Advanced Machine Learning **Group 3**
Prof. Parag Singla Date: 12/11/2019

Overview

This assignment focuses a simplified implementation of AlphaGo-Zero reinforcement learning algorithm. Using self-play between old and new version of players the algorithm attempts to defeat it's predecessor AlphaGo which represents culmination of human knowledge.

In this assignment, we are using a smaller 13x13 board for the ease of search. We are using Monte Carlo Tree search to explore the states. The neural network outputs the policy and value using a double headed architecture.

Strategies & Code

Generating episodes using short-sighted MCTS to train a player which will be used a base to do further training.

We have written a Go game wrapper over PachiPy from scratch which encompasses functions like listing all possible legal moves and such. These enable us to store the board state and board in a structured manner in form of objects and perform faster and more efficient computation.

One heuristic that we implemented was training our network on the Monte Carlo Simulations of a greedy player for better model weight initialization. After this initialization, this model was trained with self-play. This was able to consistently beat a model with a random weight initialization.

Initialization

We have reduced the size of neural network as compared to the original paper. Following is our neural network architecture:

- Layer 1 : Input 13 x 13 x 2 to Output 13 x 13 x 128 [Conv2D -> BatchNorm -> ReLU]
- Layer 2 : Input 13 x 13 x 128 to Output 13 x 13 x 128 [Conv2D -> BatchNorm -> ReLU]
- Layer 3 : Input 13 x 13 x 128 to Output 13 x 13 x 128 [Conv2D -> BatchNorm -> ReLU]
- Layer 4 : Input 13 x 13 x 128 to Output 512 [Fully Connected]
- Layer 5 : Input 512 to Output (13 x 13 + 2) [Output layer 1]
- Layer 6 : Input 512 to Output 1 [Output layer 2]

Where output layer 1 gives the action probabilities and output layer 2 predicts the value function.

We are using the following hyper-parameters for training.

- Dropout - 0.2
- Learning Rate - 0.005
- Momentum - 0.9
- Episodes - 10
- Epochs - 20

Parallelization

We have parallelized generating games (as each one uses separate MCTS trees) over CPUs and GPUs using bash scripts to spawn multiple python processes. Then neural network training is done sequentially from the game-plays obtained. Competition between new and old model is done in parallel.

Environment Setup

We have only used torch(v1.0), pachi_py libraries. Any other libraries are not required.

Group 3 Students

Udit Jain, Shashwat Shivam, Sushant Rathi
2016CS10327, 2016CS10328, 2016CS10329