Published in the Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012), IEEE Computer Society, pp. 1053-1058, 2012.

Controlling the Production of Neuro-Symbolic Rules

Ioannis Hatzilygeroudis *
Dept. of Computer Engineering & Informatics
University of Patras, School of Engineering
Patras 26500, Greece
ihatz@ceid.upatras.gr

Jim Prentzas *
Dept. of Education Sciences in Pre-School Age
Democritus University of Thrace
Nea Chili, Alexandroupolis 68100, Greece
dprentza@psed.duth.gr

***Abstract*—Neurules are a kind of integrated rules integrating neurocomputing and production rules. Neurules can be produced from existing empirical data, through the neurules production algorithm (NPA). In this paper, we present (a) an extension to NPA regarding presentation of neurules, so that they are more natural and more informative, and (b) an experimental comparison of various alternative strategies we can use at some points of NPA targeting at producing as less neurules as possible. Results of (b) show no clear winner for all cases in terms of the gain in number of neurules compared to the computational cost.**

Keywords: neuro-symbolic approaches, integrated rules, hybrid knowledge representation

I. INTRODUCTION

A popular research trend in Artificial Intelligence involves the combination or integration of (two or more) different problem solving or knowledge representation methods. The ensuing approaches have given fruitful results in many application areas. The main objective 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 [17], [16], [21], [19].

Neuro-symbolic approaches combine symbolic and connectionist methods and constitute a popular type of combinations. Advanced neuro-symbolic formalisms and systems have been developed [1], [6], [7], [10], [14], [17]. Different types of neuro-symbolic approaches have been presented. Neuro-symbolic approaches have been presented that combine neural networks with first-order logic [9], multi-valued logic or with other types of logic such as temporal logic [3]. However, combinations of connectionism with symbolic rules (of propositional type) ([5], [8], [24], [2], [22]) seem to be the most popular trend in neuro-symbolic approaches. This is mainly due to complementary advantages and disadvantages of the combined formalisms [14].

Neurules are a type of integrated rules combining symbolic rules (of propositional type) and neurocomputing [11], [13], [22]. In contrast to other approaches, neurules give pre-eminence to the symbolic part of the integration. Therefore, they retain the naturalness and modularity of symbolic rules in a large degree. Also a neurule-based system possesses an interactive inference mechanism [15] and provides explanations for drawn conclusions [23].

Neurules can be produced either from symbolic rules [11] or from empirical data [13], [20]. In this paper, we concentrate on the latter. We present an extension to the basic algorithm, presented in [13, 15], concerning presentation of neurules and a number of new experiments concerning use of alternative strategies in producing neurules.

This paper is structured as follows. Section II presents the neurule-based knowledge representation scheme. In Section III, naturalness and informativeness of rules are discussed. In Section IV, production of neurules from empirical data is presented. Section V presents various alternative strategies to be used in producing neurules, whereas in Section VI experimental results are presented. Finally, Section VII concludes.

II. NEURULES

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 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:

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

$$f(a) = \begin{cases} 1, & if \quad a \geq 0 \\ -1, & if \quad a < 0 \end{cases} \quad (2)$$

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.

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

* The names of the authors appear in alphabetic order

Published in the Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012), IEEE Computer Society, pp. 1053-1058, 2012.

<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", "night-pain") or a number (e.g "5"). <bias-factor> and <significance-factor> are (real) numbers.

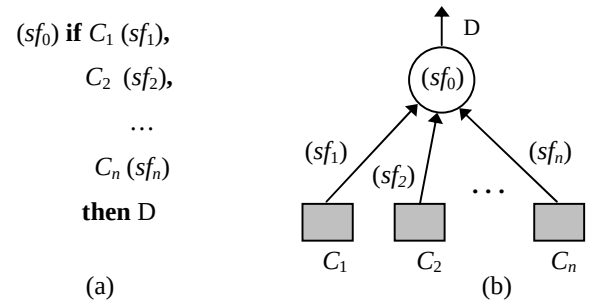

Figure 1. (a) Form of a neurule (b) a neurule as an adaline unit

For the sake of simplicity, in the following, we consider "is" as the only <l-predicate> and <r-predicate>, without loss of generality, since the others can be somehow expressed via it.

III. NATURALNESS AND INFORMATIVENESS IN RULES

As it is known, variables and values represent concepts in the problem domain. Specification of variables and their corresponding values is an important task in modeling 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.

For example, in a medical domain, we could have the following rules that refer to the same knowledge:

R1
if fever is high
and pain is local
then disease is inflammation

R2
if symptom is high-fever
and symptom is local-pain
then disease is inflammation

R3
if high-fever is true
and local-pain is true
then inflammation is true

In R1 and R2, variables with discrete values (probably more than two) are used, whereas in R3, boolean variables (with as only values 'true' and 'false') are used. The values in R2 are used as variables in R3. R2 is probably more informative than R1. Finally, R2 and R1 are probably more natural than R3. Sometimes, we may have dependency information and empirical data in a form that leads to R3-like rules, but we want to have them in a R2-like form, to be more informative and more natural. So, we need to be able to rewrite produced neurules.

IV. PRODUCING A NEURULE BASE

Production of a neurule base is achieved by applying the *neurule production algorithm* (NPA) to a dataset (see [15]). NPA tries to produce one neurule for each intermediate or output variable-value pair (having that pair in its conclusion). However, due to possible non-linearity, this is not usually feasible. So, more than one neurule having the same conclusion are produced.

First, we extend here the main algorithm, presented in [15], to include control of naturalness and informativeness of the neurules by adding a fourth step. The algorithm is as follows:

NPA Algorithm
Input:
(a) A set V of domain variables
(b) Dependency information f_{DI}^{V}
(c) Presentation information f_{PI}^{V}
(d) A set S of empirical data
Output: *A set of neurules*
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).*
4. *Rewrite neurules based on presentation information.*

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 set of discrete values $S_{V_i} = \{v_{ij}\}$, $1 \le j \le k$. We distinguish various types of domain variables as follows. First, *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}. Second, *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}. Furthermore, *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.

Dependency information is related to how concepts (i.e. corresponding domain variables) depend on each other in making decisions.

Definition 1. *The* ***dependency information*** f_{DI}^{V} *related to the set of domain variables V is a relation* $f_{DI}^{V} : V \times V \rightarrow \{T, F\}$ *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*".*

Published in the Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012), IEEE Computer Society, pp. 1053-1058, 2012.

Apart from domain variables we also specify a set of *presentation variables* $I = \{I_i\}$, $1 \le i \le p$. Presentation variables are used only for rule presentation reasons, to increase their informativeness and naturalness. They are used to rewrite the conditions of rules that include Boolean variables. To this end, alongside presentation variables, we specify corresponding *presentation information*.

Definition 2. *The **presentation information** f_{PI}^{V} related to the set of domain variables V is a relation $f_{PI}^{V}: V \times I \rightarrow \{T, F\}$ represented as a set of ordered pairs $f_{PI}^{V} = \{(I_i, V_i): I_i \in I, V_i \in V\}$. Each (I_i, V_i) is interpreted as "V_i is presented via I_i".*

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: $t_i = < v_{i1}, v_{i2}, \ldots, v_{im}>$ 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 a 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 $v_{i_d}^{k}$. So, we call *success patterns*, represented by *succ(S)*, those having '1' and *failure patterns*, represented by *fail(S)*, those having '-1' as a goal value.

V. Controlling Neurule Production

A. Splitting Strategies

In step 3 of the above algorithm, in case of a non-linear training set, we split it into two "suitable" subsets hoping that they will be linear. If not, we further split the non-linear one(s) to other two subsets and so on. To do splitting, we can use a number of strategies. Those strategies should perform splitting in a way that satisfies the following three requirements:

- Each training subset contains all the failure examples of the initial training set to protect corresponding neurule from misactivations.
- Each training subset contains at least one success example to guarantee the activation of the corresponding neurule.
- 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).

We also introduce the following definitions:

Definition 3. *The **distance** between two success patterns t_i and t_j, denoted by $d(t_i, t_j)$, is a number representing how "far" the two patterns are between each other in the hyperspace (specific metrics are discussed later on in this section).*

Definition 4. *The **set of least closeness pairs** P_k of the training set S_k of a neurule R_k is a set containing the pairs of success patterns of S_k that have the maximum distance: $P_k = \{(t_{i1}, t_{i2}): i_1 \neq i_2, t_{i1}, t_{i2} \in succ(S_k) \wedge \forall k, \forall l, k \neq l, t_k, t_l \in succ(S_k) \Rightarrow d(t_i, t_j) \geq d(t_k, t_l)\}$.*

Let $P_k = \{p_1, p_2, \ldots, p_r\}$; each $p_i \equiv (t_{i1}, t_{i2})$ is called a *least closeness pair* (LCP).

In this section, we present three alternatives splitting strategies. A splitting strategy regards implementation of step 3.3 of NPA (presented below).

According to *closeness-based strategy* (*CLOSE-SPLIT*), splitting a training set is based on a LCP, which is selected in some way (see further on), called the *splitting LCP* (SLCP). That is, each subset comprises one of the members of SLCP, called its *pivot*, the success patterns closer to it and all the failure patterns of the initial training set. This stems from the intuition that existence of quite different patterns causes non-separability (i. e. non-linearity).

More formally, step 3.3 of NPA is defined as follows:

3.3.1 Produce P_k (based on Definition 4).

3.3.2 Select $p_k \in P_k$, where $p_k \equiv (t_{k1}, t_{k2})$, as SLCP.

3.3.3 Split $succ(S_k)$ into $succ(S_k)^1$ and $succ(S_k)^2$, such that $t_{k1} \in succ(S_k)^1$, $t_{k2} \in succ(S_k)^2$ and $\forall t_i \in succ(S_k)^1$, $\forall t_j \in succ(S_k)^2$, $d(t_i, t_{k2}) < d(t_i, t_{k1})$ and $d(t_j, t_{k2}) < d(t_j, t_{k1})$.

3.3.4 $S_k^1 \leftarrow succ(S_k)^1 \cup fail(S_k)$, $S_k^2 \leftarrow succ(S_k)^2 \cup fail(S_k)$.

While CLOSE-SPLIT strategy is solely based on closeness (or distance) between success patterns, *strong pattern-based strategy* (STRONG-PAT-SPLIT), as well as next one (WEAK-PAT-SPLIT), partially focuses on closeness and partially on the misclassified patterns. More specifically, STRONG-PAT-SPLIT and WEAK-PAT-SPLIT use CLOSE-SPLIT only in cases that any of the requirements (set at the beginning of this section) would be violated if based on misclassified patterns. We give a formal description of these two alternative splitting strategies initially introduced in [20].

The STRONG-PAT-SPLIT strategy is as follows, where by $misclass(S_k)$ and $corclass(S_k)$ we denote the sets of the patterns of the training set S_k which are misclassified and correctly classified respectively by the calculated weights:

3.3.1 If $succ(S_k) \subseteq misclass(S_k)$, use CLOSE-SPLIT

3.3.2 If $misclass(S_k) \cap fail(S_k) \neq \varnothing$ and $misclass(S_k) \cap succ(S_k) = \varnothing$, use CLOSE-SPLIT

3.3.3 If $misclass(S_k) \cap succ(S_k) \neq \varnothing$ and $misclass(S_k) \cap fail(S_k) = \varnothing$, $S_k^1 \leftarrow corclass(S_k) \cup fail(S_k)$ and $S_k^2 \leftarrow misclass(S_k) \cup fail(S_k)$

3.3.4 If $misclass(S_k) \cap succ(S_k) \neq \varnothing$ and $misclass(S_k) \cap fail(S_k) \neq \varnothing$, $S_k^1 \leftarrow corclass(S_k) \cup fail(S_k)$ and $S_k^2 \leftarrow misclass(S_k) \cup fail(S_k)$.

The *weak pattern-based strategy* (*WEAK-PAT-SPLIT*) is the same as the STRONG-PAT-SPLIT strategy, except that in step 3.3.4 above CLOSE-SPLIT is used too. So, it is clear that WEAK-PAT-SPLIT lies between CLOSE-SPLIT and

Published in the Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012), IEEE Computer Society, pp. 1053-1058, 2012.

STRONG-PAT-SPLIT, with regards to how much it is based on closeness/distance.

B. Distance Metrics

In the above, something that is left undetermined is how we measure the distance between two success examples. There are various ways to do that. We have experimented with some such metrics. First, we used the well-known *Euclidean* and *Manhattan distance* metrics. An important notice here is that variables in our examples are categorical and/or ordinal; so, we used the above distance metrics only on sets with ordinal variables, after having made a mapping of their values to suitable ranking. Apart from those, we used metrics mainly suitable for categorical variables, such as the *simple distance (SD) metric* [13], [20] and, a more complicated one, the *Value Difference Metric* (VDM) [18]. More specifically, the metrics, which we experimented with, are:

- Simple distance (SD)

$$d(t_i,t_j)=\sum_{l=1}^{m-1} D(v_{il},v_{jl})$$

$$D(v_{il},v_{jl})=\begin{cases}0, v_{il}=v_{jl}\\ 1, v_{il}\neq v_{jl}\end{cases}$$

- Euclidean distance

$$d(t_i,t_j)=\left[\sum_{l=1}^{m-1}(v_{il}-v_{jl})^2\right]^{\frac{1}{2}}$$

- Manhattan distance

$$d(t_i,t_j)=\sum_{l=1}^{m-1}|v_{il}-v_{jl}|$$

- Value Difference Metric (VDM)

$$d(t_i,t_j)=\sum_{l=1}^{m-1} D(v_{il},v_{jl})$$

$$D(v_{il},v_{jl})=\frac{1}{\left|S_{V_d^k}\right|}\sum_{v_d^k\in S_{V_d^k}}\left|\frac{N(V_l=v_{il}\mid v_d^k)}{N(V_l=v_{il})}-\frac{N(V_l=v_{jl}\mid v_d^k)}{N(V_l=v_{jl})}\right|$$

where $\left|S_{V_d^k}\right|$ is the cardinality of $S_{V_d^k}$, $N(V_l=v_{il}\mid v_d^k)$ is the number of patterns with desired output value v_d^k and having as value v_{il} for variable V_l, $N(V_l=v_{il})$ is the number of all patterns having as value v_{il} for variable V_l, $N(V_l=v_{jl}\mid v_d^k)$ is the number of patterns with desired output value v_d^k and having as value v_{jl} for variable V_l and $N(V_l=v_{jl})$ is the number of all patterns having as value v_{jl} for variable V_l.

C. Selecting SLCP

A final point of interest, is how to select SLCP, in step 3.3.2 of the CLOSE-SPLIT strategy. Not all SLCPs result in the same number of final neurules. So, we are looking for an SLCP that finally produces the minimum number of sibling neurules, because it affects the efficiency of inferences [11]. We have experimented with three heuristic strategies [20]: *random choice* (RC), *best distribution* (BD) and *mean distance* (MD).

- RC chooses randomly one of the LCPs. This is the simplest and least computationally expensive of the three methods.
- 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, patterns with the greatest distance will be included in different sets, which may reduce non-linearity. More formally, the corresponding algorithm is:

  1. $p_k \leftarrow p_1$
  2. Produce $P_k^{p_k} \leftarrow \{p_m, m \neq k$: $t_{m1}, t_{m2}\in succ(S_k)^1$ or $t_{m1}, t_{m2}\in succ(S_k)^2\}$
  3. For each $p_i \in P_k$ (i >1) do
     3.1 Produce $P_k^{p_i} \leftarrow \{p_m, m \neq i$: $t_{m1}, t_{m2}\in succ(S_k)^1$ or $t_{m1}, t_{m2}\in succ(S_k)^2\}$
     3.2 if $|P_k^{p_i}| < |P_k^{p_k}|$, $p_k \leftarrow p_i$

Notice that subsets $succ(S_k)^1$ and $succ(S_k)^2$ do not need to be produced. Merely, the distances between the success patterns of LCPs p_k and p_i and the success patterns of each other LCP are calculated respectively.

MD suggests choosing the LCP that creates the two subsets with the least mean distance. Initially computes the mean distance of each of the two subsets to be created for each LCP. The mean distance of a subset is the mean distance of its patterns. Then, it calculates the mean distance of each LCP, which is the mean distance of the two subsets, and chooses the LCP with the least mean distance. More formally (where $\bar{d}$ (S) denotes the mean distance of set S):

1. Set $p_k = p_1$
2. Produce $succ(S_k)^1$ and $succ(S_k)^2$ (see step 3.3.3 of CLOSE-SPLIT)

Published in the Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012), IEEE Computer Society, pp. 1053-1058, 2012.

3. Compute $d_{k1}=\overline{d}\,(succ(S_k)^1_{p_k})$, $d_{k2}=\overline{d}\,(succ(S_k)^2_{p_k})$ and $d_k = (d_{k1} + d_{k2})/2$
4. For each $p_i \in P_k$ (i >1) do
   4.1 Produce succ $(S_k)^1_{p_i}$ and succ $(S_k)^2_{p_i}$ (see step 3.3.3 of CLOSE-SPLIT)
   4.2 Compute $d_{i1}=\overline{d}\,(succ(S_k)^1)$, $d_{i2}=\overline{d}\,(succ(S_k)^2)$ and $d_i = (d_{i1} + d_{i2})/2$
   4.3 if $d_i < d_k$, set $p_k = p_i$

where the mean distance of a subset (in steps 3 and 4.2) is calculated by the formulas:

$$\overline{d}(S_k)=\frac{\sum_{t_i,t_j\in S_k, i\neq j} d(t_i,t_j)}{N},\quad N=\frac{n!}{2(n-2)!},\quad n=|S_k|$$

It is obvious that, computationally, MD is the most expensive method.

D. Neurule Presentation

Steps 1-3 of the NPA algorithm are analyzed in [15]. Here we analyze step 4, which is the new step introduced here:

4. For each R_k do
   4.1 For each condition $C_i^{R_k}$ of R_k do
      4.1.1 If the variable $V_i^{R_k}$ of $C_i^{R_k}$ is member of an element of f_{PI}^V, replace $C_i^{R_k}$ with " I_i is $V_i^{R_k}$ "
   4.2 For the conclusion $D_i^{R_k}$ of R_k do
      4.2.1 If the variable $V_d^{R_k}$ of $D_i^{R_k}$ is member of an element of f_{PI}^V, replace $D_i^{R_k}$ with " I_i is $V_d^{R_k}$ "

To illustrate application of step 4, we give an example. In [15] the theoretical diseases of the sarcophagus dataset [5] is used to illustrate neurule production based on the first three steps of NPA. Two of the produced neurules are presented in Table I.

TABLE I. NEURULES PRODUCED FROM THREE-STEP NPA

R_1 (-0.4) if SwollenFeet is true (3.6), HairLoss is true (3.6), RedEars is true (-0.8) then Supercilliosis is true	R2 (1.4) if Dizziness is true (4.6), SensitiveAretha is true (1.8), HairLoss is true (1.8) then Namastosis is true

The (partial) presentation information given is:

f_{PI}^V ={(Symptom, SwollenFeet), (Symptom, RedEars), (Symptom, HairLoss), (Symptom, Dizziness), (Symptom, SensitiveAretha), (Disease, Supercilliosis), (Disease, Namastosis), ...}

After execution of the fourth step the rules are re-written as displayed in Table II.

TABLE II. NEURULES PRODUCED FROM FOUR-STEP NPA

R1 (-0.4) if Symptom is SwollenFeet (3.6), Symptom is HairLoss (3.6), Symptom is RedEars (-0.8) then Disease is Supercilliosis	R2 (1.4) if Symptom is Dizziness (4.6), Symptom is SensitiveAretha (1.8), Symptom is HairLoss (1.8) then Disease is Namastosis

VI. EXPERIMENTAL RESULTS

We made a number of experiments to compare the SLCP selection and splitting strategies as well as the distance metrics previously specified. To this end, we used datasets from the UCI Machine Learning Repository [4]. The comparison is based on the number of produced neurules. In [20] we presented experimental results regarding use of RC, MC and BD in conjunction with the three splitting strategies and SD as distance metric, which are not repeated here.

Tables III-V present the number of neurules produced from each of the datasets for each of the three SLCP selection and splitting strategies when the VDM, Euclidean and Manhattan metrics are used. Notice that the Euclidean and Manhattan metrics cannot be used in the first four datasets, since they contain only categorical variables whose values cannot be mapped to a ranking.

From the above results and the ones presented in [20], as to the distance metrics, it is clear that for the two largest sets, i.e. the 'car' and 'nursery' sets, VDM gives significantly better results than Euclidean, Manhattan and SD metrics, with the Euclidean and Manhattan being in the second place. For the first two sets, however, SD seems to do better than VDM. As to the SLCP selection strategies, in the two largest sets, BD and MC do better than RC.

Table VI presents summary results comparing all three splitting strategies, CLOSE-SPLIT, STRONG-PAT-SPLIT and WEAK-PAT-SPLIT in terms of distance metrics and SLCP selection strategies. Comparison is based on the minimum number of neurules produced by each method. In parentheses, the name of the corresponding SLCP selection policy (i.e. one of RC, BD, MC) and the metric (i.e., SD, Euclidean, Manhattan and VDM) that produce the minimum number of neurules are shown. CLOSE-SPLIT is generally better than the other two methods, WEAK-PAT-SPLIT being in the second place. This demonstrates the effectiveness of the notion of 'closeness'. This last conclusion is also supported by the fact that WEAK-PAT-SPLIT, which lies between STRONG-PAT-SPLIT and CLOSE-SPLIT, generally performs better than STRONG-PAT-SPLIT. The results also show that it may be worth to employ STRONG-PAT-SPLIT and WEAK-PAT-SPLIT. A further result is that BD generally performs better than RC and MC whereas MC generally performs better than RC.

TABLE III. COMPARING RC, MC, BD STRATEGIES (CLOSE-SPLIT)

Dataset	VDM			Euclidean/Manhattan		
	RC	MC	BD	RC	MC	BD
Monks1_train	16	16	16			
Monks2_train	46	52	52			
Monks3_train	9	9	9			
Tic-Tac-Toe	23	32	32			

Published in the Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012), IEEE Computer Society, pp. 1053-1058, 2012.

Car	55	54	54	112/109	100/113	99/103
Nursery	143	127	127	635/577	642/565	647/566

TABLE IV. COMPARING RC, MC AND BD (STRONG-PAT -SPLIT)

Dataset	VDM			Euclidean/Manhattan		
	RC	MC	BD	RC	MC	BD
Monks1_train	24	24	24			
Monks2_train	31	31	31			
Monks3_train	15	15	15			
Tic-Tac-Toe	51	50	50			
Car	143	110	110	138/ 155	139/ 158	139/ 158
Nursery	869	826	826	1266/ 1230	1061/ 1221	1169/ 1247

TABLE V. COMPARING RC, MC AND BD (WEAK-PAT -SPLIT)

Dataset	VDM			Euclidean/Manhattan		
	RC	MC	BD	RC	MC	BD
Monks1_train	17	17	17			
Monks2_train	47	52	52			
Monks3_train	8	10	10			
Tic-Tac-Toe	51	49	49			
Car	58	55	55	103/102	98/116	97/104
Nursery	153	126	126	597/573	646/568	651/569

TABLE VI. COMPARING SPLITTING STRATEGIES (CLOSE-SPLIT, STRONG/WEAK-PAT-SPLIT)

Dataset	CLOSE-SPLIT	STRONG-PAT-SPLIT	WEAK-PAT-SPLIT
Monks1_train	13 (BD-SD)	22 (RC-SD)	13 (BD-SD)
Monks2_train	38 (BD-SD)	31 (RC,MC,BD-VDM)	39 (BD-SD)
Monks3_train	9 (RC,MC,BD-VDM)	15 (RC, MC, BD-SD/VDM)	10 (MC,BD-VDM)
Tic-Tac-Toe	23 (RC-VDM)	40 (BD-SD)	38 (BD-SD)
Car	54 (MC,BD-VDM)	110 (MC,BD-VDM)	55 (MC,BD-VDM)
Nursery	127 (MC,BD-VDM)	826 (MC,BD-VDM)	126 (MC,BD-VDM)

VII. CONCLUSION

In this paper, we analyze certain aspects regarding the construction of neurules, a type of integrated rules.

The analysis concerns the construction process of neurules from empirical data. A difficulty that arises in the neurule-based representation scheme involves the inability of the adaline unit to classify non-separable training examples. In such situations, the training set should be split into subsets according to a splitting policy. We presented different splitting policies for splitting training sets that involve non-separable training examples. We also experimented with different distance metrics used to measure the distance among training patterns. Experiments were performed using publicly available datasets. One of the findings is that VDM, the computationally most expensive metric, although does quite better than the other metrics in large datasets, it doesn't in small datasets.

Another concern is the presentation form of neurules. The new construction algorithm allows for neurule re-writing to increase their naturalness and informativeness based on presentation information provided by the user.

So, a potential tool for constructing neurules (like e.g. [12]) should incorporate all alternative policies and metrics.

REFERENCES

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

[2] J.K. Baea, Jinhwa Kim, "Combining models from neural networks and inductive learning algorithms", Expert Systems with Applications, vol. 38, pp. 4839–4850, 2011.

[3] R.V. Borges, A. d'Avila Garcez, and L.C. Lamb, "Learning and Representing Temporal Knowledge in Recurrent Networks', IEEE Transactions on Neural Networks, vol. 22, pp. 2409-2421, 2011.

[4] A. Frank, A. Asuncion, UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, 2010.

[5] S.I. Gallant, Neural Network Learning and Expert Systems. Cambridge: The MIT Press, 1993.

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

[7] A. d'Avila Garcez, L.C. Lamb, and D.M. Gabbay, Neural-Symbolic Cognitive Reasoning. Berlin Heidelberg: Springer-Verlag, 2009.

[8] A.Z. Ghalwash, "A recency inference engine for connectionist knowledge bases", Applied Intelligence, vol. 9, pp. 201-215, 1998.

[9] M. Guillame-Bert, K. Broda, A. d'Avila Garcez, "First-order logic learning in Artificial Neural Networks", in Proceedings of the International Joint Conference on Neural Networks, IEEE, 2010.

[10] B. Hammer, P. Hitzler (Eds.), Perspectives of Neural-Symbolic Integration, Studies in Computational Intelligence, vol. 77, Berlin Heidelberg: Springer-Verlag, 2010.

[11] I. Hatzilygeroudis, J. Prentzas, "Neurules: improving the performance of symbolic rules", International Journal on AI Tools, vol. 9, pp. 113-130, 2000.

[12] I. Hatzilygeroudis, J. Prentzas, "HYMES: A HYbrid Modular Expert System with efficient inference and explanation" in Proceedings of the 8th Panhellenic Conference on Informatics, vol.1, pp. 422-431, 2001.

[13] I. Hatzilygeroudis, J. Prentzas, "Constructing modular hybrid knowledge bases for expert systems", International Journal on AI Tools, vol. 10, pp. 87-105, 2001.

[14] I. Hatzilygeroudis, J. Prentzas, "Neuro-symbolic approaches for knowledge representation in expert systems", International Journal on Hybrid Intelligent Systems, vol. 1, pp. 111-126, 2004.

[15] I. Hatzilygeroudis, J. Prentzas, "Integrated rule-based learning and inference", IEEE Transactions on Knowledge and Data Engineering vol. 22, pp. 1549-1562, 2010.

[16] I. Hatzilygeroudis, J. Prentzas (Eds.), Combinations of Intelligent Methods and Applications, Smart Innovation, Systems and Technologies, vol. 8. Berlin Heidelberg: Springer-Verlag, 2011.

[17] L.R. Medsker, Hybrid Intelligent Systems. Boston: Kluwer Academic Publishers, second reprinting 1998.

[18] T.R. Payne, P. Edwards, "Implicit feature selection with the Value Difference Metric", in H. Prade (Ed.) Proceedings of the 13th European Conference on Artificial Intelligence. John Wiley & Sons, 1998.

[19] J. Prentzas, I. Hatzilygeroudis, C. Koutsojannis, "A Web-based ITS controlled by a hybrid expert system", in Proceedings of the IEEE International Conference on Advanced Learning Technologies, pp. 239-240. IEEE Computer Society, 2001.

[20] J. Prentzas, I. Hatzilygeroudis, "Construction of neurules from training examples: a thorough investigation", in A. Garcez, P. Hitzler, G. Tamburini (Eds.) Proceedings of the Second International Workshop on Neural-Symbolic Learning and Reasoning, pp. 35-40, 2006.

Published in the Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012), IEEE Computer Society, pp. 1053-1058, 2012.

[21] J. Prentzas, I. Hatzilygeroudis, "Combinations of case-based reasoning with other intelligent methods', International Journal of Hybrid Intelligent Systems, vol. 6, pp. 189-209, 2009.

[22] J. Prentzas, I. Hatzilygeroudis, "Neurules – A type of neuro-symbolic rules: an overview", in I. Hatzilygeroudis, J. Prentzas (Eds.), Combinations of Intelligent Methods and Applications, Smart Innovation, Systems and Technologies, vol. 8, pp. 145-165. Berlin Heidelberg: Springer-Verlag, 2011.

[23] J. Prentzas, I. Hatzilygeroudis, "An explanation mechanism for integrated rules", in Proceedings of the 3rd International Workshop on Combinations of Intelligent Methods and Applications, pp. 37-42, 2012.

[24] G. Towell, J. Shavlik, "Knowledge-based artificial neural networks", Artificial Intelligence, vol. 70, pp. 119-165, 1994.