Multi-inference with Multi-neurules

Ioannis Hatzilygeroudis and Jim Prentzas

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)
{ihatz, prentzas}@ceid.upatras.gr

Abstract. Neurules are a type of hybrid rules combining a symbolic and a connectionist representation. There are two disadvantages of neurules. The first is that the created neurule bases usually contain multiple representations of the same piece of knowledge. Also, the inference mechanism is rather connectionism oriented than symbolism oriented, thus reducing naturalness. To remedy these deficiencies, we introduce an extension to neurules, called multi-neurules, and an alternative inference process, which is rather symbolism oriented. Experimental results comparing the two inference processes are also presented.

1 Introduction

There have been efforts at combining the expert systems approach and the neural networks (connectionism) one into hybrid systems, in order to exploit their benefits [1]. In some of them, called embedded systems, a neural network is used in the inference engine of an expert system. For example, in NEULA [2] a neural network selects the next rule to fire. Also, LAM [1] uses two neural networks as partial problem solvers. However, the inference process in those systems, although gains efficiency, lacks the naturalness of the symbolic component. This is so, because pre-eminence is given to the connectionist framework.

On the other hand, connectionist expert systems are integrated systems that represent relationships between concepts, considered as nodes in a neural network. Weights are set in a way that makes the network infer correctly. The system in [3] is a popular such system, whose inference engine is called MACIE. Two characteristics of MACIE are: its ability to reason from partial data and its ability to provide explanations in the form of if-then rules. However, its inference process lacks naturalness. Again, this is due to the pure connectionist inference approach.

In a previous work [4], we introduced *neurules*, a hybrid rule-based representation scheme integrating symbolic rules with neurocomputing, which gives pre-eminence to the symbolic component. Thus, neurules give a more natural and efficient way of representing knowledge and making inferences. However, there are two disadvantages of neurules, from the symbolic point of view. Neurules are constructed either from symbolic rules or from learning data in the form of training examples. In case of non-separable sets of training examples, more than one neurule with the same conditions and the same conclusion, but different significance factors exist in a neurule base. This creates neurule bases with multiple representations of the same piece of knowledge [5]. The second disadvantage is that the associated inference mechanism [6] is rather connectionism oriented, thus reducing naturalness.

To remedy the first deficiency, we introduce here an extension to neurules called *multi-neurules*. For the second, we introduce an alternative hybrid inference process, which is rather symbolism oriented. Experimental results comparing the two inference processes are presented.

The structure of the paper is as follows. Section 2 presents neurules and Section 3 mainly the inference process introduced here. In Section 4, an example knowledge base and an example inference are presented. Section 5 presents some experimental results and finally Section 6 concludes.

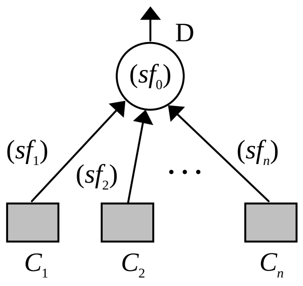
2 Neurules

2.1 Simple Neurules

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.

(sf_0) **if** C1 (sf_1),
C$_2$ (sf_2),
…
C$_n$ (sf_n)
then D

(a)



(b)

Fig. 1. (a) Form of a neurule (b) corresponding adaline 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:

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

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

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

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

2.2 Training Neurules

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). The learning algorithm employed is the standard least mean square (LMS) algorithm (see e.g. [3]).

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 divided into subsets in a way that each subset contains *success examples* (i.e. with d=1) 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 are distributed between the two subsets based on their closeness to their 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. 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, which are called *sibling neurules* (for a more detailed and formal description see [5]).

2.3 Multi-neurules

The existence of sibling neurules creates neurule bases with multiple representations of the same piece of knowledge, which is their main disadvantage. To remedy this deficiency, we introduce multi-neurules.

$$\begin{aligned} (CF_0^m)\ \textbf{if}\ & C_1\,(CF_1^m), \\ & C_2\,(CF_2^m), \\ & \ldots \\ & C_n\,(CF_n^m) \\ \textbf{then}\ & \mathrm{D} \end{aligned}$$

(a)

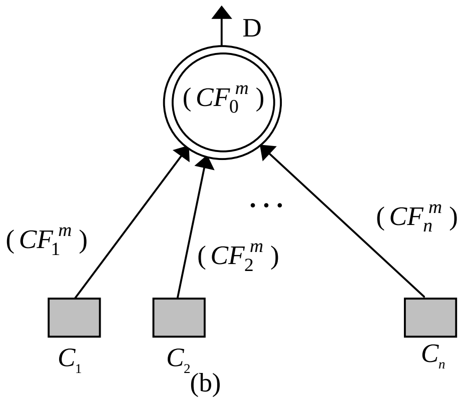


Fig. 2. (a) Form of a multi-neurule (b) corresponding multi-adaline unit

A *multi-neurule* of size *m* has the form presented in Fig. 2a and is considered as a *multi-adaline* unit (Fig. 2b), also introduced here. Each CF_i^m is called a condition *sf-tuple* that consists of *m* significance factors:

$$CF_i^m \equiv < sf_{i1}, sf_{i2}, ..., sf_{im} > .$$

A multi-adaline unit of size *m* is a merger of *m* simple adaline units. Correspondingly, a multi-neurule of *size m* is the merger of *m* simple neurules. So, in a multi-unit of size *m* we distinguish *m* different sets of weights, each corresponding to the weights of a constituent unit. Similarly, a multi-neurule of *size m* includes *m* different sets of significance factors, called rule *sf-sets*, each corresponding to the significance factors of a constituent neurule. Thus, the rule sf-set RF_i consists of the *i*th significance factors of the sf-tuples:

$$RF_i = (sf_{1i}, sf_{2i}, \ldots, sf_{ni}),\ \text{for } i = 1, m$$

Each RF_i is used to compute the activation $\mathbf{a}_i$ of the corresponding adaline unit.

The output of a multi-unit is determined by the set that produces the maximum activation value. Hence, a multi-unit is activated as soon as any of its constituent units gets activated (i.e. any $\mathbf{a}_i \geq 0$). The output D of a multi-adaline unit is calculated via the formulas:

$$D = f(\mathbf{a}),\ \mathbf{a} = \bigvee_{i=1}^{m} \mathbf{a}_i\,,\ \mathbf{a}_i = sf_{0i} + \sum_{j=1}^{n} sf_{ij} C_j \tag{2}$$

The activation function is the same as in a simple unit.

NR5: (-2.2) **if** Treatment is Placibin (-1.8),
Treatment is Biramibio (1.0)
then Treatment is Posiboost

NR6: (-2.2) **if** Treatment is Biramibio (-2.6),
Treatment is Placibin (1.8)
then Treatment is Posiboost

$\Downarrow$

NR5: (-2.2, -2.2) **if** Treatment is Placibin (-1.8, 1.8)
Treatment is Biramibio (1.0, -2.6)
then Treatment is Posiboost

Fig. 3. Merging sibling neurules into a multi-neurule

In practice, a multi-neurule is produced by simply merging all sibling neurules with the same conclusion. For example, neurule NR5, used in the example knowledge base in Section 4.1, is a multi-neurule produced from merging two sibling neurules of the old knowledge base (NR5, NR6), as shown in Fig.3. Notice that, because the conditions in each simple neurule are are sorted, so that $|sf_1| \geq |sf_2| \geq \ldots \geq |sf_n|$, for efficiency reasons, this information is also attached to multi-neurules. So, NR5 has two sf-sets, RF_1 = (-1.8, 1.0) and RF_2 = (-2.6, 1.8).

2.4 Syntax and Semantics

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

<rule>::= (<bias-factors>) **if** <conditions> **then** <conclusions>

<bias-factors>::= <bias-factor> {**,** <bias-factor>}

<conditions>::= <condition> {**,** <condition>}

<conclusions>::= <conclusion> {**,** <conclusion>}

<condition>::= <variable> <predicate> <value> (<significance-factors>)

< significance-factors >::= < significance-factor> {**,** < significance-factor>}

<conclusion>::= <variable> <predicate> <value> .

In the above definition, <variable> denotes a *variable* as in a variable declaration. <predicate> denotes a predicate, which is one of {is, isnot, <, >}. <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. The significance factor of a condition represents the significance (weight) of the condition in drawing the conclusion. A significance factor with a sign opposite to that of the bias factor of its neurule positively contributes in drawing the conclusion, otherwise negatively.

3 The Hybrid Inference Processes

The inference engine associated with neurules implements the way neurules co-operate to reach conclusions. It supports two alternative hybrid inference processes. The one gives pre-eminence to neurocomputing, and is called *connectionism-oriented* inference process, whereas the other to symbolic reasoning, and is called *symbolism-oriented inference process*. In the symbolism-oriented process, a type of a classical rule-based reasoning is employed, but evaluation of a rule is based on neurocomputing measures. In the connectionism-oriented process, the choice of the next rule to be considered is based on a neurocomputing measure, so the process jumps from rule to rule, but the rest is symbolic. In this section, we mainly present the symbolism oriented process.

3.1 Neurules Evaluation

In the following, WM denotes the working memory and NRB the neurule base.

Generally, the output of a simple neurule is computed according to Eq. (1). However, it is possible to deduce the output of a neurule without knowing the values of all of its conditions. To achieve this, we define for each simple neurule the *known sum* and the *remaining sum* as follows:

$$kn-sum = sf_0 + \sum_{C_i \in E} sf_i C_i \tag{3}$$

$$rem-sum = \sum_{C_i \in U} | sf_i | \tag{4}$$

where E is the set of evaluated conditions, U the set of unevaluated conditions and C_i is the value of condition cond$_i$. A condition is evaluated, if its value ('true' or 'false') is by some way known. So, *known-sum* is the weighted sum of the values of the already known (i.e. evaluated) conditions (inputs) of the corresponding neurule and *rem-sum* represents the largest possible weighted sum of the remaining (i.e. unevaluated) conditions of the neurule. If |*kn-sum*| > *rem-sum* for a certain neurule, then evaluation of its conditions can stop, because its output can be deduced regardless of the values of the unevaluated conditions. In this case, its output is guaranteed to be '-1' if *kn-sum* < 0, or '1', if *kn-sum* > 0.

In the case of a multi-neurule of size *m*, we define *m* different *kn-sum*s and *rem-sum*s, one for each RF_i:

$$kn-sum_i = sf_{0i} + \sum_{C_j \in E} sf_{ji} C_j \text{ , } i\text{=1, } m \tag{5}$$

$$rem-sum_i = \sum_{C_j \in U} | sf_{ji} | \text{, } i\text{=1, } m \text{ .} \tag{6}$$

It is convenient, for the connectionism-oriented process, to define the *firing potential (fp)* of a neurule as follows:

$$fp = \frac{|\,kn - sum\,|}{rem - sum}\ . \tag{7}$$

The firing potential of a neurule is an estimate of its intention that its output will become ‘±1’. Whenever *fp* > 1, the values of the evaluated conditions can determine the value of its output, regardless of the values of the unevaluated conditions. The rule then evaluates to ‘1’ (true), if *kn-sum* > 0 or to ‘-1’ (false), if *kn-sum* < 0. In the first case, we say that the neurule is *fired*, whereas in the second that it is *blocked*. Notice, that the firing potential has meaning only if *rem-sum* ≠ 0. If *rem-sum* = 0, all the conditions have been evaluated and its output is evaluated based on *kn-sum*. For a multi-neurule, we define as many *fp*s as the size of the multi-neurule.

3.2 Symbolism-Oriented Process

The symbolism-oriented inference process is based on a backward chaining strategy. There are two stacks used, a *goal stack* (*GS*), where the *current goal* (*CG*) (condition) to be evaluated is always on its top, and a *neurule stack (NS)*, where the current neurule under evaluation is always on its top. The conflict resolution strategy, due to backward chaining and the neurules, is based on textual order. A neurule succeeds if it *evaluates* to ‘true’, that is its output is computed to be ‘1’ after evaluation of its conditions. Inference stops either when a neurule with a goal variable is fired (success) or there is no further action (failure). *WM* denotes the working memory.

More formally, the process is as follows:

```
1. Put the initial goal(s) on GS.
2. While there are goals on GS do
   2.1 Consider the first goal on GS as the current goal (CG) and find the
       neurules having it as their conclusion. If there are no such neurules, stop
       (failure). Otherwise, put them on RS.
   2.2 For each neurule $NR_i$ on NS (current rule: $CR = NR_i$) do
       2.2.1 (simple neurule case) While CR is not fired or blocked, for each
             condition $C_i$ of CR (current condition: $CC = C_i$) do
             2.2.1.1 If CC is already evaluated, update the kn-sum and the rem-
                     sum of $NR_x$. Otherwise, if it contains an input variable, ask
                     the user for its value (user data), evaluate CC, put it in WM
                     and update the kn-sum and the rem-sum of CR, otherwise
                     (intermediate or output variable) put CC on the top of GS
                     and execute step 2.1 recursively until CC is evaluated. After
                     its evaluation update the kn-sum and the rem-sum of CR.
             2.2.1.2 If (|kn-sum| > rem-sum and kn-sum > 0), mark CR as ‘fired’,
                     mark its conclusion as ‘true’, put the conclusion in WM and
                     remove the current goal from GS (multi-valued variable) or
                     remove from GS all goals containing the variable (single-
                     valued variable). If (|kn-sum| > rem-sum and kn-sum < 0),
                     mark CR as ‘blocked’.
```

```
        2.2.2 (multi-neurule case) While CR is not ‘fired’ or ‘blocked’, for each $RF_i$ (current sf-set: $CRF = RF_i$) of CR do
            2.2.2.1 While CRF is not ‘fired’ or ‘blocked’, for each for each condition $C_i$ of CRF ($CC = C_i$) do
                2.2.2.1.1 The same as 2.2.1.1 (with $kn\text{-}sum_i$ and $rem\text{-}sum_i$ instead of kn-sum and rem-sum, respectively).
                2.2.2.1.2 If ($|kn\text{-}sum_i| > rem\text{-}sum_i$ and $kn\text{-}sum_i > 0$), mark CR as ‘fired’, mark its conclusion as ‘true’, put the conclusion in WM and remove the current goal from GS (multi-valued variable) or remove from GS all goals containing the variable (single-valued variable). If ($|kn\text{-}sum_i| > rem\text{-}sum_i$ and $kn\text{-}sum_i < 0$), mark CRF as ‘blocked’. If it is the last rule sf-set, mark CR as ‘blocked’.
    2.3 If all neurules on RS are blocked, mark their conclusions as ‘false’, put the conclusions in WM and remove the current goal from GS.
3. If there are no conclusions in WM containing output variables, stop (failure). Otherwise, display the conclusions in WM marked as ‘TRUE’ (output data) and stop (success).
```

3.3 Connectionism-Oriented Process

Initially, the values of the variables (conditions) may be not known to the system. The *kn-sum* for every simple neurule is then set to its bias factor, whereas its *rem-sum* is set to the sum of the absolute values of all its significance factors. For a multi-neurule, this is done for each RF_i (i=1, m). If the value of a variable becomes known, it influences the values of the conditions containing it and hence the known sums, the remaining sums and the firing potentials of the unevaluated neurules containing them, which are called *affected neurules*. As soon as an intermediate neurule becomes evaluated, the known sums, the remaining sums and the firing potentials of all the affected neurules are also updated. Updating a multi-neurule consists in updating each of its *fp*s (*kn-sum*s and *rem-sum*s). Obviously, a firing potential is updated only if the corresponding remaining sum is not equal to zero.

Unevaluated neurules that are updated, due to a new variable value, constitute the *participating* neurules. The inference mechanism tries then to focus on participating neurules whose firing potential is close to exceeding ‘1’. More specifically, it selects the one with the maximum firing potential, because it is the most likely, has a greater intention, to fire. In the case of a multi-neurule, its maximum *fp* represents the neurule. The system tries to evaluate the first unevaluated condition, which is the one with the maximum absolute significance factor (recall that the conditions of a neurule are sorted). After evaluation of the condition, *kn-sum, rem-sum* and *fp* of the neurule are computed. If *rem-sum* = 0 or *fp* > 1, it evaluates and its conclusion is put in the WM. If the system reaches a final conclusion, it stops.

A more formal description of this inference algorithm, for a NRB containing only simple neurules, can be found in [6, 11].

4 Examples

4.1 Example Knowledge Base

We use as an example to illustrate the functionalities of our system the one presented in [3]. It contains training data dealing with acute theoretical diseases of the sarcophagus. There are six symptoms (Swollen feet, Red ears, Hair loss, Dizziness, Sensitive aretha, Placibin allergy), two diseases (Supercilliosis, Namastosis) whose diagnoses are based on the symptoms and three possible treatments (Placibin, Biramibio, Posiboost). Also, dependency information is provided. We used the dependency information to construct the initial neurules and the training data provided to train them. The produced knowledge base, which contains five neurules (NR1-NR5), is illustrated in Table 1. An equivalent knowledge base forming a multilevel network is presented in [3]. It is quite clear how more natural is our knowledge base than that in [3].

Table 1.

NR1: (-0.4) **if** SwollenFeet is true (3.6), HairLoss is true (3.6), RedEars is true (-0.8) **then** Disease is Supercilliosis NR2: (1.4) **if** Dizziness is true (4.6), SensitiveAretha is true (1.8), HairLoss is true (1.8) **then** Disease is Namastosis NR3: (-2.2) **if** PlacibinAllergy is true (-5.4), Disease is Supercilliosis (4.6) Disease is Namastosis (1.8), **then** Treatment is Placibin	NR4: (-4.0) **if** HairLoss is true (-3.6), Disease is Namastosis (3.6), Disease is Supercilliosis (2.8) **then** Treatment is Biramibio NR5: (-2.2, -2.2) **if** Treatment is Placibin (-1.8, 1.8), Treatment is Biramibio (1.0, - 2.6) **then** Treatment is Posiboost

4.2 Example Inference

We suppose that the initial data in the WM is: ‘HairLoss is true’ (TRUE). Since ‘Treatment’ is the only goal variable, its possible conclusions are initially put on GS. The inference tracing, according to the symbolism oriented process, is briefly presented in Table 2 (follow the left column first in both pages, then the right).

Table 2.

Initial situation and Steps 1, 2-2.1
WM: {‘HairLoss is true’ (TRUE)}
GS: [‘Treatment is Placibin’, ‘Treatment is
Biramibio’, ‘Treatment is Posi-boost’]
CG: ‘Treatment is Placibin’
RS: [NR3]
Fired neurules:
Blocked neurules:

Step 2.2-2.2.1
CR: NR3
CC: ‘PlacibinAllergy is true’ (NR3)

Step 2.2.1.1
User data: ‘PlacibinAllergy is true’ (FALSE)
WM: {‘PlacibinAllergy is true’ (FALSE), ‘HairLoss is true’ (TRUE)}
Updated sums: kn-sum=3.2, rem-sum=6.4 (NR3)

Step 2.2.1.2
|kn-sum| < rem-sum (NR3)

Step 2.2.1
CC: ‘Disease is Supercilliosis’ (NR3)

Step 2.2.1.1
GS: [‘Disease is Supercilliosis’, ‘Treatment is Placibin’, …]
(start of recursion)

Step 2.1-2.2
CG: ‘Disease is Supercilliosis’
RS: [NR1, NR3]
CR: NR1

Step 2.2.1
CC: ‘SwollenFeet is true’ (NR1)

Step 2.2.1.1
User data: ‘SwollenFeet is true’ (FALSE)
WM: {‘SwollenFeet is true’ (FALSE), ‘PlacibinAllergy is true’ (FALSE), ‘HairLoss is true’ (TRUE)}
Updated sums: kn-sum=-4.0, rem-sum=4.4 (NR1)

Step 2.2.1.2
|kn-sum| < rem-sum (NR1)

Step 2.2.1
CC: ‘HairLoss is true’ (NR1)

Step 2.2.1.1
User data: ‘HairLossis true’ (TRUE)
WM: {‘HairLoss is true’ (TRUE),
GS: [‘Treatment is Placibin’, ‘Treatment is Biramibio’, ‘Treatment is Posiboost’]
(return from recursion)

Step 2.2.1.1
Updated sums: kn-sum=7.8, rem-sum=1.8 (NR3)

Step 2.2.1.2
|kn-sum| > rem-sum and kn-sum > 0 (NR3)
Fired neurules: NR3, NR1
WM: {‘Treatment is Placibin’ (TRUE), ‘Disease is Supercilliosis’ (TRUE), ‘RedEars is true’ (FALSE), …}
GS: [‘Treatment is Biramibio’, ‘Treatment is Posiboost’]

Step 2.1-2.2
CG: ‘Treatment is Biramibio’
RS: [NR4]
CR: NR4

Step 2.2.1
CC: ‘HairLoss is true’ (NR4)

Step 2.2.1.1
Updated sums: kn-sum=-7.6, rem-sum=6.4 (NR4)

Step 2.2.1.2
|kn-sum| > rem-sum and kn-sum < 0 (NR4)
Blocked neurules: NR4

Step 2.3
WM: {‘Treatment is Biramibio’ (FALSE), ‘Treatment is Placibin’ (TRUE), ‘Disease is Supercilliosis’ (TRUE), …}
GS: [‘Treatment is Posiboost’]

Step 2-2.1-2.2
CG: ‘Treatment is Posiboost’
RS: [NR5]
CR: NR5

Step 2.2.2-2.2.2.1
CRF: RF_1-NR5
CC: ‘Treatment is Placibin’ (RF_1-NR5)
Given that both conditions are already evaluated …
Step 2.2.2.1.1-2.2.2.1.2
… finally:
Updated sums: kn-sum_1= -5.0, rem-sum_1= 0
|kn-sum| < 0
Blocked RFs: RF_1-NR5

'SwollenFeet is true' (FALSE), …}
Updated sums: kn-sum=-0.4, rem-sum=0.8 (NR1)

Step 2.2.1.2
|kn-sum| < rem-sum (NR1)

Step 2.2.1
CC: 'RedEars is true' (NR1)

Step 2.2.1.1
User data: 'RedEars is true' (FALSE)
WM:{'RedEars is true' (FALSE), 'HairLossis true' (TRUE), …}
Updated sums: kn-sum=0.4, rem-sum=0 (NR1)

Step 2.2.1.2
|kn-sum| > rem-sum and kn-sum > 0 (NR1)
Fired neurules: NR1
WM: {'Disease is Supercilliosis' (TRUE), 'RedEars is true' (FALSE), 'HairLossis true' (TRUE), …}

Step 2.2.2-2.2.2.1
CRF: RF_2-NR5
CC: 'Treatment is Biramibio'
Given that both conditions are already evaluated …
Step 2.2.2.1.1-2.2.2.1.2
… finally:
Updated sums: kn-sum_2= 2.2, rem-sum_2= 0
|kn-sum| > 0
Fired neurules: NR5, NR3, NR1
WM: {'Treatment is Posiboost'(TRUE), 'Treatment is Biramibio' (FALSE), 'Treatment is Placibin' (TRUE), 'Disease is Supercilliosis' (TRUE), …}
GS: [ ]

Step 3
Output data: 'Treatment is Placibin', 'Treatment is Posiboost'

5 Experimental Results

In this section, we present experimental results comparing the two inference processes, the symbolism oriented and the connectionism oriented (see Table 3).

Table 3.

KB	**Connectionism oriented process**		**Symbolism oriented process**	
	ASKED	*EVALS*	*ASKED*	*EVALS*
ANIMALS (20 inferences)	162 (8.1),	364 (18.2)	142 (7.1)	251 (12.5)
LENSES (24 inferences)	79 (3.3)	602 (25.1)	85 (3.5)	258 (10.8)
ZOO (101 inferences)	1052 (10.4)	8906 (88.2)	1013 (10)	1963 (19.4)
MEDICAL (134 inferences)	670 (5)	25031 (186.8)	670 (5)	11828 (88.3)

We used four knowledge bases: ANIMALS (from [7]), LENSES and ZOO (from [8]), MEDICAL (from [9]), of different size and content. Numbers in the *ASKED* column (outside the parentheses) represent the number of inputs (variables) whose values were required/asked to reach a conclusion. The numbers within the parentheses represent the mean number of required input values (per inference). The number of inferences attempted for each KB is depicted within the parentheses in the KB column. Furthermore, the numbers in the *EVALS* columns represent the number of conditions/inputs visited for evaluation (the mean value within the parentheses). It is clear, that the symbolism oriented process did equally well or better than the connectionism oriented in all cases, except one (the shaded one). This is clearer for the condition evaluations, especially as the number of inferences increases. Given that the connectionism oriented process is better than the inference processes introduced in [3] and [10], as concluded in [6, 11], the symbolism oriented process is even better.

6 Conclusion

In this paper, we present an extension to neurules, called multi-neurules. Multi-neurules, although they make the inference cycle more complicated, increase the conciseness of the rule base. Simple neurules have the disadvantage that they may produce multiple representations (sibling neurules) of the same piece of knowledge. Multi-neurules merge those representations into one. So, although the overall naturalness seems to increase, interpretation of the significance factors becomes a tedious task, especially in cases that a large number of sibling rules participate.

We also present a new inference process, which gives pre-eminence to the symbolic inference than the connectionist one. Thus, it offers more natural inferences. This new process is proved to be more efficient than the connectionism oriented one.

References

1. Medsker L.R.: Hybrid Neural Networks and Expert Systems. Kluwer Academic Publishers, Boston (1994)
2. Tirri H.: Replacing the Pattern Matcher of an Expert System with a Neural Network. In: , Goonatilake S., Sukdev K. (eds): Intelligent Hybrid Systems. John Wiley & Sons (1995).
3. Gallant, S.I.: Neural Network Learning and Expert Systems. MIT Press (1993)
4. Hatzilygeroudis, I., Prentzas, J.: Neurules: Improving the Performance of Symbolic Rules. International Journal on AI Tools (IJAIT) **9(1**), (2000) 113-130
5. Hatzilygeroudis, I., Prentzas, J.: Constructing Modular Hybrid Knowledge Bases for Expert Systems. International Journal on AI Tools (IJAIT) **10(1-2)** (2001) 87-105
6. Hatzilygeroudis, I., Prentzas, J.: An Efficient Hybrid Rule Based Inference Engine with Explanation Capability. Proceedings of the 14th International FLAIRS Conference, Key West, FL. AAAI Press (2001) 227-231
7. Fu L.M.: Neural Networks in Computer Intelligence. McGraw-Hill (1994)
8. DataSet. ftp://ftp.ics.uci.edu/pub/machine-learning-databases/
9. Hatzilygeroudis, I., Vassilakos, P. J., Tsakalidis, A.: XBONE: A Hybrid Expert System for Supporting Diagnosis of Bone Diseases. In: Pappas, C., Maglaveras, N., Scherrer J.-R. (eds): Medical Informatics Europe'97 (MIE'97). IOS Press (1997) 295-299.
10. Ghalwash, A.Z.: A Recency Inference Engine for Connectionist Knowledge Bases. Applied Intelligence **9** (1998) 201-215
11. Hatzilygeroudis I., Prentzas, J.: HYMES: A HYbrid Modular Expert System with Efficient Inference and Explanation. Proceedings of the 8th Panhellenic Conference on Informatics, Nicosia, Cyprus, Vol.1 (2001) 422-431