Monte Carlo Methods: From Geometry to Reinforcement Learning

Dhruvin Doshi
Roll No: 24B0984

January 30, 2026

Abstract

This report presents my work during Weeks 1 to 4 of the WiDS 5.0 Monte Carlo project. The project progressed from building a complete Blackjack environment in Python, to applying Monte Carlo methods for numerical estimation, studying the theoretical foundations of Reinforcement Learning, and finally implementing Monte Carlo prediction and control to learn an effective Blackjack strategy purely through experience. The report discusses the motivation, methodology, observations, and conceptual insights gained at each stage of the project.

Contents

1 Introduction

Monte Carlo methods form a bridge between probability theory, numerical computation, and modern Reinforcement Learning. The WiDS 5.0 Monte Carlo project was structured to gradually build this connection by starting with simulation based estimation problems and culminating in a learning agent that discovers optimal behaviour through interaction with an environment.

Over the course of four weeks, I built a functional Blackjack game environment, explored Monte Carlo estimation and vectorization, studied the mathematical framework of Reinforcement Learning, and implemented Monte Carlo prediction and control algorithms. The final outcome was an agent that learned a near optimal Blackjack policy starting from zero knowledge of the game.

2 Week 1: Python Fundamentals and Blackjack Environment

2.1 Objective

The goal of Week 1 was to develop strong familiarity with Python, object oriented programming, and simulation design by building a complete Blackjack game engine from scratch. This environment was intended to be compatible with Reinforcement Learning algorithms introduced later.

2.2 Environment Design

The Blackjack game was designed using a modular and object oriented approach. Individual cards, the deck, and the overall game logic were separated conceptually to keep the design clean and easy to extend. The game logic handles card dealing, hit and stick actions, dealer behaviour, and final outcome evaluation.

A key design decision was to expose the game through a step based interface suitable for RL. Each interaction with the environment returns:

- the current state of the game
- a numerical reward
- a termination signal indicating whether the round has ended

2.3 State Representation

The game state was represented using a compact tuple consisting of:

- the player's current hand sum
- the dealer's visible card
- a boolean flag indicating the presence of a usable ace

This representation captures all information required under the infinite deck assumption and is commonly used in theoretical treatments of Blackjack in Reinforcement Learning.

2.4 Handling Aces and Edge Cases

Special care was taken to correctly handle aces, which can take values of either 1 or 11. A usable ace was defined as an ace that can be counted as 11 without causing the player to bust. Logic was implemented to downgrade a usable ace to value 1 if a subsequent hit would otherwise exceed 21.

Manual testing and a simple graphical interface were used to verify correct behaviour in edge cases such as multiple aces, near bust situations, and immediate terminal states.

2.5 Key Takeaways

Week 1 highlighted the importance of careful environment design. Building the game with RL in mind from the beginning made later algorithmic implementation significantly simpler and less error prone.

3 Week 2: Monte Carlo Estimation and Vectorization

3.1 Objective

Week 2 focused on understanding Monte Carlo methods as a general tool for numerical estimation, and on learning how computational efficiency can be dramatically improved through vectorization.

3.2 Monte Carlo Estimation of π

The value of π was estimated using random sampling in a unit square. Random points were drawn uniformly and the fraction that fell inside a quarter circle was used to estimate the area ratio.

As the number of samples increased, the estimate converged toward the true value of π. Repeating the experiment multiple times revealed the inherent randomness of Monte Carlo methods and the slow but steady convergence behaviour.

3.3 Estimating Euler's Number

Euler's number e was estimated using its integral representation. Random samples were drawn from a uniform distribution and the average value of the exponential function was used to recover e. Compared to the π experiment, this estimator exhibited lower variance due to the smoothness of the integrand.

3.4 Vectorization and Performance

A major focus of this week was rewriting loop based Monte Carlo simulations using vectorized NumPy operations. By generating all random samples at once and applying array operations, runtime was reduced dramatically.

Empirical timing comparisons showed speedups ranging from an order of magnitude to nearly two orders of magnitude for large sample sizes. This made it feasible to run experiments with millions of samples on a standard machine.

3.5 Monte Carlo Versus Deterministic Integration

Monte Carlo integration was compared with deterministic methods such as Riemann and trapezoidal sums. While deterministic methods converge faster in low dimensional problems, Monte Carlo methods maintain their convergence rate independent of dimensionality. This makes Monte Carlo particularly attractive for high dimensional problems where grid based approaches become computationally infeasible.

3.6 Key Insights

The defining characteristic of Monte Carlo methods is that estimation error decreases proportional to the inverse square root of the number of samples. Reducing error by half therefore requires approximately four times more samples, a fact that strongly motivates efficient implementations.

4 Week 3: Reinforcement Learning Foundations

4.1 Objective

Week 3 was dedicated to studying the theoretical foundations of Reinforcement Learning. The focus was on understanding how decision making problems can be formalised mathematically and how optimal behaviour can be derived.

4.2 Markov Decision Processes

Reinforcement Learning problems are formalised as Markov Decision Processes. An MDP consists of states, actions, transition probabilities, rewards, and a discount factor. The defining property is that the future depends only on the current state and action, not on the full history.

Blackjack under the infinite deck assumption satisfies this property when the state is defined appropriately.

4.3 Value Functions

Two central concepts studied were the state value function and the action value function. These functions represent the expected cumulative reward obtained by following a policy from a given state or state action pair.

Value functions provide a quantitative measure of how desirable a situation is and form the basis for computing optimal strategies.

4.4 Bellman Equations

The Bellman equations express value functions recursively in terms of immediate rewards and the values of successor states. These equations form the theoretical backbone of all Reinforcement Learning algorithms.

Understanding the Bellman optimality equation clarified how optimal policies can be derived by selecting actions that maximise expected future reward.

4.5 Monte Carlo Methods in Reinforcement Learning

Monte Carlo methods estimate value functions by averaging observed returns from sampled episodes. Unlike dynamic programming, they do not require knowledge of transition probabilities and can learn directly from experience. This makes them especially suitable for environments where the model is unknown.

5 Week 4: Monte Carlo Prediction and Control

5.1 Objective

The goal of Week 4 was to apply Monte Carlo methods to a Reinforcement Learning problem by training an agent to play Blackjack optimally through self play.

5.2 Monte Carlo Prediction

A fixed policy was first evaluated using Monte Carlo prediction. The agent followed a simple rule based strategy and complete episodes were generated. For each state encountered, the final return was recorded and averaged using first visit Monte Carlo estimation.

The resulting value estimates aligned with intuition. States with player sums close to 21 showed positive expected value, while low sums were associated with negative outcomes.

5.3 Monte Carlo Control

Monte Carlo control was then used to learn an optimal policy without prior knowledge of the game dynamics. Action value estimates were maintained for each state action pair and updated using observed episode returns.

An epsilon greedy strategy was used to balance exploration and exploitation. Early in training, random actions allowed the agent to explore the state space. Over time, the agent increasingly favoured actions with higher estimated value.

5.4 Training Behaviour and Convergence

Training was carried out over several hundred thousand episodes. Initial performance was poor due to random exploration, but steadily improved as experience accumulated. A rolling average of episode rewards was used to track learning progress and reduce the effect of high variance.

Eventually, the learning curve stabilised, indicating convergence toward a near optimal policy.

5.5 Learned Strategy

The learned policy closely matched the well known Blackjack basic strategy. The agent learned to stick on high player sums, hit on low sums, and make conditional decisions based on the dealer's visible card. Different behaviour emerged naturally for hands with a usable ace.

Importantly, this strategy was not programmed explicitly. It emerged solely from repeated trial and error and reward feedback.

6 Conceptual Discussion

6.1 Infinite Deck Assumption

In a real casino, Blackjack is played with a finite shoe, making card counting possible. In this setting, the probability of future draws depends on previously seen cards, violating the Markov property if the state does not include this information.

To model card counting, the state would need to be expanded to include a summary of deck composition. While this would restore the Markov property, it would also dramatically increase the size of the state space and slow learning.

6.2 First Visit and Every Visit Monte Carlo

First visit Monte Carlo updates value estimates using only the first occurrence of a state within an episode, while every visit Monte Carlo updates on all occurrences. Both approaches converge to the true value given sufficient data, but differ in variance and update frequency.

7 Conclusion

This project demonstrated how Monte Carlo methods connect numerical simulation and Reinforcement Learning. Starting from simple geometric estimation problems, I progressed to building an agent that learns complex decision making behaviour from experience alone.

The project highlighted key tradeoffs in Reinforcement Learning, including variance, sample efficiency, and state space design. Future extensions include experimenting with decaying exploration schedules, alternative learning rates, and modelling finite deck Blackjack using function approximation.