Reinforcement Learning Techniques for Discrete and Continuous Markov Decision Processes

Travis Latzke

November 24, 2019

1 Introduction

Reinforcement learning (RL) is one of the three primary paradigms of machine learning and the underlying mathematical framework has been proven to be a powerful framework for training agents to achieve a goal in a given environment. The types of environments in reinforcement learning range from simple toy problems, which usually consist of a discrete grid state-space, to continuous three dimensional environments that represent real life robotic problems.

Similarly, the available actions an agent can take in an environment range from a small set of discrete actions in toy problems, to a large set of actions that may take on continuous values. In general, more sophisticated algorithms are needed to solve problems with larger state-spaces and action-spaces. The methods in this paper will show how the performance of traditional methods RL like value iteration, policy iteration, and tabular q-learning is affected by different problem sizes. The following section provides a common mathematical background used in RL to help aid in the understanding of the experiments shown in later sections

2 Background

2.1 Markov Decision Process

Reinforcement learning problems are usually modeled as a Markov Decision Process (MDP) where an MDP is formally defined as a 4-tuple $\{S, A, R, P\}$:

- S is the set of all states an agent can be in.
- $A(s)$, $s \in S$ is the set of all actions an agent can take in state s.
- $R : A \times S \times S \to \mathbb{R}$ is the reward function. $R(a, s, s')$ is used to denote the reward for taking action a in state s and transitioning to state s'
- $P : A \times S \times S \to S$ is the probability function. $P(a, s, s')$ is used to denote the probability that an agent will transition to s' from s after taking action a.

The core problem of any MDP is to find a policy π that is as close as possible to the optimal policy π_*. A policy $\pi : S \to A$ is a mapping of states to actions. The optimal policy is a mapping of states to actions where each s $\in S$ is mapped to the best action to take in state s. The best action to take in state s is considered by evaluating what action leads to the next state with the highest expected reward where the highest expected reward is defined as:

$$v_\pi(s) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s], \forall s \in S$$

where $\mathbb{E}[\cdot]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step (Sutton, Barto 2017). $\gamma, 0 < \gamma < 1$, is the discount factor that controls how the agent favors long term rewards vs short term rewards. The closer γ is to 1, then the more the agent favors selecting actions that lead to higher long term rewards. If γ is closer to 0, then the agent would favor short term rewards more strongly. Different γ values are studied empirically in section 2.1, where experiments show how different γ values influence how the policy is calculated of a given MDP. The algorithms in the next section illustrate how to calculate π^* and v_{π^*}.

2.2 Policy Iteration

The idea behind policy iteration is start with a random policy π. Then find the values $v_\pi(s)$ of each state for the policy π. Then determine if there is any action a that is better to take in state s then what was return by $\pi(s)$. This is called extracting a new policy π' based on the values from v_π. Then new values $v_{\pi'}$ are calculated from π'. This process is repeated until $\pi = \pi'$. The formal algorithm is provided below:

Algorithm 1: Policy Iteration

```
Initialize $V(s) \in R$ and $\pi(s) \in A(s)$ arbitrarily for all $s \in S$;
repeat
    $policy\_has\_changed \leftarrow false$;
    repeat
        $\Delta \leftarrow 0$;
        foreach $s \in S$ do
            $v \leftarrow V(s)$;
            $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))[r + \gamma V(s')]$;
            $\Delta \leftarrow max(\Delta, |v - V(s)|)$;
        end
    until $\Delta \approx 0$;
    foreach $s \in S$ do
        $v \leftarrow V(s)$;
        $\pi'(s) \leftarrow argmax_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$;
        if $\pi(s) \neq \pi'(s)$ then
            $policy\_has\_changed = true$
        end
    end
    $\pi \leftarrow \pi'$;
until $policy\_has\_changed = false$;
```

2.3 Value Iteration

Value iteration is similar to policy iteration in that it will return the optimal policy π^* and optimal value function v_{π^*}. However, value iteration calculates v(s) by considering all actions that can be taken from s, where as policy iteration only considered the action $\pi(s)$.

Algorithm 2: Value Iteration

```
Initialize array $V$ arbitrarily (e.g., $V(s) = 0, \forall s \in S$)
repeat
    $\Delta \leftarrow 0$;
    foreach $s \in S$ do
        $v \leftarrow V(s)$;
        $V(s) \leftarrow max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$;
        $\Delta \leftarrow max(\Delta, |v - V(s)|)$;
    end
until $\Delta \approx 0$;
```

2.4 Tabular Q-Learning

MDP's provide powerful ways to create agents that follow an optimal policy π_*. However, the model of a given process is not always known up front and so there are cases when an agent needs the ability to learn model-free. Q-learning is a model-free algorithm that an agent can use to calculate the optimal policy and optimal values for each state. Q-learning is based on the trade off between exploration and exploitation. The idea behind exploration and exploitation is that at every time step t state $s_t \in S$ there is some probability:

Algorithm 3: Tabular Q-Learning

```
Initialize $\alpha, \gamma$, and $Q(s, a)$, for all $s \in S$, $a \in A(s)$,
For each episode repeat
    Initialize $s$;
    Set $\epsilon$ according to episode and decay function;
    For each step in episode repeat
        $a \leftarrow argmax_a Q(s, a)$;
        Override action $a$ with a random action $a \in A(s)$ with probability $\epsilon$
        Take action $a$, observe $r$ and $s'$;
        $Q(s, a) \leftarrow Q(s, a) + \alpha[R + \gamma max_a Q(s', a) - Q(s, a)]$;
    until $s$ is terminal;
until $\Delta Q \approx 0$;
```

.

Three important parameters that influence how the Q table converges are α, γ, and the exploration parameter epsilon. As stated previously, γ is the discount rate, which determines how much future rewards influence the value of the current state. alpha is the learning rate that determines how much the previous the previous Q value should be updated by. If alpha is set too high, then it may be possible that the Q values do not converge. If alpha is set too low, then the algorithm may need to run for many episodes more episodes to converge. epsilon is the exploration parameter, which is used to determine the probability of selecting a random action in state $s \in S$.

Ideally, agents that are unfamiliar with their environments need to explore more to learn about what rewards are given for actions taken in certain states. Conversely, the agent needs to exploit the actions that is more confident about as it becomes more familiar with its environment. The best way to model this phenomenon is through the use of some probability function that decays over time. Three different decay functions were used in the experiments in this paper.

- **Linear:** $\epsilon \leftarrow max(0.01, \epsilon_{initial} - n * \mu)$

- **Geometric:** $\epsilon \leftarrow max(0.01, \mu^n)$
- **Exponential:** $\epsilon \leftarrow max(0.01, e^{-n/\mu})$

Where n represents the n^{th} episode and μ represents the decay rate. The max function ensures that the exploration rate ϵ is at least 0.01. All of the algorithms in this section have been adapted from the reinforcement learning book by Sutton and Barto (Sutton, Barto 2017).

3 RL Example Problems

3.1 Frozen Lake Problem

The first Markov Decision Process studied in this paper is modeled after the frozen lake environment from Open AI. The frozen lake environment consists of a 4x4 grid with one state labeled as the goal state and the other states labeled as either F (frozen) or H (hole). Frozen states are safe for the agent to walk on, however hole states are not safe to walk on. If the agent walks on a hole state, then the agent will fall into the lake and the episode will terminate without the agent receiving a reward. If the agent reaches the goal state, then the episode will terminate and the agent will receive a reward of +1.

The agent can act in any given state by moving up, right, down, or left. Due to the slippery ice, the agent has a 33% of making it to the the desired state for a given action. The agent may 'slip' and end up in a state that is adjacent from previous state, but diagonal from the desired state with a 66% chance (33% chance for each adjacent state). The grid shown in table 1 shows an instance of the frozen lake problem. Table 2 shows the optimal values and optimal policy achieved from value iteration, policy iteration, and the Q-learning algorithm with a gamma value of 0.99

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

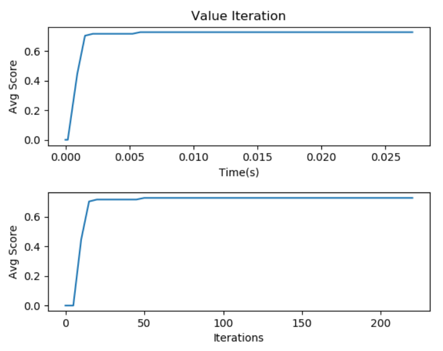
Table 1: *An instance of a frozen lake grid problem. This instance was used to obtain the results for each algorithm below.*

0.541	0.498	0.470	0.456
0.558	0.0	0.358	0.0
0.591	0.642	0.615	0.0
0.0	0.741	0.862	1.0

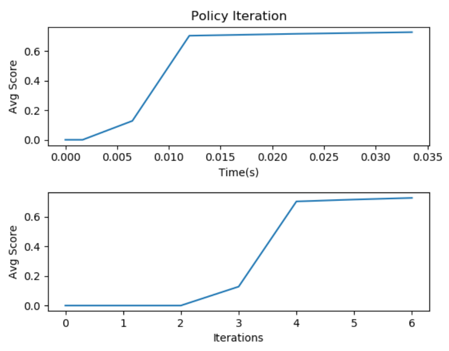
$\leftarrow$	$\uparrow$	$\uparrow$	$\uparrow$
$\leftarrow$	$-$	$\leftarrow$	$-$
$\uparrow$	$\downarrow$	$\leftarrow$	$-$
$-$	$\rightarrow$	$\downarrow$	$-$

Table 2: *The optimal values v_π and the optimal policy π^* that was calculated by the value iteration, policy iteration, and for the best Q-learner model.*

One interesting thing to note about the optimal policy found from the algorithms is the action that is mapped to the state corresponding to the second row from the top and third column from the right. The policy for this state corresponds to moving left, which corresponds to moving into a state with a hole. This may seem strange at first, but actually makes sense when you think about it. If the agent were to choose to move up or down, then there is a 66% chance that the agent will fall into a hole because the states to it's left and right consists of states with holes. However, if the



(a) Value Iteration



(b) Policy Iteration

Figure 1: *The average score vs the iterations/time that the value and policy iteration algorithms ran for. The average score was calculated by taking the policy at a certain time/iteration of the algorithm and by running 1000 episodes of the Frozen Lake with the given policy.*

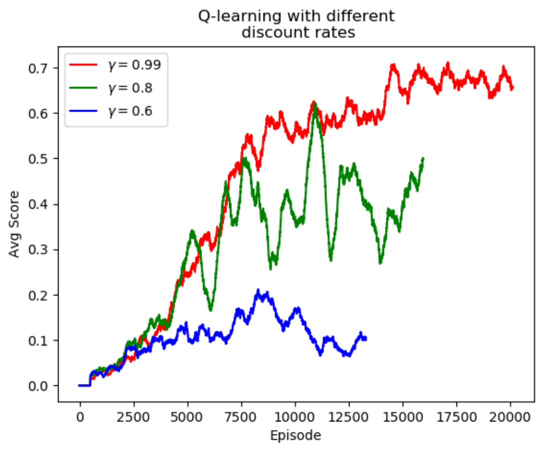


Figure 2: *The average score vs the number of episodes the Q-learner has run for with different values for gamma. The average score was calculated by using the scores from the last 100 episodes of the frozen lake problem. It is clear that the agent 'learns' a better policy when long term rewards are emphasized more.*

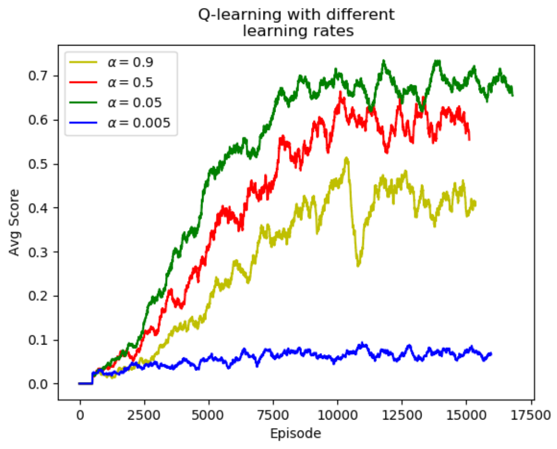


Figure 3: *The average score vs the number of episodes the Q-learner has run for with different values for alpha. The average score was calculated by using the scores from the last 100 episodes of the frozen lake problem. The plot shows no clear pattern for alpha values, but shows that the optimal value to use is close to* $\alpha = 0.05$

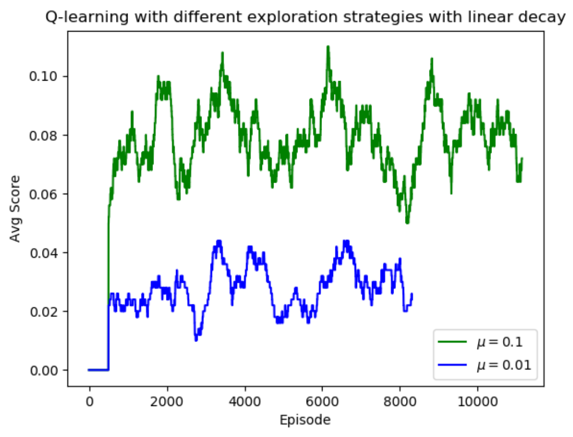


Figure 4: *The average score vs the number of episodes the Q-learner has run for using a linear decay exploration rate with different values of μ. The average score was calculated by using the scores from the last 100 episodes of the frozen lake problem.*

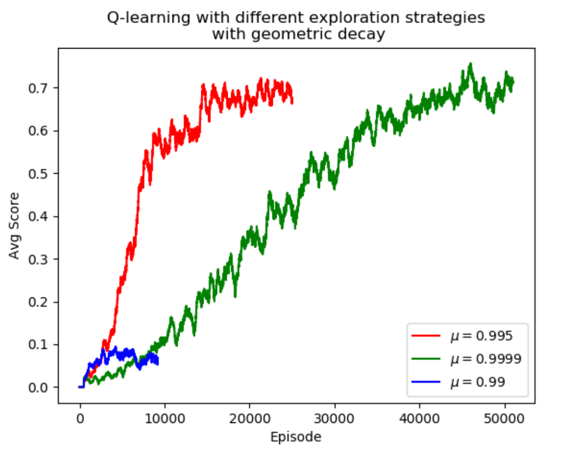


Figure 5: *The average score vs the number of episodes the Q-learner has run for using a geometric decay exploration rate with different values of μ. The average score was calculated by using the scores from the last 100 episodes of the frozen lake problem.*

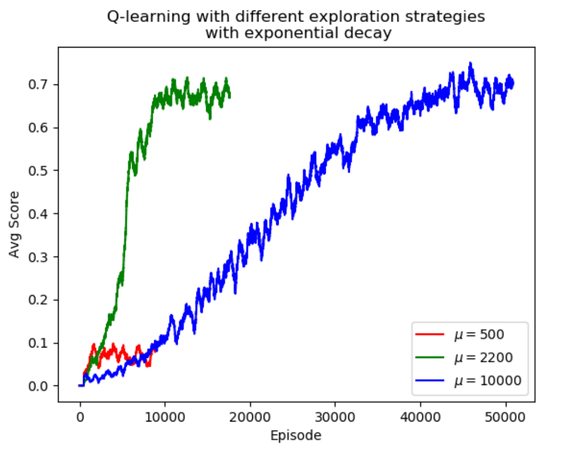


Figure 6: *The average score vs the number of episodes the Q-learner has run for using a exponential decay exploration rate with different values of μ. The average score was calculated by using the scores from the last 100 episodes of the frozen lake problem.*

agent choose to go left or right, then there is only a 33% chance that the agent will fall into a hole because the agent has a 66% chance of 'slipping' and moving to the top or bottom states. In this case, moving left or right yields the same optimality and the algorithm chose the 'left' action over the 'right' action as a tiebreaker.

Another interesting thing to note is how each exploration strategy affected the performance of each Q-learning model. For the frozen lake problem, the exploration strategy that decayed epsilon linearly was clearly the worst strategy because the model showed a clear pattern of minimal improvement over the course of 6,000 episodes. Similarly, the geometric and exponential strategies that decayed epsilon quickly performed poorly as well. This is a clear indication that the Q-learner needs a high exploration rate to continue to improve through the first few thousand episodes, which is why other exploration strategies worked better.

3.2 Mountain Car Problem

The second MDP that was studied in this paper models the mountain car problem. In the mountain car problem, there is a car (agent) that starts somewhere in the valley between two mountains. The goal of the car is to drive up to the top of one of the mountains. However, the car's engine is not powerful enough to climb the mountain in one shot, so it must sway back and forth until it has conserved enough speed and momentum to make it to the top of the mountain. The problem is scored by adding together the number of time steps it takes to reach the top of the mountain. So in contrast to the frozen lake problem, a lower score is better for the mountain car problem. The state space consists of two dimensions, where the first dimension is modeled by the cosine function $cos(x)$ with domain [-1.2, -0.5]. The second dimension specifies the velocity of the car, which can range from [-0.07, 0.07].

The car problem can perform three actions in any given state; apply a force to move in the positive direction, apply a force to move in the negative direction, or do not apply a force and let gravity be the only acting force on the agent. An action $a \in 0, 1, 2$ updates the car to the next state by the following rules:

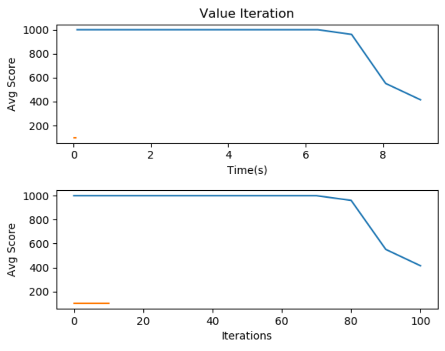
$$s'[2] \leftarrow (a - 1) * 0.001 + cos(3 * s[1]) * (-0.0025)$$
$$s'[1] \leftarrow s[1] + s'[2]$$

Being that the state space is continuous, the value iteration and policy iteration algorithms cannot be used to solve this MDP because there could be an infinite amount of states for the algorithms to iterate over. One way to accommodate for this is to discretize the state space. This can be done by creating a finite set of intervals that are used to help map a continuous value into an index where the index represents which discrete state the continuous value belongs to.

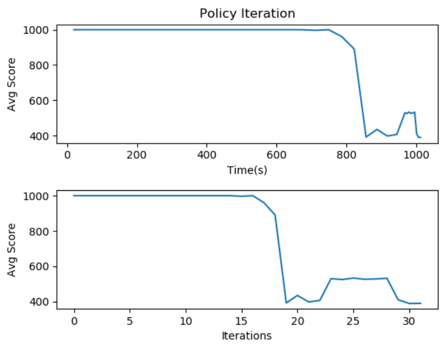
If more intervals are used to discretize the state space, then the solution of the discrete problem will be closer to the optimal solution for the continuous problem. However, the computation time to solve the MDP's with more intervals will increase as well. In the experiments below, the state space of the mountain car problem was discretized using 50 intervals in the first dimension and 50 intervals in the second dimension for a total of $50 \times 50 = 2,500$ total states.

The resulting policies and value functions from value iteration and policy iteration were the same. Figure 7 shows that the policies obtained an average score close to 400. This score is actually quite low compared to the average scores obtained by the various policies from the Q-learning algorithm. Several q-learning policies achieved an average score lower than 200 and some achieved average scores lower than 120.

The difference between the average scores obtained by Q-learning, value iteration, and policy iteration might seem strange because value iteration and policy iteration are supposed to return the optimal policies and the experiments show that they are clearly not doing so. This is because value iteration and policy iteration only consider the points that describe the edge of each interval. On the other hand, the Q-learning



(a) Value Iteration



(b) Policy Iteration

Figure 7: *The average score vs the iterations/time that the policy iteration algorithm has run for. The average score was calculated by taking the policy at a certain time/iteration of the algorithm and by running 100 episodes of the mountain car problem with the given policy.*

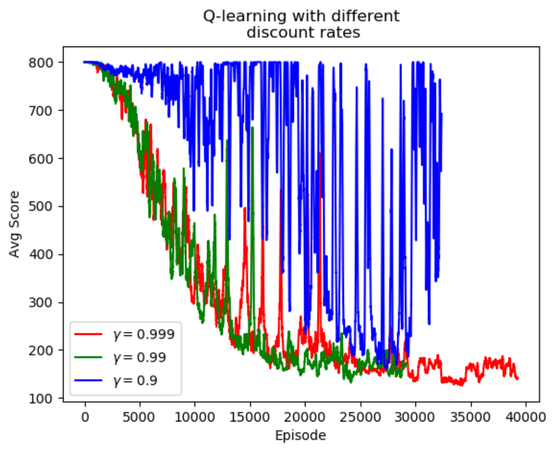


Figure 8: *The average score vs the number of episodes the Q-learner has run for with different values for γ. The average score was calculated by using the scores from the last 100 episodes of the mountain car problem. The plot shows that γ plays a significant role in how the algorithm converges.*

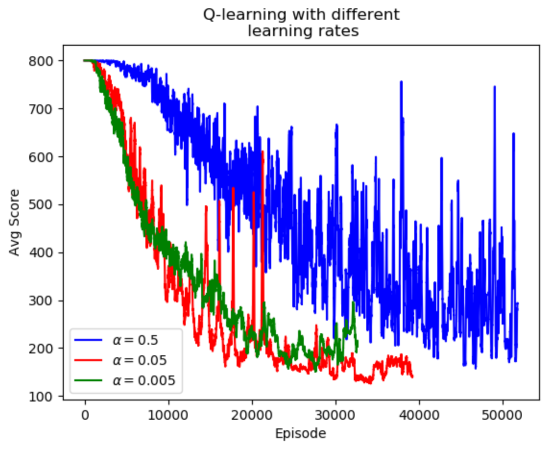


Figure 9: *The average score vs the number of episodes the Q-learner has run for with different values for alpha. The average score was calculated by using the scores from the last 100 episodes of the frozen lake problem. The plot shows how α also plays a significant role in how the algorithm converges.*

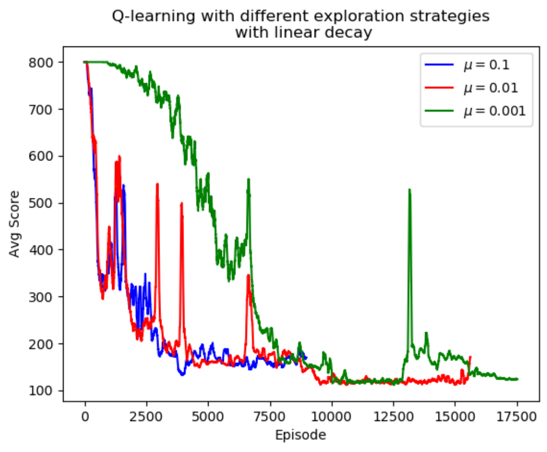


Figure 10: *The average score vs the number of episodes the Q-learner has run for using a linear decay exploration rate with different values of μ. The average score was calculated by using the scores from the last 100 episodes of the mountain car problem.*

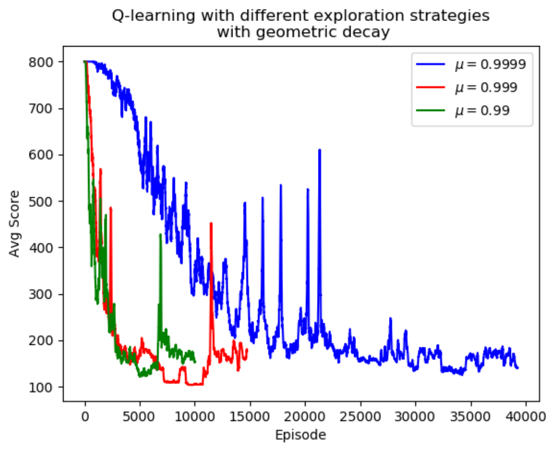


Figure 11: *The average score vs the number of episodes the Q-learner has run for using a geometric decay exploration rate with different values of μ. The average score was calculated by using the scores from the last 100 episodes of the mountain car problem.*

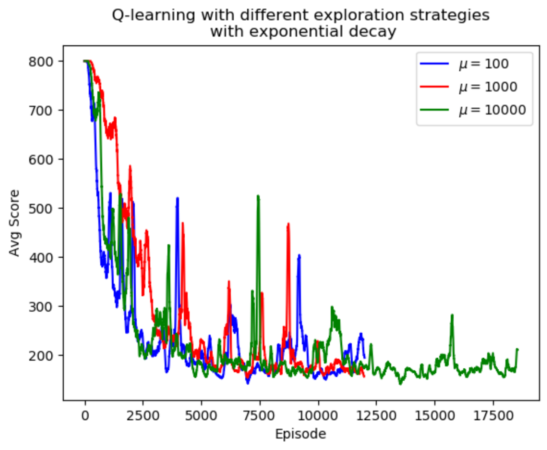


Figure 12: *The average score vs the number of episodes the Q-learner has run for using a exponential decay exploration rate with different values of μ. The average score was calculated by using the scores from the last 100 episodes of the mountain car problem.*

algorithm can encounter points throughout the entire interval of a given discrete state. To show the importance of encountering several points in a given interval, consider two points p_1 and p_2, where p_1 and p_2 belong to the same discrete state space, but $p_1 \neq p_2$. Let p_1' and p_2' denote the next points after the agent acts with some action a from the points p_1 and p_2 It is possible that p_1' and p_2' might belong to different discrete state spaces. This allows the Q-learning algorithm to better estimate the next state an agent will be in after taking a certain action. Although this problem is not stochastic, a similar analogy could be made to stochastic problems in that Q-learning captures the stochasticity of the problem, where as value iteration and policy iteration do not.

Another key difference between the performance of the Q-learning algorithm that was applied to the two problems is in regard to the linear epsilon decay exploration strategy. Figure 4 shows the results of the linear epsilon decay exploration strategy for the frozen lake problem and the results show that the agent could not obtain a better score than 0.1, when the optimal score was approximately 0.7. This illustrates an important subtlety into how fragile the algorithm is for different epsilon values at different stages of the problem. If epsilon decays too quickly, then the agent may fail to discover new and more optimal actions. If epsilon decays too slow, then the agent will seldom see a reward because the random actions that the agent selects will likely lead it to fall into a hole.

On the other hand, the linear epsilon decay exploration strategies seemed to work quite well for the mountain car problem. In fact, the Q-learning algorithm seemed to respond well to greedy methods with little exploration. This is largely due to all of the values in the Q-table being initialized to zero and how a -1 reward was issued for every step that the agent took until the agent arrived at its destination. To visualize why this naturally encourages the agent to explore different actions, imagine a new agent with an initial Q-table in some state $s_i \in S$. The agent can pick any action $a \in \{0, 1, 2\}$ because the Q-table values are all the same:

$$Q[s_i][0] = Q[s_i][1] = Q[s_i][2] = 0$$

Let's assume the agent stochastically picked action 0 as a tie breaker and this action did not lead the agent to it's destination. Then $Q[s_i][0]$ now equals -1, so the next time the agent encounters s_i, it will greedily choose between actions 1 or 2, if the corresponding values for the next states for those actions are the same.

4 Conclusion

This paper has shown how different algorithms impact how quickly and how well an agent can learn an optimal policy in a given environment to achieve some goal. For small problems grid-like problems, value iteration and policy iteration are more efficient than Q-learning because they find the optimal policy much more quickly than Q-learning does. However, value iteration and policy iteration perform less optimally for problems that have continuous state environments because the discretization of the state space might not be entirely representative of the continuous state space. One way to improve upon the basic value iteration and policy iteration algorithms used in the paper, would be to draw a random point from each discrete integral instead of using the same points to represent a given interval.

References

[1] Sutton, R., Barto, A. (2019). Reinforcement Learning: An Introduction. http://incompleteideas.net/book/bookdraft2017nov5.pdf. 11/20/2019.