Reinforcement Learning based CONNECT 4

Tarun Srinivasan
MS Robotics
Northeastern University

Husain Gittham
MS Robotics
Northeastern University

1 Introduction

For this project, I propose to build and implement two Reinforcement Learning algorithms for the classic game of Connect 4. The game is pretty simple in the high level, with a 6 x 7 grid space into which we drop coins to try to form a continuous line of 4 coins. It becomes a bit complicated due to the constraints brought to you by the opponent who basically tries to win and also hinder your chances of winning.

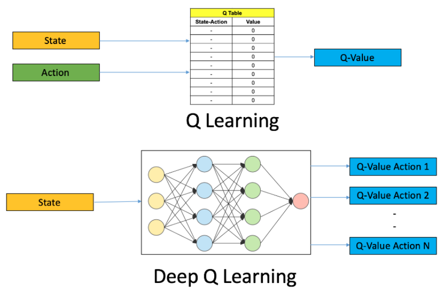
The idea behind this project is to learn to implement two or more (if time permits) and evaluate the compromises each algorithm has toward the result. I plan to choose one model-free and a model-based approach to cover both aspects of the problem. And for this the initial plan is to use **Deep Q Learning** as the model-free algorithm and **Monte Carlo Tree Search (MCTS)** for the model-based approach.

Both the models have been applied to various games with varying state and action spaces. The exploratory advantages for MCTS has been recorded across various literature. The evaluation and conclusions drawn can further be used to derive possible future improvements and further scope to find better model formulations for the application.

2 Algorithms

2.1 Deep Q Learning

We have seen how Q Learning works in class, in a general sense we try to estimate the Q table to keep track of the situations that can arise and take an action that can maximize rewards. But this approach even though it works well, has a disadvantage here as the state-action combination for connect 4 is pretty large compared to simple games like discussed in class. According to the games wiki, there are around 4,531,985,219,092 positions for this game and this complexity makes using Q learning a less attractive option. But this is where Deep Learning excels. The picture below taken from [1] shows the difference and why Deep Q Learning is preferred in this case.



(1)

2.1.1 Input to the model

To train a deep Q-learning neural network, we feed all the observation-action pairs seen during an episode (a game) and calculate a loss based on the sum of rewards for that episode. For example, if winning a game of connect-4 gives a reward of say 20, and a game was won in 7 steps, then the network will have 7 data points to train with, and the expected output for the "best" move should be 20, while for the rest it should be 0 (at least for that given training sample). This example is inspired from [2]

If we repeat these calculations with thousands or millions of episodes, eventually, the network will become good at predicting which actions yield the highest rewards under a given state of the game. Most importantly, it will be able to predict the reward of an action even when that specific state-action wasn't directly studied during the training phase. Also, the reward of each action will be a continuous scale, so we can rank the actions from best to worst.

I can also use the ε greedy for explore-exploit, for this case and do the evaluation as the solo case.

2.2 Monte Carlo Tree Search (MCTS)

A well documented approach, The core idea of MCTS [3] is to successively focus multiple simulations starting at the current state by extending the initial portions of trajectories that have received high evaluations from earlier simulations. MCTS is generally considered a model-based approach because it builds a search tree to simulate and explore the possible future states and actions based on a model of the environment. This model can be a simplified, learned model of the environment, or an actual simulator.

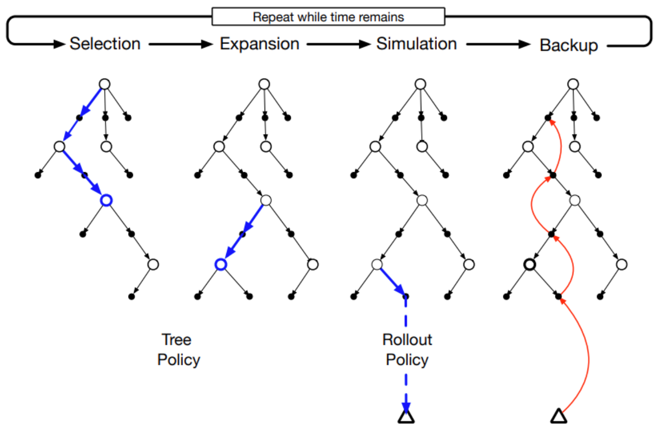


Figure 8.10: Monte Carlo Tree Search. When the environment changes to a new state, MCTS executes as many iterations as possible before an action needs to be selected, incrementally building a tree whose root node represents the current state. Each iteration consists of the four operations **Selection**, **Expansion** (though possibly skipped on some iterations), **Simulation**, and **Backup**, as explained in the text and illustrated by the bold arrows in the trees. Adapted from Chaslot, Bakkes, Szita, and Spronck (2008).

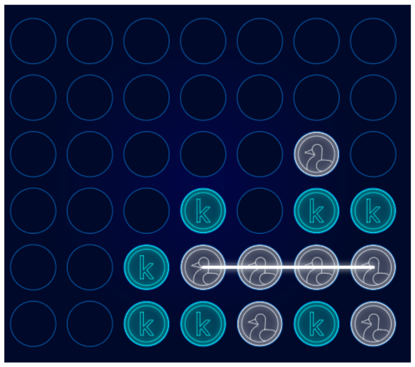
(2)

3 Environment setup, Datasets

The problem is well established, having it's own Kaggle competition where they provide the environment for the game (`https://www.kaggle.com/competitions/connectx/overview`). In case for the Deep Q Learning application where we need data to train the model, we use the Kaggle environment to gneerate the data and train itself. Every time we interact with this environment, we can pass an action as input to the game. After that, the opponent will respond with another action, and we will receive a description of the current state of the board, as well as information whether the game has ended and who is the winner. As long as we store this information after every play, we will keep on gathering new data for the deep q-learning network to continue improving. Much similar to other environment simulators like Gym.

4 Evaluation and Output expected

Since the goal of the game is to win, I plan to evaluate both models with their win percentage, against itself or a random opponent. I am still working out the details of the evaluation, but there a few papers that use win percentages as a valid metric while the agent is setup to play against itself.



(3)

5 Timeline - Project Layout

Date	Progress
2023-10-30	Initial project planning and research
2023-11-01	Project Proposal
2023-11-10	Setup Environment and Understand the algorithms involved
2023-11-20	Setup Deep Q Learning framework and Debug
2023-12-01	Implement MCTS and iron out bugs
2023-12-04	Experimentation and performance evaluation
2023-12-05	Project Presentation
2023-12-08	Final testing and minor bug fixes
2023-12-09	Documentation and report writing
2023-12-11	Project report submission.

Table 1: Project Timeline

References

[1] Ankit Choudhary. `https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/`.

[2] Louis D'hulst. `https://medium.com/@louisdhulst/training-a-deep-q-learning-network-for-connect-4-9694e56cb806`.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.