5. Reinforcement Learning: Other Solution Methods

Based on Sutton and Barto's Reinforcement Learning Book

Contents

1 Introduction to Other Solution Methods

In previous sections, we explored policy iteration and value iteration methods for solving reinforcement learning problems, which require a complete model of the environment. However, in many practical scenarios, we either don't have a complete model or it's computationally expensive to use one. In this section, we explore alternative approaches to solving reinforcement learning problems that either relax the requirement for a complete model or approach the problem from different angles.

These alternative methods include:

- Understanding the limitations of on-policy methods
- Monte Carlo techniques for estimating value functions
- Temporal Difference Learning, which combines ideas from Monte Carlo methods and dynamic programming
- Q-Learning as an off-policy temporal difference method
- Policy Gradient methods that directly optimize policy parameters

These alternative approaches are crucial for extending reinforcement learning to a wide variety of practical applications where complete models are unavailable or impractical.

2 Limitations of On-Policy Methods

On-policy methods evaluate and improve the same policy that is used to make decisions. While conceptually straightforward, this approach has several limitations:

2.1 Definition and Characteristics

On-policy methods learn the value functions and improve the policy based on actions taken according to the current policy. The key characteristic is that the policy being evaluated and the policy generating behavior are the same.

2.2 Limitations

2.2.1 Exploration-Exploitation Tradeoff

One fundamental limitation is the exploration-exploitation tradeoff. To find the optimal policy, the agent needs to explore various states and actions. However, to maximize rewards, it should exploit its current knowledge by choosing the actions it believes are best. On-policy methods must balance these competing objectives using the same policy.

For example, in ϵ-greedy policies, the agent follows the greedy policy with probability $1 - \epsilon$ and takes a random action with probability ϵ. While this introduces exploration, it also means the policy being learned is not truly optimal but is constrained by the exploration requirements.

2.2.2 Sample Inefficiency

On-policy methods can be sample inefficient because they can only learn from data collected under the current policy. When the policy changes, previously collected experience becomes less relevant or even irrelevant for evaluating the new policy. This limitation is particularly significant in environments where data collection is expensive or time-consuming.

2.2.3 Necessity for Incremental Policy Improvement

On-policy methods typically require incremental policy improvements to ensure stability. Large policy changes can lead to drastic changes in the data distribution, making it difficult to accurately estimate value functions.

2.3 On-Policy vs. Off-Policy Learning

To address these limitations, we can use off-policy methods, which allow the agent to learn from data that was collected using a different policy (behavior policy) than the one being evaluated and improved (target policy). This separation offers more flexibility but introduces its own challenges.

3 Monte Carlo Techniques

Monte Carlo (MC) methods learn directly from complete episodes of experience without requiring a model of the environment's dynamics. They are based on averaging complete returns from episodes of experience.

3.1 Properties of Monte Carlo Methods

- Only applicable to episodic tasks (tasks with a definite ending point)
- Learn from complete episodes: no bootstrapping
- Can be off-policy, learning about a target policy while following a behavior policy
- Do not require knowledge about environment dynamics
- Simple conceptually and can handle environments with unknown dynamics

3.2 Monte Carlo Prediction (Evaluation)

Monte Carlo prediction estimates the state-value function v_π for a given policy π by averaging returns observed after visits to a state.

Algorithm 1 First-visit Monte Carlo Prediction

Input: policy π to be evaluated
Initialize $V(s)$ for all $s \in \mathcal{S}$
Initialize $Returns(s)$ as an empty list for all $s \in \mathcal{S}$
for each episode **do**
    Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for $t = T-1, T-2, \ldots, 0$ **do**
        $G \leftarrow \gamma G + R_{t+1}$
        if S_t appears for the first time in the episode **then**
            Append G to $Returns(S_t)$
            $V(S_t) \leftarrow average(Returns(S_t))$
        end if
    end for
end for

Notation Overview

- π - policy to be evaluated
- $V(s)$ - estimated value function for state s
- $Returns(s)$ - list of returns observed after visiting state s
- S_t - state at time t
- A_t - action at time t
- R_t - reward at time t
- G - return (cumulative discounted reward)
- γ - discount factor
- T - final time step of the episode

The first-visit MC method estimates $v_\pi(s)$ as the average of returns following the first visits to state s, while the every-visit MC method averages returns following all visits to s.

3.3 Monte Carlo Control

Monte Carlo control methods find the optimal policy by alternating between policy evaluation and policy improvement, similar to policy iteration but using Monte Carlo methods for the evaluation step.

Algorithm 2 Monte Carlo Control with Exploring Starts

Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Initialize $\pi(s)$ arbitrarily for all $s \in \mathcal{S}$
Initialize $Returns(s, a)$ as empty lists for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for each episode **do**
    Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ such that all pairs have probability > 0
    Generate an episode from S_0, A_0 following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for $t = T-1, T-2, \ldots, 0$ **do**
        $G \leftarrow \gamma G + R_{t+1}$
        if (S_t, A_t) pair appears for the first time in the episode **then**
            Append G to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow average(Returns(S_t, A_t))$
            $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
        end if
    end for
end for

Notation Overview

- $Q(s, a)$ - action-value function for state s and action a
- $\pi(s)$ - policy that maps states to actions
- $Returns(s, a)$ - list of returns observed after taking action a in state s
- S_t - state at time t
- A_t - action at time t
- R_t - reward at time t
- G - return (cumulative discounted reward)
- γ - discount factor
- T - final time step of the episode
- $\mathcal{S}$ - set of all states
- $\mathcal{A}(s)$ - set of all actions available in state s

3.4 Off-Policy Monte Carlo Methods

Off-policy Monte Carlo methods allow learning about one policy (the target policy) while following another policy (the behavior policy). This approach addresses the exploration-exploitation tradeoff by using a separate exploratory policy for data collection.

Algorithm 3 Off-policy Monte Carlo Prediction

Input: target policy π to be evaluated
Initialize $V(s)$ for all $s \in \mathcal{S}$
Initialize $C(s)$ to 0 for all $s \in \mathcal{S}$
for each episode **do**
    Generate an episode using behavior policy b: $S_0, A_0, R_1, S_1, A_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    for $t = T-1, T-2, \ldots, 0$ **do**
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t) \leftarrow C(S_t) + W$
        $V(S_t) \leftarrow V(S_t) + \frac{W}{C(S_t)}[G - V(S_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$
        if $W = 0$ **then**
            break
        end if
    end for
end for

Notation Overview

- π - target policy to be evaluated
- b - behavior policy used to generate episodes
- $V(s)$ - estimated value function for state s
- $C(s)$ - cumulative weight for state s
- W - importance sampling weight
- S_t - state at time t
- A_t - action at time t
- R_t - reward at time t
- G - return (cumulative discounted reward)
- γ - discount factor
- T - final time step of the episode
- $\pi(A_t|S_t)$ - probability of taking action A_t in state S_t under policy π
- $b(A_t|S_t)$ - probability of taking action A_t in state S_t under policy b

The term $\frac{\pi(A_t|S_t)}{b(A_t|S_t)}$ is the importance sampling ratio, which corrects for the difference in action selection probabilities between the target and behavior policies.

4 Temporal Difference Learning

Temporal Difference (TD) learning combines ideas from Monte Carlo methods and dynamic programming. Like Monte Carlo methods, TD methods can learn directly from experience without a model of the environment. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap).

4.1 TD Prediction

The simplest TD method, known as TD(0), updates the value estimate for a state based on the observed reward and the estimated value of the next state:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{1}$$

Notation Overview

- $V(S_t)$ - estimated value of state S_t
- R_{t+1} - reward received after taking action in state S_t
- S_{t+1} - next state after taking action in state S_t
- α - step-size parameter (learning rate)
- γ - discount factor
- $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ - TD error, often denoted as δ_t

The term $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target, and $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error.

Algorithm 4 Tabular TD(0) for estimating v_π

Input: policy π to be evaluated
Parameters: step size $\alpha \in (0, 1]$
Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$
for each episode **do**
    Initialize S
    for each step of episode **do**
        Take action A according to $\pi(\Delta|S)$
        Observe reward R and next state S'
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    end for
end for

4.2 Advantages of TD Learning

TD learning has several advantages over both Monte Carlo methods and dynamic programming:

- TD methods don't require a model of the environment (unlike DP)
- TD methods can learn online, after each step, without waiting for the end of an episode (unlike MC)
- TD methods can learn from incomplete episodes, making them applicable to continuing (non-terminating) tasks
- TD methods are generally more sample efficient than Monte Carlo methods

4.3 SARSA: On-Policy TD Control

SARSA (State-Action-Reward-State-Action) is an on-policy TD control method that learns action-value functions. The update rule for SARSA is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{2}$$

Notation Overview

- $Q(S_t, A_t)$ - estimated value of taking action A_t in state S_t
- R_{t+1} - reward received after taking action A_t in state S_t
- S_{t+1} - next state after taking action A_t in state S_t
- A_{t+1} - next action taken in state S_{t+1}
- α - step-size parameter (learning rate)
- γ - discount factor

Algorithm 5 SARSA (on-policy TD control) for estimating $Q \approx q_*$

Parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
for each episode **do**
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    for each step of episode **do**
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S'$
        $A \leftarrow A'$
    end for
end for

5 Q-Learning

Q-Learning is an off-policy TD control method. It directly approximates the optimal action-value function, q_*, independent of the policy being followed.

5.1 Q-Learning Algorithm

The update rule for Q-Learning is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{3}$$

Notation Overview

- $Q(S_t, A_t)$ - estimated value of taking action A_t in state S_t
- R_{t+1} - reward received after taking action A_t in state S_t
- S_{t+1} - next state after taking action A_t in state S_t
- $\max_a Q(S_{t+1}, a)$ - maximum estimated value obtainable from state S_{t+1}
- α - step-size parameter (learning rate)
- γ - discount factor

Algorithm 6 Q-Learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
for each episode **do**
    Initialize S
    for each step of episode **do**
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
        $S \leftarrow S'$
    end for
end for

5.2 Differences Between SARSA and Q-Learning

The key difference between SARSA and Q-Learning lies in their update targets:

- SARSA uses the Q-value of the next state-action pair actually taken: $Q(S_{t+1}, A_{t+1})$, making it on-policy.
- Q-Learning uses the maximum Q-value for the next state: $\max_a Q(S_{t+1}, a)$, regardless of what action is actually taken next, making it off-policy.

This means Q-Learning can learn the optimal policy even while following an exploratory policy, while SARSA learns the optimal policy subject to the constraint of the exploration strategy being used.

5.3 Expected SARSA

Expected SARSA is a variant of SARSA that uses the expected value of the next state-action pair instead of the sample:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{4}$$

Notation Overview

- $Q(S_t, A_t)$ - estimated value of taking action A_t in state S_t
- R_{t+1} - reward received after taking action A_t in state S_t
- S_{t+1} - next state after taking action A_t in state S_t
- $\pi(a|S_{t+1})$ - probability of taking action a in state S_{t+1} under policy π
- $\sum_a \pi(a|S_{t+1})Q(S_{t+1}, a)$ - expected value of the next state-action pair
- α - step-size parameter (learning rate)
- γ - discount factor

Expected SARSA can be more computationally expensive than SARSA but can reduce variance in the updates and lead to better performance in some environments.

6 Policy Gradient Methods

Policy gradient methods take a different approach by directly parameterizing the policy and updating the parameters to maximize the expected return.

6.1 Policy Parameterization

The policy is represented as a parameterized function:

$$\pi(a|s,\theta) = P(A_t = a|S_t = s, \theta_t = \theta) \tag{5}$$

where θ is a vector of policy parameters that we adjust to optimize performance.

Notation Overview

- $\pi(a|s,\theta)$ - probability of taking action a in state s under the policy parameterized by θ
- θ - vector of policy parameters
- A_t - action at time t
- S_t - state at time t
- θ_t - policy parameters at time t

6.2 Performance Measure

The performance measure to be maximized is the expected return:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_0] \tag{6}$$

where G_0 is the return starting from the initial state.

Notation Overview

- $J(\theta)$ - expected return under the policy parameterized by θ
- $\mathbb{E}_{\pi_\theta}$ - expectation over trajectories generated by following policy π_θ
- G_0 - return (cumulative discounted reward) starting from the initial state
- π_θ - policy parameterized by θ

6.3 Policy Gradient Theorem

The policy gradient theorem provides a way to compute the gradient of $J(\theta)$ without requiring differentiation of the state distribution:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi(A_t|S_t,\theta) \cdot G_t] \tag{7}$$

Notation Overview

- $\nabla_\theta J(\theta)$ - gradient of the expected return with respect to policy parameters
- $\mathbb{E}_{\pi_\theta}$ - expectation over trajectories generated by following policy π_θ
- $\nabla_\theta \log \pi(A_t|S_t, \theta)$ - gradient of the log probability of taking action A_t in state S_t
- G_t - return (cumulative discounted reward) from time step t
- π_θ - policy parameterized by θ

6.4 REINFORCE Algorithm

REINFORCE is a Monte Carlo policy gradient method that implements the policy gradient theorem:

Algorithm 7 REINFORCE (Monte Carlo Policy Gradient)

Parameters: step size $\alpha > 0$
Initialize policy parameters θ arbitrarily
for each episode **do**
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ following $\pi(\Delta|\Delta, \theta)$
    for $t = 0, 1, \ldots, T-1$ **do**
        $G \leftarrow$ return from step t
        $\theta \leftarrow \theta + \alpha \gamma^t G \nabla_\theta \log \pi(A_t|S_t, \theta)$
    end for
end for

Notation Overview

- θ - policy parameters
- α - step-size parameter (learning rate)
- γ - discount factor
- G - return (cumulative discounted reward)
- $\nabla_\theta \log \pi(A_t|S_t, \theta)$ - gradient of the log probability of taking action A_t in state S_t
- S_t - state at time t
- A_t - action at time t
- R_t - reward at time t
- T - final time step of the episode

6.5 Advantages of Policy Gradient Methods

Policy gradient methods have several advantages:

- They can learn stochastic policies, which is important in partially observable environments
- They naturally handle continuous action spaces

- They can incorporate prior knowledge by initializing the policy to specific behaviors
- Convergence properties are often better than value-based methods
- Function approximation is more stable with policy gradients than with value-based methods

7 Comparison of Methods

Method	**Model Required**	**On/Off-Policy**	**Bootstrapping**	**Updates**
Dynamic Programming	Yes	On-policy	Yes	Sweep
Monte Carlo	No	Both	No	Episode
TD Learning	No	Both	Yes	Step
Q-Learning	No	Off-policy	Yes	Step
SARSA	No	On-policy	Yes	Step
Policy Gradient	No	Typically on-policy	Varies	Episode/Step

Table 1: Comparison of Reinforcement Learning Methods

8 Conclusion

We have explored various alternative solution methods for reinforcement learning problems beyond the classical dynamic programming approaches. Each method has its strengths and weaknesses, making them suitable for different types of problems and environments.

Monte Carlo methods are conceptually simple and learn directly from complete episodes, but they can only be applied to episodic tasks and may require more samples. Temporal Difference methods combine ideas from Monte Carlo and dynamic programming, allowing learning from incomplete sequences and bootstrapping from existing estimates. Q-Learning provides an off-policy approach that can learn the optimal policy while following an exploratory policy. Finally, policy gradient methods offer a direct approach to optimizing policies, which is particularly useful for continuous action spaces and stochastic policies.

The choice of method depends on the specific problem characteristics, such as whether the environment is episodic or continuing, whether a model is available, and whether the state and action spaces are discrete or continuous.