Artificial Neural Networks: Nim Miniproject

Bruno Ploumhans

Francesco Salvi

1. Introduction

In this project, we use Q-Learning and Deep Q-Learning to teach an artificial agent to play the game of Nim. In section 2, we implement a Q-Learning agent and run it with different training parameters, commenting the results. We then move to Deep Q-Learning in section 3, and we run similar tests. Finally, in section 4, we compare the two approaches for this specific problem. Our analysis follows the questions laid out in the project statement.

2. Q-learning

2.1. Learning from experts

Question 1

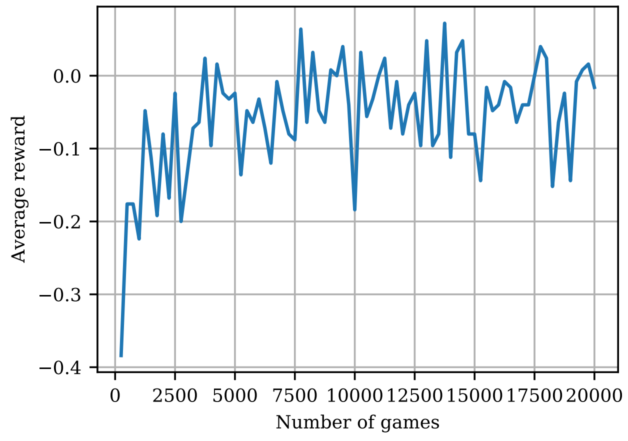

Figure: Average reward against $Opt(0.5)$, with $\epsilon = 0.5$.
We observe that our agent is learning pretty quickly to play Nim with a performance similar to the expert. In fact, with the same $\epsilon = 0.5$, the best possible performance for our agent is to equalize the expert, and we observe indeed an average reward close to an overall draw.

Question 2

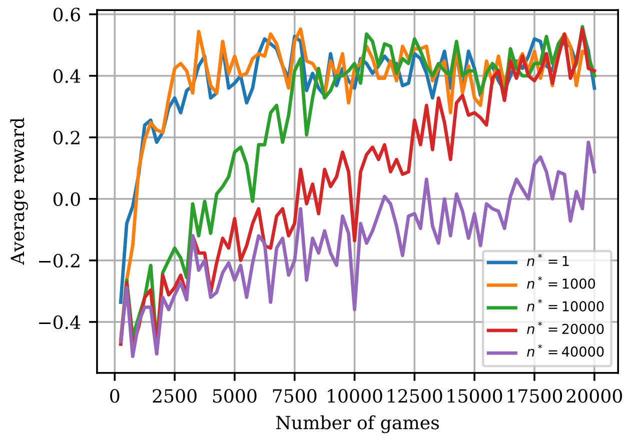

Figure: Average reward against $Opt(0.5)$, varying n^*.
We observe that our agents, except for $n^* = 40\,000$, are all converging after some time to an average reward around 0.45, learning thus to beat $Opt(0.5)$ in almost 3 out of 4 games. From this result, it would seem that higher $n*$ are in this case rather useless, increasing the time needed to reach convergence without providing any apparent benefit. However, games in this setting are biased by playing against $Opt(0.5)$, while our agents reach a much lower $\epsilon_{min} = 0.1$. This means that our agents do not need to learn the optimal move in all scenarios, especially not in states far from the end of the game, because they can count on the fact that the optimal player will randomly make a mistake much more often then them. This also explains why $n^* = 40\,000$ performs worse than the other agents, as after $20\,000$ games it will only reach $\epsilon = 0.4$. Intuitively, then, the final performance seems to only depend on the difference between 0.5 and $\epsilon(n = 20\,000)$, with exploration having very little effect for all $n^* < 20,000$. Compared to having a fixed ϵ, thus, varying it with time helps if the final ϵ is lower than the fixed one.

Question 3

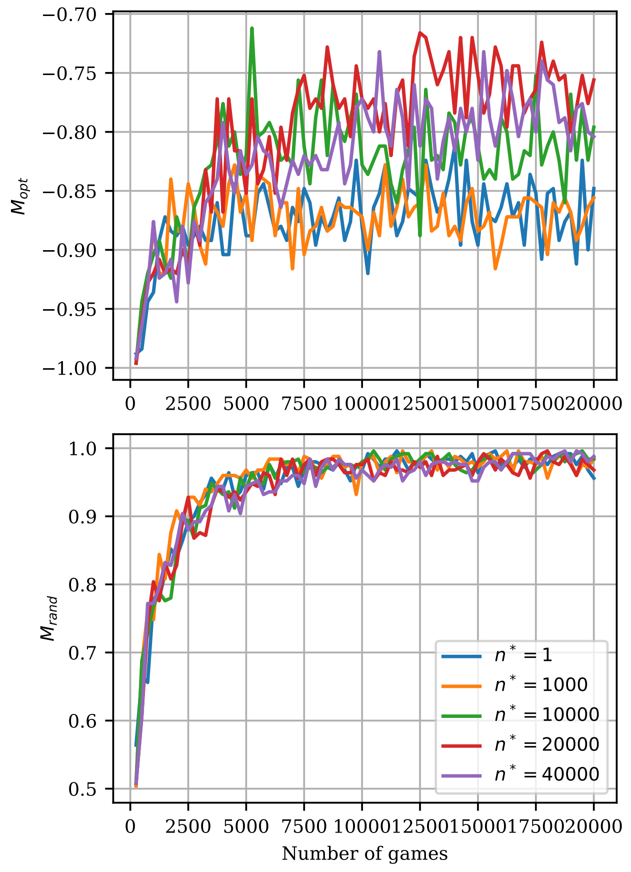

Figure: Evolution of M_{opt} and M_{rand}, varying n^*.
In general, all our agents perform very well against $Opt(1)$, winning almost every game, but play rather badly against $Opt(0)$. This can be caused by the fact that when training against $Opt(0.5)$, as argued in Q2, the agents do not need to learn the best move in every state, but this is a major disadvantage when the opponent is $Opt(0)$, which penalizes them for every mistake. With respect to the previous question, though, we now observe remarkable variations across different n^*, with exploration having a substantially positive impact by helping agents to not get stuck in a wrong policy.

Question 4

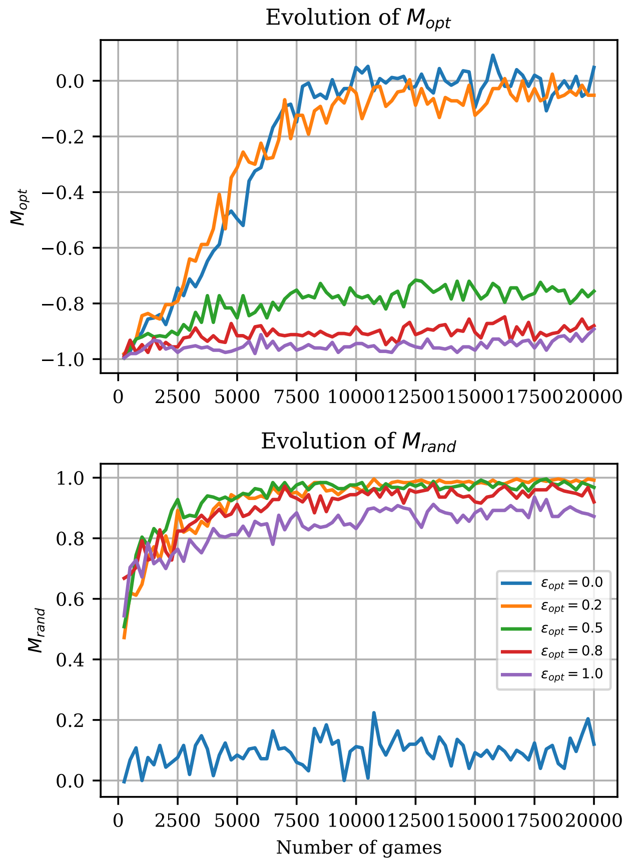

Figure: Evolution of M_{opt} and M_{rand}, with $n^* = 20\,000$.
We observe that if our agent is trained against $Opt(0)$ it learns quite quickly to play against him, equalizing the performances, but remarkably fails to outperform the naive random player $Opt(1)$. This is an interesting effect that is due to the particular way the optimal player acts when the Nim sum is already 0 and all the moves are equally losing, taking always 1 item from the first max heap. Therefore, our agent would perfectly know the optimal move in a state such as [5, 6, 0], but would have no clue what to do if the state is [6, 5, 0]. Even, it wouldn't know anything in a state such as [7, 0, 0], because it learns to win only through [1, 0, 0]. Therefore, it cannot beat the random player, because with the latter it will get into a lot of states it has never seen before. On the other hand, when the agent is trained against the random player, it achieves good performances against the latter but completely fails to play against the best player. In such a situation, in fact, the agent is learning the right move in very simple positions that are one or few moves away from the end of the game, but never learns which moves are good or bad in more complex states, as it never gets punished for mistakes. For other values of ϵ_{opt}, we observe instead a mix of the two mentioned situations.

Question 5

In general, the best possible (average) values for the two metrics are $M_{opt} = 0$ and $M_{rand} \simeq 1$. For Q4, the best (pointwise) results are $M^*_{opt} = 0.092$, for $\epsilon = 0$, and $M^*_{rand} = 0.996$, for $\epsilon = 0.2$. Note that the theoretical optimal performances can already be approximated (on average) very well with 20 000 games by picking, for example, $\epsilon_{opt} = 0.2$, allowing the agent to get punished for bad moves (hence, learning good ones) and at the same time to sufficiently explore of all the winning states. The same optimal performances could also be achieved for every value $0 < \epsilon_{opt} < 1$, but eventually with many more games.

Question 6

The two agents will never learn the same Q-values, because of the different characteristics of the two players they are trained against. Even ignoring winning states which are never explored when playing against $Opt(1)$, the agent playing against $Opt(0)$ does not get punished for bad moves and will thus assign a positive reward to every complex action. Moreover, there are particular strategies that only apply to $Opt(0)$: in a losing state such as [5, 5, 0], even if in theory all moves are equally bad, the best move would be to take only 1 item from no matter which heap, because in that way the random player will have higher chances of randomly not picking the winning move in the next turn. In conclusion, the two agents are almost playing two different games, with different strategies, states and patterns, and except for actions very close to the end the Q-values will be completely different.

2.2. *Learning by self-practice*

Question 7

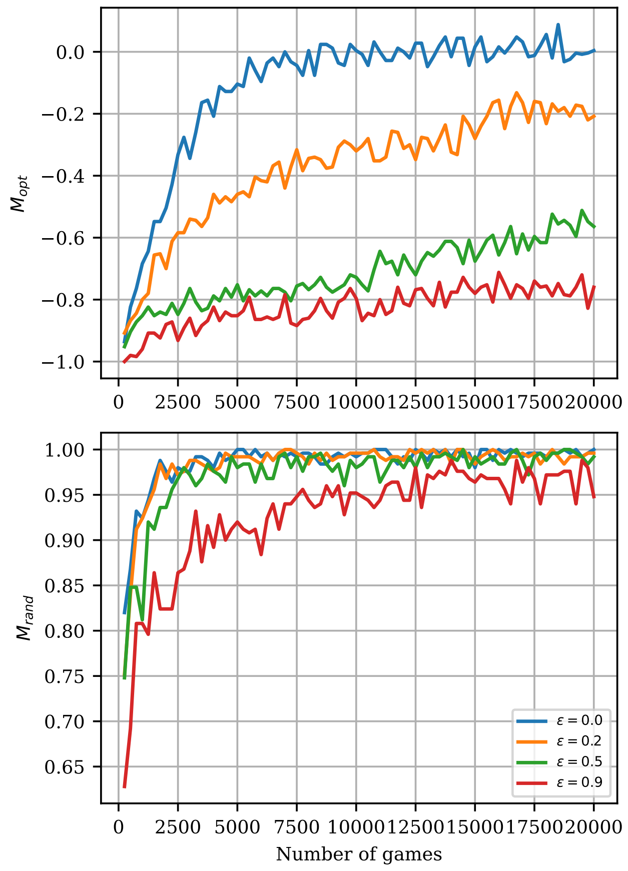

Figure: Evolution of M_{opt} and M_{rand}, varying ϵ.

Our self-learning agent is learning to play Nim pretty effectively, converging to the optimal performance both against $Opt(0)$ and $Opt(1)$, the faster the smaller ϵ is. This is logical in a solved game like Nim, where each move can binarily be only good or bad and there are no ”intermediate” suboptimal moves in which an agent could get stuck because of privileging too much exploitation, assuming that it gets punished for its mistakes. Therefore, while against $Opt(0.5)$ we still needed some exploration to balance the random component, here exploration is rather useless, and lower values of ϵ converge faster.

Question 8

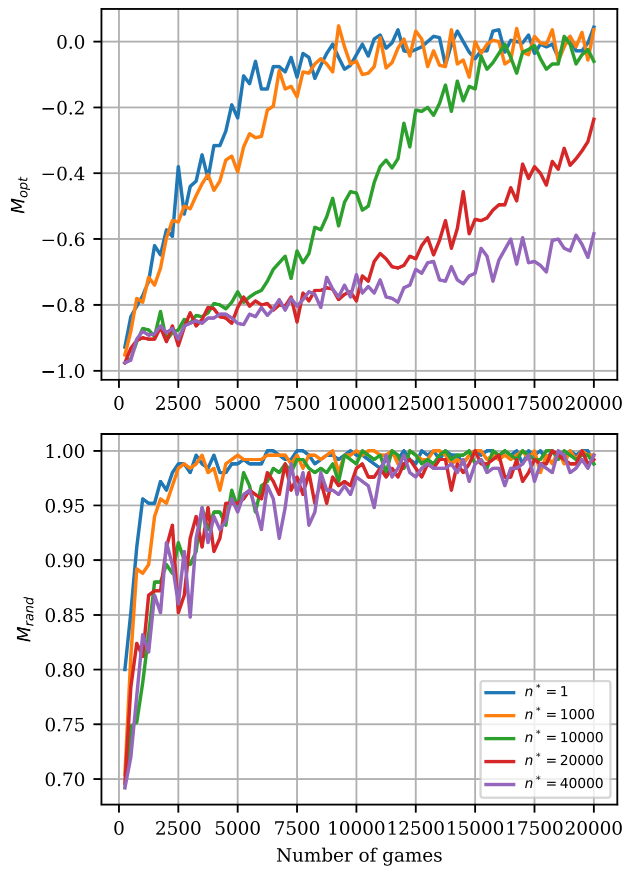

Figure: Evolution of M_{opt} and M_{rand}, varying n^*.
Again, we see that little to no exploration is more efficient in terms of convergence, and lower values of n^* are preferred. We observe, however, that decreasing ϵ over time is highly beneficial when starting with high exploration, letting the agent learn and converge faster to optimal performances.

Question 9

For Q8, we obtain at best (pointwise) $M^*_{opt} = 0.048$, for $n^* = 1000$, and $M^*_{rand} = 1.0$, for $n^* = 1$. Note that, while fixing $n^* = 1$ is conceptually similar to training against $Opt(0)$, here actions in a losing state are not chosen with the peculiar behavior of the Opt player, but rather still maximizing Q-values and breaking ties randomly. Therefore, while in Q4 training against $Opt(0)$ led to good M_{opt} but very bad M_{rand}, here $n^* = 1$ is sufficient to achieve the best possible performance.

Question 10

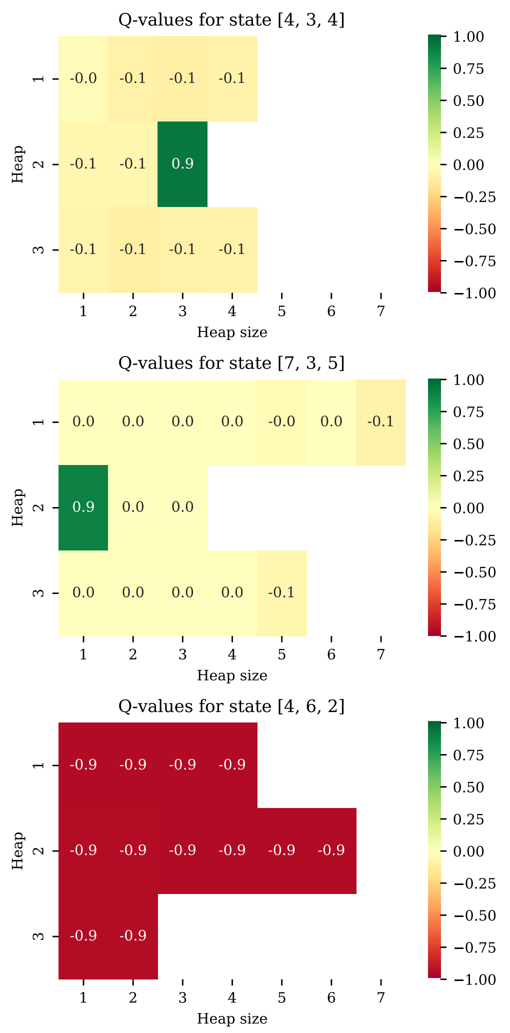

Figure: Q-values for the agent with $\epsilon = 0$.
We notice that in winning states, where the Nim sum is different from zero, the agent learned which move is the correct one, associating to it a positive and high Q-value. Since the agent is playing always with the optimal policy, as soon as the good move is found once the agent sticks to it, with the result that in winning states only that move has a Q-value significantly different from 0, and there can be even several actions that have never been tried and that have still a Q-value set to the initialization value 0. This is indeed the optimal behaviour, because, as already argued, once a good move is found there is no need for further exploration. On the other hand, in losing states where the Nim sum is zero, all the actions have a negative and low Q-value. The agent, in fact, tries unsuccessfully all the possible actions to find a good one, which of course does not exist, and thus the Q-values decrease together alternating the choice of the action. This evidence shows thus that our agent is learning to play Nim very effectively, both in winning and losing states.

3. Deep Q-Learning

3.1. Learning from experts

Question 11

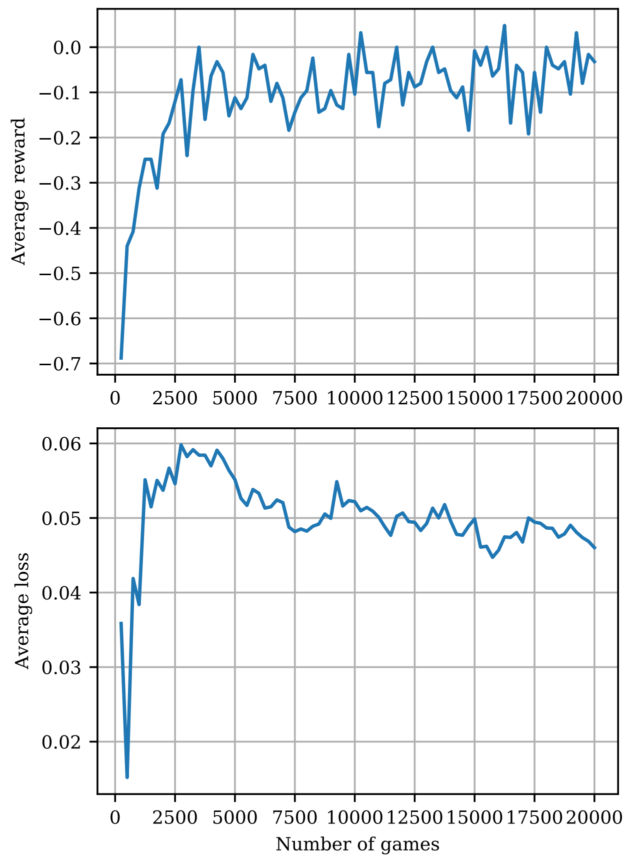

Figure : Average loss and reward against $Opt(0.5)$, with $\epsilon = 0.5$.

The agent seems to be learning to play Nim quite effectively, reaching almost equal results as the 0.5-optimal player. As already argued in Q1, with a shared $\epsilon = 0.5$ the best possible performance is indeed around an average reward of 0, which the agent seems to be approaching after some time. Concerning the loss, we observe an initial drop followed by a sharp climb until ∼2500 games are played, after which the loss starts to slowly decrease. Looking at the more fine-grained level, we notice that, at least for the first 2500 games, the loss exhibits a systematic periodic pattern of decrease-increase every two steps. This corresponds exactly to the update frequency of the target network, set to 500 games. In fact, every time that the target is updated, the Q-value data distribution that the network tries to maximize is shifting, causing the loss to likely undergo an instantaneous increase. In later stages, the shift between actor Q-values and target Q-values is likely smaller, so the loss evolution stabilizes to a slow decrease.

Question 12

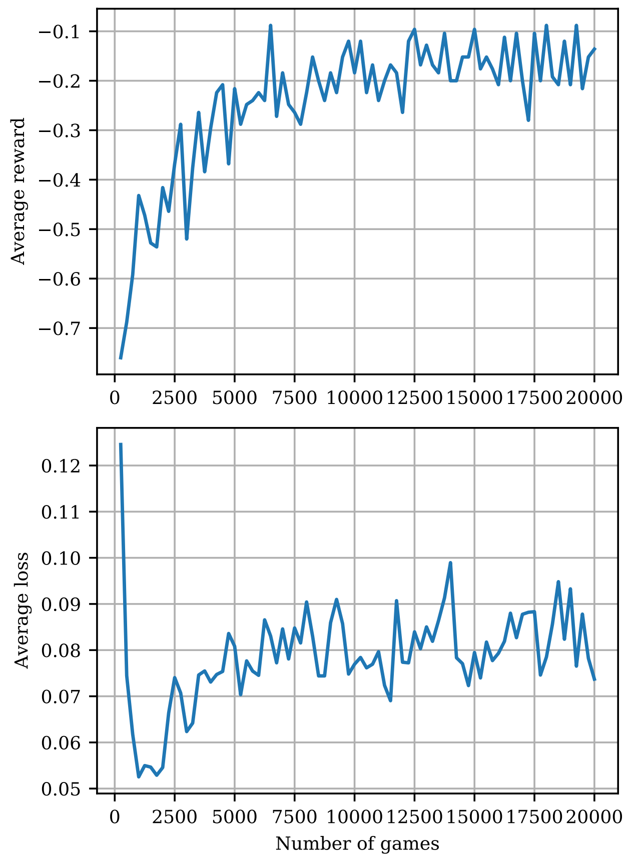

Figure : Average loss and reward against $Opt(0.5)$, with $\epsilon = 0.5$, without a replay buffer.

Without a replay buffer, we observe that the agent performs worse, stabilizing on an average reward around -0.15 and with a loss continuing to oscillate without decreasing. Indeed, without a buffer, subsequent updates are temporally highly correlated, causing the weights to go through high fluctuations and the training to be unstable.

Question 13

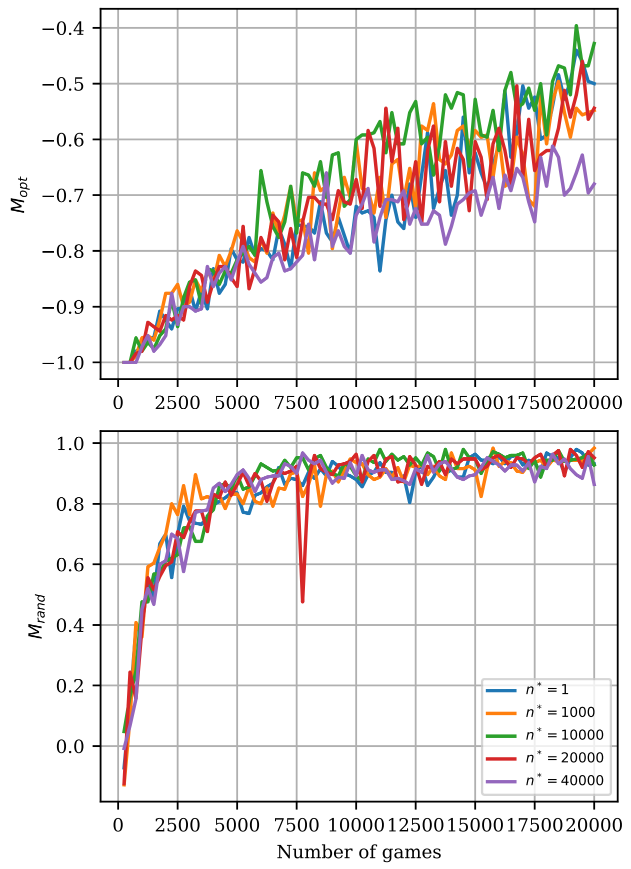

Figure: Evolution of M_{opt} and M_{rand}, varying n^*.

Running the DQN agent with different values of n^*, we obtain results qualitatively similar to those of Q3. The network, trained against $Opt(0.5)$, learns to play well against $Opt(1)$, but after 20 000 games still struggles against $Opt(0)$. We see, however, that here the growth of M_{opt} is more consistent, with the average reward linearly increasing over time. It seems, thus, that using a neural network as a function approximator of the Q-values is helping in learning some general patterns faster, since it allows for similar states to share a common representation. Moreover, we observe in this case the benefit of decreasing ϵ with time, helping in training to get faster convergence with respect to keeping it fixed (with $n^* = 1$). The tradeoff between exploration and exploitation, controlled by n^*, seems in this case to have an optimum for $n^* = 10\,000$, for which our agent gives the best results.

Question 14

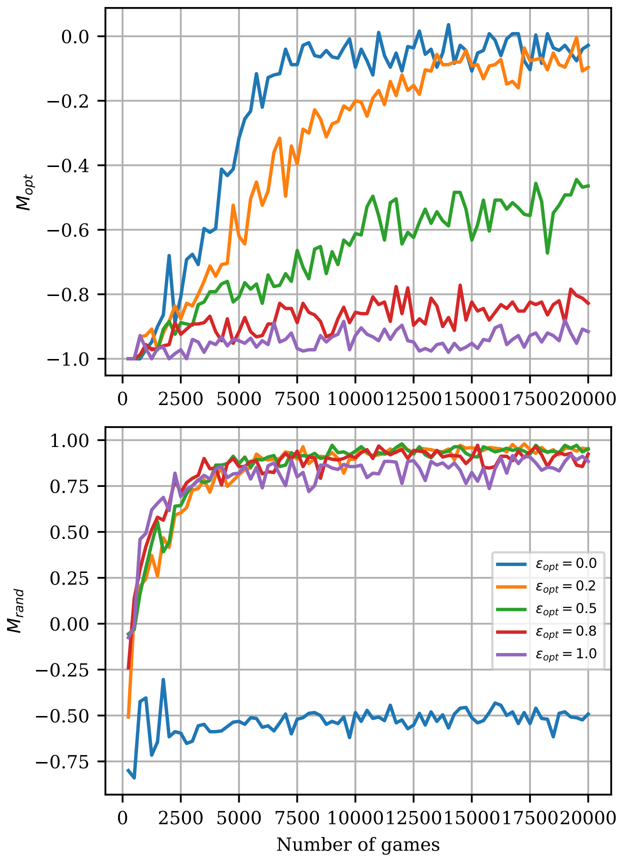

Figure: Evolution of M_{opt} and M_{rand}, with $n^* = 10\,000$.

For the "test" performances, we obtain similar results to the ones discussed in Q4. When trained against $Opt(0)$, in fact, our DQN agent learns to play against it pretty well, reaching an M_{opt} of about 0, but performs badly against $Opt(1)$. In this case, though, "badly" is much worse than in Q4, with an average M_{rand} around -0.5 instead of 0. This happens because when the agent gets into states never encountered before, as it is the case if it is trained against $Opt(0)$ because of the peculiar strategy adopted by the latter in losing positions, not only it has no idea of which moves are good, but also does not know which moves are even allowed. Therefore, in such states the agent will be likely to pick an illegal move, immediately ending the game with a negative reward. On the other hand, agents trained against Opt with some randomness ($\epsilon > 0$) learn to play very effectively against $Opt(0)$, reaching $M_{rand} \simeq 1$, at the cost however of slower convergence to $M_{opt} = 0$. Note that, as for Q4, while (convergence to) M_{opt} and ϵ are inversely proportional, it is not true that conversely M_{rand} and ϵ are directly proportional. In fact, despite the fact that to play well against the random player it is sufficient to learn good moves a few steps from the end of the game, learning good Q-values also for all other states can definitely help, and this can only happen with some punishments for bad moves.

Question 15

For Q14, we obtain at best (pointwise) $M^*_{opt} = 0.036$, for $\epsilon = 0$, and $M^*_{rand} = 0.98$, for $\epsilon = 0.2$.

3.2. *Learning by self-practice*

Question 16

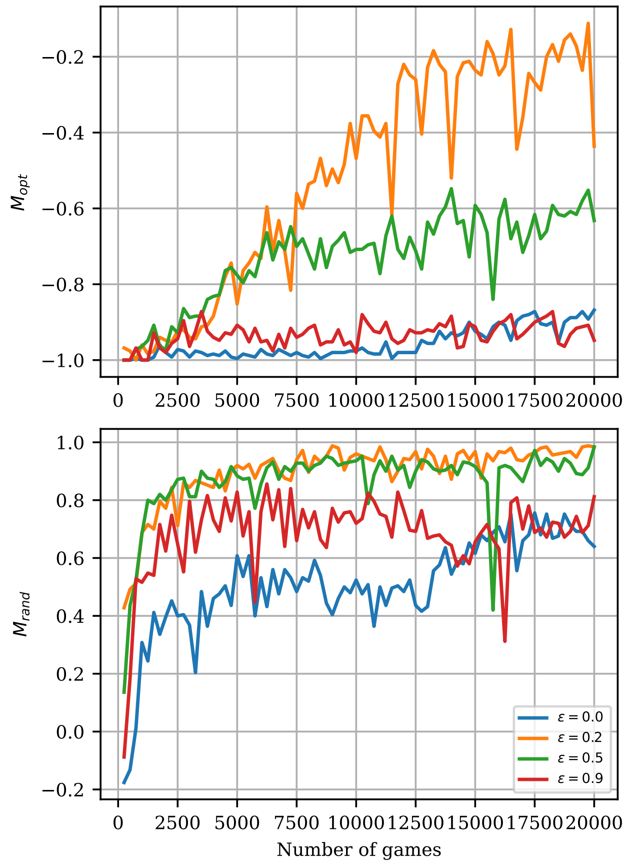

Figure: Evolution of M_{opt} and M_{rand}, varying ϵ.
Our self-learning deep agent is learning to play Nim relatively good with $\epsilon = 0.2$, but performances decrease for other values and notably also for $\epsilon = 0$, quite differently from what we observed in Q7. In this case, in fact, the network is not aware of the rules of the game, since we always ask it to pick the best move out of all 21 possibilities. Therefore, the agent needs to spend a lot of time in finding what moves are allowed, and $\epsilon \neq 0$ will speed up this process since in the random case only legal moves are sampled.

Question 17

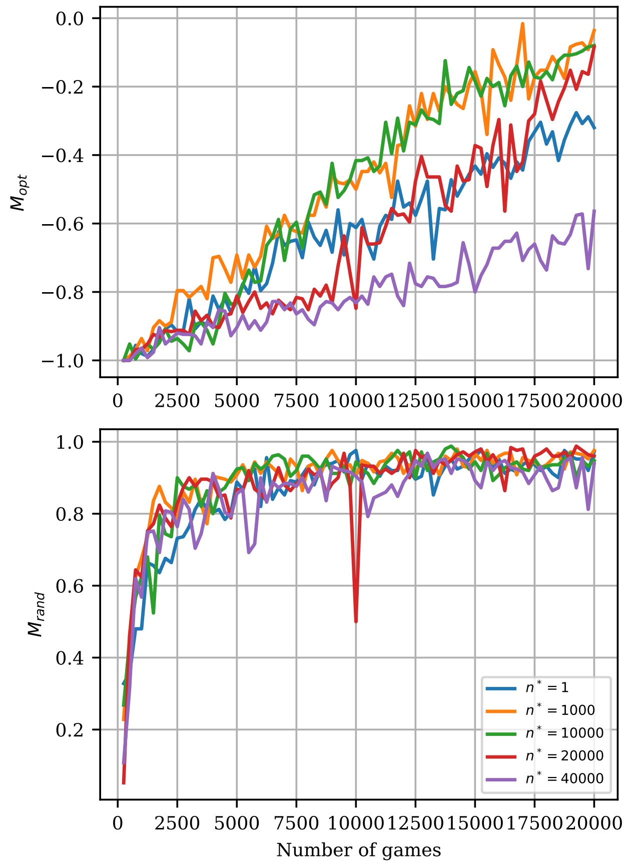

Figure: Evolution of M_{opt} and M_{rand}, varying n^*.
By decreasing ϵ, the performances seem overall much better, with peak results close to the theoretical maxima. In particular, it seems that even a very rapid decrease, with $n^* = 1000$, is helping substantially the network in its training. In fact, as argued in Q16, in DQN the agent has to learn by itself the rules of the game, and thus it is good for this to initially have a large random component (high ϵ). When legal actions have been learned, though, the situation is similar to what we had for Q-learning (cf. Q8), and it is better to maximize exploitation (low ϵ).

Question 18

For Q17, we obtain at best (pointwise) $M^*_{opt} = -0.016$, for $n^* = 1000$, and $M^*_{rand} = 0.988$, for $n^* = 10\,000$.

Question 19

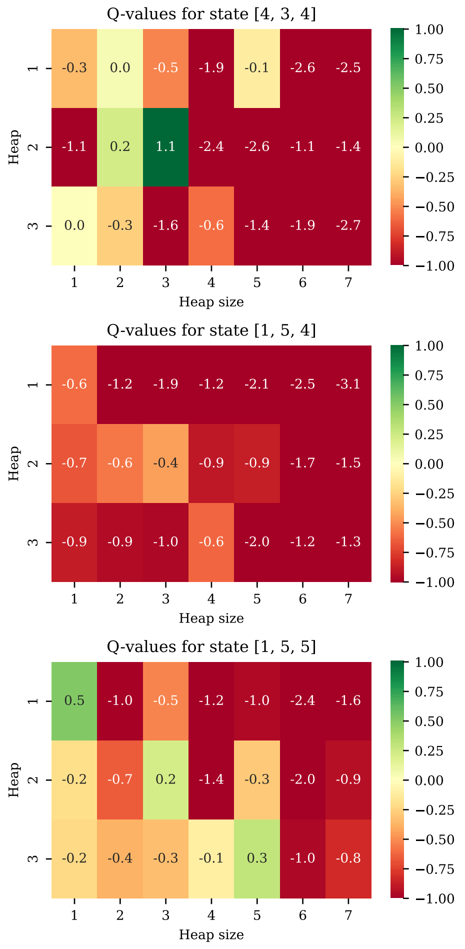

Figure: Q-values for the agent with $n^* = 1000$.

The Q-values of our agent seem to reflect the optimal behaviour, with the winning move having the maximum Q-value in states where the Nim sum is different from 0 (1st and 3rd), and Q-values being all negative where the Nim sum is already 0 (2nd). Interestingly, we observe that the agent is learning quite effectively the rules of the game, on average associating illegal moves with lower Q-values than for other losing moves. This is in fact logical, because legal losing moves could still result in a win when exploration is undergoing or in the initial phases of the training. Note also that the first state is the same as for Q10, with the same qualitative outcome. In addition, despite the fact that the second and third states differ in their 9-bits representation only by a single bit, their Q-value landscape is pretty different, showing that the network has enough representation power to learn remarkably far outcomes for similar states. Finally, we notice how it is impossible here to distinguish which states have been seen during training and which have not, since states are encoded by overlapping representations and the weights of the network are shared by all of them.

4. Comparing Q-Learning with Deep Q-Learning

Question 20

		M^*_{opt}	M^*_{rand}	T_{train}
Q-learning	Experts	0.092	0.996	7000
	Self-learning	0.048	1.000	6750
DQN	Experts	0.036	0.980	7000
	Self-learning	-0.016	0.988	14500

Table I: Performance of algorithms. T_{train} is quantized with steps of 250 games, the same frequency of plotting points.

Question 21

Overall, we observe that Q-learning and DQN, both with learning from experts or self-learning, achieve very similar final performances, with values of M^*_{opt} and M^*_{rand} close to the theoretical optima. However, the greatest difference between the two occurs in self-learning, where DQN requires much more time to converge and has a T_{train} more than two times the one of Q-learning. As already argued in Q17 and Q16, this happens because the DQN agent does not previously know the rules of the game, having thus to learn by himself which actions are allowed in addition to which actions are good. On the other hand, in Q-learning the choice is done only taking into account legal actions, and the agent manages to learn the game very quickly and effectively (cf. Q7 and Q8). Looking at the broader picture, it seems that the use of function approximations via neural networks with a shared input representation is not providing a considerable benefit, conversely introducing additional issues. In fact, while in general a low-dimensional (possibly continuous) representation of the input space helps as similar states often lead to similar decisions, in a game like Nim states that differ in a single stick can be remarkably unlike, going from losing to winning or with very different optimal actions. Consequently, the network can hardly make use of the knowledge coming from shared bits, and needs to internally learn a representation distinguishing between different states. In such a scenario, therefore, a tabular method such as Q-learning should be preferred, which is viable with a relatively low number of games since the total extended dimension of the input space is still low ($8^3 = 512$). Indeed, our experiments show that, despite similar final results, Q-learning leads in general to more stable and efficient training, with faster convergence to the optima.