CartPole-v1: Reinforcement Learning with Linear Function Approximation

A Data Mining Perspective on Control Problems

Omid Mohebi

RL: Learning by Interaction

Reinforcement Learning is a computational approach where an **Agent** learns to make decisions by performing **Actions** in an **Environment** to maximize a cumulative **Reward**.

The agent doesn't learn from labeled examples like in supervised learning. Instead, it learns through trial and error, receiving feedback (rewards or penalties) for its actions.

Key Components (The MDP Framework)

Agent

The learner and decision-maker (our Q-learning model)

Environment

The world the agent interacts with (CartPole-v1)

State (S)

A description of the current situation (position, velocity, angle, angular velocity)

Action (A)

The choices the agent can make (push cart left or right)

Reward (R)

Feedback from the environment (+1 for each step pole is upright)

Policy (π)

A mapping from states to actions that the agent learns to maximize total reward

The CartPole-v1 Environment

Task: Balance a pole on a cart by moving the cart left or right. The agent must keep the pole upright while preventing the cart from moving too far from the center.

State Space

Continuous (4 values)

- Cart position
- Cart velocity
- Pole angle
- Angular velocity

Action Space

Discrete (2 actions)

- Action 0: Push cart left
- Action 1: Push cart right

Reward

+1 for every timestep the pole remains balanced

Termination Condition

Episode ends if pole falls past **15°** or cart moves more than **2.4 units** from center

"Solved" Criteria

Average reward of **≥ 475** over 100 consecutive episodes

Maximum Episode Length

500 timesteps per episode

Learning the Value of Actions

The Q-Function

Q(S, A) represents the maximum expected future reward the agent can obtain by starting in state S and taking action A, then following the optimal policy thereafter.
It answers the question: "How good is it to take this action in this state?"

Key Insight

If we know the Q-values for all state-action pairs, we can determine the optimal policy by always choosing the action with the highest Q-value in each state.

Bellman Optimality Equation

$$Q(S,A) = R + \gamma \max_{A'} Q(S',A')$$

R: Immediate reward received

γ (gamma): Discount factor (0.99) — how much we value future rewards

$\max_{A'} Q(S', A')$: Best possible Q-value in the next state

Q-Learning Update Rule

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_{A'} Q(S',A') - Q(S,A)]$$

α (alpha): Learning rate (0.1) — step size for updates

$[R + \gamma \max_{A'} Q(S', A') - Q(S, A)]$: Temporal Difference (TD) error

This is **Temporal Difference Learning** — we update Q-values based on the difference between predicted and actual rewards

How to Choose the Next Action?

Exploitation

Choose the action with the highest known Q-value—the best action based on what we've learned so far.

Exploration

Choose a random action to discover potentially better, unknown rewards that we haven't tried yet.

The Dilemma

Too much exploitation → Agent gets stuck in local optima and misses better solutions.

Too much exploration → Agent wastes time trying random actions instead of using what it learned.

The ε-Greedy Policy

With probability (1 − ε):

Exploit

With probability ε:

Explore

This simple strategy balances exploration and exploitation. By choosing ε carefully, we ensure the agent tries enough new actions to find good solutions while still exploiting what it has learned.

Project Hyperparameter:

ε = 0.1 (10% exploration)

Fixed exploration rate throughout training

The Data Mining Connection: Function Approximation

The Problem

CartPole's state space is **continuous**. We cannot store a Q-value for every possible state-action pair in a table.

Infinite States

Position, velocity, angle, and angular velocity can take infinitely many values

Memory Impossible

A lookup table cannot store values for all possible continuous states

Generalization Needed

The agent must learn to generalize from seen states to unseen states

The Solution

Use a **function** to estimate Q-values instead of a table.

Function Approximation

$$Q(S, A) \approx f(x(S, A), w)$$

where x(S, A) is a feature vector and w are learnable weights

Linear Model

$$Q(S, A) = w^T \cdot x(S, A)$$

Simple, efficient, and connects to data mining principles

This bridges **Reinforcement Learning** with **Data Mining** concepts like feature engineering and linear regression.

Feature Engineering with RBF Kernels

The Need for Non-Linearity

A simple linear model on raw state features is often insufficient for complex control problems. The CartPole task has non-linear dynamics that require more expressive features.

Radial Basis Function (RBF) Kernels

RBF kernels are a powerful technique from **kernel methods** that map the input space into a high-dimensional feature space where the problem becomes **linearly separable**.

This is a core **data mining** technique used in Support Vector Machines (SVMs) and other kernel-based methods.

> **Key Insight:** By transforming the input space non-linearly, we can solve complex problems using simple linear models.

Random Fourier Features (RFF)

A technique to **approximate the RBF kernel feature map efficiently** without explicitly computing the kernel matrix.

- Transforms continuous state-action vectors into fixed-length, non-linear feature vectors
- Computationally efficient for large-scale problems
- Enables linear models to capture non-linear relationships

Implementation in Our Project

We use **sklearn.kernel_approximation.RBFSampler** to:

- Fit the featurizer on a large set of random state-action samples (10,000 episodes)
- Transform each state-action pair into a high-dimensional feature vector
- Learn a linear weight vector on these transformed features

> **Result:** A non-linear Q-function that is linear in the RBF feature space, combining the expressiveness of non-linear models with the efficiency of linear learning.

Project Implementation: Linear Q-Learning

Q-Function

$$Q(S, A) = w^T \cdot x(S, A)$$

- **x(S, A)** = RBF feature vector
- **w** = weight vector (learned)

The Update Rule

$$\delta = [R + \gamma \cdot \max Q(S', A') - Q(S, A)]$$

$$w \leftarrow w + \alpha \cdot \delta \cdot x(S, A)$$

- **δ** = temporal difference error
- **α** = learning rate (0.1)
- **γ** = discount factor (0.99)

The weight update is a form of **Stochastic Gradient Descent (SGD)**, a core data mining technique. The error signal δ guides the weights toward better predictions.

Error Computation: The difference between the target Q-value (what we think is correct) and the current prediction.

Gradient Direction: The feature vector x(S, A) indicates which features contributed to the error.

Weight Adjustment: We move weights in the direction of the gradient, scaled by the learning rate α.

Key Insight

By using **RBF features**, we transform a non-linear control problem into one solvable by a simple **linear model** and **gradient descent**—demonstrating how feature engineering bridges complex problems and efficient algorithms.

Hyperparameters Summary

Parameter	Description	Value	Role in Learning
γ (GAMMA)	Discount Factor	**0.99**	Controls the agent's foresight. Near 1.0 means long-term rewards are highly valued; near 0 means only immediate rewards matter. 0.99 balances immediate and future rewards.
α (ALPHA)	Learning Rate	**0.1**	Determines the step size for weight updates. Too high causes instability; too low causes slow learning. 0.1 is a standard choice for gradient descent.
ε (EPSILON)	Exploration Probability	**0.1**	Fixed probability of taking a random action. 0.1 means 10% exploration, 90% exploitation. Balances discovering new strategies with using learned knowledge.
RBF Features	Feature Vector Dimension	**100**	Number of random Fourier features. Higher dimensions capture more complex patterns but increase computation. Default is typically 100.
Episodes	Maximum Training Episodes	**2000**	Total number of learning trials. The agent stops early if it solves the environment (avg reward ≥ 475 over 100 episodes).

Hyperparameter Tuning: These values were chosen based on common practices in reinforcement learning. In practice, you would experiment with different values to find the best performance for your specific problem. The learning rate α and discount factor γ are particularly sensitive to the problem structure.

Convergence and Performance

Training Goal

Achieve an average reward of **≥ 475** over 100 consecutive episodes to declare the environment "solved."

Typical Outcome

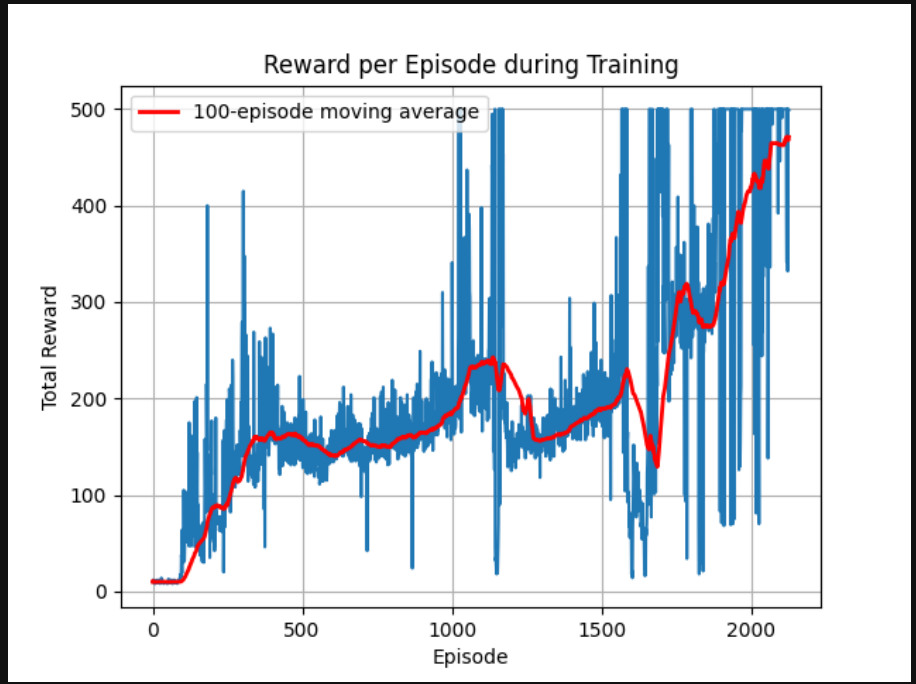
~1,000–2,000 episodes

Final Performance

≥ 475 avg reward

Test Performance

~490–500 reward



The **red moving average line** in the plot clearly demonstrates the learning progress. Early episodes show high variance and low rewards (random exploration). As training progresses, the moving average steadily increases, indicating the agent is learning an effective policy.

Conclusion & Future Work

Summary

We successfully implemented a **Q-learning agent** for the CartPole-v1 continuous control problem by leveraging **Random Fourier Features** to transform the problem into a tractable **Linear Function Approximation** task.

The agent learns an optimal policy through interaction with the environment, balancing exploration and exploitation using the ε-greedy strategy, and converges to solve the task within 1,000–2,000 episodes.

Key Takeaway for Data Mining

Reinforcement Learning, at its core, relies heavily on **Feature Engineering** and **Stochastic Optimization** (like SGD)—making it highly relevant to data mining principles. By transforming non-linear problems into linear ones through kernel methods, we demonstrate how classical data mining techniques enable modern AI systems.

Future Work & Improvements

Epsilon Decay Schedule

Start with high exploration (ε = 1.0) and gradually decay to low exploration (ε = 0.01) as the agent learns, improving the exploration-exploitation balance.

Deep Q-Learning (DQN)

Replace the linear model with a neural network to learn features automatically, eliminating the need for manual RBF feature engineering.

RBF Hyperparameter Tuning

Experiment with RBFSampler parameters (gamma, n_components) to find the optimal feature representation for CartPole.

Advanced RL Algorithms

Implement Actor-Critic methods, Policy Gradient algorithms, or other state-of-the-art RL techniques for comparison.

Thank You

Questions?

Let's discuss reinforcement learning, Q-learning, RBF kernels, and any aspects of the CartPole-v1 project.

Omid Mohebi • o.mohebi@stu.usc.ac.ir