CIS 667 – Project Report

AI-Blackjack

https://github.com/DolorHunter/AI-Blackjack

Zixiang Wang, Kevin Lin

NetID: zwang161, ltsungha

Contents

CIS 667 – Project Report 1
AI-Blackjack 1
Introduction 3
Domain 4
Tree AI description 5
Neural Network 6
Tree-based AI experiment 7
Neural Network experimental 10
    Zixiang Wang: 10
    Kevin Lin: 14
Conclusion 18
Bibliography 19

Introduction

In this project, we set out to develop an artificial intelligence (AI) player for the popular card game Blackjack. Blackjack is a strategic game that requires players to make complex decisions based on the cards they hold and the actions of their opponents. To tackle this challenge, we employed a combination of Monte Carlo Tree Search (MCTS) and Convolutional Neural Network (CNN) techniques.

MCTS is a well-known tree search algorithm that uses random sampling to evaluate and select the best move at each decision point in the game. It has been successfully applied to a wide range of strategic games, including chess, Go, and poker. In our project, we used MCTS to guide the decision-making process of our AI player and help it choose the most advantageous moves.

In addition to MCTS, we also employed a CNN, which is a type of machine learning model that is particularly effective at processing and analyzing images. CNNs have been widely used in tasks such as image classification, object detection, and face recognition, and we leveraged their ability to learn and recognize patterns to improve the performance of our AI player.

Through a series of experiments, we were able to collect a range of data and draw several important figures that illustrate the performance of our AI player. These included win rate, node count, and final score figures, as well as train error/test error figures if applicable. Our results showed that the combination of MCTS and CNN was able to effectively navigate the complex decision-making process involved in Blackjack and achieve a high level of performance. This demonstrates the potential of these techniques for developing intelligent agents that can successfully play and even outperform humans in strategic games.

Domain

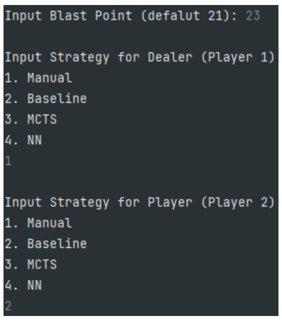
Blackjack is a card game played with a standard 52-card deck. The goal of the game is to have a hand value that is greater than the dealer's hand value without going over 21, which is known as "busting." In this version of the game, there are two players: the dealer and the player. The blasting point is set at 21 by default, but it can be modified as we want.

The state of each turn is a tuple of poker, dealer and player. Poker is class instantiation of Poker class, dealer and player are instantiation of Player class. In Poker class, there are function and attributes like cards, shuffle, turn and next card. And in Player class, there are function and attributes like points, cards on hand, stop sign, alive sign and action.

At each step, there are two action for dealer and player, which is "hit" and "stop". Hit means player want to take another card in hand, stop means player decided not take any cards.

There is a blasting point, if any player get more points than the blasting point, he lost the game. If no one exceed blasting point, the game continue, until both players choose stop action, and the player with more points in their hand win the game. If the dealer win the game, the final score is 1, and if the player win the game, score is -1, if tie score is 0.

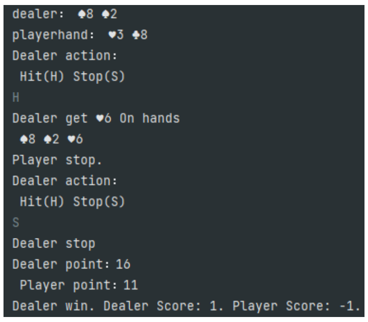
Here is an example about how game is played:



```
Input Blast Point (defalut 21): 23

Input Strategy for Dealer (Player 1)
1. Manual
2. Baseline
3. MCTS
4. NN
1

Input Strategy for Player (Player 2)
1. Manual
2. Baseline
3. MCTS
4. NN
2
```



```
dealer:  ♠8 ♠2
playerhand:  ♥3 ♣8
Dealer action:
 Hit(H) Stop(S)
H
Dealer get ♥6 On hands
 ♠8 ♠2 ♥6
Player stop.
Dealer action:
 Hit(H) Stop(S)
S
Dealer stop
Dealer point: 16
 Player point: 11
Dealer win. Dealer Score: 1. Player Score: -1.
```

As the game start, it will ask about the Blast Point, default by 21. And we choose 23. Then it will ask for the strategies for Dealer and Player. We choose manual strategy and baseline strategy.

After initialized the game, Dealer get 8 and 2 in hand, and player get 3 and 8. And Dealer choose hit at first turn, getting 6 in hand and make total points to 16. Player chooses to stop. And next turn Dealer chooses to stop. Both players not exceed blasting point, so we need to comparing points, since Dealer get 16 points and Player only get 11 points, therefore Dealer win and final score is 1.

Tree AI description

We choose MCTS as our Tree AI algorithm. MCTS is a well-known tree search algorithm that uses random sampling to evaluate and select the best move at each decision point in the game. And it is cited from MCTS.ipynb.

Pseudo code:

1. Starting at the root node, select a child node to expand based on the upper confidence bound of each child.
2. Expand the selected child node by adding a set of children to it.
3. Simulate a game from the current node until it is completed and return the result.
4. Backpropagate the playout result through the tree, updating the statistics (such as the number of visits and average score) for each node visited.

We simulate the game from node followed the rules, if player does not blast or stop, player has two children, hit and stop. If player is stop, it has one child as same as current state.

And the method we're using for choosing child is UCT, we override a rollout function called rollout2, and it can pass best children we get, and we'll use the action attribute in best children to pass action to node and determine the action for node. Also, we've manually prune the unwanted leaf from children, make sure every children is no redundancy. The score estimate is also modified, now it is dependent on the role of player, since we decided Dealer win is 1 and Player win is -1, so we also make sure if it's Dealer score convergence to 1, and Player score to -1.

We'll do 500 rollouts to get the optimized children and using the action to play. Although we won't get best results for just 500 rollouts, but it is rather efficient way to find the best children.

My baseline AI is built with a threshold with 15, also it can be called as dealer blast. It is the value of blasting point minus threshold. This idea supported by "Winning at Blackjack - 21 Points Card Counting Method.", and it has a high winning rate, sometimes even beat MCTS.

Neural Network

Neural network is a two-layer neural network (NN) with a fully connected linear layer at the end. The NN takes in a 5-element input tensor and applies a linear transformation to it using a fully connected layer with 10 output units, followed by a ReLU activation function and a flatten layer. The output is then passed through a final fully connected layer with 1 output unit to produce the final prediction. The weights and biases of the fully connected layers are learned during the training process. The output of the NN is a scalar value representing the predicted value for the current game state.

We use NN to estimate the utility of current state, like UCT. After combined with MCTS, it will help calculate the score for current state and pick the highest score children as best children. The model is inspired by. CNN_MNIST.ipynb.

The trainable parameters would include the weights and biases of the two fully connected layers. The weights and biases of these layers are learned during the training process in order to minimize the difference between the predicted output and the true output for the training data.

The optimization technique being used is mini-batch SGD, which is a variant of SGD that processes the training data in small batches rather than all at once. This allows the model to make updates to its parameters more frequently, which can often lead to faster convergence and better generalization.

And the training data is generated by all possible cases in Blackjack after abstraction. The input has 5 numbers, dealer points, player points, action, blasting point, turn. Therefore, it is not very hard to come up with all the cases under blasting point. And the output is several estimates score, if blasting then got zero, else if closer to blasting point, get higher score. The data structure of input is an array with 5 numbers.

We've tried some different configurations for the models, like adding new layers in the middle, or adding more neurons, change the shape of the network. However, since this is a simple task, the improvement is limited.

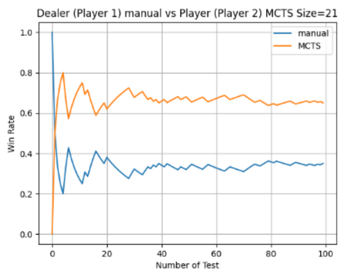
Tree-based AI experiment

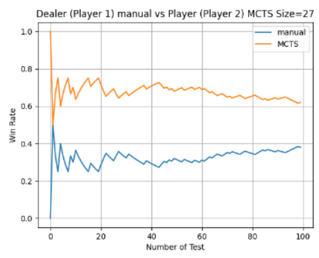
The size of the problem is Blasting Point, and it can be modified at the start of game or test. In the experiment, I've tried different blasting point like 21, 27, 34, 42, 50. We set the number of players, actions to constant. And we use one poker in the game, but it is shuffled in every game.

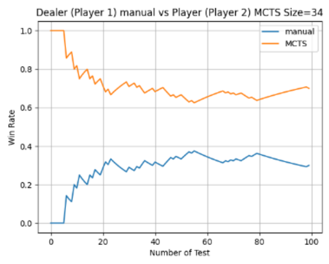
I generate three figures for the results of experiments, to show the Winning Rate, Node Visited Counts and Final Score distributions.

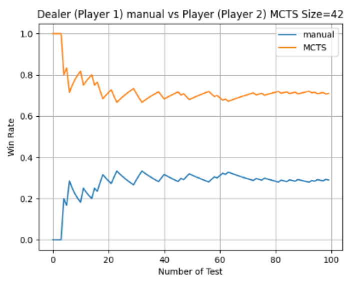
Here are the results:

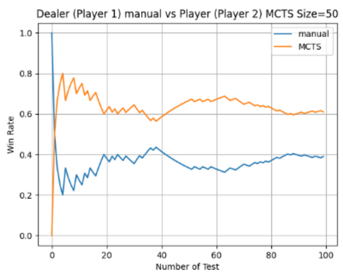
Winning Rate of Manual (random) vs MCTS:



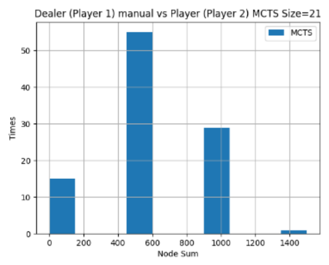


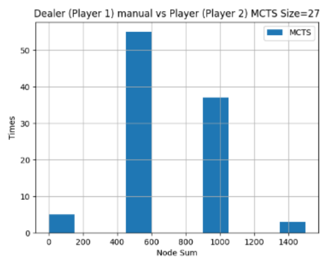


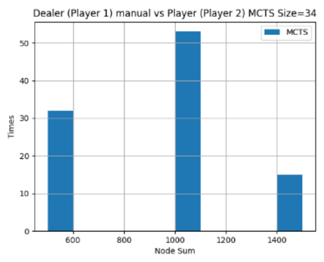


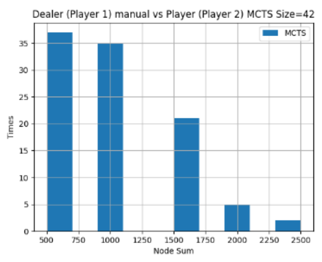


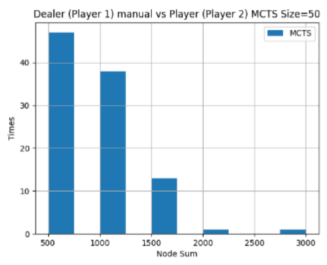
Node Counts of Manual (random) vs MCTS:



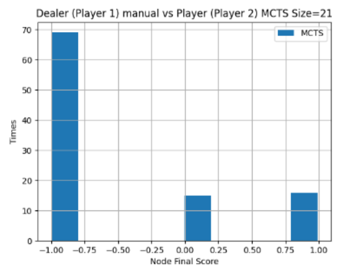


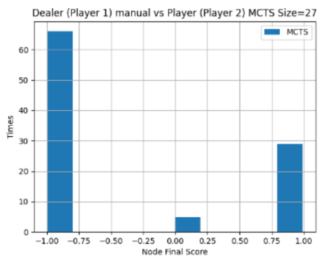


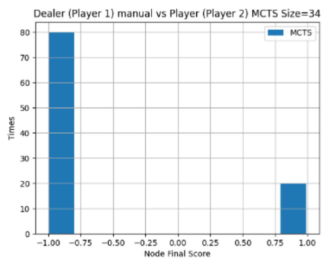


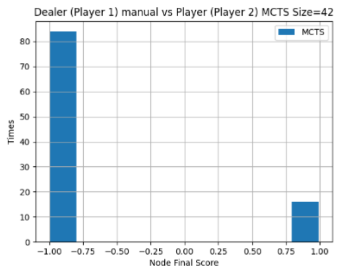


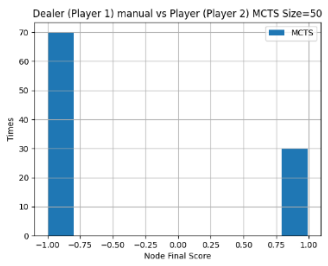
Final Score of Manual (random) vs MCTS:



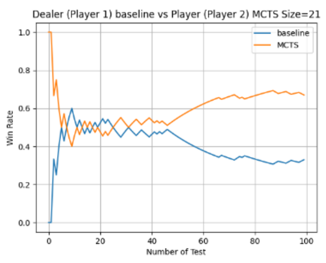


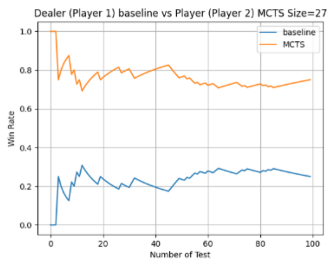


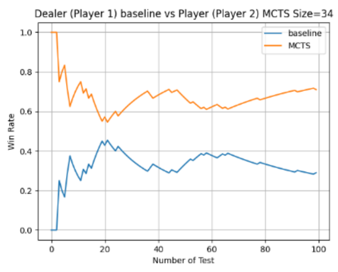


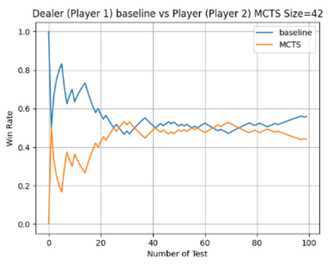


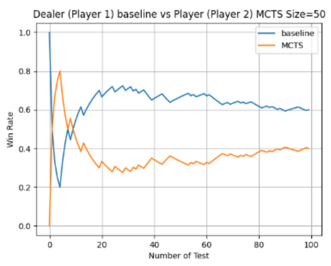
Winning Rate of Baseline (threshold) vs MCTS:



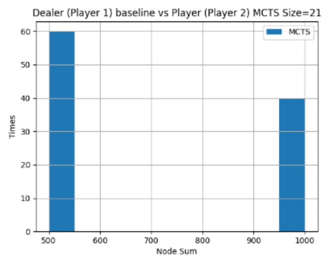


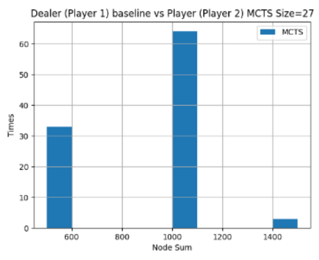


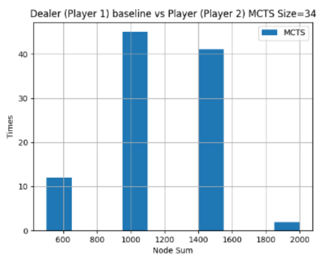


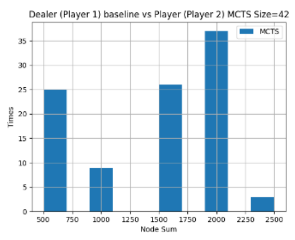


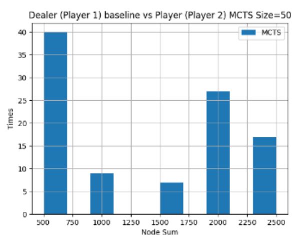
Node Counts of Baseline (threshold) vs MCTS:



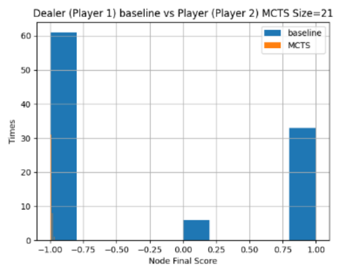


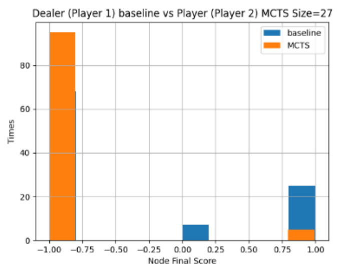


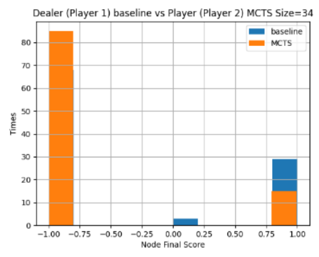


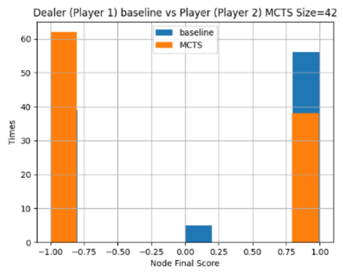


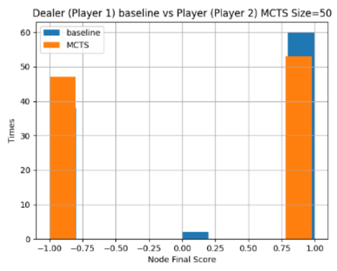
Final Score of Baseline (threshold) vs MCTS:











Conclusion:

We can tell MCTS beats Manual (random) for sure, but Baseline (threshold) performance is better than expected. For Node Count, as the size increase, the upper bound of visited node increase, the distribution feature is no obvious, but it meets normal distribution in most of the cases. For Final Score, because we only have 3 cases, Dealer win as 1, Player win as -1, tie as 0, so the higher winning rate player has more win case, e.g., for Dealer it is 1.

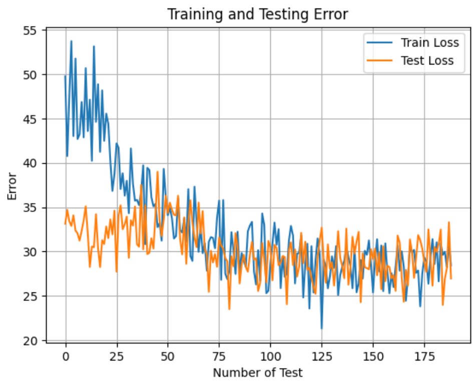
Neural Network experimental

Zixiang Wang:

```
cnn = tr.nn.Sequential(
    tr.nn.Linear(5, 10),
    tr.nn.ReLU(),
    tr.nn.Flatten(),
    tr.nn.Linear(10, 1)
).to(device)
```

Neural network is a two-layer neural network (NN) with a fully connected linear layer at the end. The NN takes in a 5-element input tensor and applies a linear transformation to it using a fully connected layer with 10 output units, followed by a ReLU activation function and a flatten layer. The output is then passed through a final fully connected layer with 1 output unit to produce the final prediction. The weights and biases of the fully connected layers are learned during the training process. The output of the NN is a scalar value representing the predicted value for the current game state.

Here is the learning curve of training and testing error:



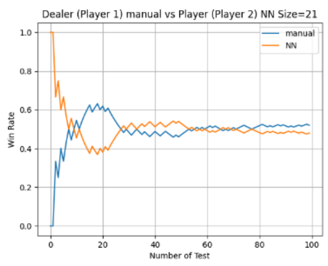
As you can see, the train loss is decreasing during the training, and test loss stay low since it is tested after the model get trained. The training set and testing set is 9:1.

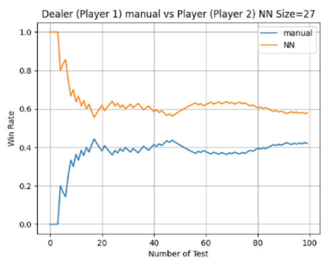
The size of the problem is Blasting Point, and it can be modified at the start of game or test. In the experiment, I've tried different blasting point like 21, 27, 34, 42, 50. We set the number of players, actions to constant. And we use one poker in the game, but it is shuffled in every game.

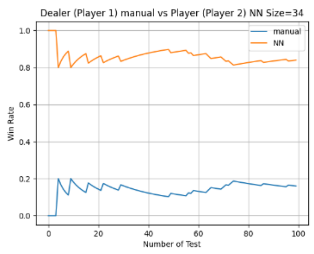
I generate three figures for the results of experiments, to show the Winning Rate, Node Visited Counts and Final Score distributions.

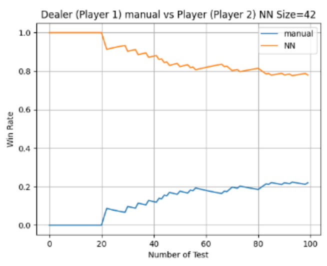
Here are the results:

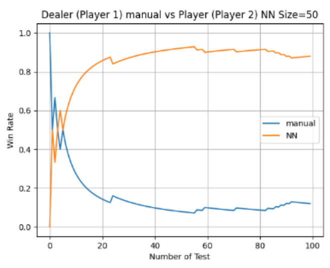
Winning Rate of Manual (random) vs NN:



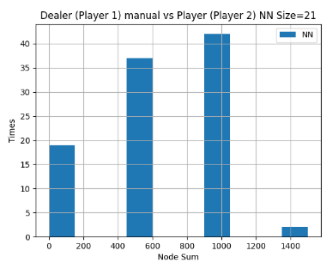


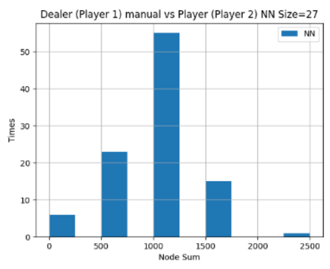


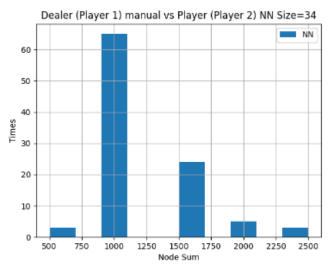


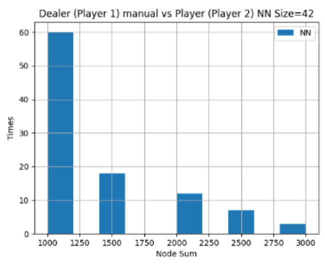


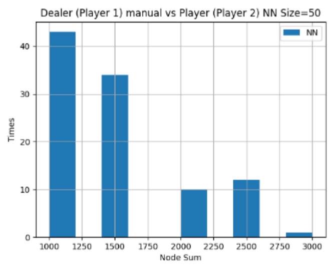
Node Counts of Manual (random) vs NN:



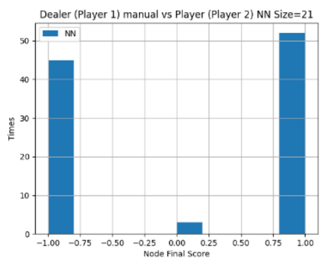


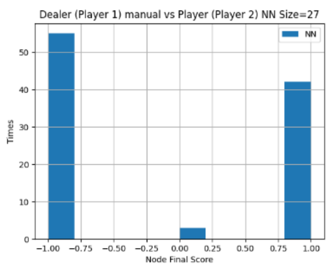


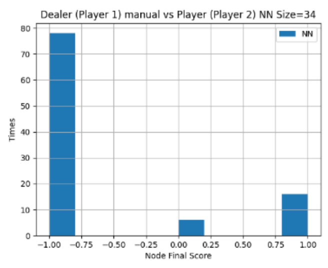


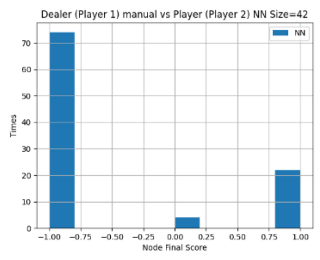


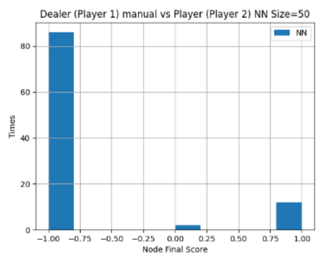
Final Score of Manual (random) vs NN:



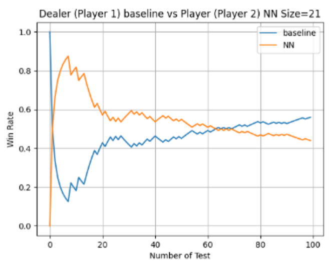


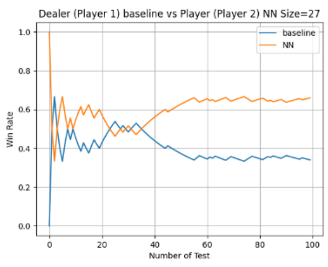


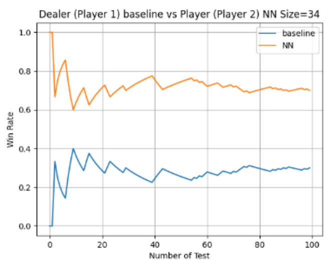


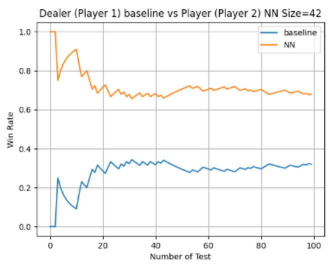


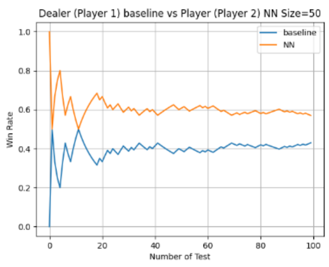
Winning Rate of Baseline (threshold) vs NN:



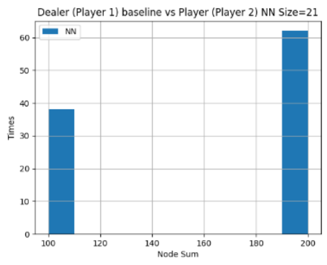


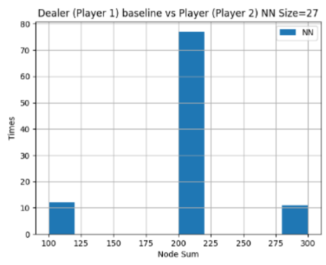


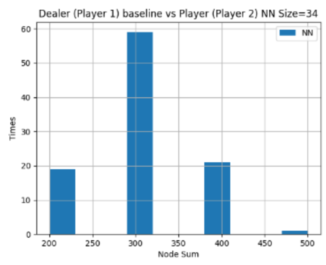


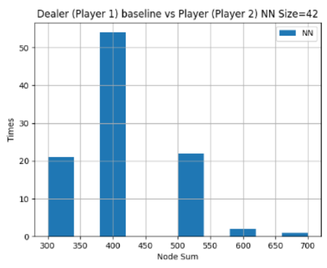


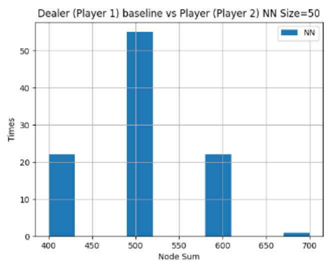
Node Counts of Baseline (threshold) vs NN:



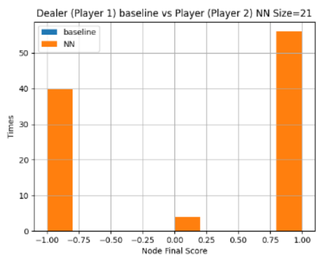


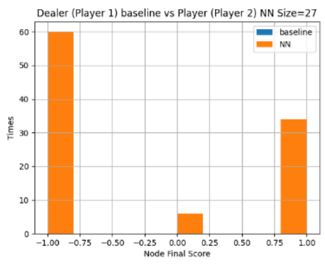


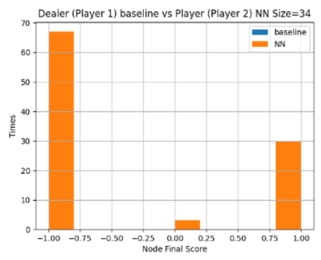


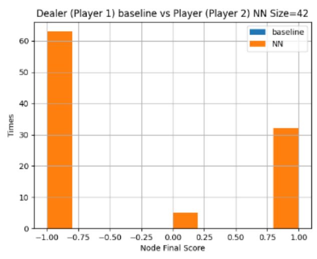


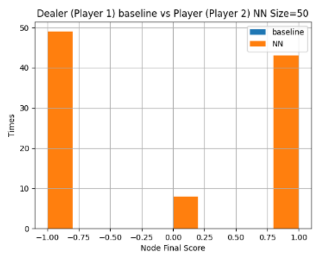
Final Score of Baseline (threshold) vs NN:









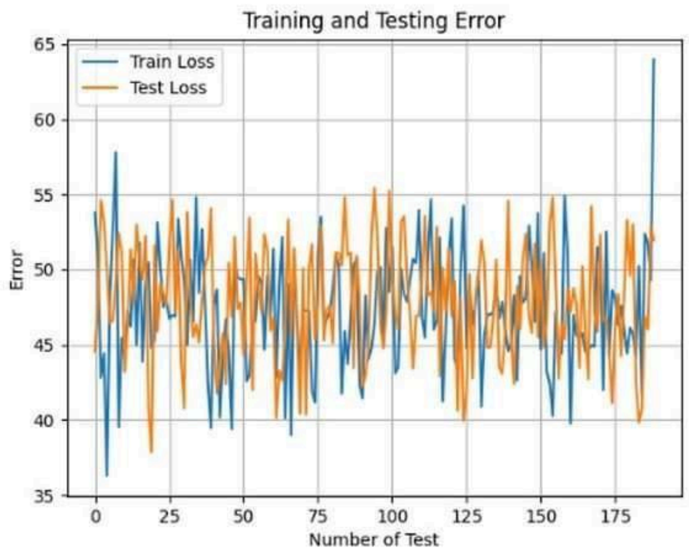


Conclusion:

We can tell NN dominate Manual (random) and Baseline (threshold) when Size (Blasting Point is large), which is better than expect. My expect is about same win rate to MCTS, but NN is clearly better and slower. For Node Count, as the size increase, the upper bound of visited node increase, the distribution is more clearly to MCTS, it meets normal distribution for sure. For Final Score, because we only have 3 cases, Dealer win as 1, Player win as -1, tie as 0, so the higher winning rate player has more win case, e.g., for Dealer it is 1.

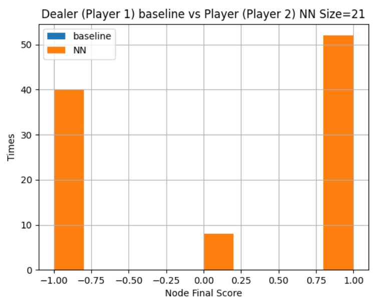
Kevin Lin:

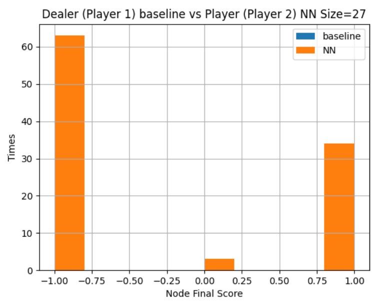
I tried two linear layer and takes 5 element input with 10 output units. Picking ELU activation function. The output is then passed through a final fully connected layer with 1 output unit to produce the final prediction.

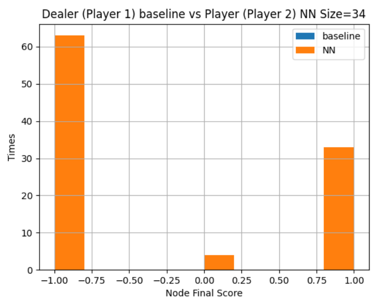


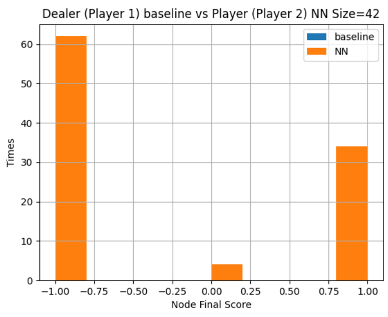
As the result the training error and the number of test still keep high after over 175 test.

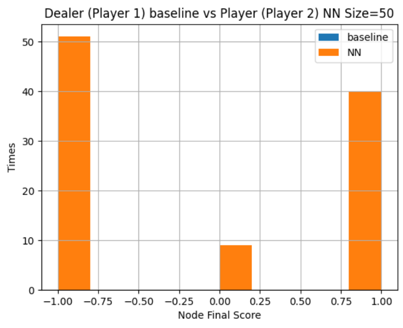
Final score baseline vs NN



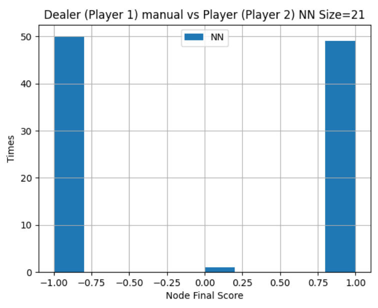


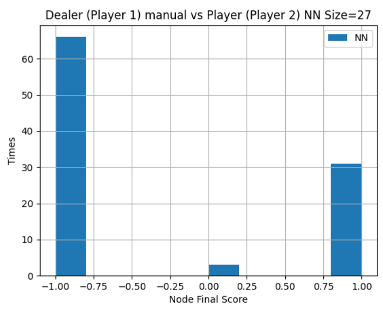


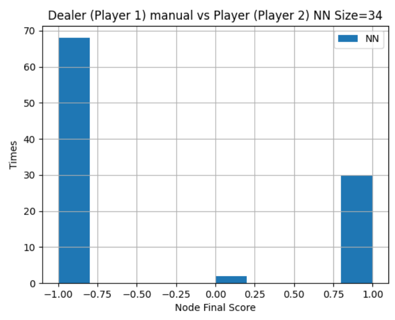


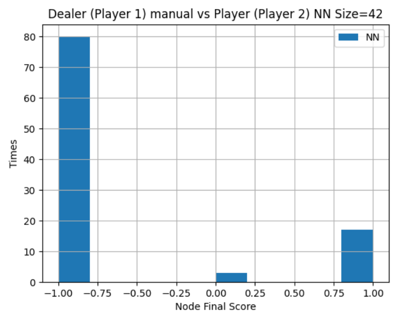


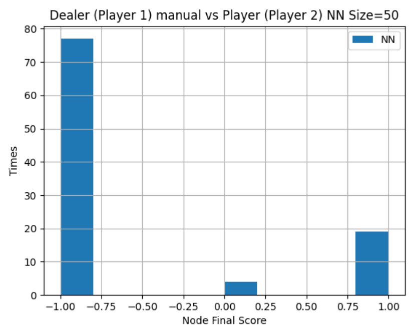
Final score manul vs NN



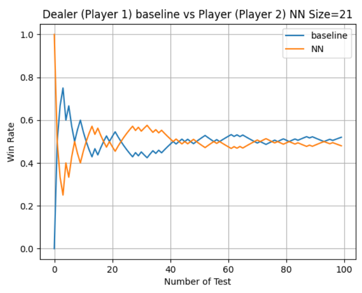


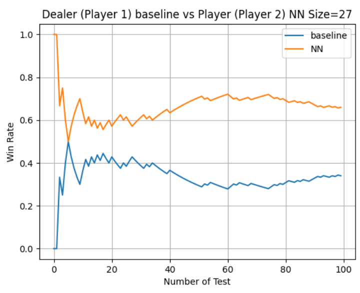


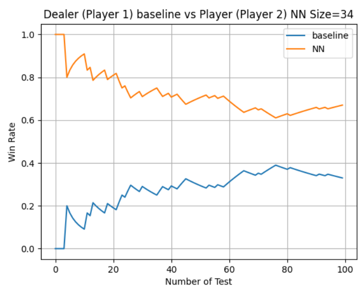


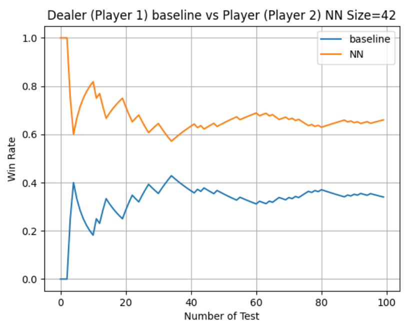


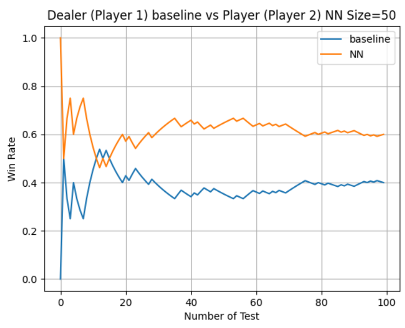
Win rate baseline vs NN



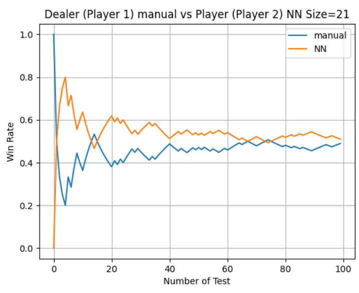


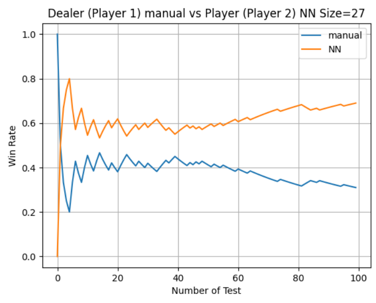


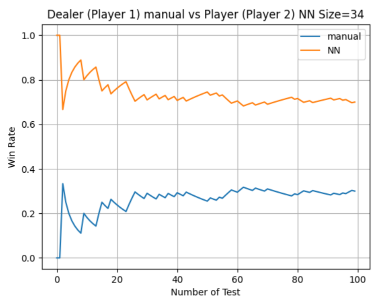


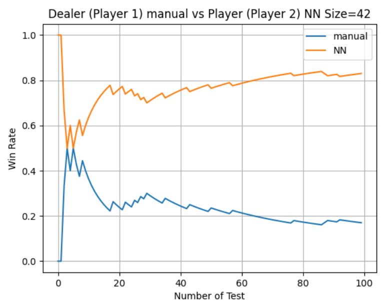


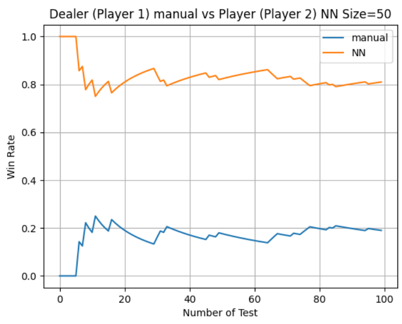
Win rate manual vs NN



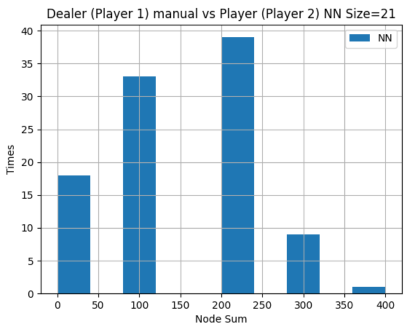


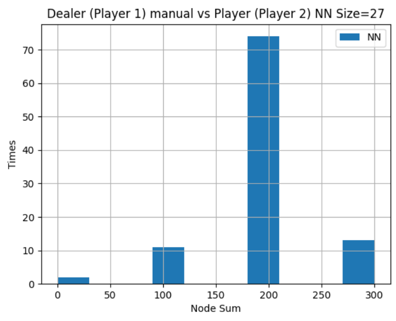


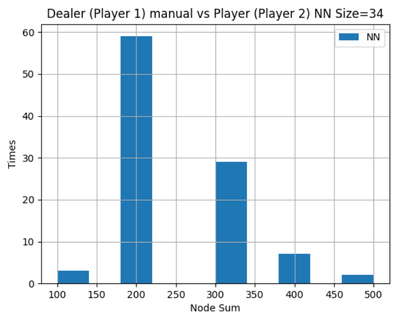


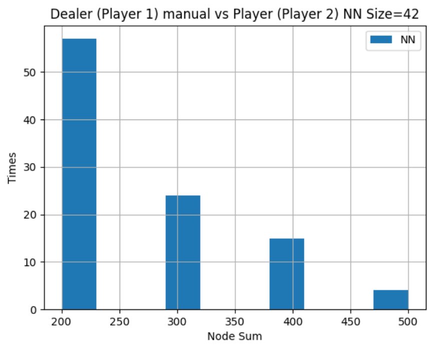


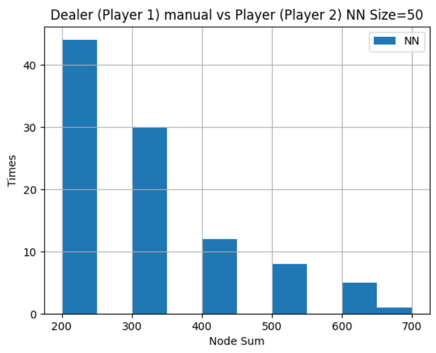
Node count mannul vs NN



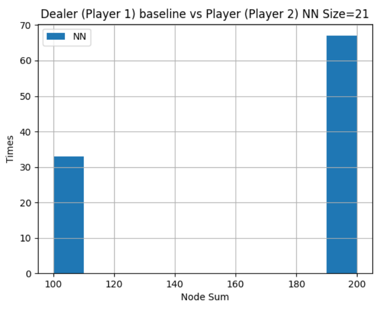


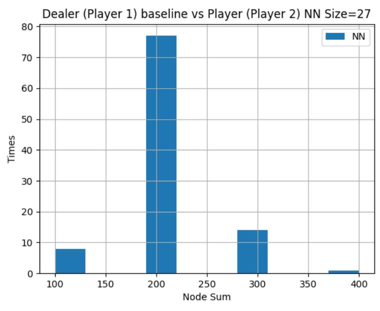


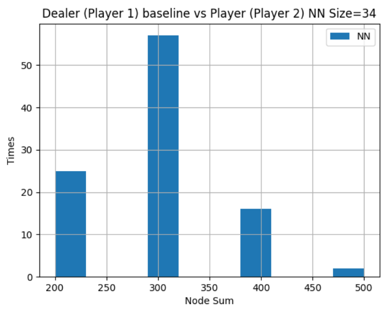


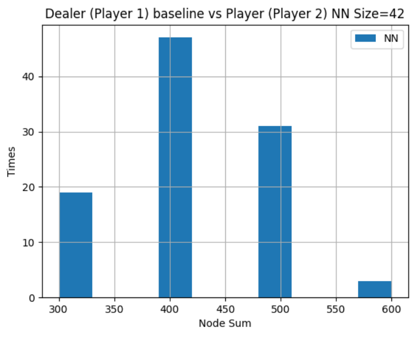


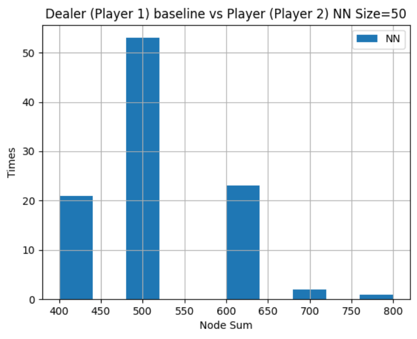
Node count baseline vs NN











Conclusion
For this experiment my neural network perform looks good, comparing to MCTS I have higher win rate. But while comparing to baseline win rate has drop a little bit.

Conclusion

In this project, we build domain for Blackjack, and four strategies for the game, including manual (random in test mode), baseline (threshold from blasting point), MCTS, and NN. The MCTS performance the best, based on the win rate and time consuming. We're using NN as a children picking algorithm like UCT, and it take more time than UCT, therefore MCTS has better performance than NN. But if we set a small threshold for baseline, the performance is really great and has highly chances beating MCTS.

In the case that NN with Flatten layer performance is better than without the layer. It will increase 5% of accuracy. And the most challenging part is MCTS and NN, we spend about a week to really understand most of the part from MCTS and NN model. NN is really hard to implement by ourself, even though tried really hard, the accuracy and performance is still no very good.

If we have more time to put into the project, the priority work is to figure out a better model for NN, can get better accuracy and more fit to the actual score in the state. We think there are huge potential for the implement of NN in the project, but sadly we couldn't play to their strengths.

Bibliography

1. Katz, Garrett. "MCTS.ipynb." Colab.
2. Katz, Garrett. "CNN_MNIST.ipynb." Colab
3. Macau, Gambling. "Winning at Blackjack - 21 Points Card Counting Method." Medium.
4. Arnaud, Buzzi, "The statistics of Blackjack." Towards Data Science.
5. Yiu, Tony. "Teaching A Neural Net To Play Blackjack." Towards Data Science.