Project 2 Report

AI Capstone

Name: Kai-Jie Lin
Student ID: 110652019
April 12, 2023

1 Abstract

I applied monte carlo tree search for this game. Originally I want to try alpha zero. I have trained my model and testing for about a week, but the action space is too complicated to train. So I finally use the pure MCTS. But I will discuss the algorithm of alpha zero and how it fail in this report. Alpha Zero is a work that trains an agent for the game through pure self-play without any human knowledge except the rules of the game. Since the game in this project has little human knowledge and strong opponent can be used to train the agent, I thought Alpha Zero was a great idea for this game. I am going to introduce the algorithm, experiment with other agent and some problem I have faced.

2 Method

2.1 Monte Carlo Tree Search

For the tree search, we maintain the following:
$Q(s, a)$: the expected reward for taking action a from state s, i.e. the Q values.
$N(s, a)$: the number of times we took action a from state s across simulations.
$P(s, \cdot) = p_\theta(s)$: the initial estimate of taking an action from the state s according to the policy returned by the current neural network. From these, we can calculate $U(s, a)$, the upper confidence bound on the Q-values as

$$U(s, a) = Q(s, a) + c_{puct} \cdot P(s, a) \cdot \frac{\sqrt{\sum_b N(s, b)}}{1 + N(s, a)}$$

Here c_{puct} is a hyperparameter that controls the degree of exploration. To use MCTS to improve the initial policy returned by the current neural network, we initialise our empty search tree with s as the root. A single simulation proceeds as follows. We compute the action a that maximises the upper confidence bound $U(s, a)$. If the next state s' (obtained by playing action a on state s) exists in our tree, we recursively call the search on s'. If it does not exist, we add the new state to our tree and initialise $Q(s', a)$ and $N(s', a)$ to 0 for all a. Instead of performing a rollout, we then propagate $v(s')$ up along the path seen in the current simulation and update all $Q(s, a)$ values. On the other hand, if we encounter a terminal state, we propagate the actual reward (+1 if player wins, else -1).

2.2 Neural Network

The neural network f_θ is parameterised by θ and takes as input the state s of the board. It has two outputs: a continuous value of the board state $v_\theta(s) \in [-1, 1]$ from the perspective of the current player, and a policy $p_\theta(s)$ that is a probability vector over all possible actions. The neural network is then trained to minimise the following loss function (excluding regularisation terms):

$$l = \sum_t (v_\theta(s_t) - z_t)^2 - \pi_t \log(p_\theta(s_t))$$

In this project, I used 4-layer CNN network followed by few feedforward layers.

2.3 Policy Iteration Through Self Play

Here is the complete training algorithm. We initialise our neural network with random weights, thus starting with a random policy and value network. In each iteration of our algorithm, we play a number of games of self-play. In each turn of a game, we perform a fixed number of MCTS simulations starting from the current state s_t. We pick a move by sampling from the improved policy π_t. This gives us a training example (s_t, π_t). The reward _ is filled in at the end of the game: +1 if the current player eventually wins the game, else -1. The search tree is preserved during a game. At the end of the iteration, the neural network is trained with the obtained training examples. The old and the new networks are pit against each other. If the new network wins more than a set threshold fraction of games (55 in the DeepMind paper), the network is updated to the new network. Otherwise, we conduct another iteration to augment the training examples.

3 Experiment

I trained an agent for the game a single GPU. Each iteration consisted of 150 episodes of self-play and each MCTS used 25 simulations. The model took around 4 days (100 iterations) for training. I evaluated the model against some agents use 100 rounds: Since vannila mcts has better result and

Agent	**Win Rate(Alpha Zero:Other)**
Random	90:10
Vannila MCTS	27:73
Minimax	31:69

Table 1: Results of Agent Competitions

I have no more time to tune the model. I finally chosed the vannila one.

4 Why NN model did not perform well?

4.1 Problem of Action Space

The total action space of this game is $12 \cdot 12 \cdot (1 + 2 * 6)$ which is relativiely large. But in practice, we do not need that much action space. There is atmost 32 empty position but every episode may change the position distribution. This is hard for neural network to the game.

4.2 Training time and computing resources

Since the state space and action space may be super large. My network design is relatively small which can not handle large dimensions. If I use larger model, it can cost more time to converge and the repond time may be too slow. It is hard for large neural network to find a great policy by mcts. In the end, I give up to use alpha zero algorithm to train model. Maybe I will try to design a newer model to play this game in the future.