Learning TicTacToe using Q-learning

Samuele Milanese s3725294 **Levente Foldesi s3980456**

Abstract

In this report we describe how we implemented a Q-Learning algorithm that learns how to play TicTacToe and maximizes performance. The Q-learning algorithm was combined with different exploration methods. In order to evaluate the efficiency of such learning agents, we let them play and learn against a random agent and each other, then compared their performances in terms of win ratios (wins over number of games played)

1 Introduction

For our final project we decided to look into TicTacToe. This is a really popular and well known game but we will give a brief introduction to the rules. There are two players who play the game. The board which the game is played on is a 3x3 grid (in our implementation the size can vary). Each player chooses one square to put either an 'X' (player 1) or an 'O' (player 2). The essence of TicTacToe is to have three of the same signs ('X' or 'O') next to each other horizontally, vertically or diagonally. The first player to achieve this wins.

In our experiment, we implemented two agents that play the game with different strategies: Q-learning and Random. Random agent was used as baseline for comparison and learning of other agents and chooses a random action. On the other hand, Q-learning tries to learn the best strategy throughout the games. We combined Q learning with different exploration strategies from assignment one. The implementation contains: Greedy, Epsilon-Greedy, Upper Confidence Bound and Optimal initialization. We will explain them in more details in the following sections.

2 Literature review

Literature on the solution of the problem of TicTacToe through reinforcement learning methods was rather scarce. We could not find any authoritative paper that could act as main point of reference for this project.
However, the game of TicTacToe is fairly simple. For this reason the framework could be implemented from scratch while the algorithms we needed could be adapted in a straightforward manner as presented in the book by Sutton and Barto (1998) and the Lecture Slides by Sabatelli (2021). Furthermore, we used the blog by Ostermiller (2004) as a source to motivate the initial optimization of an agent as it will be described in the next section.

3 Method

3.1 General setup

The setup of the python project consists of three files: **tictactoe.py**, **agent.py** and **game.py**. The **tictactoe.py** contains the main function, this is the controller part of the algorithm. The initialization of the agents, the hyperparameters and the plots are implemented here (the last one is used for evaluating the performance of each agent). The settings for the experiments (with their relative parameters) are also written and executed here. **game.py** is responsible for the actual game of

TicTacToe. In this class, the players make their steps while there is a step to make (so it is not a draw) or either one of the players win. The winner is decided by checking the diagonal, vertical and horizontal lines after each turn. Finally, in the **agent.py** class, the different types of agents are implemented.

3.2 Learning strategies

In our implementation of TicTacToe, the game is only played amongst agents that start with no knowledge of the game. The agents learn through experience the best moves to take in each configuration the board can be in.
In order to implement this idea, as with any other reinforcement learning problem, we had to frame the game of TicTacToe as a Markov Decision process (MDP) (Sabatelli, 2021). In other words, it is necessary to find a representation for the states the game could be in, the actions that can be taken, a transition function and a reward function. Although in many similar problems states are represented by the actual configuration of the board, grid or such, we found it more fitting to represent the states of the game as the set of moves that could still be made in a specific configuration. This gave us many advantages at an implementation level as states were more compact and easy to generate and actions could be selected directly from the state. It follows naturally that actions were represented as tuples of coordinates of the grid where the next mark could be set (ideally, either a X or a O, although this kind of representation is completely absent in the implementation). Figure 1 shows a graph that describes how we implemented the Q-table. It was implemented as a nested dictionary where the states are the key set of the outer dictionary. The values of the latter are the actions which are the actual tuples of coordinates (used by the framework code) associated with their action value.

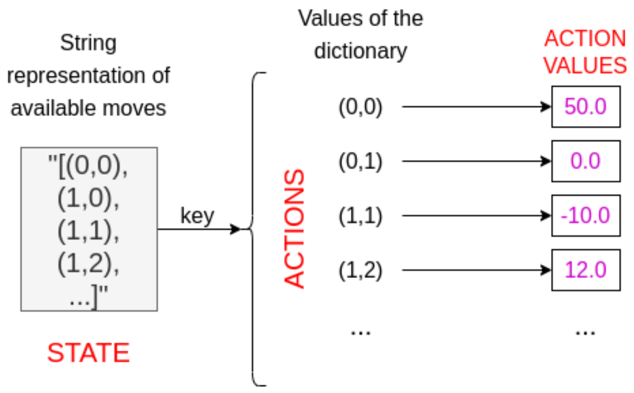

Figure 1: Graph of Q-table representation in the code

The transition function was implemented in terms of an exploration algorithm that picked the best action (or not) in a certain configuration.

Lastly, the reward function was rather simple, returning either 100 or -100 for a winning (or losing) move. Every other move was rewarded with 0, including final moves that ended the game in a draw. The reward was thought out specifically for strategies such as Q-learning, since values are computed in terms of future values, the rewards sort of spreads so that there is no need to reward positively (or negatively) intermediate steps.

Of course the MDP frame was completely overlooked by our implementation of the random agent. This agent is, in fact, the most naive implementation of a TicTacToe player possible. All it does is choose randomly from the sets of possible moves. The reason this type of agent was implemented to begin with is because we used it as a sort of control group to evaluate the performances (and learning) of the other "smarter" agents.
These "smarter" agents were implemented following the Q-Learning algorithm. We found this algorithm to be the best fit for a game like TicTacToe because it does not only evaluate the state, but the action as well. Furthermore, being an adversarial game, the look-ahead step is available, but it would have been trickier to include also the following action, making algorithms such as SARSA not

an option.
As mentioned in the introduction, we implemented the Q-Learning algorithm following Sabatelli (2021) and Sutton & Barto (1998). Namely, the formula we implemented is the following:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma * \max_{a \in A} Q(s_{t+1}, a_t) - Q(s_t, a_t)]$$

Our agents keep a Q-table that maps from configurations to actions to values for the state-action pairs, as described before and by figure 1. These state-actions pairs are then updated at each move. As mentioned earlier, as values are computed in terms of the value of the successive move, rewards for terminal states propagates so that the actions that led to the winning move become more desirable. The Q-table then informs the agent on which is the best action to take in each state. A Greedy agent would always pick the best possible action in each state. However, this was not the case for all the agents in our implementation. In fact, while all the agents (except for the random one) use Q-learning, they differ in the action selection procedure which is going to be explained in more detail in the next section.

3.3 Exploration methods

As it was just stated, our implementation presents different classes of Q-Learning agents which differ in the way the action selection is carried out. Namely, we implemented four different exploration methods that present different balances between exploration and exploitation, in order to compare what is the best exploration strategy to follow to learn the game of TicTacToe. Similarly to the Q-learning algorithm, these exploration methods were adapted directly from the book by Sutton & Barto (1998).

3.3.1 Greedy

The Greedy exploration method is the most basic one. Essentially, it does only one thing, always selects the best possible action in set of available moves. Action selection according to a Greedy strategy can be described formally as:

$$A_t = argmax_{a \in A}(Q_t(s, a))$$

3.3.2 Epsilon-Greedy

The ϵ-Greedy algorithm is quite similar to the basic Greedy except it does not always pick the best move. It is, in fact the first algorithm that involves some exploration. It achieves this by adding some randomness. In our implementation, a (pseudo)random number is sampled, if this is higher than the fixed ϵ then we exploit (so with probability $1 - \epsilon$), otherwise the agent will pick a completely random action, from those which are available, effectively achieving exploration.
This can be described formally as:

$$\pi_t(a) = \begin{cases} argmax_{a \in A}(Q_t(s, a)) & \text{if} \quad (1 - \epsilon) + \epsilon/|A \\ \text{random} & \text{if} \quad \epsilon/|A| \end{cases}$$

3.3.3 Optimal initialization

The optimal initialization method is based on choosing the top left corner as its first move every time, since Ostermiller (2004, June 15) found that it gives the highest chance for winning. The way it was implemented, is that a large reward is hard-coded for that particular first move so the agent will find it desirable at each starting point of the game. The whole idea of Optimal initialization is an alteration of Optimistic initial values strategy from the Sutton & Barto (1998) book, to make it more suitable for TicTacToe.

3.3.4 Upper Confidence bound

The Upper Confidence bound exploration is based on the following equation ((Sutton & Barto (1998)):

$$a_t = argmax[Q_t(a) + c\sqrt{\frac{ln(t)}{N(t)}}]$$

Here, $Q(a)$, is the estimated value for action a, c is a constant which controls the degree of exploration, t is the current time step and $N(t)$ is the number of times, an action has been chosen before t. The $Q(a)$ part represents the exploitation, as throughout the learning, it will have higher an higher estimated value therefore, the exploration of the agent will be decreased as the agent learns. $c\sqrt{\frac{ln(t)}{N(t)}}$ is responsible for exploration and since $N(t)$ is going to be larger over time, similarly as before, exploitation is going to be preferred.

3.4 Hyperparameters

For tuning the hyperparameters, we went through each agent and checked for the maximum percentage regarding winning. First, we started with the number of epochs and repetitions. In our implementation, epochs are the number of games played by the agents before they are reset, such that for every successive game, the agents learn a bit more about the game. Repetitions are the number of times the experiment (a cycle of N epochs or games) is ran over and over again. The numbers were obtained keeping in mind that the agents need "time" to learn hence the number of epochs was high enough (175) for the agent to show a clear learning curve when plotting the percentage of wins. On the other hand, the repetition number was increased to 1500 only to minimize the effects of randomness and to obtain a more accurate average of wins per game (or, per epoch).
Moving forward to the agents themselves, in each case, a list of possible hyperparameters were given to the agents and, as briefly mentioned before, the hyperparameters contributing to the highest winning percentage were chosen. Regarding the alpha, gamma and epsilon, the following list of values were tried: 0.1, 0.2, 0.4, 0.5, 0.6, 0.8, 1. For Greedy, the best combination was 1 and 0.6 (alpha and gamma respectively) which essentially makes sense, as 1 means the highest learning rate possible, and since there are not too many "strategies" to learn, a rapid learning can be beneficial. Understandably, the same values were used with ϵ-Greedy. The ϵ was found to be 0.1, which maximized the winning ratio. The intuition behind it is that with a low ϵ there will be more exploring which is advantages in the beginning as the agent needs to learn. (One possible development for future studies could be an ϵ-Greedy algorithm where the ϵ is continually increased as more the agent knows the less exploration is needed).
Optimal initialization agent uses the same hyperparameters as the Greedy algorithms though, we acquired different results. The best combination was 0.4 and 0.5 for alpha and gamma respectively. Since the first steps of the optimal agent is pre-coded (as explained before), it is straightforward that less learning is needed hence, the α is significantly lower. γ is not notably different so the discounting factor stays the same approximately.
Upper Confidence bound (UCB) had the same α and γ as it was found with the Greedy agents. So, the c hyperparameter had to be adjusted. The following values were tried for c: 1, 2, 3, 4, 5. The optimal was 4 however, there were only a few percentage difference between the different values. This could be explained with that in UCB the degree of exploration (at least with a small state space like in TicTacToe) does not play a momentous role.
As a final remark, we intended to test the algorithm with larger grid sizes as well but as it turned out, in that case, the vast majority of the games are ended in a draw, so we decided not to include it in our experiment. The reason behind this could be that larger grid size, it is harder to win, so more games will end in a draw

4 Discussion

In this section we are going to talk about our findings during the experiment. Figure 2 shows all the exploration methods against the control group (random agent). It seems that Greedy and the Optimal initialization strategy works the best. The reason for this is since there are not too many option to explore, being Greedy, thus choosing the maximum state, is the most advantageous strategy. Regarding Optimal initialization, it is known from Ostermiller (2004, June 15) that the best starting choice is the corner, hence it is expected that this strategy will be powerful.

We decided to plot each of the exploration methods against each other 2 by 2, so one could see the performance of each strategies not just with a dummy agent (random), but with more complex agents as well.

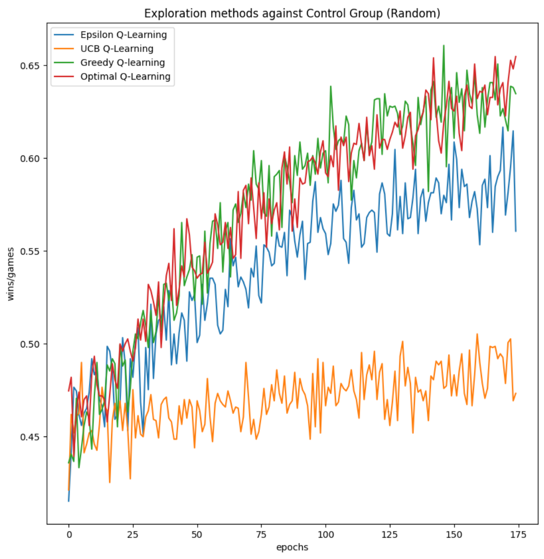

Figure 2: All the exploration methods against the control group).

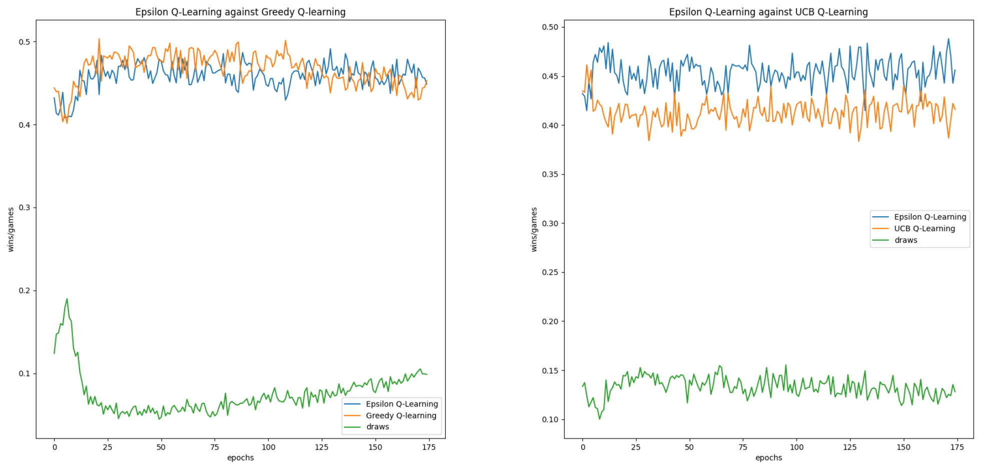

Figure 3: ϵ-Greedy against Greedy and UCB

In figure 3 we plotted the ratio of wins for ϵ-Greedy and Greedy or UCB when playing against each other respectively. As we can see the performance for the two variations of the Greedy algorithm are pretty much the same. The fact that in the beginning we have more draws shows that the two algorithms quickly learn the basic strategy to win the game, so that, in later epochs, who starts first wins. Hence, they almost perform the same way as the starting algorithm is randomized.
We have very different results from the games between UCB and ϵ-Greedy (figure 3). ϵ-Greedy, in fact clearly outperforms the UCB exploration strategy. This is consistent with the findings of the first experiment (all algorithms against the control agent) as it reflects the reduced efficacy of UCB. The reason why the Upper-Confidence-Bound strategy might not be the best for learning a game of TicTacToe is that TicTacToes is essentially not a very complex game to learn. In fact, there are very

specific ways to win or draw (e.g. start in the corner) and the configurations of the board are not so numerous either. In other words, there are not many local optima where an algorithm could "get stuck". On the opposite, finding the good strategy and exploiting that turns out to be the best bet. This explains why Greedy algorithms generally perform better for this game over more exploration-prone algorithms such as UCB.

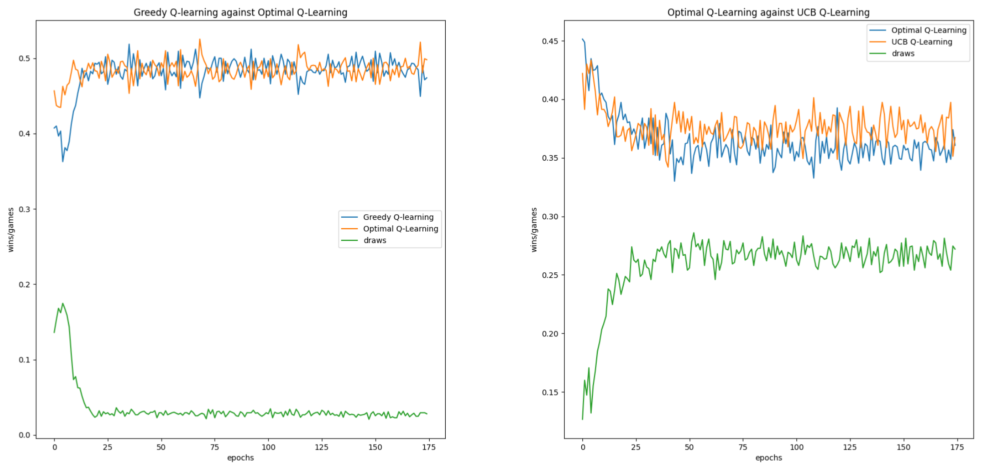

Figure 4: Greedy against optimal initialization and UCB

In figure 4, the Optimal agent against the Greedy and UCB is plotted. First, considering Greedy (on the left side), one can observe that in the very beginning of the game Optimal initialization works better as Greedy still need to learn the optimal strategy and towards the end of the experiment, Greedy catches up. The win/lose ratio is roughly 50% for both of them and the number of draws are under 5%. The reason behind this could be that it is almost always the case that the starting agent wins.
Now moving on to UCB against Optimal agent (on the right side), one can observe a different behaviour than before. Here, it seems that UCB learns how to "fight" against starting the game in the corner and brings those games to a draw. So that is why UCB outperforms Optimal initialization (just by a little) by the end as it can win it own starts but learns to draw the cases when Optimal initialization starts.

Figure 5 shows the performance of Optimal initialization against ϵ-Greedy (right side) and UCB against Greedy (left side). First, considering UCB it is still consistent with our previous findings that Greedy performs better (as it can be seen in Figure 2). The main reason behind it, is similar to the aforementioned reasons. TicTacToe has very little state space hence and the best strategy is to be Greedy.
Moving to the ϵ-Greedy against Optimal initialization, one can obverse similar behaviour as Optimal initialization against Greedy. At the start of the experiment, Optimal initialization over performers the Greedy strategy as learning has not taken place yet. Then after a few receptions, ϵ-Greedy finds a good strategy and the win/lose ratio is around 45% for both agents.

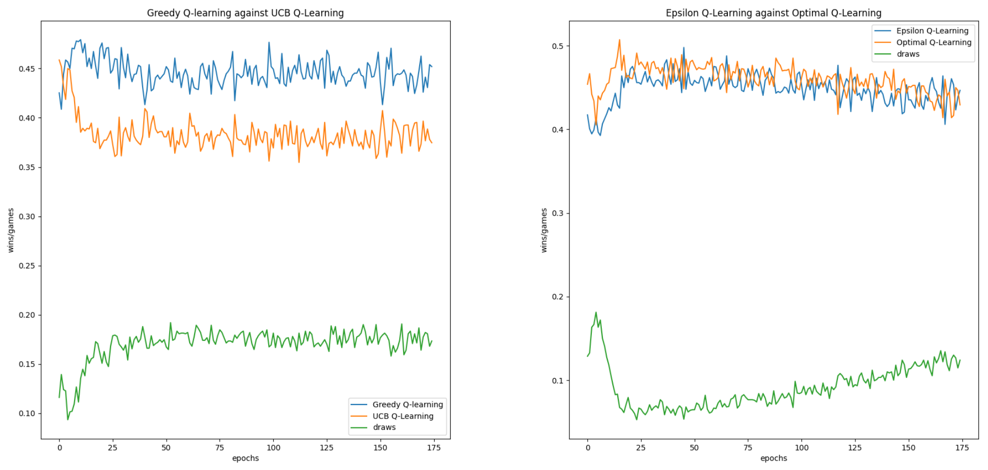

Figure 5: UCB against Greedy and Optimal initialization against ϵ-Greedy

5 Conclusion and further research

In conclusion, although we went through a deeper analysis of the exploration algorithms, figure 2 accurately summarizes our findings on the problem of TicTacToe. In short, for such a small problem, exploitation is sensitively more important than exploration.
We experimented with a rather simple reinforcement learning problem. However, this allowed us to flesh out the algorithms thoroughly and to get important insights about the workings of exploration methods and policy evaluation.
This project is, of course, very much open to further additions. Clearly many more reinforcement learning methods could be implemented as well as exploration strategies. The framework could be enhanced further with options to play against the agents we trained. However, we deemed this to be somewhat beyond our scope.
In the end we reckon that, for the sake of this project, the results we came up with, give satisfactory results as the plot show how all agents become better and better at playing the game.

6 References

[1] Sabatelli, M. (2021). Model-Free Reinforcement Learning—Learning Optimal Value Functions. Reinforcement Learning Practical Lecture, University of Groningen.

[2] Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction. MIT Press.

[3] Ostermiller, S. (2004, June 15). Tic-tac-toe strategy. Stephen Ostermiller. https://blog.ostermiller.org/tic-tac-toe-strategy/