Genetic Programming Discovers Efficient Learning Rules for the Hidden and Output Layers of Feedforward Neural Networks

Amr Radi and Riccardo Poli

School of Computer Science
The University of Birmingham
Birmingham B15 2TT, UK
{A.M.Radi,R.Poli}@cs.bham.ac.uk

Abstract. The learning method is critical for obtaining good generalisation in neural networks with limited training data. The Standard BackPropagation (*SBP*) training algorithm suffers from several problems such as sensitivity to the initial conditions and very slow convergence. The aim of this work is to use Genetic Programming (GP) to discover new supervised learning algorithms which can overcome some of these problems. In previous research a new learning algorithms for the output layer has been discovered using GP. By comparing this with *SBP* on different problems better performance was demonstrated. This paper shows that GP can also discover better learning algorithms for the hidden layers to be used in conjunction with the algorithm previously discovered. Comparing these with *SBP* on different problems we show they p rovide better performances. This study indicates that there exist many supervised learning algorithms better than *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 [13]. 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 [25,50].

However, *SBP* presents several drawbacks [10,13,28,41,42,44,46]. For example, it is extremely slow, training performance is sensitive to the initial conditions, it may be trapped in local minima before converging to a solution, and oscillations may occur during learning (this usually happens when the learning rate is too high).

As a consequence, in the past few years a number of improvements to *SBP* have been proposed in the literature (see [44] for a survey). We will review the *SBP* rule and mention some of these recent improvements in Section 2.

R. Poli et al. (Eds.): EuroGP'99, LNCS 1598, pp. 120–134, 1999.

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 it has to be tested extensively to verify its functionality and efficiency. In addition, scientists tend to search the space of possible learning algorithms for neural nets by using a kind of "gradient descent". So, most algorithms newly proposed are not very different from or much better than the previous ones. 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 demonstrate much better performance than *SBP* have been produced in the last 10 years [2,3,19,30,34,35,48].

This 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 [55] which, although not optimum for some domains, offer the broadest possible applicability. Very often the strategy adopted has been to use Genetic Algorithms (GAs) [16] 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.

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 work by Benjio [3], which is also summarised in Section 3, we decided to use Genetic Programming (GP) [21] 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.

In our previous research [39,40], GP was successful in discovering a number of learning rules for the output layers of feed-forward neural networks. Among them we found one which was better than *SBP* in all problems considered. This paper extends significantly that work by applying GP to the discovery of learning rules for the hidden layer.

We describe our approach in Section 4. Section 5 reports the experimental results obtained on six classes of standard benchmark problems: the parity, the encoder-decoder, the character recognition, the multiplexer, the vowel recognition, and the sonar 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 [13]. 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 recent improvements. More details can be found in [42,44].

2.1 Standard Backpropagation

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}^o i.e. with the i^{th} element, x_{ip}, of x_p. 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 neuron u_j^l (usually $o_{jp}^l = O(net_{jp}^l)$ with O 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, in the following, the bias will be interpreted as the weights 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

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

For the hidden layers the error ε_{ip}^l is computed recursively from the errors on other layers (see [13])

The *SBP* rule uses these errors to adjust the weights (usually initialised randomly) in such a way that the errors 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 according to the following equations:

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

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

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 and the error $\varepsilon_{ip}^{l+1}(s)$, and η is a parameter called learning rate.

2.2 Improvements to SBP

Many methods have been proposed to improve generalisation performance and convergence time of SBP. Current research mostly concentrates on: the optimum setting of learning rates and momentum [5,9,18,41,46,51,52,53]; the optimum setting of the initial weights [6,24,27]; the enhancement of the contrast in the input patterns [23,29,32,49,57]; changing the error function [1,9,17,33,45,47]; finding optimum architectures using pruning techniques [7,15]. In the following we will describe two speed-up methods which are relevant to the work described in the rest of the paper: the Momentum and Rprop methods.

The Momentum method implements a variable learning rate coefficient implicitly by adding to the weight change a fraction of the last weight change as follows:

$$\Delta w_{ij}^l(s) = \eta \frac{\partial E(s)}{\partial w_{ij}^l(s)} + \mu \frac{\partial E(s-1)}{\partial w_{ij}^l(s-1)}$$

where μ is a parameter called momentum. This method decreases the oscillation which may occur with large learning rates and accelerates the convergence. For a more detailed discussion see [9,18,51].

Rprop is one of the fastest variations of the SBP algorithm [41,44,56]. Rprop stands for 'Resilient backpropagation'. It is a local adaptive learning scheme, performing supervised batch learning. The basic principle of Rprop is to eliminate the harmful influence of the magnitude of the partial derivative $\frac{\partial E}{\partial w_{ij}^l}$ on the weight changes. The sign of the derivative is used to indicate the direction of the weight update while the magnitude of the weight change is exclusively determined by a weight-specific update-value $\Delta_{ij}^l(s)$ as follows

$$\Delta w_{ij}^l(s) = \begin{cases} -\Delta_{ij}^l(s) & \text{if } \frac{\partial E(s)}{\partial w_{ij}^l(s)} > 0, \\ +\Delta_{ij}^l(s) & \text{if } \frac{\partial E(s)}{\partial w_{ij}^l(s)} < 0, \\ 0 & \text{otherwise.} \end{cases}$$

The update-values $\Delta_{ij}^{l}(s)$ are modified according to the following equation:

$$\Delta_{ij}^{l}(s) = \begin{cases} \eta^{+}\Delta_{ij}^{l}(s-1) & \text{if } \frac{\partial E(s-1)}{\partial w_{ij}^{l}(s-1)} \cdot \frac{\partial E(s)}{\partial w_{ij}^{l}(s)} > 0, \\ \eta^{-}\Delta_{ij}^{l}(s-1) & \text{if } \frac{\partial E(s-1)}{\partial w_{ij}^{l}(s-1)} \cdot \frac{\partial E(s)}{\partial w_{ij}^{l}(s)} < 0, \\ \Delta_{ij}^{l}(s-1) & \text{otherwise.} \end{cases}$$

where η^{-} and η^{+} are constants such that $0 < \eta^{-} < 1 < \eta^{+}$.

The Rprop algorithm has three parameters: the initial update-value $\Delta_{ij}^{l}(0)$ which directly determines the size of the first weight step (default setting $\Delta_{ij}^{l}(0) = 0.1$) and the limits for the step update values Δ_{max} and Δ_{min} which prevent the weights from becoming too large or too small. Typically Δ_{max}=50.0 and Δ_{min}=0.0000001. Convergence with Rprop is rather insensitive to the choice of these parameters. Nevertheless, for some problems it can be advantageous to allow only very cautious (namely small) steps, in order to prevent the algorithm from getting stuck too quickly in local minima. For a detailed discussion see also [41,42].

Although the Momentum method and Rprop are considerably faster than *SBP*, they still suffer from same of the problems mentioned in Section 1 [11,41].

3 Previous Work on the Evolution of Neural Network Learning Rules

A considerable amount of work has been done on the evolution of the weights and/or the topology of neural networks. See for example [20,36,37,38,54]. However only a relatively small amount of previous 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 the optimum learning rules. For example, Montana [31] used GAs for training feedforward networks and created a new method of training which is similar to *SBP*. Chalmers [4] 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 [43]. 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 may be a good way of getting around the limitations inherent to fixed genetic coding which GAs suffer from. 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. However only, one attempt to use GP to induce new learning rules for neural networks has been reported before our own work

Bengio [3] used GP to find learning rules. 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 and was only tested on a very specific problem. 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. Our objective is 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:

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

where o^l_{jp} is the output of neuron u^l_j when pattern p is presented to the network and $E^l_{jp} = \sum_i w^{l+1}_{ji} \delta^{l+1}_{ip}$, 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 preliminary tests in which we tried to evolve both rules at the same time we obtain relatively poor results. This has to be attributed to the huge size of the search space and to the limited memory and CPU power of current workstations.

So, we decided to proceed in two stages. In the first stage, we used GP to evolve rules for the output layer, while the hidden layers were trained with the SBP rule (this was done in previous research [40]). In the second stage, we used GP to evolve rules for the hidden layers, while the output layer was trained with the rule discovered in the first stage. This second stage is described in this paper.

The tasks that the networks are supposed to learn with each learning rule in the population. 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 six 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 [39], 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 [3], 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, 5) the vowel recognition problem for independent speakers of the eleven steady state vowels of British English (for more information see [8]) where each training pattern has 10 input coefficients for each vowel and 4 outputs which represent the eleven different vowels, 6) the classification of sonar signals problem [12], where the task is to discriminate between the sonar signals (60 inputs) bounced off a metal cylinder and those bounced off a roughly cylindrical rock.

For the XOR problem, we used a three-layer network consisting of 2 input, 2 hidden, and 1 output neurons with hyperbolic tangent 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 logistic activation functions with output in the range of [0,1]. For the character recognition problem we used a three-layer network consisting of 7 input, 10 hidden, and 4 output neurons. We used logistic activation functions with output in the range of [-1,1]. For the display problem we used a three-layer network consisting of 4 input, 10 hidden, and 7 output neurons. We used logistic activation functions with output in the range of [0,1]. For the vowels recognition problem we used a three-layer network consisting of 10 input, 8 hidden, and 4 output neurons with logistic activation functions with output in the range of [-1,1]. For the classification of sonar signals we used a three-layer network consisting of 60 input, 12 hidden, and 1 output neurons. We used logistic activation functions with output in the range of [-0.3,0.3] (same range as in [12]). These parameters and network topologies were determined by experimenting with different configurations (number of layers, number of hidden layers, learning rate, and range of random initial weights) and selecting the ones on which the SBP algorithm worked best. We tested each problem with 100 independent runs (i.e each run with different initial weights).

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

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

where C_{min} is the minimum number of epochs needed for convergence and E_{max} and C_{max} are constants such that $f \geq 0$. λ is factor that makes the value of $(E_{max} - E)$ greater than $C_{max} - C_{min}$ in any condition. The value of E is measured at the maximum number of learning epochs. For the XOR, the encoder, character recognition, display, vowel recognition, and sonar signals we used 1000, 200, 500, 500, 300 and 500 epochs, respectively.

The experiments were performed using our own SBP and GP simulators. The simulators are written in POP11 and run on Digital Alpha machines with

233 MHz processors. In these conditions each GP run took six days of CPU time on average. For this reason we were able to perform only 25 GP runs for second stage. In the first stage GP discovered several rules[40]. The best is as follows:

$$NLR_o = \Delta w_{ij}^l = \eta o_{jp}^l \varepsilon_{ip}^{l+1}.$$

In all sample problems considered NLR_o is much faster than SBP even by applying both of NLR_o and SBP with Rprop. This suggests that NLR_o outperforms SBP by allowing bigger weight changes when the output of the neuron is nearly saturated

In the second stage GP was run for 500 to 1000 generations with a population size of 200 to 1000, and a crossover probability of 0.9. After applying crossover we applied subtree mutation with a probability of 0.01 to all of the population. We used the function set $\{+, -, \times\}$, and the terminal set $\{w_{ij}^l, o_{jp}^l, o_{ip}^{l+1}, E_{jp}^l, 1, 0.5, 0.1\}$. We used the "full" initialisation method with an initial maximum depth from 3 to 5 [22].

In these experiments GP was allowed to evolve learning rules for the hidden layers, while the learning rule for the output layer was NLR_o. GP discovered several rules such as

$$NLR_1 = \Delta w_{ij}^l(s) = \eta[1.5 o_{ip}^{l+1}(1 - o_{ip}^{l+1}) E_{jp}^l (o_{jp}^l - 0.5) o_{jp}^l o_{jp}^l]$$

$$NLR_2 = \Delta w_{ij}^l(s) = \eta[o_{jp}^l o_{ip}^{l+1} o_{ip}^{l+1}(1 - o_{ip}^{l+1}) E_{jp}^l + 0.1(o_{ip}^{l+1} - 0.5) o_{jp}^l]$$

$$NLR_3 = \Delta w_{ij}^l(s) = \eta[o_{jp}^l o_{ip}^{l+1}(1 - o_{ip}^{l+1}) E_{jp}^l + 0.1(o_{ip}^{l+1} - 0.55) o_{jp}^l]$$

and

$$NLR_4 = \Delta w_{ij}^l(s) = \eta[o_{jp}^l o_{ip}^{l+1}(1 - o_{ip}^{l+1}) E_{jp}^l + 0.1(o_{jp}^l - 0.55) o_{ip}^{l+1}]$$

We tested these rules on four problems obtaining the results in Tables 3 and 1. It was found that the last two rules (NLR_3 and NLR_4) are the best rules discovered in our runs. We studied them in different conditions to determine their reliability and efficiency with respect to SBP. It should be noted that for the XOR, Encoder, and Display problems the difference between the minimum and maximum number of epochs required by NLR to converge is smaller than for NLR_o although the average number of epochs for NLR_o is better on XOR and Character recognition problems. In any case this compares very favourably with the results obtained with SBP.

By looking to the last two learning rules (NLR_3 and NLR_4), we can see that they include two terms. The first one is the term: $\eta o_{jp}^l o_{ip}^{l+1}(1 - o_{ip}^{l+1}) E_{jp}^l$ which is the SBP rule in the case of logistic activation functions (which has derivative $o_{ip}^{l+1}(1 - o_{ip}^{l+1})$). This makes sense since in Encoder, display and character recognition problems, we use the logistic activation function. However, this is not beneficial for the XOR problem where we used the hyperbolic tangent activation function (which has derivative $1 - (o_{ip}^{l+1})^2$).

It is clear from the Table 3 that the convergence efficiency in the XOR problem increases by changing the term $o_{ip}^{l+1}(1 - o_{ip}^{l+1})$ into $1 - (o_{ip}^{l+1})^2$ to obtain $NLR_3(m)$ and $NLR_4(m)$.

The second term in $NLR_3(m)$ and $NLR_4(m)$ is $\eta 0.1(o_{ip}^{l+1} - 0.55)o_{jp}^{l}$ and $\eta 0.1(o_{jp}^{l} - 0.55)o_{ip}^{l+1}$, respectively. These two equations are similar to the Hebbian learning rule (HB) as explained in by Linsker [26]. A Hebbian rule for synaptic plasticity is one in which a synaptic strength is increased when pre-and post-synaptic firing are correlated, and possibly decreased when are anticorrelated [14]. The Hebb-type learning rule used by Linsker for weigh update is:

$$HB = (o_{ip}^{l+1} - k_1)(o_{jp}^{l} - k_2) + k_3$$

where k_1, k_2, and k_3 are constants. k_3 and either k_1 or k_2 are zero in NLR_6 and NLR_7.

So, the experiments with GP suggested that a good learning rule for the hidden layers could have the form:

$$NLR_h = \eta(\beta SBP + (1 - \beta)HB)$$

which corresponds to:

$$NLR_h = \eta[\beta o_{jp}^{l} o_{ip}^{l+1}(1 - o_{ip}^{l+1})E_{jp}^{l} + (1 - \beta)(o_{ip}^{l+1} - k_1)(o_{jp}^{l} - k_2)]$$

The complete learning rule is NLR_o for the output layer and NLR_h for the hidden layers.

The interest of this result is that GP suggested a NLR in which the weakness of a supervised learning rule (SBP) are removed by combining it with an unsupervised learning rule (HB).

Table 2 shows the results of the two algorithms (SBP and $NLR_o with NLR_h$) for the vowel and sonar problems. By changing the parameters of the NLR_h (β, k_1, and k_2) we found that $\beta = 0.01, k_1 = 0.5$, and $k_2 = 0$ give the best results on the vowel problem while on the sonar problem the best parameters are $\beta = 0.1, k_1 = 0.5$, and $k_2 = 0$.

The combination of NLR_o and NLR_h, which we will term NLR from now on, is very efficient and provides good performance on the six problems. So, in an other set of tests, we have decided to compare its convergence behaviour with the SBP with and without the Momentum and Rprop speed-up algorithms. Figure 1 shows the TSS error of SBP and NLR with and without Momentum on the character recognition problem. The results obtained indicate that NLR achieves its target output at the same epoch as SBP with Momentum, while NLR with Momentum converges much more quickly than the other algorithms. Also, Figure 2 shows that the NLR with Rprop outperforms SBP with Rprop in the character recognition problem. All runs used the same initial random weights.

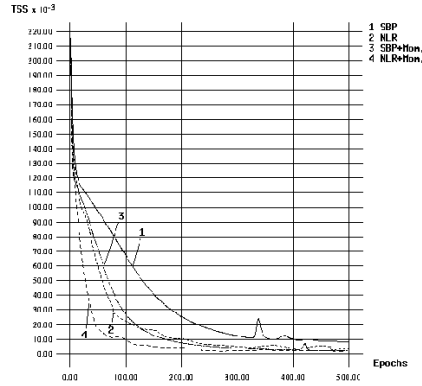

Fig. 1. Plot of the TSS error of SBP, NLR, SBP expand with Momentum, and NLR expand with Momentum in the character recognition problem.

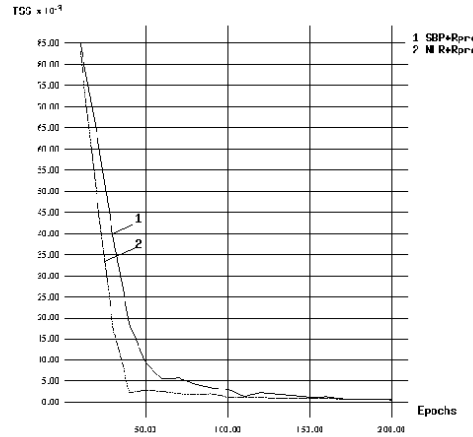

Fig. 2. Plot of the TSS error of SBP with $Rprop$ and NLR with $Rprop$ in the character recognition problem.

6 Conclusions and Future Work

In this paper we have applied GP to find new supervised learning rules for the hidden layers of neural networks. In previous researches GP has discovered for the output layer 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. Then, using this rule to train the output layer GP discovered new learning rules for the hidden layer. In particular GP has discovered a useful way of using the generalised Delta rule and the Hebbian learning rule together. These two learning rules together have performed much better than the SBP learning rule on all the sample problems considered, with and without different speed up algorithms. Whether these rules have the same performed in general remains to be seen.

This study indicates that there are supervised learning algorithms that perform better and are more stable than the SBP learning rule and that GP can discover them.

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. R. Ahmad and F. M. A. Salam. Error back propagation learning using the polynomial energy function. In *Proceedings of the International Joint Conference on Neural Networks, Iizuka, Japan*, 1992.
2. Pierre Baldi. Gradient learning algorithm overview: A general dynamical systems perspective. *IEEE Transactions on Neural Networks*, 6(1):182–195, Jan 1995.
3. 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.
4. D. J. Chalmers. The evolution of learning: An experiment in genetic connectionism. In *Connectionist Models Summer School. San Mateo, CA.*, 1990.
5. X-H Yu G-A Chen. Efficient backpropagation learning using optimal learning rate and momentum. *Neural Networks*, 10(3):517–527, 1997.
6. Vladimir Cherkassky and Robert Shepherd. Regularization effect of weight initialization in back propagation networks. In *1998 IEEE International J. Conference of Neural Networks (IJCNN'98), Anchorage, Alaska*, pages 2258–2261. IEEE, May 1998.
7. C Schittenkopf G Deco and W Brauer. Two strategies to avoid overfitting in feedforward neural networks. *Neural Networks*, 10(3):505–516, 1997.
8. D. H. Deterding. *Speaker Normalisation for Automatic Speech Recognition*. PhD thesis, University of Cambridge, 1989.

9. A. Harry Eaton and L. Tracy Oliver. Improving the convergence of the back propagation algorithm. *Neural Networks*, 5:283–288, 1992.
10. 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.
11. M. Gori and A. Tesi. On the problem of local minima in backpropagation. *IEEE Transactions on PAMI*, 14(1):76–86, 1992.
12. R. P. Gorman and T.J.Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.
13. S. Haykin. *Neural Networks: A Comprehensive Foundation*. IEEE Society Press, Macmillan College Publishing, New York 10022, 1994.
14. Donald O. Hebb. *The Organisation of Behaviour: A Neuropsychological Theory*. New York: Wiley, 1949.
15. Yoshio Hirose, Koichi Yamashit, and Shimpei Hijiya. Back propagation algorithm which varies the number of hidden units. *Neural Networks*, 4:61–66, 1991.
16. J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan, 1975.
17. M. J. J. Holt and S. Semnani. Convergence of back-propagation in neural networks using a log-likelihood cost function. *Electronics Letters*, 26(23):1964–1965, 1990.
18. Robert A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.
19. K. Kruschke John and R. Javier Movellan. Fast learning algorithms for neural networks. *IEEE Transactions on Circuits and Systems-II: Analogy and Digital Signal Processing*, 39(7):453–473, 1992.
20. H. Kitano. Neurogenetic learning: an integrated method of designing and training neural networks using genetic algorithms. *Physica D*, 75:225–238, 1994.
21. John R. Koza. *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, Massachussetts, 1992.
22. John R. Koza. *Genetic Programming II: Automatic discovery of reusable programs*. The MIT Press, Cambridge, Massachussetts, 1994.
23. 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.
24. T Denoeux R Lengelle. Initializing back propagation networks using prototypes. *Neural Networks*, 6(3):351–363, 1993.
25. L.E.Scales. *Introduction to non-linear optimization*. New York:Springer-Verlag, 1985.
26. R. Linsker. From basic network principles to neural architecture: Emergence of spatial-opponent cells. In *Proceedings of the National Acedemy of Science USA*, volume 83, pages 7508–7512, 1986.
27. F. A. lodewyk Wessels and Etienne Barnard. Avoiding false local minima by proper initialisation of connections. *IEEE Transactions on Neural Networks*, 3(6):899–905, 1992.
28. Howard Demuth Martin Hagan and Mark Beale. *Neural Network Design*. PWS Publishing Company, Boston, MA 02116, 96.
29. Rangachari Anand Kishan Mehrotra Chilukuri Mohan and Sanjay Ranka. An impproved algorithm for neural network classification of imbalanced training sets. *IEEE Transactions on Neural Networks*, 4(6):962–969, 1993.
30. Martin F. Moller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533, 1993.
31. 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.

32. Taek Mu and Hui Cheng. Contrast enhancement for backpropagation. *IEEE Transactions on Neural Networks*, 7(1):515–524, 1996.
33. A. Van Ooyen and B Nienhuis. Improving the convergence of the back propagation algorithm. *Neural Networks*, 5:465–471, 1992.
34. Alexander G. Parlos, B. Fermandez, 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.
35. 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.
36. J. Pujol and R. Poli. Efficient evolution of asymmetric recurrent neural networks using a pdgp-inspired two-dimensional representation. In *the First European Workshop on Genetic Programming(EUROGP'98), Paris, Springer Lectre Notes in Computer Science*, volume 1391, pages 130–141, 1998.
37. J. Pujol and R. Poli. Evolving neural networks using a dual representation with a combined crossover operator. In *the IEEE International Conference on Evolutionary Computation (ICEC'98), Anchorage, Alaska*, pages 416–421, 1998.
38. 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.
39. Amr Radi and Riccardo 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.
40. Amr Radi and Riccardo 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.
41. 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.
42. Martin Riedmiller and Heinrich Braun. A direct method for faster backpropagation learning: The RPROP Algorithm. *IEEE International Conference on Neural Networks 1993 (ICNN93), San Francisco*, pages 586–591, 1993.
43. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Parallel Distributed Processing.* MIT Press, Cambridge, MA, 1986.
44. Dilip Sarkar. Methods to speed up error back propagation learning algorithm. *ACM Computing Surveys*, 27(4):519–542, 1995.
45. Wolfram Schiffmann and Merten Joost. Speeding up backpropagation algorithm by using cross-entropy combined with pattern normalization. *International Journal of Uncertainity, Fuzziness and Knowledge–based Systems (IJUFKS),*, 6(2):177–126, 1998.
46. Femando F. Silva and Luis B. Almeida. *Speeding Backpropagation*, pages 151–158. Advanced Neural Computers, Elsevier Science Publishers B.V. (North-Holland), 1990.
47. Sara A. Solla, Esther Levin, and Michael Fleisher. Accelerated learning in layered neural networks. *Complex System*, 2:625–640, 1988.
48. Alessandro Sperduti and Antonina Starita. Speed up learning and network optimization with extended back propagation. *Neural Networks*, 6:365–383, 1993.
49. Ramana Vitthal P. Sunthar and Ch.Durgaprasada Rao. The generalized proportional-integral-derivative (pid)gradient descent back propagation algorithm. *Neural Networks*, 8(4):563–569, 1995.
50. J. G. Taylor. *The Promise of neural networks.* Springer-Verlag, London, 1993.

Table 1. Summary of Table 3 where Yes = better than SBP and No = worse than SBP.

Rules	Problems			
	XOR	Encoder	character recognition	Display
$NLR_o + SBP$	Yes	Yes	Yes	Yes
$NLR_o + NLR_1$	No	No	Yes	Yes
$NLR_o + NLR_2$	No	Yes	Yes	Yes
$NLR_o + NLR_3$	No	Yes	Yes	Yes
$NLR_o + NLR_4$	No	Yes	Yes	Yes
$NLR_o + NLR_3(m)$	Yes	–	–	–
$NLR_o + NLR_4(m)$	Yes	–	–	–

Table 2. Performance of $NLR_o with NLR_h$ and SBP on two hard problems.

Vowel					
		Learning Epochs			Successful
Algorithm	η	Min.	Max.	Mean	Runs
SBP	0.08	124	273	256.9	100
$NLR_o + NLR_h$	0.08	120	200	165.8	100
Sonar					
		Learning Epochs			Successful
Algorithm	η	Min.	Max.	Mean	Runs
SBP	0.1	373	420	396.7	100
$NLR_o + NLR_h$	0.1	117	167	138.1	100

51. Tom Tollenaere. Super SUB:Fast adaptive back propagation with good scaling properties. *Neural Networks*, 1:561–573, 1990.
52. W.T.Zink T.P.Vogl, J.K. Zigler and D.L.Alkon. Accelerating the convergence of the backpropagation method. *Biological Cybernetics*, 59:256–264, Sept. 1988.
53. Michael K. Weir. A method for self determination of adaptive learning rate in back propagation. *Neural Networks*, 4:(371–379), 1991.
54. 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, Georg Mason University,*, pages 391–396. Morgan Kaufmann, 1989.
55. 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.
56. Merten Joost Wolfram Schiffmann and R. Werner. Optimisation of the backpropagation algorithm for training multilayer perceptrons. Technical report 16/1992, University of Koblenz, Institute of Physics, 1992.
57. Byoung-Tak Zhang. Accelerated learning by active example selection. *International Journal of Neural Systems*, 5(1):67–75, 1994.

Table 3. Performance of the discovered learning rules on four different problems.

XOR					
		Learning Epochs			Successful
Algorithm	η	Min.	Max.	Mean	Runs
SBP	0.96	101	870	163.95	100
$NLR_o + SBP$	0.96	50	197	63.39	100
$NLR_o + NLR_1$	0.96	44	1000	832.54	18
$NLR_o + NLR_2$	0.96	1000	1000	1000	0
$NLR_o + NLR_3$	0.96	16	1000	926.8	16
$NLR_o + NLR_4$	0.96	2	1000	688.26	17
$NLR_o + NLR_3(m)$	0.96	49	1000	161.68	89
$NLR_o + NLR_4(m)$	0.96	58	88	70.95	100
Encoder					
		Learning Epochs			Successful
Algorithm	η	Min.	Max.	Mean	Runs
SBP	0.4	44	303	118.67	100
$NLR_o + SBP$	0.4	15	108	36.85	100
$NLR_o + NLR_1$	0.4	500	500	500	0
$NLR_o + NLR_2$	0.4	15	105	50.66	100
$NLR_o + NLR_3$	0.4	10	118	34.74	100
$NLR_o + NLR_4$	0.4	16	78	34.37	100
Character Recognition					
		Learning Epochs			Successful
Algorithm	η	Min.	Max.	Mean	Runs
SBP	0.3	179	358	226.54	100
$NLR_o + SBP$	0.3	34	76	51.29	100
$NLR_o + NLR_1$	0.3	43	94	61.26	100
$NLR_o + NLR_2$	0.3	71	331	149.35	100
$NLR_o + NLR_3$	0.3	54	160	83.21	100
$NLR_o + NLR_4$	0.3	40	108	64.31	100
Display					
		Learning Epochs			Successful
Algorithm	η	Min.	Max.	Mean	Runs
SBP	0.8	130	500	250.514	87
$NLR_o + SBP$	0.8	46	204	79.069	100
$NLR_o + NLR_1$	0.8	46	180	79.94	100
$NLR_o + NLR_2$	0.8	71	331	149.35	100
$NLR_o + NLR_3$	0.8	41	108	66.81	100
$NLR_o + NLR_4$	0.8	46	116	72.6	100