Optimizing the Learning of Binary Mappings

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

Abstract- **When training simple sigmoidal feed-forward neural networks on binary mappings using gradient descent algorithms with a sum-squared-error cost function, the learning algorithm often gets stuck with some outputs totally wrong. This is because the weight updates depend on the derivative of the output sigmoid which goes to zero as the output approaches maximal error. Common solutions to this problem include offsetting the output targets, offseting the sigmoid derivatives, and using a different cost function. Comparisons are difficult because of the different optimal parameter settings for each case. In this paper I use an evolutionary approach to optimize and compare the different approaches.**

I. Introduction

There is a well known potential problem when training simple sigmoidal feed-forward neural networks on binary mappings using gradient descent algorithms with a sum-squared-error cost function. It is that 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.

One obvious solution, that dates back to the re-discovery of the back-propagation learning algorithm, is to offset the targets [1]. 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 do not go to zero when the wrong target is reached. Another early solution to the problem involves offseting the sigmoid derivatives (a.k.a. sigmoid prime) directly, e.g. by adding a constant 0.1 to it [2]. This Sigmoid Prime Offset (SPO) approach clearly also prevents the weight update signals from going to zero when the wrong target is reached.

An obvious problem with both of these approaches is the need to choose an appropriate value for the offset parameters. On one hand we probably shouldn't deviate from true gradient descent more than we have to; on the other hand we want the offsets to be large enough to be maximally effective. Some 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) may depend on the type of offset used, and also on the magnitude of the chosen offset.

Another solution that has been proposed is to replace the sum-squared-error gradient descent cost function with the cross-entropy cost function [3, 4]. 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.

In recent years, increasing computational resources have made it possible to optimize such learning parameters using evolutionary strategies [5], and thus ensure that 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 of some 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 basic ideas of training feed-forward networks by gradient descent are now well known [3, 6], so I shall simply summarize my notation here. The standard weight update equation at each epoch *n* is

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

where *E* is the cost function. 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 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 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 are

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

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

Our aim 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, running, 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 described in [5]). Rather than training each member of the whole population for a fixed time and then 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. During each simulated year, each individual will learn from their own experience with the environment (i.e. set of training/testing data) and have their fitness determined. 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 [7], and the evolution of modularity in neural network systems [8].

Each genotype in our simulations will include 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. In 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 [7], 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 [8] it was more appropriate to have each individual start with random initial connection weights and allow the architecture to evolve. Here we shall have a fixed feed-forward architecture with one hidden layer of 36 units 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.

Each genotype thus contained up to twelve evolvable parameters: 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.

To obtain reliable results it is important to fix 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 one month over those that take eleven is lost, and our simulated evolution will not be very realistic, nor will it encourage faster learning. Since the networks were allowed to evolve their own learning rates, this had to be controlled by 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 a trade-off between maintaining genetic diversity and running the simulations reasonably quickly. The death rates were set in order 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 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 led to coarser simulations than one would ideally like, but otherwise the simulations would still be running.

A crucial consideration for any evolutionary simulation is the choice of fitness function. The obvious 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 an appropriate fitness measure was chosen to be 1/log(1+*ErrorCount*).

My earlier evolutionary studies [7, 8] indicated that the consequences of the evolution can depend strongly on the initial conditions, i.e. on the distribution of the innate parameters across the initial population. In particular, the populations tend to settle into a near optimal state 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. Thus, 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].

IV. Simulation Results

We can clearly expect some degree of problem dependence with the simulation results, so two representative sets of binary 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 [9, 8], 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).

Figure 1 shows how the initial weight distribution sizes and learning rates typically evolve for the 'What' task with CE cost function and no offsets. Note the wide differences in the emerging values for the different network components. Even after 45000 simulated years, the bias parameters are still drifting to ever lower values, but are already so low as to 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.

Figure 2 provides plots of performance against age for the final evolved populations. The first graph shows how the evolved population from Figure 1 is able to learn the given task by about 10 simulated years of age. We can similarly evolve the best parameters for the corresponding SSE case, but here, as the second graph in Figure 2 shows, the evolved population is unable to learn the task to perfection.

As discussed above, we can hope to do better, particularly in the SSE case, by evolving appropriate offsets. Figure 3 shows what happens in practice. For both the CE and SSE cost functions, we find that *spo1* and *toff* take on very low (effectively zero) values, indicating that their presence does not help. In the SSE case, we see that *spo2* takes on very large values, rather than the small offset that is traditionally used. What happens is that the offset totally swamps the sigmoid derivative to result in weight updates that are good approximations to the CE weight updates. The SSE cost function has evolved into the better CE cost function, with its

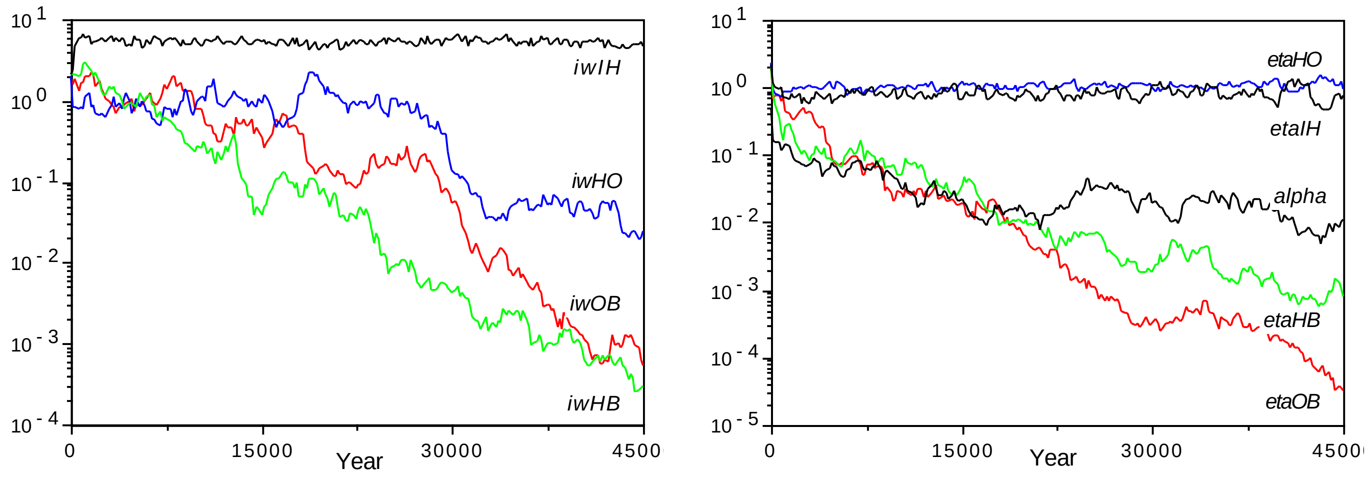


Figure 1: Evolution of the initial weight ranges and learning rates for the CE cost function and 'What' training data.

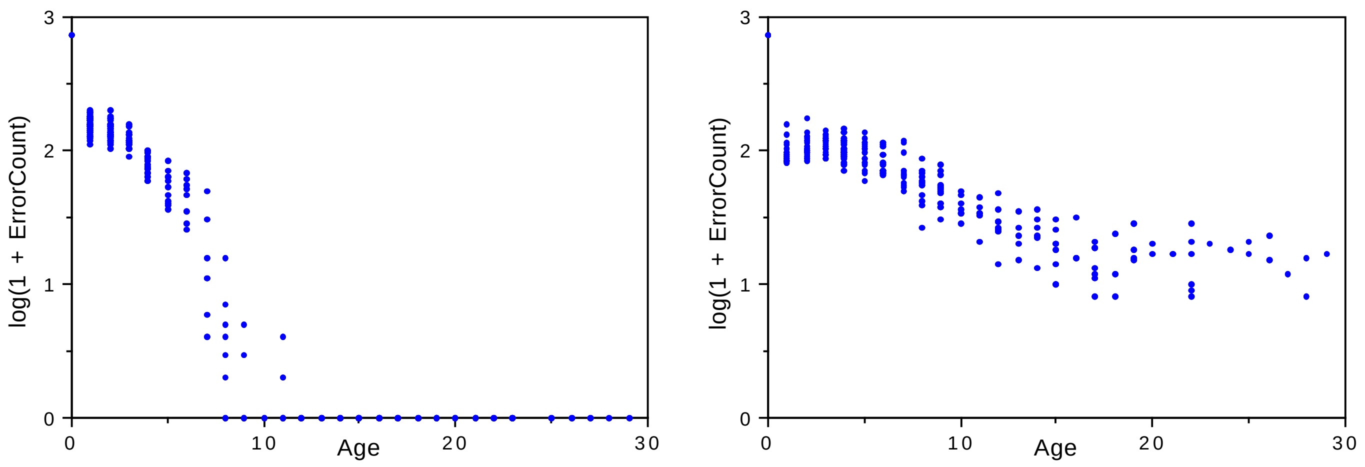


Figure 2: Learning of the 'What' training data by the evolved populations for CE (left) and SSE (right) cost function.

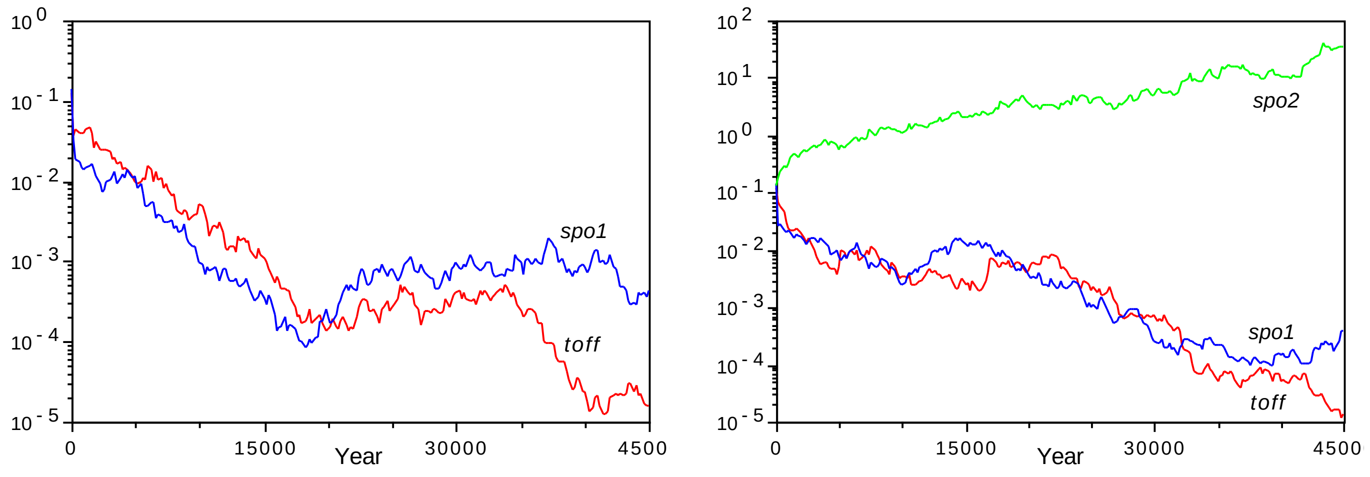


Figure 3: Evolution of the offsets for the CE (left) and SSE (right) cost functions with the 'What' training data.

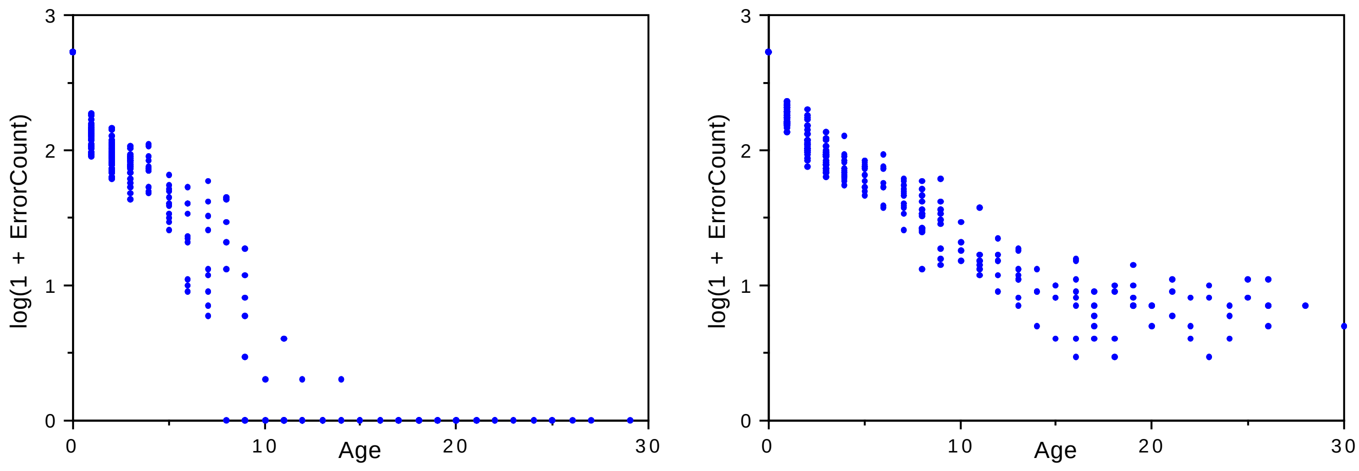


Figure 4: Learning of the ‘QuasiReg’ training data by the evolved populations for CE (left) and SSE (right) cost function.

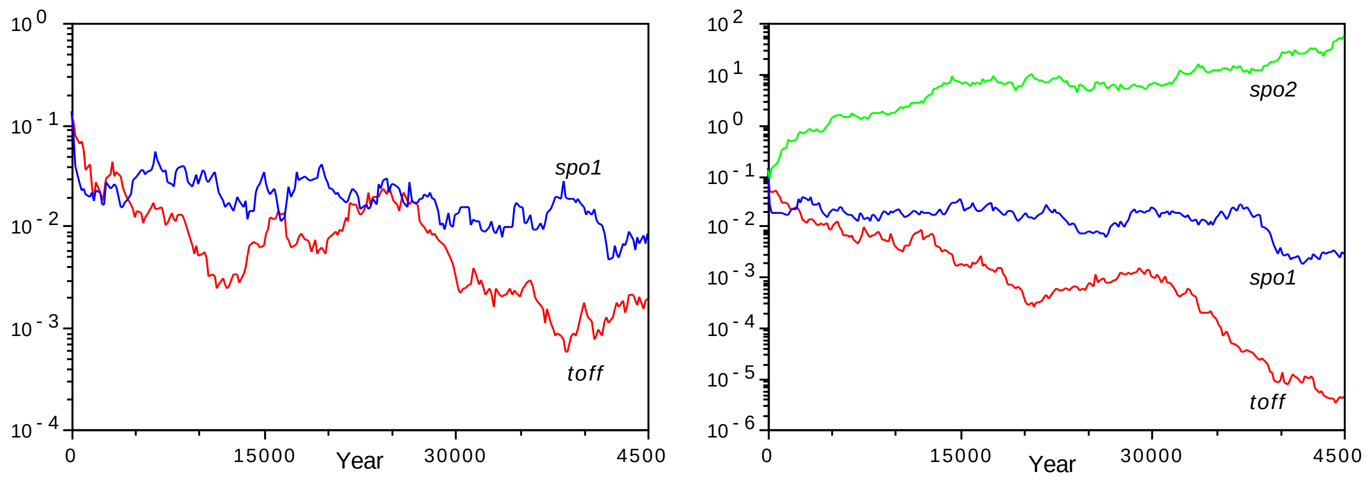


Figure 5: Evolution of the offsets with CE (left) and SSE (right) cost functions for the ‘QuasiReg’ training data.

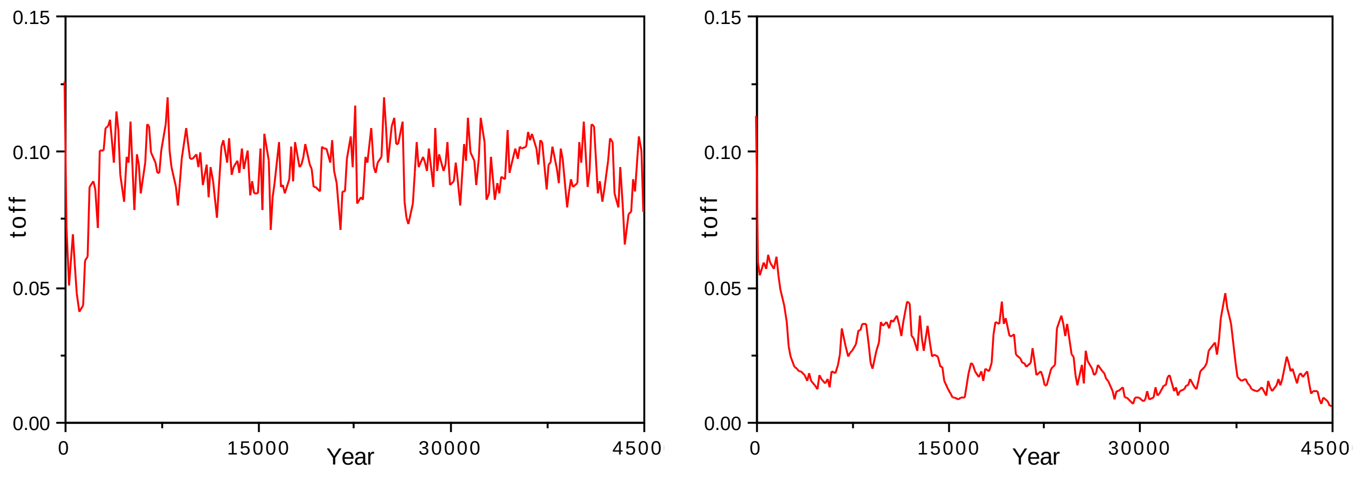
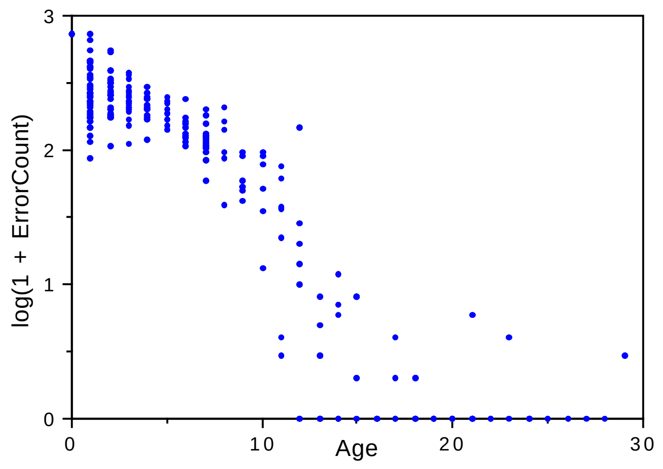


Figure 6: Evolution of target offsets for SSE when SPO is not available for the ‘What’ (left) and ‘QuasiReg’ (right) tasks.



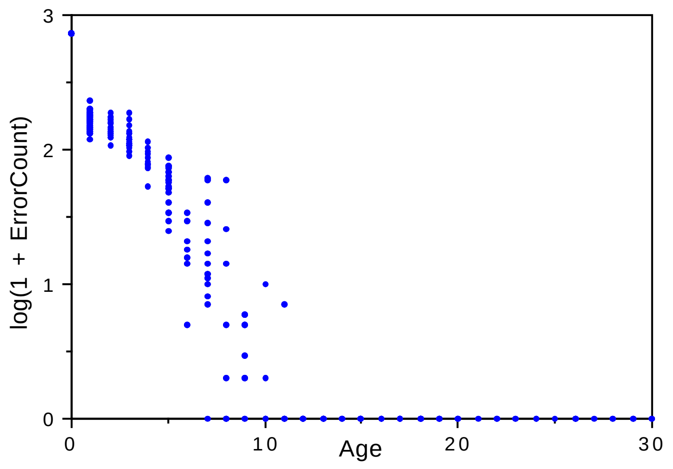


Figure 7: Learning of the 'What' training data by the evolved SSE populations for zero SPO (left) and large SPO (right).

superior learning performance.

Figure 4 shows similar patterns of learning for the 'QuasiReg' task, and Figure 5 shows the offsets that evolve to improve the performance. Again, *spo1* and *toff* take on very small values, and *spo2* allows the SSE cost function to evolve into the better performing CE cost function.

The final situation to consider is what happens if we just allow the target offset, but not the sigmoid prime offsets. Not surprisingly, for the CE case they just evolve to zero as before. For the SSE case we obtain problem dependent results as shown in Figure 6. For the 'What' task we find that *toff* evolves to take on values around 0.1, which is exactly the value traditionally used [1, 8, 9]. Learning in this case takes around 15 simulated years, as shown on the left in Figure 7, which is significantly worse than the CE case seen in Figure 2, and the SSE case with evolved SPO shown on the right in Figure 7. For the 'QuasiReg' case, Figure 6 reveals that a significant *toff* does not evolve. This is presumably because it does not offer sufficient improvement over the non-offset case. Even fixing *toff* at the 'standard' value of 0.1 doesn't result in a population that can successfully learn the task.

V. Conclusion

We have seen how an evolutionary approach can be used to generate good neural network learners. Well known techniques for preventing the problems of learning binary mappings with the SSE cost function involve introducing either target offsets or sigmoid derivative offsets. Attempts to optimize these offsets for two representative training sets result in the SSE cost function evolving into the CE cost function which has no such learning problems. This suggests that the best general strategy is to employ CE, rather than SSE, cost functions for neural networks required to learn binary mappings.

References

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

[2] S.E. Fahlman, 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*, San Mateo, CA: Morgan Kaufmann, 1988, pp. 38-51.

[3] G.E. Hinton, Connectionist Learning Procedures. *Artificial Intelligence*, **40**, 1989, pp. 185-234.

[4] A. Van Ooyen & B. Nienhuis, Improving the Convergence of the Backpropagation Algorithm. *Neural Networks*, **5**, 1992, pp. 465-471.

[5] X. Yao, Evolving Artificial Neural Networks. *Proceedings of the IEEE*, **87**, 1999, pp. 1423-1447.

[6] C.M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, UK: Oxford University Press, 1995.

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

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

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