ESANN'1993 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 7-8-9 April 1993, D-Facto public., ISBN 2-9600049-0-6, pp. 91-96

PROBABILISTIC DECISION TREES. AND MULTILAYERED PERCEPTRONS

Pascal BIGOT and Michel COSNARD
Laboratoire de l'Informatique du Parallélisme
Ecole Normale Supérieure de Lyon
69364 LYON Cedex 07 - FRANCE

Abstract: We propose a new algorithm to compute a multilayered perceptron for classification problems, based on the design of a binary decision tree. We show how to modify this algorithm for using ternary logic, introducing a Don'tKnow class. This modification could be applied to any heuristic based on recursive construction of a decision tree. Another way of dealing with uncertainty for improving generalization performance is to construct probabilistic decision trees. We explain how to modify the preceding heuristics for constructing such trees and associating probabilistic multilayered perceptrons.

1. Introduction.

Supervised learning is often used for the solution of classification problems, as for examples pattern recognition problems involving labeled training data [GS88]. Many learning algorithms have been proposed for solving this problem. For multilayered perceptrons the well-known back propagation algorithm [RM86] is such an effective solution. However, this algorithm is based on the minimization of an error function by using a gradient descent techniques which is slow to converge. Moreover it assumes that the architecture of the network (number of layers, number of units per layer, connections between layers) is known in advance.

Recently several works [Bre90], [SM90] have been devoted to the study of the relationship between neural nets and decision trees. A decision tree completely classifies the training set, by iteratively dividing the training data into subsets until each subset is homogeneous. Hence the architecture of the tree is not fixed but directly depends on the training set. Several methods already exist for computing decision trees for a given set of data. Brent [Bre90] has proposed an algorithm for associating a two-hidden-layered perceptron to a binary decision tree. The number of units of this perceptron is directly proportionnal to the number of internal nodes in the decision tree. Hence minimizing the number of units is related to the minimization of the tree size. This problem is known to be NP-complete [HR76], but Brent proposes a specific heuristic. Experimental evidences of the efficiency of this heuristic are given. Sankar and Mammone [SM90] have also proposed a new training algorithm for constructing tree structured neural networks. In this paper, we introduce two new heuristics and discuss the generalization capabilities of these learning algorithms. In order to improve these capabilities, we propose to introduce a measure of the quality of the classification. This leads to the definition of probabilistic classification trees.

2. Binary decision trees.

ESANN'1993 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 7-8-9 April 1993, D-Facto public., ISBN 2-9600049-0-6, pp. 91-96

Before studying in detail the construction of binary decision trees, we describe how a multilayered perceptron can be associated to such a tree. For simplification purpose and without loss of generality, we shall assume that the number of classes is 2. First, each node in the fan-in layer is associated to one coordinate in R^n. So there are n fan-in nodes. Second, in the first hidden layer, each node is associated to a nonterminal node of the tree. If H is the hyperplane corresponding to such a nonterminal node: $H=a_1x_1 + ...+ a_nx_n + a_0$, the weights of the node are (a_i) and the threshold is $(-a_0)$. Third, each node in the second hidden layer is associated to the path between the root and a leaf. The weigth of the connection between such a node and a node in the first hidden layer is 0 if the associated nonterminal node is not in the path, *1* (resp. *-1*) if the right (resp. left) son is in the path. The node computes the conjunction of its inputs. The fan-out layer contains only one node (logk nodes in case of k classes). The weights of the connections with the nodes in the second hidden layer are *0* if the path associated leads to a leaf labeled *0* and *1* if not. The node computes a disjunction of its inputs.

3. Learning algorithms for binary decision trees.

Let S be a finite set of training points x in R^n, each point being assigned a class $c(x)$ from a finite set C of possible classes. c is assumed to be a single valued function from R^n to C. A learning algorithm aims to construct a function χ which interpolates c on S ($\chi(x) = c(x)$ for x in S) and is able to predict the class $c(x)$ of a point x which is not in S. This second property is called the generalization capability. Using S, we construct a binary decision tree T whose nonterminal nodes are linear tests of the form $H(x) > 0 : H$ corresponds to an hyperplane which splits R^n into two halfspaces. The leaves of the tree partition R^n into polygonal regions labeled by a class. The tree is said to correctly classify S, if for any x in S, the corresponding leaf to which x belongs is labeled with $c(x)$. Several correct trees could be associated to a given training set. Brent proposes to select a tree which maximises a specific criterion related to a measure of the entropy associated with the tree. Call E this criterion (we refer the reader to [Bre90] for a precise definition of E).

First we describe the class of algorithms defined by Brent in [Bre90]. Recall that an hyperplane H is chosen for maximizing criterion E. Finding a global maximum of E is generally too difficult. Brent proposes to construct a local maximum by reducing the problem to a 1-dimensional problem. The following heuristics may be used: try the $k(k-1)/2$ possible normalized differences of centroids of two distinct classes, try a random vector of unit length, try the n possible unit axis vectors. Assume that the hyperplane is chosen parallel to the axis directions, for instance. To an hyperplane is associated a partition $\mathcal{P}$ of S. This partition is defined by the list $\mathcal{P}=(r_1,r_2,...,r_k,l_1,l_2,...,l_k)$ where r_i (resp l_i) is the number of points of class i in S on the right side of H (resp on the left side). In order to have a new partition for a given direction, H is displaced until one point moves from one side to the other and a new partition is obtained.So, for one direction of H, there are at most m possible positions of H associated to m different partitions of S. For a given partition, there are several possible positions of H between two points in S, and the algorithm has to choose this position. For instance, it may pass through the middle of the two nearest learning points. The algorithm is the following:

ESANN'1993 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 7-8-9 April 1993, D-Facto public., ISBN 2-9600049-0-6, pp. 91-96

```
Function Separate(S:learning sample) : binary decision tree
    If all the points in S belong to the same class then stop.
    Else   For all the directions computed by the heurictic do
                Choose a direction of hyperplane.
                Find the partition of S maximizing E
           Build S1= S∩ (Hmax>0) and S2= S∩ (Hmax<0).
           Tree(rightson)<-Separate (S1)
           and Tree(leftson)<-Separate(S2)
           Separate <- Tree
```

The use of the n possible unit axis vectors corresponds to a popular classification technique for symbolic learning ID3 proposed by Quinlan [Qui86]. This makes hyperplane tests cheaper but usually increases the size of the tree. Mammone and Sankar propose in [SM90] to associate a perceptron with n fan-in (one fan-in for each coordinate) and one output (*0* or *1* for the class) to each nonterminal node of the tree. This perceptron achieves a linear dichotomy of the space. The perceptron computes the classes of the learning sample points and a distance between the computed and the expected classes. Then its weights are changed with the perceptron learning algorithm in order to reduce the error of classification. As a consequence, the position of the associated hyperplane is changed, too. The algorithm returns the hyperplane of the node when some convergence criterium is reached. The learning sample is then separated as previously into *S1* and *S2* .The algorithm is recursively applied to *S1* and *S2* until all the points in the learning sample are of the same class. The algorithm is the following:

```
Function PerceptronTree(S:learning sample) : binary decision tree
    If all the poinst in S belong to the same class then stop.
    Else   Initialise weights
           Repeat
                compute the error-distance
                change the weights.
           Until convergence is reached
           Build S1= S ∩ (Hmax>0) and S2= S ∩ (Hmax<0).
           Tree(right son) <- PerceptronTree(S1)
           Tree(left son) <- PerceptronTree(S2)
           PerceptronTree(S) <- Tree
```

When the tree is built, it can be used to compute the class of any new example.The tree divides the space in $(t+1)$ polygonal regions (if t is the number of nodes in the tree). Each region is associated to one leaf in the tree and, so, to one path from the root to a leaf: the sides of the region are the hyperplanes met on the path associated and all the learning points of this area are of the same class. Hence, the classification of a new example (a point outside S) is done by traversing the tree from the root and evaluating the positions of the point with respect to the hyperplanes on the path: the point belongs to the polygonal region corresponding to the leaf and its class is the class of this leaf. This algorithm has been used for a classification problem proposed by Sejnowsky using Brent's heuristics. The problem is to classify *sonar targets* which can be *mines* or *rocks*. There are sixty parameters to analyse and we have 208

ESANN'1993 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 7-8-9 April 1993, D-Facto public., ISBN 2-9600049-0-6, pp. 91-96

examples. The algorithm runs on a fraction of this data and the classification tree is tested on the remaining examples. The learning examples are correctly classified but only 75% of the new examples.

4. Ternary decision trees.

It is known that the generalization capabilities of a multilayered perceptron depends strongly on the quality of the training. In this part, we shall propose to use ternary logic in our decision process. First we present a new algorithm based on the following principles (see Baum [Bau88] for related theoretical issues). Consider first that there are only 2 classes, labeled *0* and *1*, and divide S into S_0 (the class *0* points) and S_1. Take n points from S_0 and construct the hyperplane H through these points. If the points are not in general position, add some points until H is uniquely defined. Assuming that no class *1* point belongs to H, call δ_+ and δ_- the distances of the class *1* points nearest to H, above and below. Let H_+ (resp. H_-) the translated hyperplane $H+\delta_+/2$ (resp. $H-\delta_-/2$). The region $R(H)$ between H_+ and H_- will be given label *0*. The cardinality of $R(H)$ is the number of class *0* points in $R(H)$. If we choose at random the n points from S_0, $R(H)$ could be very small. The best choice would be to consider all the possible regions and to take the one which contains the maximum number of points of S_0. However, for practical applications, this method is too expensive. As for Brent's method, we propose to derive a new scheme by reduction to a 1-dimensional optimization problem:

```
Function MaxRegion(S0,S1:learning sample) : n-dimensional region
        Choose at random a subset Q of n-1 points in S0,
        For all the points x in S -Q do
                Using Q and x compute R(H)
                Keep R(H) of maximal cardinality
        Return R(H)
Algorithm Classif1(S0,S1:learning sample)
        While S0 is not empty do
                Classify all points of MaxRegion(S0,S1) in class 0
                S0 := S0 - MaxRegion(S0,S1)
        Classify remaining points in class 1
```

It can be shown that the computational complexity of this algorithm is polynomial. A very interesting feature of this algorithm is its dissymmetry: the classification process is centered on the *class 0* points. This could be important in the case of classes of different importance, as for example in the sonar application: it is essential that mines be correctly classified, even if there are several errors for rocks. We propose first to modify algorithm *Classif1* in order to implement it in a binary decision tree: repeat the process in the two halfspaces created by a *MaxRegion*.

```
Algorithm ClassifTree1(S0,S1:learning sample)
        If S0 is empty then return
        Else    Classify all points of MaxRegion(S0,S1) in class 0
                S0 := S0 - MaxRegion(S0,S1)
                Divide S0 into S0+ and S0- acording to H
```

ESANN'1993 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 7-8-9 April 1993, D-Facto public., ISBN 2-9600049-0-6, pp. 91-96

Divide S_1 into S_1+ and S_1- acording to H
ClassifTree1(S_0+,S_1+) ClassifTree1(S_0-,S_1-)
Classify remaining points in *class 1*

ClassifTree1 can be used in a similar way as a decision tree. Assume that the tree possesses t nonterminal nodes. These nodes correspond to *MaxRegions* and the $(t+1)$ leaves correspond to remaining regions. Hence given a new example x, traversing the *ClassifTree1* is done by testing on each internal node if x belongs to the corresponding *MaxRegion*, or to one of the halfspaces. In the former case, classify x with *class 0*. In the latter, choose the son node corresponding to the halfspace to which x belongs. If x reaches a leaf, classify x with *class 1*. Such a decision tree can be computed by a one hidden layer neural network. A couple of nodes in the hidden layer is associated to the couple of hyperplanes defining a *MaxRegion* and the outputs are 1 for both if the point belongs to the *MaxRegion*. Remark that the sum over all the nodes' outputs is 0 if the point is not in one *MaxRegion*. and is greater than 2.otherwise There is only one node in the ouptut layer computing this sum. Hence the weights are 1 and the threshold is 2.

The dissymmetry of *Classif1* has some drawback in the case of symmetry of the classes, and is difficult to generalize to more than 2 classes. In the following we propose a modification of the algorithm which is based on the introduction of a third (or k+1st) class corresponding to a *Don'tKnow* value. If the algorithm returns this value for a point x, no class will be given to x. The algorithm is the following:

Algorithm ClassifTree2(S_0,S_1:learning sample)
ClassifTree1(S_0,S_1)
ClassifTree1(S_1,S_0)
For *all points classified in both class 0 and 1* **do** change to *Don'tKnow*
Classify remaining points in *class Don'tKnow*

This algorithm could be easily extended to more than 2 classes. Its computational complexity is still polynomial. Some remarkable features of this algorithm are to be underlined: it correctly classifies all the training patterns and generalizes well on particular function classes.

5. Binary decision trees and probabilities.

In a general case with no information on the function to be learned, we do not know how reliable are the results when computing on new examples. When we apply the classification tree to compute the class of an example the algorithm has not learned, for each nonterminal node in the tree, the position of x with respect to the associated hyperplane H is computed. Since, the choosen hyperplane position is arbitrary, the answer will be arbitrary if the point is between the two learning points the hyperplane separates. We propose to introduce the probability P_H for the point to be on the left side of the hyperplane H. This probability will depend on the position of the point between the two learning points. The main requirement is that $P_H = 1$ on a training example. So, for each nonterminal node of the tree, we compute a probability to be on the left or on the right side of the associated hyperplane. Then, for each path in the tree, we compute the product of these probabilities along it. Hence we obtain the

probability for the point to be in the associated region. The sum of all the probabilities obtained for the regions of the same class gives the probability of the point to be in the class. Suppose that we can compute all the trees obtained by giving all possible positions between two points to each hyperplane of nonterminal nodes. Hence, for a new example, we can compute its class by all the trees. The fraction of *0* over the total gives the probability for this point to be in class *0*.

References

[Bau88] Baum E.B. "On the capabilities of multilayer perceptrons", Journal of Complexity 4, (1988), 193-215

[Bre90] Brent R.P "Fast Training Algorithm for MultiLayer Neural Nets", Numerical Analysis Project NA-90-03, Computer Science Department, Stanford University, (1990).

[GS88] Gorman R.P., Sejnowski T.J., "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets", Neural network, Vol 1, (1988), 75-89,

[HR76] Hyafil L., Rivest R.L., "Constructing optimal decision trees is NP-complete", Information Processing Letters, 5(1), pp 15-17, (1976).

[Qui86] Quinlan J.R., "Induction of decision trees", Machine Learning, 1,(1986), 81-106

[RM86] Rumelhart D.E., McClelland J.L., "Parallel Distributed Processing", Vol 1, MIT Press, Cambridge, Mass, (1986).

[SM90]Sankar A., Mammone R., "A Fast Learning Algorithm for Tree Neural Networks", Proceedings of the 1990 Conference on Information Sciences and Systems, Princeton, New Jersey, (1990).