Brock University

Department of Computer Science

Autonomous Life Agent Using Recurrent Neural Networks and Genetic Algorithms

Tony Abou-Assaleh and Jianna Zhang

Tech. Report # CS-00-03

Brock University
Dept. of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
www.cosc.brocku.ca

Autonomous Life Agent Using Recurrent Neural Networks and Genetic Algorithms

Tony Abou-Assaleh
Computer Science Dept., Brock University
St. Catharines, ON, Canada, L2S 3A1
Phone: (905) 688-5550 ext. 3513
Email: taa@acm.org

Dr. Jianna Zhang
Computer Science Dept., Brock University
St. Catharines, ON, Canada, L2S 3A1
Phone: (905) 688-5550 ext. 3494
Email: jianna@cosc.brocku.ca

Abstract

Studies of artificial life (alife) attempt to simulate simple living beings. On the other hand, autonomous agents researchers are interested in building agents that are able to complete a particular task without supervision. In this research, these two areas of artificial intelligence are combined together into what we call "Autonomous Life Agent" (ALA). ALA is an artificial agent that is sent to some environment to live in without any supervision or any predefined behaviour rules. The primary goal of the agent is to learn how to survive in the artificial environment it lives in. In this research, we utilized a recurrent neural network (RNN) to determine the agent's actions. An ALA Training System was developed that evolves RNNs using the genetic algorithms approach.

1 INTRODUCTION

Many studies have attempted to simulate a self-navigating robot [Yamamura and Onozuka, 1998] & [Tse et al., 1998], however, this project looks at autonomous agents from a different prospective. This research has two objectives. First, it provides the software architecture for a simulated autonomous robot that is able to maintain itself and collect the required energy to remain functional. Secondly, it illustrates the power of using genetic algorithms to evolve recurrent neural networks.

Such an agent has many practical applications. A more advanced version of this software would enable a robot to be sent to a totally unknown environment. For instance, a robot could be sent to space without requiring any monitoring or supervision from earth.

This paper focuses on the technical details related to the ongoing research by the authors regarding ALAs. Section 2 describes some key concepts related to the developed system. Section 3 introduces the ALA architecture and the ALA Training System. Section 4 presents the experimental results. Section 5 concludes with remarks about the significance of this research. Finally, Section 6 discusses future work.

2 RELATED CONCEPTS

This section provides a quick overview of neural networks, genetic algorithms, and reinforcement learning.

2.1 NEURAL NETWORKS

Artificial neural networks consist of a number of artificial neurons that simulate the characteristics of biological neurons in terms of the relationship between the input and the output [Hagan et al., 1996]. As Figure 1 illustrates, every neuron has a set of input connections, a bias b, and an output a. Every input link has an input value p and a weight w. The total input, n, is calculated by multiplying every input value by the corresponding link weight and summing the products. The output of the neuron can be viewed as function: $a = f(\sum wp+b)$. Various activation functions can be used to determine when and how the neuron fires (produces output).

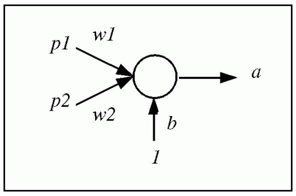

Figure 1: A Single Artificial Neuron

Neurons are grouped into layers with links from one layer to another, which forms a neural network (see Figure 2). If the links are always in one direction then the network is called "feed forward neural network." On the other hand, if the network has backward connections then it is called "recurrent neural network."

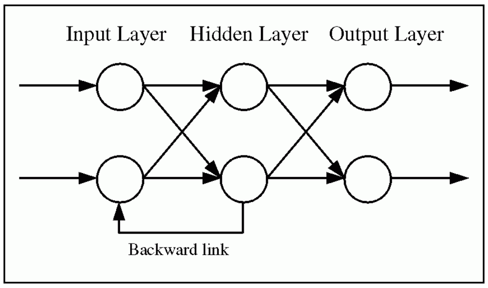

Figure 2: Three-Layer Recurrent Neural Network

2.2 GENETIC ALGORITHMS

Genetic algorithms are analogous to natural evolution [Holland, 1992]. They evolve populations consisting of individuals. Every individual is defined by his genes. After creating an initial population, individuals with higher fitness are more likely to survive and produce offsprings. When an individual is selected based on his fitness, one of the following three operations is used: i) crossover, ii) mutation, and iii) reproduction. Crossover requires two parents. A part of the genetic data is exchanged between the two parents to produce two children. The advantage of this is to combine good features from both parents to produce a better individual. Mutation changes some of the genes to random values, which allows introducing new genetic information not currently present in the population. Some individuals may be reproduced (copied) to preserve good combinations of genetic data.

2.3 REINFORCEMENT LEARNING

Reinforcement learning obligates the learning agent to maximize the accumulative reinforcement signal that it receives. A positive reinforcement signal yields a reward for the action taken by the agent, while a negative signal yields a penalty. Onat et al have a good description of reinforcement leaning [Onat et al., 1998].

3 ALA ARCHITECTURE

There are two issues to be considered when designing an autonomous agent. The first category is related to the simulated environment. Some researchers suggest that a continuous action space can be used [Lee et al., 1998]. However, the actual implementation that they propose divides the continuous action space into small discrete steps. Although a continuous action space may seem to be the ultimate goal, a discrete action space is more practical for the implemented agent. Also, when a non-Markovian domain was used in previous studies [Mizutani and Dreyfus, 1998], the domain of the problem was artificially manipulated and did not represent real world settings. For this reason, a stochastic domain is chosen in our architecture. As well, due to the nature of the problem, it is necessary to introduce hidden states where only partial information about the world is available to the agent as input. Other researchers have successfully implemented such a domain using recurrent neural networks [Pacut, 1998], [Onat et al., 1998], & [Paraskevopoulos et al., 1998].

The second issue is related to the design of the agent itself. The primary goal of the agent is to stay alive. Thus, the agent has to operate continuously without an explicit terminal state [Riedmiller, 1998]. Agents are usually implemented using an Action-Critic model that consists of two neural networks [Pacut, 1998] & [Lee et al., 1998]. The first of these is used to evaluate the external reinforcement value and produce a more accurate internal reinforcement value. This value is then fed into the second network that determines the action to be taken. Mizutani and Dreyfus have implemented the Action-Critic method using a single neural network with these outputs [Mizutani and Dreyfus, 1998]. A similar approach is used in the ALA design since, after all, in an animals brain there is no discrete distinction between these two functions.

The ALA agent has two internal variables that define its internal state: energy level and maintenance level. The agent is considered alive as long as both these variables have positive values. The agent dies when their value reaches zero. As the agent acts in its world, its energy and maintenance levels are reduced. The following section describes how the agent can acquire energy and maintenance to remain alive.

3.1 WORLD REPRESENTATION

The world in which we want our agent to survive is a 4x4 grid consisting of three types of cells: empty cells, energy cells, and home cells. The agent can travel over any cell without limitations. Moving into any of the cells results in losing some energy and causing some damage for the agent by decreasing its maintenance level. If the agent chooses to "stay" as an action while positioning over a home cell, its internal maintenance level increases by the amount provided by that cell. Similarly, staying over an energy cell consumes all the energy in that cell increasing the internal energy level up to a maximum limit. However, once the energy is consumed from an energy cell, there is delay before the cell can provide energy once again. Thus, the agent is forced to search for another energy cell.

3.2 ALA STRUCTURE

The brain of our agent is a recurrent neural network with three layers: input layer, hidden layer, and output layer. Some neurons in the hidden layer have backward links with a time delay to form what is called "context nodes". This means that the output from some nodes in the hidden layer at time t is used as input from the context nodes at time t+1. The activation function for all the neurons is linear.

The reason for selecting RNNs to represent our agent is threefold. First, they can autonomously learn the environment. No information about the world is available to the agent other than the immediate sensory input. Some aspects of the world such as the amount of energy and maintenance that a cell can provide are unknown to the agent. Second, they can produce sequences of actions where a particular action selection depends on previously selected actions. Third, the external reinforcement value need not be accurate and can be calculated implicitly within the network. For instance, when the agent receives energy or maintenance from the world, the values of the corresponding internal variables are updated and provided to the network as input. During training, the RNNs develop internal interpretations of these values and decide which actions are more desirable for the purpose of staying alive.

Instead of adapting one of the standard training algorithms to determine the weights of the RNN, the genetic algorithms method was used. This eliminated the complexity and the limitations of other algorithms. Also it enabled selecting more appropriate values for the initial outputs of the context nodes which had to be fixed in the other training algorithms.

3.3 ALA TRAINING SYSTEM

The training system is divided into three major parts: the control unit, the genetic algorithms engine, and the ALA evaluator. The control unit initializes the other two units and controls the operation of the genetic algorithms engine. It also acts as an interface between the user and the system. The genetic algorithms engine evolves recurrent neural networks and sends them to the ALA evaluator to determine their fitness values.

Below is the training algorithm used by the ALA Training System:

1. Control unit: initialize all the parameters
2. Genetic algorithms engine: generate initial random population with gene values between -1.0 and +1.0.
3. ALA evaluator: evaluate individuals and assign them scores
   (a) Read current input: sensors, external reinforcement signal (energy and maintenance levels), and internal feedback
   (b) Select the action with the highest output
   (c) Update position, energy and maintenance level
   (d) Repeat until agent dies or maximum age is reached
   (e) Return the agents age to genetic algorithms engine as a fitness score
4. Genetic algorithms engine: Select an operation probabilistically from: crossover, mutation, and reproduction; and apply it to individuals selected based on fitness.
5. Control unit: repeat from step 3 until an optimal solution is found or the maximum number of generations is reached.
6. Control unit: save the best resulting ALA definition. An optimal solution is defined as an RNN that can live in the ALA evaluator up to the maximum age specified by the user.

4 THE EXPERIMENT

The ALA architecture discussed earlier in this paper was implemented in the ALA Training System. This system was used to evolve a number of agents using different parameters.

Due to the difficulty of the search problem and the random nature of the genetic algorithms, it was not always possible to find the optimal solution within a relatively short time. This issue was solved by running multiple simulations in parallel on a multiprocessor computer. Our tests have indicated that on average 4 out of 10 runs find the solution within a few hours.

Table 1: ALA Training System Parameters

PARAMETER	VALUE
Number of generations (max.)	2000
Population size	500
Crossover rate	0.5
Mutation rate	0.4
Reproduction rate	0.1
Tournament size	5
Hidden nodes	20
Context nodes	15
World (4x4 grid) 0 = empty cell H = home cell E = energy cell H000	H000 0000 000E 0E00
Initial Location	0, 0
Energy in E cells	1.0
Maintenance in H cells	1.0
Energy growth delay	7
Initial ALA energy	1.0
Initial ALA maintenance	1.0
Change rate of energy	0.1
Change rate of maintenance	0.1
Maximum age	500

To illustrate the efficiency of our approach, we describe a sample single run in details. The experiment was conducted on an SGI Origin 2000 computer with 16 multiprocessing MIPS R12000 CPU's running IRIX 6.5 operation system. The language used to develop the training system was Java. The parameters used are listed in Table 1.

The genetic algorithms engine parameters were selected on a trial and error basis. Our observations did not show increase in efficiency when the population size was increased above 500. On the other hand, finding a solution with a population size less than 500 requires a much higher number of generations. Similarly, we found that 5 is the most appropriate tournament size. If the tournament size is less than 5 then the average age of the population keeps fluctuating and does not increase through time. Larger tournament sizes resulted in stabilizing at some age without any further progress. ALA and environment parameters were chosen in a way that permits different behaviour patterns to be valid.

Ten independent training evolutions (runs) were executed simultaneously. Five of them found the optimal solution within the maximum number of generations limit. Table 2 shows the running time for each run along with the number of generations it evolved, the

Table 2: Final Results of 10 Runs

Run No.	Running Time hh:mm:ss	Generations	Average Age	Best Age
1	02:23:34	2000	16.844	20
2	03:51:51	2000	44.806	59
3	00:08:21	84	106.114	500
4	02:18:08	2000	17.440	20
5	01:13:32	554	55.566	500
6	01:53:36	260	344.968	500
7	07:54:23	1664	271.614	500
8	01:00:02	676	99.166	500
9	02:20:19	2000	17.764	25
10	02:21:18	2000	18.608	21

average agent age of the terminating population, and the age of the best individual found by the run. The age is the number of time actions the agent made before it died or reached the maximum age. The performances of the best and the worst runs that found a solution are shown in figure 4 and 5 respectively.

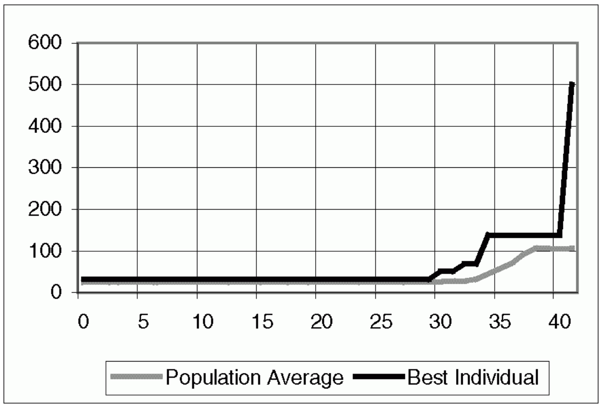

Figure 3: Run Number 3 Training Progress

It is evident in these results that when using numerous evolutions running simultaneously the solution can be found very fast. Normally, once a solution is found all the other runs are terminated, which would save a lot of computational resources. However, for the purpose of observing the performance and efficiency, we waited for the runs to normally terminate.

5 CONCLUSION

Very few papers describe the application of genetic algorithms to evolve

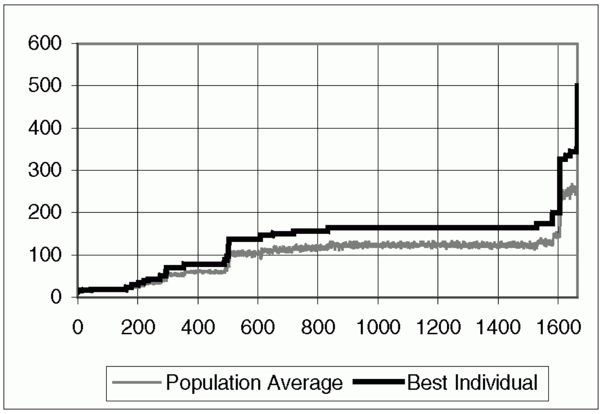

Figure 4: Run Number 7 Training Progress

neural networks [Weisman and Pollack, 1995] and recurrent neural networks [Syed, 1995]. Although Syed conducted an intensive research on optimizing the parameters for the genetic algorithms, he tested his finding only on very simple RNNs with up to 5 neurons in the hidden layer.

Our research has shown that the genetic algorithms approach is very powerful and efficient in evolving recurrent neural networks. Regardless of the relative complexity of the ALA's RNN in comparison with other studies [Syed, 1995], our training system is able to find a solution in a short time. Furthermore, the ALA architecture is greatly extendible. We are currently running experiments on larger worlds (10x10) with more complex structures.

The experiment results strongly support the arguments discussed earlier. RNNs can automatically discover hidden world states that are obscure to the agent, and the reinforcement signal can be determined internally. In addition, they can autonomously acquire world knowledge and develop behaviours without explicit predefined rules.

Our in-depth research on genetic algorithms and recurrent neural networks has led to developing a robust architecture for the ALA agent. It has also set the grounds for future work.

6 FUTURE WORK

The success of our initial studies regarding autonomous life agents indicates that the proposed ALA architecture is worth further investigation and improvements.

Our ongoing research on ALAs focuses making the system more dynamic and flexible. We are working on introducing more internal variables to the agent, use multiple hidden layers in the RNN, and evolve agents that can live in unseen environments which are similar to the training environments.

Another interesting area to investigate is to allow the agent to live in unlimited environments and let the RNN autonomously expand as the agent discovers new places. Likewise, when the agent realizes that the newly discovered places are useless then its RNN may shrink to reduce the computation time.

Acknowledgments

The first author is a recipient of the AAAI Student Travel Grant and is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under the USRA award.

The second author is supported by the NSERC of Canada under Grant 228142-2000 and by the GRG Brock Start Grant.

Special thanks and appreciation to Dr. Brian Ross from the Computer Science Department at Brock University for his help and support.

References

[Hagan et al., 1996] Hagan, M., Demuth, H., and Beale, M. (1996). *Neural Network Design.* PWS Publishing Company, Boston, MA.

[Holland, 1992] Holland, J. (1992). *Adaption in Natural and Artificial Systems.* MIT Press.

[Lee et al., 1998] Lee, J., Oh, S., and Choi, D. (1998). Td based reinforcement learning using neural networks in control problems with continuous action space. In *Proc. IEEE International Conference on Neural Networks*, pages 2028–2033.

[Mizutani and Dreyfus, 1998] Mizutani, E. and Dreyfus, S. (1998). Totally model-free reinforcement learning by actor-critic elman network in non-markovian domains. In *Proc. IEEE International Conference on Neural Networks*, pages 2016–2021.

[Onat et al., 1998] Onat, A., Kita, H., and Nishikawa, Y. (1998). Recurrent neural networks for reinforcement learning: Architecture, learning algorithms and internal representation. In *Proc. IEEE International Conference on Neural Networks*, pages 2010–2015.

[Pacut, 1998] Pacut, A. (1998). Common framework of certain reinforcement schedules. In *Proc. IEEE International Conference on Neural Networks*, pages 2004–2009.

[Paraskevopoulos et al., 1998] Paraskevopoulos, V., Heywood, M., and Chatwin, C. (1998). Modular srv reinforcement learning: An architecture for non-linear control. In *Proc. IEEE International Conference on Neural Networks*, pages 2034–2037.

[Riedmiller, 1998] Riedmiller, M. (1998). Reinforcement learning without an explicit terminal state. In *Proc. IEEE International Conference on Neural Networks*, pages 1998–2003.

[Syed, 1995] Syed, O. (1995). Applying genetic algorithms to recurrent neural networks for learning network parameters and architecture. Master's thesis, Department of Electrical Engineering, Case Western Reserve University.

[Tse et al., 1998] Tse, P., Lang, S., Leung, K., and Sze, H. (1998). Design of a navigation system for a household mobile robot using neural networks. In *Proc. IEEE International Conference on Neural Networks*, pages 2151–2156.

[Weisman and Pollack, 1995] Weisman, O. and Pollack, Z. (1995). Neural network using genetic algorithm. Internet site.

[Yamamura and Onozuka, 1998] Yamamura, M. and Onozuka, T. (1998). Reinforcement learning with knowledge by using a stochastic gradient method on a bayesian network. In *Proc. IEEE International Conference on Neural Networks*, pages 2045–2050.