Producing Modular Hybrid Rule Bases for Expert Systems

Ioannis Hatzilygeroudis, 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)
E-mails: ihatz@cti.gr, ihatz@ceid.upatras.gr, prentzas@ceid.upatras.gr
Phone: +30-61-997807
Fax: +30-61-991909

Abstract

Neurules are a kind of hybrid rules integrating neurocomputing and production rules. Each neurule is represented as an adaline unit. Thus, the corresponding rule base consists of a number of autonomous adaline units (neurules). Due to this fact, a modular and natural rule base is constructed, in contrast to existing connectionist rule bases. In this paper, we present a method for generating neurules from empirical data. We overcome the difficulty of the adaline unit to classify non-separable training examples by introducing the notion of 'closeness' between training examples and splitting each training set into subsets of 'close' examples.

1. Introduction

Recently, there has been extensive research activity at combining (or integrating) the symbolic and the connectionist approaches (see e.g. (Sun and Alexandre 1997)) for knowledge representation in expert systems. There are a number of efforts at combining symbolic rules and neural networks for knowledge representation (Fu and Fu 1990; Sun 1994; Ghalwash 1998; Boutsinas and Vrahatis 1998). The strong point of those approaches is that knowledge elicitation from the experts is reduced to a minimum. A weak point is that the resulted systems lack the naturalness and modularity of symbolic rules. For example, the systems in (Gallant 1988, Gallant 1993, Ghalwash 1998) are more or less like black boxes and, to introduce new knowledge, one has to retrain either the whole or a large part of the network. Their knowledge bases cannot be incrementally developed.

We use *neurules* (Hatzilygeroudis and Prentzas 1999), which achieve a uniform and tight integration of a symbolic component (production rules) and a connectionist one (the adaline unit). Each neurule is represented as an adaline unit. 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 data (in the form of training examples). 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 divide the training set of a neurule into subsets of examples with some degree of closeness and train as many copies of the neurule as the subsets. Failure of any copy training leads to further divisions as far as success is achieved.

The structure of the paper is as follows. Section 2 presents the hybrid formalism and corresponding system architecture. In Section 3, the algorithm for creating a hybrid knowledge base directly from empirical data is described. In Section 4 the hybrid inference mechanism is presented. Section 5 contains experimental results. Finally, Section 6 concludes.

2. The Hybrid Formalism

2.1 Neurules

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 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 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:

$$D = f \qquad sf_0 + \sum_{i=1}^{n} sf_i C_i$$

as usual (see e.g. Gallant 1993), 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.

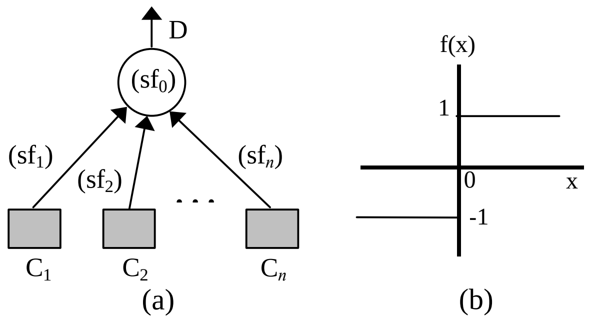

Fig.1 (a) A neurule as an adaline unit (b) the activation function

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. A variable in a condition can be either an *input variable* or an *intermediate variable*, whereas a variable in a conclusion can be either an *intermediate* or an *output variable*. 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).

2.3 The Hybrid Architecture

In Fig.2, the architecture of the hybrid rule-based expert system to incorporate neurules 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 *hybrid rule base (HRB)*, the *hybrid inference mechanism (HIM)* and the *working memory (WM)*.

The HRB contains neurules produced from empirical (training) data (see Section 3). The initial neurules, constructed by the user, are trained using the *training mechanism (TRM)* and the training examples produced from the available empirical data. The HIM is responsible for making inferences by taking into account the input data in the WM and the rules in the HRB. 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 initial conditions or intermediate/final conclusions produced during an inference course.

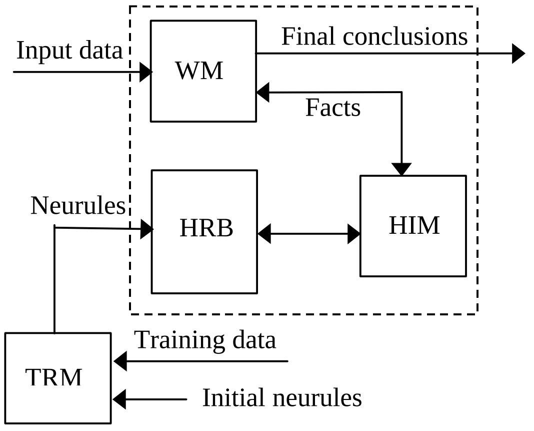

Fig.2 The hybrid expert system architecture

3. Hybrid Rule Base Construction from Empirical Data

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

1. Construct a neurule for each of the intermediate and final conclusions (variables). These neurules will be referred to as *initial neurules*.
2. Use the training examples to train each initial neurule. After training, one or more neurules will be produced from each initial neurule.
3. Put the produced neurules into the hybrid rule base.

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

3.1 Constructing the Initial Neurules

In constructing a neurule, all conditions, corresponding to the input and intermediate variables that contribute in drawing a conclusion, which corresponds to an intermediate or an output variable, constitute the inputs of a rule and the conclusion its output. Dependency information for intermediate or output variables, that is information about which input variables they depend on, is very useful. Thus, one has to produce as many rules as the different conclusions, intermediate and final to be drawn.

For example, in the medical diagnosis domain, if there are four symptoms expressed by the conditions C1, C2, C3, C4 and two diseases D1, D2, such that D1 depends on C1, C2, C3 and D2 on C3, C4, 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.

3.2 Training the Initial Neurules

From the initial training examples, we produce as many subsets as the initial neurules. Each subset contains examples of 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 subset is used to train the corresponding initial neurule and calculate its factors. The learning algorithm employed is the standard least mean square (LMS).

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. This means that the training examples correspond to a *non-separable (boolean) function*. It is known that the adaline model cannot fully represent such functions.

To overcome this problem, the set of the training examples used to train the initial neurule is divided into subsets in a way that each subset contains examples, which are "close" to each other in some degree. The *degree of closeness* between two examples is defined as the number of common component values. For example, the degree of closeness of [1 0 1 1 1] and [1 1 0 1 1] is '2'. Initially, the set is partitioned into subsets with low degree of closeness, e.g. '1'. To each subset, the examples of the other subsets with d ='-1' are added, to avoid rule misfiring. Then, as many copies of the initial neurule as the number of the subsets are trained. If the factors of a copy misclassify some of its examples, the corresponding subset is further partitioned into smaller subsets with examples of the next greater degree of closeness. This continues, until all examples are classified.

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

2.1 From the initial training examples, produce as many subsets as the number of the initial neurules.

2.2 Train each initial neurule by using the corresponding training subset. If the calculated factors classify correctly all the examples produce the corresponding neurule. Otherwise, divide the training subset into further subsets of examples with a degree of closeness equal to '1'. Add to each subset the examples with d=-1 of all other subsets.

2.3 For each new subset do the following:

2.3.1 Perform training of a copy of the corresponding 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 of examples with degree of closeness increased by one and apply step 2.2 recursively to these new subsets.

4. The Inference Mechanism

The *hybrid inference mechanism* (HIM) implements the way neurules co-operate to reach a conclusion. It is based on a backward chaining strategy. As soon as the input data is given and put in the WM, the rules having an output variable in their conclusion are considered for evaluation. One of them is selected for evaluation. Rule 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 a rule in HRB with a conclusion containing that variable is examined, in case of an intermediate variable. A condition evaluates to 'false' if there is a fact in the WM with the same variable, predicate and different value, in case of an input variable, and additionally there is no rule in the HRB that has a conclusion with the same variable, in case of an intermediate variable. Inference stops either when a rule with an output variable is fired (success) or there is no further action (failure).

To increase inference efficiency, a number of heuristics are used (Hatzilygeroudis and Prentzas 1999).

5. Experimental Results

We applied the algorithm for constructing a hybrid knowledge base to two different sets of training examples.

The first set, taken from (Mitchell 1997), is illustrated in Table 1. Here, the target attribute (output variable) *PlayTennis,* which can have values *Yes* or *No*, should be predicted based on the values of other attributes (input variables) concerning the environment.

There are two conclusions that can be drawn: (1) Playtennis is Yes and (2) PlayTennis is No. Thus, two initial neurules were produced, which were successfully trained. All the examples that came out from the rows of Table 1 were used in training of both neurules, but with inverse desired outputs. Finally, neurules PTR1 and PTR2 (see below) were produced. A few of the initial conditions in each neurule were removed, because their factors were near to zero and had no effect on the conclusion, like e.g. 'Temperature is Hot'.

Table 1

Outlook	Tempe-rature	Humidity	Wind	Play-Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

PTR1: (-0.4) **if** Humidity is High (-8.5),
Outlook is Sunny (-7.2),
Wind is Strong (-6.0),
Temperature is Cool (-4.9),
Outlook is Overcast (10.0),
Humidity is Normal (8.8),
Temperature is Mild (7.9),
Wind is Weak (5.0)
then PlayTennis is Yes

PTR2: (0.8) **if** Outlook is Sunny (7.3),
Humidity is High (6.5),
Wind is Strong (6.3),
Temperature is Cool (4.5),
Outlook is Rain (3.8),
Humidity is Normal (-7.5),
Temperature is Mild (-7.0),
Wind is Weak (-6.3)
then PlayTennis is No

The second set of training examples was taken from (Gallant 1988; Gallant 1993). They deal 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, a dependency information is provided.

So, there are two intermediate (the two diseases) and three final conclusions (the three treatments) to be drawn producing thus, five initial neurules. The training subsets, which were extracted from the examples provided in (Gallant 1988, 1993), and the corresponding neurules (DR1 to DR6) that were produced, are illustrated in the following tables.

The calculated factors of the initial neurule pertaining to the treatment Posiboost failed to classify its respective set of training examples (table DR5-6), 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 (DR5, DR6). The calculated factors of the other initial neurules successfully classified their corresponding set of training examples. Also, although a few examples containing unknown inputs were omitted, they were successfully classified.

DR1

RedEars	SwollenFeet	HairLoss	Supercilliosis
1	1	1	1
0	0	0	-1
0	0	1	1
1	1	0	-1
0	1	0	1
1	0	1	-1

DR2

HairLoss	Dizziness	Sensitive-Aretha	Namastosis
1	0	1	1
0	1	1	1
1	0	0	-1
0	0	1	-1
1	1	0	1

DR3

Placibin-Allergy	Supercilliosis	Namastosis	Placibin
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

DR4

HairLoss	Superscilliosis	Namastosis	Biramibio
1	1	0	-1
0	0	1	1
1	1	1	-1
0	0	0	-1
0	1	1	1
1	0	1	-1

DR5-6

Placibin	Biramibio	Posiboost
1	0	1
1	1	-1
0	0	-1
0	1	1

DR1: (-0.4) **if** RedEars is true (-4.4),
SwollenFeet is true (3.6),
HairLoss is true (2.7)
then Disease is Supercilliosis

DR2: (-2.2) **if** Dizziness is true (4.6),
SensitiveAretha is true (1.8),
HairLoss is true (0.9)
then Disease is Namastosis

DR3: (-0.4) **if** PlacibinAllergy is true (-5.4),
Disease is Namastosis (1.8),
Disease is Supercilliosis (1.0)
then Treatment is Placibin

DR4: (-0.4) **if** HairLoss is true (-3.6),
Disease is Namastosis (1.8),
Disease is Supercilliosis (1.0)
then Treatment is Biramibio

DR5: (-0.4) **if** Treatment is Biramibio (-4.4),
Treatment is Placibin (1.8)
then Treatment is Posiboost

DR6: (-0.4) **if** Treatment is Placibin (-3.6),
Treatment is Biramibio (1.0)
then Treatment is Posiboost

A comparison between this hybrid rule base and the equivalent connectionist knowledge base in (Gallant 1988; Gallant 1993; Ghalwash 1998) 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. Also, one can easily add new or remove old neurules making small or even no changes to the knowledge base, since neurules are functionally independent units. Thus, incremental development of the knowledge base is still supported, although by larger knowledge chunks.

On the other hand, the equivalent connectionist knowledge base in (Gallant 1988; Gallant 1993; Ghalwash 1998) is a multilevel network with some meaningless intermediate units. Thus, it lacks the naturalness of neurules. Furthermore, there are difficulties in introducing new or removing existing knowledge from the knowledge base. These actions entail changes to the knowledge base that may scale up to the retraining of the whole network, especially in cases of non-easy learning problems.

6. Conclusion

In this paper, we introduce a method for generating a kind of hybrid rules called neurules from empirical data. Neurules integrate neurocomputing and production rules. Each neurule is represented as an adaline unit. Thus, the corresponding rule base consists of a number of autonomous adaline units (neurules). In this way, the produced rule base retains the modularity of symbolic rule bases. This is in contrast to existing connectionist rule bases, which are more or less black boxes and thus do not allow for 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, we divide the training set of each neurule into subsets of 'close' examples and train as many copies of the neurule as the subsets. Failure of any copy training leads to further division until success is achieved.

Also, the naturalness of symbolic rules is retained in a degree, since significance factors represent the contribution of corresponding conditions in drawing the conclusion, a fact that makes sense.

References

Boutsinas, B., and Vrahatis, M. N. 1998. Nonmonotonic Connectionist Expert Systems. In Proceedings of the Second WSES/IEEE/IMACS International Conference on Circuits, Systems and Computers, Athens, Greece.

Fu, L-M, and Fu, L-C, 1990. Mapping rule-based systems into neural architecture, *Knowledge-Based Systems*, 3:48-56.

Gallant, S.I. 1988. Connectionist expert systems, *Communications of the ACM,* 31(2):152-169.

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

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

Hatzilygeroudis, I. and Prentzas, J. 1999. Neurules: Improving the Performance of Symbolic Rules. In *Proceedings of the Eleventh IEEE International Conference on Tools with Artificial Intelligence* (*TAI 99*), 417-424.

Mitchell, T. 1997. *Machine Learning,* McGraw-Hill.

Sun, R. 1994. *Integrating Rules and Connectionism for Robust Commonsense Reasoning*, Sixth-Generation Computer Technology, John Wiley & Sons.

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