Mathematical Applications of Reinforcement Learning

Thomas O'Neill

Advisor: Alex Iosevich

Spring 2022 Honors Thesis

Contents

1 Introduction

Consider the following problem: A “Learning Agent” is confined to an n-dimensional grid. They can move in unit vector steps in any dimension, but not diagonally, and they may not leave the grid or move multiple spaces at a time. [4]

Each space on the grid is assigned to a number by a function $P(x_1, ..., x_n)$.

The algorithm takes in the following parameters:

t - the step length used to update the estimation of q-table values

ϵ - the probability of algorithm to take a random move during the learning process

g - the discounting factor for future rewards

N - the number of steps taken during a single epoch

nEpochs - the number of epochs

P - a function that maps the grid to real numbers

The goal of the agent is to walk a path with the maximum P values. For this reason, we have created an algorithm to “reward” the agent for moving from a low P value to a higher one, and a “punishment” for moving from high to low.

2 Algorithm

The algorithm starts with q-table consisting exclusively of zeroes. Consider the grid in figure 2 as an example. Beginning at the start point, the agent has the choice of moving to any of the 4 adjacent points. Since at this point the Q-table is made up entirely of zeroes, this first choice is random.

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 10 \end{bmatrix}$$

Suppose the agent moves from a point with $P = 1$ to the point with $P = 2$. The agent is given a reward of $2 - 1 = 1$ for moving to a higher P-value. This reward is accounted for in the Q-table. The Q-table is a 100×4 table; each row corresponds to a point in the grid, and the 4 columns correspond to the 4 directions the agent may travel from this point. Of course, for corners and edges travel is not possible in all 4 directions, but this is corrected later on by the “limit_moves” function. The Q value corresponding to the point at which the agent started, and the direction the agent traveled, is updated according to the following formula:

$$(1-t)\cdot Q_{init} + t * (reward + g \cdot Q_{max})$$

Q_{init} is the initial Q-value for the point and direction that we are updating, and Q_{max} is the maximum Q-value in the row corresponding to the point that the agent traveled to.

From this equation we can see the effect of the parameters t and g. t determines the proportion that the new Q-value depends on the Q-value of the end space plus the reward, as opposed to its initial value. For this reason, t is referred to as the learning rate, as it determines how much the Q-table is to change after each move. The parameter g is responsible for how much effect the Q-value of the end space has on the new value as compared to the reward. A lower value of g means that the Q-value of a space depends less on the Q-value of surrounding spaces, and for this reason it is called the discounting factor. As we travel along the board, the effect of a Q-value on a given space would decrease exponentially due to this factor. For example, if $t = 1$, a starting space would have Q-value $reward + g \cdot Q_{max}$, but then if this space is traveled to from another space (and this were the maximum Q-value for the space), the other space would have $Q = reward_2 + g \cdot (reward_1 + g \cdot Q_{max})$, which would be on the order of $g^2 \cdot Q_{max}$ for the original value of Q_{max}.

After this first move, the Q-table is no longer entirely filled with zeroes, as now the Q-value for the start point that corresponds with moving in the direction of $P = 2$ will be increased. The next time the agent is at this point, it will with probability ϵ move to a random adjacent point, but with probability $1-\epsilon$ it will move in the direction corresponding to the largest Q-value, choosing randomly in the case of ties. This ϵ value is thus the probability that the agent will "explore" the map, rather than follow his current belief of what the optimal path is. With an ϵ value of 1, the agent would travel completely randomly, and thus obtain a roughly equal amount of information about every location on the map. However, if we lower the value of ϵ, this would give us more information on parts of the map that have high rewards, which may be more important to know.

The agent will move around as such for N steps, updating Q-table accordingly after each one. These N steps mark the completion of a single epoch. The agent returns to the start point for beginning of each epoch, starting with the newly updated Q-table. This allows the agent to go back on its previous decisions based on newly obtained knowledge of the grid.

After running the process for a given number of epochs, we would like to evaluate the learning of the agent. To do this, we call the Q-path function, which creates a path from a given start point based entirely on the optimal directions according to the Q-table. In this sense, it is similar to running the learning algorithm with $\epsilon = 0$. If at any point all the Q-values for a certain point on the grid are negative, then the path stops at this point. The learning process is successful if the agent takes a path that gives the maximum reward for the specified number of steps.

3 Theory

One of the distinguishing features of Q-learning is that it is a *model-free* reinforcement learning algorithm, meaning that it does not depend on a "loss" or "optimization" function like many reinforcement learning algorithms do [1]. Instead, Q-learning is a *values-based* algorithm, which means that it learns using only on the values that it produces, which are of course the Q-values. To put this in the context of the grid-world problem, there is no function that is specifically put into the algorithm that the agent is trying to maximize; However, the result of the algorithm is that the agent tries to find the path to maximize the P values that it travels over. This same result could be achieved with a model-based algorithm where the optimization function is the total sum of P values traveled over, but the use of a function is unnecessary in this case. Another characteristic of values-based learning algorithms is that they update each time based on a set equation.

In order to better understand the theory behind our grid-game algorithm, let us examine the fundamental equations of generic Q-learning.

$$V^\pi(s) = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, \pi]$$

[5]

This is the "V-value" function, which gives the expected return of a state s given a policy π. In the case of our grid simulation, the state is simply the space that the agent is currently on. The other variables are as follows:

π - the "policy" being followed—In our case this is the willingness of the agent to explore or exploit, and is thus dependent on the value of ϵ.

γ - the discount rate applied to future values (exactly the same as our g)

r_t - the return value at step t (the same as $P(x_t)$)

What this formula does is sum up the discounted expected return values for each future step, given that the policy π is being followed starting from state s.

Next we have the Quality Value function, or Q-value function, which is the generic form of the Q-value formula we have been using [5].

$$Q^\pi(s, a) = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi]$$

[5]

It is almost identical to the V-value function, except that it adds the parameter a—the "action" taken from a given state. This is just the direction that the agent moves from a given location.

By taking the difference of these 2 functions, we get the advantage function, which gives us the advantage of taking an action a as compared to the expected return of a given state s [5].

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

[5]

We can also write the equation for Q in terms of the equation for V as follows:

$$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s,a)$$

$\pi(a|s)$ is the probability that action a will be chosen when in state s, which depends on the policy being followed by the agent.

Most importantly, we can break down the equation for Q as such:

$$Q^\pi(s,a) = E[r_t + \sum_{k=1}^{\infty} \gamma^k r_{t+k}]$$

$$= r_t + E[\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}]$$

$$= r_t + \gamma Q^\pi_{t+1}$$

Now we can see the resemblance to our equation for updating the Q-table after each step:

$$Q_{new} = (1-t)Q_{init} + t(reward + gQ_{max})$$

Q^π_{t+1} is equivalent to Q_{max}, since the policy of the agent is to go to the adjacent space with the highest Q-value.

4 Optimization

Now at this point, the natural inclination is to find the optimal values for the parameters—mainly g, t, and ϵ. As a test, let us run the Q-learning process with the following parameters:

$$g = .2, t = 0.8, e = 0.2, N = 2000, nEpochs = 5$$

The resultant Q-table is shown in figure 1 (rounded to 1 decimal place). We see that most of the Q-values are close to 0 except for the last few rows; This is expected, as the very last entry in the grid has $P = 10$, while the surrounding entries have $P = 1$. This will cause the algorithm to produce highly positive Q-values for moving towards this space, and highly negative Q-values for moving away from it.

Next we want to check the Q-path calculated by the algorithm. When we start at the point $(0,0)$ and set the maximum number of steps to 10, we get the following path: $(0,0),(1,0),(2,0),(2,1),(2,2)$. This is exactly what we would expect, as from $(0,0)$ the agent is drawn to the closest point with high reward,

Figure 1: Q-table after Q-learning process run with g=.2, t=0.8, e=0.2, N=2000, nEpochs=5

```
[['0.0', '0.0', '0.0', '0.0'],
 ['0.0', '0.0', '0.0', '0.0'],
 ['0.2', '0.0', '0.0', '0.0'],
 ['0.0', '0.0', '0.0', '0.0'],
 ['0.1', '0.0', '0.0', '0.0'],
 ['0.0', '0.0', '0.0', '0.0'],
 ['0.2', '0.1', '0.0', '0.0'],
 ['0.8', '0.0', '0.0', '0.0'],
 ['0.2', '0.1', '0.0', '0.2'],
 ['0.4', '0.0', '0.0', '0.0'],
 ['0.0', '0.2', '0.0', '0.0'],
 ['0.0', '0.8', '0.0', '0.0'],
 ['-0.8', '-0.8', '-0.8', '-0.8'],
 ['0.4', '0.4', '0.0', '0.8'],
 ['1.8', '0.0', '0.1', '0.2'],
 ['0.0', '0.0', '0.0', '0.0'],
 ['0.1', '0.2', '0.2', '0.0'],
 ['0.4', '0.4', '0.8', '0.0'],
 ['1.8', '1.8', '0.2', '0.2'],
 ['9.0', '0.0', '0.4', '0.4'],
 ['0.0', '0.0', '0.0', '0.0'],
 ['0.0', '0.4', '0.0', '0.0'],
 ['0.0', '1.8', '0.2', '0.1'],
 ['0.0', '9.0', '0.4', '0.4'],
 ['0.0', '0.0', '-7.2', '-7.2']]
```

which is $(2, 2)$ with $P = 2$. The path then stops at this point, as all the surrounding points have lower P values, so the Q-values here are all negative (seen in the row filled with -0.8 in figure 1). If we instead start our path closer to the bottom-right corner of the map—say at point $(4, 1)$, we will get a path that leads us to $(4, 4)$, the other outstanding P value in the grid.

Instead of running the path algorithm over and over, it would be useful to have a function that gives us an idea of the resultant path starting from each point. For this, we have the QTableTopology function. This gives us a grid mirroring our original, but the values are the P value of the endpoint of the Q-path starting at that point.

We have seen that paths will tend to end at points with P values higher than the surrounding points. Thus, our resulting QTableTopology grid (shown in figure 2), all the values are either 2 or 10, the two extreme P values in the grid.

Figure 2: QTableTopology after Q-learning process run with g=.2, t=0.8, e=0.2, N=2000, nEpochs=5

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 10 \\ 2 & 2 & 2 & 2 & 10 \\ 2 & 2 & 2 & 2 & 10 \\ 2 & 2 & 2 & 10 & 10 \\ 2 & 10 & 10 & 10 & 10 \end{bmatrix}$$

We can use this topology function in the subsequent sections to help find the optimal values for the parameters.

4.1 The Discount Factor

The need for the discount factor, g, stems from the fact that in many applications of this problem, we do not want to consider an infinite number of steps in the future, but we also do not want to consider only the next immediate step. To visualise this better, let us consider both of these extreme cases.

First, consider having $g = 1$, which effectively means that there is no discount rate. Then the agent will only care about reaching the area of the grid with maximum possible reward at some point; It will not care about the rewards picked up along the way. This would be useful if we are just trying to find the maximum value of P in the grid, but there are better algorithms for doing that. If we were to calculate a Q-path from this algorithm, it may be the case that our path is too short to reach this maximum reward, and so our algorithm has failed to find the optimal path for a given length.

Now consider the opposite extreme, which is that $g = 0$, or only the immediate next reward matters to the agent. In our example grid from figure 2, if the agent started in the top-left corner, it would be completely indifferent between all the surrounding points, as all of them lead to the same reward.

Lastly, let us consider the application of an intermediate g value. Suppose we are training the agent for the following game:

On each step taken, there is exactly p probability that the game will end. Try and maximize the total reward obtained before this happens.

This means the actual value of a reward r k steps in the future is $(1-p)^k r$, as obtaining this reward is dependent on the $(1-p)^k$ probability that the game does not end before we reach it. This means we should set our discount factor to $1 - p$.

In short, the optimization of the discount factor ultimately depends on the rules of the game, and what we are trying to achieve. If the goal is to simply maximize total average reward in an infinite number of steps, then we would take the discount factor of 1, whereas if the goal is to find the optimal next step, we would take the discount factor of 0. Lastly, if we are in a situation such as the paragraph above where the game ends with probability p, we would take

Figure 3: QTableTopology after Q-learning process run with g=.1, t=0.1, e=0.1, N=2000, nEpochs=5

$$\begin{bmatrix} 2 & 2 & 2 & 10 & 10 \\ 2 & 2 & 2 & 10 & 10 \\ 2 & 2 & 2 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 \\ 1 & 10 & 10 & 10 & 10 \end{bmatrix}$$

Figure 4: QTableTopology after Q-learning process run with g=.8, t=0.1, e=0.1, N=2000, nEpochs=5

$$\begin{bmatrix} 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 \end{bmatrix}$$

$g = 1 - p$.

The topology table output is greatly impacted by the discount factor; If we have a discount factor close to 1, we would expect the topology table to contain mostly 10's, as then the algorithm would be mainly interested in finding the maximum value. Otherwise, points closer to the $P = 2$ space would have a topology value of 2, as a lower discount rate means the path is more interested in the nearest reward, not necessarily the highest. We can see this by comparing figures 3 and 4, where the only difference is the discount factor.

4.2 The Learning Rate

The next parameter to consider is the learning rate t. If we look at the update formula again:

$$(1 - t) \cdot Q_{init} + t * (reward + g \cdot Q_{max})$$

we see that t determines the proportion that the Q value depends on the newly learned information as opposed to its initial value. Thus, if we had a t value of 0, the Q values would never change at all, and if we had a t value of 1, the Q values would solely depend on their most recent updates.

In figures 5, 6, and 7 we run the same simulation with t values of .2, .5, and .8 respectively (using the P grid from 2).

The differences are not immediately obvious, but there are few that pop out. For one thing, the values in the first column for the first few rows are considerably higher when t is larger. The first column in the Q-table corresponds to moving "down" from a certain point on the grid. Thus, if we refer to the

Figure 5: Q-table after Q-learning process run with g=.8, t=0.2, e=0.2, N=2000, nEpochs=5

```
[['0.25', '0.00', '0.00', '0.00'],
 ['0.00', '0.00', '0.00', '0.00'],
 ['0.00', '0.12', '0.00', '0.00'],
 ['0.78', '0.00', '0.00', '0.00'],
 ['0.00', '0.00', '0.00', '0.00'],
 ['1.20', '0.00', '0.00', '0.00'],
 ['2.29', '0.49', '0.00', '0.00'],
 ['3.33', '0.00', '0.01', '0.84'],
 ['3.50', '0.09', '0.00', '1.12'],
 ['1.89', '0.00', '0.00', '0.85'],
 ['0.00', '2.55', '0.12', '0.00'],
 ['0.49', '3.40', '0.80', '1.07'],
 ['3.00', '1.81', '1.48', '1.66'],
 ['5.00', '2.90', '2.09', '3.40'],
 ['4.00', '0.00', '0.61', '3.19'],
 ['0.00', '0.00', '0.61', '0.00'],
 ['1.38', '3.78', '2.72', '0.04'],
 ['3.20', '5.00', '3.40', '2.69'],
 ['-5.00', '-5.00', '-5.00', '-5.00'],
 ['3.20', '0.00', '3.20', '5.00'],
 ['0.00', '0.29', '0.00', '0.00'],
 ['0.00', '3.16', '0.00', '0.00'],
 ['0.00', '4.00', '3.46', '2.23'],
 ['0.00', '3.20', '5.00', '3.20'],
 ['0.00', '0.00', '4.00', '3.91']]
```

Figure 6: Q-table after Q-learning process run with g=.8, t=0.5, e=0.2, N=2000, nEpochs=5

```
[['0.00', '1.49', '0.00', '0.00'],
 ['2.02', '0.00', '0.00', '0.00'],
 ['0.00', '1.60', '0.00', '0.22'],
 ['3.20', '0.36', '0.00', '0.00'],
 ['1.73', '0.00', '0.00', '0.00'],
 ['0.00', '0.00', '0.21', '0.00'],
 ['0.00', '2.56', '0.00', '0.00'],
 ['2.82', '3.20', '0.40', '1.95'],
 ['4.00', '2.55', '2.56', '2.56'],
 ['3.20', '0.00', '0.00', '1.60'],
 ['0.00', '0.00', '0.00', '0.00'],
 ['0.00', '2.76', '0.00', '0.00'],
 ['3.00', '2.78', '1.56', '0.51'],
 ['5.00', '3.20', '3.20', '3.40'],
 ['4.00', '0.00', '2.56', '4.00'],
 ['0.00', '0.00', '0.00', '0.00'],
 ['1.41', '4.00', '0.00', '0.00'],
 ['3.20', '5.00', '3.40', '3.19'],
 ['-5.00', '-5.00', '-5.00', '-5.00'],
 ['3.20', '0.00', '3.20', '5.00'],
 ['0.00', '0.00', '0.00', '0.00'],
 ['0.00', '3.09', '0.00', '0.00'],
 ['0.00', '3.25', '4.00', '1.75'],
 ['0.00', '3.20', '5.00', '3.20'],
 ['0.00', '0.00', '4.00', '4.00']]
```

Figure 7: Q-table after Q-learning process run with g=.8, t=0.8, e=0.2, N=2000, nEpochs=5

```
[['1.74', '0.69', '0.00', '0.00'],
 ['0.00', '2.09', '0.00', '0.00'],
 ['2.72', '0.00', '0.00', '0.00'],
 ['3.19', '0.00', '0.00', '1.72'],
 ['2.45', '0.00', '0.00', '0.00'],
 ['1.33', '2.18', '0.00', '0.00'],
 ['2.72', '1.39', '1.13', '1.39'],
 ['3.40', '0.00', '0.91', '2.18'],
 ['4.00', '2.56', '2.50', '2.72'],
 ['3.20', '0.00', '1.30', '3.05'],
 ['0.00', '2.72', '1.74', '0.00'],
 ['2.90', '3.40', '2.16', '2.12'],
 ['3.00', '3.00', '1.72', '1.72'],
 ['5.00', '3.20', '3.20', '3.40'],
 ['4.00', '0.00', '2.56', '4.00'],
 ['0.00', '0.00', '2.01', '0.00'],
 ['0.00', '4.00', '2.72', '0.89'],
 ['3.20', '5.00', '3.40', '3.20'],
 ['-5.00', '-5.00', '-5.00', '-5.00'],
 ['3.20', '0.00', '3.20', '5.00'],
 ['0.00', '0.00', '0.00', '0.00'],
 ['0.00', '3.20', '2.56', '0.00'],
 ['0.00', '4.00', '4.00', '2.56'],
 ['0.00', '3.20', '5.00', '3.20'],
 ['0.00', '0.00', '4.00', '4.00']]
```

P grid, we can see this means that the agent is more drawn to the space with $P = 2$ when we increase the learning rate.

If we look at the Q values for the points around the $P = 10$ spot on the grid, there is not as much of a difference when we change the learning rate. There is a slight difference from $t = .2$ to $t = .5$, but after that the change is barely noticeable. The explanation for this is that since the P value of 10 is already so much higher than its surrounding P values, it does not take long for the algorithm to "learn" that it should move towards this point. Thus, we can have a lower learning rate and still achieve basically the same result, as long as we run the learning process enough times.

This discovery motivates the following question: What is the advantage of choosing a lower learning rate?

If we choose a high learning rate, we could get into a situation where the Q value for a given space oscillates wildly between 2 values [2]. It has been widely established that the ideal learning should decay over time; this allows us to pick up initial values quickly with a high learning rate at the beginning, then exponentially decrease the learning rate so that we reach an "equillibrium" in the Q-table [2].

Another factor we would want to consider would be the layout of the P-grid. So far we have only considered the case of a grid with mostly zero's and a single extreme value. To see how the learning rate affects a more evenly distributed grid, consider the following grid of random values uniformly distributed between -10 and -10:

$$P = \begin{bmatrix} 10 & -10 & -10 & 3 & -8 \\ -9 & 3 & -6 & -5 & -6 \\ 5 & 2 & 0 & 7 & 8 \\ -5 & 5 & 1 & -9 & -8 \\ 0 & -8 & 1 & -10 & -4 \end{bmatrix}$$

4.3 The Exploration Rate

Choosing the ideal exploration rate depends entirely on the context for which we are using the algorithm. If the goal is for the agent to have the best overall "understanding" of the grid, then $\epsilon = 1$ is the best choice, as then we will travel to all the spaces with equal likelihood. However, in most uses of this algorithm, our goal is just to find the best possible path, and we do not care about having knowledge of the grid outside of that.

It is clear that choosing $\epsilon = 0$ would not be useful, as then as soon as a point on the table has low Q values, it will never be visited again. This leads to the potential of getting a biased view of the grid based on just the first few Q-updates.

A comparison of $\epsilon = .2$ vs. $\epsilon = .8$ is shown in figures 10 and 11 respectively. Similar to when we increased the learning rate, the Q values in the top-left are much higher in figure 11 than figure 10, while the values at the bottom of the table do not change much. The likely reason for this is that the lower

Figure 8: Q-table for randomly generated grid after Q-learning process run with g=.8, t=0.2, e=0.2, N=2000, nEpochs=5

```
[['-15.20', '-16.00', '0.00', '0.00'],
 ['12.92', '2.60', '0.00', '20.00'],
 ['5.78', '13.00', '0.00', '4.00'],
 ['-5.56', '-8.80', '0.00', '-10.40'],
 ['4.80', '0.00', '0.00', '11.00'],
 ['14.00', '11.92', '19.00', '0.00'],
 ['-0.40', '-7.22', '-9.00', '-8.20'],
 ['5.95', '3.44', '-1.39', '8.92'],
 ['12.20', '1.80', '8.00', '0.75'],
 ['14.00', '0.00', '0.20', '3.44'],
 ['-8.00', '-2.40', '-10.20', '0.00'],
 ['2.52', '-0.56', '0.92', '3.00'],
 ['1.70', '7.20', '-4.21', '2.60'],
 ['-12.76', '1.00', '-9.56', '-5.56'],
 ['-12.80', '0.00', '-11.20', '-0.80'],
 ['5.00', '9.52', '10.00', '0.00'],
 ['-10.50', '-3.30', '-2.40', '-7.95'],
 ['0.11', '-6.81', '0.44', '3.52'],
 ['0.20', '4.17', '16.20', '10.70'],
 ['4.00', '0.00', '16.00', '2.24'],
 ['0.00', '-5.49', '-2.88', '0.00'],
 ['0.00', '0.00', '12.52', '6.40'],
 ['0.00', '-7.39', '0.70', '-6.24'],
 ['0.00', '6.00', '4.02', '11.14'],
 ['0.00', '0.00', '-0.80', '-3.77']]
```

Figure 9: Q-table for randomly generated grid after Q-learning process run with g=.8, t=0.8, e=0.2, N=2000, nEpochs=5

```
[['-3.80', '-4.00', '0.00', '0.00'],
 ['15.56', '12.80', '0.00', '20.00'],
 ['12.25', '14.65', '0.00', '16.00'],
 ['2.23', '-2.20', '0.00', '-0.20'],
 ['6.93', '0.00', '0.00', '12.57'],
 ['14.96', '14.56', '19.00', '0.00'],
 ['1.85', '0.22', '3.00', '3.20'],
 ['4.80', '7.40', '8.80', '11.56'],
 ['12.80', '-0.80', '8.00', '3.94'],
 ['14.00', '0.00', '7.51', '0.00'],
 ['-1.23', '-0.15', '1.20', '0.00'],
 ['2.69', '0.44', '3.56', '0.00'],
 ['3.80', '0.00', '-0.06', '0.00'],
 ['-3.89', '1.00', '-2.29', '-4.28'],
 ['-3.35', '0.00', '-2.87', '-0.20'],
 ['5.00', '0.00', '10.96', '0.00'],
 ['-5.57', '-1.24', '-0.15', '-1.66'],
 ['0.00', '-8.00', '-0.80', '3.80'],
 ['0.00', '0.00', '16.51', '0.00'],
 ['3.20', '0.00', '15.97', '0.00'],
 ['0.00', '-1.08', '2.73', '0.00'],
 ['0.00', '10.31', '0.00', '0.00'],
 ['0.00', '-8.80', '2.84', '-1.84'],
 ['0.00', '0.00', '0.00', '12.19'],
 ['0.00', '0.00', '0.00', '0.83']]
```

exploration rate does not provide as much opportunity for the algorithm to discover the space with $P = 2$, whereas it quickly learns about the space with $P = 10$, since its P value is so high. Once the algorithm discovers that the highest value is towards the bottom-right of the graph, it will not explore other parts as much, especially with a low exploration rate.

In almost all cases, the algorithm will learn better with a higher exploration rate, but we would also want to take into account the efficiency of the algorithm and the number of epochs necessary to achieve the desired result. In situations where we are only concerned with finding the best possible path, and there is no motivation to learn other parts of the table, it would be a better use of resources to lower the exploration rate so that the algorithm could spend more time learning the optimal path.

Figure 10: Q-table Q-learning process run with g=.8, t=0.2, e=0.2, N=2000, nEpochs=5

```
[['0.41', '0.00', '0.00', '0.00'],
 ['0.00', '0.20', '0.00', '0.00'],
 ['0.60', '0.00', '0.00', '0.00'],
 ['3.00', '0.99', '0.00', '0.12'],
 ['2.21', '0.00', '0.00', '1.22'],
 ['1.27', '0.00', '0.00', '0.00'],
 ['0.00', '0.85', '0.00', '0.00'],
 ['3.40', '1.60', '0.00', '0.00'],
 ['4.00', '2.15', '1.77', '2.67'],
 ['0.00', '0.00', '0.99', '3.20'],
 ['2.33', '0.00', '0.00', '0.00'],
 ['1.60', '3.40', '0.00', '0.00'],
 ['2.98', '3.00', '1.72', '1.71'],
 ['5.00', '3.20', '3.20', '3.40'],
 ['4.00', '0.00', '2.53', '4.00'],
 ['0.00', '3.20', '0.00', '0.00'],
 ['2.49', '4.00', '2.66', '2.50'],
 ['3.20', '5.00', '3.40', '3.20'],
 ['-5.00', '-5.00', '-5.00', '-5.00'],
 ['3.20', '0.00', '3.20', '5.00'],
 ['0.00', '1.87', '0.00', '0.00'],
 ['0.00', '2.40', '3.20', '0.49'],
 ['0.00', '3.97', '4.00', '2.41'],
 ['0.00', '3.20', '5.00', '3.20'],
 ['0.00', '0.00', '4.00', '3.99']]
```

Figure 11: Q-table Q-learning process run with g=.8, t=0.2, e=0.8, N=2000, nEpochs=5

```
[['1.74', '1.74', '0.00', '0.00'],
 ['2.18', '2.18', '0.00', '1.39'],
 ['2.72', '2.56', '0.00', '1.74'],
 ['3.20', '2.05', '0.00', '2.18'],
 ['2.56', '0.00', '0.00', '2.56'],
 ['2.18', '2.18', '1.39', '0.00'],
 ['2.72', '2.72', '1.74', '1.74'],
 ['3.40', '3.20', '2.18', '2.18'],
 ['4.00', '2.56', '2.56', '2.72'],
 ['3.20', '0.00', '2.05', '3.20'],
 ['2.56', '2.72', '1.74', '0.00'],
 ['3.20', '3.40', '2.18', '2.18'],
 ['3.00', '3.00', '1.72', '1.72'],
 ['5.00', '3.20', '3.20', '3.40'],
 ['4.00', '0.00', '2.56', '4.00'],
 ['2.05', '3.20', '2.18', '0.00'],
 ['2.56', '4.00', '2.72', '2.56'],
 ['3.20', '5.00', '3.40', '3.20'],
 ['-5.00', '-5.00', '-5.00', '-5.00'],
 ['3.20', '0.00', '3.20', '5.00'],
 ['0.00', '2.56', '2.56', '0.00'],
 ['0.00', '3.20', '3.20', '2.05'],
 ['0.00', '4.00', '4.00', '2.56'],
 ['0.00', '3.20', '5.00', '3.20'],
 ['0.00', '0.00', '4.00', '4.00']]
```

5 Shortest Path Algorithm

One of the most important algorithms in computer science is the “shortest path algorithm.” Given a graph with vertices connected by edges, with weights on each edge, the objective is to find the path between any two vertices that covers the lowest sum of edge weights.

We can translate this into our grid algorithm in the following manner [3]. Designate each point on the grid as an edge. Adjacent points correspond to edges that stem from the same vertex. The P value for each point corresponds to the edge weight, except we will make it the negative of this value, as we want to have a negative reward for taking longer paths. If it is not possible to plot all the given edges in a way that properly marks adjacency, we can fix this by adding more dimensions and filling in blank spaces with P values of 0, so that they will not impact the algorithm. Consider the graph in figure 12.

Figure 12: Sample graph with 4 vertices

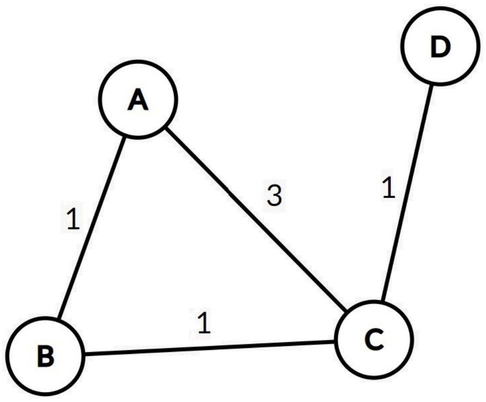


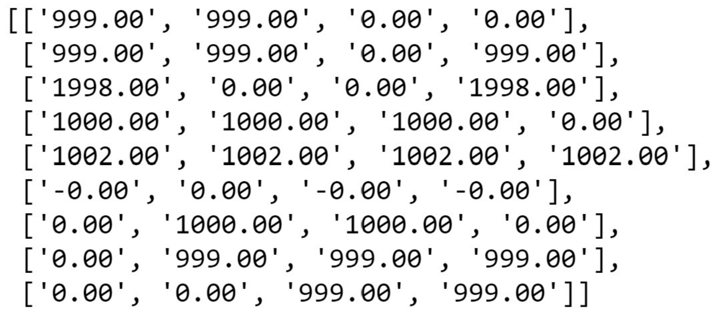
Figure 13: Grid corresponding to graph in figure 12

$$P = \begin{bmatrix} 0 & 0 & -999 \\ -1 & -3 & -1 \\ -1 & 0 & 0 \end{bmatrix}$$

We can convert this to the grid in figure 13. The -1 in the bottom-left corner represents the edge from vertex B to vertex C. The -1 above this point represents the edge from vertex A to B, the -3 represents the edge from A to C, and the last -1 represents the edge from C to D. We can see that in the grid, it is impossible to get from the top-left corner to the top-right without passing through another nonzero value. This corresponds to how in the graph, we cannot get from edge AB to edge CD without passing through one of the other edges. The adjacency also matches up, as the point corresponding to AB is adjacent to the points corresponding to both AC and BC. It appears as though the point corresponding to BC is not adjacent to that of CD, however, since we can traverse between these points without passing through any nonzero points, this counts as adjacent for our purposes. Lastly, we need to put an arbitrarily low value in the top-right corner so that it is not possible to travel from edge AB to CD without passing through any other edges.

Say we want to find the shortest path from vertex A to vertex D. In order to properly train the algorithm, we would need to start at a point adjacent to all edges touching vertex A and give a high reward for reaching any edge touching

Figure 14: Q-table after Q-learning process run on grid in figure 13 with g=.8, t=0.8, e=0.8, N=2000, nEpochs=5



```
[['999.00', '999.00', '0.00', '0.00'],
 ['999.00', '999.00', '0.00', '999.00'],
 ['1998.00', '0.00', '0.00', '1998.00'],
 ['1000.00', '1000.00', '1000.00', '0.00'],
 ['1002.00', '1002.00', '1002.00', '1002.00'],
 ['-0.00', '0.00', '-0.00', '-0.00'],
 ['0.00', '1000.00', '1000.00', '0.00'],
 ['0.00', '999.00', '999.00', '999.00'],
 ['0.00', '0.00', '999.00', '999.00']]
```

vertex D. Thus, we would need to change the -1 on the right to 999, as to give an arbitrarily high reward for reaching the end point.

Now we test the Q-learning algorithm on this grid. Since we are just trying to find the optimal total sum over a path, there is no reason for a discount rate, as future rewards matter just as much as current ones. Thus, we set $g = 1$. Using t, ϵ equal to 0.8, and start point in the top-left, we get the Q-table in figure 14.

At first glance, a few things pop out about the Q-table. There are many points where the algorithm is indifferent between 2 directions. This makes sense as there are a few different options we could traverse diagonals and still get the same path in terms of the graph.

In order to find the most efficient path, we would want to use the "Q-path" function from earlier. The output of the function is as follows:

$$(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (1, 2)$$

This corresponds to the following traversal of edges: A, AC, BC, C, C, CD; and the equivalent traversal of vertices: A, B, C, D. We can verify from figure 12 that this is in fact the shortest path from A to D in terms of total edge weights.

References

[1] *A Beginner's Guide to Q-Learning.* URL: `https://towardsdatascience.com/interactive-q-learning-9d9203fdad70`.

[2] Eyal Even-Dar and Yishay Mansour. "Learning Rates for Q-learning". In: *Journal of Machine Learning Research 5* 25.1 (2003).

[3] *Finding Shortest Path using Q-Learning Algorithm.* URL: `https://towardsdatascience.com/finding-shortest-path-using-q-learning-algorithm-1c1f39e89505`.

[4] *Train Reinforcement Learning Agent in Basic Grid World.* URL: `https://www.mathworks.com/help/reinforcement-learning/ug/train-q-learning-agent-to-solve-basic-grid-world.html`.

[5] *Value-based Methods in Deep Reinforcement Learning.* URL: `https://towardsdatascience.com/value-based-methods-in-deep-reinforcement-learning-d40ca1086e1`.