INM 426 – Software Agents

Lego Home Finding with Q-learning

Harry Li, Xin Li

Introduction

Q-learning is one of the early breakthroughs in reinforcement learning developed by Chris Watkins in 1989 when he was a graduate student at Cambridge University. The algorithm is efficient and simple. It is still one of the most popular algorithms today. This report shows the implementation of Q-learning in a navigation task. The impact of different parameters (e.g. learning rate, discount rate, exploration factor, decay) are studied. A grid search is performed to find the optimal combination. Furthermore, Double Q-learning, invented to solve overestimation from Q-learning, is implemented to understand the differences to Q-learning. In addition, the parallelism between Q-learning and psychological learning theories is discussed.

1. Basic

1.1 Domain and Task

The domain is the central part of London Underground with random starting station and a defined destination station of Leicester Square (Station 7 in Figure 2) where the Lego store is based. It is an episodic finite navigation task. The agent is set to find the shortest path from the random starting station to the destination.

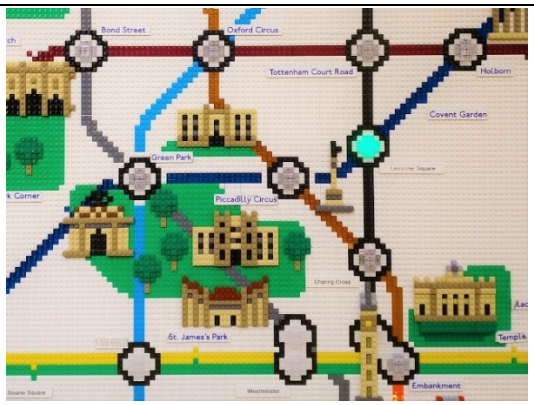

Figure 1. London Underground Lego map

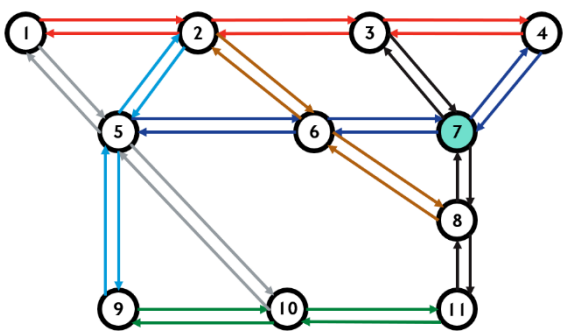

Figure 2. Graphic representation of the domain

1.2 State Transition Function

The state provides the agent basis to make an action. It has the Markov property that every state includes all the information in the past and enables the agent to make future interaction with the environment. The state transition function represents the successor state an agent could be ended up after taking action. It is part of the environment outside the control of the agent. In Dynamic Programming (DP), the state transition is controlled by the state transition probabilities. In Monte Carlo (MC) and Temporal Difference (TD) methods, the transition probabilities are generally unknown to the agent. Thus the agent has an equal probability to transit into any possible successor state including the current one. [4]

Based on graphic representation in Figure 2, the state transition function can be expressed as:

State (s) → Next State (s')			
1→ {1, 2, 5}	4→ {3, 4, 7}	7→ {3, 4, 6, 7, 8}	10→ {5, 9, 10, 11}
2→ {1, 2, 3, 5, 6}	5→ {1, 2, 5, 6, 9, 10}	8→ {6, 7, 11}	11→ {8, 10, 11}
3→ {2, 3, 4, 7}	6→ {2, 5, 6, 7, 8}	9→ {5, 9, 10}	

Table 1. State transition function

1.3 Reward Function

The reward signal is the way to communicate to the agent in terms of what to achieve. It is one of the fundamental ideas of reinforcement learning. In general, the goal of the agent is to maximise the cumulative rewards receive in the long run. The reward function is defined as R-matrix in section 1.5 below. It represents the immediate reward an agent receives after transitioning from one state to the next. And it is the input to the Bellman optimality equation follows the Markov Decision Process (MDP). [4]

1.4 Policy

Policy (π) is a mapping from states to each possible action. Policy is what the agent ultimately learns through iteratively evaluate and improve state-value function v or action-value function q. This process is called Generalised Policy Iteration. One way to improve the policy is to take a greedy approach in respect of value function follow Bellman optimality equation. However, always take the greedy approach would make agent leave many options unexplored. Therefore, to balance between exploration and exploitation, an exploration factor ($\varepsilon \in (0,1)$) is introduced. ε-greedy policy is one of the simplest ideas to balance between exploration and exploitation, but effective. With the probability of ε, the agent takes a random move instead of greedily. Furthermore, a decay factor for ε is also employed in this report. This allows the agent to explore more at the beginning and gradually less towards the end. This assumes that agent explored enough in early episodes and is claimed effective. [4]

1.5 Graphic Representation and R-matrix

The R-matrix is demonstrated on the right. Rows show the state (s) and columns show the next state (s') after taking an action. Numbers represent the immediate reward an agent receives after taking an action. 0 for every action unless arriving at the destination. "-" represents null value indicates no link between stations (i.e. invalid action). 100 for reaching the destination, Station 7.

The graphic representation of the domain is shown in Figure 2 above.

State \ Action	1	2	3	4	5	6	7	8	9	10	11
1	0	0	-	-	0	-	-	-	-	-	-
2	0	0	0	-	0	0	-	-	-	-	-
3	-	0	0	0	-	-	100	-	-	-	-
4	-	-	0	0	-	-	100	-	-	-	-
5	0	0	-	-	0	0	-	-	0	0	-
6	-	0	-	-	0	0	100	0	-	-	-
7	-	-	0	0	-	0	100	0	-	-	-
8	-	-	-	-	-	0	100	0	-	-	0
9	-	-	-	-	0	-	-	-	0	0	-
10	-	-	-	-	0	-	-	-	0	0	0
11	-	-	-	-	-	-	-	0	-	0	0

1.6 Parameters

Q-learning is an off-policy TD control algorithm which is presented below. The new $Q(S_t, A_t)$ is equal to the old $Q(S_t, A_t)$ plus TD error (i.e. immediate reward after taking an action and maximum one-step ahead action-value function minus old Q value). α is learning rate and γ is discount rate. [4]

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max Q(S_{t+1}, a) - Q(S_t, A_t)]$$

1.6.1 Learning Rate

Bootstrapping is one property of Q-learning and TD model in general. It means the current update is based on previous estimates. Learning rate ($\alpha \in (0,1]$) is used to control how much of the agent takes into account the new update providing it is an estimation. We might not want to have a full update but instead steadily steps toward the right direction. When α close to 0, the agent only takes a small step towards the updated Q value. When α approaching 1, the agent takes a full update from the new value [2,4]. The learning rate is initially set as 0.9 for simplicity as Even-Dar and Mansour [2] suggested around 0.85 is a good start. Beleznay et al. [1] suggested to use a fixed learning rate for faster convergence, thus learning rate is not decayed.

1.6.2 Discount Rate

Discount rate ($\gamma \in (0,1]$) is used to represents the uncertainty of the estimates and it determines the present value of it. It's also a mathematical convenience to turn an infinite horizon problem to a finite one. When γ close to 0, the agent tends to maximise the immediate and short-term reward. When γ approaching 1, the agent takes more consideration of future rewards. A discount rate of 0.8 is used as the initial setting. [4]

1.6.3 Exploration and Decay Factor

As noted in section 1.4 that ε-greedy policy with decay factor is implemented in this report. The larger the exploration factor and the larger the decay factor, the more agent would explore instead of acting greedily towards the short-term maximum value. We initially set exploration factor to be 0.9 and decay to be 0.999 in a view to let the agent explore more at the early episodes. The table below summarised the initial configuration of parameters.

Summary of initial parameters configuration			
Learning Rate (α)	0.9	Exploration Factor (ε)	0.9
Discount Rate (γ)	0.8	Decay factor (df)	0.999

Table 2. Summary of initial parameters configuration

1.7 Q-matrix Update

To demonstrate Q-learning, 2 episodes of Q-matrix asynchronous update is illustrated below with the parameters stated above ($\alpha = 0.9, \gamma = 0.8$). In addition, the pseudocode of Q-learning algorithm is presented in the box below [4].

Q-learning (off-policy TD control) pseudocode

Set algorithm parameters: $\alpha \in (0,1], \gamma \in (0,1], \varepsilon \in (0,1), df \in (0,1]$
Initialise $Q(s,a)$, arbitrarily set $Q(s,a) = 0$
Loop for each episode:
  Initialise S
  Loop for each step of episode:
    Choose A from S using ε-greedy policy
    Take action A, observe R, S'
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max Q(S_{t+1}, a) - Q(S_t, A_t)]$
    $S \leftarrow S'$
  until S is terminal

Episode 1: Station 6 to 7

Step 1: initial Q-matrix, and set current state at Station 6

State \ Action	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0

Step 2: choose to go to Station 7

$$Q(6,7) = Q(6,7) + \alpha\{R(6,7) + \gamma\max[Q(7,3), Q(7,4), Q(7,6), Q(7,7), Q(7,8)] - Q(6,7)\}$$

$$Q(6,7) = 0 + 0.9(100 + 0.8 \times \max[0,0,0,0,0] - 0) = 90$$

Station 7 becomes the current state. Since Station 7 is the terminal state, this episode ends here.

State \ Action	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	90	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0

Episode 1: Station 2 to 7

Step 1: go to Station 6 follow policy

$$Q(2,6) = Q(2,6) + \alpha\{R(2,6) + \gamma\max[Q(6,2), Q(6,5), Q(6,6), Q(6,7), Q(6,8)] - Q(2,6)\}$$

$$Q(2,6) = 0 + 0.9(0 + 0.8 \times \max[0,0,0,90,0] - 0) = 65$$

Station 6 becomes the current state.

State \ Action	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	65	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	90	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0

Step 2: go to Station 7 follow policy

$$Q(6,7) = Q(6,7) + \alpha\{R(6,7) + \gamma\max[Q(7,3), Q(7,4), Q(7,6), Q(7,7), Q(7,8)] - Q(6,7)\}$$

$$Q(6,7) = 90 + 0.9(100 + 0.8 \times \max[0,0,0,0,0] - 90) = 99$$

Station 7 becomes the current state. Since Station 7 is the terminal state, this episode ends here.

State \ Action	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	65	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	99	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0

1.8 Performance vs Episodes

The section above illustrated two episodes of update in Q-matrix with $\alpha = 0.9$ and $\gamma = 0.8$. To fully show the performance, we run the algorithm for 1000 episodes with parameters in Table 2. There are many ways to measure the performance. To make it comparable among all configurations. We evaluate the Q-matrix after each step by set the agent navigate from Station 9 to Station 7 (destination) using the Q-matrix just updated. The idea is that if the Q-matrix is completely updated, the agent should able to navigate with the optimal path. The total number of steps, total reward, average reward per step, and mean of average reward per step in the last n episodes are recorded. Only the relevant measurement supporting the analysis are presented in the following sections. And all the Q-matrix in this report is normalised by dividing the largest value ($Q_{norm} = Q/\max(Q) \times 100$) for comparability.

Figure 3. Reward/Steps & Steps in 1000 episodes

	Reward/Steps	Steps
count	1000.0	1000.0
mean	53.3	5.0
std	14.8	5.1
min	0.0	4.0
25%	54.4	4.0
50%	60.6	4.0
75%	61.0	4.0
max	61.0	94.0

Table 3. Statistics of Reward/Steps & Steps

Figure 3 shows the performance evaluation in 1000 episodes together with descriptive statistics in Table 3. The convergence reached around 320th episode, taking 4 steps from Station 9 to 7. The reward/steps is around 60 at convergence. The table below shows the Q-matrix by 1000 episodes and the optimal path from all stations to Station 7.

Optimal paths to Station 7
1 to 7 (steps: 4): 1→5→6→7
2 to 7 (steps: 3): 2→6→7
3 to 7 (steps: 2): 3→7
4 to 7 (steps: 2): 4→7
5 to 7 (steps: 3): 5→6→7
6 to 7 (steps: 2): 6→7
7 to 7 (steps: 1): 7→7
8 to 7 (steps: 2): 8→7
9 to 7 (steps: 4): 9→5→6→7
10 to 7 (steps: 4): 10→11→8→7
11 to 7 (steps: 3): 11→8→7

Normalised Q-matrix by 1000 episodes

State \ Action	1	2	3	4	5	6	7	8	9	10	11
1	50	63	0	0	63	0	0	0	0	0	0
2	50	63	79	0	63	79	0	0	0	0	0
3	0	63	79	79	0	0	100	0	0	0	0
4	0	0	79	79	0	0	100	0	0	0	0
5	50	63	0	0	63	79	0	0	50	50	0
6	0	63	0	0	63	79	100	79	0	0	0
7	0	0	79	79	0	79	100	79	0	0	0
8	0	0	0	0	0	79	100	79	0	0	63
9	0	0	0	0	63	0	0	0	50	50	0
10	0	0	0	0	63	0	0	0	50	50	63
11	0	0	0	0	0	0	0	79	0	50	63

Table 4. Normalised Q-matrix by 1000 episodes & optimal paths to destination

2. Advanced

2.1 Search of Learning Rate

To understand the impact of the learning rate, a wide range of values are explored: 0.01, 0.1, 0.5, 0.9, 1. A large learning rate leads to faster convergence since Q value moves faster towards the maximum value. However, too large learning rate without exploration may lead to a suboptimal policy (i.e. always greedy without exploration) [1,2]. Thus we expect a faster convergence when the learning rate is large. *[Hypothesis 1: larger learning rate leads to faster convergence]*

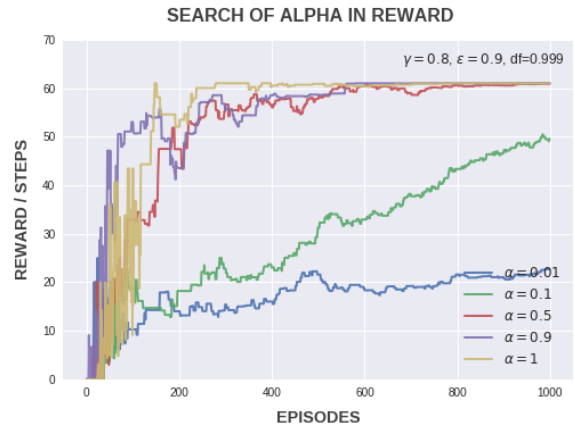

Figure 4. Average reward per step in different α

Reward/Steps

alpha	mean	std	median	max
0.01	17.3	4.3	18.5	25.0
0.10	29.4	12.7	31.2	50.5
0.50	52.1	14.8	58.4	60.9
0.90	55.2	11.3	58.6	61.0
1.00	54.5	15.3	60.7	61.0

Table 5. Statistics of reward/steps in different α

Figure 4 shows that the convergence reached by 1000 episodes when α is 0.5, 0.9 or 1.0. Table 5 also shows that the maximum reward/steps is around 60 for those 3 values. When α is 0.01 or 0.1, the convergence is not reached by 1000 episodes. In terms of the speed of convergence, the large learning rate does converge faster. Therefore, it is consistent with our hypothesis and previous studies.

2.2 Search of Discount Rate

The following discount rates are explored: 0.01, 0.1, 0.5, 0.8, 1. High discount rate (i.e. large value) means more future rewards is reflected, thus the agent is far-sighted. Therefore, we expect faster convergence can be brought by using a high discount rate. *[Hypothesis 2: larger discount rate leads to faster convergence]*

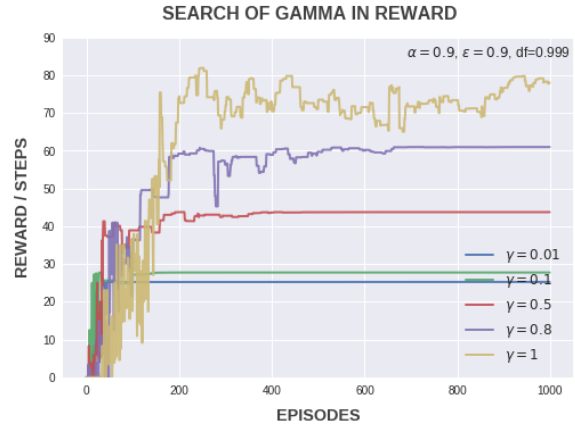

Figure 5. Average reward per step in different γ

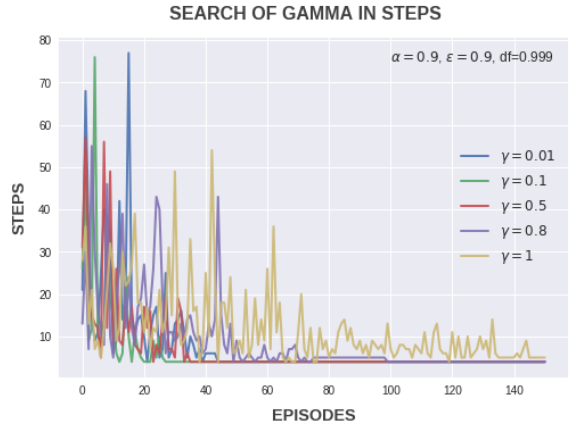

Figure 6. Total steps in different γ

Figure 5 shows the opposite to our hypothesis. Agent converges earlier with a smaller discount rate. And interestingly, the average reward per step is very unstable when γ=1. It is not even

converged by 1000 episodes. This is consistent with previous study that Q value becomes unstable when the discount rate is large [2]. When γ=1, a complete reward estimation backup gives the agent not enough incentive to find out the optimal path. As more options than actually have would give the agent the same maximum reward anyway. Figure 6 also illustrated the unstableness that there are still large steps after 60 episodes when γ=1 (yellow line).

2.3 Search of Exploration Factor

Exploration is one of the few conditions for Q-learning to learn optimal policy [2]. ε of 0.001, 0.01, 0.1, 0.5, 0.9 are explored. We expect high ε would lead to a slow convergence since the agent spend a longer time to explore. *[Hypothesis 3: larger exploration factor leads to slower convergence]*

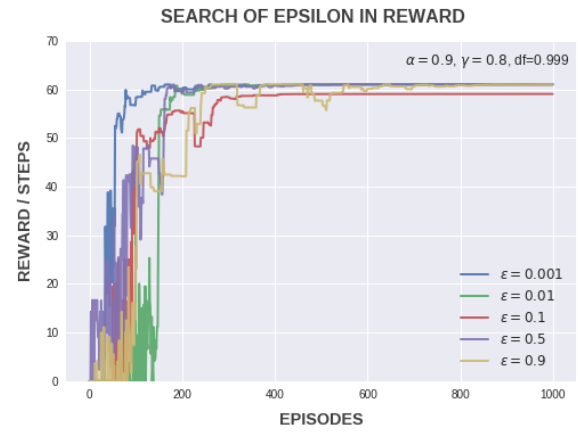

Figure 7. Average reward per step in different ε

Action

State	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	63	0	0	0	0	0	0
2	0	0	0	0	0	79	0	0	0	0	0
3	0	0	0	0	0	0	99	0	0	0	0
4	0	0	79	0	0	0	0	0	0	0	0
5	0	0	0	0	0	79	0	0	0	0	0
6	0	0	0	0	0	0	99	0	0	0	0
7	0	0	0	0	0	0	99	0	0	0	0
8	0	0	0	0	0	0	100	0	0	0	0
9	0	0	0	0	63	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	51	0	0
11	0	0	0	0	0	0	0	79	0	0	0

Table 6. Q-matrix by 1000 episodes with ε=0.001

Figure 7 shows all 5 cases are converged by 1000 episodes. And ε seems only have a marginal impact on the speed of convergence. However, only looking at performance evaluation in average reward per step is misleading in this case. Table 6 shows the normalised Q-matrix by 1000 episodes when ε is 0.001. It appears the matrix is sufficient to guide the agent navigate from Station 9 to 7 in the optimal path. But, many actions at each state are left without explored. This is a good example to show the effect when ε is too small and the necessity of exploration. In our case, Q-matrix got completely updated when ε is 0.5 or 0.9.

2.4 Search of Decay Factor

The following values of decay factor are explored: 0.5, 0.7, 0.9, 0.99, 0.999. We expect a slow convergence when the decay factor is large, which the agent explores more. This might be an oversimplified hypothesis, but we would expect the trend. *[Hypothesis 4: larger decay factor leads to slower convergence]*

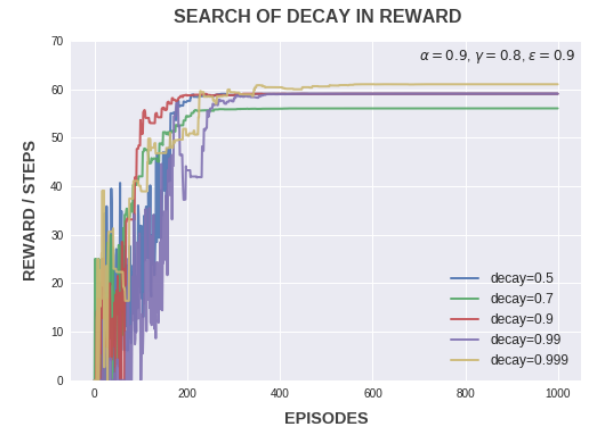

Figure 8. Average reward per step in different decay

	Steps						
	mean	std	min	25%	50%	75%	max
decay							
0.500	5.9	3.6	5.0	5.0	5.0	5.0	54.0
0.700	6.3	2.3	4.0	6.0	6.0	6.0	46.0
0.900	5.4	2.6	4.0	5.0	5.0	5.0	43.0
0.990	6.5	6.0	4.0	5.0	5.0	5.0	71.0
0.999	4.4	4.7	4.0	4.0	4.0	4.0	143.0

Table 7. Statistics of total steps in different decay

The performance evaluation in Figure 8 doesn't show any clear trend in terms of the speed of convergence with different decay factor, and little conclusion can be drawn from it. However, Table 7 reveals some insights of impact from the different decay factors. The decay factor is negatively correlated with average total steps. For example, average 4.4 steps when decay is 0.999 but 5.9 steps when decay is 0.5. Also, the 3rd quartile total steps is 4 steps only when decay is 0.999. It means slower decay (i.e. more exploration) is beneficial for Q-matrix update. When checking Q-matrix with different decay value, it is noticed that the Q-matrix is completely updated when decay value is 0.99 or 0.999. This is another good example to prove the value of exploration, especially in early episodes.

2.5 Search for an Optimal Combination

The sections above demonstrated the impact on Q-learning by varying a single parameter in each section. Based on the findings above, we can summarise the optimal value range as:

Parameter and Optimal Value Range			
Learning Rate (α)	[0.5, 0.9]	Exploration Factor (ε)	[0.5, 0.9]
Discount Rate (γ)	[0.5, 0.8]	Decay Factor (df)	[0.99, 0.999]

Table 10. Optimal value range for each parameter

To find the optimal parameters combination, a Grid Search on learning rate (0.8, 0.9), exploration factor (0.7, 0.9), and decay factor (0.99, 0.999) is performed. The discount rate is fixed at 0.8 to be comparable with other sections. The discount rate has a direct impact on the magnitude of Q value, thus varying the discount rate would make the comparison challenging.

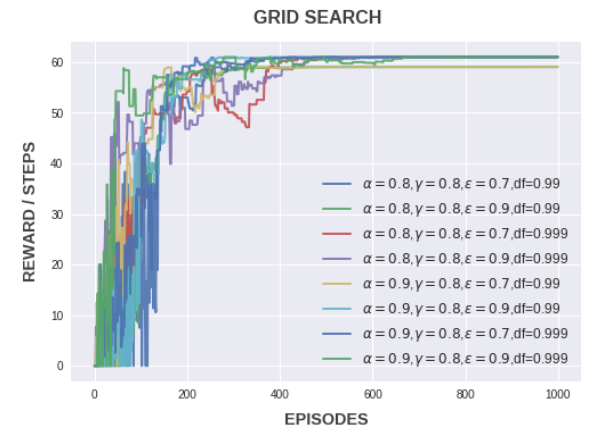

Figure 11. Average reward per step on grid search

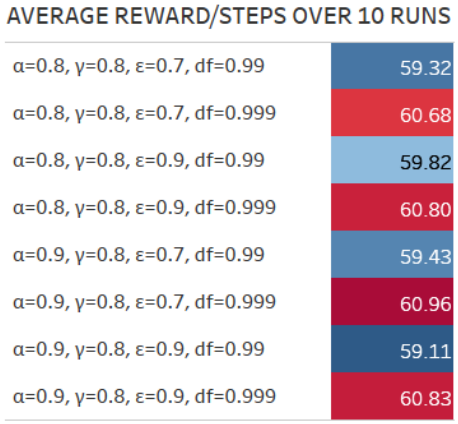

Table 11. Mean of reward/steps in the last 500 episodes

Figure 11 demonstrated the average reward per step on 8 combinations. All combinations converged by 1000 episodes and reached optimal path (i.e. reward/steps is around 60). To find out which combination perform better, we run the model for 10 times and calculated a mean value of the average reward per step in the last 500 episodes. As we can see from Figure 11 that all combinations start to converge around 500th episode. Table 11 shows the result that $\alpha = 0.9, \gamma = 0.8,\ \varepsilon = 0.7,\ df = 0.999$ achieved the highest average reward per step of 60.96. However, it is worth to caveat that this parameter does not always perform the best over 10 runs. It is ranked at top 3 out of 10 times. This reveals some randomness of the agent's behaviour which is fundamental to reinforcement learning.

2.6 Conclusions and Future Study

All parameters impact Q-learning algorithm in different ways. Learning rate has a clear impact on the speed of convergence. Not necessarily the larger the better, but too small leads slow learning. Adding decay in learning rate could be interesting to explore in the future. The impact of the discount rate is interesting. The unstable Q-value when the discount rate is large tells us to choose a smaller one. Exploration and decay factor control the same thing: exploration versus exploitation. They have the biggest impact on Q-matrix among all. And it is essential to choose the larger values to keep agent explore. We learnt the randomness of agent's move in section 2.5. This tells us to run the model multiple times for all sections may give us more stable results than we have now. And implement Q-learning in a larger task will provide us more to learn.

2.7 Double Q-Learning

The Double Q-learning algorithms is a variant of the Q-learning algorithms. It updates the Q-matrix similar to Q-learning. However, it divides the time step in two and there are two Q-matrices. The algorithm randomly updates the Q-matrices one step at a time. We can think of it as tossing a coin on each step, if the coin is coming up head, then the update is

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, maxQ_1(S_{t+1}, a)) - Q_1(S_t, A_t)]$$

Otherwise we perform the same update with Q1 and Q2 switched. Our implementation is based on the sum of the two action-value estimates. However, we can also use average or max functions. The pseudocode of Q-learning is presented in the box below. [4]

Double Q-learning (off-policy TD control) pseudocode

Set algorithm parameters: $\alpha \in (0,1], \gamma \in (0,1], \varepsilon \in (0,1), df \in (0,1]$
Initialise $Q_1(s,a)$ and $Q_1(s,a)$, arbitrarily set $Q_1(s,a) = Q_2(s,a) = 0$
Loop for each episode:
  Initialise S
  Loop for each step of episode:
    Choose A from S using ε-greedy policy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probability:
      $Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_2(S_{t+1}, maxQ_1(S_{t+1}, a)) - Q_1(S_t, A_t)]$
    else:
      $Q_2(S_t, A_t) \leftarrow Q_2(S_t, A_t) + \alpha[R_{t+1} + \gamma Q_1(S_{t+1}, maxQ_2(S_{t+1}, a)) - Q_2(S_t, A_t)]$
    $S \leftarrow S'$
    $\varepsilon' = \varepsilon \times df$
  until S is terminal

The main reason for us to choose Double Q-learning as the alternative algorithm is due to the maximisation bias in Q-learning. In Q-learning, we use the ε-greedy policy which includes a maximization step over estimated action values. Hence our result can lead to a significant positive

bias. Hasselt (2010) proved that Double Q-learning: $E[Q_2(S_{t+1}, A_t)] \leq max(Q_1(S_{t+1}, A_t))$. Hence, over iterations, Q_1 is not updated with a maximum value.

For a fair comparison, we also use the ε-greedy policy for Double Q-learning and we also set $\alpha = 0.9, \gamma = 0.8, \varepsilon = 0.7, df = 0.999$ which are the best parameters from our grid search in Q-learning. We also run the algorithm for 1000 episodes and have a similar setup as to the Q-learning implementation. The below figures are the performance/steps vs. episodes results for comparison. *[Hypothesis Statement 5: Double Q-learning will outperform Q-learning, given the same optimised parameters]*

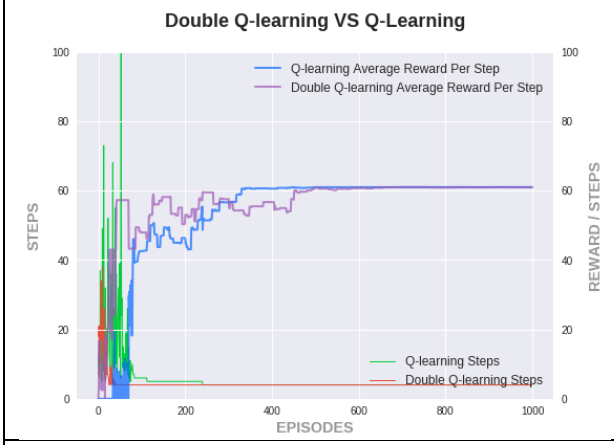

Figure 11. Q-learning algorithm in 1000 episodes

	Average Reward Per Step	Steps
count	1000.0	1000.0
mean	54.0	4.7
std	10.9	2.3
min	0.0	4.0
25%	50.6	4.0
50%	60.0	4.0
75%	60.7	5.0
max	61.0	42.0

Table 13. Descriptive statistics of 1000 episodes

Figure 11 shows the performance comparison between Q-learning and Double Q-learning for 1000 episodes. From only around 70th episode, the step number of Double Q-learning starts to converge, and agent only takes 4 steps (the optimal route) to reach the destination, whereas Q-learning took 240 episodes to converge. However, Q matrix converged faster in terms of rewards per step. For Q learning, the reward per step converged at around 300th episode and Double Q-learning converged around at 420th episodes. We believe this is due to Double Q-learning has two Q matrices and update one of them per episode randomly, therefore it would take longer for the Q matric converge. Nevertheless, our results agree with our hypothesis. We can compare the results more precisely by analysing the average reward per steps (see table 13 a table 3). We can see that Double Q-learning has a slightly higher average rewards (with mean 54) and a higher average step of mean 4.7 (0.3 less than Q-learning). Also, the standard deviation of average reward is 10.9 and average step is 2.3 respectively, which are significantly less than Q-learning (14.8 and 5.1 respectively, table 3). This demonstrates that Double Q-learning is a more stable algorithms than Q-learning. Furthermore, from figure 11, we can see that Double Q-learning converged earlier than Q-learning in term of average rewords and number of steps. Moreover, we can see than Q-learning takes a much greater number of steps in the earlier episodes compared with Double Q-learning.

However, we cannot conclude that Double Q-learning is a more efficiency algorithm than Q-learning base on our experiment. We ran our models' multiple times, and in some cases Q-learning outperformed Double Q-learning. This means that our problem maybe too small and less robust to evaluate the two algorithms. To comprehend the actual performance of Double Q-learning, we could add some noise to the environment or increase the number of states. However, when facing some stochastic environments, we should choose Double Q-learning instead as it is generally a more robust and stable algorithm. [4]

3. Essay Questions

3.1 Q-learning and error correction models in psychology

Reinforcement learning is an area of study about how software agent learns a policy through the interaction in an environment. The development of it is inspired by psychological learning theories. At the same time, the development of reinforcement learning provides psychology with a systematic way to understand experimental data and suggest new directions of study. There is a close link between the two subjects. [4]

3.1.1 Classical Conditioning

Q-learning conventionally is referred to as an off-policy TD control algorithm. To understand control algorithms, we need to first look at the simpler prediction algorithms which can be approximately connected to classical or Pavlovian conditioning in psychology despite many other complications. Classical conditioning was established by Ivan Pavlov through his famous experiment on dogs. He found that dog learnt to use conditional stimulus (CS) to predict unconditional stimulus (US, reinforcer) and thus produce conditional responses (CR) after receiving CS. [4]

3.1.2 Blocking and Higher Order-Conditioning

A property called blocking was observed in classical conditioning. Blocking occurs when an animal failed to produce CR when a potential CS is presented together with another CS that has been previously used to form CS and CR condition. Rescorla and Wagner in 1972 offered an influential error correction model to explain blocking which will be discussed below. [4]

Higher order-conditioning was also observed by Ivan Pavlov that a new CS can be established by using the current CS as US. And previous CS now becomes a secondary reinforcer. The idea is similar to bootstrapping in Q-learning that update is not based on actual reward but estimated one step ahead value function. [4]

3.1.3 Error Correction Models

Rescorla-Wagner model states that an animal only learns when an event violate its expectation. When a surprise comes, the model adjusts its associative strength of each component stimulus of a compound CS to reflect the violation. No adjustment is made when there is no surprise. This is highly similar to how Q-learning agent learn. It updates the current Q value when the new action-value function differs from the current one (i.e. when there is TD error). If the new Q value is the same as the old one, no learning/update is made. In the context of exploration and exploitation, if exploration has not done enough, the agent loss chance to see all “surprises” in an environment to develop a complete policy. [4]

Rescorla-Wagner model is an error correction supervised learning rule which is similar to the Least Mean Square and Widrow-Hoff learning rule. They share the same idea and is fundamental to Q-learning. However, there are two main differences between Rescorla-Wagner’s error correction model and TD error in Q-learning. Firstly, Q-learning is a real-time model means value function is updated after each time step, whereas Rescorla-Wagner model update error after an entire trial. Secondly, higher-order conditioning comes naturally with bootstrapping in Q-learning. But higher-order conditioning is not a mechanism in Rescorla-Wagner model. [4]

3.1.4 Instrumental Conditioning

We discussed error correction and bootstrapping in the sections above. And there is another one main missing piece: actions from the agent. Classical conditioning is responsive, whereas

instrumental conditioning is active. It states that the learning is contingent on what the animal does. In Q-learning as well as reinforcement learning control algorithms, the agent learns the cause and effect by taking different actions and updating function value according to errors. [4]

3.2 Error Correction Models and Reinforcement Learning

The essential feature of reinforcement learning can be linked to the Law of Effect by Edward Thorndike. The law is known as trial and error. In computational terms, it relates to search and memory. Search means the agent need to explore. Memory is stored in the form of an agent's policy, value function or environment through iteratively adjusting the prediction errors. [4]

As noted above, any type of reinforcement learning architecture involves taking actions and memorising errors from the actions. The way to memorise the error is through error correction. In Q-learning, $R_{t+1} + \gamma maxQ(S_{t+1}, a) - Q(S_t, A_t)$ is the error term. To generalise this concept, we look into Rescorla-Wagner model again. To understand how the error correction model can be transformed to TD model and reinforcement learning architecture in general.

In Rescorla-Wagner model, if we have a compound CS A and X. The animal may already experience stimulus A, and X might be new. Let V denote the associative strengths of stimulus. Suppose a trial involves a compound CS AX followed by US (denote as Y). R is the level of associative strength that US Y can support. Then the change in associate strengths are expressed as: $\Delta V_A = \alpha_A \beta_Y (R_Y - V_{AX})$ & $V_X = \alpha_X \beta_Y (R_Y - V_{AX})$. [4]

α_A and β_Y are step-size parameters depended on CS and US. The formulas state that V_A and V_X will keep increase until the compound V_{AX} reach the same level of R_Y. To gradually transform to TD model, we take the conditioning process above as the predicting the magnitude of US. And states need to be introduced to convert Rescorla-Wagner model to a real-time model. Assume state s is described as a vector of features $X(S) = (x_1(s), x_2(s), \dots, x_d(s))$. If the d-dimensional vector of associative strengths is w, the aggregated associative strength is: $v(s, w) = w^T X$. To update w on trail t: $w_{t+1} = w_t + \alpha \delta_t X(S_t)$. α is step-size parameter and δ is prediction error: $\delta_t = R_t - v(S_t, w_t)$. R_t is the prediction of magnitude of US on trial t. [4]

To fully arrive at TD model, we now change t to denote time step instead of a complete trial as above. And a short-term memory vector, eligibility trace z_t is also introduced to only update eligible states or actions. Now, the associative strength update becomes: $w_{t+1} = w_t + \alpha \delta_t z_t$. The full state vector $X(S)$ in Rescorla-Wagner model is replaced with eligibility trace vector. And prediction error now is: $\delta_t = R_{t+1} + \gamma v(S_{t+1}, w_t) - v(S_t, w_t)$, where γ is the discount factor, R_{t+1} is the immediate reward after take an action, and $v(S_{t+1}, w_t)$ and $v(S_t, w_t)$ are aggregated associative strength at $t+1$ and t which can be seen as value function. And this is the error correction architecture used by Q-learning: $R_{t+1} + \gamma maxQ(S_{t+1}, a) - Q(S_t, A_t)$. With different value of decay parameter λ in the eligibility trace vector $z_{t+1} = \gamma \lambda z_t + X(S_t)$, we have TD(λ) model which can unify the whole spectrum from Monte Carlo (λ=1) to one-step TD model (λ=0) (e.g. Q-learning) [4]. Therefore, we can see that the error correction model is one of the fundamental parts of TD(λ) algorithm, so as reinforcement learning architecture.

References:

[1] Beleznay, F., Grobler, T. and Szepesvari, C., 1999. Comparing value-function estimation algorithms in undiscounted problems. *Technical Report* TR-99-02, MindMaker Ltd.
[2] Even-Dar, E. and Mansour, Y., 2003. Learning rates for Q-learning. *Journal of Machine Learning Research*, 5(Dec), pp.1-25.
[3] Hasselt, H.V., 2010. Double Q-learning. *In Advances in Neural Information Processing Systems* (pp. 2613-2621).
[4] Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. MIT press.
[5] Watkins, C.J.C.H., 1989. Learning from delayed rewards (Doctoral dissertation, King's College, Cambridge).