Adaptive Behavior 2000, 8(2), 145-168

Goal Directed Adaptive Behavior in Second-Order Neural Networks: The MAXSON family of architectures

FREDERICK L. CRABBE & MICHAEL G. DYER*,
U.S. Naval Academy, Annapolis, Maryland
**University of California, Los Angeles*

The paper 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. It also enables an agent to learn vicariously, from the successes and failures of other agents. 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 paper shows that an extension of MAXSON (V-MAXSON) enables agents to learn vicariously, and this improves the overall survivability of the agent population.

Keywords: Second-order neural network; simulated autonomous agents; reinforcement learning; vicarious learning.

1 INTRODUCTION

Agents that reside in complex environments need to be able to meet their survival goals, often when several of these goals have hig priority simultaneously. When the environment is dynamic, or when the agents cannot be pre-programmed to meet these goals, they must learn to meet them.

One of the most common methods of learning for animats is to use reinforcement from the environment in the form of Temporal Difference Learning or Q-Learning (Sutton and Barto, 1998). These methods use reward from the environment to associate values to particular actions in particular states in the environment. The standard benchmarks for these techniques are navigation tasks, such as maze learning (Kaelbling et al., 1996; Dietterich, 1998; Ring, 1992), which are often situated in a grid-based simulation environment (Thrun and Schwartz, 1995; Sutton, 1996). Typically, these models take a large number of iterations before the agent behavior is suitable, and focus on a single goal at a time (Blumberg et al., 1996). The number of iterations required can be on the order of tens of iterations for simple behaviors (Araujo and Grupen, 1996) to tens of thousands for more complicated ones (McCallum, 1996). While maze learning and related tasks are important skills for animats, we are interested in investigating other skills that are also important for the survival of an autonomous animat. In particular we are interested in the task of learning about objects in the environment as they pertain to multiple
internal goals, and learning how to move toward or away from these objects.

This skill is important to a completely uninitialized animat thrust into a novel environment and forced to adapt to its surroundings, which requires the learning to be extremely fast. Typically an agent would get only one interaction with an object before it is required to act appropriately. For example, if the agent survives an interaction with a potentially dangerous object, it should immediately learn that the object is dangerous, and also begin to learn to avoid objects of that type. The agent should then be able to simultaneously avoid that type of object while pursuing other goals.

An Extended Braitenberg Architecture (Braitenberg, 1984; Pfeifer and Scheier, 1999) using higher-order connections (Giles and Maxwell, 1987) is a suitable architecture for this problem. Braitenberg animats consist of sensors connected directly or indirectly to actuators, resulting in reactive behavior. This reactivi-

ty makes this architecture an excellent starting point for such an approach/avoid task. Earlier work (Werner, 1994; Crabbe and Dyer, 1999b) has shown that extending the architecture, with the addition of second-order connections, enables the animats to exhibit behavior that: is goal directed, balances the relative priority of goals, and exhibits compromise behavior; that is, selects an action that is a compromise between actions each of which satisfy a different active goal. It has been demonstrated (Crabbe and Dyer, 1999b) that agents with this architecture are able to build nest-like structures, while simultaneously satisfying survival goals such as consuming food. Our problem here is to develop learning algorithms for this second-order architecture that the agents can use to learn approach/avoid behaviors quickly in order to ensure their survival.

This paper presents MAXSON, a flexible second-order neural architecture that: learns faster than any using traditional reinforcement learning approaches; is able to generate and apply the reinforcement in a neurally plausible manner; can balance the requirements of multiple simultaneous goals; and is flexible enough to incorporate extensions to the basic reinforcement paradigm. The paper also presents one such extension that enables agents to learn vicariously (i.e., learn about objects in the environment by observing the reactions of other agents due to their interactions with those objects) and shows that this change positively affects the quality of learning. Vicarious learning is an entirely new class of social learning which can be explored independently of the architecture. Finally, this paper discusses the advantages of MAXSON and compares it to other similar systems.

1.1 Environment and Agent Task

In order to evaluate an agent control architecture, both an environment and a task are needed. The environ- ment 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.

For the purposes of this paper, an agent lives in a two-dimensional, continuous environment. 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.

This environment is dynamic due to the actions of the agents in it. When an agent eats food, it changes the location of that piece of food as well as the overall distribution of food in the environment. A large number of agents eating food can drastically alter the nature of the environment from food-rich to food-poor in a short time. In work not discussed here, agents can also relocate objects (used as building materials) and as a result create new barriers and remove old ones (Crabbe and Dyer, 1999b).

In order 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 with which it comes into contact. 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.

2 MAXSON ARCHITECTURE

Our agents use an architecture, called MAXSON (Figure 1), (Crabbe and Dyer, 1999a), 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 create the agent's actions at each time step, while the value network is used to generate

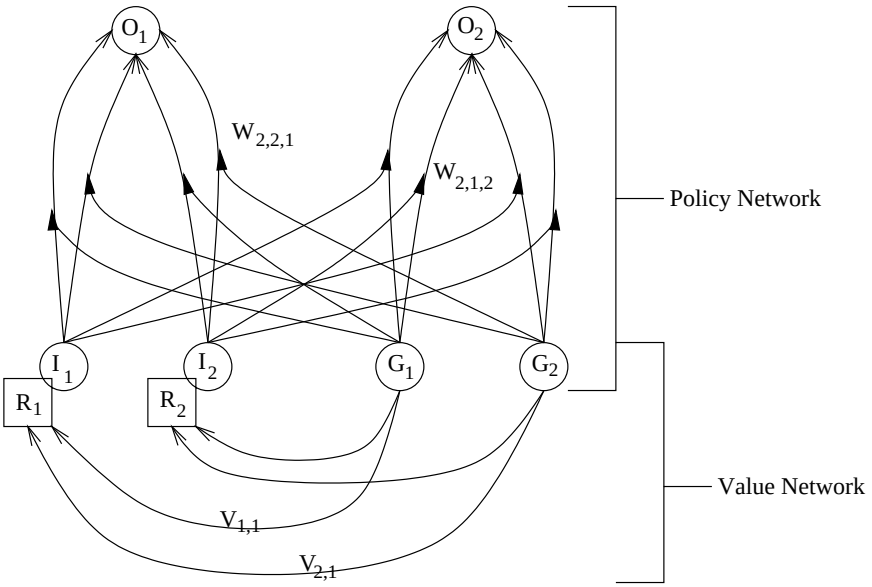

Figure 1. A small-scale MAXSON network. I-nodes receive external input. Associated with each I-node is a reinforcement node R. G-nodes receive internal goal input, and O-nodes perform motor actions. Each black triangle (joining two input connections) is a second-order connection with a weight W.

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, not only immediate survival goals such as eating, but also other goals involved in more sophisticated behavior. This can result in an agent having multiple, possibly conflicting goals that fluctuate in urgency over time (the task environment in which a MAXSON agent is designed to learn). 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 of or failure to satisfy some goal. Multiple satisfactions of a goal could be 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.

2.1 Second-Order Networks

A MAXSON-based agent uses the second-order poli-

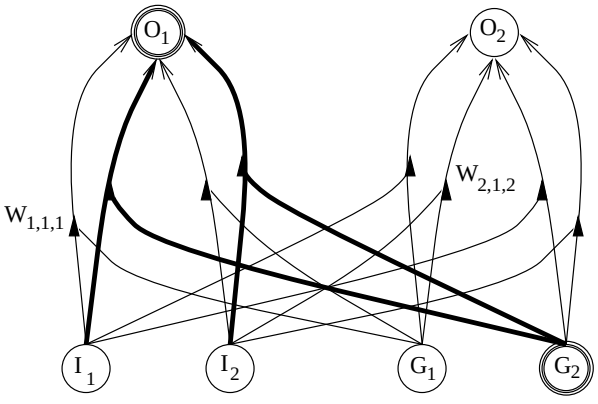

Figure 2: A second-order policy sub-network. G-nodes (goals) dynamically modulate the influence of I-nodes on O-nodes (actions) via second-order connections.

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 :

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

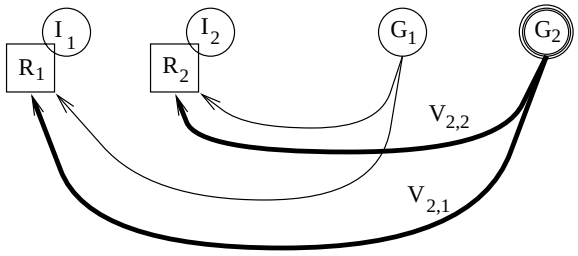

Figure 3. A first-order value sub-network. Each Ri node holds the amount of the reinforcement for the adjoining external input sensor node (Ii). Adjustment occurs only if the difference between the input on the external input sensor node and the input at the previous time step on the external input sensor node is less than a constant threshold .

cy sub-network (Figure 2) to choose what actions to take at each moment. Second-order connections multiply pairs of inputs to nodes before the inputs are summed (Giles and Maxwell, 1987; Werner, 1994). In 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 Iiis 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:

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

If there were two external sensors (food_left and

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

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

$$g \leftarrow \arg\max_g \overset{\mathsf{r}}{G}$$

for each i **in** I **do :**

$$R_i \leftarrow (I_i^t - I_i^{t-1}) \times V_{g,i} \times G_g$$

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

Figure 4. Update of weights V in value network.

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

$$i \leftarrow \arg\max_i \overset{\mathsf{r}}{I}^{t-1}$$

for each g **in** G **do :**

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

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

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: $O_{turn\_left} = (I_{food\_left} \text{ x } G_{hunger} \text{ x } W_1 + I_{food\_right} \text{ x } G_{hunger} \text{ x } W_2)$.

2.2 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 Ii 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.

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 2 as O_1 and G_2). Then the weight on each sec-

ond-order link from G_g to O_o (dark lines in Figure 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.

2.3 Reinforcement Calculation by Value Network

In MAXSON, a first-order single layer network (called the value network) re-calculates 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 3). The reinforcement is calculated as shown in algorithm 2. Activation on the maximal goal sensor G_g (dark circle G_2 in Figure 3) is multiplied by the weights $V_{g,i}$ on the links from G_g (dark lines in Figure 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).

2.4 Value Network Weight Adjustment

The value network uses Hebbian style learning (Hebb, 1949) to update its 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 4 from both goal nodes) from that goal sensor node to the external input sensor node (dark circle I_1 in Figure 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.

2.5 Architectural Design Features

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

2.5.1 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

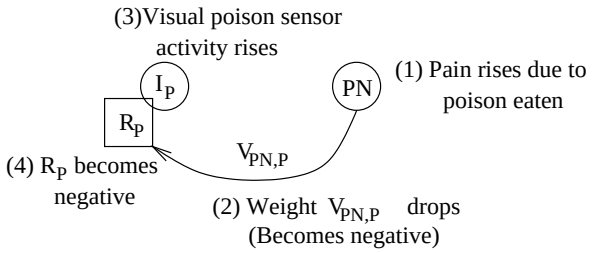

Figure 5. A value network example. PN is the pain input node, IPis the visual node that receives input about a poison object. RP is the reinforcement node for IP.

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" sub-portion of the policy network becomes very active, and, all other things being equal, the agent approaches the water.

2.5.2 Delayed Reinforcement Converted to Immediate Reinforcement via Cross-Modal Learning

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 function 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 5), when the agent eats poison, the activation on the 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 x 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 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.

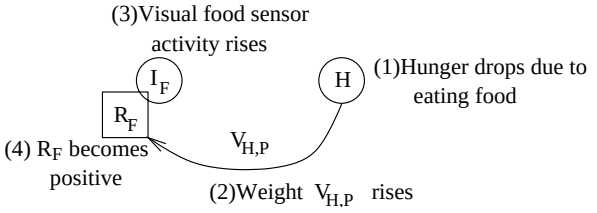

Figure 6: Another value network example. H is the hunger goal input node, IFis the visual node that receives input about a food object. RF is the reinforcement node for IF. The agent has no a priori knowledge that food satisfies hunger.

2.5.3 Using MaximumNodes 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 agents learning bad policies. In order to focus the attention of the second-order learning algorithm on

the appropriate weights, we introduced 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 was 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 does not 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.

2.5.4 Temporal Difference Based Sensor and Goal Threshold

Sometimes an atypical event occurs and it interferes with the agent's learning how to behave in the typical case. For example, if an agent that sees food on the left turns to the right, normally it should get a negative reinforcement. But if, while it is 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 from that sensor. So, if an object comes into view at some point, the large change in the sensor input causes that data not to 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 an event has occurred that should warrant a change in the agent's reinforce-ment attitude toward some object, and the goal to which this change relates. 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 a 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 found, by our own experimentation, that given the nature of the simulated environment, both thresholds worked best at 0:02. Genetic Algorithm experiments (Crabbe and Dyer, 2000) demonstrated θ_2 indeed converging on 0:02, while θ_1 stabilizes in the range between 0.18 and 0.3.

3 MAXSON EXPERIMENTAL METHOD

To test the MAXSON architecture, we ran three experiments. 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 agent's performance when always eating and drinking. We measured the agent's performance by counting the number of objects the agent ate and drank, allowing 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 poison units eaten from the total number of food units eaten and water unite drunk. 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

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

Table 1. A possible 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.

experiment, an agent should no longer eat and drink as much as it could, but only eat and drink when hungry or 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 Q-learning, the system maintains an estimate of how "good" it would be for the agent to perform a particular action when it is in a particular world state. These estimates are called Q-values. Each time the agent performs an action, it updates the Q-values for the state it was in based on any reward it gets and its prediction of future reward. In the first two experiments, reinforcement of +1 was given for eating food, -1 for eating poison and 0 otherwise. In the third experiment, the reward for eating food or drinking water was the value of the agent's hunger or thirst at that moment. The reward for eating poison was -1 x the agent's pain at that time.

The table-based Q-learner is based on the $Q(\lambda)$ learner presented in Sutton and Barto (1998). This Q-learner keeps an explicit table of all possible world states. Each state is defined directly by the agent's sensory input. The continuous input to the agent was discretized into a 3 x *m* table, where *m* is the number of inputs (12 or 15, depending on experiment) and there are 3 possible discretizations (Table 1). This results in a state space with 3^{15}, or 14,348,907 possible states.

The linear function Q-learner is also a version of (λ) Q. It uses a linear function of the inputs as an approximation of the table of Q-values. There was a separate function for each action ai, that is, a separate equation to calculate the Q-value for each possible action, such as "turn_left". The equation is shown in Equation 2.

$$Q(s, a_i) = w_1E_1 + w_2E_2 + \ldots + w_{15}I_3. \qquad (2)$$

The weights were adjusted using gradient descent (Sutton and Barto, 1998).

The MAXSON agent consisted of the policy and value networks as described in the architecture section above. In the third experiment we ran two versions of MAXSON: one with the links from its goal nodes lesioned (as in experiments 1 and 2), and one with the links from its goal nodes intact, thus using goal input.

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 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 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.

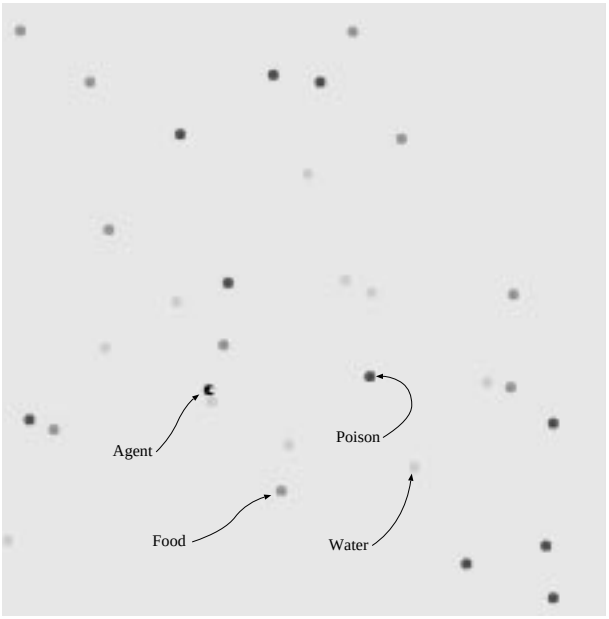

Figure 7. One of the random initial positions.

4 MAXSON RESULTS

The results of the first experiment (food and water only) are shown in Figure 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 agent. The difference between the maximum scores of the linear function Q-learner and the MAXSON agent is statistically significant at 99.92% using a Mann-Whitney U-test. The table-based Q-learner fails to learn any good policy within the time frame of the experiment.

Figure 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 is time taken by the agent to randomly consume all three types of objects. The performance of the linear function Q-learner was also similar to its performance in experiment 1. It learns more slowly and converges to a score of 16. The table-based Q-learner again fails to learn a viable policy in the time given.

Figure 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 performs about as well as the MAXSON agent that ignores its goal input. As the linear function Q-learner gets better at avoiding

poison and at collecting food and water, its survival time goes up; but it fails to account for its goal inputs properly, over-consumes, and dies from starvation. The table-based Q-learner performs comparatively better in this experiment. While it does not have a policy any different from that of the the previous experiments, its "strategy" of consuming very little food or water succeeds about as well as a strategy that involves eating too much.

5 MAXSON DISCUSSION

5.1 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 from a global location (Sun and Peterson, 1999). MAXSON was designed for mobile agents that balance multiple goals in a continuous environment. Because of this, MAXSON shows 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.

There are a number of reasons for the failure of both types of Q-learners in the various experiments. The discrete table entries of table-based Q-learning generalizes poorly in a continuous environment. Imagine an agent in a state where food is nearby, and slightly to the right. The agent is unable to notice the similarity between this state and one in which it sees food nearby to the right *and* poison distantly to the left. From the point of view of the table-based Q-learner, this is an entirely different state, about which it knows nothing. Similarly, when the agent sees food that is distantly to the right, that too is an entirely different state. The table-based Q-learner ignores the fact that distant food is similar to near food, which is also similar to near food with other extraneous input. A table-based Q-learner should be able to overcome this difficulty given enough time. Note that often the speed of Q-learning is reported in terms of iterations of the algorithm, where each iteration ends with an achievement of the goal. In the 180,000 time steps the table Q-learner was run, the agents interacted with food or water roughly 30 times, resulting in 30 iterations of the algorithm. Given the size of the state space, this is insufficient time for the table Q-learner to learn the correct behavior.

In experiment 3 the linear function Q-learner did receive goal input but was unable to form a good policy, instead behaving as it did in experiment 2. Between 40,000-60,000 time steps, the agent eats and drinks while avoiding poison, but without regard to the goal input. At that stage, it consumes all the food and water within a few thousand time steps, and then starves to death in 15,000 time steps. The reason the goal input had so little effect was that it was overwhelmed by the rest of the input. Small changes in the goal input should have large effect on the behavior of the agent, resulting in a non-linear target-function that cannot be represented by a linear Q-function.

A different way of introducing goals to the Q-learner would be to use three different Q-functions, one for each goal, and then use the Q-function that corresponds to the current maximal goal at each time step. Aside from lengthening the learning time, this approach prohibits the agent from exhibiting any compromise behavior. This can be detrimental to the agent, since when it is acting on one goal, it completely ignores the other goals. For example, if an agent's maximal goal is to approach an object, it will do so even though it might pass close to another object which it has a strong goal to avoid. The lack of the ability to consider and react to multiple goal input limits the abilities of such an agent.

To preserve the ability to perform compromise behaviors, while introducing non-linearity, a function- based Q-learner could multiply the goal and external input together as in Equation 3.

$$Q(s,a) = \sum_{i \in I, e \in E} I_i E_e W_{i,e,o}. \qquad (3)$$

This equation is non-linear, computing the same function as MAXSON does. Because each term combines both external and internal input, small fluctuations in internal goal input can have the needed large effects on the output.

MAXSON combines six features to enable it to perform well in comparison to standard Q-learning:

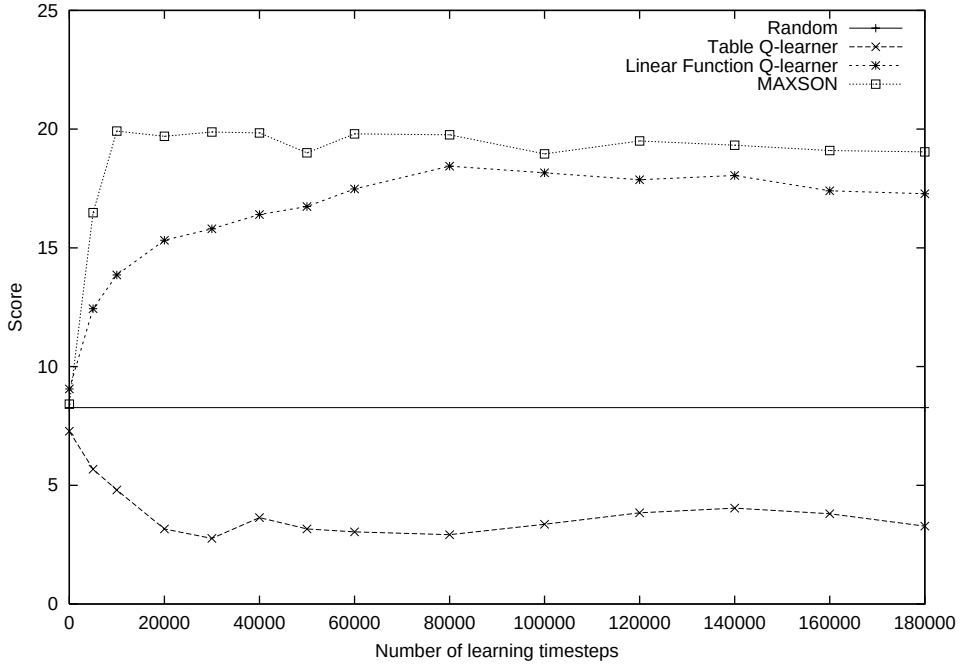

Figure 8. Experiment 1. The x-axis is the number of time steps and the y-axis is the score. The MAXSON agents converge six times faster than the linear function Q-learner and obtained a higher score (99.92% significance on a U-test). The horizontal line represents the score of a random agent performing the task.

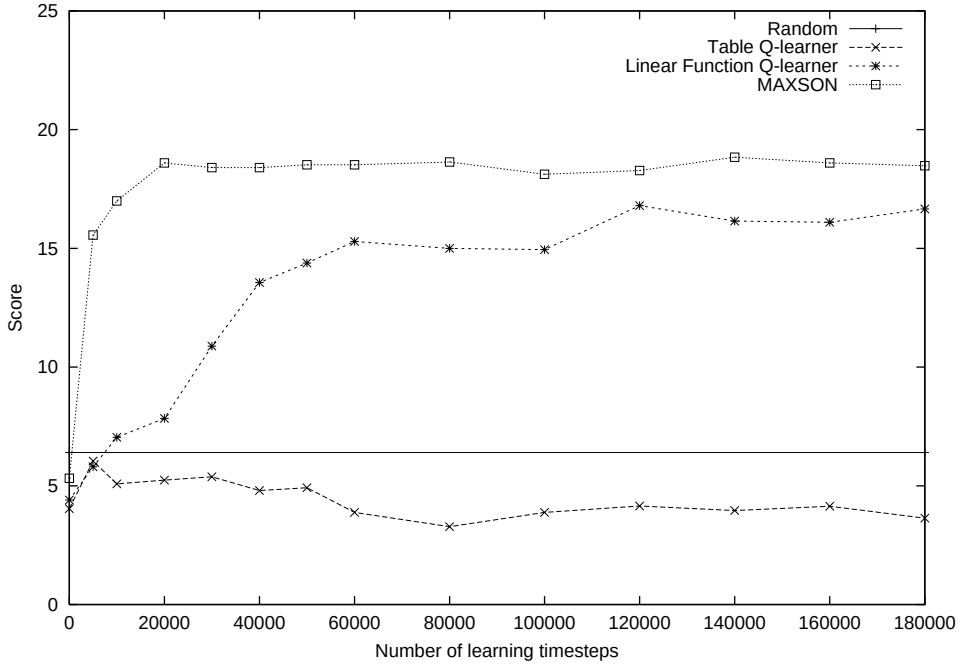

Figure 9. Experiment 2. The MAXSON agents converge to a good score (18.4) in 20,000 time steps. The linear function Q-learner reaches converges on a score of 16.4.

- Firstly, 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 works because motion in a continuous environment changes the input continuously. Actions that move the agent closer to its goal make the goal appear closer to the agent. 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.

- Secondly, 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.

- Thirdly, 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 misplaced credit. By reducing the errors in credit assignment, the network learns more rapidly and avoids learning incorrect associations.

- Fourthly, 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.

- Fifthly, 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.

- Lastly, the second-order connections allow the goals of the agent to have an important major influence over its behavior. An agent does not waste its resources by consuming them when it does not need them.

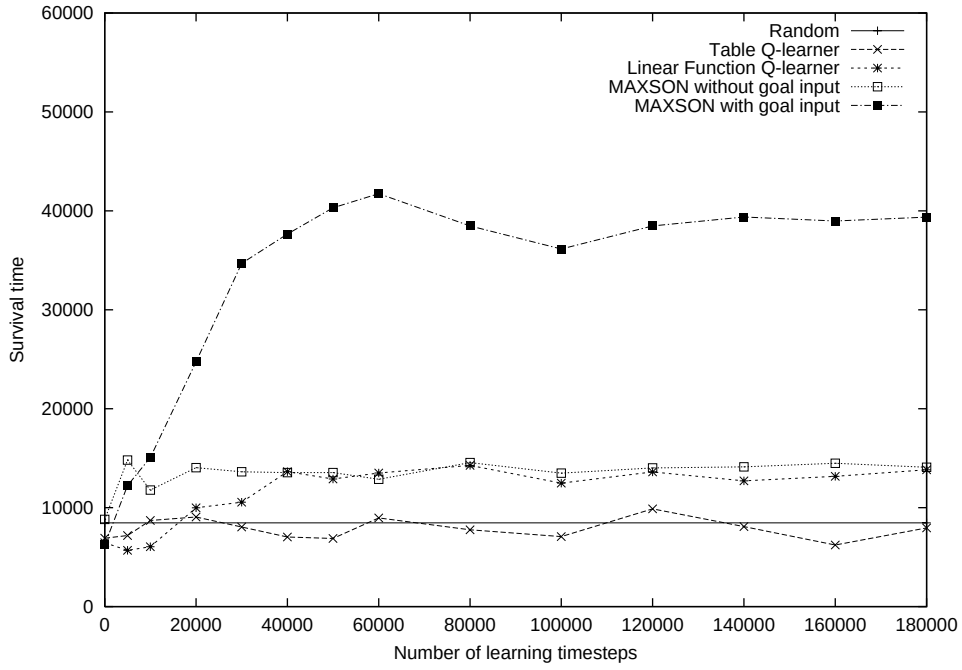

Figure 10: Experiment 3. The MAXSON agents that use their goal inputs survive the longest. Here, the y-axis is the survival time in number of time steps. Each data point indicates when an agent died (averaged over 10 trials).

5.2 How MAXSON Networks Self-Organize

Figures 11,12, and 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 poison will cause the pain input to increase while eating food/water will cause 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.

There are three ways in which the MAXSON architecture and learning algorithms are neurally inspired: (1) the second-order connections approximate axo-axonal neural connections, (2) the update rule is Hebbian- like, and (3) the network begins as a fully connected network and connections are pruned during development. Figures 12 and 13b show only the strongest connections after learning. 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. Links with a weight of 0 are effectively pruned from the network. Thus, a structured connectionist network self-organizes out of a distributed fully connected network (Figure 11). The pruning of excessive connections is common during neural development (Oppenheim, 1985; Edelman, 1987;

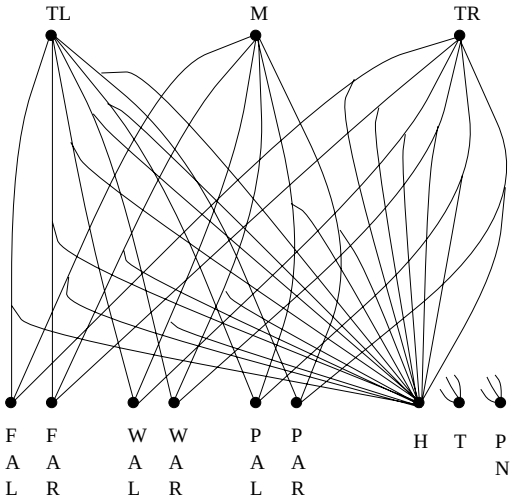

Figure 11. Initial policy network architecture with full connectivity and random weights in range of [0; 0:01]. TL = turn-left, TR = turn-right, PN = pain sensed, H/T = hunger/thirst sensed, FAL = angle to food object sighted on left, PAR = angle to poison object sighted on right, and so on. The distance visual sensor nodes were left out for clarity. Only connections from the hunger goal input node are shown.

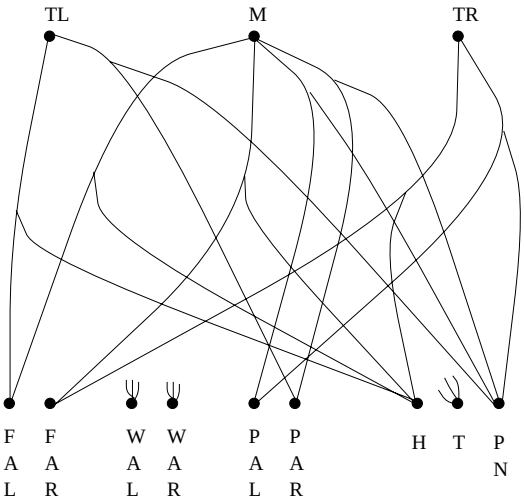

Figure 12. Simplified structure of a high-performance policy network after learning. All links from both hunger and pain with a weight greater than 0.01 are shown to highlight the learned structure. All links shown have weights greater than 0.5. Connections from thirst are omitted for clarity.

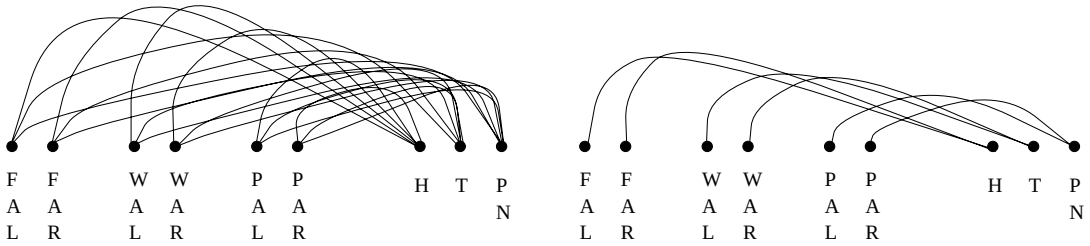

Figure 13. Value network before and after learning. Initially it has full connections, but it learns which sensors are more likely to be related to the achievement of appropriate goals.

Churchland and Sejnowski, 1992) and is theorized to improve the precision of of neural circuits (Landmesser, 1987).

6 V-MAXSON

In previous sections, we showed that MAXSON is a viable architecture to control an agent to act in an environment and to learn from its experiences. MAXSON's modular nature also makes it extensible to other types of learning, such as vicarious learning. The goal of the vicarious learning system is for one agent to learn from observing both another agent's actions and the results of those actions. For example, suppose agent A sees agent B eat a poison object and agent A sees B grimace, indicating that it did not like the poison. As a result A learns to avoid the poison without having experienced it itself. The ability to learn vicariously could be an advantage to an agent that has not yet explored its entire world.

Vicarious learning requires a number of features. The agents need to be able to signal their pleasure and displeasure with the environment, and the agents need to be able to perceive these signals from others. The agents need to reliably associate actions they see other agents performing with the signals these agents emit. The agents also need to keep vicariously learned knowledge separate from other knowledge because it is less reliable than direct experience, and it does not indicate what goals particular actions satisfy. To include these features, a small number of changes needed to be made to MAXSON.

To learn vicariously, the agents must be able to signal their reaction to objects they interact with, and they must be able to receive those signals. V-MAXSON (Crabbe and Dyer, 1999c) agents signal by gesturing positively (e.g., grin) or negatively (e.g., grimace). They also have new input sensor nodes to receive those signals. Because the signals are gestural, the agent only perceives the signal if it is turned toward the signaling agent.

V-MAXSON agents also have a small number of innate connections to and from these new sensors and output motor nodes. Whenever a goal node's activation falls more than a fixed amount, the agent automatically grins. Conversely, if one of the agent's goal nodes rises more than a fixed amount, the agent automatically grimaces. For example, if the agent eats food when it is hungry, its hunger will fall, causing it to grin.

The agents also need a new goal node to control vicariously learned behavior. This vicarious goal node (or Vic node) has innate connections from the agent's grin and grimace sensors. Whenever an agent A perceives a peer's grin, the activation on A's Vic goal node falls, and whenever A perceives a grimace, the activation on its Vic node rises. Change in the activation of the Vic node triggers learning in the value network. The vicarious goal node is different from the other goal nodes in that it normally has a activation of 0.2, and once it is changed, either up or down, it slowly returns back to 0.2.

6.1 Phase-based Vicarious Learning

In a MAXSON agent, the value network learns the association between a goal node and the maximal input node, but because we now want the agent to learn vicariously from agents that are far away, this mechanism will not always work. For Example, if agent A sees agent B eat food, but the closest object to A is poison, then the MAXSON net will associate the

positive signal emitted by B with the poison. To prevent this crosstalk, agent A must determine which objects are significant with respect to the change in its vicarious goal input. This is accomplished by binding events in the environment using temporal synchrony (Gray et al., 1989; Shastri and Ajjanagadde, 1993).

V-MAXSON input is split into a two-phase process. The first phase (the object phase) is exactly the same as the input to the MAXSON network. In the second phase (the event phase), only the input nodes that correspond to the closest event are active. An event is defined as an agent taking an action on another object. For example, if at a time step an agent can see a food object to the left and another agent eating a poison object to the right, then in the object phase, the FAL (Food Angle Left), FDL (Food Distance Left), AAR (Agent Angle Right), ADR (Agent Distance Right), PAR (Poison Angle Right), and PDR (Poison Distance Right) sensor nodes are active. In the event phase, only the AAR, ADR, PAR, and PDR node are active, indicating that these objects were involved in an event.

In V-MAXSON, all the connections in the value net adjust their weights during this event phase. By temporally separating the event from the rest of the input, it is made clear to the network which objects are important and should be used for learning.

6.2 Delayed Vicarious Learning

Realistically, it would be desirable to allow for some delay between when an agent performs an action and when it signals with a grin or a grimace (e.g., if an agent eats poison, it may take several moments before the agent begins to feel ill). We account for this delay by allowing the activations in the event phase to decay slowly over time. If an agent sees an event, the input during the event phase corresponds to the event seen. If, at the next time step, the agent does not see an event, then its input during the event phase is the linearly decayed activations corresponding to the event in the previous time step. This decay continues until it reaches 0, or a new event is observed, replacing the event input. So if agent A sees agent B eat food, then agent A will receive input during the event phase corresponding to the food. If over the next few time steps, agent A sees no other events, the input during its event phase will still be food, but with decayed activations. Finally, when agent A sees agent B grin, agent A will still be receiving input in its event phase representing food, and the agent can associate a drop in the Vicarious node with food.

The introduction of decay makes the agent more flexible with respect to time delays in vicarious learning, but introduces a problem of confusion. If an agent A observes agent B eat food, then shortly later observes B eat poison, and then observes B grin because of the earlier food, then agent A would vicariously confuse poison with the grin.

Because an agent perceives its own actions as events, the V-MAXSON agent still performs reinforcement learning as the MAXSON agent does. But, the V-MAXSON agent can also learn vicariously from other agents, such as when agent A observes agent B eating food. B eats the food and B's hunger goes down. This causes B (by an innate connection) to grin. If agent A is facing B, then A will perceive the grin, which will cause (also via an innate connection) A's vicarious goal node to drop to zero. At the same time, in A's event input phase, A sees agent B eating the food. As the activation on A's Vic node begins to rise, A's policy network begins to learn how to approach the food (Figure 14). Once A actually eats food, its hunger drops, and A thus learns to associate food with hunger. Thus through the process of vicarious learning followed by reinforcement learning, agent A learns that to satisfy hunger, it should approach food.

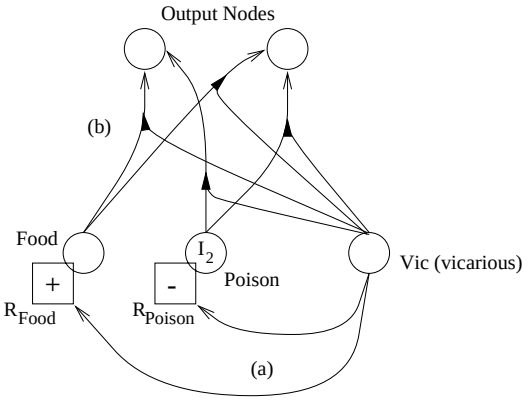

Figure 14. Portion of a V-MAXSON net learning vicariously. (a) Weights between Vic node and food sensor nodes are increased due to perceived grins, causing positive reinforcement to be generated at relevant sensor nodes. Perceived grimaces cause poison objects to become negatively reinforcing. (b) Reinforcement at sensor nodes causes learning on higher-order connections between sensor and motor nodes.

	MAX	V-MAX
Average	1656	6520
Average best	2301	17711

Table 2. V-MAXSON experiment 1, vicarious learning in an unforgiving environment. The left column contains the results for MAXSON agents and the right column contains the results for V-MAXSON agents. The first row is the average survival time of all the agents of that type. The second row shows the average survival time of the best agent in each run. Best V-MAXSON agents can still die because they run into poisons hidden behind food or water.

7 V-MAXSON EXPERIMENTS/RESULTS

To test V-MAXSON, we ran three experiments. In the first experiment we compared the average survival time of a group of V-MAXSON agents with the average survival time of a group of MAXSON agents in the same environment. The environment was "unforgiving" in that when an agent ate a poison, it would be removed from the environment (killed). The agents still had to eat the food and drink the water to stay alive. In each run, four agents of the same type were placed in an environment with 20 food units, 20 water units and 20 instant-death poison units. The simulation was run until all the agents were dead. There were 5 runs for each type of agent, varying the initial position of objects each time.

We hypothesized that, because the MAXSON agents needed to sample the deadly poison to learn about it, they would die more quickly than the V-MAXSON agents, which would have the advantage of learning vicariously from the fatal mistakes of their peers. Table 2 shows the results of the first experiment. As expected, the average survival time of the V-MAXSON agents is several times longer than the average survival time of the MAXSON agents. When looking at the best-surviving agent of each run, the difference is even greater.

In the second experiment, we wanted to test the speed of learning in groups of MAXSON agents and groups of V-MAXSON agents. The environment was made more forgiving in that, while poison harmed an agent that ate it, it did not kill it. Because the higher-order connections in the policy network prevent the agents from eating and drinking when already satiated, we disconnected the inputs from the goal nodes in the policy network, causing the agents that have learned to approach food/water to always approach food/water regardless of their hunger/thirst. This "greedy" technique decreased simulation time and emphasized learning speed. Each run consisted of alternated learning and testing. Four greedy agents of a single type (MAXSON vs. V-MAXSON) were placed in an environment with 20 food units, 20 water units and 20 weak poison units. The agents were then allowed to learn about their environment. Periodically the learning was stopped and the agents were tested. Testing was done after 500, 1,000, 2,000, 3,000, 4,000, 5,000, 10,000, 15,000, 20,000, and 25,000 time steps.

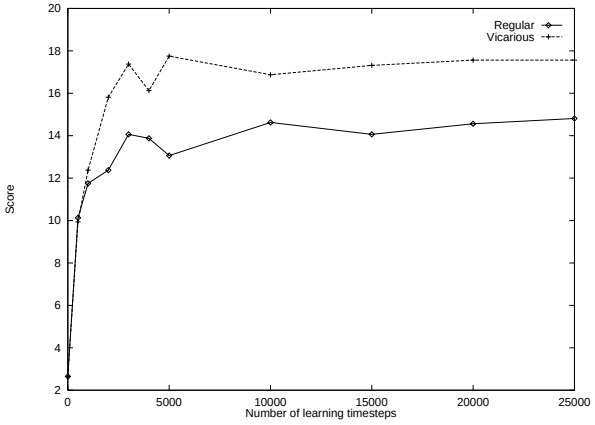

Figure 15. V-MAXSON Experiment 2, vicarious learning in a forgiving environment. The graph compares the performance of vicarious learning agents with non-vicarious agents. The x-axis is the number of time steps the agent spent learning and the y-axis is the score. The average score of the vicarious learning agents is significantly higher than their non-vicarious counte parts.

When testing, each agent was placed in a smaller environment by itself with learning turned off. The testing environment had 10 food units,10 water units and 10 poison units. The agent was allowed to interact with the testing environment for 20,000 time steps, and then given a score (the number of food units eaten plus the number of water units drunk, minus the number of poison units eaten). Testing was performed on each of the four agents. Afterwards, the four agents were placed back in the learning environment and the agents were allowed to learn until the

next test point. After 25,000 time steps, the testing was done once more, and then the run was completed. There were five runs for each type of agent (MAXSON or V-MAXSON), varying the initial positions of agents and objects in the environments.

We hypothesized that the vicarious learning agents would learn faster than their non-vicarious cousins because the vicarious learning would allow the agents to experience the effects of the objects at a distance and thus learn about them earlier in the training. Figure 15 shows the results of the second experiment. Instead of learning faster, the vicarious learning agents learned better on average. The difference between the final average scores of 14:81 for non-vicarious learners and 17:56 for vicarious learners is statistically significant with a 99.98% confidence interval using a U-test.

In the third experiment, we tested the robustness of the vicarious agents by examining their ability to recover from the effects of confusions. We initialized the value networks of individual agents as if they had incorrectly associated the satisfaction of (or failure to satisfy) a goal with the wrong type of object. For example, the weights on the connections between the vicarious node and food were set to -1, causing the agent to build a policy to avoid food. We compared a regular V-MAXSON agent to two types of agents with built-in incorrect biases. One type of agent, described above, had weights of -1 on its value network links between the vicarious node and food; the other type of agent had the weights on the links of its value network between the vicarious node and poison set to 1, causing the agent to build a policy to approach poison. We ran the agents alone in their environment so as not to introduce any other vicarious experiences. There were 10 food, 10 water, and 10 poison units. The simulation was run for 25,000 time steps, and tested at 500, 1,000, 2,000, 3,000, 4,000, 5,000, 10,000, 15,000, 20,000, and 25,000 time steps. There were 5 runs for each type of agent and each test was repeated 5 times.

We hypothesized that the agents with the mis-initialized value networks would perform similarly, but not as well as the control agent. We expected that eventually the initialized agents would experience the objects for themselves, and the weaker input from the vicarious node would yield to the stronger input from the other goal nodes. Figure 16 shows the results of the third experiment. As expected, an age with no improperly initialized weights performs better initially, scoring significantly better (99.58% on a U-test) at time 4,000. But the difference at time 25,000 is not significant (15.06% on a U-test). The two mis-initialized agents have no significant difference at any time during learning.

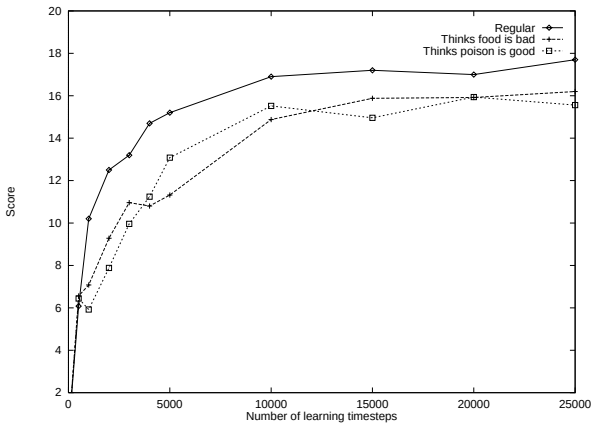

Figure 16. V-MAXSON Experiment 3: Vicarious robustness. This graph compares a V-MAXSON agent with V-MAXSON agents with initial confusions. The confusions hinder learning early on, but there is no significant difference after 25,000 time steps.

8 V-MAXSON DISCUSSION

Vicarious learning is clearly advantageous in unforgiving environments for the reasons stated above. If an agent is able to learn to avoid catastrophic situations without putting its life in danger, it has an advantage over an agent that is forced to risk its neck to learn this same information. By sacrificing a few agents initially, the vicarious agents were able to survive much longer.

In the forgiving environment, the situation is more complex. The vicarious learning agents do not learn faster initially because even though they vicariously experience the objects in the environment sooner, it still takes time to learn to approach or avoid the objects. By the time the agents have spent enough time in the env ronment to set their policy networks properly, they tend to have had an opportunity to directly encounter most of the objects.

What was more surprising is that the performance

of the V-MAXSON agents was much better than that of the MAXSON agents. Early in the learning runs of the MAXSON agents, one agent would interact with the food and water first. This agent would immediately begin to learn how to approach food and water, while the remaining agents had yet to discover these objects. The first "lucky" agent would begin to consume all the food and water as fast as it could, and leave none for the remaining agents. In the testing phase, the lucky agent would score very well, but the rest of the agents would score poorly, thus dropping the average score. The actions of peer agents made the environment dynamic, and necessitated faster learning.

In the V-MAXSON runs, the effect of one agent getting everything was avoided. When the lucky agent ate the food and water, it implicitly shared information about it with the rest of the agents. Thus, vicarious learning evens out the playing-field. It provides the less lucky agents with vicarious experience which enables them to catch up to their lucky peers, thus raising the total group score. This could be critical in environments that require cooperation for later tasks where a group of agents needs to survive, such as for later pack-hunting to bring down large prey.

With the introduction of vicarious learning, it is possible that an agent could incorrectly learn associations in the value network. The experiments show that this negatively impacts the agents initially, but that the agents can overcome the handicap. Furthermore, if the agent later vicariously learns the opposite of its incorrect earlier learning, the agent would proceed to learn a correct policy. One possible solution for this confusion is to allow the weights associated with the vicarious node to decay back to 0 over time, letting the agent forget the incorrect learning. This would be less optimal in unforgiving environments, since the agent would forget what it learned about the deadly poison and might eat it anyway. Otherwise, there is a trade-off in the speed of the decay of the event input. With a faster decay, there are fewer time steps available for confusion to occur. But if the decay is too fast, such that events always decay to 0 before the delayed reactions of the agents to food and poison, then the agents could never learn vicariously.

9 RELATED WORK

Work related to MAXSON falls into three categories: second-order networks as agent controllers, neurally-based reinforcement learning, and imitation learning.

9.1 Second-Order Controller Networks

Braitenberg (Braitenberg, 1984) 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 its environment. Werner (Werner, 1994) 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 to interrupt such behavior to avoid a danger or to take advantage of an opportunity to satisfy a less important goal. Once an agent has begun a consummatory behavior, it persists at it because 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 (1994) 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 12) similar to Werner's hand-designed structure, and weights that provide the agents with the above desirable properties. We are unaware of any systems that extend Hebbian learning to learn the weights on second-order or higher-order connections as we do in algorithm 1.

9.2 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 (Sutton and Barto, 1998). 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)- (Ackley and Littman, 1990) 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.

Araujo and Grupen (1996) described an agent architecture made up of a collection of Braitenberg-style controllers, each performing a simple operation (such as approach and avoid) in a task environment very similar to ours. They then used Q-learning to select the appropriate controller to use at each moment. Their technique does not address the way which the low-level controllers come about. In their experiments, the learning algorithm reached peak performance between 150 and 500 iterations of the algorithm. MAXSON, in contrast, learns the structure of the low-level controllers and takes an order of magnitudeless time.

9.3 Social Imitation Learning

The DRAMA architecture (Billard and Hayes, 1999) uses a technique similar to our value network in a mobile robot 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. DRAMA also performs social imitation learning (Billard and Hayes, 1997). 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 about actions. In further experiments (Billard and Dautenhahn, 1998; Billard and Dautenhahn, 1999), the imitating robot could also learn names for environmental attributes. The imitation helped ensure that the two robots shared similar input when learning the names of locations and directions. 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.

Cecconi et al. also developed agents that learned via social imitation (Cecconi et al., 1995). In their model, a child agent rides on the back of a parent agent so that both the parent and child receive identical sensory input. The child then uses the parent's action as a training signal for the child's neural network. By forcing the child's input to be the same as the parent's, they avoid the perception problem of one agent learning from another although they have different input vectors. In V-MAXSON, we address this problem through the event phase-based learning.

Kaminka and Tambe's agents learned by observing the actions of other agents in a team (Kaminka and Tambe, 1999). These agents determined that they were mis-behaving by comparing their actions with those of other members of the team, a strategy is more appropriate for groups of agents that have already learned how to perform a task, but may experience a planning failure. Models of the relationships between agents allow an agent to detect and correct an error. This team-based learning is different from vicarious learning in that, with the former, agents learn based on observations of the actions of other agents; while with the latter, agents learn based on both the actions of and the effects on other agents. Team-based learning can be powerful in specifying correct behavior to an agent, but it requires that the agents be cooperating on some task.

In imitation learning, the basic assumption is that the teaching agent knows better how to solve the task than does the learning agent. The learning agent compares its behavior with that of the teaching agent to make adjustments. In vicarious learning, the learning agent's behavior is not involved. Instead the learning agent observes the consequences of the teaching agent's actions. Both these learning techniques are useful when they are appropriate. Vicarious learning is not as powerful because, like reinforcement learning, the agent only receives a positive or negative signal evaluating an action. Imitation learning allows the

agent to learn the correct way of performing some task. In a situation where none of the agents are experts, imitation learning is less useful, while vicarious learning can afford a real advantage.

10 CONCLUSIONS AND FUTURE WORK

When agents need to learn about their environment in order 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. In the environments discussed in this paper, the agents are required to learn about their environment rapidly. For example, if an agent does not learn how to approach food when hungry within two interactions with food, the agent will die. Algorithms that require 40 interactions with poison in order to learn to avoid the poison cannot succeed in this environment. This sort of environment constitutes a different benchmark for animat learning, where rapid understanding is required for survival. This benchmark should help facilitate research into what kinds of architectures lead to the fast learning in what kinds of environments, as opposed to general purpose architectures.

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.

Our architecture as described here cannot learn to perform maze learning or other types of mapping-navigation tasks. We suggest that this is not a deficiency of the system but rather a demonstration of the fundamental differences in the two tasks. We believe that approach-avoid tasks should be performed by reactive systems, and maze navigation should be performed using an explicitly stored cognitive map, such as the one in Chao and Dyer (1999) or Lagoudakis (1999). Adding such a map to a reactive system enables an agent to learn a maze within a single exploration, as opposed to multiple explorations seen in TD and Q-learning (Russell and Norvig, 1995). This more closely matches the behavior of real animals in maze environments (Gallistel, 1990). Part of our future work is to integrate the maps of Chao and Dyer into the MAXSON architecture so that the agents will be able to navigate via the acquisition of explicit cognitive maps.

Another area of future work is to expand the learning to include learning to achieve *sequences* of goals by imitation of a teaching agent. (Crabbe and Dyer, 1999b) describes an architecture where second-order networks (structured like MAXSON networks) perform construction tasks that require the achievement of sequences of goals. Output from the sequence portions of the network activate various goal nodes, controlling the agent to meet these goals in sequence. We intend to enable learning in the sequence portion of the architecture as well. The result will be agents that learn how to approach and avoid objects on their own using MAXSON, and learn the appropriate sequence (in what order the goals should be met) via observation and imitation.

As we have seen, the MAXSON architecture can also be extended to perform other types of learning, such as vicarious learning. Agents that can learn vicariously have a survival advantage over agents that cannot. V-MAXSON agents can survive in unforgiving, instant-death environments, as well as help each other in more forgiving environments. The possible confusions that arise from vicarious learning can negatively affect the agent, but can also be overcome by additional experiences of the agent. We view V-MAXSON as merely a first step in the area of vicarious learning. Any system that can learn about objects and actions in the environment from the results of actions of other agents is learning vicariously. In this area there remain many open questions, including: How do agents internally represent actions of other agents? To what extent do animals in nature exhibit vicarious learning? is vicarious learning the basis of the development of communication? We envision further work in exploring the nature of this type of learning.

NOTES

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

units in (Cecconi and Parisi (1992).

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

ACKNOWLEDGEMENTS

We would like to thank the two anonymous reviewers, whose suggestions were especially helpful. We would also like to thank Rebecca Hwa, who made useful comments of several earlier drafts of this paper. This work is supported in part by an Intel University Research Program grant to the second author. The model runs Pentium III based computers, also donated by the Intel Corporation. Lisp code on random number generation was kindly provided by Wheeler Ruml of Harvard University.

REFERENCES

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

Araujo, E. G. and Grupen, R. A. (1996). Learning control composition in a complex environment. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pp. 333-342.

Billard, A. and Dautenhahn, K. (1998). Grounding communication in autonomous robots: an experimental study. *Robotics and Autonomous Systems*, *24*:71-79.

Billard, A. and Dautenhahn, K. (1999). Experiments in social robotics: Grounding and use of communication in autonomous agents. *Adaptive Behavior*, *7*(3/4).

Billard, A. and Hayes, G. (1997). Learning to communicate through imitation in autonomous robots. In *Proceedings of the International Conference on Artificial Neural Networks*, pp. 763-768.

Billard, A. and Hayes, G. (1999). Drama: A connectionist architecture for control and learning in autonomous robots. *Adaptive Behavior*, *7*(1):35-63.

Blumberg, B. M., Todd, P. M., and Maes, P. (1996). No bad dogs: Ethological lessons for learning in hamsterdam. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pp. 295-304.

Braitenberg, V. (1984). *Vehicles: Experiments in Synthetic Psychology*. MIT Press, Cambridge, MA.

Cecconi, F., Menczer, F., and Belew, R. (1995). Maturation and the evolution of imitative learning in artificial organisms. *Adaptive Behavior, 4*(1):29-50.

Cecconi, F. and Parisi, D. (1992). Neural networks with motivational units. In *From animals to animats: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pp. 346-355.

Chao, G. and Dyer, M. G. (1999). Concentric spatial maps for neural network based navigation. In *Proceedings of the International Conference on Artificial Neural Networks*, pp. 144-149.

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

Crabbe, F. L. and Dyer, M. G. (1999a). MAXSON: max-based second-order neural network reinforcement learner for mobile agents in continuous environments. Technical Report CSD-900009, UCLA.

Crabbe, F. L. and Dyer, M. G. (1999b). Second-order networks for wall-building agents. In *Proceedings of the International Joint Conference on Neural Networks.*

Crabbe, F. L. and Dyer, M. G. (1999c). Vicarious learning in mobile neurally controlled agents: The V- MAXSON architecture. In *Proceedings of the International Conference on Artificial Neural Networks*, pp. 904-909.

Crabbe, F. L. and Dyer, M. G. (2000). Goal directed adaptive behavior in second-order neural networks: Leaning and evolving in the maxson architecture. In Honavar and Patel, Eds., *Advances in the Evolutionary Synthesis of Neural System*. MIT Press.

Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. In *Fifteenth Interna- tional Conference on Machine Learning.*

Edelman, G. M. (1987). *Neural Darwinism*. Basic Books, New York.

Gallistel, C. R. (1990). *The Organization of Behavior*. MIT Press, Cambridge, Ma.

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

Gray, C. M., Konig, P., Engel, A. K., and Singer, W. (1989). Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties. *Nature*, 338:334-337.

Hebb, D. O. (1949). *The Organization of Behavior.* Wiley, NewYork.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*.

Kaminka, G. A. and Tambe, M. (1999). I'm ok, you're ok, we're ok: Experiments in distributed and central- ized socially attentive monitoring. In *Proceedings of the Third International Conference on Autonomous Agents* (Agents-99).

Lagoudakis, M. G. and Maida, A. S. (1999). Neural maps for mobile robot navigation. In *Proceedings of the International Joint Conference on Neural Networks*, Washington D.C.

Landmesser, L. (1987). Death of neurons during development. In Adelman, G., Ed., *Encyclopedia of Neuroscience*, pp. 303-304. Birkhauser.

McCallum, A. K. (1996). Learning to use selective attention and short-term memory. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pp. 315-324.

Oppenheim, R. W. (1985). Naturally occuring cell death during neural development. *Trends in Neuro- sciences*, 8:487-493.

Pfeifer, R. and Scheier, C. (1999). *Understanding Intelligence.* MIT Press, Cambridge, MA.

Ring, M. (1992). Two methods for hierarchy learning in reinforcement environments. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pp. 148-155.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence. A Modern Approach*. MIT Press, Cambridge, MA.

Shastri, L. and Ajjanagadde, V. (1993). From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings. *Behavioral and Brain Sciences, 16*(3):417-494.

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

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse code coding. In *Advances in Neural Information Processing Systems 8*, pp. 1038-1044.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning*. MIT Press, Cambridge, MA.

Thrun, S. and Schwartz, A. (1995). Finding struction in reinforcement learning. In *Advances in Neural Information Processing Systems 7*.

Werner, G. M. (1994). Using second order neural connection for motivation of behavioral choices. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pp. 154-161.

ABOUT THE AUTHORS

Frederick L. Crabbe

Frederick Crabbe is an Assistant Professor in the Computer Science Department at the United States Naval Academy. His current research interests focus on situated agents, and include rapid skill acquisition as well as social teaching and learning among groups of agents. He has worked at Sun Microsystems Labs, US Air Force's Rome Laboratories and Los Alamos National Labs. He received his Ph.D. in Computer Science from UCLA in 2000 as a member of the Artificial Intelligence Laboratory. He received his A.B. in Computer Science and Philosophy from Dartmouth College in 1992.

Michael G. Dyer

Michael G. Dyer received a Ph.D. in Computer Science at Yale University in 1982. afterwards he worked at Cognitive Systems, Inc. in New Haven, CT as a Senior Research Scientist. He joined UCLA in 1983 and founded an AI lab there. He is currently a full professor and has advised 16 (completed) doctoral students and numerous masters students.

He received a 1983 IBM Faculty Development Award, 1984 TRW Excellence in Teaching Award, and is on the editorial boards of the journals: Applied Intelligence, Connection Science, International Journal of Expert Systems, Knowledge-Based Systems, and Artificial Life. He is author of the book In-Depth Understanding (MIT Press, 1983) and has authored (or co-authored with his students) over 100 articles in conference proceedings, edited books and journals, in the areas of: natural language processing, cognitive modeling, naive mechanics, artificial life, adaptive agents, and neural networks. His current research interests involve building computational models of communicating and cooperating groups of agents and exploring how natural language processing relates to other cognitive skills, such as planning, reasoning, learning, perception, and creativity.