Published in the International Journal of Artificial Intelligence Tools (IJAIT),
Vol. 10, No. 1-2 (2001), 87-105.

CONSTRUCTING MODULAR HYBRID RULE BASES FOR EXPERT SYSTEMS

I. HATZILYGEROUDIS, J. PRENTZAS
University of Patras, School of Engineering
Dept of Computer Engin. & Informatics, 26500 Patras, Hellas (Greece)
Email: ihatz/prentzas@ceid.upatras.gr
&
Computer Technology Institute, P.O. Box 1122, 26110 Patras, Hellas (Greece)
Email: ihatz@cti.gr

ABSTRACT

Neurules are a kind of hybrid rules integrating neurocomputing and production rules. Each neurule is represented as an adaline unit. Thus, the corresponding neurule base consists of a number of autonomous adaline units (neurules). Due to this fact, a modular and natural knowledge base is constructed, in contrast to existing connectionist knowledge bases. In this paper, we present a method for generating neurules from empirical data. To overcome the difficulty of the adaline unit to classify non-separable training examples, the notion of 'closeness' between training examples is introduced. In case of a training failure, two subsets of 'close' examples are produced from the initial training set and a copy of the neurule for each subset is trained. Failure of training any copy, leads to production of further subsets as far as success is achieved.

Keywords: hybrid knowledge representation, symbolic representation, connectionist representation, production rules, neural networks, modularity, naturalness.

1. Introduction

Recently, there has been extensive research activity at combining (or integrating) the symbolic and the connectionist approaches[1, 2, 3]. There are a number of efforts at combining symbolic rules and neural networks for knowledge representation[4, 5, 6, 7]. What they do is a kind of mapping from symbolic rules to a neural network. Also, connectionist expert systems[8, 9, 10] are a type of integrated systems that represent relationships between concepts, considered as nodes of a neural network. The strong point of those approaches is that knowledge elicitation from the experts is reduced to a minimum. A weak point of them is that the resulted systems lack the naturalness and modularity of symbolic rules. This is mainly due to the fact that those approaches give pre-eminence to connectionism. For example, the systems in[8, 10] are more or less like black boxes and, to introduce new knowledge, one has to modify a

large part of the network. Connectionist knowledge bases cannot actually be incrementally developed.

We use *neurules*[11, 12], which achieve a uniform and tight integration of a symbolic component (production rules) and a connectionist one (the adaline unit). Each neurule is considered as an adaline unit. However, pre-eminence is given to the symbolic component. Thus, the constructed knowledge base retains the modularity of production rules, since it consists of autonomous units (neurules), and their naturalness, since neurules look much like symbolic rules. A difficult point in this approach is the inherent inability of the adaline unit to classify non-separable training examples.

In this paper, we describe a method for generating neurules directly from empirical (training) data. We overcome the above difficulty of the adaline unit by introducing the notion of 'closeness', as far as the training examples are concerned. That is, in case of failure, we produce two subsets of the initial training set of the involved neurule, which contain 'close' success examples and train a copy of the neurule for each subset. Failure of training any copy leads to production of further subsets as far as success is achieved. This paper is a revised and extended version of the one presented at FLAIRS'2000[13].

The structure of the paper is as follows. Section 2 presents neurules and the corresponding expert system architecture. Section 3 presents the basic ideas introduced in the paper. In Section 4, the algorithm for creating a hybrid knowledge base directly from empirical data is described. In Section 5 the hybrid inference mechanism is presented. Section 6 contains examples and experimental results. Section 7 discusses related work and finally, Section 8 concludes.

2. Neurules

2.1 *Structure*

Neurules (: *neur*al *rules*) are a kind of hybrid rules. Each neurule is considered as an adaline unit (Fig.1a). 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 its *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 weight of the *bias input* (C_0 = 1, not illustrated in Fig.1 for the sake of simplicity) of the unit.

Each input takes a value from the following set of discrete values: [1 (true), 0 (false), 0.5 (unknown)]. This gives the opportunity to easily distinguish between the falsity and the absence of a condition, in contrast to symbolic rules. The *output D*, which represents the *conclusion* (decision) of the rule, is calculated via the formulas:

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

as usual[14], where **a** is the *activation value* and *f(x)* the *activation function*, which is a threshold function (Fig.1b). Hence, the output can take one of two values, '-1' and '1', representing failure and success of the rule respectively.

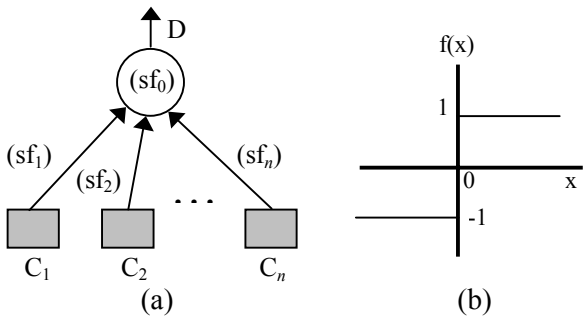

Fig.1. (a) A neurule as an adaline unit (b) the activation function

2.2 *Syntax and semantics*

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

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

where <variable> denotes a *variable*, that is a symbol representing a concept in the domain, e.g. 'sex', 'pain' etc, in a medical domain, or 'learning-ability', 'related-concept' etc in a tutoring domain. A variable in a condition can be either an *input variable* or an *intermediate variable* or even an *output variable*, whereas a variable in a conclusion can be either an *intermediate* or an *output variable*. An input variable takes values from the user (input data), whereas intermediate and output variables take values through inference, since they represent intermediate and final conclusions respectively. <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*. The significance factor of a condition represents the significance (weight) of the condition in drawing the conclusion(s). (For example neurules see Fig. 3, Section 6).

We distinguish between *input*, *intermediate* and *output neurules*. An input neurule is a neurule having only input variables in its conditions and intermediate or output variables in its conclusions. An intermediate neurule is a neurule having at least one intermediate variable in its conditions and intermediate variables in its conclusions. An output neurule is one having an output variable in its conclusions.

2.3 *The neurule based architecture*

In Fig.2, the architecture of a hybrid neurule-based expert system is illustrated. The run-time system (in the dashed rectangle) consists of three modules, functionally similar to those of a conventional rule-based system: the *neurule base (NRB)*, the *hybrid inference mechanism (HIM)* and the *working memory (WM)*.

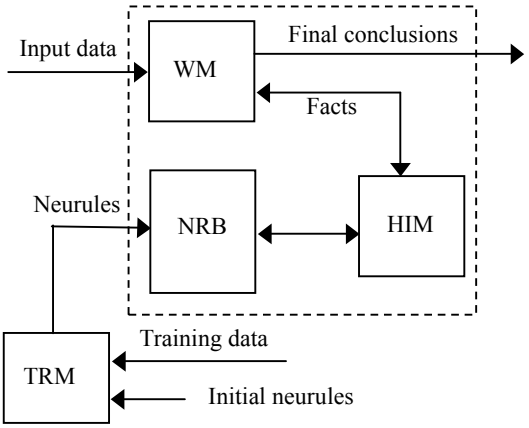

Fig.2 A neurule-based expert system architecture

The NRB contains neurules produced from empirical (training) data (see Section 4). The initial neurules, constructed by the user, are trained using the *training mechanism (TRM)* and the training examples produced from the available empirical (training) data. The HIM is responsible for making inferences by taking into account the input data in the WM and the rules in the NRB. The WM contains *facts*. A fact has the same format as a condition/conclusion of a rule, however, it can have as value the special symbol "unknown". Facts represent either conditions given by the user or intermediate/final conclusions produced during an inference course.

3. The Basic Ideas

The main objective of our approach is to produce modular hybrid knowledge bases of as fine granularity as possible. Our main choice to this end is to use neurules. As it is clear from the previous section, neurules are hybrid rules that give pre-eminence to and look much like symbolic rules. By giving pre-eminence to the symbolic component, we retain the naturalness and modularity of symbolic rules. Internally, neurules are considered as independent neural units, more specifically, as adaline neural units, which are individually trained. As a single neural unit is the finest possible neural granule, a neurule is the finest possible hybrid granule.

The main problem with this is the inherent inability of a single neural unit to classify training patterns (examples) that correspond to a non-separable (boolean) function. A well known such function is the XOR function, which is the basis of non-separability in many cases. The training examples that correspond to the XOR

function with two input variables are presented in Table 1, where v_1 and v_2 represent the input values ('0' and '1' mean 'false' and 'true' respectively). Also, d represents the output value ('-1' and '1' mean 'inactive' and 'active' respectively). We call *success examples* the patterns with d=1 and *failure examples* those with d=-1. From a knowledge representation point of view, success examples result in a cell's activation and hence in (positive) knowledge production, whereas failure examples act like a protection from misactivations (produce negative knowledge).

As it is known, there is no single-cell network that can represent the XOR function[14]. For example, a single unit cannot correctly classify all four patterns of Table 1. However, it can classify any three of them. Hence, if we remove one of the success examples, the remaining examples can be used to train a single unit (neurule). Furthermore, to be able to represent the removed success example, we need a second unit (neurule). However, to avoid misactivations we should use the failure examples, alongside the removed success example, to train the second neural unit. Thus, two independent neural units (neurules) are needed to represent the two-input XOR function. The first unit is trained to classify the examples in the set {[0 0 -1], [0 1 1], [1 1 -1]} and the second those in {[0 0 -1], [1 0 1], [1 1 -1]}.

Table 1. Two-input XOR examples

v_1	v_2	d
0	0	-1
0	1	1
1	0	1
1	1	-1

Table 2. Three-input XOR examples

v_1	v_2	v_3	d
0	0	0	-1
0	0	1	1
0	1	0	1
0	1	1	-1
1	0	0	1
1	0	1	-1
1	1	0	-1
1	1	1	1

In creating these training sets, we had two implicit requirements in mind:

- Each unit (neurule) should be activated by at least one success example.
- There should be no two units (neurules) that can be activated by the same success example.

The first requirement assures that, there is no unit (neurule) that will never be activated, that is will never produce an output. The second assures that, there are no two units (neurules) that will be activated by the same data. Each subset, in order to satisfy these requirements, contains one of the success examples and all the failure examples. This means that the success examples are the basis for the creation of the training subsets. Notice, here, that the two success examples are totally unrelated, that is they have no common input values.

Similar things hold for the three-input XOR function (see Table 2). In this case, four independent neural units (neurules) are necessary to represent it. Each of them represents a subset of the eight training examples of Table 2 that includes one of the four success examples and all of the failure examples. Again, three of the success

examples are totally unrelated, whereas the forth has only one common input value with the others.

This analysis, which shows that totally or greatly unrelated success examples may cause non-separability, and an experience with a related problem[12] led us to introduce the notion of 'closeness' between success examples (see Section 4.2) as a criterion for the creation of the training subsets with more than one success example. Closeness is used to guide distribution of success examples between subsets.

In using the independent units (neurules) coming from the same training set, if one of them gets active, there is no need to evaluate the rest ones, since there is no possibility to have other activated units (neurules) for the same input data.

4. Neurule Base Construction

The algorithm for constructing a hybrid rule base from empirical (training) data is outlined below:

1. Determine the input, intermediate and output variables and use dependency information to construct an initial neurule for each intermediate and output variable.
2. Determine the training set for each initial neurule from the training data, train each initial neurule using its training set and produce the corresponding neurule(s).
3. Put the produced neurules into the neurule base.

In the sequel, we elaborate on each of the first two steps of the algorithm.

4.1 *Constructing the initial neurules*

To construct *initial neurules*, first we need to know or determine the input, intermediate and output variables. Then, we need *dependency information*. Dependency information indicates which variables (concepts) the intermediate and output variables (concepts) depend on. If dependency information is missing, then output variables depend only on input variables, as indicated by the *training data*.

In constructing an initial neurule, all the conditions including the input, intermediate and the output variables that contribute in drawing a conclusion (which includes an intermediate or an output variable) constitute the inputs of the initial neurule and the conclusion its output. So, a neurule has as many conditions as the possible input, intermediate and output variable-value pairs. Also, one has to produce as many initial neurules as the different intermediate and output variable-value pairs specified. Let's assume that in the medical diagnosis domain there are four symptoms and two diseases. The symptoms are expressed by the conditions C1, C2, C3 and C4, and the diseases by the conclusions D1 and D2. We also know that D1 depends on C1, C2, C3 and D2 on C3, C4. Then, the following initial neurules are constructed: "(0) **if** C1 (0)**,** C2 (0)**,** C3 (0) **then** D1", "(0) **if** C3 (0)**,** C4 (0) **then** D2". A zero initial value is assigned to each factor by default, except if the user assigns non-zero ones. For specific examples, see Section 6.

4.2 *Training the initial neurules*

From the initial training data, we extract as many (sub)sets as the initial neurules. Each such set, called a *training set*, contains *training examples* in the form [v_1 v_2 … v_n d], where v_i, i= 1, …,n are their *component values*, which correspond to the n inputs of the neurule, and d is the *desired output* ('1' for success, '-1' for failure). Each training set is used to train the corresponding initial neurule and calculate its factors. The learning algorithm employed is the standard least mean square (LMS) algorithm[14].

However, there are cases where the LMS algorithm fails to specify the right significance factors for a number of neurules. That is, the corresponding adaline units of those rules do not correctly classify some of the training examples in their training sets. This means that the training examples correspond to a non-separable (boolean) function.

To overcome this problem, the training set of the initial neurule is divided into subsets in a way that each subset contains success examples, which are "close" to each other in some degree. The *closeness* between two examples is defined as the number of common component values. For example, the closeness of [1 0 1 1 1] and [1 1 0 1 1] is '2'. Also, we define as *least closeness pair* (LCP), a pair of success examples with the least closeness in a training set. There may be more than one LCP in a training set.

Initially, a LCP in the training set is found and two subsets are created each containing as its initial element one of the success examples of that pair, called its *pivot*. Each of the remaining success examples 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*.

So, step 2 of the algorithm for each initial neurule is analyzed as follows:

2.1 From the initial training data, produce as many initial training sets as the number of the initial neurules.

2.2 Train each initial neurule, by applying the LMS algorithm to its initial training set. If the calculated factors classify correctly all the examples, produce the corresponding neurule. Otherwise, find a LCP and produce two subsets of the initial set. In each subset put its pivot, the success examples of the initial training set which are closer to the pivot, and all the failure examples. In constructing the subsets, success examples with the same closeness to both pivots are put in the subset containing a success example with the greatest closeness to it, otherwise to the one with the least number of success examples.

2.3 For each subset do the following:

2.3.1 Perform training of a copy of the corresponding initial neurule and calculate its factors.

2.3.2 If the calculated factors misclassify examples belonging to the subset, further divide the subset into smaller subsets as in step 2.2 and apply step 2.3.

2.3.3 Produce the corresponding neurule.

5. The Inference Mechanism

Although the focus of this paper is not on the inference of the system, for the sake of completeness, we concisely refer to it in this section. The *hybrid inference mechanism* (HIM) implements the way neurules co-operate to reach a conclusion. HIM gives pre-eminence to symbolic reasoning, which is based on a backward chaining strategy. As soon as the initial input data is given and put in the WM, the output neurules are considered for evaluation. One of them is selected for evaluation. Selection is based on textual order. A rule succeeds if the output of the corresponding adaline unit is computed to be '1', after evaluation of its conditions (inputs).

A condition evaluates to 'true', if it matches a fact in the WM, that is there is a fact with the same variable, predicate and value. A condition evaluates to 'unknown', if there is a fact with the same variable, predicate and 'unknown' as its value. A condition cannot be evaluated if there is no fact in the WM with the same variable. In this case, either a question is made to the user to provide data for the variable, in case of an input variable, or an intermediate neurule in NRB with a conclusion containing that variable is examined, in case of an intermediate variable. A condition with an input variable evaluates to 'false', if there is a fact in the WM with the same variable, predicate and different value. A condition with an intermediate variable evaluates to 'false' if additionally to the latter there is no unevaluated intermediate neurule in the NRB that has a conclusion with the same variable. Inference stops either when one or more output neurules are fired (success) or there is no further action (failure).

In this process, because sibling neurules concern the same conclusion, if one of them fires, there is no need to evaluate any of the rest. To further increase inference efficiency, a number of heuristics are used[12].

6. Examples and Experimental Results

In this section, we present application of our algorithm for constructing neurule bases to two different sets of training data. More specifically, we present the construction process and the resulted neurule base in each case. Also, we compare three methods for choosing the LCP.

6.1 *Fitting contact lenses*

The first data set was taken from a machine learning ftp repository[15]. It consists of 24 patterns (p1-p24) and concerns empirical data for fitting contact lenses (see Table

3). There are four input variables and one output variable. The input variables (with their possible values) are: *age* (young, pre-presbyopic, presbyopic), *spectacle prescription* (myope, hypermyope), *astigmatic* (no, yes), *tear rate* (reduced, normal). The output variable is: *lenses-class* (hard-lenses, soft-lenses, no-lenses). There are no intermediate variables, so there is no dependency information. Given that the output variable can take three possible values, we need three initial neurules, corresponding to the three possible conclusions. The output variable depends on all input variables, as empirical data shows. So, each initial neurule contains all the conditions related to the input variables. Each input variable produces as many conditions as its possible values. So, each neurule has nine conditions. The initial neurules are the same as the first three final neurules (see Fig. 3), except that the initial neurules have zero factors.

Table 3. Data set for fitting contact lenses

Pat. No	age	spect-pres	astigmatic	tear-rate	lenses-class
p1	young	myope	no	reduced	no-lenses
p2	young	myope	no	normal	soft-lenses
p3	young	myope	yes	reduced	no-lenses
p4	young	myope	yes	normal	hard-lenses
p5	young	hypermetrope	no	reduced	no-lenses
p6	young	hypermetrope	no	normal	soft-lenses
p7	young	hypermetrope	yes	reduced	no-lenses
p8	young	hypermetrope	yes	normal	hard-lenses
p9	pre-presbyopic	myope	no	reduced	no-lenses
p10	pre-presbyopic	myope	no	normal	soft-lenses
p11	pre-presbyopic	myope	yes	reduced	no-lenses
p12	pre-presbyopic	myope	yes	normal	hard-lenses
p13	pre-presbyopic	hypermetrope	no	reduced	no-lenses
p14	pre-presbyopic	hypermetrope	no	normal	soft-lenses
p15	pre-presbyopic	hypermetrope	yes	reduced	no-lenses
p16	pre-presbyopic	hypermetrope	yes	normal	no-lenses
p17	presbyopic	myope	no	reduced	no-lenses
p18	presbyopic	myope	no	normal	no-lenses
p19	presbyopic	myope	yes	reduced	no-lenses
p20	presbyopic	myope	yes	normal	hard-lenses
p21	presbyopic	hypermetrope	no	reduced	no-lenses
p22	presbyopic	hypermetrope	no	normal	soft-lenses
p23	presbyopic	hypermetrope	yes	reduced	no-lenses
p24	presbyopic	hypermetrope	yes	normal	no-lenses

The training sets of the initial neurules are extracted from the empirical data (Table 3) and are given in Table 4. In that table, the following correspondences are considered: age1 → 'age is young', age2 → 'age is pre-presbyopic', age3 → 'age is presbyopic', spec-pres1 → 'spectacle-prescription is myope', spec-pres2 → 'spectacle-prescription is hypermetrope', astig1 → 'astigmatic is no', astig2 → 'astigmatic is yes', tear-rate1 → 'tear-rate is reduced', tear-rate2 → 'tear-rate is normal', lenses-class1 → 'lenses-class is hard-lenses', lenses-class2 → 'lenses-class is soft-lenses', lenses-class3 → 'lenses-class is no-lenses'.

Table 4. Initial training sets for the contact lenses example

Ex. No	age 1	age 2	age 3	spec-pres 1	spec-pres 2	astig 1	astig 2	tear-rate 1	tear-rate 2	lenses-class		
										1	2	3
p1	1	0	0	1	0	1	0	1	0	**-1**	**-1**	**1**
p2	1	0	0	1	0	1	0	0	1	**-1**	**1**	**-1**
p3	1	0	0	1	0	0	1	1	0	**-1**	**-1**	**1**
p4	1	0	0	1	0	0	1	0	1	**1**	**-1**	**-1**
p5	1	0	0	0	1	1	0	1	0	**-1**	**-1**	**1**
p6	1	0	0	0	1	1	0	0	1	**-1**	**1**	**-1**
p7	1	0	0	0	1	0	1	1	0	**-1**	**-1**	**1**
p8	1	0	0	0	1	0	1	0	1	**1**	**-1**	**-1**
p9	0	1	0	1	0	1	0	1	0	**-1**	**-1**	**1**
p10	0	1	0	1	0	1	0	0	1	**-1**	**1**	**-1**
p11	0	1	0	1	0	0	1	1	0	**-1**	**-1**	**1**
p12	0	1	0	1	0	0	1	0	1	**1**	**-1**	**-1**
p13	0	1	0	0	1	1	0	1	0	**-1**	**-1**	**1**
p14	0	1	0	0	1	1	0	0	1	**-1**	**1**	**-1**
p15	0	1	0	0	1	0	1	1	0	**-1**	**-1**	**1**
p16	0	1	0	0	1	0	1	0	1	**-1**	**-1**	**1**
p17	0	0	1	1	0	1	0	1	0	**-1**	**-1**	**1**
p18	0	0	1	1	0	1	0	0	1	**-1**	**-1**	**1**
p19	0	0	1	1	0	0	1	1	0	**-1**	**-1**	**1**
p20	0	0	1	1	0	0	1	0	1	**1**	**-1**	**-1**
p21	0	0	1	0	1	1	0	1	0	**-1**	**-1**	**1**
p22	0	0	1	0	1	1	0	0	1	**-1**	**1**	**-1**
p23	0	0	1	0	1	0	1	1	0	**-1**	**-1**	**1**
p24	0	0	1	0	1	0	1	0	1	**-1**	**-1**	**1**

L1: (-2.4) if age is young (4.8),
age is pre-presbyotic (-4.4),
age is presbyotic (-4.5),
spectacle is myope (-0.9),
spectacle is hypermetrope (-2.1),
astigmatic is no (-6.4),
astigmatic is yes (3.3),
tear-rate is reduced (-7.5),
tear-rate is normal (4.8)
then lenses-class is hard-lenses

L2: (-2.4) if age is young (-0.5),
age is pre-presbyotic (-0.3),
age is presbyotic (-2.7),
spectacle is myope (-4.0),
spectacle is hypermetrope (0.9),
astigmatic is no (2.9),
astigmatic is yes (-6.4),
tear-rate is reduced (-7.4),
tear-rate is normal (4.4)
then lenses-class is soft-lenses

L3: (0.8) if age is young (-0.6),
age is pre-presbyotic (-0.6),
age is presbyotic (1.2),
spectacle is myope (1.6),
spectacle is hypermetrope (-0.2),
astigmatic is no (2.7),
astigmatic is yes (-2.0),
tear-rate is reduced (4.4),
tear-rate is normal (-4.6)
then lenses-class is no-lenses

L4: (-0.7) if age is young (-6.2),
age is pre-presbyotic (1.6),
age is presbyotic (3.2),
spectacle is myope (-5.8),
spectacle is hypermetrope (4.7),
astigmatic is no (-4.1),
astigmatic is yes (2.6),
tear-rate is reduced (3.4),
tear-rate is normal (-4.5)
then lenses-class is no-lenses

Fig. 3. The neurule base for the contact lenses example

Each of the three training sets consists of 24 examples. The examples in each of them have the same input patterns, but different output values, which are presented in Table 4 under the columns 1, 2 and 3 of the output variable 'lenses-class'.

For the first two initial neurules, the calculated factors successfully classified all training examples. The produced neurules L1 and L2 are presented in Fig. 3. However, it didn't happen the same with the third initial neurule. So, from its initial training set two subsets were produced.

There were six LCPs found (with closeness equal to '1'): (p1, p16), (p1, p24), (p7, p18), (p9, p24), (p15, p18) and (p16, p17). The first of them was chosen as the LCP. Then, from the third initial training set, two subsets were produced. The first, with pivot p1, included the success examples p3, p5, p9, p17, p18, p19 and p21, which were closer to p1 than to p16, and all failure examples. The second subset, with pivot p16, included the success examples p7, p11, p13, p15 and p24 and all the failure examples. Training of both copies of the third initial neurule was successful and two final neurules, L3 and L4, were produced (see Fig. 3).

6.2 *Diseases of the sarcophagus*

The second set of training data was taken from[14]. It includes 8 patterns that concern acute theoretical diseases of the sarcophagus. According to the example in[14], there are six symptoms (Swollen feet, Red ears, Hair loss, Dizziness, Sensitive aretha, Placibin allergy), two diseases (Supercilliosis, Namastosis), whose diagnosis is based on the symptoms and three possible treatments (Placibin, Biramibio, Posiboost).

Table 5. Data set for the diseases of sarcophagus

Pat. No	Sym 1	Sym 2	Sym 3	Sym 4	Dis 1	Dis 2	Treat 1	Treat 2	Treat 3
p1	T	F	×	F	T	F	T	F	T
p2	F	T	T	F	F	T	T	T	F
p3	T	T	F	T	T	T	F	F	F
p4	F	F	T	F	F	F	F	F	F
p5	×	T	T	T	T	T	F	T	T
p6	F	T	T	F	T	T	T	T	F
p7	T	F	F	T	T	F	F	F	F
p8	T	F	T	T	F	T	F	F	F

For reasons that will become clear in Section 7, we omit the symptoms 'Swollen feet' and 'Red ears' as well as their related data. Thus, symptoms are reduced to four. Also, we consider that 'Supercilliosis' does not any more depend on symptoms, because they have been removed, but it is given as input information. The training data for our example are given in Table 5, where Sym1 → 'Hair loss', Sym2 → 'Dizziness', Sym3 → 'Sensitive aretha', Sym4 → 'Placibin allergy', Dis1 → 'Supercilliosis', Dis2 → 'Namastosis', Treat1 → 'Placibin', Treat2 → 'Biramibio' and Treat3 → 'Posiboost'. Also, 'T'and 'F' mean 'true'and 'false'

respectively and '×' means 'unknown'. Finally, dependency information is provided (see Table 6), which shows the dependency between concepts.

There is one input variable, namely *symptom*, which can take four possible values (the four symptoms). Also, there is one variable, called *disease*, which is both an input and an intermediate variable, depending on its value (see dependency information in Table 6). It can take two possible values (the two diseases). Finally, there is a last variable, *treatment*, which is both an intermediate and an output variable (see dependency information in Table 6) and can take three possible values (the three treatments). Because there are totally four possible values for the intermediate and output variables, four initial neurules are required.

Table 6. Dependency information for the sarcophagus diseases problem

	Sym 1	**Sym 2**	**Sym 3**	**Sym 4**	**Dis 1**	**Dis 2**	**Treat 1**	**Treat 2**
Namastosis (Dis2)	√	√	√					
Placibin (Treat1)				√	√	√		
Biramibio (Treat2)	√				√	√		
Posiboost (Treat3)							√	√

The training sets for the four initial rules, which were extracted from the training data of Table 5, are presented in Tables 7-1 to 7-4. Notice that we didn't use patterns including the 'unknown' value.

Table 7-1. Training set for D1

HairLoss (Sym3)	**Dizziness (Sym4)**	**Sensitive aretha (Sym5)**	**Namastosis (Dis2)**
1	0	1	1
0	1	1	1
1	0	0	-1
0	0	1	-1
1	1	0	1

Table 7-2. Training set for D2

Placibin allergy (Sym6)	**Supercilliosis (Dis1)**	**Namastosis (Dis2)**	**Placibin (Trea1)**
0	0	1	1
0	1	0	1
1	1	1	-1
0	0	0	-1
0	1	1	1
1	1	0	-1
1	0	1	-1

Table 7-3. Training set for D3

HairLoss (Sym3)	Superscilliosis (Dis1)	Namastosis (Dis2)	Biramibio (Trea2)
1	1	0	-1
0	0	1	1
1	1	1	-1
0	0	0	-1
0	1	1	1
1	0	1	-1

Table 7-4. Training set for D4-5

Placibin (Trea1)	Biramibio (Trea2)	Posiboost (Trea3)
1	0	1
1	1	-1
0	0	-1
0	1	1

D1: (-2.2) **if** Symptom is Dizziness (4.6),
Symptom is SensitiveAretha (1.8),
Symptom is HairLoss (0.9)
then Disease is Namastosis

D2: (-0.4) **if** Symptom is PlacibinAllergy (-5.4),
Disease is Namastosis (1.8),
Disease is Supercilliosis (1.0)
then Treatment is Placibin

D3: (-0.4) **if** Symptom is HairLoss (-3.6),
Disease is Namastosis (1.8),
Disease is Supercilliosis (1.0)
then Treatment is Biramibio

D4: (-0.4) **if** Treatment is Biramibio (-4.4),
Treatment is Placibin (1.8)
then Treatment is Posiboost

D5: (-0.4) **if** Treatment is Placibin (-3.6),
Treatment is Biramibio (1.0)
then Treatment is Posiboost

Fig. 4. The neurule base for the sarcophagus diseases example

The calculated factors of all the initial neurules, except the last one, successfully classified all the training examples, even those containing the 'unknown' value, which were not used (generalization capability). Thus, three final neurules were produced (D1-D3 in Fig. 4). The fourth initial neurule, pertaining to the treatment Posiboost, failed to classify its respective set of training examples (Table 7-4), because they correspond to a non-separable function (XOR type). Thus, two subsets were created containing the first three and the last three examples respectively. Finally, two neurules were produced (D4, D5).

6.3 *Choosing the least closeness pair*

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

However, none of them faces all cases successfully. On the other hand, *random choice method* (RC) could be an alternative. In Table 8, results of using the above two heuristics and the random choice are presented. As random choice, we got the first LCP. The data used was indirectly taken from a medical rule base of 41 symbolic rules. We extracted training patterns from the symbolic rules by the method for training sets specification described in[12]. We did so, because we knew the optimal number of the neurules to be produced. In Table 8, Opt No indicates the optimal number of neurules and OLCPs the optimal LCPs, that is the LCPs that lead to the optimal number of final neurules.

Table 8. Comparison of methods for the least closeness pair choice

Conclusion	Exam-ples	Condi-tions	LCPs	OLCPs	Opt No	MC	BD	RC
inflammation	72	8	7	5	2	2/3	2	3
arthritis	144	9	3	2	2	2	2/3	2
primary-malignant	120	10	8	3	2	2	2/3	2
secondary-malignant	72	7	2	1	2	2	2/3	3
early-inflammation	324	11	7	7	4	4	4	4
soft-tissue-early-bone- inflammation	288	11	2	2	2	2	2	2
early-soft-tissue-inflammation	270	11	2	2	3	3	3	3

As we can see from Table 8, none of the methods assures optimality of the number of the produced neurules in all cases. The expression '2/3' means that the number of neurules can be 2 or 3. This is because there were more than one pairs that met the criterion of the method (e.g. had the same mean closeness or distributed the elements of the pairs in different subsets), but they weren't all optimal. The MC method did a bit better than the BD method only in cases with a relatively small number of examples. However, random choice didn't do bad, because,as a matter of fact, OLCPs is a large part, if not all, of LCPs. On the other hand, the MC method is computationally more expensive than the BD method and this latter than RC. So, given the computational effort required in the two heuristic methods, especially in the mean closeness, RC seems to be the best choice. Of course, some more experiments, with different rule bases, would give a more confident view on that.

7. Discussion and Related Work

The main contribution of this work is the representation of non-separable empirical data in a hybrid, natural and modular way, for use in expert systems, in contrast to existing connectionist approaches. A method for representing non-separable training examples in a connectionist expert system is presented in[14]. It is called the "distributed method". What that method does is to introduce a number of intermediate cells, between the inputs and the related output, called "distributed cells", whose bias and weights are randomly generated. This extra layer between inputs and output makes representation of non-separable examples possible.

The problem with that method is that those intermediate cells have no meaning, that is there are no concepts related to the problem assigned to them, as happens with the other nodes in a connectionist knowledge base. Also, there is no specific way to determine the number of the required intermediate cells. Thus, the resulted knowledge base is unnatural and complicated.

Following the process for generating a connectionist knowledge base from empirical data described in[14], we constructed the connectionist knowledge base corresponding to the data set for fitting contact lenses (see Section 6.1). According to the process, each node is individually trained. In case of non-separable training examples, the distributed method is used. The resulted (real) knowledge base is depicted in Fig. 5 and its corresponding neural network in Fig. 6, where we used real numbers for the weights and biases, instead of integers.

The knowledge base in Fig. 5 is actually a matrix that represents the connections, their weights and the biases of the cells (concepts) in the network of Fig. 6. A zero at a position in the matrix shows that there is no connection between the input cell (variable) of its column and the intermediate or output cell (variable) of its row.

In the network of Fig. 6, there are three output cells (the cyclic ones) representing the three outputs (conclusions), three intermediate (distributed) cells (the triangles) introduced for representing the non-separable training examples of the 'no-lenses' training set and nine input cells representing the nine input values. For readability reasons, we didn't draw all the connections neither put all the weights on the net of Fig. 6. Actually, all inputs are connected to all intermediate and all output cells and the outputs of all the intermediate cells are connected to the output cells.

Lenses class 1												
-13.2	8.4	1.0	0.9	0.9	-2.1	-6.4	5.1	-5.7	4.8	0	0	0
Lenses class 2												
-11.4	3.1	3.3	2.7	-4.0	2.7	2.9	-4.6	-7.4	6.2	0	0	0
Intermediate variable 1												
3.2	0	-0.8	-3.6	7.8	-6.5	7.7	-6.7	-11	10.4	0	0	0
Intermediate variable 2												
-4.0	-3.6	2.8	3.6	4.2	-2.9	4.1	-3.1	7.0	-7.6	0	0	0
Intermediate variable 3												
-7.6	0	6.4	0	0.6	0.7	0.5	0.5	-0.2	-0.4	0	0	0
Lenses class 3												
5.0	-5.4	-2.6	1.8	-1.2	2.5	-1.3	2.3	1.6	-2.2	-9.6	9.8	-5.3
Bias	age 1	age 2	age 3	spec -pr1	spec -pr2	astig 1	astig 2	tear- r1	tear- r2	Inter 1	Inter 2	Inter 3

Fig. 5. The connectionist knowledge base for the contact lenses example

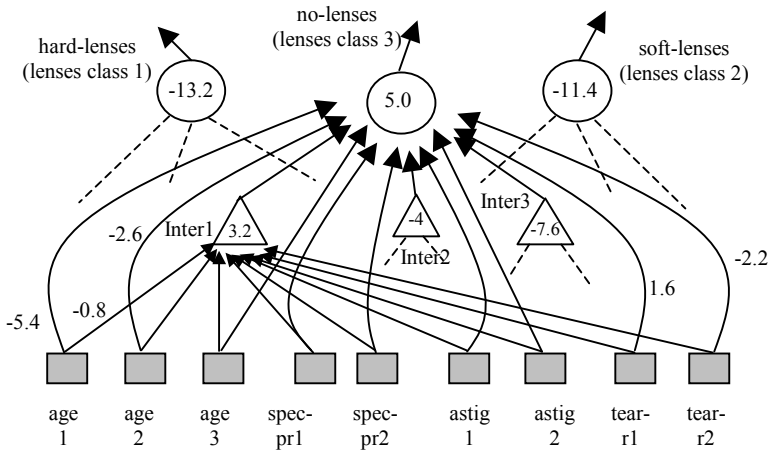

Fig. 6. The neural network for the contact lenses example

A comparison of the knowledge base in Fig. 5 to the one in Fig. 3 demonstrates the advantages of neurules. It is clear that the benefits of symbolic rule-based representation, such as naturalness and modularity are retained. Neurules are understandable, since significance factors represent the contribution of corresponding conditions in drawing the conclusion. On the other hand, the connectionist knowledge base is a multilevel network with some meaningless intermediate units. Thus, it lacks the naturalness of neurules.

The corresponding connectionist knowledge base for the diseases of the sarcophagus example is depicted in Fig. 7. It is a modified version of that presented in[14], to fit our modified example. The corresponding network is a multi-level network with three distributed cells, introduced by the training algorithm.

Let's suppose now that we get some new knowledge, which says that 'Supercilliosis' should not be given as input information, but it will be produced as an intermediate conclusion. Also, that it depends on 'Hair loss' and two new

symptoms (inputs), namely ‘Swollen feet’ and ‘Red ears’, according to the (training) data given in Table 9.

Namastosis												
-1	3	3	3	0	0	0	0	0	0	0	0	0
Placibin												
-2	0	0	0	-4	2	2	0	0	0	0	0	0
Biramibio												
-1	-4	0	0	0	1	3	0	0	0	0	0	0
Intermediate variable 1												
2	0	0	0	0	0	0	-4	5	0	0	0	0
Intermediate variable 2												
3	0	0	0	0	0	0	-2	2	0	0	0	0
Intermediate variable 3												
0	0	0	0	0	0	0	-1	-3	0	0	0	0
Posiboost												
3	0	0	0	0	0	0	-3	1	-3	-3	-1	0
Bias	Sym 1	Sym 2	Sym 3	Sym 4	Dis 1	Dis 2	Treat 1	Treat 2	Int 1	Int 2	Int 3	Treat 3

Fig. 7. The connectionist knowledge base for the diseases of the sarcophagus example.

Table 9. Training data for Supercilliosis

HairLoss (Sym1)	**SwollenFeet (Sym5)**	**RedEars (Sym6)**	**Supercilliosis (Dis1)**
1	1	1	1
0	0	0	-1
1	0	0	1
0	1	1	-1
0	1	0	1
1	0	1	-1

To introduce this new knowledge to our neurule base, we train a neurule with three conditions, corresponding to the three symptoms of Table 8, and a conclusion related to ‘Supercilliosis’. The result is the final neurule D6 depicted in Fig. 8, which is put into the neurule base.

To introduce that knowledge into the connectionist knowledge base of Fig. 7, not only training of a new unit is needed, but also modifications should be made to the knowledge base. More specifically, a new row (concerning ‘Superscilliosis’) and two new columns (concerning ‘Swollen feet’ and ‘Red ears’) should be added.

D6: (-0.4) **if** Symptom is RedEars (-4.4),
Symptom is SwollenFeet (3.6),
Symptom is HairLoss (2.7)
then Disease is Supercilliosis

Fig. 8. The new neurule concerning ‘Supercilliosis’.

Furthermore, one can easily add new neurules to or remove old neurules from a neurule base without making any changes to the knowledge base, since neurules are functionally independent units, given that they do not affect existing knowledge. Thus, a type of incremental development of the knowledge base is still supported, although by larger knowledge chunks. This corresponds to introducing one or more networks in an existing connectionist knowledge base sharing or not inputs and/or intermediate cell outputs. This is either difficult or impossible to do.

8. Conclusions

In this paper, we introduce a method for generating neurules, a kind of hybrid rules, from empirical data of binary type. Neurules integrate neurocomputing and production rules. Each neurule is represented as an adaline unit. Thus, the corresponding rule base consists of a number of neurules (autonomous adaline units). In this way, the produced neurule base retains the modularity of symbolic rule bases. Also, it retains their naturalness, since neurules look much like symbolic rules. Furthermore, incremental development is still supported. This is in contrast to existing connectionist knowledge bases, which are not modular and thus do not actually offer incremental development.

A difficult point in our approach is the inherent inability of the adaline unit to classify non-separable training examples. We overcome this difficulty by introducing the notion of 'closeness', as far as the training examples are concerned. That is, in case of failure, from the training set of the neurule two subsets of 'close' examples are produced and two copies of the neurule are trained. Failure of any copy training leads to further subsets production until success is achieved.

A weak point of the neurules is the fact that we have multiple representations of the same knowledge, in case of sibling rules.

Given the capability of producing neurules from empirical binary data and their advantages over symbolic rules, as far as inference efficiency and the rule base size are concerned[12], we can argue that neurules are more suitable for representing knowledge in web-based intelligent tutoring systems than symbolic rules. This is our current continuation on this research.

Acknowledgements

This work was partially supported by the GSRT of Greece, Program ΠΕΝΕΔ'99, Project No ΕΔ234.

References

[1] L. M. Fu (Ed), *Proceedings of the International Symposium on Integrating Knowledge and Neural Heuristics (ISIKNH'94)*, Pensacola, FL (May 1994).

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

[3] M. Hilario, *An Overview of Strategies for Neurosymbolic Integration*, ch.2 in [2].

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

[5] F. Kozato and Ph. De Wilde, *How Neural Networks Help Rule-Based Problem Solving*, Proceedings of the ICANN'91 (1991) 465-470.
[6] Tan C.L. and T. S. Quash, *Implementation of rule-based expert systems in a neural network architecture*, The World Congress on Expert Systems Proceedings (1991) 1843-1851.
[7] Kuncicky D. C., S. I. Hruska and D. C. Lacher, *Hybrid Systems: The equivalence of rule-based expert system and artificial neural network inference*, International Journal of Expert Systems, (1992) 4(3) 281-297.
[8] S. I. Gallant, *Connectionist Expert Systems*, CACM, **31** (1988) 152-169.
[9] B. Boutsinas and M. N. Vrahatis, *Nonmonotonic Connectionist Expert Systems*, Proceedings of the 2nd WSES/IEEE/IMACS International Conference on Circuits, Systems and Computers, Athens, Hellas (Oct. 1998).
[10] A. Z. Ghalwash, *A Recency Inference Engine for Connectionist Knowledge Bases*, Applied Intelligence **9** (1998) 201-215.
[11] I. Hatzilygeroudis, J. Prentzas, *Neurules: Integrating Symbolic Rules and Neurocomputing*, in D. Fotiades and S. Nikolopoulos (Eds), Advances in Informatics, World Scientific Pub., 2000, 122-133.
[12] I. Hatzilygeroudis and J. Prentzas, *Neurules: Improving the Performance of Symbolic Rules*, International Journal on Artificial Intelligence Tools (IJAIT), 9(1) (2000) 113-130.
[13] I. Hatzilygeroudis and J. Prentzas, *Producing Modular Hybrid Rule Bases for Expert Systems*, Proceedings of the 13th International FLAIRS Conference, Orlando, FL (May 2000) 181-185.
[14] S. I. Gallant, *Neural Network Learning and Expert Systems*, MIT Press (1993).
[15] ftp://ftp.ics.uci.edu/pub/machine-learning-databases/