Report for assignment 1

Jingmo Bai

February 18, 2023

1 Description of the solution

In this assignment, our task is to write a program and implement an algorithm to play a connect four game against the server, and to win the server consistently, 20 times in a row. And each move should be decided within 5 seconds so we need to make our program fast. To achieve the task, I use the alpha-beta pruning algorithm in a minimax search to find the best move in my program.

1.1 Minimax search and alpha-beta pruning algorithm

The Minimax algorithm is often used to implement an artificial intelligence program to play zero-sum games where two parties compete, and one party always chooses the option that maximizes its advantage among the available options, and the other party chooses the option that minimizes the opponent's advantage. The main idea of this algorithm is to evaluate each position of the state in the game and to compute a value to indicate how good or bad it will be for one player to reach that position. Then for the so-called maximizing player, it will always choose the options to maximize the value to have more advantage, for minimizing player, it will always choose the options to minimize the value.
In order to make our program respond within 5 seconds, I also use a variant of the minimax algorithm which called alpha-beta pruning. The benefit of this algorithm is that it can prune some of the branches in the search tree as long as some condition is met. In this algorithm, we set two variables alpha and beta, alpha is used to store the best evaluated value for maximizing player which is the maximum value, and beta is used to store the best evaluated value for minimizing player which is the minimum value. Once alpha is greater or equal to beta, the rest of the branches can be pruned. Because that means the minimizing player can always choose the previous branch in which it can at least get the beta value in the end. So whatever how high the value the maximizing player can obtain in the rest branches, it is pointless. Because the opponent will always choose the best option for itself so it will always choose the previous branch where it get the beta value. So with this method we can save up to half of the computing time for giving up those branches.
We also need to restrict our searching depth in the program. After testing, when I try to search to depth 5, it will always take more than 5 seconds to get the results. Finally I decide to set the depth as 4, and after this, we can make sure our program can respond within 5 seconds.
In the alpha-beta pruning minimax search function, I set the alpha to negative infinity and beta to positive infinity initially, and use the evaluation function to

get the evaluated values of the states of board. Then there is a for loop for every available moves which should be seven in most cases, for every available moves we apply another alpha-beta pruning minimax search function until the max depth we set. Then the max value among the first 7 states should be passed to alpha, in this step, the minimizing player can not make a choice. Then the min value among these 7 max values should be passed to beta, in this step, the maximizing player can not make a choice. And when alpha is greater or equal to beta, the rest branches can be pruned, so we break the loop. (refer to the code from line 133 to line 167)

1.2 Evaluation function

In the evaluation function, we need to detect lines in four different directions in the board, in rows, in columns, and in two-way diagonals. In each direction, we count the number of 1, -1, and 0. If the number of 1 is going to be 4 we assign +10000 to the value which means we can win by 4 in a row, we definitely want the next move to be that position. If the number of 1 is 3 we assign +12 to the value and so on. On the other hand, we also want to defend from the opponent, so if the number of -1 is going to be 4, we must block that, so we assign -10000 to the value so it won't make the next move to that possible losing position. If the number of -1 is 3, then the situation is not that urgent, so we assign -10 to the value. I want the algorithm to be more offensive so I assign more absolute value on winning moves than defensive moves.(refer to the code from line 75 to line 124)

2 How to launch and use the solution

To launch and use the program, simply type the command "python code.py -l" to play against the local opponent in the code, and type the command "python code.py -o" to play against the server.
To check the statistics you have played, type the command "python code.py -s".

3 Peer-review

My review partner is Basim Elessawi whose student ID is "Ba5266el-s".

3.1 Peer's Solution

My peer used similar methods as mine, which are alpha-beta pruning in a minimax search and an evaluation function. The difference is the value assigned for evaluating scores, and some other strategy.

3.2 Technical Differences of the Solutions

- The first difference between our codes is the scoring for evaluation. His algorithm focuses more on defending, preventing the opponent from acquiring a win. While mine algorithm assigns more scores on offensive moves than defensive moves.

- The second difference is that his code randomly initializes some of the elements during the search tree, by assigning stochastic values to them, which I didn't.

3.3 Opinion and Performance

- Which differences are most important?
  I believe the most important difference is the different strategies for evaluation. He set a higher reward value for defending the opponent. So he tried to make sure not to lose, and in the worst case result in a draw. So his record showed that there were more draws between his algorithm and the server. However, I just assigned more value to the winning moves, so my algorithm seldom gets a draw.

- How does one assess the performance?
  We can assess performance based on the ratio of games won to games played in total, or the ratio of games not lost to games played in total, which are the winning rates and not losing rates. One can also assess the performance by taking the calculation time of each move into consideration when some other requirements are applied.

- Which solution would perform better?
  Compared by the winning rate, my result is 25/26=0.96, and his is 35/37=0.94. Actually, the performance is quite close. He just lost one more game than me, considering he played more games than me, that is quite normal.

4 Paper summary AlphaGo

All games of perfect information have an optimal value function. These games may be solved by recursively computing the optimal value function in a search tree. However, in large games such as Go, exhaustive search is infeasible. Prior work has been limited.
This paper introduces a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. It also introduces a new search algorithm that combines Monte Carlo simulation with value and policy networks.
They pass in the board position as a 19×19 image and use convolutional layers to construct a representation of the position. They use these neural networks to reduce the effective depth and breadth of the search tree, evaluating positions using a value network, and sampling actions using a policy network.
First, they begin by training a supervised learning policy network from expert human moves. Next, they train a reinforcement learning policy network that improves the SL policy network by optimizing the final outcome of games of self-play. Finally, they train a value network that predicts the winner of games played by the RL policy network against itself.
With this algorithm, a huge breakthrough has been made.

4.1 Difference to the own solution

There are several differences between this approach and mine.
First, it uses deep neural networks to reduce the effective depth and breadth of the search tree, but in my connect 4 game, it's possible to do an exhaustive search.
Second, it trains the network by supervised learning from human experts and reinforcement learning from self-play, but I didn't do that.
Third, they train the value and policy network to evaluate board positions and select moves, but in my solution, I just use a simple scoring way.

4.2 Performance

There is a possibility that using the AlphaGo approach to deal with the connect 4 games will make a better behavior than my solution. But in my opinion, they train this complicated algorithm because in those large games like chess and Go, exhaustive search is infeasible. But in a simple connect 4 game, it's possible and low-cost to do it. Training those neural networks, using supervised learning and reinforcement learning, is just too costly and time-consuming. It is just not worth it.