Artificial Neural Networks - CS-456

Professor: Wulfram Gerstner

Project Report

Tic Tac Toe Project

Guillaume Parchet (283294) - Alessio Verardo (282634)

May 29, 2022

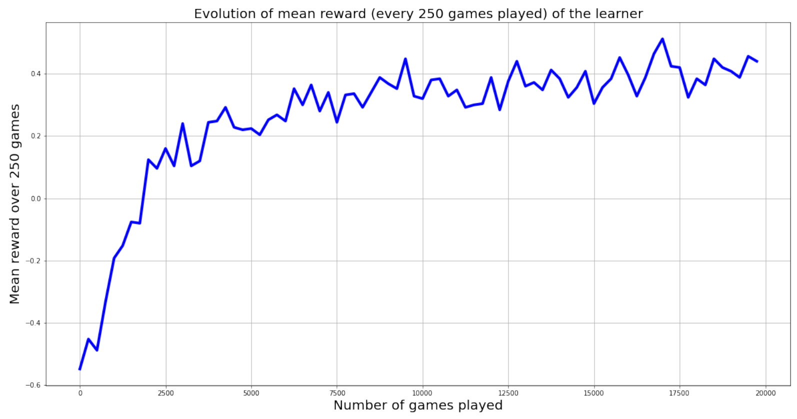
1 Introduction

Important note: in all the questions analyzing test performances against optimal M_{opt} and random M_{rnd} for a range of hyper-parameters, we were strongly advised by the TAs to plot the results into only two sub-graphs. However, in the notebook you may also find individual sub-graph display allowing another perspective for comparison. This perspective is sometimes used in the reasoning answering the questions, do not hesitate to check out the notebook if you wish to better visualize all the perspectives!

2 Q-Learning

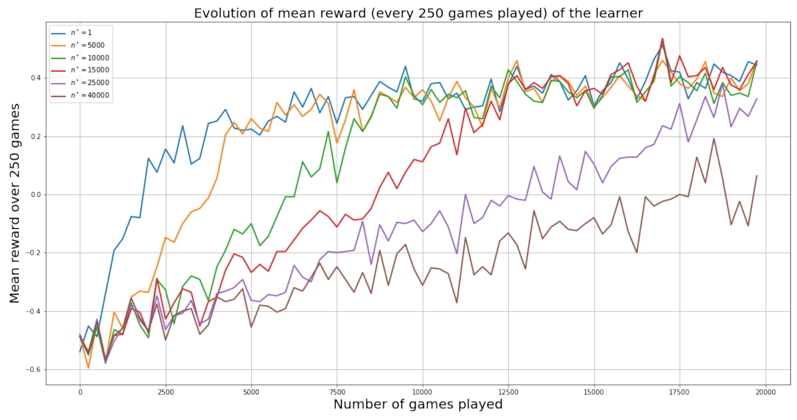
Question 1

With an adversary following Opt(0.5) and a learner with $\epsilon = 0.1$, we get the following average rewards:



From the plot, we see the agent indeed learns how to play Tic Tac Toe as its mean reward has a steep increase and fair consistency across the learning process.

Question 2

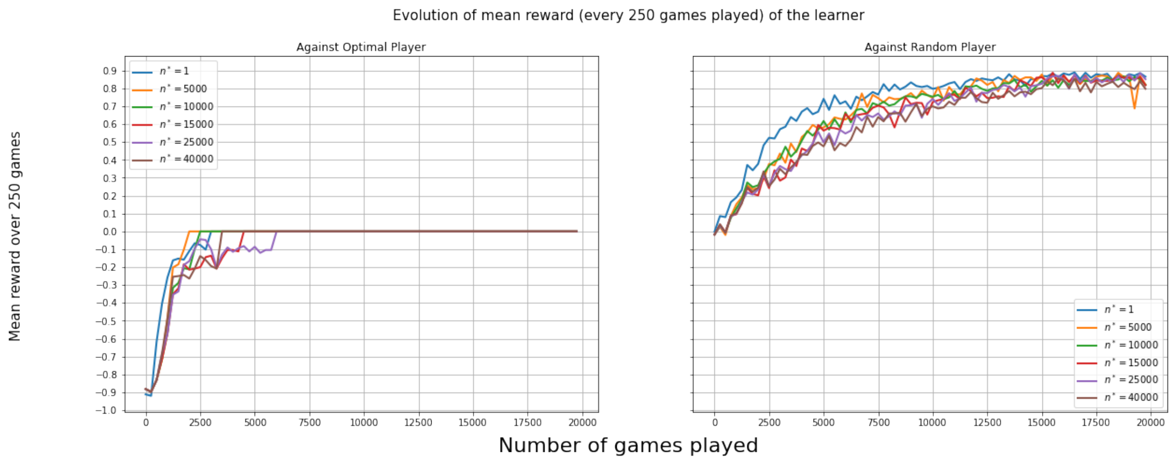


A high n^* makes the random exploratory rate fall off slowly (during most of the training we keep high rate of random actions). A low n^* has the opposite effect, quickly letting the learner only play randomly 10% of the times, i.e. play 90% of the time the optimal move.

As we can see from the figure above, it seems that keeping a low n^* helps to quickly increase the average reward **during training** ($\neq$ test reward). The higher n^*, the longer it seems to take to reach the terminal average reward level (around 0.4). This is intuitive as for high n^*, even if we explore more states, playing randomly a significant proportion of the plays does not help the learner to win when training. To conclude, decreasing ϵ indeed helps learning many Q-values (as it will be confirmed in next question) but decreasing it slowly will make it harder for the learner to achieve great average during the training process.

Lastly, note we only tested on a few n^* values but we could further refine our estimate of the best n^* by running a new search with values close to the current best n^*.

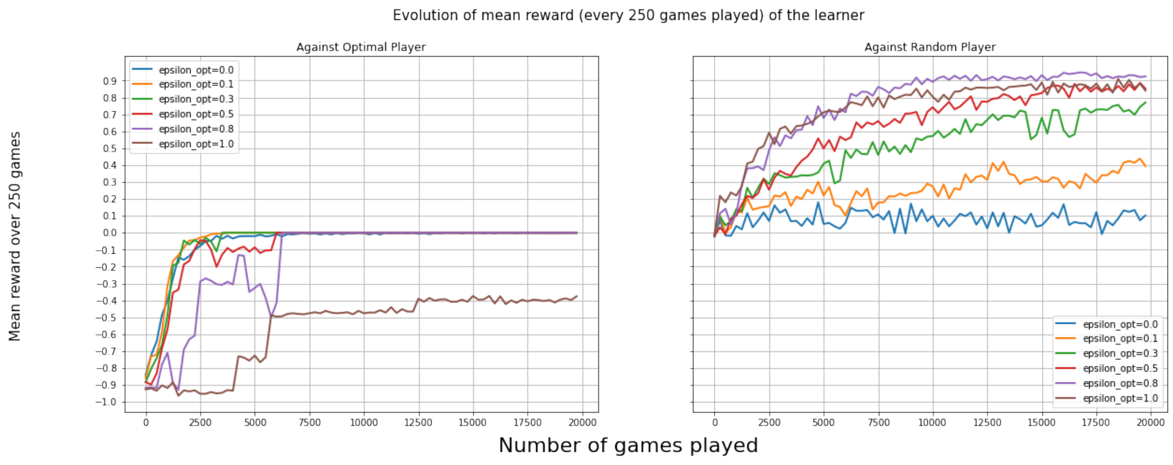
Question 3



We can see similarities with the previous question in the fact that higher n^* take longer to derive the optimal draw policy against optimal player and good moves when playing against random (as n^* increases, the curves have less steep but more constant increase across learning process). However, we can now observe that higher values of n^* also manage to reach the higher terminal performances by the end of the training (for example, with 25'000 and 40'000 which didn't reach the terminal average reward in training do when testing).

Question 4

From the last question, we picked best $n^* = 15000$ as it reaches top performances and allow greater exploration of the Q-values.



We can make the following observations:

- when playing against the optimal policy ($\epsilon = 0$), we see the learner finds very fast systematic ways to achieve a draw. However it struggles to learn how to actually win, with an average reward only around 0.1 against the random player.
- As the proportion of random moves from the adversary increases, we can see it takes increasingly longer to find out systematic ways to achieve a draw (even failing for ϵ=1). However, the learner gains better knowledge of states and Q-values that will allow it to win. The mean average reward against random player increases during the process and eventually reaches slightly above 0.9 win rate against the random player ($\epsilon = 0.8$). It makes sense the average reward keeps increasing as we both train and test against a random player. However one must note that when ϵ_{opt} gets closer to 1, we especially learn how to play against dumb players.
- We noted that by random lucky events during training, the learner sometimes finds an optimal draw policies even for ϵ close to 1. By running with different seeds we saw that indeed it was caused by randomness of the training process and that most of the times it failed to get the policy for high ϵ.

Question 5

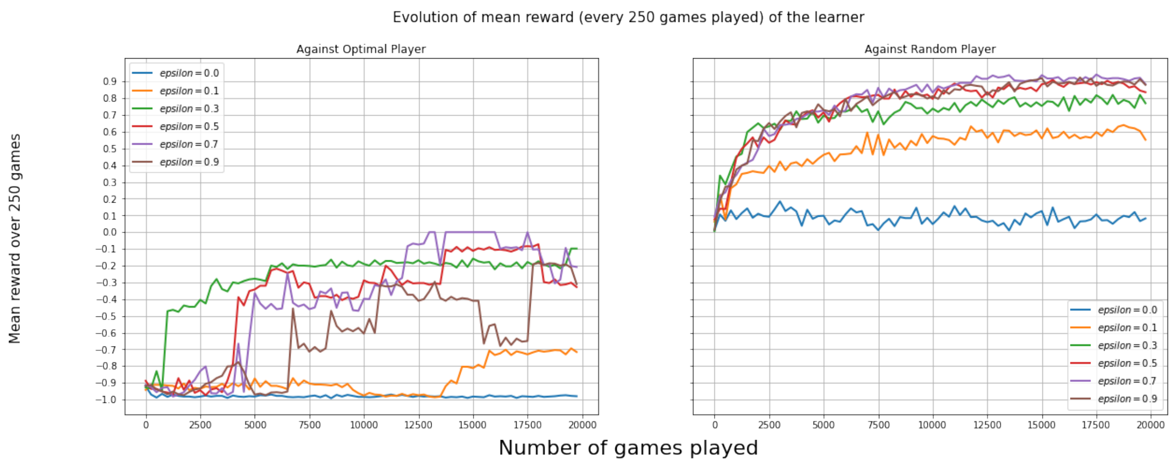
The highest values of M_{opt} and M_{rand} that were achieved after playing 20'000 games were respectively 0.932 (with ϵ_{opt} =0.8) and 0 (for any ϵ_{opt} except 1). We only took into consideration the test values after the training has converged (last 10% iterations).

Question 6

$Q_1(s, a)$ and $Q_2(s, a)$ will not have the same values. Indeed, against the optimal player Opt(0) the learner can at most hope to achieve a draw. It will therefore only see null or negative rewards at the end of its games. Its Q-values will therefore all be null or negative. On the opposite, against the random player, we can expect our learner to win non-negligible number of games, leading to higher Q-values in general, ome of them being positives (e.g. when the learner only has to complete a line to win).

This holds even if we assume the learner aims to learn the optimal playing strategy (which is unlikely against Opt(0) as he may as well learn how to easily achieve a draw).

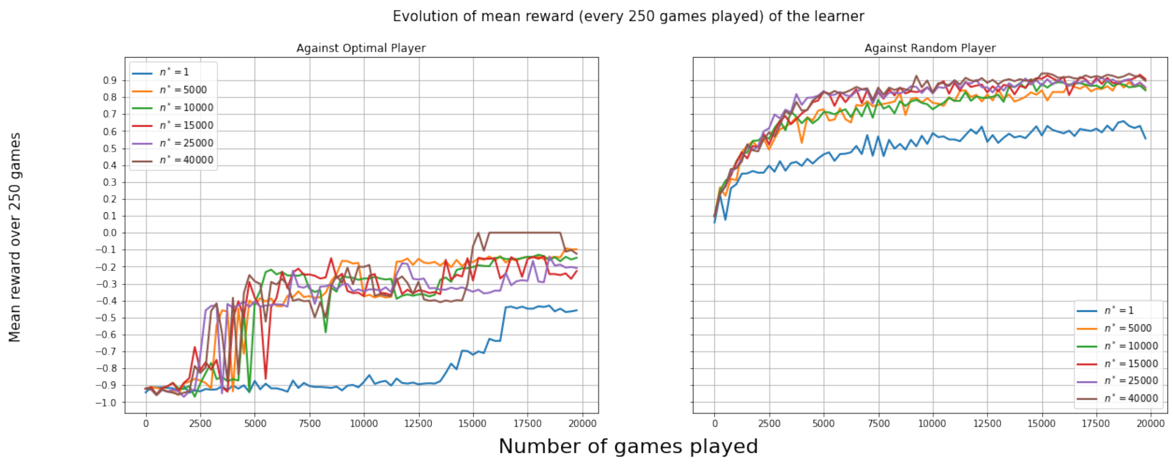
Question 7



The agents indeed seem to learn how to play TicTacToe in all situations except $\epsilon = 0$. When testing, the average reward against optimal and random player seem to increase across learning iterations (eventually stabilizing in the process).

When ϵ is small, the learner do not explore many states. Therefore, the number of Q-values it updates is also small and the learning is pretty poor. With ϵ close to 1, the test performance against random player is high as we learned by playing mostly random. However, it struggles when playing against optimal player (chaotic learning curve).

Question 8

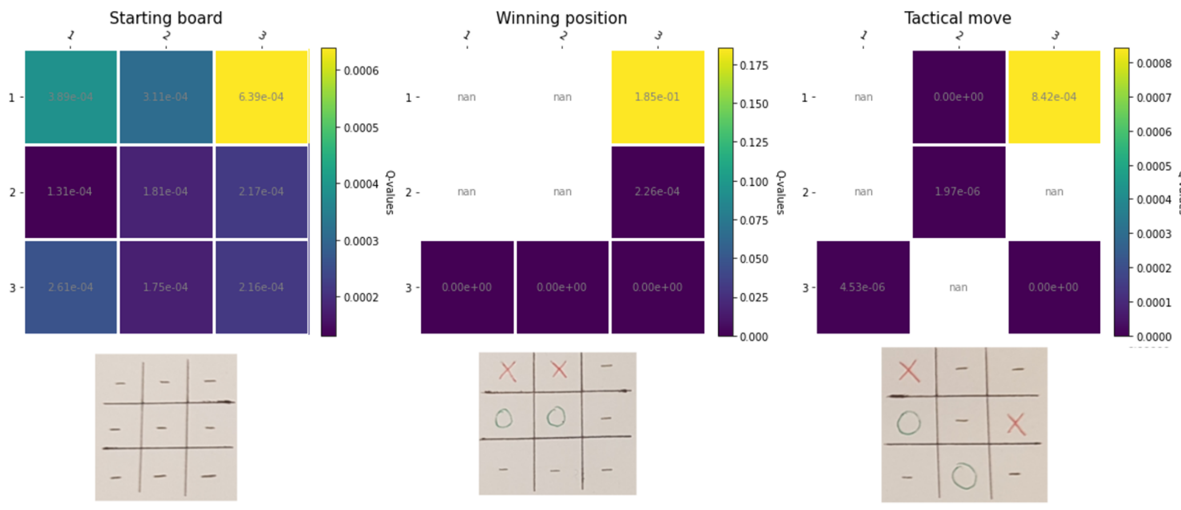


Slowly decreasing the exploratory rate ϵ from 0.8 to 0.1 helps the learners to improve their average test score against the random player. Nevertheless, in self learning it is harder to perform well against the optimal policy as it was never seen during training. Most of the learning seems to come from spikes (when luckily the learners encounter a series of good moves to achieve a draw). Allowing the learners to explore and slowly move towards exploitation seems better than having a fixed ϵ, here the performances are generally slightly better. Again, n^* plays the trade-off role between exploration-exploitation.

Question 9

The highest values of M_{opt} and M_{rand} that were achieved after playing 20'000 games were respectively 0.938 and 0, both with n^* =40'000. Again, we only took into consideration the test values after the training has converged (last 10% iterations).

Question 10



Generally speaking, the results seem coherent (the learner plays 'X'). We have by far the highest Q-value when there is a winning position (board 2), the Q-value for blocking the adversary is higher than the remaining actions. In board 3, it identifies one tactical move (ensuring victory next turn). Finally, there is a preferred starting position in a corner when considering the empty board ; the sub-optimal non-corner positions have lower Q-values.

Yet, the learned Q-values are not perfect. When presented the empty board, in theory we should have some symmetries in the Q-values (starting from any corner is equivalent). Also, the learner fails to recognize the second winning position (bottom right). Indeed, it probably never bothered exploring it and prefer to sharpen its estimation of the first winning position.

These elements allow to see the limitations of our Q-Learning approach. Even though our learners achieve good scoring metrics and general sense in their Q-values, we can see it will usually prioritize one particular move rather than gaining general "intelligence" of all the adapted moves.

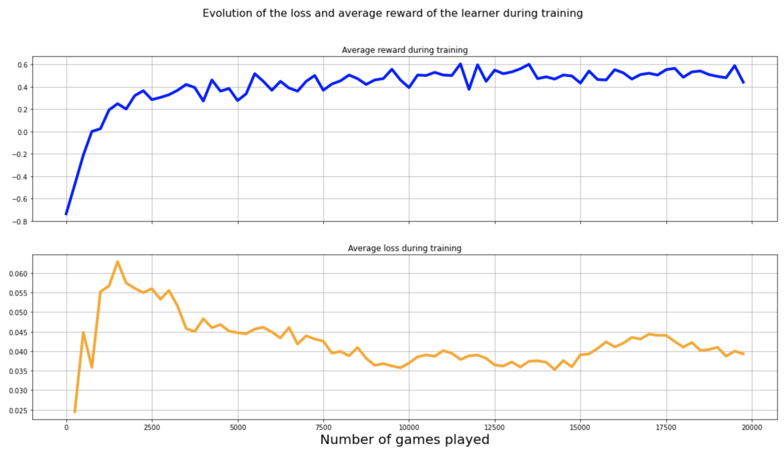
Taking into account board symmetries in the Q-value updates may be a good way to speed up the learning process of the game.

3 Deep Q-Learning

Implementation details

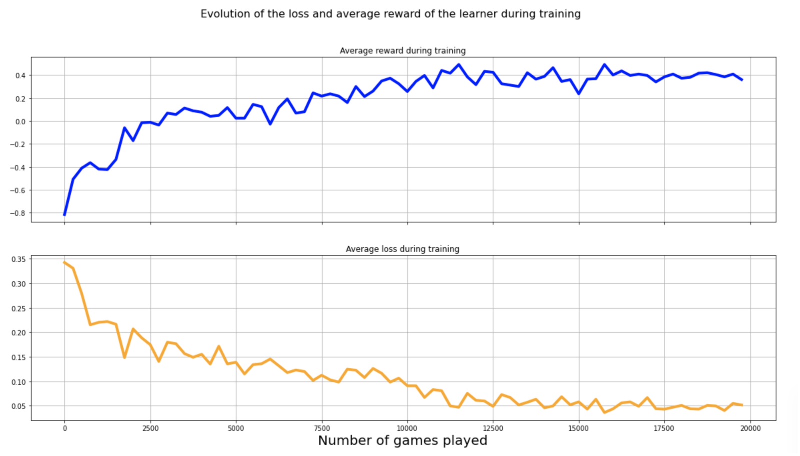
The implementation we made follows closely the PyTorch tutorial on Deep Reinforcement Learning for the sampling from Buffer Memory and update of the policy network weights. The rest is classical deep learning architecture. The main trick to stabilize the training and avoid noisy udpates was to use a learning rate scheduler starting at $5 \cdot 10^{-4}$, decreasing to $5 \cdot 10^{-5}$ after 10'000 games and decreasing to $5 \cdot 10^{-6}$ after 15'000 games. This modification helps us to be much more stable against the optimal player and increased a bit the reward against the random player.

Question 11



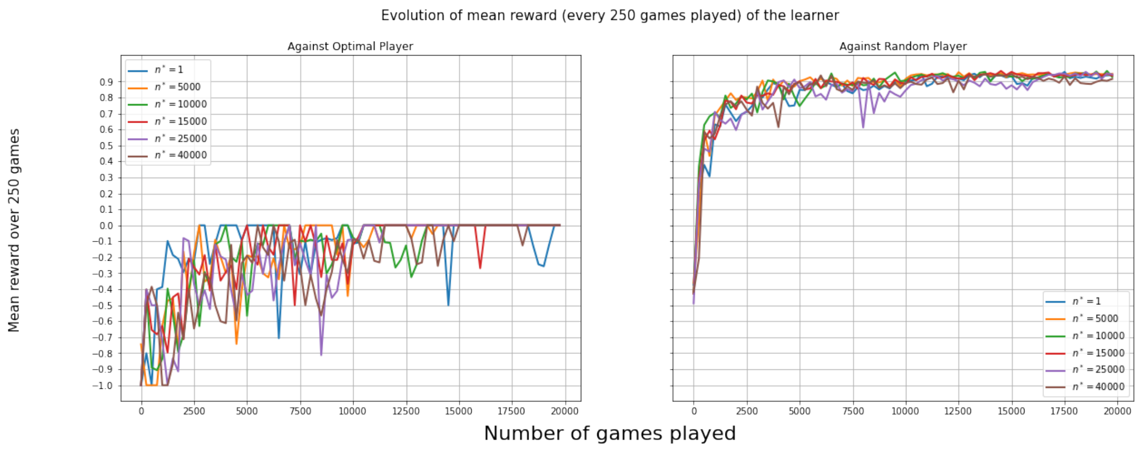
The agent ($\epsilon_{greed} = 0.1$) learns how to play Tic Tac Toe as its mean reward has a steep increase and fair consistency across the learning process. Interestingly, the loss starts by significantly increasing up to 0.065 after 1500 games played. Then, it lowers down and stabilizes around 0.04.

Question 12



The average reward now takes longer to rise and only reaches a level of 0.4 when it was around 0.5 with batches and large replay buffer. The loss starts much higher (around 0.35) and slowly reaches 0.05 (compared to 0.04 with batches).

Question 13

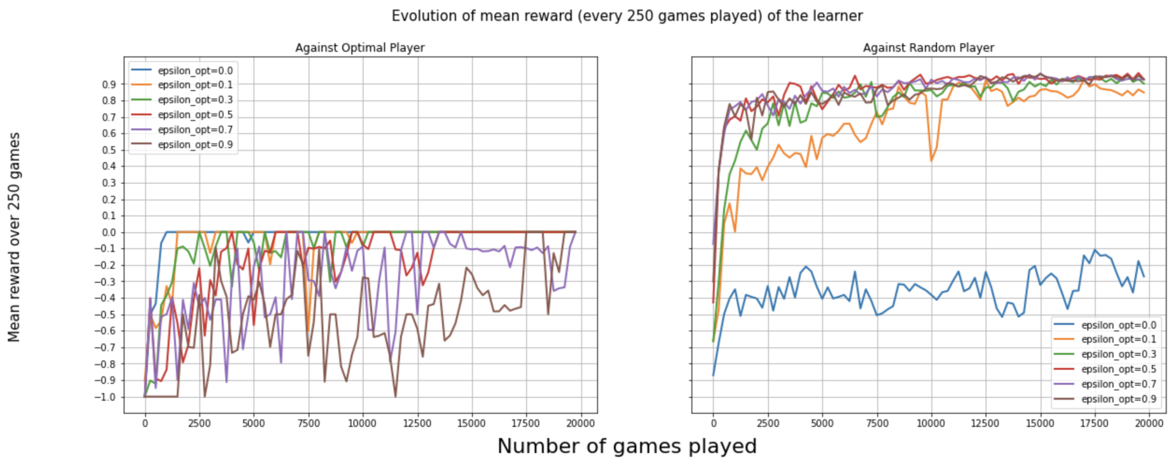


High n^* makes the random exploratory rate fall off slowly (during most of the training we keep high rate of random actions). Low n^* has the opposite effect, quickly letting the learner only play randomly 10% of the times.

The training process seems noisier compared to QLearning, especially against the optimal player. While in QLearning, we updated the Q-values one by one depending on the actions just made (one update does not impact other Q-values), we now change the whole function at each optimization step. As we sample from the "history" buffer, picking past bad moves can heavily impact on the performances as it changes the function as a whole. This explains the noisiness of the early results and why we decided to lower the learning rate later in the process as this mitigates the impact of an update coming from a sampled bad move. Thanks to the reduced oscillations, the learners achieve to converge towards optimal draw policy and high reward against random players.

Overall, it still seems useful to decrease ϵ during the learning process as doing so allows to reach slightly higher average rewards by the end of the training (for example with $n^* = 10000$).

Question 14



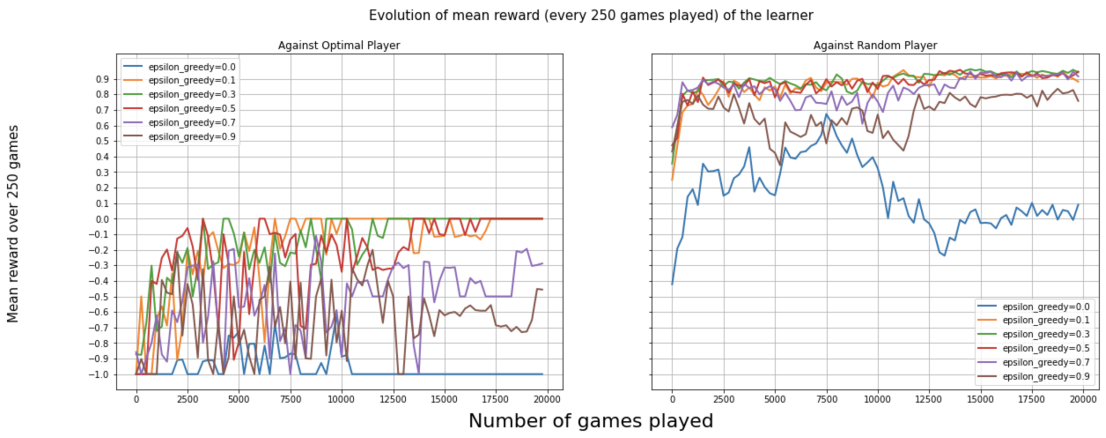
As expected, we observe that when the learner trains against a close to perfect player (low ϵ_{opt}), it quickly finds out optimal draw strategies but performs quite poorly against the random player (even staying below -0.3 average reward for $\epsilon_{opt} = 0$). On the opposite, training against a close to random opponent allows to perform well (above 90% win rate) when testing against a random player but it fails to discover perfect draw strategies when testing against optimal. The process is also extremely noisy even with the lowered learning rates after 10'000 and 15'000 iterations. These two facts make intuitive sense as, in each case, the learner performs best when tested against the type of player he trained with but struggles when confronted with strongly different playing style.

The opponents that appear as the best trainers are the ones around $\epsilon_{opt} = 0.5$. Against them, the learner manages to find and stabilize optimal draw policies (partially thanks to the reduced learning rates across the process) as well performing among the top win rates against random players (above 90%).

Question 15

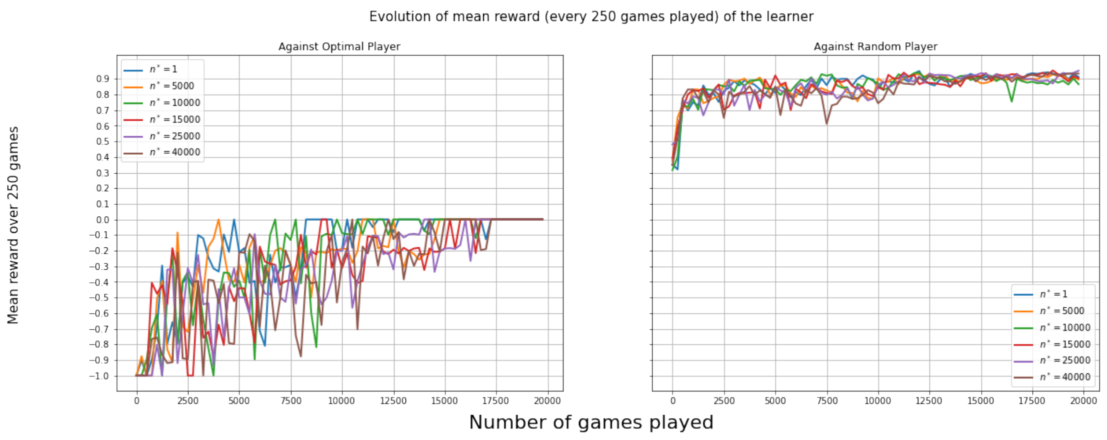
After training has converged (last 10% iterations), the highest values of M_{opt} and M_{rand} that were achieved after playing 20'000 games were respectively 0.966 (for $\epsilon_{opt} = 0.5$) and 0 (for all ϵ_{opt}).

Question 16



The agent generally learns how to play TicTacToe. It displays an increasing then stabilizing average reward against both optimal and random players. The only two exceptions occur for ϵ close to 0 or close to 1 which makes logical sense. In the first case, the two learners never explore new random moves which leads to very poor knowledge of the Q-values in most states. In the second case, both learners nearly always play random so they struggle estimating correct Q-values as defining good/bad moves is hard if both players react randomly and game winner greatly depends on randomness).

Question 17

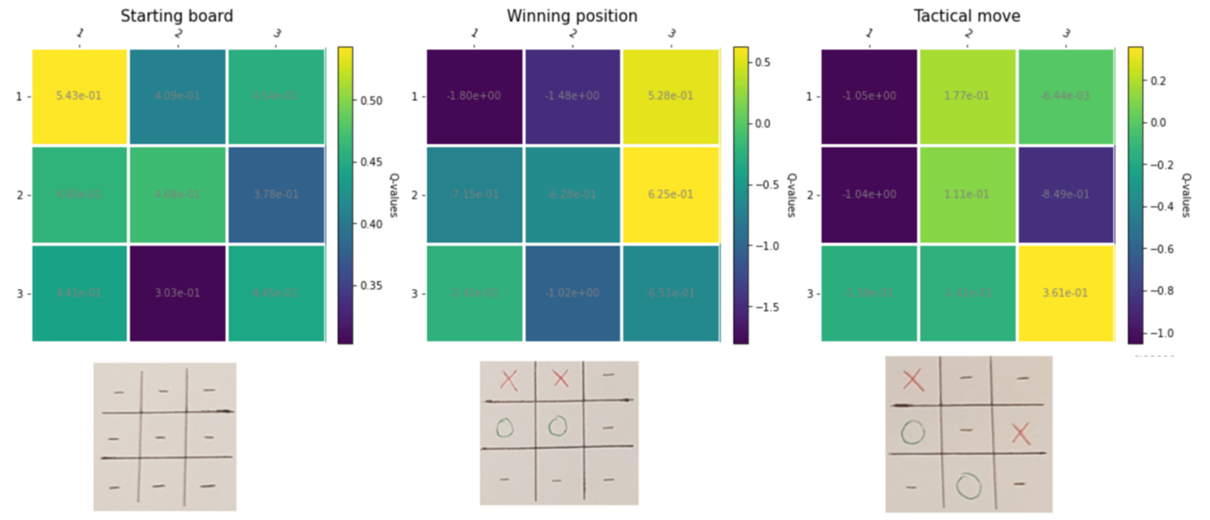


As in previous questions, n^* controls how fast the exploratory rate ϵ falls off from 0.8 to 0.1 and is the trade-off between exploration and exploitation. We can observe decreasing ϵ across learning again helps to achieve greater average rewards by the end of the training and wider Q-values knowledge. The higher n^*, the noisier the learning process when testing against optimal policy. Furthermore, the learning process seems noisier when training with self-learning compared to playing against ϵ-optimal policy (question 13).

Question 18

After training has converged (last 10% iterations), the highest values of M_{opt} and M_{rand} that were achieved after playing 20'000 games were respectively 0.954 (for $n^* = 15000$) and 0 (for all n^*).

Question 19

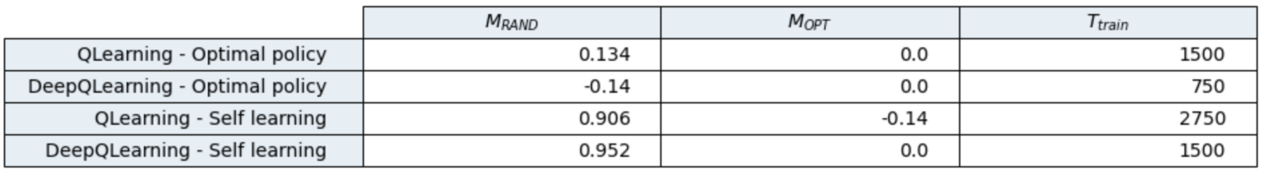


From the above boards, we can see that our agent learns the game well using self learning (in the boards, the agent plays 'X'). However, we can observe it sometimes fails to strongly recognize already taken spots and again lacks symmetry. For instance, we know starting any corner is equally good but even DQN doesn't not succeed at interpolating the symmetries: it only explores one possibility (upper left corner) in depth. On the second board arrangement, it is surprising to see that the agent thinks it is better to draw than directly winning the game. We can argue the agent learns more or less the optimal policy and didn't explore the winning positions in details. In other words, the player thinks it is safer to draw than trying to win (even with direct wins as in this case). On the third board arrangement, one can again observe the lack of symmetry in the agent's knowledge. In this position, there are two winning moves (either upper-right corner and lower-right corner) but the agent learns only the latter and fails to recognise the former.

Overall, DQN seems to suffer from similar limitations as QLearning even if it achieves better average results.

4 Comparison Q-learning vs Deep Q-Learning

Question 20



	M_{RAND}	M_{OPT}	T_{train}
QLearning - Optimal policy	0.134	0.0	1500
DeepQLearning - Optimal policy	-0.14	0.0	750
QLearning - Self learning	0.906	-0.14	2750
DeepQLearning - Self learning	0.952	0.0	1500

Question 21

When learning against a fully optimal player ($\epsilon = 0$), we can see that DQN quickly learns how to draw and so does the Q-learning. However, it is surprising to see that DQN is losing against the random player a non-negligible portion of the time whereas Q-learning has a decent positive win rate against it.

From the performance curves (especially if we compare questions 3 & 4 against questions 13 & 14), the Q-learning algorithm seems more stable when trained against the ϵ-greedy-optimal policy. In the first 10'000 games, the DQN performance against the optimal player have a really high variance. This can be explained by the fact that any update of the DQN model implies whole modification of the function approximating the Q-values whereas update of the Q-learning Q-values affects only the currently chosen action. Finally, having a replay buffer allows to make batch update but also induces the possibility to make very bad updates. This explains the drop in performance of DQN around 12'500 played games and after.

For the self learning part, we were surprised to observe this high performance with Q-learning algorithm. It achieves a 90% win rate against the random player and only lose 14% of the time against the optimal player when trained against itself. On the other hand, DQN outperforms Q-learning and scores a 95% win rate against the random player as well as draw policies against the optimal player. Nevertheless, it seems that the Q-learning learner is more confident to win a game than DQN (as shown for the second board in questions 10 and 19). As already pointed out, our "simple" learners greatly lack symmetries knowledge. On the test sets it does not seem to have massive impact but we think that real humans different AI adversaries could take advantage of this flaw to lower the win rate of our learners.