**Blackjack AI with Monte Carlo Tree Search and Neural Network**

**CIS667 Term Project Report**

Jiaqi Ji jji107

Jinsi Li jli344

Shihong Yang syang60

Xiaozhu Wang xwang240

**Introduction:**

In the project, we implement a Blackjack game, a globally popular banking game known as Twenty-One, and an AI player of the game. We use Monte Carlo Tree Search as the tree search algorithm. We will also implement Neural Network model to prioritize child nodes of the search tree. Different Neural Network models will be tested in the AI player. For example, we use different structures in the hidden layers and organize input in different ways.

**Game:**

We implement the ‘Blackjack’ game for the project. Blackjack is a globally popular banking game known as Twenty-One. It is a comparing card game between one or more players and a dealer, where each player in turn competes against the dealer. Players do not compete against each other.[1]

The game uses one or more decks of 52 cards. The basic rule of the game is similar to the most widely played version:

* Players are each dealt two exposed cards. The dealer is also dealt two cards. The value of cards 2 through 10 is their pip value. Face cards (Jack, Queen, and King) are all worth ten. Aces can be worth one or eleven. A hand's value is the sum of the card values.
* Players are allowed to draw additional cards to improve their hands, it’s called hit. Hit may bust a player.
* Different from the normal rules, in each turn, a player can swap with other players. To make sure the game won’t loop infinitely with swapping the same card many times. The swap will be totally random: When a player swaps, the other player it will swap with and the cards swapped will be chosen randomly (by program). It means, swap can also make a player bust.
* Players are allowed to stop make its hand on its turn if they want.
* If a player stops or busts, it is stopped. No more hit or swap for the player.
* Once all the players have completed their hands, it is the dealer’s turn. The dealer hand will not be completed if all players have either busted or received blackjacks. The dealer will hit until its hand has 17 or more points.
* Players win by not busting and having a total higher than the dealer, or not busting and having the dealer bust, or getting a blackjack without the dealer getting a blackjack. If the player and dealer have the same total (not counting blackjacks), this is called a "push", and the player typically does not win or lose money on that hand. Otherwise, the dealer wins.

We use a list deck to represent the cards which are not distributed to players and a list to represent the cards which have already been distributed. The deck list will be shuffled before the game starts. Whenever a player draw from the deck, a card will be popped from the list, and added to the used card list and the player’s hand.

Each player in the game is represented by a player object, and it has a list called hand, which represents the hand a player has. When a player gets a card (from the deck or from other player’s hand), the card will be added to the hand list.

The dummy player, which represents the non-AI player, will make decision randomly. And the AI player, will make decision by its algorithm and model. The action will be Hit, Swap or Stop. After the action is processed, the total of the current players hand will be calculated. If it busts, it stops.

The dealer’s hand is also represented by a list. Once all players stop, the dealer’s hand will be checked. If the total of dealer’s hand is less than 17, the dealer hits until its hand greater than 17. After that, check and compare the total between dealer and each players.

For example, we use 2 decks of 52 cards and we have 2 players. At the beginning, the dealer draws 2 cards [5, K] from the deck, player 0 draws 2 cards [2, 7], player 1 draws 2 cards [J, K]. In the first turn, the player 0 chooses Hit, so it draws a card [8] from the deck, so right now it has [2,7,8]. The player 1 choose swap, it swap J with player 0’s 7, so player 1 has [K,7] and player 0 has [2,8,J]. In the next turn, since the player 0 has already had a total 20, it decides to stop. And player 1 hits a 8, it busts. Now all players stop, because the dealer has a total 15 less than 17, dealer hits a 6 and gets [5,6,K]. The dealer gets a blackjack, so player 0 loses.

The number of decks used in the game and the number of players are two variables we will use in the project to make the size of problem varied. The number of decks means how many deck of 52 cards (2 to 10, J, Q, K and A for 4 different colors) will be used in the game as the overall deck. And the number of players is how many players join the game. This value should be greater than 1 since we have at least on AI player.

We make some reference from Github[2] for the implementation of the game.

**Tree Search:**

We use the Monte Carlo Tree Search in AI player. The focus of MCTS is on the analysis of the most promising moves, expanding the search tree based on random sampling of the search space[reference].

Each round of Monte Carlo tree search consists of four steps[3]:

* Selection: start from root R and select successive child nodes until a leaf node L is reached. The root is the current game state and a leaf is any node from which no simulation (playout) has yet been initiated. The section below says more about a way of biasing choice of child nodes that lets the game tree expand towards the most promising moves, which is the essence of Monte Carlo tree search.
* Expansion: unless L ends the game decisively (e.g. win/loss/draw) for either player, create one (or more) child nodes and choose node C from one of them. Child nodes are any valid moves from the game position defined by L.
* Simulation: complete one random playout from node C. This step is sometimes also called playout or rollout. A playout may be as simple as choosing uniform random moves until the game is decided (for example in chess, the game is won, lost, or drawn).
* Backpropagation: use the result of the playout to update information in the nodes on the path from C to R.

The Monte Carlo tree search can be written in the pseudo-code:

While (compute\_usage < compute\_budget):

Node = Selection(Root)

Expand\_node = Expand(Node)

Simulation(Expand\_node)

Backpropagation(Expand\_node)

Return BestNext(Root)

For the Blackjack game, the question can be transformed to how to get a number as close as possible to 21. And the root of the search tree is the current state of the game, which includes the information like current hand, used cards and remain cards (without order). Since our target is to get a number as close as possible to 21, and the action can make the total greater or smaller (swap a smaller number), we always consider the depth limit unless it actually get a 21, instead of a win/loss state. The maximum depth we use is 20.

**Machine Learning Model:**

We use a Neural Network model as the machine learning model in the AI player. The features used to train the model are the action a player take, the current used cards, and the current total of hand. The target is if the total after the action. Since in each turn we keep the used card list and the player’s hand, we can simply get features we want. The used cards will be represented by a vector of 13 numbers, each position represents how many cards with corresponding value have been used in the game. The actions will be represented by a 0-1 vector of 3, each position represents an action type. The data is generated by running baseline (dummy player).

Since the Blackjack game result won’t be revealed until all players stop and dealer makes its hand, in the models we only predict the total value after the action taken. If the prediction is greater than 21, it means if the player takes this action, it may bust. In that case, the AI player won’t expand this node and the sub tree of the node is pruned. If the predicted value is less than or equal to the 21, the action should be good to expand.

We tried 4 different configuration of the NN. We change the hidden layers number and size of each hidden layers. The configuration of machine learning model for each member:

1. Jiaqi Ji: A NN with 2 hidden layers. Layer 1 has 100 neural. Layer 2 has 20 neural.
2. Jinsi Li: A NN with 2 hidden layers. Layer 1 has (input size \* 2) neural. Layer 2 has 10 neural.
3. Shihong Yang: A NN with 4 hidden layers. Layer 1 has 128 neural. Layer 2 has 64 neural. Layer 3 has 32 neural. Layer 4 has 16 neural.
4. Xiaozhu Wang: A NN with 3 hidden layers. Layer 1 has (input size \* 2) neural. Layer 2 has input size neural. Layer 3 has (input size / 2) neural.

The models are pre-trained with baseline data separately and dumped into files.

**Baseline:**

We use the dummy player as the baseline and generate training data for the machine learning model. The dummy player will always choose actions randomly and compete with dealer. The baseline runs 100 times for each different game setting. The game settings will be talked about later.

**Experimental results:**

Generally, a Blackjack game will have 8 different result for a single player:

1. The player gets a blackjack, the player wins
2. The dealer gets a blackjack, the dealer wins
3. The player busts, the dealer wins
4. The dealer busts, the player wins
5. The dealer gets a higher total than the player, the dealer wins
6. The player gets a higher total than the dealer, the player wins
7. The dealer gets a blackjack after deal, the game stop immediately
8. Both the dealer and the player get a blackjack, it’s called push.

Since the 7th case is very rare and the game process doesn’t start actually in this case, we don’t consider it in the statistic. And because we use ‘get a total as close to 21 as possible’, and push result is also good for the AI, and it is very rare, we also don’t record this situation. So we only collect statistic of the first 6 cases in each game. For each different configuration of machine learning model, we run 100 games for each different game setting. The game settings we have are:

2 decks, 2 players

2 decks, 4 players

3 decks, 3 players

1 deck, 2 players

3 decks, 4 players

Below is the experimental results we get from a certain baseline data. Notice: since many processes in the game run randomly and are not predictable. The result may various.

The uniform baseline data used to train machine learning model is generated with 500 games. Each game setting provides 100 games. All baseline data in different setting is combined together to train the models. The statistic result of baseline is:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 | Case 4 | Case 5 | Case 6 | Win Rate |
| 2 decks, 2 players | 8 | 8 | 74 | 36 | 49 | 12 | 30% |
| 2 decks, 4 players | 16 | 22 | 145 | 67 | 105 | 13 | 26% |
| 3 decks, 3 players | 6 | 12 | 93 | 69 | 88 | 10 | 30% |
| 1 deck, 2 players | 12 | 5 | 58 | 35 | 62 | 12 | 32% |
| 3 decks, 4 players | 17 | 27 | 133 | 63 | 106 | 26 | 28% |

The overall winning rate is around 30%.

The tree search baseline is generated by running the game with tree search strategy but without machine learning model. It also has the 500 games data running with the 5 different game setting. The result of tree search baseline is:

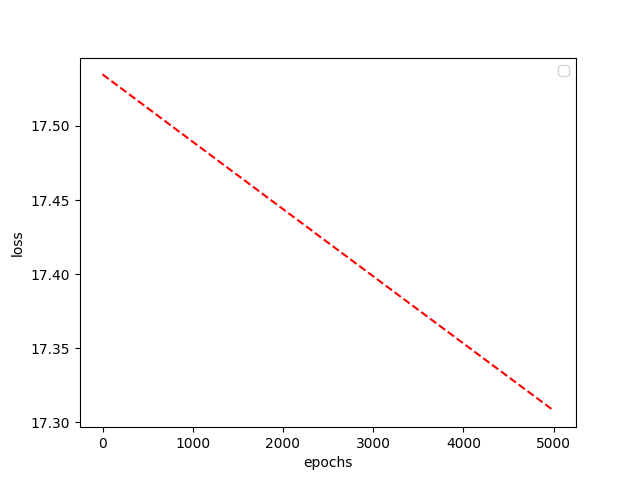
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 | Case 4 | Case 5 | Case 6 | Win Rate |
| 2 decks, 2 players | 5 | 1 | 27 | 22 | 26 | 7 | 39% |
| 2 decks, 4 players | 6 | 4 | 39 | 16 | 21 | 6 | 30% |
| 3 decks, 3 players | 2 | 5 | 23 | 22 | 31 | 6 | 33% |
| 1 deck, 2 players | 3 | 6 | 35 | 15 | 25 | 6 | 27% |
| 3 decks, 4 players | 4 | 4 | 33 | 21 | 30 | 4 | 30% |

The overall winning rate is a lit bit higher than the baseline. But for 1 deck 2 players case, the winning rate goes lower.

And the node visited for the tree search baseline is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 decks, 2 players | 2 decks, 4 players | 3 decks, 3 players | 1 decks, 2 players | 3 decks, 4 players |
| 9980 | 9917 | 9316 | 9539 | 9720 |

The first machine learning configuration is handled by Jiaqi Ji. The configuration is a NN with 2 hidden layers. Layer 1 has 100 neural. Layer 2 has 20 neural. The order of activate function is sigmoid, tanh, and no activate. The loss function is MAE. The learning curve of model 1 is:



The statistic result of running with model 1 is:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 | Case 4 | Case 5 | Case 6 | Win Rate |
| 2 decks, 2 players | 5 | 3 | 24 | 22 | 27 | 6 | 38% |
| 2 decks, 4 players | 9 | 6 | 35 | 11 | 22 | 3 | 27% |
| 3 decks, 3 players | 1 | 7 | 35 | 21 | 27 | 7 | 30% |
| 1 deck, 2 players | 6 | 2 | 30 | 16 | 39 | 3 | 26% |
| 3 decks, 4 players | 3 | 4 | 31 | 16 | 33 | 7 | 28% |

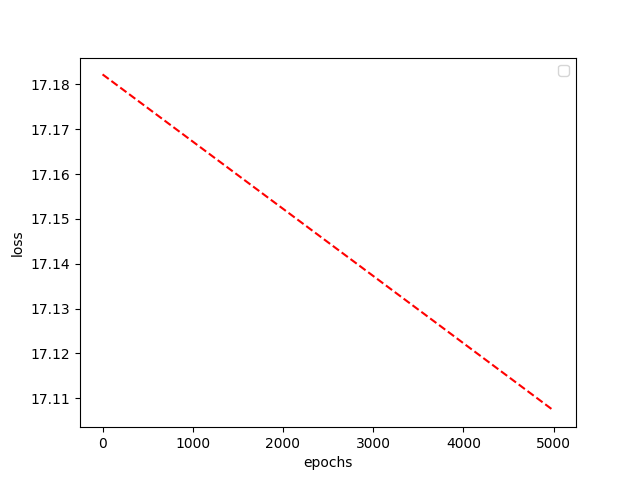
The overall winning rate is close to the tree search baseline.

And the node visited for the model 1 is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 decks, 2 players | 2 decks, 4 players | 3 decks, 3 players | 1 decks, 2 players | 3 decks, 4 players |
| 9446 | 9421 | 9797 | 9949 | 9785 |

Overall the visited nodes are less than the tree search baseline but the difference is not significant.

The second machine learning configuration is handled by Jinsi Li. The configuration is a NN with 2 hidden layers. Layer 1 has (input size \* 2) neural. Layer 2 has 10 neural. The order of activate function is sigmoid, sigmoid, and no activate . The loss function is MAE. The learning curve of model 2 is:



The statistic result of running with model 2 is:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 | Case 4 | Case 5 | Case 6 | Win Rate |
| 2 decks, 2 players | 7 | 6 | 29 | 11 | 30 | 6 | 27% |
| 2 decks, 4 players | 6 | 2 | 34 | 16 | 27 | 2 | 28% |
| 3 decks, 3 players | 3 | 3 | 31 | 18 | 32 | 6 | 29% |
| 1 deck, 2 players | 2 | 10 | 36 | 22 | 21 | 2 | 28% |
| 3 decks, 4 players | 3 | 5 | 37 | 16 | 25 | 6 | 27% |

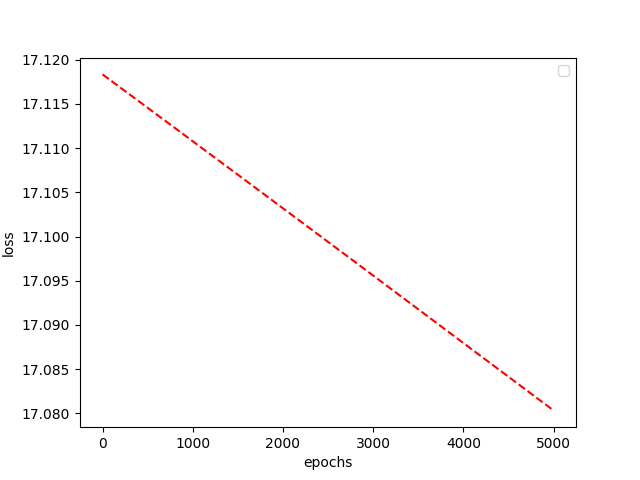
The overall winning rate is lower than tree search baseline.

And the node visited for the model 2 is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 decks, 2 players | 2 decks, 4 players | 3 decks, 3 players | 1 decks, 2 players | 3 decks, 4 players |
| 10026 | 9554 | 9578 | 10322 | 9695 |

The node visited is greater than baseline. That means the model prefer to predict the total after action less than 21.

The third machine learning configuration is handled by Shihong Yang. The configuration is a NN with 4 hidden layers. Layer 1 has 128 neural. Layer 2 has 64 neural. Layer 3 has 32 neural. Layer 4 has 16 neural. The order of activate function is tanh, sigmoid, tanh, and no activate. The loss function is MAE. The learning curve of model 3 is:



The statistic result of running with model 3 is:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 | Case 4 | Case 5 | Case 6 | Win Rate |
| 2 decks, 2 players | 6 | 5 | 28 | 18 | 34 | 6 | 31% |
| 2 decks, 4 players | 6 | 7 | 28 | 18 | 28 | 4 | 31% |
| 3 decks, 3 players | 3 | 7 | 27 | 24 | 30 | 2 | 31% |
| 1 deck, 2 players | 2 | 6 | 32 | 20 | 16 | 6 | 34% |
| 3 decks, 4 players | 2 | 7 | 32 | 20 | 25 | 6 | 30% |

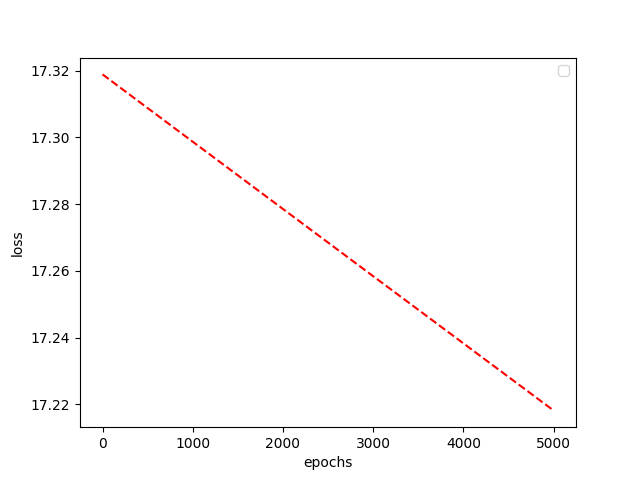
The overall winning rate is close to the tree search baseline.

And the node visited for the model 3 is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 decks, 2 players | 2 decks, 4 players | 3 decks, 3 players | 1 decks, 2 players | 3 decks, 4 players |
| 9996 | 9202 | 9408 | 9303 | 10167 |

The node visited in 2 decks, 4 players and 1 decks, 2 players cases are less than baseline, but in other cases, it’s higher than baseline.

The fourth machine learning configuration is handled by Xiaozhu Wang. The configuration is a NN with 3 hidden layers. Layer 1 has (input size \* 2) neural. Layer 2 has input size neural. Layer 3 has (input size / 2) neural. The order of activate function is sigmoid, tanh, sigmoid, tanh and no activate. The loss function is MAE. The learning curve of model 4 is:



The statistic result of running with model 4 is:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Case 1 | Case 2 | Case 3 | Case 4 | Case 5 | Case 6 | Win Rate |
| 2 decks, 2 players | 2 | 5 | 35 | 21 | 26 | 1 | 27% |
| 2 decks, 4 players | 10 | 11 | 35 | 14 | 17 | 5 | 32% |
| 3 decks, 3 players | 3 | 5 | 34 | 16 | 29 | 3 | 24% |
| 1 deck, 2 players | 4 | 4 | 27 | 23 | 28 | 8 | 37% |
| 3 decks, 4 players | 4 | 11 | 35 | 23 | 21 | 1 | 30% |

The winning rate of 2 decks, 2 players and 3 decks, 3 players are significantly lower, but for the smaller case like 1 deck, 2 players, it performs better.

And the node visited for the model 4 is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 decks, 2 players | 2 decks, 4 players | 3 decks, 3 players | 1 decks, 2 players | 3 decks, 4 players |
| 9131 | 9435 | 9194 | 9410 | 9170 |

The nodes visited is significantly less than the tree search baseline.

**Conclusion:**

Overall, we make an AI player with Monte Carlo Tree Search algorithm and a Neural Network model. For different game setting, the performance of the AI player is various. For some configuration, it performs well, but for some of them, it performs worse.

The most challenge part of it is thinking and implement the way to measure the state of the game. The Blackjack game is a Incomplete information game, and each state is not only determined by the choice of the player, and it is also determined by some random output, like the card actually drawn from the deck.

What to do next is to find a way to improve the performance/winning rate of the AI player. Based on our implementation and setting, the AI player is not significantly better than the dummy player. The reason may be various. The first possible reason is the data collection. The data may be not enough and not representative. We only use 500 games data as the training set of the Neural Network model, and due to the limit of the machine we use, the training process is also restricted. The other possible reason is the measurement of reward in the tree search. The distance between total of hand and 21 may not be a good measurement. So if we will move forward on that AI player, we may handle the two things first.

**Bibliography:**

[1]Blackjack. (2019). Retrieved 15 December 2019, from https://en.wikipedia.org/wiki/Blackjack

[2]python blackjack. (2019). Retrieved 15 December 2019, from https://gist.github.com/mjhea0/5680216

[3]Monte Carlo tree search. (2019). Retrieved 15 December 2019, from https://en.wikipedia.org/wiki/Monte\_Carlo\_tree\_search