DIRECTED RANDOM SEARCH FOR MULTIPLE LAYER PERCEPTRON TRAINING

Udo Seiffert and Bernd Michaelis
University of Magdeburg
Institute for Electronics, Signal Processing and Communications
P.O. Box 4120, 39016 Magdeburg, Germany
http://iesk.et.uni-magdeburg.de/

Abstract. Although Backpropagation (BP) is commonly used to train Multiple Layer Perceptron (MLP) neural networks and its original algorithm has been significantly improved several times, it still suffers from some drawbacks like being slow, getting stuck in local minima or being bound to constraints regarding the activation (transfer) function of the neurons. This paper presents the substitution of backpropagation by a random search technique which has been enhanced by a directed component. By means of some benchmark problems a case study shows general potential application fields as well as advantages and disadvantages of both the Backpropagation and the Directed Random Search (DRS).

1 INTRODUCTION

The *Multiple Layer Perceptron (MLP)* [1] is probably the most frequently used network architecture among supervised trained artificial neural networks [2]. It is an extension of the original *Perceptron* architecture which is able to train the output elements to classify input patterns, providing the classes are linearly separable. More complex non-linearly separable classes can be handled with perceptrons arranged in several consecutive layers.

The most important problem is to determine the weights of all neurons in all layers by means of the global network error derived from the difference between actual and desired network response to a given input stimulus. This is also known as the *Credit Assignment* problem [1]. After the invention of a suitable training algorithm called *Backpropagation* ([3], [4]), multiple layer perceptrons have proved to be able to solve a great variety of real-world problems.

A typical MLP (see Fig. 1) always has an input buffer to present an input stimulus, at least one hidden layer which is not accessible from outside the net and finally an output layer to present the network response. Usually each layer, except the output layer, is fully connected to the succeeding one. Backpropagation assumes that all neurons and weighted connections are partly responsible for an erroneous response of the entire network. The level of this responsibility is affixed by propagating the

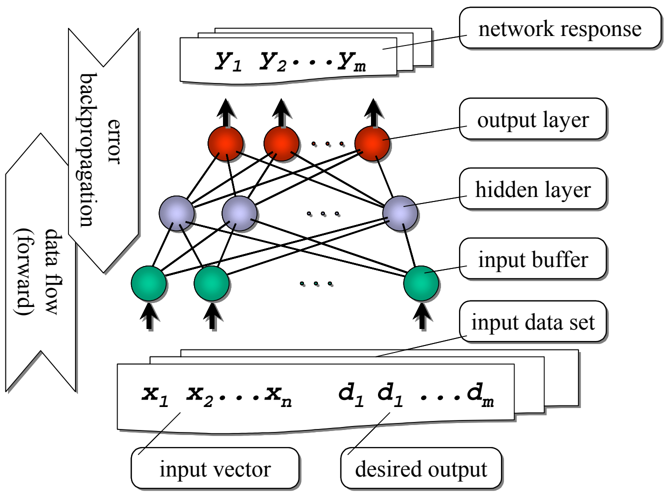

Fig. 1. Typical topology of a Multiple Layer Perceptron. The input vector is processed forward to the output layer. Based on the difference between desired output and actual network response a global error is computed. A local error for each weight is determined using the weighted connections backward.

global error backward layer by layer using the weighted connections until the weights of the first hidden layer are reached. This global error function uses the difference between actual and desired network response to a given input stimulus. In most cases a quite simple *Root Mean Square (RMS)* error is utilized.

Since the error is passed back using all forward connections including the neuron's activation function, this function has to meet some special requirements. It must be steady and differentiable in order to make the computation of its derivative possible. This restricts the available functions. To simulate the biological original, basically a step function, or its smooth mathematical equivalents, either the sigmoid function for unipolar outputs or the hyperbolic tangent for bipolar outputs are used [5].

Now all network weights are updated[1] based on this local error. At the initialisation phase of the network all weights are chosen randomly. The general aim of the learning phase is to minimize the global error for each example of the training data set. Usually a gradient descent algorithm, controlled by a few further parameters, i.e. learning rate, is utilized to determine the weight change. Now the next training vector is presented to the net via the input layer and is propagated forward to the output layer, the global error is computed and propagated backward again to determine an

1. The computed weight change is either executed instantly or cumulated until all input examples are presented to the network. This is often called *batch learning*.

appropriate weight change. This loop is executed until an a-priori defined stopping criterion is met.

2 MOTIVATION

Numerous extensions and modifications [6] have been suggested to improve the training results or to achieve some desired properties of the trained MLP network. These focus mainly on:

- ❑ acceleration of the convergence speed (i.e. *fast-backpropagation*),
- ❑ special learning rules and data representation schemes (i.e. *cumulative delta-rule*, *cascade* and *batch learning*),
- ❑ different error functions (i.e. *cubic* instead of standard *RMS error*),
- ❑ alternative activation functions of the neurons (within the frame of the above mentioned requirements only),
- ❑ numerically faster implementation (i.e. *parallel computing*),
- ❑ weight distribution (i.e. *weight pruning*)

and some more. Meanwhile a few of them even became 'standard' themselves. One key element on which Backpropagation is based, namely the gradient descent algorithm (see Fig. 2) to minimize the network error, has been changed scarcely, although there are sometimes really strong reasons to look for a substitute.

The first and probably most evident problem using Backpropagation is the fact, that the network often gets stuck in a local minimum rather than reaching the desired error threshold. Once trapped in a minimum crater, which has a bottom above the specified global error threshold, there is often no other way out than jogging the weights[1] or even restarting the complete training. This may be very time consuming and in the worst case a desired threshold may not be reached at all. If the network enters a plateau, the gradient becomes very flat, which slows down the training progress. This happens mainly on a low-dimensional error surface, because it is not very likely that a plateau is spread out over many dimensions. Fig. 3 illustrates the most common problems of Backpropagation.

For many applications the available standard activation functions (sigmoid and hyperbolic tangent) are undoubtedly sufficient. But sometimes the wish for an alternative one arises. The objective is either to simulate the original biological operation more closely or to achieve a special network behaviour [5]. This often fails because of the demand of Backpropagation for steadiness and differentiability of the utilized activation function.

1. A relatively small random number is added to each weight to move the current position of the network on the error surface. In a successful case the net hops out of the minimum crater.

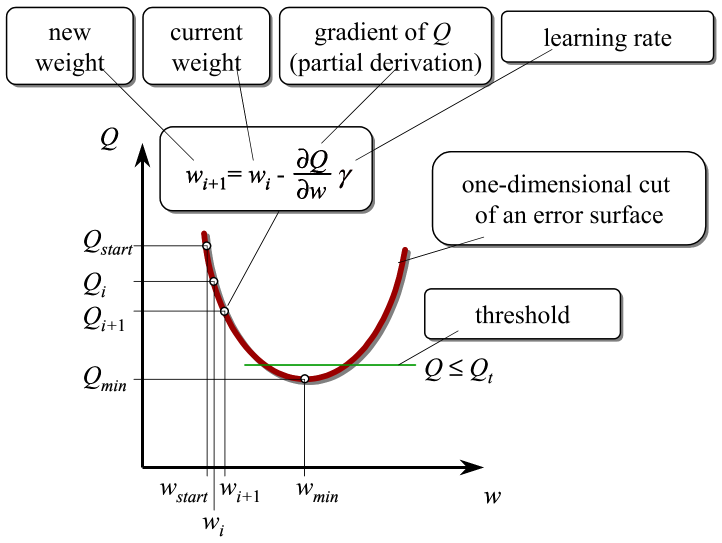

Fig. 2. Gradient descent by means of a simple one-dimensional error surface (only one weight). Beginning at an initial point (marked with index *start*) the aim of the learning process is to minimize the global error Q (indexed with i as it proceeds) until it falls below a given threshold Q_t.

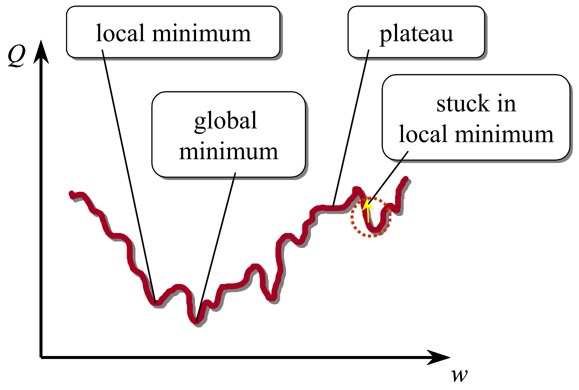

Fig. 3. Typical problems of the gradient descent algorithm caused by the existence of local minima and plateaus with a very flat gradient.

One of the most promising approaches to substitute Backpropagation is to apply genetic algorithms to optimize the weights and sometimes even the topology of the entire network [7]. As investigations have shown, especially large networks are likely to be captured in local minima, because their error surface is very high-dimensional. Since genetic algorithms perform a global rather than a local minimum search, they have an advantage over BP in particular for larger networks ([8], [9]).

However, genetic algorithms are quite slow, especially for big populations. Sometimes it is tricky to get them properly working. They require a certain degree of expert knowledge to chose appropriate parameters. It is true that genetic algorithms have proven to be a well working alternative to BP, but they are difficult to handle.

There are a few more training algorithms based on several strategies, i.e. decision trees. However, these methods seem to be inappropriate to be used in real-world applications, because they are either hard to treat or produce insufficient results.

3 RANDOM SEARCH TECHNIQUE

3.1 Overview

The discussion in the previous section shows that it is still interesting to look for alternative training methods for MLPs. Commonly the majority of training algorithms for neural networks uses a combination of random and calculus based parts. In general, training a neural network can be viewed as an optimization problem.

Matyas reported a *Random Optimization Method (ROM)* in the context of standard functional approximation in 1965 [10]. He suggested to take random steps in the search space. After each step the function to be minimized is evaluated. If the error has decreased, the new position is accepted. Otherwise the previous position is kept and from there another random step is performed. It can be demonstrated, that over a compact data set, ROM is guaranteed to converge to the global minimum [11]. In order to improve the performance of the random optimization, some extensions and modifications have been suggested ([10], [12], [13]).

The Random Optimization Method has been successfully applied to neural network training by Baba [14] for the first time. The following section describes the application of an enhanced random optimization technique to the training of MLPs in detail.

3.2 Multiple Layer Perceptron Training

Like in Backpropagation, the starting point is the global network error

$$E = \sum_{t=1}^{T_d} \sum_{j=1}^{N_o} (d_{tj} - y_{tj})^2 \qquad (1)$$

computed from the difference between the desired output d_{tj} and the actual network response y_{tj} for all neurons N_o of the output layer and all examples T_d of the training data set. It is assumed, that all weights of the network are initially set.

Random Step. A random value, which is usually drawn from an uniform or Gaussian distribution, is added to each weight w of all existing layers

$$w_{i+1} = w^* + \Delta w_i \,, \tag{2}$$

where i is the iteration counter and w^* refers to the stored best weight achieved by a previous iteration. Now the global network error is calculated according to Eq. (1). If its value E_{i+1} is less than the stored one E^* for the previous best set of weights, then the new weight w_{i+1} is retained as the new best weight w^*. Otherwise the current best weights are kept unchanged (Fig. 4, left).

Self-Adjusting Variance. Solis proposed a self-adjusting variance σ of the random distribution to adapt the size of the steps taken in the weight space [13]. This method can also be applied to the MLP training. Starting from an initial size of the variance an adaptation algorithm updates σ according to the history of success of the recent steps. If several consecutive successful steps have been recorded, σ is increased to allow larger steps. Otherwise σ is decreased to reduce the step size. Since the self-adjusting variance has no deep impact on the results of the training, it is not further explained.

Reversal Step. If a particular random step is not successful and leads only to an increased error, its reversal step may guide to a decreased error. According to Eq. (2), a reversal step is defined by *subtracting* the *same* random value from the best weight and is a simple and very fast operation. Of course, a reversal step is only performed after an unsuccessful random step. If the reversal step is lucky and the network error decreases, these weights are retained as the new best weights.

Directed Component. Up to this point, the algorithm is solely based on random steps. See (Fig. 4) for a flow chart illustrating this part of the algorithm. However, even in the original Random Optimization Method a directed component was introduced by Matyas [10]. Depending on the progress of previous successful or failed random steps (either forward or reversal), a directed component may further enhance the search. Extending Eq. (2), a third term, the directed component Δd, is added to the weight update

$$w_{i+1} = w^* + \Delta w_i + \Delta d_i \,. \tag{3}$$

The directed component forces the search in directions where successful steps have been recorded previously. It is initialised to 0, and updated after each iteration in

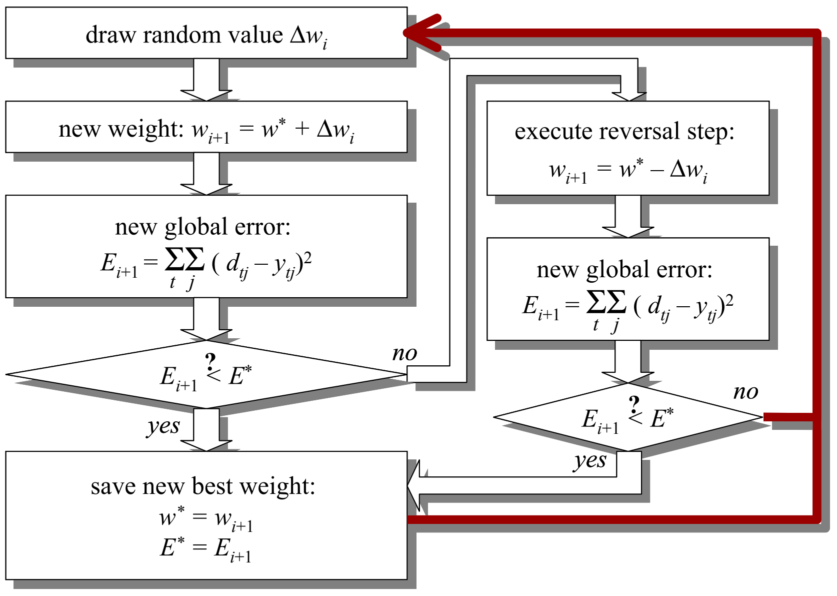
draw random value Δw_i
new weight: $w_{i+1} = w^* + \Delta w_i$
new global error: $E_{i+1} = \sum_t \sum_j (d_{tj} - y_{tj})^2$
$E_{i+1} \overset{?}{<} E^*$
no
yes
save new best weight: $w^* = w_{i+1}$ $E^* = E_{i+1}$
execute reversal step: $w_{i+1} = w^* - \Delta w_i$
new global error: $E_{i+1} = \sum_t \sum_j (d_{tj} - y_{tj})^2$
$E_{i+1} \overset{?}{<} E^*$
no
yes

Fig. 4. Flow chart of the random components of the Directed Random Search applied to Multiple Layer Perceptron training. Depending on the result of the random step, a reversal step is performed. If the global network error decreases, either after a random or a reversal step, the weights of this iteration are retained as new best values.

dependence on the results and the size of the previously taken steps according to the following equation. Refer to the next chapter for a discussion of the coefficients c.

$$\Delta d_{i+1} = c_1 \cdot \Delta d_i + c_2 \cdot \Delta w_i \qquad (4)$$

4 RESULTS

4.1 Benchmark Data Set

In order to keep the results comparable to other training methods, for the present some frequently referred benchmark problems have been used, such as the well-known *Iris* data set [15] and the *Abalone* and *Dermatology* data from [16].

4.2 Properties of the DRS Training

This section describes the results applying the DRS based training to the Iris data set and the influence of varying key parameters.

The directed component has turned out to be very important and should be used in every case (for detailed results see below). It accelerates the convergence noticeably by forcing larger steps to previously successful directions. The coefficients (Eq. (4)) for the update rules have to be chosen according to the results of the previous random resp. reversal steps. It has turned out that the DRS algorithm is not very sensitive to slightly modified coefficients. However, the general tendencies, such as positive or negative reinforcement, must take effect.

Tab. 1: Coefficients for the update of the directed component depending on the results of the previous steps.

Condition	c_1	c_2
successful forward step	0.1 … 0.3	0.2 … 0.6
successful reversal step	1	-0.2 … -0.6
both steps unsuccessful	0.4 … 0.6	0

As to be seen in Fig. 5, the best results are obtained using both the directed component and reversal steps together with a sigmoid activation function (d). For this network the RMS error after 100 epochs (7,500 iterations) is 0.0084, which is absolutely sufficient for this problem. As also clearly to be seen, the directed component improves the training result considerably (a), in this example by factor 10 $(0.074 \to 0.008)$.

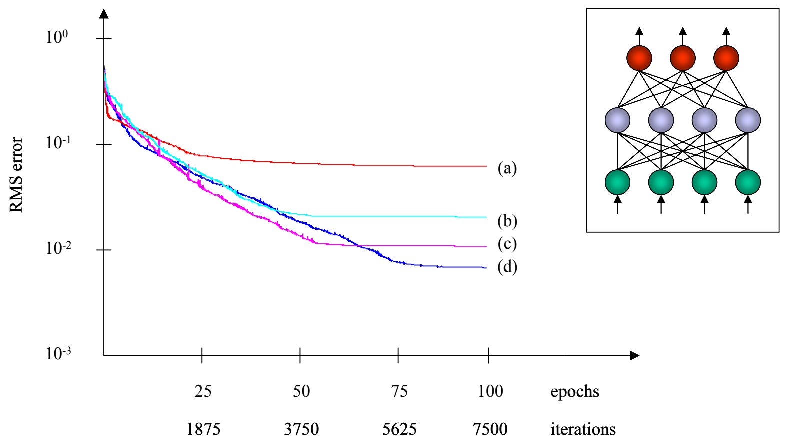

Fig. 5. Training of a Multiple Layer Perceptron (4 input, 4 hidden, 3 output neurons; see inlet) using BP with delta rule and sigmoid activation function (c); and DRS with sigmoid activation function, directed component and reversal steps (d). Based on (d), (a) is without directed component and (b) is without reversal steps.

In contrast to these results, a Backpropagation trained network (c) of the same size needs at least 400 epochs to attain this error. After 100 epochs it reached only 0.013. It occasionally gets stuck in local minima (not shown in Fig. 5) and has to be restarted then with a different initial weight set. In this context it has to be noted that one iteration of BP is much faster than one complete iteration of the DRS. The calculations to perform the random steps and to handle the directed component are very fast, but after each step the objective function has to be evaluated by performing a network recall for the entire data set. In the end this takes longer than the gradient descent of BP. Since BP needs in all more iterations to reach the desired error, in the long run it may be slower, to say nothing of the danger not to arrive there at all. The advantages of the DRS get more and more lost as the network size increases.

Tab. 2: Comparison between 2 common Backpropagation variants and the Directed Random Search for 3 benchmark data sets of different size. Mean value for 10 independent runs each.

Data set	Network size	Training method	Relative training time to reach the same RMS error	Reached RMS error at the same training time
Iris [15]	4 - 4 - 3 28 weights	Std. Backprop Quickprop DRS	1 0.82 0.68	0.05 0.0517 0.0406
Abalone [16]	7 - 5 - 1 40 weights	Std. Backprop Quickprop DRS	1 0.79 0.73	0.08 0.0721 0.0658
Dermatology [16]	34 - 15 - 1 525 weights	Std. Backprop Quickprop DRS	1 0.91 1.33	0.09 0.141 0.167

5 CONCLUSION

This paper has demonstrated the implementation details of a Directed Random Search to train Multiple Layer Perceptrons. One characteristic feature of the DRS training algorithm is that only a few parameters have to be arranged. These parameters are not very sensitive. This makes it easy to maintain the network training and to achieve reliable results. The additional implementation of a directed component is very useful. It accelerates the training in our investigations by factor 4…13.

The DRS typically needs less iterations than Backpropagation to attain a desired error. One DRS iteration takes longer, because after each search step the objective function has to be evaluated. This is usually done by performing a network recall with all examples of the training resp. test data set.

The results demonstrated by means of some benchmark problems could also be confirmed using other data sets. The DRS technique will surely never replace BP, not even for small nets, because in many cases the well-known and commonly used BP works very well. But it should be kept in mind as a promising and easily controllable alternative to BP, especially for moderately sized nets and when BP once again does not lead to satisfactory results.

REFERENCES

[1] M. Minsky and S. Papert, **Perceptrons: An introduction to computational geometry**, Cambridge: MIT Press, 1969.

[2] R.P. Lippmann, "An introduction to computing with neural nets," **IEEE ASSP Magazine**, vol. 4, no. 87, pp 4-23, 1987.

[3] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning internal representations by error propagation," Rumelhart, D.E. et al. (eds.): **Parallel distributed processing: Explorations in the microstructure of cognition**, Cambridge: MIT Press, 1986, pp. 318-362.

[4] D.B. Parker, "Learning-logic," **Report No. TR47**, Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science, 1985.

[5] M.I. Jordan, "Why the logistic function? A tutorial discussion on probabilities and neural networks," Computational Cognitive Science Report No. 9503, Massachusetts Institute of Technology, 1995.

[6] The Neural Network FAQ, ftp://ftp.sas.com/pub/neural/FAQ.html#questions.

[7] A.J.F. van Rooij, L.C. Jain and R.P. Johnson, **Neural network training using genetic algorithms**, Singapore: World Scientific, 1996.

[8] U. Seiffert and B. Michaelis, "On the gradient descent in Backpropagation and its substitution by a genetic algorithm," **Proceedings of the IASTED International Conference on Applied Informatics**, 2000, pp. 821-826.

[9] U. Seiffert, "Multiple Layer Perceptron training using genetic algorithms," **Proceedings of the European Symposium on Artificial Neural Networks**, 2001, pp. 159-164.

[10] J. Matyas, "Random optimization," **Automation and Remote Control**, vol. 26, pp. 246-253, 1965.

[11] N. Baba, T. Shoman and Y. Sawaragi, "A modified convergence theorem for a random optimization method," **Information Sci**, vol. 13, pp. 159-166, 1977.

[12] G. Schrack and M. Choit, "Optimized relative step size random searches," **Mathematical Programming**, vol. 10, pp. 230-244, 1976.

[13] F.J. Solis and R.J.-B. Wets, "Minimization by random search techniques," **Mathematics of Operations Research**, vol. 6, no. 1, pp. 19-30, 1981.

[14] N. Baba, "A new approach for finding the global minimum of error functions of neural networks," **Neural Networks**, vol. 2, pp. 367-373, 1989.

[15] B.G. Batchelor, **Practical approach to pattern recognition**, New York: Plenum Press, 1974.

[16] C.L. Blake, C.L. and C.J. Merz, C.J., **UCI Repository of machine learning databases** [http://www.ics.uci.edu/~mlearn/MLRepository.html], Irvine, CA: University of California, Department of Information and Computer Science.