Introduction

The Connect4 game is a 2 player game which contains a board with 7 columns and 6 rows. Each player takes turns where it drops one colored disk in any one of the columns, from the top, which falls to the lowest empty spot. The first one to have 4 disks with the same color in a horizontal, vertical or diagonal wins. The game has more than 10^{14} states which make it difficult to solve with tabular reinforcement learning algorithms. So for this project, I decided to try the AlphaZero algorithm developed by Google's DeepMind which is a Deep reinforcement learning algorithm in which we train a deep neural network as the *Value Function Approximator*.

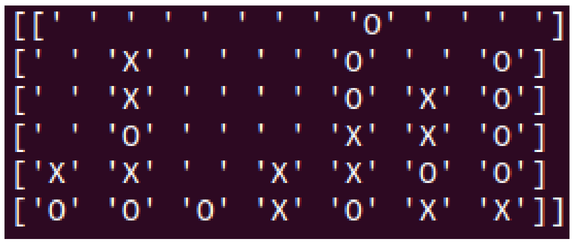
Figure 1: Connect four environment

Problem Formulation

Since the work is based on AlphaZero, the current system was also named ConnectZero. ConnectZero learns to play by looking at the board positions that would result from each available move, evaluating each of them using the neural network and then choosing the best one.

Input: The Connect4 board is represented as a 6x7x3 matrix of 1s and 0s. The 3 channels encode the presence of X(1 beign present and 0 being empty), O and player to move(0 being "O's" turn and 1 being "X's" turn).

Outputs: The output is probability distribution of possible moves(policy) and value(O wins: +1, X wins: -1, draw: 0)



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

Figure 2: Connect four environment

0.1 Deep Convolutional Neural Network

The neural network design is as follows:

1. One Convolution block - 128 filters (3x3, 1 stride) + batch_norm + relu.
2. 19 residual blocks - each block has 128 filters(3x3 kernel, 1 srtide) + batch_norm + relu + 128 filters (3x3 kernel, 1 stride) + batch_norm + residual connection + relu
3. Output block -
   - Policy: convolution of 32 filter(1x1 kernel, 1 stride) + batch_norm + relu + linear(fully connected) + softmax
   - Value: convolution of 3 filters(1x1 kernel, 1 stride) + batch_norm + relu + linear + relu + linear + tanh activation.

Monte Carlo Tree Search

The self-play policy is guided by a Monte-Carlo Tree Search to generate the dataset which is then used to train the neural network in an iterative manner. The iteration pipeline is as follows:

1. Self-play using MCTS to generate game datasets (s, p, v) where s - *board state,* p - *policy output from the network,* v - *value output from the network.* The neural network provides the prior probabilities to choose the desired action.
2. Train the neural network using the *(s, p, v)* generated from MCTS self play.
3. Evaluate the trained neural net (at pre defined checkpoints) by pitting it against the neural network from the previous iteration and keeping only the neural net that performs better.
4. Repeat

Results

The figure(3) shows the loss vs epochs for the training duration. In each iteration the neural net is trained using the play data generated using MCTS and then evaluated against the neural

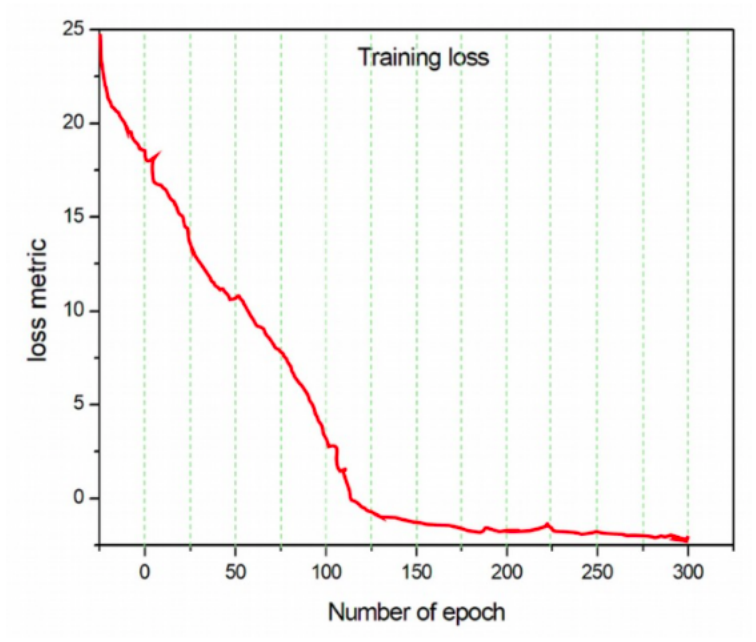


Figure 3: Caption

network from the previous iteration as the other Connect4 player. The winner neural network is then preserved for the next training iteration.

After a total of 4 iterations, 3268 MCTS self-play games was generated. A typical loss vs epoch of the neural network training for each iteration is shown above, showing that training proceeds quite well for each iteration. Once the training was finished connectZero_net_6 was pitted against connectZero_net_4 and out of 100 games played, **connectZero_net_6 won 87 and lost 13**.

Please follow this git_link to access the code for this project.

References

1. From-scratch implementation of AlphaZero for Connect4
2. Codebox Software - Learning to play Connect 4
3. Deep Reinforcement Learning and Monte Carlo Tree Search With Connect 4
4. Applying Machine Learning to Connect Four