Performance of Q-Learning and Value Iteration (CS605.449 – Project 7)

By Max Robinson

Abstract

In this experiment the performance of Q-Learning and Value Iteration are compared to themselves and each other on the race track problem, using effectively three different tracks. Specifically, the three tracks used are an L-Track, and R-Track, and a modified R-Track, R-Track-Harsh, that changes what happens to the agent upon crashing into the wall of the race track to make it more difficult. The models were trained for each algorithm on each track. The results showed that both algorithms performed best on the L-Track. Q-Learning performed significantly worse on the R-Tracks, but better on the R-Track-Harsh than the R-Track. When comparing the two algorithms, Value Iteration performed better than the Q-Learning algorithm, but overall it seemed to take a much longer time to converge to the solutions for Value Iteration, even though the number of iterations were fewer.

Problem

The problem being investigated in this experiment is how well the Q-Learning Reinforcement algorithm performs on two different tracks in the Race Track problem and when compared to the Value Iteration solutions for the race tracks. The two race tracks used for this experiment are the L-Track and the R-Track. The Race Track problem is a control problem in which an agent controls a car on a race track and attempts to reach the finish line in the fewest number of steps possible. The R-Track also has two crash settings. The simple version places the agent on the nearest valid track space when the agent crashes. The harsh version places the agent back at the start of the track should a crash occur.

For this experiment I hypothesize that the Q-Learning algorithm will perform the best on the L-Track, while needing the most number of steps to reach the goal on the R-Track with the harsh crash setting. The L-Track is a much simpler track and it is also smaller. As a result, I expect the Q-Learning algorithm to be able to create a better strategy when compared to the larger more complex track.

I also hypothesize that the Value Iteration algorithms will create far more optimal policies than the Q-learning, though I believe it will take longer to train. Policy iteration looks at every possible state and iterates over the values until the values for the states stop changing very much. As a result, it is able to look at many more states and provide many more traceable rewards to every state that is involved in the game. As a result, I expect this algorithm to provide the shortest average number of steps to the goal.

Algorithm Implementation

Value Iteration

Value Iteration is a dynamic programing approach proposed by Bellman [1] to solve Markov decision processes. The algorithm works by keeping track of an optimal policy table and a Q value table that keeps track of the utility of the states in the process. Value Iteration also makes the assumption that there is a model $T(s,a,s')$ that is known to the algorithm. Value Iteration then has the list of all states and known goal states, and incrementally updates the Q value for every state, and adjusts the policy table V(s) accordingly after each update.

The update function for the Q table can be described as follows [1] translated into Q and T variables

$$Q_t(s,a) = r(s,a) + \gamma \sum_{s' in S} T(s,a,s')V_{t-1}(s')$$

A policy is then chosen $\pi(s) = argmax_a Q_t(s,a)$, and then the policy table V(s) is updates as $V(s) = Q_t(s,\pi(s))$. This means that the policy table always has the best policy based on the Q value for the best state action pair.

To know when to be done updating the two tables the Bellman Error Magnitude is used which is characterized as ϵ. When the maximum difference between $V_t(s)$ and $V_{t-1}(s)$ is less than ϵ then the updates to the tables stops, and the best policy at time t is returned.

This process updates every state available in the state space every time updates are made. This can grow to be expensive depending on the size of the state space being worked in.

Q-Learning

Q-Learning, developed by Christopher Watkins [2], is a dynamic programming approach to "learn how to act optimally in controlled Markovian domains" [2]. Q-Learning works by keep a record or state-action table that records the utility of that state-action pair. These state-action pairs are then updated over time as the agent moves about the world. Specifically, Q-Learning operates with the following action-observation loop [2].

1. Observes its current state s
2. Select and perform an action a
3. Observer the subsequent state s'
4. Receive an immediate reward r,
5. Update $Q_{t+1}(s,a)$ using the previously known values for $Q_t(s,a)$

The update equation for $Q_{t+1}(s,a)$ is as follows [2]:

$$Q_{t+1}(s,a) = (1-\alpha)Q(s,a) + \alpha[r(s,a) + \gamma * argmax_{a'}(Q_t(s',a'))$$

α here is a learning rate, γ is a discount factor, and $r(s,a)$ is the reward for the state action pair s, a. This equation updates the Q table with the value of the current Q state, plus a discounted rate for the best known state action pair in the Q table for s' at the time. The best known state action pair is used rather than using the action that was taken from Q. This make Q-Learning an off-policy learner.

This algorithm has been proven to converge given that "the sequence of episodes that forms the basis for learning must include an infinite number of episodes for each starting state and action" [2].

For this implementation for Q-Learning, in order to explore the state space and make decisions that are something other than the greedy choice when learning the policy, this Q-Learning variant uses and Epsilon-Greedy approach to exploration and exploitation. This means that at any time where a policy is chosen for s, $\pi(s)$, the policy is chosen as follows.

$$\pi(s) = \begin{cases} argmax_a\, Q(s,a) \; with\, probability\, 1-\epsilon \\ random\, a \;\; with\, probability\, \epsilon \end{cases}$$

This allows the algorithm to control how often new states are explored and how often to exploit the algorithm's knowledge. To make this effective for training, a decay rate is also applied to epsilon s o that as the agent learns, it takes fewer random moves and uses more of its knowledge to select the policy. This discount factor is applied as

$$\epsilon = \epsilon * discountRate^t$$

were t is the episode t, where an episode is considered to be a full traversal of the agent from start to goal.

Scoring

Steps to Goal

The primary scoring mechanism for comparing Q-Learning and Value Iteration will be the average number of steps to the goal for a learned model. This is calculated by applying a model that has been learned by the algorithm and using only the information in the model to take actions in the race track. No further updates were made to the model, and no random action selection, such as epsilon -greedy were used to explore the states space. Instead exploitation of the model was used to get from the start to the goal. The average number of steps that it took using the model is then the value that is compared across tracks and across algorithms.

Epochs to Reach Minimum Steps

The number of epochs to reach a minimum step number is not something that is used to directly compare the performance of the Q-Learning learning algorithm across tracks, but it is used to describe how well the information was learned and how quickly. This metric is used when comparing th e time it takes for the Q-Learning algorithm to learn a policy to some threshold.

Experimental Approach

The approach of this experiment was to train each of the algorithms, Q-Learning and Value Iteration, on each of the tracks, L-Track, R-Track, and R-Track with harsh crashing (R-Track-Harsh). For Q-Learning, the number of steps per epoch were captured to show how many steps were needed as the algorithm progressed in learning. The process of learning the models and capturing the number of steps needed per epoch was done ten times to average out each run.

A policy was then chosen to give a sample average number of steps it would take, applying that policy, to reach the goal. This provides the metrics by which the algorithms are compared. The average number of steps required are compared across tracks, and across algorithms.

The other information considered when comparing algorithms is amongst the Q-Learning algorithms. The number of epochs needed to converge towards the average number of steps for the model is considered when comparing the tracks to one another. This is shown through the average number of steps taken when learning for each of the ten runs of the experiments.

Finding Parameters

Q-Learning requires that meta-parameters such as the learning rate, discount rate, number of episodes per epoch, and the random action selection rate for epsilon-Greedy, along with a decay rate for the epsilon are all provided for the algorithm. These parameters can have large effects on the ability for the algorithm to learn appropriately.

Unfortunately, since the amount of time it takes to train each of the models tends to be quite significant, a grid search strategy could not be applied given the current time constraints. Instead the process by which these parameters were select was on an anecdotal basis. While the selection of the parameters was based on anecdotal success, the process to find relatively good parameters had a pattern. The parameters were often first set to be very high, followed by low, and then adjusting back to high and low again until a suitable anecdotal parameter was found. This had a bracketing effect on finding parameters.

The parameters that were chosen are shown in

Table 1 Q-Learning parameters per Track

Track	Num Episodes	Alpha	Gamma	Epsilon	Decay Rate
L-Track	5000	.7	.7	.99	.9
R-Track	2000	.7	.7	.99	.9
R-Track-Harsh	5000	.7	.7	.1	.99

For Value Iteration an epsilon parameter is also needed to inform the algorithm when to stop iterating. This is the Bellman Error Magnitude. For this experiment this value was set to .1 for all models.

Results

After the running the experiment as described in the above sections, two sets of results are displayed below. The first set of results are diagrams describing how the Q-Learning algorithm learned its model by graphing the average number of steps taken for each epoch over the epoch number. The second set of results show a comparison of the Q-Learning algorithm when compared to the Value Iteration algorithm on each track.

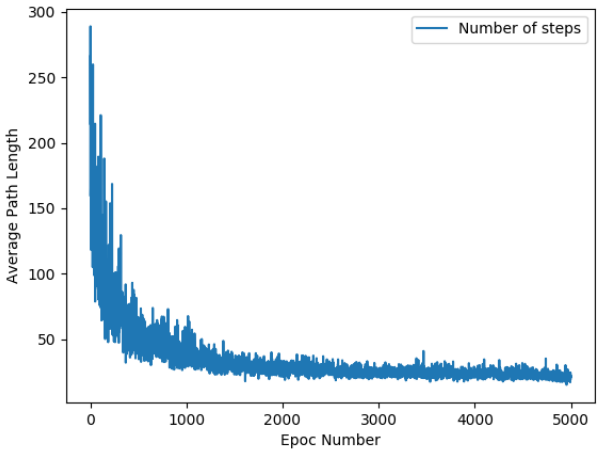

Figure 1 L-Track

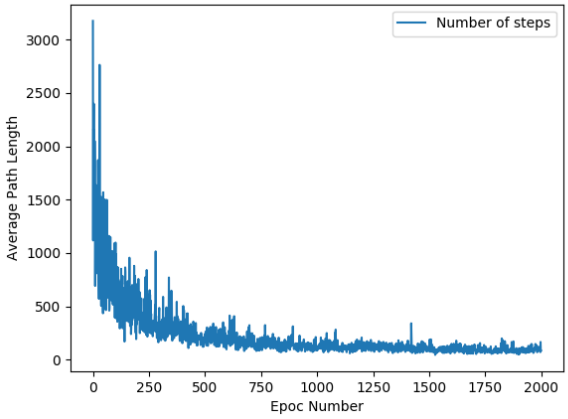

Figure 2 R-Track

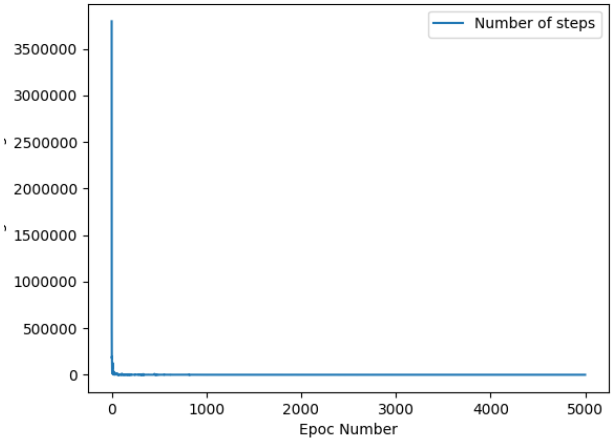

Figure 3 R-Track-Harsh

Table 2 Average Steps to Goal per algorithm and track

Algorithm	L-Track	R-Track	R-Track-Harsh
Q-Learning	19.0 steps	75.98 steps	66.95 steps
Value Iteration	11.31 steps	18.23 steps	34.98 steps

Table 3 Iterations needed for Value Iteration to Converge

Algorithm	L-Track	R-Track	R-Track-Harsh
Value Iteration	12	26	52

Behavior

The behavior of the two different algorithms on the three different tracks performed about as expected, with one surprise.

When comparing the performance of the Q-Learning algorithm across the three different tracks, the results deviated slight from what was expected. The L-Track had the fewest average number of steps to the goal, which was expected. The L-Track is significantly smaller than the R-Track and is a simpler design than the R-Track. As a result, it would make sense for the number of steps to be fewer than the other tracks for the Q-Learning algorithm.

When comparing the R-Track and the R-Track-Harsh for Q-Learning, the R-Track-Harsh model performed a little bit better on average than the R-Track. This is a bit surprising. One of two things could have happened. The R-Track-Harsh model, might have learned a more conservative approach to getting to the finish line because of how big the penalty was if a crash was caused. This could mean that the policy learned mitigates the risk of crashing more. The R-Track, might have not learned that as well. Another explanation is that the parameters that were used for the R-Track model were not sufficient to provide the full convergence to a minimal solution for the track. Since the parameters were chosen an ecdotally, it is very possible that the parameters were not tuned fine enough for this track and problem.

Comparing Q-Learning to Value Iteration, the results are as expected. The Value Iteration models are significantly better on average than the Q-Learning model. However, anecdotally, the time it took to train the Value Iteration took much longer than the time to train the Q-Learning models, even though the number of iterations needed were fewer.

The number of steps on average that are needed for each track increase as the complexity of the track increases, starting as low as 11.31 steps on average for the L-Track, and as many as 34.98 on average for the R-Track-Harsh. The number of iterations needed to converge to these policies also reflex the complexity of the track. The fewest number of iterations needed were for the L-Track, and the most for the R-Trach-Harsh.

An interesting discrepancy is how much closer the R-Track-Harsh average steps are for Q-Learning and Value Iteration, than the number of steps needed for the R-Track for the Q-Learning algorithm. The Q-Learning algorithm needs more than four times the number of steps for the R-Track than the Value Iteration. In comparison the R-Track-Harsh Q-Learning model needs only just under two times as many steps than the Value Iteration model. This would seem to point out that the parameters for the R-Track

for Q-Learning were not optimal. The Value Iteration algorithms shows that the track is easier to navigate by showing how many fewer steps on average are needed.

Summary

Overall, the algorithms performed about as expected with an odd discrepancy regarding the R-Track for Q-Learning. The L-Track was the track that both algorithms performed the best on, as expected. The simplicity of the track along with the smaller size allowed the number of steps needed to get to the goal to be overall fairly low, for both algorithms. The R-Track then required more steps due to it being a more complicated and larger track. The Q-Learning model for the R-Track while requiring more steps to get to the goal as expected, performed worse in comparison to the R-Track-Harsh and Value Iteration models than expected. This anomaly is most likely due to parameters not being tuned properly for the track. The algorithms performed as expected on R-Track-Harsh, needed the highest number of steps on average to get to the goal. Finally, while the Value iteration algorithm performed better across the board, anecdotally it took far more time to train the model than to train the Q-Learning models.

References

[1] BELLMAN, RICHARD. "A Markovian Decision Process." Journal of Mathematics and Mechanics, vol. 6, no. 5, 1957, pp. 679–684.

[2] Watkins, Christopher J. C. H., and Peter Dayan. "Q-Learning." Machine Learning, vol. 8, no. 3-4, 1992, pp. 279–292., doi:10.1007/bf00992698.