Final Checkpoint

Github link: https://github.com/hongannie/aigaming

I. How much did you achieve?

We have achieved implementing a tic-tac-toe game leveraging a reinforcement learning model to train and utilize an AI agent that takes the step that maximizes its own reward and minimizes the reward of the opponent.

The game has the option to play in 3 different modes:

1. Between two users
2. Between two AI agents
3. Between a user and an AI agent

The reinforcement learning problem consists of three components: the state, action and reward sets. The state set refers to the current state of the board, the action set refers to the set of all options the player can take, and the reward refers to a value between -1 and 1 and is assigned at the end of the game (0 if player 2 wins, 0.5 if tie, 1 if player 1 wins). The initial state is set by a 3x3 board of empty strings. As players move, the board is updated with an 'x' or 'o' in the space depending on their turn order first or second respectively. After each action, the game uses the check_winner function to continuously check if the game has ended. If the game has ended, the check_winner function judges the resulting winner, returning True if the game has ended or False if it has not.

The agent (AI) is set up to choose actions based on the current board state, record all states of the game, update the states-value estimation after each game and save and load the policy, which is a function that maps from the state to action set. The optimal policy maximizes future rewards in each state. The state value is updated using the below equation:

$$v(s) = v(s) + \alpha(v(s')+R-v(s))$$

The equation indicates that the updated value of the state equals the current value of the state adding the difference between the value of the next state and the value of the current state, which is multiplied by a learning rate α. After setting up the value update, we are able to train the agent by having two agents play against each other a certain number of times. We implement this training in a separate training.py file so that the agent can learn and save the policy to then be used against a human player.

II. How many lines of codes?

We have about 400 lines of code in total. Our code consists of 2 parts:

1. training.py → This is python code that trains the AIs to learn the best choice of action after multiple games based on a reward system. This code contains functions that simulates entire games between two AI agents. The method of reinforcement learning used is temporal difference learning. Each state has a value attached to it. Once the game is started, the agent computes all possible actions it can take in the current state and the new states that would result from each action. The states are collected in a states_value vector that contains values for all possible states in the games in a add_states function.

The rewards are stored in the p1/2states_value dictionary. These updated states help the agent choose the best course of action based on the available positions after learning previously from prior games.

2. Tictactoe.py → This python code simulates the game based on the number of players. When the game is first initiated, it will ask the user to input the number of user players who will be playing.
    a. Single player → This python code simulates a game between an AI agent and a user. This code takes input from the user who chooses either X or O. If the user chooses X, the user goes first. The simulation asks the user to select a position,represented by a number between 1-9, that they would like to mark their play. When the user makes a play, the AI agent will then use what it has learned from the training to choose a choice of action based that will provide the highest reward if possible.
    b. Two player → This python code simulates a game between two users. The code will ask each user to select a number value representing the positions on the board where they would like to place a mark and switch turns after each action until someone wins.
    c. For the 'human' players, you enter where you want to put your symbol via this system:

1	2	3
4	5	6
7	8	9

III. Lessons learned?

We changed our implementation from using a dynamic programming approach to using a reinforcement learning approach to better create an AI that makes the best possible decision and is more similar to how a real human being would play tic-tac-toe. From this experience, we learned to more thoroughly explore the options to tackle a task and be open to different solutions that may better fit the situation. Rather than using complex mathematical deduction and exploration, we learned to look for a way to solve the tic-tac-toe problem of finding the best possible move via a simple trial and learn.

Reinforcement learning makes the decision between exploitation vs. exploration. Exploitation allows us to choose the action that maximizes the reward which is what we are trying to achieve through the training and reward system. However, there are certain scenarios where we cannot choose an action regardless of the reward it provides which is exploration. There has to be a good balance between the two for the agent to make the best decisions. This was done through the epsilon-decreasing strategy where the variable epsilon was initialized to 0.3. If probability = epsilon, then we choose to explore. If probability = 1- epsilon, then we choose to exploit. We would decrease the value of epsilon over time until it becomes zero. This is a way for the agent to

explore better actions during earlier stages of training and exploit the best actions in later stages of the game.

This method not only is easier to understand, but also, as previously mentioned, more accurate to how humans learn and play tic-tac-toe. Thus, the game becomes more enjoyable for the human player, as the opponent is more challenging.

IV. Analysis of important algorithms

The program was designed for the AI agent to make choices in order to win if it is perceived to be ahead of its competitor or to draw a tie if the opponent is perceived to be ahead. Player X is typically the first to move, so statistically, player X is ahead most of the time while Player O is statistically usually behind. So if the AI agent is playing X, the agent will most likely move to win. If the agent is playing O, the agent will most likely move to draw.
The check_winner function is meant to analyze a 2D array and determine the winner of the game. All the possible (8) winning outcomes are already stored and the existing board is compared against the winning board outcomes. The time complexity is O(1). The training function is linear O(n) depending on the amount of trainings/rounds you choose to execute.

V. Sources

1. Agrawal, R. (2020, April 15). *Playing Games with Python-Part 1: Tic Tac Toe*. Medium. https://towardsdatascience.com/lets-beat-games-using-a-bunch-of-code-part-1-tic-tac-toe-1543e981fec1.
2. Choudhary, A. (2020, May 24). *Dynamic Programming In Reinforcement Learning*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2018/09/reinforcement-learning-model-based-planning-dynamic-programming/.
3. Paszke, A. (n.d.). *Reinforcement Learning (DQN) Tutorial¶*. Reinforcement Learning (DQN) Tutorial - PyTorch Tutorials 1.8.1+cu102 documentation. https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html.