Tuning Q-Learning Parameters with a Genetic Algorithm

Ben E. Cline

September 2004

Abstract

The Pond simulator provides a means of studying agents that must learn to survive in either static or dynamic environments. In this study, a genetic algorithm is used to tune the learning parameters of Q-learning agents in an environment with impediments to both survival and learning. Variations on the tuning experiments included using deterministic and nondeterministic Q-learning algorithms, static and dynamic environments, and groups of homogeneous and heterogeneous agents. Tuning improved survival rates in all the experiments.

Introduction

In the Q-learning algorithm [Watkins 1989], agents learn based on the state of the environment and a reward value. Unlike supervised learning, the agents are not told which outcomes are desired. Instead, they learn by taking actions and using feedback from the environment. The Q-value Q(s,a) in Q-learning is an estimate of the value of future rewards if the agent takes a particular action *a* when in a particular state *s*. By exploring the environment, the agents build a table of Q-values for each environment state and each possible action. Except when making an exploratory move, the agents select the action with the highest Q-value.

The Q-learning algorithm with an ε-greedy policy with inertia[1] has three parameters: the learning rate (α), the discount factor (γ), and the ε-greedy parameter. (α and γ are between 0.0 and 1.0, and ε is usually small.) The learning rate parameter limits how quickly learning can occur. In the Q-learning algorithm, it governs how quickly the Q-values can change with each state/action change. If the learning rate is too small for the task at hand, learning will occur very slowly. If the rate is too high, then the algorithm might not converge, i.e., Q-values might not stabilize near values that represent an optimal policy.

The discount factor controls the value placed on future rewards [Sutton & Barto 1998]. If the value is low, immediate rewards are optimized, while higher values of the discount factor cause the learning algorithm to more strongly count future rewards.

The value of ε is a probability of taking a non-greedy (exploratory) action in ε-greedy action selection methods. A non-zero value of ε insures that all state/action pairs will be explored as the number of trials goes to infinity. In a greedy method, $\varepsilon = 0$, the algorithm might miss optimal solutions; however, in a near greedy method, a value of ε that is too

[1] This is not a standard term; however, its meaning is described in this section.

high will cause agents to waste time exploring suboptimal state/action pairs that have already been visited.

When an agent is not making an exploratory move, it selects the action with the highest Q-value for the state. The action selected in the case where more than one Q(s, $a \in A$) has the maximum Q-value for the state (where *A* is the set of all actions) is based on inertia in this study. One of the actions, a_i is selected, and, on subsequent moves, a_i continues to be selected until Q(s,a_i) is no longer one of several Q-values with the maximum value for the state.

The heart of the Q-learning algorithm is the following formula that is evaluated each time an action is selected:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \lambda \max_{a'} Q(s',a') - Q(s,a)]$$

The agent decides on an action *a* based on the current state *s* of the environment and the ε-greedy policy. After performing the action, the new state is s', and the agent is given reward *r*. The $\max_{a'}$ term selects the maximum Q-value for state s' and $a' \in A$. This version of the algorithm has been proven to converge for deterministic Markov Decision Processes (MDP) and, as indicated by this study, it sometimes converges in environments that don't have this property.

There is a nondeterministic version of Q-learning which is guaranteed to converge in systems where the reward and state transitions are given in a nondeterministic fashion as long as they are based on reasonable probability distributions. It would appear that this algorithm would be suitable for the Pond environment as explained below; however, the deterministic algorithm finds better solutions faster for the evolved cases in this problem set. The equation for the non-deterministic algorithm is given by Mitchell [Mitchell 1997] based on work by Watkins and Dayan [Watkins & Dayan 1992]:

$$Q_n(s,a) \leftarrow (1-\alpha_n)Q_{n-1}(s,a) + \alpha_n[r + \max_{a'} Q_{n-1}(s',a')]$$

where

$$\alpha_n = \frac{1}{1 + visits_n(s,a)}$$

$Visits_n(s,a)$ is the number of times at step *n* that action *a* has been selected from state *s*. The Q-values are updated more slowly than in the deterministic version of the algorithm, and, unlike the deterministic version of the algorithm, the updates are smaller as *n* grows.

The Simulation Environment

There are three types of Pond[2] objects used in this study. There are agents that have Q-learning controllers, red veggies, and blue veggies. In the static version of the Pond, the agents gain energy when they eat red veggies and lose energy when they are poisoned by eating blue veggies. In the dynamic version of the Pond, the roles of red and blue veggies change gradually approximately half way though the simulation. At the beginning of each iteration during the environment change, the nutritional value of red veggies is decremented by one, and the nutritional value of the blue veggies is incremented by one. This change continues until the nutritional value of red veggies is reduced from +5 to -5 while the nutritional value of blue veggies increases from −1 to +8.

During each iteration of the simulator, Pond objects are selected two at a time without replacement and interacted. If one or both objects are agents, they are given a state value and produce an action. The results of the action, new states and rewards, are given to the agents as input to the Q-learning algorithm. If both objects are veggies, no state change occurs. If two agents interact, neither performs an action. During each iteration time step, 600 pairs of objects are interacted.

Each agent can have an energy value of 0 to 128 with an initial energy of 128. Each time an agent is activated, one energy unit is subtracted (called the "move penalty"). Each time an agent eats a red veggie, it gains 5 units of energy. Eating a blue veggie reduces the energy of the agent by 1. (As noted above, the role of the red and blue veggies gradually change half way through the simulation in the dynamic version of the Pond.) Agents cannot sense their energy value. Instead, the state description contains a health value that takes on the values *energy very low, energy low, energy medium,* and *energy high*. Furthermore, agents are only given a non-zero reward when their health value changes: +1 for moving to a higher health level, -1 for moving down a health level. Because the reward is generated by the agent based on its health value, no external source of reward is needed.

The veggies have a growth delay of 15 time units after being eaten. If an agent eats a growing veggie, it receives no change in energy from the veggie, but it still pays the one energy unit move penalty, and the veggie growth delay resets to 15. The agent cannot sense if the veggie is growing.

The consequence of the health level mechanism introducing a delayed reward and the veggie growth delay is that rewards are nondeterministic relative to the agents' senses. Eating the poison veggie repeatedly will produce a zero reward until the energy health threshold is passed. Then the agent will receive a -1.0 reward. Similarly, eating a beneficial veggie will produce a zero reward until a higher health level is entered. Then, the agent will receive a +1.0 reward. If the agent is at the energy threshold for passing to a lower energy level and it eats a beneficial veggie that is growing, it will receive a −1.0 reward and go to a lower health level because of the one energy unit move penalty and

[2] The Pond simulator is written in Java and was developed to study broader experiments involving cooperating learning agents.

the fact that growing veggies don't provide nutrition. In the dynamic Pond, the changing environment presents more ways for the rewards to be nondeterministic.

It should also be noted that no positive rewards are given at the highest health state because positive rewards are only given when an agent moves to a higher health state. Similarly, in the lowest health state, no negative rewards are given since negative rewards are given only when an agent enters a lower health state. Agents that have zero energy die of starvation and are removed from the list of objects in the Pond.

Besides the difficulty in learning because of the infrequent and nondeterministic rewards, the resources in the environment are not enough to support the 1200 agents that initially populate each simulation run. There are 900 red veggies and 900 blue veggies in the Pond. Due to the stochastic nature of the environment, the limited number of nutritional veggies, and the time required to learn, the number of surviving agents at the end of a run is small. Using a hand-coded agent that eats only red veggies and is run in the static environment, the average number of survivors is 297. Less greedy controllers failed to produce an improvement over this value.

Table 1 shows the meaning of the 8-bit state value. The agents do not "know" what the bits represent. The state value is only used as an index into the Q-table.

State Bits	**Description**
Agent's health (2-bits)	Energy value quantized as health
Attachment object (2-bits)	Object adjacent to agent (agent or veggie). (In other experiments, the attachment object can be NOTHING or OTHER.)
Attached object's energy or color (2-bits)	Health of agent or color of veggie
Communication bits (2-bits)	Always zero in this environment

Table 1 - Agent state information

Each time an agent is selected for a move, it is given the 8-bit state value, and based on Q-learning and its policy, it selects one of two actions: don't eat or eat. Attempting to eat another agent produces no result other than the one energy unit move penalty. After the action is processed, the agent is given an updated state and reward. It uses this information to update its Q-Table.

Agents and veggies have virtual location. One can think of the agents and the food floating in a pond. When two objects are selected randomly for interaction, it is as if they were floating into each others' vicinity. This technique removes the agents' need to plot moves in either 2 or 3 dimensions. (A planned version of the Pond will support both real and virtual locations.)

Tuning Learning Parameters

Aljibury and Arroyo [Aljibury & Arroyo 1999] describe an attempt to evolve learning parameters for a Q-learning controller for a simulated mobile robot. They had problems with convergence and indicated that the effectiveness of a set of learning parameters was sensitive to minor changes to the environment.

In this study, I used a genetic algorithm (GA) to tune the learning parameters of agents. Tuning experiments were performed over a set of variations in the learning algorithm, the environment, and the configuration of the GA. A summary of the variations follows:

- Deterministic and nondeterministic Q-learning algorithms
- Static and dynamic environments
- GA populations of heterogeneous and homogeneous agents

In both the heterogeneous and homogeneous agent experiments, each chromosome in the GA represents values of the learning parameters γ, α, and ε. In the heterogeneous case, the value in each chromosome in a population is given to a single agent, and the population of agents is run in the Pond simulator. The fitness of surviving agents and their corresponding chromosomes is related to the energy values of the agents. (The evaluation function peaks around 90 units of energy.) A GA population is evaluated with one simulation. Figure 1 shows the mapping between chromosomes and agents in the simulation for one GA population.

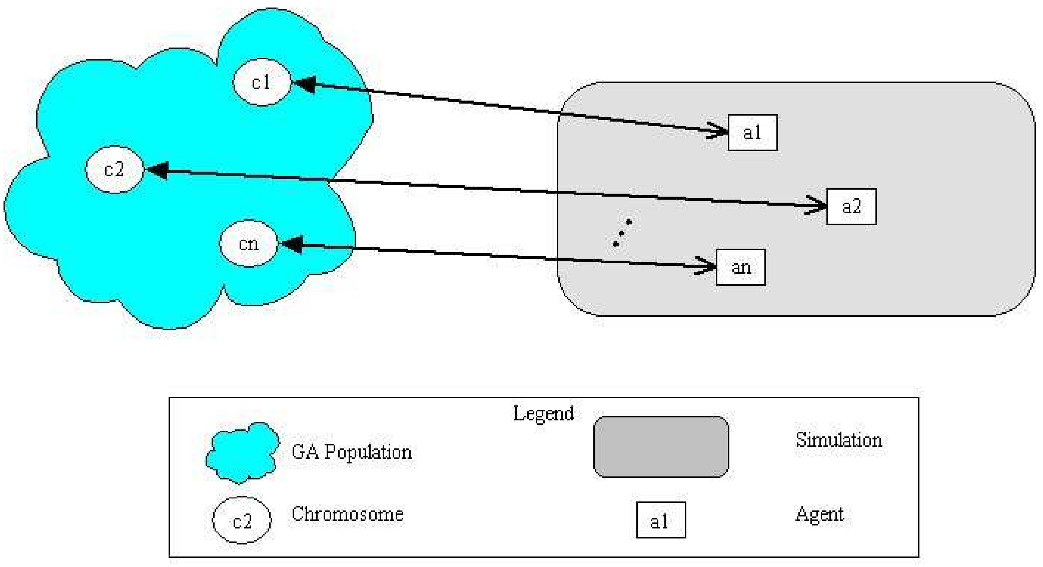

Figure 1 - Chromosome to agent mapping in the heterogeneous version of the GA

In the homogeneous case, the learning parameters from a particular chromosome are replicated on 1200 agents, and these agents are run in a simulation. The fitness value of the chromosome is directly proportional to the number of surviving agents. To evaluate a GA population, a complete simulation is run for each chromosome. Figure 2 illustrates the mapping between chromosomes and populations. With the population size of 30 used in the homogeneous GA runs, 30 simulations were required to evaluate a single population in the GA.

There are three primary differences in these two cases. First, the amount of CPU time needed to evaluate a population of homogeneous agents is much greater than evaluating a population of heterogeneous agents. Second, the fitness functions for both styles of genetic algorithms measure different quantities. Third, since the fitness of individuals is used by the GA in the heterogeneous case, this case more closely mirrors natural evolution. The latter difference is not significant as the GA is being used as a software search tool.

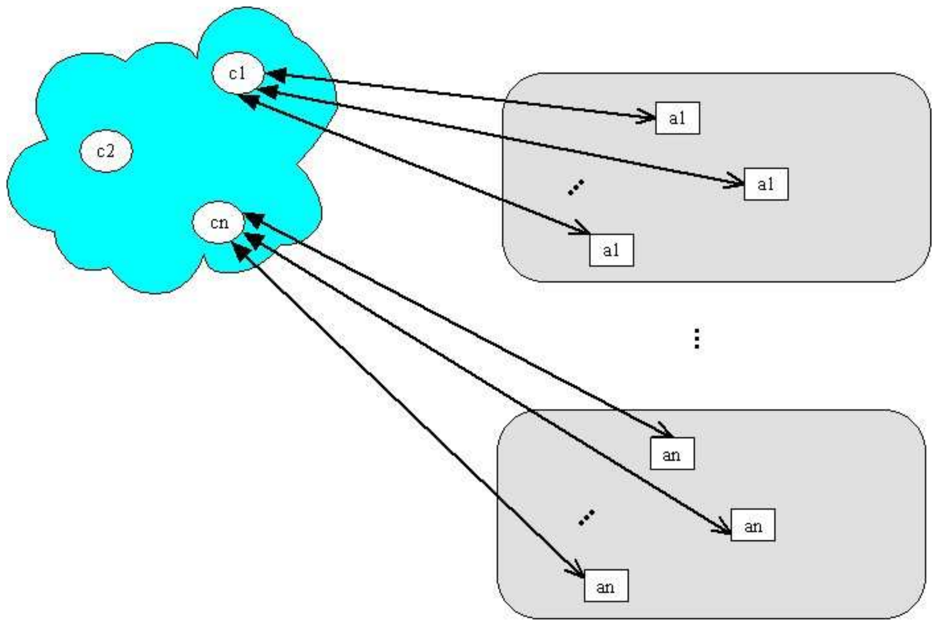

Figure 2 - Chromosome to agent mapping in the homogeneous version of the GA. See Figure 1 for the legend.

The goal of these experiments is to find good learning parameters such that the survival rate of a group of agents with the same learning parameters approaches the survival rate of a group of hand-coded agents that makes good decisions. The GA using homogeneous agents uses a fitness function that is directly related to the survival rates of agents based on one instance of learning parameters. The GA using heterogeneous agents has a fitness function that is indirectly related to improving survival rates as it measures the health of surviving agents. In both cases, populations of agents are simulated using the Pond simulator for 30,000 time steps.

Sutton and Barto [Sutton & Barto 1998] suggest starting values of α=0.1, γ=0.1, and ε=0.01 for several Q-Learning examples. I used the survival rate of simulations using these parameters, which I call the standard parameters, as the basis for comparing simulations running evolved parameters taken from the GA runs. These values are a reasonable starting point for some problems. With experience, one could perhaps guess at better parameters for a particular learning environment; however, with the GA approach, I hope to reduce the guesswork in selecting learning parameters.

Results

The following tables summarize the results of the tuning exercise. Runs were made with the standard parameters for each configuration of the environment and Q-learning algorithm. Then the GA was run for each configuration. Using the same simulator configuration that the GA used, the best learning parameters from the GA were replicated on agents, and 200 simulations were run. The tables show the average survival rates for the 200 simulations. Note that the survival rates for runs using the standard parameters are repeated in the tables for easier comparison.

Tables 2 and 3 show the number of surviving agents in both static and dynamic environments for both the deterministic and nondeterministic versions of the Q-learning algorithm. The evolved parameters were produced by a GA where each chromosome corresponded to a set of agents with the same learning parameters running in a single simulation.

Deterministic Q-Learning, Homogeneous GA Run			
Static Environment		Dynamic Environment	
Standard γ=0.1, α=0.1, ε=0.01	Evolved γ=0.9846, α=0.1548, ε=0.0	Standard γ=0.1, α=0.1, ε=0.01	Evolved γ=1.0, α=0.7006, ε=0.1144
172.87	218.39	102.84	120.24

Table 2 - The number of surviving agents (deterministic/homogeneous)

Nondeterministic Q-Learning, Homogeneous GA Run			
Static Environment		Dynamic Environment	
Standard γ=0.1, α=0.1, ε=0.01	Evolved γ=0.9965, α=0.2518, ε=0.0	Standard γ=0.1, α=0.1, ε=0.01	Evolved γ=1.0, α=0.3521, ε=0.1213
160.19	212.35	18.21	77.74

Table 3 - The number of surviving (nondeterministic/homogeneous)

Deterministic Q-Learning, Heterogeneous GA Run			
Static Environment		Dynamic Environment	
Standard γ=0.1, α=0.1, ε=0.01	Evolved γ=0.9995, α=0.0570, ε=0.1416	Standard γ=0.1, α=0.1, ε=0.01	Evolved γ=1.0, α=0.3789, ε=0.0580
172.87	216.81	102.84	112.59

Table 4 - The number of surviving agents (deterministic/heterogeneous)

Nondeterministic Q-Learning, Heterogeneous GA Run			
Static Environment		Dynamic Environment	
Standard γ=0.1, α=0.1, ε=0.01	Evolved γ=1.0, α=0.5793, ε=0.0	Standard γ=0.1, α=0.1, ε=0.01	Evolved γ=0.8428, α=0.8321, ε=0.0961
160.19	205.46	18.21	54.41

Table 5 - The number of surviving agents (nondeterministic/heterogeneous)

Tables 4 and 5 are similar to similar to tables 2 and 3 except that the evolved parameters were produced by a GA where each chromosome represents the learning parameters of a single agent running in a simulation of heterogeneous agents.

In all cases, populations of agents with evolved learning parameters outperformed the standard starting parameter values. Statistical t-tests were confirmed that the improvements gained by tuning with a GA were significant. The comparisons between the deterministic and nondeterministic Q-learning algorithms and evolved parameter runs from GAs using homogeneous and heterogeneous agent mixes were also statistically significant.

Analysis

The results are interesting in several ways. First, it should be noted that the evolved γ parameter is near 1.0 in all cases. The best performers place a high value on the potential for future rewards. This result makes sense because agents can receive immediate misleading negative rewards as noted in the introduction. The large γ value moderates these misleading rewards by offsetting them with reward estimates learned over time. In the dynamic case, the large γ might hinder relearning as the roles of red and blue veggies change, but generally it is good for survival.

Less can be said about the α values as their values vary across the different configurations of the simulations. α is larger in the dynamic environment cases over the corresponding static cases. This observation is reasonable because learning must be quicker in environments where the underlying environment changes.

It is surprising that the ε values are 0.0 for three of the four static environments for the evolved parameters. This would seem detrimental to the agent as it might miss good solutions by not exploring. But, because of the simple nature of the environment and the action selection policy, the agents can learn without exploring untried actions in the static environment. Initially, at each health level, the agent will select some action, eat or don't eat, for blue and red veggies. Due to inertia, it will continue using this action until a heath level change. At that point, the Q-value of the current action for the current state will either be lowered or made larger. If a positive reward were received for the action, then that will become the action of choice whenever this type of veggie is encountered again until another reward is received. If a negative reward is received, then the opposite action will be the action of choice. So, because there are only two Boolean-related actions, agents can learn without exploring in a static environment.

In the dynamic environment, exploring seems to be a useful action. It probably helps during the change of veggie roles by allowing the agent to more quickly get positive rewards for doing actions it had previously learned were not advantageous.

In comparing the deterministic and nondeterministic versions of the Q-learning algorithm, it is noteworthy that the deterministic version of the algorithm converges in an environment where rewards are given in a nondeterministic fashion. One reason for this effect is that the frequency of misleading rewards is small with respect to non-misleading rewards. This is especially true as the population size decreases as some agents starve, and the likelihood of eating a growing veggie decreases. It is also helpful that non-poisonous veggies give a much higher nutrient value than the negative effect of poisonous veggies. Even in the dynamic environment where the roles of veggies change, most agents have adequate energy reserves to survive while relearning the proper actions.

It is important that the Q-learning algorithm converges for the GA to work properly since the fitness value is based on how well agents learn to survive. If the survival rate changes greatly among simulation runs with the same set of learning parameters, then the GA cannot find a good set of parameters. In this narrow experiment, the deterministic algorithm did converge; however, it might not have if the frequency of good rewards were not higher than the frequency of misleading rewards. In such cases, the use of the nondeterministic version of the Q-learning algorithm would be appropriate if the underlying distributions support learning.

The GA running sets of homogeneous agents found the best sets of learning parameters at a significant increase in run time. But, in several cases, it was not obvious that the improvements over the heterogeneous cases were significantly better without performing a statistical analysis. The GA using homogeneous performed 29 more simulations per population than the heterogeneous case.

Conclusions and Future Work

The Pond experiments indicate that survival of learning agents is difficult in an overpopulated environment where misleading rewards occasionally occur. However, improvements over the standard starting values for learning parameters were found by the GA in every case in this study. More study is needed to determine how well tuning works in other environments such as those that are not constrained by overpopulation or in which more actions are available and all are not Boolean related.

In learning environments where Q-learning converges, then tuning Q-learning with a GA shows promise. In cases where real learning robots are to be sent into a difficult environment, learning can be improved by first simulating the environment and using a GA to tune the learning parameters.

The Pond was originally developed to study the broader topic of cooperating learning agents that explore previously unexplored environments. Experiments are planned to study cooperating groups of functionally heterogeneous agents. Sharing of learning trials among agents will be studied, and a parallel processing version of the Pond simulator is planned.

References

1. Aljibury, H. and Arroyo, A. (1999). Creating Q-table parameters using genetic algorithm. Florida Conference on Recent Advances in Robotics.
2. Mitchell, T. (1997). *Machine Learning.* McGraw-Hill, Boston, MA.
3. Sutton, R. and Barto, A. (1998). *Reinforcement Learning.* MIT Press, Cambridge, MA.
4. Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* Ph.D. thesis, Cambridge University.
5. Watkins, C. and Dayan, P. (1992). Q-learning. *Machine Learning, 8, 279-292.*