ESANN'2001 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 25-27 April 2001, D-Facto public., ISBN 2-930307-01-3, pp. 159-164

Multiple Layer Perceptron Training Using Genetic Algorithms

Udo Seiffert

University of South Australia, Adelaide
Knowledge-Based Intelligent Engineering Systems Centre (KES)
Mawson Lakes, 5095, Adelaide, Australia
udo.seiffert@unisa.edu.au

Multiple Layer Perceptron networks trained with backpropagation algorithm are very frequently used to solve a wide variety of real-world problems. Usually a gradient descent algorithm is used to adapt the weights based on a comparison between the desired and actual network response to a given input stimulus. All training pairs, each consisting of input vector and desired output vector, are forming a more or less complex multi-dimensional error surface during the training process. Numerous suggestions have been made to prevent the gradient descent algorithm from becoming captured in any local minimum when moving across a rugged error surface. This paper describes an approach to substitute it completely by a genetic algorithm. By means of some benchmark applications characteristic properties of both the genetic algorithm and the neural network are explained.

1 Introduction

Multiple layer perceptrons (MLP) [1] commonly trained with *backpropagation* (BP) [2], [3] are very frequently used to solve a great variety of real-world problems. Among the group of supervised trained networks this paradigm can be considered meanwhile as the standard architecture. That's why numerous extensions and modifications have been suggested to improve the results or to achieve some required properties of trained nets. This focuses mainly on

- the 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 transfer (activation) functions of the neurons,
- weight distribution (i.e. *weight pruning*)

and some more. One key element, backpropagation is based on - the gradient descent algorithm to minimize the network error - has been changed scarcely.
Usually a gradient descent algorithm is used to adapt the weights based on a comparison between the desired and actual network response to a given input stimulus. All training pairs, each consisting of input vector and desired output vector, are forming a more or less complex multi-dimensional error surface. Some suggestions have been made to prevent the gradient descent algorithm from getting stuck in local minima when moving across a rugged error surface. This paper describes an approach to substitute it completely by a genetic algorithm.

ESANN'2001 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 25-27 April 2001, D-Facto public., ISBN 2-930307-01-3, pp. 159-164

2 Motivation

Despite its popularity as an optimization tool, not only for neural network training, the gradient descent has several drawbacks. It is dependent on the shape of the error surface, the starting point (values of the usually randomly initialised weights) and some further parameters. A common error surface has many local minima usually not meeting the desired convergence criterion.
Of course there are some approaches to prevent the gradient descent algorithm from becoming stuck in any local minimum when moving across the error surface. Some of them, being frequently used, are the introduction of a *momentum term* (useful for points of high curvature) and the utilization of a *synchronized decreasing learning rate*. Once the system is stuck in any local minimum, the weights can be *jogged*; a small random offset is added before the next iteration continues. However, all this is not able to really overcome the many existing problems.
Recently some investigations into neural network training using GAs have been published. Often only selective problems with their particular solution are in the focus of attention. This paper demonstrates a more general way.
Genetic algorithms (GA) [4] offer an efficient search method for a complex problem space and can be used as powerful optimization tools. Potential solutions (single individuals resp. chromosomes in terms of genetic algorithms) to a problem compete with each other in order to achieve increasingly better results. With regard to the above mentioned problems of the gradient descent a complete substitution of it by a GA might be advantageous [5]. Besides the neural network design optimization using genetic algorithms [6], [7], [8] this is another very interesting and biologically motivated symbiosis of artificial neural networks and evolutionary computation.

3 Implementation

It would go beyond the scope of this paper to explain the basic terminology, implementation and features of genetic algorithms. Therefore a basic knowledge is assumed and only those aspects concerning the neural network training are mentioned.
In order to get an appropriate chromosomal representation of the network's weights [9], these have to be randomly initialised multiple times and accordingly coded into a linear structure. Each chromosome (individual) represents one neural weight set (see Figure 1). Since the architecture of the neural network is predefined and remains fixed after the initialisation, the chromosome solely consists of the weight values and does not contain any topological or structural information. All these individuals represent the initial population.
Now the fitness function of every member of the current population is computed by performing a recall of the network for each weight set. In order to match the network's topology, all chromosomes have to be decoded again. If there is a recall result satisfying the required quality criterion, defined by the overall network error, the process stops. Otherwise, genetic operations using the reproduction and / or selection operator are executed to form an intermediate population. Finally a new population is generated by applying crossover and maybe mutation to this intermediate population. A new iteration cycle starts again with the computation of the fitness functions.
By selecting suitable parameters to control the GA, high efficiency and good performance can be achieved. The neural network is run only in recall mode which can be easily implemented and guarantees a rather simple adaptation to any demands on its behaviour made by the task to be

ESANN'2001 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 25-27 April 2001, D-Facto public., ISBN 2-930307-01-3, pp. 159-164

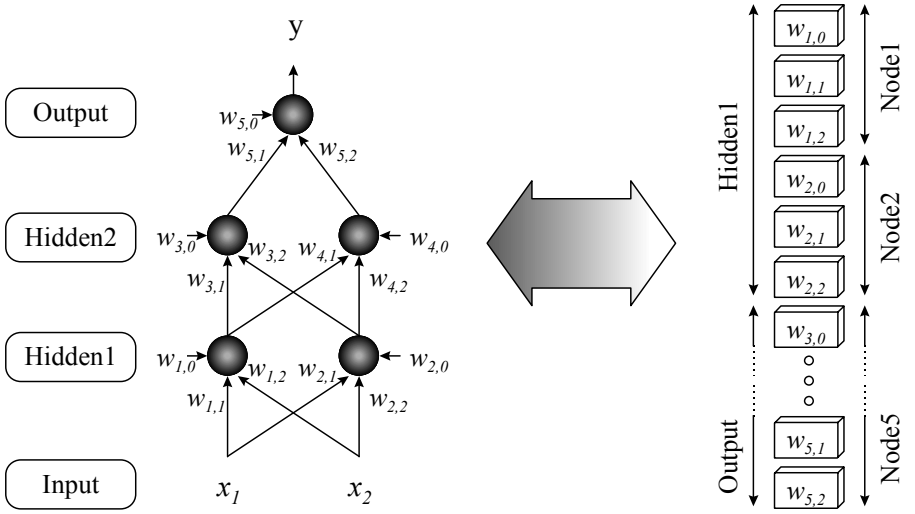

Figure 1: Mapping the weights of the neural network from problem space (left hand side) into a chromosome (representation space) and v.v. by means of a network with two inputs, two hidden layers with two neurons (nodes) each and an one neuron output layer. Including the bias, a total amount of *15* weights is coded into the linear chromosome. The order of weights within the chromosome is chosen arbitrarily, but must be left unchanged while the network is trained.

solved. In the long run at least one individual should meet the stopping (convergence) criterion. It represents the weight set of the best trained network. This can be applied to the actual problem just as it had been trained by backpropagation or any other learning sequence. From now on it does not matter how the network was actually trained.

4 Results

4.1 A simple XOR

In order to have the results comparable to other studies, particularly to conventionally (with BP) trained MLPs, at first some frequently referred benchmark problems have been applied to evaluate performance and properties of the evolutionary trained net. For illustration purposes the well-known *exclusive-or* (xor) problem has been selected. Based on a multitude of experiments evaluating different parameters to control the genetic algorithm, a standard set has been defined (see Table 1). In order to be able to solve the two-bit xor,

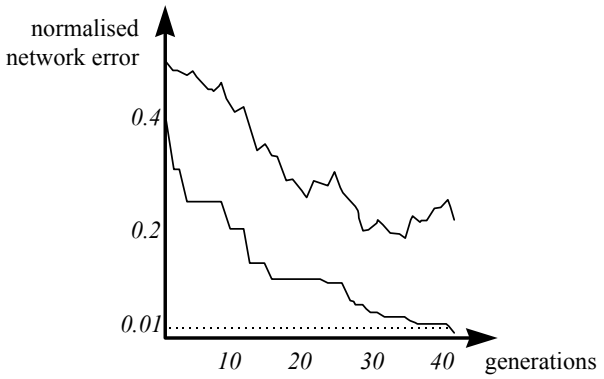

Figure 2: Training progress by means of normalised average network error (upper line) and network error of the fittest individual (lower line). The desired network error to stop training is marked with a dotted line.

ESANN'2001 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 25-27 April 2001, D-Facto public., ISBN 2-930307-01-3, pp. 159-164

the network size has been set to *2* inputs, *2* neurons in the hidden and *1* in the output layer with a *sigmoid* transfer function. Initially *50* weight sets have been created and coded into corresponding chromosomes. The stopping criterion is an *or*-combination of the network error during recall and a maximum number of generations. The *normalised average network recall error* (mean value of the average recall errors using the complete training data set (*4* examples) from all *50* chromosomes) of the initial population is about *0.5*. The fittest individual of the initial population has an error of about *0.4*. After *42* generations the error falls below the desired threshold and the training is stopped at an error of *0.0061*.

Figure 2 shows the training progress of this selected characteristic example. This result has to be compared with a conventionally trained network of the same size and under comparable circumstances (computation speed, implementation details). Of course, numerous networks with varying parameter combinations as well as with identical parameters and only different weight initialisation to reduce the role of chance have been run. The applied *save best* strategy sets fair conditions for the backpropagation and can be compared with the elitism scheme in genetic algorithms. Doing this leads to the following basic statements.

- Neither backpropagation (after a few restarts) nor the genetic algorithm experienced problems solving this particular test problem.
- The backpropagation net needs significantly more iteration cycles (ratio approx. *50:1 ... 100:1*, depending on initialised weights).
- Each single iteration runs considerably faster than one generation of the evolutionary algorithm (ratio approx. *1:20 ... 1:50*).
- It follows that the GA finally has a speed lead of max. *5:1* over the backpropagation. There are also cases yet, where backpropagation outperforms the genetic algorithm by *2:1 ... 4:1*.

Table 1: Standard parameter set used for reference training.

Parameter	Value
transfer function of the neurons	tanh
population size	pop_size = *50*
weight initialisation routine	Gaussian; mean = *0.0*; std = *5.0*
stopping criterion	network_error = *0.01*; max_iter = *500*; conv = *20*; eps = 10^{-4}
fitness normalization	norm; rank
selection operator	roulette
elitism	elit = *1*
crossover	p_c = *0.8*; two-point; uniform;
mutation	p_m = *0.1*; Gaussian; mean = *0.0*; std = *1.0*

The great range of these numbers indicates that all given values are strongly dependent on some initially set values and the actually used implementation, i.e. influence of numerical details, source code, compiler. This must not be undervalued while reading and comparing the numbers.

4.2 Larger networks

This simple *xor* example demonstrates basic properties and handling of the evolutionary MLP. It could be expected yet, that advantages become even more evident when dealing with larger net-

works and more complex problems. The more weights to optimize during the network training, the higher is the dimension of the corresponding error surface. Along with then often more rugged error surfaces the risk to become stuck within a local minimum increases. Since a genetic algorithm does not suffer from this, it should still reasonably perform with larger nets.
In order to test this, several more complex benchmark problems have been used. Among them are for instance *continuous xor*, the extension of the binary xor to continuous values and the *Iris* flower classification based on [10] with *4* inputs, *3* outputs and *75* training examples.
For the first two tests, the results are similar to the previous ones. Actually there was no problem to find an appropriate solution, neither for the MLP nor for the GA. Since the trained nets are larger, the computation time has increased for both methods proportionally. In other words, these problems are not large enough to challenge the tested algorithms.
In order to really challenge the evolutionary MLP algorithm, several data sets have been built by extracting features and statistical parameters from a three-dimensional image data set of a confocal microscope [11]. The length of input vectors varies within a range from *5* to *13*, with several hundred training examples each. Backpropagation (trained in standard configuration with *momentum term*) encountered some serious problems and was in most cases not able to meet the desired error criterion. There was no further error decrease, even if the networks were allowed to run for much more iterations. Obviously the networks were stuck in local minima. The analysis of the weights, achieved from several runs with the same parameters except for different initialised weights, reported different values for each trial, what supports this theory too.
In contrast to BP, the GA based nets have been able to solve these tasks successfully and reached the required error. The genetic algorithms continue to decrease the recall error if run for more generations. This supports the assumption that GAs do not suffer from becoming stuck in local minima of the error surface. However, the consumed resources, particularly computation time, increase exponentially, because longer chromosomes require additionally larger populations.
On the other hand, BP always required various (*5...10*) starts with different weight initialisation to find a path from its given starting point, defined by the randomly initialised weights, to a desired minimum. The higher the demands on the error threshold to be reached, the more problems have been faced. Keeping this in mind, the lead on the genetic algorithm caused by faster execution of each BP iteration is destroyed. Thus the utilisation of a GA based network seems to be the easier and more promising approach, when dealing with large and rather complex problems.

5 Conclusions

The symbiosis of artificial neural networks and genetic algorithms is not really a new idea. Nevertheless, using genetic algorithms in neural network training is still worth being investigated, particularly if some drawbacks of neural network training can be avoided. Actually the backpropagation algorithm for training multiple layer perceptrons sometimes encounters some serious problems and its complete substitution by a genetic algorithm may be a very promising solution. The main questions this paper was focusing on is under which circumstances might this be useful and how can a suitable implementation be achieved and handled.
Assuming both backpropagation and genetic algorithms are run with nearly optimal parameters under comparable conditions, backpropagation outperforms genetic algorithms in case of simple standard applications. Considering all aspects, including simulation time, finding and handling

parameters, necessary previous knowledge and software availability, there is no need to substitute the well-tried and reliable backpropagation training algorithm.
But as soon as problems to be solved are getting more complex, backpropagation more and more fails due to its inherent gradient descent. Backpropagation needs in these cases several starts with varying initial weights to meet the desired error or is not able to solve the required task at all, while genetic algorithms are still performing very well.
On the field of genetic algorithms the results given in this paper show the important dualism between the length of the chromosome, depending on the number of weights to be coded, and the average number of necessary generations. The longer the chromosomes the more generations are required. In general, genetic algorithms are inherently slower than backpropagation. This could be expected due to their global search technique compared to the highly directed gradient descent learning of backpropagation. But even this makes it valuable and helpful in those cases BP fails.
Besides this, the chance to use any transfer function for the neurons, without being bound to the demand for a differentiable function, it might also be advantageous to consider the substitution of backpropagation by a genetic algorithm.
The numbers given in this paper are strongly dependent on many simulation conditions. Other problems under different circumstances may lead to slightly different results. The basic effects should be always the same. Comparing this work with related work supports this conclusion.

References

[1] M. Minsky and S. Papert, *Perceptrons: An introduction to computational geometry* (Cambridge MA.: MIT Press, 1969).
[2] 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 MA.: MIT Press, 1986), 318-362.
[3] D.B. Parker, Learning-logic, *Report No. TR47, Massachusetts Inst. of Technology, Center for Computational Research in Economics and Management Science*, 1985.
[4] J.H. Holland, Adaptation in natural and artificial systems, *University of Michigan Press*, 1975.
[5] A.J.F. van Rooij; L.C. Jain and R.P. Johnson, *Neural network training using genetic algorithms* (Singapore: World Scientific, 1996).
[6] J. Branke, Evolutionary algorithms for neural network design and training. *Technical Report No. 322, University of Karlsruhe, Institute AIFB*, 1995.
[7] P. Robbins; A. Soper and K. Rennolls, Use of genetic algorithms for optimal topology determination in back propagation neural networks, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms* (Vienna: Springer, 1993), 726-730.
[8] P.J. Angeline; G.M. Saunders and J.B. Pollack, An evolutionary algorithm that constructs recurrent neural network, *IEEE Transactions on Neural Networks*, 5(1), 1994, 54-65.
[9] K. Balakrishnan and V. Honavar, Properties of genetic representations of neural architectures, *Proceedings of the World Congress on Neural Networks* (Mahwah NJ: Lawrence Erlbaum / INNS Press, 1995), 807-813.
[10] B.G. Batchelor, *Practical approach to pattern recognition* (New York: Plenum Press, 1974).
[11] A. Herzog; G. Krell; B. Michaelis; J. Wang; W. Zuschratter and K. Braun, Three-dimensional quasi-binary image restoration for confocal microscopy and its application to dendritic trees, *Lecture Notes in Computer Science 1296* (Berlin: Springer, 1997), 114-121.