An optimal solution for Black Jack

Introduction to Reinforcement learning and control theory 02465

Authors

Andreas Råskov Madsen - s183901

March 22, 2025

Git repostory:
`https://gitlab.gbar.dtu.dk/s183901/Black_Jack`

Danmarks Tekniske Universitet DTU

1 Introduction

This assignment sees Black Jack as a control problem to train a planning agent by calculating the exact action value, and compare it with agents trained via Monte Carlo simulation. Thereby I was able to learn the optimal policy for Black Jack (With tho usual disclaimer that my code and ma thematic is correct). I was also able to significant reduce computing time, by combining some of the ideas from RL into the planing problem.

1.1 Acknowledgement

I have written my own code (Also for the MC agent) because i want to build a Git hub that show my coding skill's, and i had a lot of vacation to practise code. However when that is said i draw a lot of inspiration from the course code base.

Other inspiration has been found on the python version of Sutton's [2] code, that can be found on: `https://github.com/ShangtongZhang/reinforcement-learning-an-introduction/blob/master/chapter05/blackjack.py#L56`

I use the Black Jack environment in Open AI gym to train my MC agent, and evaluate all agent. This environment can be found on: `https://github.com/openai/gym/blob/master/gym/envs/toy_text/blackjack.py`

1.2 Motivation

Well the stock trading algorithm didn't do that well, so now i need to find new creative ways to earn money (or get a job).

2 Theory

2.1 Finite horizon

A simple prof for the finite horizon problem is that only cards with a value equal or greater than 1 can be drawn. Thus if we keep drawing cards, we will sooner or later hit a sum greater than 21.

2.2 Drawing from a fix probability

We are drawing cards from a distribution that don't change depending on cards drawn before.

And example is a scenario where the agent draws 22 aces is possible, and thus should be evaluated when doing a full tree search. Other practical consequences of this method is in the discussion.

3 Method

3.1 Planning Agent

The planning agent is doing a full tree search in order to find the value of every state and every action. Since Black Jack is a final horizon MDP with relatively few states. The planning agent uses an extended version of the state used internal composed as

$$Si = (AgentSum, Agentace, DealerSum, DealerAce)$$

. The agent also uses the Open AI state space defined as

$$S = (Agentsum, dealerfaceupcard, AgentAce)$$

. Without terminal states there are 588 internal states and 280 non in the Open AI state space.

a conversion from S to Si can easily be made, a conversion the other way around is not needed.

For each state the optimal action is given as:

$$a = \arg\max(Q_{stand}(s), Q_{hit}(s))$$

Evaluating Q_{stand} and Q_{hit} is for this agent two different problems.

3.1.1 Q_{stand}

When the agent stand the state is terminal and the agent can no longer influence the game. Thus we just need to calculate the value of a Markov chain that will follow the dealers actions.

The dealer will turn around his hidden card or draw a new card (When every card is drawn from the same distribution those two cases is identical). The state space then split to all possible sates we can be in:

$$Si_i = sum(Si, i)$$

Where i is the value of cards that are possible to be draw in the set (ace to 10) and Si_i is the dealers sum and ace after drawing/turning the card i, The function sum add the value of i to the dealers sum but can handle counting ace as either 1 or 11, see the function in planning agent class.

For each Si_i a value of the state $V(Si_i)$ is found if dealer sum is greater than 17 by comparing the dealer and agent sum $V(Si_i) = comp(Si_i)$ (see function comp for details), or return -1 if the dealer is bust, or the value can be found by calling the function $V(Si_i) = Q_{stand}(Si_i)$ recursively until the dealers sum is greater than 17. This can be done because we have a final horizon problem (else the algorithm would run forever).

Finally for the function Q_{stand} a value returned by the equation:

$$Q_{stand}(Si) = \sum_i V(Si_i) \cdot p(i)$$

3.1.2 Q_{hit}

The hit action like the stand also transform the state from one state to 10 possible states:

$$Si_i = sum(Si, i)$$

But this time Si_i is refers to the agents part of the state, the agent's sum and ace.

$V(Si_i)$ is -1 if agent sum is over 21, else the state value can be found according to the MDP properties by:

$$V(Si_i) = \max(Q_{hit}(Si_i), Q_{stand}(Si_i))$$

Note that we are calling Q_hit recursively, thus the final horizon argument should be made to argue the algorithm would not run infinitely.

The action value returned by Q_{hit} is thus:

$$Q_{stand}(Si) = \sum_i V(Si_i) \cdot p(i)$$

3.1.3 Using a bit of learning

The reader may have noted that recursive function is use many times and also that for each step the tree branches out with a factor of 10. This means really many recursions even if it stops eventually. In an early version of my code i tried to evaluate the state $S(12, 1, True)$ and it ran for about an hour (This is also the hardest state of the game to evaluate)

However this run time can be significantly reduced by introducing a Q matrix from reinforcement learning. By saving the values of $Q_{hit}(Si)$ and $Q_{stand}(Si)$ the first time they are calculated, science we are often getting back to the same states 588 states we can just get the Q value from memory instead of evaluating it again.

This is best given by an example, imagine an agent going form a sum of 12 and no ace to a sum of 18, this can be done by by going many ways, a it could draw a 6 or (2,4),(2,2,2),(1,5) and so on. For each combination the state of 18 would be evaluated, but since drawing a card does not change the cards distribution the state, the value of 18 would be the same no matte how the agent got there, thus the agent only need to evaluate it once.

3.2 MC Agent

The MC agent does not have any significant difference from the every visit MC agent, used in the course, even though i tried to write the code to look more like the pseudo code in the book: [2]. The original problem was solved with a first visit agent, given the nature of Black Jack (and other finite horizon problems) returning to a state previous state is not possible, thus there is no difference between first visit and every visit.

I use epsilon =0.2, i did some pilot study's with different values without much different result.

4 Environment/testing

4.1 Planing environment

The planing agent simulate the environment internally. Thus it doesn't need to play the game but just iterate over all the 280 possible states, in a Black Jack game.

4.2 RL environment

The MC agent was trained in the same way as done in the course, and by using the open AI environment

To get an idea of how much train the effect of Monte Carlo sampling 3 experiment was run. One with 10^4, 10^5 and 10^6 episodes.

4.3 Testing environment

The performance of the found policy was evaluated, by running 10^5 episode in the open AI environment, and returning the mean reward.

5 results

Agent	planning agent	MC 10^4	MC 10^5	MC 10^6
Mean reward	-0.04667	-0.08691	-0.05784	-0.04618
Training time	<1s	1s	23s	20m 6s

Table 1: Results of testing, run time is evaluated on a Intel i7 processor

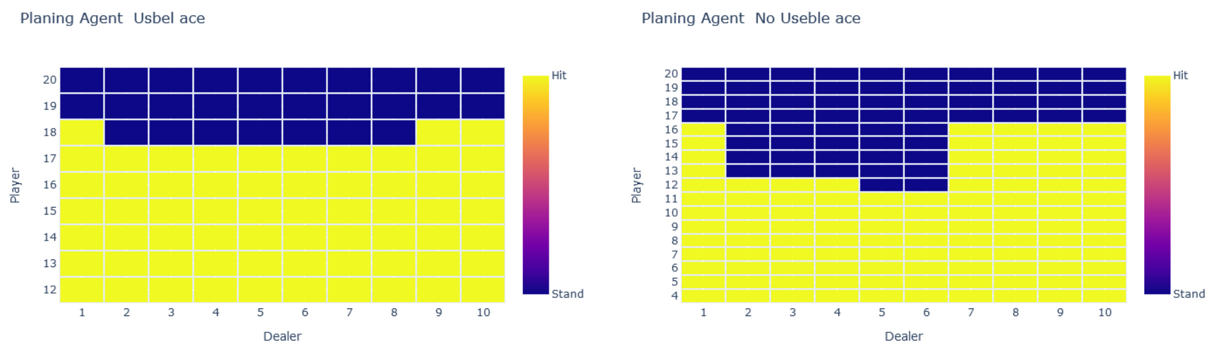


Figure 1: Policy of planing agent

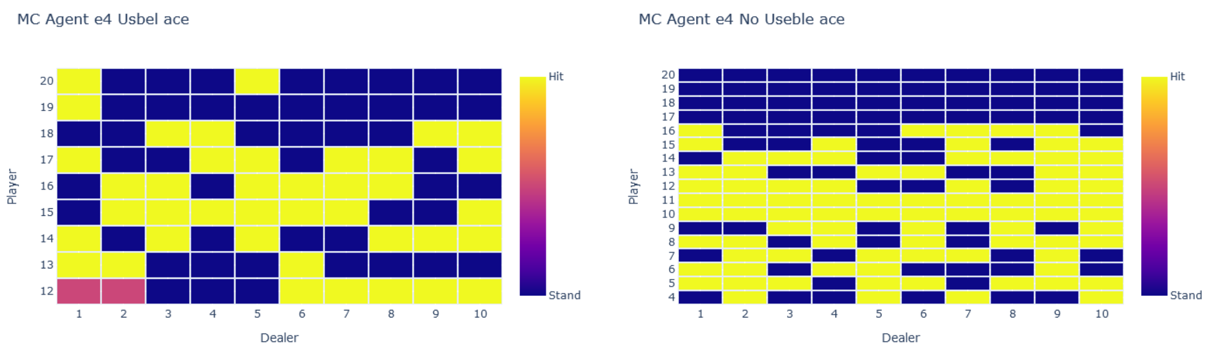


Figure 2: Policy after 10^4 episodes purple is states not visited.

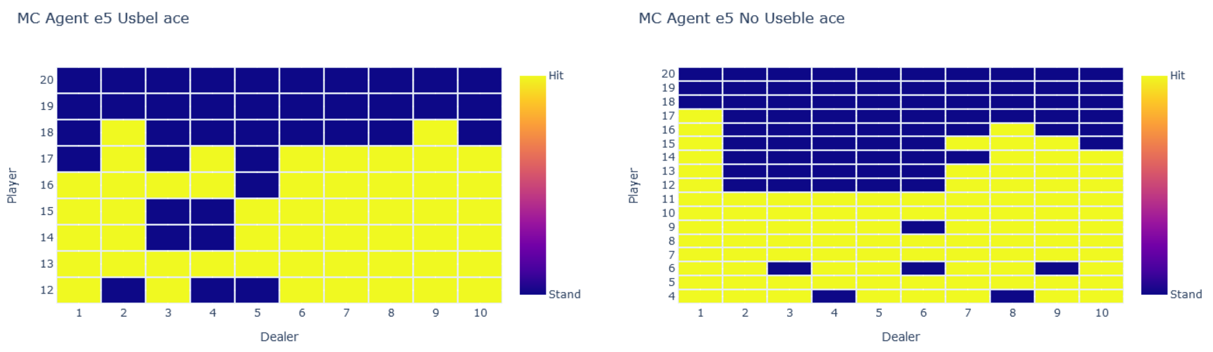


Figure 3: Policy after 10^5 episodes

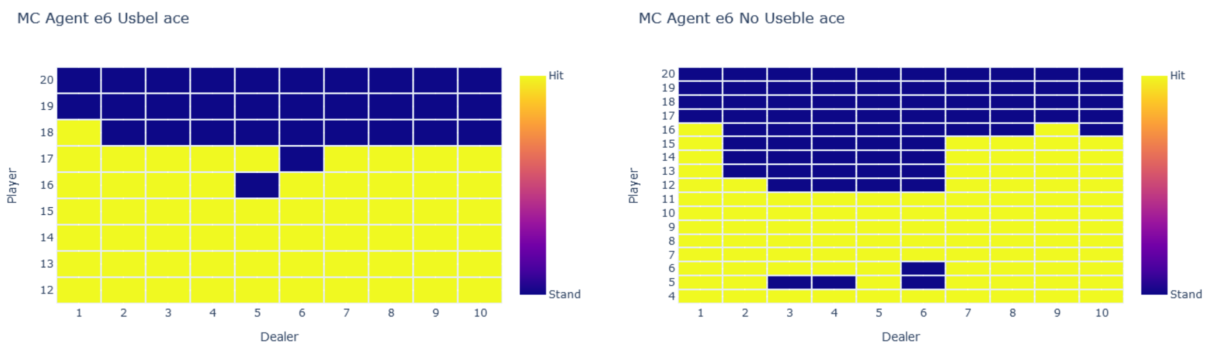


Figure 4: Policy after 10^6 episodes

Note: It should be possible to open and interactive version of all plots if download from the repository.

6 Conclusion

Even though the MC agent with 10^6 episodes performed slightly better i is not significant to conclude it's better.

An other argument for optimally is that i get the same result as [2] except in the state (12,4,no Ace) however my planing agent claim that both standing and hitting have the action value -0.22 making it a question of settling ties.

Therefor i would conclude that my planing agent can find the optimal policy. It can also be concluded that Monte Carlo simulation also will find the close to optimal policy given enough Episodes.

7 Discussion

7.1 Planing VS Learning

Even though planning seems Superior in a Black Jack problem it has some serious drawbacks if the state space becomes to big, and more complex solution would be need for an infinite horizon problem.

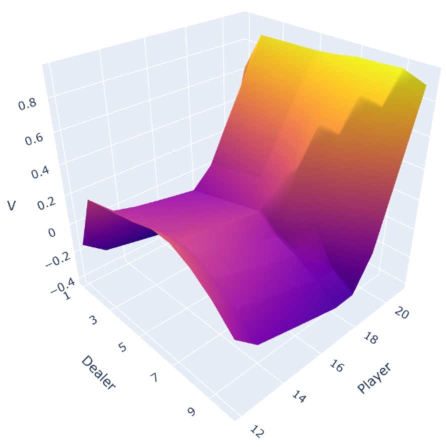
Also the code for a planing agent is more complicated to write, and unlike the MC agent this code can only solve this specific problem.

However from a pedagogical standpoint it help me a lot to solve a problem both at an control problem and a RL problem.

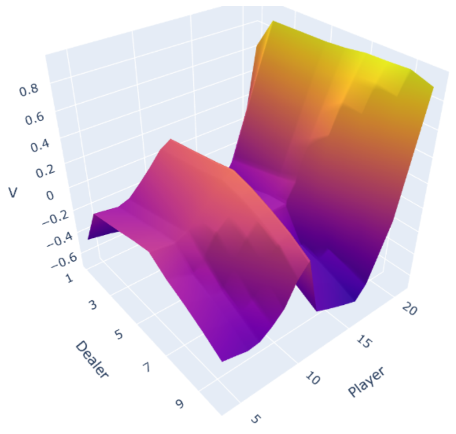
8 Drawing from a constant distribution

In a real game of Black Jack the card deck is slowly depleted as cards is drawn changing the probability of drawing the next card. In this assignment and in all both Sutton and Open AI gym code the distribution does not change. A problem with depleting a deck is that the problem become more complex, as each combination of card left can be considered a new state thereby expanding the state space. By using MC it should be possible to find an average policy on over all likely deck compositions without change the state space, that policy should correspond with the one found by Thorp [1] i think that's why Sutton an Torp, disagree, they are solving two different problems. However here we should note that this would not be and optimal policy since different deck compositions may require different policy, further studies should be done by students who wish to rip of casinos.

9 Appendix

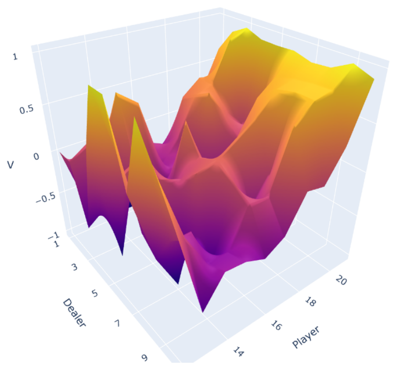


(a) Usable Ace

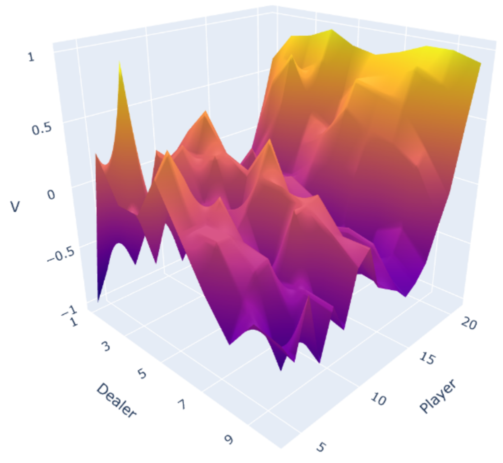


(b) No usable Ace

Figure 5: Calculated state value of planing agent

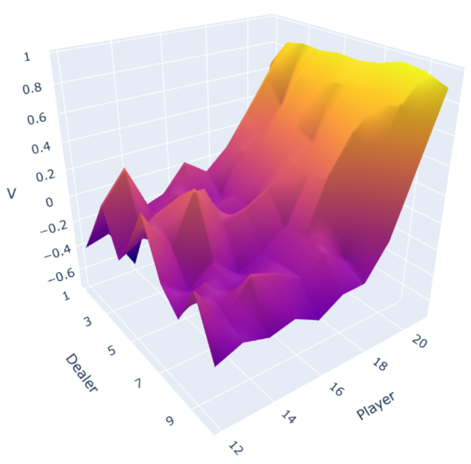


(a) Usable Ace

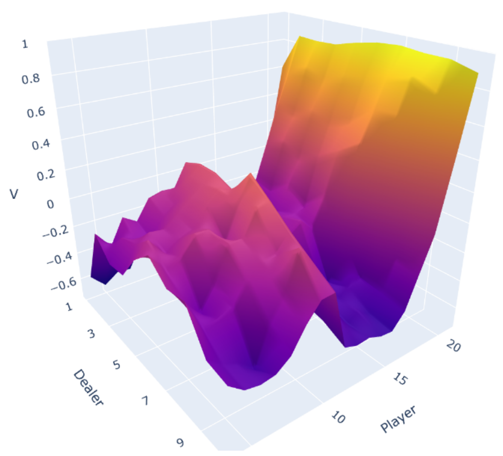


(b) No usable Ace

Figure 6: Value after 10^4 episodes

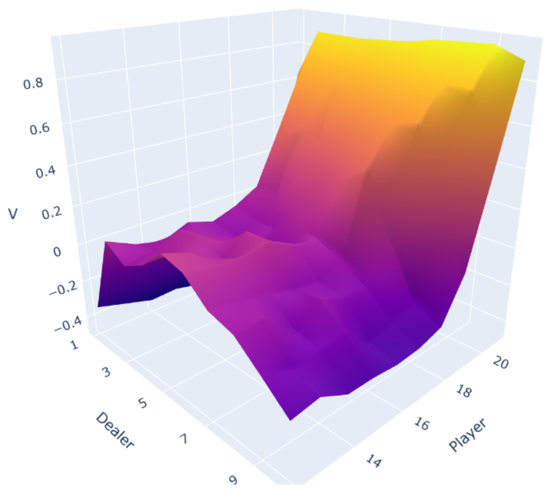


(a) Usable Ace

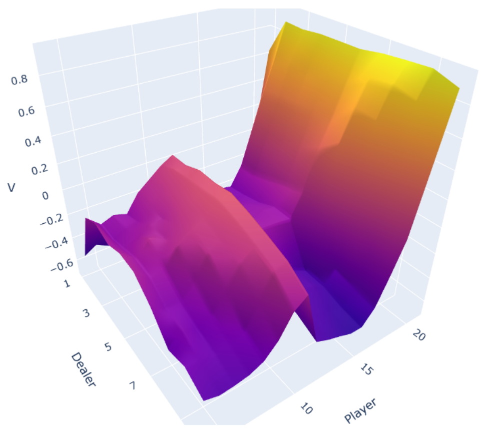


(b) No usable Ace

Figure 7: Value after 10^5 episodes



(a) Usable Ace



(b) No usable Ace

Figure 8: Value after 10^6 episodes

References

[1] Thorp Edward O. *Beat the dealer : a winning strategy for the game of Twenty-One*. Vintage Books, New York, vintage book editions edition, 1966.

[2] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction second edition*. The MIT Press.