CONSTRUCTIVE METHODS FOR A NEW CLASSIFIER BASED ON A RADIAL-BASIS-FUNCTION NEURAL NETWORK ACCELERATED BY A TREE

Philippe GENTRIC, Heini C.A.M. WITHAGEN
Laboratoires d'Electronique Philips
22, av. Descartes
94453 LIMEIL-BREVANNES
FRANCE

Abstract

We present a new constructive algorithm for building Radial-Basis-Function (RBF) network classifiers and a tree based associated algorithm for fast processing of the network. This method, named Constructive Tree Radial-Basis-Function (CTRBF), allows to build and train a RBF network in one pass over the training data set. The training can be in supervised or unsupervised mode. Furthermore, the algorithm is not restricted to fixed input size problems. Several construction and pruning strategies are discussed. We tested and compared this algorithm with classical RBF and multilayer perceptrons on a real world problem: on-line handwritten character recognition. While instantaneous incremental learning is the major property of the architecture, the tree associated to the RBF network gives impressive speed improvement with minimal performance losses. Speed-up factors of 20 over classical RBF have been obtained.

1 Introduction

Classification is a very general task that may be used in a great number of fields. The desired properties for a classifier are: high generalization rate, high processing speed, low memory requirement and the ability to give a likelihood measurement of some sort. Another important requirement is a short training time. Also, some tasks require an additional capability: very fast incremental learning (for example, on-line user adaptation in handwriting recognition modules for small computer).

Many of the most popular neural algorithms are only able to take into account a new information through a full re-training. If we take for example a multilayer perceptron trained by error back-propagation; even with special acceleration techniques [1] such a learning would require several passes over the training set which is simply not feasible if one is to stick to the requirements of real-world "cheap" applications.

CTRBF is a method that allows to build very rapidly a structure made of a RBF network and a tree that will be much faster than a classical RBF in resolution mode with minimal performance loss. The network is able to provide a very reliable likelihood estimate and can be easily adapted to changes (incremental learning). Furthermore, *all* the information stored in the network and the tree can be very precisely accounted for (we know exactly and declaratively the purpose of every component of the net) and consequently, the network and/or the tree can be pruned very efficiently when a smaller system is required, with control over the subsequent loss of performance.

In this paper, we will describe the tree and network structure as well as the algorithms needed to build and optimize this structure. Finally we will present a real-world application: on-line handwritten character recognition.

2 Radial-Basis-Function network

Radial-Basis-Function Networks are known to be capable of universal approximation [2] and the output of a RBF network can be related to Bayesian properties.

A RBF net has 2 layers. The hidden layer is (usually) fully connected to the input units $X = (X_i)$ of size N_{input}. A hidden unit j has a input vector of synaptic weights $Win^j = (Win^j_i)$ and is evaluated using a metric and a nonlinear function, here for example the Euclidean metric:

$$OUT_j = f(||Win^j - X||) = f(\sum_{i=0}^{N_{input}-1} (X_i - Win^j_i)^2/\sigma_j)$$

where f(x) is a function such as exp(-x) or 1/(x+a) and σ_j is an adjustable parameter. As this process has a radial symmetry of center Win^j, the output of a hidden unit j will increase when the input pattern vector X comes "closer" (according to the metric) to the synaptic weights vector Win^j .

The next layer gathers the activities of the hidden units with the purpose of taking a decision on the class of the input prototype.

For classification tasks with C classes, the output layer will have C output units. The classification result is obtained from this layer on a "winner-take-all" (WTA) basis: the class of the input pattern will be given by the most active output unit. This layer is usually made of linear units because the computation of a monotonic non-linear function adds useless computations in a winner-take-all context and/or leads to useless confusions (loss of information) in case of hard limiting function or limited precision outputs.

Assuming N_{hidden} hidden units, the activity of output unit k is:

$$OUT_k = \sum_{j=0}^{N_{hidden}-1} (Wout_j^k * f(||Win^j - X||))$$

and the resulting class $w \mid OUT_w = max_{(among\ k)}(OUT_k)$.

One of the most interesting properties of RBF networks is that they provide intrinsically a very reliable rejection of "completely unknown" patterns. Indeed, in a classical multilayer perceptron there is no guaranty that a prototype very far from any previously presented data will not produce a positive output.

Note also that the synaptic vector Win^j *stores a location in the problem space*, in other words, it stores an input pattern. It is then straightforward to imagine how (incremental) learning can be performed by creating a new hidden unit whose input synaptic weight vector will store the new training pattern. The synaptic weight from this new hidden unit to the output unit corresponding to the class of the new pattern is set to a positive value (say 1). One way to build a RBF network for a given classification problem is thus to create one hidden unit per training prototype. In real-world problems though, the number of prototypes can get very large. For this reason we developed CTRBF, a technique that allows to reduce the number of stored patterns accessed for each classification. Note that pruning techniques can also be used in order to reduce the number of hidden units (see below).

Most of the RBF literature [3] [4] is devoted to the use of optimization techniques in order to compute an optimized set of synaptic weights in the RBF network. Our experience is that for real world problems these methods demand a great amount of computations and in the end produce systems that do not perform much better than CTRBF.

3 Tree RBF

The idea of associating a tree to a RBF network is based upon the simple remark that all of the hidden units of the RBF network are not active at the same time. More precisely, only the hidden units storing a pattern that is "close" to the input pattern are active.

Consequently, when a pattern is presented to the input, instead of computing the full network, we will search which hidden units store patterns that are close to this input pattern. Then, just like a classical RBF, we will evaluate these units and propagate their activity to the output. Of course, the aim is to find these active units with as little distance computations as possible.

3.1 Tree structure

The tree structure used is binary and is illustrated in figure 1. To each branch is associated a stored pattern and a distance threshold named "radius" equivalent to σ_j. Each branch may lead to a son branch and a brother branch, and this branch is then the father of both. If the son branch exists, its radius is smaller than its father radius and the distance between a branch and any branch up the tree starting with the son of this branch is *smaller* than the radius of this branch. If the brother branch exists, its radius is equal to the father radius (hence the name "brother") and all branches up the tree starting with the brother are at a distance from the father *greater* than the father radius. If a branch has no son then it is a "leaf" and each leaf points to one hidden unit of the RBF net. If a branch has no brother then it is a terminal branch. The radius of the root branch should be bigger than the biggest possible distance between two patterns.

3.2 Tree evaluation

When an input pattern is presented we start the evaluation of the tree at its root. Then we compute the distance between the input pattern and the pattern stored by the current branch. If the distance is bigger than the branch radius, we go to the brother, otherwise, we go to the son.

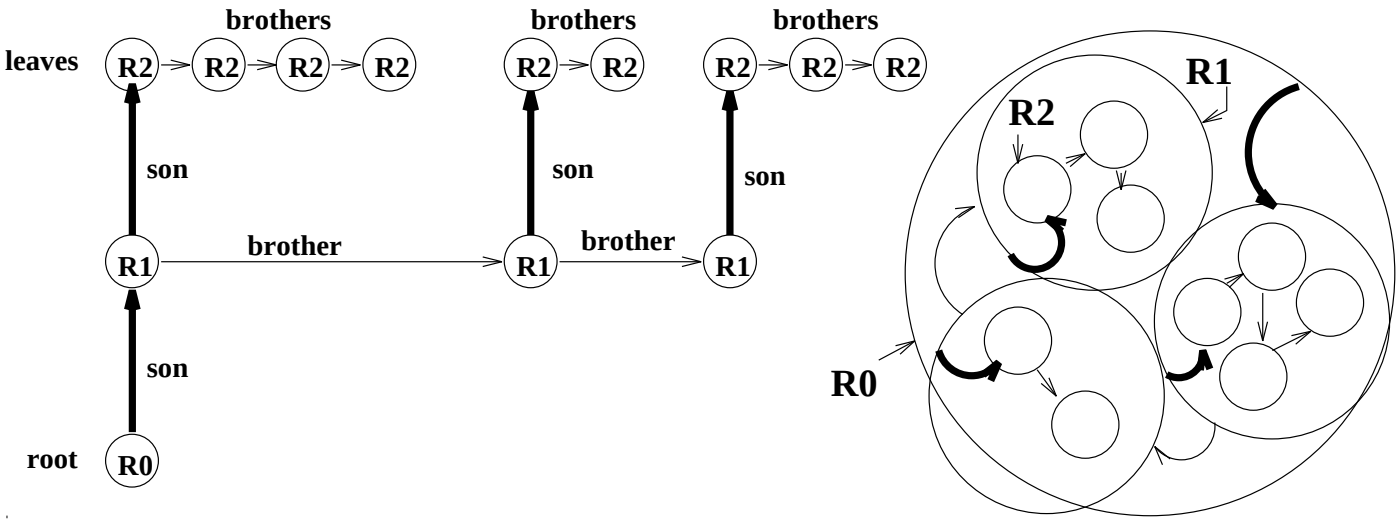

Figure 1: *Tree structure: an example of very simple tree. On the right, a representation of what the corresponding topology could be with 2 dimensional input data and a Euclidean metric*

When the evaluation of the tree leads to a leaf, the corresponding hidden unit of the RBF network is evaluated. When the evaluation of the tree leads to a terminal branch, the search is ended (see below for exception: extended search).

In order to know how much acceleration the tree gives, we compute the speed-up factor α : the number of hidden units divided by the average number of distance computations performed per prototype (tree + network distance computations); hence we have $\alpha = 1$ for a classical RBF network and $\alpha < 1$ if the tree adds computationsœ50z instead of preventing some.

3.2.1 Extending the search

We may extend the search in order to get a bigger number of selected hidden units. First, the tree may be seen as a variable-K-NN: when a leaf is reached the test (brother or son ?) is not made: all leaves along the line from brother to brother are evaluated. In other words, all hidden units storing prototype that are closer to the last branch than this branch radius are evaluated. In the following CTRBF will refer to this "basic" extended search.

Also, the search may be extended in the following fashion: when a terminal branch is reached, instead of ending the search we backtrack down the tree and at each branch we test the brother. Note that the tree is not completely explored: only branches which store patterns that are closer to the input pattern than their radius are then searched. We call this mode of extended search "backtracking".

Another way to extend the search is to force a bigger radius than the stored radius [5]. The factor by which the radii are multiplied is called the *external search factor*.

Experimentally, the results show that extending the search using these methods is efficient in terms of performance while allowing to keep a good speed-up factor α.

3.2.2 Rejection

Rejection is provided in three different ways. First, the tree search may end while no leaf was selected: this rejection mode is typical of a pattern that is "very different" from what was in the training data set. Secondly, the maximum output unit (winner of the WTA) activity may be below a given threshold : this is a likelihood rejection. Thirdly, the difference of activity between the winner and the second best is less than a second prefixed threshold: this is a confusion rejection.

4 Tree Construction algorithm

There are several ways to build the tree. Whether the class of a pattern is used or not, the construction is supervised or unsupervised.

4.1 Unsupervised learning by dichotomy

This method has the advantage of making few hypothesis on the structure of the data. We set the following constraints: the maximum number of brothers up the tree for any branch $MAX_{brother}$ is fixed (this parameter is very important, it will determine the speed-up factor α). The lowest branch radius (root radius) is fixed (the biggest possible distance between two prototypes is a good estimation) . The highest branch radius (leaf radius) is fixed (this parameter is very important, it is equivalent to the resolution of the model), it may be set by trial and error or by prior knowledge on the problem [6]. Then, build a RBF net by storing all the patterns in the training data set. Make a tree with only one branch (the root) and as many leaves as there are hidden units in the RBF net. Start at the tree root and for all the branches do the following:

1. Count the number of brothers up to the top (go from brother to brother until a terminal branch is reached). This measure of the size of the branch is a worst case estimation of the amount of computation that would be performed during tree search if this branch was reached.
2. If this number is less than $MAX_{brother}$ continue the search. When the whole tree has been tested, stop.
3. If this number is greater than $MAX_{brother}$ optimize this branch (or create an intermediate tree level) as follows: For a branch B_0, we need to keep track of 2 locations in the tree: B_i designates the son of B_0 and then the line of brothers up to the terminal. B_k designates the line of brothers created up to the terminal. Set B_i as the son of B_0:
   (a) Create a new son branch for B_0 (or a new brother along the line of brothers starting at the son of B_0) named B_k of radius R_k. R_k must be intermediate between the radius of B_i and the radius of B_0(for example take the average $R_k = (R_0 + R_i)/2$). Associate to this branch the prototype of B_i .
   (b) Find all the B_i which store prototypes closer to the prototype of B_k than R_k. Retrieve them from the line of brother B_i and insert them in the line of brother of the son of B_k. Go to 1.

4.2 Supervised learning by distribution

We start with the following constraints: all the radius hierarchy is fixed so that the lowest branch radius (root radius) is fixed, the highest branch radius (leaf radius) is fixed, the number and value of intermediate radius levels are fixed. Then proceed as follows:

1. Get at random a prototype from the training data set, until the set is empty. Use it as input pattern for an evaluation.
2. If the classification decision is correct, discard the pattern.
3. Otherwise, create a RBF hidden unit and a leaf storing this pattern with the leaf radius.
4. Find through the tree the first branch which stores a prototype that is closer to this new pattern than the its radius (this search was already done during evaluation) assign the new branch (or leaf) as a son (or a brother up the line of brothers starting with the son).
5. If no branch can be found, create (recursively up the tree) a branch(s) (leaf) storing this new pattern with radius(ii) corresponding to the son of the last found branch (root branch in the worst case).

Note that this tree and network construction mode is the mode used in case of incremental learning. Also, this is a supervised algorithm but removing the test in step 2 provides an unsupervised strategy.

4.3 Tree pruning

Branch pruning may be needed because, during construction, branches that have no brother but only a son may be created: these branch are useless and should be pruned (the connection from their father to their son is made directly). Also, after a network pruning (see below), some branch may loose all their upper branches, of course they should also be pruned.

4.4 Network pruning versus efficient construction

The most simple pruning methods are based upon the "leave-one-out" strategy: if the global system performs just as well without a given unit then this unit is not needed and must be pruned. However, this method leads to N^2 complexity (if N is the training data set size) because every hidden unit must be evaluated every time a hidden unit is tested for pruning. You may benefit from a kinetic factor: the network gets smaller as the pruning proceeds, thus less computation is needed. This is dual to the "efficient construction" obtained by distribution (see above).

Another strategy is the self-leave-one-out strategy. When a hidden unit is tested for pruning, the temptation is to perform the test with as few patterns as possible: ultimately, the unit is tested only with its own stored pattern. With CTRBF one can choose to test one hidden unit using only the tree-designated closest patterns. If α is the speed-up factor the overall complexity is then reduced to N^2/α^2.

The CTRBF architecture allows to choose any of these strategies, depending on the amount of data, time and processing power available.

4.5 Metric and *a priori* information

Very often we have *a priori* information about the problem to be solved. The possibility to incorporate such information into the classification task is known to be fundamental for the final performance. For example, in multi-layer perceptrons the *a priori* information can be built into the system using local connections, constrained weights and back-propagation of special quantities depending on known properties in the data such as translation invariance [7].

In RBF networks this kind of information can also be incorporated by changing the metric used (Euclidean in the original RBF). For example an image metric may be computed allowing local translations. Or a dynamic programming technique may be used to compute a metric between two character strings, which also allows to compute a metric between patterns that are not described with the same amount of information (in case of character strings, one can compare the word "wagon" and the word "wgon") etc..

5 Application to on-line handwritten character recognition

The handwritten characters considered have been acquired using an electronic paper interface, and pre-processed into a pattern vector having 481 components. A detailed report on this application will be published elsewhere [8]. Here we present two problems, the first problem is the "bars" problem. There are 4 kinds of bars: vertical, horizontal, slanted to the left, slanted to the right. We used 544 prototypes for learning and 544 others for testing. The second problem is a upper case character recognition problem. We used 3698 prototypes for learning and 3697 for testing. For both problems we compare the results obtained with various parameter values against the performance of multi-layer perceptrons. For the "bars" problem, the results are summarized in table 1. For the upper case characters problem, the results are summarized in table 3. One can see that the performances are still good even for α as big as 20.

In table 2 we compare the rejection ability. Here we tested the systems learned with the "bars" data (4 output classes) with the upper case letter test set: all the test patterns should be rejected. One can clearly see that the CTRBF is far superior to the multi-layer perceptron.

The evaluation times of CTRBF and perceptrons are comparable. For a CTRBF network learning and evaluation takes about the same time. On the contrary, the learning time of the perceptron is much bigger. For example the learning time of the perceptron with 50 hidden units in the second problem is far superior to all the learning and evaluation times of the tests on the CTRBF for the same problem (on a SPARC station, a few minutes against several hours). Of course, the instantaneous incremental learning is not possible with a perceptron.

6 Conclusion

CTRBF has been software-implemented in a user-adaptable on-line handwritten character recognition and runs at 3 characters per second on a 386 PC. CTRBF allows to explore constructive neural network techniques in supervised and unsupervised mode. The tree structure provides a clustering of the training data. The fact that all the information in the system can be accounted for, the intrinsic robust rejection capability, the short training time and the instantaneous incremental learning coupled with a high processing speed are the assets of this new architecture. Speed-up factors of 20 over a classical RBF can be obtained on a real world application. Furthermore, the parameters of the constructive algorithm allow to tailor the system to the application requirements. Many question are still to be answered, such as: how can we systematically build a metric taking into account *a priori* information ? Is CTRBF suitable for efficient hardware implementation ? Nevertheless, we already know that CTRBF is a very versatile new architecture for pattern classification.

References

[1] S. Makram-Ebeid, and al., A rationalized error backpropagation algorithm, *Proc. INNS*, Washington, DC, June 18-22,(1989)

[2] J.Park, I.W. Sandberg, Universal Approximation Using Radial-Basis-Function Networks, *Neural Computation* 3,246-257 (1991)

[3] E.Hartmann, J.D.Keeler, Predicting the Future: Advantage of Semilocal Units, *Neural Computation* 3,566-578 (1991)

[4] J. Moody and C. Darken, Learning with localized receptive fields, *Proceedings of the 1988 Connectionist Models Summer School*, ed. D. Touretzky, Morgan Kaufmann Publishing, San Mateo, CA, pp. 133-143, (1988).

[5] A. Saha and J.D. Keeler, Algorithms for better representation and faster learning in radial basis function networks, *Neural Information Processing Systems*, ed. D. Touretzky, Morgan Kaufmann, San Mateo, CA , 482-489, (1990).

[6] M.T. Musavi and al. On the training of radial basis function classifiers *Neural Networks* 5, 595-603 (1992).

[7] P. Simard, Y. Le Cun, J. Denker, B. Victorri, An efficient algorithm for learning invariances in adaptive classifiers. To be published in IAPR.

[8] P. Gentric, On-line handwriting recognition for small computer. Submited to the Sixth International Conference on Handwriting and Drawing, Paris July 5-6-7, (1993).

network used	branches	hidden units	error rate	rejection rate	α
CTRBF	149	544	1.1 %	9.2 %	33
idem + backtracking	149	544	2.0 %	1.5 %	10
idem + external search factor 1.2	149	544	0.9 %	1.1 %	6.2
idem + external search factor 2	149	544	1.1 %	0.6 %	2.7
idem + external search factor 10	149	544	1.5 %	0.6 %	0.9
normal RBF	1	544	1.5 %	0.6 %	1
1 layer perceptron (no hidden neurons)		0	1.1 %	1.1 %	
2 layer perceptron (10 hidden neurons)		10	1.8 %	1.2 %	

Table 1: Compared performances on a 4 classes problems: the bars. One can see that a speed-up factor from 5 to 10 does not affect the performance. Also the perceptrons do not perform much better although (especially the two-layer perceptron) they are much slower to learn.

network used	error rate	rejection rate	α
CTRBF + backtracking + external search factor 1.2	4.4 %	95.6 %	33
2 layer perceptron (10 hidden neurons)	45 %	55 %	

Table 2: Compared performances on the 4 bars problems: rejection capability, this test was performed using upper-case data (3697 prototypes): on this problem the error rate cannot go much below 6 % because there are 240 'I' among which many that may be confused with a vertical bar. This is a clear confirmation of the superiority of classifiers based on distance-to-pattern over classifiers based on distance-to-linear-separator

network used	branches	hidden units	error rate	rejection rate	α
CTRBF	1099	3698	7.9 %	25.5 %	200
idem + backtracking	1099	3698	12.5 %	4.1 %	42
idem + external search factor 1.2	1099	3698	5.4 %	0.8 %	21
idem + external search factor 2	1099	3698	4.3 %	0.5 %	5
idem + external search factor 10	1099	3698	4.3 %	0.6 %	1
normal RBF	1	3698	4.3 %	0.6 %	1
1 layer perceptron (no hidden neurons)		0	10.4 %	12.3 %	
2 layer perceptron (10 hidden neurons)		10	16 %	20 %	
2 layer perceptron (50 hidden neurons)		50	7.8 %	8.6 %	

Table 3: Compared performances on a 26 classes problems: the upper case letters. As the problem size and number of classes grow the size of the totaly connected