1 Goal Directed Adaptive Behavior in Second-Order Neural Networks: Leaning and Evolving in the MAXSON Architecture

Frederick L. Crabbe and Michael G. Dyer

This chapter presents a neural network architecture (MAXSON) based on second-order connections that can learn a multiple goal approach/avoid task using reinforcement from the environment. The paper shows that MAXSON can learn certain spatial navigation tasks much faster than traditional Q-learning, as well as learn goal directed behavior, increasing the agent's chances of long-term survival. The chapter also shows that portions of the *learning rules* can be evolved using genetic algorithms.

1.1 Problem

Agents that reside in complex environments need to be able to meet their survival goals, often when several of these goals simultaneously have high priority. When the environment is dynamic, or when the agents cannot be preprogrammed to meet these goals, they must *learn* to meet them. This chapter presents MAXSON, a flexible neural architecture that: learns faster than traditional reinforcement learning approaches, generates and applies the reinforcement in a neurally plausible manner, and can balance the requirements of multiple simultaneous goals. The chapter also describes two genetic algorithm experiments: 1) evolving the value of constants in the learning rules of the algorithm, and 2) evolving portions of the learning rules themselves.

Environment and Task

In order to evaluate an agent control architecture, it needs both an environment and a task. The environment includes the agent's body (sensors, effectors, and any physiological mechanisms) as well as the external world. The task allows for a metric to measure the relative success of the control architecture.

In this chapter, an agent lives in a two-dimensional, continuous environment, where the objects in the environment are food, water, and poison. To survive, the agent needs to eat the food, drink the water, and avoid the poison. The agent receives goal input in the form of hunger, thirst, and pain[1]. Each goal input is a single value between 0 and 1. As an agent's store of food and water goes down, its hunger and thirst rises. When the agent eats a poison, its pain input rises.

1 The goal input corresponds to input coming from within the agent's body, similar to motivational units in [6].

To detect external objects, the agent has a primitive visual system for object detection. The agent can see in a 180 degree field in front of it. Vision is broken up into four input sensors for each type of object the agent can see. Since the world contains three types of external objects, the agent has twelve visual sensors. The agent is bilateral with two visual sensors on each side for each type of object. These sensors perceive the closest object of that type. For instance, the FDL (Food Distance Left) sensor measures the distance to the closest food on the agent's left side, while the FAL (Food Angle Left) sensor measures the angle of this food relative to the direction the agent is facing.

An agent's possible actions are: turn left or right smoothly (up to four degrees), and move forward up to six units, where one unit is one twelfth the agent's size. The agent automatically consumes any object it comes in contact with. Any or all of an agent's actions may be carried out in parallel. For example, if the agent turns right 3.2 degrees, turns left 1.8 degrees, and moves forward 4.6 units, the result is that the agent simultaneously moves forward 4.6 units and turns right 1.4 degrees.

The basic task for an agent is to eat when hungry, drink when thirsty, and always avoid eating the poison. An agent begins knowing nothing about how to approach any of these items, nor which items are suitable for satisfying which goals. The agent must learn to behave appropriately to satisfy these goals.

1.2 MAXSON Architecture

Our agents use an architecture, called MAXSON (figure 1.1) [8], that is made up of two sub-networks: a second-order *policy network*, and a first-order *value network*. The policy network is used to dynamically generate the agent's actions at each time step, while the value network is used to generate and apply reinforcement to the policy network. At each time step the following occurs: (1) the policy network determines the agent's action, (2) the action is performed in the environment, (3) the value network calculates reinforcement for the policy network, (4) the weights on the policy network are adjusted based on that reinforcement, and finally, (5) the weights of the value network are adjusted based on the external reinforcement.

In order to behave intelligently, an agent needs to be able to act based upon its goals, both immediate survival goals such as eating, as well other goals involved in more sophisticated behavior, such as nest construction [9]. This can result in an agent that has multiple possibly conflicting goals that fluctuate

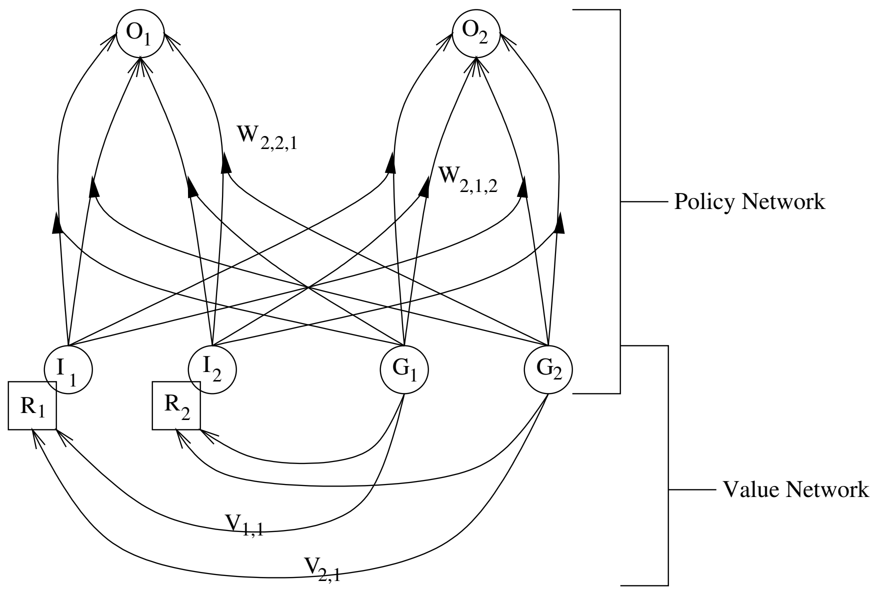

Figure 1.1
A small-scale MAXSON network. I-nodes receive external input. Associated with each I-node is a reinforcement node R. G-nodes receive internal goal input, and O-nodes perform motor actions. Each black triangle (joining two input connections) is a second-order connection with a weight W.

in urgency over time (which is the task environment a MAXSON agent is designed to learn in). The second-order connections in the policy network help the MAXSON agent satisfy these multiple simultaneous goals.

Reinforcement from the external world appears intermittently, upon the satisfaction or failure of some goal. This could require multiple satisfactions of a goal for an agent to learn correctly. The MAXSON agent should be able to learn from a single interaction with an object in the environment, so that an agent needs to eat a poison only once before it begins to avoid poisons. The value network serves this purpose by converting the intermittent reinforcement to reinforcement at each time step for the policy network.

Second-Order Networks

A MAXSON-based agent uses the second-order policy sub-network (figure 1.2) to choose what actions to take at each moment. Second-order connections multiply pairs of inputs to nodes before the inputs are summed [11, 18]. In

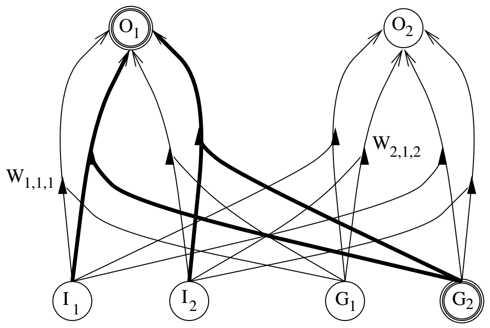

Figure 1.2
A second-order policy sub-network. G-nodes (goals) dynamically modulate the influence of I-nodes on O-nodes (actions) via second-order connections.

the policy network, the sensor nodes are broken into two groups: the external input and the internal goal sensors. The set of external input sensors I consist of all the visual sensors (for food, water, and poison objects); the goal sensors G consist of hunger, thirst, and pain. When the agent receives input ($I \cup G$), the associated sensor nodes are activated (activation between 0 and 1). Then, activation from each external input sensor I_i is multiplied with activation from each goal sensor G_g and a weight $W_{i,g,o}$ (weights between 0 and 1), via second-order connections, and then summed at an output node O_o, as shown in Equation 1.1:

$$O_o = \sum_{i \in I, g \in G} I_i G_g W_{i,g,o}. \tag{1.1}$$

If there were 2 external sensors (food_left and food_right), one internal sensor (hunger), one output (turn_left), and two weights:
($W_1 = W_{food\_left,hunger,turn\_left}$ and $W_2 = W_{food\_right,hunger,turn\_left}$)
the activation of turn_left would be:
$O_{turn\_left} = (I_{food\_left} \times G_{hunger} \times W_1 + I_{food\_right} \times G_{hunger} \times W_2)$.

Policy Network Weight Adjustment

The agent learns to approach food and water while learning to avoid poison by adjusting the second-order weights based on *immediate distributed reinforcement*. By immediate, we mean that reinforcement is given to the agent at each time step, rather than only when the agent satisfies some goal. By distributed, we mean that each external sensor generates a separate reinforcement signal, rather than having a single reinforcement signal for the whole organism. The reinforcement R_i at external input sensor node I_i is continuously re-calculated by a function of the difference between the activation on I_i at the current time step and the activation on I_i at the previous time step: $R_i = f(I_i^t - I_i^{t-1})$. We describe the neural mechanism that dynamically generates this distributed reinforcement signal in the next section. The second-order weights in the policy network are adjusted as shown in algorithm 1.

algorithm 1 : adjusts the second-order policy network weights.

W is the 3D weight matrix,
$\vec{O}$ is the network output vector,
$\vec{R}$ is the reinforcement vector,
$\vec{I}^t$ is the external input sensor vector at the current time step,
$\vec{I}^{t-1}$ is the external input sensor vector at the previous time step and,
$\vec{G}$ is the goal sensor input vector at the current time step.

$o \leftarrow \arg\max_o \vec{O}$
$g \leftarrow \arg\max_g \vec{G}$

for each i **in** I **do**:
$w_{i,g,o} \leftarrow w_{i,g,o} + R_i \times O_o \times G_g \times \delta_1$, where:

$$\delta_1 = \begin{cases} 1, & \text{if } |I_i^{t-1} - I_i^t| < \theta_1 \\ 0, & \text{otherwise.} \end{cases}$$

First, the *maximally* responding output node O_o and the *maximally* responding goal sensor node G_g are identified[2] (these are highlighted via dark circles in figure 1.2 as O_1 and G_2). Then the weight on every second-order link from G_g to O_o (dark lines in figure 1.2) is adjusted by the product of the activation of O_o, the activation of G_g, and the reinforcement R_i from the external input sensor node for that weight. Adjustment occurs only if the difference between the input on the external input sensor node I_i^t and the input at the pre-

2 The name MAXSON is an acronym for Max-based Second-Order Network.

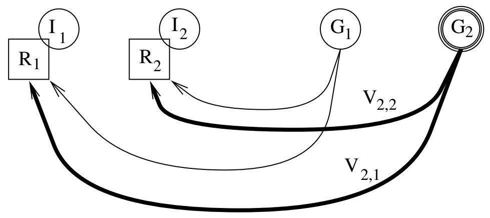

Figure 1.3
A first-order value sub-network. Each $\mathrm{R_i}$ node holds the amount of the reinforcement for the adjoining external input sensor node ($\mathrm{I_i}$).

vious time step on the external input sensor node I_i^{t-1} is less than a constant threshold θ_1.

Reinforcement Calculation by Value Network

In MAXSON, a first-order single layer network (called the *value network*) recalculates the reinforcement for the policy network at each time step. It consists of connections from the goal sensors to the reinforcement nodes associated with the external input sensors (figure 1.3).

The reinforcement is calculated as shown in algorithm 2. Activation on the *maximal* goal sensor G_g (dark circle G_2 in figure 1.3) is multiplied by the weights $V_{g,i}$ on the links from G_g (dark lines in figure 1.3) and propagated to the reinforcement nodes. Each external input sensor multiplies its input from the value network by the change in its activation over time ($I_i^t - I_i^{t-1}$) to generate the local reinforcement value (R_i).

algorithm 2 : calculates reinforcement at each external input sensor node.

V is the 2D weight matrix on the value-network.

$g \leftarrow \arg\max_g \vec{G}$

for each i **in** I **do**:
$R_i \leftarrow (I_i^t - I_i^{t-1}) \times V_{g,i} \times G_g$

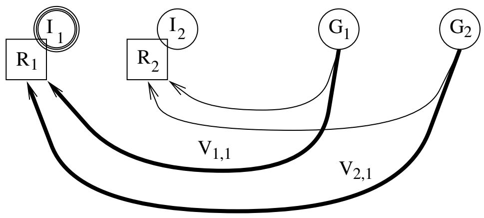

Figure 1.4
Update of weights V in value network.

Value Network Weight Adjustment

The value network uses Hebbian style learning [13] to updates it weights V based on the temporal difference at the *goal* sensor nodes (algorithm 3). When the change of activation at any goal sensor node G_g is greater than a constant threshold θ_2, then the weight $V_{g,i}$ on the link (dark lines in figure 1.4 from both goal nodes) from that goal sensor node to the external input sensor node (dark circle I_1 in figure 1.4) that was maximally responding at the previous time step I_i^{t-1}, is modified by the difference $(G_g^{t-1} - G_g^t)$ times the activation of the external input sensor node at the previous time step.

---

algorithm 3 : adjusts the first-order value network (FOV) weights

---

$i \leftarrow \arg\max_i \vec{I}^{t-1}$

for each g **in** G **do**:
$V_{g,i} \leftarrow V_{g,i} + (I_i^{t-1} \times \delta_2)$, where:

$$\delta_2 = \begin{cases} G_g^{t-1} - G_g^t, & \text{if } |G_g^{t-1} - G_g^t| > \theta_2 \\ 0, & \text{otherwise.} \end{cases}$$

Architectural Design Features

The unique features of the architecture described above were developed in response to requirements of an efficient neural mechanism for learning the multi-goal task. This section describes these requirements and how they were met.

On-the-fly Management of Multiple Goals.

The second-order connections (combining the goal and external input sensor input) give a MAXSON agent the ability to properly achieve *multiple* fluctuating goals. The second-order network enables changes in the goal input to radically affect the overall behavior. If a goal changes, a distinct portion of the policy network is given more (or less) control over the output. Using second-order learning (algorithm 1), different sub-networks are modified automatically, one for each goal sensor. When a goal sensor becomes active, the portion of the policy network that meets that goal becomes more active, and when that goal node becomes inactive, the portion of the network that meets that goal is turned off. For example, imagine an agent with moderate hunger, no thirst, food to the left and water to the right. The thirst input is 0, thus turning off the connections from the water sensors to the output nodes that would make the agent move toward the water. Thus the agent moves toward the food. But if the agent suddenly becomes extremely thirsty, the ‘approach water’ subportion of the policy network becomes very active, and, all other things being equal, the agent approaches the water.

Conversion of Delayed to Immediate Reinforcement.

The second-order policy network requires *immediate reinforcement* in order to learn. Unfortunately for the agent, reinforcement is only available when an object is eaten or drunk and this lasts for just a single time step. The value network solves this delayed reinforcement problem. The purpose of the value network is to convert short duration delayed reinforcement into the continuous immediate reinforcement used in the second-order learning algorithm. A weight ($V_{g,i}$) on the value network reflects the agent's positive or negative reinforcement attitude toward the external object represented by the external input sensor (I_i) when the goal sensor (G_g) is active. By associating the input that immediately led to a large positive reinforcement with that reinforcement, the agent is able to adjust the value of that object. Later, when the agent moves relative to the object, the value network can generate more appropriate immediate reinforcement. The cross-modal association of one input with another converts the delayed reinforcement into immediate reinforcement.

For example (figure 1.5), when the agent eats poison, the activation on the

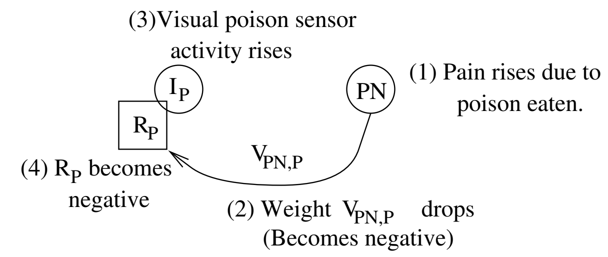

Figure 1.5
A value network example. PN is the pain input node, $\mathrm{I_P}$ is the visual node that receives input about a poison object. $\mathrm{R_P}$ is the reinforcement node for $\mathrm{I_P}$.

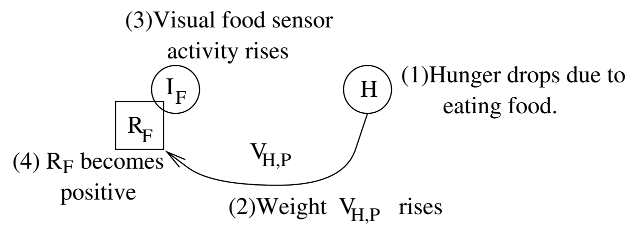

Figure 1.6
Another value network example. H is the hunger goal input node, $\mathrm{I_F}$ is the visual node that receives input about a food object. $\mathrm{R_F}$ is the reinforcement node for $\mathrm{I_F}$. The agent has no a priori knowledge that food satisfies hunger.

goal pain sensor rises (1). The change in activation on that sensor triggers the downward weight adjustment (2) between the pain sensor and the reinforcement node of the external input sensor that detects the poison. Later, when the agent moves toward a unit of poison, the activation on the poison visual sensor changes (3) from time $t-1$ to time t. That temporal difference combines with the value network ($G \times V$) to generate an immediate negative reinforcement (4) at that external input sensor node. The policy network can now use the resulting immediate vision-based negative reinforcement to learn how to avoid the poison. Thus an intermittent negative *gustatory* experience has caused the agent to experience negative reinforcement from more common *visual* experiences.

In the case when the agent eats food (figure 1.6), the activation on its goal hunger sensor goes down (1). The change in activation on that sensor triggers the upward weight adjustment (2) between the hunger sensor and the reinforcement node of the external input sensor that detects the food. Later, when the agent moves toward a unit of food, the activation on the food visual sensor changes over time (3). That temporal difference combines with the value network to dynamically generate an immediate positive reinforcement (4) at that external input sensor node. Thus the intermittent positive gustatory experience has caused the agent to experience a positive reinforcement from more common visual experiences. The policy network can use this immediate visual reinforcement to learn how to approach the food.

USING MAXIMUM NODES FOR FOCUS OF ATTENTION.

During second-order learning, attempting to adjust *all* the weights at each sense/act time step resulted in problems in assigning credit to individual weights. Here are two example problems: (1) When an agent outputs a large activation for 'turn_left', and a medium activation for 'turn_right', the agent's resulting action is to turn left. If the agent receives a positive reinforcement, and the algorithm were to adjust weights to *all* the output nodes, then the agent would adjust weights to both 'turn_right' and 'turn_left'. Because the weights are capped at 1 (to prevent unbounded growth), enough of these errors would cause the weights on both links to grow to 1. (2) Suppose that the agent is thirsty but even more hungry; it sees food to the left and turns left, causing the 'food left' sensor to receive positive reinforcement. If the weight between the 'food left' sensor node, the 'thirst' goal node, and the 'turn left' output node were to be adjusted based on the positive reinforcement, then the agent will learn to incorrectly approach food when thirsty. The credit for the positive reinforcement would be improperly given to that link.

Experimentation showed us that in sufficiently complicated environments, these errors occur often and are not compensated for, thus resulting in the agent learning bad policies. In order to focus the attention of the second-order learning algorithm on the appropriate weights, we introduce the idea of only adjusting the weights associated with the *maximum* nodes of a particular group. By only adjusting weights connected to the maximum output node, the credit assignment problem in problem 1 is alleviated. Since the 'turn left' node is the maximum output node, only the weight causing the agent to turn left is the one that is increased. Similarly, by only adjusting the weight connected to the maximum goal sensor, the agent doesn't learn to approach food when thirsty. Since, in this case, hunger is the maximum goal sensor, only the weights on the links emanating from hunger are modified, thus handling problem 2.

TEMPORAL DIFFERENCE BASED SENSOR AND GOAL THRESHOLD. Sometimes an atypical event occurs and it interferes with the agent learning how to behave in the typical case.

For example, if an agent that sees food on the left then turns to right, normally it should get a negative reinforcement. But, if while turning to the right a new piece of food happens to come into view then the agent would get a positive reinforcement. In this case, the agent might learn that when there is food on the left, turning to the right is a good idea. The same sort of situation can occur when an occluded object suddenly comes into view.

In order to filter out the effects of these discontinuous events during learning, we use the threshold θ_1 shown in algorithm 1. If the change of input at any sensor is greater than a threshold, no change is made to the weights on the links emanating of that sensor. So, if an object comes into view at some point, the large change in the sensor input causes that data to not be used for learning. Thus θ_1 acts as a filter to disregard non-smooth sensor fluctuations.

Because the reinforcement signal is generated by any large change of an agent's internal goal sensors, the value network must detect when such an event has occurred that should warrant a change in the agent's reinforcement attitude toward some object, and what goal this change is in relation to. To select the appropriate goal to associate with that external reinforcement, we choose a goal G_g whose change is greater than a threshold θ_2. The rationale for this threshold in algorithm 3 is that, when an goal sensor such as hunger changes gradually, it is a part of the normal process of hunger increasing over time, but when it drops suddenly, something special has occurred to satisfy the hunger. It is at these times that the agent should take note of its surroundings and perform cross-modal reinforcement learning.

Both these thresholds are a function of properties outside the MAXSON architecture: i.e. the sensors and physiology of the agent, as well as the physics of the world. We posit that such factors are naturally part of any situated system, and would be set in nature as a product of evolution. We initially found by our own experimentation that given the nature of the simulated environment, both thresholds worked best at 0.02.

1.3 MAXSON Experimental Method

To test our architecture, we ran three experiments (that are in increasing complexity). In the first experiment, the environment contained 10 food units and 10 water units. The task was to gather as much of the food and water as an agent could in 20,000 time steps. A single agent was placed in the environment, and that agent received external visual inputs, but the connections from the goal nodes in the policy network were lesioned, thus the MAXSON policy network became a first-order network. We lesioned these links because agents that were not hungry or thirsty would not eat and drink as fast as possible, and we wanted to measure the agents performance when always eating and drinking. We measured the agent's performance by counting the number of objects the agent ate and drank, giving a maximum score of 20. In the second experiment, the environment contained 10 food units, 10 water units, and 10 poison units, with the goal input still lesioned. The task was still to eat as much food and drink as much water as the agent could, but also to eat as little poison as possible. We measured an agent's performance by subtracting the number of poisons eaten from the total number of food and water eaten. In the third experiment, the environment was the same as in the second experiment, but the links from the goal nodes in the policy network were left intact. In this experiment, an agent should no longer eat and drink as much as it could, but only when it is hungry and thirsty, respectively. The agent should still refrain from eating poison. Each agent was given a maximum food and water storage capacity, and any amount eaten beyond this capacity was lost. We measured an agent's performance by measuring how long it survived in the environment. The maximum time an agent could survive was 60,000 time steps because of the limited amount of food and water.

In each experiment, we compared the MAXSON agent with three other agents: random, table-based Q-learner, and linear function Q-learner. The random agent selected its action from amongst 'turn left', 'move forward', and 'turn right', with equal probability. Q-learning was selected because it is a popular reinforcement learning technique for agents in similar environments. In the Q-learning models, reinforcement of +1 was given for eating food, -1 for eating poison and 0 otherwise.

The table-based Q-learner is based on the Q(λ) learner presented in [17]. The continuous input was discretized into a $3 \times m$ table, where m is the number of inputs (12 or 15, depending on experiment) and there are 3 possible discretizations (table 1.1).

The linear function Q-learner uses a linear function of the inputs as an approximation of the Q-function, as suggested in Russell & Norvig, p. 615 [15]. There was a separate function for each action a_i, that is, a separate equation to calculate the Q-value for each possible action, such as 'turn left'.

Table 1.1
An example discretization to form the table for the table-based Q-learner. P = poison, F= food, A = angle, L = left, R = right, H = hunger, T = thirst, and PN = pain.

	between 0.0 and .33	between 0.33 and .66	between 0.66 and 1.0
FAL	1	0	0
FAR	0	0	0
FDL	0	1	0
FDR	0	0	0
WAL	0	0	0
WAR	0	0	0
WDL	0	0	0
WDR	0	0	0
PAL	0	0	0
PAR	0	1	0
PDL	0	0	0
PDR	0	0	1
T	1	0	0
H	0	0	1
PN	1	0	0

The equation is shown in Equation 1.2.

$$2Q(s, a_i) = w_1 E_1 + w_2 E_2 + ... + w_{15} I_3. \tag{1.2}$$

The weights were adjusted using gradient descent, as described in Sutton & Barto [17].

The MAXSON agent consisted of the policy and value networks as described in the architecture section above. In the third experiment we ran both a MAXSON agent with the links from its goal nodes in tact, and one with those links lesioned.

For each type of agent, we ran 10 training sessions and report the average of the 10. In each training session, the agent was trained over 180,000 time steps. Periodically, learning was halted and the agent was tested for 20,000 time steps in 10 environments with different starting configurations. One of the random starting configurations is shown in figure 1.7.

We hypothesized that in the first experiment: (a) the table-based Q-learner would not perform well because its simple method of generalizing the input would not adapt to the continuous environment; (b) the linear function Q-learner would learn better; and (c) the MAXSON agent would learn as well as the linear function Q-learner, but do so faster as a result of the conversion of intermittent gustatory reinforcement to continuous visual reinforcement.

We hypothesized that in the second experiment: (a) the table-based Q-learner would continue to do poorly; (b) the linear function Q-learner would

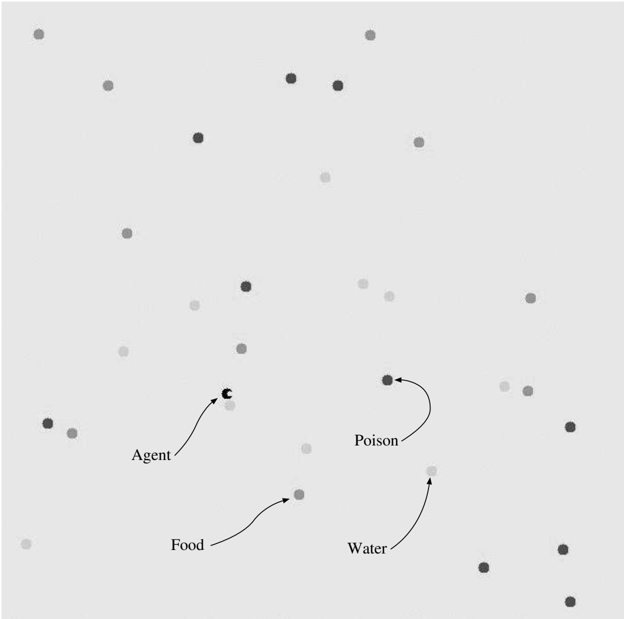

Figure 1.7
One of the random initial positions.

learn the function, but more slowly; and (c) the MAXSON agent would learn at roughly the same speed as before, since learning about the poison happens in parallel with learning about the food and water.

We hypothesized that in the third experiment the MAXSON agent with goal input would out-live all of the other agents because it could conserve food and water while avoiding the poison.

1.4 MAXSON Results

The results of the first experiment (food and water only) are shown in figure 1.8. The MAXSON agents performed the best, converging on a score of nearly 20 by 10,000 time steps. Once a MAXSON agent interacts with all of the different types of objects, it quickly (within a few thousand time steps) learns to consume almost all the food and water. The linear function Q-learner converges in 60,000 time steps to a score around 18, slightly less than the MAXSON

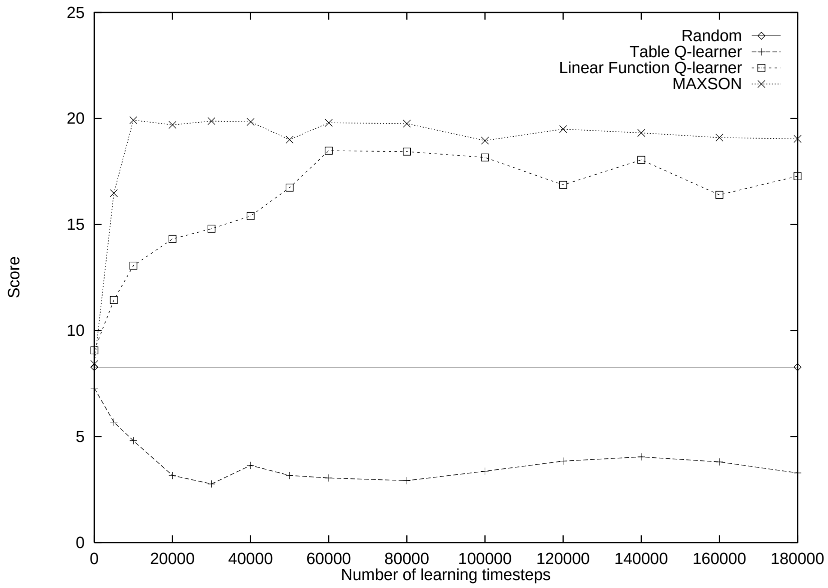

Figure 1.8
Experiment 1. The x-axis is the number of time steps and the y-axis is the score. The MAXSON agents converge six times faster than the linear function Q-learner and obtained a higher score (99.92% significance on a U-test).

agent. The difference between the maximum scores of the linear function Q-learner and the MAXSON agent is statistically significant at 99.92% using an Mann-Whitney U-test. The table-based Q-learner fails to learn any good policy within the experiment time frame. This is most likely because of its inability to generalize the input.

Figure 1.9 shows the results for the second experiment (food, water and poison). Again, the MAXSON agents had the best performance, converging to a score of 18 in 20,000 time steps. Much of the extra time taken to learn this task comes from the time it takes the agent to randomly consume all three types of objects. The linear function Q-learner starts well, but peaks at 60,000 time steps with a score of 15.29, and surprisingly begins to drop. The agents begin to move faster toward food and water, but also eat more poison. They eventually converge on a policy that consumes nearly everything in the environment, resulting in a score close to 10. The table-based Q-learner again fails to learn a viable policy in the time given.

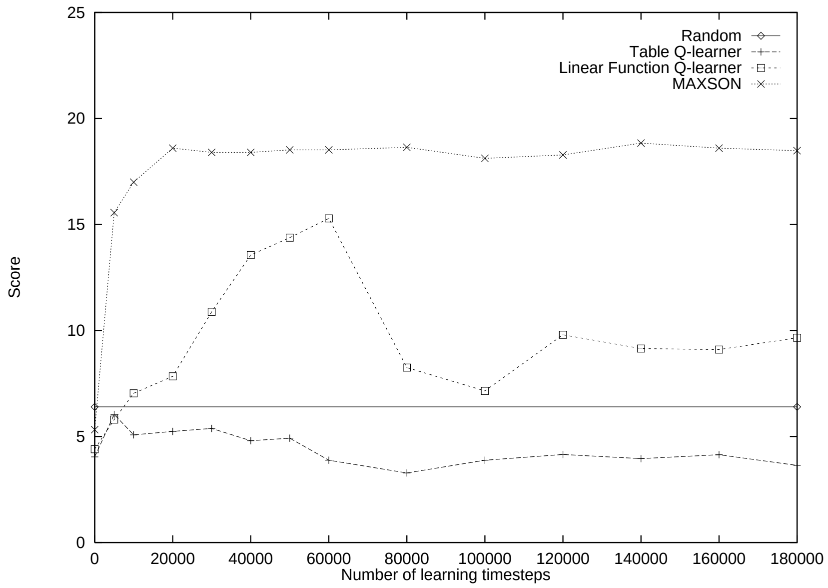

Figure 1.9
Experiment 2. The MAXSON agents converge to a good score (18.4) in 20,000 time steps. The linear function Q-learner reaches 15.29, but eventually converges on a score of 10.

Figure 1.10 shows the results for the third experiment (conserve food and water while avoiding poison). The MAXSON agents that take advantage of their goal input have the longest survival time. They are able to approach food and water, but only do so when hungry or thirsty, saving resources for later on. As expected, the MAXSON agent that ignores its goal input does not survive as well. It can eat and drink while avoiding poison, but it consumes all of its resources quickly, and dies by the 15,000th time step. The linear function Q-learner initially learns to perform as well as the MAXSON agent that ignores goal input. It does not improve beyond this because the effect of the goal inputs is small compared to the rest of the Q-function. Then, as it begins to eat more poison, the linear function Q-learning agents poison themselves to death in 10,000 time steps. The table-based Q-learner performs comparatively better in this experiment. While it does not have a policy any different from the previous experiments, its 'strategy' of consuming very little food water or poison performs about as well as a strategy that involves eating lots of poison.

1.5 Learning Discussion

Comparison to Q-learning

Q-learning was developed as a very general method for learning arbitrary functions. Often, general methods do not work as well in a particular domain as methods designed for that domain. Q-learning has been shown to have particular difficulties when the input to the agent is from the agent's point of view, rather than a global location [16]. MAXSON was designed for mobile agents that balance multiple goals in a continuous environment. Because of this, MAXSON has much better performance in our experiments. Not only do the MAXSON agents surpass the Q-learning agents in performance, they also reach the Q-learners' maximum performance an order of magnitude faster.

MAXSON combines six features to give it good performance compared to Q-learning. First, it implicitly takes advantage of the *continuous spatial nature* of this mobile robotic task. The reinforcement at each sensor node generated by the change in input over two time steps is valid because motion in the continuous environment changes the input continuously. This would not be true if the input were an arbitrary function, where moving toward an object could cause the input at a distance sensor node to go down or fluctuate. Second, a MAXSON agent uses credit assignment in the form of *distributed reinforcement* across all the sensors. By distributing the reinforcement, it becomes easier to pick out which part of the network affects the actions with respect to the external objects in view. Third, the weights associated with only the *maximum* nodes are adjusted. By adjusting only the weights associated with the maximum nodes, the system can concentrate on learning one thing at a time and reduce miss-placed credit. By reducing the errors in credit assignment, the network learns more rapidly and avoids learning incorrect associations. Fourth, the *cross-modal association* that occurs in the value network converts a single gustatory reinforcement event, such as eating a unit of food, into immediate visual reinforcement that is active over a period of time. With Q-learning, learning takes place only when the food is eaten, while in MAXSON, after the food is eaten, the agent learns about approaching food during each time step that it is observing food at a distance.

Fifth, the input is separated into goal sensors and external input sensors. By separating out the goal sensors from the external input sensors, the system can treat then differently during learning. Sixth, the second-order connections allow the goals of the agent to have an important major influence over its

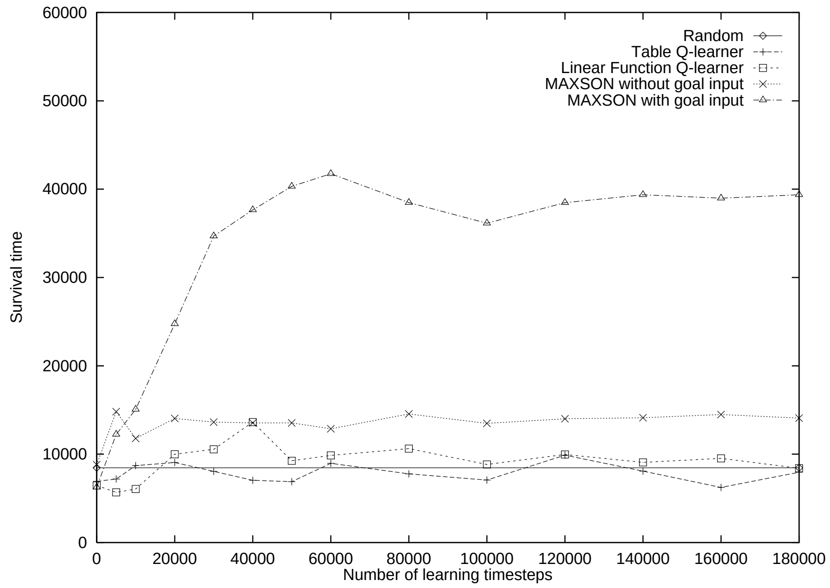

Figure 1.10
Experiment 3. The MAXSON agents that use their goal inputs survive the longest. Here, the y-axis is the survival time in number of time steps. Each data point indicates when an agent died (averaged over 10 trials).

behavior. An agent does not waste its resources by consuming them when it doesn't need them. This last feature could be incorporated into Q-learning. A function based Q-learner could multiply the goal and external input together as in Equation 1.3.

$$Q(s,a) = w_{1,1}E_1I_1 + w_{1,2}E_1I_2 + w_{1,3}E_1I_3 + ... + w_{12,3}E_{12}I_3. \qquad (1.3)$$

It would, however, be more difficult to incorporate the other features without fundamentally altering the general nature of Q-learning.

How MAXSON Networks Self-Organize

Figures 1.11,1.12,1.13 illustrate how a typical MAXSON network self-organizes as a result of the multi-goal approach/avoid task. The agent starts out randomly connected and with no knowledge of objects in its environment. The only innate structure (besides the architecture itself) is that eating

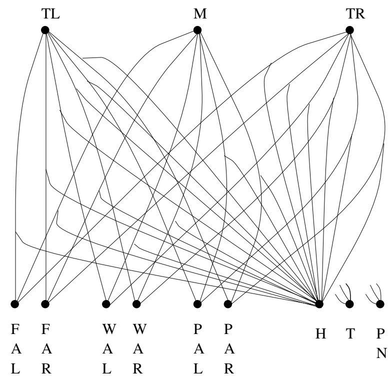

Figure 1.11
Initial policy network architecture with full connectivity and random weights in range of $[0, 0.01]$. TL = turn-left, TR = turn-right, PN = pain sensed, H/T = hunger/thirst sensed, FAL = angle to food object sighted on left, PAR = angle to poison object sighted on right, and so on. The distance visual sensor nodes were left out for clarity. Only connections from the hunger goal input node are shown.

poison will cause the pain input to increase while eating food/water causes hunger/thirst to diminish. Over time, the MAXSON agent learns the following through interaction with its environment:

1. It learns which sensory input should control which action output. For instance, it learns that the sensor for food on the left side should drive the 'turn left' output while poison on the left should drive the 'turn right' output.
2. It learns which goals should modulate which external sensor-to-action connections, via the second-order connections. For instance, it learns that the sensation of Thirst (T) should control connections between the water-sensors and the motor outputs while the sensation of Pain (PN) should control the connections between the poison-sensors and the motor outputs.
3. Through adjustments in V weights (e.g., due to experiencing pain when eating poison) it learns to become negatively reinforced merely by the *sight* of poison at a distance and positively reinforced by the sight of food/water at a distance.

Figures 1.12 and 1.13b show only the strongest connections. All weights tend toward extremes: in the policy network, weights are either close to 0, or close to 1. In the value network, the weights are close to 1, -1, or 0, but not in-between. Thus, a structured connectionist network self-organizes out of a

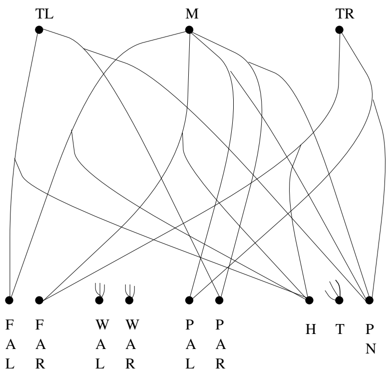

Figure 1.12
Simplified structure of a high-performance policy network after learning. All links from both hunger and pain with a weight greater than 0.01 are shown to highlight the learned structure. All links shown have weights greater than 0.5. Connections from thirst are omitted for clarity.

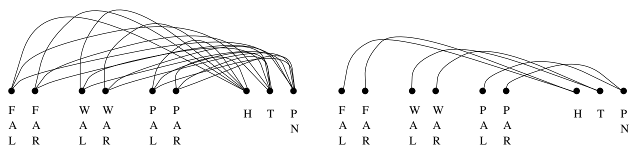

Figure 1.13
Value network before and after learning. Initially it has full connections, but it learns which sensors are more likely to be related to the achievement of appropriate goals.

distributed fully connected network. The pruning of excessive connections is common during neural development [14, 10, 7].

1.6 Evolving the Learning Rules

The learning algorithms in MAXSON depend on two threshold parameters (θ_1 and θ_2) and their relation to the difference over time in the agent's input. The purpose of these parameters was to exclude certain perceptory inputs from the learning process. We calculated what we felt were good values that correctly excluded all of the perceptually misleading events, while minimizing the number of excluded events. For example, θ_2 was needed to discriminate between goal meeting events such as eating food, with normal physiological changes, such as hunger rising. The previous sections showed that the learning rules worked with the thresholds set at 0.02. In this section, we discuss Genetic Algorithm (GA) experiments which first evolve the values of the thresholds and then evolve the threshold portion of the rules themselves in order to build them automatically.

In the first experiment we hoped to generate θ_1 and θ_2 with values similar to what we had set by hand. In doing so, the traditional GA model of individually testing each agent to determine if it should be bred in the next generation would take prohibitively long. This is because the agents exist in a continuous environment where actions need to be recalculated more frequently than in a grid-like environment and because, to properly evaluate an agent, the agent must be allowed time to both learn about the environment as well as interact with it to demonstrate the agents's learning. The combination of these factors results in simulations that would run for several months on our hardware. To alleviate this, we hoped to run an entire population together, without explicit evaluation of the agents. At each time step, each agent had a small probability of replicating, resulting in the best agents having more offspring over time.

We ran 50 agents in a 2160 by 2160 environment, with 100 units each of food, water, and poison. When an agent consumed one of the environment objects, it was replaced by one of the same type at a random location. If an agent died, it was replaced by an offspring of a an agent randomly selected from the 15 oldest agents. At each time step, an agent would produce an offspring with probability $p(o) = a \times 10^{-7}$ when the agent's age a is $< 10^4$, and probability 10^{-3} then a is $> 10^4$. The agent genome was two 7-bit numbers, representing the two θs. The θs were calculated from the genome by subtracting 12 and dividing by 100, resulting in a range of -0.12 to 1.11. All values less than zero were treated as zero and all values greater than one were treated as one. This mechanism allowed the use of a binary number genome, while not biasing the thresholds to larger values. The mutation rate ranged from 0.01 to 0.001.

Figure 1.14 shows the average value of θ_2 in the population varying over time in a typical run. Despite varying the density of objects in the environment, as well as the reproduction probabilities, and mutation rates, neither threshold ever converged to useful values. When we examined the longest lived agents in these experiments, we discovered that the majority were performing random walk. The density of the food and water allowed the random walk agents to survive just as well as learning agents, resulting in no selective pressure to evolve the agents. But when the environment was made more challenging

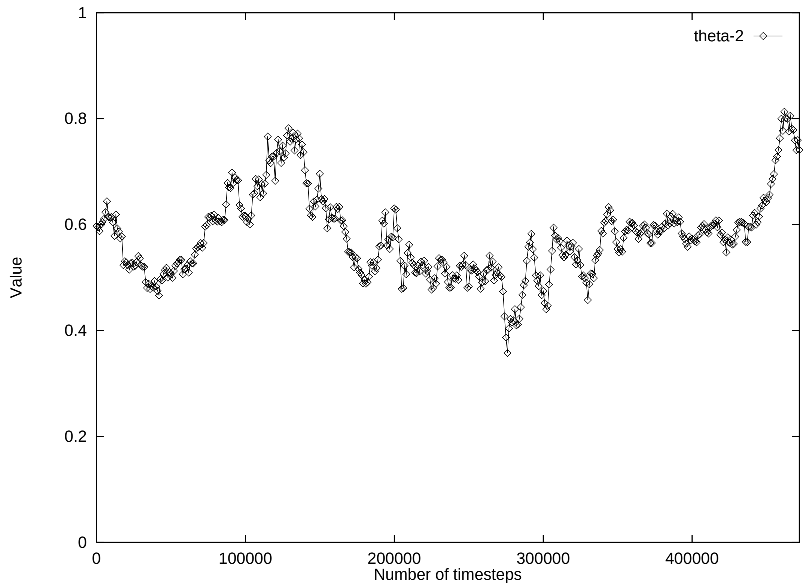

Figure 1.14
Plot of the population average value of θ_2 over time in the first GA experiment.

by decreasing the density of food and water, while increasing the density of poison, the environment was too difficult for new born agents who have yet to learn how to survive. Despite varying the densities of objects, we were unable to develop a suitable compromise. We felt a better solution was to *increase* the difficulty of the environment as the agents age to increase selectional pressure on agents as they have had a chance to learn.

In order to increase the selectional pressure, we reduced the density of food and water as the agents age. This requires that the simulation synchronize generations so that all members of the next generation begin at the same time. Then by not replacing food and water as it is consumed, the difficulty of the environment automatically increases. After all but the last 15 agents had died, those top 15 agents who lived the longest were used to populate the next generation, using the same parameters as above. Figure 1.15 plots θ_1 and θ_2 in each generation in one of five runs. θ_2 drops quickly to 0.2 and eventually converges to 0.02. Once θ_2 had a reasonable value, θ_1 begins to drop as well, because the agent needs to get reasonable reinforcement from the value network before it can begin to evolve the policy network. θ_1 does

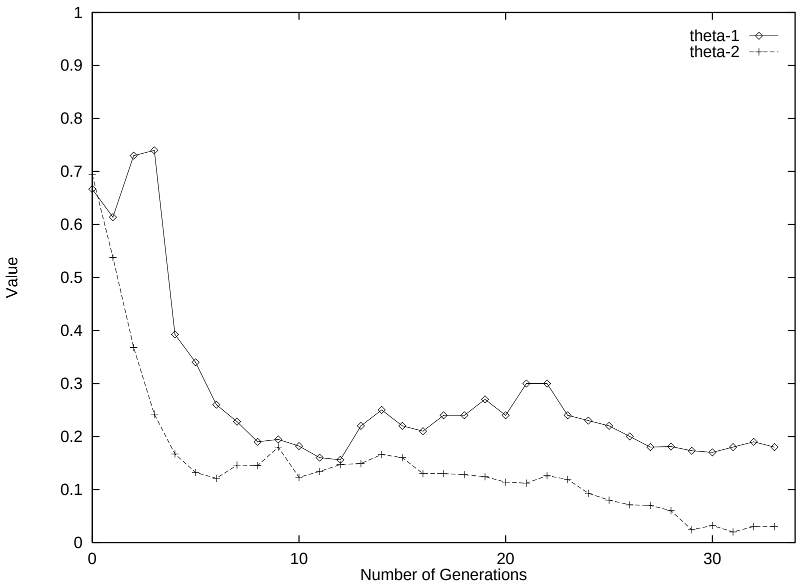

Figure 1.15
Plot of the population average value of θ_1 and θ_2 over the number of generations in the second GA experiment. The average generation length was 23000 time steps.

not converge to as close a range as θ_2, hovering between 0.18 and 0.3. The other four runs had similar results, with an average time to convergence of 91 generations.

The evolution of the thresholds introduces the question of what other parts of the learning rules can be evolved. In particular, the learning rule for each netwrok link uses the difference in the activation between 2 time steps of one of the nodes connected to that link and compares this difference to a threshold. We wondered what would happen if a link could compare the temporal difference in the activation of all of the nodes connected to it with a threshold? To investigate this, we changed the learning rules to accommodate more parameters. In the policy network, the learning rule became:

$$w_{i,g,o} \leftarrow w_{i,g,o} + R_i \times O_o \times G_g \times \delta_1 \times \delta_3 \times \delta_4, where:$$

$$\delta_1 = \begin{cases} 1, & \text{if } |I_i^{t-1} - I_i^t| \lesseqgtr_1 \theta_1 \\ 0, & \text{otherwise.} \end{cases}, \delta_2 = \begin{cases} 1, & \text{if } |G_g^{t-1} - G_g^t| \lesseqgtr_3 \theta_3 \\ 0, & \text{otherwise.} \end{cases},$$

$$\delta_3 = \begin{cases} 1, & \text{if } |O_o^{t-1} - O_o^t| \lesseqgtr_4 \theta_4 \\ 0, & \text{otherwise.} \end{cases}$$

where θ_1, θ_3, and θ_4 are specified by the genome, as well as $\lesseqgtr_1$, $\lesseqgtr_3$ and $\lesseqgtr_4$ which determine whether the rule tests for $<$ or for $>$.

The learning rule in the value network became:

$$V_{g,i} \leftarrow V_{g,i} + (I_i^{t-1} \times \delta_2 \times \delta_5)$$

$$\delta_2 = \begin{cases} 1, & \text{if } |G_g^{t-1} - G_g^t| \lesseqgtr_2 \theta_2 \\ 0, & \text{otherwise.} \end{cases}, \delta_4 = \begin{cases} 1, & \text{if } |R_i^{t-1} - R_i^t| \lesseqgtr_5 \theta_5 \\ 0, & \text{otherwise.} \end{cases}$$

where θ_2, θ_5, and each $\lesseqgtr$ are specified by the genome.

The above rules allow temporal difference thresholds on any of the nodes connected to a link to affect the learning on that link.

We expanded the agents' genome to five 7 bit numbers to specify the five thresholds, and 5 more bits to represent whether each $\lesseqgtr$ was $<$ or $>$. We predicted that the rules would evolve to resemble the ones we had designed, such that θ_1 and θ_2 would converge on values similar to the last experiment with $\lesseqgtr_1$ as $<$ and $\lesseqgtr_2$ as $>$, while $\theta_3, \theta_4, \theta_5, \lesseqgtr_3, \lesseqgtr_4$ and $\lesseqgtr_5$ would evolve such that they would have no effect on the learning, in a sense evolving away.

Figure 1.16 shows the value of θ_1, θ_2, θ_3, θ_4 and θ_5 in each generation of one of the runs. θ_1 and θ_2 converge to values similar to the previous experiment, and for every agent in the population, $\lesseqgtr_1$ and $\lesseqgtr_2$ are $<$ and $>$, respectively. θ_3 and θ_4 fall to 0.0, and $\lesseqgtr_3$ and $\lesseqgtr_4$ are $<$, while θ_5 rises to 1.0 and $\lesseqgtr_4$ is $>$. As predicted, learning rules evolved to a form where only θ_1 and θ_2 are used in learning. However, in two of the five runs, θ_2 converged to 1.0, with a $\lesseqgtr_2$ being $<$. Essentially, θ_2 evolved away as a threshold as well. All other values converged as expected.

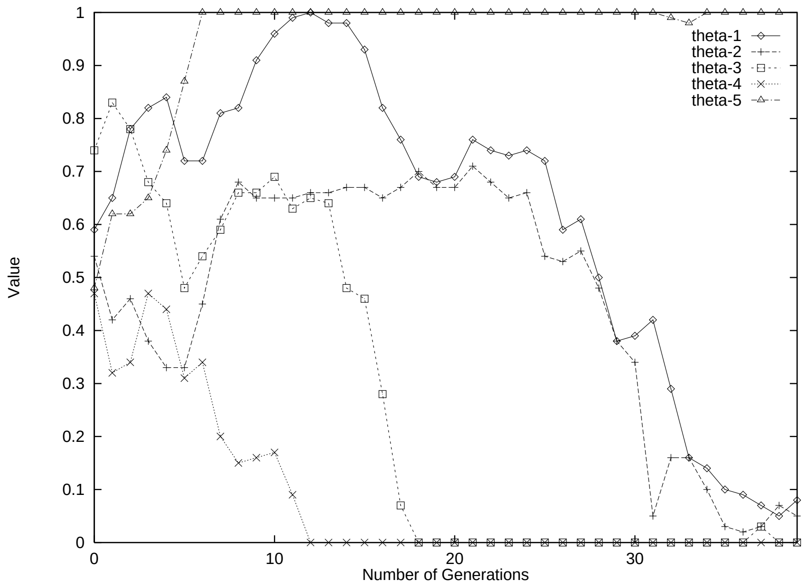

Figure 1.16
Plot of the population average value of θ_1, θ_2, θ_3, and θ_4 over time in the third GA experiment.

1.7 Evolution Discussion

Because it was the learning rules of the networks being evolved, rather than the weights or structures, it was the learning ability of the agents, rather than their behavior that was being tested. The experiments show that in static environments, it may be difficult to strike a balance between strong selectional pressure, and initial ease of survival. By gradually increasing the selectional pressure on learning agents, they can evolve to learn their tasks. In a more natural environment, the absolute level of difficulty of the environment can remain relatively static over an agent's lifetime, however, increasing difficulty can be simulated with social interaction between agents, such as parenting. Parent agents can provide protection to child agents as they learn about the environment, and the parent can gradually lower the amount of protection as the child agent's ability increases, providing the increasing selectional pressure needed for evolution of learning. This implies that in natural evolution, parenting precedes learning; parenting is necessary for learning to come about. This is clearly an area that requires much research.

That θ_1 does not converge to as precise a value as θ_2 implies that the *value* of θ_1 is less important for the success of the agent. There is usually a large difference between the events that θ_1 should include in learning and the set of events that it should exclude. This provides a range of equally valid values for θ_1 to be in.

When the learning rules themselves were allowed to vary, the convergence of the thresholds was not guaranteed, such as when θ_2 unexpectedly evolved away as a threshold. Closer inspection showed that without θ_2, the agents were still able to learn to behave reasonably well. They would incorrectly adjust their value networks based on naturally changing goal input, but the effect on the policy network was such that they were still able to find food and water, if less efficiently. At that point, θ_2 was stuck in a local minimum, because any movement away from 1.0, and the learning rule would begin throwing out data that the agent would otherwise have been using for learning, reducing the agent's fitness.

1.8 Related Work

Work related to MAXSON falls into four categories: second-order networks as agent controllers, neurally based reinforcement learning, neural associative networks, and genetic algorithm techniques.

Second-Order Controller Networks

Braitenberg [4] first showed that an agent with direct connections from left and right sensors to left and right wheels could be made to approach and avoid objects in it's environment. Werner [18] noted that second-order connections combining Braitenberg-like connections with goal input have advantageous properties for controlling agents. He noted that the combination of goal and sensory input enables an agent to pursue goal directed behavior, as well as interrupt such behavior to avoid a danger or take advantage of an opportunity to satisfy a less important goal. Once an agent has begun a consummatory behavior, it persists at it because of the proximity of the object to be consumed dominates the behavior. The agents can prioritize based on the activations of the inputs as well as the weights on the links. Finally, the agents can perform simultaneous actions, and do not need to exclusively choose a single action at each time step.

Werner hand designed the structure of his second-order networks and then used a genetic algorithm to set the weights. MAXSON's learning from fully connected networks results in both a structure (figure 1.12) similar to Werner's hand designed structure and weights that provide the agents with the above desirable properties.

Neural Reinforcement Learning

Often, neural networks are used within a reinforcement learning system as a function approximator for the value, policy, or Q-values function [17]. These networks replace a lookup table of values to scale up to large or continuous problem spaces, while the agent selects the single maximum score action (based on the network output) at each time step. In contrast, MAXSON takes the maximum of input for learning, but can output multiple simultaneous actions. A Q-learning system with an embedded neural network still has all the difficulties of Q-learning described above.

The complementary reinforcement backpropagation algorithm (CBRP) [1] is an example of a neural network that specifies the output directly, making it more similar to MAXSON. CBRP, however, requires that the agent receive immediate reward after each action, as opposed to the delayed reinforcement of the MAXSON environment. CBRP has no way of converting delayed reinforcement to visual immediate reinforcement.

Associative Networks

Billard and Hayes [3] use a technique similar to our value network in a mobile robot (DRAMA) to associate events cross-modally, as well as across time in order to label landmarks, and to learn series of perceptions. In DRAMA, event recognition systems (made up of feed forward networks) identify sensory events that are fed into an associative module. This fully connected module associates events that co-occur or occur in similar time frames by increasing the weights on the links between the nodes representing each event. They also applied DRAMA to social imitation learning [2]. A mechanism external to the architecture enabled the robotic agent to imitate the actions of another robot. By associating auditory input with imitated motor actions, the agents could learn a simple language. While DRAMA does learn about its environment, it neither contains an action selection mechanism such as MAXSON's policy network, nor does it use the output of the associator to generate reinforcement.

Genetic Algorithm Techniques

Cecconi, Menczer and Belew [5] used genetic algorithms to evolve neural networks that learned from parental example. They found that prolonging immaturity of agents was selected for when agents had the ability to learn from their parents.

Gomez and Miikkulainen [12] discuss making an environment incrementally more difficult because the final environment would be too difficult for the initial agents to start off in.

1.9 Conclusion

When agents need to learn about their environment to survive, it is important that they do so quickly and accurately. They should take advantage of the sort of environment they live in to maximize their chances of success. Our architecture uses second-order connections between goal and external input sensors, converts delayed discrete reinforcement to immediate continuous reinforcement across sensory modalities, applies max-based credit assignment strategies, and makes effective use of the spatial continuous nature of the task. As a result, the MAXSON neural network architecture is able to learn to approach food and water, while learning to avoid poison. It learns to achieve these multiple simultaneous goals much faster and more flexibly than the more general Q-learning techniques.

Genetic Algorithms can be used not only to evolve constants in the learning rules of MAXSON, but they can also evolve portions of the rules themselves. This technique can confirm both the sufficiency, as well as the necessity of aspects of the learning rules, helping to clarify the nature of the learning algorithm. It also can provider insight on the nature of learning and the role it plays among social agents.

Acknowledgments

This work is supported in part by an Intel University Research Program grant to the second author. The model runs Pentium II based computers, also donated by the Intel Corporation.

References

[1]D. Ackley and M. Littman. Generalization and scaling in reinforcement learning. In *Advances in Neural Information Processing Systems 2*, 1990.

[2]A. Billard and G. Hayes. Learning to communicate through imitation in autonomous robots. In *Proceedings of the International Conference on Artificial Neural Networks*, 1997.

[3]A. Billard and G. Hayes. Drama: A connectionist architecture for control and learning in autonomous robots. *Adaptive Behavior*, 7(1), 1999.

[4]V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, Cambridge, MA, USA, 1984.

[5]F. Cecconi, F. Menczer, and R. Belew. Maturation and the evolution of imitative learning in artificial organisms. *Adaptive Behavior*, 4(1), 1996.

[6]F. Cecconi and D. Parisi. Neural networks with motivational units. In *From animals to animats: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 346–355, 1993.

[7]P. S. Churchland and T. J. Sejnowski. *The Computational Brain*. Bradford Book/MIT Press, Cambridge, MA, 1992.

[8]F. L. Crabbe and M. G. Dyer. Maxson: Max-based second-order neural network reinforcement learner for mobile agents in continuous environments. Technical Report CSD-900009, UCLA, 1999.

[9]F. L. Crabbe and M. G. Dyer. Second-order networks for wall-building agents. In *Proceedings of the International Joint Conference on Neural Networks*, Washington D.C., July 1999.

[10]G. M. Edelman. *Neural Darwinism*. Basic Books, New York, 1987.

[11]C. L. Giles and T. Maxwell. Learning, invariance, and generalization in high-order neural networks. *Applied Optics*, 26(23):4972–4978, 1987.

[12]F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5(3-4):317-347, 1997.

[13]D. O. Hebb. *The Organization of Behavior*. Wiley, NewYork, 1949.

[14]R. W. Oppenheim. Naturally occuring cell death during neural development. *Trends in Neurosciences*, 8:487–493, 1985.

[15]S. Russell and P. Norvig. *Artificial Intelligence. A Modern Approach*. MIT Press, Cambridge, MA, 1995.

[16]R. Sun and T. Peterson. Partitioning in reinforcement learning. In *Proceedings of the International Joint Conference on Neural Networks*, 1999.

[17]R. S. Sutton and A. G. Barto. *Reinforcement Learning*. MIT Press, Cambridge, MA, 1998.

[18]G. M. Werner. Using second order neural connection for motivation of behavioral choices. In *From animals to animats3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, 1994.