Integrated Rule Based Learning and Inference

Ioannis Hatzilygeroudis, *Member, IEEE,* **and Jim Prentzas**

Abstract— Neurules are a kind of integrated rules integrating neurocomputing and production rules. Each neurule is represented as an adaline unit. Thus, the corresponding neurule base consists of a number of autonomous adaline units (neurules). In this paper, we present the contstruction process and the inference mechanism of neurules and explore their generalization capabilities. The construction process, which also implements corresponding learning algorithm, creates neurules from a given empirical dataset. The inference mechanism of neurules is integrated in its nature; it combines neurocomputing with symbolic processes. It is also, interactive, i.e. it interacts with the user asking him/her to provide values for some variables necessary to carry on inference. As shown via experiments, the neurules integrated inference mechanism is more efficient than the inference mechanism used in connectionist expert systems. Furthermore, neurules generalize much better than its constituent neural component (adaline unit) and is comparable to the back-propagation neural net (BPNN).

Index Terms—Neuro-symbolic integration, Integrated inference, Rule based reasoning, Neurocomputing

1 Introduction

The combination of different problem solving methods is a very active research area in Artificial Intelligence. The aim is to create combined formalisms or systems that benefit from each of their components. It is generally believed that complex problems can be easier solved with such combinations [37].

One of the most popular types of combination is that of symbolic and connectionist approaches, usually called the *neuro-symbolic* approach, which has yielded advanced problem solving formalisms and systems [7], [29], [36], [51], [10], [8], [15], [27], [2], [16], [20], [23]. Although within neuro-symbolic approaches there have been advanced efforts at combining neural networks with first-order logic [43], [22], [4] or with multi-valued logic [33] or even with non-classical logics [17], [18], [19], [34], those that combine symbolic rules (of propositional type) and neural networks still possess a great part [13], [50], [31], [48], [14], [24], [25] and seem to have given more applied results [38], [52], [53], [54].

The success of such combinations is based on the fact that the two component formalisms/methods have complementary advantages and disadvantages (see [27] for a detailed account of them). Symbolic rules offer a number of advantages for knowledge representation such as, naturalness and modularity (see e.g. [42]). Naturalness refers to how easily one can interpret expressions that represent knowledge, whereas modularity refers to the fact that each rule is an autonomous unit. The latter allows for incremental construction of a rule base, which is a quite important feature. Furthermore, rule based systems provide an interactive inference mechanism, which guides the user in supplying input values, and an explanation mechanism, which justifies the reached conclusions. However, symbolic rules have also some deficiencies. The most important of them is the difficulty in acquiring rules from the experts, a problem known as the knowledge acquisition bottleneck.

Neural networks represent a totally different approach to problem solving, known as connectionism (see e.g. [13]). The main advantages of neural networks are their ability to obtain their knowledge from training examples (thus reducing the interaction with the experts to a minimum), their ability to generalize and represent complex and imprecise knowledge and their high level of efficiency. Their main drawbacks, compared to symbolic rules, are the lack of naturalness and modularity and the difficulty (if not inability) in providing explanations. This latter fact makes them to look like black boxes, unnatural and thus incomprehensible.

The combination of symbolic rules and neural networks can result into various, so called, neuro-symbolic representations. We concentrate on combinations that result in a uniform, seamless combination of the two component approaches. This type of combinations are called unified, according to [29], or integrated, according to [2]. However, most (if not all) of existing approaches that integrate rules and neural networks see the integration from the point of view of connectionism, i.e. the main component is the connectionist one in which somehow symbolic processing is incorporated in or mapped to. So, they do not provide the functionalities of a rule-based system, like e.g. interactive inference.

We have introduced neurules [24], [25], a type of symbolic-neural rules combining symbolic rules (of propositional type) and neurocomputing. Their high-level characteristic is that they give pre-eminence to the symbolic part

- *I. Hatzilygeroudis is with the Department of Computer Engineering and Informatics, University of Patras, Greece. E-mail: ihatz@ceid.upatras.gr*
- *J. Prentzas is with the Department of Education Sciences in Pre-School Age, Democritus University of Thrace, Greece. E-mail: dprentza@psed.duth.gr.*

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.

of the integration. As a result, they retain the naturalness and modularity of symbolic rules to a large degree and are able to possess an interactive inference mechanism. One way of constructing neurules is from empirical data. Although we presented the corresponding construction process [25], we haven't formally and completely presented the corresponding inference mechanism.

So, the main concern of this paper is the inference mechanism of neurules, initial aspects of which were introduced in [26]. Neurules possess an interactive, integrated inference mechanism, which is proved to be more efficient than other actually pure connectionist mechanisms. For the sake of completeness, we also present the construction process, in a more formal way than in [25]. Finally, we explore the generalization capabilities of neurules. Neurules are proved to generalize quite well.

The structure of the paper is as follows. Section 2 treats related work. Section 3 presents the neurule-based knowledge representation scheme. In Section 4, production of neurules from empirical data is presented. Section 5 deals with the inference mechanism. Section 6 presents experimental results regarding efficiency and generalization capabilities of neurules. Section 7 presents use of neurules in an application and finally, Section 8 concludes.

2 Related work

A main research direction at combining rules and neural networks involves use of prior domain knowledge in neural network configuration [31, 50]. Such approaches are applied when both domain theory and training data are available and can be summarized in two steps: map the domain theory onto a neural network and then use training data to train the network. The term Knowledge-Based Neural Networks (KBNNs) is usually used to describe such approaches. Most often prior domain theory involves a core of propositional rules.

Two seminal works, in the above research direction, has led to two different research trends. The one trend stems from [31], where a connectionist network is developed that implements the meaning function of a propositional (definite) logic program. Following that line of research, a number of more advanced systems have been proposed. Direct successors of [31] extend it to first-order logic programs [32], [3], [4]. A feed-forward network, called 'core', is used to approximate the meaning function of a first-order logic program. Having connected the output layer of the 'core' network to its input layer, after a number of iterations, it converges to a stable state that corresponds to the model of the logic program [4]. Furthermore, the work in [33] extends the work in [31] to multi-valued logic based programs. Also, extensions to non-classical logics have been proposed. For example, in [17] a language for a connectionist temporal logic of knowledge (CTLK) is presented alongside a translation algorithm that translates CTLK theories into ensembles of neural nets. Similarly, in [18], intuitonistic reasoning is achieved via ensembles of neural nets. Also, in [19] a modal logic is modeled via ensembles of neural nets, which represent different modalities.

Approaches in the above research trend primarily focus on modeling symbolic processes in a neural way. So, they do not really provide an integrated inference mechanism and an interactive inference process.

The other trend stems from [50], where a background domain theory in the form of propositional rules is used to construct an initial feed-forward neural network, which is then finally trained through a training dataset. This leads to more efficient training and theory refinement [49], [30]. Such approaches differ among each other in various aspects such as the type of prior rules, weight semantics in the initially constructed neural network, learning algorithm, node characteristics and type of neural network. Prior rules can be categorical as in [50], [11], [6], [52], certainty factor rules as in [35], [11], fuzzy rules or in the form of deterministic finite state automata as in [39]. Training in such approaches is usually based on supervised learning. However, other types of learning such as reinforcement learning may be employed, as in [47]. KBNNs constructed with prior rules have been used in various applications such as medicine, bioinformatics, recognition of handwritten words [38], monitoring and diagnosis of manufacturing processes [53], product definition [54] and travel mode choice [52]. Approaches in the above research vein primarily focus on learning and not on reasoning or inference. The produced network can be used afterwards for producing outputs via neurocomputing methods. So, integrated inference capabilities are missing from the above approaches.

On the other hand, connectionist expert systems are considered as integrated systems that represent relationships between concepts, associated with nodes in a neural network. Weights are set in a way that makes the network to infer correctly. MACIE [12], [13] is such a system. Two characteristics of MACIE are: its ability to reason from partial data and its ability to provide explanations in the form of if-then rules. Inference in MACIE is controlled by the 'confidence' metric for each cell, which is an indication of how much close the corresponding cell is to be activated. The output cell with the maximum confidence is next considered. To improve performance of connectionist expert systems, the 'recency inference engine' (RIE) is introduced in [21]. RIE uses 'convergent ratio', a metric similar to 'confidence', for any cell in the network. The cell with maximum 'convergent ratio' is next considered. Furthermore, Sima presented a connectionist expert system shell called EXPSYS [45]. EXPSYS is also an improvement of MACIE, since it provides interactive inference engine and explanation mechanism for multilayer neural networks trained with back-propagation and using a differentiable activation function [44]. To handle partial input information, the concept of interval state is introduced for the network neurons and back-propagation is generalized for neural networks with such neurons. The introduction of the interval states, though, degrades the comprehensibility of the network compared to Gallant's approach and, furthermore, makes the inference process more complicated. The inference process provides the user with partial conclusions and confidences when some input values have been supplied.

All the above approaches have a common ground: they map or simulate or implement one or more symbolic models or processes into a neural network. In other words, they give pre-eminence to the connectionist component of the integration. This creates problems regarding the naturalness and modularity of their knowledge bases [25]. The produced representation remains unatural, hence incomprehensive. Also, in most of them modularity is not an attribute. Some kind of modules are created in [17], [18] and [19], in the form of ensembles of simpler neural nets. Also, there are modules in [43], in the form of functional focal-clusters (FFCs), which are clusters of connectionist nodes. However, all those types of modules are less fine grained than neurules, more complicated and not totally independent.

Also, functionalities like interactive inference and provision of explanations are not supported by the above approaches, except connectionist expert systems (e.g. MACIE, RIE). However, inference is done in an implicit and incomprehensive way and some of the explanation rules are meaningless. This is due to the fact that during construction they use some random cells that have no concepts assigned to them. The presence of those random cells also makes representation meaningless at those points. Additionally, corresponding conditions in the explanation rules are meaningless.

3 NEURULES

3.1 Syntax and Semantics

Neurules are a kind of integrated rules. The form of a neurule is depicted in Fig.1a. Each condition C_i is assigned a number sf_i, called its *significance factor*. Moreover, each rule itself is assigned a number sf_0, called its *bias factor*. Internally, each neurule is considered as an adaline unit (Fig.1b). The adaline unit uses the LMS algorithm for learning, which more safely converges for nonlinear training sets and generalizes better than perceptron (see e.g. [13]). The *inputs* C_i (i=1,...,n) of the unit are the conditions of the rule. The weights of the unit are the significance factors of the neurule and its bias is the bias factor of the neurule. Each input takes a value from the following set of discrete values: [1 (true), -1 (false), 0 (unknown)].

The *output D*, which represents the conclusion (decision) of the rule, is calculated via the standard formulas:

$$D = f(a),\ a = sf_0 + \sum_{i=1}^{n} sf_i C_i \qquad (1)$$

$$f(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{otherwise} \end{cases} \qquad (2)$$

where ***a*** is the *activation value* and *f(x)* the *activation function*, which is a threshold function. Hence, the output can take one of two values ('-1', '1') representing failure and success of the rule respectively. The significance factor of a condition represents the significance (weight) of the condition in drawing the conclusion.

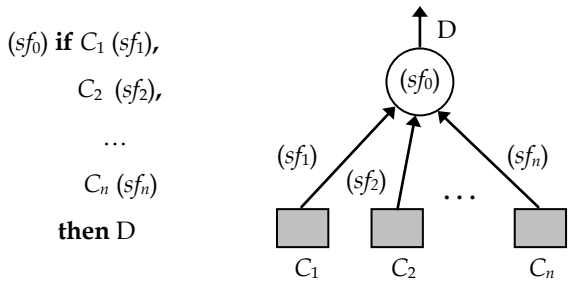

Fig. 1. (a) Form of a neurule (b) a neurule as an adaline unit

The general syntax of a neurule (in a BNF notation, where '< >' denotes non-terminal symbols) is:

<rule>::= (<bias-factor>) **if** <conditions> **then** <conclusion>
<conditions>::= <condition> | <condition>,<conditions>
<condition>::= <variable> <l-predicate> <value>
(<significance-factor>)
<conclusion>::= <variable> <r-predicate> <value> .

where <variable> denotes a *variable*, that is a symbol representing a concept in the domain, e.g. 'sex', 'pain' etc in a medical domain, and <l-predicate> denotes a symbolic or a numeric predicate. The symbolic predicates are {is, isnot}, whereas the numeric predicates are {<, >, =}. <r-predicate> can only be a symbolic predicate. <value> denotes a value; it can be a symbol (e.g. "male", "nightpain") or a number (e.g "5"). <bias-factor> and <significance-factor> are (real) numbers.

For the sake of simplicity, in the sequel, we consider "is" as the only <l-predicate> and <r-predicate>, without loss of generality, since the others can be somehow expressed via it. For example, the condition "pain isnot low" can be replaced by two conditions: "pain is medium", "pain is high", given that the set of values of 'pain' is {low, medium, high}. On the other hand, the datasets used in our experiments fit better in this predicate, so that no modifications are required.

Also, we use the following notation and definitions for a neurule R_k:

- n_{R_k} : the number of conditions of R_k.
- $sf_0^{R_k}$: the bias factor of R_k.
- $C_i^{R_k}$: the i^{th} condition of R_k, ($C_i^{R_k} \equiv$ "$V_i^{R_k}$ is $v_i^{R_k}$")
- $V_i^{R_k}$: the variable involved in $C_i^{R_k}$
- $v_i^{R_k}$: the value involved in $C_i^{R_k}$.
- $sf_i^{R_k}$: the significance factor of $C_i^{R_k}$.
- D^{R_k} : the conclusion of R_k ($D^{R_k} \equiv$ "$V_d^{R_k}$ is $v_d^{R_k}$")
- $V_d^{R_k}$: the variable involved in D^{R_k}.
- $v_d^{R_k}$: the value involved in D^{R_k}.
- $V_{C_k}^{R_k}$: the variable involved in condition C_k of R_k.

In the rest of the paper, we use the terms 'neurule' and 'rule' interchangeably. Also, we use plain capital letters

with (or without) a capital letter index to represent sets or lists of elements (e.g. sets of variables or rules) and italicized capital letters with (or without) a small letter index to represent individual elements (like a variable or a rule). Finally, a list is considered as an ordered set of elements.

3.2 Neurule-Based System Architecture

In Figure 2, the run-time part of a neurule-based system is illustrated. The run-time system consists of three modules, functionally similar to those of a conventional rule-based system: *neurule base (NRB)*, *integrated inference engine (IIE)* and *working memory (WM)*.

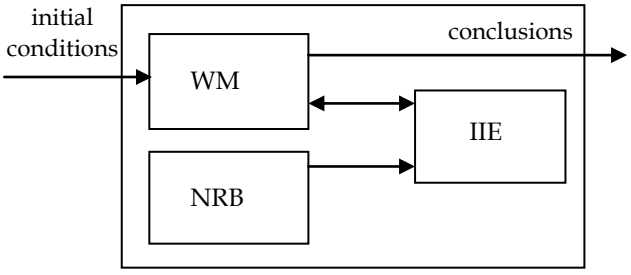

Fig. 2. The run-time architecture of a neurule-based system

NRB contains neurules produced from empirical data in a way outlined in the next section. IIE is responsible for making inferences by taking into account the data in WM and the rules in NRB. WM contains *fact assertions*. A fact assertion has the following structure:

$$(F_i, ass(F_i))$$

where F_i is a *fact* and $ass(F_i)$ is the assertion value related to it, which can be one of {TRUE, FALSE, UNKNOWN}. A *fact* has the same format as a condition or a conclusion of a rule:

$$F_i \equiv \text{“}V_{F_i} \text{ is } v_{F_i}\text{”}$$

where V_{F_i} is the variable and v_{F_i} is the corresponding value involved in the fact.

Fact assertions are either given by the user, as initial input data or during an inference course, or produced by the system, as intermediate or final conclusions during an inference course.

4 Producing a Neurule Base

4.1 Required Data and definitions

To be able to produce a neurule base, we need three types of data: (a) a set of domain variables, with their possible values, (b) dependency information and (c) a set of empirical data.

As is known, variables and values represent concepts in the problem domain. Specification of variables and their corresponding values is an important task in formulating or modelling a problem. Alongside variables, we should also specify the sets of the (discrete) values they can take. Specification of which concepts should be considered as variables and which as values is not unique. A concept that is a value in one specification/representation may be a variable in another one. This is related to how concepts depend on each other in making decisions, i.e. on *dependency information*, and perhaps on the form of available empirical data.

Let us consider a finite set of *domain variables* $V = \{V_i\}$, $1 \le i \le m$, which represent the concepts of the problem domain involved in making inferences. Each variable V_i can take values from a (corresponding) set of discrete values $S_{V_i} = \{v_{ij}\}$, $1 \le j \le k$.

Dependency information is given by the expert and indicates how variables depend on each other.

Definition 1. *Dependency information* f_{DI}^V related to the set of domain variables V is a relation:

$$f_{DI}^V : V \times V \to \{T, F\}$$

represented as a set of ordered pairs

$$f_{DI}^V = \{(V_i, V_j): V_i, V_j \in V, i \neq j\}$$

Each (V_i, V_j) is interpreted as “V_i depends on V_j”. Dependency information can be illustrated by means of a matrix (see Table 1 in Section 4.2).

We define various types of variables as follows. *Input variables* are those whose values are given by the user as input to the system. The set of input variables is denoted by V_{INP}. *Intermediate variables* represent concepts related to intermediate conclusions that are conclusions inferred via some rules, so their values depend on the values of input variables or other intermediate variables. The set of intermediate variables is denoted by V_{INT}. *Goal* (or *output*) *variables* are related to final conclusions, so their values depend on the values of input or intermediate variables and constitute the output of the system. The set of goal variables is denoted by V_G. Intermediate and goal variables are called *inferable variables*. The set of inferable variables V_{INF} is defined as: $V_{INF} = V_{INT} \cup V_G$. Of course: V_{INP}, V_{INT}, V_G and $V_{INF} \subset V$.

The set S of *empirical data* in general consists of a number of *patterns* (or *examples*) t_i:

$$S = \{t_i, 1 \le i \le n\}.$$

Each pattern t_i is an m-tuple of values:

$$t_i = < v_{i1}, v_{i2}, \ldots, v_{im} >$$

where $m = |V|$ and each $v_{ij} \in S_{V_j}$, $V_j \in V$. Each v_{ik} denotes that the fact (condition) “V_k is v_{ik}” is true.

A subset S_k of S is usually used for training a neurule. In such as set, the last value of a pattern t_i corresponds to a goal variable, thus called the *goal value* of the pattern and denoted by v_{id}. Also, we call ‘negative’ examples those having ‘-1’ and ‘positive’ those having ‘1’ as a goal value.

4.2 Neurules Production Algorithm

Production of a neurule base (NRB) is achieved by applying the *neurule production algorithm* (NPA). NPA tries to produce one neurule for each intermediate or output conclusion. However, due to possible non-linearity of the data, this is not usually feasible. So, more than one neurule having the same conclusion may be produced for each intermediate or output conclusion.

NPA Algorithm

Input:
- (a) *A set V of domain variables*
- (b) *Dependency information* f_{DI}^{V}
- (c) *A set S of empirical data*

Output: *A set of neurules (NRB)*

Body:
1. *Construct initial neurules, based on dependency information.*
2. *Extract an initial training set for each initial neurule from S.*
3. *Train each initial neurule individually and produce corresponding neurule(s).*

In the following, we analyse each step of the algorithm, introducing some definitions, where necessary.

The *initial neurules* are constructed by mainly using the dependency information. One initial neurule is constructed for each value of each intermediate or output variable. Initial neurules represent the possible intermediate or final conclusions. The conditions of each initial neurule include the variables that contribute in drawing the corresponding conclusion, as specified by the dependency information. In case of a boolean variable V_k, we consider S_{V_k} = {true} as its corresponding set of values (i.e. not including 'false'). So, step 1 of NPA can be analysed as follows:

1. For each inferable variable V_d
  1.1 For each possible value v_d of V_d
    1.1.1 Construct a neurule with conclusion "V_d is v_d" and conditions "V_i is v_i (0)", where $(V_d, V_i) \in f_{DI}^{V}$ and $v_i \in S_{V_i}$

Let us suppose that each constructed initial rule R_k has n_k conditions. Then, for each initial neurule its corresponding initial training set is extracted from S, in step 2, which can be more formally defined as follows:

2. For each initial rule R_k extract an initial training set S_k from S. Each pattern in S_k consists of as many values as the number of different variables in R_k. The last value is the goal value.

After the training set for each initial neurule has been determined, each initial neurule is individually trained, via the Least Mean Square (LMS) algorithm, using its own training set. Training is not always successful, that is a set of significance and bias factors cannot always be found that correctly classify all of the training examples. This is the case when the training patterns constitute a non-linear set and are therefore inseparable. When the algorithm succeeds, that is values for the bias and significance factors are calculated that classify all training patterns, one neurule is produced. When it fails, due to inseparability of the training examples, a splitting process is followed. More specifically, the initial training set of the neurule is split into two subsets and two copies of the initial neurule are trained, each using one of the training subsets. If training of either neurule copy fails, its subset is further split into two other subsets and so on, until there is no failure. In this way, more than one neurule are produced, having the same conditions with different bias and significance factors and the same conclusion, called *sibling neurules*.

So, step 3 of NPA is analysed as follows:

3. For each R_k do
  3.1 Train R_k using the LMS algorithm with S_k as the training set.
  3.2 If training is successful, produce R_k with the calculated $sf_i^{R_k}$ and stop (success).
  3.3 Split S_k into two suitable subsets, S_k^1 and S_k^2.
  3.4 Apply steps 3.1 to 3.3 with $S_k = S_k^1$ and $S_k = S_k^2$ separately.

In the above algorithm, a point left unspecified is how we split a training set into "suitable" subsets. Splitting is based on the following requirement: the patterns in each of the two subsets are closer between each other than between each one of them and each one of those in the other subset. The 'closeness' between patterns is estimated via their 'distance', based on some distance metric, like Hamming, Euclidian, Manhattan or Value Difference Metric-VDM [40]. Experiments have shown that VDM metric is the best, but most computationally expensive, metric, although Hamming distance metric has had quite good results, while being much less expensive [41]. Criterion for the comparison is the number of produced neurules: the less neurules the better the metric. This is because less neurules lead to more efficient ineferences.

To demonstrate NPA, we use an example problem from [13] and its corresponding dataset (called ACUTE). It is related to a medical diagnosis problem that concerns acute theoretical diseases of the sarcophagus. There are six symptoms (Swollen Feet, Red Ears, Hair Loss, Dizziness, Sensitive Aretha, Placibin Allergy), two diseases (Supercilliosis, Namastosis), whose diagnoses are based on the symptoms, and three possible treatments (Placibin, Biramibio, Posiboost). It's a small size, but adequately complicated problem.

TABLE 1
DEPENDENCY INFORMATION

	sf	**re**	**hl**	**dz**	**sa**	**pa**	**sc**	**nm**	**pl**	**bi**
sc	×	×	×							
nm			×	×	×					
pl						×	×	×		
bi			×				×	×		
po									×	×

We consider one variable for each symptom, disease and treatment: 'sf' for 'SwollenFeet', 're' for 'RedEars', 'hl' for 'HairLoss', 'dz' for 'Dizziness', 'sa' for 'SensitiveAretha', 'pa' for 'PlacibinAllergy', 'sc' for 'Supercilliosis', 'nm' for 'Namastosis', 'pl' for 'Placibin', 'bi' for 'Biramibio', 'po' for 'Posiboost'. All are considered Boolean, thus take as value 'true' or 'false'. The dependency information is depicted in Table 1 (where × means 'depends on'). The empirical dataset of the problem is presented in Table 2 (where T means 'true', F means 'false' and × means 'uknown'). So,

V = {sf, re, hl, dz, sa, pa, sc, nm, pl, bi, po}, S_{V_i} ={true}

f_{DI}^{V} ={(sc, sf), (sc, re), (sc, hl), (nm, hl), (nm, dz), (nm, sa), (pl, pa), (pl, sc), (pl, nm), (bi, hl), (bi, sc), (bi, nm), (po, pl), (po, bi)}

TABLE 2
THE ACUTE DATASET

sf	re	hl	dz	sa	pa	sc	nm	pl	bi	po
T	T	T	F	×	F	T	F	T	F	T
F	F	F	T	T	F	F	T	T	T	F
F	F	T	T	F	T	T	T	F	F	F
T	T	F	F	T	F	F	F	F	F	F
T	F	×	T	T	T	T	T	F	T	T
T	F	F	T	T	F	T	T	T	T	F
T	T	T	F	F	T	T	F	F	F	F
F	T	T	F	T	T	F	T	F	F	F

By applying step 1 of NPA, five initial neurules are produced. They are the same as the first five of those depicted in Table 4, but having all *sf*is equal to '0'.

By applying step 2 of NPA, we get five training sets, one for each of the five initial neurules, which are depicted in Table 3 (where duplicate patterns have been removed and cases with uknown values have not been taken into account).

TABLE 3
THE INITIAL TRAINING SETS

S_1	S_2	S_3	S_4	S_5
T T T 1	T F T 1	F T F 1	T F T -1	T F 1
F F F -1	F T T 1	F F T 1	F T F 1	T T -1
F T F 1	T F F -1	T T T -1	T T T -1	F F -1
T F T -1	F T F -1	F F F -1	F F F -1	F T 1
T F F 1	T T F 1	F T T 1	F T T 1	
F T T -1		T T F -1	T T F -1	
		T F T -1		

By applying step 3 of NPA, we get the neurules presented in Table 4. We can see that only two neurules have the same conclusion. This means that the training dataset used for their initial neurule (S_5) was non-linear (it actually represents the behavior of the XOR function).

5 THE INTEGRATED INFERENCE ENGINE

The inference engine associated with neurules implements the way neurules co-operate to reach conclusions. The choice of the next rule to be considered is based on a neurocomputing measure, but the rest is symbolic.

5.1 Evaluation Aspects

The inference process requires a way of evaluating a neurule, which in turn requires a way of evaluating its conditions. These are presented in this section.

TABLE 4
THE PRODUCED NEURULES

R_1
(-0.4) **if** SwollenFeet is true (3.6),
HairLoss is true (3.6),
RedEars is true (-0.8)
then Supercilliosis is true

R_2
(1.4) **if** Dizziness is true (4.6),
SensitiveAretha is true (1.8),
HairLoss is true (1.8)
then Namastosis is true

R_3
(-2.2) **if** PlacibinAllergy is true (-5.4),
Supercilliosis is true (4.6),
Namastosis is true (1.8)
then Placibin is true

R_4
(-4.0) **if** HairLoss is true (-3.6),
Namastosis is true (3.6),
Supercilliosis is true (2.8)
then Biramibio is true

R_5
(-2.2) **if** Placibin is true (-1.8),
Biramibio is true (1.0)
then Posiboost is true

R_6
(-2.2) **if** Biramibio is true (-2.6),
Placibin is true (1.8)
then Posiboost is true

5.1.1 Condition evaluation

Evaluation of conditions takes place during each inference session. Evaluation of conditions that refer to the same variable is done at the same time. We call those conditions 'sibling' conditions:

Definition 2. Two conditions $C_i \equiv$ "V_i is v_i" and $C_j \equiv$ "V_j is v_j"are called *sibling* iff $V_i \equiv V_j$.

A condition C_i can evaluate to TRUE, FALSE or UNKNOWN. Conditions are evaluated based on either the contents of the working memory (*WM*) or the user responses or firings of rules. Recall that

$$WM = \{(F_i, ass(F_i)), i=1, n\}$$

where $F_i \equiv$ "V_{F_i} is v_{F_i}" is a fact and $ass(F_i) \in$ {TRUE, FALSE, UNKNOWN}.

Evaluation of a condition, when based on WM contents, is done via the following rules:

(a) A condition C_i evaluates to TRUE, denoted by $ass(C_i)$ = TRUE, if there is a fact assertion (F_i, $ass(F_i)$) in WM with $F_i \equiv C_i$ and $ass(F_i)$ = TRUE.
(b) A condition C_i evaluates to FALSE, denoted by $ass(C_i)$ = FALSE, if there is a fact assertion (F_i, $ass(F_i)$) in WM with $F_i \equiv C_i$ and $ass(F_i)$ = FALSE.
(c) A condition C_i evaluates to UNKNOWN, denoted by $ass(C_i)$ = UNKNOWN, if there is a fact assertion (F_i, $ass(F_i)$) in WM with $F_i \equiv C_i$ and

$ass(F_i)$ = UNKNOWN.

When a user is asked to evaluate a condition $C_i \equiv$ "V_i is v_i" (where $V_i \in V_{INP}$) he/she is actually called to assign a value $value(V_i)$ to the corresponding variable V_i, where

$$value(V_i) \in S_{Vi} \cup \{\text{UNKNOWN}\}.$$

As soon as that value is given by the user (i.e the variable is evaluated), the sibling conditions $C_j \equiv$ "V_i is v_j" related to that variable (where $v_j \in S_{Vi}$) are consequently evaluated according to the following formula:

$$ass(C_j) = \begin{cases} \text{TRUE} & \text{if } v_j = value(V_i) \\ \text{UNKNOWN} & \text{if } value(V_i) = \text{UNKNOWN} \\ \text{FALSE} & \text{otherwise} \end{cases} \quad (3)$$

and corresponding fact assertions, called *sibling fact assertions* are added to WM. Sibling fact assertions are specified as:

$$sibl\text{-}fact\text{-}ass(V_i, value(V_i)) = \{(F_j, ass(F_j)): V_{F_j} \equiv V_i,\ v_{F_j} \in S_{V_i}\} \quad (4)$$

where $ass(F_j)$ is evaluated via formula (3). The same happens when an (intermediate) neurule is fired and its conclusion is produced, i.e. becomes TRUE for the specific variable and the specific value of its conclusion.

The above are based on the following remarks:

- If $ass(C_i)$ = TRUE, then the sibling conditions of C_i evaluate to FALSE, because a variable cannot take more than one value simultaneously.
- If $ass(C_i)$ = UNKNOWN, then the sibling conditions of C_i evaluate to UNKNOWN too, since 'unknown' means that there is no evidence about any value of a variable.

For example, let us suppose that in evaluating condition 'age is young' of a neurule the user gives for 'age' the value 'presbyopic', i.e. *value*('age') = 'presbyopic'. Also, let suppose that S_{age} ={young, pre-presbyopic, presbyopic}. Then, based on (3), the system results in *ass*('age is young') = FALSE and based on (4) the following sibling fact assertions:

sibl-fact-ass('age', 'presbyopic') = {('age is young', FALSE), ('age is presbyopic', TRUE), ('age is pre-presbyopic', FALSE)}

are added to the WM.

At any time, of an inference process, we distinguish between two sets of conditions for a neurule R_k : the *set of evaluated conditions* $C_E^{R_k}$ and the *set of unevaluated conditions* $C_U^{R_k}$. $C_E^{R_k}$ refers to conditions that have already been evaluated and the results of their evaluation have been taken into account in deducing the conclusion of R_k, whereas $C_U^{R_k}$ to the rest ones.

5.1.2 Neurule evaluation

A neurule evaluates to either '1' or '–1' (see Section 3.1). In the first case, we say that the neurule is *fired*, whereas in the second that it is *blocked*. In both cases the neurule is considered *evaluated*. The set of fired rules during an inference process is denoted by R_F, whereas that of blocked rules by R_B. Also, the set of evaluated rules is denoted by R_E and that of unevaluated by R_U.

Normally, the output of a neurule R_k is computed according to Eq. (1) and (2) (Section 3.1). To do that, all conditions of the neurule should be evaluated, to be able to compute $a(R_k)$, the *activation value* of R_k. So, we start evaluating the conditions of R_k one by one including its contribution in the *current activation value* of R_k :

$$a_c(R_k) = sf_0^{R_k} + \sum_{C_i^{R_k} \in C_E^{R_k}} sf_i^{R_k}\ assv(C_i^{R_k}) \quad (5)$$

where

$$assv(C_i^{R_k}) = \begin{cases} 1 & \text{if } ass(C_i^{R_k}) = \text{TRUE} \\ -1 & \text{if } ass(C_i^{R_k}) = \text{FALSE} \\ 0 & \text{if } ass(C_i^{R_k}) = \text{UNKNOWN} \end{cases} \quad (6)$$

This process should continue until $a(R_k)$ is computed. However, the interesting thing in this process is that it is possible to deduce the output of a neurule without having evaluated all of its conditions. To formalise and prove it, we introduce for each neurule the following metrics:

Definition 3. The *known sum* of a neurule R_k, at some time, represents its current potential to fire and is defined as the weighted sum of the values of the already "known", i.e. evaluated, conditions (inputs) of it:

$$ks(R_k) = sf_0^{R_k} + \sum_{C_i^{R_k} \in C_E^{R_k}} sf_i^{R_k}\ assv(C_i^{R_k}) \quad (7)$$

Definition 4. The *remaining sum* of a neurule R_k, at some time, represents its remaining potential to fire and is defined as the largest possible weighted sum of the "remaining", i.e. unevaluated, conditions of it:

$$rs(R_k) = \sum_{C_i^{R_k} \in C_U^{R_k}} |\ sf_i^{R_k}\ | \quad (8)$$

It is convenient to define another metric related to a neurule, the 'firing ratio'.

Definition 5. The *firing ratio (fr)* of a neurule R_k, at some time, is an estimate of its intention to be fired (or blocked) and is defined as the ratio of the absolute value of its known sum over the value of its remaining sum, given that it is non-zero:

$$fr(R_k) = \frac{|\ ks(R_k)\ |}{rs(R_k)}\ ,\ rs(R_k) \neq 0 \quad (9)$$

'Firing ratio' is similar to 'convergent ratio' used in RIE [21].

Now, we introduce and prove the following two simple theorems.

Theorem 1 (*neurule success or firing principle*). *The output of a neurule R_k is evaluated to '1', i.e. R_k succeeds or is fired, as soon as $ks(R_k) \geq 0$ and $|ks(R_k)| \geq rs(R_k)$.*

Proof. Let $a_c(R_k)$ the current activation value of R_k, when $m > 0$ conditions of R_k have been evaluated. It is obvious from the above that $a_c(R_k) = ks(R_k)$. Let $a_r(R_k)$ be

the remaining part of $a(R_k)$, i.e. the part of of $a(R_k)$ that will be finally computed when the rest (or remaining) conditions of R_k will be evaluated. So, at any step

$$a(R_k) = a_c(R_k) + a_r(R_k) = ks(R_k) + a_r(R_k) \quad (1\text{-}1)$$

Let assume that

$$ks(R_k) \geq 0 \quad (1\text{-}2)$$

To obtain an output equal to '1', $a(R_k) \geq 0$ should hold (see Section 3.1, Equation (2)). From (1-1), we get $ks(R_k) + a_r(R_k) \geq 0$ or

$$ks(R_k) \geq -a_r(R_k) \quad (1\text{-}3)$$

If we would be able to assure that (1-3) holds in the worst possible case, before all remaining conditions are evaluated, then R_k could be evaluated to '1' without any further evaluation.

Given (1-2), the worst possible case to satisfy (1-3) is when $-a_r(R_k)$ gets its maximum (positive) value, i.e. when $-a_r(R_k) \geq 0$ and $a_r(R_k)$ gets its minimum (negative) value, i.e.

$a_r(R_k) = -\max(\sum_{i=m}^{n_{R_k}} sf_i^{R_k}\ assv(C_i^{R_k})) = -\sum_{i=m}^{n_{R_k}} |\ sf_i^{R_k}\ | = -\ rs(R_k)$.

So, (1-3) becomes $ks(R_k) \geq rs(R_k)$ or equivalently

$$|ks(R_k)| \geq rs(R_k) \quad (1\text{-}4)$$

Theorem 2 (*neurule failure or blocking principle*). *The output of a neurule R_k is evaluated to '-1', i.e. R_k fails or is blocked, as soon as $ks(R_k) < 0$ and $|ks(R_k)| > rs(R_k)$.*

Proof. Under the same assumptions, again

$$a(R_k) = ks(R_k) + a_r(R_k) \quad (2\text{-}1)$$

Let assume that

$$ks(R_k) < 0 \quad (2\text{-}2)$$

To obtain an output equal to '-1', $a(R_k) < 0$ should hold (see Section 3.1, Equation (2)). From (2-1), we get $ks(R_k) + a_r(R_k) < 0$ or

$$ks(R_k) < -a_r(R_k) \quad (2\text{-}3)$$

If we would be able to assure that (2-3) holds in the worst possible case, before all remaining conditions are evaluated, then R_k could be evaluated to '-1' without any further evaluation.

Given (2-2), the worst possible case to satisfy (2-3) is when $-a_r(R_k)$ gets its minimum (negative) value, i.e. when $-a_r(R_k) < 0$ and $a_r(R_k)$ gets its maximum (positive) value, i.e.

$a_r(R_k) = \max(\sum_{i=m}^{n_{R_k}} sf_i^{R_k}\ assv(C_i^{R_k})) = \sum_{i=m}^{n_{R_k}} |\ sf_i^{R_k}\ | = rs(R_k)$.

So, (2-3) becomes $ks(R_k) > rs(R_k)$ or $-ks(R_k) > rs(R_k)$ or finally (given (2-2))

$$|ks(R_k)| > rs(R_k) \quad (2\text{-}4)$$

Based on the above theorems, we introduce the following definitions:

Definition 6. *Success or firing condition* of a neurule R_k is the situation where $|ks(R_k)| \geq rs(R_k)$ (or equivalently $fr \geq 1$, $rs(R_k) \neq 0$) and $ks(R_k) \geq 0$.

Definition 7. *Failure or blocking condition* of a neurule R_k is the situation where $|ks(R_k)| > rs(R_k)$ (or equivalently $fr > 1$, $rs(R_k) \neq 0$) and $ks(R_k) < 0$.

For example, consider neurule R_2 from Table 4 (Section 4). The initial values of $ks(R_2)$ and $rs(R_2)$ are: $ks(R_2) = 1.4$ and $rs(R_2) = |4.6| + |1.8| + |1.8| = 8.2$. Let assume that the first condition "Dizziness is true" is evaluated to be true. Then, $ks(R_2) = 1.4 + 1*4.6 = 6.0$ and $rs(R_2) = 8.2 - |4.6| = 3.6$. So, $ks(R_2) > 0$ and $|ks(R_2)| > rs(R_2)$ and the rule is fired without any further evaluation, because the firing condition is met.

Similarly, consider neurule R_3 from Table 4 (Section 4). The initial values of $ks(R_3)$ and $rs(R_3)$ are: $ks(R_3) = -2.2$ and $rs(R_3) = |-5.4| + |4.6| + |1.8| = 11.8$. Let assume that the first condition "PlacibinAllergy is true" is evaluated to be true. Then, $ks(R_3) = -2.2 + 1*(-5.4) = -7.6$ and $rs(R_3) = 11.8 - |-5.4| = 6.4$. So, $ks(R_3) < 0$ and $|ks(R_3)| > rs(R_3)$ and the rule is blocked without any further evaluation, because the blocking condition is met.

We experimented with a number of ways of arranging conditions to achieve more efficient response. We finally found out that an overall ordering of the conditions of neurules make inferences more efficient. More specifically, that happens when the conditions in a neurule R_k consisting of n_{R_k} conditions are ordered in a way that $|sf_1| \geq |sf_2| \geq \ldots \geq |sf_{n_{R_k}}|$. So, internally, the conditions of each neurule are ordered in that way.

5.1.3 Goal facts

The inference procedure uses a goal stack, where the possible solutions-answers are stored in the form of facts, which we call 'goal facts'. Goal facts are actually the conclusions of the neurules that contain a goal variable. To be more precise, we give the following definition:

Definition 7. A *goal fact* G_i related to a neurule base is an expression of the form "V_{G_i} is v_{G_i}" where $V_{G_i} \in \mathrm{V_G}$ and $v_{G_i} \in S_{V_{G_i}}$.

5.2 The Integrated Inference Process

The integrated inference process focuses on the firing potential of each neurule. Initially, the *fr*s of all neurules are computed. In this process, after evaluation of a condition of a neurule, the *fr*s of the neurules that contain that variable (called *affected rules*) are updated. In the next step, the neurule with the maximum *fr* is considered, given that it is the neurule most likely (i.e. with the greatest intention) to fire. This means that its first unevaluated condition is considered as the next goal and so on. As soon as a neurule is fired, the WM is updated with corresponding conclusions. A similar thing happens when a neurule is blocked. So, at each step of the process, rules are competing between each other towards which is closer to fire, based on their *fr*s. The one with the greatest *fr* takes the lead. Inference stops either successfully, when there are facts in WM containing goal variables and assigned the TRUE value and no further action can be taken, or unsuccessfully, otherwise.

There is one stack used, the *goal stack* (GS), which initially includes the goal facts. The basic inference algorithm is as follows:

1. *Set the goal(s) on GS, the initial facts in the WM and compute (initial) fr for each neurule.*
2. *For each fact in WM do*
   Find all affected neurules and update their frs.
3. *While there are still goals on GS do*
   3.1 Choose the neurule with the maximum fr from affected rules, if there are such, or from the unevaluated neurules, otherwise.
      3.1.1 If the firing condition is satisfied for the rule, update WM with the sibling assertions related to the conclusion of the rule, update affected rules, update their frs and remove from GS the goals having the same variable as the conclusion of the fired rule.
      3.1.2 If the blocking condition is satisfied and all the sibling rules of the rule have been already evaluated and blocked, update WM with the fact related to the falsity of the conclusion of the rule, update affected rules, update their frs and remove from GS the goal corresponding to the conclusion of the blocked rule.
      3.1.3 If neither of the above happens, choose the first unevaluated condition of the rule.
         3.1.3.1 If the condition contains an input variable, ask the user for a value, update WM with the sibling assertions related to the condition and update affected rules and their frs.
         3.1.3.2 If the condition contains an intermediate variable, consider from the sibling rules having that variable in their conclusions the one with the maximum fr, choose its first unevaluated condition and go to step 3.1.3.1.
4. *If WM contains any goal facts assigned a 'TRUE' value, return those goal facts, otherwise return failure.*

To briefly demonstrate the function of the above inference algorithm, we consider the problem used in Section 4.2. As an example inference, let us assume that the input data (i.e. the data for sf, re, hl, dz, sa and pa) of the second row of Table 2 is given. This means that the following facts are available in the WM: {('sf is true', F), ('re is true', F), ('hl is true', F), ('dz is true', T), ('sa is true', T), ('pa is true', F)}.

After starting inference, R_1 and R_2 are examined:

- R_1 is blocked, because at some point $ks(R_1)$ = -0.4+ (-1)*3.6+(-1)*3.6=-7.2<0 and |-7.2|> $rs(R_1)$=0.8. So, ('sc is true', F) is added to WM.
- R_2 is fired, because at some point $ks(R_2)$ = 1.4+ 1*4.6=6.0>0 and |6.0|> $rs(R_2)$=1.8+1.8=3.6. So, ('nm is true', T) is added to WM.

After that, at second level, R_3 and R_4 are examined:

- R_3 is fired, because at some point $ks(R_3)$ = -2.2+ (-1) *(-5.4)+(-1)*4.6+1*1.8=0.4>0 and $rs(R_3)$=0. So, ('pl is true', T) is added to WM.
- R_4 is fired too, because at some point $ks(R_4)$ = -4.0+ (-1)*(-3.6)+1*3.6=3.2>0 and |3.2|> $rs(R_4)$=2.8. So, ('bi is true', T) is added to WM.

Finally, R_5 and R_6 are examined:

- R_5 is blocked, because at some point $ks(R_5)$ = -2.2+ 1*(-1.8)=-4.0<0 and |-4.0|> $rs(R_5)$=1.0. So, ('po is true', F) is added to WM.
- R_6 is blocked too, because at some point $ks(R_6)$ = -2.2+ 1*(-2.6)=-4.8<0 and |-4.8|> $rs(R_5)$=1.8. So, ('po is true', F) is added to WM.

It is easy to see that the last five values of the selected row of Table 2 have been proved.

After testing the above algorithm using all the patterns of the example dataset of Table 2, we find that all results (intermediate and final) are correct.

6. Experimental Results and Discussion

In this section, experimental results regarding the performance of neurules are presented. They refer to the following:

- The neurules integrated inference engine (IIE) is compared to the inference mechanisms used in connectionist expert systems and presented in [12], [13] (MACIE) and [21] (RIE).
- The generalization capabilities of neurules are explored and compared to those of a back-propagation neural net (BPNN) and the adaline unit.

The experiments were performed on a Pentium IV, operating at 3 GHz.

6.1 Neurules Inference Process vs Connectionist Expert Systems Inference

A bunch of experiments were conducted to compare IIE with MACIE and RIE. The inference mechanisms are compared as far as the required computational cost for drawing conclusions is concermed. More specifically, comparison of the inference mechanisms is made in terms of the mean computational time required and the number of computations made to draw conclusions.

Ghalwash compared RIE with Gallant's inference mechanism in terms of the 'convergent rate', i.e. the ratio of the number of necessary (i.e. the least required) inputs and the total number of asked inputs. However, this is a questionable criterion. First, it is difficult, if not imposible, to estimate the necessary inputs for each inference, except they are known or come from short chains of symbolic rules, so are easy to specify. This might be the reason why in [21] only two rather simple problems are used in the experiments. Unfortunately, it is not explained how the necessary inputs are determined. Second, even if it is possible, it shouldn't be the only criterion, since that rate is not directly related to the computational cost, which mainly depends on the length of the inference chains and not on the number of used inputs, which anyway is usually small. On the other hand, we believe that computational cost is more important in such cases and is the one that is typically used (see e.g. in [43]). Furthermore, a comparison between MACIE and RIE mechanisms based on the computational cost has not been made. So, such a comparison can produce fruitful results both for neurules and connectionist expert systems. Despite the above, we also conducted an experiment concerning 'convergent rate'.

We have used four datasets and two symbolic rule bases. The first dataset (ACUTE) is the one used as an

example in Section 4 and is taken from [13]. The dataset is incomplete. It consists of 8 input data patterns out of 64 possible. For most of the input combinations no final conclusion can be drawn since all the output cells of the connectionist knowledge base become false and all the output rules of the corresponding neurule base become blocked. We give results for 27 input combinations for which final conclusion(s) can be drawn that is, one or more output cells of the connectionist knowledge base become true and one or more output rules of the corresponding neurule base fire. The second dataset (called LENSES) is taken from the UCI Machine Learning Repository [1] and regards a database for fitting contact lenses. This dataset is complete and contains 24 input patterns each consisting of four input and one output attribute (variable). The third and fourth datasets (called CAR and NURSERY respectively) are also from the UCI Machine Learning Repository. However, we also used the dependency information provided by the donators of the datasets [5], which is not included in the UCI Machine Learning Repository.

The above datasets were chosen to satisfy at least one of the requirements set:

- be categorical
- there is dependency information available.

TABLE 5
COMPARISON OF INFERENCE MECHANISMS (COMPUTATIONAL COST)

	IIE		*MACIE*		*RIE*	
Dataset	*Runtime (msec)*	*Computations*	*Runtime (msec)*	*Computations*	*Runtime (msec)*	*Computations*
ACUTE (27)	0.041	11.56	0.050	17.48 + 15.11	0.046	17.48
LENSES (24)	0.045	25.50	0.073	42.04 + 18.67	0.061	42.04
CAR-DEP (1728)	0.154	117.96	0.241	158.05 + 68.07	0.203	160.32
NUSERY-DEP (12960)	0.187	138.91	0.281	195.50 + 84.18	0.250	196.84
RB1	0.256	121.5	0.470	172.43+100.77	0.409	179.70
RB2	0.497	282.60	1.450	603.96+310.16	1.299	599.28

The two rules bases (RB1 and RB2) are related to diagnosis of bone diseases and have been created via interviews with expert doctors (they were also used in [24]). They consist of 59 and 134 symbolic rules respectively. To be used in the experiments, the two rules bases were preprocessed. Rules with the same conclusion were grouped and the truth table of the combined logical function of the rules of each group was extracted. The valid rows of the truth table constituted the patterns of the corresponding dataset. The produced neurule bases performed all inferences correctly.

IIE was applied to the four neurule bases produced from the above mentioned four datasets. MACIE and RIE were implemented in a simulated way and also applied to the four connectionist knowledge bases, which were created from the same datasets according to the guidelines presented in [13]. In the case of the two rule bases, the inference mechanisms were applied to the integrated/connectinist knowledge bases produced from the datasets of each rule group.

Table 5 presents experimental results regarding the performance of the inference mechanisms as far as computational cost is concerned. 'Runtime' represents the mean computational time required to draw the conclusions. 'Computations' represents the mean number of computations required to reach conclusions. In the case of IIE, those computations involve the mean number of times that a product of a significance factor and a corresponding condition value were added to the known sum of a neurule. In the cases of MACIE and RIE, they involve the mean number of times that a product of a weight and a corresponding input value were added to the known (weighted) sum of a node. MACIE requires additional computations for the calculation of confidences. These computations are displayed besides the '+' symbol in each case.

The inference mechanisms, except for the ACUTE dataset, were tested for all input combinations (the total number of input combinations is shown in parentheses beside the dataset names). As shown in Table 5, IIE performs better than the two inference mechanisms used in connectionist expert systems. One reason for that is that the learning algorithm used for the construction of the corresponding networks in both cases introduces some random cells in their connectionist knowledge bases, as hidden layer nodes, in order to deal with inseparability. The introduction of these cells increases the number of required computations and hence the mean computational time. On the other hand, in terms of symbolic reasoning, while the other two approaches use a mixed forward and backward chaining, neurules inference process follows a backward chaining based strategy, which is usually more efficient. More specifically, in the forward chaining mode, in MACIE and RIE, when an input is given, computation is spread to almost all nodes of the network, whereas in neurules mechanism only specific rules are

involved. In general, a neurule base is quite different from a connectionist one. From a connectionist point of view, it consists of independent cells each of which corresponds to a neurule. Each such cell has a structure, consisting of identifiable conditions. This gives the flexibility to the inference mechanism to choose the most suitable cell (neurule) to move on, without being commited to follow all connections between cells.

Another reason is the existence of sibling rules. Sibling rules play the role of the random cells, i.e. solve the non-linearity problem. When a neurule fires during an inference, then all its sibling rules are not considered in the rest of the inference. This reduces the number of neurules to be considered in the sequel; this contributes in reducing computational time.

Another finding from Table 5 concerns connectionist expert systems: the time performance of RIE is better than the time performance of MACIE. This is mainly due to the following: (a) RIE focuses on the nodes most relevant to the computation of the outputs as has been shown in [21] and (b) the computation of the convergent ratios in RIE is less expensive than the computation of the confidences in MACIE (the confidence of a node requires the computation of the confidences of its input nodes).

A difficulty arising when constructing the knowledge base of the connectionist expert system following Gallant's approach is finding a proper random function for the intermediate (i.e. hiden layer) cells so that the overall network (after training) will classify correctly all patterns. As the number of patterns increases, this difficulty increases as well.

Table 6 presents experimental results concerning the performance of the inference mechanisms as far as the 'convergent rate' is concerned. We have used two of the datasets and the two rule bases, because it was easier to determine the necessary inputs for them.

TABLE 6
COMPARISON OF INFERENCE MECHANISMS (CONVERGENT RATE)

Dataset	*IIE*	*MACIE*	*RIE*
ACUTE	0.639	0.709	0.709
LENSES	0.825	0.944	0.944
RB1	0.877	0.934	0.842
RB2	1.000	1.000	1.000

An outcome of Table 6 is that MACIE gives better results than IIE and RIE. The latter is in contrast to what is claimed in [21], but we guess that this is so, because the set of 8 patterns is used in [21] instead of that of 27 patterns we use here. This means that MACIE's inference mechanism does a better selection of inputs. As to neurules, it seems that IIE requires more inputs than MACIE does, whereas compared to RIE things are not so clear. It seems that sibling rules play a negative role here. In case a conclusion is examined, all related sibling rules should be examined one by one, until one is fired or all found to be blocked. So, in case the conclusion of a set of sibling rules cannot be derived (i.e. it is proved false), all sibling rules will be examined. This may result in using unnecessary inputs.

6.2 Generalization Capabilities of Neurules

To test the generalization capabilities of neurules, we conducted a number of experiments. Table 7 presents results regarding the classification accuracy (generalization) of neurules on unseen test examples compared with the ones of the adaline unit and BPNNs. The results for each dataset (except for the two monks datasets) were produced by using 75% of the examples as training set and 25% of the examples as testing set in four different runs. Different and disjoint test sets were used in each run, so that the union of the four test sets formed the whole dataset (4-fold cross validation). The classification accuracy was computed as the mean value of the accuracies obtained from the four tests. For 'monks1' dataset the procedure for creating training and test sets was applied to the corresponding test sets of 432 training examples available in the UCI repository. For the 'monks3' dataset, the training and test set available in the UCI repository were used, since the training set is reported to contain noise. It should be mentioned that we were not able to construct a BPNN for the 'Nursery' dataset with competitive generalization capability.

For the training of BPNNs, the standard back-propagation algorithm was employed using a momentum in adjusting the weights and one layer of hidden nodes. The values of these three back-propagation parameters along with the average error threshold were tuned separately for the training sets of each dataset after a number of experiments (based on error-and-trial). Training stopped when either the number of training epochs reached an upper threshold or the average squared error became less than or equal to the average error threshold. If the activations of multiple output nodes exceeded 0.5 (when a test example was given as input), then the example took the category of the most active output node (i.e., the one with the greatest activation) [50].

TABLE 7
GENERALIZATION EXPERIMENTAL RESULTS

Dataset	*Adaline Unit (%)*	*Neurules (%)*	*BPNN (%)*
LENSES	58.33	70.83	70.83
MONKS1	66.44	99.77	100
MONKS3	87.50	93.98	97.22
TIC-TAC-TOE	57.93	97.50	98.23
CAR	72.57	93.63	95.72
NURSERY	80.52	99.31	

The results in Table 7 show that neurules generalize quite well. It shows that neurules well outperform the adaline unit and are a bit worse than BPNNs. These results are very promising. It was expected that the generalization capability of neurules would be somewhere between the adaline unit and the BPNN. This is due to the nature of the three approaches: the adaline unit is a single unit for performing classification, a neurule base consists of a number of autonomous adaline units (neurules) and a

back-propagation neural network is a multi-layer network containing hidden nodes useful for the computation of non-separable functions. It should be noted here that there are also other, in some sense similar, hybrid systems, like fuzzy ARTMAP [9] and Cascade ARTMAP [46], that have achieved highly competitive generalization performance with regards to BPNNs.

A parameter not shown in Table 7 involves the total effort in constructing the corresponding knowledge base. The construction of a neurule base is easier than the construction of a BPNN. Construction of neurules is straightforward. On the other hand, in the case of a BPNN, one should simultaneously adjust three different parameters (based on error-and-trial): the number of hidden nodes (assuming one hidden layer), the learning rate and the momentum. The number of hidden nodes is an integer, whereas the learning rate and the momentum are real numbers lying between 0.0 and 1.0. Simultaneously adjusting those three parameters can be a non-trivial and time-consuming task. However, the adjustment of those parameters plays an important role in the classification accuracy of the neural network regarding the training and test sets.

According to the above results about generalization capabilities of neurules, neurules could be a choice in applications with available training examples and in which naturalness and modularity of the knowledge base as well as provision of interactive inference are desirable factors. Obviously in applications in which generalization is the only concern, one could choose BPNNs, but in that case the effort required for tuning the network should be taken seriously into account.

7. An Application of Neurules

7.1 Developing a Web Based ITS

We have used neurules in developing a web-based intelligent tutoring system (ITS) [28]. ITSs are knowledge intensive systems that require a number of discrete knowledge bases used in different units of the systems, as illustrated in Figure 3, where the basic inference structure of the ITS, including its knowledge based components, is depicted. For example, the pedagogical unit contains three neurule bases, one for teaching method selection, one for the next concept to be taught and the last one for selecting the appropriate course units from domain knowledge. Also, the user modeling unit includes two neurule bases, one for selecting the user stereotype and the other for evaluating student's knowledge level about concepts.

Neurules greatly facilitated the development and performance of the ITS in a number of ways:

- Neurules support incremental development of neurule-bases, because they retain the modularity of symbolic rules. One can easily add new neurules or remove old neurules from a neurule base without making any other changes to it. ITSs need this kind of capability. This is not offered by neural nets.
- Neurules can be produced in a straight-forward, semi-automated way from empirical data or decision tables. Also, no extra effort for tuning the knowledge base is required. These are very important for ITSs, because knowledge acquisition for them is harder than for conventional expert systems, due to the existence of more than one knowledge-based module. Neither symbolic rules nor neural nets or existing integrations offer both above facilities.

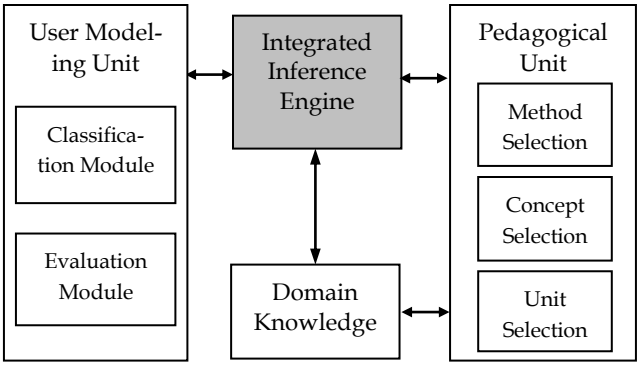

Fig. 3. The inference architecture of the developed ITS.

- Neurules can make robust inferences. In contrast to symbolic rules, neurules can derive conclusions from partially known inputs. This is due to the fact that neurules integrate a connectionist component. This feature is useful, because, during a teaching session, certain parameters related to the user may be unknown.
- Neurules provide a time-efficient inference engine. This means that they require fewer computations compared to classical symbolic rules and even to other hybrid approaches in order to derive the same conclusions. This is very important, since an ITS is a highly interactive knowledge-based system requiring time-efficient responses to users' actions. Furthermore, the Web imposes additional time constraints.

7.2 Bank Loan Application Evaluation System

We have also used neurules in the development of an expert system, which evaluates the applications of the clients of a bank who apply for getting a loan. We implemented two versions of the system, one using fuzzy rules via FuzzyCLIPS and the other using neurules. The fuzzy expert system was based on empirical knowledge, elicitated from experts. The neurule-based expert system was based on empirical (real) data taken from the data base of a bank.

The system consists of three discrete neurule bases, as illustrated in Figure 4. One of them deals with evaluating the trust of the applicant, based on his income, age, profession etc. The second evaluates the trust of the surety of the applicant based on similar parameters. Finally, the third one results in an evaluation of the trust of the pair

applicant-surety, based on the results of the other two knowledge bases.

Again, neurules facilitated the development and performance of that system in similar ways as for the ITS, especially in acquiring knowledge, where real (empirical) data was used instead of the time consuming expert based elicitation, which was the case in the version of the system where FuzzyCLIPS was used. Finally, comparison of the two systems in terms of accuracy showed that the neurule-based system did (slightly) better than the fuzzy one and with less effort, whereas the fuzzy expert system took a long time for fuzzy parameters tuning.

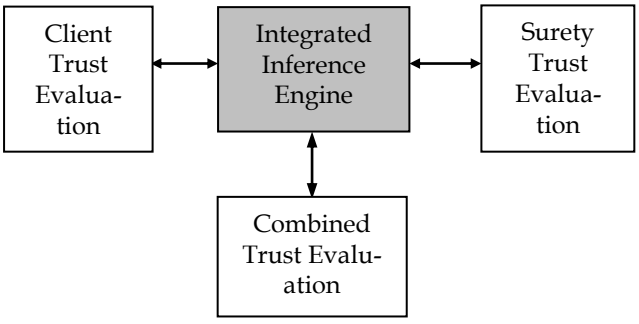

Fig. 4. The inference architecture of the loan application evaluation system.

8. Conclusions

In this paper, we present the construction process and the inference mechanism of neurules, a type of integrated rules, integrating symbolic rules with neurocomputing. A special feature of neurules is that they combine the symbolic and the connectionist approach in a way which is not the usual one: they incorporate connectionist computation within the symbolic framework, not the other way round. This creates an integration that refers to all aspects of a knowledge representation language: the syntax, the semantics and the inference mechanism. Another interesting point of neurules is that the associated inference mechanism is also integrated, in contrast to other approaches, where the inference mechanism remains connectionist. This makes the inference process quite explicit and comprehensive. So, even natural explanations can be produced [26] (however, this is not treated here). A further attractive feature of the neurules, which stems from the way integration is made, is that compared to other connectionist-oriented integrations they retain the modularity and to some degree the naturalness of symbolic rules, which are higly desired features.

The inference mechanism draws conclusions based on facts and asking the user to provide values for some inputs, i.e. it supports interactivity with the user, something not supported by other similar efforts. Experimental results have shown a superiority of the neurule-based inference engine compared to those used in well-known connectionist expert systems, in terms of efficiency, but not in terms of the number of the involved inputs.

Also, generalization capabilities of the neurules have been explored in this paper. Results show that neurules do much better than its neural component (adaline unit) itself and is a bit behind the BPNN. However, the effort required for constructing and tuning a neural net is much more than that required for neurules. So, neurules remain a strong alternative, even when generalization is demanding.

Given the above, we can say that neurules offer learning and inference in a single paradigm. This is another important feature of neurules not met in other efforts.

Finally, neurules have been successfully used in a web-based intelligent tutoring system and a bank loan application evaluation intelligent system.

Our future work is directed to finding (a) an even more efficient way for making inferences with neurules, (b) a way of providing natural explanations and (c) a way of incorporating fuzziness.

References

[1] A. Asuncion and D.J. Newman (2007). UCI Machine Learning Repository [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science.

[2] S. Bader and P. Hitzler (2005). "Dimensions of neural-symbolic integration – a structured survey". In S. Artemov, H. Barringer, A. S. d'Avila Garcez, L. C. Lamb, and J. Woods, editors, We Will Show Them: Essays in Honour of Dov Gabbay, volume 1, pages 167–194. International Federation for Computational Logic, College Publications.

[3] S. Bader, P. Hitzler, S. Holldobler and A. Witzel (2007). A Fully Connectionist Model Generator for Covered First-Order Logic Programs, Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), 666-671.

[4] S. Bader, P. Hitzler and S. Holldobler (2008). Connectionist model generation: A first-order approach, Neurocomputing, 71, 2420-2432.

[5] M. Bohanec and B. Zupan (1997). AI Lab Datasets [http://magix.fri.uni-lj.si/blaz/hint/datasets.htm]. Slovenia, University of Ljubljana, Faculty of Computer and Information Science.

[6] G. Bologna (2003). A Model for Single and Multiple Knowledge Based Networks, Artificial Intelligence in Medicine, 28(2), 141-163.

[7] L. Bookman and R. Sun (eds.) (1993), Special Issue on Integrating Neural and Symbolic Processes, Connection Science, 5(3-4).

[8] A. Browne and R. Sun (2001). Connectionist Inference Models, Neural Networks, 14(10), 1331-1355.

[9] G. A. Carpenter and A.-H. Tan (1995). Rule extraction: From neural architecture to symbolic representation. Connection Science 7, 3-27.

[10] I. Cloete and J.M. Zurada (eds.), (2000). Knowledge-Based Neurocomputing, The MIT Press, Cambridge.

[11] L.M. Fu (1993). Knowledge-Based Connectionism for Revising Domain Theories, IEEE Transactions on Systems, Man, and Cybernetics, 23(1), 173-182.

[12] S. I. Gallant (1988). Connectionist expert systems, Communications of the ACM, 31(2), 152-169.

[13] S. I. Gallant (1993). Neural Network Learning and Expert Systems, MIT Press.

[14] A. S. d'Avila Garcez, K. Broda and D. M. Gabbay (2001). Symbolic Knowledge Extraction from Trained Neural Networks: A Sound Approach. Artificial Intelligence, 125(1-2):155–207.

[15] A. d'Avila Garcez, K. Broda and D.M. Gabbay (2002). Neural-Symbolic Learning Systems: Foundations and Applications, Perspectives in

Neural Computing, Springer-Verlag.

[16] A. d'Avila Garcez, D. Gabbay, S. Holldobler and J. Taylor (2004). Special Issue on Neural-Symbolic Systems, Journal of Applied Logic, 2(3).

[17] A. d'Avila Garcez and L. C. Lamb (2006). A connectionist computational model for epistemic and temporal reasoning, Neural Computation 18(7), 1711-1738.

[18] A. d'Avila Garcez, L. C. Lamb and D. M. Gabbay (2006). Connectionist computations of intuitionistic reasoning, Theoretical Computer Science 358(1), 34-55.

[19] A. d'Avila Garcez, L. C. Lamb and D. M. Gabbay (2007). Connectionist modal logic: Representing modalities in neural networks, Theoretical Computer Science 371(1-2), 34-53.

[20] A. d'Avila Garcez, L. C. Lamb and D. M. Gabbay (2008). Neural-Symbolic Cognitive Reasoning, Springer-Verlag.

[21] A. Z. Ghalwash (1998). A Recency Inference Engine for Connectionist Knowledge Bases, Applied Intelligence, 9(3), 201-215.

[22] H. Gust, K.-U. Kuhngerger and P. Geibel (2007). Learning Models of Predicate Logical Theories with Neural Networks Based on Topos Theory. In [23], 233-264.

[23] B. Hammer and P. Hitzler (Eds) (2007). Perspectives of Neural-Symbolic Integration, Studies in Computational Intelligence 77, Springer-Verlag.

[24] I. Hatzilygeroudis and J. Prentzas (2000). Neurules: Improving the Performance of Symbolic Rules. International Journal on AI Tools (IJAIT), 9(1), 113-130.

[25] I. Hatzilygeroudis and J. Prentzas (2001). Constructing Modular Hybrid Knowledge Bases for Expert Systems. International Journal on AI Tools (IJAIT), 10(1-2), 87-105.

[26] I. Hatzilygeroudis and J. Prentzas (2001). An Efficient Hybrid Rule Based Inference Engine with Explanation Capability, Proceedings of the 14th International FLAIRS Conference, 227-231.

[27] I. Hatzilygeroudis and J. Prentzas (2004). Neuro-Symbolic Approaches for Knowledge Representation in Expert Systems, International Journal on Hybrid Systems (IJHIS), 1(3-4), 111-126.

[28] I. Hatzilygeroudis and J. Prentzas (2004). Using a Hybrid Rule-Based Approach in Developing an Intelligent Tutoring System with Knowledge Acquisition and Update Capabilities. Journal of Expert Systems with Applications, 26(4), 477-492.

[29] M. Hilario (1997). An Overview of Strategies for Neurosymbolic Integration", in R. Sun and E. Alexandre (Eds.), Connectionist-Symbolic Integration: From Unified to Hybrid Approaches, Lawrence Erlbaum, ch.2.

[30] M. Hilario, A. Rida (1997). The Use of Prior Knowledge in Neural Network Configuration and Training. In Proceedings of the International Work-Conference on Artificial and Natural Neural Networks: Biological and Artificial Computation: From Neuroscience to Technology, Lecture Notes in Computer Science, vol. 1240, 227-236.

[31] S. Holldobler and Y. Kalinke (1994). Towards a Massively Parallel Computational Model for Logic Programming. In Proceedings of the ECAI94 Workshop on Combining Symbolic and Connectionist Processing, pp. 68–77, Amsterdam, The Netherlands, ECCAI 1994.

[32] S. Holldobler, Y. Kalinke and H.-P. Storr (1999). Approximating the Semantics of Logic Programs by Recurrent Neural Networks. Applied Intelligence, 11(1), 45–58.

[33] E. Komendantskaya, M. Lane and A. K. Seda (2007). Connectionist Representation of Multi-Valued Logic Programs. In [23], 283-313.

[34] L. C. Lamb, R. V. Borges and A. d'Avila Garcez (2007). A Connectionist Cognitive Model for Temporal Synchronisation and Learning, Proveedings of the 22nd Conference on Artificial Intelligence (AAAI 2007), 827-832.

[35] J. J. Mahoney and R. Mooney (1993). Combining Connectionist and Symbolic Learning to Refine Certainty-Factor Rule Bases. Connection Science, 5(3-4), 339-364.

[36] K. McGarry, S. Wermter and J. MacIntyre (1999). Hybrid Neural Systems: From Simple Coupling to Fully Integrated Neural Networks, Neural Computing Surveys, vol. 2, pp. 62-93.

[37] L. R. Medsker (1995). Hybrid Intelligent Systems, Kluwer Academic Publishers, Second Printing, 1998.

[38] L. Souici-Meslati and M. Sellami (2006). Toward a Generalization of Neuro-Symbolic Recognition: An Application to Arabic Words. International Journal of Knowledge-based and Intelligent Engineering Systems, 10(5), 347-361.

[39] C. W. Omlin and C. L. Giles (1996). Rule Revision with Recurrent Neural Networks. IEEE Transactions on Knowledge and Data Engineering, 8(1), pp. 183-188.

[40] T. R. Payne and P. Edwards (1998). Implicit Feature Selection with the Value Difference Metric. In Proceedings of the 13th European Conference on Artificial Intelligence), 450-454, Henri Prade (Ed), John Wiley & Sons.

[41] J. Prentzas and I. Hatzilygeroudis (2006) "Construction of Neurules from Training Examples: A Thorough Investigation", Proceedings of the ECAI-06 Workshop on "Neural-Symbolic Learning and Reasoning" (NeSy'06), A. Garcez, P. Hitzler and G. Tamburini (Eds), Riva del Garda, Italy, 28 Aug-1Sept 2006, 35-40.

[42] H. Reichgelt (1991). Knowledge Representation, An AI perspective, Ablex.

[43] L. Shastri (2007). SHRUTI: A Neurally Motivated Architecture for Rapid, Scalable Inference, In [23], 183-203.

[44] J. Sima (1995). Neural Expert Systems. Neural Networks, 8(2), pp. 261-271.

[45] J. Sima and J. Cervenka (2000). Neural Knowledge Processing in Expert Systems. In I. Cloete and J.M. Zurada (eds.), Knowledge-Based Neurocomputing, The MIT Press, Cambridge, pp. 419-466.

[46] A.-H. Tan (1997). Cascade ARTMAP: Integrating neural computation and symbolic knowledge processing. IEEE Transactions on Neural Networks, 8(2), 237-250.

[47] T.-H. Teng, Z.-M. Tan, A.-H. Tan (2008). Self-Organizing Neural Models Integrating Rules and Reinforcement Learning. In Proceedings of IEEE International Joint Conference on Neural Networks, 3771-3778.

[48] H. Tirri (1995). Replacing the Pattern Matcher of an Expert System with a Neural Network. In Intelligent Hybrid Systems, Goonatilake S. and Sukdev K. (Eds), John Wiley & Sons.

[49] G. Towell and J. Shavlik (1993). Extracting Refined Rules from Knowledge-based Neural Networks, Machine Learning, 13(1), 71-101.

[50] G. Towell and J. Shavlik (1994). Knowledge-based Artificial Neural Networks, Artificial Intelligence, 70(1-2), 119-165.

[51] S. Wermter and R. Sun (eds) (2000). Hybrid Neural Systems, Springer-Verlag.

[52] J. C. Xianyu, Z. C. Juan and L. J. Gao (2008), Knowledge-Based Neural Networks and its Application in Discrete Choice Analysis. In Proceedings of the Fourth International Conference on Networked Computing and Advanced Information Management, pp. 491-496, IEEE Computer Society.

[53] J. Yu, L. Xi and X. Zhou (2008), Intelligent Monitoring and Diagnosis of Manufacturing Processes using an Integrated Approach of KBANN and GA, Computers in Industry, 59(5), 489-501.

[54] L. Yu, L. Wang and J. Yu (2008), Identification of Product Definition Patterns in Mass Customization using a Learning-based Hybrid Approach, International Journal of Advanced Manufacturing Technologies, 38(11-12), 1061-1074.