BACKPROPAGATION AND MONTE CARLO ALGORITHMS FOR NEURAL NETWORK COMPUTATIONS

R. JUNCZYS AND R. WIT*

Institute of Physics, Jagellonian University
Reymonta 4, 30-059 Kraków, Poland
e-mail address: wit@ztc386a.if.uj.edu.pl

Results of teaching procedures for neural network for two different algorithms are presented. The first one is based on the well known backpropagation technique, the second is an adopted version of the Monte Carlo global minimum seeking method. Combination of these two, different in nature, approaches provides promising results.

(Received July 12, 1996)

PACS numbers: 89.80.+h

1. Introduction

Neural networks are quite useful when solving some practical problems [1]. They have also found many new applications. In particular, neural networks may be used to identify the ancestor of a hadron jet in e^+e^- high energy collisions [2]. Already with one hidden layer (*cf.* Fig. 1) one may attack problems which seem to be rather complicated. Therefore effective methods for 'training' multilayer neural networks are of certain importance.

In this paper we would like to present the results of our numerical experiments concerning weights' calculations for a given neural network. In addition to the standard *backpropagation* algorithm we introduce a *Monte Carlo* one related to the clustering technique [3]. It can be easily implemented and connected to other subroutines in use and yields encouraging results.

The function of a neuron in an artificial neural network is to sum its weighted input signals and to present the output according to the *activation*

* Supported in part by KBN grant No 2 P03B 196 09.

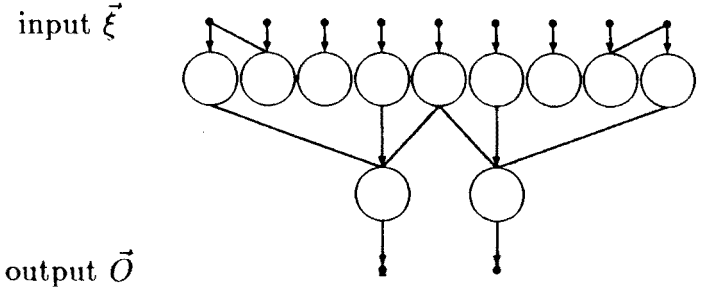

Fig. 1. A simplified structure of a neural architecture. For clarity not all connections are drawn.

function:

$$n_i = f(\sum_j w_{ij} n_j).$$

In this way we describe node's activity [4]. The weight $w_{\alpha\beta}$ represents the synapse connection strength between neuron α and β; for convenience we collect the $w_{\alpha\beta}$'s in one vector denoted by $\vec{w}$.

The neural network training procedure consists of three obvious steps:

1. the feed forward of the input pattern,
2. the backpropagation of the resulting error,
3. the iterative change of the relevant weights.

The error function is defined in the following way

$$E : R^N \longrightarrow R_+ : E(\vec{w}) = \sum_c \sum_i (O(\vec{w})_i^c - \zeta_i^c)^2 . \tag{1}$$

The subscript c labels elements of the teaching sequence, the subscript i describes elements of the output vector.

The global error is normalized by the factor

$$\delta(\vec{w}) = \frac{E(\vec{w})}{\sum_c \sum_i 1} . \tag{2}$$

Such normalization is an appropriate one since the single unit (neuron) maximal error is equal 1 and here we are summing over all possible errors.

In this paper we use the following activation function $f(h)$

$$f(h) = \frac{1}{1 + \exp(-\beta \cdot (h - \mu))} , \tag{3}$$

which has an obvious sigmoidal structure (logistic sigmoid).

The (\$-shaped) diffused step functions are well suited for use in neural dynamics algorithms due to a simple relationship between their values and the values of their derivatives. There exists some freedom in taking the values of the parameters β and μ. They influence the convergence rate of the teaching procedures and this is the main limitation for their possible variation. The value $\mu = 0.5$ was fixed during our numerical experiments.

There are many update rules for weights' change. One of the best algorithms used in this case is based on the *backpropagation* technique, closely related to the *gradient steepest descent* method taken from minimisation theory. The backpropagation algorithms differ in some technical details but the underlying theory remains the same.

The main problem of the training procedure is connected with the structure of the weight space, which very often looks like that given in Fig. 2. Once on the flat plateau the searching algorithm, when based on gradient calculations, does not get valuable hints where and how far to move to obtain the requiered weight changes.

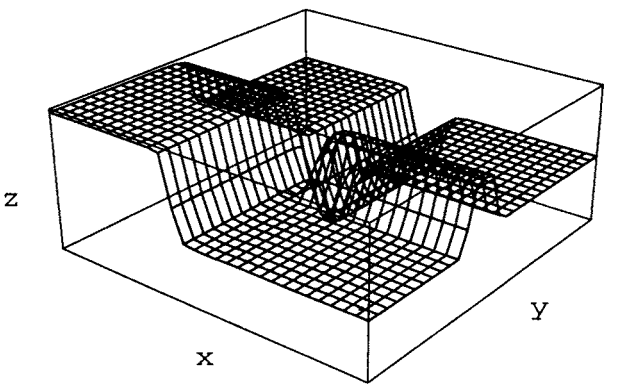

Fig. 2. Typical weight space structure of the error function.

Another difficulty is related to *local* and *global* minima. Roughly speeking, the minimum finding calculations may get stuck either somewhere on the plateau or at some local minimum, far away from the global one. Repeated calculations may clarify the situation (different starting point, different searching strategy *etc.*) but this is certainly not a very systematic approach.

An entirely different attitude to the minimum finding procedure is provided by the Monte Carlo global optimization technique, introduced by Törn [3]. We explain its essence using the following numerical example,

which serves also as a very good cross-check for our procedures and relevant links among them.

2. Numerical example

Consider the following function $g : R^N \longrightarrow R_+$:

$$g(\vec{x}) = \sum_{i=1}^{N} (\text{int})|x_i - a_i| \sum_{i=1}^{N} (\text{int})|x_i - b_i| \times \sum_{i=1}^{N} (\text{int})|x_i - c_i| \sum_{i=1}^{N} (\text{int})|x_i - d_i| \,. \tag{4}$$

By (int) we denoted the integer part of the subsequent simple expression. The appropriate parameters are described in the following way:

dimension: $N = 2$;
minima: $\vec{a} = (5.0, 5.0)$; $\vec{b} = (5.0, -5.0)$; $\vec{c} = (-5.0, 5.0)$; $\vec{d} = (-5.0, -5.0)$;
boundaries for independent variables: $x_1 \in (-10, 10)$; $x_2 \in (-10, 10)$.

The structure of this function is presented in Fig. 3. The four symmetric minima are well seen.

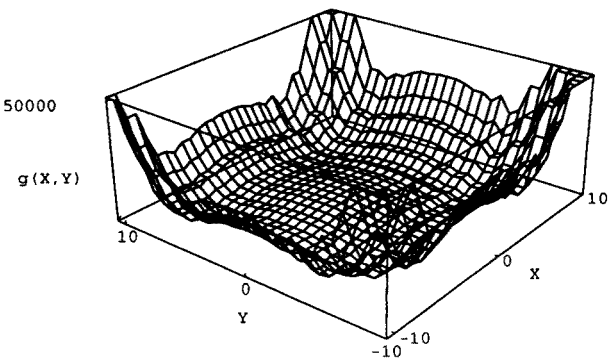

Fig. 3. The structure of the function $g(\vec{x})$ (*cf.* (4)).

Considering a general case assume that no *a priori* information about the location of minima is available. One starts the Monte Carlo minimizing procedure by *scattering at random* the so called *seed points* within certain boundaries (*cf.* Fig. 4). Next, we move each of them towards a local minimum. The procedure used for this purpose consists of two steps for each point:

1. find a random direction in which the minimum searching calculations should proceed,
2. use a (crude) linear minimisation procedure to move the seed points towards some minimum along the chosen direction,

and a general stopping criterion

1. stop when some point density condition is satisfied.

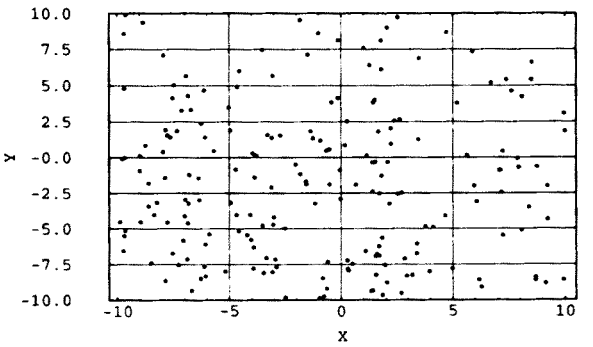

Fig. 4. The location of the starting seed points.

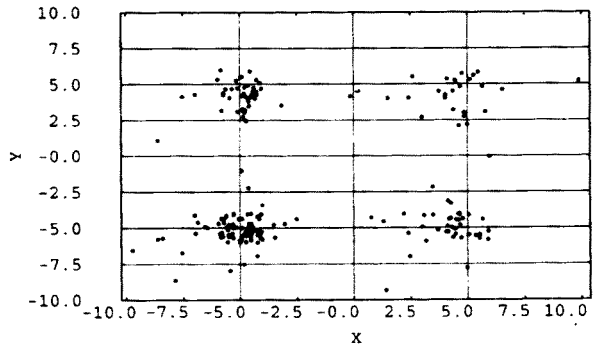

Fig. 5. The location change of the starting points.

The results concerning function $g(\vec{x})$ are presented in Fig. 5. We observe a *cluster* formation and therefore it is enough to take only *one* representative of each cluster to perform further calculations. Obviously the cluster

technique is used to prevent multiple determination of a local minimum and repetitive, unnecessary calculations.

From now on we may either use again the Monte Carlo minimum finding method or we may switch, *e.g.*, to the backpropagation algorithm. If corresponding computing resources are available one may initialize at this stage *parallel* calculations for each starting point representing the above mentioned clusters.

A further remark connected with a cluster structure seems to be worth mentioning. The weight choice describing a well defined (deep) minimum of the error function may not be a good one if we expect also certain flexibility from our network. Therefore Monte Carlo calculations could be a useful tool when generalization properties of a neural network are of great value.

3. XOR and pattern recognition test problems

In this section we would like to present some of the results obtained when solving the XOR [4] and simple pattern recognition problems by means of the backpropagation and seed points (Monte Carlo) methods. The standard network structure for solving the XOR problem is given in Fig. 6.

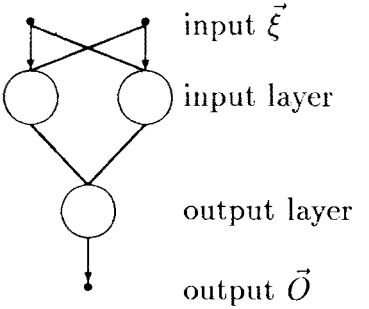

Fig. 6. Standard network architecture for the XOR problem.

With a given starting point $\vec{w_0}$ and $\beta = 10$ (what corresponds to a sharp sigmoidal activation function) we see in Fig. 7 that there is no error reduction when using the backpropagation algorithm. However, we observe a real change if the calculations are performed by means of the Törn prescription [3] (*cf.* Fig. 8).

We see, however, that if for the same starting point we change the shape of the activation function putting $\beta = 5$ we get some improvement for the backpropagation algorithm. This is seen in Fig. 9 and to some extend could be expected: for smaler β values the activation function is smoother.

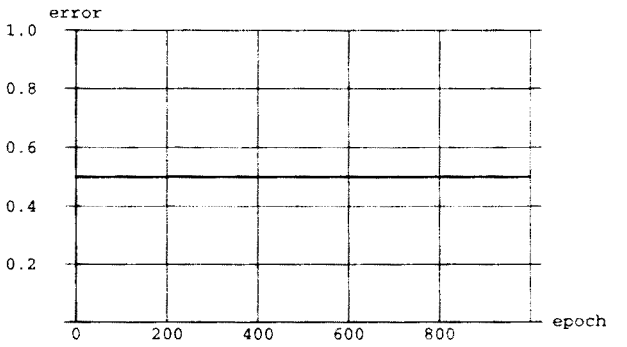

Fig. 7. Solution to the XOR problem — $\beta = 10$. For a given starting point the backpropagation does not work.

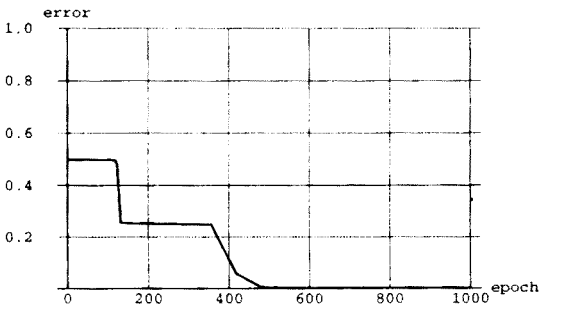

Fig. 8. Solution to the XOR problem — $\beta = 10$. the Törn seed points method.

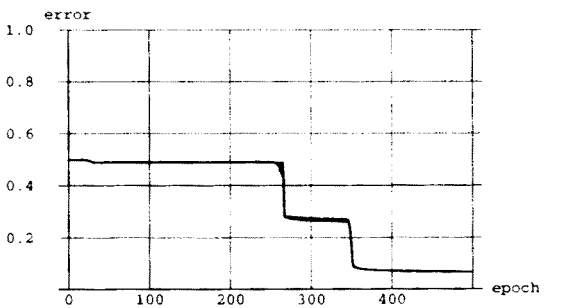

Fig. 9. Again the *XOR* problem — this time with $\beta = 5$. Backpropagation works better.

It is, however, evident that we have obtained the best results when starting with the Monte Carlo calculations and then switching to the back-

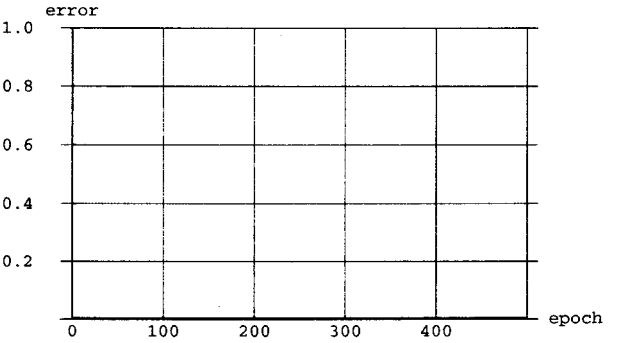

Fig. 10. Start with Monte Carlo to solve the XOR problem ($\beta = 10$) and continue by backpropagation. The result is excelent.

propagation algorithm. The corresponding numerical results are presented in Fig. 10.

Fig. 11. Simple 3×3 patterns

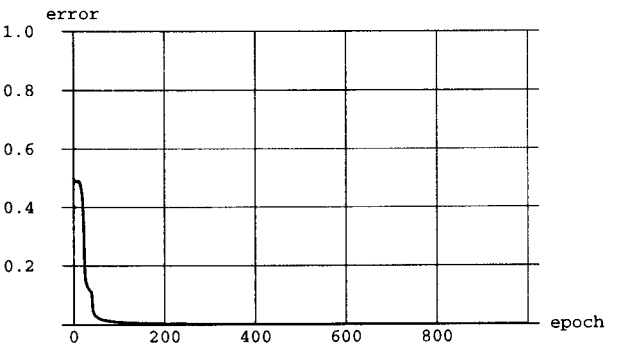

Fig. 12. Pattern recognition — backpropagation.

Another example we would like to discuss here is a simple pattern recognition problem (see Fig. 11). The backpropagation method (Fig. 12) works now better than the Törn method (Fig. 13) but the combination of these two (Fig. 14) leads again to encouraging results.

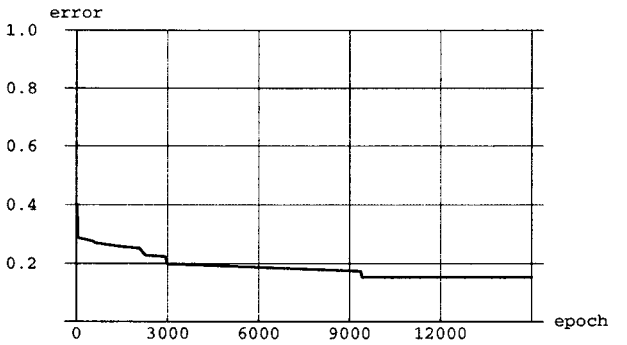

Fig. 13. Error change for the pattern recognition problem using the Törn algorithm.

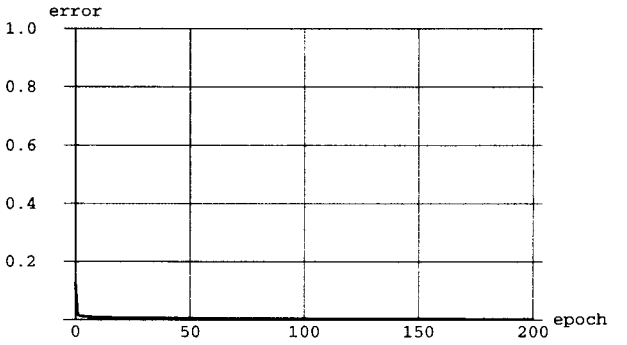

Fig. 14. Pattern recognition — hybrid solution method.

4. Final conclusions

As we have shown the inclusion of Monte Carlo technique into the weights calculations for neural networks provides one with quite interesting perspectives and results.

All the examples we discussed above are, of course, low dimensional ones. Programs, written in $C++$, were well adjusted to be used on *PC's*. Dealing with more complicated neural networks we should expect additional problems connected with the 'curse of dimensionality'. In particular we feel that an introduction of a more elaborated (but not time consuming) procedure for the directional minimisation would be necessary. Also a different organisation of our numerical procedures could be unavoidable.

What seems, however, to be attractive in the presented approach is that right from the beginning one tries to get here a general look at the structure of local minima (metastable memory points [5]). This sort of information is often of great value during the planning stage of a neural network architecture. A possible inclusion of parallel calculations might also be interesting.

One of the crucial elements in the Monte Carlo algorithm is a proper choice of the relevant cluster analysis technique. Here we may gain a lot of computational effort. The cluster analysis, by itself an important part of scientific activities in different areas (*cf.* [6]), when properly applied to neural network calculations may lead to further improvements.

At the end we would like to emphasize that the positive examples we presented here should be considered as some hints for further investigations rather than an indication for something closed and definite.

We are grateful to J. Wosiek for reading the manuscript of this paper. A. Burzyński was very helpful fixing some software problems we met when performing this work.

REFERENCES

[1] S. Haykin, *Neural Networks, A Comprehensive Foundation*, Macmillan College Company, Inc., New York 1994.

[2] L. Lönnblad, C. Peterson, T. Rögnvaldsson, *Nucl. Phys.* **B349**, 675 (1991).

[3] A.A. Törn, *A Program for Global Optimization. Multistart with Clustering (MSC)*, in IFIP Proceedings, ed. P.A. Samet, North-Holland, 1979, p.427.

[4] L. Fausett, *Fundamentals of Neural Networks*, Prentice Hall, Englewood Cliffs, NJ 07632 1994.

[5] J. Hertz, A. Krogh, R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City CA 1992.

[6] *STATISTICA*, StatSoft, Tulsa OK 1995.