ENPM690: Robot Learning

Homework 4

Prateek Arora
Masters of Engineering in Robotics
University of Maryland, College Park
Email: pratique@umd.edu
Link to github repo

I. Abstract

Connect Four is a two-player connection board game in which the players first choose a color and then take turns dropping one colored disc from the top into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the lowest available space within the column. The first one to form a horizontal, vertical, or diagonal line of four of one's own discs wins the game [1]. The objective of this project is to train a RL agnet to play the game through self-play and reinforcement learning. Deep convolutional residual neural network have been used to learn the policy and value functions. Further, Monte-Carlo tree search has been used to generate game dataset to train the neural network. Lastly, evaluation of neural network is performed where two network play against each other and the model for the winner network is saved.

II. Setup

To simulate the connect four board game a 6×7 grid is created which is updated using drop_piece function with "X" and "O" as each player plays. Each location in the grid represents a state, however, our network is trained on sequence of images Rather than location of each piece. An action is dropping a piece from the top of the grid and thus total possible actions equals the number of columns which is 7. A sample of environment can be seen in figure 1.

```
[[' ' ' ' ' ' ' ' ' ' ' ' ' ']
[' ' ' ' ' ' ' ' ' ' ' ' ' ']
[' ' ' ' ' ' ' ' ' ' ' ' ' ']
[' ' ' ' ' ' ' ' ' ' ' ' ' ']
[' ' ' ' 'O' ' ' 'X' 'X' ' ']
[' ' 'O' 'O' 'X' 'O' 'X' ' ']] 0
```

Figure 1: Connect four environment

III. Method

In order to train an agent to learn the game Monte-Carlo tree search and deep recurrent neural networks have been used. The approach has been unpaired Alphazero and the code was refered from Wee Tee Soh [2]. There are two main components in the pipeline which is described below:

A. Network Architecture

A deep convolutional residual neural network (using PyTorch) with the above architecture similar to AlphaZero to map an input Connect4 board state to its associated policy and value has been used. The policy is essentially a probability distribution of next moves the player should move from the current board state (the strategy), and the value represents the probability of the current player winning from that board state. This neural net is an integral part of the MCTS, where it helps guide the tree search via its policy and value outputs as we will see later. The structure of the network used has been described below:

```
Input: Board state (encoded)
├─ Convolution block with 128 filters
│  ├─ Batch Norm
│  └─ Relu
├─ 10 Res blocks each with:
│  ├─ Conv layer
│  ├─ Batch Norm
│  └─ Relu
└─ Output block
   ├─ Policy output
   │  ├─ Batch Norm
   │  └─ Relu
   └─ Value output
      ├─ Batch Norm
      └─ Relu
```

The raw Connect4 board is encoded into a 6 by 7 by 3 matrix of 1's and 0's before input into the neural net, where the 3 channels each of board dimensions 6 by 7 encode the presence of "X", "O" (1 being present and 0 being empty), and player to move (0 being "O" and 1 being "X"), respectively. Finally, to properly train this neural net which has a two-headed output, a custom loss function is defined as simply the sum of the mean-squared error value and cross-entropy policy losses.

B. Monte-Carlo Tree Search

A game can be described as a tree in which the root is the board state and its branches are all the possible states that can result from it. In a game such as Go where the number of branches increase exponentially as the game

progresses, it is practically impossible to simply brute-force evaluate all branches. Hence, the Monte-Carlo Tree Search (MCTS) algorithm is devised to search in a smarter and more efficient way. Essentially, one wants to optimize the exploration-exploitation tradeoff, where one wants to search just exhaustively enough (exploration) to discover the best possible reward (exploitation)

The process of Select, Expand and Evaluate and Backup represents one search path or simulation for each root node for the MCTS algorithm. In AlphaGo Zero, 1600 such simulations are done. For our Connect4 implementation, we only run 777 since it's a much simpler game. After running 777 simulations for that root node, we will then formulate the policy p for the root node which is defined to be proportional to the number of visits of its direct child nodes. This policy p will then be used to select the next move to the next board state, and this board state will then be treated as the root node for next MCTS simulations and so on until the game terminates when someone wins or draw. The whole procedure in which one runs MCTS simulations for each root node as one moves through until the end of the game is termed as MCTS self-play.

IV. Network Evaluation and Results

After one iteration in which the neural net is trained using MCTS self-play data, this trained neural net is then pitted against its previous version, again using MCTS guided by the respective neural net. The neural network that performs better (eg. Wins the majority of games) would then be used for the next iteration. This ensures that the net is always improving.

Below are plots of the performance of network while training (see figure 2 and figure 3)

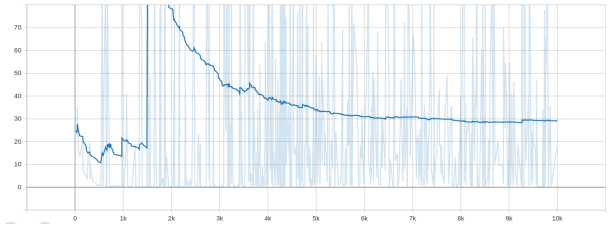

Figure 2: loss vs epoch graph

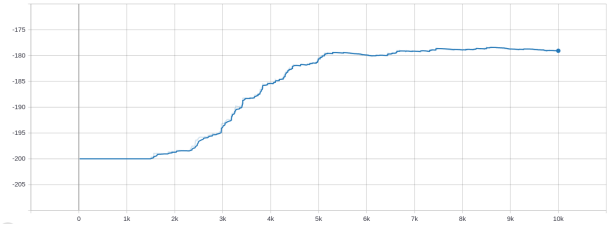

Figure 3: win vs epoch graph

References

[1] Connect 4 wiki. https://en.wikipedia.org/wiki/Connect_Four.

[2] Medium post: Towards data science. https://towardsdatascience.com/from-scratch-implementation-of-alphazero-for-connect4-f73d4554002