Exploring Reinforcement Learning Algorithms in Deterministic and Continuous State Spaces: Blackjack and Cartpole

Tyler Dickson
Georgia Institute of Technology
tdickson30@gatech.edu

***Abstract*—This paper analyzes interesting properties involved in the Markov Decision Process (MDP) through the analysis of two classic reinforcement learning environments as part of the gymnasium library: Blackjack and Cartpole. By comparing model-based approaches such as Policy Iteration (PI) and Value Iteration (VI) and model-free approaches such as Q-Learning, this work investigates the computational efficiency, convergence behavior, and policy outcomes across deterministic and continuous state-space environments. Results of the study demonstrate that PI and VI consistently converged to optimal policies in Blackjack, highlighting their reliability in structured, deterministic environments. However, in Cartpole, the computational trade-offs of state-space discretization revealed VI's sensitivity to hyperparameters and computational cost. Q-Learning demonstrated flexibility and computational efficiency, converging more quickly than PI and VI in both environments, as model-free approaches like Q-Learning are better suited for solving CartPole because of their ability to learn policies directly from interactions with the environment without requiring explicit knowledge of the transition probabilities. These results highlight the adaptability of reinforcement learning algorithms to various problem domains and the importance of selecting the right algorithm to fit various domains, hyperparameter tuning, and reward structure design. The study reaffirms the value of these well established environments as benchmarks for testing reinforcement learning techniques.**

I. Introduction

Reinforcement Learning (RL) is a branch of machine learning that enables an agent to interact with its environment, learn from the outcomes of its actions, and make decisions that maximize its long-term rewards. The agent achieves this by associating mathematical values with distinct states and actions within a framework known as a Markov Decision Process (MDP). MDPs provide a structured way to model decision-making problems under uncertainty by capturing the relationships between states, actions, rewards, and transitions. [4]

This paper explores the application of RL techniques to solve two MDP problems: Blackjack and CartPole, both of which are included in the Gymnasium library. Blackjack is selected to represent a small MDP problem space due to its discrete and finite state space, making it computationally feasible to apply MDP solving model-based approaches like Value Iteration (VI) and Policy Iteration (PI). Conversely, CartPole represents a large MDP problem space with its continuous state variables where outcomes are dictated by laws of physics. This makes it a more computationally intensive and challenging environment to solve.

First, we begin by discussing the state and action spaces of both MDPs and discussing their interesting properties. Blackjack, as a game of probabilistic decision-making, highlights the nuances of balancing risk and reward, while CartPole emphasizes control under continuous environmental factors. Next, we apply Value Iteration and Policy Iteration to each problem, comparing their convergence behaviors, computational efficiency, and the quality of the derived policies. Finally, we employ Q-Learning, a model-free reinforcement learning algorithm to solve each MDP problem. This allows us to assess how a model-free approach compares to model-based methods like VI and PI, specifically in cases where the underlying transition dynamics and rewards are unknown. Additionally, we explore different exploration strategies to optimize learning in each environment and discuss their impact on performance and convergence speed. These comparisons will demonstrate the strengths and limitations of various RL approaches in handling both small and large MDPs.

II. Problem Space: Understanding Our MDP Problems

*a) **Blackjack**:* Blackjack is a classic card game where the agent's goal is to beat the dealer by accumulating a hand value closer to 21 than the dealer's hand, without exceeding 21 (referred to as "busting"). The state space in the context of Blackjack encapsulates all possible combinations of the player's hand, the dealer's visible card, and the presence of a usable Ace. This state space is represented as a tuple with three discrete components: **Player's current hand sum:** The total value of the player's hand, ranging from 4 to 21. **Dealer's showing card:** The value of the card visible to the player, ranging from 1 (Ace) to 10. **Usable ace:** A binary value indicating whether the player has a usable Ace (1 for true, 0 for false). A "usable" Ace can be counted as 11 without causing the hand to exceed 21. The action space for Blackjack is discrete and consists of two possible actions: **Stick (0):** Ends the player's turn, allowing the dealer to play according to predefined rules. **Hit (1):** The player requests an additional card to increase their hand total. [1]

Blackjack's state and action spaces make it a relatively small and manageable MDP problem, as the finite and discrete nature of the state space allows for a relatively straightforward application of Value Iteration (VI) and Policy Iteration (PI). The game's probabilistic nature adds an interesting layer of complexity that requires the agent to balance risk and reward when deciding between actions. [1]

*b) **Cartpole**:* CartPole is a control problem where a pole is attached to a cart via an unactuated joint on a frictionless track. The objective is to keep the pole balanced upright while the cart moves back and forth by applying forces in the left or right direction. The observation space for CartPole is represented as a 4-dimensional continuous vector, with the following components: **Cart Position:** The position of the cart on the track, with a range of **[-4.8, 4.8]**. However, the episode terminates if the cart moves outside range **[-2.4, 2.4]**. **Cart Velocity:** The velocity of the cart, which is unbounded **[-Inf, Inf]**. **Pole Angle:** The angle of the pole with respect to the vertical axis, ranging from approximately **[-0.418, 0.418]** radians. The episode terminates if the pole angle exceeds these bounds. **Pole Angular Velocity:** The rate of change of the pole's angle, which is also unbounded **[-Inf, Inf]**. The action space for CartPole consists of two discrete actions: **Left (0):** Applies force to push the cart to the left. **Right (1):** Applies force to push the cart to the right.

Unlike Blackjack, CartPole's state space is continuous, making it a much larger and more complex MDP problem. This complexity arises from the infinite possible values for state variables within their defined ranges, as well as the continuous interaction of dynamics governed by forces and gravity. The Gymnasium library simulates these dynamics. CartPole is particularly interesting because it demonstrates the need for more advanced RL techniques such as custom reward shaping and convergence criteria. As we explore this problem space, the need to discretize the state space will arise for VI and PI, which introduces computational challenges. Conversely, model-free approaches like Q-Learning are better suited for solving CartPole, as they learn policies directly from interactions with the environment without requiring explicit knowledge of the transition probabilities. [2]

A. *Hypothesis: Blackjack*

Policy Iteration (PI) will converge faster than Value Iteration (VI) due to PI's iterative improvement of a fixed policy, which generally requires less iterations over the state space. Both VI and PI will converge to the same optimal policy, as the Blackjack environment is a deterministic MDP where the outcomes are governed by predefined rules and probabilities associated with drawing cards.

For Q-learning, since it is a model-free approach, I expect it to converge more slowly compared to VI and PI because it requires interacting with the environment to approximate the state-action value function. However, Q-Learning should ultimately approximate the optimal policy obtained by VI and PI.

The discrete nature of Blackjack's state and action spaces should make it feasible for all three algorithms to converge on an effective policy, with PI likely being the most computationally efficient.

B. *Hypothesis: Cartpole*

Value Iteration (VI) will converge faster than Policy Iteration (PI) in terms of iterations because it direct updates the value function, but its computational cost will significantly increase with the discretization of the continuous state space. PI will require more iterations and computation per policy refinement but should ultimately converge to the same optimal policy as VI since the environment is deterministic.

Q-Learning will likely converge more slowly than both VI and PI since it relies on interaction with the environment to learn the state-action values and feel out the reward shaping functionality. The discretized state space and reliance on exploration may lead to variability and slower convergence. However, with enough episodes, Q-Learning should approximate the optimal policy derived by VI and PI (pending no updates to reward shaping functions).

III. Blackjack & Cartpole: PI & VI Methodology

In solving the both the Blackjack and Cartpole environments, both Policy Iteration (PI) and Value Iteration (VI) aim to learn an optimal control policy that maximizes cumulative rewards. The agent achieves this by iteratively refining the policy that maximizes the value function until it converges to the optimal policy. The optimal policy defines the best action to take in every state to maximize long term rewards. The agent's objectives can be expressed mathematically through a reward function, which assigns numerical values to outcomes based on the agent's actions and transitions in the environment. [4]

In both environments, the agent must address the exploration vs. exploitation tradeoff. The agent can only perceive a sequence of immediate reward values as the agent executes its sequence of actions. Therefore, the agent is faced with the problem of temporal credit assignment—determining which of the actions in its sequence can be credited with producing the eventual rewards. [3] The learner faces a tradeoff in choosing whether to favor exploration of unknown states and actions (to gather new information), or exploitation of states and actions that it has already learned will yield high reward (to maximize its cumulative reward).

Initially, the agent may favor exploration to gather information about the environment and learn the consequences of different actions. However, as the policy stabilizes and converges toward optimality, the agent increasingly relies on exploitation, favoring actions that yield the highest rewards based on the learned policy. [3]

Both PI and VI leverage **Bellman's Equation** (1) to guide this process, ensuring that the value of each state reflects the maximum achievable reward based on the agent's decisions. Since both Blackjack and Cartpole are deterministic MDPs, where state transitions, rewards, and rules are predefined,

there is a single mathematically optimal policy for each environment. [4] This determinism simplifies the problem and guarantees that both PI and VI will converge to the same optimal policy. However, CartPole's continuous state space and the need for discretization make it more computationally demanding than Blackjack, which operates in a smaller, discrete state space.

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right] \quad (1)$$

[3]

IV. Blackjack: Policy Iteration, Value Iteration, & Q-Learning

A. *Blackjack: PI & VI Approach*

*a) **Defining Convergence**:* For both PI and VI, convergence is determined by monitoring the maximum change in the value function (ΔV) across all states during policy evaluation. [3] Specifically, the algorithm checks that delta remains below a small threshold (θ) for **10** consecutive iterations. This ensures that the value function has stabilized, and no further updates would impact the policy.

This approach works particularly well for Blackjack's deterministic environment because the state transitions and rewards are explicitly defined and follow fixed probabilities. Since there is no randomness in the outcomes of actions under a given policy, the value function converges with consistency. The deterministic nature of the environment guarantees that once the policy reaches convergence, the associated value function remains unchanged across iterations, satisfying the convergence criteria.

By requiring stability over multiple iterations (**10** consecutive iterations), the stopping condition also accounts for any potential oscillations or minor fluctuations in the value updates. This ensures that the algorithm does not terminate prematurely and consistently identifies the optimal policy.

*b) **Reward Shaping**:* Of note, implementing reward shaping modifications was not necessary to achieve convergence for PI and VI

*c) **Hyperparameter tuning**:* Using Optuna, I conducted a hyperparameter search over **100** trials to tune the discount factor (γ) and convergence threshold (θ). The results showed that values of gamma in the range **[0.90, 0.99]** and theta values between [10^{-5}, 10^{-2}] yielded meaningful results. These ranges allowed the policy evaluation to reach convergence within a reasonable number of iterations while maintaining an accurate value function approximation.

B. *Blackjack: Policy Iteration Results*

The Policy Iteration implementation successfully converged after **49** iterations, with a total wall clock time of **0.13** seconds. This rapid convergence validates the hypothesis that Policy Iteration would be able to leverage the deterministic nature of the Blackjack environment, and would efficiently identify the optimal policy with minimal computational overhead. The results align with the expectation that Policy Iteration would achieve convergence relatively quickly by iteratively refining a fixed policy.

While the complete optimal policy is extensive and unsuitable for inclusion in this paper, its structure aligns with the principles of an optimal Blackjack strategy:

For states with high hand values (**18** and above), the policy mostly favors sticking (action **0**) to minimize the risk of busting. For lower hand values (**12–16**), the policy varies depending on the dealer's visible card and whether the player has a usable Ace. The presence of a usable Ace allows for more aggressive actions (hitting, action **1**) due to its flexibility in avoiding busts.

Figure 1 depicts mean and max value's returned across iterations until convergence is reached at iteration **49**. It demonstrates how the mean and maximum values of the state-value function (V_s) stabilized over successive iterations. Notably, the value function rapidly increased during the initial iterations as the policy was iteratively improved, followed by a plateau where minimal changes occurred. This behavior reflects the agent quickly learning effective strategies and then refining them for minimal gains.

Figure 5 depicts the delta convergence. The plot tracks the maximum change (Δ) in the value function across all states during policy evaluation. The sharp spikes indicate significant updates to the value function in the early iterations, followed by a steady decrease as convergence approached. The algorithm satisfied the convergence criteria after stabilizing, confirming the effectiveness of the stopping condition.

Figure 3 highlights the average rewards obtained during the policy improvement phase until convergence is reached at iteration **49**. While the initial policy updates led to slight fluctuations in rewards, the final policy demonstrated consistent improvement, aligning with the goal of maximizing cumulative rewards.

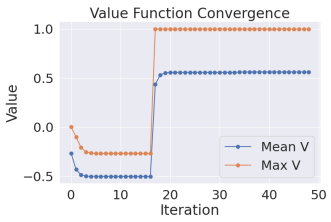

Fig. 1. PI: Value Function Convergence

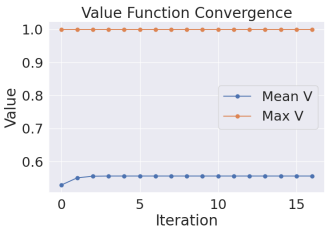

Fig. 2. VI: Value Function Convergence

This optimal policy supports the hypothesis that Policy Iteration would reliably converge to a single, mathematically optimal strategy in Blackjack, where deterministic rules and transitions ensure consistent results. The minimal changes required in the later iterations further reflect the efficiency of the algorithm in such environments. As seen in the convergence plots 1 and 3.

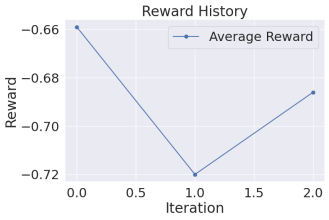

Fig. 3. PI: Reward History

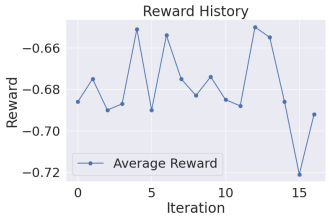

Fig. 4. VI: Reward History

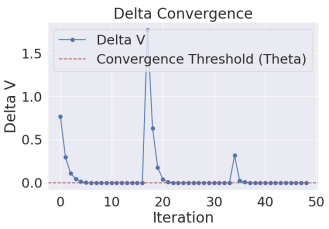

Fig. 5. PI: Delta Convergence

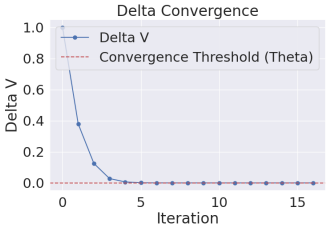

Fig. 6. VI: Delta Convergence

C. *Blackjack: Value Iteration Results*

In Value Iteration (VI), the agent's goal is to directly compute the optimal value function for all states to derive the optimal policy. Unlike Policy Iteration (PI), which alternates between policy evaluation and improvement, VI iteratively applies Bellman's optimality equation 1 to update the value of each state until convergence. Upon stabilization, the optimal policy is determined by choosing actions that maximize the value for each state.

Similar to PI, I used Optuna — tuning parameters over **100** trials before settling on optimal values for γ: **0.944** and θ: **9.470**x10^{-5}. Value Iteration converged after **17** iterations, which is significantly faster than the **49** iterations required for Policy Iteration. Despite requiring fewer iterations, VI had a higher total wall clock time of **0.69** seconds, compared to **0.13** seconds for PI, a notably small difference in wall clock time. The difference is likely because VI performs more extensive computations in each iteration, updating the value function for all states at once, whereas PI evaluates a single policy at a time. The faster iteration count for VI aligns with its theoretical advantage in environments with small or moderately sized state spaces. Notably, both algorithms converged to the same optimal policy. Consistent with the hypothesis, this is expected for this deterministic Blackjack environment. This consistency demonstrates that both methods reliably solve the MDP.

The value function convergence (Figure 2) depicts the mean and max value's returned across iterations until convergence is reached at iteration **17**. The plot shows the max value was initially reached early in the process. This is expected because VI aggressively updates all state values to reflect the maximum possible return from any state at each step. However, the mean value lagged behind, resulting in a wider gap between max and mean. This difference occurs because VI prioritizes maximizing the value of reachable states, overestimating their values in the early iterations. By comparison, PI's policy evaluation phase provides a more gradual adjustment to the value function.

Figure 6 depicts the delta convergence. Notably, VI produces smoother convergence in comparison to PI (Figure 5), as the policy evaluation step stabilizes the value function before improving the policy. This leads to a more gradual but computationally efficient refinement process.

Figure 3 depicts the reward history across iterations until convergence is reached at iteration **17**. VI inherently balances exploration and exploitation by propagating maximum rewards backward through the state space in every iteration. This contrasts with PI, which explicitly evaluates and improves the policy in separate steps. VI's aggressive updating can lead to quicker identification of high-reward paths but may cause more variation in value estimates.

D. *Blackjack: Q-Learning Approach to Solving*

I took an ϵ-greedy approach (further explained in Section V-D) to maximize rewards while integrating specific adjustments to handle convergence challenges. A key difference between Q-Learning and the model-based methods (PI and VI) is the reliance on experience sampling. This introduces variability, making it harder to converge without careful reward shaping and hyperparameter tuning. In my initial implementations, the algorithm struggled to converge due to ambiguous feedback from the environment.

*a) **Reward Shaping**:* To address subobtimal policies and failure to converge, I introduced several reward shaping functions that incentivized the agent to favor safer, reward-maximizing actions in challenging states and mitigated the risk of the agent converging to suboptimal policies.

Penalty for sticking with low hands: If the dealer showed an Ace and the agent's hand was below **16**, the agent was penalized **-4** for sticking but rewarded **+4** for hitting. **Reward for high, non-busting hands:** The agent received a reward of **+5** for achieving a hand value above **18** and less than or equal to **21**. **Penalty for busting:** Any hand value above **21** incurred a penalty of **-5**.

Another significant challenge was the decay of the exploration rate (ϵ). Early implementations saw ϵ decay too quickly, leading the agent to prematurely favor exploitation over exploration before the Q-values were sufficiently learned. To counter this, I implemented a function to slow the decay rate, allowing the agent more time to explore and sample from the state space effectively.

*b) **Defining Convergence**:* I set a fixed threshold of **0.02** for the change in Q-values (ΔQ) and required this threshold to hold for a minimum of **20** consecutive episodes. If ΔQ fell below this threshold, the algorithm terminated, indicating the Q-values had stabilized. This approach worked well in ensuring reliable convergence.

Finally, I tuned hyperparameters, including the learning rate (α), discount factor (γ), exploration rate (ϵ), and epsilon decay. This resulted in best parameters: α: **0.0127**, γ: **0.1163**, ϵ: **0.9684**, and epsilon decay: **0.8286**. These adjustments

collectively enabled the Q-Learning agent to overcome initial failures and converge to an effective policy for Blackjack.

E. Blackjack: Q-Learning Results

The Q-Learning implementation successfully converged at episode **33**, reaching the convergence criteria established Section IV-D0b. Convergence occurred due to effective reward shaping, epsilon decay adjustments, and optimized hyperparameters. The total wall clock time for this process was negligible (**0.00** seconds), demonstrating the computational efficiency of Q-Learning for the relatively small state space of Blackjack.

Figure 7 depicts Delta Convergence, and shows the maximum change in Q-values (ΔQ) per episode. The initial episodes exhibit large fluctuations as the algorithm explores the state space and updates Q-values aggressively. Notably, a sharp spike in ΔQ is observed around episode **15**, indicating a significant update to Q-values for specific states, likely due to newly learned high-reward paths. After this spike, ΔQ rapidly declines, stabilizing below the convergence threshold by episode **33**. This stabilization confirms that the Q-values have effectively converged.

Figure 8 depicts the Mean vs Convergence over iterations, and tracks the average value of all states across episodes. The gradual increase in the mean state values reflects the agent's improved understanding of the environment over time, as it learns to favor high-reward actions. The values stabilize as the policy converges, with the mean values reaching their highest point near the detected convergence episode.

Figure 9 illustrates the total reward obtained by the agent in each episode. Early episodes show high variability as the agent explores and occasionally receives significant penalties for suboptimal actions like busting. An example of this occurs around episode **15**, resulting in a large dip. This corresponds to poor policy decisions during exploration. However, as the policy converges, the agent's rewards stabilize, indicating consistent decisions aligning with the learned policy.

Figure 10 depicts Epsilon Decay, which shows the decay of the exploration rate (ϵ) over episodes. Initially, ϵ is high, encouraging exploration of the state space. The decay is gradual due to the tuned epsilon decay parameter, allowing necessary exploration before shifting toward exploitation. By the time the agent converges, ϵ levels off, prioritizing exploitation of the learned policy.

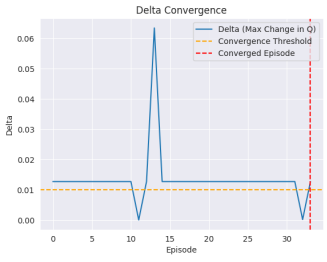

Fig. 7. QL: Delta Convergence

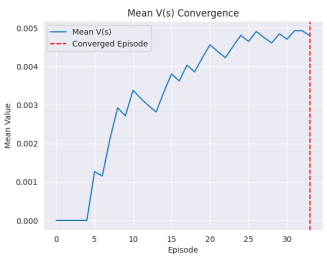

Fig. 8. QL: Mean vs Convergence

Fig. 9. QL: Reward History

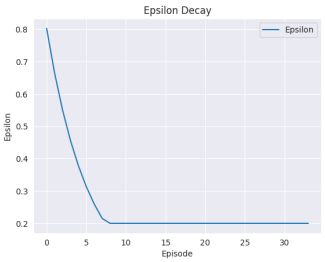

Fig. 10. QL: Epsilon Decay

Hypothesis Discussion The Q-Learning algorithm converged to a different optimal policy than the policies derived from Policy Iteration (PI) and Value Iteration (VI). While PI and VI produced identical policies as expected, Q-Learning deviated due to the introduction of reward shaping functions that influenced the learning process. This outcome partially aligns with the hypothesis. The reward shaping functions introduced biases, causing the learned policy to deviate. These functions, designed to guide the agent by penalizing risky decisions and rewarding favorable outcomes, improved convergence speed and stability but altered the reward structure of the MDP. This adjustment led the agent to prioritize specific actions that differed from the mathematically optimal policy derived by PI and VI.

V. CartPole: Policy Iteration, Value Iteration, & Q-Learning

A. Cartpole: PI & VI Approach

*a) **Initial Issues**:* My initial implementation of CartPole presented challenges due to its continuous state space. Unlike Blackjack's discrete environment, CartPole's states (cart position, velocity, pole angle, and angular velocity) exist on a continuous spectrum. This requires implementation of discretization to make the problem controllable for all implementations. As a result, the state space was divided into bins using a resolution parameter, allowing each state variable to map to a discrete bin. Discretization introduced tradeoffs between computational efficiency and accuracy. Smaller bins represent smaller divisions of a continuous state space. This produces higher precision but is computationally more expensive. Conversely, larger bins are less precise, but are less computationally complex, which can result in less optimal policies. [4]

*b) **Defining Convergence**:* To ensure robust policy convergence, two criteria were employed.

Value Function Stability: The maximum change in the value function ΔV across all states during one iteration had to fall below a predefined threshold θ. After tuning parameters, theta is defined as **0.0016**.

Policy Stability: The policy was considered stable when no state's best action changed after the policy improvement step. This was enforced using a boolean flag to track policy changes across iterations. Convergence was achieved only when both conditions were satisfied.

*c) **Reward Shaping**:* Both PI and VI were initially implemented without reward shaping, but this led to difficulties in achieving convergence. To address this, reward shaping was introduced to better guide the agent's learning process. These rewards were combined with the default environment rewards to provide more robust feedback:

Penalized large pole angles: A penalty proportional to the pole's deviation from the vertical position was added:

$$\text{pole\_angle\_penalty} = -|\text{pole\_angle}| * \text{penalty\_weight} \quad (2)$$

Reward centered cart position: The agent was rewarded for maintaining the cart near the center of the track:

$$\text{cart\_pos\_reward} = -|\text{cart\_pos}| * \text{pos\_weight} \quad (3)$$

Encourage high angular velocity for recovery: Angular velocity aiding in recovery was rewarded:

$$\text{recovery\_reward} = \text{pole\_ang\_velocity} * \text{velocity\_weight} \quad (4)$$

*d) **Hyperparameter tuning**:* Hyperparameters state bin resolution, gamma (γ), and theta (θ), were tuned using Optuna. The state bins were created with the following ranges for discretization: cart position: **[-2.4, 2.4]**, cart velocity: **[-3.0, 3.0]**, pole angle: **[-0.2, 0.2]**, and pole angular velocity: **[-2.0, 2.0]**. After tuning, optimal parameters were calculated for γ: **0.8819**, θ: **0.0016** (convergence threshold), and state bin resolution: **12**.

B. *Cartpole: Policy Iteration Results*

Policy Iteration for the CartPole environment demonstrated significant computational challenges. Initial runs with Optuna using multiple trials resulted in excessive computational strain, requiring adjustments to the number of trials. As a compromise, the number of trials was reduced to **1** for initial tuning, followed by a focused re-run over **70** iterations using the best hyperparameters obtained. Despite these constraints, Policy Iteration successfully converged after **49** iterations when the convergence criteria outlined in Section V-A0b were both satisfied.

An average reward of **10.2** was acheived, confirming the effectiveness of the tuned parameters in balancing the exploration vs. exploitation tradeoff. However, the computational cost was notable, with a total wall clock time of **85.69** seconds for convergence. Many iterations leading prior to reaching convergence lasted over **200** seconds. This result aligns with the hypothesis that Policy Iteration is computationally expensive in CartPole's continuous environment.

Delta convergence (Figure 11) shows the gradual reduction of V_s over **49** iterations. Initially, the value function drops significantly, reflecting rapid policy updates during the early stages. By iteration **30**, V_s falls below the predefined threshold for θ (**0.0016**), but fails to satisfy both convergence criteria outlined until iteration **49**.

Figure 13 captures fluctuations in average reward across iterations. Unlike Blackjack, the CartPole's continuous environment introduces more variability, as seen in the oscillating reward values. Although never fully stabilizing, the algorithm reaches terminal state at iteration **49**, aligning with the convergence criteria. This highlights how the environment's continuous nature adds complexity to learning an optimal policy.

Figure 15 depicts the mean and maximum value functions across iterations. Both metrics displayed a steady decline, reflecting the CartPole environment's rewards structure and penalty system for instability. Initially, both metrics showed a sharp initial decrease. Around iteration **25** they began to stabilize, showing that the value function accurately reflects the long term expected rewards for each state action pair.

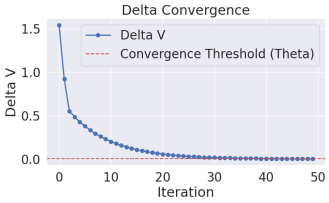

Fig. 11. PI: Delta Convergence

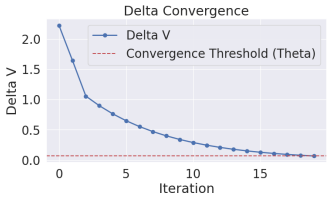

Fig. 12. VI: Delta Convergence

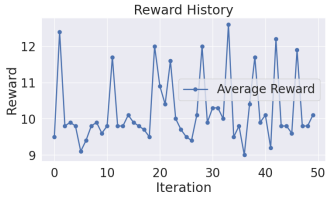

Fig. 13. PI: Reward History

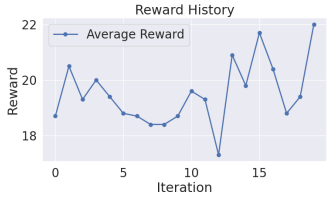

Fig. 14. VI: Reward History

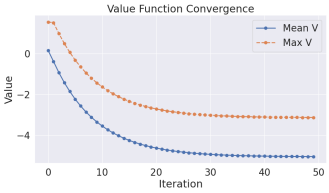

Fig. 15. PI: Value Function Convergence

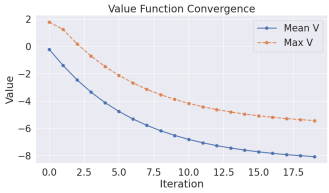

Fig. 16. VI: Value Function Convergence

C. *Cartpole: Value Iteration Results*

Similar to issues faced during Policy Iteration (PI) implementation, Value Iteration (VI) also presented a computational challenge. I used Optuna to tune using just 1 trial due to high computational demands. Optuna reported tuned parameters of γ: **0.8505**, θ: **0.0723** (convergence threshold), and state bin resolution: **17**. Using these parameters, VI managed to converge after **19** iterations, significantly fewer than the **49** iterations required by PI. The final wall clock time was **97.58** seconds, which was higher than the **85.69** seconds recorded for PI, contrary to initial expectations based on the Blackjack results. This discrepancy can be attributed to the higher computational cost of VI, which directly updates the value function for all

states in each iteration. Unlike PI, which evaluates a fixed policy during its iterations, VI requires more intensive updates to all action state pairs in Cartpole's continuous state space.

Figure 11 depicts the delta convergence across the specified θ threshold. The curve shows a steady decrease in the maximum change in V_s. By iteration **19**, V_s falls below the convergence threshold for θ: **0.0723**, signaling convergence. The steep decline in the initial iterations indicates that VI rapidly improves the value estimates early on but requires additional iterations to tune values.

The reward history graph (Figure 14) depicts fluctuations in average reward over iterations, with an upward trend as the algorithm converges. By iteration **19**, the average reward stabilizes at **22.0**, reflecting the effectiveness of the optimized policy in maintaining the pole's balance. The early instability likely results from the exploration of suboptimal policies prior to convergence.

The Value Function Convergence (Figure 16) graph highlights the decreasing mean and maximum value function results across iterations. The gap between mean and maximum values narrows as the iterations progress, indicating improved consistency across the state space.

Hypothesis Discussion The results somewhat align with the hypothesis II-B. As expected, VI converged in fewer iterations compared to PI due to its direct updates to the value function. However, the computational cost of VI was higher, leading to a longer wall clock time despite the reduced iteration count. This is consistent with the hypothesis that VI's efficiency is offset by the computational expense of updating a discretized continuous state space. Contrary to my hypothesis (II-B), the policies returned by PI and VI were different. The policy generated by PI was very long—far too lengthy to include in the report—while the policy from VI was so large it exceeded heap memory allocation when attempting to print. This divergence suggests that despite the deterministic nature of the CartPole environment, nuances in the implementation, such as discretization or convergence criteria, may have influenced the resulting policies.

D. *Cartpole: Q-Learning Approach to Solving*

The Q-learning implementation used an ϵ-greedy strategy for action selection. At each step, the agent selects the action

$$A_t = \arg\max_a Q_t(a) \tag{5}$$

where $Q_t(a)$ is the estimated value of taking action a in the current state. Conversely, with probability ϵ, the agent explores by randomly sampling an action instead of greedily exploiting the current knowledge.

The formula emphasizes how ϵ-greedy balances exploitation, which uses the action with the highest current $Q_t(a)$ value to maximize immediate rewards, and exploration, which investigates other actions to discover potentially better long-term strategies. [5] In Cartpole's implementation, optimal performance required sampling actions that helped the cart recover balance even when such actions appeared suboptimal initially.

The high ϵ value **(0.9576)** combined with a moderate decay rate ensured that the agent extensively explored the environment in early episodes. This likely led to faster learning of optimal state action pairs. This strategy was particularly effective in CartPole's continuous environment.

*a) **Reward Shaping**:* Similar to PI and VI implementations, I imposed similar reward shaping methods as highlighted in Section V-A0c. Penalty criteria were included in parameter tuning, which resulted in optimal parameters for reward shaping of penalty weight: **0.0225**, position weight: **0.0290**, and velocity weight: 0.4919,

*b) **Defining Convergence**:* Convergence was defined as the maximum change in ΔQ being less than a threshold of **0.02** for **20** consecutive iterations. Upon meeting this criteria, the algorithm transitioned to a terminal state and printed final results. Despite using a seed for reproducibility, convergence behavior varied slightly across iterations during development. This variability stems from the stochastic nature of Q-learning, as it relies on sampling transitions from the environment. This can introduce slight deviations depending on exploration vs exploitation tradeoffs.

*c) **Tuning parameters**:* Using Optuna, I tuned parameters over **20** trials to optimize Q-learning performance. The best parameters identified were α (learning rate): **0.0119**, γ: (discount factor): **0.0691**, ϵ (exploration rate): **0.9576** ϵ-decay: **0.9421**, and state bin Resolution: **9**.

E. *Cartpole: Q-Learning Results*

To provide a consistent analysis, I selected an implementation where convergence occurred in later iterations, enabling better visualization of the algorithm's learning patterns.

The Q-learning algorithm converged at **413** episodes, with a total runtime of just **0.43** seconds, significantly faster than both Policy Iteration (**85.69** seconds) and Value Iteration (**97.58** seconds). This significant reduction in runtime is attributed to Q-learning's sample based updates, which avoid iterating over all states and actions in every step. Instead, Q-learning updates only the current state action pair, resulting in fewer computations per iteration compared to the full state-action space updates required in PI and VI. Additionally, the epsilon-greedy strategy and discretization resulted in less complex computations by limiting the number of state transitions the agent evaluates at each step. [3]

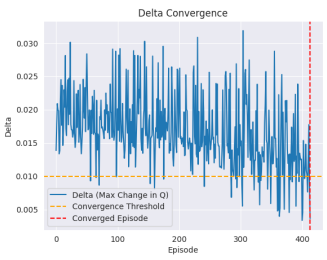

Fig. 17. QL: Delta Convergence

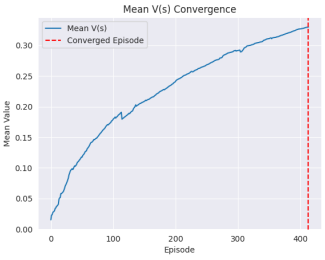

Fig. 18. QL: Mean vs Convergence

Figure 18 depicts the mean ΔQ over iterations until reaching convergence. There is a consistent increase in ΔQ as

Fig. 19. QL: Reward History

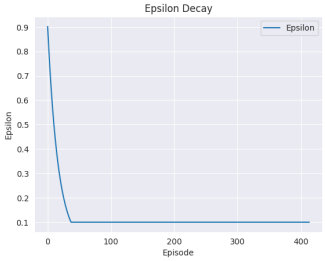

Fig. 20. QL: Epsilon Decay

the Q-Learning algorithm updates the Q-table. The upward trend indicates that the algorithm is learning increasingly better policies over time. Convergence is illustrated at episode **413** with the red dashed line, which corresponds to the defined convergence criteria.

Delta convergence (Figure 17) tracks the maximum change in Q-values across all states during each episode. Early iterations display fluctuations due to exploration and learning adjustments. However, in later iterations, delta stabilizes below the convergence threshold of **0.02** for **20** consecutive iterations upon reaching the the **413th** episode. The variability in earlier iterations reflects the influence of the ϵ-greedy policy, as exploratory actions create larger Q-value updates before the policy refines itself.

Reward vs. Episode: Figure 19 shows the total reward accumulated per episode. The spikes in reward indicate episodes where the agent performed well, while the dips correspond to less successful episodes due to exploration or poorly optimized actions. In early iterations, fluctuations are more wide. However, over time the reward becomes less volatile, reflecting the agent's improved consistency in balancing the cart as the Q-values converge.

The epsilon decay plot (Figure 20) demonstrates how exploration probability diminishes over time. Starting at a high value which corresponds to our tuned parameter of **0.9576**, epsilon gradually decays to its lower bound, which was hard coded at **0.1**. This transition reduces exploratory actions in favor of exploiting the learned policy, contributing to the stability of the Q-values and reward patterns in later episodes.

*a) **Hypothesis Discussion**:* My hypothesis predicted that Q-Learning would converge more slowly than both VI and PI due to its reliance on environment interactions and stochastic exploration. However, this was incorrect. Q-Learning converged much faster than both VI (**19** iterations, **97.58** seconds) and PI (**49** iterations, **85.69** seconds).

This discrepancy arose because Q-Learning avoids the exhaustive updates across all state action pairs required by VI and PI. Instead, it incrementally updates the Q-values for visited state action pairs, resulting in significantly fewer computations per iteration. The faster convergence was achieved without sacrificing computational efficiency, as the algorithm focused only on sampled transitions rather than the full state-action space. [1]

Additionally, the reward shaping functions improved learning efficiency, prioritizing recovery and balancing actions, which likely accelerated convergence. Interestingly, the policy returned by Q-Learning was significantly smaller than the policies generated by PI and VI. This smaller policy highlights the algorithm's focus on sampled transitions rather than exhaustive evaluation of the entire state action space, which further contributes to its computational efficiency. However, the smaller policy also reflects its departure from the optimal policies derived by PI and VI, diverging from the original hypothesis.

Conclusion

The results for Policy Iteration (PI) and Value Iteration (VI) across both Blackjack and Cartpole demonstrate the strengths and challenges of model-based reinforcement learning in deterministic environments. In Blackjack, both PI and VI converged to the same mathematically optimal policy as expected, with PI requiring more iterations but less computational cost than VI. Conversely, in Cartpole, VI converged in fewer iterations than PI but took longer in wall clock time, underscoring the computational overhead introduced by discretizing the continuous state space. These findings highlight how the environment's complexity affects the efficiency of these algorithms.

For the model-free Q-Learning approach, the results reveal its adaptability but also its reliance on careful tuning of hyperparameters and reward shaping. In both Blackjack and Cartpole, Q-Learning converged to a slightly different policy than PI and VI due to the influence of reward shaping, achieving faster computational times but sacrificing alignment with the optimal policy derived by PI and VI. Similarly, in both Blackjack and Cartpole, Q-Learning excelled in computational efficiency, converging in significantly less time than PI and VI while approximating a strong policy. This underscores the algorithm's suitability for high-dimensional or continuous state spaces.

Despite these problems having been extensively studied and refined over the years, they remain indispensable case studies for exploring the nuances of reinforcement learning. They Provide a foundational lens through which researchers can evaluate and innovate RL techniques beyond the scope of these specific environments.

References

[1] Farama Foundation, "Blackjack Environment," Gymnasium, [Online]. Available: https://gymnasium.farama.org/environments/toy_text/blackjack/. [Accessed: Nov. 30, 2024].

[2] Farama Foundation, "CartPole Environment," Gymnasium, [Online]. Available: https://gymnasium.farama.org/environments/classic_control/cart_pole/. [Accessed: Nov. 30, 2024].

[3] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997, pp. 367-387.

[4] A. Vikram and T. LaGrow, "Mastering Markov Decision Processes: A Practical RL Journey with OpenAI Gym," Georgia Tech OMSCS 7641, Apr. 9, 2024. [Online]. Available: https://sites.gatech.edu/omscs7641/2024/04/09/mastering-markov-decision-processes-a-practical-rl-journey-with-openai-gym/. [Accessed: Nov. 30, 2024].

[5] "ROBOT LEARNING, edited by Jonathan H. Connell and Sridhar Mahadevan, Kluwer, Boston, 1993/1997, xii+240pp., ISBN 0-7923-9365-1 (Hardback, 218.00 Guilders, $120.00, £89.95)," *Robotica*, vol. 17, no. 2, pp. 21–235, 1999, doi:10.1017/S0263574799271172.