Mini-project 1: Tic Tac Toe

Federico Betti, Ivan Bioli
CS-456 Artificial Neural Networks, EPFL Lausanne, Switzerland

***Abstract*—We present the results obtained from training an agent to play Tic-Tac-Toe both against quasi-optimal strategy (up to some user-defined degree of randomness) and by self-practice, using standard Reinforcement Learning algorithms. Results obtained by tabular Q-Learning and by Deep Q-Learning are shown, and the two approaches are compared. The performance of the agent is tested against the optimal strategy and the fully random strategy, and different configurations of the training hyper-parameters are tested.**

I. Introduction

A. Notation

Throughout the report we use the same notation as in the project description, if not stated otherwise.

Moreover, in the following, we define as a *win-booking state* every state in which the current player has the chance to win the game, either with the next move or in two moves with a fork. Two examples of *win-booking state* are provided in Figure 7a and Figure 7c.

II. Q-Learning

In this section, we present the results obtained by training the agent with tabular Q-Learning. As the training trends may not be represented by a single training run, we performed 10 complete training runs for all the requested experiments. This helped in particular to dump the oscillations observed for the performance measure M_{opt}. Throughout this section, in the plots, solid lines represent the median over the training runs, while shaded regions show the 25 and 75 percentiles. Moreover, we present only those values of the parameters which are significant and representative of the main trends, referring to the notebook attached to the submission for the results of all the tested values.

Question 1

See Figure 1.

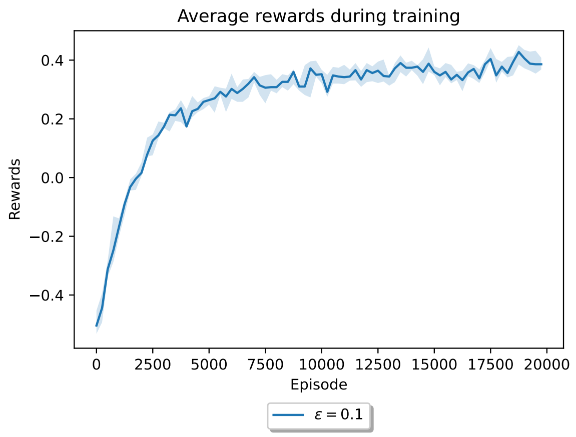


Fig. 1: Training with $\epsilon = 0.1$, an increase in average reward with experience is observed. At the end of training the average reward is positive, indicating that the agent wins more games than it loses against `Opt(0.5)`. Thus, the agent learns to play Tic-Tac-Toe.

Question 2

Figure 2 shows the average reward during training for different values of $n^\star$. It can be observed that using $n^\star \gg 1$ initially actions are chosen at random more frequently, therefore the average reward is lower than the one obtained with fixed $\epsilon = \epsilon_{\min}$. However, when $\epsilon(n)$ reaches $\epsilon_{\min}$ the average reward approaches the same values observed for fixed exploration rate, with the advantage of a having explored more exhaustively the state space. On the other hand, for very large values of $n^\star$ we never reach the constant $\epsilon_{\min}$-greedy policy, and therefore the average reward attains lower values as the agent keeps paying the price of exploration.

In conclusion, having a decaying exploration rate helps training compared to having a fixed one, if the chosen value of $n^\star$ guarantees that the agent attains the $\epsilon_{\min}$-greedy policy towards the end of training. Indeed, the final average reward is comparable, but more state-action pairs are explored. On the other hand, having a too high exploration rate throughout the whole training negatively affects the agent's reward.

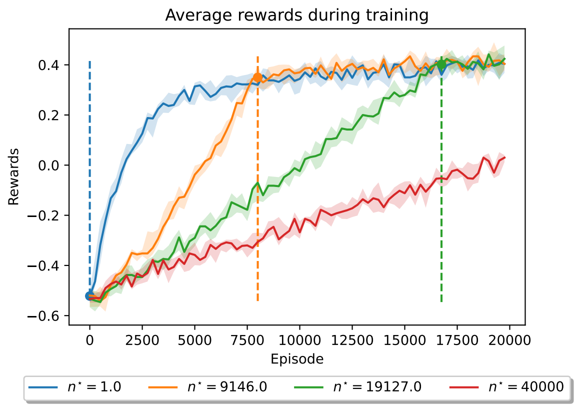


Fig. 2: Average rewards during training against `Opt(0.5)` for different values of $n^\star$: the vertical lines correspond to the episode at which the agent starts choosing moves with the minimum rate of exploration $\epsilon_{\min}$.

Question 3

As shown in Figure 3, for all the experimented values M_{opt} reaches the optimal value $M_{\text{opt}} = 0$, with a more gentle growth for higher values of $n^\star$ as exploration is more frequent (in expectation) during learning. Similarly to the average rewards shown in *Question 2*, higher values of $n^\star$ reach their plateau later during training. On the other hand, in this case also $n^\star = 40000$ reaches good performances in terms of M_{opt} and M_{rand}. This is because M_{opt} and M_{rand} are measured with the agent playing greedily, contrarily to the average reward which is an on-policy performance measure.

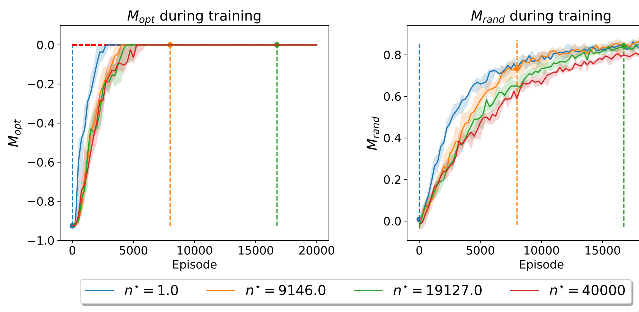


Fig. 3: Behaviour of M_{opt} and M_{rand} every 250 training episodes for different values of $n^\star$.

Question 4

As shown in Figure 4, when training against `Opt(0)` the agent quickly learns to draw against the optimal player, and M_{opt} reaches 0. In contrast, the performance against the random player M_{rand} does not improve significantly because there is no chance of ending up in a *win-booking state*. Therefore, all the Q-values $Q(s, a)$ with s a *win-booking state* remain equal to their initialization value zero. As a consequence, when the agent plays against the random policy and possibly reaches a *win-booking state*, it chooses the next action randomly, thus a lower M_{rand}.

Conversely, when training against `Opt(1)` M_{rand} quickly increases, while M_{opt} improves slowly without reaching zero. In this case, the agent learns to try to get a $+1$ reward by taking advantage of possible mistakes of his opponent. This implies leaving also to the adversary the chance to win, thus a lower M_{opt}. Training against `Opt(0.5)` both M_{opt} and M_{rand} increase significantly: the agent is able to draw against the optimal policy and to win against the random one. The values $\epsilon_{\text{opt}} = 0.2$ and $\epsilon_{\text{opt}} = 0.8$ show intermediate results.

In conclusion, the agent learns to gain optimal rewards (in expectation) against the opponent it is trained against. Indeed the optimal Q-values depend on the transition probabilities, which vary with the opponent. Compared with the agent trained against `Opt(0.5)`, choosing extreme values of ϵ_{opt} makes the agent gain little in one of the two performance measures and lose much in the other, resulting in a worse general performance.

Question 5

The highest values of M_{opt} and M_{rand} training with Q-Learning for 20000 games against `Opt(`ϵ_{opt}`)` have been achieved for $\epsilon_{\text{opt}} = 0.5$ using decreasing exploration with $n^\star = 9146$. They are equal to $M_{\text{opt}} = 0$, $M_{\text{rand}} = 0.85$ (median over 10 runs).

Question 6

We claim that $Q_1(s, a)$ and $Q_2(s, a)$ do not have the same values, as suggested by Figure 4. Indeed, the optimal Q-values,

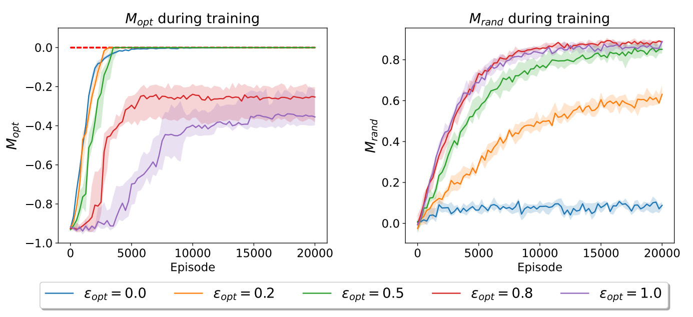


Fig. 4: Behaviour of M_{opt} and M_{rand} every 250 training episodes for different values of ϵ_{opt} and $n^\star = 9146$.

i.e. those satisfying the Bellman equation, represent the total expected (discounted) reward and depend on the transition probabilities, which are different if playing against `Opt(0)` or `Opt(1)`.

During training, Agent 1 never encounters *win-booking states*, hence at the end of training $Q_1(s, a)$ is equal to the initialization value zero for every *win-booking state* s and every action a. This is because the transition probability to any of these states is zero when playing against `Opt(0)`. Moreover, since Agent 1 can at most draw against `Opt(0)`, all the rewards are non-positive, thus also all the Q-values.

Conversely, Agent 2 encounters *win-booking states* during training and hence we expect $Q_2(s, a) = 1$ (possibly discounted by γ) for actions that make the agent win. As there exist strictly positive $Q_2(s, a)$, this proves that $Q_2(s, a) \neq Q_1(s, a)$ for some state-action pairs.

Question 7

Figure 5 shows the behaviour of M_{opt} and M_{rand} for different exploration rates ϵ of the learning agent. We observe that when the agent is trained without any exploration both performance measures are low, as the agent gets stuck performing sub-optimal actions. Increasing the exploration rate ϵ the performance measures improve and the agent learns to play Tic-Tac-Toe. However, picking a too large exploration rate negatively affects M_{opt}, while a low one degrades M_{rand}. The best trade-off between the two metrics is given by $\epsilon = 0.5$.

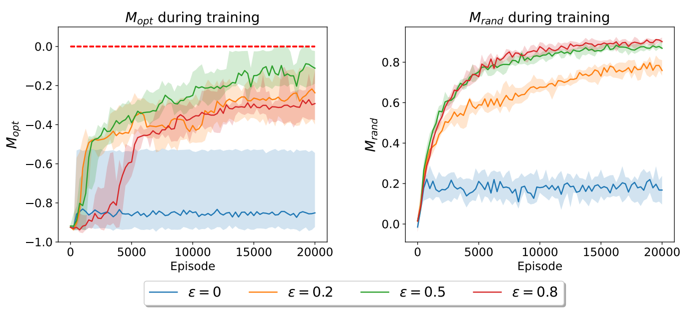


Fig. 5: Behaviour of M_{opt} and M_{rand} every 250 training episodes for different exploration rates ϵ.

Question 8

Similarly to *Question 3*, from Figure 6 it can be seen that decreasing ϵ helps training compared to a fixed one if the chosen value of $n^\star$ is representative of a well-designed exploration-exploitation trade-off. By prioritizing exploitation during training, i.e. using $n^\star = 1$, the agent gets stuck choosing sub-optimal actions. Using larger values of the $n^\star$ helps in improving M_{rand}, but using too high values negatively affects M_{opt}. The best trade-off between the two performance measures is given by $n^\star = 24460$, and the final performance is superior to the one obtained using fixed exploration rate $\epsilon = 0.5$.

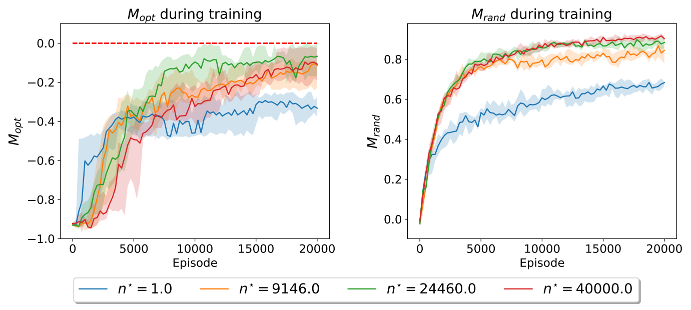


Fig. 6: Behaviour of M_{opt} and M_{rand} over the training episodes for different values of $n^\star$.

Question 9

The highest values of M_{opt} and M_{rand} training with Q-Learning by self practice for 20000 games have been achieved using decreasing exploration with $n^\star = 24460$. They are equal to $M_{\text{opt}} = -0.07$, $M_{\text{rand}} = 0.87$ (median over 10 runs).

Question 10

In Figure 7a the highest Q-value correctly corresponds to the action that makes the agent (player X) win, but it is not equal to 1. Furthermore, picking any of the other two empty corners would correspond to a fork, but the estimated Q-values are much lower than $\gamma = 0.99$. Similarly, in Figure 7c the agent correctly picks the center doing a fork, which brings to victory in one more move, but the Q-value of this action is far from γ, and the estimates for other win-leading moves such as the south-east corner are even worse. In Figure 7b the agent (player O) correctly blocks the win of the adversary, predicting a negative reward for any other action.

It can be argued that the agent learned the game fairly well, as in the above described scenarios it plays one of the possible correct actions. However, the number of state-action pairs is too high to be sufficiently explored in 20000 training episodes with tabular Q-Learning, hence the estimated Q-values did not converge to their true values, i.e. the ones satisfying the Bellman equation.

III. Deep Q-Learning

In this section, we present the results obtained by training the agent with Deep Q-Learning (DQN). Similarly to the previous section, the results are obtained from an average over multiple training runs. Due to longer training times, we limited ourselves to 4 training runs for each of the question. In the following plots solid lines represent the median, while shaded areas the 25 and 75 percentiles. Again, we show only the values of the parameters which are representative of the main training trends,

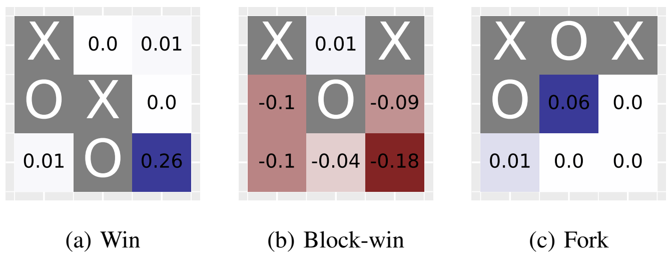


Fig. 7: Q-values of available actions in three different board arrangements. The player moving first is always X.

referring to the notebook attached to the submission for the results of all the tested values.

We fine tuned the learning rate and the optimal value turned out to be $\alpha = 10^{-4}$ for both the learning by experts and the learning by self-practice.

Question 11

See Figure 8.

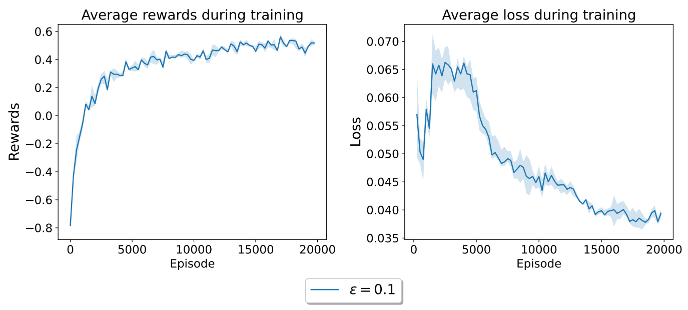


Fig. 8: Training with $\epsilon = 0.1$, the loss initially increases, but it flattens when the replay buffer is filled up and then decreases. The reward is initially negative as the agent chooses unavailable actions, but it increases with experience and becomes positive, indicating that the agent learns to play Tic-Tac-Toe.

Question 12

See Figure 9.

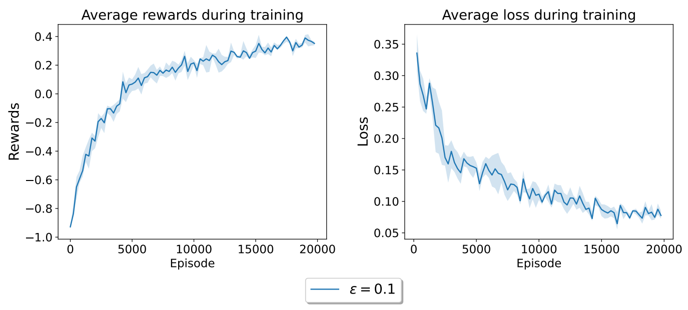


Fig. 9: Using only the latest transition, the loss is significantly higher than in Figure 8 and decreases from the beginning. The reward increases less steeply and attains a lower value. Hence using a replay buffer and a larger batch-size helps learning by avoiding temporally correlated weight updates.

Question 13

Figure 10 shows the effect of the exploration decay rate $n^\star$ on the performance measures M_{opt} and M_{rand}. It can be seen that for $n^\star = 1$, i.e. constant rate of exploration $\epsilon = \epsilon_{\text{min}}$, M_{opt} increases rapidly in the initial stages of training. In general, the higher the value of $n^\star$, the slower the convergence of M_{opt} to the optimal value $M_{\text{opt}} = 0$. For all the experimented values of $n^\star$ the performance measure M_{rand} rapidly grows above the value $M_{\text{rand}} = 0$, i.e. the agent learns the rules of the game and stops choosing unavailable actions. When using $n^\star = 1$ M_{rand} has a more gentle growth and flattens after reaching a sub-optimal plateau. Using larger values of $n^\star \gg 1$ helps in achieving a higher M_{rand}, with a comparable final performance for all values shown in the plot. Furthermore, in the initial stages of training the metric M_{opt} shows larger oscillations for higher values of $n^\star$, due to a greater rate of random moves.

In conclusion, using a decreasing exploration rate ϵ helps compared to a fixed rate $\epsilon = \epsilon_{\text{min}}$, which results in a lower final value of M_{rand}. However, choosing $n^\star$ too high, e.g. $n^\star = 40000$, increases the training variance and does not provide benefits in terms of the performance measures M_{opt} and M_{rand}. Intermediate values of $n^\star$, in our case $n^\star = 10000, 20000$, are the ones providing the best trade-off. Thus for the following question we choose $n^\star = 20000$.

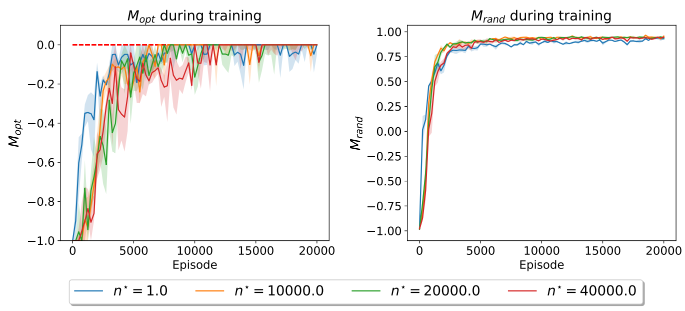


Fig. 10: *Question 13*: behaviour of M_{opt} and M_{rand} every 250 training episodes for different values of $n^\star$.

Question 14

As shown in Figure 11, training with $\epsilon_{\text{opt}} = 0$ the agent quickly learns to draw against the optimal player and M_{opt} stabilizes at 0 in the early stages of training. Conversely, the performance against the random player M_{rand} does not improve significantly and remains below zero. Indeed `Opt(0)` has a deterministic policy, thus when training against it there is no chance of visiting some of the possible states, such as *win-booking states*. When, during testing, the agent plays against the random policy and possibly reaches one of these states, it chooses the next move almost randomly and might even choose one of the unavailable actions, thus the negative average reward.

Increasing ϵ_{opt}, M_{rand} significantly improves for all values of ϵ_{opt}, with a slightly more gentle growth for smaller values of ϵ_{opt}. However, increasing it too much makes learning unstable and degrades the performance measure M_{opt}. Indeed, for the extreme case $\epsilon_{\text{opt}} = 1$, the agent learns to try to get a $+1$ rewards by taking advantage of possible wrong moves of the adversary. This implies exposing itself to the risk of losing the game: while the chance is frequently not taken by the random player during training, it negatively affects M_{opt} when testing.

Coherently with the results of *Question 4*, it can be argued that when learning by experts the agent learns to gain optimal rewards against the opponent it is trained against, as the transition probabilities depend on the opponent. The best trade-off between the two performance measures is obtained for ϵ_{opt} around 0.5.

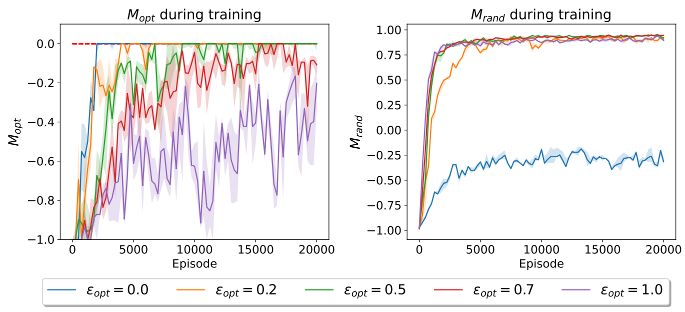


Fig. 11: Behaviour of M_{opt} and M_{rand} for different values of ϵ_{opt} and $n^\star = 20000$.

Question 15

The highest values of M_{opt} and M_{rand} training with DQN for 20000 games against `Opt(`ϵ_{opt}`)` have been achieved for $\epsilon_{\text{opt}} = 0.5$, using decreasing exploration with $n^\star = 20000$ and learning rate $\alpha = 10^{-4}$. They are equal to $M_{\text{opt}} = 0$, $M_{\text{rand}} = 0.94$ (median over 4 runs).

Question 16

Figure 12 shows that for $\epsilon = 0$, i.e. greedy policy, the agent does not learn to play as it gets stuck playing sub-optimal actions. On the contrary, taking $\epsilon > 0$ both M_{opt} and M_{rand} significantly increase during training and the agent learns to play Tic-Tac-Toe. Coherently with the results of *Question 7*, too high values of ϵ lead to a worse performance against the optimal player, while too low values result in a less sharp increase of M_{rand}. Thus, the best trade-off between the two performance measures was obtained with $\epsilon = 0.5$.

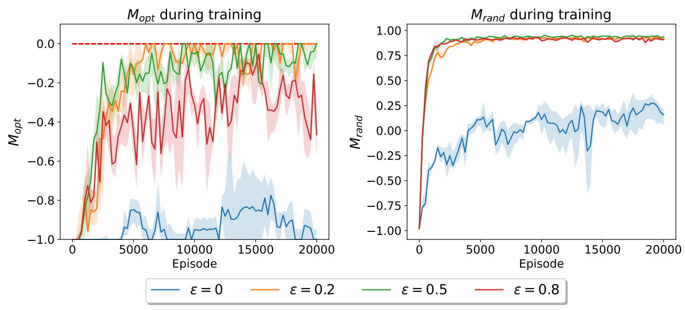


Fig. 12: Behaviour of M_{opt} and M_{rand} for different values of the exploration rate ϵ.

Question 17

Figure 13 shows that all values of the decaying exploration rate have similar performances in terms of M_{opt}, reaching the optimal value $M_{\text{opt}} = 0$ towards the end of training. However, a too high exploration rate, e.g. using $n^\star = 40000$, results in a more oscillating trend due to a greater rate of random moves.

On the other hand, choosing constant rate of exploration $\epsilon = \epsilon_{\text{min}}$, M_{rand} flattens at a sub-optimal level, while the performance is similar for $n^{\star} = 10000, 20000, 40000$. Hence, using decreasing exploration with intermediate values of $n^{\star}$ helps training. The best trade-off between the two performance measures was given by $n^{\star} = 10000$.

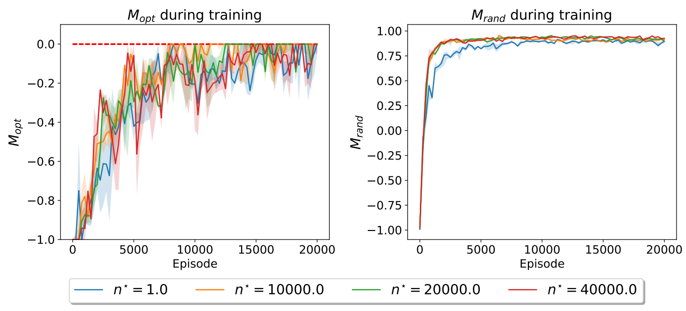


Fig. 13: Behaviour of M_{opt} and M_{rand} for different values of $n^{\star}$.

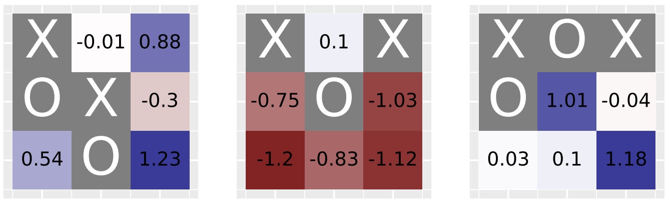
Question 18

The highest values of M_{opt} and M_{rand} training with DQN by self practice for 20000 games have been achieved using decreasing exploration with $n^{\star} = 10000$ and learning rate $\alpha = 10^{-4}$. They are equal to $M_{\text{opt}} = 0$, $M_{\text{rand}} = 0.91$ (median over 4 runs).

Question 19

In Figure 14a, the highest Q-value correctly corresponds to the move that makes the agent (player X) win, but it is higher than the reward $r = +1$. Moreover, playing in any of the other two empty corners corresponds to a fork, thus the related Q-values are correctly positive, even if not equal to the true value $\gamma = 0.99$. In Figure 14b the agent (player O) blocks the opponent's win and predicts a strongly negative reward for all other actions. Finally, in Figure 14c the agent (player X) correctly picks one of the two possible fork moves, and both have a Q-value close to one.

In conclusion, it can be argued that the agent learned the game well: in the above described board arrangements, the agent picks one of the possible correct actions and the Q-values do make sense. However, as expected since we are using a neural network to approximate the Q-values, the Q-values do not exactly match the true ones and in particular they do not necessarily range in the interval $[-1, 1]$.



(a) Win (b) Block-win (c) Fork

Fig. 14: Q-values of available actions in three different board arrangements. The player moving first is always X.

IV. Comparing Q-Learning with Deep Q-Learning

Question 20

See Table I.

Learning algorithm	M_{opt}	M_{rand}	T_{train}
Q-Learning by experts	0.00	0.85	6625
Q-Learning self-practice	-0.07	0.87	6625
DQN by experts	0.00	0.94	3500
DQN self-practice	0.00	0.91	4000

TABLE I: Best performance and corresponding training time for Q-Learning (median over 10 runs) and DQN (median over 4 runs).

Question 21

From Table I it can be observed that when the learning agent has a "teacher", both Q-Learning and DQN reach the optimal value $M_{\text{opt}} = 0$, but the final performance of DQN is superior in terms of M_{rand}. However, DQN provides the greatest advantages when learning by self-practice: in addition to a higher M_{rand}, DQN stably reaches the optimal value $M_{\text{opt}} = 0$, while Q-Learning does not.

DQN showed a more oscillatory behaviour during training, especially in terms of M_{opt} and in the early stages of training. This is mainly due to the fact that the actions' Q-values for different states are not independent, as if using a look-up table, but share the same weights. Therefore, the weight updates affect all the Q-values, leading to a more unstable learning. Moreover, in the early stages of training the replay buffer is not completely full and instability might also be caused by partially temporally correlated weight updates.

Furthermore, the training time was consistently lower for DQN and the heatmaps showed estimated Q-values closer to the true ones for DQN. This is even more striking if one takes into account that the agent trained with DQN had also to "learn the rules of the game", because actions' selection was not constrained to only available moves. This faster learning can be explained by the representation of the Q-values using shared weights, which allows to modify also the Q-values of unseen state-action pairs, and by the use of replay buffer, which reduces the sample complexity.

In conclusion, it can be argued that since the state space is discrete but contains a too high number of state-action pairs to be exhaustively explored in 20000 games, modelling Q-values by a neural network and learning the weights, such as in DQN, allows to have a faster and more reliable learning compared to tabular Q-Learning.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.