 1984 International Conference on Parallel Processing*, pp. 74-82, 1984.

[22] Gupta A., "Implementing OPS5 Production Systems on DADO", *Proceedings of the 1984 International Conference on Parallel Processing*, pp. 83-91, 1984.

[23] Shaw D.E., "NON-VON: A Parallel Machine Architecture for Knowledge-Based Information Processing", *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pp. 961-963, 1981.

[24] Gupta A., Forgy C.L., Kalp D., Newell A., Tambe M., "Parallel OPS5 on the Encore Multimax," *Proceedings of the 1988 International Conference on Parallel Processing, Vol. 1*, pp. 271-280, 1988.

[25] Sartori M.A., Antsaklis P.J., "Neural Computing and Production Systems," *Proceedings of the Third Annual Symposium on Intelligent Control*, Aug. 1988.

[26] Passino K.M., Sartori M.A., Antsaklis P.J., "Neural Computing for Numeric/Symbolic Conversion in Control Systems," *IEEE Control Systems Magazine*, pp. 44-52, April 1989.

[27] Passino K.M., Sartori M.A., Antsaklis P.J., "Neural Computing for Information Extraction in Control Systems," *Proceedings of the Twenty-Sixth Annual Allerton Conference on Communications, Control, and Computing*, University of Illinois at Urbana-Champaign, pp. 1172-1180, September 1988.

[28] Lippmann R.P., "An Introduction to Computing with Neural Networks," *IEEE ASSP Magazine*, pp.4-22, April 1987.

[29] McDermott J., Forgy C., "Production System Conflict Resolution Strategies," D.A. Waterman, F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems*, Academic Press, New York, 1978.

[30] Hopfield J., "Neurons with Graded Response have Collective Computational Properties like Those of Two-State Neurons", *Proceedings of the National Academy of Science*, pp. 3088-3092, May 1984.

[31] Kohonen T., *Self-Organization and Associative Memory*, Springer-Verlag, New York, NY, 2nd edition, 1988.

[32] Rumelhart D.E., Hinton G.E., Williams R.J., "Learning Internal Representations by Error Propagation", in Rumelhart D.E., McClelland J.L. (Eds.), *Parallel Distributed Processing: Explorations in the MicroStructrure of Cognition. Vol. 1: Foundations*, MIT Press, Cambridge, MA, 1986.

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

[34] Sartori M.A., Passino K.P., Antsaklis P.J., "An Artificial Neural Network Solution to the Match Phase Problem in Rule-Based Artificial Intelligence Systems," Technical Report #89-09-02, Department of Electrical and Computer Engineering, University of Notre Dame, September 1989.