Published in the Proceedings of the 14th IEEE International Conference on AI Tools (ICTAI-02), 4-6 Nov. 2002, Washington, DC, USA, 9-15.

Updating a Hybrid Rule Base with New Empirical Source Knowledge

Jim Prentzas, Ioannis Hatzilygeroudis, Athanasios Tsakalidis
University of Patras, School of Engineering
Dept of Computer Engin. & Informatics, 26500 Patras, Hellas (Greece)
&
Computer Technology Institute, P.O. Box 1122, 26110 Patras, Hellas (Greece)
{prentzas, ihatz}@ceid.upatras.gr, {ihatz, tsak}@cti.gr

Abstract

Neurules are a kind of hybrid rules that combine a symbolic (production rules) and a connectionist (adaline unit) representation. Each neurule is represented as an adaline unit. One way that the neurules can be produced is from training examples (empirical source knowledge). However, in certain application fields not all of the training examples are available a priori. A number of them become available over time. In these cases, updating the corresponding neurules is necessary. In this paper, methods for updating a hybrid rule base, consisting of neurules, to reflect the availability of new training examples are presented. The methods are efficient, since they require the least possible retraining effort and the number of the produced neurules is kept as small as possible.

1. Introduction

There has been extensive research activity at combining (or integrating) the symbolic and the connectionist approaches for knowledge representation in expert systems [3, 15, 16, 19]. Especially, there are a number of efforts combining symbolic rules and neural networks that map rules into neural networks [4, 9, 18]. In addition, connectionist expert systems [6, 7, 17] are a type of integrated systems that represent relationships between concepts, considered as nodes of a neural network. The above approaches give pre-eminence to connectionism and use a neural network as a knowledge base. The strong point of those approaches is that knowledge elicitation from experts is reduced to a minimum. A weak point is that their knowledge base lacks the naturalness and modularity of symbolic rules; it is incomprehensible. Therefore, often explanations are provided in the form of if-then rules by rule extraction methods [1].

Neurules [10] integrate symbolic rules and connectionism, but in a different way. They give pre-eminence to the symbolic component. Neurocomputing is used within the symbolic framework to improve the performance of symbolic rules. The constructed knowledge base retains the modularity of production rules, since it consists of autonomous units (neurules), and also retains their naturalness in a great degree, since neurules look much like symbolic rules. Also, the inference mechanism is a tightly integrated process, which results in more efficient inferences than those of symbolic rules and explanations in the form of if-then rules can be produced [12].

One way that the neurules, called the *target knowledge*, can be produced is from training examples [11], called the *empirical source knowledge*. However, in certain application fields (e.g. user modeling/profiling, intelligent agents, intelligent user interfaces and robotics) not all of the training examples are available a priori. A number of them become available over time. This happens either because the environment changes with time or because the rate at which examples become available may be too slow [8]. Therefore, methods should be developed for the various types of classifiers dealing with this problem [5]. These methods may or may not require re-examination of all or part of the empirical source knowledge [14]. The methods must be effective as far as the retraining effort and the size of the target knowledge are concerned.

In this paper, we present methods for efficient maintenance of the target knowledge, due to changes to the empirical source knowledge of a neurule-based expert system. Section 2 presents neurules. The methods are introduced in Section 3. Section 4 gives some examples and Section 5 presents experimental results. Finally, Section 6 concludes.

2. Neurules

2.1 Syntax and semantics

Neurules are a kind of hybrid 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 (see e.g. [6]):

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

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

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.

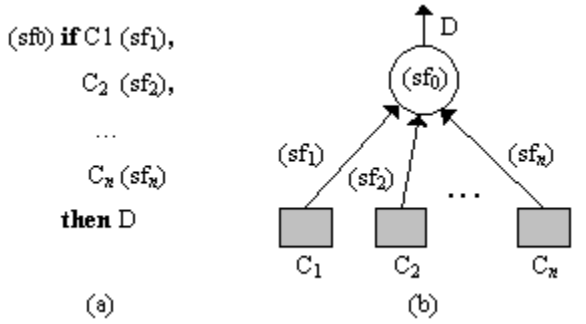

Figure 1. (a) Form of a neurule (b) a neurule as an adaline unit

The general syntax of a condition C_i and the conclusion D is:

<condition>::= <variable> <l-predicate> <value>
<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. <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.

2.2 Constructing a neurule-base

One way of constructing neurules is from empirical data (i.e. training examples /patterns) [11]. What is required is the *dependency information* concerning the domain variables and a set of empirical data, which we call the *source set*. They constitute the *empirical source knowledge*. Based on the dependency information, the *initial neurules* are constructed. Then, for each initial neurule its corresponding *initial training set* is extracted from the source set (for details see [11]). A *training example/pattern* has the form [v_1 v_2 … v_n d], where d is the desired value of a variable related to a partial or final conclusion and v_i, i= 1, …,n are the values of the variables it depends on, called *component values*. We distinguish between *success examples* and *failure examples* in a training set. Success examples are those having '1' as their d value, whereas failure examples those having a '-1'. Furthermore, the *closeness* between two examples is defined as the number of their common component values. So, a *least* closeness *pair* (*LCP*) consists of two success examples that have the least closeness between them. There may be more than one LCP in a training set.

Each initial neurule is individually trained via the Least Mean Square (LMS) algorithm (see e.g. [6]) using its own training set. When the algorithm succeeds, that is values for the bias and significant factors are calculated that classify all training examples, a 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. Splitting a training set is based on a LCP. That is, each subset comprises one of the members of a LCP (randomly chosen), the success examples closer to it and all the failure examples of the initial training set. 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* (for details, also see [11]). The conditions of the neurules are organized according to the descending order of their significance factors. This increases inference efficiency [11].

For reasons that will become clear in the sequel, we introduce here the notion of a *splitting tree*. For each initial training set, its splitting process (if there is one) is stored as a tree, which is called its splitting tree. The root of the tree corresponds to the initial training set. The intermediate nodes and leaves correspond to the subsequent subsets into which the initial training set was split in. An intermediate node denotes a subset from a split, due to training failure, whereas a leaf denotes a subset that was successfully trained and produced a neurule. The members of the least closeness pair that guided each split are attached to the corresponding edges of the tree. It can be easily seen that the training (sub)set of the root or an intermediate node is a superset of the training subsets related to its descendant nodes. Furthermore, the nearer one gets to the leaves, the greater the mean closeness between the training examples of the corresponding training subsets. Tree information is assigned to each initial training set that had to split.

Table 1. An example training set

C1	*C2*	*C3*	*C4*	*C5*	*C6*	*C7*	*C8*	*C9*	*C10*	*C11*	*C12*	*C13*	*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**

Table 2. Example conditions and conclusion

symbol	description
C1	arterial-concentration is slight
C2	blood-concentration is normal
C3	scan-concentration is normal
C4	capillary-concentration is moderate
C5	venous-concentration is high
C6	arterial-concentration is moderate
C7	blood-concentration is high
C8	capillary-concentration is slight
C9	venous-concentration is slight
C10	venous-concentration is normal
C11	blood-concentration is moderate
C12	blood-concentration is slight
C13	venous-concentration is moderate
D	disease is inflammation

Table 3. 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]

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 (arterial-concentration, blood-concentration, scan-concentration, capillary-concentration, venous-concentration) and a conclusion 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), presented in Table 2. D corresponds to the conclusion.

For presentation reasons, names (P1-P5) are assigned to the five success examples/patterns (of Table 1), as presented in Table 3. Also, let F be the set of failure examples in the training set.

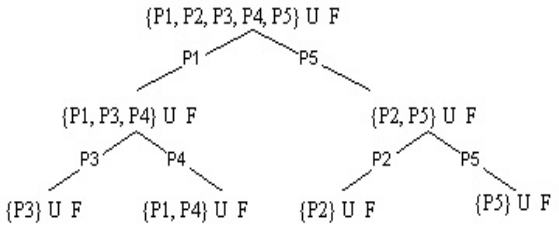

Figure 2. The splitting tree for the training set of Table 1

From the above training set, according to the process presented in [11], four neurules are finally produced (two of them are depicted in Table 4). Figure 2 illustrates the corresponding splitting tree, which represents the splits that take place during the process.

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 (NR1, Table 4). Subset {P1, P4} ∪ F produces another neurule (NR2, Table 4). Similarly, from subset {P2, P5} ∪ F two other neurules are produced (corresponding to its two leaves).

Table 4. Two of the neurules produced from the example training set

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
NR2 (-14.6) **if** blood-conc is normal (14.0), venous-conc is slight (13.5), arterial-conc is moderate (10.6), cap-conc is slight (10.4), scan-conc is normal (10.1), venous-conc is normal (9.9), blood-conc is high (9.9), venous-conc is moderate (6.5), blood-conc is moderate (6.3), blood-conc is slight (3.2), venous-conc is high (-1.0), cap-conc is moderate (-0.5), arterial-conc is slight (-0.4) **then** disease is inflammation

3. Insertion of a new example

Often, new empirical data may become available over time. This entails that the neurule-base should be updated. Availability of new data in the source set means insertion of a new example (either success or failure) into all the initial training sets that are affected. Given the modularity of a neurule-base, when a new example is available, it does not affect the whole base, but one or more sets of sibling rules. So, the basic problem is how to update a set of sibling neurules, due to availability of an extra training example/pattern, which should be taken into account alongside their initial training set.

The updating process, in order to be efficient, should encompass two features: (a) the least possible subset of corresponding sibling neurules should undergo (re)training and (b) the number of corresponding sibling neurules should remain as small as possible. The first feature guarantees that the computational cost of the update will be as low as possible. The second one assures that the efficiency of the inferences will not significantly decrease.

At the insertion of a new example in an initial training set, two cases can be distinguished: (a) There is no splitting tree associated with the training set, (b) There is a splitting tree. Case (a) is a simple one. The fact that there is no splitting tree means that the initial training set was not split, hence only one neurule was produced. To handle this case, the existing neurule is removed from the neurule-base and (re)training with the updated training set is performed. If training is successful, one new (updated) neurule is produced. If training fails, two new neurules are produced. Case (a) is handled in the same way for both, the insertion of a success and the insertion of a failure example.

Case (b) is a difficult one. The existence of a splitting tree means that there was at least one splitting of the initial training set and two or more sibling neurules were produced. There can be various approaches to handle this case, which are presented in the next subsections.

3.1. Insertion of a success example

At inserting a success example/pattern P in an initial training set, there can be three approaches to handle case (b).

(i) Corresponding sibling neurules are removed from the neurule-base, the new example is inserted into the initial training set and retraining is performed to produce the new (sibling) neurules. This approach is actually based on retraining the whole set of the sibling neurules, therefore is inefficient, especially when more than two neurules are produced from the initial training set. The reason is that it discards the information stored in the splitting tree, thus performing extra training and splitting. However, it probably produces the least number of sibling neurules. We call this approach Sb1.

(ii) Leave all corresponding sibling neurules intact and insert into the neurule-base an extra (sibling) neurule produced from a training set containing the new example and the failure examples of the initial training set. This method is computationally efficient, but definitely increases the number of neurules in the neurule-base, negatively affecting the inference process. Again, this

method does not take into account the information stored in the splitting tree. We call this approach Sb2.

(iii) The third approach exploits the information stored in the splitting tree. It focuses on the training subset containing the success examples that are closer to P. To this end, the splitting tree is traversed starting from the root and ending at a leaf or an intermediate node. Traversing is based on the closeness between the new success example P and the least closeness pairs attached to the edges of the splitting tree. We call this approach Sb3. More formally, the corresponding algorithm is as follows:

Starting from the root, traverse the splitting tree and do

1. If the current node is not a leaf, check whether the training (sub)set corresponding to the node contains an example P' whose closeness to P is less than the least closeness of the (sub)set. If there is no such example, insert P into the training (sub)set of the node and execute this step recursively for the child of the node on the branch denoted by the member of the least closeness pair which is closer to P. If there is such an example P', do

1.1. Stop traversing the splitting tree.

1.2. Remove from the neurule-base all (sibling) neurules corresponding to the leaves descending from this node.

1.3. Insert P into the corresponding training (sub)set and split it in two subsets with as least closeness pair (P, P').

1.4. Perform (re)training based on the two training subsets, produce the corresponding neurules (reusing parts of the initial splitting tree to avoid unnecessary training or splitting), insert the produced neurules into the neurule-base and update the splitting tree.

2. If the current node is a leaf, remove the corresponding neurule, insert P into its training set and perform (re)training.

2.1 If training fails, split the training set, produce the two neurules, insert them into the neurule-base and update the splitting tree.

2.2 If training is successful, insert the neurule produced from the leaf's new training set into the neurule-base and update the splitting tree.

Approach Sb3 results in changes to the least possible subset of the corresponding sibling neurules set and therefore requires less (re)training effort than approach Sb1. Furthermore, Sb3 produces less sibling neurules than Sb2. Sb2 inserts a neurule into the neurule-base when a new success example is inserted into the initial training set. On the contrary, if the insertion of a success example is handled according to approach Sb3, the number of the produced neurules may not increase. More specifically, if traversing reaches a leaf and training of the subsequent subset is successful, the number of produced neurules remains the same (steps 2.1-2.2).

3.2 Insertion of a failure example

The insertion of a failure example into an initial training set affects all the sibling neurules produced from this set. Therefore, compared to the insertion of a success example, it requires more (re)training effort. At inserting a failure example/pattren in an initial training set, there can be two approaches to handle case (b).

(i) Corresponding sibling neurules are removed from the neurule-base, the new example is inserted into the initial training set and retraining is performed to produce the new neurules. Again, this approach is actually based on retraining the whole set of the sibling neurules, therefore is inefficient. It discards the information stored in the splitting tree, thus performing extra training and splitting. We call this approach Fb1. This approach is similar to Sb1.

(ii) The second approach exploits the information contained in the splitting tree. It removes corresponding sibling neurules from the neurule-base and inserts the example into the training subsets corresponding to the leaves of the splitting tree. It performs training of the subsets and the produced neurules are inserted into the neurule-base. We call this approach Fb2. It is obvious that approach Fb2 usually requires more training effort than its corresponding approach Sb3, handling the insertion of success examples. However, it requires less training effort than approach Fb1.

4. Examples

Consider the initial training set presented in section 2.2. Suppose that the success example P6 = [-1, 1, 1, 1, -1, 1, -1, -1, 1, -1, -1, -1, -1, 1] is to be inserted into the initial training set. Given that more than one sibling neurule were produced from the initial training set, it is a (b) case.

Following approach Sb3, traversing ends at the leaf corresponding to subset {P3} ∪ F (see Figs 2 and 3). Training based on the new training subset {P3, P6} ∪ F is successful and the process stops. The splitting tree takes the form in Fig. 4. So, the total number of neurules in the neurule-base after the insertion of P6 remains four (corresponding to the leaves of the splitting tree in Fig. 4).

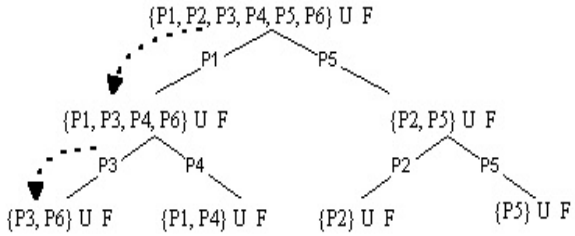

Figure 3. Traversal of the splitting tree for the insertion of example P6

Notice that, approach Sb3 finally ends at subset {P3} ∪ F, which includes the success example closest to P6. So, only one of the corresponding sibling neurules requires (re)training. The rest of them remain intact. Approach Sb1 produces the same number of neurules, requiring though unnecessary training and splitting. Approach Sb2 inserts the neurule produced from subset {P6} ∪ F into the neurule-base. So, Sb2 produces more neurules than Sb3.

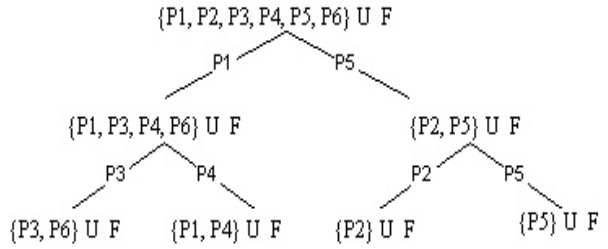

Figure 4. The splitting tree after insertion of example P6

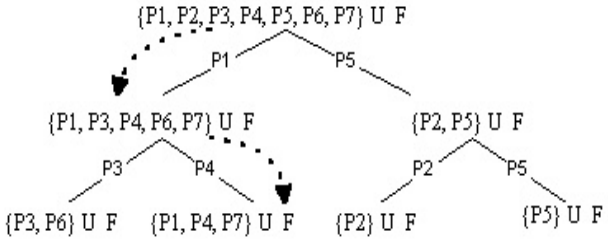

Figure 5. Traversal of the splitting tree for the insertion of example P7

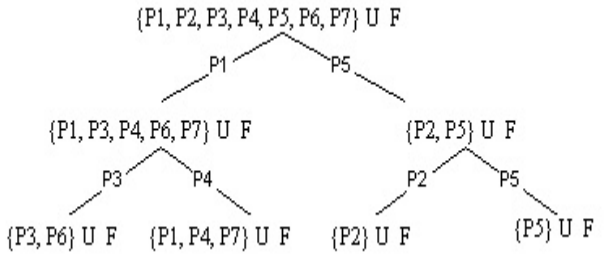

Figure 6. The splitting tree after insertion of example P7

Suppose that (after the insertion of P6) the success example P7 = [-1, 1, 1, -1, -1, 1, -1, 1, 1, -1, -1, -1, -1, 1] is to be inserted into the training set {P1, P2, P3, P4, P5, P6} ∪ F. Following approach Sb3, traversing ends at the leaf related to subset {P1, P4} ∪ F (see Figs 4 and 5). Training based on the new training subset {P1, P4, P7} ∪ F is successful and the process stops. The splitting tree takes the form shown in Fig. 6. The total number of neurules in the neurule-base, after inserting P7, remains again four (corresponding to the leaves of the splitting tree in Fig. 6). Once again only one of the sibling neurules requires (re)training. The rest remain intact.

Approach Sb1 produces the same neurules, requiring though unnecessary training and splitting. Approach Sb2 inserts the neurule produced from subset {P7} ∪ F into the neurule-base. So, approach Sb2, in order to update the neurule-base due to insertion of both P6 and P7, increases the number of neurules from four to six, whereas Sb3 results in no change to that number; they remain four.

5. Experimental Results

In this section we present experimental results comparing the different approaches for handling case (b) of an example insertion. For this purpose, we use training sets produced from datasets taken from the UCI Repository of Machine Learning and Domain Theories [2]. More specifically, we used the *lenses* dataset containing 24 examples/patterns of 9 component values and the *tic-tac-toe* dataset containing 958 examples/patterns of 27 component values. Additionally, we used three datasets of ours produced from a medical domain [13]: the *inflammation* dataset, containing 96 examples/patterns of 9 component values, the *arthritis* dataset, containing 144 examples/patterns of also 9 component values, and the *primary_malignant* dataset, containing 120 examples/patterns of 10 component values.

Table 3. Experimental results

Dataset	*Comp Set*	*Incomp Set*	*Sb1*	*Sb2*	*Sb3*
Lenses	4	4 (3)	4	7	4
Tic-tac-toe	30	28 (4)	30	32	30
Inflammation	2	2 (1)	2	3	2
Arthritis	3	3 (1)	3	4	3
Primary-malignant	2	2 (1)	2	3	2

Comparison of the approaches is based on the number of the produced neurules. Initial training sets, called *complete sets*, were formed from the datasets and corresponding neurule-bases were produced from them. Afterwards, a few success examples were removed from the complete sets. The resulting training sets are called *incomplete sets*. Then, neurule-bases corresponding to the incomplete sets were produced. Afterwards, the removed examples were inserted one by one into the incomplete sets and the neurule-bases were updated using the

approaches related to case (b). So, at the end, the incomplete sets became the same as the complete sets.

Table 3 presents the experimental results. For each dataset, the number of neurules of the neurule-base produced from the complete and the incomplete sets are presented ('Complete Set' and 'Incomplete Set' columns). Also, the final number of neurules, after the insertion of the removed examples, using the approaches Sb1, Sb2 and Sb3, are presented. The number of removed examples is shown within parentheses in the 'Incomplete Set' column. As it is clear, the use of approach Sb3 results in the same number of neurules as if recreation of the neurule-base has been made (approach Sb1). On the contrary, approach Sb2 results in increasing the number of the neurules, directly depending on the number of inserted examples.

6. Conclusions

In this paper, we present methods for efficiently updating a hybrid rule base (target knowledge) due to changes to its empirical source knowledge. The hybrid rule base consists of neurules, a type of hybrid rules integrating symbolic rules with neurocomputing. Its empirical source knowledge consists of training examples/patterns. Updates refer to insertion of new training examples and are handled quite efficiently. That is, only the necessary part of the target knowledge has to be retrained and the number of the neurules remains as small as possible. This is achieved by exploiting the notion of closeness, used to handle inseparability in the construction of the target knowledge, and the introduction of a structure called the splitting tree, which stores information regarding the construction process of the target knowledge.

In this paper, we assume that the inserted example is not conflicting with an existing one. If this is not the case, the existing example should be removed and the new one should be inserted instead. So, the problem of removing an example/pattern should be tackled. This is a more difficult problem and a challenge for further research. Furthermore, in this paper, we are dealing with consecutive insertion of examples. Simultaneous insertion of more than one example seems to be another interesting problem for further research.

References

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

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

[3] d'Avila Garcez A. S., Broda K. B. and Gabbay D. M., *Neural-Symbolic Learning Systems*, Springer-Verlag, Heidelberg, 2002.

[4] Fu, L-M, Fu, L-C, "Mapping rule-based systems into neural architecture", *Knowledge-Based Systems* 3, 1990, 48-56.

[5] Fu, L-M, "Incremental Knowledge Acquisition in Supervised Learning Networks", *IEEE Transactions on Systems, Man and Cybernetics*-Part A: Systems and Humans 26, 1996, 801-809.

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

[7] Ghalwash, A. Z., "A Recency Inference Engine for Connectionist Knowledge Bases", *Applied Intelligence* 9, 1998, 201-215.

[8] Giraud-Carrier, C., "A Note on the Utility of Incremental Learning", *AI Communications* 13, 2000, 215-224.

[9] Hall L. O., K. Sanou and S. Romaniuk, "An Encoding of Production Rules in a Connectionist Network", *Journal of Intelligent and Fuzzy Systems*, 4 (1), 1996, 1-18.

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

[11] Hatzilygeroudis, I., Prentzas, J., "Constructing Modular Hybrid Rule Bases For Expert Systems", *International Journal on AI Tools* 10, 2001, 87-105.

[12] Hatzilygeroudis, I., Prentzas, J., "An Efficient Hybrid Rule-Based Inference Engine with Explanation Capability", *Proceedings of the 14th International FLAIRS Conference*, AAAI Press, 2001, 227-231.

[13] Hatzilygeroudis I., Vassilakos P. J. and Tsakalidis A., "XBONE: A Hybrid Expert System for Supporting Diagnosis of Bone Diseases", C. Pappas, N. Maglaveras and J-R Scherrer (Eds), Medical Informatics Europe'97 (MIE'97), IOS Press, 1997, 295-299.

[14] Maloof, M. A., Michalski, R. S., "Selecting Examples for Partial Memory Learning", *Machine Learning* 41, 2000, 27-52.

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

[16] Neagu C-D and Palade V., "Modular Neuro-Fuzzy Networks Used in Explicit and Implicit Knowledge Integration", *Proceedings of the 15th International FLAIRS Conference*, AAAI Press, 2002, 277-281.

[17] Quah T-S, Tan C-L, Krishnamurthy R. S. and Srinivasan B., "Towards integrating rule-based expert systems and neural networks", *Decision Support Systems*, 17 (2), 1996, 99-118

[18] Towell, G., Shavlik, J., "Knowledge-Based Artificial Neural Networks", *Artificial Intelligence* 70, 1994, 119-165.

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