Construction of Neurules from Training Examples: A Thorough Investigation

Jim Prentzas[1] and Ioannis Hatzilygeroudis[1,2]

Abstract. Neurules are a type of hybrid rules combining a symbolic and a connectionist representation. A neurule base consists of a number of autonomous adaline units (neurules), in contrast to existing neuro-symbolic knowledge bases. A neurule base is constructed from training examples. To overcome the inability of the adaline unit to classify non-separable training examples, the notion of 'closeness' between training examples has been used to split the initial training set into subsets that can be successfully trained. In this paper, we investigate previously unexplored aspects regarding the construction of neurules from training examples. First, we compare different splitting policies, i.e. policies using different criteria for splitting the training set. We also introduce two alternative approaches to splitting not solely relying on closeness and compare them with our initial approach, which is solely based on closeness. The comparison demonstrates the effectiveness of the notion of 'closeness' in splitting the initial non-separable training set. Finally, we evaluate the generalization capability of neurules.

1 INTRODUCTION

Recently there has been extensive research activity at combining (or integrating) the symbolic and the connectionist approaches for problem solving in intelligent systems [3, 4, 5, 12, 13, 14, 15, 19, 21]. Especially, there are a number of efforts at combining symbolic rules and neural networks for knowledge representation [6, 20]. What they do is a kind of mapping from symbolic rules to a neural network. Also, connectionist expert systems are a type of integrated systems that represent relationships between concepts, considered as nodes of a neural network [7, 8]. The strong point of those approaches is that knowledge elicitation from experts is reduced to a minimum. A weak point of them is that the resulted systems lack the naturalness and modularity of symbolic rules. This is mainly due to the fact that those approaches give pre-eminence to connectionism. So, explanations are often provided in the form of if-then rules by rule extraction methods [1, 2].

Neurules constitute a hybrid rule-based representation scheme achieving a uniform and tight integration of a symbolic component (production rules) and a connectionist one (the adaline unit) [8, 9]. In contrast to other integrated approaches, neurules give pre-eminence to the symbolic component. Each neurule is considered as an adaline unit. Thus, neurules give a more natural way of representing knowledge since the constructed knowledge base retains the modularity and (to some degree) the naturalness of symbolic rules. Also, the corresponding inference mechanism, which is a tightly integrated process, results in more efficient inference than those of symbolic rules, and explanations, in the form of if-then rules, can be provided [11]. Mechanisms for efficiently updating a neurule base, given changes to its source knowledge, have also been developed [17, 18].

One way of constructing neurules is from empirical data (i.e., training examples) [10]. A difficult point in this approach is the inherent inability of the adaline unit to classify non-separable training examples. To overcome this difficulty of the adaline unit, we introduced the notion of 'closeness', as far as the training examples are concerned. That is, when the LMS algorithm fails to produce weights that classify all the examples, due to non-separability, we split the initial training set of the involved neurule in two subsets, which contain 'close' examples, and train a copy of the neurule for each subset. Failure of training any copy leads to further splitting as far as success is achieved.

In this paper, we investigate previously unexplored aspects regarding the construction of neurules from training examples. First, we compare different splitting policies, i.e. policies using different criteria for splitting the training set. Second, we introduce alternative approaches to constructing neurules from training examples, not solely relying on closeness to perform splitting. We also compare these alternative approaches with our initial approach, which is solely based on closeness. Finally, we present experimental results evaluating the generalization capability of neurules and comparing it with the generalization capability of a back-propagation neural network and a single adaline unit.

The structure of the paper is as follows. Section 2 presents neurules, the mechanism for their construction from training examples and different splitting policies (based on closeness). Section 3 introduces alternative approaches to splitting (not solely relying on closeness). Section 4 presents experimental results and finally Section 5 concludes the paper.

[1] Technological Educational Institute of Lamia, Department of Informatics and Computer Technology, 35100 Lamia, Greece, email: dprentzas@teilam.gr.
[2] University of Patras, Dept of Computer Engineering & Informatics, 26500 Patras, Greece, email: ihatz@ceid.upatras.gr.

2 NEURULES

2.1 Syntax and Semantics

Neurules (: *neu*ral *rules*) are a kind of hybrid rules. Each neurule (Fig. 1a) is considered as an adaline unit (Fig.1b). The *inputs* C_i (i=1,...,n) of the unit are the *conditions* of the rule. Each condition C_i is assigned a number sf_i, called a *significance factor*, corresponding to the weight of the corresponding input of the adaline unit. Moreover, each rule itself is assigned a number sf_0, called the *bias factor*, corresponding to the *bias* of the unit.

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* of the rule, is calculated via the formulas:

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

where **a** is the *activation value* and *f(x)* the *activation function*, which is a threshold function:

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

Hence, the output can take one of two values, '-1' and '1', representing failure and success of the rule respectively.

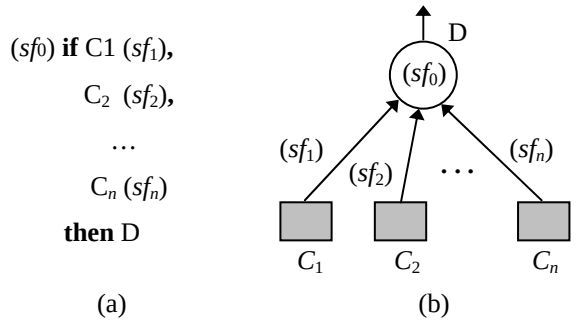

Figure 1. (a) Form of a neurule (b) corresponding adaline unit

The general syntax of a neurule (in a BNF notation, where '{}' denotes zero, one or more occurrences and '<>' denotes non-terminal symbols) is:

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

In the above definition, <variable> denotes a *variable*, that is a symbol representing a concept in the domain, e.g., 'sex', 'pain' etc, in a medical domain. A variable in a condition can be either an *input variable* or an *intermediate variable*, whereas a variable in a conclusion can be either an *intermediate* or an *output variable*. <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 or a number. <bias-factor> and <significance-factor> are (real) numbers. The significance factor of a condition represents the significance (weight) of the condition in drawing the conclusion.

2.2 Constructing Neurules from Training Examples

Each neurule is individually trained via a *training set*, which contains *training examples* in the form [v_1 v_2 … v_n d], where v_i, i= 1, …,n are their *component values*, corresponding to the n inputs of the neurule, and d is the *desired output* ('1' for success, '-1' for failure). We call *success examples* the examples with d=1 and *failure examples* the ones with d=-1. The learning algorithm employed is the standard least mean square (LMS) algorithm.

However, there are cases where the LMS algorithm fails to specify the right significance factors for a number of neurules. That is, the adaline unit of a rule does not correctly classify some of the training examples. This means that the training examples correspond to a non-separable (boolean) function. To overcome this problem, the initial training set is split into two subsets in a way that each subset contains success examples, which are 'close' to each other in some degree. The *closeness* between two examples is defined as the number of common component values. For example, the closeness of [1 0 1 1 1] and [1 1 0 1 1] is '2'. Also, we define as *least closeness pair* (LCP), a pair of success examples with the least closeness in a training set. There may be more than one LCP in a training set.

Initially, a LCP in the training set is found and two subsets are created each containing as its initial element one of the success examples of that pair, called its *pivot*. Each of the remaining success examples is distributed between the two subsets based on its closeness to the pivots. More specifically, each subset contains the success examples, which are closer to its pivot. Then, the failure examples of the initial set are added to both subsets, to avoid neurule misfiring. After that, two copies of the initial neurule, one for each subset, are trained employing the LMS learning algorithm. If the factors of a copy misclassify some of its examples, the corresponding subset is further split into two other subsets, based on one of its LCPs. This continues, until all examples are classified. This means that from an initial neurule more than one final neurule may be produced, called *sibling neurules* (for details see [10]).

To illustrate how splitting is performed, we use as an example the training set presented in Table 1. As it is clear, the majority of the examples in the training set are failure examples, whereas success examples, which are shown in bold, are a minority. The training set has been extracted from empirical data concerning five input (domain) variables and an output variable (disease) that depends on the five domain variables. Given that each input variable can take more than one discrete value, each initial neurule has thirteen conditions (C1-C13). D corresponds to the conclusion. Actually Table 1, for simplicity reasons, shows only a subset of the failure examples.

Table 1. An example training set

C 1	*C 2*	*C 3*	*C 4*	*C 5*	*C 6*	*C 7*	*C 8*	*C 9*	*C 10*	*C 11*	*C 12*	*C 13*	*D*
-1	-1	1	-1	-1	1	1	1	-1	1	-1	-1	-1	-1
-1	**-1**	**1**	**-1**	**-1**	**1**	**1**	**1**	**1**	**-1**	**-1**	**-1**	**-1**	**1**
-1	-1	1	1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	1	-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	**-1**	**1**	**1**	**-1**	**1**	**-1**	**-1**	**-1**	**-1**	**-1**	**1**	**1**	**1**
-1	-1	1	1	-1	1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1	1	-1	1	-1	-1
-1	-1	1	1	-1	1	-1	-1	1	-1	-1	1	-1	-1
-1	**-1**	**1**	**1**	**-1**	**1**	**-1**	**-1**	**1**	**-1**	**1**	**-1**	**-1**	**1**
-1	1	1	-1	-1	-1	-1	1	-1	-1	-1	-1	1	-1
-1	1	1	-1	-1	1	-1	1	-1	-1	-1	-1	-1	-1
-1	1	1	-1	-1	1	-1	1	-1	-1	-1	-1	1	-1
-1	**1**	**1**	**-1**	**-1**	**1**	**-1**	**1**	**-1**	**1**	**-1**	**-1**	**-1**	**1**
1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
1	1	1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1
1	1	1	1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1
1	**1**	**1**	**1**	**1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**-1**	**1**

For presentation reasons, names (P1-P5) are assigned to the five success examples/patterns (of Table 1), as presented in Table 2. Also, let F be the set of failure examples in the training set.

Table 2. Success examples

symbol	description
P1	[-1, -1, 1, -1, -1, 1, 1, 1, 1, -1, -1, -1, -1, 1]
P2	[-1, -1, 1, 1, -1, 1, -1, -1, -1, -1, -1, 1, 1, 1]
P3	[-1, -1, 1, 1, -1, 1, -1, -1, 1, -1, 1, -1, -1, 1]
P4	[-1, 1, 1, -1, -1, 1, -1, 1, -1, 1, -1, -1, -1, 1]
P5	[1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, 1]

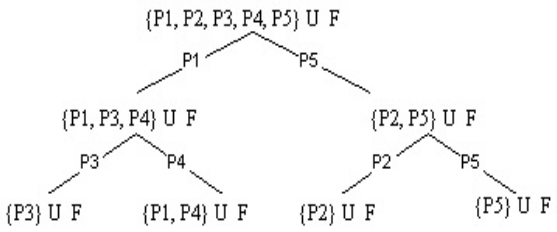

Figure 2. Splittings of the training set of Table 1

Due to inseparability, the initial training set {P1, P2, P3, P4, P5} ∪ F is split in two subsets: {P1, P3, P4} ∪ F and {P2, P5} ∪ F with as least closeness pair (P1, P5). Subset {P1, P3, P4} ∪ F is subsequently split into subsets {P3} ∪ F and {P1, P4} ∪ F. Subset {P3} ∪ F produces a neurule (see Figure 3). Subset {P1, P4} ∪ F produces another neurule. Similarly, from subset {P2, P5} ∪ F two other neurules are produced (corresponding to its two leaves). The performed splittings are illustrated in Figure 2, as a tree.

In creating the training subsets, some requirements were implicitly satisfied. Each training subset contains: (a) all the failure examples of the initial training set to protect from misactivations and (b) at least one success example to guarantee the activation of the corresponding neurule. Furthermore, the two subsets created by splitting a (sub)set do not have common success examples to avoid having different neurules activated by the same success example(s). In the following sections, the approach to splitting based on closeness will be called CLOSENESS-SPLIT.

A point of interest in training a neurule with a non-separable training set is how to choose a least closeness pair (LCP), in the process of producing the two subsets of the initial training set. Not all LCPs result in the same number of final neurules. So, we are looking for the LCP that finally produces the minimum number of sibling neurules. We tried three heuristic methods for that: the *random choice*, the *best distribution* and the *mean closeness method*. The *random choice method* (RC) chooses randomly one of the LCPs and is the simplest and least expensive of the three methods. The *best distribution method* (BD) suggests choosing the LCP that assures distribution of the two elements of all the other (or most of the other) LCPs in different sets. So, examples with least closeness will be included in different sets, which may assure separability. The *mean closeness method* (MC) initially computes the mean closeness of each of the two subsets to be created from each LCP. Then, it calculates the mean closeness of each LCP,

which is the mean closeness of the two subsets, and chooses the LCP with the greatest mean closeness. It is obvious that MC is (computationally) the most expensive method.

NR1
(-13.5) **if** venous-conc is slight (12.4),
blood-conc is moderate (11.6),
art-conc is moderate (8.8),
scan-conc is normal (8.4),
cap-conc is moderate (8.4),
blood-conc is slight (8.3),
venous-conc is moderate (8.2),
venous-conc is normal (8.0),
arterial-conc is slight (-5.7),
cap-conc is slight (4.5),
blood-conc is normal (4.4),
blood-conc is high (1.6),
venous-conc is high (1.2)
then disease is inflammation

Figure 3. One of the produced neurules

3 ALTERNATIVE APPROACHES

In this section, we present two alternative approaches to splitting a non-separable training set not solely relying on closeness. The two alternative approaches will be called ALTERN-SPLIT1 and ALTERN-SPLIT2 respectively. Both of these approaches satisfy the implicit requirements mentioned in the previous section. The idea behind both approaches is simple. More specifically, they focus on the examples which are misclassified by the weights calculated by LMS and try to split the training set into two subsets: one containing the correctly classified success examples (along with all failure examples) and one containing the misclassified success examples (along with all failure examples). This process can be followed only if some (not all) success examples (and possibly failure examples) are misclassified. If all success examples are misclassified or if only failure examples are misclassified, there is no alternative but to split based on closeness. Therefore, in this process one should distinguish the following cases: (a) all of the success examples are misclassified, (b) only failure examples are misclassified, (c) only some of the success examples and none of the failure ones are misclassified, (d) failure examples and some of the success examples are misclassified. In cases (a) and (b) splitting is based on closeness. The two approaches differ only in the way of handling case (d).

More formally, approach ALTERN-SPLIT1 is as follows:

1. If all success examples are misclassified by the calculated weights, split the training set based on closeness.
2. Else, if only failure examples are misclassified, split the training set based on closeness.
3. Else, if only some of the success examples (and none of the failure examples) are misclassified, split the training set in two subsets: one containing the correctly classified success examples (along with all failure examples) and one containing the misclassified success examples (along with all failure examples).
4. Else, if failure examples and some of the success examples are misclassified, split the training set in two subsets: one containing the correctly classified success examples (along with all failure examples) and one containing the misclassified success examples (along with all failure examples).

Approach ALTERN-SPLIT2 does the same as ALTERN-SPLIT1 in steps 1, 2, 3 and handles step 4 based on closeness. It can be easily seen that ALTERN-SPLIT2 lies between CLOSENESS-SPLIT and ALTERN-SPLIT1.

4 EXPERIMENTAL RESULTS

In this section, we present various experimental results using datasets from the UCI Machine Learning Repository [15]. The experimental results involve the following aspects: (a) evaluation of the three different splitting policies based on closeness (i.e., RC, BD, MC), (b) comparison of the three approaches to splitting, CLOSENESS-SPLIT, ALTERN-SPLIT1 and ALTERN-SPLIT2 and (c) evaluation of the generalization capability of neurules and comparison with the generalization capabilities of the back propagation neural networks and the adaline unit.

Table 3. Number of neurules produced by the RC, MC and BD policies

Dataset	Condi-tions	Conclu-sions	RC	MC	BD
Monks1_train (124 patterns)	17	2	17	17	**13**
Monks2_train (169 patterns)	17	2	46	47	**38**
Monks3_train (122 patterns)	17	2	14	**11**	12
Tic-Tac-Toe (958 patterns)	27	2	26	26	**24**
Car (1728 patterns)	21	4	**151**	163	153
Nursery (12960 patterns)	27	5	830	839	**823**

Table 3 depicts experimental results for CLOSENESS-SPLIT comparing RC, MC and BD. Comparison is based on the number of neurules produced from each splitting method, shown in columns 'RC', 'MC' and 'BD'. Column 'Conditions' denotes the number of conditions for each sibling neurule and column 'Conclusions' the number of different (final) conclusions. For the 'monks' datasets we used the training sets provided in the UCI Repository. Based on the results of Table 3, none of the three methods is clearly better than the others for all datasets. Further on, there is no great difference in the number of neurules produced by the three methods. BD performs better in most of the cases. RC, the simplest of the

three methods, performs quite well even in the large datasets compared to the other two more complex methods. On the other hand, MC, which is computationally the most expensive method, does not perform quite well compared to the other methods to justify its use. So, BD or RC can be considered as better alternatives as far as the number of produced neurules is concerned. The number of produced neurules is the basic criterion of the comparisons, because it plays a crucial role in inference efficiency and neurule-base size.

Table 4 presents experimental results regarding ALTERN-SPLIT1 and ALTERN-SPLIT2. RC, MC and BD play a role for subsets in which splitting based on closeness is used.

Table 5 presents summary results comparing the three approaches to splitting, CLOSENESS-SPLIT, ALTERN-SPLIT1 and ALTERN-SPLIT2. Comparison is based on the minimum number of neurules produced from each method. In parentheses, the name of the splitting policy (i.e., RC, BD, MC) used, when producing the minimum number of neurules is shown. CLOSENESS-SPLIT is generally better than the other two methods. This demonstrates the effectiveness of the notion of 'closeness'. This last conclusion is further intensified by the fact that ALTERN-SPLIT2 that lies between ALTERN-SPLIT1 and CLOSENESS-SPLIT generally performs better than ALTERN-SPLIT1. The results also show that it may be worth to employ ALTERN-SPLIT1 and ALTERN-SPLIT2. A further result is that BD generally performs better than RC and MC.

Table 4. Number of neurules produced by ALTERN-SPLIT1 and ALTERN-SPLIT2

Dataset	ALTERN-SPLIT1			ALTERN-SPLIT2		
	RC	MC	BD	RC	MC	BD
Monks1_train	**22**	24	24	19	16	**13**
Monks2_train	34	**32**	33	43	49	**39**
Monks3_train	**15**	**15**	**15**	14	**11**	13
Tic-Tac-Toe	44	41	**40**	43	41	**38**
Car	189	171	**169**	**152**	161	154
Nursery	**1330**	1382	1378	837	842	**821**

Table 5. Number of neurules produced by CLOSENESS-SPLIT, ALTERN-SPLIT1 and ALTERN-SPLIT2

Dataset	CLOSENESS-SPLIT	ALTERN-SPLIT1	ALTERN-SPLIT2
Monks1_train	**13 (BD)**	22 (RC)	13 (BD)
Monks2_train	38 (BD)	**32 (MC)**	39 (BD)
Monks3_train	**11 (MC)**	15 (RC, MC, BD)	11 (MC)
Tic-Tac-Toe	**24 (BD)**	40 (BD)	38 (BD)
Car	**151 (RC)**	169 (BD)	152 (RC)
Nursery	823 (BD)	1330 (RC)	**821 (BD)**

Tables 6 and 7 present results regarding the classification accuracy (generalization) of neurules on unseen test examples. Table 6 compares the classification accuracy of neurules produced from the three splitting policies based on closeness. Table 7 compares the classification accuracy of neurules (i.e., the best result of Table 6) with the ones of the adaline unit and back-propagation neural networks. The results for each dataset (except for the three 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. Needless to say that the training examples in the test sets were not included in the training sets. Different and disjoint test sets were used in each run, so that the union of the four test sets formed the whole dataset. The classification accuracy was computed as the mean value of the accuracies obtained from the four runs. For 'monks1' and 'monks2' datasets this 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 back-propagation neural network for the 'Nursery' dataset with competitive generalization capability.

For the training of back-propagation neural networks, 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. Furthermore, no cross-validation was used when training the adaline unit, the neurules or the back-propagation neural network (perhaps with cross-validation the results of Table 7 for all approaches would have been slightly better). Also, 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) [20].

Table 6. Generalization of neurules produced from RC, MC, BD policies

Dataset	RC	MC	BD
Monks1	100%	100%	100%
Monks2	96.30%	96.99%	**97.92%**
Monks3	92.36%	93.52%	**96.06%**
Tic-Tac-Toe	**98.85%**	97.50%	98.12%
Car	94.44%	**94.56%**	94.50%
Nursery	**99.63%**	99.53%	99.52%

Table 7. Generalization of adaline unit, neurules and back-propagation neural network

Dataset	Adaline Unit	Neurules	BPNN
Monks1	67.82%	100%	100%
Monks2	43.75%	97.92%	100%
Monks3	92.13%	96.06%	97.22%
Tic-Tac-Toe	61.90%	98.85%	98.23%
Car	78.93%	94.56%	95.72%
Nursery	82.26%	99.63%	

The results in Tables 6 and 7 show that neurules generalize quite well. Table 6 shows that none of the three splitting policies performs better than the others in all datasets. Comparing the results of Table 5 and Table 6, it can be said that it is not unlikely that a splitting policy may generalize better

than the other policies although it produced a greater number of neurules. Table 7 shows that neurules outperform the adaline unit and are worse than back-propagation neural networks. These results are very promising. It was expected that the generalization capability of neurules would be somewhere between the adaline unit and the back-propagation neural network. 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.

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 back-propagation neural network. When constructing neurules, one should only try out the different splitting approaches. So, construction of neurules is straightforward. On the other hand, in the case of a back-propagation neural network, 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.

It should be also mentioned that when we developed a method for producing neurules from training examples [10], we did not have generalization as our primary intention. Our effort was to develop an alternative method to the one producing neurules through conversion from existing symbolic rule bases [9]. In this way, the knowledge acquisition process is facilitated since neurules can be constructed from two alternative sources, existing symbolic rule bases and training examples. However, according to the results of this paper, regarding generalization capability of neurules, neurules could be a choice in applications with available training examples and in which naturalness, modularity of the knowledge base and provision of interactive inference and explanation mechanisms are desirable factors besides generalization. Obviously in applications in which generalization is the only concern, one should choose back-propagation neural networks.

5 CONCLUSIONS

In this paper, we investigate previously unexplored aspects regarding the construction of neurules from training examples. Results validate our initial choice, demonstrating the effectiveness of solely using the notion of 'closeness' to handle non-separable training sets. Alternative splitting approaches performed worse. Furthermore, experimental results show that neurules generalize quite well even compared to back-propagation neural networks. Our future research will involve investigation of possible improvements to the construction and generalization capability of neurules.

REFERENCES

[1] R. Andrews, J. Diederich and A. Tickle, 'A survey and critique for extracting rules from trained ANN', *Knowledge-Based Systems*, **8**, 373-389, (1995).

[2] A.S. d'Avila Garcez, K. Broda, D.M. Gabbay, 'Symbolic knowledge extraction from trained neural networks: A sound approach', Artificial Intelligence 125, 155-207, 2001.

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

[4] I. Cloete, J. M. Zurada (eds.), *Knowledge-Based Neurocomputing*, MIT Press, 2000.

[5] A.S. d'Avila Garcez, K. Broda, D.M. Gabbay, *Neural-symbolic Learning Systems: Foundations and Applications*, *Perspectives in Neural Computing*, Springer-Verlag, Heidelberg, 2002.

[6] L-M Fu, *Neural Networks in Computer Intelligence*, McGraw-Hill, 1994.

[7] S.I. Gallant, *Neural Network Learning and Expert Systems*, MIT Press, 1993.

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

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

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

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

[12] I. Hatzilygeroudis and J. Prentzas, 'Neuro-Symbolic Approaches for Knowledge Representation in Expert Systems', *International Journal of Hybrid Intelligent Systems*, **1**, 111-126, (2004).

[13] M. Hilario, 'An Overview of Strategies for Neurosymbolic Integration', in [16].

[14] K. McGarry, S. Wertmer, and J. MacIntyre, 'Hybrid neural systems: from simple coupling to fully integrated neural networks', *Neural Computing Surveys*, **2**, 62-93, (1999).

[15] L.R. Medsker, *Hybrid Neural Networks and Expert Systems*, Kluwer Academic Publishers, Boston, 1994.

[16] D.J. Newman, S. Hettich, C.L. Blake, C.J. Merz, 'UCI Repository of machine learning databases' [http://www.ics.uci.edu/~mlearn/ MLRepository.html]. Irvine, CA, University of California, Department of Information and Computer Science (1998).

[17] J. Prentzas, I. Hatzilygeroudis and A. Tsakalidis, 'Updating a Hybrid Rule Base with New Empirical Source Knowledge', Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence, Washington, DC, USA, 9-15, (2002).

[18] J. Prentzas and I. Hatzilygeroudis, 'Rule-based Update Methods for a Hybrid Rule Base', *Data and Knowledge Engineering*, **55**, 103-128, (2005).

[19] R. Sun and E. Alexandre (eds), *Connectionist-Symbolic Integration: From Unified to Hybrid Approaches*, Lawrence Erlbaum, 1997.

[20] G. Towell and J. Shavlik, 'Knowledge-Based Artificial Neural Networks', *Artificial Intelligence* **70(1-2)**, 119-165, (1994).

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