Vicarious learning in mobile neurally controlled agents: The V-MAXSON architecture.*

Frederick L. Crabbe Michael G. Dyer
cardo@cs.ucla.edu dyer@cs.ucla.edu
UCLA Computer Science Department, Los Angeles, CA 90095

Abstract

This paper extends MAXSON (a neural reinforcement learning system) to include *vicarious* learning. It shows that V-MAXSON can improve individual survival as well as promote survival of the group.

1 Environment and Task

This paper presents V-MAXSON, an extension to the MAXSON [3] neural architecture for mobile agents. The agents live in a two-dimensional, continuous environment where there are objects in the environment to be approached and avoided (called food, water, and poison). To survive, the agents need to eat the food, drink the water, and avoid the poison. V-MAXSON agents learn vicariously about their environment, i.e. they use observations of peers to learn about objects in the environment.

2 MAXSON

First we will present the MAXSON architecture, and then describe the V-MAXSON enhancements.

2.1 Input, Output, and Task A MAXSON agent receives goal input representing the agents internal states[1] (called hunger, thirst, and pain). 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.

To detect external objects, the agent has a front-facing 180 degree visual system. 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 which items should be approached or avoided, nor which items are suitable for satisfying which goals. The agent must learn to behave appropriately to satisfy these goals.

2.2 Network Architecture The neural network of a MAXSON agent is made up of two sub-networks: a second-order *policy network*, and a first-order *value network* (figure 1). The policy network is used to dynamically generate the agent's actions at each time-step, while the value network is used to generate and apply reinforcement to the policy network.

2.3 Second-Order Networks A MAXSON based agent uses the second-

*This work is supported in part by an Intel University Research Program grant to the second author. The model runs Pentium II based computers, also donated by the Intel Corporation. Lisp code on random number generation was kindly provided by Wheeler Ruml of Harvard University.

[1]The goal input corresponds to input coming from within the agent's body, similar to motivational units in [2].

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

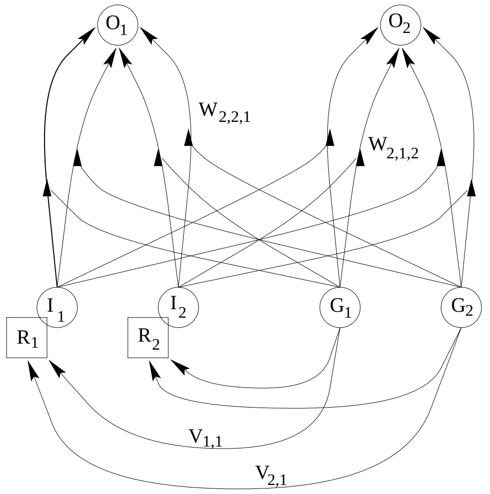

Figure 1: A small-scale MAXSON network. I-nodes receive external input. Associated with each I-node is a reinforcement node R. G-nodes receive internal goal input, and O-nodes perform motor actions. Each black triangle is a second order connection with a weight W.

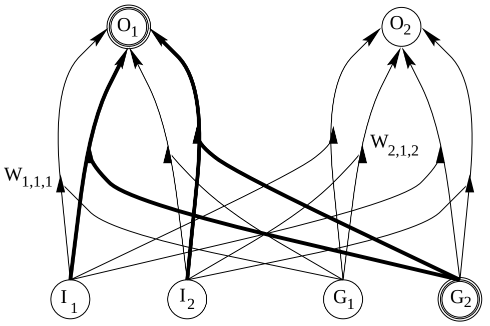

Figure 2: A second-order policy sub-network. G-nodes (goals) dynamically modulate the influence of I-nodes on O-nodes (actions) via second-order connections.

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 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, pain, and fear. 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 [4, 7], and then summed at the output node (O_o): $O_o = \sum_{i \in I, g \in G} I_i G_g W_{i,g,o}$.

2.4 Policy 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 policy network. The reinforcement R_i at external input sensor node I_i is continuously re-calculated by a function of the difference between the activation on I_i at the current time-step and the activation on I_i at the previous time-step: $R_i = f(I_i^t - I_i^{t-1})$. We describe the neural mechanism that dynamically generates this distributed reinforcement signal in the next section. The second-order weights in the policy network are adjusted as shown in algorithm 1.

---

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

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

$o \leftarrow \arg\max_o \vec{O}$
$g \leftarrow \arg\max_g \vec{G}$
for each i **in** I **do**:
$w_{i,g,o} \leftarrow w_{i,g,o} + R_i \times O_o \times G_g \times \delta_1,$

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

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 ex-

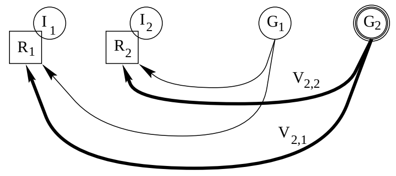

Figure 3: A first-order value network. Each R_i node holds the value of the reinforcement for the adjoining external input sensor node (I_i).

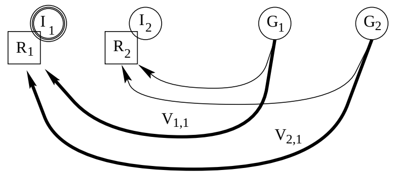

Figure 4: Update of weights V in value network.

ternal 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.

2.5 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*) recalculates the reinforcement at each timestep. It consists of connections from the goal sensors to the reinforcement nodes (R_i) 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**:
$R_i \leftarrow (I_i^t - I_i^{t-1}) \times V_{g,i} \times G_g$

2.6 Value Weight Adjustment The value-network updates its weights V based on the temporal difference at the *goal* sensor nodes (algorithm 3) and the external input. 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.

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:

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

2.7 MAXSON Features The MAXSON architecture exhibits several features: 1) The second order connections enable the agents' changing goals to modulate their behavior. As a result, agents learn to eat only when hungry and drink only when thirsty. 2) The value network converts sporadic reinforcing events into continuous reinforcement for the policy network. For instance, eating food creates a positive reinforcement on visual sensors so that seeing distant food now also is reinforcing. 3) By adjusting only the weights attached to maximal nodes, the attention of the learning is focused on those nodes that contributed the most to the situation, and reduces the number of times the credit was incorrectly assigned to the wrong weight in learning. As a result, MAXSON agents learn faster and perform better than Q-learning [6] agents. For more information, see [3].

3 V-MAXSON

The goal of the vicarious learning system is for one agent to learn from observing both another agent's actions and the result of those actions. An example is when agent A sees agent B eat a poison object. B grimaces, indicating that it did not like the poison. As a result A learns to avoid the poison without having experienced it itself. In order to enhance MAXSON to V-MAXSON, several features needed to be added: new external sensors and motor outputs, a small number of innate connections, a new internal goal node (the Vicarious or Vic node) and a synchrony-based learning mechanism.

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 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.

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 value 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 value of 0.2, and once it is changed, either up or down, it slowly returns back to 0.2.

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 about objects that are far away, this mechanism does not always work. 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 vicarious goal input. This is accomplished by binding events in the environment using temporal synchrony [5]. V-MAXSON visual input is split into a two phase process.

The first phase (the object phase) is exactly the same as the visual input to the MAXSON network. In the second 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 timestep 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.

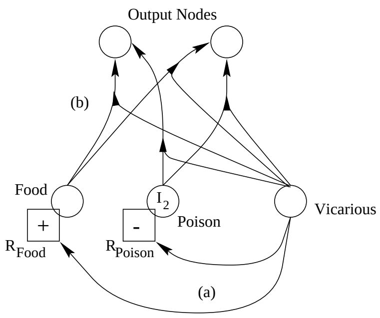

Figure 5: Portion of a V-MAXSON net learning vicariously. (a) weights between Vic node and food sensor nodes are increased due to perceived grins, causing positive reinforcement to be generated at sensor nodes. Perceived grimaces cause poison to become negatively reinforcing. (b) Reinforcement at sensor nodes causes learning on higher-order connections between sensor and motor nodes.

In V-MAXSON, the value net adjusts its 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. Because an agent sees 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. Agent A perceives the grin, which causes (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 eat 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 5). Once A eats food, its hunger drops, and the agent learns to associate food with hunger. Thus through the process of vi-

	MAX	V-MAX
Average	1656	6520
Average best	2301	17711

Table 1: The results of the first experiment. 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.

carious learning followed by reinforcement learning, agent A learned that to satisfy hunger, it should approach food.

4 Experiments/Results

To test V-MAXSON, we ran two 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 1 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, 1000, 2000, 3000, 4000, 5000, 10000, 15000, 20000, and 25000 time steps.

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 20000 time steps, and then given a score (the number of foods eaten plus the number of water units drunk, minus the number of poisons 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 25000 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 6 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 99.95% confidence interval using a t-test and 99.98% significant using a U-test [1].

5 Discussion

Vicarious learning is clearly advantageous in unforgiving environments for the reasons stated above. If an agent is able to

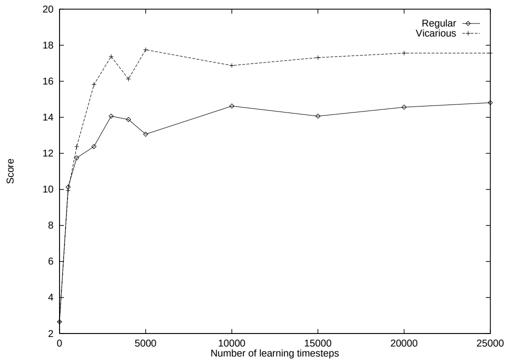

Figure 6: Experiment 2, comparing the performance of vicarious learning agents with non-vicarious agents. The x-axis is the number of time-steps the agent spent learning and the y-axis is the score. The average score of the vicarious learning agents is significantly higher than their non-vicarious counterparts.

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 environment 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 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.

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 is needed to survive, such as pack hunting to bring down large prey.

6 Conclusion

Agents that can learn vicariously have a survival advantage over agents that cannot. The MAXSON architecture, which outperforms Q-learning in a continuous multi-goal environment, has been extended to incorporate vicarious learning. V-MAXSON agents can survive in unforgiving, instant death environments, and help each other in more forgiving environments.

References

[1] V. Cangelosi, P. Taylor, and P. Rice. *Basic Statistics*. West Co., St. Paul, 1979.

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

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

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

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

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

[7] G. M. Werner. Using second order neural connection for motivation of behavioral choices. In *From animals to animats: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 154–161, 1995.