Evolving Crossover, Mutation and Training rates in a Population of Neural Networks

Dara Curran and Colm O'Riordan
Dept. of Information Technology,
National University of Ireland, Galway.

Abstract. This paper describes a method of determining the rates of crossover, mutation and training employed in the evolution of a population of neural networks. The genetic codes of the population are modified to include rate data which evolves with the population to attain optimum levels. We compare these results to experiments performed to determine optimum rate values by trial and error.

1 INTRODUCTION

The combination of genetic algorithms and neural networks has been shown to be successful in a variety of problem domains [1, 2, 3, 4]. The evolutionary approach of genetic algorithms has been shown to guide the neural network's data processing capability to high levels of performance. In a typical implementation, the genetic algorithm is employed to generate a population of random neural network architectures, each of which is trained and tested to evaluate its performance. The genetic codes of successful networks are then combined to create the next generation. As the genetic algorithm can be designed to select particular network characteristics (such as small number of nodes and links), the approach yields very efficient neural network architectures with little or no human intervention.

The genetic algorithm uses several operators including crossover and mutation. Crossover allows offspring to inherit portions of each parent's genetic code, thus probing new search spaces while the mutation operator modifies the offspring's genetic code in order to both open up un-explored search spaces and slow the process of stagnation.

Typically, the behaviour of both these operators is determined by a rate, usually expressed as a percentage or probability. A crossover rate of 0.75, for instance, means that when any two agents mate, their genetic codes will undergo the process of crossover with a probability of 75%. Similarly, a mutation rate of 0.5 indicates that each bit in the offspring's genetic code will be modified with probability 50%. These rates seriously affect a population's performance. For instance, a setting of 0.00 crossover, would create a stagnant population of cloned networks, while a population evolved using a mutation rate of 0.8 would exhibit very erratic behaviour as a result of constant genetic changes.

The third rate examined in this paper is that of training. Each network is given the opportunity to reduce its error via learning using error back propagation. Generally speaking, the more a network is trained, the better it will perform its task — although, over–training is possible.

It can be difficult to determine values for any of these rates in advance of a given experiment. Usually several trial and error attempts must be made before an appropriate setting is found. This paper presents a method of evolving optimal rate values for crossover, mutation and training for a population of neural networks. All experiments assume that rates are independant and that optimum values can be determined for each individually. Future work will concentrate on experiments attempting to evolve all rates simulatenously.

The next section of the paper describes related research which has been performed. Section 3 briefly describes the artificial life simulator used in this set of experiments. Section 4 presents results for the first set of experiments, where rates are determined in an iterative manner. Section 5 describes the method employed for the evolutionary rate determination experiments as well as their results. Finally, Section 6 ends with a conclusion and an indication of future work.

2 RELATED WORK

The combination of neural networks and genetic algorithms originally stemmed from the desire to generate neural network architectures in an automated fashion using genetic algorithms [1, 2]. The advantage of this approach is that neural networks can be selected according to a variety of criteria such as number of nodes, links and overall accuracy. The neural network component, on the other hand, provides computational functionality at an individual level within the genetic algorithm's population. The combination of genetic algorithms and neural networks has since been proven successful in a variety of problem domains ranging from the study of language evolution [3] to games [4].

Some research has also been performed on the evolution of crossover in a genetic algorithm, notably by Spears [5]. Experiments were performed to allow populations to determine the crossover type to employ for a given problem. The populations were allowed to choose between 1–point and 2–point crossover. This was achieved by appending extra bits to each genetic code in the population to represent either 1 or 2–point crossover. As the population evolved, individuals carrying the most successful crossover type emerged as dominant. Thus, at the end of each experiment it was possible to observe the evolved preference to each crossover method.

3 ARTIFICIAL LIFE SIMULATOR

The experiments outlined in this paper were performed using a previously developed artificial life simulator [6, 7, 8]. The simulator allows populations of neural networks to evolve using a genetic algorithm and each network can also be trained during each generation of an experiment to simulate life–time learning.

The mapping of neural network to genetic code required for the genetic algorithm is achieved using a modified version of marker based encoding. This allows networks to develop any number of nodes and interconnecting links, giving a large number of possible neural network architecture permutations.

Marker based encoding represents neural network elements (nodes and links) in a binary string. Each element is separated by a marker to allow the decoding mechanism to distinguish between the different types of element and therefore deduce interconnections [9, 10].

In this implementation, a marker is given for every node in a network. Following the node marker, the node's details are stored in sequential order on the bit string. This includes the node's label and its threshold value. Immediately following the node's details, is another marker which indicates the start of one or more node–weight pairs. Each of these pairs indicates a back connection from the node to other nodes in the network along with the connection's weight value. Once the last connection has been encoded, the scheme places an end marker to indicate the end of the node's encoding.

The networks undergo various stages throughout their lifetime. First, the gene codes are decoded to create their neural network structure. Training is then performed using error back–propagation for a given number of iterations (training cycles). Each network is tested to determine its fitness and the population is ranked using linear based fitness ranking. Roulette wheel selection is employed to generate the intermediate population. Crossover and mutation operators are then applied to create the next generation.

Since the goal of this paper is to demonstrate rate evolution, not complex problem solving, the problem set chosen for these experiments is 3–bit parity, a problem of modest complexity but which allows numerous experiments to be undertaken in a short space of time.

4 EXPERIMENTAL RATE DETERMINATION

These experiments were designed to give an approximation of the optimal rates of crossover, mutation and training. Each of the rates was modified for each experiment to observe to what degree each rate affects the population's performance. The experiments use 500 networks which evolve for 500 generations. The rates for each experiment were set as follows:

- Crossover Experiment — Mutation set at 0.02, training cycles set at 10.
- Mutation Experiment — Crossover rate set at 0.5, training cycles set at 10.
- Training Experiment — Crossover rate set at 0.5, mutation set at 0.02.

Each experiment was carried out 20 times to ensure an accurate estimation of each of the rates.

4.1 Crossover

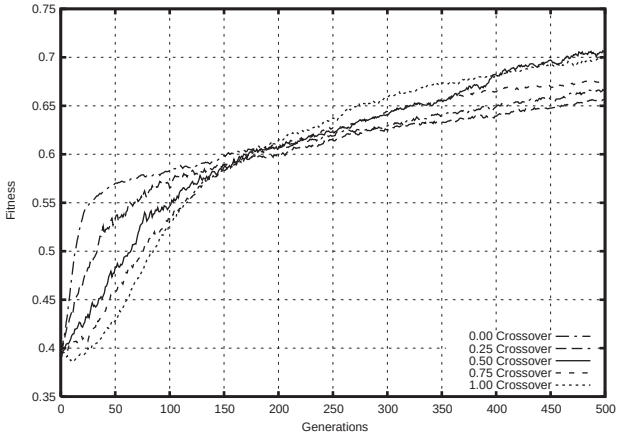

Figure 1. Crossover Rate Determination

The chosen encoding scheme does not permit crossover to operate at the bit level as this could result in the generation of invalid gene codes. Therefore the crossover points are restricted to specific intervals — only whole node or link values may be crossed over.

These results show that at certain levels, the crossover operator begins to negatively affect the population's performance (fig.1). At low levels of crossover this can be interpreted easily: network offspring are often merely clones of their parents thereby impeding the progress of the genetic algorithm. It is interesting to note that the highest level of fitness achieved occurs when crossover is set at 0.5, narrowly out–performing the 1.0 crossover rate. The fact that the fitness drops considerably when crossover is set at 0.75 would seem to indicate that the optimal level is somewhere between 0.5 and 0.75. Thus, allowing crossover to occur too often has negative effects on the population's fitness. This may be because valuable individuals are sometimes lost in the process of crossover and the population is unable to achieve high levels of fitness as a result.

4.2 Mutation

As a result of the encoding scheme employed, the mutation operator in the artificial life simulator may only operate at the weight encoding level and bit–level mutation is not possible as it could result in the generation of invalid gene codes. Instead, weight encodings are modified by a percentage in the range -200% to +200%.

Since mutation is essentially a disruptive element in the genetic algorithm, one would expect that high levels of mutation would cause a decrease in the population's fitness. This expectation appears to be justified by the results presented in fig. 2. The 0.02 mutation rate appears to be the best available option, as it performs better than no mutation and pushed to higher levels, the population's fitness begins to fall. However, 0.02 mutation may not be necessarily the optimum level. It is

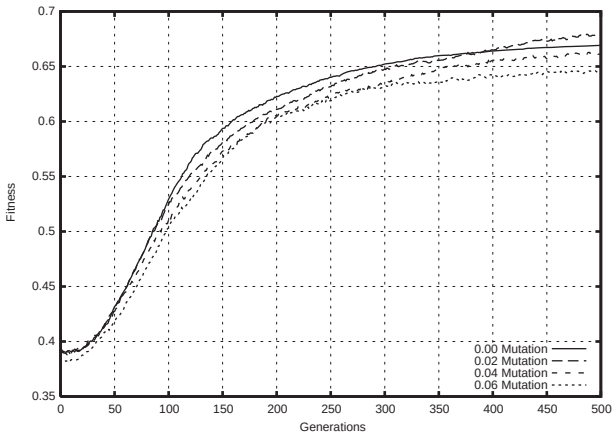

Figure 2. Mutation Rate Determination

possible that a better performance could be obtained by setting the mutation level at between 0.00 and 0.02 or between 0.02 and 0.04.

4.3 Training Cycles

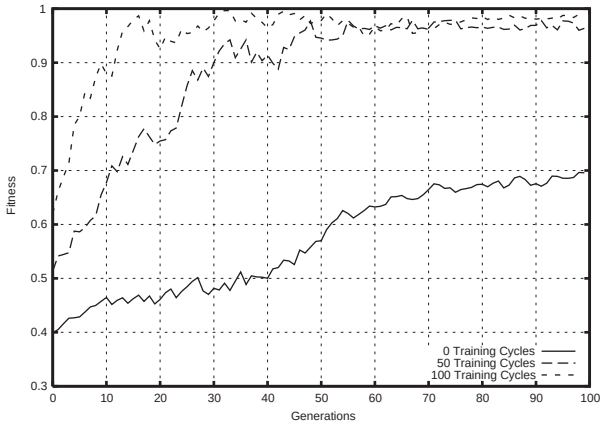

Figure 3. Training Rate Determination

The training cycle rate determines how often error back propagation is performed on a neural network. As is evident from fig. 3, the population performs better as more training is applied. It would seem that the more training is applied to the population, the faster the initial fitness jump. In the case of 50 training cycles the population's fitness reaches 0.9 in 30 generations. When the training cycles are increased to 100, the population reaches the same level in a mere 12 generations. It is interesting to note, that despite this increased acceleration, the population's fitness converges at generation 60 for both these rates. Only after generation 75 does the additional training finally overtake the 50 training cycle graph. It would appear from these results that 50 training cycles should be ample to give the population a high level of fitness.

5 EVOLUTIONARY RATE DETERMINATION

This set of experiments was designed to allow populations to adapt the rates of crossover, mutation and training to determine the optimal level of each. In order to achieve this goal, changes were made to the structure of the genetic codes of each member of the population to allow the incorporation of rate data.

At the beginning of an experiment, randomly determined rate values are appended to each individual's genetic code. These values are integers in the range 0 – 100 and are encoded into 6–bit strings. The encoded rate values undergo separate crossover and mutation processes following those of the main genetic string and using the same probability values.

The behaviour of each rate value varies according to its operator. When two networks are selected as parents, the crossover rate values for each is examined and an average is taken to determine the crossover rate to be used. The parent gene codes will then undergo crossover according to this probability value.

The mutation operator is used on the offspring gene codes generated following the crossover process. The operator uses the inherited rate value of an individual offspring to determine its mutation rate. Each weight encoding in the offspring's gene code will be modified according to this probability value.

The training rate is also inherited and represents the number of training cycles that a neural network receives. At each cycle, the network's error is reduced using error back propagation, so in general, the more cycles a network receives, the better it performs.

Enough randomly generated networks must be produced in order to create a sufficiently large spread of values to allow potentially beneficial values to propagate across generations. The population size must also be small enough to allow the experiments to be undertaken in a reasonable length of time. After several trial runs, it was determined that the optimum population size to set the experiments was 500.

As the population evolves, individuals are selected according to their ability to solve the given problem. At the same time, rate values which they have inherited from past generations also evolve. Values that cause networks to perform poorly (such as excessively high mutation rates) should become extinct from the population as the badly performing individual is less likely to be selected for mating. Similarly, values which create the least disruption and the most benefit to an individual's performance will become more prominent in the population.

5.1 Experiment setup

500 networks are randomly generated at the start of each experiment to allow a suitably larger spread of rate values to be present in the initial population. The networks are then allowed to evolve to solve the 3–bit parity problem for 100 generations. The crossover rate was set to 0.5, the mutation rate to 0.02 and training at 10 training cycles, where these rates where not the focus of the experiment. Each experiment was carried out 20 times.

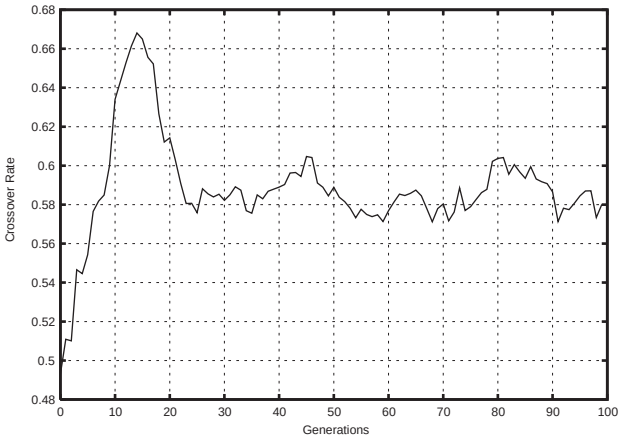

Figure 4. Crossover Rate Evolution

5.2 Evolution of Crossover Rates

The results for the evolution of the crossover rate are illustrated in fig. 4. The population begins with an average crossover rate of 0.48 which quickly ascends to over 0.66. However, the rate begins to fall sharply before stabilising somewhat at around 0.58—0.59. The initial sharp rise is most likely a result the initial population's poor fitness — a high crossover rate is useful in such situations because it allows more comprehensive probing of the search space to occur. In response to an increase in population fitness, the crossover rate then drops as exhaustive probing is no longer desirable. The final crossover rate evolved by the population seems to fit with the predicted rate of crossover (between 0.5 and 0.75) determined in the previous experiment set.

5.3 Evolution of Mutation Rates

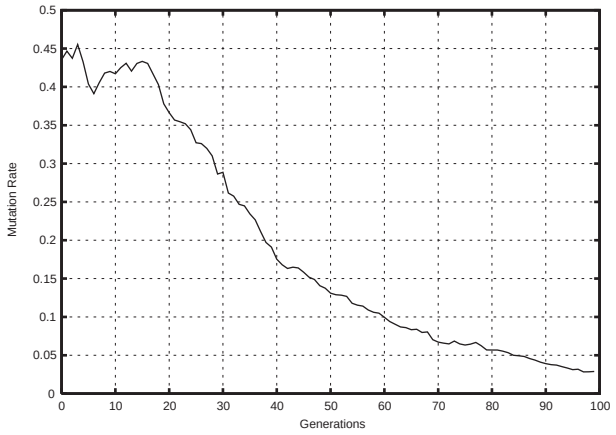

Figure 5. Mutation Rate Evolution

Fig. 5 shows the evolution of the population's mutation rate. This experiment begins with an average mutation rate of 0.45 – one which is large enough to have disastrous consequences on any offspring subjected to it. This is reflected by the results, which show that the population rejects this value and forces it to plummet down to below 0.05. At this point, the drop slows down considerably indicating that the population has found a near optimum level of mutation. Following the generation 90, the mutation rate levels to around 0.025. This value lies in the range 0.02 – 0.04, one of the ranges predicted from the previous experiment set.

5.4 Evolution of Training Cycles

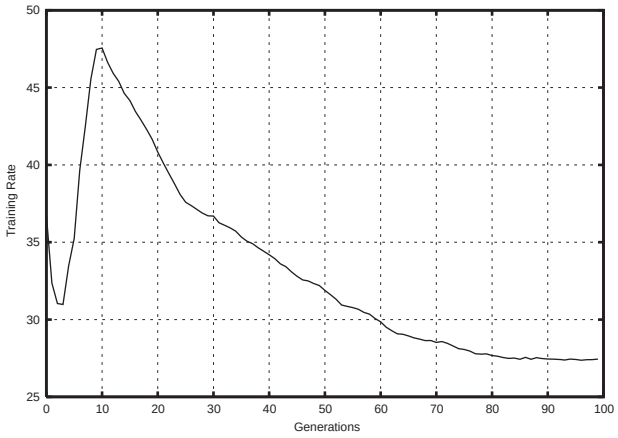

Figure 6. Training Rate Evolution

The evolution of the training rate was treated slightly differently from the others to favour networks able to acheive high fitness without the need for excessive training. Since the training process is very time consuming, it would be useful to adapt the training rate to give the maximum fitness improvement for the least effort. To encourage the evolution of such a training rate, the fitness function was altered to reward networks with lower training rate values.

The results illustrated in fig. 6 show an initial sharp increase in training rate, followed by a gradual fall to only 27 training cycles. It should be noted that the population's fitness during this gradual fall remained above 0.95 at all times. The population spends the initial 10 generations undergoing intensive training and once the networks have acheived a sufficiently high fitness, the population's reliance on training appears to subside.

5.5 Comparison of Performance of Evolved Rates to that of Designed Rates

A final experiment was performed to observe the performance of the evolved rates versus the designed rates. The performance of two populations are illustrated in fig. 7. The first population uses the crossover, mutation and training rates arrived at from iterative experimentation in section 4, that is crossover at 0.625, mutation at 0.02 and training at 100 cycles.

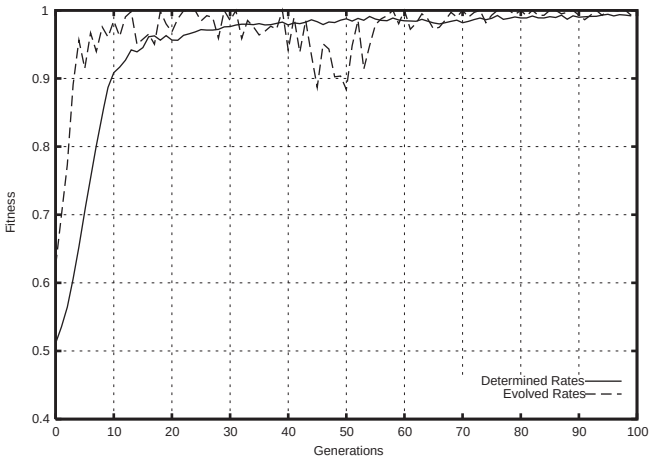

Figure 7. Comparison of Performance of Evolved Rates to that of Designed Rates

The second population uses the rates evolved for crossover (0.59) and mutation (0.025) but is allowed to evolve and adapt the training rate as in section 5.4. This is because the evolution of training rate does not appear to result in an optimum value, rather it is the process of adaptation that results in improved performance. Both populations were allowed to evolve for 100 generations and the experiment was repeated 20 times.

The results show that the evolved rates slightly outperform the designed rates. However, the designed rates population is using a training rate of 100 cycles for the entire experiment, thus inhibiting its performance in terms of time, while the evolved rates population is adapting its training rate as each generation passes, allowing it to reduce the training rate considerably and thus greatly out–performing the determined rates population in terms of execution time.

6 CONCLUSION

The results of the evolutionary rate determination experiments show that it is possible to determine the optimal setting for crossover and mutation for a given problem set. This effectively eliminates the requirement for trial and error estimation of these rates. The values achieved are very close to the values predicted by the first set of experiments, where the rates were modified in order to find the optimum by trial and error.

In addition, the results show that allowing a population to determine its own training rate dynamically can produce a better performance both in terms of fitness and execution time. Future work will examine the relationship between each rate and attempt to evolve all rates simulatenously in both static and changing environments.

ACKNOWLEDGEMENTS

This research is funded by the Irish Research Council for Science, Engineering and Technology.

REFERENCES

[1] P. J. Angeline, G. M. Saunders, and J. P. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65, January 1994.

[2] R. K. Belew, J. McInerney, and N. N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, pages 511–547. Addison-Wesley, Redwood City, CA, 1992.

[3] B. MacLennan. Synthetic ethology: An approach to the study of communication. In *Artificial Life II: The Second Workshop on the Synthesis and Simulation of Living Systems, Santa Fe Institute Studies in the Sciences of Complexity*, pages 631–635, 1992.

[4] D. Moriarty and R. Miikkulainen. Discovering complex othello strategies through evolutionary neural networks. *Connection Science*, 7(3–4):195–209, 1995.

[5] W. M. Spears. Adapting crossover in evolutionary algorithms. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proc. of the Fourth Annual Conference on Evolutionary Programming*, pages 367–384, Cambridge, MA, 1995. MIT Press.

[6] D. Curran and C. O'Riordan. Learning in artificial life societies. In *Report number nuig-it-220202. Technical Report, Dept. of IT.* NUI, Galway, 2002.

[7] D. Curran and C. O'Riordan. On the design of an artificial life simulator. In R.J.Howlett V.Palade and L.C.Jain, editors, *Proceedings of the Seventh International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES 2003)*, University of Oxford, United Kingdom, 2003.

[8] D. Curran and C. O'Riordan. Artificial life simulation using marker based encoding. In *Proceedings of the 2003 International Conference on Artificial Intelligence (IC-AI'03)*, volume II, pages 665–668, Las Vegas, Nevada, USA, 2003.

[9] H. Kitano. Designing neural networks using genetic algorithm with graph generation system. In *Complex Systems, 4, 461-476*, 1990.

[10] P. M. Todd G. F. Miller and S. U. Hedge. Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, pages 379–384, 1989.