Evolution and Design of Distributed Learning Rules

Thomas Philip Runarsson
Department of Mechanical Engineering
University of Iceland
Hjardarhagi 2-6, 107 Reykjavik, Iceland.
tpr@verk.hi.is

Magnus Thor Jonsson
Department of Mechanical Engineering
University of Iceland
Hjardarhagi 2-6, 107 Reykjavik, Iceland.
magnusj@verk.hi.is

Abstract- The paper describes the application of neural networks as learning rules for the training of neural networks. The learning rule is part of the neural network architecture. As a result the learning rule is non-local and globally distributed within the network. The learning rules are evolved using an evolution strategy. The survival of a learning rule is based on its performance in training neural networks on a set of tasks. Training algorithms will be evolved for single layer artificial neural networks. Experimental results show that a learning rule of this type is very capable of generating an efficient training algorithm.

1 Introduction

The work describes a novel approach for the training of neural networks. In essence we propose that a learning rule be part of the neural network structure and as such itself a network and globally distributed. Although the method is a non-local learning rule it does not require an 'error' signal to be propagated directly to the neuron in question. The 'error' signal will instead be channeled through the learning rule, i.e. the network. The viewpoint expressed challenges the idea that 'learning rules' must be local for biological plausibility. Indeed how can a simple local rule give rise to anything cognitively significant like "learning"? If it does, it must be the result of some higher-order organization of the system as a whole.

The 'biological implausibility' of learning algorithms is commonly debated in the literature [1], for example: "it is unlikely that animal networks employ any of the mechanisms used in training back-propagation networks (gradient descent, stochastic descent, genetic algorithms, etc.) because they all use information from across the entire network to decide how to adjust the value of an individual weight" [2]. It is for this reason that Hebb-like [3] local learning rules are regarded as biologically plausible. However, there do exist many return pathways in the brain, although they may not carry all the information needed for back-propagation learning [4]. Another move towards biological relevance [5] are reinforcement learning methods [6, 7]. With these methods learning may be achieved by a single reinforcer. A part from these arguments there is the issue of the implementation of the 'learning rule' itself.

Previous attempts for evolving learning rules have relied on predefined functions [4, 2, 8]. For example, a simple quadratic function (where the delta rule could be found as a special case) with evolvable parameter was used by Chalmers [4]. Chalmers recommends genetic programming for the evolution of even more complicated learning rules [4]. We will show that sophisticated learning rules may be found using the neural networks themselves. It is well known that a neural network is a capable function approximator and for this reason quite able to both approximate and implement any learning rule. A single hidden layer is sufficient for a multilayer perceptron to be a universal approximator [9, 10, 11]. The learning rules will be discovered by an evolutionary search method. The implications of such a system may be the following:

- faster trainers for given problem classes
- a new approach to learning
- different types of feedback (reinforcers)
- a greater variety of learning rules

In the current work we look at the single layered neural network only. Because the decision boundary is linear, the single-layer network can only be used to recognize patterns that are linearly separable. However, the working principles developed for this type of network is fundamental to more complicated neural network structures. Section 2 describes the learning rule and then the evolutionary algorithm used to evolve this rule is formulated in section 3. Experimental studies for some linearly separable problems are presented in section 4 and the paper is concluded with a summary and discussion.

2 Learning Rule

A learning rule is a procedure for modifying the weights and biases of a network, this procedure is also referred to as the training algorithm. The weight updating rule at time (t) is given by:

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij} \tag{1}$$

where w_{ij} is the weight of the connection from node j to node i. The commonly used learning rule for the change

in the weights Δw_{ij} is

$$\Delta w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}} \tag{2}$$

where E_p is the error for pattern p, or by the delta rule [12]:

$$\Delta w_{ij} = \eta \delta_{ip} o_j \tag{3}$$

The change in the weights is a function of the *learning rate* η which is typically less than 1 (e.g. 0.1 or 0.3). A small learning rate is chosen because the final weights we seek should work for all patterns and so we must be cautious when changing the weights per pattern. The error term δ_{ip} is the product of the actual error output, which is the difference between the target for node i, denoted by t_{ip}, and the actual output of that node o_{ip}, multiplied by the derivative of the node's activation function. The error for the node is also related to the signal from the sending node, which is o_j. For a pure linear activation function we have for the output layer:

$$\Delta w_{ij} = \eta (t_{ip} - o_{ip}) o_j \tag{4}$$

and for the commonly used sigmoid activation function:

$$\Delta w_{ij} = \eta o_{ip}(1 - o_{ip})(t_{ip} - o_{ip}) o_j \tag{5}$$

In other words the learning rule is a function of the input and output node activations and the target value:

$$\Delta w_{ij} = \eta \ f(o_j, o_{ip}, t_{ip}) \tag{6}$$

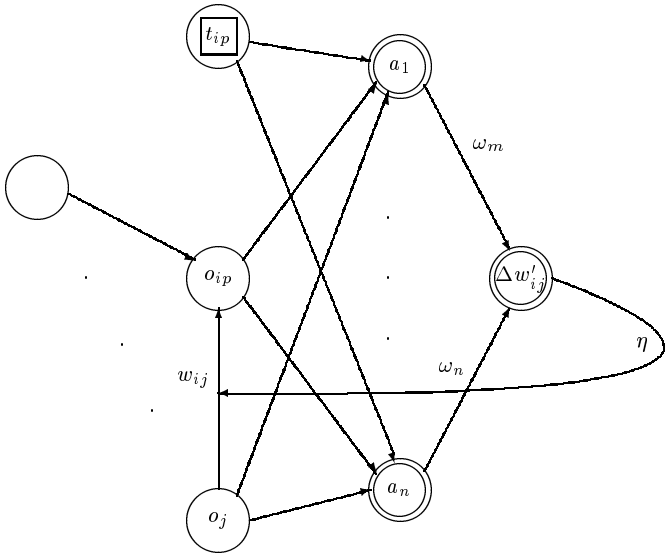

Figure 1: The neural network structure (bias is not shown). The double circular nodes are part of the learning rule, the node with the square inside is the target (or reinforcer) and the rest of the nodes are part of the neural network being trained.

This function may be approximated by a neural network and hence our proposal to use another part of the network to model the learning rule. Computing the error term for hidden nodes is usually done by letting them inherit the errors of all nodes that they activate and so a more general learning rule would take the form:

$$\Delta w_{ij} = \eta \ f(o_j, o_{ip}, \delta_{ip}) \tag{7}$$

where now δ_{ip} reflects in some way the 'error' assigned to w_{ij} by the principles of credit and blame [12],[6], or reinforcement [7].

The neural network approximating equations (6) or (7) would have an output value of $\Delta w'_{ij}$, which may be either positive of negative. For this reason the output node's activation function must allow for both signs, using the tan-sigmoid function the learning rule becomes:

$$\Delta w_{ij} = \eta \Delta w'_{ij} = \eta \left(\frac{2}{1 + \exp(-2 \sum_k \omega_k a_k)} - 1 \right) \tag{8}$$

where ω_k are the weights of the learning rule and a_k are the signals from the hidden nodes' activation. Other activation functions may be equally acceptable but the tan-sigmoid function will be used in our study. The inputs for the neural network, the learning rule, are the function arguments in equation (6) or (7). In Chalmers' [4] quadratic function the weights themselves, w_{ij}, are part of the learning rule, however, we see no reason for including them here. The complete neural network structure is given in figure 1 for a learning rule with one hidden layer training a single layer network.

3 Evolutionary Algorithm

The evolutionary optimization algorithm presented is based on the evolution strategy, ES(μ, λ), [13]. It follows the typical cycle for the evolution of learning rules [14] and may be formulated as follows:

1. Initialize a population of λ individuals randomly, set the generation counter to $g = 0$ and the desired maximum number of generations G. Each individual is a vector $(\omega_1, \ldots, \omega_n, \sigma_1, \ldots, \sigma_n)$ where $\vec{\omega} = (\omega_1, \ldots, \omega_n)$ are the neural network weights for the learning rule, and $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ are the 'mean step' sizes adapted during the search. The initial weights are randomly distributed between -1 and 1 and the initial mean step sizes are $2/\sqrt{n}$.

2. Use each learning rule $\vec{\omega}_i \ \forall \ i \in \{1, \ldots, \lambda\}$ to train a neural network on a given set of training patterns. The initial weights $\vec{w}$ of the network to be trained are randomly distributed between -1 and 1. A number of different training patterns are used. The fitness of the training algorithm, $\vec{\omega}_i$, is the averaged 'mean root square error' of the last epoch.

Table 1: Chalmers' eight linearly separable tasks [4].

o_1	o_2	o_3	o_4	o_5	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}
1	1	1	1	1	1	0	0	0	1	0	1	0
0	0	0	0	0	0	0	1	0	0	1	0	1
0	1	1	1	0	0	0	1	0	1	0	1	0
1	1	0	0	0	0	1	1	1	0	1	1	1
1	0	1	0	1	1	0	0	0	0	0	1	1
0	1	1	0	0	0	0	1	0	1	1	0	1
0	1	1	1	1	1	0	1	0	1	0	1	0
0	1	0	0	0	0	0	1	1	1	1	0	1
1	1	0	0	1	1	0	0	1	0	1	1	1
1	0	0	1	0	0	0	1	1	1	0	1	1
1	0	1	1	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	0	0	1

3. Select the best μ individuals to generate on average λ/μ offspring for the next generation, we set $\mu/\lambda \approx 1/7$ [15, page 79].

4. The 'mean step sizes' are updated according to Schwefel's log-normal update rule [13]: $i = 1, \ldots, \mu$, $h = 1, \ldots, \lambda$, and $j = 1, \ldots, n$,

$$\sigma_{h,j}^{(g+1)} = \sigma_{i,j}^{(g)} \exp(\tau' N(0,1) + \tau N_j(0,1))), \quad (9)$$

where $N(0,1)$ is a normally distributed one-dimensional random variable with an expectation of zero and variance one. The subscript j in $N_j(0,1)$ indicates that the random number is generated anew for each value of j. The 'learning rates' τ and τ' are approximately equal to $\varphi^*/\sqrt{2\sqrt{n}}$ and $\varphi^*/\sqrt{2n}$ respectively where φ^* is the expected rate of convergence [13, page 144] and is usually set to one [15, page 72].

5. The neural network weights are varied using the normal distribution:

$$\omega_{h,j}^{(g+1)} = \omega_{i,j}^{(g)} + \sigma_{h,j}^{(g+1)} N_j(0,1) \quad (10)$$

6. Increment the generation counter $g = g + 1$ and go to step 2 unless the maximum number of generations (G) has been reached.

The algorithm presented is a simple evolution strategy which does not use any recombination. Some of the factors which we expect will influence the search for learning rules are:

- whether the training samples can be learned by the chosen network architecture
- the number of training samples used (the set must cover the problem space 'sufficiently')
- the number of epochs used in training
- the complexity of the learning rules (the number of hidden nodes)
- repeated training on the same task in order for the results to be statistically significant.

Some of these issues will be examined in the following experimental study.

4 Experimental Study

Linear prediction is a task for which single-layer networks are commonly used. Here we will experiment with Chalmers' [4] eight linearly separable tasks given in table 1. The tasks have only one output unit since a single layer network with more than one output unit is equivalent to a number of disjointed networks with a single output unit [4]. Task 3 is, however, not linearly separable and therefore will not be used in our experiments. We will examine the effect of learning rule complexity and training time in this study.

The input and outputs are zeros and ones and the activation function for the output node of this single layer network is the log-sigmoid activation function:

$$o_{ip} = \frac{1}{1 + \exp(-\sum_j w_{ij} o_j)} \quad (11)$$

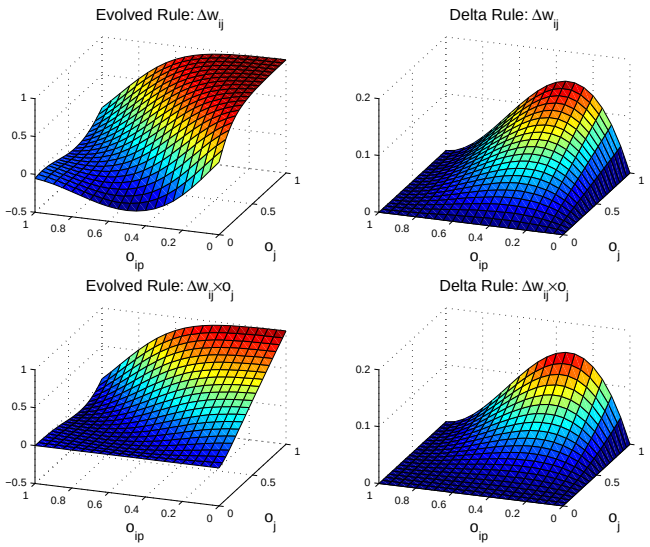

Figure 2: The target value is $t_{ip} = 1$ and the learning rate is one. The learning rule has 20 hidden nodes and a training time of 60 epochs during the evolution.

Table 2: The last epoch error for the best rules found for each experiment.

F	I	G	B	E	H	A	C	D
0.0059	0.0070	0.0086	0.0120	0.0127	0.0137	0.0138	0.0227	0.0240

Both the learning rule and the network being trained use a bias which is not shown in figure 1.

The evolution strategy, ES$(30, 200)$, was run for 400 generation on the following experimental setups:

	10	20	30	hidden nodes
10	A	D	G	
30	B	E	H	
60	C	F	I	
epochs				

where $A, \ldots, I$ denote the experiment number. Each of the experiments was conducted five times with a learning rate of $\eta = 1/2$.

In all of the experiments similar response surfaces were evolved. We begin by comparing the response surface for experiment F with that of the delta rule given by equation (5). This comparison is shown in figure 2 for a target value of 1 and in figure 3 for a target value of 0. The figures also illustrate the actual change in the network behaviour by multiplying the change in weight by the input signal o_j.

The striking difference between the evolved networks and the delta rule is that the delta rule does not change its weights when the output node is zero or one, regardless of the error signal. For the evolved network, when the target value is $t_{ip} = 1$ (see figure 2), the change in weights is minimal when $o_{ip} \approx 1$, but when $o_{ip} \approx 0$ the change is maximal but again little for $o_j \approx 0$. Similar behaviour is observed for when the target value is 0 (see figure 3), that is maximal change when $o_{ip} \approx 1$ but again minimal for $o_{ip}, o_j \approx 0$. When we multiply the input signal to the weight change as shown in both figures 2 and 3 we notice that the major difference is that *the delta rule does not modify weights when the error is at the extremum, however, the evolved network does.*

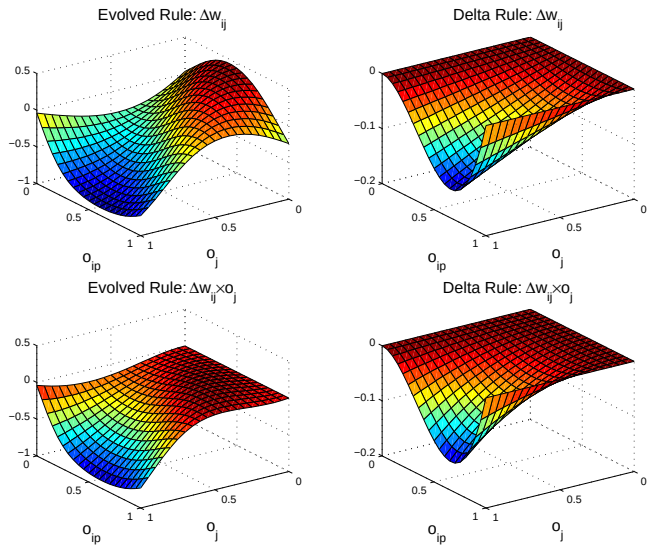

Figure 3: The target value is $t_{ip} = 0$ and the learning rate is one. The learning rule has 20 hidden nodes and a training time of 60 epochs during the evolution.

The best evolved learning rules for each of the experiments conducted was used to train a network on Chalmers' tasks again but now for 200 epochs. The last epoch error for these learning rules, sorted in order of best to worst performance, are given in table 2 above. None of the training rules diverged but fluctuated slightly about a fixed error level. From these experiments is seems that the more complicated learning rules, with 20 and 30 hidden layers, performed better. Also, a longer training time resulted in a learning rule with lower errors.

Figure 4 shows the evolution of error as a function of epoch for the best learning rule from experiments A and F. For comparison these tasks were trained using the delta rule with a η value five times higher than that used for the evolved rule (see figures 2 and 3). Indeed, a very large η value is needed for the delta rule to achieve the same learning rate as the evolved rules, however, in this case the weights do oscillate [16, page 81].

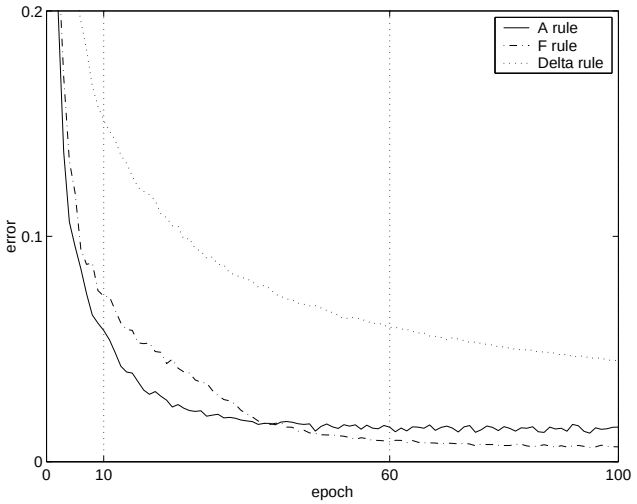

Figure 4: The learning process for the evolved rules from experiment A and F and using the delta rule on Chalmers' tasks.

The performance of the evolved rules on other unseen tasks are also good, even for Chalmers' third task, which is not linearly separable, all but one pattern was solved. A similar result is obtained using the delta rule on this task. As a further example consider the following two cases: the logical `OR` function and the logical `AND` function. The training results for these cases are depicted in figure 5. It is clear from this demonstration that the evolved learning rules are biased.

5 Summary and discussion

The analysis presented in this paper are preliminary. The experimental results may be summarized as follows: 1.) a neural network is capable of representing a learning rule, 2.) the evolved rules are fast neural network trainers, 3.) the training algorithms do not diverge when a longer period of learning is used, 4.) the accuracy achieved is limited, and 5.) the learning rule is biased. Our results indicate also that the shorter the training time given to the evolved learning rule the less accurate the results but faster the learning. Conversely the longer the training time the more accurate the results but slower the learning. This seems reasonable since even an algorithm like back-propagation requires a certain number of epochs to reach a quality solution. All of the learning rules evolved have a limited accuracy and appear to be noisy. However, a network approximating equation (5) would also make for a 'noisy' learning rule.

The learning rule is biased. It is 'tuned' to solve a given problem class fast. Even using the back-propagation algorithm the tuning of learning and momentum rates is often required when new problems are solved. Note that a recurrent neural network structure could be used to model a learning rule with momentum.

The next steps are to model rules using similar inputs to those used in reinforcement learning for the training of neural networks with hidden layers. We believe that the approach taken here opens up many possibilities for evolving fast problem specific training algorithms.

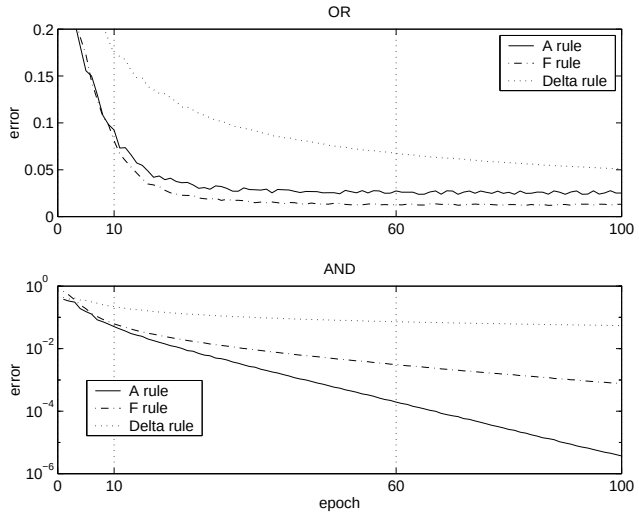

Figure 5: The learning process for the evolved rules from experiment A and F and using the delta rule on the logical `OR` (top) and `AND` (bottom) functions.

Acknowledgement

The first author would like to thank professors Jorgen Pind and John Stuart for many helpful discussions. This work has been supported by the Research Council of Iceland, hereby also gratefully acknowledged.

References

[1] I.E. Dror and D.P. Gallogly. Computational analysis in cognitive neuroscience: In defence of biological implausibility. *Psychonomic Bulletin & Review*, 6(2):173–182, 1999.

[2] J. Baxter. The evolution of learning algorithms for artificial neural networks. *Complex Systems*, 1992.

[3] D.O. Hebb. *The Organization of Behaviour*. Wiley, New York, 1949.

[4] D. Chalmers. The evolution of learning: An experimentation in genetic connectionism. In *Connectionists models: Proceedings of the 1990 summer school*, pages 81–90. San Mateo, CA: Morgan Kaufmann Publishers, 1990.

[5] P. Mazzoni, R.A. Andersen, and M.I. Jordan. A more biological plausible learning rule for neural networks. *Proc. Natl. Acad. Sci. USA*, 88:4433–4437, May 1991.

[6] B. Widrow, N.K. Gupta, and S. Maitra. Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Trans. Syst. Man Cybern.*, 4:455–465, 1973.

[7] A.G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Trans. Syst. Man Cybern.*, 4:360–375, 1985.

[8] S. Bengio, Y. Bengio, and J. Cloutier. On the search for new learning rules for ANNs. *Neural Processing Letters*, 2(4):1–5, 1995.

[9] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[10] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward network are universal approximators. *Neural Networks*, 2:359–366, 1989.

[11] G. Cybenko. Approximation by superposition of a sigmoid function. *Mathematics of Control, Signal and Systems*, 2:303–314, 1989.

[12] D.E. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Vol. 1. Foundations*, pages 318–362. Cambridge, MA: MIT Press, 1986.

[13] H-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New-York, 1995. Series: Sixth-Generation Computer Technology.

[14] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[15] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[16] K. Mehrotra, C.K. Mohan, and S. Ranka. *Elements of Artificial Neural Networks*. The MIT Press, Cambridge, Massachusetts, 1997.