Evolving Efficient Learning Algorithms for Binary Mappings

John A. Bullinaria
School of Computer Science, The University of Birmingham
Birmingham B15 2TT, UK

Abstract- **Gradient descent training of sigmoidal feed-forward neural networks on binary mappings often gets stuck with some outputs totally wrong. This is because a sum-squared-error cost function leads to weight updates that depend on the derivative of the output sigmoid which goes to zero as the output approaches maximal error. Although it is easy to understand the cause, the best remedy is not so obvious. Common solutions involve modifying the training data, deviating from true gradient descent, or changing the cost function. In general, finding the best learning procedures for particular classes of problem is difficult because each usually depends on a number of interacting parameters that need to be set to optimal values for a fair comparison. In this paper I shall use simulated evolution to optimise all the relevant parameters, and come to a clear conclusion concerning the most efficient approach for learning binary mappings.**

I. Introduction

It is inevitable that the most efficient neural network learning algorithm for a given problem will depend on the problem class being considered. Simple gradient descent based learning algorithms using sum-squared-error cost functions work well for a wide range of problems, but are well known to run into difficulties when used with sigmoidal feed-forward neural networks that need to learn binary mappings. This is because the weight updates depend linearly on the derivative of the output activation function, and for sigmoidal activation functions that derivative tends to zero as the sigmoids saturate at either the correct or the maximally incorrect output. This means it is theoretically possible for the network to get stuck with some outputs totally wrong, and in practice this is actually quite common in realistic applications. Early on in training some patterns (e.g. the most regular or frequent) will dominate the weight changes and force the other patterns into errors. By the time the errors on the dominant patterns have reduced to small values, the other patterns are stuck. We could consider abandoning the sigmoids and use linear output units instead, but then we would lose the natural interpretation of the outputs in terms of posterior probabilities (e.g. Golden, 1988; Bishop, 1995).

Perhaps the most obvious solution that keeps the sigmoids, which dates back to the re-discovery of the back-propagation learning algorithm, is to simply offset the target outputs (Rumelhart, Hinton & Williams 1986). Instead of using the actual binary targets of 0 and 1, one offsets them slightly to 0.1 and 0.9 (say) so the sigmoids never saturate and the weight update signals no longer go to zero when the wrong target is reached. We effectively restrict the outputs to the near linear central region of the sigmoids. If we then set output response thresholds at 0.2 and 0.8 (say) it will make no difference to the binary network responses, but it may have other side effects, such as slowing down the training, and destroying the interpretation of the output values.

Another early solution to the problem involves offseting the sigmoid derivatives (a.k.a. sigmoid prime) directly by adding a small constant (say 0.1) to them (Fahlman, 1988). This Sigmoid Prime Offset (SPO) approach will clearly prevent the weight update signals from going to zero when the wrong target is reached, but it also means that the learning algorithm is no longer performing true gradient descent on the cost function, and it is not obvious what effect that will have on the speed of learning.

It will obviously be rather important for both of these approaches to choose good values for the offset parameters, that are large enough to be sure of preventing the outputs from getting stuck at the wrong values, but small enough to result in minimal loss of learning efficiency. Some degree of experimentation, and perhaps a liking for round numbers, has led to offsets of 0.1 being more or less standard for all problems. However, it is not clear that these really are the optimal values, nor is it clear which form of offset is best to use. The problem is that the best values of the other learning algorithm parameters (such as the learning rates and momentum) and the magnitude of the chosen offset will depend on each other, and also on the type of offset used.

To complicate matters further, another proposed solution has been to replace the sum-squared-error gradient descent cost function with the cross-entropy cost function (Hinton, 1989; Van Ooyen & Nienhuis, 1992). This is known to be a more appropriate cost function for classification problems with network outputs that are to be interpreted as specifying probability distributions over binary vectors (Hinton, 1989; Bishop, 1995). In this case the sigmoid derivative simply cancels out of the weight update equation, and so we never have the problem of it going to zero for totally wrong outputs. However, again we have a problem comparing this against the two offset approaches because the optimal learning algorithm parameters are likely to be different.

Whichever approach is used to avoid getting stuck with the wrong outputs, one should presumably then continue the training with the right cost function and no offsets if we want

to recover the interpretation of non-binary output activations as probabilities. Once all the output errors are relatively small, there is much less chance of patterns being forced back into error and getting stuck again. The big question is: which approach provides the most efficient (i.e. fastest) learning to get us to that final stage, and how do we determine the appropriate learning and offset parameters?

We are fortunate that increasing computational resources have, over recent years, rendered it feasible to optimize such learning parameters using evolutionary strategies (Yao, 1999), and thus ensure than we are comparing each approach when performing as best they can. In the remainder of this paper I shall describe the Target Offset, Sigmoid Prime Offset, and Cross Entropy approaches in more detail, and present the results from a series of evolutionary simulations that optimize each case for a representative pair of binary mappings. We end with a clear conclusion concerning which approach is best.

II. The Neural Network Models

The procedures for training feed-forward neural networks by gradient descent are now well known (e.g. Hinton, 1989; Bishop, 1995), so I shall simply summarize my notation here. We define an appropriate cost or error function E, and compute a smoothed iterative series of updates

$$\Delta w_{ij}(n) = -\eta_L \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}(n-1)$$

for each weight w_{ij} which reduce that function, where n is the epoch of training. Past experience indicates that networks learn better if they have different learning rates η_L for each connection layer, and each bias set. So, to ensure that each network learns at its full potential, each has five learning parameters: the learning rate η_{IH} for the input to hidden layer, η_{HB} for the hidden layer biases, η_{HO} for the hidden to output layer, and η_{OB} for the output biases, and the momentum parameter α. The initial network weights $w_{ij}(0)$ are generated randomly with a uniform distribution from the range $[-iw_L, +iw_L]$. Naturally, different initial weight range parameters iw_L are allowed for the input to hidden layer connections, the hidden layer biases, the hidden to output layer connections, and the output biases.

With the standard sum squared error (SSE) cost function

$$E = \tfrac{1}{2} \sum_j \left| t_j - o_j \right|^2$$

the output layer weight derivatives are

$$\frac{\partial E}{\partial w_{ij}} = h_i.(t_j - o_j).\left[(1-o_j).o_j\right]$$

where t_j are the binary target network outputs, o_j are the actual outputs, and h_i are the hidden unit activations. The term in square brackets is the problematic sigmoid derivative that goes to zero as the sigmoids saturate. We thus consider

$$\frac{\partial E}{\partial w_{ij}} \approx h_i.(t_j - o_j).\left[(1-o_j).o_j + spo2\right]$$

where *spo2* is the sigmoid prime offset that would be zero if the derivative were performed exactly. We can also allow the possibility of a similar sigmoid prime offset *spo1* at the hidden layer. The second approach we consider is to offset the output targets and take them to be *toff* and 1–*toff*, rather than 0 and 1, with appropriate outputs beyond these targets deemed errorless.

The third approach is to employ a better network cost function. For the cross-entropy (CE) cost function

$$E = -\sum_j \left[t_j.\log(o_j) + (1-t_j).\log(1-o_j) \right]$$

the output layer weight derivatives simplify to

$$\frac{\partial E}{\partial w_{ij}} = h_i.(t_j - o_j)$$

Clearly, we should need no offsets here, and indeed there is no place for an *spo2*, but we can still check to see if there is any advantage in having a non-zero *spo1* at the hidden layer, or an output target offset *toff*.

III. Evolving The Models

The goal here is to optimise our neural network models using evolution by natural selection. We can simulate such a process for the systems discussed above by taking a whole population of individual instantiations of each model and allowing them to learn, procreate and die in a manner approximating these processes in real (biological) systems. The genotype of each individual will depend on the genotypes of its two parents, and contain all the appropriate innate parameters. Then, throughout its life, the individual will learn from its environment how best to adjust its weights to perform most effectively. Each individual will eventually die, perhaps after producing a number of children.

For biological evolution, the ability of an individual to survive or reproduce will depend on a number of factors which can vary in a complicated manner on that individual's performance over a range of related and unrelated tasks (food gathering, fighting, fleeing, and so on). For current purposes it is appropriate and sufficient to assume a simple relation between our single task performance and the survival or procreation fitness. Whilst any monotonic relation should result in similar evolutionary trends, we often find that, in simplified simulations, the details can have a big effect on what evolves and what gets lost in the noise.

A more natural approach to procreation, mutation and survival will be followed than many evolutionary simulations have used in the past, such as those described by Yao (1999). Rather than simultaneously training all members of the current population and picking the fittest to breed and form the next generation, the populations will contain competing learning individuals of all ages, each with the potential for dying or procreation at each stage. The alternative earlier approaches typically set the number of epochs of training per generation and base the fitness on the remaining error, or base the fitness on the number of epochs required to reduce the error to a set value. Both of these procedures are problematic in that they involve parameters (the number of epochs or the target fitness) that are difficult to fix when the ability to learn is changing from generation to generation.

In my approach, each individual will learn from their own experience with the environment (i.e. set of training/testing data) and have their fitness determined each simulated year. A fitness biased random subset of the least fit individuals, together with a flat random subset of the oldest individuals, will then die. These are replaced by children, each having one parent chosen randomly from the fittest members of the population, who randomly chooses a mate from the rest of the whole population. Each child inherits characteristics from both parents such that each innate free parameter is chosen at random somewhere between the values of its parents, with sufficient noise (or mutation) that there is a reasonable possibility of the parameter falling outside the range spanned by the parents. There are many other aspects of biological evolution that could be incorporated into our simulations, but this simplified approach proves adequate. A similar regime has already been employed successfully elsewhere to study genetic assimilation and the Baldwin effect in the evolution of adaptable control systems (Bullinaria, 2001a), and the evolution of modularity in neural network systems (Bullinaria, 2001b).

In the simulations, each genotype includes all the innate parameters needed to specify the details of the associated individual network, namely the architecture, the initial connection weights, the learning algorithm, the learning rates, the offsets, and so on. In real biological evolution, all these parameters will be free to evolve. For simulations that are designed to explore particular issues, it makes sense to fix some of these parameters to avoid the complication of un-foreseen interactions (and also to speed up the simulations). In my earlier study of the Baldwin effect (Bullinaria, 2001a), for example, it made sense to keep the architecture fixed and to allow the initial innate connection weights and learning rates to evolve. In my study of modularity (Bullinaria, 2001b) it was more appropriate to have each individual start with random initial connection weights and allow the architecture and learning rates to evolve. Here we shall have a fixed fully connected feed-forward architecture with one hidden layer and random initial weights, and allow all the learning algorithm parameters to evolve, i.e. the four learning rates, the momentum, and the offsets. Then, since the appropriate ranges for the random initial weights may well depend on the evolved learning parameters, and vice versa, we must allow the four initial weight distributions to evolve as well. In total, we thus have up to twelve evolvable parameters in each genotype: four to control the individual's distribution of random initial weights, five to control its learning rates, and up to three to determine the offsets. All the other network parameters, such as the number of hidden units, were fixed across the whole population.

IV. Simulation Results

We can clearly expect some degree of problem dependence with the simulation results, so two representative sets of training data were used:

'What' – A simplified pattern recognition task that maps simple images (5×5 binary matrices) to a representation of 'what' (a 9 bit binary vector with one bit 'on'). Following earlier studies (Rueckl, Cave & Kosslyn, 1989; Bullinaria, 2001b), 9 patterns consisting of different 3×3 arrays with 5 cells 'on' were used as images, and these could appear in any of 9 positions in the full input array, giving 81 training patterns in all.

'QuasiReg' – A quasi-regular mapping from 9 binary input units to 9 binary output units. There were 60 training patterns in all, 48 regular (permuted identity maps) and 12 irregular (random maps). The irregular patterns will naturally be harder to learn than the regulars, and hence prime candidates for learning problems of the type discussed above.

Obtaining reliable results relies on fixing all the evolutionary parameters appropriately according to the details of the problem and the speed and coarseness of the simulations. For example, if all the individuals were able to learn the given task perfectly by the end of their first year, and we only tested their performance once per year, then the advantage of those that learn in two months over those that take ten would be lost, and our simulated evolution would not be very realistic, nor would it encourage faster learning. Since the networks are allowed to evolve their own learning rates, this had to be controlled by fixing the number of hidden units at 36 (which is plenty, but not excessive, for our two tasks), and restricting the number of presentations of the training data set to two per simulated year for each individual. A fixed population size of 100 was chosen as a trade-off between maintaining genetic diversity and running the simulations reasonably quickly. The death rates were chosen to result in reasonable age distributions, and to prevent the population from becoming dominated by skilled adults who killed off most of the children before they had the chance to learn how to perform

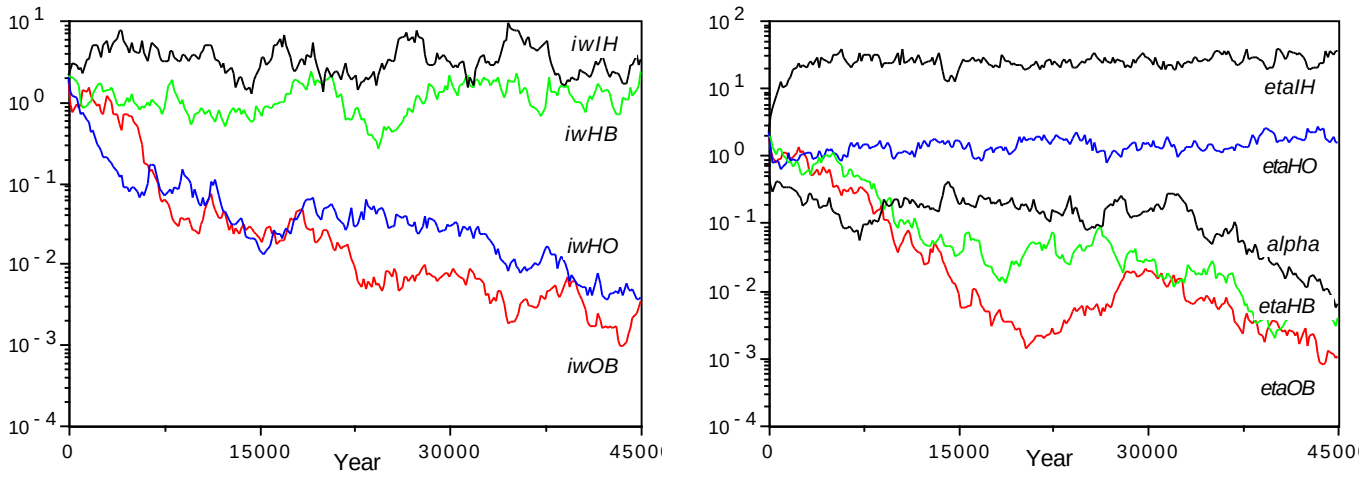

Figure 1: Evolution of the initial weight ranges and learning rates for the SSE cost function and 'What' training data.

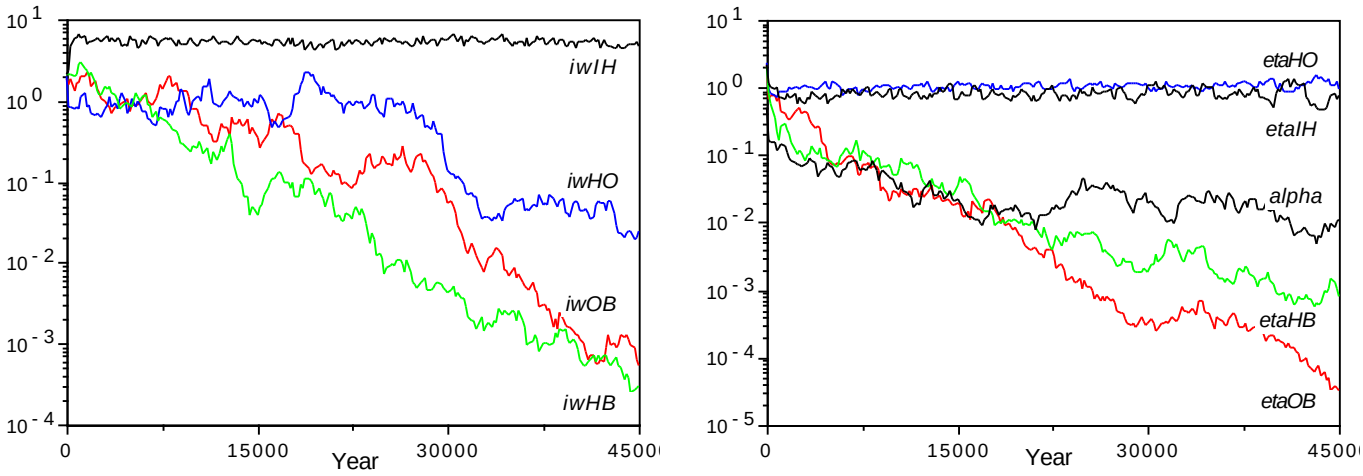

Figure 2: Evolution of the initial weight ranges and learning rates for the CE cost function and 'What' training data.

well. This meant about 10 deaths per year due to competition, and another 3 individuals over the age of 20 dying each year due to old age. The mutation parameters were chosen to speed the evolution as much as possible by maintaining genetic diversity without introducing an excessive amount of noise into the process. These parameter choices inevitably led to coarser simulations than one would ideally like, but otherwise the simulations would have taken too long to complete.

For any evolutionary simulation, the choice of fitness function is obviously a crucial factor. Relating it to the gradient descent cost function was not feasible, because it prevents fitness comparisons between the sum-squared-error and cross-entropy cases, and because the target offsets just grow so large that any output value is deemed errorless. The natural fitness measure here is in terms of the total or average number of network output bits that are significantly wrong (e.g. more than 0.2 from their binary targets) over the whole training set. This works well, but the distribution of errors actually becomes very skewed over the population, so the fitness measure was chosen to be 1/log(1+*ErrorCount*).

Previous evolutionary studies (e.g. Bullinaria, 2001a,b) have shown that the results of evolution can depend strongly on the initial conditions, i.e. on the distributions of innate parameters across the initial population. In particular, the populations tend to settle into near optimal states more quickly and reliably if they start with a wide distribution of initial learning rates, rather than expecting the mutations to carry the system from a state in which there is little learning at all. Consequently, in all the following experiments, the initial population learning rates η_L were chosen randomly from the range [0.0, 4.0], the momentum parameters α randomly from the range [0.0, 1.0], and the random initial weight ranges iw_L from the range [0.0, 4.0]. The results are then consistent enough for us to present typical runs, rather than averages which tend to obscure the interesting details.

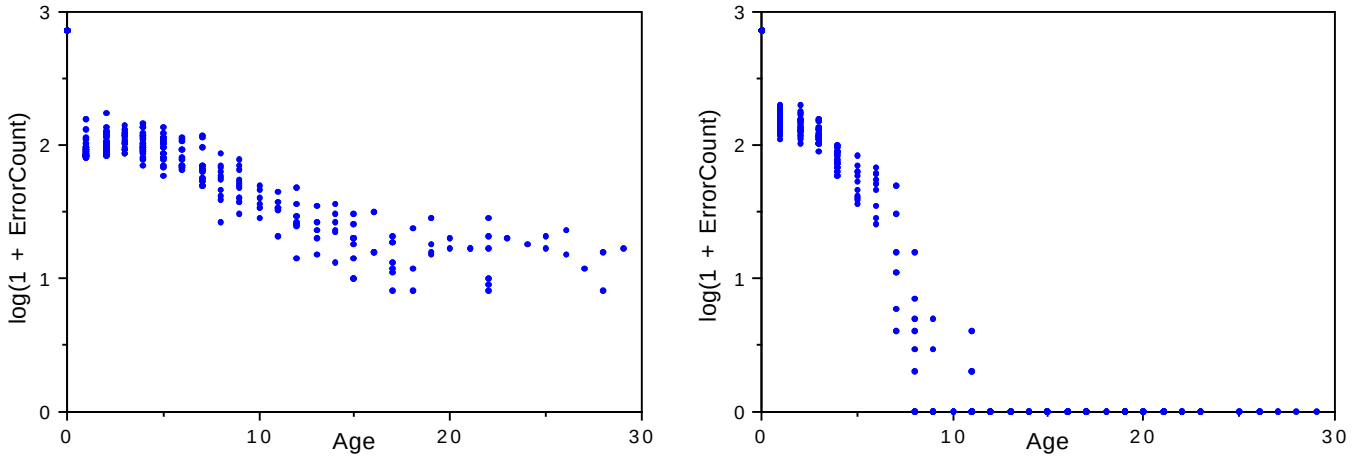

Figure 3: Learning of the 'What' training data by the evolved populations for SSE (left) and CE (right) cost function.

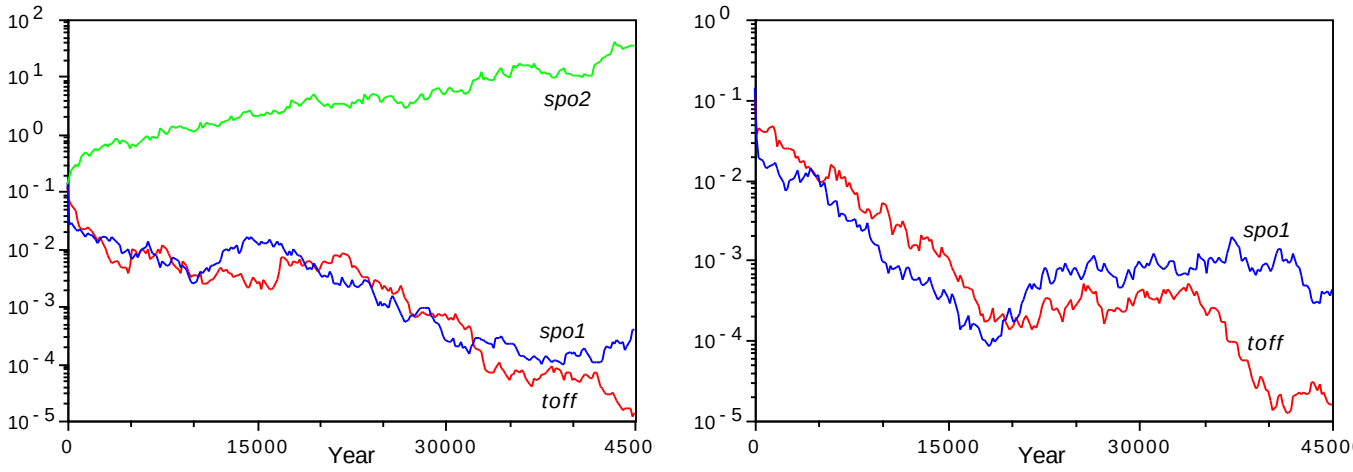

Figure 4: Evolution of the offsets for the SSE (left) and CE (right) cost functions with the 'What' training data.

The starting point is to evolve the standard learning algorithms with no offsets. Figures 1 and 2 show how the initial weight distribution sizes and learning rates evolve for the 'What' task with the SSE and CE cost functions. Note the wide differences in the emerging values for the different network components, and also for the different cost functions. We see that even after 45000 simulated years, some parameters are still drifting to ever lower values, but are already so low that they have very small effect on what the networks are learning. It is clear that the common practice of setting the same initial weight distributions and learning rates throughout a given network is unlikely to result in the optimal performances we can arrive at with an evolutionary approach.

Naturally it is the resultant learning abilities that we are primarily interested in. In Figure 3 we have plots of the performance against age for the final evolved populations. The left graph shows that, as expected, the evolved SSE population from Figure 1 is unable to learn the task to perfection. In fact, a more detailed analysis shows that, by the simulated age of 30, approximately 96% of the remaining output unit errors are greater than 0.99, which confirms that the main cause of the learning problems is the outputs getting stuck with totally wrong values. The right graph of Figure 3 shows how the evolved CE population corresponding to Figure 2 has no such problem, and is typically able to learn the given task by about 10 simulated years of age.

As discussed above, we can hope to do better, particularly in the SSE case, by evolving appropriate offsets. In Figure 4 we can see what the evolutionary pressures do produce in practice. For both the SSE and CE cost functions, we find that the hidden layer sigmoid derivative offsets *spo1* and the output target offsets *toff* take on very low (effectively zero) values, indicating that their presence does not help. In the SSE case, we see that the other potential offset, the output layer sigmoid derivative offset *spo2*, takes on very large values (around 50 after 45000 simulated years and still

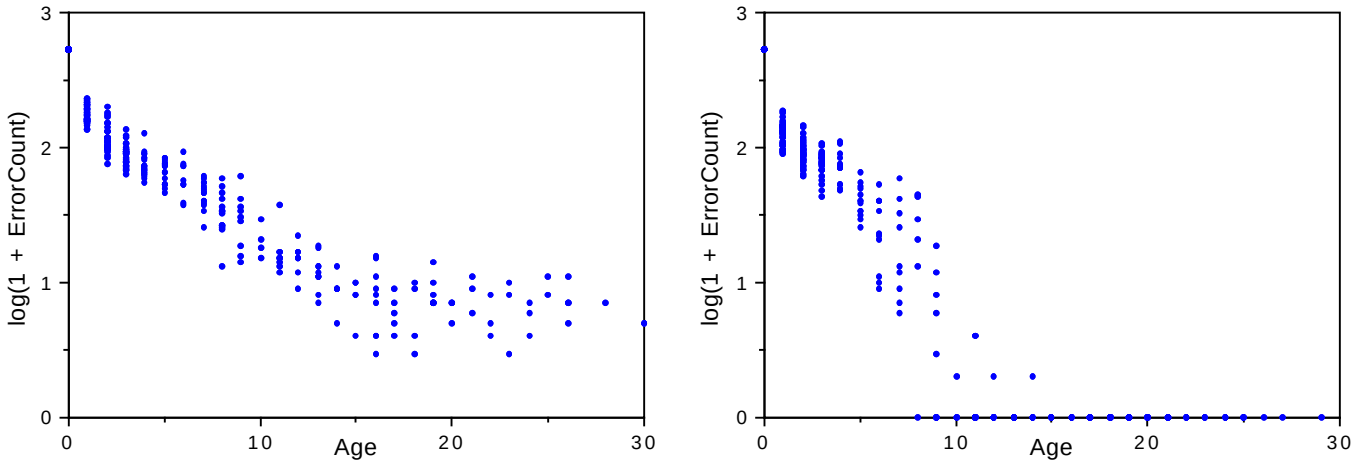

Figure 5: Learning of the 'QuasiReg' training data by the evolved populations for SSE (left) and CE (right) cost function.

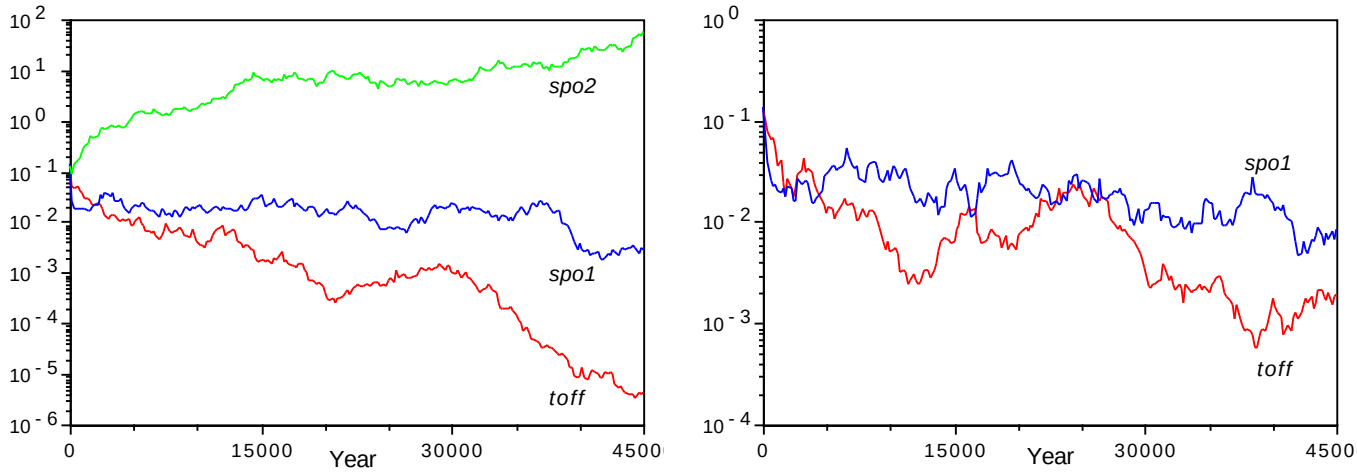

Figure 6: Evolution of the offsets with SSE (left) and CE (right) cost functions for the 'QuasiReg' training data.

rising). This was rather surprising given the relatively small offsets of around 0.1 that are traditionally used (Fahlman, 1988). What happens is that the offset ends up totally swamping the sigmoid derivative to result in weight updates that are good approximations to the standard CE weight updates, but with the learning rates multiplied by *spo2*. Normally, multiplying the learning rates by such large numbers would be detrimental to learning, but because the base learning rates η_L are also allowed to evolve, they adjust themselves in inverse proportion to *spo2* to give appropriate effective learning rates at each stage. In effect, the SSE cost function has discovered how to evolve into the better CE cost function, with its superior learning performance.

We clearly need to check that this result is not dependent on the particular training set used. The set of simulations corresponding to Figures 1 to 3 were therefore repeated with the 'QuasiReg' task. Similar patterns emerged for the evolution of the initial weights and learning rates, and Figure 5 shows the learning performance for the individuals in the evolved populations. Again the CE cost function results in all the individuals learning the task easily, but the SSE population is unable to learn the task to perfection. Here approximately 92% of the remaining output unit errors are greater than 0.99 at the simulated age of 30, so again the main cause of the learning problems appears to be outputs getting stuck with totally wrong values.

As before, we can hope to improve the learning by allowing appropriate offsets to evolve. The results shown in Figure 6 are not as clear cut as we had for the 'What' training data in Figure 4, but the same general pattern emerges. The target offset *toff* and hidden layer sigmoid derivative offset *spo1* take on effectively zero values indicating that they are not useful, while the output layer sigmoid derivative offset *spo2* takes on the ever increasing values that allows the SSE cost function to evolve into the better performing CE cost function as we found previously.

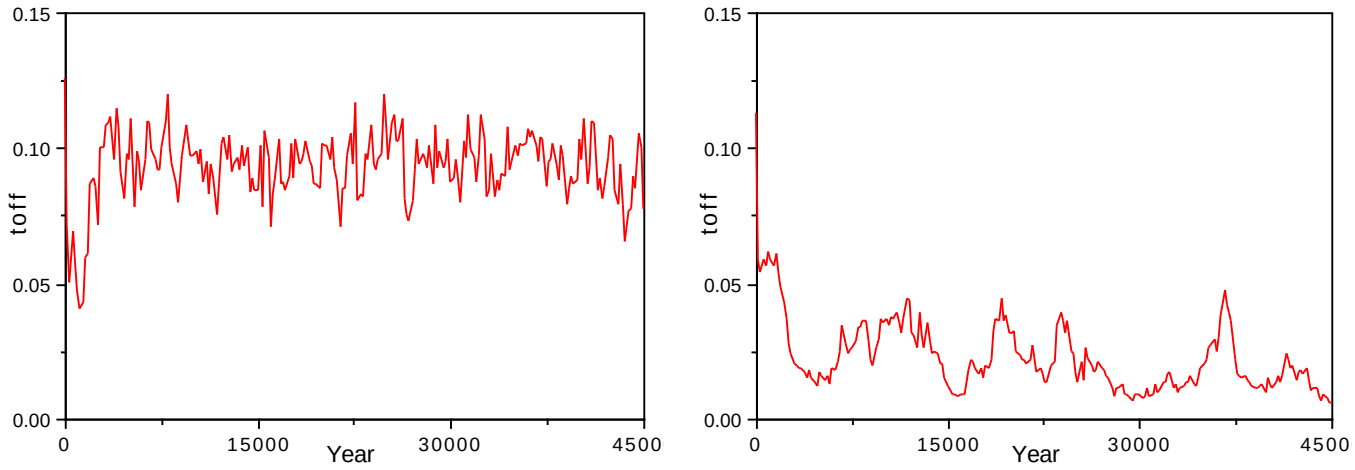

Figure 7: Evolution of target offsets for SSE when SPO is not available for the 'What' (left) and 'QuasiReg' (right) tasks.

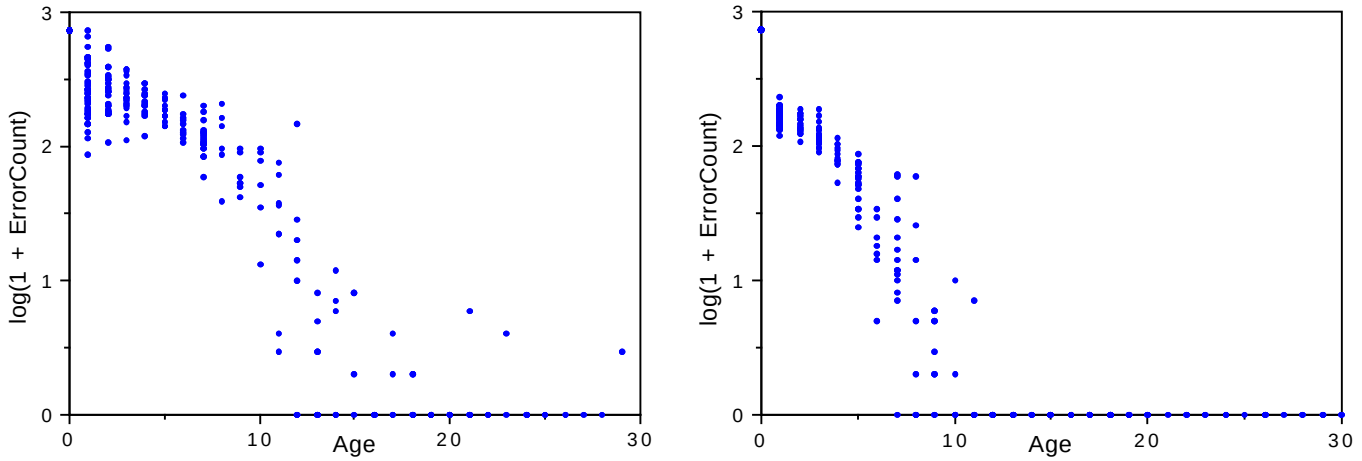

Figure 8: Learning of the 'What' training data by the evolved SSE populations for zero SPO (left) and large SPO (right).

The final situation we need to consider is that in which we do not allow the evolution of sigmoid derivative offsets, and thus prevent the ability of the SSE networks to evolve into CE networks. Will significant target offsets evolve in this situation? Not surprisingly, simulations of the CE case result in the target offsets evolving towards zero values as before. For the SSE case we obtain the problem dependent results shown in Figure 7. For the 'What' task we find that *toff* evolves to take on values around 0.1, which is exactly the value traditionally used (Rumelhart, Hinton & Williams 1986; Rueckl, Cave & Kosslyn, 1989; Bullinaria, 2001b). Learning in this case takes around 15 simulated years, as shown on the left in Figure 8, which is significantly slower and less reliable than the CE case seen in Figure 2, and the SSE case with evolved SPO shown on the right in Figure 8. For the 'QuasiReg' task, on the other hand, Figure 7 reveals that a significant *toff* does not evolve. This is presumably because it does not offer sufficient improvement over the no offset case. Even carrying out the evolution with *toff* fixed at the 'standard' value of 0.1 fails to produce a population that can successfully learn the task.

V. Conclusion

This paper has demonstrated how an evolutionary approach can be used to generate efficient neural network learning algorithms by discovering appropriate initial weight distributions and learning parameters. Explicit simulations for two representative binary mappings were carried out for both the SSE and CE cost functions. The evolved CE networks learned quickly and reliably, while the SSE networks ran into a well known problem in which some outputs get stuck with totally wrong values. The same evolutionary approach enabled the optimization of two common techniques for avoiding such problems in SSE networks, namely the introduction of target offsets and/or

sigmoid derivative offsets. Not surprisingly, for the CE case, evolution caused the unnecessary offset parameters to approach zero. For the SSE networks, evolution failed to produce offsets in the range traditionally used, but rather generated values that transformed the networks into increasingly good approximations of the CE networks that have no such learning problems.

This study has two clear conclusions. First, appropriate evolutionary strategies provide an extremely powerful approach for optimizing neural network systems. Second, when performing gradient descent training of neural networks on binary mapping, it is better to employ the CE cost function rather than using approaches that attempt to fix the problems inherent in using the SSE cost function.

REFERENCES

Bishop, C.M. (1995). *Neural Networks for Pattern Recognition*. Oxford, UK: Oxford University Press.

Bullinaria, J.A. (2001a). Exploring the Baldwin Effect in Evolving Adaptable Control Systems. In R.F. French & J.P. Sougné (Eds), *Connectionist Models of Learning, Development and Evolution* (pp. 231-242). London: Springer.

Bullinaria, J.A. (2001b). Simulating the Evolution of Modular Neural Systems. In *Proceedings of the Twenty-Third Annual Conference of the Cognitive Science Society* (pp. 146-151). Mahwah, NJ: Lawrence Erlbaum Associates.

Fahlman, S.E. (1988). Faster Learning Variations of Back Propagation: An Empirical Study. In D. Touretzky, G.E. Hinton & T.J. Sejnowski (Eds), *Proceedings of the 1988 Connectionist Models Summer School* (pp. 38-51). San Mateo, CA: Morgan Kaufmann.

Golden, R.M. (1988). A Unified Framework for Connectionist Systems. *Biological Cybernetics*, **59**, 109-120.

Hinton, G.E. (1989). Connectionist Learning Procedures. *Artificial Intelligence*, **40**, 185-234.

Rueckl, J.G., Cave, K.R. & Kosslyn, S.M. (1989). Why are "What" and "Where" Processed by Separate Cortical Visual Systems? A Computational Investigation. *Journal of Cognitive Neuroscience*, **1**, 171-186.

Rumelhart, D.E., Hinton, G.E. & Williams, R.J. (1986). Learning Internal Representations by Error Propagation. In D.E. Rumelhart & J.L. McClelland (Eds), *Parallel Distributed Processing, Volume 1* (pp. 318-362). Cambridge, MA: MIT Press.

Van Ooyen, A. & Nienhuis, B. (1992). Improving the Convergence of the Backpropagation Algorithm. *Neural Networks*, **5**, 465-471.

Yao, X. (1999). Evolving Artificial Neural Networks. *Proceedings of the IEEE*, **87**, 1423-1447.