Artificial Life Simulation Using Marker-Based Encoding

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

keywords: Artificial life, simulation, neural networks, genetic algorithms

Abstract

This paper describes the design of an artificial life simulator. The simulator uses a genetic algorithm to evolve a population of neural networks to solve a presented set of problems. The simulator has been designed to facilitate experimentation in combining different forms of learning (evolutionary algorithms and neural networks). We present results obtained in simulations where the population is evolved to solve certain problems. The simulations are designed to show the population's progress when presented with problems of increasing difficulty using evolutionary algorithms and neural networks both individually and in combination.

I. Introduction

The combination of genetic algorithms and neural networks provides a framework for tackling often intractable problems which would be difficult to approach using either method in isolation. Combining the evolutionary approach of genetic algorithms with the active data processing capability of neural networks has proved successful in the past. In a typical implementation, the genetic algorithm is utilised to generate a population of random 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 focus of this paper is the development of an artificial life simulator combining genetic algorithms and neural networks to solve a given problem set. The purpose of the simulator is to ascertain to what degree the addition of the neural network capability aids the discovery of optimum solutions. This was achieved through a set of experiments with problem sets of increasing complexity.

The remainder of the paper is organised as follows. The next section describes both neural networks and genetic algorithms and approaches to combine the two. Section 3 outlines the architecture and functionality of the simulator. Section 4 describes the set of experiments used as well as the results obtained.

II. Related work

Genetic algorithms have long been used as an efficient approach to solution optimisation[1]. Based on a Darwinian evolutionary scheme, potential solutions are mapped to genetic codes which, in the case of the canonical genetic algorithm, are represented by bit patterns. Each of these solutions is tested and a portion are selected to be combined to create the next generation of solutions. Using this mechanism in a iterative manner, the approach has been shown to solve a variety of problems[2].

Artificial neural networks are an alternative method of machine learning based on the biological structure of the nervous systems of living organisms. A neural network is composed of nodes and interconnecting links with adjustable weights which are modified in order to alter a network's response to outside stimuli. In artificial neural networks, these stimuli take the form of bit patterns which are fed to the network via entry input nodes. The stimulus signal is altered by its movement through the network arriving finally at the output nodes as the network's response. Through a process of training, the network adjusts its links, thereby altering its response to particular stimuli. This modification process occurs through a learning algorithm which typically examines the network's output and compares it with pre-defined correct responses. A measure of error is thereby extracted and links are altered to reduce the error for the next training iteration.

While the two approaches have benefits in their own right, there are disadvantages associated with their use. Not all problems and solutions may be readily mapped to the genetic code structure which the genetic algorithm requires. Artificial neural networks implementations may also suffer from this problem, but the real issue is the design of the network's architecture and link structures. It can be very difficult to ascertain the optimal number of nodes, links and weight values for a network for a given problem.

To address this latter issue, a combination of genetic algorithms and neural networks has been proven to be successful in a variety of problem domains ranging from neural network design optimisation [3], [4], [5] to games[6], [7].

III. Experimental Setup

The architecture of the artificial life simulator can be seen as a hierarchical structure. At the top-level of the simulator is a command interpreter which allows users to define an experiment's variables including the number of networks, the number of generations to run the experiment, mutation and crossover rates and the actual problem set which the population will be attempting to solve.

The neural network layer takes the variables set using the command interpreter and initialises a given number of neural networks. The layer then performs training and testing of the networks according to the parameters of the experiment. These network memory structures are then passed to the encoding layer which transforms them into genetic code structures for use in the genetic algorithm. The encoding mechanism used for this set of experiments is a modified version of marker based encoding.

The genetic algorithm layer uses the genetic codes and the data retrieved from the neural network layer's testing of the networks to perform its genetic operators on the population. A new population is produced in the form of genetic codes. These are passed to the decoding layer which transforms each code into a new neural network structure. These structures are then passed up to the neural network layer for a new experiment iteration. Once the required number of generations has been reached the experiment finishes.

The simulator therefore only provides a framework for which each component must be built. The standard simulator as used in these experiments contains a number of default components for each of the layers, but these can easily be expanded upon to include more complex functionality. The next sections outline the functionality of the default components of the simulator used in the experiments.

A. Genetic Operators

In order to generate new genetic material, the parents of new networks undergo the process of crossover. The operator selects points along each parent gene code and swaps portions of the gene code within each parent. Two new networks are created in this manner. Two-point crossover is employed in this implementation (any type could be employed).

The mutation operator introduces additional noise into the genetic algorithm process thereby allowing potentially useful and unexplored regions of problem space to be probed. The mutation operator usually functions by making alterations on the gene code itself, most typically by altering specific values randomly selected from the entire gene code.

In this set of experiments, weight mutation is employed. A bit-level approach utilising Gray encoding was considered for this operator but was abandoned after several trial experiments. Instead a more broad mutation operator is now used. This takes the weight value and increases/decreases the value according to a random percentage (200%). This approach was found, empirically, to be more successful and was adopted for this set of experiments.

IV. Experiments

The problem set employed for these experiments was n-bit parity. Each network was exposed to 3 and 4 bit patterns and trained to determine the parity of each pattern. The number of training iterations was also varied from 0, 10 and 100 iterations. Six experiments were carried out with populations of 500 networks spanning 500-1000 generations although in some cases the experiment was cut short because of evident fitness stagnation. The general aim of these experiments was twofold: to demonstrate the validity of the simulator and to ascertain how much the training or learning process affects each population's fitness with an increasingly difficult problem set.

A. 3 bit parity

This initial set of experiments exposes the population to 3-bit parity patterns and represents a problem of modest difficulty since a neural network must evolve to solve all 9 solutions to attain high levels of fitness.

A.1 No Training

With no training, it is clear that the population has difficulty in attaining high levels of fitness (figure 1). The graph begins with a stagnation period after which a fitness jump occurs, but does not result in a large scale fitness climb. The fitness of the population has not stagnated however, rather it appears to be climbing at a much reduced rate. It may be possible that given enough time, the experiment would have resulted in higher levels of fitness than those illustrated.

A.2 10 Training Iterations

When the networks are given some training, an interesting phenomenon occurs. The graph appears to be much smoother than the previous experiment, but the shape of the population's fitness progress is almost identical (figure

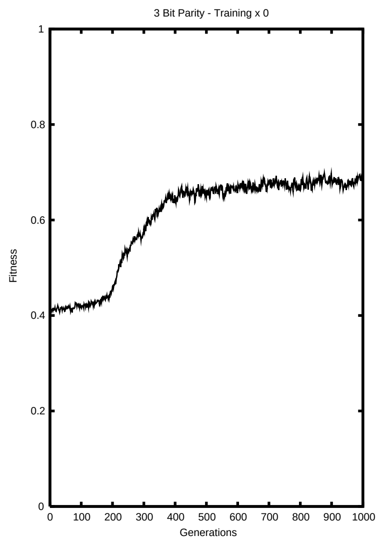

Fig. 1. *3 bit parity-no training*

2). There is a clear stagnation of fitness even after the jump occurs. It would seem that the learning process is only fulfilling one of its characteristic tasks, that of smoothing the population's progress. It has not, however aided in the actual fitness of the population.

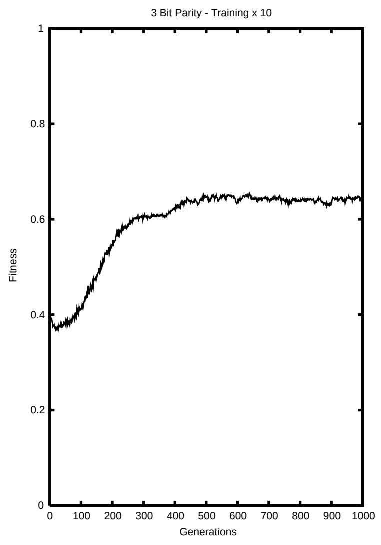

Fig. 2. *3 bit parity-10 training iterations*

A.3 100 Training Iterations

Given more training iterations, the population immediately rises to very high levels of fitness (figure 3). This indicates that some threshold of training iterations exists for this problem. Clearly, for the 3-bit parity problem, a higher level of learning is required to make good progress.

B. 4 bit parity

This second set of experiments contains the most interesting results, as it represents the most difficult problem of the set. The number of solutions has increased to 16, thus requiring a significantly more complex neural network architecture to be evolved.

B.1 No Training

The difficulty of the 4-bit parity problem can be clearly witnessed when examining the results of the no training experiment (figure 4). There is little progress in the population's fitness throughout the experiment. Only after 500

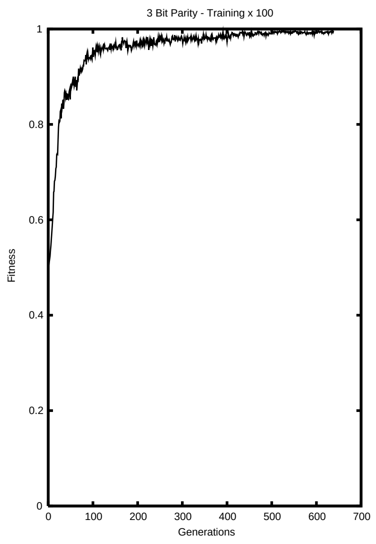

Fig. 3. *3 bit parity-100 training iterations*

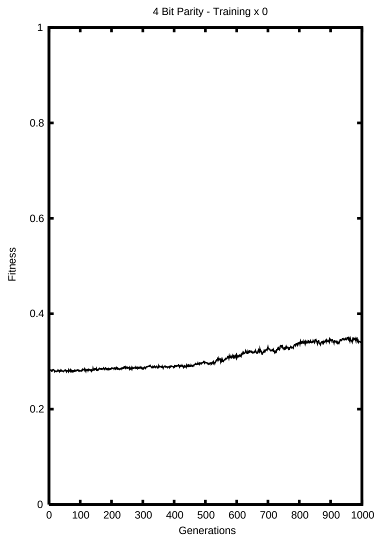

Fig. 4. *4 bit parity-no training*

generations is there any indication that the genetic operators are functioning at all. Clearly the problem is too complex for the genetic algorithm to solve unaided in any reasonable length of time.

B.2 10 Training Iterations

When some training is introduced, a certain amount of progress is made by the population (figure 5). A substantial fitness jump occurs at generation 450, but the fitness flattens out soon after at around 0.4. The learning process is aiding the genetic operators, but in this case it is not enough. The population's fitness following the initial jump shows no signs of improving, even if the experiment was continued further.

B.3 100 Training Iterations

The final figure illustrates the fact that the more difficult a problem becomes, the more the learning process is needed to guide the evolution of the neural network population (figure 6). There is a huge jump in fitness towards the beginning of the experiment, giving rise to a continual rise in fitness reaching to more than 0.9 indicating that the problem has been solved by the population.

V. Conclusion

The results achieved with the Artificial Life simulator indicate that for simple problems, the genetic algorithm component of the simulator is capable of guiding the population towards acceptable levels of fitness. However, as the

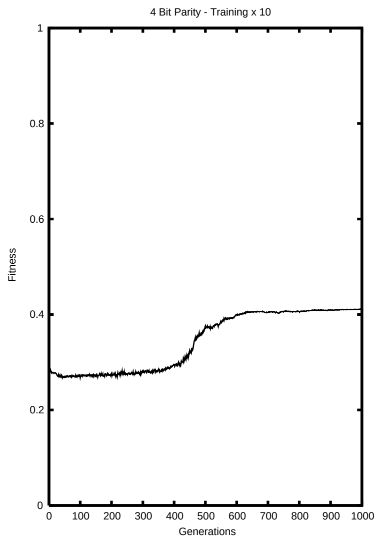

Fig. 5. *4 bit parity-10 training iterations*

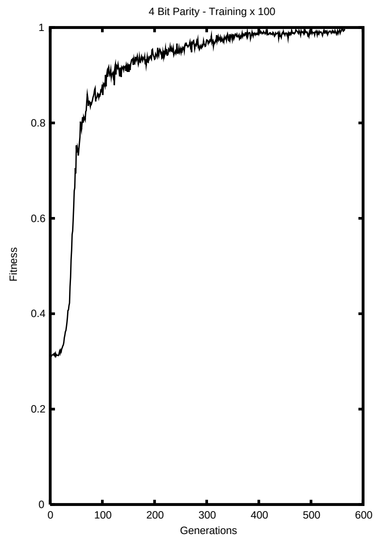

Fig. 6. *4 bit parity-100 training iterations*

problem difficulty increases, this component becomes less and less capable of sustaining population fitness growth. In these cases, it is apparent that the learning mechanism is increasingly required and that without it, the population is not capable of survival. In conclusion, the simulator has shown itself capable of solving this set of problems. Future work will concentrate on problems of increased complexity and the effects of environment changes on the robustness of populations.

VI. Acknowledgements

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

References

[1] J. H. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor MI: The University of Michigan Press, 1975.
[2] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA, Addison-Wesley, 1989.
[3] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity, 1990.
[4] R. K. Belew, J. McInerney, and N. N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 511–547. Addison-Wesley, Redwood City, CA, 1992.
[5] X. Yao. Evolving artificial neural networks. *PIEEE: Proceedings of the IEEE*, 87, 1999.
[6] D. Moriarty and R. Miikkulainen. Discovering complex othello strategies through evolutionary neural networks. *Connection Science*, 7(3–4):195–209, 1995.
[7] N. Richards, D. Moriarty, P. McQuesten, and R. Miikkulainen. Evolving neural networks to play go. In *Proceedings of the 7th International Conference on Genetic Algorithms*, East Lansing, MI, 1997.