Experimenting Q-Learning with Function Approximation

Syed Mohammed Umar Farooq

Department of Electrical Engineering
Indian Institute of Technology, Madras

April 28, 2025

Overview

1. Markov Decision Process
2. Dynamic Programming
3. Q-Learning
4. Q-Function Approximation
5. Experiments & Results

Markov Decision Process(MDP)

- A framework for modeling decision-making in environments with stochastic outcomes.
- Provides a mathematical structure for reinforcement learning problems.
- Used in robotics, game theory, economics, and other areas requiring sequential decision-making.

MDP Terminology

- **Agent**: The entity which we are training to make correct decisions.
- **Environment**: The surroundings with which the agent interacts.
- **State** (S_t): Defines the current situation of the agent.
- **Action** (A_t): The choice that the agent makes at the current time step.
- **Reward** (R_t): The feedback signal received after an action, indicating the immediate gain or loss.
- **Policy** (π): A strategy or thought process behind selecting an action, mapping states to actions.
- **Discount Factor** (γ): A value in $[0, 1]$ that determines the importance of future rewards. A smaller γ prioritizes immediate rewards, while a larger γ values long-term rewards.

MDP Terminology

- **Agent**: The entity which we are training to make correct decisions.
- **Environment**: The surroundings with which the agent interacts.
- **State** (S_t): Defines the current situation of the agent.
- **Action** (A_t): The choice that the agent makes at the current time step.
- **Reward** (R_t): The feedback signal received after an action, indicating the immediate gain or loss.
- **Policy** (π): A strategy or thought process behind selecting an action, mapping states to actions.
- **Discount Factor** (γ): A value in $[0, 1]$ that determines the importance of future rewards. A smaller γ prioritizes immediate rewards, while a larger γ values long-term rewards.

MDP Terminology

- **Agent**: The entity which we are training to make correct decisions.
- **Environment**: The surroundings with which the agent interacts.
- **State** (S_t): Defines the current situation of the agent.
- **Action** (A_t): The choice that the agent makes at the current time step.
- **Reward** (R_t): The feedback signal received after an action, indicating the immediate gain or loss.
- **Policy** (π): A strategy or thought process behind selecting an action, mapping states to actions.
- **Discount Factor** (γ): A value in $[0, 1]$ that determines the importance of future rewards. A smaller γ prioritizes immediate rewards, while a larger γ values long-term rewards.

MDP Terminology

- **Agent**: The entity which we are training to make correct decisions.
- **Environment**: The surroundings with which the agent interacts.
- **State** (S_t): Defines the current situation of the agent.
- **Action** (A_t): The choice that the agent makes at the current time step.
- **Reward** (R_t): The feedback signal received after an action, indicating the immediate gain or loss.
- **Policy** (π): A strategy or thought process behind selecting an action, mapping states to actions.
- **Discount Factor** (γ): A value in $[0, 1]$ that determines the importance of future rewards. A smaller γ prioritizes immediate rewards, while a larger γ values long-term rewards.

MDP Terminology

- **Agent**: The entity which we are training to make correct decisions.
- **Environment**: The surroundings with which the agent interacts.
- **State** (S_t): Defines the current situation of the agent.
- **Action** (A_t): The choice that the agent makes at the current time step.
- **Reward** (R_t): The feedback signal received after an action, indicating the immediate gain or loss.
- **Policy** (π): A strategy or thought process behind selecting an action, mapping states to actions.
- **Discount Factor** (γ): A value in $[0, 1]$ that determines the importance of future rewards. A smaller γ prioritizes immediate rewards, while a larger γ values long-term rewards.

MDP Terminology

- **Agent**: The entity which we are training to make correct decisions.
- **Environment**: The surroundings with which the agent interacts.
- **State** (S_t): Defines the current situation of the agent.
- **Action** (A_t): The choice that the agent makes at the current time step.
- **Reward** (R_t): The feedback signal received after an action, indicating the immediate gain or loss.
- **Policy** (π): A strategy or thought process behind selecting an action, mapping states to actions.
- **Discount Factor** (γ): A value in $[0, 1]$ that determines the importance of future rewards. A smaller γ prioritizes immediate rewards, while a larger γ values long-term rewards.

MDP Terminology

- **Agent**: The entity which we are training to make correct decisions.
- **Environment**: The surroundings with which the agent interacts.
- **State** (S_t): Defines the current situation of the agent.
- **Action** (A_t): The choice that the agent makes at the current time step.
- **Reward** (R_t): The feedback signal received after an action, indicating the immediate gain or loss.
- **Policy** (π): A strategy or thought process behind selecting an action, mapping states to actions.
- **Discount Factor** (γ): A value in $[0, 1]$ that determines the importance of future rewards. A smaller γ prioritizes immediate rewards, while a larger γ values long-term rewards.

MDP Terminology

- **Agent**: The entity which we are training to make correct decisions.
- **Environment**: The surroundings with which the agent interacts.
- **State** (S_t): Defines the current situation of the agent.
- **Action** (A_t): The choice that the agent makes at the current time step.
- **Reward** (R_t): The feedback signal received after an action, indicating the immediate gain or loss.
- **Policy** (π): A strategy or thought process behind selecting an action, mapping states to actions.
- **Discount Factor** (γ): A value in $[0, 1]$ that determines the importance of future rewards. A smaller γ prioritizes immediate rewards, while a larger γ values long-term rewards.

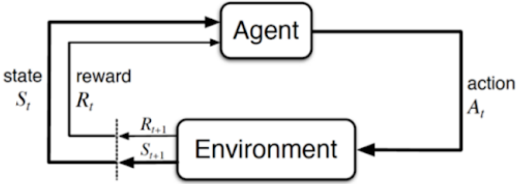

Figure: A typical interaction in a Markov Decision Process (MDP).

- The Return(G_t) is the total accumulated reward from time step t onward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Markov Decision Process

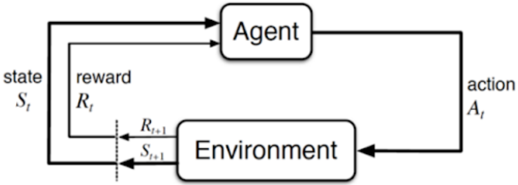

Figure: A typical interaction in a Markov Decision Process (MDP).

- The Return(G_t) is the total accumulated reward from time step t onward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Value Functions

1. **State Value Function ($V^\pi(s)$):**
   - Expected return when starting in state s and following policy π.
   - $$V_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$
2. **Action Value Function ($Q^\pi(s, a)$):**
   - Expected return when starting in state s, taking action a, and then following policy π.
   - $$Q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

These functions form the foundation for most reinforcement learning algorithms.

Bellman Expectation Equations

- The Bellman Expectation Equation for the State Value Function $V_\pi(s)$ is:

$$V_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s\right]$$

- The Bellman Expectation Equation for the Action Value Function $Q_\pi(s, a)$ is:

$$Q_\pi(s, a) = \mathbb{E}_\pi\left[R_{t+1} + \gamma \mathbb{E}_\pi\left[V_\pi(S_{t+1}) \mid S_{t+1}\right] \mid S_t = s, A_t = a\right]$$

- Bellman Optimality Condition is

$$V_*(s) = \max_a q_*(s, a)$$

Bellman Expectation Equations

- The Bellman Expectation Equation for the State Value Function $V_\pi(s)$ is:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s \right]$$

- The Bellman Expectation Equation for the Action Value Function $Q_\pi(s, a)$ is:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma \mathbb{E}_\pi \left[V_\pi(S_{t+1}) \mid S_{t+1} \right] \mid S_t = s, A_t = a \right]$$

- Bellman Optimality Condition is

$$V_*(s) = \max_a q_*(s, a)$$

Bellman Expectation Equations

- The Bellman Expectation Equation for the State Value Function $V_\pi(s)$ is:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s \right]$$

- The Bellman Expectation Equation for the Action Value Function $Q_\pi(s, a)$ is:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma \mathbb{E}_\pi \left[V_\pi(S_{t+1}) \mid S_{t+1} \right] \mid S_t = s, A_t = a \right]$$

- Bellman Optimality Condition is

$$V_*(s) = \max_a q_*(s, a)$$

Bellman Expectation Equations

- The Bellman Expectation Equation for the State Value Function $V_\pi(s)$ is:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s\right]$$

- The Bellman Expectation Equation for the Action Value Function $Q_\pi(s, a)$ is:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma \mathbb{E}_\pi \left[V_\pi(S_{t+1}) \mid S_{t+1}\right] \mid S_t = s, A_t = a\right]$$

- Bellman Optimality Condition is

$$V_*(s) = \max_a q_*(s, a)$$

Bellman Expectation Equations

- The Bellman Expectation Equation for the State Value Function $V_\pi(s)$ is:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s \right]$$

- The Bellman Expectation Equation for the Action Value Function $Q_\pi(s, a)$ is:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma \mathbb{E}_\pi \left[V_\pi(S_{t+1}) \mid S_{t+1} \right] \mid S_t = s, A_t = a \right]$$

- Bellman Optimality Condition is

$$V_*(s) = \max_a q_*(s, a)$$

Dynamic Programming (DP) is a method used to compute optimal policies for Markov Decision Processes when the model (transition probabilities and rewards) is known.

Key Concepts:

- Uses value functions to evaluate and improve policies.
- Relies on the Bellman equations for recursive updates.
- Iteratively updates value estimates until convergence.

Common DP Algorithms:

- **Policy Evaluation:** Computes $V^{\pi}(s)$ for a fixed policy π.
- **Policy Improvement:** Improves the current policy using $V^{\pi}(s)$.
- **Policy Iteration:** Alternates between evaluation and improvement.
- **Value Iteration:** Repeatedly applies Bellman optimality update.

Q-Learning

Q-Learning is a model-free reinforcement learning algorithm used to learn the optimal action-value function $Q^*(s, a)$.

- Maintains a Q-table with estimates of the expected return for each state-action pair.
- Initializes all $Q(s, a)$ values arbitrarily (often to zero or random values).
- Updates the Q-values using the following rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

Where:

- α is the learning rate,
- s' is the next state.

Why Approximation?

- Traditional Q-learning stores Q-values in a table.
- For large or continuous state-action spaces, visiting every (s, a) pair becomes infeasible.

Solution: Function Approximation

- Approximate the Q-function using a parameterized function: $\hat{Q}(s, a; \mathbf{w})$
- $\mathbf{w}$ represents the weights or parameters of the function (e.g., linear weights or neural network parameters).
- Instead of learning all Q-values explicitly, we learn the parameters $\mathbf{w}$ such that $\hat{Q}(s, a; \mathbf{w}) \approx Q^*(s, a)$.
- The update rule is:
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[R + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}) - \hat{Q}(s, a; \mathbf{w}) \right] \nabla_{\mathbf{w}} \hat{Q}(s, a; \mathbf{w})$$

Linear Function Approximation:

- The Q-function is approximated as:

$$\hat{Q}(s, a; \mathbf{w}) = \mathbf{w}^T \phi(s, a)$$

where $\phi(s, a)$ is the feature vector representing the state-action pair.

- The gradient with respect to the weights is:

$$\nabla_{\mathbf{w}} \hat{Q}(s, a; \mathbf{w}) = \phi(s, a)$$

Update Rule:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \underbrace{\left[R + \gamma \max_{a'} \hat{Q}(s', a'; \mathbf{w}) - \hat{Q}(s, a; \mathbf{w}) \right]}_{\text{TD Error}} \cdot \phi(s, a)$$

Experiments

The environment is modelled as:

- **State Space:** $\mathcal{S} = \{0, 1, \ldots, p-1\}$, where p is a natural number.
- **Action Space:** $\mathcal{A} = \{0, 1, \ldots, p-1\}$.
- **Noise Space:** A uniformly distributed space $\mathcal{N} = \{0, 1, \ldots, p-1\}$ with $\mathbb{P}(n = i) = \frac{1}{p}$ for all i.(We can use any probability distribution)
- **Transition Dynamincs**: $s_{t+1} = (s_t + a_t + n_t) \bmod p$.

Reward Model: $R(s_t, a_t) = s_t.a_t$

All experiments were conducted using $\gamma = 0.9$ & $p = 100$, meaning that each of the state, action, and noise spaces contained 100 elements.

Q-learning with constant learning rate

- The update rule for Q-learning:

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

- Here $\alpha = 0.001$.
- **Convergence:** Achieved after 10^8 iterations.
- **Time Taken:** 2137 seconds.
- **MSE** is around 3.6×10^7 & **RMS** is around 6000.
- **Accuracy:** Many Q-values were significantly different from the true values obtained via Dynamic Programming (DP).

- **Mean Squared Error** is calculated from Q values obtained from Q-learning algorithm and optimal Q* values from Dynamic Programming.
- The MSE graph for this case is:

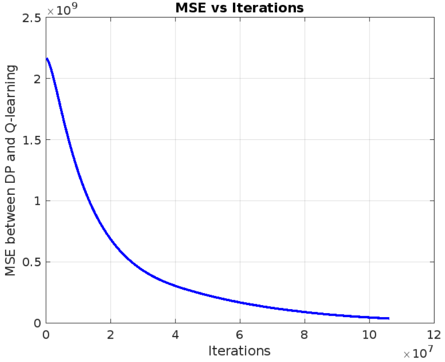

Figure: For constant Learning rate $\alpha = 0.001$

Q-learning with Robbins-Monro Algorithm

- Condition for Robbins-Monro Algorithm:

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty,$$

- For our case:

$$\alpha(s,a) = \frac{1}{1 + N(s,a)}$$

  where $N(s,a)$ counts how often a state-action pair has been visited.

- **Convergence:** Achieved after 5×10^7 iterations.
- **Time Taken:** 696 seconds.
- **MSE** is around 39611 & **RMS** is around 199
- **Accuracy:** Most Q-values are near to those computed by DP.

Q-learning with Robbins-Monro Algorithm

- The MSE graph for this case is:

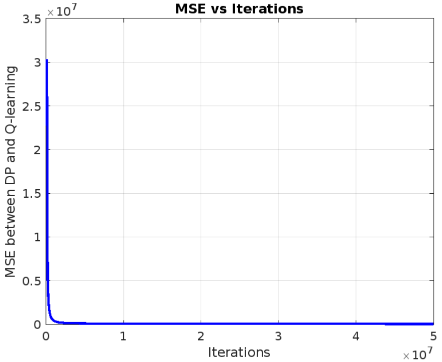

Figure: MSE for Robbins-Monro learning rate

Q-Learning with Linear Function Approximation

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

- **Feature Vector Construction:** Considering polynomial of degree 10 as function approximator. The feature vector be

$$\phi(s, a) = [1,\ \tilde{s},\ \tilde{a},\ \tilde{s}^2,\ \tilde{s}\tilde{a},\ \tilde{a}^2,\ \ldots,\ \tilde{s}^{10},\ \tilde{s}^9\tilde{a},\ \ldots,\ \tilde{a}^{10}]^T$$

  where $\tilde{s} = \frac{s}{p-1}, \quad \tilde{a} = \frac{a}{p-1}$.

- For k degree polynomial function approximator, the number of features will be $\frac{(k+1)(k+2)}{2}$.
- For our case the number of features are 66. So, the weight vector $\mathbf{w} \in \mathbb{R}^{66\times 1}$
- By learning 66 parameters we can find Q-table of 100×100.

Function Approximation with const learning rate

- The update rule:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\big(R + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}) - \hat{Q}(s, a; \boldsymbol{w})\big)\phi(s, a)$$

- $\alpha = 0.001$
- **Convergence:** Not achieved.
- **Time Taken:** For 5×10^7 1115 seconds.
- **MSE** is around 3.5×10^7 & **RMS** is around 6000

Function Approximation with const learning rate

- MSE between optimal values(calculated from DP) and Q values from function approximation.
- The MSE graph for this case is:

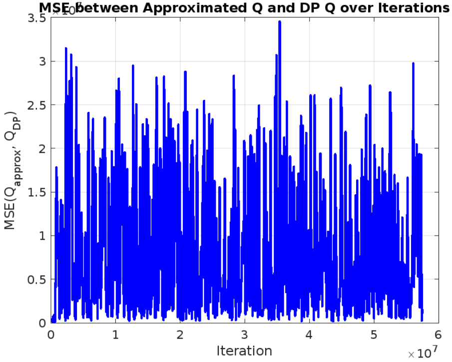

Figure: Caption

Function Approximation with Robbins-Munro Algorithm

- For our case:

$$\alpha(s,a) = \frac{1}{1+N(s,a)}$$

- Using Robbins-Monro algorithm, condition for convergence is $\|\phi(s,a)\|_2 \leq 1 \quad \forall (s,a)$.
- To satisfy this condition,

$$\phi(s,a) \leftarrow \frac{\phi(s,a)}{\|\phi(s,a)\|_2} \quad \forall (s,a)$$

- **Convergence:** Achieved after 3×10^7 iterations.
- **Time Taken:** 570 seconds.
- **Accuracy:** Many Q-values were closer to DP results than with constant learning rate.

Function Approximation with Robbins-Munro Algorithm

- MSE between optimal values(calculated from DP) and Q-function approximation is around 2.8×10^6, the RMS error is around 1673.
- MSE for this case is:

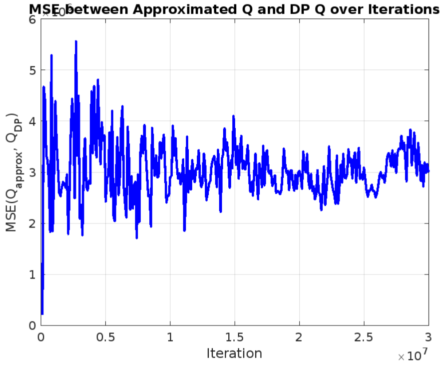

Figure: Caption

Results

- Function approximation remains a powerful approach for scaling reinforcement learning to large domains.
- Function approximation gives faster convergence when compared to Q-learning.
- Using Robbins-Monro algorithm gives us better and faster convergence for both Q-learning and funtion approximation than with constant learning rate.
- Function approximation with constant learning rate does not always guarantees convergence.
- Learning rates must be chosen carefully. For instance, while the Robbins-Monro condition (e.g., $\alpha_t = 1/t$) guarantees convergence, such learning rates can lead to slower convergence in practice.

Thank you