

Discovery of Optimal Backpropagation Learning Rules Using Genetic Programming

Amr Radi and Riccardo Poli
School of Computer Science
The University of Birmingham
Birmingham B15 2TT, UK
E-MAIL: {A.M.Radi,R.Poli}@cs.bham.ac.uk

1998

Abstract

The development of the backpropagation learning rule has been a landmark in neural networks. It provides a computational method for training multilayer networks. Unfortunately, backpropagation suffers from several problems. In this paper, a new technique based upon Genetic Programming (GP) is proposed to overcome some of these problems. We have used GP to discover new supervised learning algorithms. A set of such learning algorithms has been compared with the Standard BackPropagation (SBP) learning algorithm on different problems and has been shown to provide better performances. This study indicates that there exist many supervised learning algorithms better than, but similar to, SBP and that GP can be used to discover them.

1 Introduction

Supervised learning algorithms are by far the most frequently used methods to train artificial neural networks. The Standard BackPropagation (SBP) algorithm represents a computationally effective method for the training of multilayer networks which has been applied to a number of learning tasks in science, engineering, finance and other disciplines. The SBP learning algorithm has indeed emerged as the standard algorithm for the training of multilayer net-

works, against which other learning algorithms are often benchmarked [5, 26].

The standard backpropagation algorithm suffers from a number of problems [20, 18, 19, 22, 4, 5]. The main problems are: it is extremely slow, training performance is sensitive to the initial conditions, it may be trapped in local minima before converging to a solution. oscillations may occur during learning (this usually happens when the learning rate is high), and If the error function is shallow, the gradient is very small leading to small weight changes. This is known as the “flat spot” problem.

For these reasons in the past few years a number of improvements to SBP have been proposed in the literature. They can be divided into five main groups as discussed in [20].

We will review the SBP rule and some of the recent improvements in Section 2.

All these algorithms are generally faster than the SBP rule (up to one order of magnitude) but tend to suffer from some of the problems of SBP. Efforts continue in the direction of solving these problems to produce faster supervised learning algorithms and more importantly to improve their reliability. However, the progress is extremely slow because any new rule has to be imagined/designed firstly (using engineering and/or mathematical principles) by a human expert and then has to be tested extensively to verify its functionality and efficiency. Also, most algo-

rithms newly proposed are not very different from or much better than the previously known types, as scientists tend to search the space of possible learning algorithms for neural nets by using a kind of “gradient descent”. This way of searching may take a long time to lead to significant breakthroughs in the field. Indeed, by looking critically at the huge literature on this topic, it can be inferred that only a few really novel algorithms which demonstrated much better performance than SBP have been produced in the last 10 years [9, 3, 25, 16, 14, 1].

This has led some researchers to use optimisation algorithms to explore, at least locally, the space of the possible learning rules. Given the limited knowledge of such a space, the tools of choice have been evolutionary algorithms [24] which, although not optimal for some domains, offer the broadest possible applicability. Very often the strategy adopted has been to use Genetic Algorithms (GAs) [7] to find the optimal parameters for prefixed classes of learning rules. The few results obtained to date are very promising. We recall them in Section 3.

In preliminary work (not reported) we have replicated part of these results and applied GAs to find the optimal parameters for the SBP algorithm, the SBP algorithm with momentum, and the Rprop algorithm. However, in that work we have realised that by fixing the class of rules that can be explored we biased the search and prevented the evolutionary algorithm from exploring the much larger space of rules which we, humans, have not thought about. So, in line with some recent work by Benjio [1], which is also summarised in Section 3, we decided to use a technique known as Genetic Programming (GP) developed by Koza [12]. GP allows the direct evolution of symbolic learning rules with their coefficients (if any) rather than the simpler evolution of parameters for a fixed learning rule.

We briefly summarise how GP works and describe our approach in Section 4. Section 5 reports the experimental results obtained on two classes of standard benchmark problems, the parity problems and the encoder-decoder problem. We discuss these results and draw some conclusions in Section 6.

2 Standard Backpropagation Algorithm and Recent Improvements

A multilayer perceptron is a fully connected feed-forward neural network in which an arbitrary input vector is propagated forward through the network, causing an activation vector to be produced in the output layer. The network behaves like a function which maps the input vector onto the output vector. This function is determined by the connection weights of the net.

The backpropagation algorithm follows the principle of gradient descent to perform a kind of error-correction learning. In SBP, the objective is to tune the weights of the network so that the network performs a desired input/output mapping.

Let u_i^l be the i^{th} neuron in the l^{th} layer (the input layer is the 0^{th} layer and the output layer is the k^{th} layer). Let n_l be the number of neurons in the l^{th} layer. The weight of the link between neuron u_j^l and neuron u_i^{l+1} is denoted by w_{ij}^l . Let $\{x_1, x_2, \dots, x_m\}$ be the set of input patterns that the network is supposed to learn and let $\{t_1, t_2, \dots, t_m\}$ be the corresponding target output patterns. The pairs (x_p, t_p) $p=1, \dots, m$ are called training patterns.

The output o_i^0 of a neuron u_i^0 when pattern x_p is presented coincides with its net input net_{ip}^l i.e. it is the i^{th} element, x_{ip} , of x_p . For the other layers, the net input net_{ip}^{l+1} of a neuron u_i^{l+1} for the input pattern x_p is usually computed as follows:

$$net_{ip}^{l+1} = \sum_{j=1}^{n_l} w_{ij}^l o_{jp}^l - \theta_i^{l+1},$$

where o_{jp}^l is the output of the neuron u_j^l , which is a non-linear activation-function (the sigmoid logistic function or symmetric function), and θ_i^{l+1} is the bias of neuron u_i^{l+1} . For the sake of a homogeneous representation, θ_i is often substituted by a weight to a ‘bias unit’ with a constant output 1. This means that biases can be treated like weights. We assume this in the rest of the paper.

The error ε_{ip} for the j^{th} neuron u_i^k of the output layer for the training pair (x_p, t_p) is computed as

$$\varepsilon_{jp} = t_{jp} - o_{jp}^k.$$

The SBP rule uses this error to adjust the weights in such a way that the error gradually reduces. The training process stops when the error of every neuron for every training pair is reduced to an acceptable level, or when no further improvement is obtained.

The network performance can be assessed using the total sum of squared (TSS) errors given by the following function:

$$E = \sum_{p=1}^m \sum_{i=1}^{n_k} \varepsilon_{ip}^2.$$

In the batched variant of the SBP which is based on the gradient of E the weight update of w_{ij}^l in the s^{th} learning step is given by:

$$\Delta w_{ij}^l(s) = \eta \frac{\partial E}{\partial w_{ij}^l}$$

where η is a parameter called learning rate.

The SBP algorithm suffers from several problems. Then, many methods of speeding up the SBP algorithm have been proposed [21, 20, 18, 22, 13, 17, 6].

Rprop is one of the fastest variation of the SBP algorithm [21, 20, 18]. Rprop stands for 'Resilient backpropagation'. It is a local adaptive learning scheme, performing supervised batch learning in multilayer perceptrons. For a detailed discussion see also [18, 19]. Rprop suffers from the same problems as any adaptive learning algorithm [18].

3 Previous Work on the Evolution of NN Learning Rules

A relatively small amount of previous work has been reported on the evolution of learning rules for NNs. Given the topology of the network, GAs have been used to find the optima learning rules. For example, Kitano [10] combined GAs with neural networks and proposed the NeuroGenetic Learning algorithm. This consists of five stages: 1) evolution, 2) developmental, 3) weight distribution, 4) network pruning and reduction, and 5) gradient-descend training. He

demonstrated that the method provides an order of magnitude speed up with respect to other methods. Whitley [27] showed that GAs can make a positive and competitive contribution in the neural networks area. As, although they have trouble getting an exact solution, all the solution are nearly correct. Chalmers [2] applied GAs to discover supervised learning rules for single-layer neural networks. He discussed the role of different kinds of connections systems and verified the optimality of the Delta rule. This author noticed that discovering more complex learning rules like the SBP using GAs is not easy because either a) one uses a highly complex genetic coding, or b) one uses a simpler coding which allows SBP as a possibility. In the first case the search space would be huge in the second case we bias the search using our own prejudices.

All these methods choose a fixed number of parameters and a rigid form for the learning rule. Koza [12, 11] has demonstrated that GP may be a good way of getting around the limitations inherent to fixed genetic coding which GAs suffer from. GP is a very natural way of encoding dynamic algorithmic processes of the kind we are investigating here. We will briefly summarise how GP works and describe our approach to discovering learning rules in next Section.

4 Evolution of NN Learning Rules with GP

GP has been applied successfully to a large number of difficult problems like automatic design, pattern recognition, robotics control, synthesis of neural networks, symbolic regression, music and picture generation, etc, but only one attempt to use GP to induce new learning rules for NNs has been reported to date.

Bengio [1] used GP to find learning rules which had optimal number of parameters. Bengio used the output of the input neuron, the error of the output neuron and the first derivative of the activation function of the output neuron as the terminals and algebraic operators as the functions. GP found a better learning rule compared to the rules discovered by simulated annealing and GAs. The new learning rule

suffers from the same problems as SBP and was only tested on special problems.

Our work is an extension of Bengio’s work our objective is to explore a larger space of rules using different parameters and different rules for the hidden and output layers. We hope to obtain a rule which is general like SBP but faster, more stable, and which can work in different conditions. We want to discover learning rules of the following form:

$$\Delta w_{ij}^l(s) = f(w_{ij}^l, x_{jp}^l, t_{ip}, o_{ip}^l)$$

Using GP the fitness of each learning rule is $f = E_{max} - E$ the TSS errors function and E_{max} is a constant such that $f \geq 0$. The value of E is measured at the maximum number of learning epochs. The task that the networks are supposed to learn through to each learning rule in the population are linearly separable mapping from input patterns to output patterns. These are described in the next section together with the function and the terminal nodes used by GP.

5 Experimental Results

It is not easy to establish a fair comparison between the many variants of supervised learning techniques. There are as many benchmark problems reported in the literature as there are new learning algorithms.

Here, we consider two problems which have been widely used: The ‘exclusive or’ (XOR) problem and its more general form, the N-input parity problem and the family of the N-M-N encoder problems force network to generalise and to map input patterns into similar output activation [4].

For the XOR problem, we used a three-layer network consisting of 2 input, 2 hidden, and 1 output neurons. We applied symmetric activation functions with output in the range [-1,1]. The weights were randomly initialised within the range [-1,1]. For the Encoder problems we used a three-layer network consisting of 10 input, 5 hidden, and 10 output neurons. Here, we used sigmoid activation functions with output in the range of [0,1]. The weights were randomly initialised within the range [-1,1].

In our approach we used two different learning rules for the output and hidden layers like in the SBP learning rule. In a first set of experiments we tried to use GP to evolve rules for the output layer, while the hidden layers were learning with the SBP rule. Poor results were obtained. We achieved much better results by changing the learning rule of the hidden layer from SBP to the Delta rule:

$$\Delta w_{ij}^l(s) = \eta o_{jm}^l \varepsilon_{im}^l$$

In these experiments, we expected that GP would rediscover an enhancement of the Delta learning rule, and then would find a new learning rule.

The fitness of each learning rule was computed using the TSS error the two problems mentioned above. In the case of the XOR problem we measured the TSS error after 100 learning epochs. For the encoder problem a limit of 200 epochs was chosen.

In a preliminary set of experiments with these problems we varied the initialisation procedure (using either the grow method or the full method) and the maximum depth of the initial trees (using a depth of 3,4 or 5). We found that the “full” method is more efficient on these problems than the “grow” method and that an initial maximum depth of 3 is sufficient to find good learning rules.

Then, GP was run for 500 generations with a population size of 1000, crossover probability is 0.9, mutation probability is 0.01, function set $\{+, -, \times\}$, and terminal set $\{w_{ij}^l, x_{jp}^l, t_{ip}, o_{ip}^l, 1, 2\}$.

All of the experiments were run using our own SBP and GP simulators. The simulators were written in POP11 and run on Sun SS20 machine with two RT625-180 MHZ processor. In these conditions each GP run takes five days of CPU time on average. For this reasons we were able to perform 10 GP runs in total.

In all runs, we realised that great differences in fitness occur when the terms ε_{ip}^l (i.e. $(t_{ip} - o_{ip}^l)$) and o_{jp}^l appear in the learning rule. In other words, the genetic search process tends to rediscover the Delta rule which corresponds to a gradient descent using Log-likelihood cost function (logarithmic error function) instead of energy function (quadratic error function) [8, 23, 15].

Table 1: The results on the parity problem.

2 input-parity			
Algorithm	η	epochs	error
SBP	0.5	500	0.61
NLR(1)	0.5	500	0.0346
NLR(2)	0.5	500	0.000016

Table 2: The results on the encoder problem.

10-5-10 Encoder			
Algorithm	η	epochs	error
SBP	0.1	200	0.31
NLR(1)	0.1	200	0.000194
NLR(2)	0.1	200	0.0

The result of our GP runs was a small set of promising new learning rules (NLRs) which we needed to study in different conditions to determine their reliability and efficiency. We performed two kinds of tests.

In a first series of tests we fixed the number of training epochs and we compared the TSS error of the NLRs with the TSS of the SBP. In these experiments we first found the learning rate which provided the best performance for SBP and then we used it also for the NLRs. Then the performances of SBP and the rules evolved by GP were tested in ten independent runs. Each test run had different initial random weights.

Tables 1 and 2 report the average TSS errors for SBP and two NLRs discovered in our GP runs. The rules are:

$$\text{NLR}(1) = 2x_{jp}^l(t_{ip}^l - o_{ip}^l) - t_{ip}^l,$$

and

$$\text{NLR}(2) = (t_{ip}^l - o_{ip}^l)(x_{jp}^l(x_{jp}^l + o_{ip}^l) - t_{ip}^l).$$

It is clear that with the same number of epochs and the same learning rate the TSS error is much smaller for the NLRs. As a matter of fact the NLRs tend to converge to the minimum of the error function in very few epochs with respect to SBP. For example

in 10-5-10 Encoder problem, the average error with NLR(2) was as small as 0.00026 after 20 epochs, only.

NLR(2) is very simple and provides good performance on both problems. So, in a second set of tests, we have decided to use it extensively on the XOR problem and to compare it with the SBP with and without different speed-up algorithms (the momentum method and the Rprop). In these tests each algorithm was run with the best set of parameters which we determined in preliminary runs. The results are shown in Table 3. The table reports the six algorithms with their learning rate η , momentum α , initial learning rate η_o , minimum (Min.), maximum (Max.), the mean number of epochs (the period during which every pattern of the training set is presented once) which the algorithm needs to converge in 100 independent runs and the number of successes. If the network had not converged within 1000 epochs, the run was declared unsuccessful. The learning rate was kept in the range [0.1,1].

It should be noted that the minimum and maximum number of epochs required by NLR(2) to converge is 22 and 49, respectively. This compares very favourably with the minimum and maximum number of epochs required by the SBP (101 and 870, respectively). NLR(2) outperforms SBP also when used with the momentum method and the Rprop algorithm. Indeed the average number of epochs necessary for convergence with the NLR(2) and Rprop is 20.5 which compares well with the results reported by Fahlman for this problem using Quikprop [4] which required on average 24.22 epochs to converge.

The table shows that the best results are achieved using NLR(2) with momentum, which learned the XOR function in 12.71 epochs on average and succeeded in all runs.

It is important to note that the performance achieved by NLR(2) is not sensitive to the particular choice of learning rate as shown in Figure 1. The figure clearly shows that NLR(2) is much more stable than SBP, at least on the XOR problem.

In Table 4 SBP_s and NLR_s are the successes number of SBP and NLR respectively. Table 4 shows a comparison between SBP and NLR(2) on three parity problems. The table shows the learning rates, the ratio of the mean number of epochs necessary to con-

Table 3: Performance of 6 different learning rules on the XOR problem.

Algorithm	η	α	η_o	Learning Epochs			Successful Runs
				Min.	Max.	Mean	
<i>SBP</i>	0.96	—	—	101	870	163.95	100
<i>SBP + Mom.</i>	0.96	0.8	—	19	176	57.45	99
<i>Rprop</i>	—	—	0.9	12	56	25.93	86
<i>NLR(2)</i>	0.3	—	—	22	49	30.45	100
<i>NLR(2) + Mom.</i>	0.3	0.65	—	9	28	12.71	100
<i>NLR(2) + Rprop</i>	—	—	1.85	2	69	20.5	88

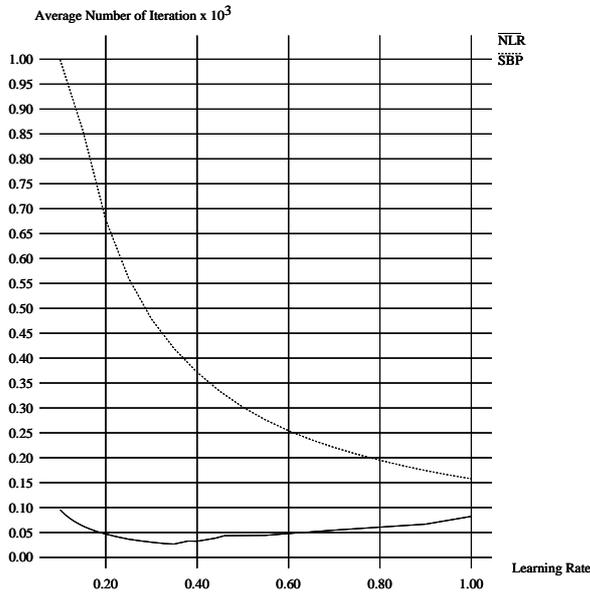


Figure 1: Average number of epochs necessary for convergence with SBP and NLR(2) as a function of the learning rate used.

Table 4: Comparison between SBP and NLR on three parity problems.

Parity problems					
Problems	η_{SBP}	η_{NLR}	Perf. ratio	SBP_s	NLR_s
2 input	0.96	0.3	5.2	100	100
3 input	0.3	0.052	4.9	66	100
4 input	0.03	0.011	3.0	55	87

verage for SBP and NLR (performance ratio), and the number of successes in 100 runs. The maximum number of epochs was 1,000 in the case of 2-parity problem, 5,000 in the case of the 3-parity problem and 10,000 for the 4-parity problem. These experimental show that NLR(2) generalises to other problems and converges considerably faster than SBP.

6 Conclusions and Future Work

In this paper, we have applied GP to find new supervised learning rules for neural networks. GP has discovered a number of rules, among which is the Delta learning rule. Some of these rules have performed much better than the SBP learning rule on all the sample problems considered. Whether these rules work better than SBP on all problems remains to be seen.

In case of the parity problem, we tested NLR(2) to examine its convergence ability in the presence of different of learning rates. These experiment have shown very promising stability results.

This study indicates that there are many supervised learning algorithms that perform better and are more stable than the SBP learning rule and that GP can discover them. In addition to providing us with new general learning rules which can be applied to different problems. We hope that GP, through the examination of the resulting rules, will help us find new theories for supervised learning.

Acknowledgements

The authors wish to thank the members of the EE-BIC (Evolutionary and Emergent Behaviour Intelligence and Computation) group for useful discussions and comments.

References

- [1] S. Bengio, Y. Bengio, and J. Cloutier. Use of genetic programming for the search of a learning rule for neural networks. In *Proceedings of the First Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pages 324–327, Orlando, Florida, USA, 1994.
- [2] D. Chalmers. The evolution of learning: An experiment in genetic connections. In Elman Touretsky, editor, *Proceedings of the 1990 Connectionist Models Summer School*, pages 81–90, San Mateo, CA, 1990. Morgan-Kaufman.
- [3] Harris Drucker and Yann Le Cun. Improving generalisation performance using double back-propagation. *IEEE Transactions on Neural Networks*, 3(6):991–997, 1992.
- [4] S. Fahlman. An empirical study of learning speed in back propagation networks. Technical Report CMU-CS-88-162, Carnegie Mellon University, Pittsburgh, PA, 1988.
- [5] S. Haykin. *Neural Networks: A Comprehensive Foundation*. IEEE Society Press, Macmillan College Publishing, New York 10022, 1994.
- [6] Yoshio Hirose, Koichi Yamashit, and Shimpei Hijiya. Back propagation algorithm which varies the number of hidden units. *Neural Networks*, 4(1):61–66, 1991.
- [7] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.
- [8] M. Holt and S. Semnani. Convergence of back-propagation in neural networks using a log-likelihood cost function. *Electronics Letters*, 26(23):1964–1965, 1990.
- [9] D. Karras and S. Perantonis. An efficient constrained training algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 6(6):1420–1434, Nov 1995.
- [10] H. Kitano. Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms. *Physica D*, 75:225–238, 1994.
- [11] John Koza. *Genetic Programming II: Automatic discovery of reusable programs*. The MIT Press, Cambridge, Massachusetts, 1994.
- [12] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
- [13] John Kruschke and Javier Movellan. Benefits of gain: Speeded learning and minimal hidden layers in back propagation networks. *IEEE Transactions on Systems, Man and Cybernetics.*, 21(1):273–280, 1991.
- [14] Martin Moller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533, 1993.
- [15] A. Ooyen and B. Nienhuis. Improving the convergence of the back propagation algorithm. *Neural Networks*, 5:465–471, 1992.

- [16] Alexander Parlos, B. Fernandez, Amir Atiya, J. Muthusami, and Wei Tsai. An accelerated learning algorithm for multilayer perceptron networks. *IEEE Transactions on Neural Networks*, 5(3):493–497, 1994.
- [17] Donald Prados. New learning algorithm for training multilayered neural networks that uses genetic-algorithm techniques. *Electronics Letters*, 28(16):1560–1561, 1992.
- [18] Martin Riedmiller. Advanced supervised learning in multi-layer perceptrons from backpropagation to adaptive learning algorithms. *Computer Standards and Interfaces Special Issue on Neural Networks*, 16(3):265–275, 1994.
- [19] Martin Riedmiller and Heinrich Braun. A direct method for faster backpropagation learning: The RPROP Algorithm. In *IEEE International Conference on Neural Networks 1993 (ICNN93)*, San Francisco, pages 586–591, 1993.
- [20] Dilip Sarkar. Methods to speed up error back propagation learning algorithm. *ACM Computing Surveys*, 27(4):519–542, 1995.
- [21] W. Schiffmann, M. Joost, and R. Werner. Application of genetic algorithms to the construction of topologies for multilayer perceptrons. In *Proceedings of International Conference on Neural Networks and Genetic Algorithms (ICNNGA)*, pages 675–682, University of Innsbruck, Austria, 1993.
- [22] F. Silva and L. Almeida. Speeding up backpropagation. In R. Eckmiller, editor, *Advanced Neural Computers*, pages 151–158, Amsterdam, 1990. Elsevier Science Publishers.
- [23] Sara Solla, Esther Levin, and Michael Fleisher. Accelerated learning in layered neural networks. *Complex System*, 2:625–640, 1988.
- [24] W. Spears, K. De Jong, T. Bäck, D. Fogel, and H. de Garis. An overview of evolutionary computation. In Pavel B. Brazdil, editor, *Proceedings of the European Conference on Machine Learning (ECML-93)*, volume 667 of *LNAI*, pages 442–459, Vienna, Austria, April 1993. Springer Verlag.
- [25] Alessandro Sperduti and Antonina Starita. Speed up learning and network optimisation with extended back propagation. *Neural Networks*, 6:365–383, 1993.
- [26] J. Taylor. *The Promise of Neural Networks*. Springer-Verlag, London, 1993.
- [27] D. Whitley and T. Hanson. Optimizing neural networks using faster, more accurate genetic search. In J. Schaffer, editor, *Third International Conference on Genetic Algorithms*, pages 391–396, George Mason University, 1989. Morgan Kaufmann.