MAXSON: Max-Based Second-Order Neural Network Reinforcement Learner for Mobile Agents in Continuous Environments

Frederick L. Crabbe and **Michael G. Dyer**
Computer Science Department
University of California, Los Angeles, CA
Tech Report Number: UCLA-CSD-990009

Abstract

We present a neural network architecture (MAXSON) that controls a mobile agent in a continuous environment. MAXSON uses second-order connections to allow the agent to smoothly balance multiple goals. It generates its own novel reinforcements which it uses to learn to control the agent to approach useful objects in the environment while avoiding harmful objects. The learning performance of MAXSON compares favorably to two types of Q-learning.

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 the agents cannot be pre-programmed to meet these goals, they must *learn* to meet them. Reinforcement learning is a standard mechanism agents can use to learn to survive. Q-learning, a common type of reinforcement learning (Watkins 1989), can take time to converge and typically has no way to smoothly handle multiple goals. This paper presents a neural architecture that, for the task of learning to approach and avoid objects in its environment, learns faster than traditional Q-learning approaches, generates and applies the reinforcement in a neurally plausible manner, and can balance the requirements of multiple simultaneous goals.

Environment and Task

In this paper, 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 (Cecconi & Parisi 1993).

To detect external objects, the agent has a visual system. 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 (up to four degrees), and move forward up to six units[2]. 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 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.

Architecture

Our agents use an architecture, called MAXSON (figure 1), that is made up of two 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 and apply reinforcement to the policy network.

Second-Order Networks

A MAXSON-based agent uses the second-order policy network (figure 2) to choose what actions to take at each moment. In this network, the sensor nodes are broken into two groups: the external input and the

[2] One unit is one twelfth the agent's size.

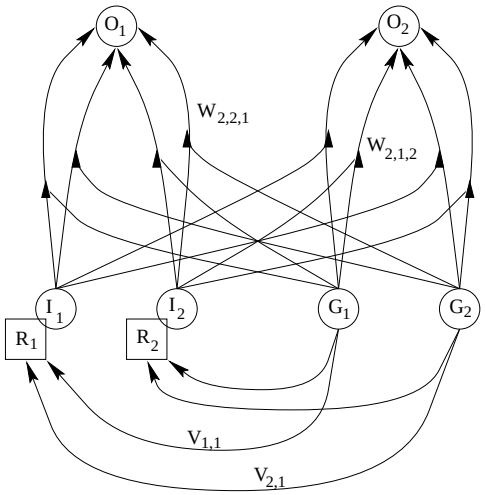

Figure 1: A small-scale MAXSON network. I-nodes receive external input. Associated with each I-node is a reinforcement node R. G-nodes receive internal goal input, and O-nodes perform motor actions. Each black triangle is a second order connection with a weight W.

goal sensors. The external input sensors ($\vec{I}$) consist of all the visual sensors (for food, water, and poison objects); the goal sensors ($\vec{G}$) consist of hunger, thirst, and pain. When the agent receives input ($\vec{I} \cup \vec{G}$), the associated sensor nodes are activated. Then, activation from each external input sensor (I_i - individual nodes are identified by a subscript) is multiplied with activation from each goal sensor (G_g) and a weight ($W_{i,g,o}$), via second-order connections (Giles & Maxwell 1987; Werner 1994), and then summed at the output node (O_o), as shown in Equation 1:

$$O_o = \sum_{i \in I, g \in G} I_i G_g W_{i,g,o}. \qquad (1)$$

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. 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

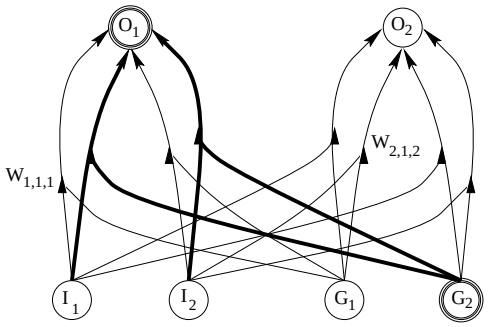

Figure 2: A second-order policy sub-network. G-nodes (goals) dynamically modulate the influence of I-nodes on O-nodes (actions) via second-order connections.

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 (these are highlighted via dark circles in figure 2). Then the weight on every second-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. Adjustment occurs only if the difference between the input on the external input sensor node I_i^t and the input at the previous time-step on the external input sensor node I_i^{t-1} is less than a threshold θ_1.

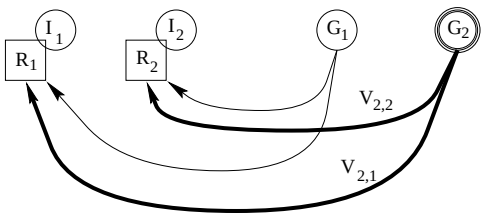

Figure 3: A first-order value network. Each R_i node holds the value of the reinforcement for the adjoining external input sensor node (I_i).

Reinforcement Calculation

Unlike traditional reinforcement learning algorithms, reinforcement in MAXSON is not pre-specified to the system from an external source. Instead, a first-order single layer network (called the *value network*) re-calculates the reinforcement 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 in figure 3) is multiplied by the value of 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).

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

Value Network Weight Adjustment

The value-network updates its weights V based on the temporal difference at the *goal* sensor nodes (algorithm 3). When the change of activation at an goal sensor node G_g is greater than a threshold θ_2, then the weight $V_{g,i}$ on the link (dark lines in figure 4) from that goal sensor node to the external input sensor node (dark circle in figure 4) that was maximally responding at the previous timestep I_i^{t-1}, is modified by the difference times the activation of the external input sensor node at the previous timestep.

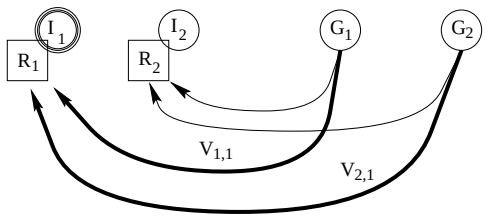

Figure 4: Update of weights V in value network.

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 specific difficulties in developing an efficient neural mechanism for learning the multi-goal task. This section describes these problems and how they were solved.

On-the-fly management of multiple goals. 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 (Crabbe & Dyer 1998). This can result in an agent that has multiple possibly conflicting goals that fluctuate in urgency over time (which is the task environment a MAXSON agent is designed to learn in). The second-order connections (combining the goal and external input sensor input) give the agent the ability to properly achieve these fluctuating goals. The second-order network allows 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 is turned on, 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. Instead

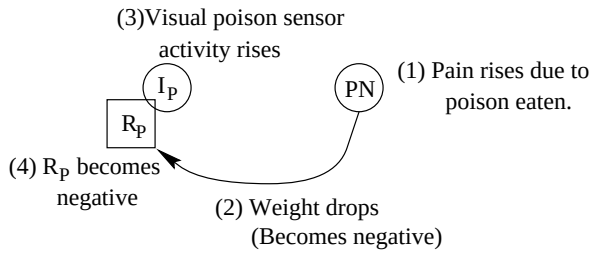

Figure 5: A value network example. PN is the pain input node, I_P is the visual node that receives input about a poison object. R_P is the reinforcement node for I_P.

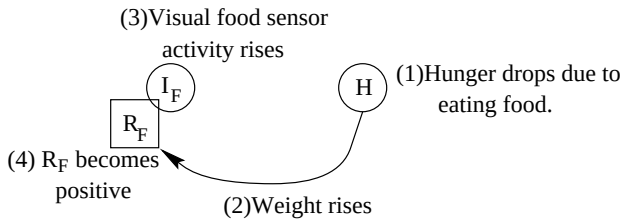

Figure 6: Another value network example. H is the hunger goal input node, I_F is the visual node that receives input about a food object. R_F is the reinforcement node for I_F. The agent has no a priori knowledge that food satisfies hunger.

the agent moves toward the food. But if the agent suddenly becomes extremely thirsty, the 'approach water' sub-portion of the policy network becomes active, and, all other things being equal, the agent approaches the water.

Delayed reinforcement converted to immediate reinforcement via cross-modal learning. The second-order policy network requires *immediate reinforcement* in order to learn. Unfortunately, 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 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 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 value on its goal pain sensor goes up (1). The change in value 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 reinforcement to learn how to avoid the poison.

In the case when the agent eats food (figure 6), the value on its goal hunger sensor goes down (1). The change in value 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. The policy network can use this immediate visual reinforcement to learn how to approach the food.

Using maximum nodes for focus of attention. During 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 value for 'turn left', and a medium value for 'turn right', the agent's resulting action is to turn left. If the agent receives a positive reinforcement, and the algorithm adjusts weights to *all* the output nodes, then the agent would adjust weights to both 'turn right' and 'turn left', and the effective behavior would not change. (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 is 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 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 example 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. Sim-

ilarly, 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 from hunger are modified, thus handling example 2.

Temporal difference based sensor threshold. Sometimes an atypical event occurs which could interfere with the agent learning how to behave in the typical case.

For example: If an agent that sees food on the left 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 events in the 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. For the experiments in this paper, θ_1 is set to 0.02.

Temporal difference based goal threshold. Because the reinforcement signal is not provided from outside the system, the value network must detect when an event has occurred that should warrant a change in the agent's 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 G_g whose change is greater than a threshold θ_2. The rationale for the 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 these times that the agent should take note of its surroundings. For all the experiments in this paper, the threshold constant θ_2 is set to 0.02.

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. A single agent was placed in the environment, and that agent received external visual inputs, but the goal input was turned off (The MAXSON policy network became a first-order network). The task was then to gather as much of the food and water as an agent could in 20,000 time-steps. 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 turned off. 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 agent's goal input was turned on. 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 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.

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.

The table-based Q-learner is based on the Q(λ) learner presented in (Sutton & Barto 1998). 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:

	FAL	FDL
$0.0 \leq x < .33$	×	
$.33 \leq x < .66$		×
$.66 \leq x \leq 1.0$		

...

PAR	PDR	H	T	PN
	×		×	
×				×
		×		

where P = poison, F= food, A = angle, L = left, R = right, H = hunger, T = thirst, and PN = pain.

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 (1995). 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'. The equation is shown in Equation 2.

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

The weights were adjusted using gradient descent, as described in Sutton & Barto (1998).

The MAXSON agent was as described in the architecture section above. In the third experiment we ran both a MAXSON agent with its goals turned on, and one with its goals turned off.

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 for 180,000 time-steps. Periodically, learning was stopped 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

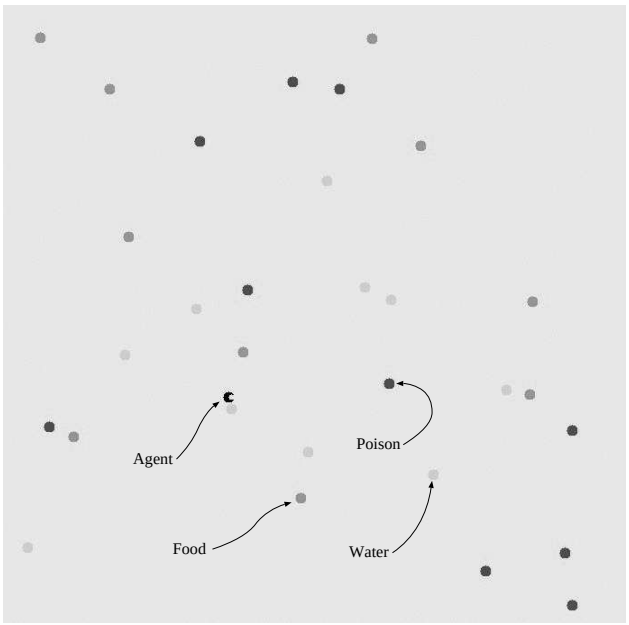

Figure 7: One of the random initial positions.

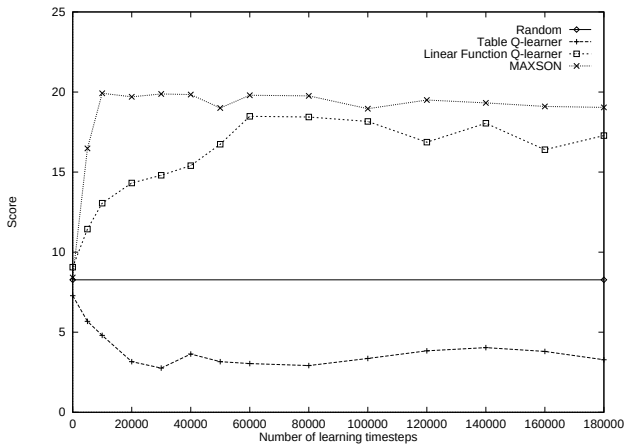

Figure 8: Experiment 1. The x-axis is the number of time-steps and the y-axis is the score. The MAXSON agents converge six times faster than the linear function Q-learner.

Results

The results of the first experiment 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 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 9 shows the results for the second experiment. 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 function 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. After this, 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 10 shows the results for the third experiment. 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.

Discussion

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. 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. Sec-

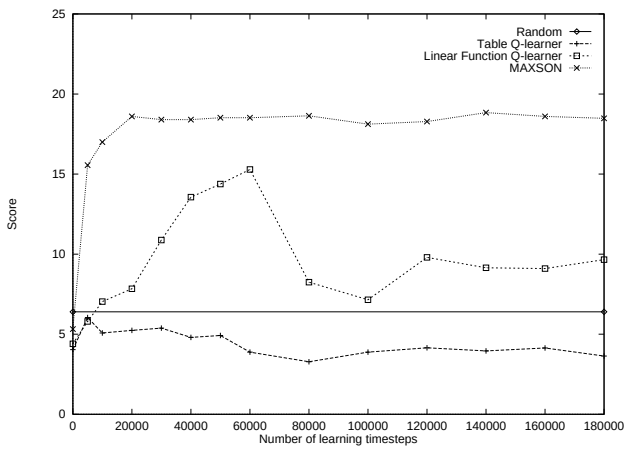

Figure 9: Experiment 2. The MAXSON agents converge to good score (18.4) in 20,000 time-steps. The linear function Q-learner reaches 15.29, but eventually converged on 10.

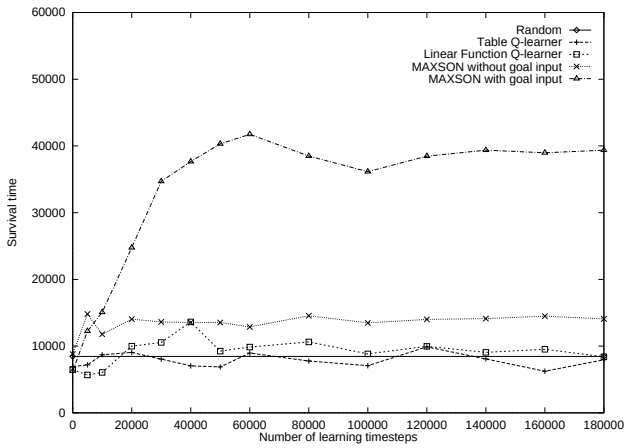

Figure 10: Experiment 3. The MAXSON agents that use their goal inputs survive the longest. Here, the y-axis is the survival time in number of time steps. Each data point indicates when an agent died (averaged over 10 trials).

ond, 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 avoids learning incorrect associations. Fourth, the cross-modal association that occurs in the value network converts a single 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 sees food.

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. Sixth, 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 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 3.

$$Q(s,a) = w_{1,1}E_1I_1 + w_{1,2}E_1I_2 + w_{1,3}E_1I_3 + \quad (3)$$
$$w_{2,1}E_2I_1 + ... + w_{12,3}E_{12}I_3.$$

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

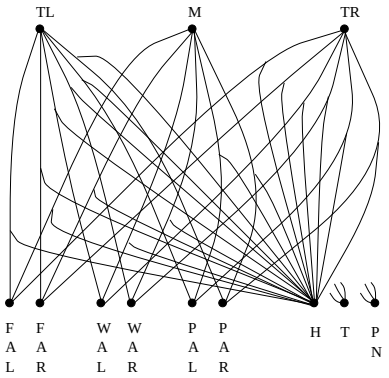

Figure 11: Initial policy network architecture with full connectivity and random weights in range of $[0, 0.01]$. TL = turn-left, TR = turn-right, PN = pain sensed, H/T = hunger/thirst sensed, FAL = angle to food object sighted on left, PAR = angle to poison object sighted on right, and so on. The distance visual sensor nodes were left out for clarity. Only connections from the hunger goal input node are shown.

Figures 11,12,13 illustrate how a typical MAXSON agent 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 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

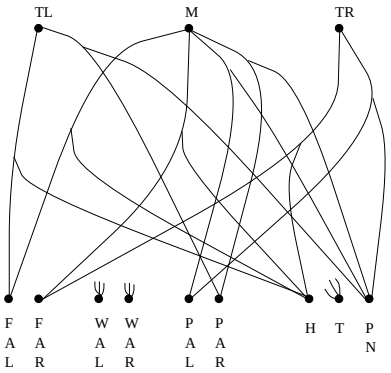

Figure 12: Simplified structure of a high-performance policy network after learning. All links from both hunger and pain with a weight greater than 0.01 are shown to highlight the learned structure. All links shown have weights greater than 0.5. Connections from thirst are omitted for clarity.

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 links between the water-sensors and the motor outputs while the sensation of Pain (PN) should control the links 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.

Figure 12 shows only the strongest connections. If the environment were to change then the weaker connections could be readjusted to adapt to the new environment.

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, 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 does this much faster and more flexibly than the more general Q-learning techniques.

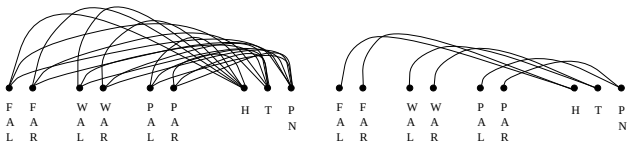

Figure 13: Value network before and after learning. Initially it has full connections, but it learns which sensors are more likely to be related to the achievement of appropriate goals.

Acknowledgments

This work is supported in part by an Intel University Research Program grant to the second author. The model is implemented in Allegro Common Lisp and runs under Linux on Pentium II based computers, also donated by the Intel Corporation. Some lisp code was kindly provided by Wheeler Ruml of Harvard University.

References

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

Crabbe, F. L., and Dyer, M. G. 1998. Cooperative behavior, communication and task decomposition in protective structure construction by neurally controlled agents. Technical Report CSD-980019, UCLA.

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

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

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

Watkins, C. J. C. H. 1989. *Learning from Delayed Rewards.* Ph.D. Dissertation, Cambridge University.

Werner, G. M. 1994. 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*, 154–161.