Creating Q-table parameters using Genetic Algorithms

Hal Aljibury, A. Antonio Arroyo
Machine Intelligence Laboratory
Dept. of Electrical Engineering
University of Florida, USA
Tel. (352) 392-6605
Email: luck@mil.ufl.edu,arroyo@mil.ufl.edu

1999 Florida conference on Recent Advances in Robotics
April 29-30 University of Florida

Abstract:

This paper covers the use of a genetic algorithm to create ideal Q-learning parameters for the creation of a Q-table used by a simulated mobile agent for object avoidance. An arena simulation was created that would handle the agent, as well as calculating its Q-table. Compared to a real-world environment, evaluation proceeded much faster, with a 40-minute experiment reduced to some 6 seconds in the simulation. Despite this speed increase, the simulation was too slow to definitively answer the question of whether ideal Q-table parameters can be evolved. Results suggest that the evolved parameters are highly specified, according to the configuration of the arena.

Introduction:

Reinforcement schedules for Q-learning depend upon the physical structure of the robot, its sensors and actuators, the environment and the resultant task to be realized. Reinforcement schedules often have subtle effects on a robot's resultant behavior. Designing suitable reinforcement schedules to accomplish specific tasks tend to be more art than engineering. Measuring cumulative rewards or summing the Q-table maximum values provide mostly techical information about the method, not the utility of Q-learning for improving task performance as perceived by a human observer. …Without a guiding theory, we must develop reinforcement schedules intuitively and then compare the resultant behaviors. Techniques for systematic mapping a reinforcement schedule to a resultant (emergent) function of the robot remains an open question. [Burrati]

I have been exploring an interesting method of generating reinforcement schedules for Q-learning using a genetic algorithm to select for an optimum learning rate. I selected object-avoidance as the emergent behavior, since Q-learning is well-suited for generating this behavior. Q-learning originated as an algorithm for solving Markov decision processes, which have a finite number of states, with each state having a finite number of possible actions. The basic Q-learning algorithm is stated below:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r(s,a) + \gamma \max Q(s',a')$$

The value of the action associated with state-action pair (s,a) is Q(s,a). The new value of Q(s,a) is determined by the learning rate alpha in combination with dependence upon past results gamma. The max operator selects the action with the highest value for the state in the Q-table. The state itself is determined by the sensor inputs. The agent in this simulation uses infra-red (IR) detection to determine its proximity to an obstacle in its path. To reduce the number of states involved, the IR sensors were quantized to 2 bits each, and the bump detector to 1 bit, resulting in 32 possible input states.

This effort has its roots in reducing the art involved in constructing suitable reinforcement schedules.
All parameters that are involved in the Q-learning algorithm and reinforcement schedule

were put through a genetic algorithm which evaluated the robots behavior in terms of collisions per unit time, with collisions become progressively more undesireable later in the learning process.

Procedure:

A simulated robot arena was central to this project. Briefly, the way this arena worked was to take one robot at a time, and then evaluate each robot by the 'let it roam' method, producing a Q-table in accordance with the parameters that were given to it. This arena could perform 5000 movement steps, which should be enough to give an excellent estimate as to the fitness of the particular setup. However, the time involved with evolving populations was close to unmanageable (24+ hours on a pentium pro) for this simulation length. Therefore, in later runs the length was reduced to a mere 500 steps.

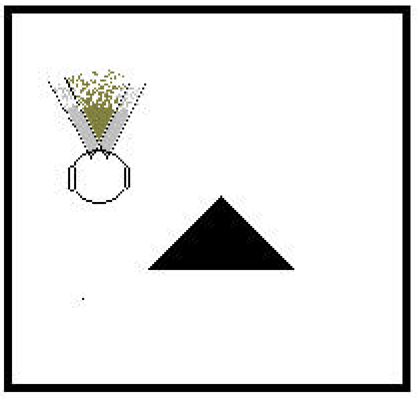
Fig. 1 (arena representation)

The observed distribution of fitnesses, instead of a specified fitness value, stems from the nature of how the Q-table is constructed. It necessarily depends upon random exploration of the state space, which implies that some robots of otherwise equal fitness would collide a different number of times, and thus have different fitness values.

However, over the long run, it is statistically likely that a particular combination of learning parameters and reinforcement schedules will result in a fewer number of collisions.

Assumption: a particular set of parameters and reinforcement schedule can result in a Q-table produced from a smaller number of collisions than a different set of parameters. (based on material from Sutton 1988)

Equivalently: This Q-table had more early collisions than late collisions, in other words, it learned the environment, then exploited its knowledge by not colliding.

Given these assumptions, one more assumption must be made, upon which rests the foundation of this approach. The results, while not strictly repeatable, must be statistically repeatable. In other words, it is assumed that the evaluation of the parameters will be similar to repeatedly flipping a weighted coin, in that one side is not guaranteed to come up, but over many flips, its overall likelihood can be determined with < epsilon certaincy.

Reducing the number of steps from 5000 to 500 would therefore cause some problem with accurately determining the real fitness of a schema, but this was necessary to cut the run time to hours instead of days.

Program details:

The arena simulation is scaleable, with several obstacles and robots allowable inside. To keep runtimes down, only one robot, and one triangular object were used. The robot follows the following procedure: First, it evaluates the infra-red (IR) reflections from the objects. The reflectivity of the obstacles and walls were fixed at 1 (perfect), but this can be changed as necessary. The algorithm divides up the side-length of each obstacle and wall into roughly pixel-sized segments, then evaluates the distance and whether or not the segment is in a detection arc of 30 degrees for either the left or right IR sensors. Secondly, the robot evaluates its Q-table to determine the appropriate action for its state, as well as rewarding the previous state,action pair. The IR values are quantized to 2 bits each, and the bump detector is quantized to one bit, giving 32 possible states. Once an action is chosen, the collision detection routine tells the robot whether or not it can move in that direction. To solve a minor bug, the allowable movements are only in an restricted arc facing away from the collision direction. Occasionally, the robot would escape from the arena by sliding along a wall and out the corner, since only one collision direction was reported. Allowing the robot only to step away from the collision was effective in solving this.

Details of the Genetic Algorithm:

The genetic algorithm used in this project used a single chromosome, with real-valued genes. Since the genetic operators in the program manipulate the data in a matter independent of representation, this should not affect the results. [Mitchell 157] A proportionate selection method was used for reproduction, since this method seemed to offer the best way to handle the indeterminate fitness of a particular schema. It seems intuitively better to select individuals that are statistically more fit with an algorithm that statistically weights reproduction in favor of fitter individuals. Selection based on rank did not seem appropriate, since the fitness values seemed to form a clustered distribution, and a steady-state population would have taken too long to evaluate.

Crossover could occur at any point between the 9 genes for, (in order) alpha, gamma, bump penalty, right turn, left turn, forward reward, right reward, left reward, and backward reward. The values for alpha and gamma ranged between 0.001 and .997. The values for the bump penalty ranged from –100 to –1, since under no circumstances should the robot be rewarded for collisions. The values for turning ranged from 0 to PI, with left turns being negative, and right turns positive. The values for the reinforcement schedule are as follows: Forward ranged from 1 to 100, Right, left, and back ranged from –100 to 99, but always at least 1 less than the forward reward. This restriction was made to handle the highly fit, but undesirable conditions of interminable circling, see-saw motion, etc.

Conclusion:

I started this project with the aim of producing a set of Q-learning parameters and reinforcement schedules that would result in rapid learning. I had hoped that this set of parameters would result in rapid learning for other similar environments. By similar environments, I am referring to an arena which is different from the original by some amount, but in which a robot with a Q-table learned in the old arena could be placed into the new arena without having problems with collisions. By my experimentation with Q-learning in the University of Florida Machine Intelligence Lab (MIL), I have observed that changing an arena configuration in any reasonable manner does not result in in more collisions until the robot has learned the new environment. A reasonable manner implies that spacing between objects is not reduced below the IR sensing capabilities, that non-reflective obstacles are not placed in the arena, and so forth.

I had hoped that the Q-learning parameters were equally as robust to an environmental change. In lieu of changing the simulation environment (which would have entailed more day-long runs), I had changed the starting orientation of the agent from -.10 radians (below horizontal, facing right) to being entirely random. This seemingly minor change is actually rather significant, because the robot's first learning experience will come from a random direction, instead of slanting across the arena into a wall the same way every time. However, on the basis of experience, I had thought that this would only affect the first few steps, and that highly fit parameters would deal easily with this.

However, during the runs, I observed that the program rarely reached the same sets of parameters, although the learning rate for all the solutions had improved considerably. On average, the finishing collision rate was roughly a third of the starting parameters. Some consistency was noted in terms of the bump penalty, and the right and left turns and rewards. This can be directly traced to the arena setup, showing that at least part of my hypothesis was correct, in that highly fit parameters could be evolved.

The shape of the arena, as well as the starting orientation had an influence on the size of the left turn compared to the right turn. With the fixed starting orientation, the left turn was considerably larger than the right by about a 2-1 ratio. This can be traced to the robot's first collision, assuming it had been traveling in a straight line. The robots first collision with a wall was at an oblique angle. If the robot had collided at a 90 degree angle, I think that the evolved right and left turns would be almost equal (as well as slightly more than 90 degrees). I also think it is interesting that the bump penalty did not go to the maximum, instead stabilizing in a range from about –75 to –90 on various runs. This shows that, while collisions were deemed to be quite bad by the algorithm, on occasion they were beneficial to learning (especially early on).

However alpha and gamma, the parameters which I had particularly hoped to have converge to a solution, instead gave multiple results. Based upon the sparseness of the arena, I had thought that gamma would converge to near maximum, giving the most weight to previous experience, but in only 2 occasions did this happen.

In summary, the results from this project can best be regarded as preliminary, with more evaluation needed. In the respect that Q-learning parameters can be used for multiple environments, this project seems to indicate that the parameters themselves are not transitive, though the generated q-tables might well be.

References:

1. [Burrati 1997] Davide Burrati, et al. “Reactive, Sequenced Q-learning of Multiple Tasks by an Autonomous Mobile Robot” to be published.

2. [Chapman 1997] Kevin Chapman and John Bay, “Task decomposition and Dynamic Policy Merging in the Distributed Q-learning Classifier System” Subm. to 1997 IEEE International Symposium on Computational Intelligence in Robotics.

3. [Mitchell 1996] Melanie Mitchell, An Introduction to Genetic Algorithms, the MIT Press, Cambridge MA, 1997.

4. [Kaebling, 1996] Leslie Kaebling and Michael Littman, “Reinforcement Learning: A survey”, Journal of Artificial Intelligence Research, May 1996. pp. 238-277.

5. [Sutton 1988] Richard S. Sutton, “Learning to predict by the Methods of Temporal Differences”, Machine Learning 3:9 44, 1988. pp. 10-43.