Blackjack Bot with Reinforcement Learning

Ashok Koduru, Bhushan Mohite

ABSTRACT

The purpose of this paper is to implement Reinforcement techniques with Q Learning and Feature extraction in to the world famous game of Blackjack. The world of Blackjack and the agent are implemented as a Markov Decision Process. The agent receives training in different modes and will be played random sets of games to determine the winning probability at the end of a series of games.

KEYWORDS

MDP; Reinforcement Learning; Feature Extraction; Blackjack

1 INTRODUCTION

Blackjack is the most widely played card game in casinos around the world. It is usually played between 2 or more players and a dealer, where every player individually plays against the dealer, but players do not play against each other. Complex probability of determining the winner amongst the players by the output from choosing a particular sequence of cards makes the game a major subject of study to mathematics and scholars

The goal of the player is to beat the dealer in one of the ways

(1) Get a natural blackjack (21 points in first two cards), without a dealer blackjack
(2) achieve a total score higher than the dealer and less than 21
(3) dealers total score exceeds 21

2 TRADITIONAL BLACKJACK

There are many variations of Blackjack played in casinos around the world. Rules for traditional blackjack played across the US are as follows:

(1) Every game has a deck of 52 cards which are shuffled at the start of every game.
(2) At the start of every game 2 cards are dealt to every player, including the dealer, one card face up and the other face down and dealer plays at the end
(3) At every player's turn his second card is revealed
(4) Each card value is the numeric value printed on the card. Face cards viz. Jacks, Queens, Kings have value 10. Ace can have two values, 1 or 11, depending on the total sum of rest of the cards in the hand.
(5) If either the player or the dealer has a blackjack they win provided the opponent doesn't have a blackjack.
(6) If both the player and the dealer have same value of hands, it is a tie.
(7) At every point a player has following choices in the game:
    (a) Hit: to draw an additional card from the deck
    (b) Stand: player wants to stay with the current score
    (c) Double: to draw one more card and double his bet, then stand
(8) Usually dealers follow a standard strategy to hit until they reach a total greater than or equal to 17
(9) At the end, value of both the dealer and player hand is compared, one with the higher value wins the game.

2.1 Why Blackjack

Blackjack, as a problem, is fairly simple and has straight forward rules. It also has relatively small number of actions at every step to reach an outcome that leads to a reward. These factors make blackjack a suitable problem to be solved using Artificial Intelligence. Blackjack in AI terminology is a Markov Decision Process because given any state, you could easily trace back previous states, as well as actions leading to those states.

3 IMPLEMENTATION

In our implementation, we first created a playable version of Blackjack.

(1) Every card consists of two values:
    (a) Suit - represented as an integer from 1 to 4 (Clubs, Spades, Hearts, Diamonds)
    (b) Rank - represented as an integer from 1 to 13 (Ace, 2 - 10, Jack, Queen, King)
(2) A deck is represented as a list of cards. A deck could be shuffled and drawn cards from.
(3) A dealer class contains all the traditional blackjack rules and keeps track of the player and dealer hand.
(4) Unlike traditional Blackjack, in our implementation the actions available to the player are hit or stand.

A value of hand function keeps track of the value of player and dealer hand. It gives a value of 10 to all cards with a rank above 10. An ace flag would keep track of the current value of ace(1 or 11) used to calculate the value of the hand.

4 ALGORITHM

4.1 Markov Decision Process

Markov Decision Process provides a framework to model problems of stochastic nature where outcomes are partly random in nature. More formally stated, MDP is a discrete time stochastic control process. At every time step, the process is in some state s, and the decision maker may choose any action a that is available in current state S_p. The process responds at the next time step by randomly moving into a new state S_n, giving the decision maker a corresponding reward R for that action a from state s which leads to the transition $S_p \rightarrow S_n$. So MDP is a better choice for this project compared to any other modelling approaches. The game of blackjack could be modelled using MDP due to its stochastic nature.

MDP for blackjack has following features:

- **State:** Every state consists of

ACM Conference, July 2017, Washington, DC, USA

...$ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

  - Value of Player Hand
  - Value of Dealer Hand
  - 2 Ace Flags
- **Action:** Actions available at each state are
  - Hit
  - Stand
- **Reward:** 1 for win, -1 otherwise

We used the following two approaches to solve the blackjack MDP represented above:

- Reinforcement Learning using Q-Learning
- Q-Learning with Feature Extraction

4.2 Reinforcement Learning

Reinforcement learning(RL) is concerned with the way in which the software, bot in this case, interacts with the environment to maximize the notion of cumulative reward. In RL the agent teaches itself without any user supervision. The core principle of RL is based on rewards for a sequence of actions taken or a sequence of state transitions. Given a state S_p, based on its knowledge of the environment, the agent will take an action A. This will eventually lead to a change in state, and eventually a reward, either positive or negative. Depending on the reward received for every action sequence or state transitions the agent will update its knowledge about the environment. Due to this feature of reinforcement learning we could use it to solve the blackjack MDP where certain action(hit or stand) at every state(hand value) leads to a win or lose.

Bot consists of a map of state to q-values, where every q-value holds the value for an action(hit or stand) for that state. Every state consists of the value of player hand, the value of dealer hand and the two ace flags. Epsilon is a user modulated value which decides whether the bot will select the actions randomly or one that has the highest q-value. When epsilon is set to 1 the bot explores its the state space by selecting randomly between actions. When epsilon is set to 0 the bot exploits its knowledge by choosing the best action at each state.

4.3 Q Learning

In the beginning, q_{value} for every action from each state is set to zero. The bot slowly explores and learns about the world by iterating through each state thousands of times, gradually updating its knowledge during a transition from state to state occurs. Q_{value} function used to update the map:

$$Q_{k+1}(S_p, a) \leftarrow \sum_{S_n} T(S_p, A, S_n)[R(S_p, A, S_n) + \gamma max_a Q_k(S_n, a_n)]$$

In each new iteration, we receive a sample (s_p, a, s_n, r) and then compute new sample estimate

$$Sample = [R(S_p, A, S_n) + \gamma max_a Q_k(S_n, a_n)]$$

After estimating the new sample, update the q_{value} map using the follwing formula

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha)[sample]$$

- $Q(s, a)$ - returns the q-value for a state-action pair
- α - user controlled learning rate
- $R(S_p, A, S_n$ - returns 1 for terminal action if bot wins the game, -1 otherwise
- γ - constant applied to q_{values} of future states

Q-learning alone works good enough to for the bot to play with an acceptable accuracy, though it requires a significant number of plays in training phase before it could achieve this. To reduce the time taken in training phase we implemented the Feature Extraction bot.

4.4 Feature Extraction

Feature extraction is like Q-learning as values are propagated backwards till the bot learns enough about the world to make right decisions. It differs in the way it uses the weighted features within the state to make inferences about its current value. This helps in reducing the number of times every state is visited during the training phase.

The features used in the early detection of similarity in the states depends on how close a particular state is to the winning state. Since the game play is independent of the permutation of the cards i.e., it does not matter which order the cards are drawn as long as the final state of cards match, we can also use the sum of the cards taken till the moment as a mapping from initial state to the closeness of the final state.

We used the following features as a means to detect the similarity of the different states starting with a different sum total of cards

- The value of hands proximity to number 21
- Average total of cards played till the moment

5 RESULTS AND ANALYSIS

Both the bots with Q learning and Feature Extraction are fairly successful in the end. Both the bots can be adjusted with a group of variables to achieve the optimum set of results through trial and error. After multiple runs successfully, we decided that 1000 episodes of play will train the bot successfully with a winning rate approximately 4200 out of the next 10000 games at which the learning rate is 0.8 and the rate of exploitation is 0.9. We would say that although the learning rate and exploiting rate might contradict each other, the rate at which agent trains in the Q learning is actually a pitfall when Feature extraction is applied the to Q learning bot. Our conclusion also depends number of independent runs of the bot with different sequences of learning across the two procedures implemented.

5.1 References

The following papers are referenced while finishing this project

- Blackjack using MDP, *Brenden Reilly*
- Blackjack with Q Learning, *Benjamin Cheevor*