Mini-project 1: Tic Tac Toe

Benedek Harsányi[a] **and Mels Loe Jagt**[b]

[a]School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne; [b]Life Sciences Engineering, École Polytechnique Fédérale de Lausanne

This manuscript was compiled on June 5, 2022

2. Q-Learning

2.1 Learning from experts.

Question 1. Different Q-learning agents with varying ϵ are trained against Opt(0.5) (Figure 1A). As observed, Q-learning agents with lower ϵ perform better and overall learns rather quick. This plot depicts the exploration vs exploitation dynamic where we specifically do not plot for $\epsilon = 0$.

Question 2. It can be observed that increasing n^* slows down the learning rate of the Q-learning agent (Figure 1B). This is well in line with the result of **Question 1**. A higher n^* means that the rate in which ϵ decreases is lower. In **Question 1** we observed that for $\epsilon \in [0.1, 0.9]$, a lower ϵ leads to a higher average reward and thus improved learning. Corollary a higher n^*, where $\epsilon_{min} = 0.1$ and $\epsilon_{max} = 0.8$, leads to slower learning as ϵ_{min} is reached at a slower rate. Similarly, decreasing n^* leads to a faster learning.

Question 3. Two different dynamics are observed when we test either against M_{opt} or M_{rand} (Figure 1C). For M_{opt} it can be seen that the performance converges to 0, whereas for M_{rand} it is more slowly reaching for 1. Both values are in line with the expectation as playing against Opt(0) and Opt(1) will respectively lead to always tying or most of the times winning. Moreover since Opt(0) is more deterministic than Opt(1) we see more stable convergence for M_{opt} than for M_{rand}. In addition we see again that lowering n^* increases the learning rate in accordance with **Question 2**.

Question 4. So far $n^* = 1000$ and $n^* = 1$ have shown similar performance. As we favor a bit more exploration during the learning process, we selected $n^* = 1000$ to be optimal for this exercise. When training against Opt(ϵ_{opt}) for different values of ϵ_{opt}, again a different behavior between M_{opt} and M_{rand} can be observed (Figure 1D). That is, for M_{opt} we again see quick converge to 0 where we only find that training against Opt(0) has a slightly lower learning rate. In contrast, for M_{rand} we see that learning performance substantially depend on Opt(ϵ_{opt}). In fact, it can be observed that learning performances increases as ϵ_{opt} decreases. Hence, training against an optimal player with a lower degree of randomicity improves performance. This makes sense as playing against a player that only moves with random actions does not make the Q-learning agent better. In contrast playing against an optimal player where ϵ_{opt} is low does punish and reward bad and good moves respectively more adequately. Hence, in this scenario the Q-learning agent learns better.

Question 5. The optimal values $M_{rand} = 0.892$ and $M_{opt} = 0$ are found.

Question 6. $Q_1(s, a)$ and $Q_2(s, a)$ will not have the same values as they play against opponents with a substantially different policy. Playing against Opt(0) generates a rather deterministic opponent. Here every wrong move by the Q-learning agent will be punished directly. Accordingly, the Q-learning agent will explore a rather deterministic subspace of moves that he can play. In contrast playing against Opt(1) generates an opponent with completely random moves. For this reason some moves that would lead to a loss against Opt(0) can eventually lead to a win against Opt(1). Hence another subspace of moves will be discovered and rewarded, punished differently. Hence the optimal landscapes of Q-values are not the same.

2.2 Learning by self-practice.

Question 7. Overall, it can be observed that the Q-learning agent definitely learns by playing games against itself (Figure 2A). The degree of learning depends on ϵ. Interestingly, it seems that for M_{rand} a pattern can be observed where increasing ϵ (i.e., higher randomicity) yields better performance. For M_{opt} however this trend is not observed. In fact having a large ϵ rather slows down learning and no further distinct pattern can be find here. Strikingly, as depicted, M_{opt} does not consistently converge to 0 as seen before. This can be explained as we do not play against an optimal policy with a deterministic strategy. Learning by self-practice can lead to an optimized Q-value value landscape that does not perfectly match the predefined optimal policy.

Question 8. As decreasing n^* accelerates the rate where ϵ decreases, we expect based on **Question 7** a proportional relationship between n^* and the learning rate for M_{rand}. This can indeed be observed if viewed very closely Figure 2B where a larger n^* has slightly improved performance. Moreover, similar to **Question 7** we do not observe a pattern for M_{opt} other than a rather stochastic behavior for $n^* = 40'000$. The latter relates to having a large ϵ for an extended period of time. This could explain the pattern observed as playing against a random version of yourself results in random strategies against the optimal policy that sometimes work out better than other times.

Question 9. The optimal values $M_{rand} = 0.754$ and $M_{opt} =$ -0.186 are found.

Question 10. We visualize three states and the learned Q-values of the self-learning agent with $n^* = 20000$. In position one and three the agent finds the optimal move (for more info see Q20), but the agent never encountered the second position before, it is unable to generalize and find the optimal move here (2,1) setting up a fork.

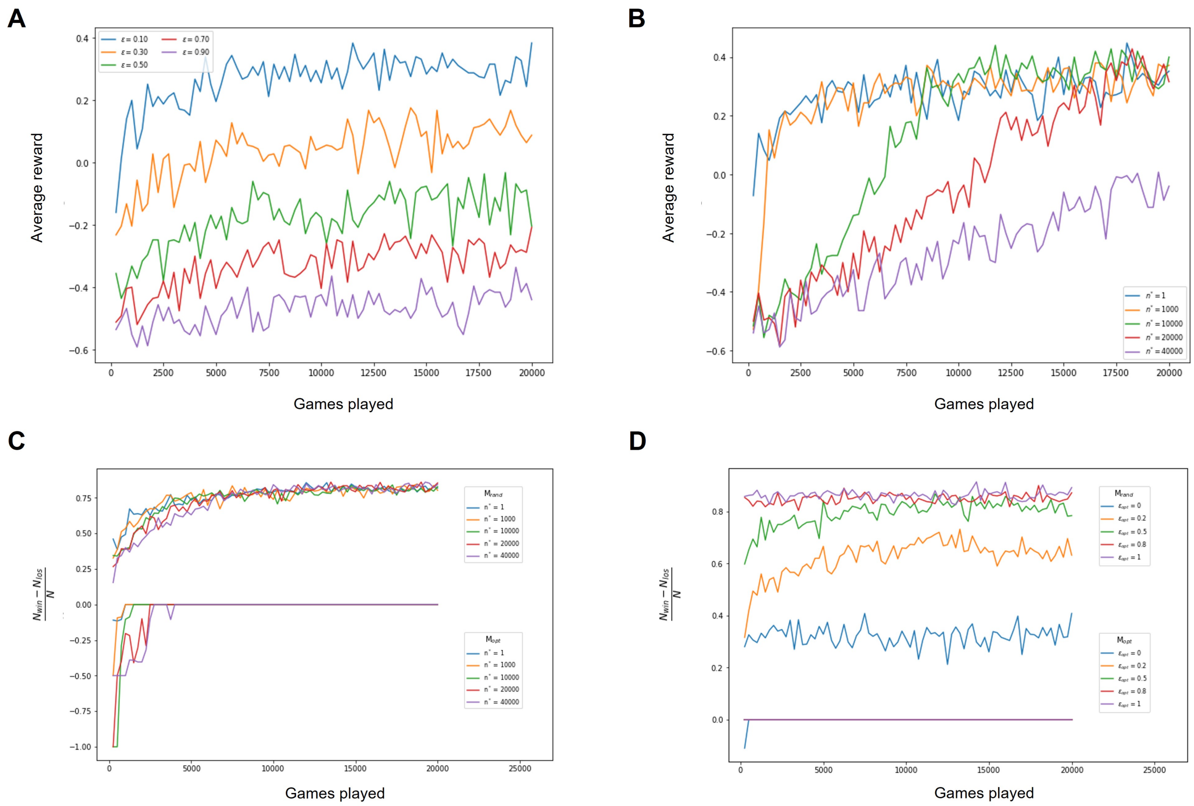

Fig. 1. Figures complementing part **2.1 Learning from experts**

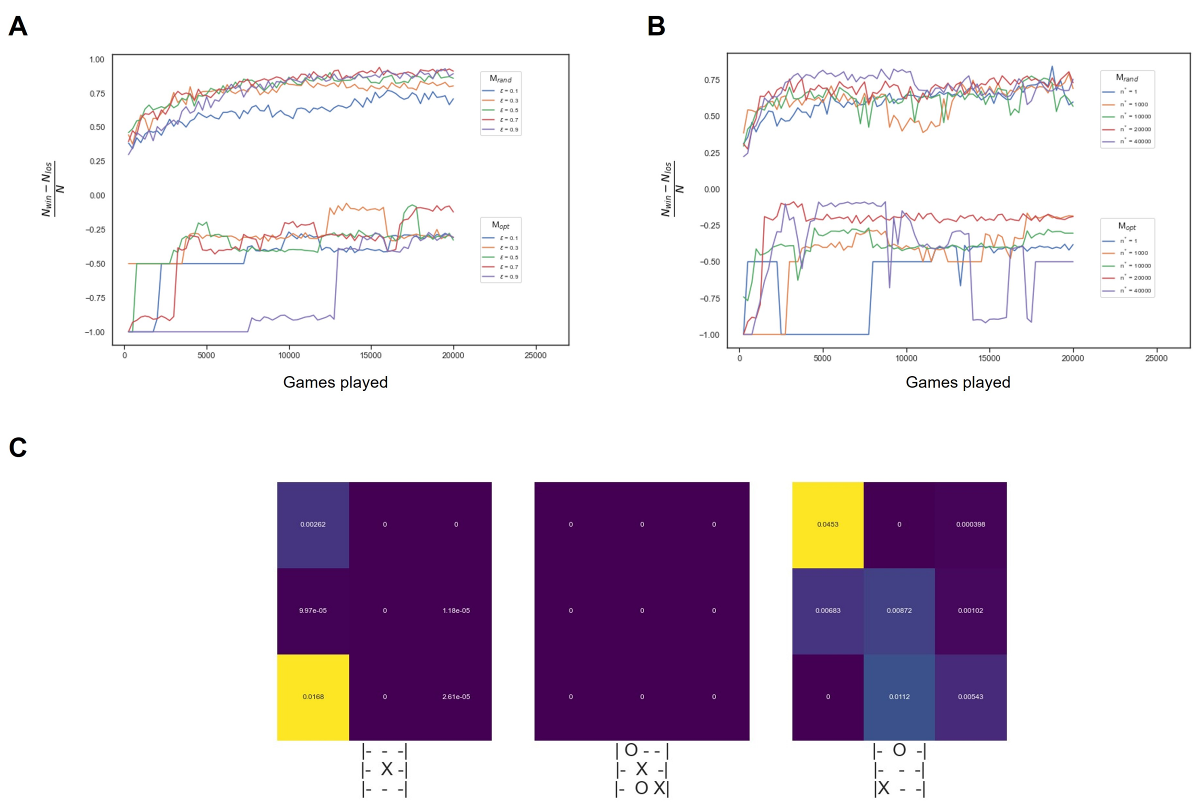

Fig. 2. Figures complementing part **2.2 Learning by self-practice**

3. Deep Q-Learning

3.2 Learning from experts.

Question 11. The Deep Q agent is able to produce similar results as Tabular Q learning in terms of average reward. The loss function seems to decrease until 10000 played games. A possible explanation could be, that we found a local minima, which we cannot escape with a batch size of 64.

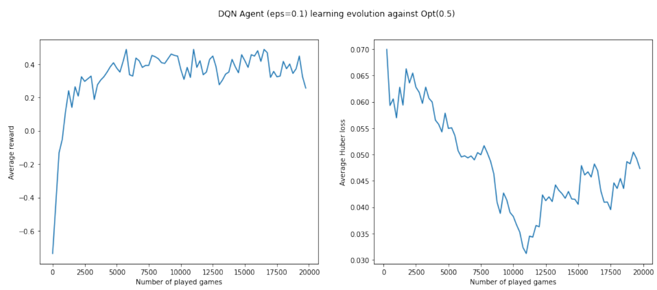

Fig. 3. Question 11

Question 12. The agent is again able to achieve the 0.4 average reward, however convergence time is significantly longer, it cannot use historical playouts to learn. On the other hand the agent can fit the Q-values almost six times better compared to buffer DQN, because it always optimizes against one training example

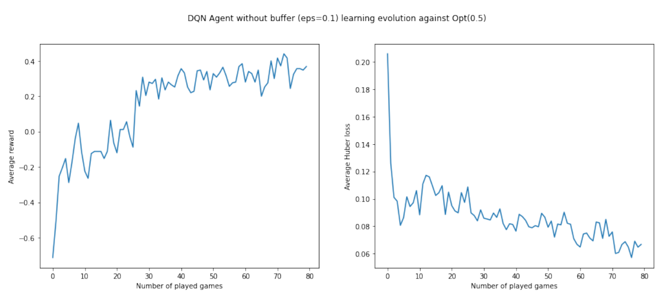

Fig. 4. Question 12

Question 13. The agents learn to beat the random opponent quickly, they are all able to achieve 0.75 score, which is close to the ability of Tabular Q. Smaller n^*s seem to work better, with 1 (using constant exploration rate) and 1000 the curves are less noisy and the agents can win almost every game against the random opponent. As for the optimal opponent the results are much noisier. At some point of the training all agents are able to draw every game, but later with larger n^* the neural network may sample non-optimal moves, which may influence the Q-values of the optimal moves as well. Again $n^* = 1000$ seems to produce the best results with barely any noise. We point out, that the performance usually drops to -0.5, this corresponds to the fact that the agent can draw with X, but loose the games with O or vice versa. All in all decreasing epsilon slowly does not help learning, keeping it fixed at 0.1

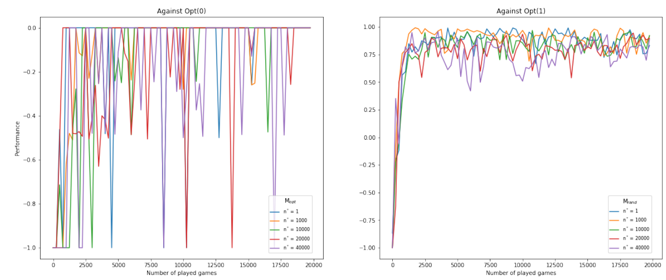

Fig. 5. Question 13

or quickly decrease it to 0.1 works best for DQN. We choose $n^* = 1000$, because of its stability against the optimal player.

Question 14. If our opponent uses some exploration level, our agent is able to beat the random player after some iterations, however if we train against the optimal player, the agent is not able to generalize, we overfit the optimal strategy (this can be seen from the other plot, in the testing phase we can achieve $M_{opt} = 0$ quickly and steadily) and cannot win against random moves. On the other hand if we train against random or almost random opponent ($\epsilon \in \{0.8, 1\}$) our agent will perform even more noisier against the optimal player. Sometimes it is able to draw every game, but other times it just looses every game. The best option seems to be choosing a low exploration rate for the opponent ($\epsilon \in \{0.2, 0.5\}$). In this setting DQN quickly learns to beat the random player, but also draw against the optimal policy as well. Based on the results the best choice if using $n^* = 1000$ for DQN and training against Opt(0.2).

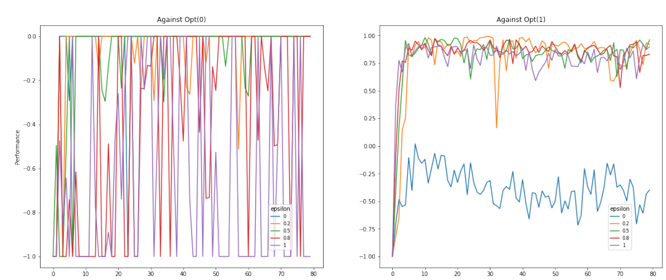

Fig. 6. Question 14

Question 15. To conclude, an agent using a neural network for learning -values with memory buffer is able to learn Tic Tac Toe, it is able to achieve the best possible results $M_{opt} = 0$ against optimal policy, and almost perfect result $M_{rand} = 0.996$ against random policy.

1. Learning by self-practice

Question 16. Using random policy ($\epsilon = 1$), the agent is not able to learn to play neither against the random, nor against the optimal player. Without any exploration the agent performs poorly against the optimal player, but achieves high consistent scores against the random player. The in-between exploration rates show similar behaviour, against random player they quickly figure out how to play for a win, but as the training continues it tends to forget, what it learned before. As for the optimal player, the agents perform similarly as before. The performance is very noisy, sometimes it is able to draw every game, but other times it just looses.

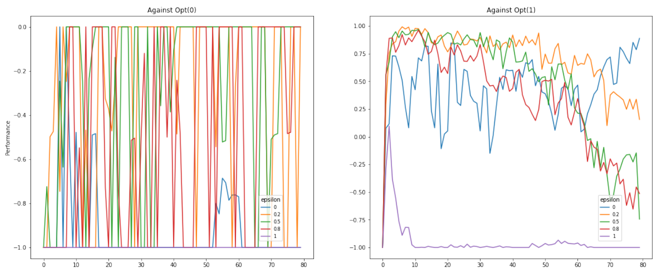

Fig. 7. Question 16

Question 17. Decreasing the exploration level seems to stabilize learning against random opponent. Setting $n^* = 1000$ yields the best performance. In the early stages it is beneficial to explore, but quickly (around 2500 games), the performance reaches a plateau. With the reduction of epsilon DQN agent is able to keep its performance level. As for the optimal player, decreasing the exploration level does not help either, the results are still really noisy.

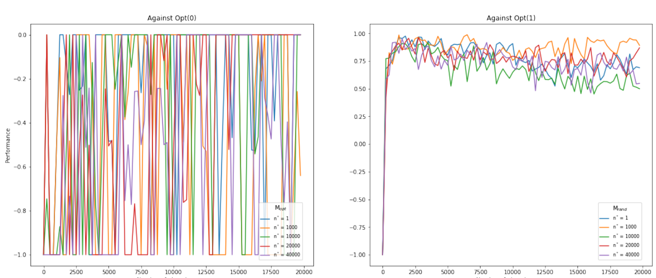

Fig. 8. Question 17

Question 18. At some stage of the learning self-learning agent is able to produce a value 0.988 for M_{rand}, and 0 against optimal player. The best choice seems to be using $n^* = 1000$, and simply stopping the training at an intermediate point where $M_{opt} = 0$.

Question 19. We visualize three states and the learned Q-values of the self-learning agent with $n^* = 40000$.

- In the first position, with optimal play the game ends with a draw if the agent chooses to play in some of the corners. As it can bee seen from the Q-values it finds one of the best moves. The agent learns to play defensively.

- In the second case the agent confidently finds the fork possibility, it is able to convert a winning position. It can play offensively.

- This is a tricky position, which requires long-term planning. By playing (0,0), the opponent is forced to block on (1,0). The resulting state will present with a fork-possibility on (2,2). The DQN agent was able to propagate the reward from the final position and it can plan ahead multiple moves.

Fig. 9. Question 19

4. Comparing Q-Learning with Deep Q-Learning

Question 20. Summary results and training times.

	model	Training time of M_opt	training time of M_rand	best M_opt	best M_rand
0	DQN against Opt(0.5)	9	5	0.000	0.996
3	DQN self-practice	8	3	0.000	0.988
1	Q-Learning against Opt(0.5)	2	10	0.000	0.892
2	Q-Learning with self-practice	5	12	-0.186	0.754

Fig. 10. Question 20

Question 21. We investigated the question, whether it is possible to learn Tic Tac Toe, an abstract strategy game,which involves no hidden information using various reinforcement learning techniques. An important degree of freedom during learning is the policy function. After several simulations, we can conclude, that in the early phase of training it is beneficial to explore more, and later aim towards exploitation of the learned policy, this is achieved through a clever choice of decreasing exploration function combined with the greedy policy. The policy function is based on the Q-values, which are numerical representation of a state-action pair. Q-learning is a tabular method to approximate these values. It requires to store (at most) $9 \cdot 9^3 = 6561$ Q-values. An other method, DQN uses neural nets to approximate the Q-values, in our case using $[18 \cdot 128 + 128] + [128 \cdot 128 + 128] + [128 \cdot 9 + 9] = 20105$ parameters. DQN is able to learn faster against random opponent, on the other hand both self-practice and against experts training favors Q-learning as regards the test phase against the optimal player. DQN by the virtue of NNs able to generalize easily and finds the optimal move even in previously unseen positions as well, while Q-learning can only solve encountered situations. In both cases choosing both the expert's and the agent's exploration level is critical. As far as self learning is concerned it turned out to be a competitive alternative form of training. With Q-learning we observed slower convergence time, while DQN produced a really noisy performance level against the optimal player, this may be reduced by playing with different hyper-parameters, such as buffer size, learning and discount rate, number of hidden layers, neurons or the criterion and the optimization method.