CS 4100 Project Report

Blackjack Bot

Runyu Wang

Introduction:

In this project, I designed and implemented an agent that plays blackjack automatically (The rule of the game will be explained in detail later). This project is interesting because of the stochastic nature of the game. It would also be profiting if my agent can exhibit consistent positive (higher than 0.5) net win rate. Net win rate is given by:

$$\text{net win rate} = \frac{\text{win rate}}{\text{win rate} + \text{loss rate}} \qquad (1)$$

Although simple, the development of the game is highly volatile. The volatility comes from mainly two sources. First, the card drawn at any point is volatile, and each game of blackjack contains at least four card draws. Second, the nature of the game drives a confliction in gameplays: players want higher total than the dealer, yet a total larger than 21 counts as immediate loss. This confliction further magnifies the volatility in the game. To model this game is difficult: the expectation of any hidden cards is with high standard deviation. The average win rate of the game is as follow (https://wizardofodds.com):

Event	Win	Draw	Loss	net win
Probability	42.43%	8.48%	49.09%	46.36%

I first use large sample simulation trials (Monte Carlo Tree Search method) to combat the volatility. My MCTS agent with random simulation was able to yield the following result:

Event	Win	Draw	Loss	net win
Probability	43.05%	8.75%	48.2%	47.18%

I then compare the result to that of (1) a random agent, (2) a MCTS agent with simulation based on heuristics, and (3) a simple reflex agent that execute according to general heuristics.

Related Work:

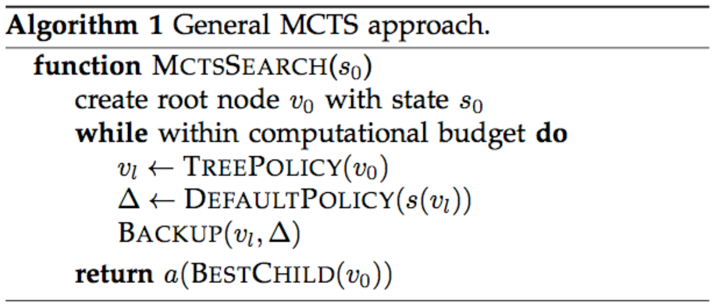
In *An Expectation-Maximization Algorithm to Compute a Stochastic Factorization from Data*, 2015, Barreto et al. used Hidden Markov Model, Stochastic Factorization Model, and reinforcement learning to teach their agent to play blackjack. In *Model-free Q-learning of Blackjack* by Finn, the author used model free method to teach his agent to play blackjack.

My method differs from both as it is not a learning agent: it simply runs simulations to form data that directs its action. It is much simpler in nature but also powerful.

The original blackjack environment is acquired freely from https://gist.github.com/mjhea0/5680216. It is modified from a functional program to an object-oriented program for an easier connection with agents.

Monte Carlo Tree Search uses large simulated sample to make decisions. In *Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search,* 2006, Coulom introduces the term Monte Carlo Tree Search, However Monte Carlo Method dates back to 1940s. Figure 1 is the pseudocode for Monte Carlo Tree Search algorithm

Figure 1



Algorithm 1 General MCTS approach.

```
function MCTSSEARCH($s_0$)
    create root node $v_0$ with state $s_0$
    while within computational budget do
        $v_l \leftarrow$ TREEPOLICY($v_0$)
        $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
        BACKUP($v_l, \Delta$)
    return $a$(BESTCHILD($v_0$))
```

Approach:

The rules of Blackjack are as follow: The game uses a deck of playing cards including only 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, and A, each with four copies. We assume that the dealer always resets such deck before each game. At the beginning of the game, both player and the dealer are dealt two cards. Both of the player's cards are shown and only one card from the dealer's hand is shown. Each number card counts as its face value. J, Q, and K are each worth 10. An Ace is worth 1 or 11 by player's choice. Suits are irrelevant. After the initial dealing, player has the option to hit (ask for one more card) or stand. If the total of a hand is greater than 21, such hand is a "bust" meaning an immediate loss. After the player chooses stand, dealer gets to decide whether he wants more cards or not. The dealer is a simple reflex agent where it always hit for a hand lower or equal to 17, and stand otherwise. The same bust rule applies to the dealer as well. After the dealer chooses to stand, the player and the dealer will compare their hand. The higher hand wins. We do not consider the options to split, double down, or surrender.

At each decision, the agent would have the following information: the player's hand and the showing card of the dealer's hand. Before it makes a decision, it will simulate 15000 black jack games from the given state. The essence is in the simulation policy. If we use a random simulation policy where it randomly returns ‘hit’ or ‘stand, it wouldn’t be able to have a good judgement since all future actions are random as well. The agent needs to perform sensible actions at the simulations for the agent to yield positive result. In other words, the agent is as good as its simulation policy. Based on common heuristics on the game (see strategy chart on appendix A), a decision is made based on: player current total, whether player has a soft hand (meaning that the player has an Ace that can be counted as 11 without bust), whether the showing dealer card is high (7 to A) or low, whether the dealer card is an Ace (calls for riskier plays). These factors generate a probability that the simulation will return ‘hit’. For example, if the player has a hand total of 19, it will have very low chance to return ‘hit’; on the other hand, if the player has a hand with total of 11, it will have a very high chance to return ‘hit’ in the simulation.

For the second trial, we use a random agent to randomly return ‘hit’ or ‘stand’. This trial is to eliminate trivial methods. For the third trial, we use the strategy chart as a simple reflex agent for the simulations. For the final trial, we use a simple reflex agent that plays according to exact heuristics instead of using Monte Carlo methods.

Evaluation/Results/Testing:

For testing, each agent will play 10,000 games, each with random dealing, and return the number of wins, number of draws, and number of losses. We can then calculate the net win rate of each trial using equation (1).

Results:

Average by chance:

Event	Win	Draw	Loss	Net win
Probability	42.43%	8.48%	49.09%	46.36%

Random agent:

Event	Win	Draw	Loss	Net win
Probability	28.50%	3.68%	67.82%	29.59%

MCTS agent with probability-based simulation policy:

Event	Win	Draw	Loss	Net win
Probability	43.05%	8.75%	48.20%	47.18%

MCTS agent with heuristic-based simulation policy:

Event	Win	Draw	Loss	Net win
Probability	42.98%	9.60%	47.42%	47.54%

Simple reflex agent using heuristics from appendix A:

Event	Win	Draw	Loss	Net win
Probability	42.98%	9.25%	47.77%	47.36%

All three agents performed respectably. Both MCTS agents and the simple reflex agent yields similar result. MCTS agent with heuristic-based simulation policy yields the highest net win rate. On average, using heuristics helps out perform the population by 1 percent in net win rate.

Conclusions and Future work:

Without taking advantage of using split, surrender, or double down, the MCTS agent with random simulation policy was able to out perform benchmark by 1 percent. The heuristic that guided the simulation policy is the key player as all heuristic-based agent were able to outperform the benchmark.

MCTS agents with smaller simulation size (1500) yields similar results but with a much higher variance. Due to the limitation of the machine, I could not further increase the amount of iteration while keeping the test runs manageable.

Although outperforming average players, the agents are still far from profiting. The next step would be taking in consideration of the neglected player options. It is expected to increase the net win rate if all options are considered.

Acknowledgements:

The game of blackjack is originally written by https://gist.github.com/mjhea0/5680216.

Reference Section:

(1) Austin Finn, *Model-free Q-learning of Blackjack*, Stanford University

(2) Andr´e M. S. Barreto, Rafael L. Beirigo, Joelle Pineau and Doina Precup, *An Expectation-Maximization Algorithm to Compute a Stochastic Factorization from Data*, Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

(3) Cameron Browne, *Monte Carlo Tree Search Slide*, Computational Creativity Group, Imperial College London

(4) Stuart J. Russell and Peter Norvig, *Artificial Intelligence: a modern approach,* Prentice Hall

(5) Justin Berkman, *Perfect Blackjack Strategy: 15 Charts to Help You Master the Game*, https://blog.prepscholar.com/blackjack-strategy

(6) *Variance in Blackjack*, https://wizardofodds.com/games/blackjack/variance/

Appendix A - Strategy Card

Hard	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**A**
4-7	H	H	H	H	H	H	H	H	H	H
8	H	H	H	Dh	Dh	H	H	H	H	H
9	Dh	Dh	Dh	Dh	Dh	H	H	H	H	H
10	Dh	Dh	Dh	Dh	Dh	Dh	Dh	Dh	H	H
11	Dh	Dh	Dh	Dh	Dh	Dh	Dh	Dh	Dh	Dh
12	H	H	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	Rh
16	S	S	S	S	S	H	H	H	Rh	Rh
17	S	S	S	S	S	S	S	S	S	Rs
18+	S	S	S	S	S	S	S	S	S	S

Soft	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**A**
13	H	H	Dh	Dh	Dh	H	H	H	H	H
14	H	H	Dh	Dh	Dh	H	H	H	H	H
15	H	H	Dh	Dh	Dh	H	H	H	H	H
16	H	H	Dh	Dh	Dh	H	H	H	H	H
17	Dh	Dh	Dh	Dh	Dh	H	H	H	H	H
18	S	Ds	Ds	Ds	Ds	S	S	H	H	S
19	S	S	S	S	Ds	S	S	S	S	S
20	S	S	S	S	S	S	S	S	S	S