Asymptotic performance of a constructive algorithm

Florence d'Alché-Buc and Jean-Pierre Nadal*
Laboratoires d'Electronique Philips
B.P. 15, 22 avenue Descartes - 94453 Limeil-Brévannes Cedex France
e-mail: florence@lep.lep-philips.fr

Abstract

We present a numerical study of a neural tree learning algorithm, the "trio-learning" strategy. We study the behaviour of the algorithm as a function of the size of the training set. The results show that a limited number of examples can be used to estimate both the network performance and the network complexity that would result from running the algorithm on a large data set.

Appeared in *Neural Processing Letters*
Sorry: this is a version without the figures...

*Permanent address: Laboratoire de Physique Statistique, Ecole Normale Supérieure, 24, rue Lhomond - 75231 Paris Cedex 05 France

Asymptotic performance of a constructive algorithm

Abstract

We present a numerical study of a neural tree algorithm, the "trio-learning" strategy. We study the behaviour of the algorithm as a function of the size of the training set. The results show that a limited number of examples can be used to estimate both the network performance and the network complexity that would result from running the algorithm on a large data set.

1 Introduction

In the context of supervised learning, progress has been made in the recent years allowing for efficient learning strategies with multilayer perceptrons. As an alternative approach to learning with, say, backpropagation algorithms applied to a multilayer perceptron of a fixed architecture, constructive algorithms have been developed. In particular, there has been a recent interest in "neural trees"[1, 2, 3], which can be seen as generalizations of decision trees [4, 5]. In the present letter we are concerned with the estimation of the performance of such constructive algorithms.

As for any learning strategy, one should test the consistency and validity of the algorithm by asking basic questions. First, the rates of success measured on the training set and on the test set should extrapolate to the same asymptotic value when the size of the data set goes to infinity. This is obvious from the theoretical point of view, but a given learning strategy (even when the architecture of the network is fixed in advance) may not satisfy the conditions assumed in the theoretical studies. Next, one would like to know how typical are the results obtained in a test of the algorithm done on a given application with a fixed data set. Again, a relevant indication is the behaviour of the algorithm as the size of the data set is increased. Finally, one would like to make use of this behaviour in order to estimate the possible performance of the algorithm if run on a much larger data base. For the case of a multilayer perceptron trained with, say, backpropagation, theoretical studies have been used to extract very useful information from scaling experiments [6, 7].

At least for us, it is not clear how these and related results apply or not to constructive algorithms. The main purpose of the present work is therefore to test a particular constructive algorithm, namely the trio-learning strategy [8, 9], in terms of its behaviour as the size of the training set increases. We will consider the scaling of the percentage of success as a function of the size of the training set. We will also pay attention to the dependence of the tree complexity on the data size.

In the section 2 we will first shortly introduce the particular neural tree algorithm we are using. Then in section 3 we will present the numerical simulations, and we will discuss the main results in the last section.

2 Building a hybrid tree with the trio strategy

We consider a particular neural tree algorithm that we introduced recently [8], and which is related to the hierarchical mixture of experts proposed in [10]. Our algorithm is based on a recursive learning strategy called "trio-learning" [8]. The idea is as follows.

The data are a set of patterns, each one belonging to one of two classes, and defined as a vector in some space (the *data* or *input space*) typically of large dimension. At step zero, one try to discriminate the two classes with a single and simple classifier of a given family. It may be any neuron-like unit such as a linear separator or a radial basis function, or even a small neural

Figure 1: 1a: A hybrid tree architecture. The last built trio is singled out. 1b: Classification induced in data space, with hyperplanes as separators and radial basis functions as classifiers.

network. If the resulting performance are not good enough, one replaces this classifier by a *trio* composed of a separator and two new classifiers: the separator, e.g a hyperplane implemented by a standard formal neuron, cuts the space into two domains; for each one, one classifier processes the data that fall into that domain (see Figure 1). This procedure is repeated iteratively: within each domain, one evaluates the classification success of the classifier according to some a priori chosen criterion; if the latter is not good enough, one replaces this classifier by a new trio: the domain is cut into two new domains, and at the same time in each one a new classifier is introduced. The parameters of a trio, that is the set of parameters defining the separator (the hyperplane) together with the two new classifiers, are computed with a supervised learning scheme, typically a gradient method based on an appropriate cost function.

The algorithm we have shortly described is thus building a hybrid tree, that is a binary decision tree composed of two kinds of nodes, *internal* nodes (the separators - the linear separators in Figure 1), and *terminal* nodes (the classifiers - the radial basis functions in Figure 1).

Whereas the trio-learning strategy improves the building of a neural decision tree in terms of complexity, it is still necessary to prune the tree after building in order to avoid overfitting[4]. The pruning algorithm that we use aims at finding the subtree of the tree generated by trio-learning, that gives the best classification rate on the test set. It requires to store the parameters and the classification rate obtained at each node of the current tree before that node is removed and replaced by the father node of a new trio. We will see, in the numerical illustration presented in the next section, how crucial the role of pruning is.

3 Asymptotic rates

We illustrate our approach on a particular difficult binary discrimination problem of handwritten characters, "H" versus "M". In the framework of dynamic coding each character is coded as a vector of dimension 20, corresponding to the 2D coordinates of a sequence of ten points taken along the drawing. The dimension of the input space is thus 20. We run the algorithm on training sets of sizes $S = 271, 339, 453, 679, 1018$, using the $1359 - S$ remaining examples for the test set. In each case we limited arbitrarily the growth of the tree to a maximal depth of 7. For each experiment, we measured the performance both before and after pruning. The mean success rate is shown on Figure 2 as a function of $\frac{1}{S}$, the inverse of the size of the training set. On Figure 3, we show the mean depth of the tree (each branch depth being weighted by the number of examples classified by this branch). A first remark is that, without pruning, it is not possible to reasonnably extrapolate the success rates measured on the training set: it is not even clear that the performance on the training set and on the test set extrapolate to a common value. Moreover, the mean depth of the tree increases with the size of the data base, obviously converging to 7, the maximal allowed depth. On the contrary, we observe an interesting behaviour after pruning. Quite importantly, the mean depth of the tree seems to saturate, at a value around 5.5. Although we have large error bars, the results are consistent with a linear convergence of the success rate as a function of $1/S$, with a common limit for $S \rightarrow \infty$. The scaling of the success rate with S is similar to the one obtained with a perceptron of a fixed architecture [6, 11].

The results presented above have been obtained with a hybrid-tree built with radial basis

Figure 2: Success rate *versus* inverse of data size S, before and after pruning. Each point is an average over 10 experiments. For clarity, the error bars are only shown for the rates after pruning. The straight lines are from a joint regression over the test and training set rates obtained after pruning.

Figure 3: Mean depth of the network *versus* inverse of data size S, before and after pruning.

functions as terminal nodes. We made similar experiments with other classifiers: linear separators (standard McCulloch and Pitts neurons) and quadratic (second order) neurons. In each case we obtained results similar to those presented on Figures 2 and 3. The success rates (after pruning) for the three choices of classifiers are presented on Figure 4. One sees that the performance depend drastically on the choice of the classifier, being much better for second order neurons than for the two other choices. Then, it will not come as a surprise that the mean depth of the tree is shorter with second order neurons (with a saturation at about 2.5) than with radial basis functions.

We presented our results on one particular task, but we obtained similar results for other choices of pairs of characters, in particular "A" versus "R", "S" versus "5", "n" versus "r" and "1" versus "7". All these numerical experiments have been performed with a relatively small number of examples, leading to large error bars (as shown on Figure 2). However, in each case, and for each choice of classifier, a very clear linear scaling is observed for the mean success rates (after, and only after, pruning), together with a saturation of the mean depth of the tree.

4 Conclusion

The main consequence of the numerical simulations presented in this letter is that one can expect a consistent behaviour of a constructive learning strategy: we observe that the extrapolated value at infinite size is consistent with the one obtained from the performance on the training set; moreover the performance in generalization increases linearly with the inverse size of the data set. This is very similar to what has been obtained in numerical and theoretical studies of networks of a given, fixed, architecture[6, 11].

One should stress that the consistent behaviour is obtained *only* if the growth of the network is well controlled. In the present work we made use of a pruning strategy in order to control the growth of a neural tree. As a result, one observes that the network complexity, as measured by the mean depth of the tree, saturates at a finite value: the structure of the tree tends to adapt itself to the complexity of the task. We expect that similar results would be obtained with alternative approaches - such as the use of a cost function taking into account the tree complexity [12].

A practical consequence is that a limited number of examples can be used to estimate *both* the network performance and complexity that would result from running the algorithm on the full data set - or conversely, one can estimate the number of examples and the size of the network that would be needed in order to achieve a given performance. From this point of view, the particular algorithmic strategy that we have tested can be very valuable: by varying the choice of the classifier (used at the terminal nodes of the tree), one can find the one which will

Figure 4: Success rate (after pruning) *versus* inverse of data size S for three choices of classifiers.

lead to the best performance for the particular application considered (in the particular case discussed in the previous section, quadratic neurons gave much better performance than radial basis functions or linear separators).

Acknowledgements

This work originated from a collaboration with Didier Zwierski. It is a pleasure to thank him for stimulating discussions.

References

[1] J. A. Sirat and J.-P. Nadal, Neural Trees: a New Tool for Classification, NETWORK Vol.1, pp. 423-438, 1990.

[2] M. Golea and M. Marchand, A growth algorithm for neural network decision trees, Europhys. Lett. Vol. 12, pp. 105-109, 1990

[3] M. Frean, The Upstart Algorithm: a method for constructing and training feedforward neural networks, Neural Computation, Vol. 2, pp. 198-209, 1990.

[4] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. Stone. Classification and regression Trees. Wadsworth International Group, Belmont, CA, 1984.

[5] J.R. Quinlan, Induction of Decision Trees, Machine Learning, Vol.1, pp. 81-106, 1986.

[6] Cortès, C., Jackel, L. D., Solla, S., Vapnik, V., and Denker J. S (1993). Learning curves: asymptotic values and rates of convergence. In *Neural Networks for Computing Conference, Snowbird, Utah.*

[7] V. Vapnik, E. Levin, Y. Le Cun, Measuring the VC-Dimension of a learning machine, Neural Computation, Vol. 6, pp. 851-876, 1994.

[8] F. d'Alché-Buc, D. Zwierski and J.-P. Nadal, Trio learning: a new strategy for building hybrid neural trees, *Neural Networks for Computing (Snowbird)*, 1993, and LEP preprint 1994 submitted to IJNS

[9] F. d'Alché-Buc, J.-P. Nadal and D. Zwierski Hybrid trees for supervised learning: towards extraction of decision rules *Proc. ECAI'94 Workshop on Combining Symbolic and Connectionist Processing (Amsterdam)*, pp. 133-142, 1994

[10] M. I. Jordan and R. A. Jacobs Hierarchical mixtures of experts and the EM algorithm, *Neural Networks for Computing (Snowbird)*, 1993, and Neural Computation, Vol. 6, pp. 181-214, 1994.

[11] S. I. Amari, N. Murata, Statistical theory of learning curves under entropic loss criterion, Neural Computation, Vol. 5, pp. 140-153, 1993.

[12] J. Rissanen, *Stochastic complexity in statistical inquiry*, World Scientific, Teaneck, New Jersey, 1989.