Artificial Neural Networks (CS-456)
Mini-project 1: Tic Tac Toe

EPFL

Mickaël Achkar: 322190, Yehya El Hassan: 325932

June 6, 2022

1 Q-Learning

1.1 Learning from experts

Question 1

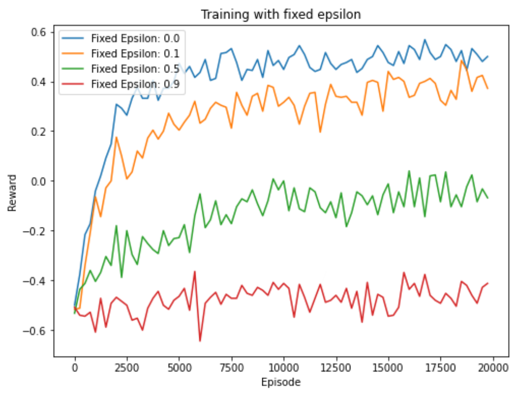

Figure 1: Plot of RL Agent's reward over the episodes for fixed ϵ trained against `Opt(0.5)`, $\epsilon_{\text{fixed}} \in \{0, 0.1, 0.5, 0.9\}$, $\alpha = 0.05, \gamma = 0.99$.

From the plot in Fig. 1, the average reward increases over the number of games. For $\epsilon_{\text{fixed}} = 0.0$ and $\epsilon_{\text{fixed}} = 0.1$, the agent surpasses the average reward level of 0, which means that it manages to accumulate more wins than losses. With that, we can validate that the RL agent learns how to play the Tic Tac Toe game.

1.1.1 Decreasing exploration

Question 2

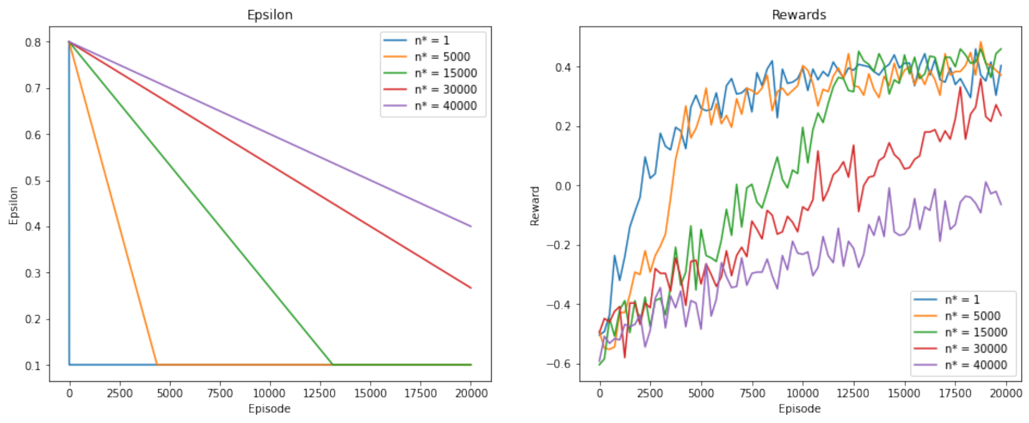

Figure 2: Plot of $\epsilon(n)$ and RL Agent's reward over the episodes for a decaying $\epsilon(n)$ trained against `Opt(0.5)`, $\epsilon \in [0.1, 0.8], n^* \in \{1, 5000, 15000, 30000, 40000\}$, $\alpha = 0.05, \gamma = 0.99$.

Decreasing ϵ through out the training procedure helps in improving the evolution of the average reward over the number of games. As it can be seen in Fig. 2, starting with a high exploration probability and decreasing it over the number of games allows the average reward to surpass that of a fixed exploration probability ($n^* = 1$). The reward curve for $n^* = 1$ corresponds to $\epsilon_{\text{fixed}} = 0.1$ in Fig. 1, because:

$$\epsilon(n, n^* = 1) = \max\{\epsilon_{\min}, \epsilon_{\max}(1 - \frac{n}{1})\} = \epsilon_{\min} = 0.1, \forall n \in \mathbb{N}^+ \tag{1}$$

In more details, the maximum average reward with $\epsilon_{\text{fixed}} = 0.1$ is below 0.4, while by training with a monotonically decaying ϵ up to the same constant ϵ_{fixed}, the agent is able to surpass the 0.4 mark (for $n^* \in \{5000, 15000\}$) within the 20000 games frame.
The effect of n^*, the number of exploratory games, is to decay ϵ up to a certain constant value $\epsilon_{\min}$ as the number of games increases. The larger n^* is, the longer the decay from a large ϵ to a minimal $\epsilon_{\min}$ value, and the longer the initial exploration phase, since a higher ϵ translates to a higher exploration probability.
In essence, the higher the initial exploration, the larger the coverage of the state-action pairs, and thus the higher the exploitation performance later on (we expect this behavior at $N > 20000$ for $n^* \in \{30000, 40000\}$).

Question 3

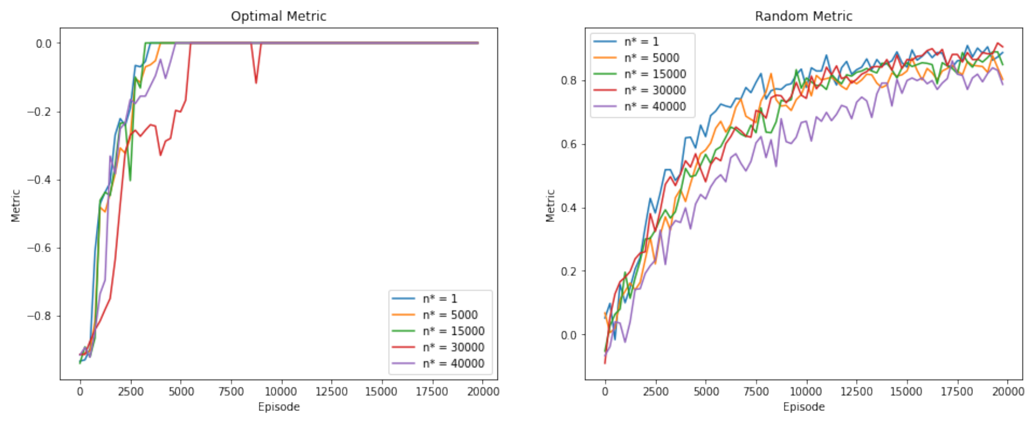

Figure 3: Plot of RL Agent's M_{opt} and M_{rand} over the episodes for a decaying $\epsilon(n)$, trained against `Opt(0.5)`, $\epsilon \in [0.1, 0.8], n^* \in \{1, 5000, 15000, 30000, 40000\}$, $\alpha = 0.05, \gamma = 0.99$.

In Fig. 3, it can be seen that all agents manage to reach the maximum level of expertise with respect to the optimal metric, with different number of games ($n^* = 15000$ being the fastest). For the random metric, the performance is not as good but it is similar across agents, with $n^* = 30000$ ending up with the best performance.
If we compare Fig. 3 with Fig. 2, the noted similarity is that $n^* = 15000$ is able to surpass $n^* = 1$ in rewards is also reflected on the random metric performance. While, the difference can be spotted by noting that the RL agents learn to play Tic Tac Toe with similar levels of expertise, as shown in the optimal metric, even though they had accumulated different rewards and utilized different exploration level decays.

1.1.2 Good experts and bad experts

Question 4

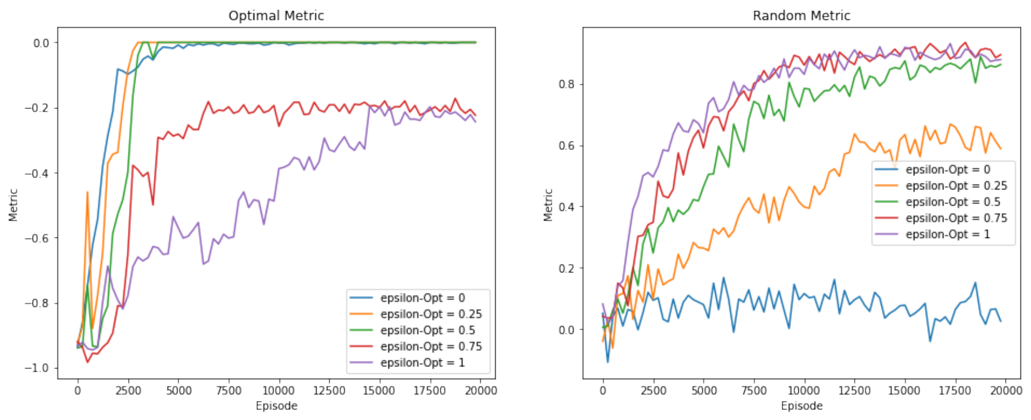

Figure 4: Plot of RL Agent's M_{opt} and M_{rand} over the episodes for a decaying $\epsilon(n)$ trained against `Opt(`ϵ_{opt}`)`, $\epsilon_{\text{opt}} \in \{0, 0.25, 0.5, 0.75, 1\}$, $\epsilon \in [0.1, 0.8], n^* = 15000$, $\alpha = 0.05, \gamma = 0.99$.

For M_{opt}, it can be observed that all agents except the ones trained with `Opt(0.75)` and `Opt(1)` manage to reach the maximum level of expertise with respect to the optimal metric. This can be explained by the fact that the ones trained with `Opt(0.75)` and `Opt(1)` did not observe many optimal moves and hence cannot learn to counteract them.
Now, for M_{rand}, the result is striking: the RL agent trained against the optimal `Opt(0)` is the worst performing, with a random metric indicating that the game is almost always a draw with a random agent `Opt(1)`. This can be explained by the fact that this RL agent has only seen optimal moves, and did not explore unexpected random moves by other types of adversaries. Hence, the learning of this RL agent is overfit to counteract optimal agent moves (RL agent does not know how to win). Moreover, the other RL agents achieve better results on the random metric with a decreasing level of opponent optimality, with $\epsilon_{\text{opt}} = 0.5$ achieving the best trade-off between speed of learning and regularization of learning on both metrics. With that, a trend can be drawn: the higher the optimality level of the teacher, the lower the ability of the trained agent to win a game.

Question 5

The highest values of the optimal and random metrics that we can achieve after playing 20'000 games are: $M_{\text{opt}} = 0.0$ (for $\epsilon_{\text{opt}} \in \{0, 0.25, 0.5\}$) and $M_{\text{rand}} = 0.894$ (for $\epsilon_{\text{opt}} = 0.75$), which approach the best results ($M_{\text{opt}} = 0$ and $M_{\text{rand}} = 1$).

Question 6

No, Agent 1 and Agent 2 will not have the same Q-values, because the two agents are not trained to value certain actions in the same fashion. In fact, training an agent with an optimal player `Opt(0)` allows at best to learn to *draw* with the opponent, which yields rewards capped at 0. In contrast, training an agent against a random player `Opt(1)`, which has the ability to lose, provides rewards capped at 1. In that sense, that agent will be able to learn to *win* the game, and $Q_2(s, a)$ will be higher than $Q_1(s, a)$.

1.2 Learning by self-practice

Question 7

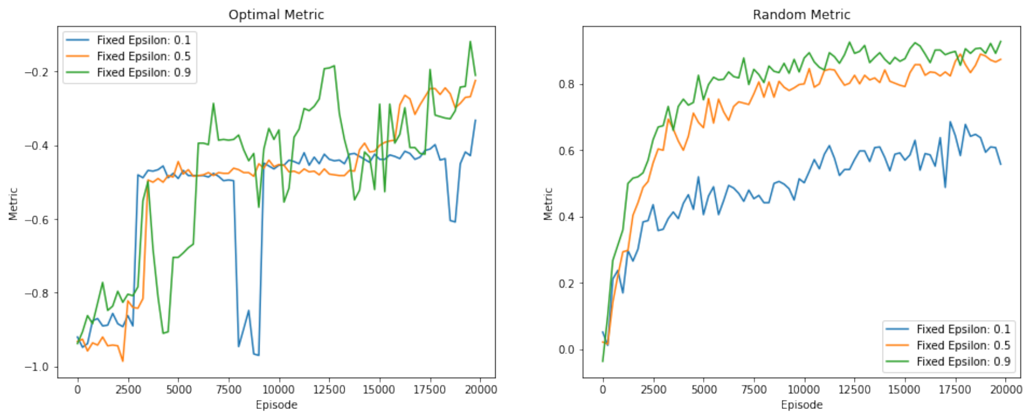

Figure 5: Plot of RLAgent's M_{opt} and M_{rand} over the episodes for fixed ϵ trained against itself, $\epsilon_{\text{fixed}} \in \{0.1, 0.5, 0.9\}$, $\alpha = 0.05, \gamma = 0.99$.

When training against itself, testing the RL agent's performance shows an increase over the number of games in both the M_{opt} and M_{rand} metrics. Hence, the agent learns to play Tic Tac Toe by self-practice.
In addition to that, as it can be seen from Fig. 5, a higher ϵ results in higher M_{opt} and M_{rand} scores. This can be explained by the fact that as ϵ increases, the RL agents explore a larger scope of state-action pairs which will allow it better find the optimal strategy without learning it from a teacher.

Question 8

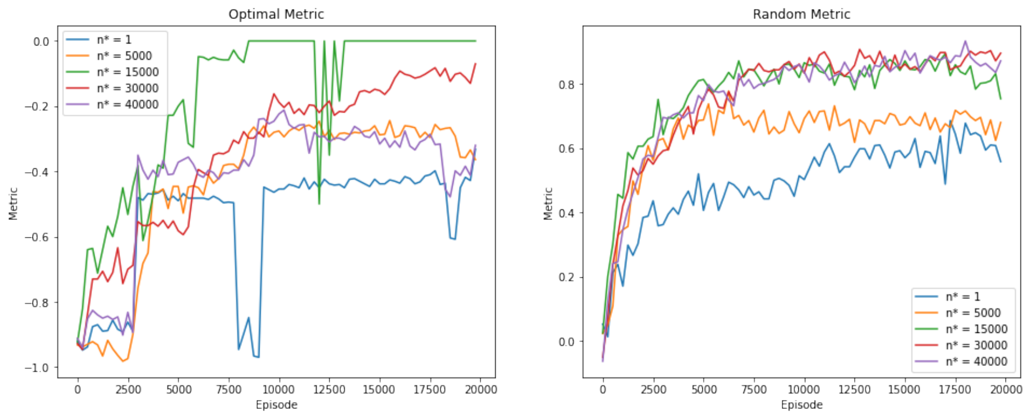

Figure 6: Plot of RL Agent's M_{opt} and M_{rand} over the episodes for a decaying $\epsilon(n)$ trained against itself, $\epsilon \in [0.1, 0.8], n^* \in \{1, 5000, 15000, 30000, 40000\}$, $\alpha = 0.05, \gamma = 0.99$.

The reward curve for $n^* = 1$ in Fig. 6 corresponds to a fixed $\epsilon = 0.1$, because of Eq. 1. Considering that, $n^* = 1$ has a worse training out of all other values of $n^* \in \{5000, 15000, 30000, 40000\}$. Thus, decreasing ϵ appears to be helping the training compared to having a fixed one.
The decay allows agents with a high number of exploratory games n^* to quickly catch up in learning after many games, as exploration allows the agent to try and evaluate different policies before exploiting the best estimated one. In fact, the RL agent self-trained with $n^* = 15000$ achieve the best trade-off after 20'000 games on both M_{opt} and M_{rand}, and agents with a high n^* are the best against a random agent.

Question 9

The highest values of the optimal and random metrics that we can achieve after playing 20'000 games are: $M_{\text{opt}} = 0.0$ (for $n^* = 15000$) and $M_{\text{rand}} = 0.896$ (for $n^* = 30000$), which approach the best results ($M_{\text{opt}} = 0$ and $M_{\text{rand}} = 1$).

Question 10

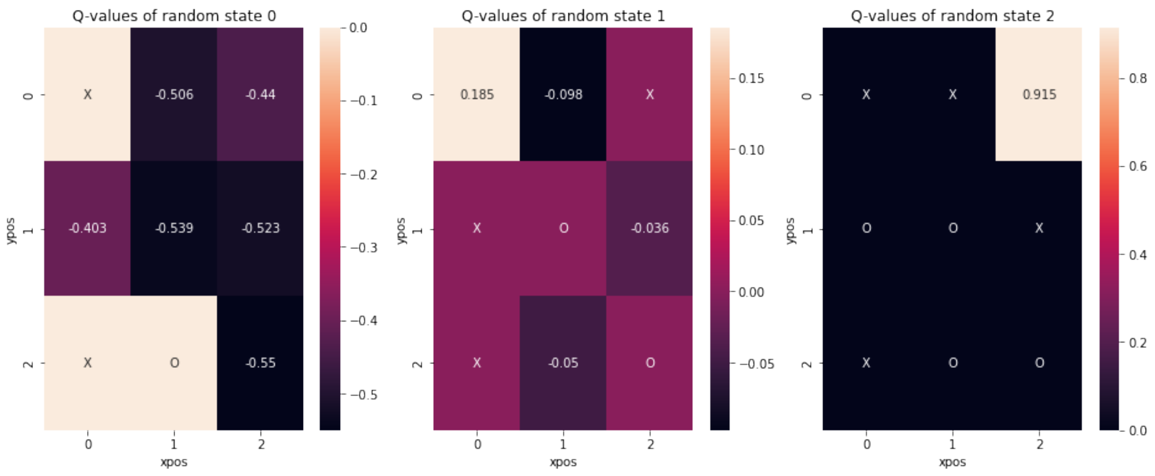

Figure 7: Heat map of Q-values of available actions for some three states on the Tic Tac Toe board for the best self-learning Q-learning agent, $\epsilon \in [0.1, 0.8], n^* = 15000, \alpha = 0.05, \gamma = 0.99$.

For the three board arrangements in Fig. 7, it can be seen that the best RL agent favors winning or blocking the opponent. As a reminder, player ╳ always moves first, and player ◯ always moves second, hence an equal number of ╳s and ◯s in the grid means that player ╳ is next, and a number of ╳s larger than a number of ◯s means that player ◯ is next. Each presented board configuration is a state observed by the RL agent before it plays.
Considering that, in the first arrangement, the highest Q-value is the middle left corner, which means that the next playing agent ◯ would attempt to block the winning move of the opponent by favoring that available space. In the second arrangement, the next playing agent ◯ would favor the top left corner which would secure a win in addition to blocking the move of the opponent. Finally, in the third arrangement, the next playing agent ╳ would attempt to win the opponent by playing the top right space.
We can conclude that the results make sense and that the RL agent learned the game well by self-practice.

2 Deep Q-Learning

2.1 Learning from experts

Question 11

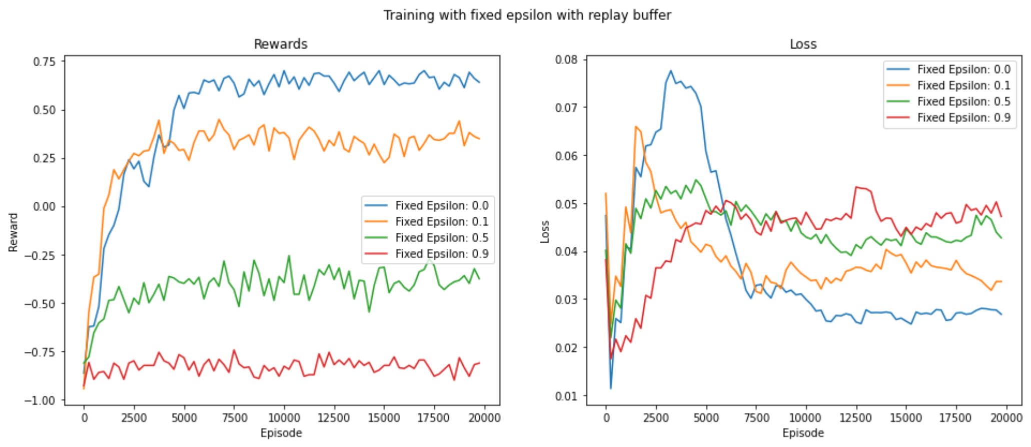

Figure 8: Plots of Deep RL Agent's rewards and training loss over the episodes for fixed ϵ trained against `Opt(0.5)`, $\epsilon_{\text{fixed}} \in \{0, 0.1, 0.5, 0.9\}$, $\alpha = 5 \cdot 10^{-4}$, $\gamma = 0.99$, with a replay buffer size of 10000, and batch size of 64.

From the plot in Fig. 8, the average reward increases over the number of games. The average training loss first increases during the first portion of the games but then decreases over the number of games. With a choice of $\epsilon_{\text{fixed}} = 0.1$, the final reward is positive and around 0.45. This means that the RL agent is learning how to play Tic Tac Toe.

Question 12

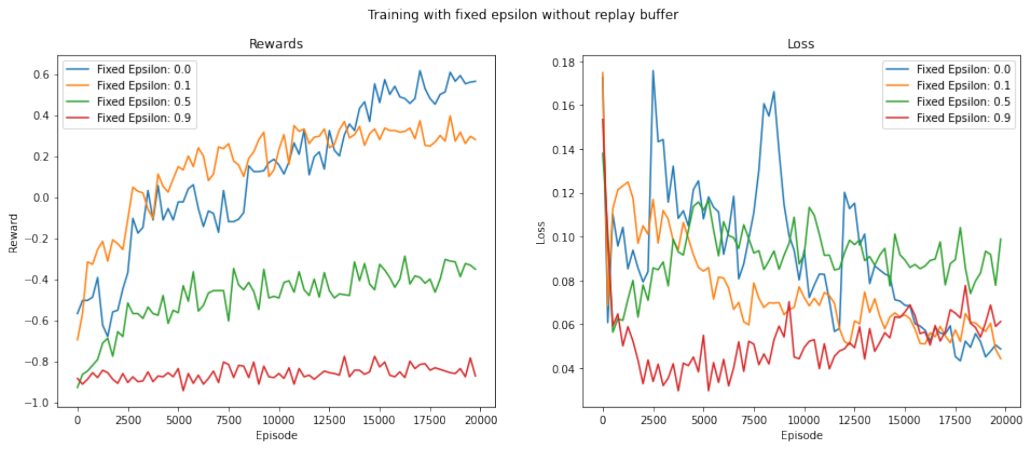

Figure 9: Plots of Deep RLAgent's rewards and training loss over the episodes for fixed ϵ trained against `Opt(0.5)`, $\epsilon_{\text{fixed}} \in \{0, 0.1, 0.5, 0.9\}$, $\alpha = 5 \cdot 10^{-4}$, $\gamma = 0.99$, without a replay buffer, and a batch size of 1.

When updating the DQN network by using only the latest transition, without the replay buffer and a batch size of 1, we can observe that the loss evolution over the number of games is less smooth. In addition to that, the resultant trained agent average reward after 20'000 games, for a given ϵ, is less than that of the agent trained using a replay buffer.

Question 13

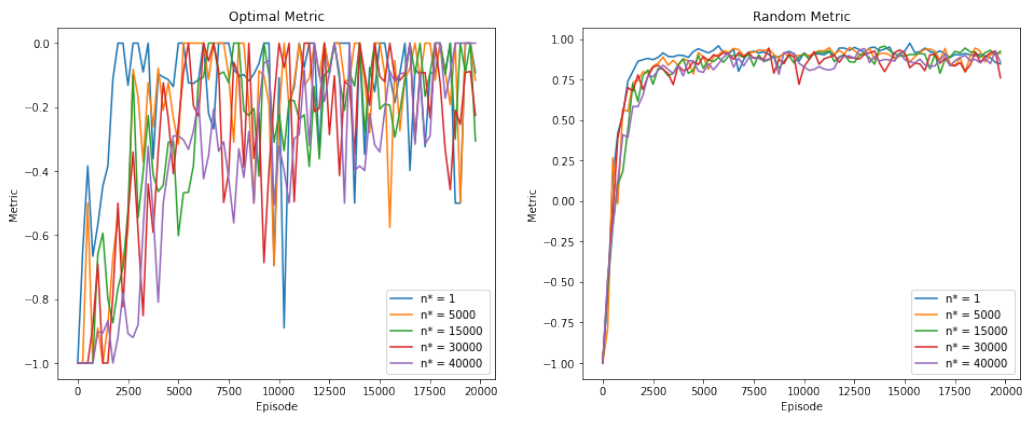

Figure 10: Plot of Deep RL Agent's M_{opt} and M_{rand} over the episodes for a decaying $\epsilon(n)$, trained against `Opt(0.5)`, $\epsilon \in [0.1, 0.8], n^* \in \{1, 5000, 15000, 30000, 40000\}$, $\alpha = 5 \cdot 10^{-4}$, $\gamma = 0.99$, with a replay buffer size of 10000, and batch size of 64.

As it can be seen in Fig. 10, with different ϵ decay rates (i.e. different initial exploration phases), there is no significant difference between agent performance over the optimal and random metrics. However, the performance of some agents is smoother than others. In addition to that, the rough evolution of the M_{opt} metric over the number of games can be explained by the fact that the DQN training algorithm outputs a model that, given any arbitrary state of the game, should output the corresponding Q-values so as to maximize the total return received. Considering that, the same weights should generalize well to different states as well as to different policies (random move, optimal move). However, as the model learns with different randomly selected batches (small batch sizes) from the replay buffer, it might see transitions not seen before and optimize its weights so as to decrease the loss to those observed training samples, which might cause the performance to fluctuate a lot over the games. Accordingly, we expect that using a larger batch size can smooth out the M_{opt} metric curve and avoid excessive fluctuations.

Question 14

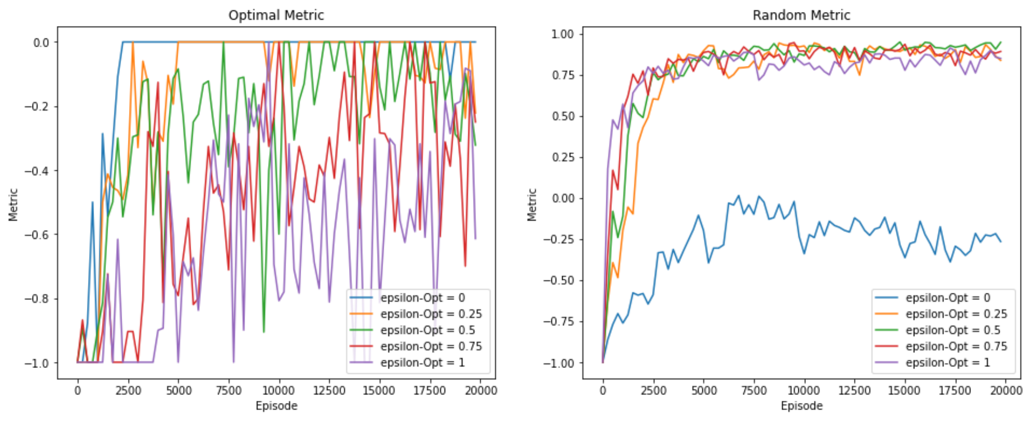

Figure 11: Plot of Deep RL Agent's M_{opt} and M_{rand} over the episodes for a decaying $\epsilon(n)$ trained against `Opt(`ϵ_{opt}`)`, $\epsilon_{\text{opt}} \in \{0, 0.25, 0.5, 0.75, 1\}$, $\epsilon \in [0.1, 0.8], n^* = 15000$, $\alpha = 5 \cdot 10^{-4}$, $\gamma = 0.99$, with a replay buffer size of 10000, and batch size of 64.

For M_{opt}, it can be observed that all agents except the one trained with `Opt(1)` manage at some point to reach the maximum level of expertise with respect to the optimal metric, and the speed of learning of other Deep RL agents worsens as ϵ_{opt} increases.

Now, in a similar fashion to Tabular Q-learning for M_{rand}, the Deep RL agent trained against the optimal `Opt(0)` is the worst performing with a random metric indicating that the agent was not able to win against a random agent `Opt(1)`. This can also be explained by the fact that this Deep RL agent has only seen optimal moves and thus was never able to explore a winning move. Hence, the learning of this Deep RL agent is overfit to counteract optimal agent moves.
In addition to that, the higher the optimal level, the lower the fluctuations in the evolution of the optimal metric. This can be explained by the fact that, as the random policy probability increases, more random states are observed and accordingly higher amount of information for the DQN model to learn and optimize its weights.
The optimality level of $\epsilon_{\text{opt}} = 0.25$ achieves the smoothest performance and has the best trade-off between speed of learning and regularization of learning.

Question 15

The highest values of the optimal and random metrics that we can achieve after playing 20'000 games are: $M_{\text{opt}} = 0.0$ (for $\epsilon_{\text{opt}} = 0$) and $M_{\text{rand}} = 0.946$ (for $\epsilon_{\text{opt}} = 0.5$), which approach the best results ($M_{\text{opt}} = 0$ and $M_{\text{rand}} = 1$).

2.2 Learning by self-practice

Question 16

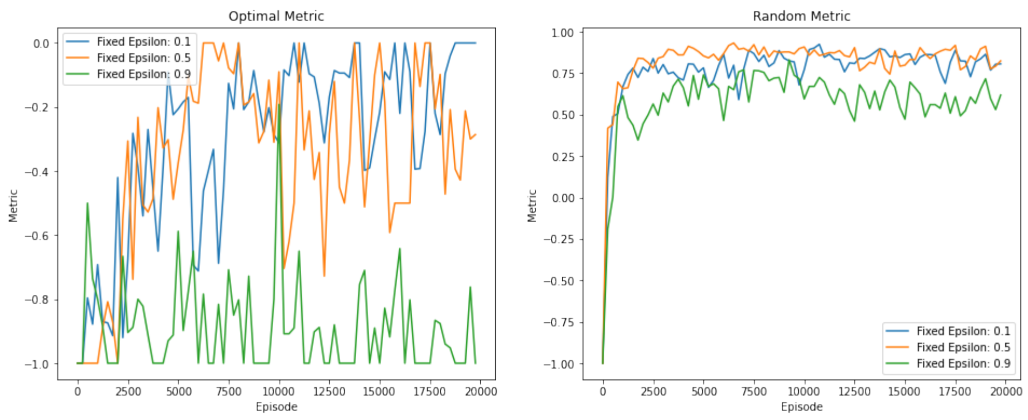

Figure 12: Plot of Deep RLAgent's M_{opt} and M_{rand} over the episodes for fixed ϵ trained against itself, $\epsilon_{\text{fixed}} \in \{0.1, 0.5, 0.9\}$, $\alpha = 5 \cdot 10^{-4}$, $\gamma = 0.99$, with a replay buffer size of 10000, and batch size of 64.

When training against itself, testing the Deep RL agent's performance shows that $\epsilon_{\text{fixed}} = 0.1$ achieves the best trade-off between the two performance measures, attaining a perfect score on the optimal metric, and a high score on the random metric. Hence, the agent learns to play Tic Tac Toe by self-practice, with small levels of exploration. For the optimal metric, the effect of a higher ϵ increases the exploration level which leaves the self-learning agents exploring random moves with little room for improving currently known strategies. Evaluating against the random metric shows a better performance for low/middle range ϵ-greedy agents, which better estimates good policies by exploiting them more.

Question 17

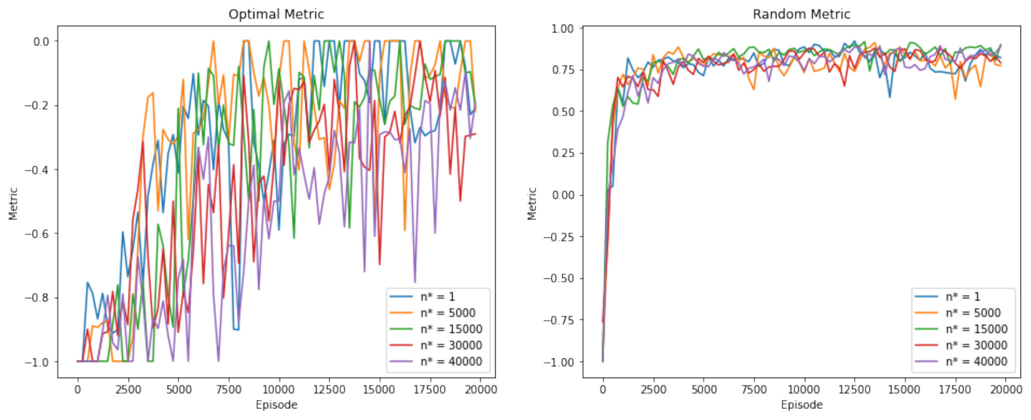

Figure 13: Plot of Deep RL Agent's M_{opt} and M_{rand} over the episodes for a decaying $\epsilon(n)$ trained against itself, $\epsilon \in [0.1, 0.8], n^* \in \{1, 5000, 15000, 30000, 40000\}$, $\alpha = 5 \cdot 10^{-4}$, $\gamma = 0.99$, with a replay buffer size of 10000, and batch size of 64.

As it can be seen in Fig. 13, with different ϵ decay rates (i.e different initial exploration phases), there is no significant difference between agent performance over the optimal and random metrics. However, the performance of some agents is smoother than others. The larger n^* is, the longer the decay from a large ϵ to a minimal ϵ_{min} value, and the slower the initial learning phase, since a higher ϵ translates to a higher exploration probability.
The decay allows agents with a high number of exploratory games n^* to explore more and proceed by exploiting the best estimated policy. In fact, the Deep RL agents self-trained with $n^* = 5000$ achieve the best results and smoothest learning after 20'000 games.

Question 18

The highest values of the optimal and random metrics that we can achieve after playing 20'000 games are: $M_{\text{opt}} = -0.19$ (for $n^* = 40000$) and $M_{\text{rand}} = 0.898$ (for $n^* = 15000$), which approach the best results ($M_{\text{opt}} = 0$ and $M_{\text{rand}} = 1$).

Question 19

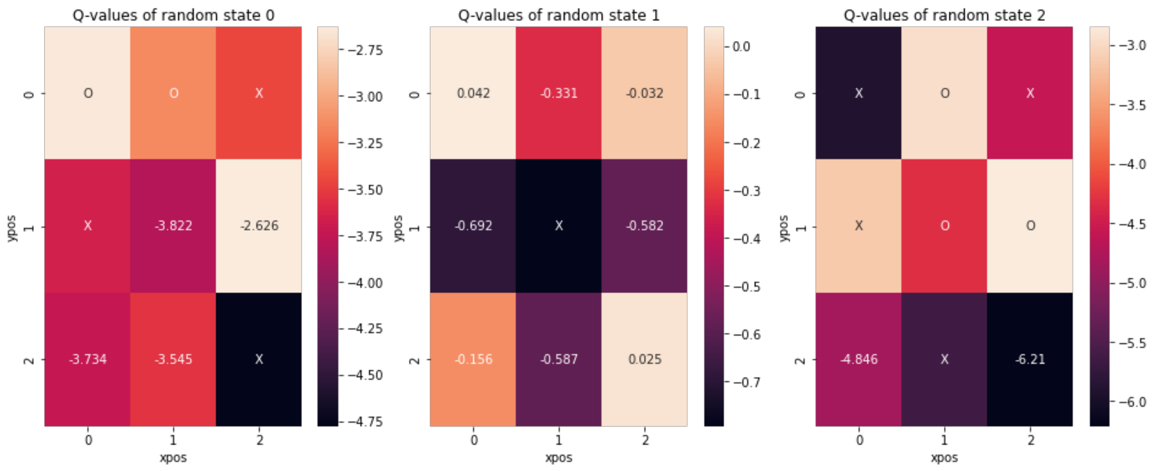

Figure 14: Heat map of Q-values of available actions for some three states on the Tic Tac Toe board for the best self-learning Deep Q-learning (DQN) agent, $\epsilon \in [0.1, 0.8], n^* = 15000$, $\alpha = 5 \cdot 10^{-4}$, $\gamma = 0.99$, with a replay buffer size of 10000, and batch size of 64.

For the three board arrangements in Fig. 14, it can be seen that the best Deep RL agent favors winning or blocking the opponent. As a reminder, player × always moves first, and player ○ always moves second, hence an equal number of ×s and ○s in the grid means that player × is next, and a number of ×s larger than a number of ○s means that player ○ is next. Each presented board configuration is a state observed by the Deep RL agent before it plays.
Considering that, in the first arrangement, the highest Q-value is the middle right corner, which means that the next playing agent ○ would attempt to block the winning move of the opponent × by favoring that available space. In the second arrangement, the center was marked first by ×, and according to the optimal strategy (from `tic_env.py`), the next playing agent ○ should favor a corner, and all corners of this arrangement have a much higher Q-value than the edges. Finally, in the third arrangement, the next playing agent ○ would attempt the block the opponent by playing the bottom left corner.
Furthermore, it can be noted that compared to the heat map in Fig. 7, filled cells can have a non-zero Q-value. This is explained by the fact that the deep network considers filled cells as valid actions, but with an end game penalty of $r_{\text{unav}} = -1$.
We can conclude that the results make sense and that the Deep RL agent learned the game well by self-practice.

3 Comparing Q-Learning with Deep Q-Learning

Question 20

	M_{opt}	M_{rand}	T_{train} **(games)**
Learning from experts	0.0	0.894	6750
Learning by self-practice	0.0	0.896	5500

Table 1: Best performances of the Q-learning agents and the corresponding training time.

	M_{opt}	M_{rand}	T_{train} **(games)**
Learning from experts	0.0	0.946	2250
Learning by self-practice	−0.19	0.898	6250

Table 2: Best performances of the Deep Q-learning (DQN) agents and the corresponding training time.

Question 21

In this project, we explored 2 different algorithms capable of training an agent to play the Tic Tac Toe game by reinforcement learning. One algorithm focused on estimating the Q-values and stores them in a look-up table while the other algorithm attempted to train a model which maps the state-action pairs to Q-values. The second algorithm is highly valued with problems with huge state-action spaces.
Considering that, by observing the results in Table 1 and Table 2, we can observe that the DQN approach resulted in higher performance when the agent is trained against an expert. Specifically, the resulting agent of both approaches learned perfectly the optimal policy as it can be seen through the values of M_{opt}, while the resulting agent of the DQN approach achieved a 5.8% increase in the random metric M_{rand}. Furthermore, the training time of the DQN algorithm against an expert is a third of that of Tabular Q-learning algorithm in the same training setup.
On the other hand, when observing the differences in performance in the two algorithms when training the agent using self-practice, it can be seen that there is no significant improvement compared to training the agents against a teacher. In fact, the resulting trained agent of the DQN model could not perfectly learn the optimal policy at the end of the 20'000 games; however, it still managed to get a perfect score in previous games (i.e. before reaching 20'000 games). In addition to that, training time increased by 13.6% (mainly due to the difficulty to learn the optimal policy).
Accordingly, we can conclude that the agents where able to learn the Tic Tac Toe game both against a teacher and against themselves. Furthermore, the DQN algorithm attempts to scale the Tabular

Q-learning algorithm in scenarios where building a Q-table is infeasible. The disadvantage of the DQN algorithm is that the training procedure might be less stable (more performance fluctuations) compared to the one of Tabular Q-learning. However, many tricks still exist to further stabilize the training procedure, such as using a larger batch size, different training hyper-parameters, and other tricks of the trade.