# EPFL ANN Project Report

Yanbo Xu(349254) Xiang Bai(337255)

## 1. Q-Learning

### 1.1. Question 1

As shown in Fig. 1, while $\epsilon$ increases from 0.0 to 0.9, the average rewards are decreasing. We can only conclude the agent learned to play with the Opt(0.5). There would need some Validation and Test to get a better evaluation.

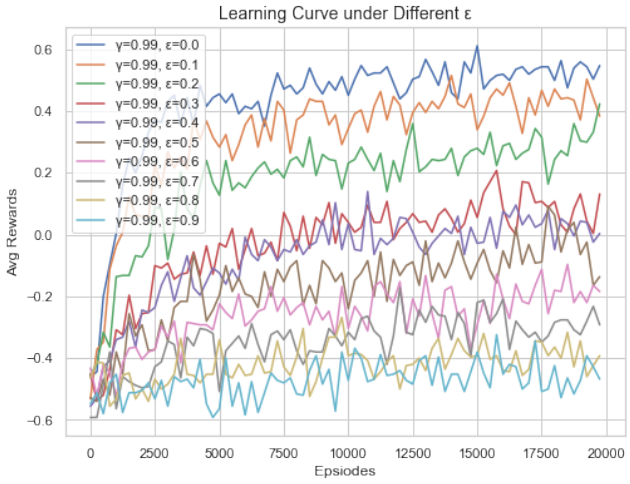


Figure 1. Average rewards with different $\epsilon$

### 1.2. Question 2

Decreasing $\epsilon$ with a reasonable $n^*$ (10000, 20000) helps the training compared with the setting with large $\epsilon$(0.7, 0.9), which could be observed from Fig. 2.

For $n^* = 10000$ or 20000, because their $\epsilon$ will decay to 0.1, the final average rewards will all converge to 0.4 roughly. For $n^* = 30000 or 40000$, the final decayed will be $\epsilon$ will decrease to 2.75 and 0.4 respectively, and this will lead to a lower average reward. It works the same as a fixed $\epsilon(0.1)$ when $n^* = 1$. A fixed $\epsilon(0.1)$ converges fast than the decay methods.

The effect of $n^*$ is a balance parameter for exploration and exploitation, at the fist beginning, agent should emphasizes on exploration, then tends to make use of exploitation.

### 1.3. Question 3

Result is shown in Fig. 3.The similarity is, a fixed $\epsilon(0.1)$ converges really fast. The difference is even some bigger $n^*$ have a poor performance on average reward curve, they still get a qualified test/validation results under both Random and Optimal players. The training results do not always accurately reflect the generalization.

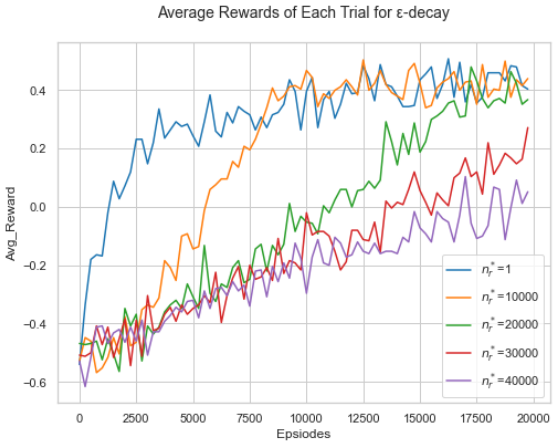


Figure 2. Average reward with $n^*$

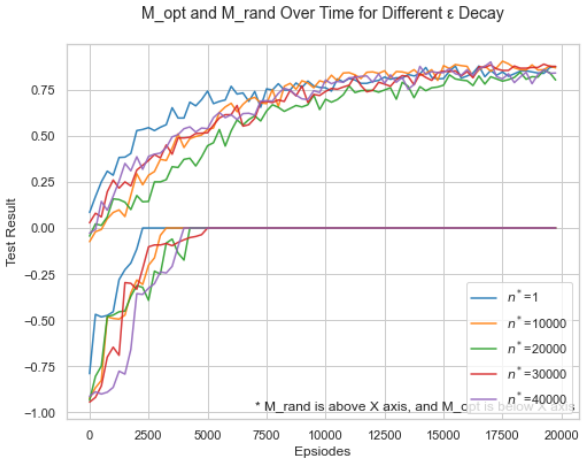


Figure 3. $M_{opt}$ and $M_{rand}$ with different $n^*$

### 1.4. Question 4

After choosing 20000 as the best $n^*$, we observe an approximate reverse between $M_{opt}$ and $M_{rand}$ (see Fig. 4). For agents trained with a player with small $\epsilon$ (0, 0.1), we will get a high $M_{opt}$ and a low $M_{rand}$. If trained with high $\epsilon$ (0.7, 0.9), the $M_{opt}$ is lower.

The reason is that when we implement the Q-learning algorithm with a two-player-game, the game itself is not the environment in our theory; instead, the environment is the game plus the counterpart player. So every time we change player's $\epsilon$, the training environment is changing accordingly.

### 1.5. Question 5

The highest $M_{opt}$ and $M_{rand}$ are 0.0($\epsilon_{opt} \in \{0, 0.1, 0.3, 0.5\}$) and 0.92($\epsilon_{opt} = 0.9$), respectively.

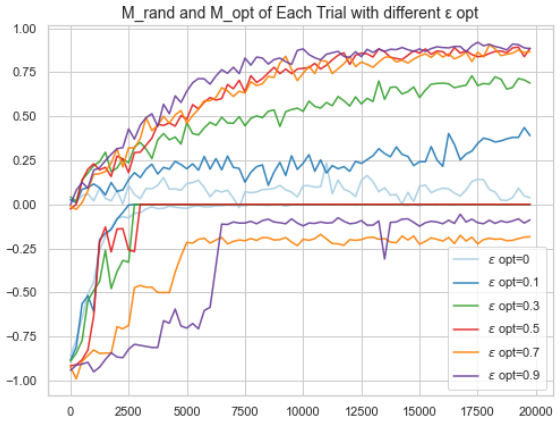


Figure 4. $M_{opt}$ and $M_{rand}$ with different $\epsilon_{opt}$

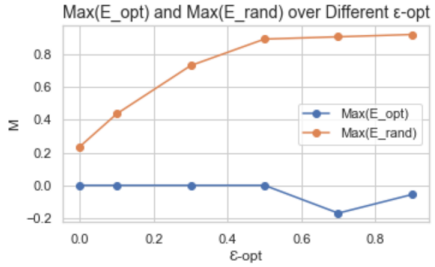


Figure 5. Relation between $M_{opt}^{best}$ and $M_{rand}^{best}$ with different $\epsilon_{opt}$

## 1.6. Question 6

$Q1(s, a)$ and $Q2(s, a)$ have different values. This is because the environment is not the TicTacToe game itself, instead it's the game plus the other player's round (See Fig. 6). So it is obvious the state Stochasticity $P^a_{s \to s'}$ is changing with different $Opt(\epsilon)$.

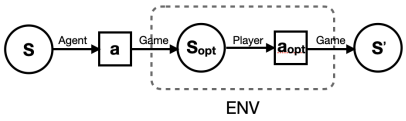


Figure 6. The True Environment

## 1.7. Question 7

If $\epsilon$ is too small (such as 0), the agent does not learn to play TicTacToe. This is because the agent will be trained under the exploitation mode. With an absolute 'greedy' method, the agent will get stuck with a suboptimal strategy due to lack of exploration. When $\epsilon$ is too big (0.9), performance also suffers due to not learning from itself (it performs random actions). The effect of $\epsilon$ is to determine the degree of exploration and exploitation. From our trials, $\epsilon = 0.3$ is the best value.

## 1.8. Question 8

As in Fig. 8, decreasing $\epsilon$ will help training compared to having a fixed and large $\epsilon$ (0.7, 0.9). In different training stages, we focus on exploration and exploitation differently and $n^*$ decides the ratio. Giving an optimal $n^*$ (such as 30000), we can explore more at the beginning, and exploit more gradually to the end.

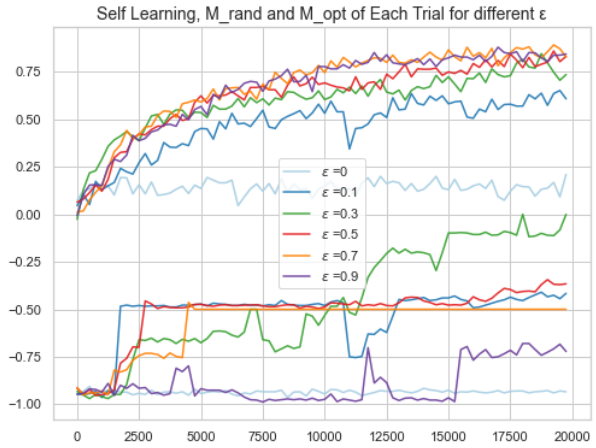


Figure 7. Self-Practice $M_{opt}$ and $M_{rand}$ with different $\epsilon$

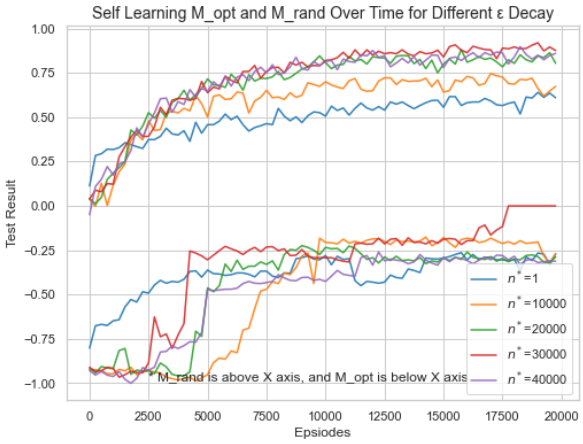


Figure 8. Self-Practice $M_{opt}$ and $M_{rand}$ with different $n^*$

## 1.9. Question 9

The highest values of $M_{opt}$ and $M_{rand}$ is 0.0 ($n^* = 30000$) and 0.92 ($n^* = 30000$), respectively. (See the relation from Fig. 9.)

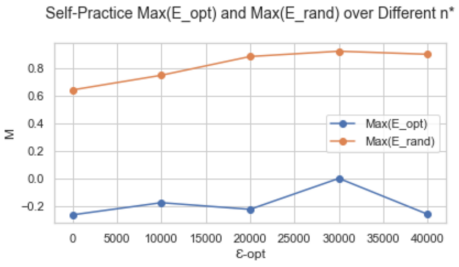


Figure 9. Relation between $M_{opt}^{Max}$ and $M_{rand}^{Max}$ with different $n^*$

## 1.10. Question 10

For Board1 in Fig. 9, by placing the piece on the cross position, we would have a better chance to win, which is corresponding with its Hotmap.

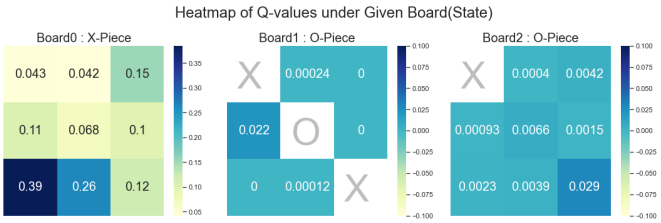


Figure 10. Board Heat map Examples

## 2. Deep Q-Learning

### 2.1. Question 11

As shown in Fig. 11, for most $\epsilon$, the average reward increases and the loss decreases. The average reward approaching 0 or even becoming positive shows that the agent learns to play the game. From the results, $\epsilon = 0$ gives the highest average reward and the lowest loss.

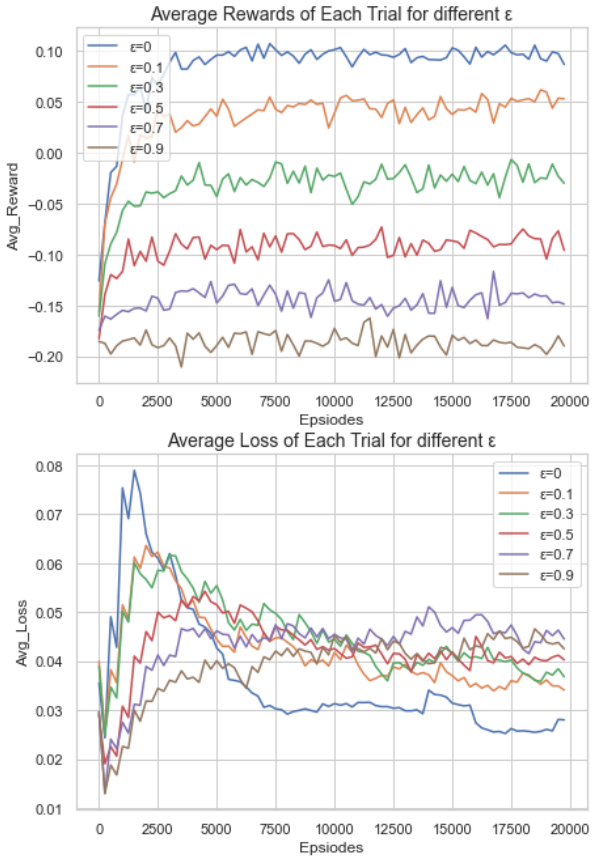


Figure 11. Average reward and loss with fixed different $\epsilon$

### 2.2. Question 12

Fig. 12 shows the result. From the graph, we observe that the loss will become more fluctuating and unstable compared with experiments with a larger replace buffer. Additionally, although the reward increases, the convergence is clearly slower.

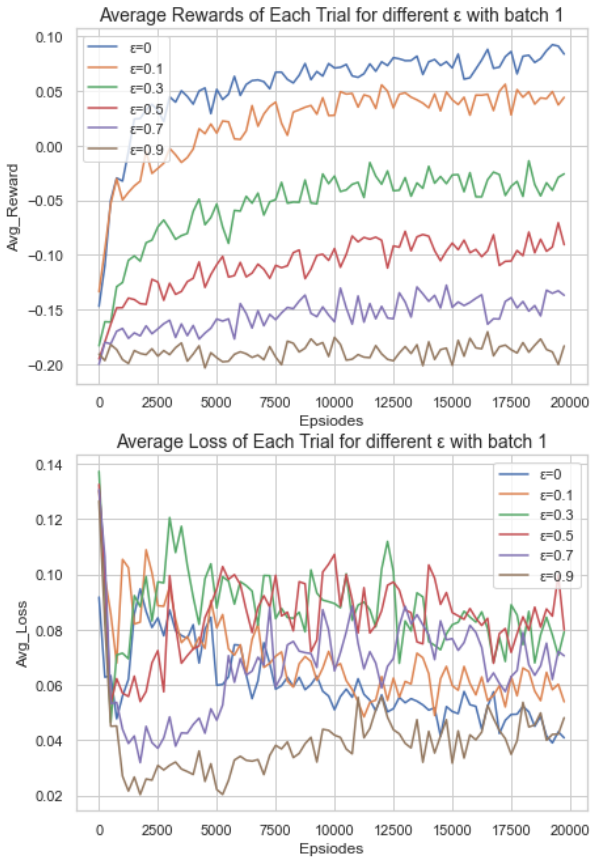


Figure 12. Average reward and Average loss using replace buffer of size 1 and different $\epsilon$

### 2.3. Question 13

Fig. 13 shows the $M_{opt}$ and $M_{rand}$ with different decay factor $n^*$, uniformly sampled from [1, 40000]. A larger $n^*$ makes the decay of $\epsilon$ slower, while a small $n^*$ gives faster decay. We chose $n^* = 10000$ as its result is both good and stable. Although not shown here, having a decayed $\epsilon$ will give a better average reward and loss curve compared with the experiments with large fixed $\epsilon$ in Sec. 2.1.

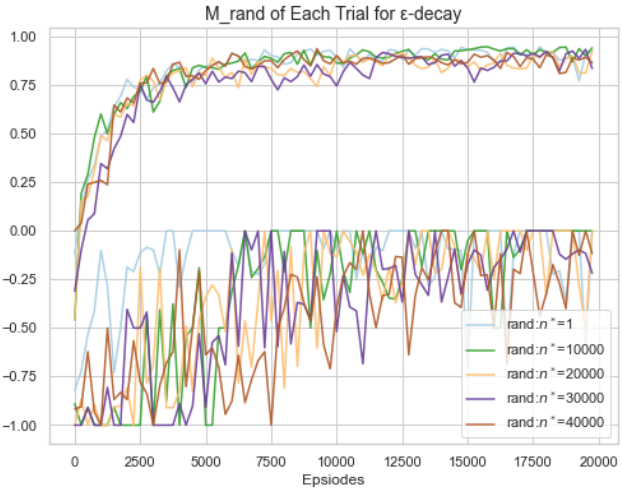


Figure 13. $M_{opt}$ and $M_{rand}$ with different decay factor $n^*$

### 2.4. Question 14

With the chosen decay factor $n^* = 10000$, we sample $\epsilon_{opt}$ uniformly from [0, 1). From the result in Fig. 14, it could be observed that using $\epsilon_{opt} = 0$ will harm the $M_{rand}$ score, since the agent had only seen optimal actions and thus fails when encountering unseen actions given by the random player. On the other hand, using a large $\epsilon_{opt}$ such as 0.9 will harm the $M_{opt}$ score, since the expert chooses random actions too often. An good expert should have a degree of randomness, yet not too random. Values of $\epsilon_{opt}$ such as 0.2, 0.3 show good results in both $M_{opt}$ and $M_{rand}$.

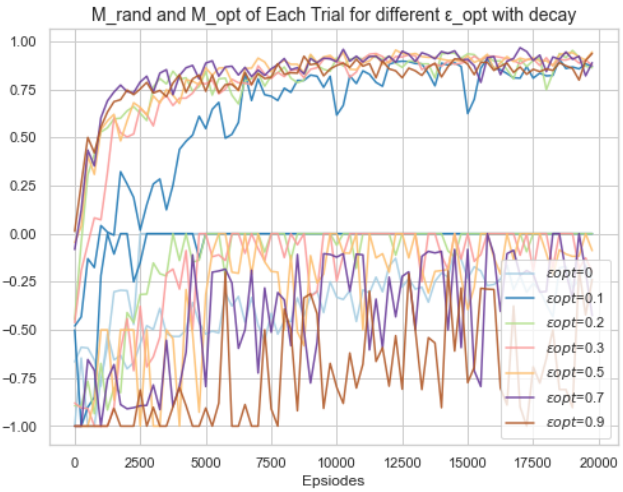


Figure 14. $M_{opt}$ and $M_{rand}$ with different $\epsilon_{opt}$

### 2.5. Question 15

The best $M_{opt}$ and $M_{rand}$ we achieved after 20000 games is 0.0 and 0.966, respectively.

### 2.6. Question 16

The result in Fig. 15 shows that with proper $\epsilon$ (such as 0.1), the agent could learn to play the game well. If $\epsilon = 0$, the agent will only explore limited actions, resulting in a very bad result against both random and optimal players. If $\epsilon$ is very large such as 0.9, although exploring more possible actions, the policy itself has trouble learning, since most actions are not chosen from the current policy network. A good $\epsilon$ should guarantee exploration while giving the policy network enough chance to learn from itself.

### 2.7. Question 17

As shown in Fig. 16, having a decay factory will improve the performance compared the experiments with a large fixed $\epsilon$ (such as 0.8, 0.9) in Fig. 15. To be specific, results of experiments with decay factor will be more stable. $n^*$ controls the proportion of exploration and exploitation. If decay too slow ($n^* = 40000$ for example), there will be fluctuation at the end.

### 2.8. Question 18

The best $M_{opt}$ and $M_{rand}$ we achieved after 20,000 games with the decay factor $n^*$ is 0.0 and 0.932, respectively. With a fixed $\epsilon$, we could achieve 0.0 and 0.95.

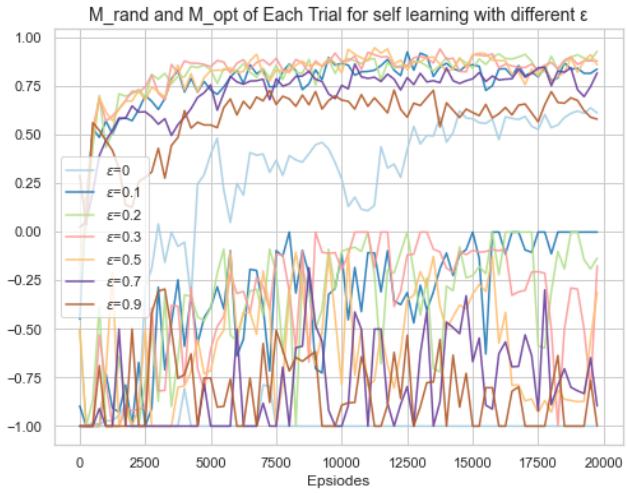


Figure 15. Self learning, $M_{opt}$ and $M_{rand}$ with different $\epsilon$

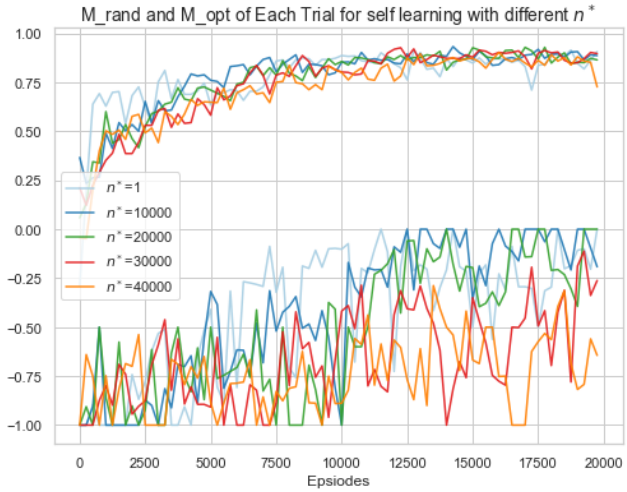


Figure 16. Self learning, $M_{opt}$ and $M_{rand}$ with decay

### 2.9. Question 19

Fig. 17 visualizes the result of self training. The agent does lean to play the game. In the middle graph, when the board is occupied, the Q-value is large negative, which is consistent with our design. And the actions with the large Q-value are indeed good choice . However, although the board is symmetrical, the Q-values are not. This happens in self-training that the agent tends to favor some positions since it's learning from itself. With an optimal player, the Q-values are actually more symmetrical (result not shown here).

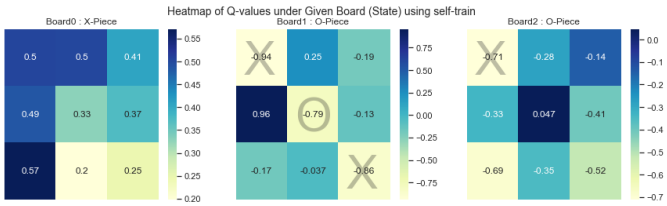


Figure 17. Q-value Heatmap Examples

## 3. Comparing Q-Learning with Deep Q-Learning

### 3.1. Question 20

See Tab. 1.

Table 1. Comparing Q-learning and DQN

| | | Q-learning | DQN |
|---|---|---|---|
| Exp | $M_{opt}$ | 0.0 | 0.0 |
| | $M_{rand}$ | 0.9 | 0.966 |
| | $T_{train}$ | 6500 | 3250 |
| Self | $M_{opt}$ | 0.0 | 0.0 |
| | $M_{rand}$ | 0.92 | 0.95 |
| | $T_{train}$ | 12250 | 2750 |

### 3.2. Question 21

From Tab. 1, We found that the DQN is slightly better than Tabular Q-learning. The most obvious difference is the convergence speed. As seen in the result, DQN converges much faster than Q-learning.

Below is the process and update rule for Q-learning.

$$\begin{aligned} Q(s,a) &\leftarrow Q(s,a) + \alpha(R + \gamma max_{a'} Q(s',a') - Q(s,a)) \\ &= (1-\alpha)Q(s,a) + \alpha[R + \gamma max_{a'} Q(s',a')] \end{aligned} \tag{1}$$

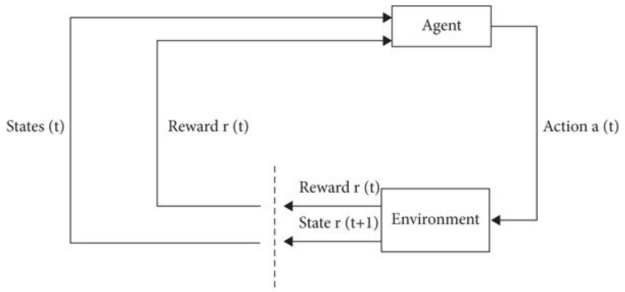


Figure 18. Q-learning Flow Chart

Comparing with Q-Learning, DQN introduce two method, *replay buffer* and *target network*.

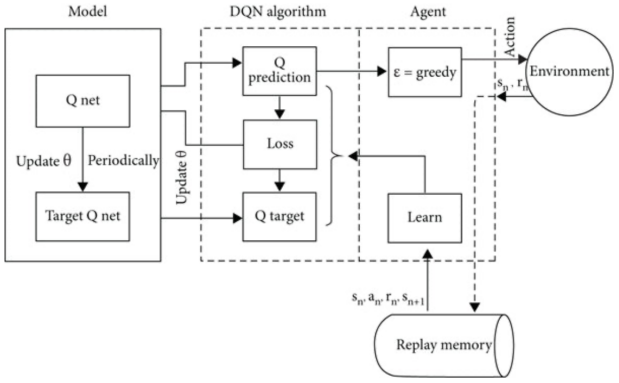


Figure 19. DQN Flow Chart

Q-learning only use the played samples once then throw it away, which makes the training less efficient. Additionally, most of the time there is a strong correlation between the current state and the previous one, which is undesirable for deep networks, which need the i.i.d. distributed data.

Randomly sampling a mini-batch of (s,a,r,s') evenly from the replay buffer is used to avoid the correlation of the same trajectory data and to make convergence faster. But the replay buffer only support off-policy algorithm.

Theoretically, the target network $Q_{\theta'}$ is used to solve the problem of unstable targets, which is exactly the same as the Q-net, except that its parameters $\theta'$ are not updated so frequently, and are usually copied from $\theta'$ only once after a certain period of time, to ensure the stability of the targets. We observe a more stable training average reward for DQN, but not on the test process. The instability on the test especially with the optimal player may be because the sparsity of our play buffer in which most (s,a,r,s')-s are with a 0 return.