CIS667 Project Final Report

Jiayu Ding (jding14), Kefu Wu (kwu129), Shuangding Zhu (szhu03)
Github: https://github.com/FrankWuwuwu/Checkers

1. Introduction

In this project, we implemented a checker board game and proposed an experiment that compared the performance of 3 different types of decision trees. The checker board game defined the size of the domain and actions that could be applied. Besides the user's action input, we implemented 3 types of AI's that decide the next action: a baseline AI that randomly picks an action from the feasible range, a Monte Carlo Tree Search AI and three Convolutional Neural Networks that were trained based on the data generated from MCTS. The three CNNs differ in the number of layers and the structure of each layer. Experiments also consist of two parts: MCTS vs the baseline AI and CNNs vs the baseline. In the first part, we compared the performance of MCTS and the baseline AI for 5 different domain sizes among 500 games, e.g. 6x6,8x8,10x10,12x12,14x14. In the second part, performance of 3 different configurations of CNN vs baseline AI was evaluated by the scores among 100 games. The MTCS showed high performance among 500 games, while the CNNs only showed a win rate around 50%. In the end, a conclusion was drawn based on the performance of each AI and discussions about the results were proposed.

2. Domain Implementation and Description

Domain：**Checkers**

Checker is a board game with two antagonistic players. The game can be implemented with larger or smaller instances. We implement the game by our original code and we can set the size of the board at the beginning of the game to experiment.

Rule modification: each turn, there is a 10% chance that a player's piece automatically teleports to a random nearby location. This will always happen after the player makes their move.

State

In our game, the state of the game at each turn is represented by a tuple **(player, board)**, where 'player' is the current player to make a move and 'board' is a 2D list representing each grid on the game board. The value of each grid represents what is on the grid.

0 represents an empty grid.

-1 represents the unreachable grid, (which is also empty).

1 represents the piece of player 1.

2 represents the piece of player 2.

For example, the initial 'board' for a 6*6 games will be:

```
[[ 2 -1  2 -1  2 -1]
 [-1  2 -1  2 -1  2]
 [ 0 -1  0 -1  0 -1]
 [-1  0 -1  0 -1  0]
 [ 1 -1  1 -1  1 -1]
 [-1  1 -1  1 -1  1]]
```

To make it readable, we implement a function show_board() to print the board state for the user after each move. The initial game board of 6*6 games will be shown like this.

```
2 X 2 X 2 X
X 2 X 2 X 2
- X - X - X
X - X - X -
1 X 1 X 1 X
X 1 X 1 X 1
```

Valid action

We determined all the valid actions in function valid_actions().

The function will return a list of all valid actions for the current player. An action in the 'valid action' list includes the action type (0 for move, 1 for jump), the coordinates of the piece and its moves.

An example of 'valid action' list is shown below:

```
[(0, (4, 0), (3, 1)),
 (0, (4, 2), (3, 1)),
 (0, (4, 2), (3, 3)),
 (0, (4, 4), (3, 3)),
 (0, (4, 4), (3, 5))]
```

During turns, players can move any of his/her pieces to an empty diagonal grid or jump through a grid held by the rival's piece to eliminate the piece of the rival.

A piece can only move forward, except the piece reaches the bottom of the rival board and becomes king piece.

If a jump chance exists, the player must jump.

If a jump chance exists after a jump, the player must jump again, which is called multiple jump.

The player's turn ends after he/she makes a diagonal move or finishes all the jumps.

Game over

The game is over when there is no valid action for the current player.

At this time, the player with more pieces alive on board will be the winner.

If the number of pieces for both players is the same, it will be a tie.

Gamesize and player option

At the start of each game, you can choose the board size of this game. The minimal game size is 6*6. Then, you will be able to choose one of the following control strategies: human, baseline AI, tree-based AI, or tree+NN AI.

The game will randomly choose a player to make the first move, as moving first may provide potential advantages.

```
C:\Users\52466\Desktop\Checkers>python play_checker.py
Please enter an integer for the board size(e.g 8 for 8*8 game):6
player 2 go first.
Please choose the control for Player 1 (enter index).
1. human
2. baseline AI
3. Tree-based AI
4. Tree+NN AI
You choice:1
Please choose the control for Player 2 (enter index).
1. human
2. baseline AI
3. Tree-based AI
4. Tree+NN AI
You choice:2
```

Typical step during game-play

I start of a 6*6 game for human VS simple AI:

```
C:\Users\52466\Desktop\Checkers>python play_checker.py
2 X 2 X 2 X
X 2 X 2 X 2
- X - X - X
X - X - X -
1 X 1 X 1 X
X 1 X 1 X 1
--- Player's turn --->
Player 1, choose an action (enter the index):
0    move the checker of (4, 0) to (3, 1)
1    move the checker of (4, 2) to (3, 1)
2    move the checker of (4, 2) to (3, 3)
3    move the checker of (4, 4) to (3, 3)
4    move the checker of (4, 4) to (3, 5)
Your action:
```

The game shows 5 valid moves that the player can perform. Players can enter the index of that move to perform the move. Here I enter 0. The result will be shown, and its simple AI's term.

```
C:\Users\52466\Desktop\Checkers>python play_checker.py
2 X 2 X 2 X
X 2 X 2 X 2
- X - X - X
X - X - X -
1 X 1 X 1 X
X 1 X 1 X 1
--- Player's turn --->
Player 1, choose an action (enter the index):
0    move the checker of (4, 0) to (3, 1)
1    move the checker of (4, 2) to (3, 1)
2    move the checker of (4, 2) to (3, 3)
3    move the checker of (4, 4) to (3, 3)
4    move the checker of (4, 4) to (3, 5)
Your action:0
2 X 2 X 2 X
X 2 X 2 X 2
- X - X - X
X 1 X - X -
- X 1 X 1 X
X 1 X 1 X 1
--- AI's turn --->
AI action: move the checker of (1, 3) to (2, 4)
```

At the end of game, winner will be shown like this:

```
--- AI's turn --->
AI action: move the checker of (3, 3) to (4, 2)
1 X - X - X
X - X - X -
- X - X - X
X - X - X -
2 X 2 X - X
X - X 2 X -
Game over, player 2 wins.
```

As a special rule modification, a random transfer will happen with 10% chance for each term. It will look like this.

```
Opps! A random transfer happened.
One of your piece automatically teleports to a random nearby location.
Your piece at ( 1 , 3 ) teleports to ( 2 , 2 )
2 X - X 2 X
X 2 X - X 2
1 X 2 X 2 X
X - X - X -
- X 1 X 1 X
X 1 X 1 X 1
```

The code can be found in our github repository. Enjoy playing our games.

3. Baseline AI Implementation

The baseline AI will choose a move from all valid actions uniformly at random.

4. Tree AI Implementation and Description

We implemented the tree AI by **Monte-Carlo Tree Search(MCTS)**, which is in the MCTS.py file.

Algorithm

We use the MCTS code for tic-tac-toe from the course as a base and gradually adapt every aspect of this algorithm to serve for our game. The core idea of this code is rollout. In each searching process, a number of rollouts will be performed to give an evaluation of current state based on a consequence of child selection. The explored area of the tree will increase after each term of rollout so that later search can make benefit of previous rollout. As a result, the tree-AI can make better decisions based on the evaluation of potential future state.

```
def rollout(node):
    if node.state.is_leaf(): result = node.state.score_for_max_player()
    else: result = rollout(node.choose_child())
    node.visit_count += 1
    node.score_total += result
    node.score_estimate = node.score_total / node.visit_count
    return result
```

Child selection

We provide three kinds of children selection strategies in our code: exploitation, exploration and **Upper Confidence bounds applied to Trees(UCT)**. UCT is our final choice.
Upper Confidence bounds applied to Trees is a children selection strategy developed based on the Upper Confidence Bounds (UCB1) formula (Ziad SALLOUM,2019):

$$v_i + C \times \sqrt{\frac{\ln N}{n_i}}$$

In the formula, vi stands for the value of state Si, ni stands for the total number of states that has been visited, and N is the sum of the number of visits of the nodes at the same level. C is an optimized constant selected for fine running. The child with the highest value of the UCB1 equation will be chosen.
exploitation, exploration and Upper Confidence bounds applied to Trees(UCT).
Currently, our game uses UCT as our children's selection strategy.
As we mentioned in section 3, our baseline AI will simply choose the valid action uniformly at random.

5. NN implementation and description

We choose to use **convolutional neural networks(CNN)** for this part. We integrate our CNN into our tree-based AI to build a tree+NN AI.
The general structure of CNN takes the state of the board as the input and the utility of the given state as the output. For convenience, we take the 10x10 board size as the example, therefore the input size will be 1x10x10 and the output size will be 1. For the batched version of N data, the size of input and output will be Nx1x10x10 and Nx1 respectively. All the data were transformed into tensor format before they were fed to CNN.
format before they are fed to the NN.
General CNN consists of two parts: convolutional structure and linear structure, which are connected sequentially. The layers of each structure can be modified as well as the total number of layers. When creating CNN, theCNN function takes 4 inputs, including number of input layers (should always be 1 for the given domain), board size, hidden features and the kernel size.
The SGD function was selected as our optimizer, and the learning rate between 0.0001 and 0.001 was picked. Using gradient descent method, the optimized weight and bias term was calculated.
As we are developing tree+NN AI on middle game size(10*10). We performed a 10000 rollout on MCTS tree of a 10*10 games and select 1000 nodes from the tree that have a big enough visit count. With enough visit count, these nodes can present the typical game state with a meaningful evaluation of utility. We divide these 1000 pairs of nodes and utility into two 500 dataset. One as a training dataset and the other as a testing dataset for neural networks.
The configuration of each group member's implementations are listed with the training and experimental results in section 7.

6. Tree AI experiments

As we use random action for baseline AI, the experiments we did are not identical.
Several board sizes are tested in these experiments. We used 6 by 6, 8 by 8, 10 by 10, 12 by 12 and 14 by 14 board sizes. Each game has two players, player1 presents baseline AI and player2 presents tree-based AI. One hundred games were run for each board size, so totally five hundred games were tested. The value of rollout was set to 500 for 6 by 6 and 8 by 8 board size, but the value was reduced to 100 or 200 to test 10 by 10, 12 by 12 and 14 by 14. As the large game size involves hundreds/thousands of turns, a game may take a long time even if the AI costs less than 5 seconds to make a decision for each turn. Since the program running time spent several hours for each of the last three board sizes, we had to reduce the value of rollout to save some time. The 100 game of 14 by 14 board size cost more than six hours to run. It has to be admitted that the tree algorithm can be optimized in the future.
At the current stage, the result of our Tree AI is quite good. The tree AI has almost a 100% winning rate against baseline AI.
Here we show the result for five different board sizes below:

```
X - X 1 X -
- X - X - X
X 2 X 2 X -
- X - X - X
X 2 X 2 X -
--- Player 2's turn --->
treebased AI action: jump the checker of (0, 4) to (2, 2)
- X 1 X - X
X - X - X -
- X 2 X - X
X 2 X 2 X -
- X - X - X
X 2 X 2 X -
Game over, player 2 wins.
45665 nodes produced by MCTS
100 game finished
player 1 wins 9 game
player 2 wins 87 game
Tie 4 game
```

Figure: Game Result for 6 by 6 Board Size

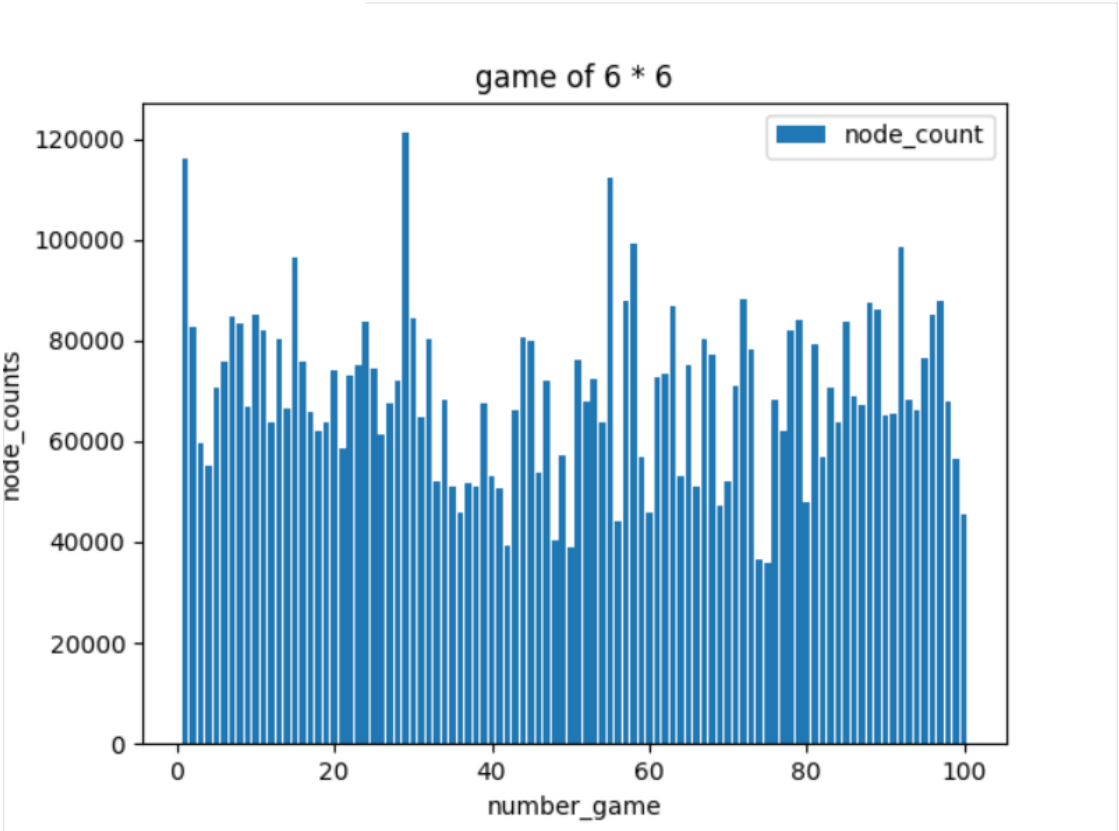


Figure: Node Counts for 6 by 6 Board Size

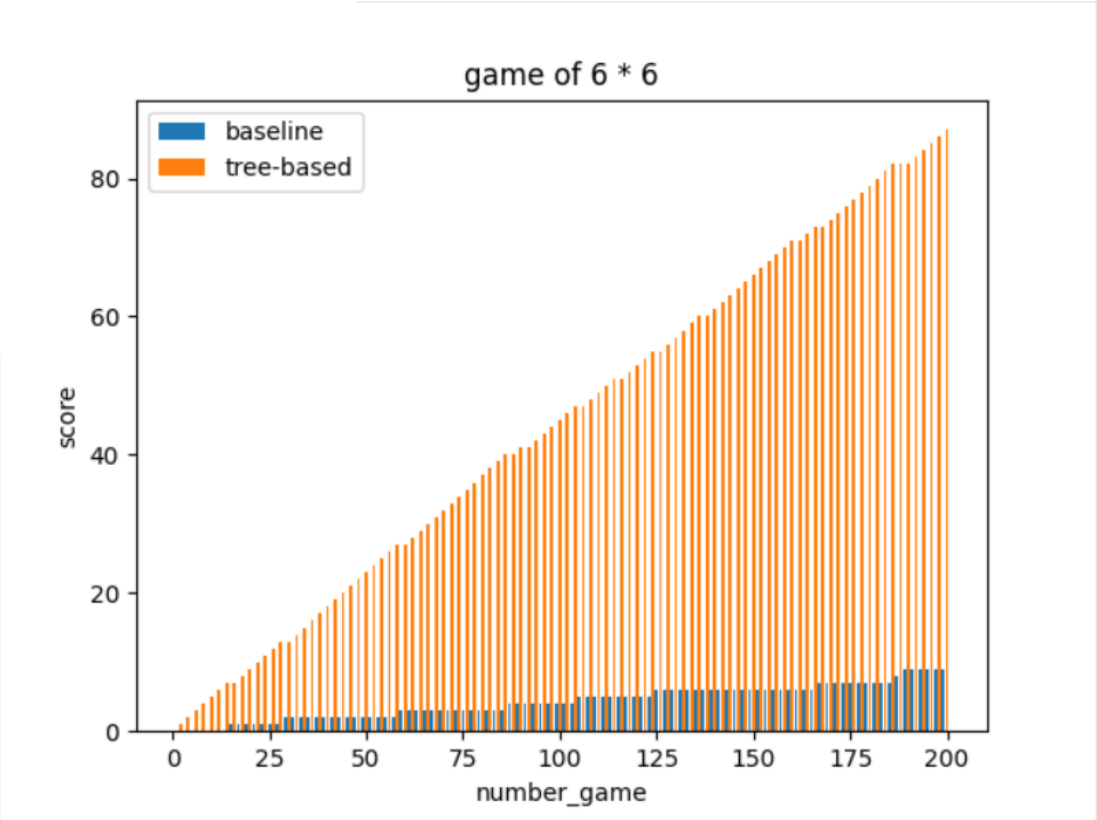


Figure: Two Players Score for 6 by 6 Board Size

```
X - X - X 2 X -
2 X - X - X 1 X
X - X - X - X -
--- Player 2's turn --->
treebased AI action: jump the checker of (5, 5) to (7, 7)
- X - X - X - X
X - X - X 2 X 2
2 X - X 2 X 2 X
X - X - X - X -
- X - X - X - X
X - X - X - X -
2 X - X - X - X
X - X - X - X 2
Game over, player 2 wins.
860137 nodes produced by MCTS
100 game finished
player 1 wins 1 game
player 2 wins 99 game
Tie 0 game
```

Figure: Game Result for 8 by 8 Board Size

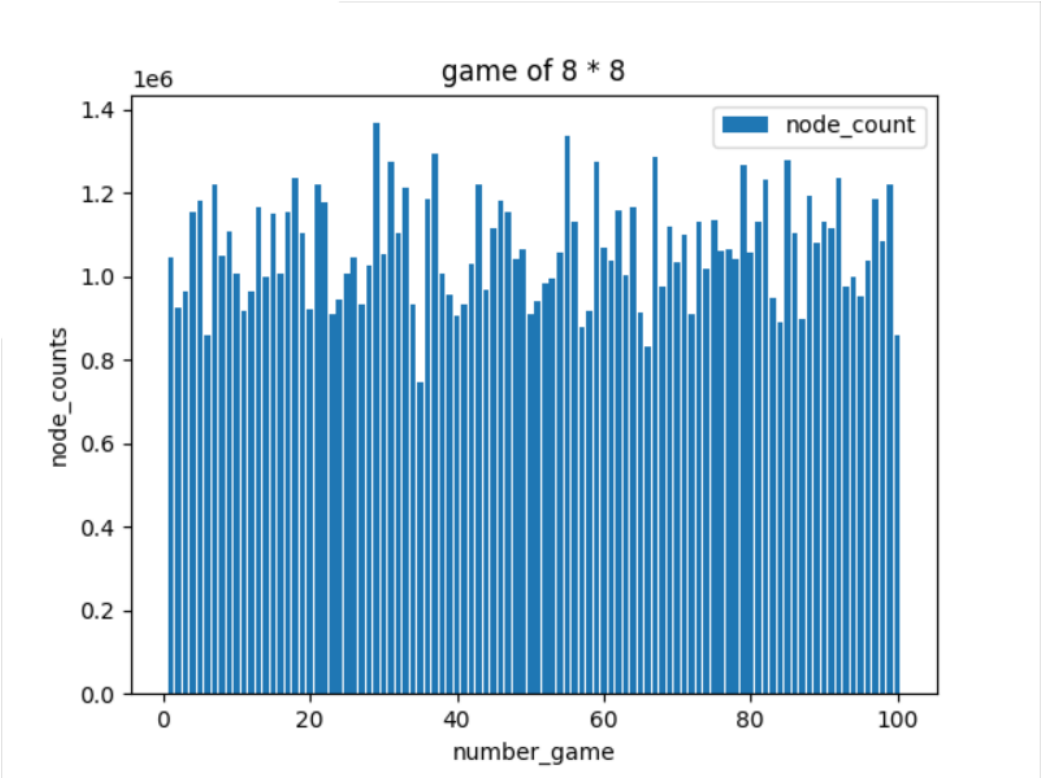


Figure: Node Counts for 8 by 8 Board Size

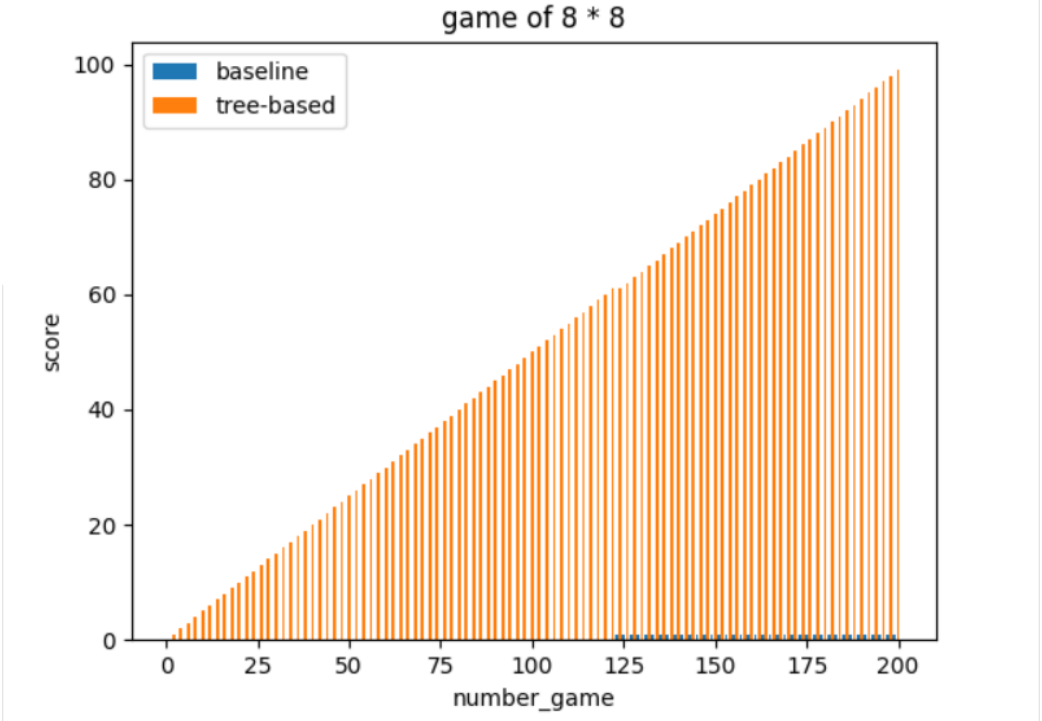


Figure: Two Players Score for 8 by 8 Board Size

```
X - X - X - X - X -
--- Player 2's turn --->
treebased AI action: jump the checker of (3, 5) to (5, 3)
- X - X - X - X - X
X - X - X - X - X 2
2 X 2 X - X 2 X 2 X
X 2 X - X - X 2 X 2
- X - X - X - X - X
X - X 2 X - X - X -
- X 2 X - X - X - X
X - X 2 X - X 2 X -
- X 2 X - X - X - X
X - X - X - X - X -
Game over, player 2 wins.
1275334 nodes produced by MCTS
100 game finished
player 1 wins 1 game
player 2 wins 98 game
Tie 1 game
```

Figure: Game Result for 10 by 10 Board Size

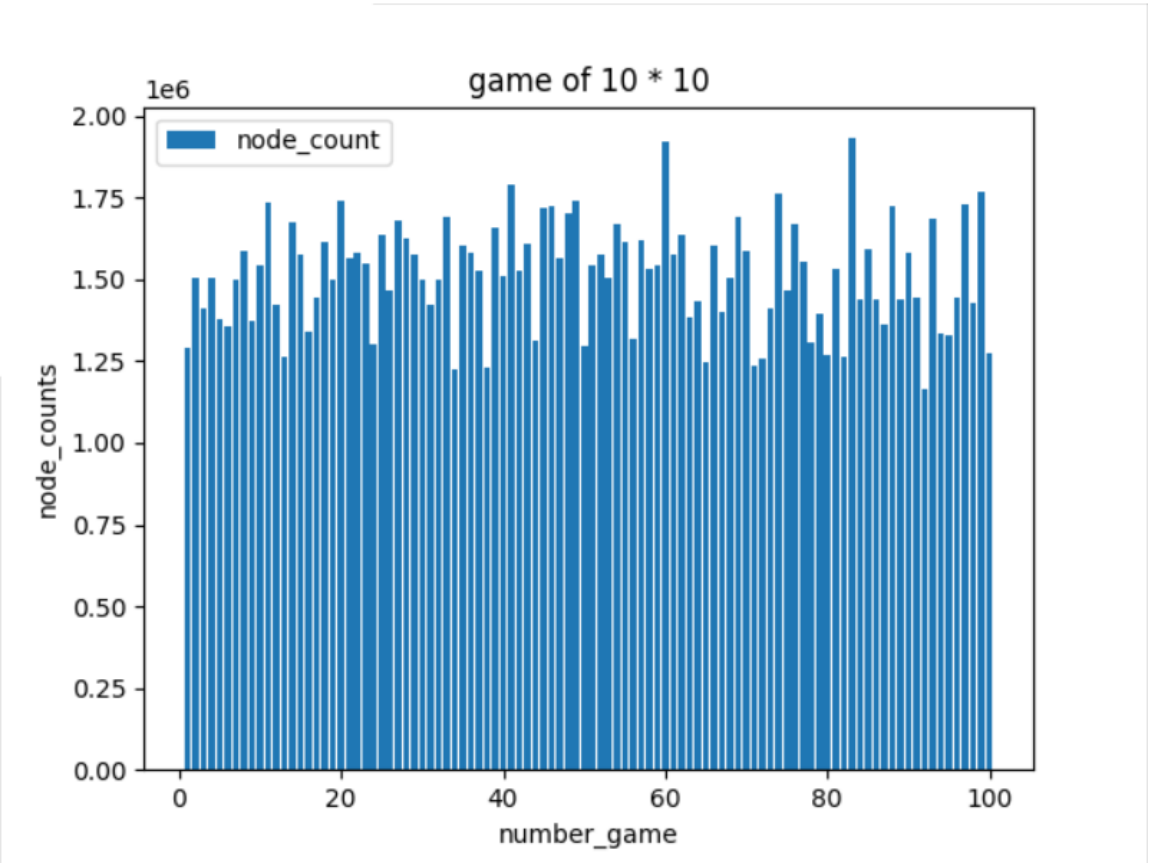


Figure: Node Counts for 10 by 10 Board Size

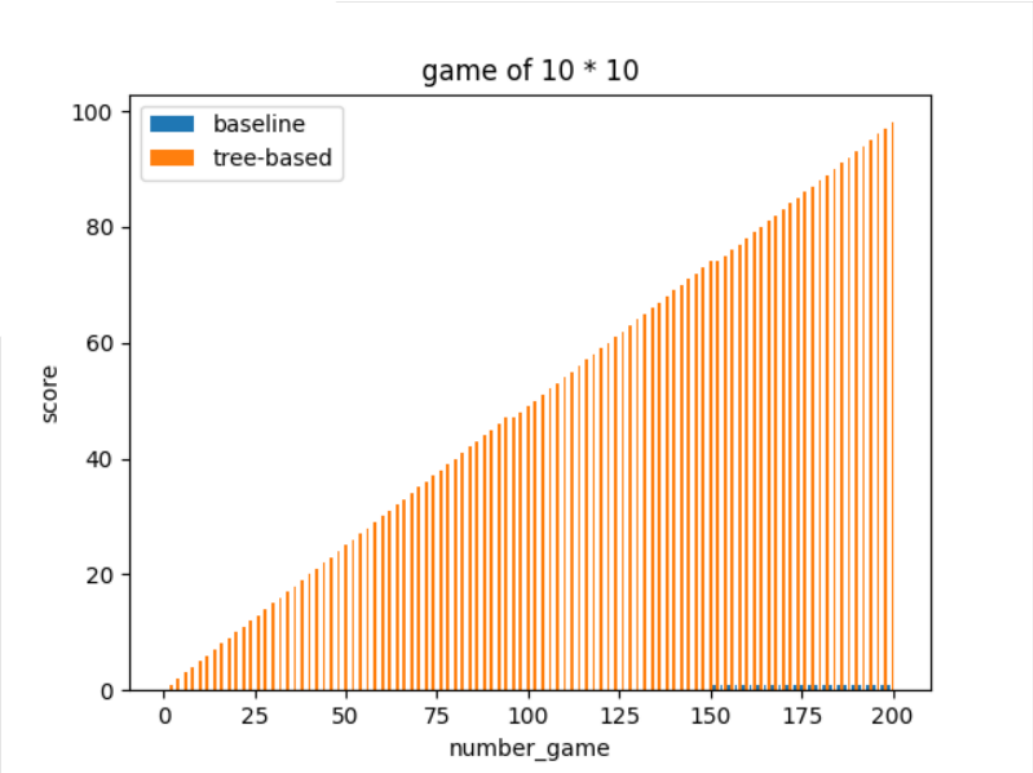


Figure: Two Players Score for 10 by 10 Board Size

```
- X - X - X - X - X 2 X
X - X - X 2 X 2 X 2 X -
- X - X - X - X - X - X
X 2 X - X 1 X - X - X 2
2 X - X - X - X - X - X
X - X - X - X - X - X -
- X 2 X - X - X - X - X
X - X - X - X - X 2 X 2
--- Player 1's turn --->
baseline AI action: move the checker of (7, 5) to (6, 4)
1 X 1 X - X - X 1 X - X
X - X - X - X - X - X -
- X - X - X - X - X 2 X
X - X - X - X 2 X 2 X -
- X - X - X - X - X - X
X 2 X 2 X - X - X - X 2
2 X - X - X - X - X - X
X 2 X - X - X - X - X -
- X - X - X - X - X - X
X - X - X - X - X 2 X 2
Game over, player 2 wins.
6745586 nodes produced by MCTS
100 game finished
player 1 wins 1 game
player 2 wins 98 game
Tie 1 game
```

Figure: Game Result for 12 by 12 Board Size

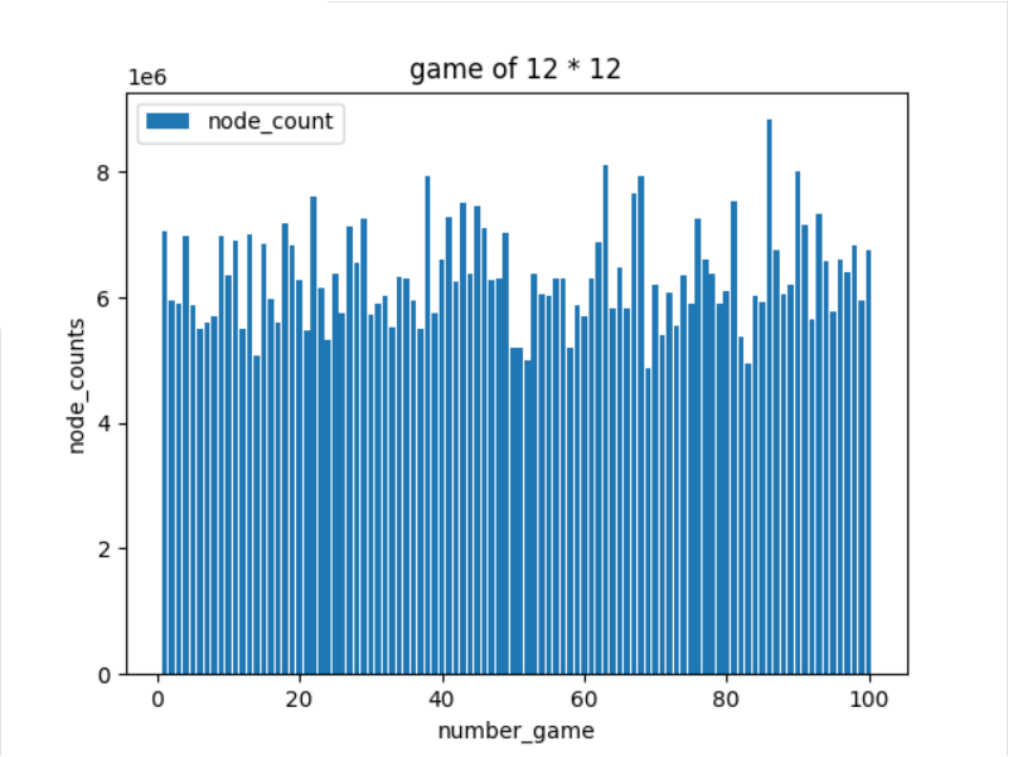


Figure: Node Counts for 12 by 12 Board Size

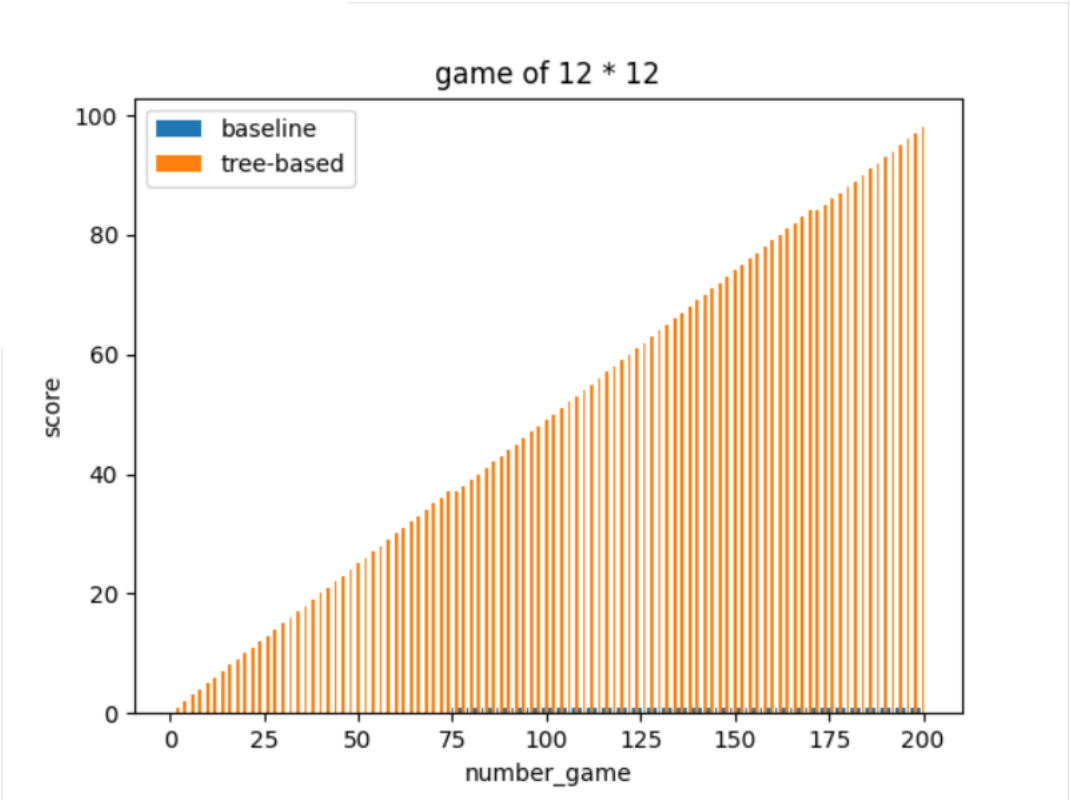


Figure: Two Players Score for 12 by 12 Board Size

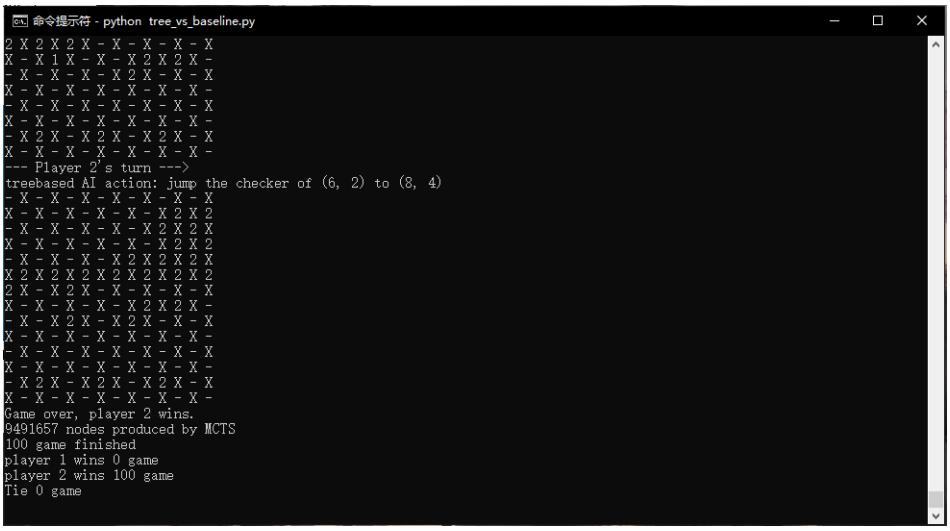


Figure: Game Result for 14 by 14 Board Size

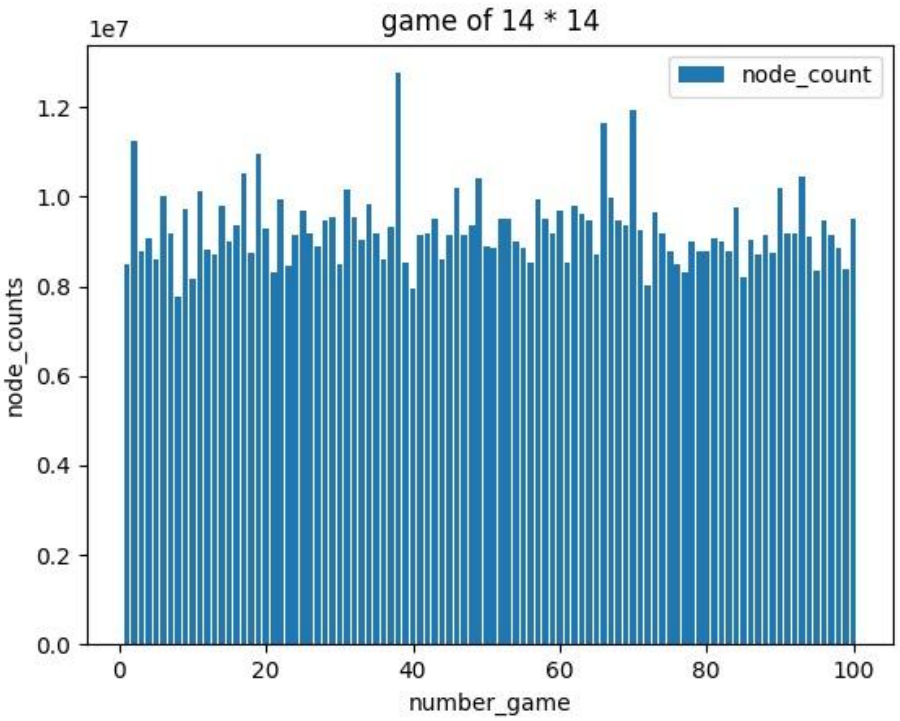


Figure: Node Counts for 14 by 14 Board Size

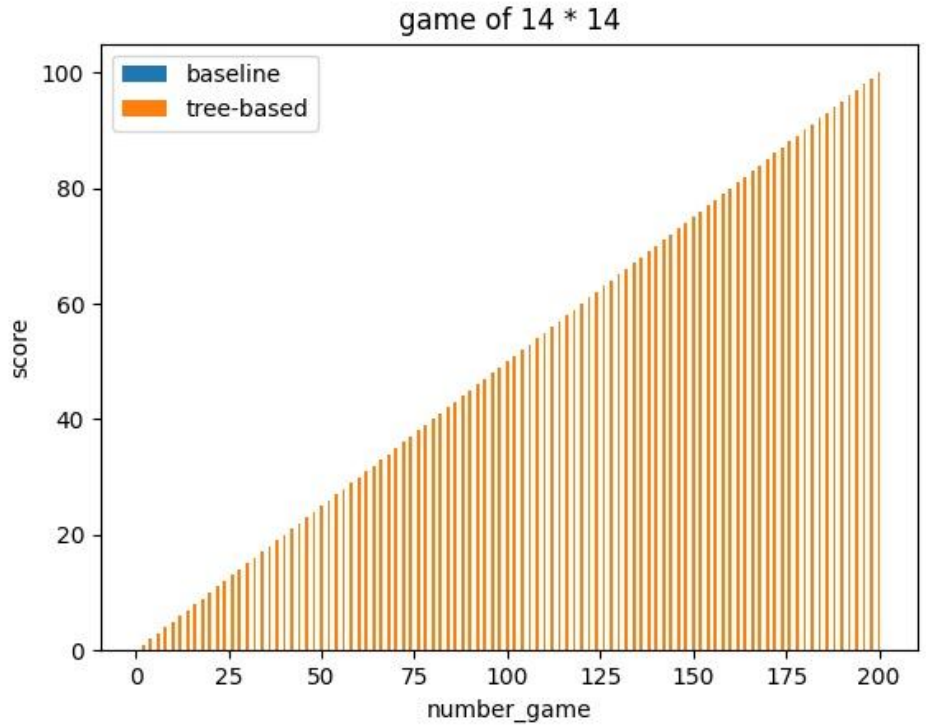


Figure: Two Players Score for 14 by 14 Board Size

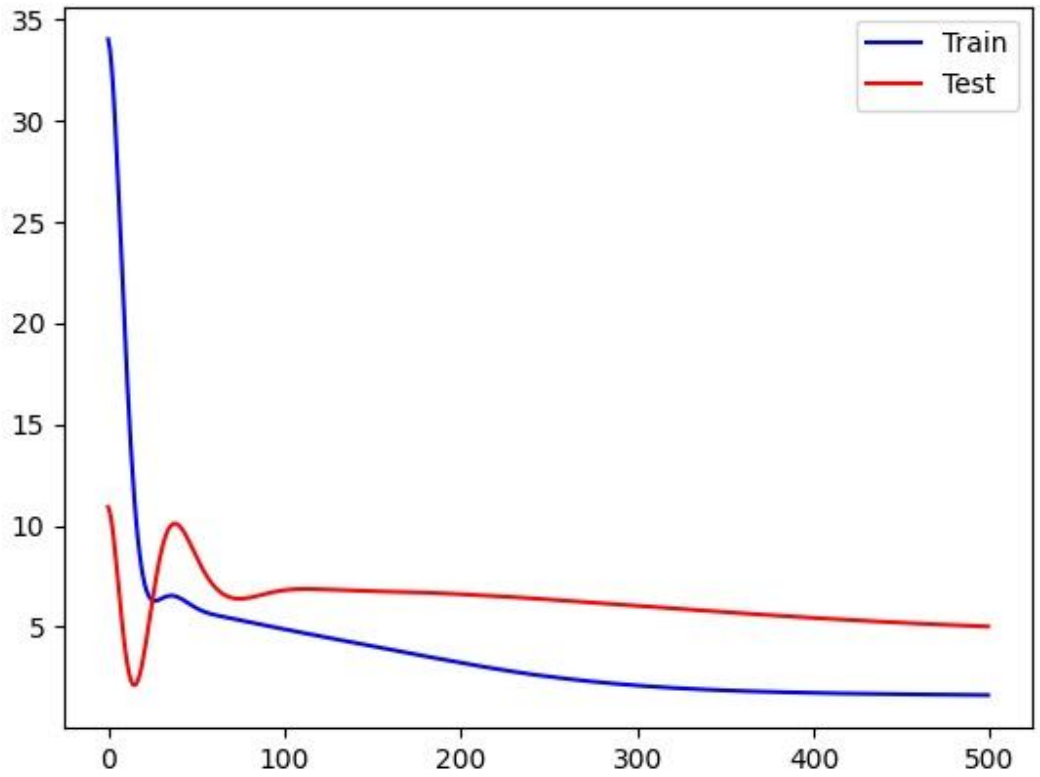
In the 100 games of 6 by 6 board size, the baseline can win several games by chance as there is not much random space for small games. When the board size increased, the advantage of strategy increased. Baseline can not win a game any more. We can also find out that the number of nodes produced by the game is gradually increased while we extend the game size. It is because it takes more turns for a state to reach an end-game state. And the number of possible child states of each state grows exponentially.
To sum up this section, the performance of our tree AI “crushed” the baseline AI with almost 100% win rate.

7. Tree+NN AI experiments

Version 1 - Jiayu Ding

the number of possible child states of each state grows exponentially.
Configuration: Two CNN layers, where each of them consists of a convolution layer, a batch norm layer, a RELU layer and a pooling layer + Two linear layers
CNN implementation and training parameters:
net = ConvNet(inputlayer=1,board size=10,hid_features=10,kernel_size=3)
optimizer = tr.optim.SGD(net.parameters(), lr=0.00001, momentum=0.9)

Learning curve of training and testing error:



Tree+NN AI vs Baseline AI experiment result (100 game):

```
- X 1 X - X - X - X
X - X - X - X - X 1
- X - X 1 X - X - X
X - X - X - X - X -
1 X - X - X - X - X
X - X - X - X 1 X -
- X - X 1 X - X - X
X - X - X - X - X -
- X - X 1 X - X - X
X - X - X - X 2 X -
Game over, player 1 wins.
16528 nodes produced by MCTS
100 game finished
player 1 wins 53 game
player 2 wins 43 game
Tie 4 game
```

Figure: Result of TreeNN vs Baseline for Version 1

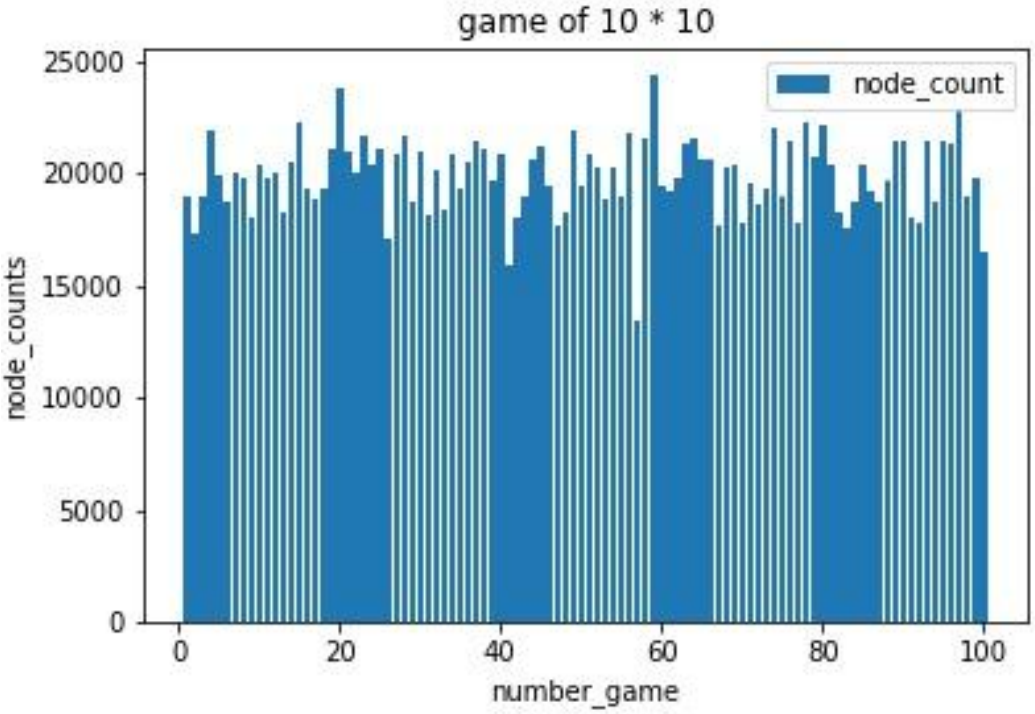


Figure: Node Counts of TreeNN vs Baseline for Version 1

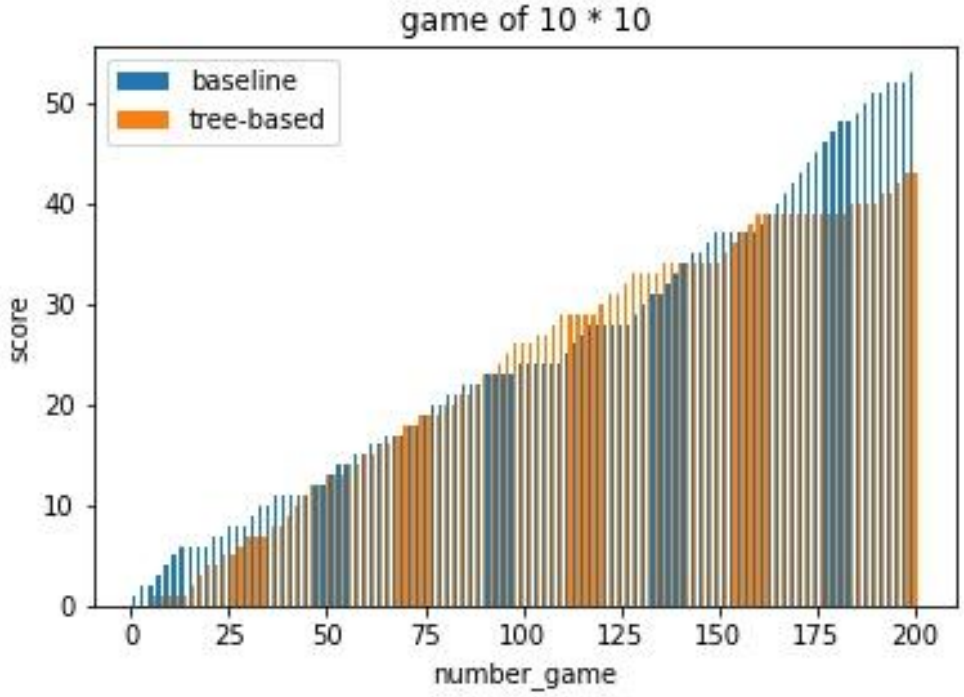


Figure: Score of TreeNN vs Baseline for Version 1

Version 2 - Kefu Wu

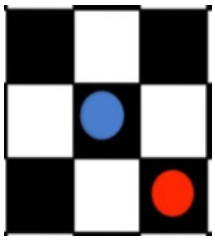
My Configuration: Two CNN layers, where each consists of a convolution layer a pooling layer + One linear layer

CNN implementation and training parameters:

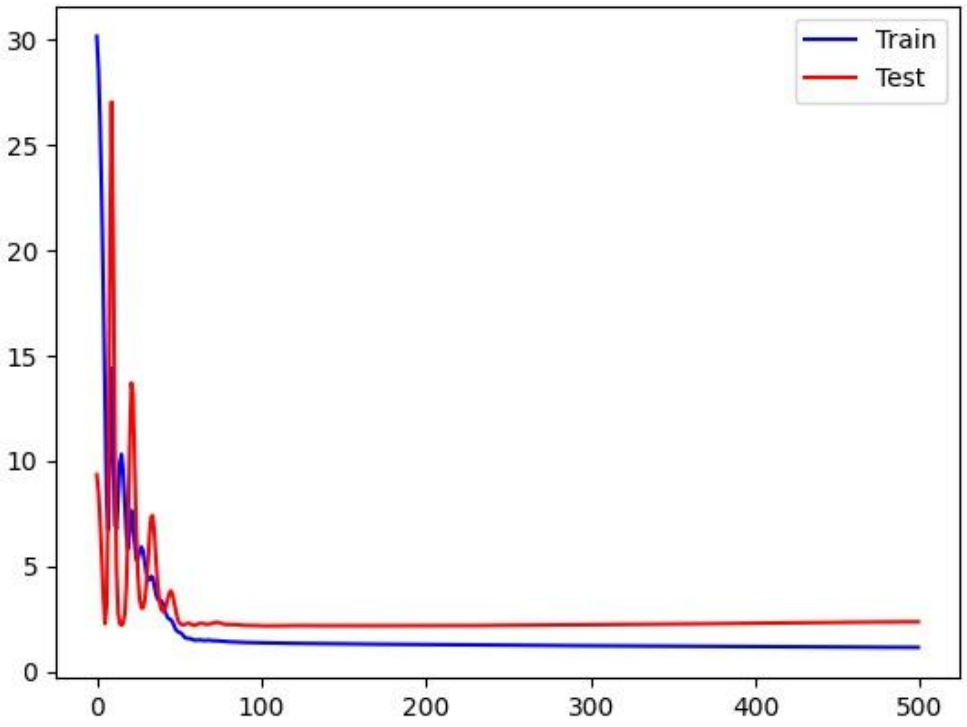
net = ConvNet(inputlayer=1,board size=10,hid_features=8,kernel_size=3)

optimizer = tr.optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

I insist to use 3*3 as the size of the filter kernel as ‘jump’ action is the most important action towards win. The ‘jump’ action can be identify in a 3*3 sub matrix on game board. Set 3*3 filter kernels will give a chance for convolution neural networks to learn this feature.



Learning curve of training and testing error:



Tree+NN AI vs Baseline AI experiment result (100 game):

```
X 2 X - X - X - X -
Game over, player 1 wins.
17509 nodes produced by MCTS
100 game finished
player 1 wins 51 game
player 2 wins 46 game
Tie 3 game
```

Figure: Result of TreeNN vs Baseline for Version 2

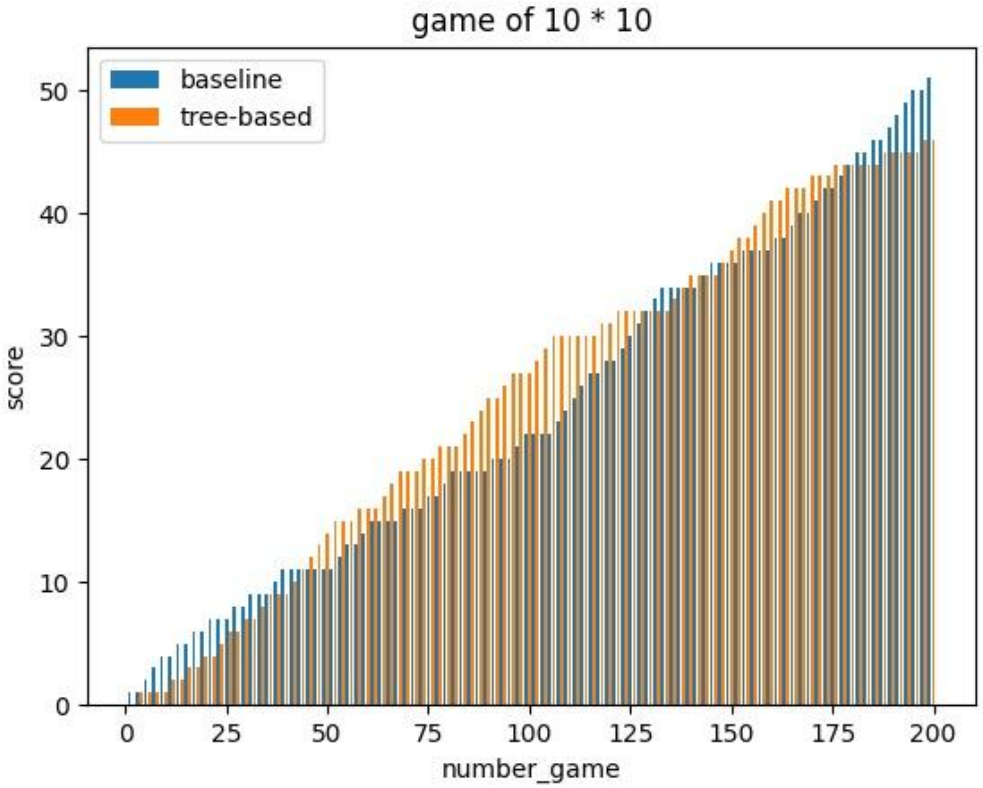


Figure: Score of TreeNN vs Baseline for Version 2

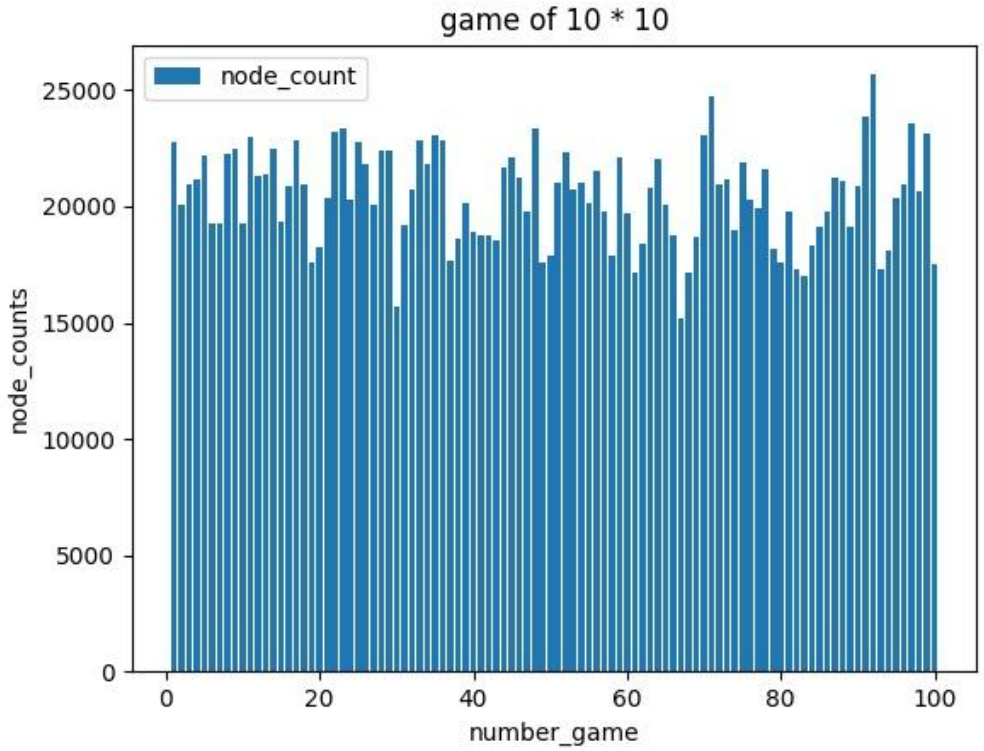


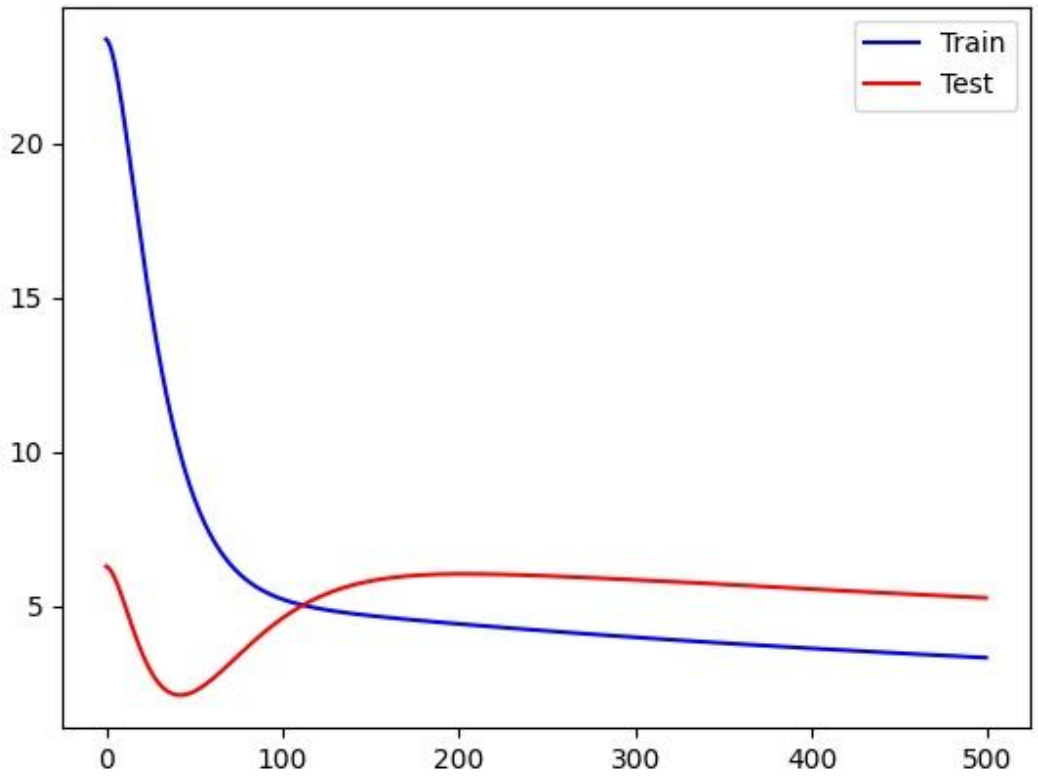
Figure: Node Counts of TreeNN vs Baseline for Version 2

The result of the trained AI is bad. It cannot beat the baseline AI firmly. It may be due to the limitation of the training dataset. We only use a small (500) dataset to train and the quality of the dataset is actually not good. We may be able to improve it in the future. Also, the configuration of my CNN can be further improved. As a result, although we did all the required process of this project, the trained AI did not learn features of this game and seemed to perform simi-random selection on action.

Version 3 - Shuangding Zhu

Configuration: One CNN layer consists of a convolution layer, a batch norm layer, a RELU layer and a pooling layer + One linear layer
CNN implementation and training parameters:
net = ConvNet(inputlayer=1,board size=10,hid_features=4,kernel_size=2)
optimizer = tr.optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

Learning curve of training and testing error:



Tree+NN AI vs Baseline AI experiment result (100 game):

```
X - X - X - X - X 2
2 X - X - X - X - X
X - X 2 X - X - X -
- X - X - X - X - X
X - X - X - X - X 2
- X - X - X - X - X
X 2 X 2 X 2 X - X 2
Game over, player 2 wins.
14910 nodes produced by MCTS
100 game finished
player 1 wins 25 game
player 2 wins 73 game
Tie 2 game
```

Figure: Result of TreeNN vs Baseline for Version 3

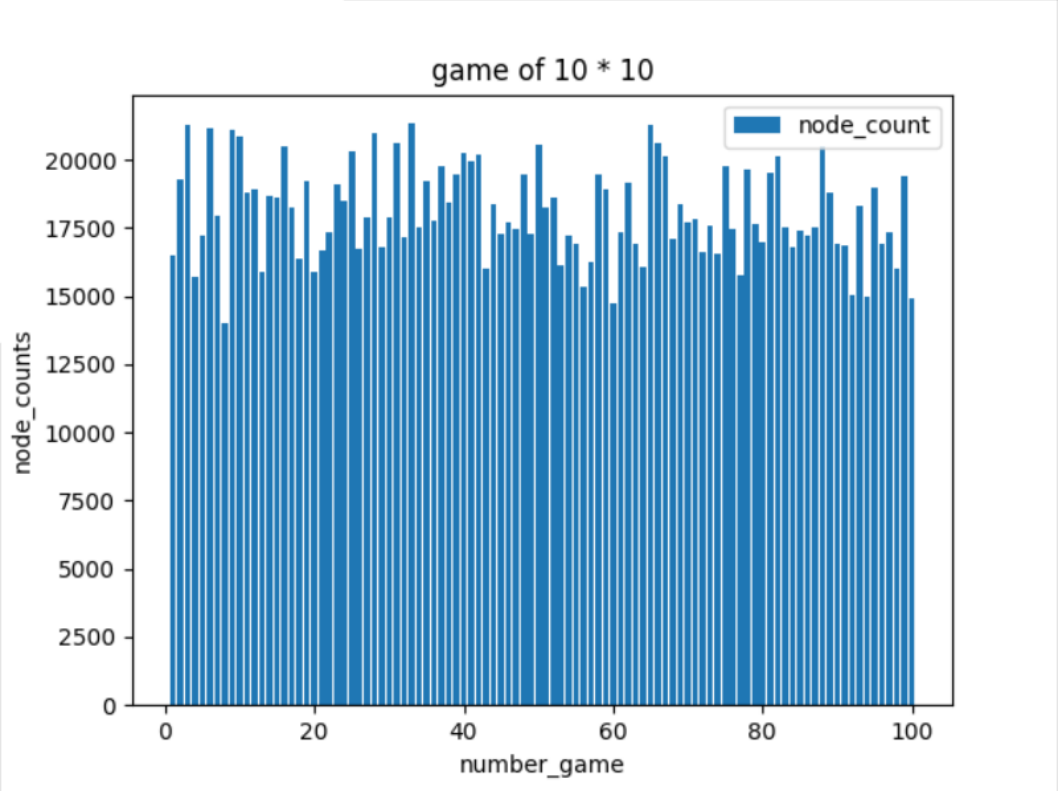


Figure: Node Counts of TreeNN vs Baseline for Version 3

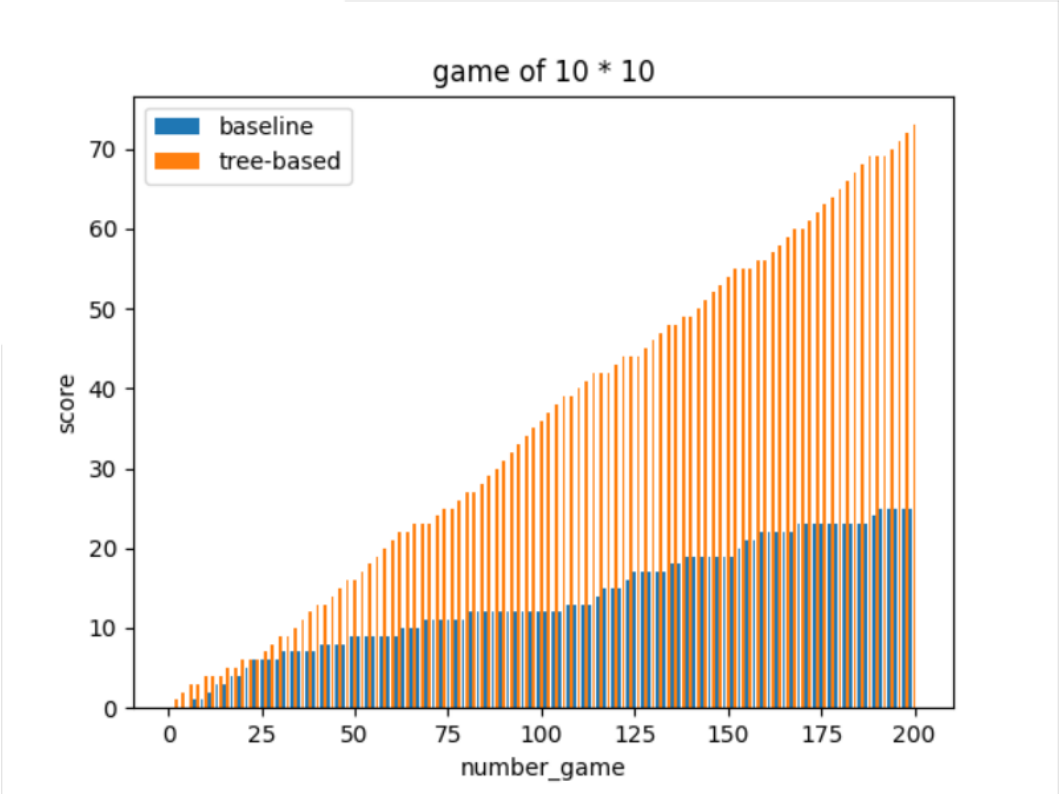


Figure: Score of TreeNN vs Baseline for Version 3

8. Conclusion

We successfully implemented all the required implementation and experiment including the modified domain for checkers, baseline AI, tree-based AI and tree+NN AI. In most of our experiments, the results are quite good. The most significant result is our tree-based AI. It has almost a 100% win rate against baseline AI and the performance is better in large games. The most challenging part will be constructing an efficient small training dataset. We do not know what kinds of data can give our neural network better understanding of the game strategy. So it's hard for us to propose a 500 dataset that can give our CNN a good performance.

For further improvement, there are several aspects that we can do in the future. We can propose a better evaluation for the MCTS child selection process. The configuration of our neural network can be improved. We only use 500 pairs of data as training data, which is relatively small for a neural network training. A bigger dataset will surely improve the performance to a large extent. Thus, constructing a larger and better training dataset will be the highest priority for future work.

In conclusion, monte-carlo tree search and convolutional networks are very good strategies for a board game. Looking forward to making more in the future.

Reference

1. Joris Duguépéroux, Ahmad Mazyad, Fabien Teytaud, Julien Dehos. Pruning playouts in Monte-Carlo Tree Search for the game of Havannah. The 9th International Conference on Computers and Games (CG2016), Jun 2016, Leiden, Netherlands. Ffhal-01342347f
2. Ziad SALLOUM,Monte Carlo Tree Search in Reinforcement Learning.Feb 17, 2019,https://towardsdatascience.com/.