CS7641: Machine Learning
Obtaining Blackjack Policies via Reinforcement Learning

Kathryn Schuetze

I. Introduction

Blackjack is a popular probability-based card game, where the player and dealer take turns drawing from a deck of cards. Each card drawn is added to the sum of the hand, with the goal of reaching a higher sum than the other participant while staying under a sum of 21. If the sum of a player's hand exceeds 21, they are said to have a blackjack, and lose the game. To avoid a blackjack, a player can choose to pass their turn instead of drawing a card; In the context of the game, these actions are referred to as a "stand" or a "hit" respectively[1]. Reformatting this game as Markov Decision Process (MDP) is a fairly simple process; Each of the four components of a MDP can be easily mapped to a component of blackjack (Table I). This concept was the basis of assignment 4 of CS 7641: Machine Learning, which explored the application of value iteration and policy iteration algorithms to obtain the optimal policy for a game of blackjack. These two algorithms represent different approaches to solving the Bellman equation, and are considered basic reinforcement learning algorithms. By creating 'optimal' versions of each algorithm, we seek to identify how their individual performances are affected by hyperparameter variation and their combined performances compare and contrast.

MDP component	Blackjack component)
Agent	The player or RL agent
Environment	The deck of cards both player and dealer draw from
Actions	"Stand" or "Hit"
Rewards	+1 for a win, 0 for a draw, and -1 for a loss

TABLE I
Components of Blackjack As A Markov Decision Process (MDP)

II. Hypotheses

A. Value iteration

Value iteration attempts to calculate the optimal value function directly, by substituting values in the Bellman equation until an optimal value is found. Once the optimal value is found, the values are extracted from the Bellman equation and an optimal policy is derived[2]. By trying many values in quick succession, value iteration often converges faster in raw wall clock time than policy iteration.

B. Policy iteration

Policy iteration consists of two steps. Step one, policy evaluation calculates the value of the current policy. Then step two, policy improvement, selects a new value to maximize the value function's gains. These steps are repeated until the algorithm converges upon a maximally optimum policy[2]. Despite its two-step nature, policy iteration should converge faster than value iteration when measured in iterations, due to the efficiency of each individual policy iteration vs that of individual value iterations.

C. Optimal policies

Despite their differing approaches, both value and policy iteration should eventually converge upon a single globally optimal policy[2]. Therefore, the policies output by both algorithms should be identical, as they should represent the global optima of the blackjack MDP.

D. Gamma

Gamma allows for specification as to whether the algorithm should prefer short term rewards or long ones. This is important for the agent, because one often comes at the cost of the other; This idea is often exemplified by a game of chess, in which a pawn is used to take another piece but is put in a position to be taken itself (maximization of short-term rewards) vs another game where the same pawn is sacrificed in order for another piece to take a more valuable piece from their opponent (maximization of long-term rewards).
In this way, greater rewards may be obtained by playing the 'long game', but only if there is a 'long game' to play. Blackjack is a short game that can be lost in as few as two turns; Because of this, sacrificing pawns (or equivalent) may result in loss more often than a greater long-term reward.
In addition, blackjack is a game of probability; Even when taking a risk, there is no guarantee it will pay off.

E. Theta

The theta hyperparameter specifies how small improvements between iterations must be in order for a policy to be accepted as 'final'. The smaller the changes between iterations are, the more likely the policy is to be globally optimal. Therefore, a smaller theta should be guaranteed to result in better solutions; However, doing so should also result in longer run times. The simplicity of blackjack makes it a good environment with which to experiment on the magnitude of changes in run time from variations in theta, without consuming unnecessary computational resources.

III. Methodology

A. Libraries

Bettermdptools, created by John Mansfield, was created to facilitate implementation of value iteration and policy

iteration for CS7641 assignments[3]. In addition to the base value iteration and policy iteration methods, it also contains a wrapper for the gym blackjack environment, which provided a stable base to build off of for obtaining optimal blackjack policies.

In addition to bettermdptools and gym, the following libraries were used for implementation of the library and visualization of results. If a specific version was used, it is documented; Otherwise, assume the most recent release.

- gymnasium version 0.27.1
- jax and jaxlib versions 0.4.13
- matplotlib version 3.8
- numpy
- pandas
- seaborn
- time
- wget

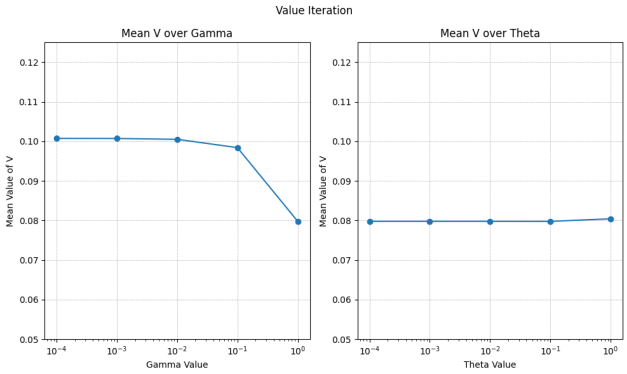
B. Modifications to bettermdptools

The code for this project was created and executed via a Jupyter notebook in Google's Colab application rather than a local directory, which created its own unique challenges. Colab struggled with appropriately accessing the pickles blackjack reward and transition matrix "blackjack-envP.pickle"; In order to access it properly, wget was used to download the matrix to the directory each run, and bettermdptools' "blackjack_wrapper.py" was overwritten to call the correct copy rather than the GitHub version of the matrix. Otherwise, the blackjack_wrapper.py in the Colab file is identical to that of the original.

C. Scoring

Both value iteration and policy iteration measure the value of individual iterations using a V function, which outputs V values used to represent the value of the state. For blackjack, the value of each state is bounded by the range [-1, 1].

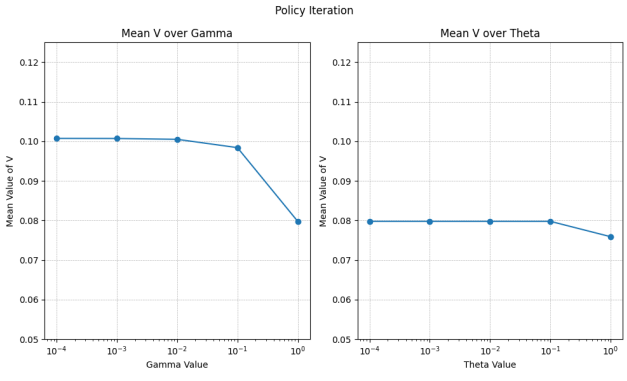
IV. Value Iteration



Figures 1.0 and 1.1

Tracking of mean V scores over various gamma values (Figure 1.0) shows a significant preference for lower gamma values vs higher values, proven by the steep drop off at 10^0. As alluded to by the hypothesis section, this is likely a function of blackjack itself, rather than an innate quality of the value iteration algorithm; Since blackjack is a game that can be won or lost in as little as two turns in a worst case scenario (drawing two aces, each worth 11 points for the sum of the hand), gambling for long-term rewards is more likely to result in a loss than anything else. Therefore, a lower gamma score makes sense because it indicates the leaner should prioritize shorter-term rewards, which are more likely to result in a win. By contrast, mean V scores over theta values (Figure 1.1) displays a mild preference for the opposite phenomena. Mean scores remain consistent across 10^-4, 10^-3, 10^-2, and 10^-1, but 10^0 represents a slight uptick in mean V scores. This shows that higher theta values resulted in a higher mean score, which is interesting because it goes against the established knowledge that allowing large changes between iterations for acceptance of a policy via value iteration may result in subpar scores since the algorithm is still making large changes. Regardless, this larger theta score was used for the 'optimal' value iteration hyperparameter set of gamma=10^-4 and theta=10^0.

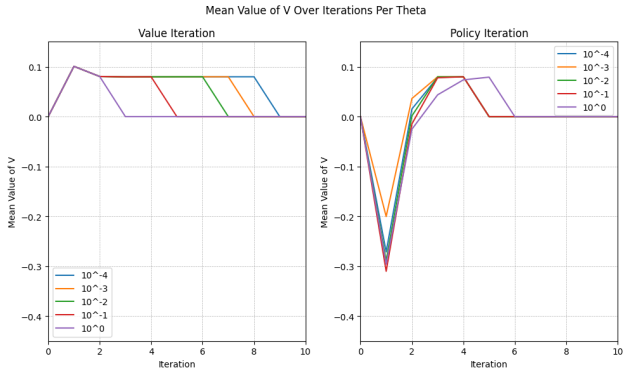
V. Policy Iteration



Figures 2.0 and 2.1

Mean V scores over gamma values (Figure 2.0) follow the same patterns and logic as that of Figure 1.0, establishing that this analysis is innate to the problem itself rather than a property of value iteration or policy iteration alone. However, mean V scores over theta values (Figure 2.1) shows opposite results as that of Figure 1.1; Similarly, mean scores remain consistent over 10^-1, 10^-3, 10^-2, and 10^-1, but show a downtick at 10^0. This implies policy iteration prefers a lower theta score, in contrast to value iteration's preference for higher theta scores. However, viewing only final V scores does not accurately show the full picture.

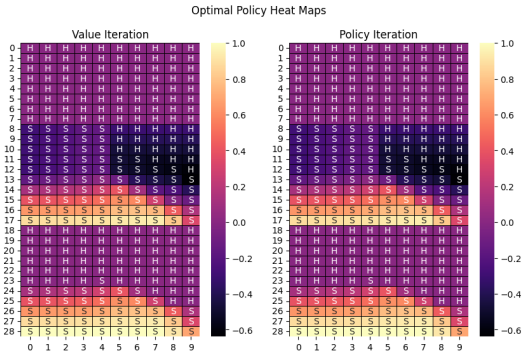
VI. Further Theta Exploration



Figures 3.0 and 3.1

Due to the low variation in final mean V scores for theta values, mean V was graphed over iterations for the same range of theta values in Figures 3.0 and 3.1. These graphs yielded much more interesting results, and allowed for a more complex analysis of theta's behavior in the context of iterations to convergence. Value iteration (Figure 3.0) shows that run time in iteration grows linearly as theta is increased, but retain the same general pattern. This indicates that theta has a lesser effect on V scores, and mostly affects run time in iterations for value iteration. This tracks with the internal logic of value iteration, which tries values until changes are sufficiently small as according to theta; By default, changing theta should result in a different number of iterations to acceptance and convergence. With the largest theta value, the optimal policy can be achieved in as little as two iterations. Mean V over theta for policy iteration (Figure 3.1) shows different results; Rather than a large theta representing the smallest number of iterations to convergence and growing as theta shrinks, iterations to convergence appears to be the same for 10^-1, 10^-3, 10^-2, and 10^-1. 10^0 was proven to result in outlier scores for policy iteration only during the course of the original project[4], so by discarding it, we can assert that policy iteration will always take 2 iterations to converge. This is indicative of the policy evaluation and improvement steps of policy iteration, in contrast to value iteration's approach. While value iteration strives to reduce changes to a size equal to or less than theta, policy iteration is repeated until the policy becomes stable. Since this process does not rely on theta, it makes sense theta values would have a lesser effect on the runtime in iterations for policy iteration.

VII. Value Iteration vs Policy Iteration



Figures 4.0 and 4.1

Figures 4.0 and 4.1 show heat maps of the policies produced by the 'optimal' combinations of hyperparameters determined by the analyses above. From both the code output and the graphical representations, we can conclude that both value iteration and policy iteration converged to the same optimal policy. This is an innate property of the algorithms, which can be proven via mathematical proof[1].

Both optimal policies achieve a final V score of 0.100745, which is approximately 55% of the total score range of [-1,1]. This implies that use of value and policy iteration on the blackjack MDP resulted in only slightly higher scores than chance, which is 50% due to the binary action set of "hit" vs "stand". Ultimately, these low scores are a result of the game's probabilistic nature; Since there are 11 possible card values for each hit, each with a 7.69% chance of being pulled per turn. Since each run is, on average, only 2.71828 turns[5], the agent does not have time to learn and apply more advanced learning strategies to get better at the game.

Algorithm	Run Time (Milliseconds)	Iterations to Convergence
Value Iteration	24.978	At best 2
Policy Iteration	38.014	Always 2

TABLE II

Time Complexities of Value Iteration vs Policy Iteration

As hypothesized, value iteration's brute force approach allowed it to run faster than policy iteration in sheer wall clock time. In a best case scenario, value iteration managed to also tie with policy iteration for time complexity in iterations; However, as noted in Figure 3.0, this result can only be achieved by a theta value of 1.0 for value iteration. Doing so means accepting large swings between iterations; In a more complex game, this would likely mean accepting much worse solutions than those converged upon by policy iteration in the same number of total iterations.

VIII. Conclusion

Ultimately, the behavior of both value iteration and policy iteration tracked strongly with previous knowledge of the algorithms' tendencies. The strongest deviations from this previous knowledge occurred most strongly in the context of gamma and time complexity in iterations between value

iteration and policy iteration; Increasing gamma detracted from total; rewards rather than increasing them due to the structure of the blackjack game itself, while value iteration benefitted from the simplicity of the game and was able to tie with policy iteration in a best case scenario. In the context of another MDP, these results would likely not hold true.

REFERENCES

[1] [1] R. Sutton and A. Barton, “Reinforcement Learning: An Introduction,” Stanford, https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf (accessed Apr. 15, 2025).

[2] T. Mitchell, “Machine Learning,” McGraw-Hill, 1997, [Online]. Available: https://www.cs.cmu.edu/ tom/files/MachineLearningTomMitchell.pdf. [Accessed: April. 6, 2025].

[3] “bettermdptools/bettermdptools/algorithms/planner.py at master · jlm429/bettermdptools,” GitHub. https://github.com/jlm429/bettermdptools/blob/master/bettermdptools/algorithms/planner.py. [Accessed April 7th, 2025].

[4] Christian Charles Robert Young, "Clarification on policy" EdStem (CS 7641/4641: Reinforcement Learning discussion), Sep. 3, 2023. [Online]. Available: https://edstem.org/us/courses/71185/discussion/6454318. Accessed: Apr. 13, 2025.

[5] eyeballfrog, "How to calculate the average number of turns until going bust for a simplified gambler’s ruin problem," Math Stack Exchange, Burreto, 2022. [Online]. Available: https://math.stackexchange.com/questions/4457096/how-to-calculate-the-average-number-of-turns-until-going-bust-for-a-simplified-b. Accessed: Apr. 12, 2025.

[6] Google AI, “Gemini,” Google AI Blog, [Online]. Available: https://ai.google.dev/gemini. [Accessed: April 1, 2025].