MINIPROJECT REPORT

CS-456

Artificial Neural Networks

Elia FANTINI (336006)
Félix KLEIN (344259)

ECOLE **P**OLYTECHNIQUE **F**ÉDÉRALE DE **L**AUSANNE

May 2022

Introduction

In this project we used Q-Learning and Deep Q-Learning methods to train an artificial agents to play the famous game of Nim.

The optimal policy of Nim is known, so we are going to see whether RL algorithms learn to play Nim by playing against optimal policy, or against itself (self-learning), and what are the advantages and drawbacks of Deep Q-Learning compared to Q-Learning.

A. Performance measures

For a given policy π, we define two quantities M_{opt} and M_{rand} to measure the performance of π in playing Nim:

- M_{opt} measures the performance of π against the optimal policy. To compute M_{opt}, we run π against `Opt(0)` (we refer to the ϵ-greedy optimal policy by `Opt(`ϵ`)`) for N = 500 games for different random seeds. π makes the first move in 250 games, and `Opt(0)` makes the first move in the rest. We count how many games π wins (N_{win}) and loses (N_{los}) and define $M_{opt} = \frac{N_{win} - N_{los}}{N}$.
- M_{rand} measures the performance of π against the random policy. To compute M_{rand}, we repeat what we did for computing M_{opt} but by using `Opt(1)` instead of `Opt(0)`.

I. Q-Learning

The first algorithm we are using is Q-Learning with ϵ-greedy policy, learning rate $\alpha = 0.1$, discount factor $\gamma = 0.99$ and zero-initialization for Q-values.

A. Learning from experts

In this section we test how our Q-agent copes when playing against `Opt(`ϵ_{opt}`)` with $\epsilon_{opt} = [0, 1]$.

Question 1 By running the algorithm with fixed ϵ_q = 0.1 to play against `Opt(0.5)` for 20'000 games while switching the first player every game, we get the following plot.

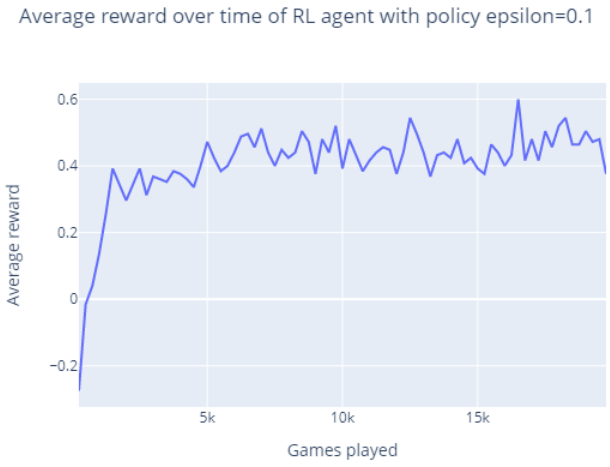


Figure 1: Learning from `Opt(0.5)`, fixed exploration rate 0.1

Mathematically, the winner among two `Opt(0)` players is known from the start. Random initial heaps means ~50% chances to win for `Opt(0)`, ~75% against `Opt(0.5)`, leading to a mean reward of 0.5. Since the learning agent reaches such mean reward, it means it has learned to play as `Opt(0)`.

Decreasing exploration

Decreasing exploration rate ϵ over time makes training more efficient. If we define $\epsilon(n)$ to be ϵ for game number n, then one feasible way to decrease exploration during training is to use $\epsilon(n) = \max\{\epsilon_{min}, \epsilon_{max}\left(1 - \frac{n}{n^*}\right)\}$.

Question 2 Plotting the mean reward every 250 games for different values of n^*, we get the following.

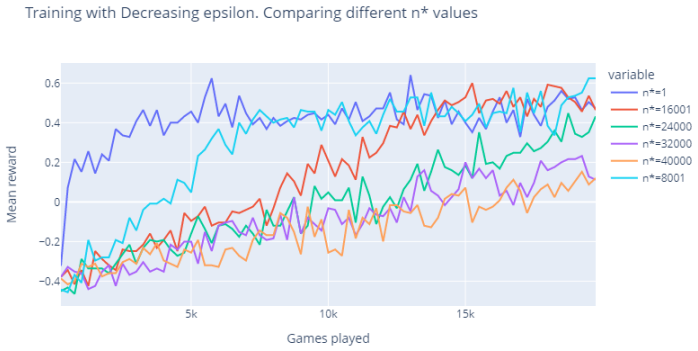


Figure 2: Learning from `Opt(0.5)`, decreasing exploration rate

It seems like $n^* = 1$ gives an extra boost in convergence speed at the very beginning, making the mean reward curve grow slightly faster than the solution with fixed exploration rate 0.1. On the other hand, higher values of n^* let the agent explore more actions and scenarios before converging to plateau. The sweet spot is $n^* = 16000$, converging to the same mean reward values in 20'000 games while managing to explore much more in the previous games thanks to the higher learning rate, while $n^* > 16000$ slows down the learning process too much, achieving a lower final mean reward.

Question 3 After every 250 games during training on different n^* values, we compute the test M_{opt} and M_{rand} for the agents on different. Plots of M_{opt} and M_{rand} over time follows.

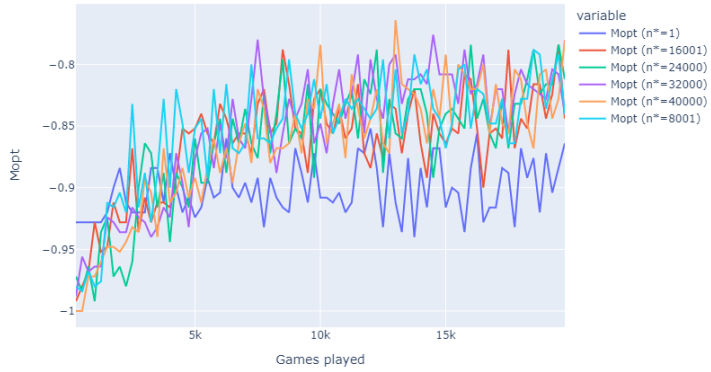


Figure 3: M_{opt} against `Opt(0.5)` on different n^*

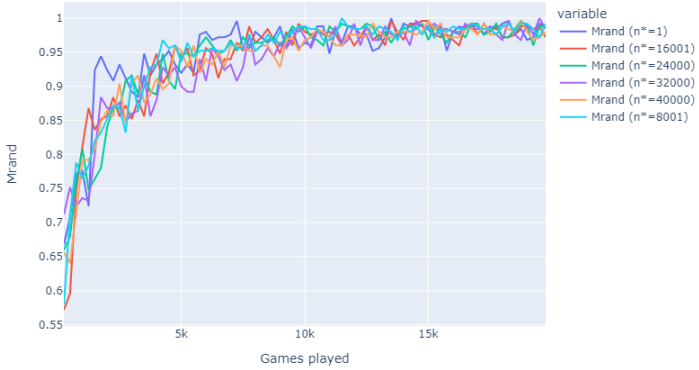


Figure 4: M_{rand} against `Opt(0.5)` on different n^*

The learning agent has certainly learned something, since it always manages to beat a random player, regardless of the n^* value. On the other hand, it loses around 3 out of 4 times against `Opt(0)`. This is due to the fact that it learned from a `Opt(0.5)` player that half of the times did a random and likely wrong move. If on M_{rand} the performance is not so different across n^* values, $n^* = 1$ is clearly worse than the others on M_{opt}.

B. Good experts and bad experts

Question 4 After every 250 games during training, we compute the test M_{opt} and M_{rand} for the agent. We repeated the training for 11 different ϵ_{opt} values for the adversary `Opt(`ϵ_{opt}`)`, while sticking to a decreasing exploration with $n^* = 16000$. Plots for M_{opt} and M_{rand} over time follows.

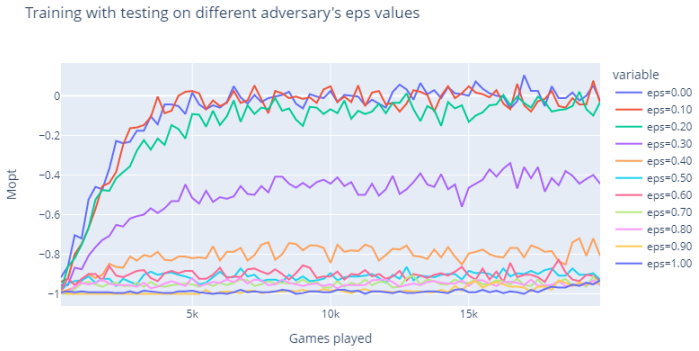


Figure 5: M_{opt} with different ϵ_{opt} for `Opt(`ϵ_{opt}`)`, using $n^* = 16000$

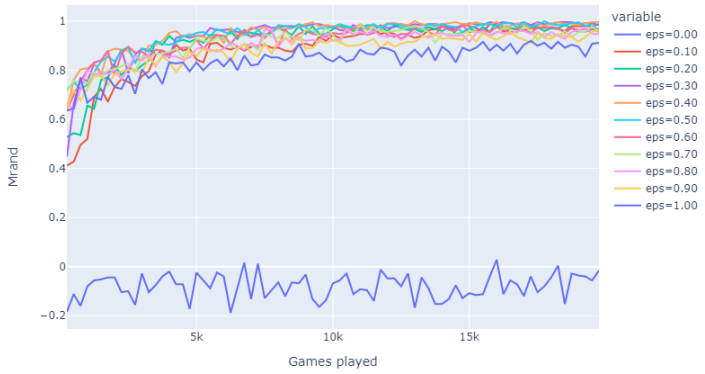


Figure 6: M_{rand} with different ϵ_{opt} for `Opt(`ϵ_{opt}`)`, using $n^* = 16000$

For every value of ϵ_{opt} except for 0.0 the agent shows to have learned something, since it always defeats the random player. On the other hand, for $\epsilon_{opt} = 0$ M_{rand} is much lower. This is because the player has always played against the optimal player, which always does the perfect move: this way the learning agent never explores the scenarios where he finds itself while playing against the random player, hence it has learned no proper Q-values and doesn't know how to play, even though it is perfectly capable to win against `Opt(`ϵ_0`)`. For this reason, it is preferable to learn against `Opt(`$\epsilon_{0.1}$`)`, reaching best M_{rand} and M_{opt} compared to all other ϵ_{opt} values.

Question 5 The highest value of M_{opt} and M_{rand} are achieved by learning against `Opt(`$\epsilon_{0.1}$`)`, reaching M_{rand} = 1 and M_{opt} = -0.004. M_{rand} can't get any higher, whereas M_{opt} reached is peak after 18'250 games, with a value of 0.064.

Question 6 Assume that Agent 1 learns by playing against `Opt(`ϵ_0`)` and find the optimal Q-values $Q_1(s, a)$. In addition, assume that Agent 2 learns by playing against `Opt(`ϵ_1`)` and find the optimal Q-values $Q_2(s, a)$. Do $Q_1(s, a)$ and $Q_2(s, a)$ have the same values?
No, $Q_1(s, a)$ and $Q_2(s, a)$ do not have the same values. Q_1 values

are learned by playing against a player that always does the best move, whereas Q_2 values are learned by playing against a player that does totally random moves. Hence, Q_2 values will be totally meaningless, since every time the agent wins or loses is because of luck. On the other hand, the agent who played against `Opt(0)` understands how to win, but it never explores those scenarios that happen when the adversary does not the optimal move, so most of the values in the Q-table are still zero.

C. Learning by self-practice

In this section, we test our algorithm by *self-practice*, i.e. when the agent plays against itself. Hence, both players use and update the same set of Q-values.

Question 7 Plotting M_{opt} and M_{rand} for different values of ϵ, we get the following.

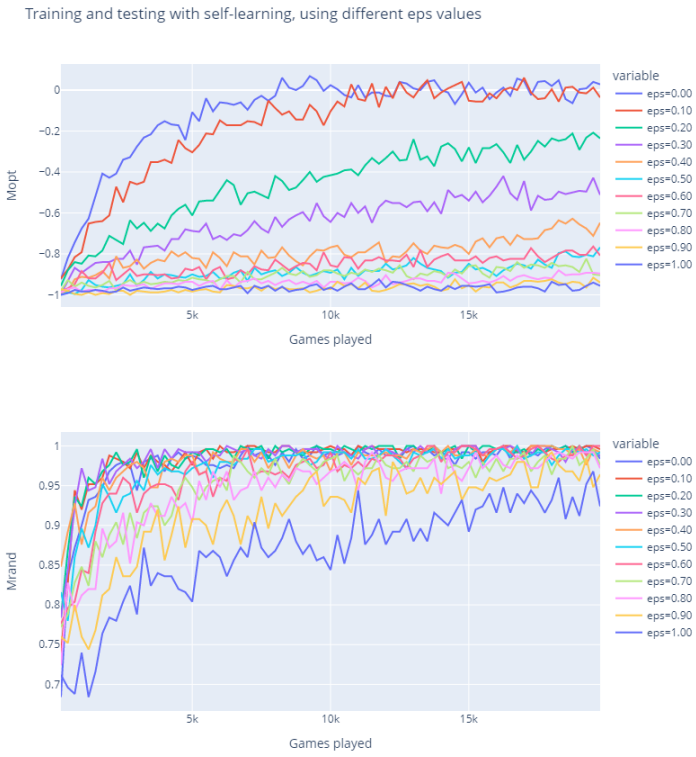


Figure 7: self-practice: M_{opt} and M_{rand} for different values of ϵ

Since the players play in the same way, they have the same probability of winning, so the mean reward (not shown in plots) oscillates around 0. Again, our agent is always better than the random player, but only low ϵ values seem to learn and play as good as an `Opt(0)` player. Compared to not self-practice method, now also $\epsilon = 0$ agent defeats the random player. This is because the agent has not played against a perfect player, so it has also seen scenarios due to wrong moves, especially in the first games. $\epsilon = 0.1$ is still the best.

Question 8 After every 250 games during training,we compute the test M_{opt} and M_{rand} for the agent, using the same decreasing epsilon method as before. Plots for M_{opt} and M_{rand} over time follows.

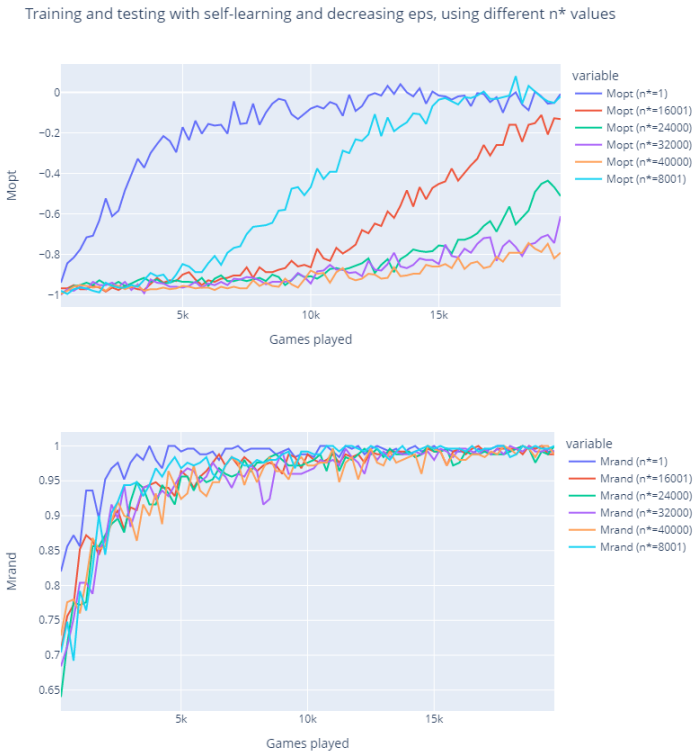
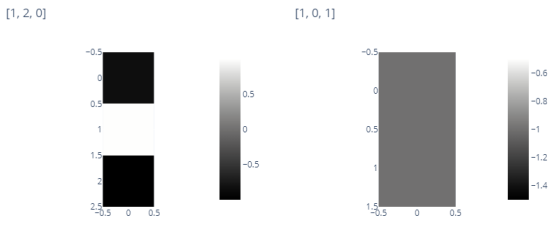


Figure 8: Self-learning with decreasing ϵ, on different n^* values

As seen previously, $\epsilon = 0.1$ is the best option, so now all agents end up with that minimum ϵ, but they first explore more with a higher exploration rate. We can see the improvements with M_{rand} since now every agent performs a score really close to 1, while with M_{opt} only agents trained with $n^* \leq 6000$ reach a score of 0, since the others never reach $\epsilon = 0.1$ in 20'000 games. Among those, $n^* = 8000$ is the best since it reaches the best scores while exploring more scenarios than the $n^* = 1$ agent.

Question 9 All agents trained with all n^* finish 20000 games with M_{rand} roughly equal to 1. Regarding M_{opt}, the highest value is achieved by the agent trained with $n^* = 8000$, achieving $M_{opt} = -0.01$

Question 10 Visualizing Q-values for three board arrangements ([1,2,0], [1,0,1] and [3,4,7]) using heatmaps, it is possible to analyse how the agent has learned to play.



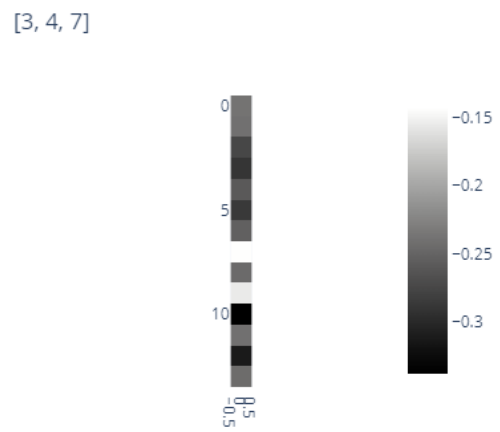


Figure 9: Q-value heatmaps for 3 board arrangements

The values make sense: in the first case, [1,2,0], the moves that make you lose in the very next move have Q-value equals to -1, whereas the best move has 0.99. In the second case, [1,0,1], both moves make you lose and they both have a -1 Q-value. Last case is less intuitive, [3,4,7], but the optimal policy tells us that removing 1 stick from heap 3 is the best move, which is action 7, the one with the highest Q-value. Hence, we can deduce that the agent has learned how to play.

II. Deep Q-Learning

In this part, the same task of game learning is performed but using Neural Networks and Deep Reinforcement Learning, more specifically the DQN algorithm combined with ϵ-greedy policy.

A. Implementation details

The network's architecture and hyperparameters are the same as the ones suggested by the guidelines. We have tried augmenting the number of neurons as well as the number of layers, but we measured no significant improvement. We have also tried different learning rates for Adam, as well as using a scheduler to reduce learning rate over time, but it didn't help and the suggested learning rate $5 \cdot 10^{-4}$ proved to be the best choice.

B. Learning from experts

We run the DQN agent with a fixed and arbitrary $\epsilon = 0.1$ against `Opt(0.5)` for 20000 games, switching the first player after every game.

Question 11 Running our DQN algorithm for 20'000 games against `Opt(0.5)`, switching the first player after every game, we have the following plot.

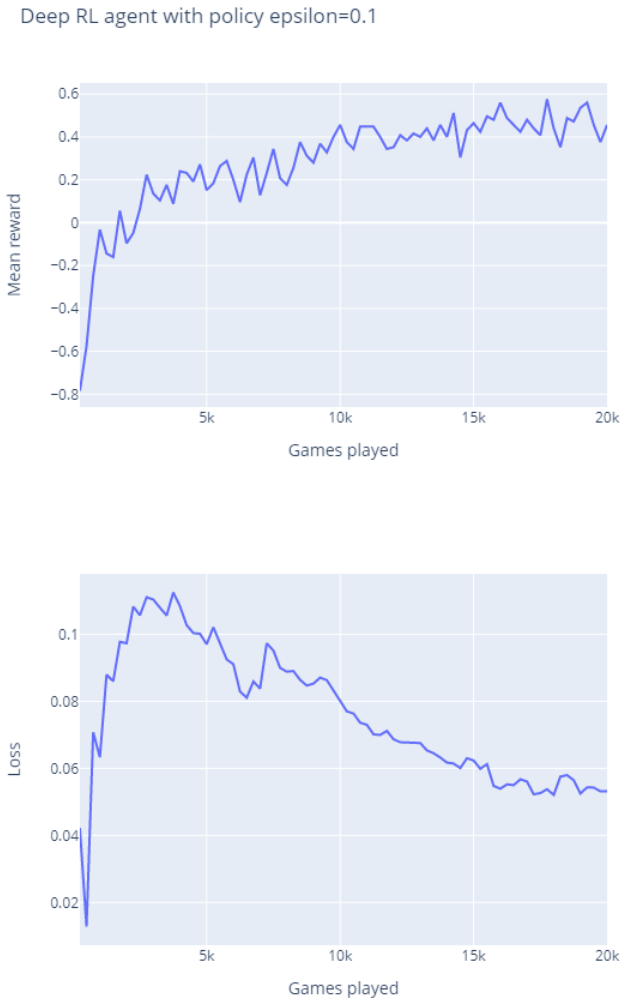


Figure 10: Deep RL agent playing against `Opt(0.5)` (mean reward and loss)

We can observe that the agent learns how to play Nim against a player that makes optimal moves 50% of the time: the mean reward after 20'000 games is in the same range as the Q-Learning agent, which leads us to the same reasoning as question 1. On the other hand, loss does not decrease dramatically and stays relatively stable. This is because the DQN algorithm is model-free: to obtain a decreasing training loss we need a model-based approach. Furthermore, the training loss soars after a few games because the expected result differs from the one our agent was implicitly predicting. Then, once it actually has learned how to play, it goes down again.

Question 12

We now repeat the same training but without the replay buffer and with a batch size of 1. The result is shown in the following plot.

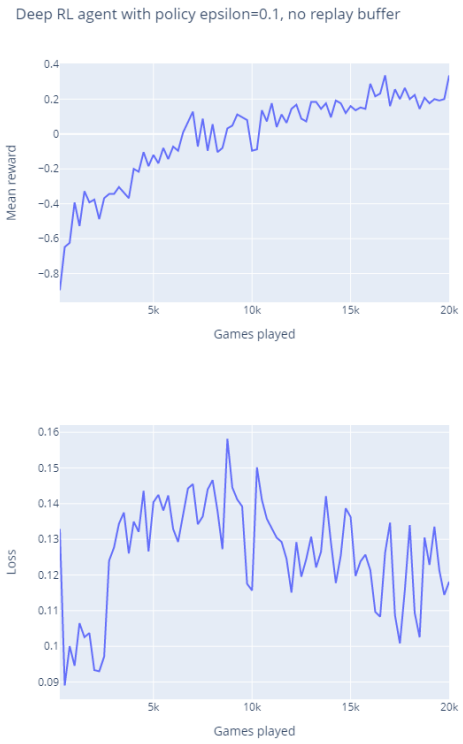
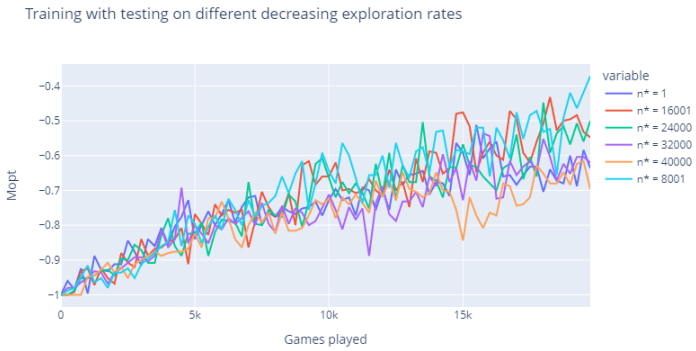


Figure 11: Deep RL agent playing against `Opt(0.5)`, batch size of 1

Setting the batch size to 1 makes the learning slower, reaching a lower mean reward. The gradient over the loss is computed on just one sample, so the update of the model's weights is more unreliable, as we can in the loss plot, which keeps oscillating.

Question 13

Instead of fixing ϵ, we now use decreasing ϵ, using different values of n^* from a reasonably wide interval between 1 to 40000 run. Again, we train the agent against `Opt(0.5)`.



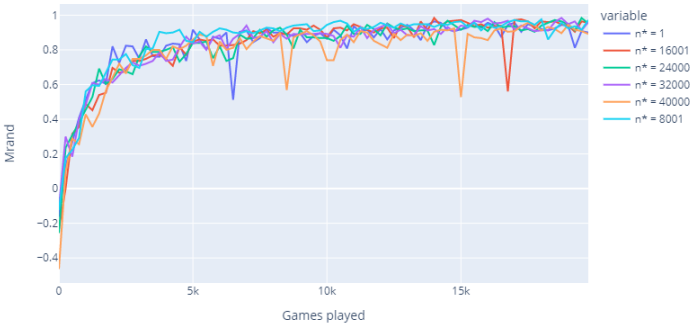
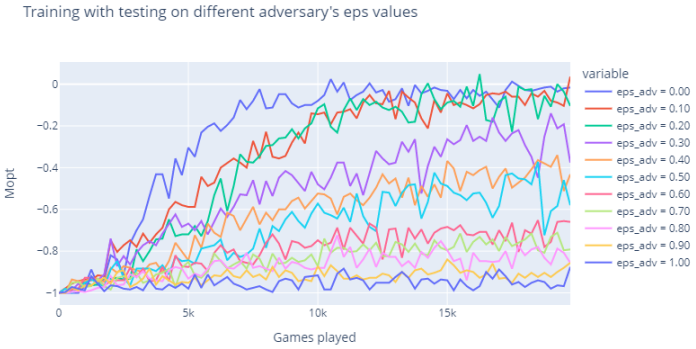


Figure 12: Training with testing on different n against `Opt(0.5)`

Playing against `Opt(0.5)`, we notice that decreasing exploration does not help compared to having a fixed ϵ. Indeed, the maximum value we would like to reach after 20'000 games is 0 while we only reach -0.4 with the best configuration, which is $n^* = 8000$. On the other hand, this is normal because we're playing not against the perfect player, but one that does random moves half of the times. Comparing the results with the Q-learning method, we can see that M_{opt} is almost double for the Deep RL agent, while M_{rand} remains really close to 1 in both cases.

Question 14 Keeping decreasing exploration and $n^* = 8000$, we run DQN against Opt(ϵ_{opt}) for different values of ϵ-opt for 20000 games.



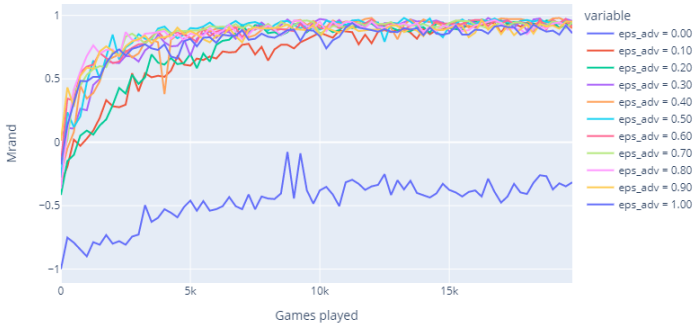


Figure 13: Training with testing on different adversary ϵ

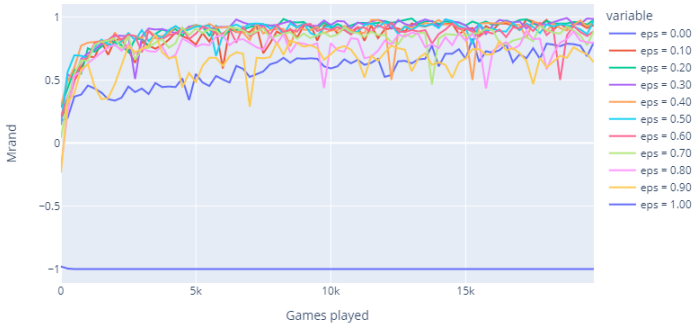


Figure 14: Training self-learning with different ϵ values

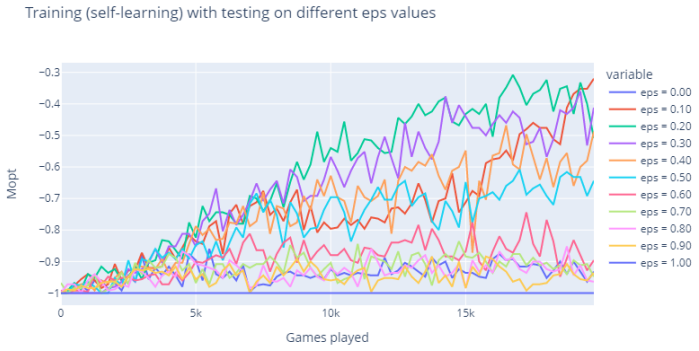
Results are very similar to the ones from question 4, hence the observations are the same. On the other hand, it's worth noting that the convergence speed is slower than the Q-learning counterpart: if Q-learning agents reached almost peak performance after 5'000 games, DQN agents reach it between 10'000 to 15'000. This is probably because the DQN agent doesn't know any rule about the game and so it has to learn which actions are valid first, whereas Q-learning agent can do only valid actions.

Question 15 The highest values we could reach are around 0.04 for M_{opt} ($\epsilon_{opt} = 0.1$) and 0.95 for M_{rand} ($\epsilon_{opt} \in [0.1, 1]$).

C. Learning by self-practice

For different values of $\epsilon \in [0, 1)$, we run a DQN agent against itself for 20000 games: both players use the same neural network and share the same replay buffer.

Question 16 After every 250 games during training, compute the test M_{opt} and M_{rand} for different values of $\epsilon \in [0, 1)$. Plot of M_{opt} and M_{rand} over time follows.



The agent learns to play Nim, but slowly. Indeed, the best mean reward is achieved with $\epsilon = 0.1$ at roughly -0.32 after 16'750 games. Running the training on 40'000 games, the agents converges to $M_{opt} = 0$ as expected, so 20'000 games are not enough. The amount of games needed to reach peak performance is double the amount needed without self-learning, as it was for Q-learning. Again, being slow makes sense because the DQN agent can also play invalid moves, and by playing against itself it really explores a lot of scenarios before converging to the game's optimal policy.

Question 17 Instead of fixing ϵ, we now use decreasing exploration with different values of n^*.

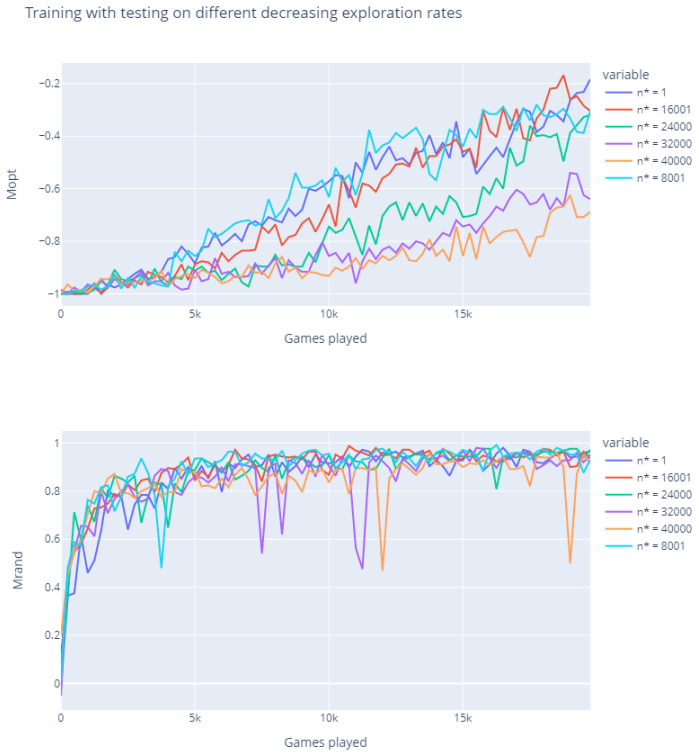


Figure 15: Training with testing on different decreasing exploration rates

Results are roughly the same as with a fixed ϵ approach. We expected that, since decreasing exploration slows down convergence to let the agent explore more scenarios. In this case though, the amount of scenarios is already very big, hence decreasing exploration can only slow down convergence, with a lower or equal M_{opt} after 20000 games. A peak of $M_{opt} = -0.18$ is reached with $n^* = 1$, which makes sense because $n^* = 1$ makes ϵ immediately go down to the minimum.

Question 18 For question 16, we reach -0.32 for M_{opt} ($\epsilon = 0.1$) and 0.992 for M_{rand} ($\epsilon = 0.3$).

For question 17, we reach -0.168 for M_{opt} ($n^* = 16000$) and 0.992 for M_{rand} ($n^* = 8000$).

Question 19 Testing with the same board arrangements as in part I with decreasing exploration,

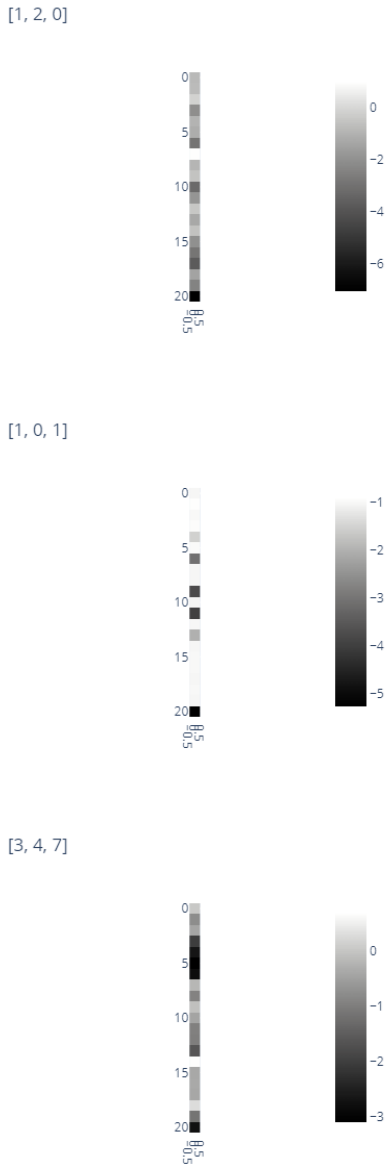


Figure 16: Q-value heatmaps for 3 board arrangements

Here we visualized all 21 options because, especially during the first few games with an untrained agent, it can choose unavailable actions. For the first case, the optimal action is taking one stick away from heap 2, which is actually the one with the highest Q-value for the agent. For the second case, the agent would lose either way and its Q-values are quite confused: many cells contain high values, probably because it has never tried those actions before. In fact, whatever the action chosen, it would receive a -1 reward, either because it has lost or because it has done an invalid move For the third case, the agent picks the best action, which is removing 1 stick from heap 3, corresponding to the fourteenth cell of the Q-values array.

This shows the DQN agent has actually learned to play Nim quite well, even though it does not know the rules of the game at the beginning.

III. Comparison between Q-Learning and Deep Q-Learning

Question 20 The following table shows the best performance (the highest M_{opt} and M_{rand}) of Q-Learning and DQN (both for learning from experts and for learning by self-practice) and their corresponding training time. We define the training time T_{train} as the number of games an algorithm needs to play in order to reach 80% of its final performance according to both M_{opt} and M_{rand}.

	M_{opt}	M_{rand}	T_{train} (k games)
Q-learning	0.01	0.99	~5
Self Q-learning	-0.06	0.99	~13.7
Deep Q-learning	0.03	0.90	~10
Self Deep Q-learning	-0.28	0.93	~15.7

Table I: Best M_{opt} and M_{rand}, with corresponding training time

Question 21: Conclusions

Through a lot of different game configurations (playing against good and bad experts, by self-learning, using decreasing exploration, modifying the batch size for Deep Q-Learning), our agents learned how to play the game of Nim more or less well. With the appropriate ϵ and n^* though, some agents actually converged towards the optimal policy. Among the two methods, Q-Learning turned out to be preferable for learning such a simple game, since it manages to achieve similar results in much less time. In fact, not only the number of games to learn the optimal policy is much less, but also the computation time: one Q-Learning training takes 10 seconds on a Ryzen 7 3700x CPU, whereas one Deep Q-Learning training takes roughly 10 minutes on a RTX 3070 GPU (4 minutes without testing and self-learning). On the other hand, if time is not a problem, for sure opting for a Deep Q-Learning solution on 40'000 games with self-learning and decreasing epsilon will give the most robust model, since it will have explored much more states and scenarios than the simpler Q-Learning method.

Link to GitHub Repository