Performance Evaluation of Neural Architectures for Sequential Tasks

Genci Capi[1], and Kenji Doya[2]

[1] Faculty of Information Engineering
Fukuoka Institute of Technology
3-30-1 Wajiro-Higashi, Higashi-ku, Fukuoka, 811-0295, Japan
[2] CREST, JST, ATR, Computational Neuroscience Laboratories
"Keihanna Science City", Kyoto, 619-0288, Japan
capi@fit.ac.jp

Abstract

This paper considers a sequential task where the agent has to alternatively visit two rewarding sites to obtain food and water after first visiting the nest. To achieve a better fitness, the agent must have a working memory to reach the target position, must ignore irrelevant sensory inputs, and at a higher level, it has to deal with the non-Markovian order of sequential task in which the preceding state alone does not determine the next action. We compare the performance of neural control architectures in different environment settings and analyze the neural mechanisms and environment features exploited by the agents to achieve their goal. Simulation and experimental results using the Cyber Rodent robot show that a specific architecture outperformed the general recurrent controller.

Introduction

The performance of sequential actions requires the storage and update of internal states. In connectionist studies, supervised learning algorithms were applied to recurrent neural networks (Doya, 2002). Despite encouraging early results in training simple neural oscillators and state machines (Doya et al. 1989), it has turned out to be very difficult to train a network to process complex sequences with long-range interactions (Bengio, 1994) by fine tuning the architecture and parameters, though not impossible (Hochreiter, 1997). Reinforcement learning algorithms have also been applied to recurrent neural networks for sequential behaviors (Nakahara et al. 2000).

Evolution of recurrent networks has also been explored for sequential behaviors (Urzelai et al. 2001). For example, Urzelai et al. 2001 evolved neural controllers for a Khepera robot to travel back and forth between lighted and dark target areas, by utilizing the proximity, light and visual sensors.

Yamauchi et al. (1996) showed that continues time recurrent neural networks capable of sequential behavior and learning can be evolved to display reinforcement learning-like abilities. The task studied was generation and learning of short bit sequences based on reinforcement from the environment. Blynel further extended this work to an artificial agent task where a simulated Khepera robot has to navigate first in a simple and then a double T-Maze (Blynel 2003). Tuci et al. (2002) has demonstrated that it is possible to evolve an integrated dynamic neural network that successfully controls an agent engaged in a simple learning task where the robot learn the relationship between the position of a light source and the location of its goal. The networks have fixed synaptic weights and leaky integrator neurons.

In this work, we consider a more complex non-Markovian sequential task in which the memory of a preceding event alone does not determine the next action to be taken. Therefore, in addition to ordinary working memory of the most recent event, the agent must store and update a higher-order working memory. We compare the performance of a feedforward neural network (FFNN) controller and two recurrent neural networks, globally-recurrent neural network (GRNN), also known as Elman's network (Elman, 1990), and locally-recurrent neural network (LRNN), which has memory units with self-excitation and lateral inhibition. We used a real-valued GA to set the connection weights as well as the number of hidden and memory units.

The task is for a Cyber Rodent (CR) robot (Capi et al. 2002) to navigate between three places, marked by different colors, in a given order: food-nest-water-nest-food-nest-water-nest-... The results show that FFNNs performed fairly well when the locations of food, water, and nest were fixed throughout the course of evolution. Therefore, it did not need to rely on explicit internal memory. However, in the second environment, when the positions of the food and water were changed during agent life, agents with FFNN performed badly. GRNN also failed to capture the higher-order structure of the sequence; LRNN gave the best performance by successfully switching between two-levels of memory and by utilizing local recurrent connections. The output of memory units showed a strong relation with the agent intention. When the reward units were substituted by one touch sensor, the task was not completely solved. However, the LRNN neural controller performed better trying to focus its intention by memory units.

The neural controllers evolved in simulation were then tested in the real hardware of the CR robot. Despite their differences, especially with regard to timing, the robot performed the sequential navigation task well.

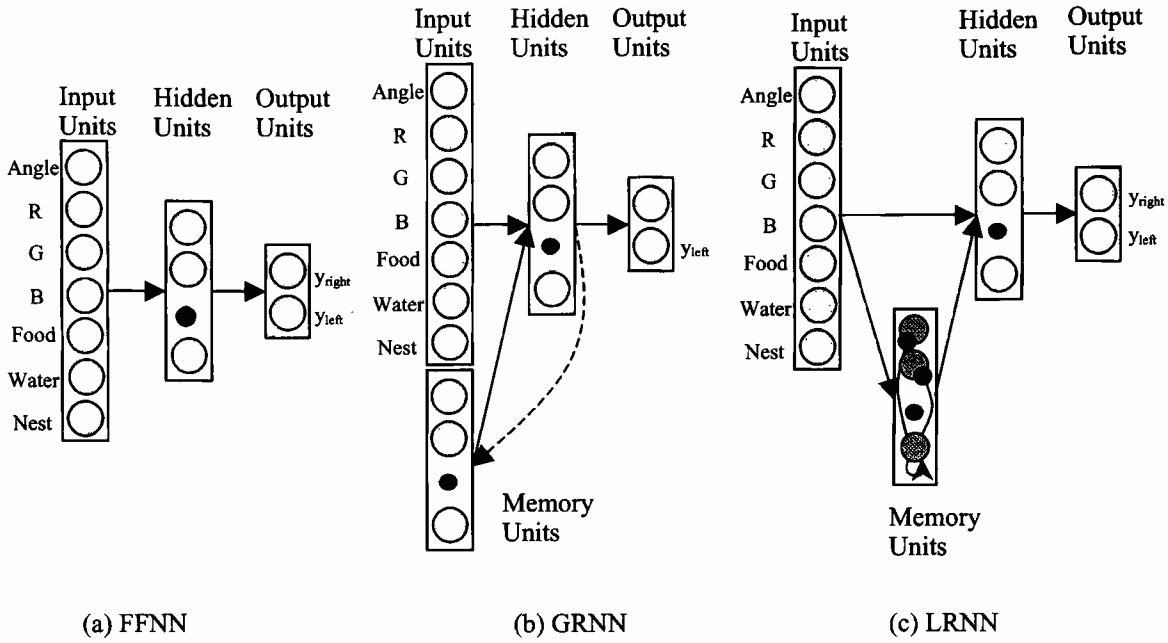

Figure 1. Neural architectures.

Evolving Dynamic Networks

Neural Architectures

The FFNN, GRNN and LRNN control architectures are shown in Figures 1(a), (b) and (c), respectively. The FFNN has one hidden layer. The GRNN architecture is similar to Elman's recurrent network, where the hidden units activation values are copied back and used as extra inputs in the next time step. The LRNN contains a memory layer where neurons with self-excitation and lateral inhibition receive inputs from the input layer. The neural architectures do not have bilateral symmetry.

The inputs of neural controllers are the angle to the nearest landmark (food, water or nest), three units (R, G, B) for color-coding of the nearest landmark, and three reward units (food, water or nest), each of which is activated only when the agent reaches the landmark according to the sequential order. The angle to the nearest landmark varies from 0 to 1 where 0 corresponds to 45° to the right and 1 to 45° to the left.

The hidden, memory and output units use sigmoid activation functions. The output of memory units of LRNN is 1 if the activation is larger then threshold and 0 otherwise. The output units directly control the right and left wheel angular velocities

$$\omega_{right} = \omega_{\max} * y_{right}, \quad (1)$$

$$\omega_{left} = \omega_{\max} * y_{left}, \quad (2)$$

where $\omega_{\max}$ is the maximum angular velocity and y_{right} and y_{left} are the neuron outputs. The maximum forward velocity of CR robot is considered to be 0.5 m/s.

Genetic Encoding

The genome of every individual of the population is designed to encode the structure of each neural controller. In addition to the weight connections, we also optimized the number of hidden and memory units varying between 1 and 5. The maximum genome length of the FFNN controller is 46. The first 35 genes encode synaptic weights of input-to-hidden connections, while the next 10 genes encode those of hidden-to-output connections, and the last gene encodes the number of hidden units. The respective genome lengths of GRNN and LRNN are 76 and 147. The GRNN genome includes weight connections (35 for input-to-hidden, 10 hidden-to-output and 25 memory units-to-hidden), initial values of memory units (5) and the number of hidden units (1). The LRNN genome encodes synaptic weights (35 for input-to-hidden, 10 hidden-to-output and 35 input-to-memory, 25 interconnected memory units, 25 memory-to-output), thresholds (5) and initial values of memory units (5). In addition, one gene for each memory unit determines whether the neuron has self and lateral connections. The last two genes encode the number of hidden and memory units. The range of input, self and lateral connection weights are from -10 to 10, 0 to 10 and -10 to 0, respectively.

Rather than using variable-length genotypes to allow varying numbers of hidden and memory neurons, we use fixed-length genotypes with the maximum number of hidden and memory nodes. When the number of hidden or memory units is smaller than the maximum, some parts of the genome are ignored in constructing a network.

Figure 2. Cyber Rodent robot.

Non-Markovian Foraging Task

Cyber Rodent Robot

In our experiments, we used the CR robot, which is a two-wheel-driven mobile robot as shown in Figure 2. The CR is 250 mm long and weights 1.7 kg. The CR is equipped with:

- Omni-directional C-MOS camera.
- IR range sensor.
- Seven IR proximity sensors.
- 3-axis acceleration sensor.
- 2-axis gyro sensor.
- Red, green and blue LED for visual signaling.
- Audio speaker and two microphones for acoustic communication.
- Infrared port to communicate with a nearby agent.
- Wireless LAN card and USB port to communicate with the host computer.

Five proximity sensors are positioned on the front of robot, one behind and one under the robot pointing downwards. The proximity sensor under the robot is used when the robot moves wheelie. The CR contains a Hitachi SH-4 CPU with 32 MB memory. The FPGA graphic processor is used for video capture and image processing at 30 Hz.

Food-Nest-Water-Nest task

The task for the CR robot is to alternately visit food and water locations after first visiting the nest. The agent must learn low-level reactive responses. In addition, the agent must have working memory to reach the specific landmark according to the sequential order. When the agent visits the nest, a higher memory level is needed to remember if the previous visited landmark was food or water.

The environment is a square of 5 m x 5 m. The food, water and nest are placed randomly in the environment, where the distance between them varies from 1.8 m to 2.3 m. We considered the performance of neural controllers in different environmental settings. In the first environment, food, water and nest are fixed during the agent's life and throughout generations. In the second setting, the food and water positions change after one third of the evaluation time and then change back after two thirds. In simulated environments, food, water and nest positions are considered squares of 0.05 x 0.05 m and their respective colors are red, blue and green, enabling the agent to distinguish them.

Evaluation and evolution

The CR robot is initially placed in a random position and orientation. Each individual of the population controls the agent during a lifetime of 100 seconds (50 ms x 2000 time steps). The agent receives a positive reward of 1 if the visited position matches the sequential order, and is penalized by -1 otherwise. The individuals are selected for crossover and mutation operations based on their performance.

A real-value GA was employed in conjunction with the selection, mutation and crossover operators. Many experiments comparing real-value and binary GA show that real-value GA generates superior results in terms of the solution quality and CPU time (Michalewich, 1994). The GA selection, mutation and crossover functions and their respective parameters are given in Appendix. Initially, 500 individuals were randomly created. The evolution terminated after 100 generations.

Table 1 Real-number GA functions and parameters.

Function Name	Parameters
Arithmetic Crossover	2
Heuristic Crossover	[2 3]
Simple Crossover	2
Uniform Mutation	4
Non-Uniform Mutation	[4 GNmax 3]
Multi-Non-Uniform Mutation	[6 GNmax 3]
Boundary Mutation	4
Normalized Geometric Selection	0.08

Results

Fixed environment

Figure 3 shows the performance of the agent controlled by FFNN. Under this environment setting, where the food nest and water are aligned almost linearly, the FFNN with three hidden neurons achieves fairy good performance. Due to the lack of memory, the agent reaches any landmark that enters the field of its visual sensor. After moving to the food and then to the nest positions, the agent reaches the water place that happens to be within its field of vision. When the visual sensor detects no landmarks, the agent rotates clockwise, due to strong connection weights

to the left wheel. After reaching the water position, the agent rotates clockwise and finds the food first. The agent then continues to rotate clockwise, and as soon as the nest becomes visible, it heads for this landmark. If the angle between the food and nest seen from the water location is larger than the agent's view angle, the FFNN-controlled agent heads to the food and cannot complete the task according to the sequential order.

The GRNN and LRNN also solve the sequential task in fixed environments. Figure 4 shows the behavior of the agent controlled by GRNN. The agent does not find irrelevant landmarks after reaching the food position nor after reaching the nest coming from the food location. In addition, the agent reaches the nest coming from the food location in such a way that the food location would be visible only for a very short time, and the agent starts moving toward the water location. The agent ignores the food after reaching the water location.

In a fixed environment, the easiest evolved strategy is to avoid seeing landmarks that do not match the sequential order. When avoidance is not possible, the agent ignores them.

Changing environment

In changing environment, the evolved FFNN and GRNN neural controllers contained four units in the hidden layer, while LRNN had three units in the hidden layer and four units in the memory layer. Three of the four units in the memory layer had lateral and self-connections, and one unit was not interconnected.

The population average of fitness value of the three neural controllers is shown in Figure 5. The fitness during the first generations is low because there are agents, which have no attraction to any landmark or only rotates passing through the same landmark. These agents are removed from the population during the course of evolution. The fitness of FFNN controller improved quickly during the first few generations and converged after 20 generations. The fast convergence of FFNN was due to the short length of the genome and it's evolved behavior. The average fitness of GRNN is higher than that of the FFNN controller. The LRNN fitness improves slower compared to other architectures, but achieved the best final performance. The GA already converges after the 80 generations.

The agent controlled by FFNN controller, due to its lack of memory, has the worst performance. The agent rotates in the environment and approaches whichever landmark that appears in its visual sensor. The trajectories of evolved GRNN and LRNN controllers are shown in Figure 6 (a) and (b), respectively. Three different environments with different food and water positions are shown. The GRNN controller performs better than FFNN. In the first and third environments, the agent successfully ignores the water and nest locations when it should reach the food. However, in the second environment, the agent reached the water before moving to the nest. Furthermore, after reaching the nest from the food moves again to the food place.

Figure 6b shows the trajectory of the agent controlled by LRNN. The agent follows the sequential order in three different environments. When the food and water positions change, the agent that was heading toward the food ignores the water location.

Figure 7 illustrates the unit activation values of LRNN. This figure shows how the agent utilized the angle and color of the nearest landmark and output of memory units in order to reach or avoid them. The activation of memory neurons shows that the first memory neuron is active when the agent has to visit the nest place. The second neuron is higher in the hierarchical level and deals with the higher-order structure of the sequence. The third and forth units specify whether to go to food and water, respectively.

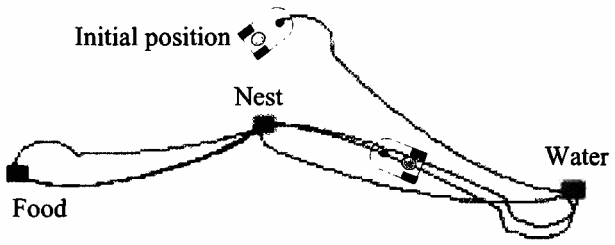

Figure 3. Performance of FFNN in fixed environment.

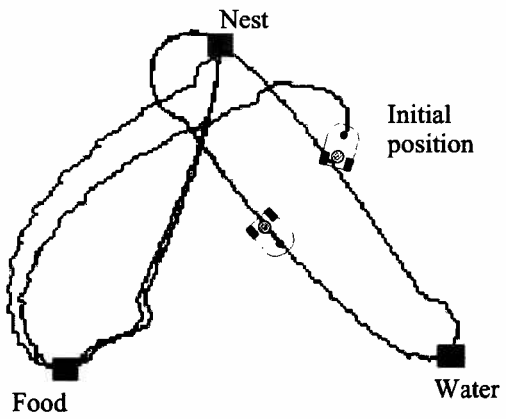

Figure 4. Performance of GRNN in fixed environment.

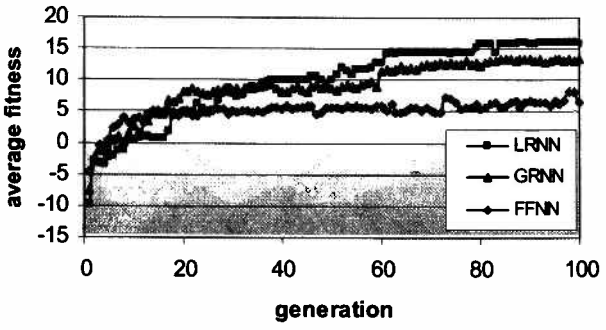

Figure 5. Average of fitness value.

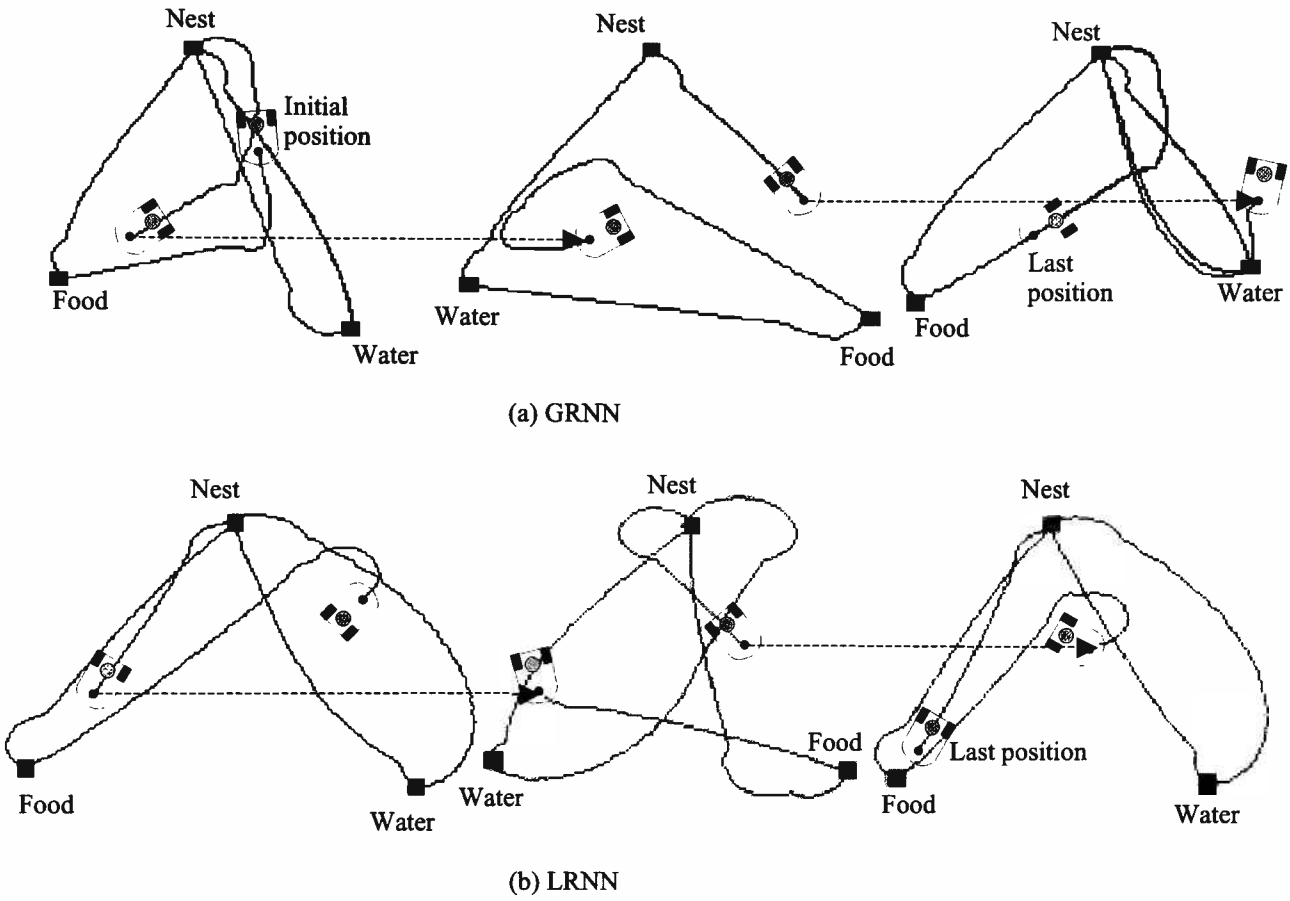

Figure 6. Performance of neural controllers.

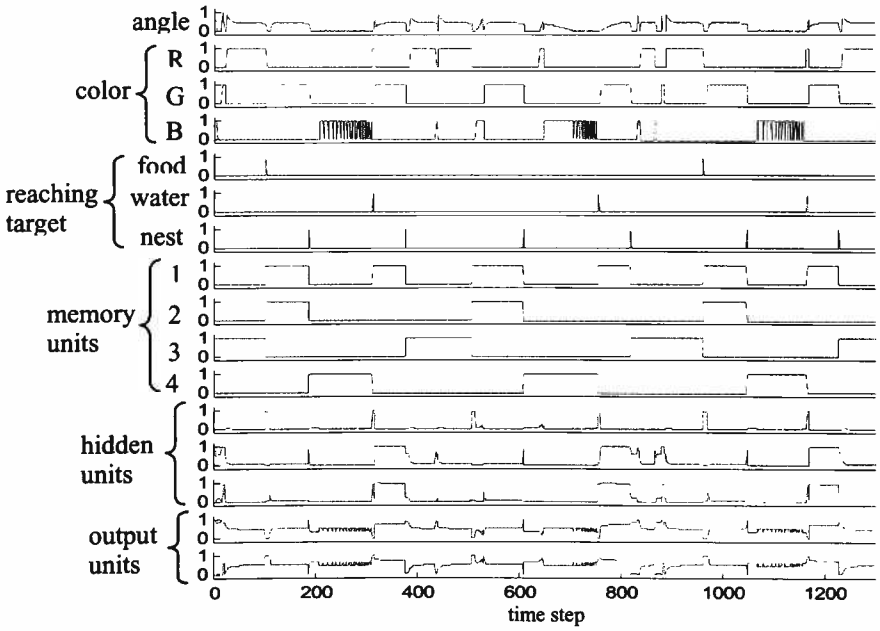

Figure 7. Neurons activations of LRNN.

Figure 8. Video capture of hardware implementation.

We implemented the evolved LRNN controller with reward units, on the real hardware of the CR robot (Figure 8), using the visual and proximity sensors. The visual and proximity sensors in front of the CR robot are utilized to activate the input units when reaching the food, water or nest location according to the sequential order. Despite some differences due to the task and CR hardware specifications, the neural controller performed well. In addition, because there were three different colors in the environment, the FPGA needed time to analyze the captured image. Therefore, the sampling time needed to obtain the sensor data and calculate the wheels velocity was 90ms. Despite these differences, the CR robot completed the sequential task well.

Conclusions

In this paper, we presented the performance of three evolved neural controllers for a sequential task, in which the agent had to obtain food and water after first visiting the nest. We evaluated the controllers' performance with simulation and hardware implementation using the CR robot. Based on these results, we conclude:

- The agent performance is strongly related to neural network size, structure and architecture.
- LRNN is more suitable for complex sequential tasks in dynamic environments, because the self-excitation and lateral inhibition of memory units has a great effect on the long term working memory and appropriate time switching among different agent intentions.
- The training environment influences the agent's evolved behavior.
- The evolved neural controller performed well when implemented onto the CR robot.

In the future, we plan to combine learning end evolution in order to deal with environmental changes.

References

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult, *IEEE Transactions on Neural Networks*, 5(2):157-166.

Blynel, J., (2003). Evolving reinforcement learning-like abilities for robots. In A. Tyrrell, P.C. Haddow, and J. Torresen: *Proc. of 5th Int. Conference on Evolvable Systems: From Biology to Hardware*, (pp. 320-331).

Capi, G., Uchibe, E., & Doya, K., (2002). Selection of neural Architecture and the environment Complexity. *Proceedings of International Symposium on Human and Artificial Intelligent Systems* (pp. 231-237), Fukui, Japan.

Doya, K., (2002). Recurrent networks learning algorithms, in: The Handbook of Brain Theory and Neural Networks.

Doya, K. & Yoshizawa, S., (1989). Adaptive neural oscillator using continuous-time backpropagation learning. *Neural Networks*, 2:375–386.

Elman, J., (1990). Finding structure in time, *Cognitive Science*, 14:179-211.

Hochreiter, S., (1997). Recurrent neural net learning and vanishing gradient. *Proceedings of Fuzzy-Neuro Workshop*, Soest, Germany.

Nakahara, H., Doya, K., & Hikosaka, O., (2001). Parallel cortico-basal ganglia mechanisms for acquisition and execution of visuomotor sequences-A computational approach. *Journal of Cognitive Neuroscience,* 13:626-647.

Michalewich, Z., (1994). Genetic algorithms + data structures = evaluation programs. Springer-Verlag.

Tuci, E., Quinn, M., & Harvey, I., (2002). An evolutionary ecological approach to the study learning behavior using a robot-based model. *Adaptive Behavior,* 10:201-221.

Urzelai, J., & Floreano, D., (2001). Evolution of adaptive synapses: robots with fast adaptive behavior in new environments. *Evolutionary Computation,* 9:495-524.

Yamauchi, B., & Beer, R., (1996). Sequential behavior and learning in evolved dynamical neural networks. *Adaptive Behavior,* 2(3):219-246.