Inlämningsuppgift (final project): Tic-Tac-Toe

Description of the problem:

The famous tic-tac-toe game is played on a 3x3 grid, between two players who sequentially fill the so far empty spaces of this grid with O and X marks (O for the start player, X for the other player). A player who placed three of their marks in a row, column, or on the diagonal wins. See https://en.wikipedia.org/wiki/Tic-tac-toe for instructions and explanations of this game. The question is now simply "what is the best strategy" – meaning that we are looking for an algorithm or data structure that returns the optimal move given one of the many possible states the board can be in.

Of course, there is the minimax algorithm, which you will have experienced in this course. Here we are asking you to use reinforcement learning to train a policy to play tic tac toe optimally, and then demonstrate the ability of your solution to play against a random strategy, and a perfect strategy derived from minimax (solution provided).

Part 1: **Create the proper computational framework to play, test, and evaluate tic-tac-toe**

We will start doing this in the first lab as a training exercise, but in the end, you need code to play tic-tac-toe between two humans, play a human versus a random strategy, and play a human versus a "policy" - we will describe in detail what this is in the course. This includes functions that create a functional board, make moves, and evaluate the board for example.

This part in itself does not provide any credits but is necessary for the second part.

Part 2: **Train a policy to play tic-tac-toe as perfectly as possible**

In this part, you will need to play tic-tac-toe policies against each other and use their experience to train a policy to play as optimally as possible. As you will see, this might take some time, and due to randomness, you might not even get to 100% performance. Thus, illustrating that your method works, improving performance significantly and over time would arrive at the optimal solution is sufficient. For this, you test your improving policy versus a random player and the perfect strategy during training and plot performance over training time, illustrating the progression of your training method. We will provide a "perfect strategy policy" you should use to confirm that your policy's performance improves over time. You don't have to use this one specifically, as long as you test your trained policy against another perfect opponent.

While there is a "textbook" solution for this problem, using the Bellman equation, other derivative solutions are also possible, and sometimes even faster, easier, or "cooler" – we will accept all those, as long as they resemble reinforcement learning in principle. As a challenge, you might even consider deep-q learning.

Submission Instructions:

You can work on this inlämningsuppgift in groups of 2. When submitting, **each group member** should submit:

1. The code.
2. A video recording of how you went about the solution, and explaining your code, and where things happen in your project. Either you make two separate videos (one per student) or both explain the code in equal proportions, sufficient that we

can evaluate you. I typically say 5 minutes long - after all I end up watching 90 videos - but that might be a bit short, so 10 minutes is fine, be reasonable if you have to go over.

Trying to explain what you did in a comprehensive manner is part of the education, doing this recording a couple of times until you nail it, is an expected behavior. However, don’ t make 20 repeats just to get it under 10 minutes. This is a loose cutoff and not strict.

3. Also mention, in a note, what grade you are aiming for and who you worked with.

Grades:

For passing (G) you need to explain how reinforcement learning works in the context of a tic-tac-toe policy and implement code that trains a tic-tac-toe policy improving its performance against a random and a perfect policy (provided).

To receive a VG, your code should be concise (short and to the point), well documented, and your explanation must be excellent. Using a novel approach, original idea, optimizing a parameter for the method you use, creative use of code, or information, etc. is a bonus and if properly explained also warrants a VG.

You can of course use all material/code created during labs, as well as, libraries that support your efforts like numpy, matplotlib, etc. However, you are not allowed to use libraries that solve the problem for you. If you attempt deep-q learning, you can use tensorflow, pytorch, or keras.

OBSERVE: we have seen all tutorials on reinforcement learning and tic-tac-toe ourselves, copying one of them or recapitulating an existing solution is considered plagiarism (see below). The same is true for using solutions from other students.

Plagiarism:

There is a difference between reading a couple of tutorials followed by implementing a similar solution yourself, and copy/pasting code. The first is okay, the second constitutes plagiarism, as well as, copying code from other students. Observe that we know those tutorials as well, and it is extremely easy for us to spot plagiarism. Plagiarism must be reported by us and can lead to being expelled from the University!