Evolutionary Discovery of Learning Rules for Feedforward Neural Networks with Step Activation Function

Amr Radi
School of Computer Science
The University of Birmingham
Birmingham, B15 2TT, UK
A.M.Radi@cs.bham.ac.uk
Phone: +44-121-414-3736

Riccardo Poli
School of Computer Science
The University of Birmingham
Birmingham, B15 2TT, UK
R.Poli@cs.bham.ac.uk
Phone: +44-121-414-3739

Abstract

Neural networks with step activation function can be very efficient ways of performing non linear mappings. However, no standard learning algorithm exists for training this kind of neural networks. In this work we use Genetic Programming (GP) to discover supervised learning algorithms which can train neural networks with step activation function. Thanks to GP, a new learning algorithm has been discovered which has been shown to provide good performance.

1 INTRODUCTION

Supervised learning algorithms are by far the most frequently used methods to train artificial neural networks [4]. 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 [10, 24]. 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 improvements in Section 2.

A major drawback of SBP is that it can not train networks using step activation functions. This kind of networks are very important for several reasons. Firstly, they can be implemented very efficiently in software and hardware. Secondly, they can perform complex non-linear mappings. Lastly, but perhaps more importantly, historically they have been the first non-linear neural networks to be studied. After their introduction thanks to McCulloch and Pitts [11], they were studied for a number of years. However, the study of such networks (and of all neural networks in general) was nearly abandoned in the Seventies due to Minsky and Papert's book on perceptrons [12], which clearly indicated the limitations of single-layer neural networks with step activation function. These limitations were due to the fact at the time learning algorithms were available only for single-layer networks. For example Rosenblatt's algorithm [19] could train single layer networks with step activation function. The field of neural networks reexpanded in the Eighties and Nineties thanks to the work of Hopfield [6] and to the discovery of the Backpropagation algorithm [20]. However, no-one has been able to overcome the original limitation of networks with step activation function: the lack of a learning algorithm for multi-layer nets.

Finding new learning rules is a very difficult task. Indeed, a critical analysis of the huge literature on this topic shows that only a few really novel algorithms which demonstrated much better performance than SBP have been produced in the last 10 years [1, 7]. This slow progress has led some researchers to use optimisation algorithms to explore the space of the possible learning rules. Given the limited knowledge of such a space, the tools of choice have been evolutionary algorithms [23] which, although not optimum for some domains, offer the broadest possible applicability. Very often the strategy adopted has been to use Genetic Algorithms (GAs) [5] to find the optimum parameters for prefixed classes of learning rules. The few results obtained to date are promising. We recall them in Section 3.

GAs require fixing the class of rules that can be explored. This biases the search and prevents the algorithm from exploring the much larger space of possible rules. So, in line with some work by Benjio [1], also summarised in Section 3, we decided to use GP [9] as this 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. This paper describes the application of GP to the discovery of learning rules for both the output and the hidden layers of neural networks with step activation function.

We describe our approach in Section 4. Our earlier work on the evolution of learning rules for neural networks with continuous activation functions is summarised in the same section. Section 5 reports the experimental results obtained on three classes of standard benchmark problems: the parity, the encoder-decoder, and the character recognition problems. We discuss these results and draw some conclusions in Section 6.

2 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 [4]. 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. In this section we briefly recall the basic concepts of multilayer feed-forward neural networks, the SBP and some of its improvements. The reader should refer to [21, 18] for more details.

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 connection between neuron u_j^l and neuron u_i^{l+1} is denoted by w_{ij}^l. Let $\{x_1, x_2, ..., x_m\}$ be the set of input patterns that the network is supposed to learn and let $\{t_1, t_2, ..., t_m\}$ be the corresponding target output patterns. The pairs (x_p,t_p) $p = 1, .., m$ are called training patterns. Each x_p is an n_o-dimensional vector with components x_{ip}. Each t_p is an n_k-dimensional vector with components t_{ip}.

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}^0, i.e. with x_{ip}. For the other layers, the net input net_{ip}^{l+1} of neuron u_i^{l+1} (when the input pattern x_p is presented to the network) 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_{ip}^k for neuron u_i^k of the output layer for the training pair (x_p,t_p) is computed as

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

The SBP rule uses these errors to adjust the weights (usually initialised randomly) in such a way that the error gradually reduce.

The network performance can be assessed using the Total Sum of Squared (TSS) errors given by the following function:

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

The training process stops when the error E is reduced to an acceptable level, or when no further improvement is obtained.

In the batched variant of the SBP the updating of w_{ij}^l in the s^{th} learning step (often called an "epoch") is performed accordingly to the following equations:

$$w_{ij}^l(s+1) = w_{ij}^l(s) + \Delta w_{ij}^l(s)$$

$$\Delta w_{ij}^l(s) = \eta \delta_{ip}^{l+1}(s) o_{jp}^l(s)$$

where $\delta_{ip}^{l+1}(s)$ refers to the error signal at neuron i in layer $l+1$ for pattern p at epoch s, which is the product of the first derivative of the activation function f' and the error $\varepsilon_{ip}^{l+1}(s)$, and η is a parameter called learning rate.

The output of neurons with step activation function is:

$$o_{jp}^l = f(net_{jp}^l) = \begin{cases} 1 & \text{if } net_{jp}^l > 0.5, \\ 0 & \text{otherwise.} \end{cases}$$

Since this is not differentiable in 0 and its derivative is 0 elsewhere, the SBP algorithm cannot be used to train networks with step activation function.

Many methods of speeding up the SBP algorithm have been proposed [18, 21, 22, 14]. Rprop is one of the fastest variation of the SBP algorithm [18, 21, 22]. Rprop stands for 'Resilient backpropagation'. It is a local adaptive learning scheme, performing supervised batch learning in multilayer perceptrons. For a detailed discussion see [18]. Although Rprop is considerably faster than SBP, it still suffers from many problems [3] and, like SBP, it cannot train networks with step activation function.

3 PREVIOUS WORK

A considerable amount of work has been done on the evolution of the weights and/or the topology of neural networks.

See for example [8, 15]. However only a relatively small amount of work has been reported on the evolution of learning rules for neural networks. Given the topology of the network, GAs have been used to find optimum learning rules. For example, Montana [13] used GAs for training feedforward networks and created a new method of training which is similar to SBP. 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 one uses a highly complex genetic coding, or 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 the methods mentioned above are limited as they choose a fixed number of parameters and a rigid form for the learning rule. 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. GP may be a good way of getting around the limitations inherent to fixed genetic coding which GAs suffer from. However, only one attempt to use GP to induce new learning rules for neural networks has been reported before our own work.

Bengio [1] used GP to find learning rules for neural networks with sigmoid activation function. Bengio used the output of the input neuron o^l_{jp}, the error of the output neuron ε^{l+1}_{ip} and the first derivative of the activation function of the output neuron as terminals and algebraic operators as functions for GP. Bengio used a very strong search bias towards a certain class of SBP-like learning rules as only 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, was only tested on a very specific problem and can not train networks with step activation function. We will describe our approach to discovering learning rules based on GP in the next section.

4 EVOLUTION OF NEURAL NETWORK LEARNING RULES WITH GP

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 also want to train networks using step activation function. Our objective is to obtain rules which are general, like SBP, fast, stable, and which can work in different conditions. We want to discover learning rules of the following form:

$$\varDelta w^l_{ij} = \begin{cases} F(w^l_{ij}, o^l_{jp}, t_{ip}, o^{l+1}_{ip}) & \text{for the output layer,} \\ F(w^l_{ij}, o^l_{jp}, o^{l+1}_{ip}, E^{l+1}_{ip}) & \text{for the hidden layers,} \end{cases}$$

where o^l_{jp} is the output of neuron u^l_j when pattern p is presented to the network and $E^{l+1}_{ip} = \sum_j w^{l+1}_{ji} \delta^{l+2}_{jp}$. So, in our approach we used two different learning rules one for the output layer and one for the hidden layers like in the SBP learning rule. In our previous research [17, 16], GP was successful in discovering a number of learning rules for the output and hidden layers of feed-forward neural networks with sigmoid activation functions. Among them we found one which was better than SBP in all problems considered. This paper extends that work by applying GP to evolve rules for both the output and hidden layers of networks with step activation function.

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

5 EXPERIMENTAL RESULTS

5.1 EXPERIMENTAL SETUP

We considered five problems from the following three classes which have been widely used in the literature: a) the 'exclusive or' (XOR) problem and its more general form, the N-input parity problem with 3 and 4 inputs, b) the family of the N-M-N encoder problems which force the network to generalise and to map input patterns into similar output activations [16], c) 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].

For the XOR problem, we used a three-layer network consisting of 2 input, 2 hidden, and 1 output neurons. For the 3-input parity problem, we used a three-layer network consisting of 3 input, 3 hidden, and 1 output neurons. For the 4-input parity problem, we used a three-layer network consisting of 4 input, 4 hidden, and 1 output neurons. 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. For the character recognition problems we used a three-layer network consisting of 7 input, 10 hidden, and 4 output neurons. The weights were randomly initialised within the range [0,1].

The fitness of each learning rule was computed after applying the rule to the XOR problem for 1000 iterations 20 times. Each time the network was initialised with different random weights. If the rule was able to train the network (i.e., the network was capable of reproducing correctly all the patterns in the training set) in at least 1 case, the fitness of the rule was the number of times (out of 20) the rule successfully trained the network. If the rule was unable to train the network in all cases, then the fitness of the rule was $f = \lambda(E_{max} - E)$, where E is the average TSS error over the 20 cases (the value of E is measured at the maximum number of learning epochs), E_{max} is a constant such that $f \geq 0$, and λ is factor such that $f < 1$. When a rule was successful on the XOR problem in at least 19 cases, we also tested the rule on the 3-input parity problem (initialising the network 20 times with different random weights and applying the rule for 5000 epochs) to see if the rule would generalise. The results of these tests were not used in the fitness function, but were used to later select good rules which deserved further off-line testing on all the problems considered (for the 4-input parity, the encoder, and character recognition problems we used 10000, 500, and 500 epochs, respectively).

GP was run for 30-100 generations with a population size of 2000-5000 and a crossover probability 0.9. After applying crossover, subtree mutation was applied with a probability of 0.01 to all the population. In these experiments, GP was allowed to evolve learning rules for both the output and the hidden layers as in the SBP but for networks with step activation function. We used the function set $\{+, -, \times\}$, and the terminal set $\{w^l_{ij}, o^l_{jp}, t_{ip}, o^{l+1}_{ip}, 0.5, 0.1\}$ for the output layer while for the hidden layers we used $\{w^l_{ij}, o^l_{jp}, o^{l+1}_{ip}, E^{l+1}_{ip}, 0.5, 0.1\}$.

We used the "full" initialisation method with a maximum depth of 3 or 5, and tournament selection with a tournament size of 4. The experiments were performed using our own neural network and GP simulators. The simulators are written in POP11 and run on a Digital Alpha machine with 233 MHz processor and on two Sun UltraSPARC 5 and 10 running at 270 and 300 MHz, respectively. In these conditions a GP run takes ten days of CPU time on average.

5.2 RESULTS

In the experiments GP discovered several New Learning Rules (NLRs). The following are same of these (after manual simplification):

$$NLR^O_1 = \eta[0.1 o^l_{jp} \varepsilon^{l+1}_{ip} - 0.0005],$$
$$NLR^H_1 = \eta[0.11 o^{l+1}_{ip} - (0.1 - E^{l+1}_{ip})(o^l_{jp} o^{l+1}_{ip} - o^l_{jp})],$$
$$NLR^O_2 = \eta[0.1 o^l_{jp} \varepsilon^{l+1}_{ip} - 0.001],$$
$$NLR^H_2 = \eta[0.11 - (0.01 - E^{l+1}_{ip})(o^l_{jp} o^{l+1}_{ip} - o^l_{jp})],$$

and

$$NLR^O_3 = \eta[0.1 \varepsilon^{l+1}_{ip}(o^l_{jp} - 0.5)(o^l_{jp} - 0.1)],$$

$$NLR^H_3 = \eta[(o^l_{jp} - 0.51)\cdot ((o^{l+1}_{ip} E^{l+1}_{ip}) + (o^l_{jp} - 0.1)(E^{l+1}_{ip} + 0.1))],$$

where $\varepsilon^{l+1}_{ip} = (t_{ip} - o^{l+1}_{ip})$ and the superscripts O and H refer to the output and hidden layer, respectively.

By testing them on the five problems mentioned above we obtained the results in Table 1 which indicate the reliability and efficiency of NLR_1, NLR_2 and NLR_3 in different conditions.

In these tests each algorithm was run with the best set of parameters which we determined empirically. The table reports, for each algorithm, the learning rate η, the minimum, maximum 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 20 independent attempts to train the network, the number of successes (i.e. times in which the learning algorithm succeeded in training the network), and the range of learning rates within which the algorithm converges with a probability of at least 50%. If the network had not converged within the maximum number of epochs, the run was declared unsuccessful. It should be noted that NLR_1 is the best rule discovered in our runs since it works for the XOR, 3-parity, 4-parity, Encoder, and Character Recognition problems converging in 100% of the cases. It should be noted that for the XOR problem the minimum and maximum number of epochs required by NLR_1 to converge is 3 and 42, respectively. This compares very favourably with the minimum and maximum number of epochs required to train an equivalent network with sigmoid activation function using SBP (101 and 870, respectively) or SBP with Rprop (12 and 56, respectively)[16]. Indeed the average number of epochs necessary for convergence with the NLR_1 is 15.8 which compares well with the results reported in [16] for this problem using SBP with Rprop which required on average 25.9 epochs to converge. Similar performance improvements were obtained for the other problems. Also, the range of learning rates within which NLR_1 converges is quite large for most problems indicating that the rule is very stable. The learning rule is also very simple.

By looking at all the learning rules, we can see that each one includes the term $\eta o^l_{jp} \varepsilon^{l+1}_{ip}$ in the output learning rule. This term is the Delta learning rule. So, the experiments with GP suggested that a good learning rule for the output layer of networks with step activation function should include the Delta learning rule. Table 2 analyses the strategy used by NLR_1 to train the hidden layers.

The strategy is simple to understand considering that in our tests the variables o^l_{jp} represent the inputs to the network. Let us consider what happens to the connection between neuron j in the input layer and neuron i in the hidden layer.

Table 1: Performance of the new learning rules discovered by GP.

XOR(2-input parity)						
		Learning Epochs			Successful	
Algorithm	η	Min	Max	Mean	Runs	Range
NLR_1	2.35	3	42	15.8	20	[0.25-10]
NLR_2	2.35	5	35	15.3	20	[0.15-10]
NLR_3	2.35	4	76	28.5	20	[0.15-10]
3-input parity						
		Learning Epochs			Successful	
Algorithm	η	Min	Max	Mean	Runs	Range
NLR_1	0.95	25	364	147.5	20	[0.15-2.25]
NLR_2	0.95	16	266	105.4	20	[0.25-2.25]
NLR_3	0.08	26	401	167.6	20	[0.25-1.55]
4-input parity						
		Learning Epochs			Successful	
Algorithm	η	Min	Max	Mean	Runs	Range
NLR_1	0.42	132	3535	1285.1	20	[0.3-0.5]
NLR_2	0.3	353	10000	2234.8	5	–
NLR_3	–	10000	10000	10000	0	[0.005-1]
Encoder						
		Learning Epochs			Successful	
Algorithm	η	Min	Max	Mean	Runs	Range
NLR_1	0.52	42	305	171.8	20	[0.34-0.82]
NLR_2	0.52	17	391	179.9	20	[0.34-0.82]
NLR_3	0.505	40	288	123.4	20	[0.45-0.68]
Character recognition						
		Learning Epochs			Successful	
Algorithm	η	Min	Max	Mean	Runs	Range
NLR_1	0.2	67	398	194.5	20	[0.1-0.36]
NLR_2	0.2	25	475	190.7	20	[0.1-0.3]
NLR_3	0.28	244	500	387.5	16	[0.28-0.6]

If neuron i is active, the weight will be increased whatever the state of the input neuron. This means that the learning rule tries to increase the probability of the hidden neurons to be active, because only if the hidden neurons are active the error on neurons in the output layer can be reduced (by appropriately setting the weights of the output layer). This in turn causes a reduction of the errors in the hidden layer. However, this strategy of activating the neurons in the hidden layer must be used carefully, since the neurons in the hidden layer have to transmit information from the input layer so that the output neurons can make the right decisions. So, they cannot always be on. This is achieved by the remaining two cases, i.e. when neuron i is inactive.

If no input is present ($o^l_{jp} = 0$) and the state of the hidden neuron is 0 ($o^{l+1}_{ip} = 0$) there is no reason to change the connection. In fact, even if there is an error on neuron i, this could be changed (in future epochs) only by changing the state of neuron i, which cannot be done since $o^l_{jp} = 0$.

Table 2: Analysis of NLR_1.

o^l_{jp}	o^{l+1}_{ip}	NLR^H_1
0	0	0
0	1	0.11 η
1	0	$\eta(E^{l+1}_{ip} - 0.1)$
1	1	0.11η

The most important case, is when the hidden neuron is inactive but the input neuron is active. In this case, the rule considers the error signal on the hidden neuron E^{l+1}_{ip}. If the error is smaller than 0.1 (i.e. it is negative or only weakly positive), then the strength of the connection will be decreased. This is to reduce the probability that the hidden neuron become active. This prevents the hidden neurons from being always active thus allowing the transmission of information through the network. However, if the error is positive and large, then it is better to help this hidden neuron to become active, since this is the only way in which the error on the output layer can be reduced. The information to the output neurons will have to be transmitted by some other hidden neuron.

6 CONCLUSIONS AND FUTURE WORK

In this paper we have applied GP to find supervised learning rules for the output and hidden layers of multi-layer neural networks using step activation functions. No rules exist in the literate to train this kind of neural networks. GP has discovered a useful way of using the Delta learning rule (originally developed for single-layer neural networks) for the output layer. Also, GP has discovered an effective learning rule for the hidden layers. These two learning rules together have performed well on a number of problems. Whether these rules are general remains to be seen.

This study indicates that there are efficient supervised learning algorithms for multilayer neural networks with step activation function and that GP can do better than humans at discovering them.

If the rule discovered by GP to train multi-layer neural nets with step activation function is general, then GP will have done what neural network researchers have been unable to do in the last 40 years: it will have continued the research of the fathers of the field of neural networks, reestablishing the usefulness and reputation of neural networks with step activation function.

Acknowledgements

The authors wish to thank the members of the EEBIC 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, Orlando-Florida, USA*, 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] M. Gori and A. Tesi. On the problem of local minima in backpropagation. *IEEE Transactions on PAMI*, 14(1):76–86, 1992.

[4] S. Haykin. *Neural Networks: A Comprehensive Foundation*. IEEE Society Press, Macmillan College Publishing, New York 10022, 1994.

[5] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.

[6] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982.

[7] K. K. John and R. J. Movellan. Fast learning algorithms for neural networks. *IEEE Transactions on Circuits and Systems-II: Analogy and Digital Signal Processing*, 39(7):453–473, 1992.

[8] H. Kitano. Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms. *Physica D*, 75:225–238, 1994.

[9] J. Koza. *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, Massachussetts, 1992.

[10] L.E.Scales. *Introduction to non-linear optimization.* New York:Springer-Verlag, 1985.

[11] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[12] M. Minsky and S. Papert. Introduction. In *Perceptrons*, pages 1–20, and p,73 (figure 5.1). Cambridge, MA:MIT Press, 1969.

[13] D. J. Montana and L. Davis. Training feedforaward neural networks using genetic algorithms. In *Proceedings of Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89), Detroit, MI*, pages 762–767. Morgan Kaufmann, Palo Alto, CA, 1989.

[14] D. L. Prados. New learning algorithm for training multilayered neural networks that uses genetic-algorithm techniques. *Electronics Letters*, 28(16):1560–1561, 1992.

[15] J. Pujol and R. Poli. Evolving the topology and the weights of neural networks using a dual representation. *Special Issue on Evolutionary Learning of the Applied Intelligence Journal*, 8(1):73–84, 1998.

[16] A. Radi and R. Poli. Discovery of backpropagation learning rules using genetic programming. In *1998 IEEE International Conference of Evolutionary Computational (ICEC'98), Anchorage, Alaska*, pages 371–375. IEEE, May 1998.

[17] A. Radi and R. Poli. Genetic programming can discover fast and general learning rules for neural networks. In *Third Annual Genetic Programming Conference (GP'98), Madison, Wisconsin*, pages 314–323. Morgan Kaufmann, July 1998.

[18] M. 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] F. Rosenblatt. The perceptron: A probabiliistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Parallel Distributed Processing*. MIT Press, Cambridge, MA, 1986.

[21] D. Sarkar. Methods to speed up error back propagation learning algorithm. *ACM Computing Surveys*, 27(4):519–542, 1995.

[22] 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.

[23] W. M. Spears, K. A. D. Jong, T. Baeck, D. Fogel, and H. de Garis. An overview of evolutionary computation. In *Proceedings of the European Conference on Machine Learning*, pages 442–459, 1993.

[24] J. G. Taylor. *The Promise of neural networks*. Springer-Verlag, London, 1993.