

Genetic Programming Can Discover Fast and General Learning Rules for Neural Networks

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

Technical Report CSRP-98-3

January 20, 1998

Abstract

The Standard BackPropagation (SBP) algorithm for training neural networks 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 new learning algorithm has been discovered and compared with SBP 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 networks, against which other learning algorithms are often benchmarked [5, 24].

SBP suffers from a number of problems [5, 4, 18, 19, 21, 22]. 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.

For these reasons in the past few years a number of improvements to SBP have been proposed in the literature (see [21] for a survey). We will review the SBP rule and mention some of these 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 algorithms 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 [15, 3, 23, 14, 12, 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 [26] which, although not optimum 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.

By using GAs, we have realised that fixing the class of rules that can be explored biases the search and prevents 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 GP [10] as 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 describe our approach in Section 4. Section 5 reports the experimental results obtained on four classes of standard benchmark problems: the parity, the encoder-decoder, the character recognition, and the multiplexer problems. 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 an output vector. This function is determined by the connection weights of the net. The objective of SBP is to tune the weights of the network so that the network performs the 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_{ip}^0 of a neuron u_i^0 in the input layer when pattern x_p is presented coincides with its net input net_{ip}^l i.e. with 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 (usually $o_{jp}^l = f(net_{jp}^l)$ with f a non-linear activation-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 interpreted as the weight of a connection to a 'bias unit' with a constant output 1.

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

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

The error for neurons in hidden layer l , ε_{jp}^l , is a weighted sum of the errors ε_{jp}^{l+1} in the following layer.

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 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.

Many methods of speeding up the SBP algorithm have been proposed [18, 21, 22, 6, 8, 11, 16]. One of the fastest variations of the SBP algorithm is Rprop [18, 21, 8]. 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 is considerably faster than SBP but it suffers from many of the problems mentioned in Section 1.

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 optimum learning rules. For example, Kitano [9] combined GAs with neural networks and proposed the NeuroGenetic learning algorithm. This consists of five stages: 1) evolution, 2) development, 3) weight distribution, 4) network pruning and reduction, and 5) gradient-descent training. He demonstrated that the method provides an order of magnitude speed up with respect to other methods. Montana [13] used GAs for training feedforward networks and created a new method of training which is similar to SBP. Whitley [25] 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 connectionist systems and verified the optimality of the Delta rule, a simpler variant of SBP applicable to single-layer neural networks [20]. The 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 is huge, in the second case we bias the search using our own prejudices.

All these methods are limited as they choose a fixed number of parameters and a rigid form for the learning rule. We believe that GP may be a good way of getting around the limitations inherent to fixed genetic coding which GAs suffer from. We will describe our approach to discovering learning rules based on GP in the 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 an optimum number of parameters. Bengio used the output of the input neuron o_{jp}^l , the error of the output neuron ε_{ip}^{l+1} and the first derivative of the activation function of the output neuron $f'(net_{ip}^l)$ as terminals and algebraic operators as functions for GP. So all the ingredients to rediscover the SBP algorithm were used. GP found a better learning rule compared to the rules discovered by simulated annealing and GAs. However, the new learning rule suffered from the same problems as SBP and was only tested on a very specific problem.

Our work is an extension of Bengio's work with the objective 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:

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

In our approach we used two different learning rules for the output and hidden layers like in the SBP learning rule. In the experiments we used GP to evolve rules for the output layer, while the hidden layers were learning with the SBP rule.

This work is a continuation of our previous work [17]. In that work, we applied GP to find new supervised learning rules for neural networks. GP discovered a number of rules, among which is the Delta learning rule. Some of these rules performed much better than the SBP learning rule on the sample problems considered (XOR and Encoder) but on other problems (character recognition) oscillated. So, we decided to use more problems to infer more general rules.

The tasks that the networks are supposed to learn through each learning rule in the population are linearly separable mappings. These are described in the next section together with the functions and the terminals used by GP.

5 Experimental Results

It is not easy to perform 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 four problems which have been widely used: 1) the 'exclusive or' (XOR) problem and its more general form, the N-input parity problem, 2) the family of the N-M-N encoder problems which force the network to generalise and to map input patterns into similar output activations [17], 3) the character recognition problem with 7 inputs representing the state of a 7-segment light emitting diode (LED) display and 4 outputs which represent the digits 1 to 9 binary encoded [1], and 4) the display problem with 4 inputs which represent a digit from 1 to 9 in binary and 7 outputs which represent the LED configuration to visualise the digit.

For the XOR problem, we used a three-layer network consisting of 2 input, 2 hidden, and 1 output neurons with 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 $[0,1]$. For the character recognition problems we used a three-layer network consisting of 7 input, 10 hidden, and 4 output neurons. We used sigmoid activation functions with output in the range of $[0,1]$. For the display problems we used a three-layer network consisting of 4 input, 10 hidden, and 7 output neurons. We used sigmoid activation functions with output in the range of $[0,1]$.

The fitness of each learning rule was computed using the TSS error E for the four problems mentioned above:

$$f = \begin{cases} E_{max} - E & \text{if the network does not learn,} \\ C_{max} - C_{min} & \text{if otherwise} \end{cases}$$

where C_{min} is the minimum number of epochs needed for convergence and E_{max} and C_{max} are constants such that $f \geq 0$. The value of E is measured at the maximum number of learning epochs. In the case of the XOR problem we measured the TSS error after 100 learning epochs. For the encoder, character recognition, and display problems the limits 200, 100 and 100 epochs were used respectively.

GP was run for 500 generations with a population size of 1000, crossover probability 0.9, mutation probability 0.01, function set $\{+, -, \times\}$, and terminal set $\{w_{ij}^l, x_{jp}^l, t_{ip}, o_{ip}^l, 1, 2\}$. We used the "full" initialisation method with an initial maximum depth of 3.

The experiments were performed using our own SBP and GP simulators. The simulators are written in POP11 and run on Alpha Digital machine with 233 MHZ processors. In these conditions each GP run takes four days of CPU time on average. For this reason we were able to perform only 20 GP runs in total.

Table 1: Performance of NLR on four different problems.

XOR					
		Learning Epochs			Successful Runs
Algorithm	η	Min.	Max.	Mean	
<i>SBP</i>	0.96	101	870	163.95	100
<i>NLR</i>	0.96	50	197	63.39	100
Encoder					
		Learning Epochs			Successful Runs
Algorithm	η	Min.	Max.	Mean	
<i>SBP</i>	0.4	14	60	29.663	100
<i>NLR</i>	0.4	6	18	11.821	100
Character Recognition					
		Learning Epochs			Successful Runs
Algorithm	η	Min.	Max.	Mean	
<i>SBP</i>	0.3	295	500	392.079	89
<i>NLR</i>	0.3	45	84	58.51	100
Display					
		Learning Epochs			Successful Runs
Algorithm	η	Min.	Max.	Mean	
<i>SBP</i>	0.8	130	500	250.514	87
<i>NLR</i>	0.8	46	204	79.069	100

In these experiments GP was allowed to evolve learning rules for the output layer, while the learning rule for the hidden layers was the SBP and was fixed. The best rule discovered in our runs is extremely simple:

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

In the following we will call it New Learning Rule (NLR). This rule is actually the well-known Delta rule which is used to train neural networks without hidden neurons. The interest of this result is that GP is using the Delta rule in conjunction with SBP to improve performance, in a multi-layer neural network. We studied the NLR in different conditions to determine its reliability and efficiency with respect to SBP. We performed four kinds of tests.

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 1. The table reports the two algorithms with their learning rate η , the minimum (Min.), maximum (Max.) and 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 500 epochs, the run was declared unsuccessful.

It should be noted that for the XOR problem the minimum and maximum number of epochs required by NLR to converge is 50 and 197, respectively.

This compares very favourably with the minimum and maximum number of epochs required by the SBP (101 and 870, respectively). Also, the average number of epochs necessary for convergence with the NLR is only 2/5 of the corresponding SBP value. Similar performance improvements were obtained for the other problems.

NLR is very simple and provides good performance on the four problems. So, in a second set of tests, we have decided to use it on the four problems and to compare its convergence behaviour with the SBP with and without the Rprop speed-up algorithm. Figures 1-4 show typical runs of SBP, NLR, SBP with Rprop and NLR with Rprop. All runs used the same initial random weights

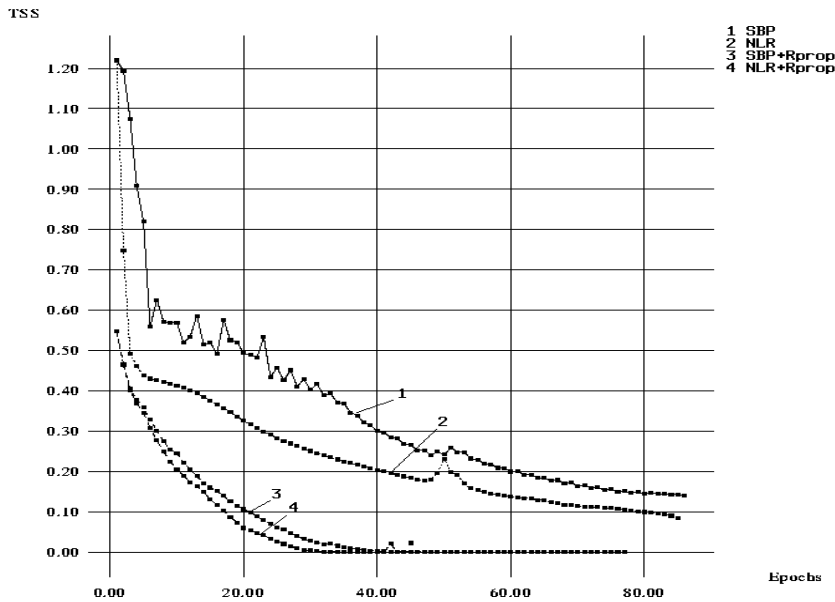


Figure 1: Plot of the TSS error of SBP and NLR in the character recognition problem.

Figure 1 shows the TSS error in of the SBP and NLR with and without Rprop on the character recognition problem. The results obtained indicate that NLR+Rprop achieves its target output at approximately 30 epochs while the SPB+Rprop takes 40 epochs.

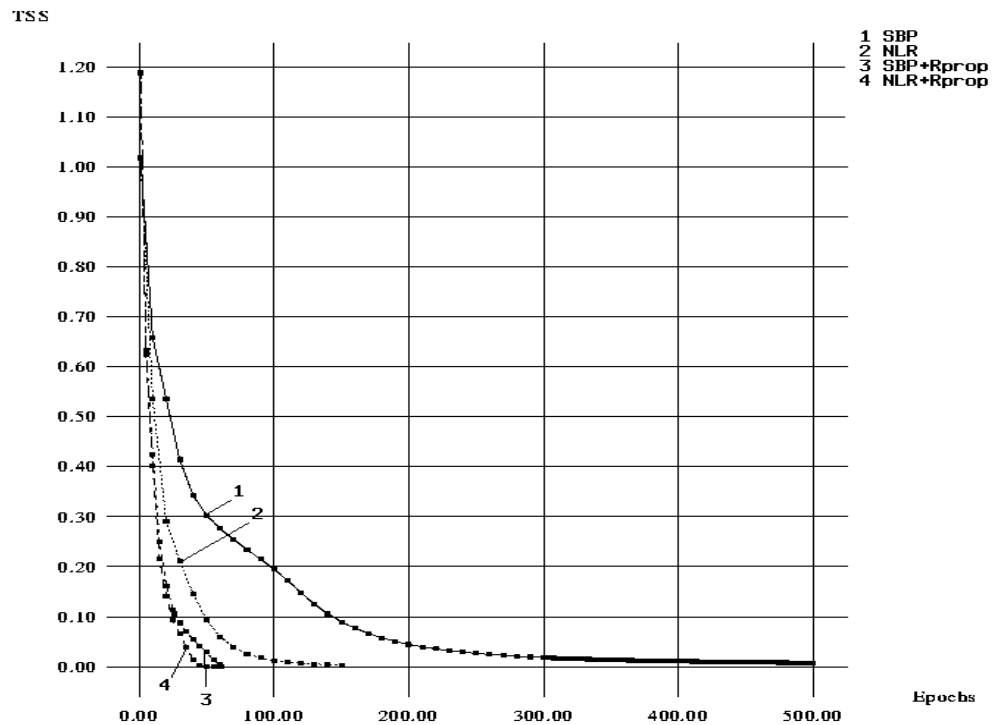


Figure 2: Plot of the TSS error of SBP and NLR in the displayproblem.

Figure 2 shows the same data for the display problem. SBP achieves its target value after 500 epochs and the NLR achieves its target value after 150 epochs. This is as three fold speed-up. Also, NLR+Rprop achieves its target error at approximately 40 epochs while SBP+Rprop takes 60 epochs.

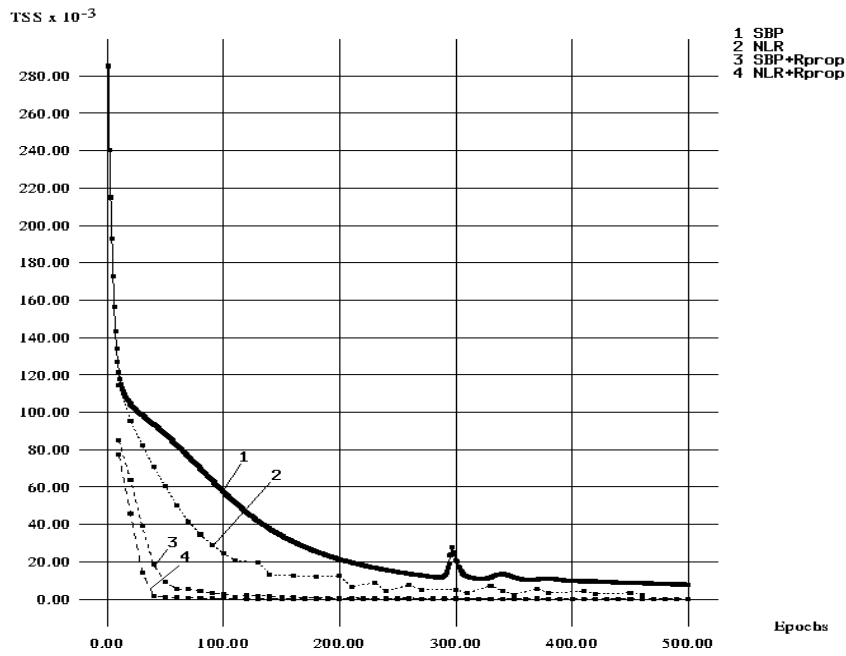


Figure 3: Plot of the TSS error of SBP and NLR in the XOR problem.

Figure 3 confirms that NLR outperforms SBP also on the XOR problem. In this case SBP+Rprop required 110 epochs to converge, while NLR+Rprop took 40 epochs.

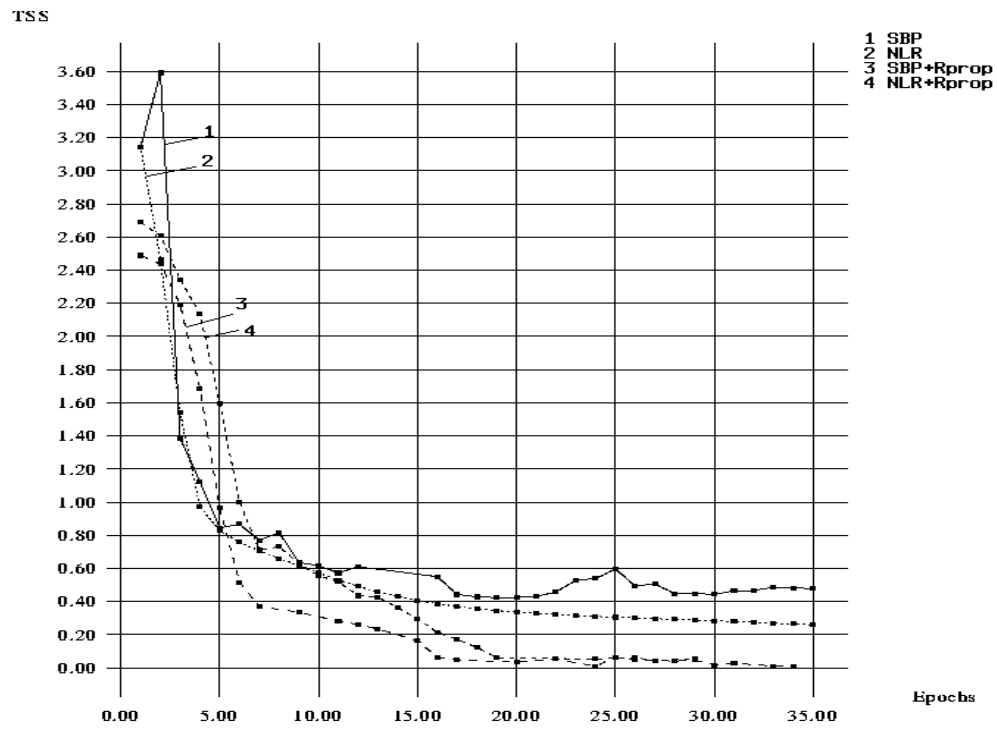


Figure 4: Plot of the TSS error of SBP and NLR in the Encoder problem.

Figure 4 shows that NLR outperforms SBP on the Encoder problem as well.

As illustrated in Figures 1-4 NLR is much faster than SBP. NLR is much faster than SBP also when applied with Rprop. Interestingly the NLR+Rprop algorithm seems to be able to solve all problems in nearly constant time despite the considerable differences in complexity of such problems.

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 useful way of using the Delta learning rule originally developed for single-layer neural networks to speed up learning. This rule has performed much better than the SBP learning rule on all the sample problems considered being up to 4 times faster. Whether this rule outperforms SBP in general remains to be seen.

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, through the examination of the resulting rules, in the future GP will help us find new theories for supervised learning.

7 Acknowledgements

The authors wish to thank the members of the EEBIC (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, 1994.
- [2] D. J. Chalmers. The evolution of learning: An experiment in genetic connectionism. In *Connectionist Models Summer School. San Mateo, CA.*, 1990.
- [3] Harris Drucker and Yann Le Cun. Improving generalisation performance using double backpropagation. *IEEE Transactions on Neural Networks*, 3(6):991–997, 1992.
- [4] S. E. 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*, 4:61–66, 1991.
- [7] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.
- [8] W. Schiffmann M. Joost and R. Werner. Optimisation of the backpropagation algorithm for training multilayer perceptrons. Technical report 16/1992, University of Koblenz, Institute of Physics, 1992.
- [9] H. Kitano. Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms. *Physica D*, 75:225–238, 1994.
- [10] John R. Koza. *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, Massachusetts, 1992.
- [11] John K. Kruschke and Javier R. Movellan. Benefits of gain: Speeded learning and minimal hidden layers in back propagation networks. *IEEE Transactions on Systems, Man and Cybernetics.*, 21(1), 1991.
- [12] Martin F. Moller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533, 1993.

- [13] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of Eleventh International Joint Conference on Artificial Intelligence*, pages 762–767, San Mateo, CA, 1989. Morgan Kaufmann.
- [14] Alexander G. Parlos, B. Fernandez, Amir Atiya, J. Muthusami, and Wei K. Tsai. An accelerated learning algorithm for multilayer perceptron networks. *IEEE Transactions on Neural Networks*, 5(3):493–497, 1994.
- [15] S. J. Perantonis and D. A. Karras. An efficient constrained training algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 6(6):237–149, Nov 1995.
- [16] Donald L. Prados. New learning algorithm for training multilayered neural networks that uses genetic-algorithm techniques. *Electronics Letters*, 28(16):1560–1561, 1992.
- [17] Amr K. Radi and Riccardo Poli. Discovery of neural network learning rules using genetic programming. In *IEEE International Conference on Evolutionary Computation (ICEC'98)*, 1998.
- [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. *IEEE International Conference on Neural Networks 1993 (ICNN93)*, pages 586–591, 1993.
- [20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Parallel Distributed Processing*. MIT Press, Cambridge, MA, and London, England, 1986.
- [21] Dilip Sarkar. Methods to speed up error back propagation learning algorithm. *ACM Computing Surveys*, 27(4):519–542, 1995.
- [22] Femando F. Silva and Luis B. Almeida. *Speeding Backpropagation*, pages 151–158. Advanced Neural Computers, Elsevier Science Publishers B.V. (North-Holland), 1990.
- [23] Alessandro Sperduti and Antonina Starita. Speed up learning and network optimization with extended back propagation. *Neural Networks*, 6:365–383, 1993.
- [24] J. G. Taylor. *The Promise of neural networks*. Springer-Verlag, London, 1993.

- [25] D. Whitley and T. Hanson. Optimizing neural networks using faster, more accurate genetic search. In J. D. Schaffer, editor, *Third International Conference on Genetic Algorithms*, pages 391–396, San Mateo, CA, 1989. Morgan Kaufmann.
- [26] M. Spears William, K. A. De Jong, T. Baeck, David Fogel, and Hugo de Garis. An overview of evolutionary computation. In *Proceedings of the European Conference on Machine Learning*, pages 442–459, 1993.