Integrating Reactive, Sequential, and Learning Behavior Using Dynamical Neural Networks

Brian Yamauchi[1,3] and Randall Beer[1,2]

Department of Computer Engineering and Science[1]
Department of Biology[2]
Case Western Reserve University
Cleveland, OH 44106

Navy Center for Applied Research in Artificial Intelligence[3]
Naval Research Laboratory
Washington, DC 20375-5000

yamauchi@alpha.ces.cwru.edu
beer@alpha.ces.cwru.edu

Abstract

This paper explores the use of dynamical neural networks to control autonomous agents in tasks requiring reactive, sequential, and learning behavior. We use a genetic algorithm to evolve networks that can integrate these different types of behavior in a smooth, continuous manner. We apply this approach to three different task domains: landmark recognition using sonar on a real mobile robot, one-dimensional navigation using a simulated agent, and reinforcement-based sequence learning.

A novel feature of the learning aspects of our approach is that we assume neither an a priori discretization of states or time nor an a priori learning algorithm that explicitly modifies network parameters during learning. Instead, we expose dynamical neural networks to tasks that require learning and allow the genetic algorithm to evolve network dynamics that generates the desired behavior.

1. Introduction

Animals and robots need many different types of behavior to operate effectively in a dynamic environment. Consider, for example, an animal that needs to forage for food and return that food to its nest. As it explores its environment, it needs to react to any threats or obstacles that it encounters. When it finds food, it needs to remember the way back to its nest. Finally, it needs to learn the location of the food source and remember how to return, so that it can come back for more. So, in order to forage effectively, such an animal would require the capability for reactive and sequential behavior, as well as the ability to learn from experience. Similar capabilities would be required for a planetary rover collecting rock samples and returning those samples to its lander, or a mobile robot retrieving objects in an office environment.

Of the three types of behavior mentioned above, reactive behavior has been the primary focus of most autonomous agents research. The capability to react to the immediate situation *is* of central importance to an agent that must operate in an unpredictable world, but other capabilities are important as well. Agents also need the capability to perform sequences of activities, where each action may depend not only on the agent's current perceptions, but previous perceptions and actions as well. This broad category spans a wide range of behavior -- from stereotyped sequences where each action depends only on the previous action, to recognition tasks where an agent must make a decision based upon sequences of perceptions, to spatial navigation where an agent's motions depend not only on previous motions but also on landmarks and goals in the environment, to reinforcement learning where an agent's actions are shaped by the degree to which they elicit a reward. The common quality that these different behaviors share is their requirement that the agent be able to integrate perceptions over time into some form of internal memory, and that the agent be able to use this memory to modify its actions.

Most attempts to combine sequenced and learned behavior with reactive behavior have assumed a sharp division between the components that are reactive and

those that are in charge of sequencing or learning. In animals, however, such divisions are far less distinct. For example, the boundary between instinctual and learned behavior is a fuzzy one -- instincts can be modified by learning, and learning often takes place within a limited range of possible behaviors. This provides organisms with the ability to learn what they need to learn within their lifetimes, but also with the innate behaviors that have been evolved over previous generations -- and a continuum of intermediate behaviors that exhibit varying degrees of plasticity. We would like to give robots the same capability that animals possess to integrate reactive, sequential, and learned behavior in a manner that is smooth and continuous.

In animals, *all* behavior is evolved behavior -- including behavior that incorporates learning -- and this idea has motivated the research described in this paper. In this research, we explored the possibilities for using genetic algorithms to evolve dynamical neural networks that are capable of performing tasks that require both sequential behavior and learning.

A unique aspect of our approach is that the processes of sequencing and learning are internalized in the activation dynamics of the individual networks. No external algorithm is used to modify network weights during an agent's lifetime. Instead, perceptions integrated over time modify the internal state of the network, and these changes modify the dynamical behavior of the network and the agent that it controls.

Previous research in our group (Beer & Gallagher, 1992) has shown that dynamical neural networks can be evolved for tasks requiring reactive behavior and simple sequential motor control, but there are a number of open questions regarding the applicability of this approach to learning tasks. How general is this approach? Can neural networks use internal state to integrate perception? What types of sequential behavior can be generated? What types of learning?

In order to investigate these questions, we applied this approach to three different types of tasks: landmark recognition, one-dimensional navigation, and sequence learning. These tasks fall at very different points in the continuum of behavioral plasticity defined above, but they all share a requirement for agents that can use previous perceptions and actions to modify their behavior. In this paper, we describe these tasks, the methods used to evolve solutions, and the results obtained. Finally, we summarize with our conclusions about the strengths and weaknesses of this approach.

2. Methods

2.1 Dynamical Neural Networks (DNNs)

In the experiments described in this paper, we employed continuous-time recurrent neural networks whose behavior is governed by a system of differential equations of the following form:

$$\frac{dv_i}{dt} = \frac{1}{\tau_i}\left(-v_i + \sum_j w_{ij} f(v_j) + \sum_j s_{ij} I_j\right)$$

where v_i is the voltage of neuron i, τ_i is the time constant of unit i, w_{ij} is the weight of the connection from neuron j to neuron i, I_i is the input from sensor i, and s_{ij} is the weight of the connection from sensor j to neuron i.

The firing rate of neuron i is given by the sigmoid function $f(v_i) = (1 - e^{(t_i - v_i)})^{-1}$ where t_i is the threshold of neuron i.

2.2 The Genetic Algorithm

A real-valued genetic algorithm with a modular encoding scheme was used to search the parameter space of the above dynamical neural networks. Network parameters were encoded as a vector of real numbers, with the time constant, bias, sensor weights and incoming connection weights for each neuron represented as an indivisible unit for the purposes of crossover. This modular encoding is motivated by our belief that a neuron's intrinsic parameters and input weights form a useful building block that would not be preserved by a more conventional binary coding of parameters which allowed crossover to occur at arbitrary points.

The initial population was typically produced by randomly selecting a real value for each parameter of each network with a uniform probability distribution over the range of allowable values. The fitness of each individual in the population was then evaluated in simulation. An agent controlled by each network was run through a series of test trials, and the average score on these trials was used as the network's fitness.

Once fitness scores were assigned to all individuals in the population, a new population was generated. Individuals were selected for reproduction with a probability proportional to their fitness. If crossover did not occur, a single parent was randomly selected. If crossover did occur, two parents were randomly selected and a crossover point x was randomly chosen. The child then received the parameters for neurons 1 through x from its first parent and the parameters for the remaining neurons from its second parent.

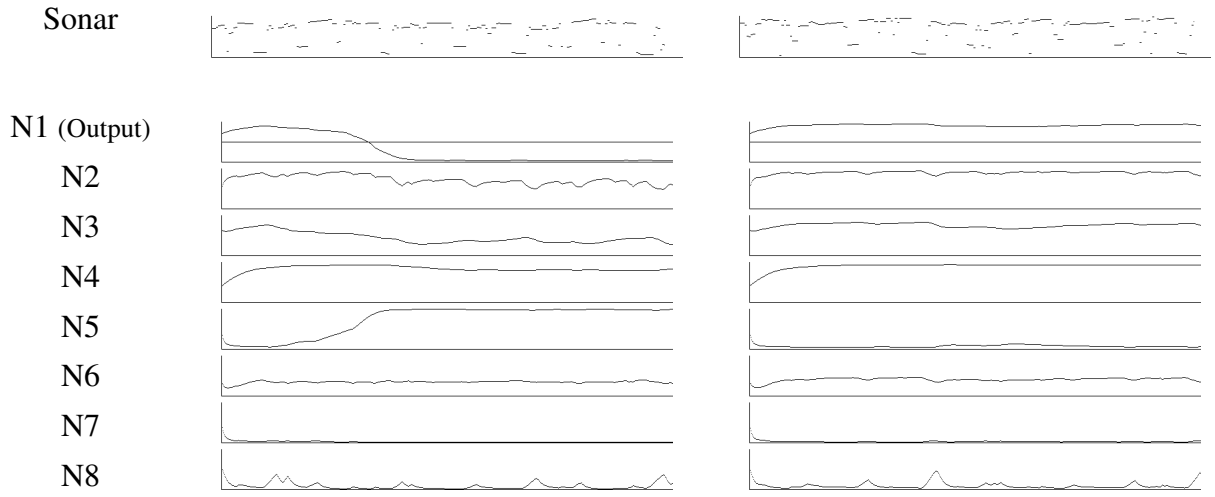

Figure 1: Behavior of landmark recognition network on (a) a landmark 1 trial (left), and (b) a landmark 2 trial (right). The top trace shows the sonar sensor input. The remaining traces show the neuron outputs. The first neuron is the classifier neuron.

Regardless of whether or not crossover occurred, there was a small probability (mutation rate) that any particular parameter of the child would be mutated. When mutation occurred, a new value was randomly chosen for the parameter to be mutated with a uniform probability distribution over the range of allowable values.

3. Landmark Recognition

3.1 Task

In reactive tasks, an agent's actions are determined by its immediate perceptions, but many other types of tasks require that an agent integrate those perceptions over time before it can decide what action to take. Consider the task of landmark recognition. The identity of the landmark may determine which way the agent should move. If the agent can identify a landmark based on perceptions received at a single moment in time, it can make a reactive decision about which way to move. However, if a single set of perceptions is not sufficient to identify the landmark, the agent will need to collect sensory information over time as, for example, it views the landmark from different directions, and eventually the agent needs to reach a decision based on the time-varying pattern of sensor data that it has received. For this reason, we chose the landmark recognition task to investigate whether dynamical neural networks can be evolved for tasks requiring the integration of perceptions over time.

For these experiments, we used a Nomad 200 mobile robot in an indoor environment. The task consisted of identifying one of two landmarks based on the time-varying sonar signals received as the robot circles the landmark. The focus of this research was on the ability of the neural network to perform the recognition task itself, rather than on the integrated control of the robot, so a simple behavior-based control system was used to allow the robot to find a wall and follow the wall counterclockwise around the perimeter of the room. The range signals received from the single sonar on the left side of the robot were used as the single input to the network. After integrating this information over a period of time, the robot was required to identify this landmark as one of two possible landmarks. Landmark 1 consisted of a long rectangle formed from two crates pushed together. Landmark 2 consisted of two squares formed from the same two crates separated by a distance of one crate width. Since a single sonar reading is not sufficient to differentiate between these two landmarks, the robot needed to use information in the temporal sequence of sonar returns in order to perform this task.

3.2 Methods

An eight-neuron dynamical neural network was evolved to solve this task. All of the neurons in this network received input from the left side sonar sensor. Neuron 1 was designated the output unit, and its firing rate $f(v_0)$ was treated as the network's output. After a fixed period of time (45 seconds of simulator time, corresponding to 135 seconds of time for the real robot), the output of neuron 1 was examined and used to classify the landmark. An output of less than 0.5 identified the landmark as landmark

1. An output of 0.5 or greater identified the landmark as landmark 2.

A population of 100 networks was evolved in simulation with a crossover rate of 0.5 and a mutation rate of 0.025. Six test trials were run for each individual in each generation -- three trials on landmark 1 and three trials on landmark 2. In each trial, the agent was started at a random position and orientation in the environment and allowed to run for a fixed length of time .

The score for each trial was computed as follows:

$$score = \begin{cases} 1 - f(v_0) & for\ landmark\ 0 \\ f(v_0) & for\ landmark\ 1 \end{cases}$$

The average of these test scores was used as each network's fitness value, and the genetic algorithm was applied to evolve populations that maximized this fitness function.

3.3 Results

After 15 generations, a network was evolved that was capable of recognizing these landmarks in simulation. This network was then transferred to the real mobile robot, where it was able to correctly classify landmarks in 17 out of 20 test trials. The network correctly identified landmark 1 in all 10 out of 10 trials, and it correctly identified landmark 2 in 7 out of 10 trials.

The top track in Figure 1a shows the sonar input that the agent received during a trial with landmark 1. Since the output of the sonar sensor is inversely proportional to the range reading, a high output indicates a nearby obstacle (the landmark) while a low output indicates a distant obstacle (the walls of the room). The remaining tracks in Figure 1a show the corresponding neuron outputs for this trial. The first of these tracks shows the firing rate for the output neuron, with the central horizontal line indicating a firing rate of 0.5. A neuron firing rate below this line indicates the agent is currently identifying the landmark as landmark 1 -- while a neuron firing rate above this line indicates that the agent is identifying the landmark as landmark 2. The identification made at the end of the trial is the one used for classification and scoring purposes. In this trial, the agent starts out by identifying the landmark as landmark 2, but after circling the landmark it changes its identification to landmark 1, and this remains unchanged for the duration of the trial. Figure 1b shows the sonar sensor output that the agent received during the trial with landmark 2 along with the corresponding neuron firing rates. Again the agent starts out by classifying the landmark as landmark 2, but in this case, the classification remains the same, even after the agent has circled the landmark.

We also investigated the application of dynamical neural networks to two other mobile robot tasks: predator avoidance and reactive navigation. For predator avoidance, DNNs were evolved to allow a robot to escape from moving pursuers while avoiding static obstacles. For reactive navigation, a hand-designed DNN was used in combination with algorithms for detecting static and cyclic behavior to allow a robot to navigate to a destination while avoiding obstacles. Further details about these tasks can be found in Yamauchi (1993).

4. One-Dimensional Navigation

4.1 Task

The results from the landmark recognition task showed that dynamical neural networks could be evolved to integrate sensory perceptions over time to arrive at a decision. The next question we explored was whether DNNs could integrate the ability to make decisions based on learning with the ability to control the motor behavior of an autonomous agent.

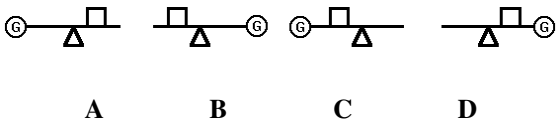

Figure 2: Environments for the navigation task. The triangle represents the agent, the circle represents the goal, and the rectangle represents the landmark. The landmark-far environments (A and B) are on the left, and the landmark-near environments (C and D) are on the right.

We chose the simplest possible continuous environment for our first experiments in this domain, a one-dimensional continuum along which the agent could move in either direction. Each environment contains a goal and a landmark. At the start of each trial, the agent is positioned in the center of the continuum, and the goal is positioned randomly at either the left end or the right end. In landmark-far environments (Figures 2A and 2B), the landmark is located on the opposite side of the agent from the goal. In landmark-near environments (Figures 2C and 2D), the landmark is located on the same side of the agent as the goal.

The agent possesses a landmark sensor and a goal sensor. Both of these provide only proximal information. The agent is treated as a point body and can only detect the landmark or the goal when it is directly in contact.

The agent's task is to learn, over a series of successive trials, whether the current environment is landmark-far or landmark-near, and then to reach the goal. Each trial lasts

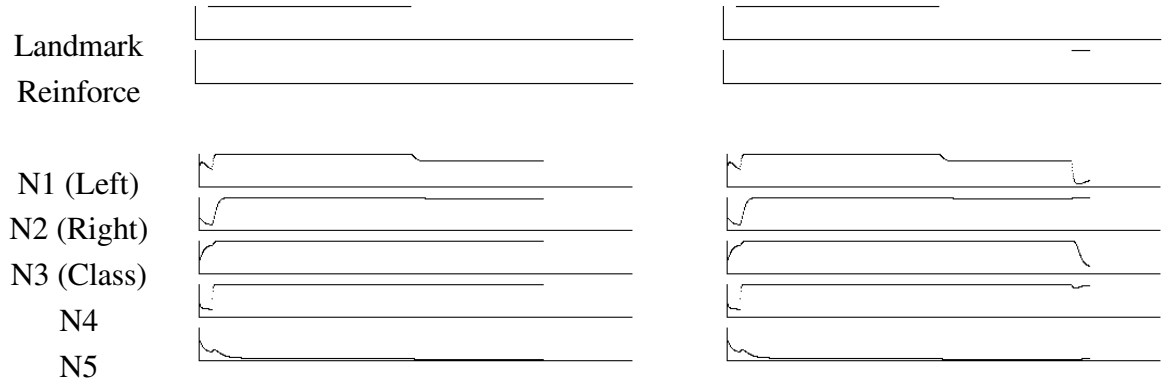

Figure 3: Behavior of the classifier network on (a) a landmark-near, goal-left trial (left) and (b) a landmark-far, goal-right trial (right). The upper two traces describe the landmark sensor input and the reinforcement sensor input. The remaining traces show the neuron outputs. The first two neurons are the left and right motor neurons. The third neuron is the classifier neuron.

until either the agent reaches the goal, the agent reaches the non-goal end of the continuum, or the time limit (1000 timesteps) is exceeded. If the agent reaches the goal, it receives reinforcement for 50 timesteps. If the agent does not reach the goal -- because it went to the wrong end or because it ran out of time -- then it does not receive any reinforcement, but the network is updated for 50 timesteps without any input.

In this task, as in many more complex tasks, the elements of reactive behavior, sequential behavior, and learning are all closely related rather than separated by sharp boundaries. Agents need to react to their perception of the landmark, but this action must be part of a sequence that will lead the robot to the goal. Since the goal location is randomly chosen, the agent cannot simply move left or right and expect to find the goal. In addition, the agent cannot search both the left and right ends of the continuum, because the trial ends as soon as the agent touches either end. As a result, the agent needs to learn the type of environment in which it is situated, based upon its perceptions of the landmark and the goal. Since the agent never perceives the landmark and the goal at the same time, it must integrate its perceptions over time in order to determine the position of the goal relative to the landmark. Once it has learned the environment type, it can then use the perception of the landmark in each individual trial to guide it toward the goal.

4.2 Methods

Attempts to evolve a single network capable of solving the entire task were unsuccessful, so a modular, incremental approach was used instead. The network architecture contains three separate subnetworks: one for goal-seeking behavior in landmark-far environments, one for goal-seeking behavior in landmark-near environments, and one for environment classification. Each of these subnetworks was evolved separately, then the three networks were assembled into an integrated system. The classifier network controls the agent during the first trial in each environment. At the end of the trial, the classification output determines which of the other two networks is used to control the agent for the remaining trials.

Each subnetwork consisted of a five-neuron, fully-interconnected DNN. All of these subnetworks have a sensor input for landmark detection and two motor outputs -- a left motor neuron and a right motor neuron. In addition, the classifier network has a sensor input for reinforcement, and a classification output used to determine the environment type. All of the neurons in all of the networks receive inputs from all of the available sensors.

The agent's velocity is equal to the difference in output between the left and right motor neurons: $dx/dt = f(v_R) - f(v_L)$, where x is the agent's current location, dx/dt is the agent's velocity, $f(v_R)$ is the output of the right motor neuron, and $f(v_L)$ is the output of the left motor neuron. The classification of the environment is landmark-far if the output of the classification neuron at the end of the trial is less than 0.5, and landmark-near otherwise.

Both of the goal-seeking networks were evolved using similar methods. A population of 1000 networks was evolved with a crossover rate of 0.5 and a mutation rate of 0.025. For each generation, each network was run for two trials -- one with the goal left, and one with the goal right. The landmark-far network was evolved exclusively in landmark-far environments, and the landmark-near network was evolved exclusively in landmark-near environments. The performance of a network on a given trial was 1 if the goal was reached and 0 if the goal was not

reached. The fitness of a network was equal to its average performance over both trials.

The classifier network was evolved to identify the environment type in a single test trial. In each generation, each network was run on four separate trials. These trials covered all of the possible combinations of environment type and goal location: landmark-far/goal-left, landmark-far/goal-right, landmark-near/goal-left, and landmark-near/goal-right. The performance of the network was equal to the error between the optimal classification (0 for landmark-far, 1 for landmark-near) and the actual classification output at the end of the trial:

$$P = \begin{cases} f(v_C) & for\ landmark-far \\ 1-f(v_C) & for\ landmark-near \end{cases}$$

where P is the network's performance and $f(v_C)$ is the network's classification output. The fitness of each classifier network was equal to its average performance over all four test trials.

4.3 Results

Networks were successfully evolved for all three of these tasks. One of the networks in the initial population for the landmark-far goal-seeking task was able to find the goal for both the goal-left and goal-right trials. Initially, this network moves left for a short period of time (not far enough to contact a left-side landmark), then it starts moving right. If the agent *does* encounter the landmark, then it reverses direction again, moving left until it reaches the goal (Figure 2A). If the agent does *not* encounter the landmark, then it continues moving right until it reaches the goal (Figure 2B).

After two generations, a network was evolved to solve the landmark-near goal-seeking task. This network starts moving right until it has moved far enough to encounter a right-side landmark, if one exists. If it does *not* encounter a right-side landmark, then it reverses direction and moves left until it reaches the goal (Figure 2C). If it *does* encounter the landmark, the it continues moving right until it reaches the goal (Figure 2D).

After five generations, a network was evolved that could correctly classify all of the trials in both of the environment types. Figure 3a shows the sensor and neuron traces for a landmark-far trial where the goal is located on the right (Figure 2B). Figure 3b shows the sensor and neuron traces for a landmark-near trial where the goal is located on the left (Figure 2C). In landmark-far environments, the landmark is located on the opposite side of the environment from the goal, and in landmark-near environments, the landmark is located on the same side of the environment as the goal -- so in both cases, the landmark will be located to the left of the agent.

The network starts with its classification output high, and its left motor output higher than its right motor output, so the agent moves left until it encounters the landmark. At this point, the landmark sensor becomes active, and the left and right motor outputs both increase to roughly equal levels, but the right output is actually slightly higher, so the agent moves right slowly. When the agent moves past the right edge of the landmark, the landmark sensor becomes inactive, the left output drops slightly, and the agent moves right at a somewhat faster rate of speed. In the trial shown in Figure 3a, the goal was located on the left so the agent never reached the goal, and its classification output remained high, identifying the environment as landmark-near. In the trial shown in Figure 3b, the goal was located on the right, so the agent was reinforced when it reached the goal, and its classification output went low, identifying the environment as landmark-far.

If the landmark had been on the right side of the agent instead, then the agent would not have reversed direction, and would have continued moving until it reached the left edge of the environment. In a landmark-near environment, a right landmark means that the goal is located on the right, so the agent will not be reinforced, and will continue to generate a high classification output, identifying the environment as landmark-far. In a landmark-far environment, a right landmark means that the goal is located on the left, so the agent *will* be reinforced, and will change its classification output to identify the environment as landmark-near.

Thus, the classifier network can correctly identify all four possible combinations of environment type and goal location. This network was then integrated with the two goal-seeking networks to create a system that could navigate to the goal in all four cases.

5. Sequence Learning

5.1 The Task

The results from the one-dimensional navigation task demonstrated that a dynamical neural network could control an autonomous agent and integrate perceptions over time to learn the identity of an environment, and that based on this identification, one of two other DNNs could be used to control the agent appropriately for that environment. The next question we addressed was whether this selection process could be internalized. Instead of using the output of one network to activate or deactivate others, could a single network switch between several different modes of behavior based on what it learned about its environment?

Consider a rat attempting to navigate a maze. As it traverses a maze, it is presented with a series of choice points at variable intervals of time. Each time it reaches an intersection, it must choose to go either left or right. While each intersection may be partially distinguished by

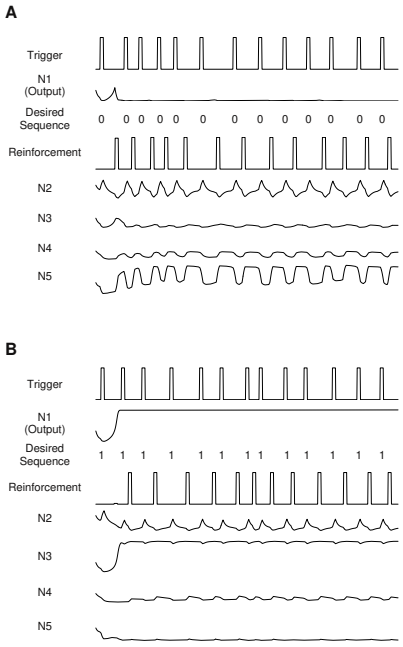

Figure 4: Behavior of the one-bit sequence learning network on (a) a 0* positive trial and (b) a 1* positive trial.

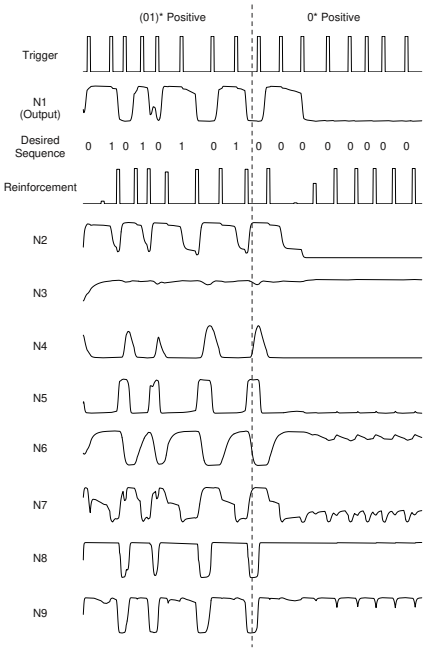

Figure 5: Behavior of the two-bit sequence learning network on a (01)* positive trial followed by a 0* positive trial.

perceptual cues, such as visual appearance or smell, in general perceptual information will be insufficient to uniquely determine the appropriate action. Instead, the rat must integrate these partial perceptual cues with its previous decisions in that run in order to determine the most appropriate action. Furthermore, in order to learn its way through new mazes, a rat must also be capable of learning new decision sequences. If we idealize the decision trigger from the environment as a binary input and the decision to be made as a binary choice, then the sequential learning problem becomes the problem of learning to generate a particular sequence of binary digits in response to a sequence of environmental triggers occurring at variable intervals of time.

In the sequence learning task, the network is required to generate one of a set B of possible sequences based on environmental reinforcement. The reinforcement of a given decision was delayed by a variable amount of time, but always occurred before the next environmental trigger. Thus, while the network could not rely upon receiving reinforcement at a fixed point in time, we did not address the general problem of delayed reinforcement. If a sequence repeats with a period n, then we will refer to it as an n-bit sequence learning task. For example, using regular expression notation, a maze for which the path to the goal consisted of alternating left and right turns would pose a two-bit sequence learning task of the form (01)* or (10)* depending upon whether 0 or 1 was the correct decision for the first intersection.

Fully interconnected dynamical neural networks with two inputs, one binary (the trigger) and one continuous (the reinforcement signal), and one continuous output (the decision) were employed. The decision trigger input I_1 normally had a value of 0. At variable intervals of time, however, the environment signaled that a decision must be made by setting I_1 to 1 for 10 time steps. The time interval between triggers was drawn from a uniform probability distribution over the range [10,40] time steps. By convention, neuron 1 always served as the output of the network. For the duration of a trigger, the output of neuron 1 was compared to the corresponding bit in the desired sequence.

Following a decision trigger, reinforcement was presented for a fixed duration of 10 time steps with a delay uniformly distributed over the range [10,40] time steps. For a given stimulus and response, the reinforcement is:

$$R = 1 - \frac{1}{end - start} \int_{start}^{end} |p - f(v_1)|$$

where R is reinforcement signal, p is the desired output, $f(v_1)$ is the firing rate of the output neuron, and start and end are the starting and ending times of the stimulus input The interval between the presentation of the reinforcement

signal for the networks response to one trigger and the next trigger was uniformly distributed over the range [10,40] time steps.

The overall performance of the network on a given trial is given by:

$$P = 1 - perf_{base} - \frac{1}{n}\sum_{i=1}^{n} R_i$$

where P is the performance value, $perf_{base}$ is the the base performance (usually the performance score expected from a random or constant output), n is the number of stimuli in the trial, and R_i is the ith reinforcement signal. Performance scores of less than zero were treated as zero.

The results of these experiments are summarized below. A more detailed treatment can be found in Yamauchi and Beer (1994).

5.2 One-Bit Sequence Learning

In our first learning experiment, we sought to evolve five-neuron networks that could learn to generate the set of one-bit sequences B = {0*, 1*} based on environmental reinforcement. The learning problem here is relatively simple -- the network must utilize the reinforcement signal in order to determine which constant output state to adopt. Two experiments were run on a population of 1000 networks for 500 generations, with a crossover rate of 0.3, a mutation rate of 0.01, and $perf_{base}$ set to 0.5. Both experiments consisted of four trials.

In the first experiment, networks were evaluated on two 0* positive and two 1* positive trials. The behavior of one network is shown in Figure 4. This network initially adopts an output state of 0. If this output is reinforced, then it remains in this state as long as it is reinforced (Figure 4a). If, however, this output is not reinforced, then the network adopts an output of 1 and then remains in this state permanently (Figure 4b). Using the techniques of dynamical systems theory, we were able to analyze the behavior of these networks in terms of fixed point and limit cycle attractors. This analysis is described in Yamauchi and Beer (1994).

5.3 Two-Bit Sequence Learning

In our second set of learning experiments, we sought to evolve dynamical neural networks that could learn to generate the set of sequences B = {0*, 1*, (01)*, (10)*} based on environmental reinforcement. Unfortunately, attempts to directly evolve such networks were unsuccessful. However, we were able to produce solutions to this problem using an incremental approach.

First, we evolved five-neuron networks that could learn to generate the more limited set of two-bit sequences {(01)*, (10)*}. We already have five-neuron networks that can learn to generate the sequences 0* and 1* from our previous experiments. Then, we seeded a population with ten-neuron networks formed by fully interconnecting {0*, 1*} one-bit sequence learning networks with {(01)*, (10)*} networks and evolved this population on the full two-bit sequence learning task, with the output neuron of the {0*, 1*} network serving as the output neuron for the composite network. Two such experiments were run on a population of 1000 networks for 500 generations, with a crossover rate of 0.3, a mutation rate of 0.002 and $perf_{base}$ set to the average performance of each generation.

The behavior of one solution to the full two-bit learning task is shown in Figure 5. Note that this particular network made use of only nine of its neurons. Figure 5 shows this network on a (01)* positive trial followed by a 0* positive trial. After receiving one low reinforcement following its output of near 1 during the first trigger, the network locks into the (01)* sequence. When the pattern of reinforcement changes from (01)* positive to 0* positive after eight triggers, it takes the network three trigger-reinforcement pairs before it successfully makes the transition to 0*. Part of this delay is due to the fact that the networks output of 0 on the ninth trigger is fortuitously reinforced because 0* and (01)* share the same next bit. It is not until the tenth trigger that a low reinforcement following an output of 1 signals that a switch in behavior is necessary. This network is also capable of learning the other two two-bit sequences, 1* and (10)*.

5.4 Three-Bit Sequence Learning

In our final set of learning experiments, we sought to evolve networks that could learn the set of three-bit sequences B = {(011)*, (101)*, (110)*} based on environmental reinforcement. All of the members of this particular set are related by a phase shift. A population of 1000 ten-neuron networks was evolved, and after 700 generations, with a crossover rate of 0.3. a mutation rate of 0.002, and $perf_{base}$ set to 0.666, a network was evolved that was capable of successfully learning all three sequences.

6. Related Work

Many researchers have used genetic algorithms to evolve autonomous agents, and a few researchers have evolved recurrent neural networks capable of reactive behavior. Jefferson and Collins evolved agents capable of following a fixed trail (Jefferson, et al. 1992) and performing reactive foraging (Collins & Jefferson, 1991). Beer and Gallagher (1992) evolved agents capable of chemotaxis and legged locomotion. Cliff, Harvey, and Husbands (1993) evolved a network capable of using vision to center a simulated robot in a cylindrical environment. What makes our work different from these research efforts is that we evolve

agents that can not only react to their environment, but also integrate information over time to modify their future behavior. Ackley and Littman (Ackley & Littman, 1992) combine evolutionary techniques with reinforcement learning by evolving an evaluation network that modifies the weights on an action network, based on the amount of reinforcement received. The primary differences between our work and theirs are that they use a separate learning algorithm (an extension of back-propagation) to modify weights during learning, and that both the world state and agent actions are discretized in their simulation.

Much research has been done in the area of reinforcement learning. Our work differs from traditional approaches (for example, Sutton, 1988; Kaelbling, 1993), in several fundamental ways. In traditional approaches, both world states and agent actions are assumed to be discrete in both space and time, while in our experiments, continuous perceptions and actions occurred in continuous space (for the navigation and landmark recognition tasks) and continuous time (for all tasks). In traditional approaches, a learning algorithm is used to adjust parameters that allow the agent to learn from experience. In our approach, the genetic algorithm is used to adjust parameters between generations, but no external algorithm is used to adjust parameters during an agent's lifetime. Instead, the internal dynamics of the neural network are used to integrate the perceptions of the agent, and to determine the agent's actions based upon these perceptions.

Some work has also been done in combining reinforcement learning with the techniques from behavior-based robotics. Maes and Brooks (1990) have used reinforcement learning to learn to coordinate the activity of hand-designed behaviors for legged locomotion. Mahadevan and Connell (1990) have used reinforcement learning to learn reactive behaviors in a system where the behavior decomposition and arbitration was designed by hand. Our work differs not only in the control mechanism (dynamical neural networks vs. finite-state behaviors), but more fundamentally, in the use of network dynamics rather than an external learning algorithm to allow the agent to learn from experience. Maes (1992) has also used reinforcement-based weight adjustment in a network of behaviors to learn behavior selection. Our work is similar in using internalized, distributed learning, but different in that our learning emerges directly from neuron activation dynamics, while Maes' algorithm is based upon statistics accumulated from the co-activation of behavior nodes. In addition, the individual behaviors in Maes' network are hand-designed, while the behaviors in our networks are evolved.

7. Conclusions

The principal aim of this research was to determine whether dynamical neural networks could provide an effective control mechanism for integrating reactive, sequential, and learning behavior in autonomous agents. In order to investigate this question, we attempted to evolve DNNs that could solve tasks involving landmark recognition, one-dimensional navigation, and sequence learning.

In the landmark recognition task, networks were evolved that could learn the identity of different landmarks based on time-varying sonar input on a real mobile robot. In the one-dimensional navigation task, networks were evolved that could learn the relationship between a landmark and the goal, then use this relationship to guide the agent to the goal. In the sequence learning task, networks were evolved that could learn to generate different sequences of output based on the reinforcement received.

Our primary conclusion, based on the results from these experiments, is that dynamical neural networks can provide a means of integrating reactivity, sequencing, and learning within a single control system.

In addition, we also conclude that genetic algorithms provide a means of generating these networks, at least for simple problems. The primary limitation that we encountered was one of scaling -- *not* in the DNNs themselves, but in the time required for the genetic algorithm search.

One way to deal with the scaling problem is to use a modular approach. In this approach, small networks are evolved to solve subtasks, and then these networks are assembled to solve the entire task. A good problem decomposition can minimize the amount of search that the genetic algorithm needs to perform, and in some cases (such as the landmark-far goal-seeking task) even random generation combined with a single round of selection can find solutions. The results from the two-bit sequence learning task and the one-dimensional navigation task indicate that this approach can be used to build networks that can perform tasks that would otherwise present scaling problems.

Acknowledgments

This work was supported in part by grant N00014-90-J-1545 from the Office of Naval Research. The robot experiments were performed at the Navy Center for Applied Research in Artificial Intelligence at the Naval Research Laboratory.

References

Ackley, David and Michael Littman (1992). Interactions between learning and evolution. In *Artificial Life II,* Chris Langton, et al., eds. Redwood City, CA: Addison-Wesley.

Beer, Randall and John Gallagher (1992). Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, Vol. 1, No. 1.

Cliff, Dave, Inman Harvey, and Phil Husbands (1993). Explorations in evolutionary robotics. *Adaptive Behavior*, Vol. 2, No. 1.

Collins, Robert and David Jefferson (1991). Representations for artificial organisms. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, Cambridge, MA: MIT Press.

Jefferson, David, Robert Collins, Claus Cooper, Michael Dyer, Margot Flowers, Richard Korf, Charles Taylor, and Alan Wang (1992). Evolution as a theme in artificial life: the Genesys/Tracker system. In *Artificial Life II,* Chris Langton, et al., eds. Redwood City, CA: Addison-Wesley.

Kaelbling, Leslie (1993). *Learning in Embedded Systems*. Cambridge, MA: MIT Press.

Mahadevan, Sridhar and Jonathan Connell (1990). Automatic programming of behavior-based robots using reinforcement learning. Research Report RC16359, IBM Research Division, Yorktown Heights, NY.

Maes, Pattie (1992). "Learning behavior networks from experience", *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, Francisco Varela and Paul Bourgine, eds., Cambridge, MA: MIT Press.

Maes, Pattie and Rodney Brooks (1990). Learning to coordinate behaviors. In *Proceedings of the Eighth National Conference on AI*, San Mateo, CA: Morgan Kaufmann.

Sutton, Richard (1988). Learning to predict by the method of temporal differences. *Machine Learning*, Vol. 7, pp. 227-252.

Yamauchi, Brian (1993). Dynamical neural networks for mobile robot control. NRL Memorandum Report AIC-033-93, Naval Research Laboratory, Washington, DC.

Yamauchi, Brian and Randall Beer (1994). Sequential behavior and learning in evolved dynamical neural networks. *Adaptive Behavior,* Vol. 2, No. 3.