# Evolutionary Search, Stochastic Policies with Memory, and Reinforcement Learning with Hidden State

**Matthew R. Glickman** GLICKMAN+@CS.CMU.EDU
Computer Science Dept., Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213-3891

**Katia Sycara** KATIA+@CS.CMU.EDU
The Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213-3891

## Abstract

Reinforcement Learning (RL) problems with hidden state present significant obstacles to prevailing RL methods. In this paper, we present experiments conducted using a straightforward approach to solving such problems that trains artificial neural networks with recurrent connections to represent action policies using evolutionary search. We apply this method to two different benchmark problems. The first problem, involving maze navigation, has original features. Key among these features are that it is scalable, and thus provides a benchmark for investigating the performance of algorithms on problems with increasingly larger state spaces, and that it facilitates the study of inter-task transfer of search effort by providing for the generation of multiple, related tasks. The second problem, New York Driving, was introduced by McCallum (1995). Previously reported results for this task provide a basis for comparison with the evolutionary approach.

Singh et al. (1994) demonstrated that in RL problems with hidden state, the best *memory-less* policy may be a stochastic one. Of particular interest in this study is our finding that in practice, the ability to represent stochastic policies can significantly enhance the performance of evolutionary search for policies *with* memory. We explore this phenomenon via the use of recurrent networks composed of stochastic units as a means for representing policies.

## 1. Introduction

### 1.1 RL, Hidden State, and Policy-Space Search

In Reinforcement Learning (RL), an agent sequentially senses and acts upon its environment over time while receiving *rewards*. The goal is to learn behavior that maximizes some function of the reward over time. This behavior or *action policy* is typically expressed as a mapping from world states to actions. Instead of directly learning the action policy, the prevailing approach is to apply dynamic programming-based techniques to learn a *value function* which maps world states and actions to expected future rewards. Action policies are derived via value functions. For a large set of problems there exists a suite of very effective algorithms with theory that provides for specific theoretical guarantees (Kaelbling et al. (1996) provide a review).

RL problems have *hidden state* when task-relevant aspects of the world-state are not always observable at any moment, but must be inferred by also considering past observations. Such problems violate the Markov property on which the great majority successful algorithms depend. Such problems usually necessitate agents with some memory for storing features of past observations in order to optimize performance. Despite the difficulties that such problems present, some innovative algorithms have been presented for such problems. Some make use of a pre-specified model of the environment (*e.g.* Littman et al., 1995) while others are operable when relevant observations are known to extend no more than a fixed window of time into the past (*e.g.* Lin & Mitchell, 1992). One proposed algorithm which does not rely on such pre-existing domain information is McCallum's U-Tree algorithm (McCallum, 1995).

An alternative to the value-function approach is to search directly in policy-space via general optimization methods. Because such methods may fail to take advantage of the special structure of the RL domain, they can possibly suffer in comparison with value-function-based methods in terms of expected solution quality, computational efficiency, and/or theoretical guarantees. However, because there are exceptions to each of these cases (*e.g.* Moriarty et al., 1999; Baird & Moore, 1999) the role of direct policy-space search in RL remains an open issue. This is particularly the case where value-function based methods are not as readily applicable, as in problems with hidden state.

### 1.2 RANN's and EA's

In the experiments reported here, we have employed an Artificial Neural Network (ANN) with Recurrent connections (RANN) to represent action policies. Among the reasons why ANN's themselves are an appealing representation is because of their inherent parallelism and because their performance may gracefully degrade when they are perturbed, a property of significance for a hill-climbing search method such as the one we used. RANN's specifically are necessary in this case for their capacity to retain state-information over time.

RANN's are often difficult to train, particularly in the absence of explicit target outputs. We have had good success at training using a simple Evolutionary search Algorithm (EA). There are many variants of EA's (principal prototypes include Genetic Algorithms (Holland, 1975), Evolutionary Programming (Fogel et al., 1966), and Evolutionary Strategies (Schwefel, 1981)), but in general they may be characterized as stochastic hill-climbers that make use of a population of candidate solutions for search. Examples of the application of EA's to RL problems with hidden state include (Teller, 1994) and (Schmidhuber et al., 1997), both of which represent policies as programs in a chosen language.

While this study does provide further information on the use of EA's for both RL and training RANN's, it is primarily distinguished from others in three respects: (1) It introduces a new benchmark problem for RL with hidden state, the Start-Anywhere Maze (SAM) task; (2) it presents results from applying the EA to another large-scale hidden-state benchmark problem that was previously explored using a value function-based technique, providing for an informative comparison; and (3) it provides evidence for (a) the utility of learning stochastic action policies *with* memory for hidden state problems using (b) recurrent, stochastic networks as a policy representation.

### 1.3 The Start-Anywhere Maze task

The Start-Anywhere Maze (SAM) task is a scalable benchmark RL problem with hidden state. The principal difference between the SAM task and other maze problems that have been used to study problems of hidden state (*e.g.* Littman et al., 1995) is that rather than pertaining to a single, hand-designed maze, random mazes of variable scale are automatically generated. This feature permits the study of algorithm behavior in the face of increasingly large underlying state spaces. Moreover, the generation of differing mazes of the same (or different) scale provides multiple, related problem instances, which facilitate the study of issues such as "learning to learn" and/or the inter-task transfer of search effort.

### 1.4 New York Driving and U-Tree

In addition to the results from the SAM task, we present results from evolving RANN's to solve a task, New York Driving, introduced by McCallum (McCallum, 1995) in connection with a value-function-based approach to RL problems with hidden state, the U-Tree algorithm. While interacting with its environment, U-Tree uses statistical tests to construct an instance-based tree-structure that facilitates the selection of actions based upon observations made arbitrarily far in the past. The New York Driving task is one of the few hidden-state problems studied in the RL literature that has a true state space much larger than 100 states.

### 1.5 Learning stochastic action policies using stochastic RANN's

As shown by Singh et al. (1994), in the absence of memory, stochastic policies may be optimal in the face of hidden state. But the ability to represent stochastic policies may also be advantageous for a learning algorithm for other reasons, even with memory. Stochasticity may be of use when there is limited memory, or the memory capacity available is difficult to fully exploit. Stochasticity may also be useful for search, because it may allow an algorithm to escape from local optima. Our results show some substantial advantages from the ability to represent stochastic policies. To represent such policies, we employ recurrent, stochastic networks, composed primarily of units similar to those used in a Boltzmann machine (Ackley et al., 1985).

The SAM task, along with associated experiments, is presented in section 2, followed by the New York Driving task in section 3. The results and issues they raise are discussed in section 4, followed by a summary section 5.

## 2. The Start-Anywhere Maze Task

### 2.1 Description

We first presented the "Start-Anywhere" Maze (SAM) task in (Glickman & Sycara, 1999)). An instance of this task involves navigation in a static, randomly-

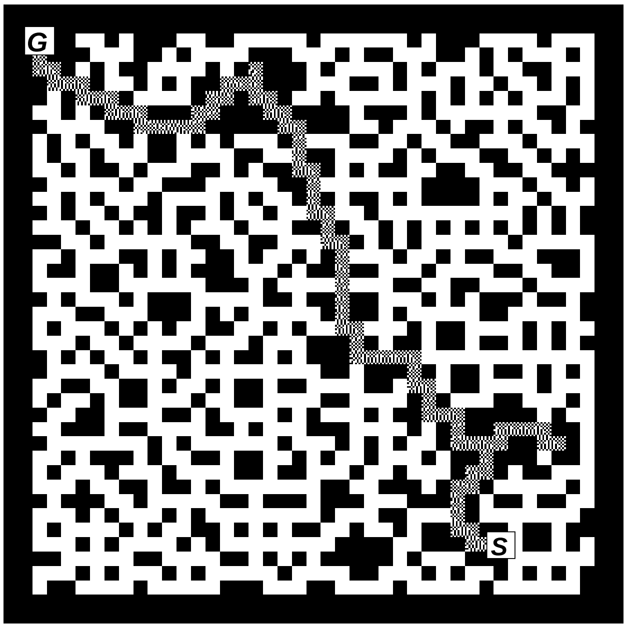


*Figure 1.* Navigating a typical 40×40 maze, starting from a randomly chosen start point (`S`) and proceeding to the goal point `G`, always in the upper, left corner of the maze.

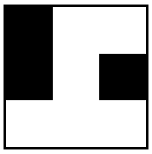

*Figure 2.* Example of an agent's field-of-view in a maze (note–orientation is significant).

generated maze[1] At each time-step, agents have a choice of three actions: (1) advance one step forward, (2) turn left and advance one step, or (3) turn right and advance one step. In many maze problems studied both in computer science and in animal behavior, the agent starts at the designated start point with the goal of finding a path–perhaps the most efficient path–to the designated goal point. In this case, an optimal policy could be specified via a fixed sequence of moves. However, in the task studied here an agent is rewarded according to how quickly it can find its way to the designated goal when started from *any* random point in the maze (*e.g.* figure 1).

If the agent were provided at all times with knowledge of its exact location in the maze (as is common among maze problems explored in RL), an optimal policy could be specified via a mapping from each location in the maze to corresponding move. Instead, the agent is provided with no such knowledge and only "sees" the eight squares immediately surrounding its position (as well as the square it occupies–see figure 2). It is this perceptual limitation that is responsible for the hidden state aspect of this problem.

Magnifying this difficulty is the fact that agents have not only a location, but an orientation (*e.g.* at any given point in time, it is facing up, down, left, or right). With different orientations, the same location can appear up to four different ways in the sensory field (agents have no sense of their "absolute" orientation, *i.e.* North, East, *etc.*). Each location can thus appear multiple ways, and each percept can correspond to multiple different locations. For example, the maze in figure 1 has 931 occupiable locations, which at 4 orientations each corresponds to 3,724 states, with only 98 distinct percepts. Thus, immediate sensory information alone is very unlikely to exactly identify where the agent is in the maze. Rather, the agent must begin by exploring, and progressively integrate observations to narrow in on a strategy to quickly and reliably reach the goal.

Another difficult aspect is that the agent only receives a reward if and when it reaches the goal; if, during a single trial, one agent manages to come within one step of the goal without actually reaching it while another never even gets close, the two receive equally poor scores. This aspect significantly limits the information available to the selection process for differentiating between alternate behavioral policies.

To select the more fit solutions, their quality must somehow be measured. We accomplish this by dropping each member of the population into the maze at each of a set of starting points and providing it with a fixed amount of time in which to reach the goal. Each such "opportunity to reach the goal" is known as a *trial*. For each generation, each individual's fitness or *training score* is calculated by summing up, for each trial, the square of the number of time-steps spent before reaching the goal. The higher its score, the less efficiently the given agent has managed to arrive at the goal. The set of starting points is randomly resampled from a uniform distribution over all starting points possible in the given maze. For 25×25 mazes, we used trials of length 400, and length 1000 for 40×40's. 10 starting points were selected at random for each generation, and hence each individual was scored on the basis of 10 trials.

### 2.2 The RANN's

Other studies applying evolutionary search to training RANN's (*e.g.* Angeline et al., 1994) frequently search the space of network topologies, possibly concurrently with the connection weight space. For reasons of simplicity, we chose a different approach: Simply choose a single, very powerful topology, and explore only in

[1] For those interested in studying the same or similar mazes to those used in the experiments reported here, the maze-generation algorithm is available from the authors.

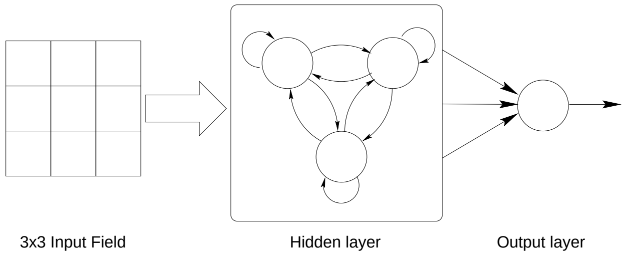


*Figure 3.* An Elman Network, consisting of a single hidden layer that is completely connected to its inputs (the 3x3 visual field), the output layer (a single unit), and itself.

connection weight-space. This arrangement yields a search space of fixed, finite dimensions.

Individuals in the population (of size 100) are encoded as vectors of real-values each corresponding to a connection weight in a network of functional units with a fixed, recurrent topology (an Elman (1990) network), consisting of an input layer (9 inputs, 1 for each of the pixel in the $3\times3$ visual field), fully connected to a single hidden layer, which is fully connected to both the single output unit *and to itself* (see figure 3). The recurrent connections in the hidden layer are effectively time-delayed by one time-step: All of the hidden units' outputs are determined in parallel, and thus they each see each others' output values from the previous time-step when calculating their current output value. By granting each hidden unit access to the activation levels of all the hidden units in the previous time-step, recurrent connections provide the capacity to retain state over time. What information is retained and how it is encoded is left to be determined via evolution.

We explored three network variants: those in which the hidden layers were entirely composed of either sigmoid, step-function (threshold), or stochastic units. In the stochastic units, as in the sigmoid units, a sigmoid function is applied to the weighted sum of the inputs. However, instead of directly taking the resulting value as the output of the unit, this value instead determines the probability of the unit producing a 1 instead of a 0 as output.

In all three variants, the output unit is sigmoid. If the value produced by the output unit is below a particular threshold, the agent turns left and tries to advance one grid-unit, if above a different threshold, to the right and forward one unit, and if between these two thresholds, straight ahead one unit. Unless otherwise noted, results presented here are for agents with a total of 12 hidden units, which yields a complete weight vector of 277 weights in length (including bias unit connections).

### 2.3 The Evolutionary Algorithm

An EA consists of two primary processes, selection and variation, that are alternately performed on a population of candidate solutions.

The specific form of selection used is *tournament selection*. Given a population of size $n$ and user-defined parameter, $k$ ($n = 100$ and $k = 10$ for our purposes here), $n$ "tournaments" are conducted whereby $k$ individuals are randomly chosen from the current population and the one with the best score is copied into the next population.

In the manner often favored by EA's such as Evolution Strategies (ES, Schwefel, 1981), and Evolutionary Programming (EP, Fogel et al., 1966) the variation process then simply mutates each new population member via the addition of Gaussian noise (sampled from a distribution of a fixed standard deviation defined by the user–0.2 in this case) to each weight. Again for simplicity, the EA used here does not employ any form of crossover operator.

### 2.4 Experimental Methodology

Many properties, such as the mean shortest-path-length to the goal over all potential starting points, the total number of potential starting points, the distribution of visible states along possible paths, *etc.*, vary from maze to maze of the same size. Consequently, what constitutes a "good" score for an agent may vary between mazes as well. To evade results biased for a particular maze, we've run experiments over multiple, differing mazes and summarized the results. However, instead of conducting each and every run on a different maze, we selected one fixed set of five randomly-chosen $25\times25$ mazes (and another of $40\times40$'s), and then conducted multiple runs on each maze in order to promote faster convergence of performance measures.

### 2.5 Results

Measures of performance for trained networks with hidden layers composed of step-function, sigmoid, and stochastic units, along with an agent that randomly chooses its actions, are shown in table 1. Each value is a mean for 20 networks. The 20 networks were trained on 5 $25\times25$ mazes, *i.e.* 4 separate networks were trained on each of the 5 mazes. Each network was tested from *all* possible start states in the given maze. For the five $25\times25$ mazes employed in these experiments, there are a mean number of 342.8 starting places, which together with orientation yield a mean of 1371.2 distinct states.

While the SAM task does appear to be challenging, we have not determined how well an optimal policy might perform. The mean number of steps to reach the goal required for a randomly wandering agent provides a form of lower bound. In an attempt to find

*Table 1.* Mean performance measures for an agent that chooses actions randomly, and for networks with hidden layers composed of step-function, sigmoid, and stochastic units. Coverage refers to the percentage of all possible start states from which the goal was reached in the allotted time, and is reported with 95% confidence intervals based upon the t-statistic. Mean steps to the goal was calculated for those start states from which the goal was successfully reached.

| Networks | Percent Coverage | Mean Steps to Goal |
|---|---|---|
| Random | N/A | approx. 8100 |
| Step | $94.2 \pm 2.4$ | 67.2 |
| Sigmoid | $94.7 \pm 3.0$ | 75.6 |
| Stochastic | $98.9 \pm 1.2$ | 73.6 |

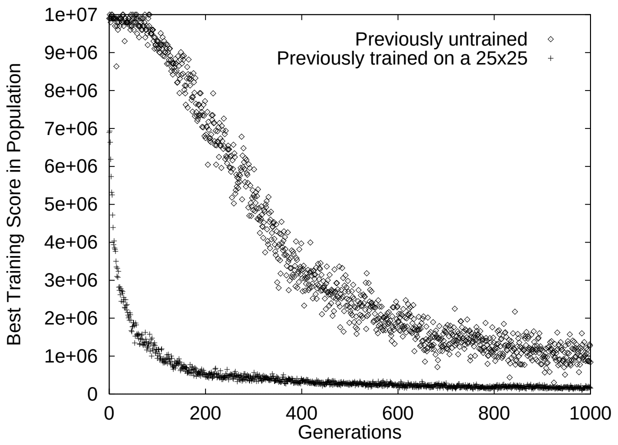


*Figure 4.* Comparison of mean population-best scores during training on a 40x40 maze for (1) starting from previously untrained networks, and (2) starting from networks previously trained on an unrelated 25x25 maze.

what the best memory-less policy might be, we have conducted experiments with the Sarsa($\lambda$) algorithm (shown by Loch and Singh (1998) to be relatively robust to problems with hidden state) on one the five mazes used here. The best-found policy in terms of coverage was able to reach the goal from only 60.53% of starting states, but by giving up on the rest of the states–presumably those furthest from the goal–when it did reach the goal it was able to get there in only 33.2 steps on average.

Enumeration of the number of distinct output vectors produced by the step-function hidden nodes provides a means to count the size of the internally represented state space arrived at by evolution. A step-function network can completely solve a $25 \times 25$ maze with about 100 states.

An intriguing property of evolving RANN's in this domain is depicted in figure 4. This figure shows the best training score (described in section 2.1) in the population during training of RANN's to navigate $40 \times 40$ mazes, averaged over several different runs on a fixed set of five mazes. The "previously untrained" runs were started with populations of weight vectors initialized to near-zero random values (as is standard). In contrast, the populations in the "previously trained" runs were copied directly from the final generation of a previous run on an unrelated $25 \times 25$ maze. Networks trained to navigate one maze in the initial runs were thus trained a second time on a different, much larger maze. The previously-trained runs adapt much more quickly and end up at a better fitness level than the first runs. We have found this effect to be quite robust.

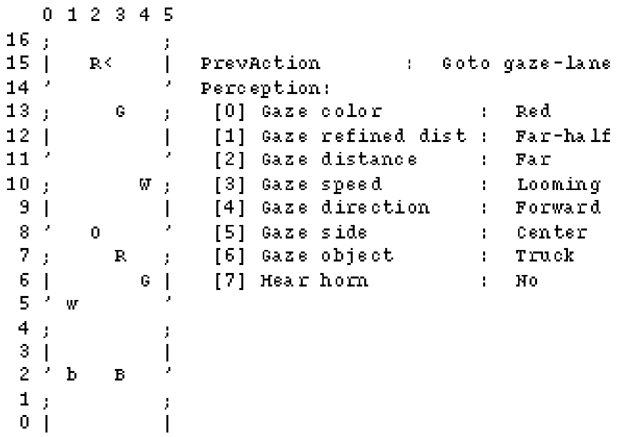


*Figure 5.* The New York Driving simulator showing a view from above of trucks traveling along a 4-lane highway (direction of travel is "up"). The agent is represented by `0`, slower trucks by other upper-case letters, and faster trucks by lower-case letter. The `<` indicates the object at which the agent is currently looking. (Reprinted with permission from (McCallum, 1996))

## 3. The New York Driving Task

### 3.1 Description

McCallum's New York Driving task (McCallum, 1995) simulates driving on a 4-lane highway (figure 5). The agent drives at a fixed speed along the freeway. Other trucks (introduced at each time-step with a probability of 0.5 into one of the four lanes) travel at one of two fixed speeds–one faster than the agent, one slower. Slower trucks first are introduced on the forward horizon and appear to move toward the agent as it catches up to them. Faster trucks are first introduced on the rear horizon and and also appear to move toward the agent as they catch up.

The agent is presented with a reward signal that penalizes both coming into contact with other trucks and impeding the progress of faster trucks. Contact with other trucks are designated as *collisions*, although they are not fatal[2] The agent and the other vehicle manage to "squeeze by" each other, and the agent receives a reward (or rather, a penalty) of $-5$. When the progress of a faster truck is impeded by the agent, it honks until the agent moves out of its way. For each honk, the agent receives a reward of $-1$. For each time-step in which the agent neither experiences a collision nor receives a honk, it receives a reward of 0.1.

There are four driving lanes, with a shoulder on each side. At each time-step, the driver has a choice of five actions, four of which select the direction in which the agent is looking ("gazing"): gaze-forward left, center, or right, and gaze-backward. The gaze-forward actions direct the driver's attention to the first object forward of its position in the lane to its left, the lane it is in, and the lane to its right, respectively. The gaze-backward action directs attention to the first object behind the agent in the same lane. The fifth action, goto-gaze-lane, moves the object into the lane where its gaze is directed.

The agent senses 8 features, 7 of which pertain to the object: gaze-object (truck, shoulder, or road), gaze-side (left, center, or right), gaze direction (forward or backward), gaze speed (looming or receding), gaze distance (far, near, or nose), gaze-refined-distance (far-half or near-half), and gaze-color (red, blue, yellow, white, gray, or tan). The eighth sensory feature is whether or not the agent hears a horn (a sign that a faster truck is stuck behind it).

### 3.2 RANN's and the EA

The RANN's are almost entirely the same as described for the SAM task. The state of each sensor is presented to the network as an integer, *e.g.* for gaze distance, 0 represents `far`, 1 is `near`, and 2 is `nose`. The only topological change to the network was to add two additional sigmoid output units to allow specification of the five possible actions. The first unit determines whether the agent will either perform a gaze action or move to lane where it is looking. If the agent is moving, then the output of the other units is irrelevant. Otherwise, the second unit determines whether the gaze is forward or backward. If the gaze is backward, then the third unit's output is irrelevant. Otherwise, the third output unit determines whether the forward gaze is to the left, center, or right.

The search algorithm is unaltered. The score of an individual in the population is its accumulated reward after 1000 steps in the simulator.

[2]Collisions between two faster and slower trucks (independent of the agent) are different in that the faster trucks subsequently disappear.

*Table 2.* Results after 5000 steps in the NYD simulator are shown for the highest-scoring evolved network, the means for those stochastic sigmoid networks which failed to become trapped in the low-scoring local optimum described at the beginning of section 3.3 (the proportions of successful runs are shown in parentheses), the best results reported from the U-Tree algorithm (McCallum, 1997), and the means for the aggregate of the remaining 7 out of 20 networks which did become trapped in the described local optimum. The intervals specified for scores are 95% confidence intervals based upon the t-statistic. "Colls" refers to Collisions.

| Networks | Colls | Honks | Score |
|---|---|---|---|
| Best | 26.7 | 97.0 | 123.6 |
| Stochastic ($\frac{9}{10}$) | 53.5 | 186.7 | $-245.3 \pm 179.9$ |
| Sigmoid ($\frac{4}{10}$) | 47.8 | 172.3 | $-172.1 \pm 225.5$ |
| U-Tree | 161.0 | 94.0 | -1229.5 |
| Failed | 854.4 | 371.6 | $-4193.1 \pm 450.4$ |

### 3.3 Results

On the New York Driving task, comparison with previously reported results (McCallum, 1997) derived from a distinctly different method permits a stronger assessment of the quality of the policies found.

Unlike in the maze task, some percentage of runs we conducted on this task were unsuccessful, in that the performance of the best networks found only moderately exceeded that of untrained networks. Examination of these policies revealed that they simply drove to the lane in which the least slow trucks appeared (the far left, *i.e.* the "fast lane") and then stayed there. The attraction of this very poor local optimum is presumably due to the fact that it manages to avoid some number of collisions, which incur a much higher penalty than honks, and requires only a minimally trained network.

The data in table 2 suggest that stochastic networks are much less likely to become trapped in this local optimum than sigmoid networks. However, a two-tailed Fisher's Exact Test yields a probability of 94.3% that these data (9 out of 10 vs. 4 out of 10) reflect a true difference in the success rate, so it is not possible to conclude this advantage for stochastic networks at the traditional 95% confidence level.

McCallum has reported scores on the NYD task from two U-Tree training sessions: one set for a single policy (McCallum 1995; 1996), and another, improved set (shown in table 2) in a later paper (McCallum, 1997) using a more directed exploration mechanism. On average, among successful runs, the evolved networks avoided many more collisions than the best U-Tree-derived policy, but tended to incur significantly more honks (table 2). Because collisions have a much higher penalty than honks, the actual reinforcement scores garnered by the networks were considerably higher than those of U-Tree[3].

At the same time, these higher-reward policies were found with a computational effort several orders of magnitude greater than that required by U-Tree. While U-Tree required only $3 \times 10^4$ steps in the simulator to arrive at the policy in table 2, the EA used a total of $5 \times 10^8$ steps (a population of size 100, each evaluated for $1,000$ simulator time-steps, over a total $5,000$ generations.

In terms of state-space size, McCallum (1996) refers to one successful NYD policy learned by U-Tree that has 143 leaves, but it is not strictly clear whether this is representative. Enumeration of the vectors of hidden-unit outputs reveal that the RANN's seemed to embody NYD policies that have far fewer states than either the reported U-Tree policy or the policies learned for the SAM task, *e.g.* the best policy found exhibits only 16 states.

## 4. Discussion

### 4.1 Benefits of Stochastic Networks

Two reasons the ability to represent stochastic policies may enhance the performance of search for action policies with memory are:

1. Permitting search through otherwise unrepresentable regions of policy-space provides smoother paths to more optimal regions.
2. With limited memory, or when full memory capacity is difficult to exploit, stochastic policies may be optimal

In the case of the New York Driving, the first reason appears to be operative because the benefit of stochastic networks would appear to be to boost the likelihood of finding a high quality ("successful") policy (although the current data do not permit this conclusion at the 95% confidence level ). The fact that the mean score of successful runs with sigmoid networks is not significantly different than that for stochastic networks would tend to suggest that the second reason given is less of a factor for the NYD task.

For the SAM task, the first reason is also likely to at least partially explain the observed results, but there is greater support for the second reason.

(Singh et al., 1994) shows there exist problems with hidden state where the optimal memoryless policy is stochastic. In such problems, the inability to distinguish between 2 world states may be arbitrarily costly for an agent. It should thus be possible to construct problems where the inability to distinguish between $n$ world states may be arbitrarily costly. Under such conditions, even with limited memory the optimal policy may be stochastic.

Note that RANN's used in this study have (for practical purposes) limited memory, and thus are limited in the number of state distinctions they may practically make. Furthermore, the adaptive landscape may be such that configurations which permit large numbers of state distinctions (or particularly difficult ones) are difficult to achieve, effectively limiting memory capacity even further.

The possibility that the best achievable policies may be stochastic seems more likely in case of the SAM task not only because the mean score of policies found with stochastic networks is highest, but also because successful policies appear to require more internal states than those for the NYD task. Stochasticity, in effect, makes it possible to "fake" a state distinction via a coin flip, and may be important to help RANN's achieve the number of state distinctions necessary for the best performance.

### 4.2 Efficiency

The EA appears effective, but its effective utility is dependent upon whether or not the training time is affordable. In situations where the cost of evaluating policies is at a premium, more RL-specific methods (such as U-Tree) are most likely to be preferable.

However, even when such domains are the ultimate goal, the EA/RANN method may be very useful for the refinement of the target algorithm. For instance, this method has revealed that there exist relatively simple policies in the NYD domain that achieve much higher reward than those found by U-Tree. It would now be interesting to determine under what conditions U-Tree itself could find such policies.

While the apparent cost in redundancy from searching with a population may appear to be high, we found that evolutionary search was vastly more effective than simple, non-population-based hill-climbers. Moreover, the cost of evaluating the population is trivially parallelized, which on 100-node system would reduce the wall-clock time by two orders of magnitude.

The observed benefit from solution re-use in training time and solution quality is also of great interest, particularly for scaling up to increasingly larger problems. Training for a particularly large-scale task might be best achieved via training on a series of well-chosen related tasks of increasing complexity (*i.e.* a learning curriculum (O'Sullivan, in prep)).

## 5. Summary

These experiments have provided testimony that a simple evolutionary search algorithm can (1) effec-

[3] We calculated the U-Tree reinforcement scores from the reported number of collisions, honks, and time-steps

tively train RANN's under difficult conditions, and (2) find action policies that effectively solve RL problems of challenging scale and with significant hidden state.

For the New York Driving task, previously reported results obtained via a value-function-based approach (U-Tree), revealed that for a much higher investment in training time, the EA/RANN approach found policies with significantly higher expected rewards. Such an approach would thus appear suitable in cases where the higher computational cost is affordable in relation to the potential benefit of higher quality policies.

Most hidden-state problems studied in the literature (New York Driving being an exception) have an underlying state space of less than 100 states. The SAM task described herein provides an additional hidden-state benchmark that is quite straightforward, trivially scaled to arbitrarily large state spaces, and supports the study of training on sets of related tasks.

On both of the problems to which we have applied this method, we have found the best results by using stochastic RANN's, networks that have a hidden layer that is composed of stochastic units. The ability of these networks to represent stochastic policies seems to transform the policy search-space in way favorable to the performance of EA's and perhaps other hill-climbing search algorithms.

Other possible applications of this method include calibrating the performance of other methods in development. The observed benefits of re-using previously trained solutions on new, related tasks are intriguing for use in scaling up to increasingly complex tasks.

## Acknowledgements

The authors wish to thank Andrew McCallum for the design and implementation of the New York Driving simulator as well as use of figure 5, James Thomas and Corey Kosak for many helpful discussions, and 3 anonymous reviewers and the area chair. This research was partially supported by the Office of Naval Research, Contracts N00014-95-1-0591 and N00014-96-1222.

## References

Ackley, D., Hinton, G., & Sejnowski, T. (1985). A learning algorithm for boltzmann machines. *Cognitive Science*, *9*, 147–169.

Angeline, P. J., Saunders, G. M., & Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, *5*, 54–65.

Baird, L. C., & Moore, A. W. (1999). Gradient descent for general reinforcement learning. *Advances in Neural Information Processing Systems 11*.

Elman, J. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial intelligence through simulated evolution*. New York, NY: Wiley.

Glickman, M., & Sycara, K. (1999). Evolution of goal-directed behavior from limited information in a complex environment. *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*.

Holland, J. H. (1975). *Evolution in natural and artificial systems*. Ann Arbor: U. of Michigan Press.

Kaelbling, L., Littman, M., & Moore, A. (1996). Reinforcement learning: A survey. *J. of Artificial Intelligence Research*, *4*, 237–285.

Lin, L.-J., & Mitchell, T. M. (1992). Reinforcement learning with hidden states. *Proceedings of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats*.

Littman, M. L., Cassandra, A. R., & Kaebling, L. P. (1995). Learning policies for partially observable environments: Scaling up. *The Proceedings of the Twelfth International Machine Learning Conf.*.

Loch, J., & Singh, S. (1998). Using eligibility traces to find the best memoryless policy in partially observable markov decision processes. *Machine Learning: Proc. of the Fifteenth International Conference*.

McCallum, A. K. (1995). *Reinforcement learning with selective perception and hidden state*. Doctoral dissertation, University of Rochester.

McCallum, A. K. (1996). Learning to use selective attention and short-term memory in sequential tasks. *From Animals to Animats 4: Proc. of the Fourth Intl. Conf. on Simulation of Adaptive Behavior*.

McCallum, A. K. (1997). Efficient exploration in reinforcement learning with hidden state. *AAAI Fall Symposium on Model-directed Autonomous Systems*.

Moriarty, D., Schultz, A., & Grefenstette, J. (1999). Evolutionary algorithms for reinforcement learning. *J. of Artificial Intelligence Research*, *11*, 199–229.

O'Sullivan, J. (in prep.). *Curriculum-based learning*. Doctoral dissertation, Carnegie Mellon University.

Schmidhuber, J., Zhao, J., & Schraudolph, N. (1997). Reinforcement learning with self-modifying policies. In S. Thrun and L. Pratt (Eds.), *Learning to learn*, 293–309. Kluwer.

Schwefel, H. P. (1981). *Numerical optimization of computer models*. Chichester: Wiley.

Singh, S., Jakkola, T., & Jordan, M. (1994). Learning without state-estimation in partially-observable markovian decision processes. *Machine Learning: Proc. of the Eleventh International Conference*.

Teller, A. (1994). The evolution of mental models. In K. Kinnear (Ed.), *Advances in genetic programming*. MIT Press.