
**Addis Ababa University**

**Department of Artificial Intelligence**

# Report on Lab 4: Monte Carlo Control with Linear Function Approximation vs Deep Q-Learning

**Submitted by:**

Tsion Bizuayehu

GSR/9235/17

Submission Date: February 5, 2026

Submitted to: Dr. Natnael and Azmeraw

This laboratory exercise investigated and compared the performance of Monte Carlo (MC) control with linear function approximation and Deep Q-Learning (DQL) in a simple one-dimensional continuous-state navigation environment. The MC agent employed polynomial basis functions to approximate action-value functions, while the DQL agent relied on a neural network, experience replay, and a target network. Experimental results demonstrated that the MC agent achieved faster convergence, greater stability, and superior final performance in this environment. In contrast, the DQL agent exhibited high variance, poor convergence, and sensitivity to hyperparameters. The lab highlights the importance of matching algorithmic complexity to problem structure and illustrates the strengths and limitations of linear and deep function approximation methods in reinforcement learning.

# 1. Introduction

Reinforcement Learning (RL) focuses on enabling agents to learn optimal behavior through interaction with an environment, guided by reward signals. When state spaces are continuous, explicit tabular methods become infeasible, motivating the use of function approximation. This lab compares two such approaches: Monte Carlo control with linear function approximation and Deep Q-Learning (DQL).

Monte Carlo methods estimate action values from complete episode returns and are known for conceptual simplicity and stability in episodic tasks. Linear function approximation allows MC methods to generalize across continuous state spaces with relatively low computational cost. Deep Q-Learning, on the other hand, replaces linear approximators with deep neural networks, enabling representation of highly non-linear value functions at the cost of increased computational and tuning complexity.

The objective of this lab was to empirically evaluate both approaches in the same controlled environment and analyze their learning dynamics, convergence properties, and practical limitations.

# 2. Environment Description

The environment is a custom one-dimensional navigation task with the following characteristics:

- **State Space**: A continuous scalar representing the agent's position on a line.
- **Action Space**: Discrete actions {move left, move right}.
- **Goal**: Reach a predefined goal_position.

- **Reward Structure**: Positive reward for reaching the goal, with penalties or zero reward otherwise.
- **Termination**: An episode ends when the agent reaches the goal or exceeds a maximum number of steps.

This environment is intentionally simple, making it well-suited for evaluating the efficiency and appropriateness of different function approximation strategies.

# 3. Monte Carlo Control with Linear Function Approximation

### 3.1 Methodology

The Monte Carlo agent used an on-policy control approach with ε-greedy exploration. Since the state space is continuous, a linear function approximator was used to estimate Q-values instead of a tabular representation.

Each state-action pair was mapped to a feature vector using polynomial basis functions of degree three. This transformation enabled the agent to capture moderate non-linearities while preserving the computational simplicity of linear models. The Q-function was defined as:

$Q(s, a; w) = w^T \phi(s, a)$

where $\phi(s, a)$ is the feature vector and w is the weight vector learned via gradient descent using Monte Carlo returns.

### 3.2 Results and Observations

**Episode Rewards**: The agent demonstrated rapid learning, with episode rewards converging to the maximum value of 10.0 after an initial exploration phase. This indicates successful policy optimization.

**Episode Lengths**: Episode lengths decreased sharply over training, showing that the agent learned increasingly efficient paths to the goal.

**Learned Q-Values and Value Function**: The learned Q-functions for left and right actions converged smoothly. The derived value function V(s) exhibited a distinct M- or W-shaped profile, peaking near the goal position. This reflects higher expected returns for states closer to the goal.

**Learned Policy**: The policy visualization showed clear directional behavior: the agent moved right when left of the goal and left when right of the goal. Minor inconsistencies near the goal are expected, as the episode terminates upon goal attainment.

Overall, the Monte Carlo agent with linear function approximation learned an optimal and interpretable policy efficiently and stably.

# 4. Deep Q-Learning Implementation

**4.1 Algorithm Overview**

**4.1 Algorithm Overview**

Deep Q-Learning extends classical Q-learning by approximating the Q-function with a neural network. The DQL agent implemented in this lab consisted of the following components:

- **Q-Network**: A feedforward neural network mapping states to Q-values for each action.
- **Experience Replay Buffer**: A memory buffer storing past transitions (s, a, r, s', done), from which mini-batches were randomly sampled.
- **Target Network**: A delayed copy of the Q-network used to compute stable Bellman targets.

These mechanisms are designed to reduce instability caused by correlated updates and non-stationary targets.

**4.2 Training Behavior and Results**

**Episode Rewards**: The DQL agent exhibited highly variable rewards with no clear convergence. Rewards often remained negative or near zero, with only occasional positive spikes.

**Episode Lengths**: Most episodes reached the maximum step limit (200), indicating frequent failure to reach the goal.

**Loss Curve**: The training loss decreased over time, suggesting that the network was fitting its targets. However, this reduction in loss did not translate into improved policy performance.

These results indicate that, despite learning in a supervised sense, the DQL agent struggled to discover a consistently successful control policy in this environment.

# 5. Limitations of Linear Approximation

**5.1 Visualization Insight**

The linear approximation visualization provides an important conceptual lesson. From the plotted graph, the following aspects should be emphasized:

- **True Function Shape**: The cubic function exhibits multiple regions of curvature and inflection points.
- **Linear Fit Behavior**: The straight-line approximation captures only a coarse global trend and ignores local structure.
- **Error Regions**: The largest approximation errors occur near the extremes of the input range and around inflection points.

This visualization should be interpreted as evidence that linear models fundamentally lack the expressive power needed to represent complex non-linear relationships, even when noise is present.

This limitation directly motivates the use of neural networks in deep reinforcement learning when environments exhibit highly non-linear dynamics.
To illustrate the representational limits of linear models, a separate experiment attempted to approximate a cubic non-linear function using linear regression.

The resulting fit captured only the global trend of the data and failed to model local curvature, inflection points, and extreme regions accurately. This visualization reinforces the theoretical limitation that linear models cannot represent complex non-linear relationships, motivating the use of deep networks in more complex RL tasks.

# 6. Deep Q-Learning Challenges

### 6.1 Learning Rate Sensitivity

Conceptual reward curves demonstrated the impact of different learning rates:

- Low learning rates resulted in very slow convergence.
- High learning rates caused unstable and oscillatory learning.
- An intermediate learning rate provided the best balance between speed and stability.

### 6.2 Hyperparameter Sensitivity

DQL performance was highly sensitive to parameters such as learning rate, discount factor, exploration schedule, batch size, replay buffer size, and target update frequency. Identifying suitable values often requires extensive experimentation.

### 6.3 Computational Constraints

Compared to linear MC methods, DQL requires significantly more computational resources:

- Neural network training involves costly gradient computations.
- Experience replay consumes substantial memory.
- Training time increases dramatically with network size and episode count.

These constraints make DQL less practical for small or simple problems.

# 7. Comparative Analysis

### 7.1 Quantitative Results Summary

To facilitate a clearer comparison between Monte Carlo (MC) control with linear function approximation and Deep Q-Learning (DQL), the key experimental results are summarized in Table 1.

**Table 1: Performance Comparison Between MC and DQL Agents**

| Metric | Monte Carlo (Linear Approx.) | Deep Q-Learning |
|---|---|---|
| **Final Average Episode Reward** | ~10.0 (Optimal) | ~0 to Negative |
| **Reward Convergence** | Fast and Stable | No Clear Convergence |
| **Reward Variance** | Low after convergence | High throughout training |
| **Average Episode Length (Final)** | Very Low (few steps) | Near Maximum (200 steps) |
| **Goal Reached Consistently** | Yes | Rarely |
| **Training Stability** | High | Low |
| **Computational Cost** | Low | High |

These results clearly demonstrate that the MC agent outperformed the DQL agent across all practical performance metrics in this environment.

### 7.2 Interpretation of Learning Curves

### 7.3 Interpretation of Learning Curves

When comparing the learning curves of both agents, several important visual patterns emerge:

- **Monte Carlo Rewards Curve**: A rapid increase in rewards followed by a flat plateau at the maximum value indicates successful convergence to an optimal policy.
- **Deep Q-Learning Rewards Curve**: Large oscillations without a stable upward trend indicate unstable learning and failure to converge.

The stark contrast between these curves visually reinforces the numerical results shown in Table 1.
When plotted together, the contrast between the two methods is clear:

- **Monte Carlo**: Fast, stable convergence to optimal rewards.
- **Deep Q-Learning**: High variance, poor convergence, and inconsistent performance.

### 7.2 Convergence and Stability

The Monte Carlo agent converged reliably with smooth learning dynamics. The DQL agent showed oscillatory behavior and failed to stabilize within the same training horizon.

### 7.3 Final Performance

The MC agent consistently solved the task, while the DQL agent frequently failed. This demonstrates that for simple continuous-state, discrete-action problems, linear approximation can outperform deep methods when appropriately designed.

## 8. Key Takeaways and Future Work

### 8.1 Main Insights

- Monte Carlo control with linear function approximation is efficient, stable, and effective for simple environments.
- Deep Q-Learning is powerful but can be unnecessarily complex and unstable for low-dimensional problems.
- Linear models are limited in representational capacity, but their simplicity can be a major advantage.
- Hyperparameter tuning is critical, particularly for deep RL methods.

### 8.2 Potential Improvements

**Monte Carlo Agent**:

- Explore richer feature representations such as Fourier bases, radial basis functions, or tile coding.
- Evaluate performance in moderately more complex environments.

**Deep Q-Learning Agent**:

- Perform systematic hyperparameter tuning.

- Experiment with deeper or wider networks.
- Implement advanced variants such as Double DQN, Dueling DQN, or Prioritized Experience Replay.
- Test both methods on higher-dimensional benchmarks such as CartPole or LunarLander.

# 9. Conclusion

This lab demonstrated that algorithmic sophistication does not guarantee superior performance. In a simple 1D navigation task, Monte Carlo control with linear function approximation significantly outperformed Deep Q-Learning in terms of convergence speed, stability, and final performance. The results emphasize the importance of aligning method complexity with problem structure and highlight key practical considerations in reinforcement learning algorithm selection.

**References**

[1] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.