Mini-project 1: Tic Tac Toe

1 Introduction

What is the aim?

In our 1st mini-project, we use Q-Learning and Deep Q-Learning to train artificial agents that can play the famous game of Tic Tac Toe. Interestingly, the optimal policy for Tic Tac Toe is known, so our aim is to answer these three questions:

1. Can RL algorithms learn to play Tic Tac Toe by playing against optimal policy?
2. Can an RL algorithm learn to play Tic Tac Toe only by playing against itself?
3. What are the pros and cons of Deep Q-Learning compared to (Tabular) Q-Learning?

Environment and optimal policy:

You are given a Python script (`tic_env.py`) and a Jupyter notebook (`tic_tac_toe.ipynb`). The script contains an implementation of the environment of Tic Tac Toe and an ϵ-greedy version of its optimal policy, and the notebook presents a brief tutorial on how to use the environment and optimal policy. We refer to the ϵ-greedy optimal policy by `Opt(`ϵ`)`.

Performance measures:

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

- M_{opt} measures the performance of π against the optimal policy. To compute M_{opt}, we run π against `Opt(0)` for $N = 500$ games – for different random seeds. π makes the 1st move in 250 games, and `Opt(0)` makes the 1st move in the rest. We count how many games π wins (N_{win}) and loses (N_{los}) and define $M_{\text{opt}} = \frac{N_{\text{win}} - N_{\text{los}}}{N}$.
- 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)`.

For a good policy, M_{opt} is close to 0 and M_{rand} is close to 1.

What should you submit? Only one submission per group: A single `zip` file containing

1. Report (in `pdf` format):
   Report should be at most 5 pages. It should include the results of your analyses in answers to the questions below. Please respect the word limit for each question and choose a reasonable size for your figures.
2. Jupyter notebook (in `ipynb` format):
   The Jupyter notebook should contain the code for reproducing your results. The code should be well-documented and readable.

Naming format: `[Sciper number of student 1]_[Sciper number of student 2].zip`.

When should you submit? You have two options:

1. Submit by **May 30** (before 23:30) and give the fraud detection interview on June 2 or June 3.
2. Submit by **June 6** (before 23:30) and give the fraud detection interview on June 9 or June 10.

2 Q-Learning

As our 1st algorithm, we use Q-Learning combined with ϵ-greedy policy – see section 6.5 of Sutton and Barto (2018) for details. At each time t, state s_t is the board position (showing empty positions, positions taken by you, and positions taken by your opponent; c.f. `tic_tac_toe.ipynb`), action a_t is one of the available positions on the board (i.e. ϵ-greedy is applied only over the available actions), and reward r_t is only non-zero when the game ends where you get $r_t = 1$ if you win the game, $r_t = -1$ if you lose, and $r_t = 0$ if it is a draw.

Q-Learning has 3 hyper-parameters: learning rate α, discount factor γ, and exploration level ϵ. For convenience, we fix the learning rate at $\alpha = 0.05$ and the discount factor at $\gamma = 0.99$. We initialize all the Q-values at 0; if you are curious, you can explore the effect of α, γ, and initial Q-values for yourself.

2.1 Learning from experts

In this section, you will study whether Q-learning can learn to play Tic Tac Toe by playing against `Opt(`ϵ_{opt}`)` for some $\epsilon_{\text{opt}} \in [0, 1]$. To do so, implement the Q-learning algorithm. To check the algorithm, run a Q-learning agent, with a fixed and arbitrary $\epsilon \in [0, 1)$, against `Opt(0.5)` for 20'000 games – switch the 1st player after every game.

Question 1. Plot average reward for every 250 games during training – i.e. after the 50th game, plot the average reward of the first 250 games, after the 100th game, plot the average reward of games 51 to 100, etc. Does the agent learn to play Tic Tac Toe?
Expected answer: A figure of average reward over time (caption length < 50 words). Specify your choice of ϵ.

2.1.1 Decreasing exploration

One way to make training more efficient is to decrease the exploration level ϵ over time. If we define $\epsilon(n)$ to be ϵ for game number n, then one feasible way to decrease exploration during training is to use

$$\epsilon(n) = \max\{\epsilon_{\min}, \epsilon_{\max}(1 - \frac{n}{n^*})\}, \tag{1}$$

where $\epsilon_{\min}$ and $\epsilon_{\max}$ are the minimum and maximum values for ϵ, respectively, and n^* is the number of exploratory games and shows how fast ϵ decreases. For convenience, we assume $\epsilon_{\min} = 0.1$ and $\epsilon_{\max} = 0.8$; if you are curious, you can explore their effect on performance for yourself. Use $\epsilon(n)$ as define above and run different Q-learning agents with different values of n^* against `Opt(0.5)` for 20'000 games – switch the 1st player after every game. Choose several values of n^* from a reasonably wide interval between 1 to 40'000 – particularly, include $n^* = 1$.

Question 2. Plot average reward for every 250 games during training. Does decreasing ϵ help training compared to having a fixed ϵ? What is the effect of n^*?
Expected answer: A figure showing average reward over time for different values of n^* (caption length $<$ 200 words).

Question 3. After every 250 games during training, compute the 'test' M_{opt} and M_{rand} for your agents – when measuring the 'test' performance, put $\epsilon = 0$ and do not update the Q-values. Plot M_{opt} and M_{rand} over time. Describe the differences and the similarities between these curves and the ones of the previous question.
Expected answer: A figure showing M_{opt} and M_{rand} over time for different values of n^* (caption length $<$ 100 words).

2.1.2 Good experts and bad experts

Choose the best value of n^* that you found in the previous section. Run Q-learning against `Opt(`ϵ_{opt}`)` for different values of ϵ_{opt} for 20'000 games – switch the 1st player after every game. Choose several values of ϵ_{opt} from a reasonably wide interval between 0 to 1 – particularly, include $\epsilon_{\text{opt}} = 0$.

Question 4. After every 250 games during training, compute the 'test' M_{opt} and M_{rand} for your agents – for each value of ϵ_{opt}. Plot M_{opt} and M_{rand} over time. What do you observe? How can you explain it?
Expected answer: A figure showing M_{opt} and M_{rand} over time for different values of ϵ_{opt} (caption length $<$ 250 words).

Question 5. What are the highest values of M_{opt} and M_{rand} that you could achieve after playing 20'000 games?

Question 6. (Theory) Assume that Agent 1 learns by playing against `Opt(0)` and find the optimal Q-values $Q_1(s, a)$. In addition, assume that Agent 2 learns by playing against `Opt(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? Justify your answer. (answer length $<$ 150 words)

2.2 Learning by self-practice

In this section, your are supposed to ask whether Q-learning can learn to play Tic Tac Toe by only playing against itself. For different values of $\epsilon \in [0, 1)$, run a Q-learning agent against itself for 20'000 games – i.e. both players use the same set of Q-values and update the same set of Q-values.

Question 7. After every 250 games during training, compute the 'test' M_{opt} and M_{rand} for different values of $\epsilon \in [0, 1)$. Does the agent learn to play Tic Tac Toe? What is the effect of ϵ?
Expected answer: A figure showing M_{opt} and M_{rand} over time for different values of $\epsilon \in [0, 1)$ (caption length $<$ 100 words).

For rest of this section, use $\epsilon(n)$ in Equation 1 with different values of n^* – instead of fixing ϵ.

Question 8. After every 250 games during training, compute the 'test' M_{opt} and M_{rand} for your agents. Does decreasing ϵ help training compared to having a fixed ϵ? What is the effect of n^*?
Expected answer: A figure showing M_{opt} and M_{rand} over time for different values of speeds of n^* (caption length $<$ 100 words).

Question 9. What are the highest values of M_{opt} and M_{rand} that you could achieve after playing 20'000 games?

Question 10. For three board arrangements (i.e. states s), visualize Q-values of available actions (e.g. using heat maps). Does the result make sense? Did the agent learn the game well?
Expected answer: A figure with 3 subplots of 3 different states with Q-values shown at available actions (caption length $<$ 200 words).

3 Deep Q-Learning

As our 2nd algorithm, we use Deep Q-Learning (DQN) combined with ϵ-greedy policy. You can watch again Part 1 of Deep Reinforcement Learning Lecture 1 for an introduction to DQN and Part 1 of Deep Reinforcement Learning Lecture 2 (in particular slide 8) for more details. The idea in DQN is to approximate Q-values by a neural network instead of a look-up table as in Tabular Q-learning. For implementation, you can use ideas from the DQN tutorials of `Keras` and `PyTorch`.

3.1 Implementation details

State representation: We represent state s_t by a $3 \times 3 \times 2$ tensor `x_t`. Each element of `x_t` takes a value of 0 or 1. The 3×3 matrix `x_t[:,:,0]` shows positions taken by you, and `x_t[:,:,1]` shows positions taken by your opponent. If `x_t[i,j,0] = x_t[i,j,1] = 0`, then position (i, j) is available.

Neural network architecture: We use a fully connected network. State `x_t` is fed to the network at the input layer. We consider 2 hidden layers each with 128 neurons – with ReLu activation functions. The output layer has 9 neurons (for 9 different actions) with linear activation functions. Each neuron at the output layer shows the Q-value of the corresponding action at state `x_t`.

Unavailable actions: For DQN, we do not constraint actions to only available actions. However, whenever the agent takes an unavailable action, we end the game and give the agent a negative reward of value $r_{\text{unav}} = -1$.

Free parameters: DQN has many hyper parameters. For convenience, we fix the discount factor at $\gamma = 0.99$. We assume a buffer size of 10'000 and a batch size of 64. We update the target network every 500 games. Instead of squared loss, we use the Huber loss (with $\delta = 1$) with Adam optimizer (c.f. the DQN tutorials of `Keras` and `PyTorch`). You can fine tune the learning rate if needed, but we suggest 5×10^{-4} as a starting point.

Other options? There are tens of different ways to make training of deep networks more efficient. Do you feel like trying some and learning more? You are welcome to do so; you just need to explain the main features of your implementation and a brief summary of your reasoning in less than 300 words – under the title 'Implementation details' in your report.

3.2 Learning from experts

Implement the DQN algorithm. To check the algorithm, run a DQN agent with a fixed and arbitrary $\epsilon \in [0, 1)$ against `Opt(0.5)` for 20'000 games – switch the 1st player after every game.

Question 11. Plot average reward and average training loss for every 250 games during training. Does the loss decrease? Does the agent learn to play Tic Tac Toe?
Expected answer: A figure with two subplots (caption length < 50 words). Specify your choice of ϵ.

Question 12. Repeat the training but without the replay buffer and with a batch size of 1: At every step, update the network by using only the latest transition. What do you observe?
Expected answer: A figure with two subplots showing average reward and average training loss during training (caption length < 50 words).

Instead of fixing ϵ, use $\epsilon(n)$ in Equation 1. For different values of n^*, run your DQN against `Opt(0.5)` for 20'000 games – switch the 1st player after every game. Choose several values of n^* from a reasonably wide interval between 1 to 40'000 – particularly, include $n^* = 1$.

Question 13. After every 250 games during training, compute the 'test' M_{opt} and M_{rand} for your agents. Plot M_{opt} and M_{rand} over time. Does decreasing ϵ help training compared to having a fixed ϵ? What is the effect of n^*?
Expected answer: A figure showing M_{opt} and M_{rand} over time for different values of speeds of n^* (caption length < 250 words).

Choose the best value of n^* that you found. Run DQN against `Opt(`ϵ_{opt}`)` for different values of ϵ_{opt} for 20'000 games – switch the 1st player after every game. Choose several values of ϵ_{opt} from a reasonably wide interval between 0 to 1 – particularly, include $\epsilon_{\text{opt}} = 0$.

Question 14. After every 250 games during training, compute the 'test' M_{opt} and M_{rand} for your agents

– for each value of ϵ_{opt}. Plot M_{opt} and M_{rand} over time. What do you observe? How can you explain it?
Expected answer: A figure showing M_{opt} and M_{rand} over time for different values of ϵ_{opt} (caption length < 250 words).

Question 15. What are the highest values of M_{opt} and M_{rand} that you could achieve after playing 20'000 games?

3.3 Learning by self-practice

For different values of $\epsilon \in [0, 1)$, run a DQN agent against itself for 20'000 games – i.e. both players use the same neural network and share the same replay buffer. *Important note:* For one player, you should add states s_t and $s_{t'}$ as `x_t` and `x_tp` to the replay buffer, but for the other player, you should first swap the opponent positions (`x_t[:,:,1]` and `x_tp[:,:,1]`) with the agent's own positions (`x_t[:,:,0]` and `x_tp[:,:,0]`) and then add them to the 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 M_{opt} and M_{rand} over time. Does the agent learn to play Tic Tac Toe? What is the effect of ϵ?
Expected answer: A figure showing M_{opt} and M_{rand} over time for different values of $\epsilon \in [0, 1)$ (caption length < 100 words).

Instead of fixing ϵ, use $\epsilon(n)$ in Equation 1 with different values of n^*.

Question 17. After every 250 games during training, compute the 'test' M_{opt} and M_{rand} for your agents. Plot M_{opt} and M_{rand} over time. Does decreasing ϵ help training compared to having a fixed ϵ? What is the effect of n^*?
Expected answer: A figure showing M_{opt} and M_{rand} over time for different values of speeds of n^* (caption length < 100 words).

Question 18. What are the highest values of M_{opt} and M_{rand} that you could achieve after playing 20'000 games?

Question 19. For three board arrangements (i.e. states s), visualize Q-values of available actions (e.g. using heat maps). Does the result make sense? Did the agent learn the game well?
Expected answer: A figure with 3 subplots of 3 different states with Q-values shown at available actions (caption length < 200 words).

4 Comparing Q-Learning with Deep Q-Learning

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}.

Question 20. Include a table showing 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.
Expected answer: A table showing 12 values.

Question 21. Compare your results for DQN and Q-Learning (answer length < 300 words).