Artificial Neural Networks
Tic-Tac-Toe Miniproject

Gianni Lodetti Arina Lozhkina

***Abstract*—In this project we have implemented Q-Learning and Deep Q-Learning algorithms for the game Tic Tac Toe. Within our work we have tried to study the effectiveness of the algorithm under different configurations of M_{opt} and M_{rand} learning metrics, and answer the following questions:**

- **Can RL algorithms learn to play Tic Tac Toe by playing against optimal policy?**
- **Can an RL algorithm learn to play Tic Tac Toe only by playing against itself?**
- **What are the pros and cons of Deep Q-Learning compared to (Tabular) Q-Learning?**

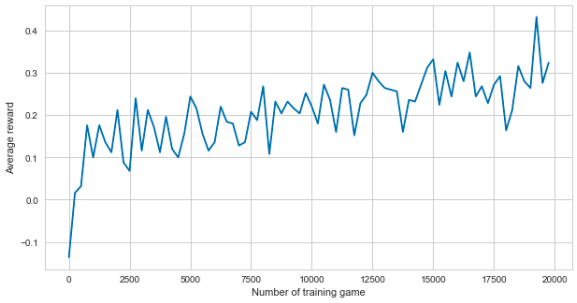

Figure 1: Average rewards.

I. Q-Learning

We first implemented the Q-Learning algorithm with hyperparameters: α = 0.05 and the discount factor = 0.99. We trained the algorithm over 20,000 games against an optimal player with an epsilon parameter of 0.5, with different players alternately making the first move.

II. Question 1. Can RL algorithms learn to play Tic Tac Toe by playing against optimal policies?

A. 2.1 Learning from experts

- **Question 1** We have chosen a value of 0.1 as the exploration level for the epsilon greedy strategy. This is a small value in order to follow the strategy of the Q-Learning algorithm, but add a stochastic element to reveal new states of the system. When representing the average values for every 250 games, we got Figure1. Judging by its dynamics, we can conclude that the algorithm is learning. After a sharp increase, the graph practically comes to a plateau with the value of 0.4.

B. 2.1.1 Decreasing learning

To improve the effectiveness of learning, we have tried the strategy of decreasing the exploration level as the learning progresses, with values of maximal and minimal epsilon 0.1 and 0.8. We have tried 20 different values of n^*: from 1 to 40000.

- **Question 2** The results of average rewards, M_{opt}, and M_{rand} for the 5 values (for clarity) are shown in Figure 2. The best result was shown by the algorithm with n^*=16000. When comparing the averages, we can conclude that the technique of reducing the epsilon value improves the learning efficiency of the algorithm. As n^* increased, the growth rate changed: the smaller the value, the faster the function plateaued. Thus, after 16000 within 20000 episodes, the function does not plateau at all.

Figure 2: Average rewards, M_{opt}, and M_{rand} for different $n*$.

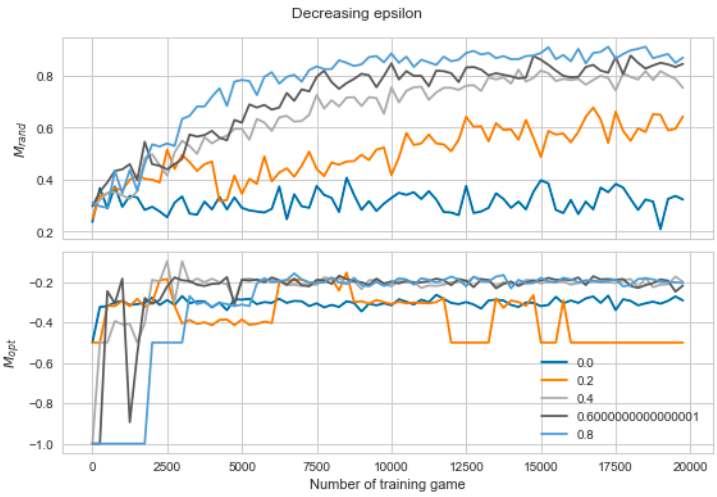

Figure 3: M_{opt}, and M_{rand} for different ϵ.

- **Question 3** In Figure2, we can see that a comparison of M_{opt}/M_{rand} and rewards plots show that the growth dynamics is the same, but M values reach the plateau faster.

C. 2.1.2 Good experts and bad experts

- **Question 4** We observe that training against Opt(0) achieves ok results for M_{opt}, however bad M_{rand} performance. This is probably because training against an optimal player with low epsilon, the learning agent is exposed to a more predictable opponent, furthermore it learns more how to tie(not to lose) then how to win since it rarely gets rewards for winning. Furthermore we observe the learning algorithm achieves better M_{rand} results when he trained against an increasingly random player, while still achieving some of the best M_{opt}, probably due to the fact that playing against a random player also allows the learning algorithm to win and thus learn to win through the rewards and play better against a random player.

- **Question 5** As a result the highest values were $M_{rand} = 0.9, M_{opt} = 0$.
- **Question 6**: The learned Q-values will be different due to the fact that the opponent's parameters affect the values of the state of the environment.
  When playing against an optimal opponent, for every "dangerous" situation encountered by our algorithm he will end up losing, which will train the algorithm to avoid such situations. Furthermore the optimal opponent always plays with the same strategy and this limits the states our learning algorithm is exposed to, thus limiting exploration.
  Meanwhile when playing against a random player the possibility of winning remains, and our algorithm receives different rewards(i.e rewards for winning), which will teach our algorithm to learn how to win and not only how to avoid losing. Furthermore playing against a random player will expose us to a wider range of states that we may learn Q-values for.

- **Conclusion:** Based on our learning observations, we can conclude that our tabular q-learning algorithm can learn when playing against an optimal policy algorithm who sometimes makes random moves with a given epsilon rate. In the end the best results were obtained by using Decreasing exploration with parameter 16000 against an optimal player with epsilon 0.5.

III. Question 2. Can an RL algorithm learn to play Tic Tac Toe only by playing against itself?

A. 2.2 Learning by self-practice

- **Question 7** We now trained our tabular q-learning algorithm against itself rather than against the optimal strategy. We trained with epsilon greedy and fixed epsilon ranging from 0 to 1. We found that the angent does learn to play and epsilon of 0.6 makes it possible to plateau $M_{opt} = 0$ and $M_{rand} = 0.9$. We furthermore observe that lower values of epsilon achieve bad results for both M_{rand} and M_{opt}, meanwhile higher values of epsilon achieve bad results for M_{opt}, however the M_{rand} values show that the agent still learns to perform well against a random player.

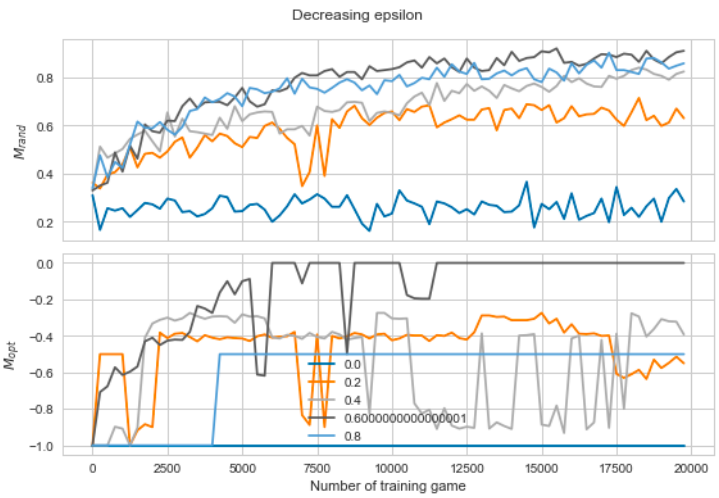

Figure 4: M_{opt}, and M_{rand} for different ϵ.

- **Question 8** When applying Decreasing exploration we got the best results $M_{rand} = 0.9, M_{opt} = -0.2$. The M_{opt} values for $n^* = 16000$ peaked, but M_{rand} was lower at about 0.8. As n^* increases we observe better performance for M_{rand} but M_{opt} peaks at $n^* = 16000$ then begins to degrade and become unstable for higher values. Thus decreasing exploration helps achieve better results in this case then a fixed epsilon, however we have different best values of n^* for M_{rand} and M_{opt}

Figure 5: M_{opt}, and M_{rand} for different n^*.

- **Question 9** The highest values were $M_{rand} = 0.9, M_{opt} = -0.2$.
- **Question 10** When analyzing the heat map of Q-values, the results make sense: the maximum corresponding to the optimal move on the grid for the current state where the agent should play. The other values being 0 or very close to 0. So we can conclude the agent learned to play well.

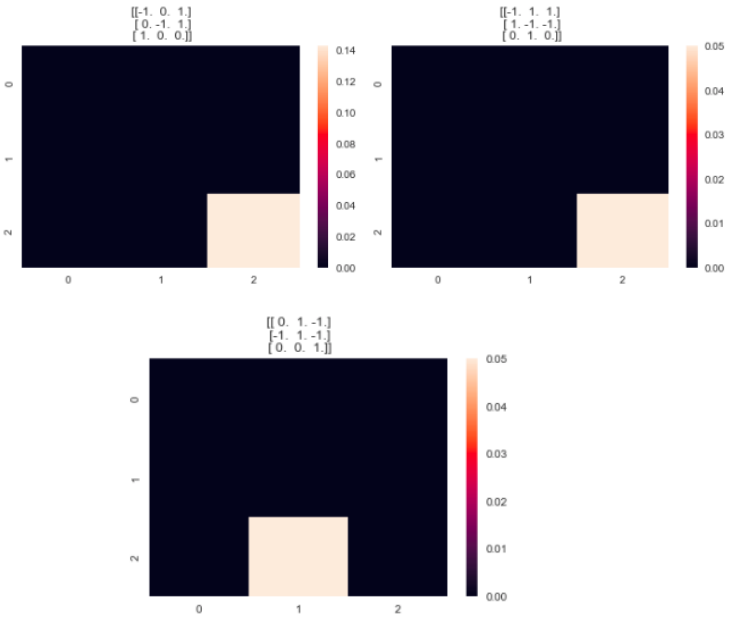

Figure 6: Q-values for 3 different states.

As a result of the experiments, we can conclude that the algorithm trained when playing with itself shows high results on the M_{opt} and M_{rand} metrics.

B. 3.2 Learning from experts

- **Question 11** The reward graph shows that the algorithm learned during the first 2500 games to the level of 0.5 , while the loss was growing.

After reward reached a plateau, loss decrease to 0.03 and maintained that value as well. Thi suggests that the algorithm is learning to pla tic-tac-toe

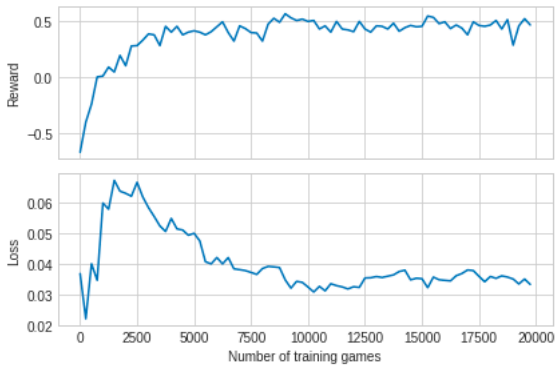

Figure 7: Average reward and training loss.

- **Question 12** Decreasing the batch size to 1 makes the behavior of the graphs more stable - reward increases to 0.5, while loss constantly decreases down to 0.03.

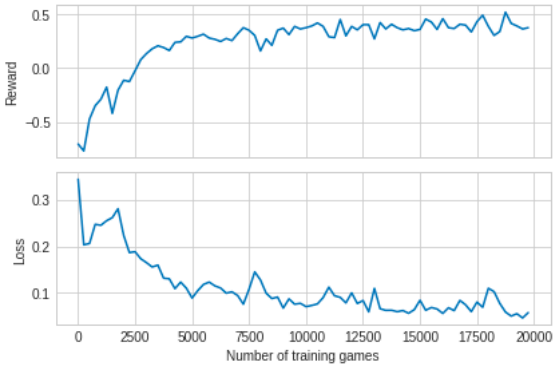

Figure 8: Average reward and training loss for batch-size=1.

- **Question 13** We notice that n^* = 1, which essentially gives us a fixed epsilon of 0.1, achieves the slightly better results for M_{rand} and better and more stable results for M_{opt} compared to other values of n^*. Thus decreasing exploration does not provide provide a big advantage in this case. We notice that that smaller value of n^* gives the best results - the maximum for $n^* = 1$, $M_{rand} = 0.96$ and $M_{opt} = 0$. As n^* increases, the average M_{rand} decreases, and M_{opt} is affected to a greater extent, up to -0.5. and also more unstable.

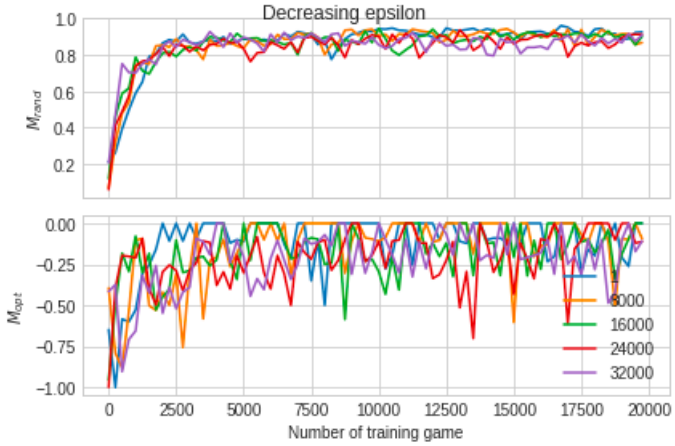

Figure 9: M_{opt}, and M_{rand} for different n^*.

- **Question 14** For different epsilon, at a minimum value of 0 the worst M_{rand} performance of 0.2, but the best M_{opt} of 0. At the highest epsilon value of 0.8 we get best M_{rand} performance close to 1, but the worst and most unstable M_{opt}. This may be due to the fact that if the algorithm trains against an optimal player with epsilon 0, it learns to tie very well and plays effectively against optimal opponen, who plays in a predicable fashion. However, such an algorithm shows low M_{rand} metrics because it has learned only the strict patterns that the optimal algorithm uses, and doesnt learn to win since it cannot win and thus never get the rewards for winning. Overall we observe that training against an optimal player with epsilon of around 0.4-0.6 is a good compromise as it achieves good and more stable results and faster training, when considering both M_{rand} and M_{opt}.

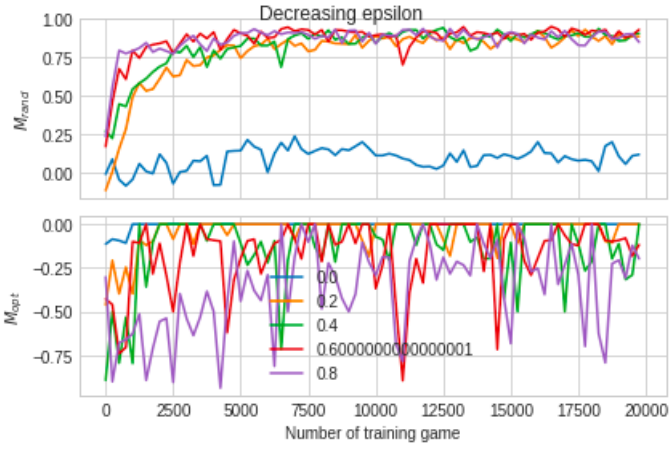

Figure 10: M_{opt}, and M_{rand} for different ϵ.

- **Question 15** The highest values were $M_{rand} = 0.96, M_{opt} = 0$.

C. 3.3 Learning by self-practice

- **Question 16** We can see that the algorithm learns to play tic-tac-toe and reaches values 0 and 0.9 for M_{opt} and M_{rand} respectively. For M_{rand} all values of epsilon achieve good results however when looking at M_{opt} too high or low values of epsilon lead to more unstable training.

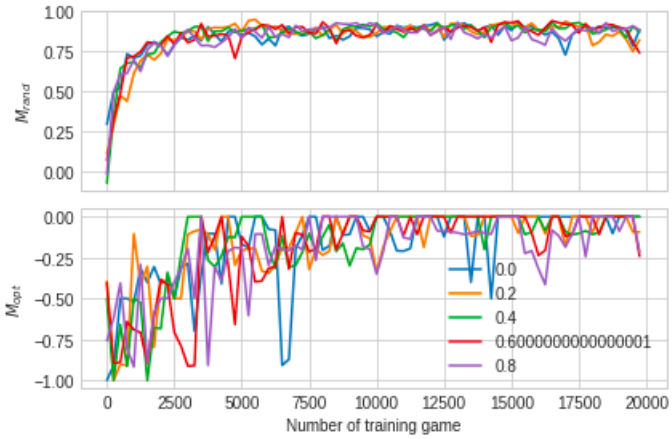

Figure 11: M_{opt}, and M_{rand} for different ϵ.

- **Question 17** We notice that $n^* = 1$, which essentially gives us a fixed epsilon of 0.1, ultimaitely achieves similar M_{rand} and M_{opt} performance, however having a n^* of 20000 or 3000 train faster for M_{rand}.

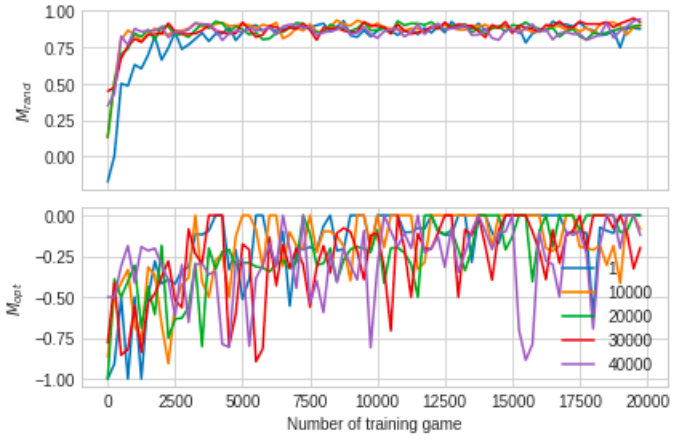

Figure 12: M_{opt}, and M_{rand} for different n^*.

- **Question 18** The highest values were $M_{rand} = 0.947, M_{opt} = 0$.
- **Question 19** For the experiment we took the same states as for Question 10. As a result, we obtained the same optimal solutions, but the degree of certainty of other options increased, which introduces doubts in the proposed solution. However, the proposed solutions(i.e the position with the highest Q-value) is indeed the best move, and prove that the algorithm learns.

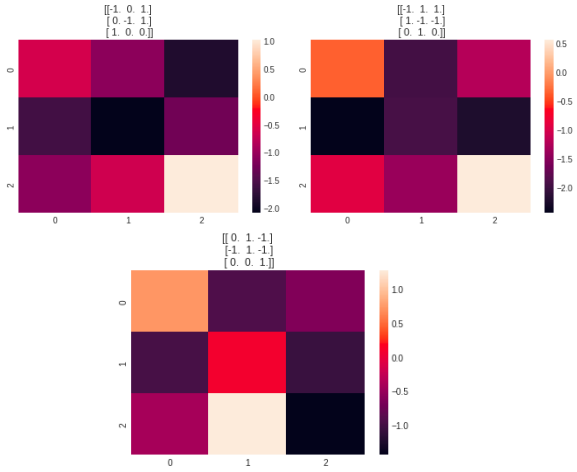

Figure 13: Q-values for 3 different states.

D. Comparing Q-Learning with Deep Q-Learning

- **Question 20**

Method	M_{opt}	M_{rand}	Time
Q-learning from experts	0	0.87	9000
Deep Q-Learning from experts	0	0.97	1400
Q-learning by self-practice	-0.2	0.9	6000
Deep Q-Learning by self-practice	0	0.952	1200

Table I: The best performance of Deep Q-Learning and Q-learning algorithms.

- **Question 21** As a result of the comparison, we can conclude that the DQN method showed the best results: when applied, we got the fastest and closest to the best option. However, the self-practice strategy slightly reduced the results within a few percent, but compared to the Q-Learning values the result remains both significantly faster and more efficient. For the Q-Learning strategy, the self-practice method was more efficient.
We can see from the graphs that the advantage of deepQ-learning is fast convergence and high efficiency. However, the disadvantage compared to Q-Learning demonstrated in heatmaps is the overestimation of Q-values.