Generational versus Steady-State Evolution for Optimizing Neural Network Learning

John A. Bullinaria
School of Computer Science, The University of Birmingham, Birmingham, B15 2TT, UK
E-mail: j.bullinaria@physics.org

***Abstract-* The use of simulated evolution is now a commonplace technique for optimizing the learning abilities of neural network systems. Neural network details such as architecture, initial weight distributions, gradient descent learning rates, and regularization parameters, have all been successfully evolved to result in improved performance. In this paper I investigate which evolutionary approaches work best in this field. In particular, I compare the traditional *generational* approach with a more biologically realistic *steady-state* approach.**

I. Introduction

We are all familiar with the idea that formulating powerful neural network systems requires the specification of a number of details that will be rather problem dependent [1]. For example, if we require a simple feed-forward network to learn particular classes of input-output mappings, we will need to set an appropriate architecture, initial weight distributions, learning rates, regularization parameters, and so on, to optimize its performance. How we measure that performance will also generally be problem dependent. Usually it is the generalization performance that we are most concerned about, but we often require fast learning as well. In this paper I shall consider the general problem of how to set up neural network systems that learn as quickly as possible, to generalize as well as possible, on data drawn randomly from a particular class of data distributions. An increasingly common and successful approach is to evolve such systems, rather than trying to build them by hand [2]. However, despite a considerable literature in this field, there appears to be no general investigation into which are the best evolutionary strategies to use for this problem. It is this particular issue that I wish to explore in this paper.

In my previous investigations of evolving neural networks [3, 4, 5], I have adopted a nature inspired evolutionary approach in which the populations consist of competing learning individuals of all ages that procreate and die according to their fitness and age. I have always assumed that such a *steady state* approach [6, 7] will be more efficient than a traditional *generational* approach in which whole new generations are produced at the same time [2]. In this paper, I present a systematic investigation of this matter. Some comparisons of steady state and generational approaches have been carried out previously [6, 7, 8], but these have not involved systems like neural networks where individuals use learning to increase their fitness during their own lifetimes.

In the remainder of this paper I shall begin by describing in more detail the class of neural network systems I am considering, and specify the three broad classes of evolutionary strategies I wish to compare. I will then present a series of simulation results, and end with some discussion and conclusions as to the advantages of each approach.

II. The Neural Network Models

For this paper, I shall restrict myself to standard fully connected feed-forward networks of sigmoidal units with one hidden layer trained using gradient descent to perform simple classification tasks. The standard weight w_{ij} update equation at each training 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 [1]. Past experience [3, 4] indicates that the networks learn better if they have different learning rates η_L for each connection layer and bias set L. 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 in the range $[-r_L, +r_L]$. Naturally, to match the learning rates, different range parameters r_L are allowed for the input to hidden layer connections, the hidden layer biases, the hidden to output layer connections, and the output biases.

As is appropriate for classification, I use the cross-entropy cost function with a weight decay regularization term [1]

$$E = -\sum_j \left[t_j . \log(o_j) + (1-t_j) . \log(1-o_j) \right] + \lambda \sum_{j,k} (w_{jk})^2$$

which leads to the output layer weight derivatives

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

where t_j are the binary target network outputs, o_j are the

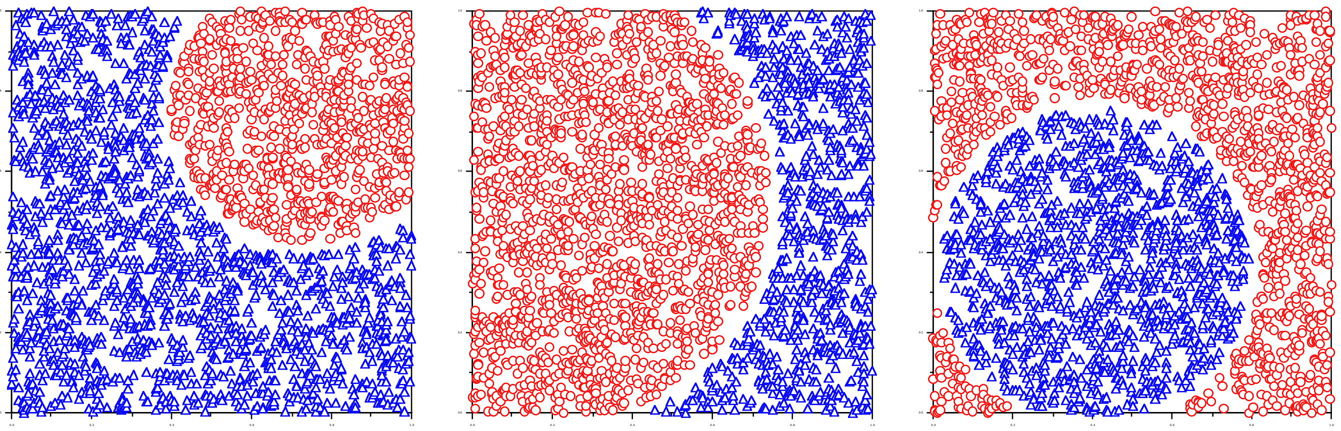

Figure 1: Typical two-class classification data distributions which the neural networks must learn from random samples.

actual outputs, and h_i are the hidden unit activations.

Obviously there is no point in evolving a system to learn one particular training set quickly – it will be quicker to put up with a slow learner and not bother with the evolution process. We want to evolve systems that can learn quickly to generalize from samples that are drawn randomly from data distributions that are themselves drawn randomly from a class of different data distributions. For this study I shall consider inputs in a continuous two dimensional space $[0.0, 1.0]^2$, and have two outputs corresponding to two classifications specified by random circles in the input space. Figure 1 shows three typical data distributions. Each individual has its own data distribution, and during each simulated year draws, and learns from, a new sample from that distribution. It must learn to generalize so it can perform well on each year's data, before learning from its mistakes. The obvious performance measure is the total number of network output bits that are significantly wrong (e.g. more than 0.2 from their binary targets) over the whole of each years' training set. Extension to more input and output dimensions is straightforward.

III. Evolving The Models

The aim here is to optimise our neural network systems using evolution by natural selection. To simulate evolution we take a whole population of individual instantiations of each model, and allow them to learn, procreate and die in a similar manner to natural (biological) systems. The genotype of each individual will specify all the appropriate innate parameters, and depend on the genotypes of its two parents plus random mutations. Then, throughout its life, each 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.

The ability of a biological individual to survive or reproduce will be a complex function of its performance on a range of tasks (feeding, fighting, fleeing, and so on). For our neural networks it is appropriate and sufficient to assume a simple relation between our single task performance and the survival or procreation fitness. It is also sufficient for each child to inherit 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. Each genotype will contain ten evolvable parameters: four to control the individual's distribution of random initial weights, four to control its learning rates, one for its momentum, and one to specify its regularization parameter. All the other network parameters, such as the number of hidden units, are fixed across the whole population and all generations.

It is less obvious how we should generate each generation from the previous. The object of this paper is to investigate which strategy works best for our chosen neural network systems. In particular, I shall consider the nature inspired *steady state* approach that I have used previously [3, 4, 5]:

SS Each individual learns to improve their performance for as long as they are alive. High performance means high fitness. In each simulated year, a small random subset individuals over a certain simulated age die of old age, and a small random subset of the least fit individuals are killed. These are replaced by children with at least one parent chosen from among the fittest individuals.

and the two traditional *generational* approaches [2]:

G1 Each individual learns to improve their performance for a set number of simulated years. High performance means high fitness. At each generation, all individuals die and are replaced by children with parents chosen from among the fittest individuals.

G2 Each individual learns to improve their performance until it reaches a set target level. Fast learning means high fitness. At each generation, all individuals die and are replaced by children with parents chosen from among the fittest individuals.

Whichever approach used, obtaining reliable results requires fixing the evolutionary parameters according to the details of the problem and the speed and coarseness of the simulations.

To maintain fair comparison, some parameters were kept the same in all simulations: A fixed population size of 200 was a trade-off between maintaining genetic diversity and running the simulations reasonably quickly. 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. The other parameters depended on the choice of generational strategy.

In the *SS* approach, if all the individuals were able to learn their 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 would not encourage faster learning. This is easily controlled by restricting the amount of training data available in each simulated year for each individual. Then the death rates need to be set to result in reasonable age distributions, with the best performing adults having plenty of time to reproduce, but not dominating the population and killing off most of the children before they have had a chance to learn how to perform well. This is easily achieved by fixed tournament competition death rates (e.g. 10% per simulated year), and old age death rates dependant on how fast the best individuals learn (e.g. 20% of individuals each year aged more than three times the average earliest age of best performance).

For the *G1* approach it is N_G, the number of simulated years per generation, that needs to be fixed. The problem is that, if we set this number too low, none of the individuals will ever get close to finishing learning, and if we set it too high, there will be nothing to drive the individuals from reaching their maximum performance any earlier. While this approach might be good for evolving other factors [2], it is clearly no good for our problem of evolving fast learners. What we can do, however, is fix N_G to be large, but measure the average performance over the last N_M years. We can start with $N_M = 1$, and increase N_M by one whenever a reasonable fraction (say half) of each generation consistently settle down to a clear maximum performance over that period. The $N_M = 1$ phase will optimize the performance, and the $N_M > 1$ phase will speed the reaching of that performance.

In the *G2* approach it is the set target level of performance that we must specify. Clearly, if we do not set that to be the maximum level of performance possible, the individuals will never reach it. The problem is that generally we won't know in advance what that level is, and even if we did know it (e.g. because we knew that perfect performance was possible), it is unlikely that that performance would be possible at all for individuals in the first generation. It would appear that this approach on its own is not feasible at all for evolving fast learners. However, if we start off with a large N_G and $N_M = 1$ in the *G1* approach as before, we can switch to the *G2* approach once a reasonable fraction (say half) of each generation consistently settle down to a clear maximum performance at the end of that period, with the set target level of performance equal to the maximum. The *G1* phase with $N_M = 1$ again optimizes performance, but now a *G2* phase is used to optimize the speed of reaching that performance.

The rest of this paper aims to determine if either the two stage *G1+G1* generational approach, or the hybrid *G1+G2* generational approach, can out-perform the steady-state *SS* approach that I have used previously.

IV. Simulation Results

My previous evolutionary studies [3, 4, 5] have indicated that the evolutionary efficiency depends strongly on the initial conditions, i.e. on the distribution 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 wide distributions of initial learning rates, rather than expecting the mutations to carry the system from a state in which there is little learning at all. So, for each evolutionary run, the random initial population learning rates η_L were chosen from the range [0.0, 4.0], the momentum parameters α from [0.0, 1.0], the regularization parameters λ from [0.0, 0.1], and the random initial weight ranges r_L from [0.0, 4.0]. In all simulations, every individual had 20 hidden units and experienced 1200 training data points per year.

Figure 2 shows the *steady state* evolution of the initial weight ranges and learning rates averaged over 20 runs. The learning rates appear to settle down quickly, but actually the variances across simulations are enormous (such that plotting the standard deviations would render the graphs unreadable). A full investigation reveals bimodal distributions, with each run settling down into one of two final states. Fourteen runs resulted in a poor average performance population, and six in a good average performance population. The learning rate means and standard deviations for those two sets are shown in Figure 3. It turns out that the evolution of the initial weight ranges, momentum and regularization parameter have little effect in this study, so I shall not discuss them further. Naturally, it is the evolved populations' performance that we are primarily interested in. To obtain an accurate and statistically reliable measure of that, it was necessary to train each individual from each population on each of 50 different random data distributions. Even for the 'poor' performance populations, the peak in the error count distribution for trained networks was at zero errors, but the distributions have long tails which make the mean and standard deviations seem misleadingly large. Figure 4 shows the median, upper and lower quartiles of errors per year at each age (i.e. amount of

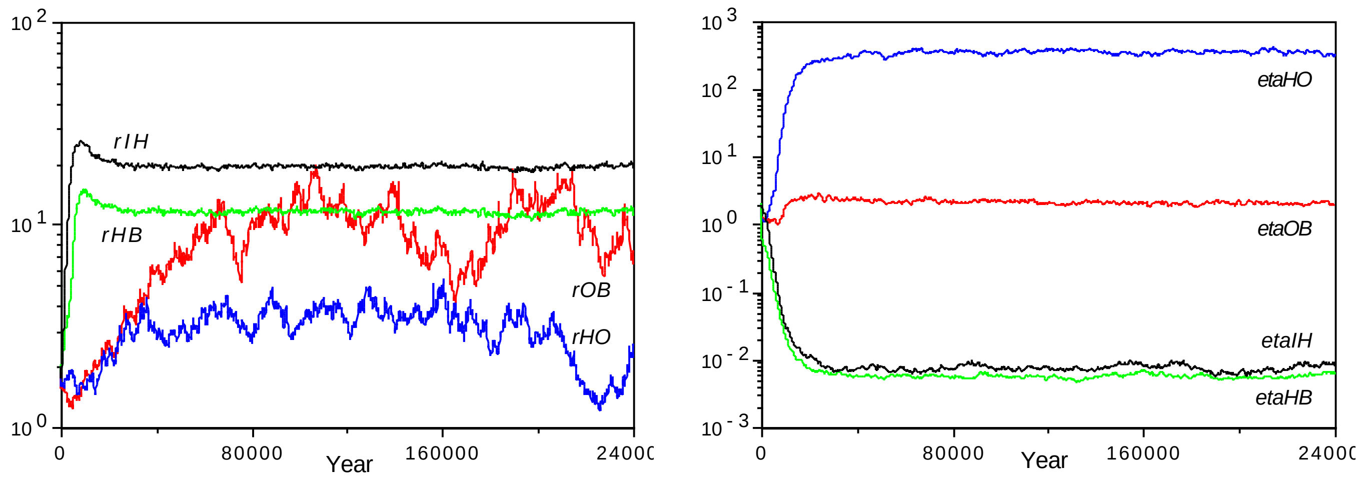

Figure 2: Evolution of the initial weight ranges and learning rates for the *Steady State* approach (averaged over 20 runs).

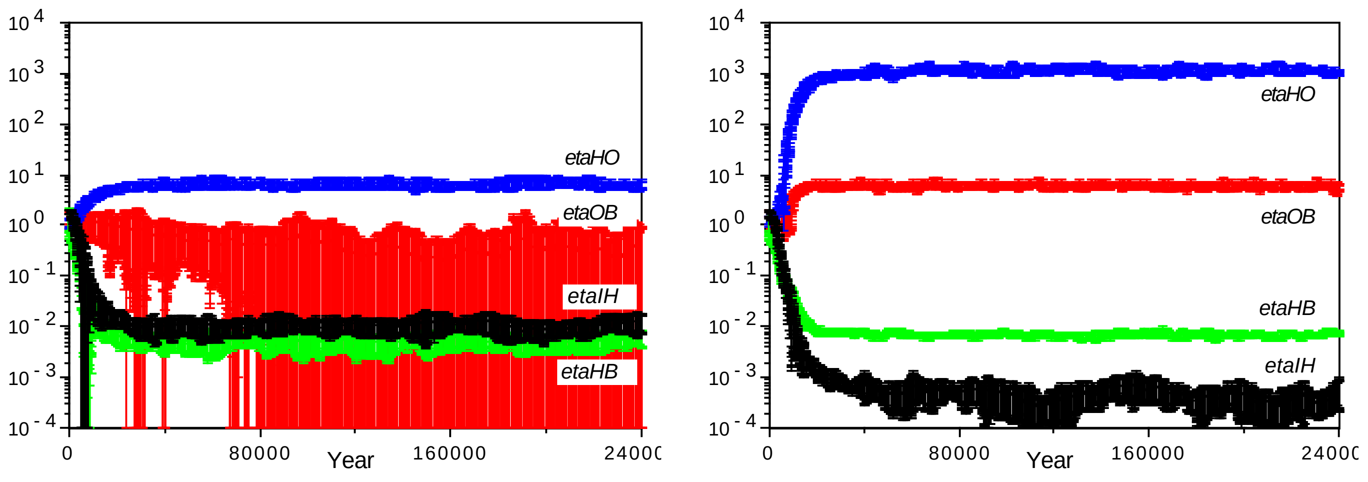

Figure 3: Evolution of the learning rates for the poor (left) and good (right) performance *Steady State* populations.

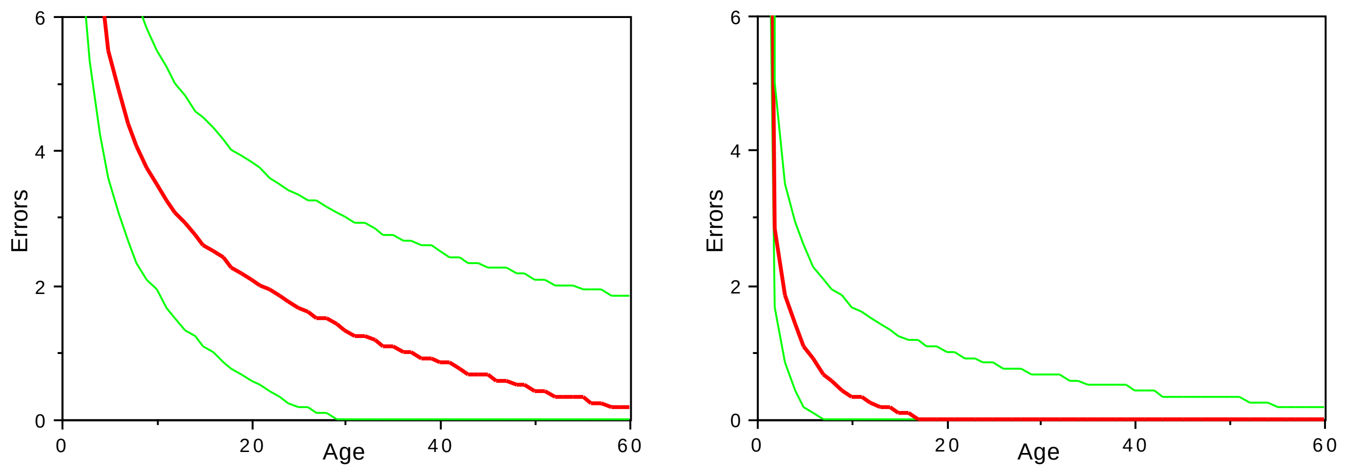

Figure 4: Median and quartile error rates for the poor (left) and good (right) *Steady State* populations at year 240,000.

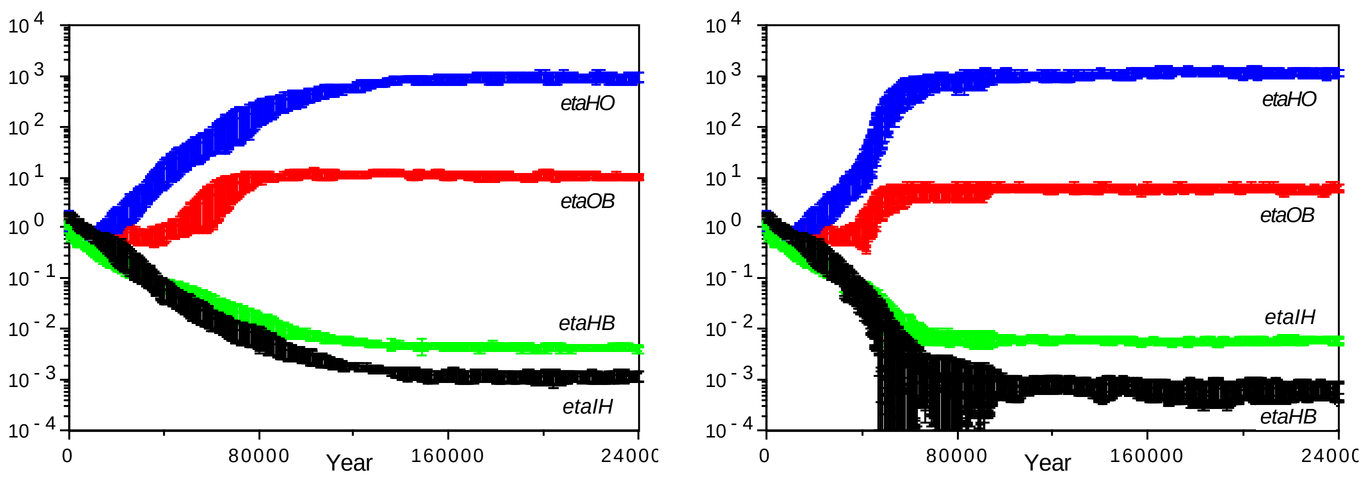

Figure 5: Evolution of the learning rates for the *G1+G1* (left) and *G1+G2* (right) *Generational* populations.

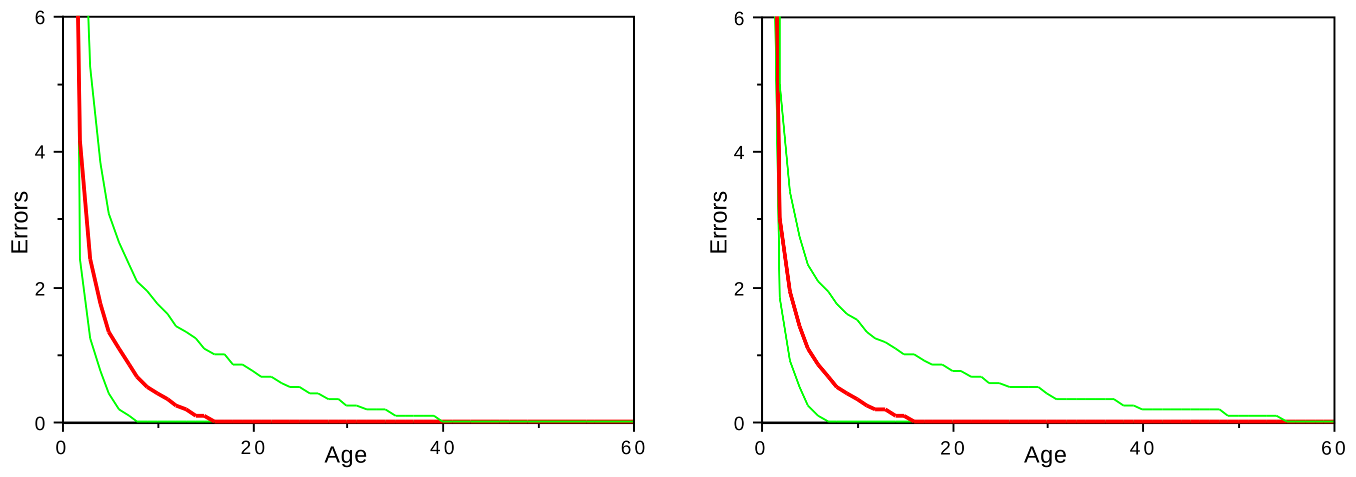

Figure 6: Median and quartile error rates for the *G1+G1* (left) and *G1+G2* (right) *Generational* populations at year 240,000.

training) for the two evolved steady state populations.

We can see that there are big differences in learning rates and performances between the two cases, and also from the standard hand-built networks in which we usually set the same learning rate and initial weight range across all parts of the network. The big question now is: how do these results compare with those of our two *generational* approaches? I set the generational time-scale at $N_G = 60$, and matched the number of neural network computations per simulated year with the steady state case above. The learning rate evolution for the generational approaches are shown in Figure 5. Here the populations take considerably longer to settle down into their final states, but we no longer end up with multi-modal distributions. Figure 6 shows the error rates per year for the evolved populations in each case. We see that the median and lower quartile error rates are similar to each other, and also with the better performing steady state populations, but the upper quartiles reach zero at much earlier ages.

Another way to compare the performances is to plot the distributions of errors per simulated year after training. Figure 7 shows two views of the average error count distributions between the ages of 50 and 60. On the left we see the peak of the distributions at zero and, as we would expect from the quartiles plotted above, the poor steady state populations have a significantly wider peak than the other three cases. On the right we see the tails of the distributions. The good average performance steady state populations have around thirty times as many large error counts (above 20) as the poor average performance steady state populations, and the generational populations fall in between them.

V. Discussion and Conclusions

The aim of this paper was to explore the effect of using different evolutionary strategies to produce neural networks which learn quickly to generalize well. It is clear from the

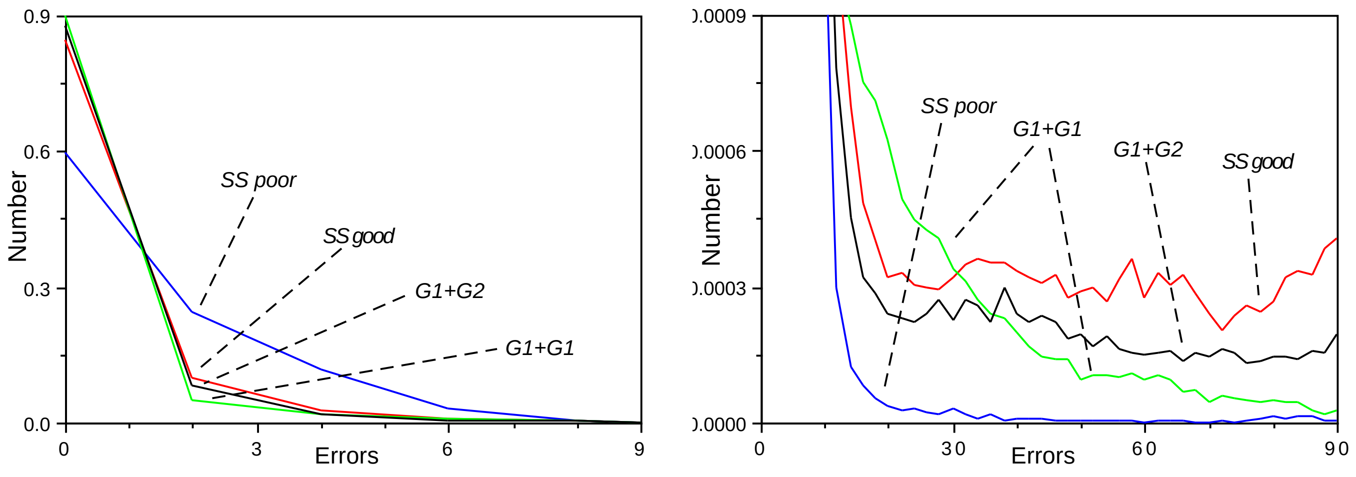

Figure 7: Two views of the mean error count distributions for ages 50-60 for the four evolved populations of Figures 3, 4, 5, 6.

simulation results presented that the different approaches do optimize different things, and consequently yield different evolved behaviours.

We have previously seen that evolution can create high performance neural networks for many applications [2, 3, 4], but this paper now shows how important it is for us to choose the right evolutionary strategy for each particular problem. We need to decide several things: do we care about simulating natural biological evolution (*SS*), do we want to obtain good overall performance (*SS good*), do we wish to reduce the instances of very poor performance at the expense of poorer average performance (*SS poor*), is it more important to learn good performance quickly (*G1+G2*), or to learn to maintain good performance over long periods (*G1+G1*)? Some approaches effectively lead to different species evolving on different runs (*SS good/poor*), others end up with very similar populations emerging on every run (*G1+G2*, *G1+G2*). If we do not tailor our evolutionary strategy to our particular requirements, we will clearly not achieve the best possible results.

A potential difficulty for our two generational approaches is the identification of the 'maximal performance level', and knowing when to switch from the first to second phase of evolution. For this paper I deliberately chose a task that allowed perfect (zero error) performance to make this easy. It will require further investigation to be sure of how to deal best with different types of noisy data, and situations where the best performance levels are more variable.

An important practical consideration, for all evolutionary approaches, is the computational costs involved. The tasks used in this study were chosen to be simple enough to make a full systematic investigation feasible, but those classifications are typical of more realistic problems, and it is reasonable to expect analogous results for larger scale situations. Preliminary studies indicate that the evolutionary efficiency scales well with problem and network size, but we cannot avoid the increased training times that are required for larger networks and more complex training data. We should probably not worry too much about the relatively slow rates of the generational evolution approaches. There are many variations one could try to apply to speed them up: different selection schemes, different elitism proportions, alternative learning regimes, and such like. However, space restrictions prevent a full analysis of these factors here.

References

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

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

[3] 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.

[4] J.A. Bullinaria, Evolving Efficient Learning Algorithms for Binary Mappings. *Neural Networks*, **16**, 2003, pp. 793-800.

[5] J.A. Bullinaria, From Biological Models to the Evolution of Robot Control Systems. *Philosophical Transactions of the Royal Society of London* A, **361**, 2003, pp. 2145-2164.

[6] D. Whitley, The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In J.D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann, 1989, pp. 116-123.

[7] G. Syswerda, A Study of Reproduction in Generational and Steady-State Genetic Algorithms. In G. Rawlins (Ed.), *Foundations of Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann, 1991, pp. 94-101.

[8] K.A. De Jong & J. Sarma, Generation Gaps Revisited. In L.D. Whitley (Ed.), *Foundations of Genetic Algorithms 2*, San Mateo, CA: Morgan Kaufmann, 1993, pp. 19-28.