IMPLEMENTATION OF OTHELLO

CIS 667 - Intro To Artificial Intelligence

Project Report

Advait Ramesh Iyer (330623485) - Aiyer01

Aniruddha Ghosh (782510200) - Aghosh03

Sanjeev Kulkarni (222950080) - Skulka06

Yash Jain (878756149) - Yajain

Contents

1 INTRODUCTION

Othello (formerly called as Reversi) is a strategy board game for two players, played on a 8x8 uncheckered board. In a play, if there are player's discs between opponent's discs, then the discs that belong to the player become the opponent's discs.The objective of the game is to end the game with the most discs on the board, and the least of opponent's discs.

If the computer were to search all the possible states of the game, the search space is of the order of $10\hat{2}8$, which is an impractical order of computation when the computer has to play a game live, with a human.Therefore, to decrease the space of the search, the project includes implementation of the following algorithms:

- **Minimax Tree Search with AB pruning:** In this algorithm, the opponent always tries to minimize the utility of the player, and the player tries to maximize their own utility.We have also used alpha-beta pruning in the implementation.
- **Convolutional Neural Network:** The neural network which uses images of the board states to make a decision about the next move on the basis of the heuristics defined by the network.

For the experiment, we will try to build an AI Agent which, after receiving a given set of inputs, will be able to calculate the estimated tree size for all the possible legal cases to win, lose, or draw. Moreover, the board constructed in the experiment will be generalized to n×n blocks.

The GitHub link for the code and README file is :

2 GAME

The goal of the game is to win by having more discs than your opponent, by the end of the game. Therefore, each player tries to place as many disks on the board as possible, and their objective is to maximize the difference between the winner's disks against the loser's. Simply winning is the basic goal, and maximizing the 'disk differential' is considered an ancillary.

Multiple iterations have been conducted on the rules of the game since its origin, and as per the current universal practice, placing two black-sides and two white-sides diagonal to each other at the center of the board is considered the default state of the board.

In the implementation, we have used 2D array to store the steps.We have tried to vary the problem instance by providing the user with 3 different difficulty levels.

- One star being the easy level uses minimax till depth 1.
- Two star being the moderate level uses minimax till depth 4.
- Three star being the difficult level uses minimax till depth 6.

The initial state of the board is given by the following screenshot where the player is asked to choose the difficulty level of the game.

Figure 1: Choosing the difficulty level

Depending on the level of difficulty that the player chooses to play, the algorithm tries to search deeper into the trees. The more trees that the computer searches, the more possibilities it has reviewed before making a decision, therefore the level of difficulty accordingly changes.

The game starts with white and black pieces placed diagonally at the center of the board. The green trace-points are meant for the white to decide the legal positions which can be taken in the next step.

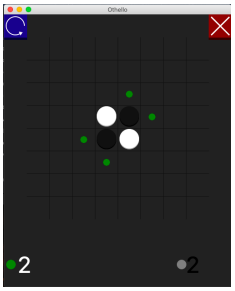

Figure 2: Initial State of the Board

Considering the left-most trace-point, let's play the white piece. Once the left position is filled with white, the center black-piece is flipped to white.After the white has played, the minimax algorithm decides that the next step should be black at the leftmost position, which flips the central white piece. Following is the result after one step.

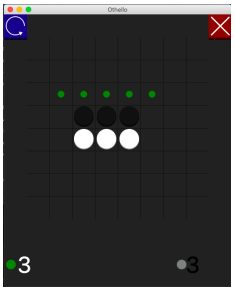

Figure 3: Both players having equal number of pieces

After a couple of moves, it is clear that white is dominating the board at this step

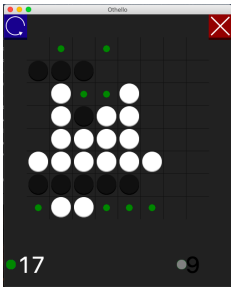

Figure 4: White player dominating the board

White has won the match, as it conquered the most part of the board. The legal moves will be

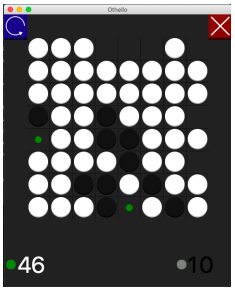

Figure 5: White player moving towards win

decided by referencing to a list of moves categorized as legal and not-legal. This way, the illegal moves are all eliminated from the search algorithm's search space. The board can be extended to nXn as per our implementation.

3 TREE SEARCH

We have used minimax algorithm with alpha beta pruning for tree search.
In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.

Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game. We have used alpha beta pruning with limited depth and depth is dependent on the difficulty level chosen.For ex for one star, the depth is 1. For two star, the depth is 4 and for three star, the depth is 6.

The pseudocode for the minimax algorithm is as follows:

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

```
(* Initial call *)
minimax(origin, depth, TRUE)
```

4 MACHINE LEARNING MODEL

This part of the implementation uses Convolutional Neural Networks (CNN) to play, and train the game on multiple games. The objective is to run various kinds of optimization techniques after training to calculate the local minimas, such that the loss function gets minimized, and the error between the actual value and the prediction is minimized.

The base algorithm is standard Q-learning, and the output value function is an 8*8 matrix, with invalid moves pruned. The value function is trained, and the optimizers used are (1) Vanilla Stochastic Gradient Descent (SGD), (2) Stochastic Gradient Descent, (3) Adam Optimizer. The CNN consists of 8-steps of convolutional layers, and 3 steps of fully-connected multi-layer perceptrons.

Inputs of the CNN are 4-features, black pieces, white pieces, valid moves, and a constant zero-padding.

Vanilla SGD calculates the gradients, and accordingly finds the minima with respect to the weights, biases, and the input values. Depending on the gradient, the value is accordingly updated.

w = w - [U+25BD]Qi(w)

where, w = weight, Q(w) = the function representing the equation of the output layer. When differential is calculated, the chain rule is applied.

Stochastic GD and Adam Optimizer are more sophisticated versions of Vanilla SGD, the difference being SGD uses an adaptive learning rate, and Adam optimizer uses the first and second order moments (Taylor's first and second approximation).

5 BASELINE OPPONENT

We implemented a human baseline opponent that randomly selects any of the possible legal moves that are shown on screen without thinking of any consequences or applying any knowledge of the game or trying for a high score. Therefore the baseline opponent randomly selects a possible move out of the legal ones displayed on screen by the computer.

6 EXPERIMENTAL RESULTS

The major takeaways from our project are:

- Model 1 provides a 3 difficulty levels of play depending on how deep the algorithm searches for the node.
- CNN has been able to decrease loss the most at a learning rate of 0.1.

Learning rate is higher when the gradient is not very high, so we can say that the gradient is not too steep, so we keep a relatively high learning rate of 0.1, instead of 0.01 or 0.001 for convergence.

Setting the Epsilon Greedy exploration rate to 0.5, i.e., there's a 50% probability at every instance of the algorithm exploring further states, 4 cases were explored with various learning rates:

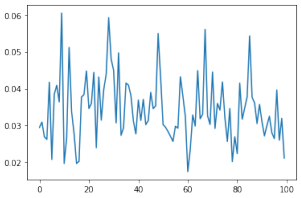

Figure 6: Case 1: Learning rate = 0.001, epsilon greedy rate = 0.5

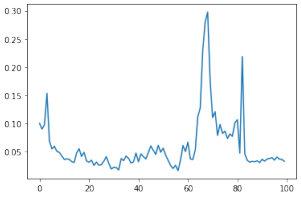

Figure 7: Case 2: Learning rate = 0.005, epsilon greedy rate = 0.5

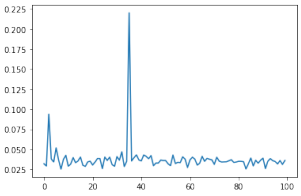

Figure 8: Case 3: Learning rate = 0.01, epsilon greedy rate = 0.5

Work Division

1. Aniruddh: Implemented minimax/alpha-beta pruning variant from online source for the first model of our project

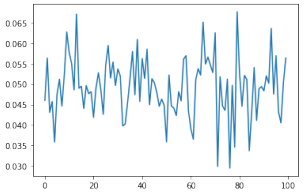

Figure 9: Case 4: Learning rate = 0.05, epsilon greedy rate = 0.5

2. Yash - Literature review, variation of difficulty extension in the first model, research and formulation of and model evaluation methodology for both models

3. Advait - Research and implementation of the variation of the CNN from online source, iterations of CNN to minimize loss function

4. Sanjeev - Research on CNN implementation, Project consolidation, Report formulation, work allocation and team management, and creating Visualizations

7 CONCLUSION

In our project, we have implemented a two player Othello game where a player gets to choose the mode. In the initial state, there are equal number of coins belonging to each player. The possible legal moves are shown to the player out of which he chooses the next step.In this way the game goes on till one player has maximum coins on board and there are no more legal moves left.

Based on the analysis of the results,our AI model performed best in the three star mode where the depth is 6. The challenging part of our project was to analyze the results by varying the configurations as it required us to retrain the data and change the instances.The future scope of this project is to extend it to 3 players.