Analyzing (Deep) Q-learning for Tic-Tac-Toe

Olav Førland, Benjamin Rike
EPFL, Switzerland

I. Q-LEARNING

A. Exercise 1

Fig. 1. The average reward for every 250th game during training using $\epsilon = 0.1$ against $\epsilon_{opt} = 0.5$.

We observe that the agent has a positive average reward after around 2500 games and reaches an average reward of around 0.4. Indeed, the agent learns to play Tic-Tac-Toe.

B. Exercise 2

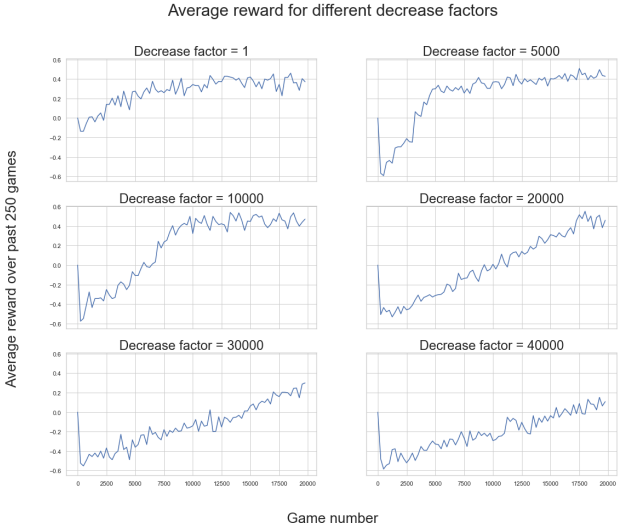

Fig. 2. The average reward for the past 250 games for varying decrease factors (n^*).

Decreasing ϵ during training does indeed seem to improve training. For a fixed ϵ, i.e. $n^* = 1$, the agent finds it best performance from game 7500 and onward, where it has an average reward of around 0.4. For $n^* \in \{5000, 10000, 20000\}$ the agent reaches its optimum progressively later, for the benefit of increased performance compared to the fixed case. For larger $n*$, the agent does not converge in 30 000 games. This seems to be because the convergence speed is reduced as $n*$ is increased. This makes intuitive sense as one uses longer time in the exploratory phase. As a side note, the effect of increasing n^* appears very similar to the effect of decreasing the learning rate; the agent converges slower, but seems to achieve better overall performance. For large n^*, this does not happen in this plot, but if we increased the number of games, we would expect large n^* to reach the same or higher level of performance.

C. Exercise 3

Fig. 3. M_{opt} and M_{rand} per 250th game for varying decrease factors ($n*$).

For the M_{opt} and M_{rand} curves we note a similar convergence trend as in the previous question. As the decrease factor increases, the algorithm spend more time converging. We also see that for low decrease factors, the agent learns to play optimal against the optimal player in a stable manner. In contrast to the previous question however, all decrease factors reaches approximately the same level of performance. I.e., for all decrease factors, the agent learns to play Tic-Tac-Toe.

D. Exercise 4

We observe that for increasing ϵ_{opt}, the algorithm performs better against the random player. This intuitively makes sense as a higher ϵ_{opt} makes the opponent behave more as a random player. That way, the agent trains against a random player and thereby learns how to beat a random player. For playing

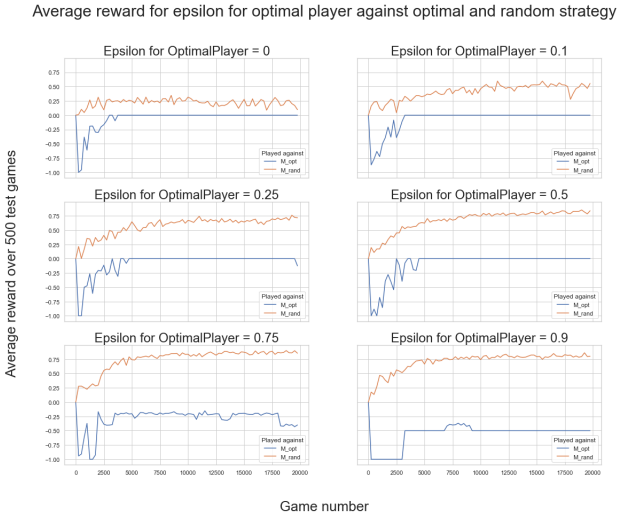

Fig. 4. M_{opt} and M_{rand} per 250th game for varying ϵ_{opt}.

against the optimal player, we see signs of the opposite trends. The agent learns to play optimal against an optimal players for all values, but it converges slower for $\epsilon_{opt} \in \{0.75, 0.9\}$ than for $\epsilon_{opt} \in \{0, 0.1, 0.25, 0.5\}$. This has the same explanation, for higher ϵ_{opt}, the agent has less training playing against optimal moves and therefore slower learns how to play against an optimal player. In summary, the agent learns how to win when training against a random player, while it learns how to not lose when training against the optimal player.

E. Exercise 5

The best value for M_{opt} was 0.000 and the best value for M_{rand} was 0.862 after 20 000 games. The agent used $n^* = 5000$ and $\epsilon_{opt} = 0.5$

F. Exercise 6

No, they do not have the same values. Consider the example in Figure 5 with the respective Q-values for agent 1 and agent 2. Agent 2 have higher Q-values for actions 3, 5, and 8 as $Opt(1)$ plays randomly. Agent 2 can therefore win even when it plays a move which $Opt(0)$ would win on. As the Q-values model the expected reward of performing an action given a state, the moves of the other player influences the expected reward of the agent. Different opposition thus leads to different expected rewards.

G. Exercise 7

In Figure 6 we observe that the agent reaches an average reward against the random player above 0.75 for $\epsilon \in \{0.1, 0.25, 0.5\}$. Against an optimal player however, the agent never seems to reach the optimum performance, i.e. 0. For $\epsilon = 0$, the agent does not reach a good performance, probably because it does not explore enough. Otherwise, the performance gets worse with higher ϵ.

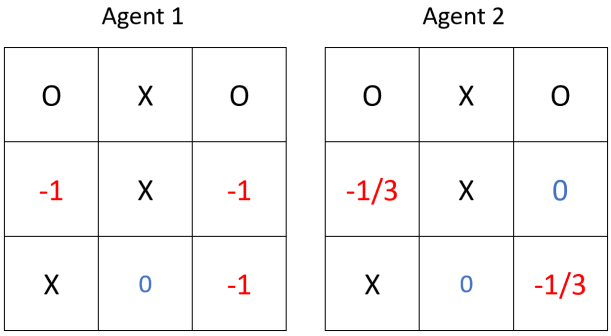

Fig. 5. An example where agent 1 and agent 2 have different Q-value for the same board configuration. The numbers indicate Q-values.

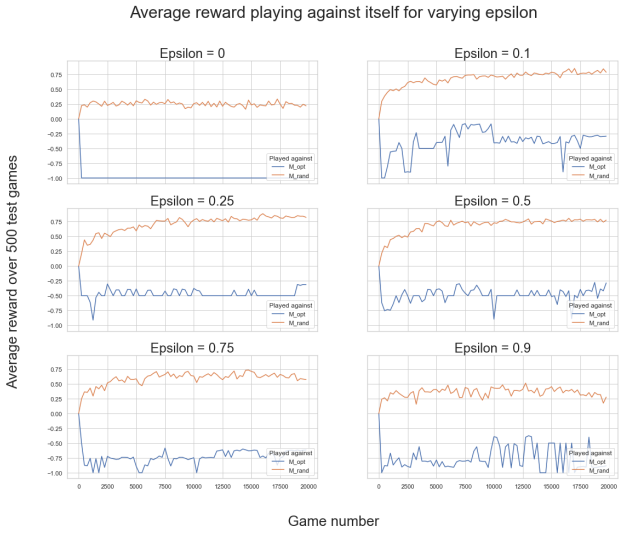

Fig. 6. M_{opt} and M_{rand} playing against itself for varying ϵ

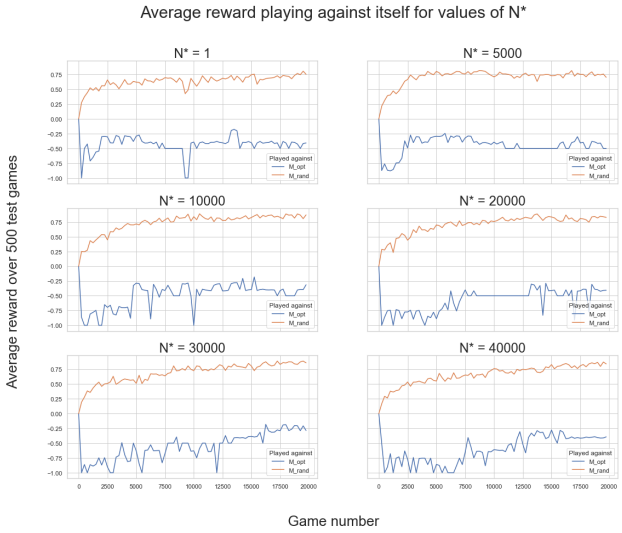

Fig. 7. M_{opt} and M_{rand} playing against itself for varying decrease factors (n^*)

H. Exercise 8

Decreasing ϵ seems to have the same performance as using a fixed but small ϵ. Figure 7 shows that the higher n^*, the slower the agent reaches strong performance.

I. Exercise 9

The best value for M_{opt} was 0.000 and the best value for M_{rand} was 0.876 after 20 000 games. The agent used decrease factor of 30 000 and $\epsilon_{opt} = 0.5$

J. Exercise 10

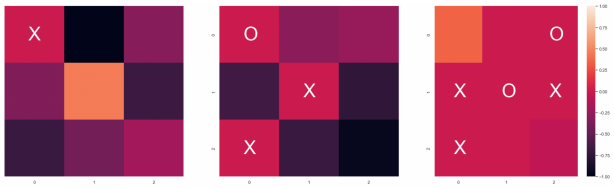

Fig. 8. Q-values for three board arrangements for the agent trained in exercise 9. Lighter values indicate higher Q-values.

From Figure 8, the agent identifies the best moves for each board configuration. However, there are some problems. Particularly, in the third configuration all other moves than the optimal move have a Q-value of around 0. However, these should be much lower, as placing the move in any of these boxes most likely results in immediate defeat. After inspecting the code, the model has only encountered one of these state-action pairs during training, hence why it defaults to the Q-value 0. This shows that further training is needed to achieve an agent that models the expected reward function accurately. To streamline the training process, we could have encoded the Q-values as V-values instead, i.e. only encoding the resulting state of a state-action pair. As several state-action pairs lead to the same state in Tic-Tac-Toe, this would effectively reduce the learned functions domain, leading to faster training. We chose not to do this as the project description explicitly asks for Q-values.

II. Deep Q-learning

A. Exercise 11

From Figure 9 the agent has a positive average reward after around 2500 games and converges to a reward between 0.5 and 0.6. Indeed, the agent seems to learn to play Tic-Tac-Toe. The loss jumps up and down, but this is because we update the Target net every 500th game which changes the values the loss is calculated relative to.

B. Exercise 12

Changing the batch size and memory buffer capacity to 1 leads to worse performance, visible in Figure 10. As the memory only remembers the last move of each game, it disregards every step up to that point.

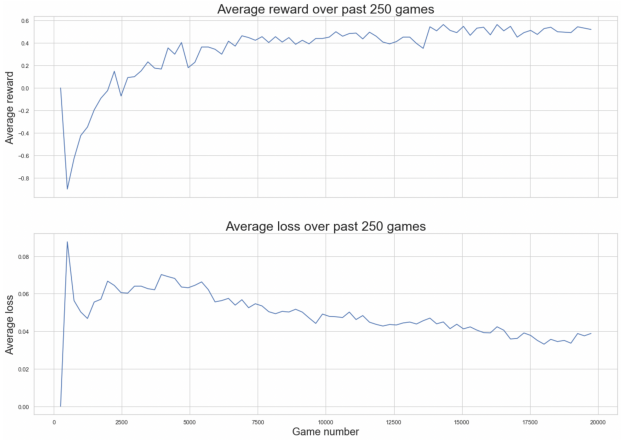

Fig. 9. The average reward and loss for every 250th game during training using $\epsilon = 0.1$ against $\epsilon_{opt} = 0.5$.

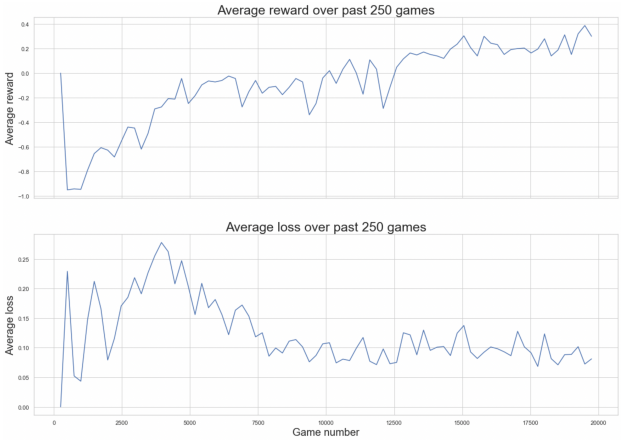

Fig. 10. Average reward and loss per 250 game during training. $\epsilon = 0.1$ against $\epsilon_{opt} = 0.5$

C. Exercise 13

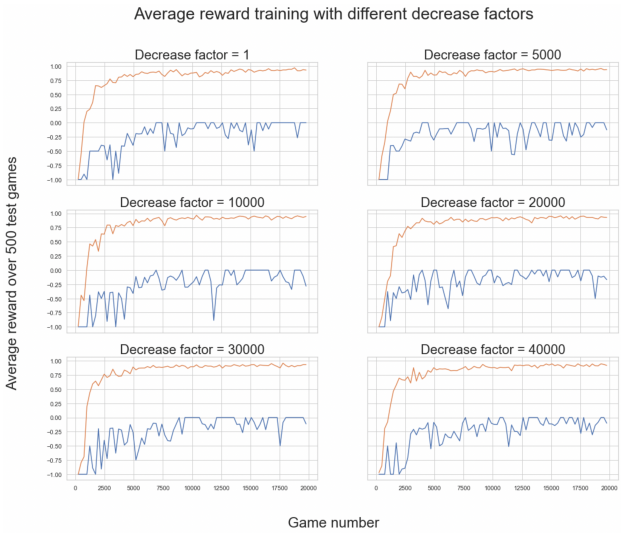

Fig. 11. M_{opt} and M_{rand} training with varying decrease factors n^*

Decreasing ϵ does not seem to improve training as the best performance is for decrease factor 1 which is equivalent to a constant ϵ. The reward versus a random player is similar for all decrease factors, but for higher decrease factors, the performance against the optimal player is more unstable.

D. Exercise 14

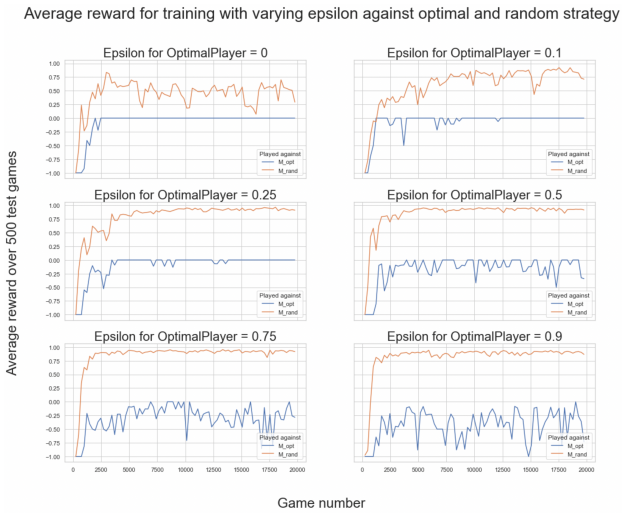

Fig. 12. M_{opt} and M_{rand} training with varying value for ϵ_{opt} using n^* = 1

We observe the same trade-off as in *Exercise 4*, lower ϵ_{opt} yields better performance against the optimal player, but worse performance against the random player. This is due to low values of ϵ_{opt} leading to the agent learning how to not lose against an optimal player, but not how to win against a random player. The opposite is true for higher values of ϵ_{opt}. In this case, it seems like $\epsilon_{opt} = 0.25$ is the correct trade-off.

E. Exercise 15

The best value for M_{opt} was 0.000 and the best value for M_{rand} was 0.962 after 20 000 games. The agent used $n^* = 1$ and $\epsilon_{opt} = 0.25$.

F. Exercise 16

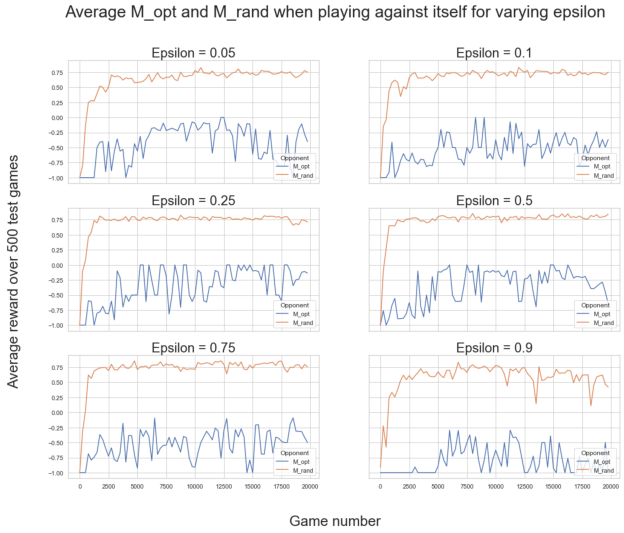

Fig. 13. M_{opt} and M_{rand} playing against itself for varying ϵ

The agent learns to play Tic-Tac-Toe against the random player in a good way, but does not achieve optimal performance against an optimal player. Interestingly, the average reward is stable against a random player, regardless of ϵ, while it is highly unstable against an optimal player. This makes the effect of ϵ hard to determine. It seems to perform best for $\epsilon = 0.25$, but not enough to convince us that it is not by chance.

G. Exercise 17

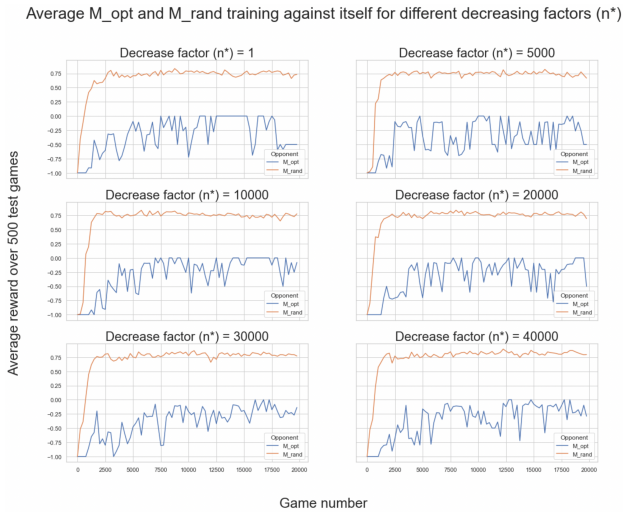

Fig. 14. M_{opt} and M_{rand} playing against itself for varying decrease factors

In the case of DQN training against itself, there does not seem to be any performance boost by decreasing ϵ vs using a fixed ϵ. There does not seem to be any specific effect from n^* either. As in the previous question, we reach strong performance against the random player, but not optimal

performance against the optimal player as there seems to be a lot of noise in the rewards. We therefore continue with a fixed epsilon.

H. Exercise 18

The best value for M_{opt} was 0.000 and the best value for M_{rand} was 0.812 after 20 000 games. The model was trained using $n^* = 1$ and $\epsilon_{opt} = 0.5$.

I. Exercise 19

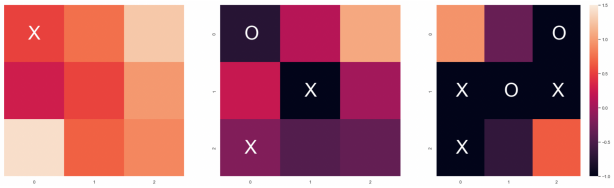

Fig. 15. Q-values for three board arrangements for the agent in exercise 18. Lighter color represents higher Q-values.

From Figure 15, we observe that the agent seems to correctly identify the best move in board two and three. It manages to play the winning move in board 3 and the only defending move in board 2. In board 1, the optimal move would be to place in the middle as that is the only defense strategy against an optimal player which does not end in defeat. However, the model prefers to place the tile in the corners, which is wrong.

Furthermore, in board 1 and board 2, we get Q-values larger than 1. This indicate that we have made some mistake in our implementation, as the expected reward cannot exceed 1. Another indication of this is that positions already occupied receive positive values when they should be negative. This problem is in states where there is only one occupied square (such as in board 1).

Even though the agent was able to identify the correct moves in board 2 and board 3, it gives us wrong Q-values for the actions. We therefore conclude that the agent hasn't properly learned Tic-Tac-Toe.

III. Q-learning vs. Deep Q-learning

A. Exercise 20

Agent	$\mathbf{M_{opt}}$	$\mathbf{M_{rand}}$	$\mathbf{T_{rand}}$	$\mathbf{T_{opt}}$
Q-learning from experts	0.000	0.864	3000	5000
Q-learning self-practice	-0.190	0.906	16250	4750
DQN from experts	0.000	0.962	2750	2750
DQN self-practice	0.000	0.812	3500	2750

TABLE I
THE BEST PERFORMANCE FOR THE VARIOUS MODELS AND THEIR CORRESPONDING TRAINING TIME

B. Exercise 21

The first thing to compare is the performance. Q-learning and DQN perform similar for the M_{opt} and M_{rand} values for both experts and self-practice. Q-learning is best for M_{rand} from self-practice while the DQN from experts have a higher Q-value for M_{opt}. Both Q-learning and DQN perform very good when learning for experts which shows that both Q-learning and DQN can teach an agent to play Tic-Tac-Toe.

For training time, we see more interesting differences. To reach 80% of the performance against the optimal player, DQN seem to converge around 2 times faster than normal Q-learning both for experts and self-practice. For the random player, it is only for self-practice we see the same difference. Additionally, while training the agents, we found the DQN to faster play 20 000 games than the normal Q-learning. This indicate that a strength of DQN vs. Q-learning is that we faster reach an acceptable performance.

Another element that we do not see in Table I is that Q-learning seem more robust to choice of hyperparameters. In exercise 3, we saw that for all n^*, the agent reached optimal performance against the optimal player. However, for DQN we first reach optimal performance against an optimal player when we also tune ϵ_{opt}.

The best performing agent overall was DQN from experts. It achieved the best score on all four criteria, and is the agent we would choose if we were to bet our money on a RL agent playing Tic-Tac-Toe.