Behavior of Reinforcement Learning Algorithms on Unique Markov Decision Processes

Joey Bishop

I. OVERVIEW OF MDPs

The two explored Markov Decision Processes (MDPs) are Blackjack and CartPole.

A. Blackjack

Blackjack is a discrete, stochastic MDP where the agent must obtain cards that sum to closer to 21 than the dealer without going over 21. The state space is a 3-tuple containing the player's current sum, the value of the dealer's one visible card, and whether the player holds a usable ace. The action space consists of $\{0, 1\}$, where the agent can either stick (0) or hit (1). The reward signals consist of $+1$ (win game), -1 (lose game), 0 (draw game), or $+1.5$ if the agent wins the game with a natural blackjack.

B. CartPole

CartPole is a continuous, deterministic MDP where the agent must balance a pole attached to a cart by moving the cart left or right. The state space is an array of length 4. Index 0 represents the cart position, which is of range $[-4.8, 4.8]$, index 1 represents the cart velocity, which is of range $[-\infty, \infty]$, index 2 holds the pole angle, which is of range $[-24, 24]$, and index 3 provides the pole angular velocity, which is also of range $[-\infty, \infty]$. The action space consists of $\{0, 1\}$, where the agent can either push the cart left (0) or right (1). The reward signal is simply $+1$ for every step taken, including the terminating step, for maximum possible reward of $+500$. The episode terminates if the pole angle is greater than ± 12, the cart position is greater than ± 2.4, or the episode length is greater than 500.

II. OVERVIEW OF ALGORITHMS

The two MDPs are explored through Value Iteration (VI), Policy Iteration (PI), and SARSA. A brief explanation of each reinforcement learning algorithm is provided below.

A. Value Iteration

Value Iteration is a model-based reinforcement learning algorithm that iterates on reward values by applying a policy in state once, and evaluating that policy through the Bellman optimality equation. Because Value Iteration is acting on values, learning becomes more direct; however, it becomes more difficult to transform this into a policy. Value Iteration converges when the value function "by only a small amount" after the policy is evaluated [1].

B. Policy Iteration

Policy Iteration is another model-based reinforcement learning algorithm that iterates to an optimal policy by applying and re-evaluating a policy [2]. For each iteration, a value function is evaluated on the current policy, which is then applied to improve the policy. "Each policy is guaranteed to be a strict improvement" on previous policies, unless an optimal policy was already found [1].

C. SARSA

SARSA is a model-free, on-policy, temporal-difference reinforcement learning algorithm. Rather than transitioning from state to state and learning value, SARSA transitions from state-action pair to state-action pair, learning the value from those pairs. The algorithm updates estimates with partial information, without waiting for full episodes, and builds a policy by trying actions. For each episode, the algorithm Q is continually updated for the policy, while the policy is pushed toward a greedier approach with respect to Q. Greediness is controlled by ϵ, and is often decays as episodes continue [1].

III. METHODS

Firstly, several executions of Value Iteration on CartPole were performed to identify the optimal bin sizes for the environment's cart position, cart velocity, and pole angular velocity. Bin sizes for cart position and cart velocity were $p \in \{4, 6, 8\}$ and $v \in \{4, 6, 8\}$, respectively, and pole angular velocity was $a \in \{6, 8, 12\}$. γ was set to 1.0 for all permutations. The permutation which maximized score for Value Iteration informed the final bin sizes for CartPole.

Then, each MDP environment was applied to Value Iteration, Policy Iteration, and SARSA. Each reinforcement algorithm was trained on each environment with $\gamma \in \{0.8, 0.9, 0.99, 1.0\}$, for a total of 4 trainings per iteration per environment. The γ that scored the best for each algorithm was recorded. On Value Iteration and Policy Iteration, a constant θ of 0.1 was used. On the Blackjack MDP, SARSA was executed on 300,000 episodes. On the CartPole MDP, SARSA was executed on 5,000 episodes. On SARSA, an initial $\epsilon = 0.3$ was used with a 99.995% decay ratio to a minimum $\epsilon = 0.1$.

IV. RESULTS

Firstly, the final bin counts for CartPole are reported. Then, for each algorithm for each MDP, results per γ are reported. Policy maps for Blackjack are reported for each algorithm. Policy maps are omitted for CartPole due to the continuous and multidimensional nature of the environment.

CartPole Bin Count Performance on Value Iteration with $\gamma = 1.0$					
p bins	v bins	a bins	V(s)	i	Time (ms)
4	4	6	2.83832	9	69
		8	2.95279	9	95
		12	3.10734	10	153
	6	6	2.99469	10	119
		8	3.22645	10	156
		12	3.37168	12	273
	8	6	3.32518	11	168
		8	3.70754	12	246
		12	3.87580	13	398
6	4	6	3.59903	11	126
		8	3.91780	11	170
		12	4.08062	12	275
	6	6	3.77053	12	205
		8	4.21981	12	281
		12	4.37621	14	475
	8	6	4.04363	12	272
		8	4.67142	13	406
		12	4.85809	15	678
8	4	6	4.29960	12	183
		8	4.85139	13	267
		12	5.01291	14	422
	6	6	4.47373	13	301
		8	5.16757	14	421
		12	5.32412	16	725
	8	6	4.73245	14	424
		8	5.60445	15	609
		12	5.79495	17	1,017

TABLE I

MEAN VALUE SCORES ON VALUE ITERATION FOR EVERY PERMUTATION OF CARTPOLE BIN COUNTS. AS THE NUMBER OF BINS BECOME MORE GRANULAR, SCORE, ITERATIONS, AND TIME INCREASE.

A. *CartPole Bin Counts*

As bin count sizes increased for each bin type, the reported mean V(s) for Value Iteration also increased. However, this also incurred more iterations to convergence and longer execution times. The smallest permutation with $p = 4, v = 4, a = 6$ had a mean V(s) of ~ 2.83832 after 9 iterations in 69ms. The largest permutation with $p = 8, v = 8, a = 12$ had a mean V(s) of ~ 5.79495 after 17 iterations in 1,017ms. All permutation results can be found in Table I. With these results, a final permutation of $p = 4, v = 4, a = 15$ is selected.

B. *Value Iteration*

For the Blackjack MDP, all γ behaved similarly on Value Iteration. The mean score V(s) was maximum at 2 iterations, stabilized by 5 iterations, then fell to 0 at iteration 6. Between 2 and 5 iterations, $\gamma = 0.8$ had the highest mean V(s), and $\gamma = 1.0$ had the lowest mean V(s). The maximum score V(s) remained at 1.0 from iteration 2 through iteration 5, then fell to 0 at iteration 6 for all γ, as seen in Fig. 1. The highest mean V(s) was 0.10075 for $\gamma = 0.8$ at iteration 2, which is slightly higher than an average of drawing each game ($V = 0$). Value Iteration executed on all γ in 59ms.

For the CartPole MDP, each γ appeared to achieve a similar shape for mean and maximum V(s), but the magnitude of the shape increased as γ increased. For all γ, the mean score V(s) increased logarithmically before reaching a plateau, then dropped to 0 at iteration 13, as seen in Fig. 2. $\gamma = 0.8$ began to plateau at iteration 8 with a mean value $V(s) \approx 2.25$, which means that on average slightly more than 2 steps were taken per iteration. $\gamma = 1.0$ plateaued at iteration 10 with the highest mean value $V(s) = 3.13424$, which means that on average slightly more than 3 steps were taken per iteration. For maximum score V(s), all γ increased before plateauing at iteration 11, then falling to 0 at iteration 13. $\gamma = 0.8$ had the lowest maximum V(s) at ≈ 4.5, and $\gamma = 1.0$ had the highest maximum V(s) at 10.0, which is equivalent to 10 cart movements taken, including the movement that terminated the game. Value Iteration executed on all γ in 679ms.

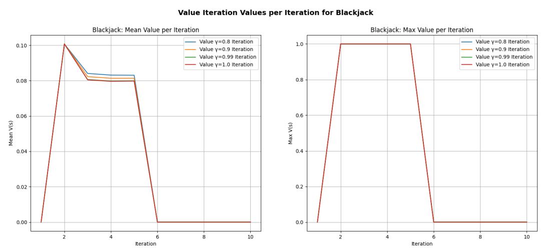


Fig. 1. The mean and max V(s) for Value Iteration on the Blackjack MDP

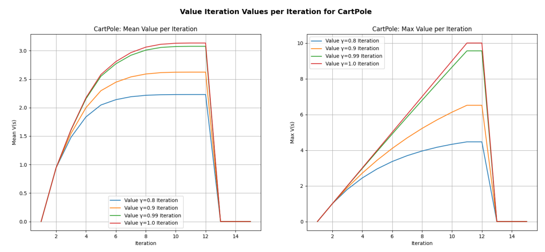


Fig. 2. The mean and max V(s) for Value Iteration on the CartPole MDP

C. *Policy Iteration*

For the Blackjack MDP, each γ followed a similar trajectory for mean and maximum V(s), with some variance in mean magnitude and persistence. At iteration 2, the mean value V(s) was negative for all γ, with $\gamma = 0.8$ reporting ≈ -0.225, and $\gamma = 0.9$ reporting ≈ -0.31. By iteration 3, all γ trended positive, with $\gamma = 0.8$ reporting the highest mean V(s) at ≈ 0.049. All γ converged at iteration 4 at 0.08150, which is marginally better than drawing on average each game. Most γ fell to 0 at iteration 5, but $\gamma = 0.99$ remained stable for one more iteration before falling to 0 at iteration 6, as seen in Fig. 3. For maximum V(s), all γ held the maximum of 1.0 from iterations 2 to iterations 4, and $\gamma = 0.99$ remained stable for one more iteration before falling to 0 at iteration 6. Policy Iteration executed on all γ in 143ms.

For the CartPole MDP, most γ followed a similar trajectory for mean and maximum V(s), with some variance in magnitude before dropping to 0 at iteration 7. However, $\gamma = 0.99$ degenerated faster than other γ, and dropped to 0 at iteration 6, as seen in Fig. 4. $\gamma = 0.8$ had the lowest mean V(s) at ≈ 2.25 at iteration 6, while $\gamma = 1.0$ had the highest mean V(s) at 3.13424 at iteration 6, which is equivalent to

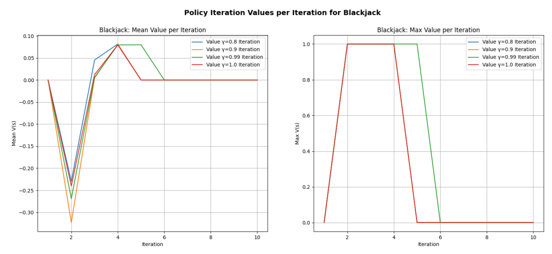


Fig. 3. The mean and max V(s) for Policy Iteration on the Blackjack MDP

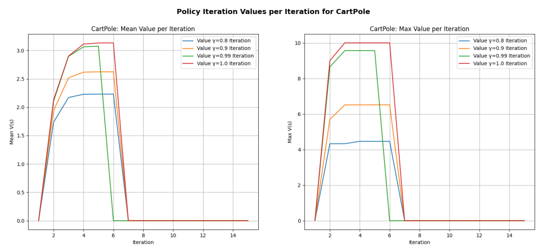


Fig. 4. The mean and max V(s) for Policy Iteration on the CartPole MDP

slightly more than 3 steps taken on average per iteration. Similarly, $\gamma = 0.8$ at the lowest maximum V(s) at ≈ 4.5 at iteration 6, while $\gamma = 1.0$ at the highest maximum V(s) at 10.0 at iteration 6, equivalent to 10 cart moves taken, including the movement that terminated the game. Policy Iteration executed on all γ in 1,197ms.

D. SARSA

For the Blackjack MDP, the best $\gamma = 0.99$ yielded a cumulative reward per episode which trended downward as episodes continued before reaching a final total reward of $-23,525$ at 300,000 episodes. However, the average reward per episode increased before converging to -0.0566 at around episode 200,000, as seen in Fig. 5. This average reward is equivalent to performing marginally less than drawing on average per game. SARSA executed with $\gamma = 0.99$ in 33.2 seconds.

For the CartPole MDP, the best $\gamma = 1.0$ yielded a cumulative reward per episode which trended upward as episodes increased before reaching a final total reward of $886,020$ at 5,000 episodes. Average reward climbed steadily until convergence at around episode 2,000 with value 212.3, as seen in Fig. 6. This average reward is equivalent to applying just over 212 movements to the cart, on average,

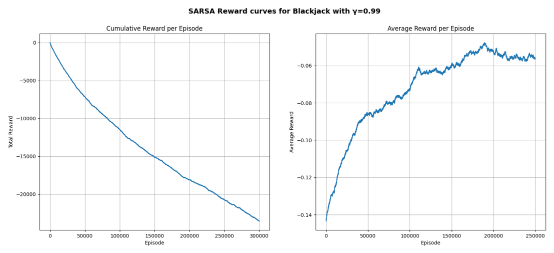


Fig. 5. The total and average reward for SARSA on the Blackjack MDP

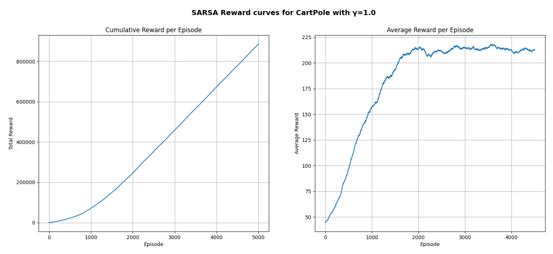


Fig. 6. The total and average reward for SARSA on the CartPole MDP

and the game terminating. SARSA executed with $\gamma = 1.0$ in 65.5 seconds.

E. Blackjack Policy Maps

For each algorithm, a policy map was generated on the Blackjack MDP using the γ that maximized value. These can be seen in Fig. 7. Each cell indicates a policy to take, where H means to "hit", and S means to "stick", and the color of the cell indicates the confidence in the particular policy. Each column represents the face-up dealer card in the current state, where columns 0 through 7 represents cards 2 through 9, column 8 represents 10 or any face card, and column 10 represents an ace. Each row represents the card sum and action of the player's state. Rows 0 through 17 represent sums from 4 to 21 and the player can choose to hit or stick, denoted H4 through H21. Rows 18 through 27 represent sums from 12 to 21 and the player has already stuck, denoted S12 through S21. Row 28 represents a blackjack, denoted simply as BJ.

For Value Iteration and Policy Iteration, 5 distinct policy partitions form in the policy map. Rows 0 through 7, which represent H4 through H11, create a partition which call to hit. Similarly, rows 18 through 23, which represent S12 through S17, create a partition which call to hit. Rows 24 through 28, which represent S18 through S21 and BJ, create a partition which call to stick. From rows 8 to 12 and columns 5 to 9, which represent player states H12 through H16 and dealer states 7 through 10, face cards, and ace, create a partition which call to hit. The remaining partition which spans rows 8 to 17 excluding the previous partition calls to stick.

The policy map formed by SARSA largely retains the same partition, but there is more blending in the decision boundaries. In particular, more calls to hit are present between rows 8 to 12, which represent H12 through H16.

V. ANALYSIS

The results for each algorithm for each MDP were analyzed to determine how the algorithm performed on different values for γ. Then, the algorithms were compared for each MDP to identify differences between algorithm performance.

A. Intra-algorithm Analysis

1) Value Iteration: On the Blackjack MDP, $\gamma = 0.8$ performed the best with a mean V(s) of 0.10075 and held the highest mean V(s) for all γ between iterations 2 through 5. Within the context of the environment, this mean score

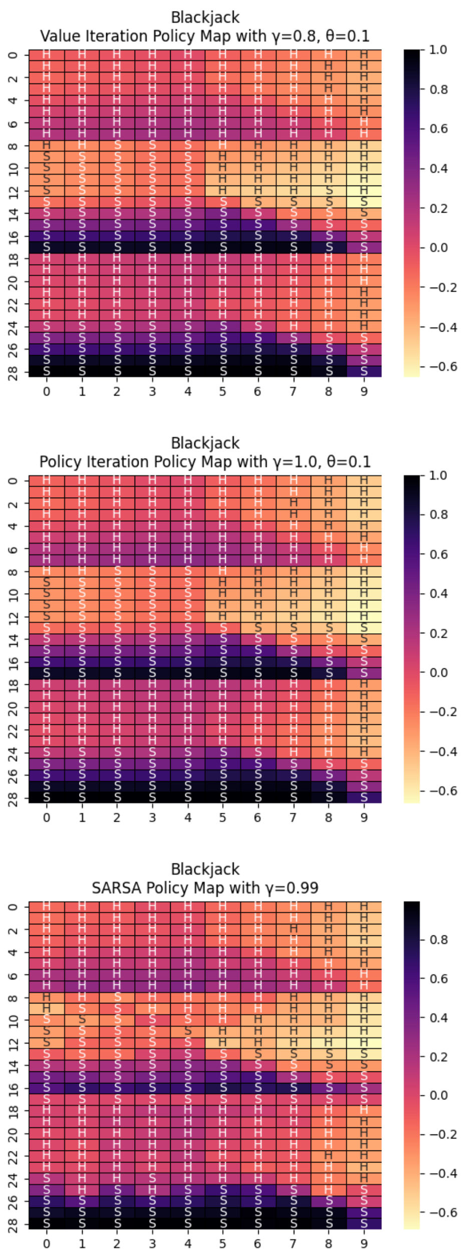


Fig. 7. Value Iteration and Policy Iteration have 5 clear policy partitions. SARSA's policy somewhat follows these partitions, but the decision boundaries are more complex.

is slightly better than drawing on average for every game. Because the lowest γ performs best, this suggests that the agent behaves better when applying some emphasis on immediate reward in addition to future reward. In Blackjack, this can be thought of as playing a more conservative hand; if the agent has a sum of 18 or 19, they do not want to hit in the hopes of getting a 2 or 3. This behavior can be seen in the policy map as well. When the agent can choose to hit or stick, and the sum is between 12 and 21, there is an increased emphasis on selecting to stick.

All γ dropped from a positive score to 0 at iteration 6, and remained at 0 for all future iterations. This can suggest that a degenerate policy is forming, where the agent converges but leads to terminal states with no reward. However, this might also be a symptom of the environment. Blackjack has a short horizon, so episodes typically terminate after a few actions. This further explains why Value Iteration prefers lower γ with Blackjack; because there are fewer future states, immediate rewards matter more.

On the CartPole MDP, $\gamma = 1.0$ performed the best with a mean V(s) of 3.13424 and held the highest mean V(s) for all γ between iterations 0 through 12. Within the context of the environment, this mean score indicates that the agent applied a little over 3 actions on average for every simulation. Because the highest γ performs best, this suggests that the agent behaves better when applying full emphasis on future reward. Since the reward function for CartPole is simply the length of an episode, it makes sense that Value Iteration would place the most emphasis on maximizing the possibility of extending the episode.

All γ dropped from a positive score to 0 at iteration 13, and remained at 0 for all future iterations. In an environment like CartPole, where the objective is to stay alive as long as possible, this is surprising. This suggests that the agent is forming a degenerate policy that is too simple or brittle to generalize to the environment. Poor lifetime might be a symptom of the bin count strategy selected for the environment's variables. Coarse bin counts where equally distributed across cart position and velocity; applying more bins that offer more granularity in the middle and broader representation on the edges might permit the agent to learn a better policy.

2) Policy Iteration: On the Blackjack MDP, $\gamma = 0.99$ performed the best with a mean V(s) of 0.08150 and held the highest mean V(s) for all γ between iterations 4 through 6. Within the context of the environment, this mean score is slightly better than drawing on average for every game. The best γ suggests that the agent behaves best when applying most emphasis on future reward, but gives some consideration to immediate reward. In Blackjack, this can be thought of as playing a riskier hand.

All γ dropped from a positive score to 0 by iteration 6, and remained at 0 for all future iterations. Similar to the discussion with Value Iteration, this might be a symptom of the environment, where episodes are terminated after only a few actions. Despite this, Policy Iteration still prefers a higher γ as it evaluates full policies and can better leverage future returns.

On the CartPole MDP, $\gamma = 1.0$ performed the best with a mean V(s) of 3.13242 and held the highest mean V(s) for all γ between iterations 0 through 7. Within the context of the environment, this mean score indicates that the agent applied a little over 3 actions on average for every simulation. Because the highest γ performed best, this suggests that the agent behaves better when applying full emphasis on policies that maximize future reward. Similar to the behavior seen in Value Iteration, this behavior also makes sense, as the reward provided by CartPole simply equates to the length of the episode.

All γ dropped from a positive score to 0 by iteration 7, and remained at 0 for all future iterations. Similar to what was discussed on Value Iteration with CartPole, this might be an effect of poor bins on the continuous variables in the observable space of the environment.

3) SARSA: On the Blackjack MDP, $\gamma = 0.99$ performed the best with a final average reward of -0.0566. Within the context of the environment, this score is slightly worse than drawing on average for every game. This is indicative of how Blackjack works, where the average score is around -0.0667 per game for humans. Applying a greedy $\epsilon = 0.3$ to this environment seemed to allow the agent to emulate human behavior, but not maximize reward in the game. Because Blackjack is a stochastic game, both in the shuffling of the deck and the dealer's hand, it is important for the agent to learn on multiple states so actions can be generalized. Applying an even greedier ϵ and a more gradual ϵ decay might allow more random actions and states to be visited, which might lead to a strategy which performs better than humans.

On the CartPole MDP, $\gamma = 1.0$ performed the best with a final average reward of 212.3. Within the context of the environment, this score indicates that the agent applied a little over 212 actions on average for every simulation. This is slightly better than 40% of the life cycle of the environment, as CartPole automatically terminates after 500 actions in an episode. Applying a greedy $\epsilon = 0.3$ to this environment seemed to greatly benefit the algorithm. The average reward quickly climbs to the 200s, as seen in Fig. 6, before stabilizing. By allowing the agent to explore several random states early on, it identified actions that maximized reward. It is possible that an even greater ϵ, or a more gradual ϵ decay, would permit even greater average reward.

B. *Inter-algorithm Analysis*

1) Performance on Blackjack: On the Blackjack MDP, Value Iteration converged faster, as seen in Fig. 8. Value Iteration with $\gamma = 0.8$ stabilized on a mean score of 0.08150 after 3 iterations, whereas Policy Iteration with $\gamma = 1.0$ did not stabilize until 4 iterations. There are a few variables that impact this. Firstly, since Value Iteration is functioning under a lower γ, its effective horizon is shorter, which can speed up value propagation. In addition, the stochastic nature of Blackjack makes policy updates in Policy Iteration more expensive, whereas Value Iteration can handle this uncertainty with fewer updates.

Comparing their policy maps, Value Iteration and Policy Iteration yielded the same policies, with variance only in some of the weights assigned to actions.

SARSA performed worse than Value Iteration and Policy Iteration. With $\gamma = 0.99$, the algorithm converged to a lower average reward of -0.0566. While this is slightly better than the average reward for a human player per game, it still results in a net loss, while the policies returned by Value Iteration and Policy Iteration provide a net win. Additionally, the algorithm was significantly slower; SARSA took 33.2 seconds to complete, while the other algorithms executed in under 1 second each.

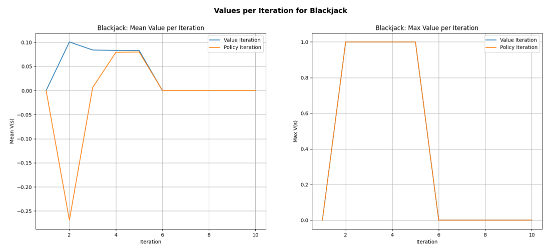


Fig. 8. Algorithm performance on Blackjack, where Value Iteration has $\gamma = 0.8$ and Policy Iteration has $\gamma = 1.0$

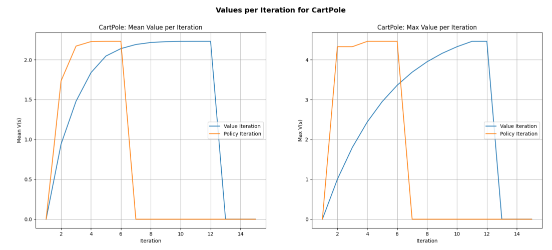


Fig. 9. Algorithm performance on CartPole, where Value Iteration has $\gamma = 1.0$ and Policy Iteration has $\gamma = 1.0$

2) Performance on CartPole: On the CartPole MDP, Policy Iteration converged faster, as seen in Fig. 9. Policy Iteration with a $\gamma = 1.0$ stabilized on a mean score of 3.13424 after 6 iterations, whereas Value Iteration with $\gamma = 1.0$ did not stabilize until 12 iterations. Since CartPole is a discrete state space, this is unsurprising, as Policy Iteration performs best when the environment has well-behaved structure.

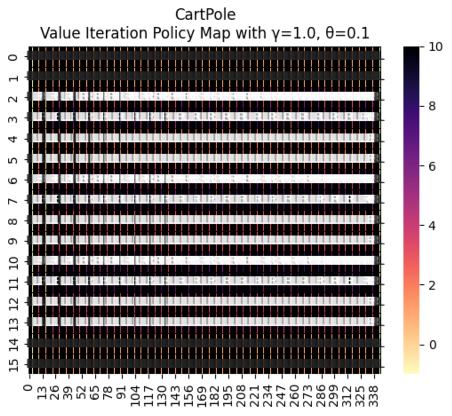
The resulting policy maps for Value Iteration and Policy Iteration on CartPole are included in Fig. 10. While impossible to read, it can be determined visually that the same policy is synthesized from each algorithm.

SARSA scored significantly better than Value Iteration or Policy Iteration, but took significantly more time to execute. With $\gamma = 1.0$, the algorithm converged to a much higher average reward of 212.3, which is more than 670% greater than the average reward from the other algorithms. However, this spike in performance came at a cost. While Value Iteration and Policy Iteration executed in under 2 seconds each, SARSA completed in 65.5 seconds.

VI. CONCLUSION

Two different Markov Decision Processes were executed on Value Iteration, Policy Iteration, and SARSA.

Blackjack is a discrete, stochastic MDP that rewards the agent based on how well it played the game against an opponent with a random shuffled deck. A Value Iteration algorithm with $\gamma = 0.8$ and $\theta = 0.1$ converged fastest on the MDP, generating a policy that prioritized playing conservatively with a final mean V(s) of 0.08150. Value Iteration converged faster than Policy Iteration, which can be attributed to Value Iteration executing on a lower γ and performing better with stochastic environments. A SARSA



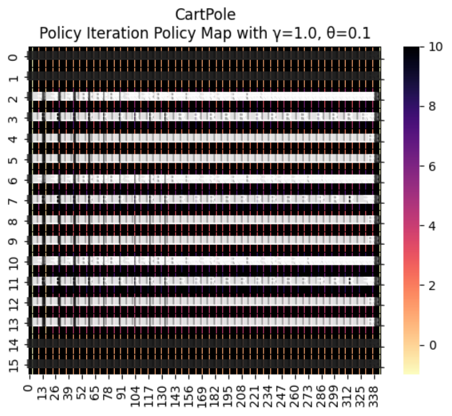


Fig. 10. Value Iteration and Policy Iteration for CartPole are visually identical. The rows represent a combination of angular velocity bins, while the columns represent a combination of cart position and cart velocity bins.

algorithm with $\gamma = 0.99$ and an initial $\epsilon = 0.3$ produced the lowest average score of -0.0566 after 300,000 episodes.

From working with the Blackjack MDP, it has become evident that stochastic environments work better with greedier ϵ for model-free algorithms. Similar to temperature in simulated annealing, applying a greedy ϵ with gradual decay allows the algorithm to explore more of the state space and provide more opportunity to discover the global optima. This is an opportunity for improvement on this work with SARSA; by providing a greedier ϵ, the algorithm might have learned a policy that performed much better than humans, and possibly better than the model-based algorithms.

CartPole is a continuous, deterministic MDP that rewards the agent for surviving longer in each iteration, and requires continuous variables such as cart position, cart velocity, and pole angular velocity, to be divided into bins. For all algorithms, cart position and cart velocity were evenly divided into 4 bins each, and pole angular velocity was evenly divided into 15 bins. A Policy Iteration algorithm with $\gamma = 1.0$ converged fastest on the MDP with a final mean V(s) of 3.14242. Policy Iteration converged faster than Value Iteration, which can be attributed to Policy Iteration preferring discrete state spaces. A SARSA algorithm with $\gamma = 1.0$ and an initial $\epsilon = 0.3$ produced the highest averaged score of 212.3 after 5,000 episodes.

Working with the CartPole MDP highlighted how bin counts affect the performance of all reinforcement learning algorithms. From exploring different permutations of bin counts, increasing bin counts always improved reward at the cost of execution time. However, extremely coarse bins, such as the bin count of 4 used for theses experiments, restricted algorithms from gaining insightful feedback and thus capped the resulting policies.

Creating bins with more granularity would provide space for more meaningful insight to propagate. Additionally, partitioning bins to have a variety of sizes would permit the agent to execute very small adjustments that might stabilize the pole. Aside from applying these bins, there could have been more thorough investigation on the permutations of bin sizes. The permutations limited some variables to only coarse bins, and all permutations were executed on Value Iteration. From these experiments we conclude that Policy Iteration performed better on CartPole; applying this algorithm over Value Iteration might yield more meaningful results.

REFERENCES

[1] R. S. Sutton, A. G. Barto, *Reinforcement Learning: an Introduction*, 2nd. ed. Cambridge, The MIT Press, 2015. pp. 96–100, pp. 154–157. Accessed: Jul 17, 2025. Online. Available: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
[2] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, O. G. Younis. Gymnasium Documentation. https://gymnasium.farama.org/ Code version: v1.2.0. (accessed Jul. 9, 2025).
[3] J. Mansfield. Better MDP Tools Documentation. https://jlm429.github.io/bettermdptools/bettermdptools.html Code Version: v0.8.1. (accessed Jul. 9, 2025).
[4] The Matplotlib development team. Matplotlib Documentation. https://matplotlib.org/3.8.4/index.html Code Version: v3.8.0. (accessed Jul. 9, 2025).
[5] NumPy Developers. NumPy Documentation. https://numpy.org/doc/1.26/index.html Code Version: v1.26.0. (accessed Jul. 10, 2025).