École Polytechnique Fédérale de Lausanne

**Artificial Neural Network**

Professor Wulfram Gerstner

# Mini-Project 1: Tic Tac Toe



| **Students** | **Sciper** |
|---|---|
| Gabriel Bessette | 283364 |
| Xavier Palle | 272991 |



June 6, 2022

# Table of Contents

# 1 Q-Learning

## 1.1 Learning from experts

### 1.1.1 Question 1

Setting an epsilon of 0, the agent's average reward increases over the number of epochs and stabilizes around 0.5. The agent therefore learns how to play TicTacToe since it is losing less and less (positive average reward).

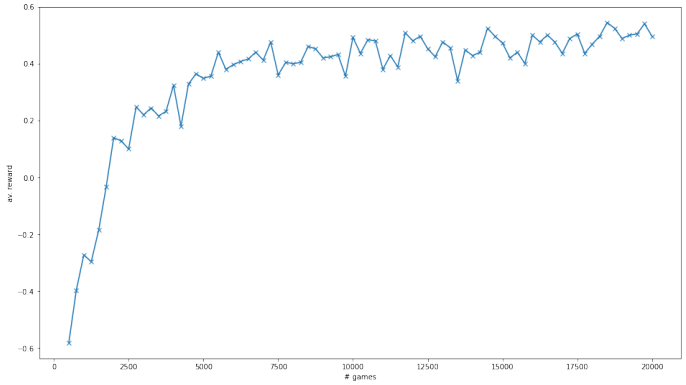


**Figure 1:** *Average Reward ($\epsilon = 0$)*

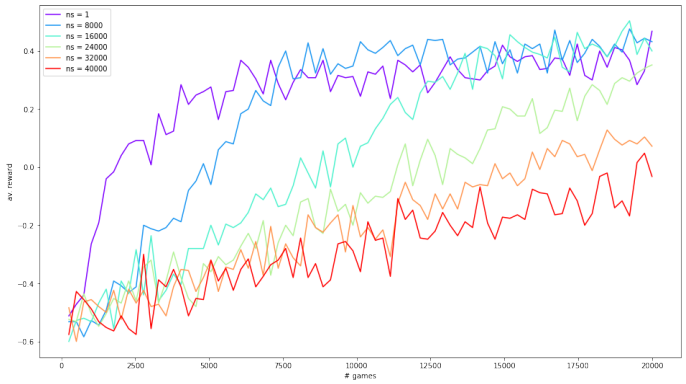


**Figure 2:** *Average Reward for different n**

### 1.1.2 Question 2

Increasing the value of n* should make the agent explore the state/action pair domain more than with a small n*. As one can see from Figure Figure 2, on one side, a high value of n* reduces the rate of learning of the agent (red curve did not reach a steady state, whereas blue curve did), however on the other side, the agent explores more possibilities. Considering a maximum number of 20'000 epochs, the optimal n* would be the one used for the green curve $n^* = 16000$. This way, the agent explores a maximum number of possibilities before reaching a steady average reward against Opt(0.5). Decreasing epsilon over the number of games makes training last longer compared to training with fixed epsilon: for example, here with $n^* = 16'000$, the steady state is reached after 7500 games played versus 5000 games for $\epsilon = 0$ fixed.

### 1.1.3 Question 3

The curves on Figure Figure 3 are similar to the ones of the previous question: exploring the state/action space during training has a cost on the learning rate throughout the number of games resulting in a higher learning rate for small values of n*. The difference appears when computing against Opt(0) and Opt(1). Indeed, the agent has a higher average reward (max 0.7) when competing against Opt(1): the agent wins more often because the optimal player chooses random actions. On the contrary, the agent learns to make draws against Opt(0) as the latter is supposed to always win and reaches a maximum average reward of -0.2.

### 1.1.4 Question 4

From the curves on Figure Figure 4, one can see that when training against an optimal player with a low $\epsilon_{opt}$, the agent is capable of winning more often when playing against the best player Opt(0): for example, at the end of the training against Opt(0), the agent has a performance of -0.2 (purple line), whereas it has a smaller performance of -0.5 at the end of the training against Opt(1) (red line). This can be explained by the fact that the agent updates more often the right Q-values when training against Opt(0) compared to training against Opt(1). From the curves representing Mrand (dotted lines), a similar but surprising result occurs: the agent who trains against Opt($\epsilon_{opt}$) with an $\epsilon_{opt}$ close to 1 shows the best performance (red dotted line). Whereas the agent who trained against the best optimal player Opt(0) shows only poor performances when playing against a player with a random policy Opt(1). This can be explained by a poor exploration of the state/action space during training against Opt(0).

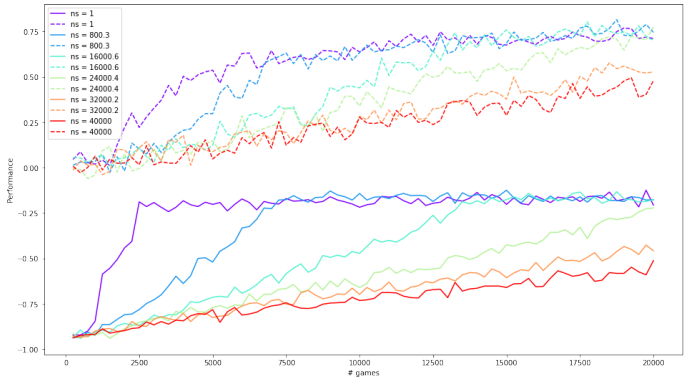


**Figure 3:** *Mopt [-] and Mrand [–] for different values of n**

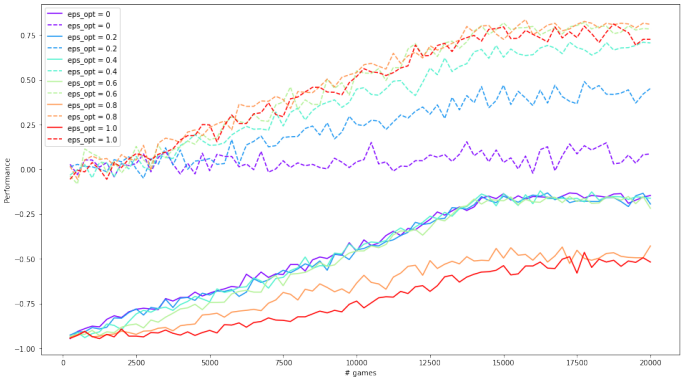


**Figure 4:** *Mopt [-] and Mrand [–] for different values of $\epsilon_{opt}$*

#### 1.1.5 Question 5

The highest value of Mopt is $-0.12$. The highest value of Mrand is 0.83.

#### 1.1.6 Question 6

We have seen that when training against Opt(0), the agent exploits the state/action space (s,a) more often rather than exploring it. Hence for a given state s and action a, the agent will update a given Q(s,a) more often. On the other hand, training against Opt(1) will make the agent explore more Q-values but it will not update them as frequently as when training against Opt(0). Hence, for a given state/action pair (s,a), $Q_1(s,a)$ will be different than $Q_2(s,a)$ since $Q_1(s,a)$ will be updated more often if it is the optimal Q-value.

### 1.2 Learning by self-practice

#### 1.2.1 Question 7

The agent learns how to play Tic Tac Toe since Mopt and Mrand increase throughout the training for all values of epsilon except for epsilon is equal to 1. Note that the best epsilon is 0.2. Also note that when epsilon is equal to zero, then the self training is not performing well: this can be seen on the graph (purple lines) where Mopt and Mrand have a lower value compared to $\epsilon = 0.2$. This can be explained by the fact that once the agent has find a path, it will only update the Q-values of this path and thus, it does not explore the environment enough. Reducing epsilon increases the performance except for $\epsilon = 0$.


**Figure 5:** *Mopt [-] and Mrand [–] for different values of $\epsilon$*


**Figure 6:** *Mopt [-] and Mrand [–] for different values of n**

#### 1.2.2 Question 8

Compared to the previous graph, decreasing epsilon improves self-training giving higher values for both Mopt and Mrand. All configurations seem to make the agent learn. Increasing n* makes the agent explore more. Decreasing n* makes the agent exploit more. There is however a trade-off between exploration and exploitation when computing the performance of the agent against the optimal players. The optimal n* equal to 16'000 gives the best Mopt (0.8) and Mrand ($-0.2$) results for this training.

#### 1.2.3 Question 9

The highest value of Mopt is $-0.22$. The highest value of Mrand is 0.81.

#### 1.2.4 Question 10

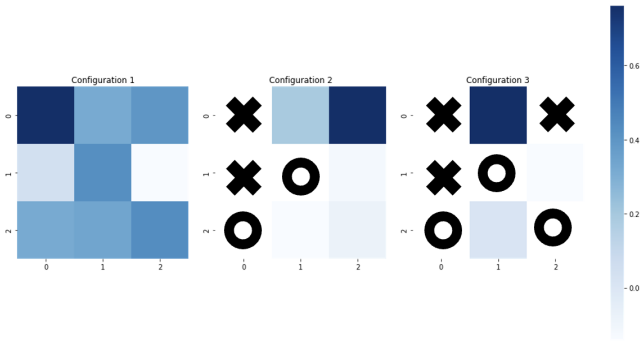


**Figure 7:** *Heatmap for three states*

For the optimal $n^* = 16'000$, and for the three configurations given by the 'X' and 'O' in Figure 7, the agent learned the best action. Indeed, it starts by playing in a corner (left), defends when it needs to (middle) and wins the game on the given occasion (right).

## 2 Deep-Q-Learning

### 2.1 Learning from experts

#### 2.1.1 Question 11

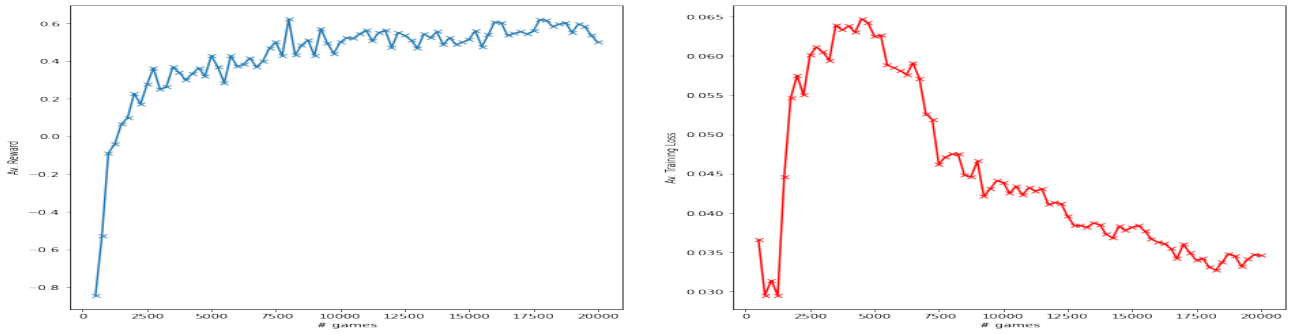


**Figure 8:** *Average reward (left) and average training loss (right) against number of games played for epsilon = 0 and a replay buffer of size 64.*

Setting an epsilon of 0, the DQL's average reward increases over the number of epochs and stabilizes around 0.6. In addition, the average training loss decreases over time until reaching 0.035 at 20'000 games. The agent therefore learns how to play TicTacToe since it is losing less and less (positive average reward and decreasing loss).

### 2.1.2 Question 12

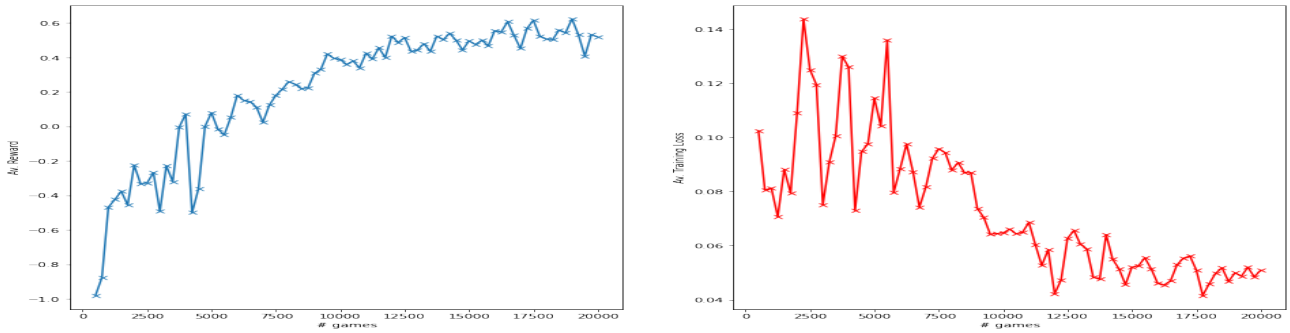


**Figure 9:** *Average reward (left) and average training loss (right) against number of games played for epsilon = 0 and a replay buffer of size 1.*

For a batch size of 1, the behavior of both the average reward and training loss remain similar to the case with a batch size of 64. However, the learning rate seems to be lowered (more time to reach a steady state for both graphs in Figure 9). Indeed, decreasing the batch size results in increasing the training time and decreasing the accuracy.

### 2.1.3 Question 13

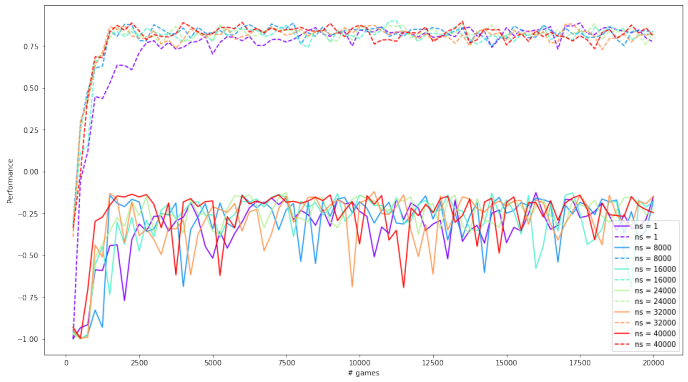


**Figure 10:** *Mopt [-] and Mrand [–] for different values of n**

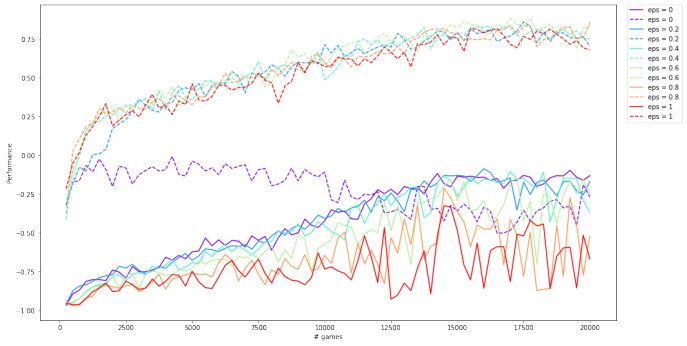


**Figure 11:** *Mopt [-] and Mrand [–] for different values of $\epsilon_{opt}$*

The Figure 10 illustrates that modifying $\epsilon$ over time does not affect much the training of our agent compared to a fixed value of $\epsilon$ ($n^* = 1$). The curves all converge at the same rate to the same values. $M_{opt}$ and $M_{rand}$ end to approximately -0.2 and 0.8 respectively. More precisely, the learning rate for $n^* = 1$ is slightly smaller than all the other ones. Additionally, we observe that $M_{opt}$ has more variance than $M_{rand}$. Hence, $n^*$ has no real effect compared to the standard Q-learning.

### 2.1.4 Question 14

The Figure 11 is similar to the one of Question 4 Figure 4. When training against Opt(0), the agent learns how to play against Opt(0) ($M_{opt}$ has a maximum value of -0.2) but it fails playing against Opt(1) ($M_{rand}$ has a minimum value of -0.5). This can be explained by the fact that when training against Opt(0), the agent *exploits* the state/action space more than it *explores* it. It is the contrary for when the agent trains against Opt(1). For Opt($\epsilon_{opt}$), increasing $\epsilon_{opt}$ leads to a lower $M_{opt}$

at training. This is however not the case for $M_{rand}$: except for $\epsilon_{opt} = 0$, all dotted curves tend to the same value ($M_{rand} = 0.75$). Hence, a value of epsilon higher than zero is sufficient to explore the state/action space and to win against Opt(1).

#### 2.1.5 Question 15

The highest value of Mopt is $-0.09$. The highest value of Mrand is 0.88.

### 2.2 Learning by self-practice

#### 2.2.1 Question 16

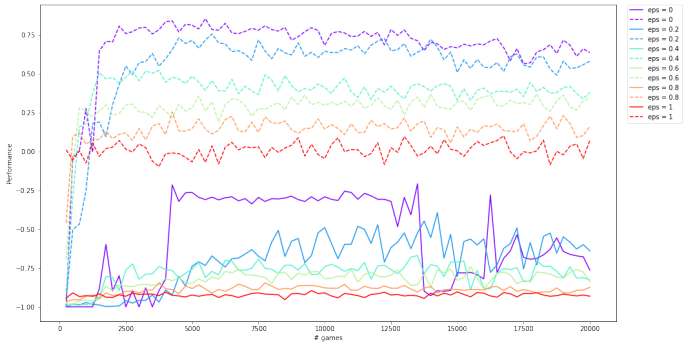


**Figure 12:** *Mopt [-] and Mrand [–] for different values of $\epsilon$*

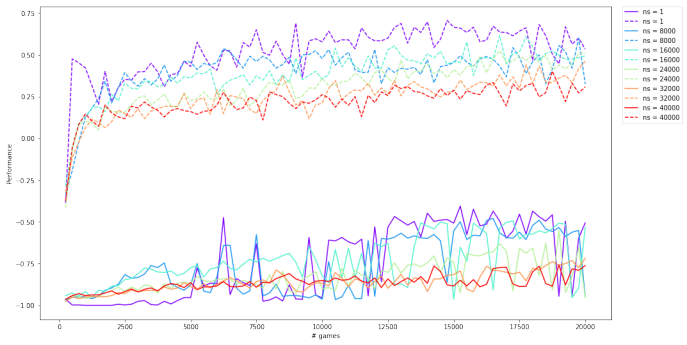


**Figure 13:** *Mopt [-] and Mrand [–] for different values of n**

Mopt graph un peu bizarre: eps= 0 arrive à train mieux que q-learning standard. decroit eps, decreôit performance pour mopt et mrand
From Figure 12, decreasing $\epsilon$ for self learning improves the performance of both $M_{opt}$ and $M_{rand}$. For instance, using $\epsilon = 0$, $M_{opt}$ reaches a maximum value of -0.23 whereas $M_{rand}$ reaches a maximum value of 0.8. The agent seems to struggle to learn how to play TicTacToe against Opt(0) since $M_{opt}$ curves have a lower values than the ones on Figure 5 and show more variance, especially for $\epsilon = 0$. This may be caused by the agent not exploring enough the state/action space.

#### 2.2.2 Question 17

From Figure 13, one can see that having a fixed value of *epsilon* ($n^* = 1$) leads to the best performances for both Mrand and Mop (purple curves in Figure 13). Hence, increasing $n^*$ reduces the performance of the agent.
Figure 13

#### 2.2.3 Question 18

The highest value of Mopt is $-0.41$. The highest value of Mrand is 0.71.

#### 2.2.4 Question 19

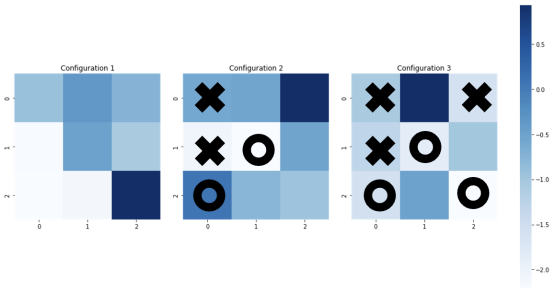


**Figure 14:** *Heatmap for the three states in DQL*

For the optimal "n* = 16'000", and for for the three configurations given by the 'X' and 'O' in Figure 14, the agent learned to play quite well. Indeed, it starts by playing in a corner (left), defends when it needs to (middle) and wins the game on the given occasion (right) as the Q-learner was doing.

# 3 Comparing Q-Learning with Deep Q-Learning

### 3.0.1 Question 20

| | Q vs opt | Q vs Q | DQL vs opt | DQL vs DQL |
|---|---|---|---|---|
| $M_{opt}$ | -0.12 | -0.22 | -0.09 | -0.41 |
| $M_{rand}$ | 0.83 | 0.81 | 0.88 | 0.71 |
| $T_{train}$ | 13'000 | 12'500 | 12'000 | 11'000 (for $M_{rand}$) |

**Table 1:** *Best performances for Q-learning (Q in table) and deep-Q-learning (DQL in table) when playing against the optimal player (opt in table) and against themselves. For the last value, the optimal training time was not readable, we selected only the time for the $M_{rand}$*

### 3.0.2 Question 21

The Table 1 summarizes all the best performances of our learning algorithms. The following points are interesting:

- Playing against the optimal player Opt(0.5) seems for both models to have better results than the self-learning. Indeed both $M_{opt}$ and $M_{rand}$ are smaller as well as the learning time.
- The deep-Q-learning seems to lead to better performances but with a higher variance.

The Q-learning method uses a lookup table and updates the values of Q(s,a) inside. When finding a new path, a new line is appended in the table. The deep-Q-learning has contrarily a fixed number of parameters (the neural network parameters) which are updated. This difference has a direct impact on the computational cost of the algorithms and leads to a faster convergence for the DQL. This is reflected by the lower values of $T_{train}$ when training with deep neural networks.

Learning against the optimal player allows our models to exploit the most interesting paths of the state/action space which makes them more effective. However they tend to explore less the state/action space as only the interesting paths are updated.

The variance of the DQL algorithm seems to come from the stochasticity of the optimization. Indeed, updating our model through batches induces this probabilistic aspect which does no appear in the Q-learning optimization.

To conclude, the Deep Q-Learning algorithm trained against the optimal player Opt(0.5) shows the best performances at TicTacToe (see values highlighted in Table 1).