CSC111 Project Proposal: A Reinforcement Learning Approach to Connect 4

Elsie Chan, Annie Tan, York Hay Ng

Tuesday, March 8, 2023

Problem Description and Research Question

Our project aims to develop an AI model that can play Connect 4 through Q-learning and effectively compete against human players.

Connect 4 is a two-player strategy game where players take turns placing coloured tokens into a 7 by 6 grid. The tokens fall vertically to the lowest possible space in any given column. The objective of Connect 4 is to be the first to form a horizontal, vertical, or diagonal line using 4 of their own coloured tokens. For the purposes of our project, we will add an additional way to win: when a player's coloured tokens form a wrap-around horizontal line in the grid.

Q-learning is a type of model-free reinforcement learning algorithm that operates by learning the value of an action at a particular state. The algorithm maintains a Q-table that contains the expected quality or reward of all possible action-state pairs. The agent then selects actions that maximise its expected reward over time. Through training the agent, the agent will learn to select actions that maximise its expected reward, resulting in an optimised algorithm for decision-making. [1]

To calculate the Q-value in a given state S_t applying an action a_t, ie $Q_{new}(S_t, a_t)$, we must first define α - the learning rate, which determines how strongly a new Q-value would affect the existing value, and γ - discount, which depicts the extent of importance of immediate reward r_t over future rewards. The action a_t with the highest Q-value is the optimal action. [1]

$$Q_{new}(S_t, a_t) = (1 - \alpha)Q(S_t, a_t) + \alpha(r_t + \gamma max_a Q(S_{t+1}, a))$$

We chose this project goal as we were inspired by CSC111's Assignment 2. We want to use the concept of a game tree to develop a more comprehensive AI for playing a mathematical game. Connect 4 is a strategic yet relatively complex game, and therefore, developing an AI model that can play Connect 4 would involve understanding the game's rules and potential strategies.

Additionally, we believe that this could be a fun and engaging project to see if we can develop an AI model that could challenge human players.

In conclusion, the project question that we want to answer is:

How can we use Q-learning to train an AI model that plays Connect 4 with a decent performance, especially compared to human players?

Computational Plan

Game Class

The game class will be responsible for the logic involving the Connect Four game itself. It will keep track of a NumPy ndarray (simulating a matrix) which will represent the state of the board, i.e. which player has placed which disks in which spots. Players will interact with the game through two methods: one for processing players' moves, by updating the game board, and another for checking for winners, by checking for four consecutive disks in horizontal, vertical or diagonal arrangements.

GraphPlot Class

We plan to use plotly to implement methods to plot graphs to analyse the AI's performance against a TrainingAgent instance or a trained AIPlayer instance. We plan to use the data analysis to find trends between the AI's performance and other variables such as learning rate, discount factor, etc, so as to optimise our AI's performance.

We plan to use plotly graph objects module (2) to visualise the outcomes of all the games played or to visualise the player or AI player win percentage by using functions like go.Scatter(). Additionally, we can use px.line() to create line charts or px.scatter() to create scatter plots, and px.bar() to create bar charts to visualise the number of wins or losses by the AI player. We can also use these functions to visualise the AI's q-values for a set of learning rate or discount factor values and find the optimal parameters. We also plan to use matrix heatmaps to visualise the game board and for example, the player's optimal first move. With Plotly Express, we can also animate these graphs to demonstrate how the performance of the AI changes over time.

Game Interface Class

We will display the current game using the information from the Game class (such as the players' moves and checking for winners) and use pygame to provide a GUI (Graphical User Interface) for the human player to interact with the AI model.

We plan to use functions like pygame.draw() to create the game board and the tokens, pygame.event.get() to handle the events that occur in the game, and pygame.display.update() to update the game board. Additionally, there are several useful pygame.event types like MOUSEBUTTONDOWN and MOUSEMOTION to allow the player to visualise and place their tokens, and QUIT to end the game. (3)

GameTree Class

The GameTree class is a tree representing all the possible states in the game. We define a State S_t as a game instance right after the opponent places a move, and an action a_t as a tuple indicating the board coordinates where the AI wishes to place the next move. The root of each tree (_curr_state) will be a Game instance and the subtrees (_possible_states) will be a dict where each key is a possible move, mapped to a Game instance where that move has been recorded. An additional attribute (q_values) will store the Q-value $Q(S_t, a_t)$ of each unoccupied position on _curr_state, with a default value defined by another attribute (initial_q). This class will include methods for adding subtrees and modifying Q-values in specific positions. Each leaf node will represent a finished game instance.

Player Class

This class consists of instance attributes indicating whether a certain Player instance is player 1 or 2 and the Game instance this player is playing on. It also has a method for placing a move on the Game instance. The human player should be implemented as an instance of this class.

AIPlayer Class

The AIPlayer class inherits from the Player class. It has additional attributes move_sequence, which is a list of GameTrees representing all game states throughout the game, and other attributes representing learning rate, discount, and exploration probability. This class contains a method for making a move based on the largest q-value in the current state. If the AIPlayer decides to explore or cannot find a move in the q-table, then it randomly picks a move and creates new GameTree(s) based on the move. There will also be a method for training the AIPlayer with any agent, first as player 1 then as player 2, while utilizing GraphPlot to display the training data.

TrainingAgent Class

The main purpose of this class is for training an AIPlayer instance. It includes all the attributes and methods of the Player class. To pick the next move, it reads one step ahead. In other words, if it sees an immediate win or lose, it will place a move that allows it to win or prevents its opponent from winning. Otherwise, it picks a move randomly.

References

1) Tic-tac-toe with tabular Q-learning. Nested Software. (2019, July 25). Retrieved March 6, 2023, from https://nestedsoftware.com/2019/07/25/tic-tac-toe-with-tabular-q-learning-1kdn.139811.html

2) Badr Mario, Liu David, and Bernuy Angela Z. (2023) CSC111 Assignment 2 source code. Retrieved March 6, 2023.

3) Pygame Documentation. Pygame Front Page - pygame v2.1.4 documentation. (n.d.). Retrieved March 7, 2023, from https://www.pygame.org/docs/