HYPERPLANE-BASED NEURAL NETWORKS FOR REAL-VALUED DECISION TABLES

Marcin Szczuka szczuka@mimuw.edu.pl

Dominik Ślęzak slezak@alfa.mimuw.edu.pl

The University of Warsaw
Institute of Mathematics
Banacha 2, PL-02-097 Warsaw, Poland

Abstract: *Different approaches to real-valued decision tables are considered. Possible profits of combining neural networks and hyperplane-based methods are discussed.*

1. Introduction

Among other problems in decision support systems the task of dealing with real-valued cases seems to be one of key issues. Many approaches have been developed so far. One quite widely used method is application of neural networks of some kind. Although this approach is very powerful computationally, it has some restrictions.

First, one may have difficulties with finding proper layout of the network (number of neurons or layers, strategy of learning). Second, network itself does not provide us with clear interpretation of knowledge it contains.

In both above cases referring to some supporting techniques is needed. Therefore, in this paper we want to draw connections between neural networks and other methods.

At the beginning we introduce the framework for hyperplane approach to real-valued decision tables, as stated in [3],[4],[5].

Then, in Section 3, we construct the neural network corresponding to given hyperplane decision tree ([5]). Possible variations of this model are discussed in view of learning and outputs' interpretation.

Section 4 includes the outline of connections to fuzzy methods ([2]).

Further profits of combining described techniques are considered.

2. Basic notions

Let $\mathbf{A} = \left(U, A \cup \{d\}\right)$ be a decision table ([6]) where $U = \{u_1,\ldots,u_n\}$ is a finite set of objects called the *universe*, $A = \{f_1,\ldots,f_k\}$ is a set of real valued *conditional attributes (features)* determined on objects i.e. $f_i : U \to \Re$ for any $i \in \{1,\ldots,k\}$ and *d* is the decision $d : U \to \{1,\ldots,r\}$. The set of values for attribute $f_i \in A$ is denoted by V_{f_i}, the set of decision values by $V_d = \{1,\ldots,r\}$.

With this definition we can treat any object u_i in our decision table as a point in affine space $\Re^k$

$$P_i = \left(f_1(u_i), f_2(u_i),\ldots, f_k(u_i)\right)$$

for $i \in \{1,\ldots,k\}$.

Assuming that conditional attributes discern our objects (points) we define *decision classes* $C_1, C_2,\ldots, C_r$ as follows:

$$C_i = \left\{u \in U : d(u) = i\right\} \text{ for } i \in \{1,\ldots,k\}$$

Any hyperplane in $\Re^k$:

$$H_i = \left\{\left(x_1,\ldots,x_k\right) \in \Re^k : a_1x_1+\ldots+a_kx_k + a = 0\right\} \quad (1)$$

for $a, a_1,\ldots,a_k \in \Re$, allows us to define new binary attribute $H_i : U \to \{0,1\}$ over the set of objects:

$$H_i(u) = \begin{cases} 1 \text{ iff } a_1 f_1(u)+\ldots+a_k f_k(u) + a \geq 0 \\ 0 \text{ iff } a_1 f_1(u)+\ldots+a_k f_k(u) + a < 0 \end{cases} \quad (2)$$

A *decision rule* is a formula of the form $\alpha \Rightarrow d = i$, where $i \in V_d$ and α is a Boolean conjunction of descriptors i.e. expressions $a = v$, where $a \in A, v \in V_a$.

In case of binary attributes produced from hyperplanes, the decision rules are Boolean, so any set of them can be represented as the binary tree (see [5]).

As some notions from neural network theory will be used further, here we provide them. By three layer network we mean the one containing input, hidden and output layer. Of those three the input layer contains non-computational neurons which only broadcast the input signal to second, hidden layer. All the networks considered are fully connected. The excitation functions of neurons vary between layers and are not necessarily the standard sigmoid. For reference see e.g. [1].

3. Hyperplanes, decision trees and networks

For decision table with real-valued attributes we can use hyperplanes to obtain set of binary attributes that allows us to construct decision rules which approximate decision classes. The key issue in that approach is the choice of possibly most relevant hyperplanes. Of course there should be not too many of them in order to make our decision process simpler actually. In practice the choice of such hyperplanes can be made by genetic algorithm as in [3],[4]. The algorithm mentioned is also able to generate decision rules which describe decision classes using hyperplane-generated, binary attributes.

Once the hyperplanes and decision rules are constructed for given decision table ***A***, we may put them into the neural network.

PROPOSITION

Let the decision table ***A***, the set of hyperplanes ***H*** = $\{H_1,\ldots,H_m\}$ defined over affine space $\Re^k$ and the set of decision rules ***R*** for binary attributes defined by ***H*** be given. Then one can construct three-layer neural network with *n* inputs, *r* outputs and *m* neurons in hidden layer, such that it recognizes all objects from *U* similarly to given decision rules.

Step 1. The network has *k* inputs corresponding to conditional attributes. There is also one additional constant input called *bias*. Every input neuron sends its signal to all neurons in hidden layer.

For every hyperplane form ***H*** we construct one neuron in hidden layer. This neuron has weights equal to coefficients describing corresponding hyperplane. Literally, if we have hyperplane like in (1), with coefficients $a, a_1, \ldots, a_k$, then they are simultaneously the weights of *i*-th neuron, where weight *a* corresponds to the bias. Such a neuron has then the excitation function corresponding to binary attribute defined by formula (2), so the output of neuron is 0 if the inputted point lays below H_i and 1 in opposite case.

Step 2 For every decision value we construct one neuron in output layer, so together *r* outputs from network. Each output neuron is supposed to give high signal if the point inputted to the network belongs to corresponding decision class and low signal if not. To achieve such a behavior of neuron we will first encode all the decision rules we have for hyperplane attributes. One possible encoding is the following:

To any binary attribute H_i we attach weight 2^{-i}, $i \in \{1,\ldots,m\}$. For decision rule from ***R*** its code is given by $t_1 \cdot 2^{-1}+\ldots+t_m \cdot 2^{-m}$, where t_i=1 if the component $"H_i = 1"$ is present in the formula and t_i=0 otherwise. We put it into the network by setting the weights of all connections coming out of *i*-th hidden neuron to 2^{-i}.

With such encoded rules we construct the 0-1 excitation functions for output neurons. For *i*-th decision class (*i*-th output neuron) this function will be equal 1 iff the argument is equal to encoded value of any rule supporting this decision.

ILLUSTRATIVE EXAMPLE

Let us consider the example of iris classification from [7]. There are 150 objects, 3 decision classes, 4 real-valued, conditional attributes. The data is spliced to 60 (learning set) and 90 (testing set).

By using previously mentioned genetic algorithm the accuracy over 97% was achieved with only two hyperplanes. For each of three decision classes there is only one supporting rule. So, the corresponding network has 4 inputs, 2 hidden neurons and three outputs. All functions in output neurons are characteristic for single points as there is only one decision rule for each class.

Of course the straight method presented above is quite artificial. It is also inflexible. The network constructed in such a way is vulnerable for any kind of disturbed, imperfect data. It also cannot guarantee good behavior for new cases. Finally, it is very inconvenient for learning.

Thus, the several changes in presented method can be introduced to obtain the network with more practical abilities.

First of them is rather simple; the replacement of step functions in hidden layer by continuous ones. For instance, we can take sigmoidal function

$$f(x) = \frac{1}{1+e^{-\beta x}}$$

to have values between 0 and 1. By manipulating functions in hidden layer we allow less restrictive borders between positive and negative regions. That weakening of the border corresponds to the fuzzy approach. The excitation function of each neuron in hidden layer can be treated as the fuzzy membership function for the set of points laying above given hyperplane. That second function, for points laying beneath, is determined complementary. By adjusting the β coefficient in sigmoidal functions we are able to decide how sharp the pass between positive and negative region should be.

The second possible modification is more serious and has deeper consequences. The excitation function of output neuron can be changed to some continuous, widely used one. We propose the sine function $f(x) = \sin(x) + \lambda$. The λ coefficient allows us to determine the interval of output values in case we want them to be e.g. between 0 and 2.

To be able to use sine functions we have to posses the confidence that they will give us proper answers. So, we have to adjust the weights in output neurons and introduce some tolerance measure for outputs. In general, it can be proven that there is no way to adjust the weights to have exact fitting of encoded rules into minimums and maximums of sine function. Therefore, we will only expect that for positive cases corresponding to its decision class the output neuron will return a value between $1+\lambda-\varepsilon$ and $1+\lambda$, where $\varepsilon \geq 0$ is the range of tolerance.

Obtained flexibility of neuron functions enables to look at similarities between construction of the network and algorithm used to generate hyperplanes from new point of view. Genetic algorithm that we are basing on ([3],[4]) is constructed in a way that hyperplanes are chosen orderly. The first chosen is the one potentially most important i.e. the one that separates possibly largest number of pairs of points belonging to different decision classes. Therefore we set our encoding of decision rules according to the precedence of hyperplanes - the more important hyperplane, the bigger its weight. It can be seen that for the most significant hyperplanes, which separate relatively big number of points from different decision classes, it is easier to design output neurons paying the attention mostly to those rules which are to be positively recognized. Such an approach assures us that if two encoded rules for different decision classes are very close to each other as real numbers, then, even if we miss to choose the proper function discerning them and negative case will be classified as positive one, the possible losses would be minor in terms of number of wrongly recognized objects. It follows the fact that small differences of encodings correspond to least important (in terms of the number of discerned objects) hyperplanes.

Further changes we may want to introduce into our network model are to allow it to learn. With all modifications proposed above, our network is able to learn utilizing some version of backpropagation algorithm.

There are two possible ways of learning in such a network. We may train only the weights of connections between hidden and output layers. The weights between input and hidden layers are taken from hyperplane equations and do not change. Such a training is equivalent to searching for encoding of decision rules.

The other way is to perform regular training procedure for all weights in our network. In that case the information acquired from algorithm generating hyperplanes and rules is used only for establishing the network architecture. The rest, choice of weights in hidden layer which corresponds to division of attribute value space and in output layer which corresponds to decision tree is left to the learning algorithm. The knowledge that comes with hyperplanes and decision rules may be used as the reference for network performance benchmark.

The network with some or all above changes may behave more flexibly but also it may lack some preciseness. Therefore, we were considering different methods of naming the final decision. For example, instead of handling the tolerance, if the network do not give us the exact answer, then we can accept the neuron with highest output as the one corresponding to current decision value. The simple experiments showed that this approach allows to improve network performance in most cases. However, it does not enable the network to answer with more than one output neuron, just like it can be done by manipulating tolerance thresholds.

It is worth mentioning that for both above methods of interpreting output sine functions there is no difference whether we use continuous functions in hidden layer or not - in case of tolerance thresholds, they must be adjusted accurately enough, when in case of “the highest output value” method it is the same for any increasing excitation functions. The difference occurs when we want to replace described approaches by some continuous way of interpreting the answers.

4. The fuzzy connection

In previous section some connections between our approach and elements of fuzzy set theory were mentioned. Here we want to point them out more precisely.

While considering the continuous functions in hidden layer neurons, we may say that now every neuron represents some fuzzy variable ([2]). This variable is in fact fuzzyfied version of binary attribute generated by hyperplane. The value of output of hidden neuron directly corresponds to the degree with which the given element belongs to one of two classes established by hyperplane.

In the first version of our network the calculations done by output layer are equivalent to performing decision process based on binary decision rules. This kind of calculations, done with fuzzy variables and continuos, but tolerance-equipped functions in output neurons, does not correspond to so called fuzzy rules yet.

EXAMPLE

If we consider binary decision rule of the form:

$$\left(H_1(u)=1\right)\&\left(H_4(u)=1\right)\&\left(H_6(u)=0\right)\Rightarrow d=5$$

then the corresponding, then corresponding linguistic rule for fuzzy approach would be:

“*If* H_1 *is large and* H_4 *is large and* H_6 *is small then decision is 5*”

Now, although we have fuzzy membership functions for states $H_1(u)=1$, $H_4(u)=1$ and $H_6(u)=0$, we don’t give as an answer numerical degree to which $d=5$ - the first method tells us that d is *enough* 5 to fit in some tolerance threshold and the second , where the output with the highest value is chosen, says that d is *most likely* 5.

In fact, the straightforward relation to fuzzy reasoning occurs when we treat d as a real function like all other attributes. Then, there may be several ways of interpreting network outputs of network according to fuzzy methods. We want only to point out the some possibilities.

If decision is initially prescaled, then we can perform the hyperplane and rule search and construct neural network. By interpreting output neuron answers in fuzzy way we will be able to return to real-valued decision e.g. by taking weighted sum of output values. It is worth remembering that the network may be influenced by the procedure we want to apply to its outputs. Possible repercussions should reflect in special adjustment of weights, response functions in output neurons, or diversification of learning process.

Just like it could be said in case of considering discrete decision attributes, there is significant difference between the way of calculating rules in classical fuzzy approach and in our network. While the network performs all calculations in parallel, in fuzzy inference the decision is reached sequentially. It means that we first consider consequences of H_1 being large and after that go further to other conditions. This sequential approach can be in some way reflected by structure of weights in our network due to potentially bigger influence of signals possessing larger weights. However, the network may also learn some other way of encoding decision rules. That fact justifies the bigger expressiveness of our network. On the other hand, it determines the border for its interpretation in terms of hyperplane decision trees.

5. The feedback

In previous sections the usefulness of hyperplanes and decision rules for construction of neural network has been discussed. Herein we want to outline possible feedback profits for hyperplane approach coming from networks.

One important is the possibility to deal with continuous decision. That ability is very crucial for many real life applications like e.g. control processes. The generation of real-valued decision may be done using fuzzy techniques as mentioned in preceding section. We can also deal with continuous decision introducing some custom error function during the process of network training.

Other possible way is to construct the network for discretized decision and then perform some learning involving a kind of measure of difference between discrete and actual values. As this measure we may take e.g. weighted variance or its modifications.

One more possible profit for hyperplane approach is the ability of weakening the strict, binary conditions. In many situations we may face the problem that constraints introduced through hyperplanes and binary rules are too sharp. In that case we may construct the network and then, using some small learning steps, “tune up” the system to obtain more tolerant and flexible one.

Furthermore, we may consider reassignment of some hyperplanes by performing the learning of certain parts of network if the current ones don’t fit. It may save us from necessity of running genetic algorithm from the very beginning in order to deal with new situation.

6. Conclusions

We presented the method for establishing neural network based on knowledge about data coming from genetic algorithm. The simple experiments showed that network approach seem to be the good silk glove for iron fist of hyperplane-generating algorithm. The next task is to perform some more advanced tests. It will be also interesting to compare the expressive power or all mentioned methods. It may also lead to some conclusions about meaning of knowledge included in certain kind of neural networks.

References

[1] Fausett L.(1994). *Fundamentals of Neural Networks*. Prentice-Hall, Engelwood Cliffs, NJ.

[2] Kruse R., Gebhardt J., Klawonn F. (1994). *Foundations of Fuzzy Systems*. Wiley, Chichester.

[3] Nguyen H.S., Nguyen S.H., Skowron A.,(1996). *Searching for Features defined by Hyperplanes.* In Proceedings of ISMIS’96, Lecture Notes in AI **1097.** Springer Verlag, Berlin, pp.366-375

[4] Nguyen H.S., Nguyen S.H., (1996). *Some efficient algorithms for rough Set Methods*. In Proceedings of IPMU’96, Granada, Spain, pp.1452-1456.

[5] Nguyen H.S., Skowron A.,(1995). *Quantization of real value attributes. Rough Set and Boolean Reasoning Approaches.* Proc. of 2nd Joint annual Conference on Information Sciences. Wrigthswille Beach, NC, pp.34-37.

[6] Pawlak Z.(1991) *Rough Sets: Theoretical aspects of reasoning about data*. Kluwer, Dortrecht.

[7] ftp://ftp.ics.uci.edu/machine-learning-databases/iris University of California, Irvine.