Reinforcement Learning

Lenny Pelhate

April 2025

Contents

1 Introduction

Reinforcement Learning (RL) is a machine learning paradigm concerned with how intelligent agents should take actions in an environment to maximize cumulative reward. Unlike supervised learning, RL doesn't rely on labeled examples but instead learns from interaction with the environment. This learning process involves:

- An **agent** that makes decisions (**actions**)
- An environment (**states**) that responds to these decisions
- A **transition function** that defines how the environment moves from one state to another given an action
- A **reward** signal that indicates the quality of decisions

The goal of RL is to develop algorithms that enable agents to learn optimal behavior through experience. This presentation explores the mathematical foundations of RL, from basic formalism to advanced deep learning approaches.

2 Formalism

2.1 Markov Decision Processes

Reinforcement learning problems are typically formalized as **Markov Decision Processes (MDPs)**.

An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where:

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{A}$ is a (finite) set of actions
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability function
  - $T(s, a, s') = P(s'|s, a)$ is the probability of transitioning to state s' when taking action a in state s
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function
  - $R(s, a, s')$ is the immediate reward after transitioning from s to s' due to action a
  - In many cases, the reward function is simplified to $r(s, a)$, especially when the environment is deterministic and the next state s' is uniquely determined by s and a
- $\gamma \in [0, 1]$ is the discount factor, which favors shorter term rewards. The closer γ is to 1, the more importance far rewards have.

So, the system is in state $s \in \mathcal{S}$, the agent chooses an action $a \in \mathcal{A}$ and gets the reward $r(s, a)$. Then, the system transits stochastically to a new state s', this new state being drawn from the conditional probability $P(\cdot\,|s, a)$.

2.2 Policy and Value Functions

2.2.1 Policy Function

A **policy** is a mapping from states to actions. The policy defines the behavior of an agent:

- If the policy is **deterministic**, then $\pi : \mathcal{S} \to \mathcal{A}$, (also written as $\mathcal{A}^{\mathcal{S}}$) assigning a unique action to each state:
$$\pi(s) = a$$
- If the policy is **stochastic**, then $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, and $\pi(a|s)$ gives the probability of taking action a in state s:
$$\pi(a|s) = \mathbb{P}(a_t = a \mid s_t = s)$$

2.2.2 Value Function

2.2.2.1 State-value Function

The state-value function $V^\pi(s)$ represents the expected return starting from state s and following policy π:
$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, s_0 = s \right]$$

- If π is **deterministic**, then $a_t = \pi(s_t)$, and the expectation is over the stochastic transitions $P(s_{t+1} = s' \mid s_t, a_t)$.
- If π is **stochastic**, then $a_t \sim \pi(\cdot \mid s_t)$, and the expectation is taken over both the policy π and the stochastic transitions $P(s_{t+1} = s' \mid s_t, a_t)$.

2.2.2.2 Action-value Function

The action-value function $Q^\pi(s, a)$ (also called **Q-function** or **quality function**) represents the expected return starting from state s, taking action a, and then following policy π:
$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, s_0 = s, a_0 = a \right]$$

- $s_0 = s$, $a_0 = a$, and for $t \geq 1$, $a_t = \pi(s_t)$ or $a_t \sim \pi(\cdot \mid s_t)$, depending on whether the policy is deterministic or stochastic.
- The expectation is taken over all possible trajectories $(s_1, a_1, s_2, a_2, \dots)$ under the transition dynamics $P(s_{t+1} \mid s_t, a_t)$.

2.2.2.3 Relationship Between State-value and Action-value Functions

The state-value function $V^\pi(s)$ can be expressed in terms of the action-value function $Q^\pi(s,a)$ as the expected value of $Q^\pi(s,a)$ under the policy π:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, Q^\pi(s,a)$$

Note: For a deterministic policy $\pi(s)$, the value function $V^\pi(s)$ simplifies to:

$$V^\pi(s) = Q^\pi(s, \pi(s))$$

Proof (stochastic policy case)

By definition, the state-value function $V^\pi(s)$ is the expected return starting from state s and following policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, s_0 = s \right]$$

We can break this expectation into two steps: choosing the first action $a_0 \sim \pi(\cdot \mid s_0)$, and then following the trajectory from there.

By the **law of total expectation**:

$$\begin{aligned} V^\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a \mid s)\, \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, s_0 = s, a_0 = a \right] \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid s)\, Q^\pi(s,a) \end{aligned}$$

2.3 Optimal Policies and the Objective of MDPs

The central objective in a Markov Decision Process (MDP) is to find an **optimal policy** π^* that maximizes the expected cumulative reward (also called the return) over time. In other words, we seek a policy under which the agent's long-term performance is as high as possible. Whether the policy is deterministic or stochastic, solving the MDP involves determining the best possible way to act in each state so as to maximize value functions like $V^\pi(s)$ or $Q^\pi(s,a)$.

A value function allows quantifying the quality of a policy, and it allows comparing policies as follows:

$$\begin{aligned} \pi_1 \geq \pi_2 &\iff \forall s \in \mathcal{S}, && V^{\pi_1}(s) \geq V^{\pi_2}(s) \\ &\iff \forall s \in \mathcal{S},\, a \in \mathcal{A}, && Q^{\pi_1}(s,a) \geq Q^{\pi_2}(s,a) \end{aligned}$$

An **optimal policy** π^* is one that achieves the highest expected return from every state. Formally, this means:

$$\forall s \in \mathcal{S}, \quad \pi^*(s) \in \arg\max_{\pi \in \mathcal{S}} V^\pi(s)$$

Or, in terms of the action-value function:

$$\forall s \in \mathcal{S}, \quad \pi^*(s) \in \arg\max_{a \in \mathcal{A}} Q^{\pi^*}(s,a)$$

2.4 Bellman Equations

The Bellman equations express the relationship between the value of a state and the values of its successor states. They enable backward induction, a key technique in Dynamic Programming. DP algorithms like Value Iteration and Policy Iteration use them to iteratively compute the optimal solution starting from terminal states.

2.4.1 Bellman Equations for V^π

2.4.1.1 Bellman Expectation Equation

Stochastic Policy, Stochastic Transitions:

Reward depends on (s, a, s')

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left[R(s, a, s') + \gamma V^\pi(s') \right]$$

Deterministic Policy, Stochastic Transitions:

Reward depends on (s, a)

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s))\, V^\pi(s')$$

Deterministic Policy, Deterministic Transitions:

Reward depends on (s, a)

$$V^\pi(s) = r(s, \pi(s)) + \gamma V^\pi(s')$$

Proof (deterministic policy and stochastic transitions case)

The value function is defined as:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s \right]$$

Expanding the first time step, and since $\forall t \geq 0,\ a_t = \pi(s_t)$:

$$\begin{aligned}
V^\pi(s) &= \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, s_0 = s \right] \\
&= \mathbb{E}_\pi \left[r(s_0, \pi(s_0)) \,|\, s_0 = s \right] + \gamma\, \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, \pi(s_t)) \,\middle|\, s_0 = s \right] \\
&= r(s, \pi(s)) + \gamma\, \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_{t+1}, \pi(s_{t+1})) \,\middle|\, s_0 = s,\ s_1 = s' \right] \\
&= r(s, \pi(s)) + \gamma\, \mathbb{E}_\pi \left[V^\pi(s_1) \,|\, s_1 = s' \right] \\
&= r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \,|s, \pi(s))\, V^\pi(s')
\end{aligned}$$

2.4.1.2 Bellman Optimality Equation

Stochastic Policy, Stochastic Transitions:

Reward depends on (s, a, s')

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left[R(s, a, s') + \gamma V^*(s') \right]$$

Deterministic Policy, Stochastic Transitions:

Reward depends on (s, a)

$$V^*(s) = \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V^*(s') \right]$$

Deterministic Policy, Deterministic Transitions:

Reward depends on (s, a)

$$V^*(s) = \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma V^*(s') \right]$$

2.4.2 Bellman Equations for Q^π

2.4.2.1 Bellman Expectation Equation

Stochastic Policy, Stochastic Transitions:

Reward depends on (s, a, s')

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left[R(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q^\pi(s', a') \right]$$

Deterministic Policy, Stochastic Transitions:

Reward depends on (s, a)

$$Q^\pi(s, \pi(s)) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) Q^\pi(s', \pi(s'))$$

Deterministic Policy, Deterministic Transitions:

Reward depends on (s, a)

$$Q^\pi(s, \pi(s)) = r(s, \pi(s)) + \gamma Q^\pi(s', \pi(s'))$$

Proof (deterministic policy and stochastic transitions case)

The Q-value function is defined as:

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right]$$

Expanding the first time step, and since $\forall t \geq 0$, $a_t = \pi(s_t)$:

$$\begin{aligned}
Q^\pi(s,a) &= \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right] \\
&= \mathbb{E}_\pi \left[r(s_0, a_0) \mid s_0 = s, a_0 = a \right] + \gamma \mathbb{E}_\pi \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \mid s_0 = s, a_0 = a \right] \\
&= r(s,a) + \gamma \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_{t+1}, \pi(s_{t+1})) \mid s_0 = s, s_1 = s', a_0 = a \right] \\
&= r(s,a) + \gamma \mathbb{E}_\pi \left[Q^\pi(s_1, a_1) \mid s_1 = s' \right] \\
&= r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q^\pi(s', a')
\end{aligned}$$

2.4.2.2 Bellman Optimality Equation

Stochastic Policy, Stochastic Transitions:

Reward depends on (s, a, s')

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \left[R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \right]$$

Deterministic Policy, Stochastic Transitions:

Reward depends on (s, a)

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \max_{a' \in \mathcal{A}} Q^*(s', a')$$

Deterministic Policy, Deterministic Transitions:

Reward depends on (s, a)

$$Q^*(s,a) = r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a')$$

2.5 Bellman Operators

The Bellman equations can be expressed concisely using operators that transform value functions. These operators are fundamental to the theoretical analysis of reinforcement learning algorithms.

2.5.1 Bellman Policy Operator

Stochastic Policy, Stochastic Transitions:

Reward depends on (s, a, s')

$$\mathcal{T}^{\pi}V(s) = \sum_{a\in\mathcal{A}} \pi(a \mid s) \sum_{s'\in\mathcal{S}} P(s' \mid s, a) \left[R(s, a, s') + \gamma V(s')\right]$$

$$\mathcal{T}^{\pi}Q(s, a) = \sum_{s'\in\mathcal{S}} P(s' \mid s, a) \left[R(s, a, s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a' \mid s')Q(s', a')\right]$$

Deterministic Policy, Stochastic Transitions:

Reward depends on (s, a)

$$\mathcal{T}^{\pi}V(s) = r(s, \pi(s)) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s, \pi(s))V(s')$$

$$\mathcal{T}^{\pi}Q(s, \pi(s)) = r(s, \pi(s)) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s, \pi(s))Q(s', \pi(s'))$$

Deterministic Policy, Deterministic Transitions:

Reward depends on (s, a)

$$\mathcal{T}^{\pi}V(s) = r(s, \pi(s)) + \gamma V(s')$$

$$\mathcal{T}^{\pi}Q(s, \pi(s)) = r(s, \pi(s)) + \gamma Q(s', \pi(s'))$$

2.5.2 Bellman Optimality Operator

Stochastic Policy, Stochastic Transitions:

Reward depends on (s, a, s')

$$\mathcal{T}^{*}V(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s' \mid s, a) \left[R(s, a, s') + \gamma V(s')\right]$$

$$\mathcal{T}^{*}Q(s, a) = \max_{a'\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s' \mid s, a) \left[R(s, a, s') + \gamma Q(s', a')\right]$$

Deterministic Policy, Stochastic Transitions:

Reward depends on (s, a)

$$\mathcal{T}^{*}V(s) = \max_{a\in\mathcal{A}} \left[r(s, a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s, a)V(s')\right]$$

$$\mathcal{T}^{*}Q(s, \pi(s)) = \max_{a'\in\mathcal{A}} \left[r(s, a) + \gamma \sum_{s'\in\mathcal{S}} P(s' \mid s, a)Q(s', a')\right]$$

Deterministic Policy, Deterministic Transitions:

Reward depends on (s, a)

$$\mathcal{T}^{*}V(s) = \max_{a\in\mathcal{A}} \left[r(s, a) + \gamma V(s')\right]$$

$$\mathcal{T}^{*}Q(s, \pi(s)) = \max_{a'\in\mathcal{A}} \left[r(s, a) + \gamma Q(s', a')\right]$$

2.5.3 Theoretical Properties of Bellman Operators

In this subsection, we consider the case where the policy π is **deterministic**, i.e., $a = \pi(s)$, and the environment dynamics are governed by **stochastic transitions**. Specifically, the transition model is given by $P(s' \mid s, a)$, and the reward function depends only on the current state and action, $r(s, a)$.

2.5.3.1 Monotonicity Property

The Bellman operators T^π and T^* are monotone. That is, for any value functions $U, V \in \mathbb{R}^{\mathcal{S}}$:

$$U \leq V \quad \Rightarrow \quad T^\pi U \leq T^\pi V \quad \text{and} \quad T^* U \leq T^* V$$

Proof for T^π

Assume $U \leq V$ pointwise, i.e., $U(s) \leq V(s)$ for all $s \in \mathcal{S}$. Then:

$$\begin{aligned} T^\pi U(s) &= r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) U(s') \\ &\leq r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) V(s') \\ &= T^\pi V(s) \end{aligned}$$

This holds for all $s \in \mathcal{S}$, so $T^\pi U \leq T^\pi V$.

Proof for T^*

Assume $U \leq V$ pointwise. For any state $s \in \mathcal{S}$:

$$\begin{aligned} T^* U(s) &= \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) U(s') \right] \\ &\leq \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V(s') \right] \\ &= T^* V(s) \end{aligned}$$

This holds for all $s \in \mathcal{S}$, so $T^* U \leq T^* V$.

2.5.3.2 Offset Property

Let $V : \mathcal{S} \to \mathbb{R}$ be a value function and $c \in \mathbb{R}$ a constant. Then for any Bellman operator $\mathcal{T}$, we have:

$$\mathcal{T}(V + c)(s) = \mathcal{T}V(s) + \gamma c$$

Proof for $\mathcal{T}^\pi$

$$\begin{aligned}
\mathcal{T}^\pi(V+c)(s) &= r(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) \left[V(s') + c\right] \\
&= r(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) V(s') + \gamma c \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) \\
&= r(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) V(s') + \gamma c \cdot 1 \\
&= \mathcal{T}^\pi V(s) + \gamma c
\end{aligned}$$

Proof for $\mathcal{T}^*$

$$\begin{aligned}
\mathcal{T}^*(V+c)(s) &= \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a)(V(s') + c) \right] \\
&= \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V(s') + \gamma c \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \right] \\
&= \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V(s') + \gamma c \cdot 1 \right] \\
&= \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V(s') \right] + \gamma c \\
&= \mathcal{T}^* V(s) + \gamma c
\end{aligned}$$

2.5.3.3 Contraction Mapping Property

The Bellman operators T^π and T^* are γ-contraction mappings with respect to the max-norm (supremum norm). That is, for any two value functions $U, V \in \mathbb{R}^{\mathcal{S}}$:

$$\|T^\pi U - T^\pi V\|_\infty \leq \gamma \|U - V\|_\infty$$

$$\|T^* U - T^* V\|_\infty \leq \gamma \|U - V\|_\infty$$

Proof for T^π

Let $U, V \in \mathbb{R}^{\mathcal{S}}$. For any state $s \in \mathcal{S}$:

$$\begin{aligned}
\Big|T^\pi U(s) - T^\pi V(s)\Big| &= \Bigg| r(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) U(s') \\
&\qquad - \Big(r(s,\pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) V(s') \Big) \Bigg| \\
&= \left| \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) \left(U(s') - V(s') \right) \right| \\
&\leq \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) \Big| U(s') - V(s') \Big| \\
&\leq \gamma \|U - V\|_\infty \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) \\
&= \gamma \|U - V\|_\infty
\end{aligned}$$

Taking maximum over all $s \in \mathcal{S}$, we obtain:

$$\|T^\pi U - T^\pi V\|_\infty \leq \gamma \|U - V\|_\infty$$

Hence, T^π is a γ-contraction mapping under the max-norm.

Proof for T^*

Let $U, V \in \mathbb{R}^{\mathcal{S}}$. For any state $s \in \mathcal{S}$, define:

$$a_U = \arg\max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) U(s') \right]$$
$$a_V = \arg\max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V(s') \right]$$

Then,

$$T^*U(s) = r(s,a_U) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a_U) U(s')$$
$$T^*V(s) = r(s,a_V) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a_V) V(s')$$

Now consider the difference:

$$\begin{aligned} T^*U(s) - T^*V(s) &\leq r(s,a_V) + \gamma \sum_{s'} P(s' \mid s,a_V) U(s') \\ &\quad - \left(r(s,a_V) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a_V) V(s') \right) \\ &= \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a_V)(U(s') - V(s')) \\ &\leq \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a_V) |U(s') - V(s')| \\ &\leq \gamma \|U - V\|_\infty \end{aligned}$$

Similarly:

$$T^*V(s) - T^*U(s) \leq \gamma \|V - U\|_\infty = \gamma \|U - V\|_\infty$$

Hence:

$$\left| T^*U(s) - T^*V(s) \right| \leq \gamma \|U - V\|_\infty$$

Taking max over all $s \in \mathcal{S}$, we get:

$$\|T^*U - T^*V\|_\infty \leq \gamma \|U - V\|_\infty$$

Therefore, T^* is also a γ-contraction mapping with respect to the max-norm.

2.5.3.4 Fixed Point Property

- The value function V^π is the unique fixed point of T^π:

$$T^\pi V^\pi = V^\pi$$

- The optimal value function V^* is the unique fixed point of T^*:

$$T^*V^* = V^*$$

Proof for T^π

By definition, $V^\pi(s)$ is the expected return starting from state s and following policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \,\middle|\, s_0 = s \right]$$

We isolate the first reward term:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi \left[r(s_0, \pi(s_0)) + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, \pi(s_t)) \,\middle|\, s_0 = s \right] \\ &= \mathbb{E}_\pi \left[r(s, \pi(s)) + \gamma V^\pi(s_1) \,\middle|\, s_0 = s \right] \\ &= r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) V^\pi(s') \end{aligned}$$

By definition of the Bellman operator for deterministic π:

$$T^\pi V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi(s)) V^\pi(s')$$

Thus,

$$T^\pi V^\pi = V^\pi$$

Since T^π is a contraction mapping, by the **Banach fixed-point theorem**, this fixed point is unique.

Proof for T^*

Let π^* be an optimal deterministic policy such that for all $s \in \mathcal{S}$:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^*(s') \right]$$

Then the value function under this policy satisfies:

$$V^*(s) = r(s, \pi^*(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi^*(s)) V^*(s')$$

By the definition of T^*:

$$T^*V^*(s) = \max_{a \in \mathcal{A}} \left[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^*(s') \right]$$

Since $\pi^*(s)$ attains this maximum, we have:

$$T^*V^*(s) = r(s, \pi^*(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, \pi^*(s)) V^*(s') = V^*(s)$$

Thus,

$$T^*V^* = V^*$$

And since T^* is also a contraction mapping, by the **Banach fixed-point theorem**, the fixed point is unique.

3 Dynamic Programming

Dynamic Programming (DP) methods solve reinforcement learning problems by recursively computing value functions, assuming full knowledge of the Markov Decision Process (MDP). These methods are capable of computing the best possible control when the MDP is known, as they rely on exact transition probabilities and reward functions to evaluate policies and find the optimal solution.

3.1 Linear Programming

Given a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ and a Markov Decision Process with stochastic transitions $P(s' \mid s, a)$, the corresponding value function V^π can be found by solving the following linear program:

$$\begin{cases} \min\limits_{V} & \sum_{s\in\mathcal{S}} V(s) \\ \text{s.t.} & V(s) \geq r(s,\pi(s)) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,\pi(s))V(s'), \quad \forall s \in \mathcal{S} \end{cases}$$

Correctness

Let V^π be the value function corresponding to the deterministic policy π. Then, the Bellman equation for V^π is:

$$V^\pi(s) = r(s,\pi(s)) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,\pi(s))V^\pi(s')$$

Any feasible solution V to the linear program must satisfy:

$$V(s) \geq r(s,\pi(s)) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,\pi(s))V(s'), \quad \forall s \in \mathcal{S}$$

Minimizing $\sum_{s\in\mathcal{S}} V(s)$ ensures that these inequalities become tight at the optimal solution, yielding:

$$V(s) = r(s,\pi(s)) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,\pi(s))V(s')$$

Hence, the solution satisfies the Bellman equation for π. Since this equation has a unique solution, the LP must return V^π.

3.2 Policy Iteration

Policy Iteration alternates between **policy evaluation** and **policy improvement**.

3.2.1 Policy Evaluation

Policy evaluation is the process of computing the value function V^π for a given policy π, which represents the expected return from each state when following that policy under the environment's dynamics.

Algorithm 1 Policy Evaluation Algorithm

Input: Policy π, transition probabilities $P(s'|s,a)$, reward function $R(s,a,s')$, discount factor γ, tolerance ϵ
Initialize: $V^\pi(s) = 0$ for all $s \in \mathcal{S}$
repeat
  $\Delta \leftarrow -\infty$
  for each state $s \in \mathcal{S}$ **do**
    $v \leftarrow V^\pi(s)$

$$V^\pi(s) \leftarrow \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s'|s,a) \left[R(s,a,s') + \gamma V^\pi(s')\right]$$

    $\Delta \leftarrow \max(\Delta, |v - V^\pi(s)|)$
  end for
until $\Delta < \epsilon$
Output: $V^\pi(s)$ for all $s \in \mathcal{S}$

3.2.2 Policy Improvement

Policy improvement involves iteratively enhancing a policy to maximize expected returns. The process uses the current policy to evaluate state values, and then improves the policy by selecting actions that maximize these values. It ensures that the value function of the improved policy will always be greater than or equal to the original policy, thus validating the policy improvement method. This iterative process converges to the optimal policy.

Algorithm 2 Policy Improvement Algorithm

Input: State-value function $V^\pi(s)$, transition probabilities $P(s'|s,a)$, reward function $R(s,a,s')$, discount factor γ
Initialize: Policy $\pi(s)$ for all $s \in \mathcal{S}$
repeat
  For each state $s \in \mathcal{S}$:
    Update the policy $\pi(s)$ by choosing the action that maximizes the expected value:

$$\pi(s) \leftarrow \arg\max_{a \in A} \sum_{s' \in S} P(s'|s,a) \left[R(s,a,s') + \gamma V^\pi(s')\right]$$

until the policy π converges (i.e. no further updates are made)
Output: The improved policy $\pi(s)$

3.2.3 Policy Iteration

Policy iteration is an iterative algorithm that alternates between evaluating the current policy by computing the state values (**policy evaluation**) and improving the policy by updating it based on the current value function (**policy improvement**) until convergence to the optimal policy.

Algorithm 3 Policy Iteration Algorithm

Input: Transition probabilities $P(s'|s,a)$, reward function $R(s,a,s')$, discount factor γ, tolerance ϵ
Initialize: Arbitrary policy π_0, $V^{\pi_0}(s) = 0$ for all $s \in \mathcal{S}$
repeat
  Policy Evaluation:
    Compute $V^{\pi_k}(s)$ by solving:

$$V^{\pi_k}(s) = \sum_{s' \in S} P(s'|s, \pi_k(s)) \left[R(s, \pi_k(s), s') + \gamma V^{\pi_k}(s') \right]$$

  Policy Improvement:
    Update the policy:

$$\pi_{k+1}(s) = \arg\max_{a \in A} \sum_{s' \in S} P(s'|s,a) \left[R(s,a,s') + \gamma V^{\pi_k}(s') \right]$$

until $\pi_{k+1} = \pi_k$
Output: Optimal policy $\pi_{k+1} = \pi^*$

3.3 Value Iteration

Value Iteration is an algorithm for computing the optimal value function and policy in a Markov Decision Process (MDP). It iteratively updates the value function for each state by considering the maximum expected return achievable from that state, until convergence. By the Banach fixed-point theorem, repeated application of the Bellman operator T converges to a unique fixed point, which is V^*.

Algorithm 4 Value Iteration Algorithm

Input: Transition probabilities $P(s'|s,a)$, reward function $R(s,a,s')$, discount factor γ, tolerance ϵ
Initialize: $V(s) = 0$ for all $s \in \mathcal{S}$
repeat
  $\Delta \leftarrow -\infty$
  for each state $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    Update $V(s)$ by:

$$V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a) \left[R(s,a,s') + \gamma V(s') \right]$$

    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \epsilon$
Output: The optimal value function $V^*(s)$ for all $s \in \mathcal{S}$

4 Approximate Dynamic Programming

While classical approaches like dynamic programming offer theoretical solutions for small problems, approximate methods and deep learning techniques extend these ideas to complex, high-dimensional domains.

In reinforcement learning and dynamic programming, exact methods like value iteration or policy iteration become computationally intractable when dealing with **large or continuous state spaces**. Approximate Dynamic Programming (ADP) addresses this challenge by using function approximation techniques to estimate value functions, policies, or models instead of explicitly computing them for every state.

In large or continuous state spaces, where it is impossible to enumerate all possible states, Approximate Dynamic Programming (ADP) allows us to approximate the value function using methods such as linear or nonlinear function approximators (e.g., neural networks), enabling the solution of problems in high-dimensional spaces. These techniques facilitate scaling reinforcement learning methods to real-world problems where exact solutions would be infeasible.

4.1 Approximate Policy Iteration

Approximate Policy Iteration (API) combines policy evaluation and policy improvement with function approximation techniques. The policy evaluation step involves approximating the value function using a parameterized function $\tilde{V}(s|w)$ or the action-value function $\tilde{Q}(s, a|w)$, depending on whether the method focuses on state values or action-values.

Algorithm 5 Approximate Policy Iteration Algorithm (V-function)

1: **Input:** Transition model $P(s'|s, a)$, reward function $R(s, a, s')$, discount factor γ, function approximator $\tilde{V}(s|w)$, evaluation method, tolerance ϵ
2: **Initialize:** Policy π_0, parameters w_0
3: **repeat**
4: **Policy Evaluation:**
5: Use a policy evaluation method to compute parameters w_k such that:

$$\tilde{V}(s|w_k) \approx V^{\pi_k}(s)$$

6: **Policy Improvement:**
7: Update the policy:

$$\pi_{k+1}(s) = \arg\max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s'} P(s'|s, a) \tilde{V}(s'|w_k) \right)$$

8: **until** convergence (i.e., $\pi_{k+1} = \pi_k$)
9: **Output:** Final policy π^* and value function $\tilde{V}(s|w^*)$

Algorithm 6 Approximate Policy Iteration Algorithm (Q-function)

1: **Input:** Transition model $P(s'|s,a)$, reward function $R(s,a,s')$, discount factor γ, function approximator $\tilde{Q}(s,a\,|\,w)$, evaluation method, tolerance ϵ
2: **Initialize:** Policy π_0, parameters w_0
3: **repeat**
4: **Policy Evaluation:**
5: Use a policy evaluation method to compute parameters w_k such that:

$$\tilde{Q}(s,a|w_k) \approx Q^{\pi_k}(s,a)$$

6: **Policy Improvement:**
7: Update the policy:

$$\pi_{k+1}(s) = \arg\max_{a\in\mathcal{A}} \tilde{Q}(s,a\,|\,w_k)$$

8: **until** convergence (i.e., $\pi_{k+1} = \pi_k$)
9: **Output:** Final policy π^* and Q-function $\tilde{Q}(s,a\,|\,w^*)$

4.1.1 Approximate Policy Evaluation Methods

For each method, we provide update rules and convergence bounds for either the state value function (V-function) or the state-action value function (Q-function). We choose to focus on one of these functions depending on the specific method, and do not apply the update rules to both simultaneously. The choice between using $V(s)$ or $Q(s,a)$ depends on the underlying approach and objectives of the method, as some methods are designed to estimate value functions, while others target action-value functions.

4.1.1.1 Temporal Difference Methods

1. TD(0):

TD(0) is the simplest temporal difference method for approximate policy evaluation. It updates the parameters based on the TD error after each transition.

V-function Approximation:

$$\begin{aligned} \text{Update Formula:} \quad & \theta_{t+1} = \theta_t + \alpha_t \delta_t \nabla_\theta V_\theta(s_t) \\ \text{TD Error:} \quad & \delta_t = r_{t+1} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t) \end{aligned}$$

Q-function Approximation:

$$\begin{aligned} \text{Update Formula:} \quad & \theta_{t+1} = \theta_t + \alpha_t \delta_t \nabla_\theta Q_\theta(s_t,a_t) \\ \text{TD Error:} \quad & \delta_t = r_{t+1} + \gamma Q_\theta(s_{t+1},a_{t+1}) - Q_\theta(s_t,a_t) \end{aligned}$$

Convergence Bound: (for linear function approximation with appropriate step sizes)

$$\begin{aligned} \text{V-function:} \quad & ||V_{\theta^*} - V_\pi||_D \leq \frac{1}{1-\gamma} \min_\theta ||V_\theta - V_\pi||_D \\ \text{Q-function:} \quad & ||Q_{\theta^*} - Q_\pi||_D \leq \frac{1}{1-\gamma} \min_\theta ||Q_\theta - Q_\pi||_D \end{aligned}$$

where $||.||_D$ is the weighted norm with respect to the stationary distribution D.

2. TD(λ):

TD(λ) balances between TD and Monte Carlo approaches using eligibility traces.

V-function Approximation:

$$\begin{aligned}
\text{Update Formula:} \quad & \theta_{t+1} = \theta_t + \alpha_t \delta_t e_t \\
\text{Eligibility Trace:} \quad & e_t = \gamma\lambda e_{t-1} + \nabla_\theta V_\theta(s_t) \\
\text{TD Error:} \quad & \delta_t = r_{t+1} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)
\end{aligned}$$

Q-function Approximation:

$$\begin{aligned}
\text{Update Formula:} \quad & \theta_{t+1} = \theta_t + \alpha_t \delta_t e_t \\
\text{Eligibility Trace:} \quad & e_t = \gamma\lambda e_{t-1} + \nabla_\theta Q_\theta(s_t, a_t) \\
\text{TD Error:} \quad & \delta_t = r_{t+1} + \gamma Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t)
\end{aligned}$$

Convergence Bound:

$$\begin{aligned}
\text{V-function:} \quad & ||V_{\theta^*} - V_\pi||_D \leq \frac{1}{1-\gamma\lambda} \min_\theta ||V_\theta - V_\pi||_D \\
\text{Q-function:} \quad & ||Q_{\theta^*} - Q_\pi||_D \leq \frac{1}{1-\gamma\lambda} \min_\theta ||Q_\theta - Q_\pi||_D
\end{aligned}$$

3. Gradient TD:

Gradient TD methods are designed to be stable with off-policy learning and function approximation.

V-function Approximation (TDC):

$$\begin{aligned}
\theta_{t+1} &= \theta_t + \alpha_t \delta_t \phi(s_t) - \alpha_t \gamma \phi(s_{t+1})(\phi(s_t)^T w_t) \\
w_{t+1} &= w_t + \beta_t(\delta_t - \phi(s_t)^T w_t)\phi(s_t) \\
\delta_t &= r_{t+1} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)
\end{aligned}$$

Q-function Approximation (GTD2-Q):

$$\begin{aligned}
\theta_{t+1} &= \theta_t + \alpha_t \delta_t \phi(s_t, a_t) - \alpha_t \gamma \phi(s_{t+1}, a_{t+1})(\phi(s_t, a_t)^T w_t) \\
w_{t+1} &= w_t + \beta_t(\delta_t - \phi(s_t, a_t)^T w_t)\phi(s_t, a_t) \\
\delta_t &= r_{t+1} + \gamma Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t)
\end{aligned}$$

Convergence Bound:

$$\begin{aligned}
\text{V-function:} \quad & ||V_{\theta^*} - V_\pi|| \leq \frac{1}{\sqrt{1-\gamma^2}} \min_\theta ||V_\theta - V_\pi|| \\
\text{Q-function:} \quad & ||Q_{\theta^*} - Q_\pi|| \leq \frac{1}{\sqrt{1-\gamma^2}} \min_\theta ||Q_\theta - Q_\pi||
\end{aligned}$$

4.1.1.2 Least-Squares Methods

1. Least-Squares Temporal Difference (LSTD):

LSTD directly computes the fixed point of TD learning without using incremental updates.

V-function Approximation:

$$\theta = A^{-1}b$$
$$A = \sum_{t=0}^{n-1} \phi(s_t)(\phi(s_t) - \gamma\phi(s_{t+1}))^T$$
$$b = \sum_{t=0}^{n-1} \phi(s_t)r_{t+1}$$

Q-function Approximation (LSTD-Q):

$$\theta = A^{-1}b$$
$$A = \sum_{t=0}^{n-1} \phi(s_t, a_t)(\phi(s_t, a_t) - \gamma\phi(s_{t+1}, a_{t+1}))^T$$
$$b = \sum_{t=0}^{n-1} \phi(s_t, a_t)r_{t+1}$$

Algorithm 7 LSTD-Q Algorithm with Non-Terminal State Handling

Input: Learning rate α, discount factor γ, state space $\mathcal{S}$, action space $\mathcal{A}$, feature vectors $\phi(s, a)$, and a dataset of transitions $\{(s_t, a_t, r_{t+1}, s_{t+1})\}$

Initialize:

- $\theta \in \mathbb{R}^d$ (parameter vector) arbitrarily
- $A \in \mathbb{R}^{d \times d}$ (matrix), initialized to 0
- $b \in \mathbb{R}^d$ (vector), initialized to 0

for each episode in the dataset **do**
  Initialize state s_0 and action a_0
  for each step of the episode **do**
    Observe reward r_t and next state s_{t+1}
    Compute feature vectors $\phi(s_t, a_t) \in \mathbb{R}^d$ and $\phi(s_{t+1}, a_{t+1}) \in \mathbb{R}^d$
    if state s_{t+1} is non-terminal **then**
      Update matrix $A \in \mathbb{R}^{d \times d}$ and vector $b \in \mathbb{R}^d$:

$$A \leftarrow A + \phi(s_t, a_t)(\phi(s_t, a_t) - \gamma\phi(s_{t+1}, a_{t+1}))^T$$

$$b \leftarrow b + \phi(s_t, a_t) r_{t+1}$$

    else
      (Optional) If state s_{t+1} is terminal, no future rewards can be considered:

$$A \leftarrow A + \phi(s_t, a_t)(\phi(s_t, a_t))^T$$

$$b \leftarrow b + \phi(s_t, a_t) r_{t+1}$$

    end if
  end for
end for
Output:

$$\theta = A^{-1} b$$

Convergence Bound: (for a finite sample of size n)

$$\text{V-function:} \quad \mathbb{E}[||V_{\theta_n} - V_\pi||_D] \leq O\left(\frac{d}{\sqrt{n}}\right)$$

$$\text{Q-function:} \quad \mathbb{E}[||Q_{\theta_n} - Q_\pi||_D] \leq O\left(\frac{d}{\sqrt{n}}\right)$$

where d is the dimension of the feature space.

2. Least-Squares Bellman Residual Minimization (LSBR):

LSBR minimizes the squared Bellman residual using least-squares techniques.

V-function Approximation:

$$\theta = \left(\sum_{t=0}^{n-1} \psi_t \psi_t^T\right)^{-1} \left(\sum_{t=0}^{n-1} \psi_t r_{t+1}\right)$$
$$\psi_t = \phi(s_t) - \gamma\phi(s_{t+1})$$

where $V(s|\theta) = \phi(s)^\top\theta$, $\phi(s)$ is the feature vector for the state s, and θ is the parameter vector learned during training.

Q-function Approximation:

$$\theta = \left(\sum_{t=0}^{n-1} \psi_t \psi_t^T\right)^{-1} \left(\sum_{t=0}^{n-1} \psi_t r_{t+1}\right)$$
$$\psi_t = \phi(s_t, a_t) - \gamma\phi(s_{t+1}, a_{t+1})$$

where $Q(s, a|\theta) = \phi(s, a)^\top\theta$, $\phi(s, a)$ is the feature vector for the state-action pair (s, a), and θ is the parameter vector learned during training.

Convergence Bound:

$$\text{V-function:} \quad ||V_{\theta^*} - V_\pi||_\infty \leq \frac{2\gamma}{(1-\gamma)^2} \min_\theta ||V_\theta - T^\pi V_\theta||_\infty$$
$$\text{Q-function:} \quad ||Q_{\theta^*} - Q_\pi||_\infty \leq \frac{2\gamma}{(1-\gamma)^2} \min_\theta ||Q_\theta - T^\pi Q_\theta||_\infty$$

where T^π is the Bellman operator for policy π.

4.1.1.3 Residual Methods

1. Bellman Residual Minimization (BRM):

BRM directly minimizes the squared difference between the value function and its Bellman backup.

V-function Approximation:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \alpha_t \nabla_\theta (V_\theta(s_t) - r_{t+1} - \gamma V_\theta(s_{t+1}))^2 \\ &= \theta_t - \alpha_t \cdot 2(V_\theta(s_t) - r_{t+1} - \gamma V_\theta(s_{t+1})) \cdot (\nabla_\theta V_\theta(s_t) - \gamma \nabla_\theta V_\theta(s_{t+1}))\end{aligned}$$

Q-function Approximation:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \alpha_t \nabla_\theta (Q_\theta(s_t, a_t) - r_{t+1} - \gamma Q_\theta(s_{t+1}, a_{t+1}))^2 \\ &= \theta_t - \alpha_t \cdot 2(Q_\theta(s_t, a_t) - r_{t+1} - \gamma Q_\theta(s_{t+1}, a_{t+1})) \cdot (\nabla_\theta Q_\theta(s_t, a_t) - \gamma \nabla_\theta Q_\theta(s_{t+1}, a_{t+1}))\end{aligned}$$

Convergence Bound:

$$\text{V-function:} \quad ||V_{\theta^*} - V_\pi||_\infty \leq \frac{1}{1-\gamma} \min_\theta ||V_\theta - T^\pi V_\theta||_\infty$$
$$\text{Q-function:} \quad ||Q_{\theta^*} - Q_\pi||_\infty \leq \frac{1}{1-\gamma} \min_\theta ||Q_\theta - T^\pi Q_\theta||_\infty$$

4.1.1.4 Model-Free Monte Carlo Methods

1. Monte Carlo Regression (MC):

Monte Carlo regression directly fits the value function to observed returns.

V-function Approximation:

$$\theta_{t+1} = \theta_t + \alpha_t(G_t - V_\theta(s_t))\nabla_\theta V_\theta(s_t)$$

where G_t is the return from state s_t until the end of the episode.

Q-function Approximation:

$$\theta_{t+1} = \theta_t + \alpha_t(G_t - Q_\theta(s_t, a_t))\nabla_\theta Q_\theta(s_t, a_t)$$

where G_t is the return following the action a_t in state s_t.

Convergence Bound:

$$\text{V-function:} \quad \mathbb{E}[||V_{\theta^*} - V_\pi||_D] \leq O\left(\frac{1}{\sqrt{n}}\right)$$

$$\text{Q-function:} \quad \mathbb{E}[||Q_{\theta^*} - Q_\pi||_D] \leq O\left(\frac{1}{\sqrt{n}}\right)$$

where n is the number of episodes.

4.1.1.5 Linear Programming Approaches

1. Approximate Linear Programming (ALP):

Approximate the value function using a parametric linear function. The value function is represented as $\tilde{V}(s|\theta) = \phi(s)^T\theta$, where $\phi(s)$ is a feature vector that captures the state's characteristics and θ is a parameter vector that we optimize. ALP seeks to minimize the discrepancy between the approximated value function and the Bellman operator's value.

V-function Formulation:

$$\min_\theta \sum_{s\in\mathcal{S}} \mu(s)\phi(s)^T\theta$$

$$\text{s.t. } \phi(s)^T\theta \geq \sum_{s'\in\mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma\phi(s')^T\theta], \ \forall s \in \mathcal{S}, a \in \mathcal{A}$$

Q-function Formulation:

$$\min_\theta \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \mu(s,a)\phi(s,a)^T\theta$$

$$\text{s.t. } \phi(s,a)^T\theta \geq \sum_{s'\in\mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s')\phi(s',a')^T\theta], \ \forall s \in \mathcal{S}, a \in \mathcal{A}$$

Convergence Bound:

$$\text{V-function:} \quad ||V_{\theta^*} - V_\pi||_{1,\mu} \leq \frac{2}{1-\gamma}\min_\theta ||V_\theta - V_\pi||_\infty$$

$$\text{Q-function:} \quad ||Q_{\theta^*} - Q_\pi||_{1,\mu} \leq \frac{2}{1-\gamma}\min_\theta ||Q_\theta - Q_\pi||_\infty$$

where $||.||_{1,\mu}$ is the weighted L1 norm with respect to the state relevance weights μ used in the objective function of the ALP formulation.

4.1.2 Key Differences Between V-function and Q-function Approximation

- **Representation:**
  - V-function: Parameters represent $V_\pi(s)$ with features $\phi(s)$ depending only on states.
  - Q-function: Parameters represent $Q_\pi(s, a)$ with features $\phi(s, a)$ depending on both states and actions.
- **Sample Complexity:**
  - V-function: Requires fewer parameters (lower dimensionality), leading to faster learning.
  - Q-function: Requires more parameters to represent action-dependent values, potentially needing more samples.
- **Application:**
  - V-function: Primarily used for policy evaluation when the policy is already defined.
  - Q-function: Can be used directly for control (policy improvement) by selecting $a = \arg\max_a Q(s, a)$.
- **Policy Representation:**
  - V-function: Requires model knowledge or additional policy representation to select actions.
  - Q-function: The policy can be directly derived from Q-values, making it suitable for model-free control.

4.1.3 Comparative Analysis of the Methods

These methods differ in their trade-offs between computational complexity, sample efficiency, stability, and asymptotic performance:

- **TD Methods** (TD(0), TD(λ)): Low computational cost per update but may require many samples to converge.
- **Least-Squares Methods** (LSTD, LSBR): More sample-efficient but higher computational cost per update.
- **Residual Methods** (BRM): Can be biased without double sampling but often more stable with function approximation.
- **Monte Carlo Methods**: Unbiased but high variance, requiring complete episodes.
- **Linear Programming Approaches** (ALP): Can incorporate constraints naturally but difficult to scale to large problems.

The choice between these methods depends on factors such as available computational resources, amount of data, desired accuracy, whether the task is on-policy or off-policy, and whether the goal is policy evaluation or control.

4.2 Approximate Value Iteration

Approximate Value Iteration (AVI) is an approach to solving large-scale reinforcement learning problems where the state space is too large for exact value iteration. In AVI, we approximate the value function using a parametric form $\tilde{V}(s|\theta)$, where θ represents the parameters of the approximation. This allows for the estimation of the value function without explicitly considering every state. Similarly, AVI can also be applied to the action-value function, where we approximate $\tilde{Q}(s, a|\theta)$ instead of $\tilde{V}(s|\theta)$, enabling the estimation of the action-value function for large-scale problems.

Given a set of state-action pairs $(s_1, a_1), (s_2, a_2), \ldots, (s_n, a_n)$, the feature matrix Φ for the estimation of Q is defined as:

$$\Phi = \begin{bmatrix} \phi(s_1, a_1)^T \\ \phi(s_2, a_2)^T \\ \vdots \\ \phi(s_n, a_n)^T \end{bmatrix}$$

where $\phi(s, a)$ is the feature vector corresponding to the state-action pair (s, a), and each row of the matrix is the transpose of the feature vector for that specific state-action pair.

Algorithm 8 Approximate Value Iteration Algorithm (V-function)

1: **Input:** Transition probabilities $P(s'|s, a)$, reward function $R(s, a, s')$, discount factor γ, initial value function $\tilde{V}_0$, features $\phi(s)$, tolerance ϵ
2: **Initialize:** Parameters θ_0 for linear function approximation
3: **repeat**
4: **For each state** $s \in \mathcal{S}$:
5: Compute the Bellman backup for state s:

$$T_k(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s, a)[R(s, a, s') + \gamma \tilde{V}(s'|\theta_k)]$$

6: Update the parameter vector θ_{k+1} by solving:

$$\theta_{k+1} = (\Phi^T \Phi)^{-1} \Phi^T T_k$$

7: **until** the change in θ is smaller than ϵ (i.e., $\|\theta_{k+1} - \theta_k\|$ is below a threshold)
8: **Output:** Approximate value function $\tilde{V}(s|\theta^*)$

Algorithm 9 Approximate Value Iteration Algorithm (Q-function)

Input: Transition probabilities $P(s'|s,a)$, reward function $R(s,a,s')$, discount factor γ, initial action-value function $\tilde{Q}_0$, features $\phi(s,a)$, tolerance ϵ
Initialize: Parameters θ_0 for linear function approximation
repeat
  For each state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$:
    Compute the Bellman backup for state-action pair (s,a):

$$T_k(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a' \in A} \tilde{Q}(s',a'|\theta_k)$$

    Update the parameter vector θ_{k+1} by solving:

$$\theta_{k+1} = (\Phi^T \Phi)^{-1} \Phi^T T_k$$

until the change in θ is smaller than ϵ (i.e., $\|\theta_{k+1} - \theta_k\|$ is below a threshold)
Output: Approximate action-value function $\tilde{Q}(s,a|\theta^*)$

4.2.1 Approximate Value Iteration Methods

For each method, we provide update rules and convergence bounds for either the state value function (V-function) or the state-action value function (Q-function). We choose to focus on one of these functions depending on the specific method, and do not apply the update rules to both simultaneously. The choice between using $V(s)$ or $Q(s,a)$ depends on the underlying approach and objectives of the method, as some methods are designed to optimize value functions, while others target action-value functions.

4.2.1.1 Fitted Value Iteration

1. Fitted Value Iteration (FVI):

Fitted Value Iteration applies supervised learning to approximate the Bellman optimality backup in each iteration, using a batch of transition samples.

V-function Approximation:

$$\text{Target Computation:} \quad y_i = \max_{a \in A} [r(s_i,a) + \gamma \mathbb{E}_{s' \sim P(s'|s_i,a)}[V_{\theta_t}(s')]]$$

$$\text{Parameter Update:} \quad \theta_{t+1} = \arg\min_{\theta \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} (V_\theta(s_i) - y_i)^2$$

Q-function Approximation:

$$\text{Target Computation:} \quad y_i = r(s_i,a_i) + \gamma \max_{a' \in A} Q_{\theta_t}(s_i', a')$$

$$\text{Parameter Update:} \quad \theta_{t+1} = \arg\min_{\theta \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} (Q_\theta(s_i,a_i) - y_i)^2$$

Convergence Bound: (for certain function approximation classes)

$$\text{V-function:} \quad ||V_{\theta^*} - V^*||_\infty \leq \frac{2\gamma\varepsilon}{(1-\gamma)^2} + \frac{2\gamma}{1-\gamma} \min_\theta ||V_\theta - V^*||_\infty$$

$$\text{Q-function:} \quad ||Q_{\theta^*} - Q^*||_\infty \leq \frac{2\gamma\varepsilon}{(1-\gamma)^2} + \frac{2\gamma}{1-\gamma} \min_\theta ||Q_\theta - Q^*||_\infty$$

where ε is the approximation error at each iteration.

3. Fitted Q-Iteration (FQI):

FQI applies iterative supervised learning to a fixed batch of samples to approximate the optimal Q-function.

Q-function Approximation:

$$\text{Target Computation:} \quad y_i^{(t)} = r_i + \gamma \max_{a' \in A} Q_{\theta_t}(s_i', a')$$

$$\text{Parameter Update:} \quad \theta_{t+1} = \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} (Q_\theta(s_i, a_i) - y_i^{(t)})^2$$

Convergence Bound:

$$\text{Q-function:} \quad ||Q_{\theta^*} - Q^*||_\rho \leq \frac{2\gamma\varepsilon_{\text{FQI}}}{(1-\gamma)^2} + \frac{C_{\rho,\nu}\gamma}{1-\gamma} \min_\theta ||Q_\theta - Q^*||_\nu$$

where ε_{FQI} is the regression error, ρ and ν are distributions over state-action pairs, and $C_{\rho,\nu}$ is a concentration coefficient.

4.2.1.2 Approximate Modified Policy Iteration

1. Approximate Modified Policy Iteration (AMPI):

AMPI generalizes both value and policy iteration by applying m policy evaluation steps followed by a policy improvement step.

V-function Approximation:

$$\text{Policy Evaluation:} \quad V_{\theta_{t+1}} = \arg\min_\theta ||V_\theta - (T_{\pi_t})^m V_{\theta_t}||^2$$

$$\text{Policy Improvement:} \quad \pi_{t+1}(s) = \arg\max_{a \in A} [r(s,a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)}[V_{\theta_{t+1}}(s')]]$$

Q-function Approximation:

$$\text{Policy Evaluation:} \quad Q_{\theta_{t+1}} = \arg\min_\theta ||Q_\theta - (T_{\pi_t})^m Q_{\theta_t}||^2$$

$$\text{Policy Improvement:} \quad \pi_{t+1}(s) = \arg\max_{a \in A} Q_{\theta_{t+1}}(s,a)$$

Convergence Bound:

$$\text{V-function:} \quad ||V_{\theta^*} - V^*||_\infty \leq \frac{2\gamma(\varepsilon_{\text{eval}} + \gamma^m \varepsilon_{\text{aprx}})}{(1-\gamma)(1-\gamma^m)}$$

$$\text{Q-function:} \quad ||Q_{\theta^*} - Q^*||_\infty \leq \frac{2\gamma(\varepsilon_{\text{eval}} + \gamma^m \varepsilon_{\text{aprx}})}{(1-\gamma)(1-\gamma^m)}$$

where $\varepsilon_{\text{eval}}$ is the policy evaluation error and $\varepsilon_{\text{aprx}}$ is the approximation error.

2. Classification-based Modified Policy Iteration (CBMPI):

CBMPI uses a classifier to represent the policy and regression to approximate the value function.

V-function and Policy Approximation:

$$\begin{aligned} \text{Value Estimation:} \quad & V_{\theta_{t+1}} = \arg\min_{\theta} ||V_\theta - (T_{\pi_t})^m V_{\theta_t}||^2 \\ \text{Policy Improvement:} \quad & \pi_{t+1} = \arg\min_{\pi \in \Pi} \mathbb{E}_{s\sim\rho}[\max_{a\in A} Q^{\pi_t}(s,a) - Q^{\pi_t}(s,\pi(s))] \end{aligned}$$

Convergence Bound:

$$||V^{\pi_{t+1}} - V^*||_{1,\rho} \leq ||V^{\pi_t} - V^*||_{1,\rho} - \alpha + 2\frac{\gamma}{1-\gamma}(\varepsilon_{\text{class}} + \varepsilon_{\text{eval}})$$

where $\varepsilon_{\text{class}}$ is the classification error, $\varepsilon_{\text{eval}}$ is the evaluation error, and α is the policy improvement factor.

5 Online Learning

Online learning methods learn directly from experience without a complete model of the environment. These methods update the agent's knowledge incrementally based on real-time feedback, such as rewards and next states, rather than relying on predefined transition models.

This approach is useful when the environment's dynamics are unknown or too complex to model. Algorithms such as Q-learning, SARSA, and Actor-Critic use online learning to iteratively improve the agent's action-value function $Q(s, a)$, or both the policy and value function in the case of Actor-Critic, through direct interaction with the environment.

Online learning relies on:

- Incremental updates after each action or episode.
- Exploration and exploitation of the environment.
- Continuous adaptation to new data.

5.1 SARSA

5.1.1 Presentation

SARSA (State-Action-Reward-State-Action) is an **on-policy** Temporal Difference (TD) control algorithm used to estimate the action-value function $Q(s, a)$. It updates the Q-function based on transitions (s, a, r, s', a'), where the next action a' is chosen according to the current policy (typically ϵ-greedy with respect to Q).

The update rule for SARSA is given by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right]$$

where:

- α is the learning rate
- r is the immediate reward
- γ is the discount factor
- $Q(s', a')$ is the estimated value of the next state-action pair
- a' is the action chosen in the next state s'

5.1.2 Convergence Theorem

SARSA converges to Q^π for a fixed policy π with probability 1 if the following conditions hold:

1. The state-action space is finite.
2. The learning rate satisfies $\sum_t \alpha_t(s, a) = \infty$ and $\sum_t \alpha_t^2(s, a) < \infty$.
3. The variance of rewards is bounded.
4. The discount factor $\gamma < 1$.

5.1.3 SARSA Algorithm

Algorithm 10 SARSA Algorithm

Input: Learning rate α, discount factor γ, exploration policy π, state space $\mathcal{S}$, action space $\mathcal{A}$
Initialize: Q-values $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
for each episode **do**
  Initialize state s_0
  Choose action a_0 based on policy π (e.g., ϵ-greedy)
  for each step $i = 0, 1, 2, \ldots$ of the episode **do**
    Take action a_i, observe reward r_i and next state s_{i+1}
    Choose next action a_{i+1} based on policy π (e.g., ϵ-greedy)
    Update the Q-value:

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \left[r_i + \gamma Q(s_{i+1}, a_{i+1}) - Q(s_i, a_i) \right]$$

    Set $s_i \leftarrow s_{i+1}$, $a_i \leftarrow a_{i+1}$
  end for
end for
Output: Action-value function $Q(s, a)$

5.2 Q-Learning

5.2.1 Presentation

Q-Learning is an **off-policy** Temporal Difference (TD) control algorithm used to learn the optimal action-value function $Q^*(s, a)$, which represents the maximum expected return achievable from state s and action a. It updates the Q-values based on transitions (s, a, r, s') using the following update rule:

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

where:

- α is the learning rate
- r is the immediate reward
- γ is the discount factor
- $\max_{a'} Q(s', a')$ is the estimated maximum future value in the next state s'

5.2.2 Convergence Theorem

Define the Bellman operator:

$$T^*(Q)(s, a) = \sum_{s'} P(s'|s, a) \left[R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q(s', a') \right]$$

The Q-learning update can be written as:

$$Q_{t+1}(s_t, a_t) = (1 - \alpha_t)Q_t(s_t, a_t) + \alpha_t \left[r_t + \gamma \max_{a' \in \mathcal{A}} Q_t(s_{t+1}, a') \right]$$

This is a stochastic approximation of the iteration $Q_{t+1} = T^*(Q_t)$. Under suitable conditions on the learning rate, and by the properties of T^* as a contraction mapping, Q-learning converges to the optimal Q-function Q^* with probability 1.

5.2.3 Q-Learning Algorithm

Algorithm 11 Q-Learning Algorithm

1: **Input:** Learning rate α, discount factor γ, exploration policy π, state space $\mathcal{S}$, action space $\mathcal{A}$
2: **Initialize:** Q-values $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
3: **for** each episode **do**
4: Initialize state s_0
5: **for** each step $i = 0, 1, 2, \ldots$ of the episode **do**
6: Choose action a_i based on ϵ-greedy policy with respect to Q
7: Take action a_i, observe reward r_i and next state s_{i+1}
8: Update the Q-value:

$$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \left[r_i + \gamma \max_{a' \in \mathcal{A}} Q(s_{i+1}, a') - Q(s_i, a_i) \right]$$

9: Set $s_i \leftarrow s_{i+1}$
10: **end for**
11: **end for**
12: **Output:** Optimal Q-function $Q^*(s, a)$

5.3 Comparison of SARSA and Q-Learning

- **SARSA**: **on-policy** algorithm:
  - It updates the action-value function based on the **action actually taken** by the current behavior policy (e.g., ϵ-greedy).
  - SARSA learns the value of the policy being followed (including exploratory actions).
- **Q-learning**: **off-policy** algorithm:
  - It updates the action-value function assuming the agent will act **greedily** in the next state, regardless of how it actually behaves.
  - Q-learning learns the value of the optimal policy, regardless of the agent's current behavior.

On-policy vs. Off-policy:

- **On-policy learning** evaluates and improves the **same policy** that the agent is currently using to make decisions.
- **Off-policy learning** evaluates or improves a **different (target) policy** than the one used to generate the data — typically aiming directly at the optimal policy.

Algorithm	Type	Update Rule
SARSA	On-policy	$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \left[r_i + \gamma Q(s_{i+1}, a_{i+1}) - Q(s_i, a_i) \right]$
Q-learning	Off-policy	$Q(s_i, a_i) \leftarrow Q(s_i, a_i) + \alpha \left[r_i + \gamma \max_{a'} Q(s_{i+1}, a') - Q(s_i, a_i) \right]$

5.4 Exploration-Exploitation Dilemma

The exploration-exploitation dilemma is a fundamental challenge in Reinforcement Learning. The agent needs to balance between two competing objectives:

- **Exploitation**: Selecting the action that is known to yield the highest reward based on current knowledge.
- **Exploration**: Trying new actions to discover potentially better long-term rewards.

The central question is: *When should the agent exploit its current knowledge, and when should it explore new possibilities?*

5.4.1 ϵ-Greedy Policy

One of the simplest strategies to address this dilemma is the ϵ-greedy policy, which strikes a balance between exploration and exploitation. The idea is to choose actions based on the current estimates of their values, but occasionally select random actions to explore new possibilities.

- With probability ϵ, select a random action (exploration).
- With probability $1 - \epsilon$, select the action that maximizes the estimated value (exploitation).

This approach ensures that, over time, the agent will explore enough of the action space to improve its estimates, but also exploit its knowledge to maximize rewards.

5.4.2 Softmax Exploration Strategy

An alternative to the ϵ-greedy approach is the **softmax** method, which selects actions probabilistically based on their estimated values. The probability of selecting an action a at time t is given by the softmax function:

$$P(a_t = a) = \frac{e^{Q_t(a)/\tau}}{\sum_{a' \in A} e^{Q_t(a')/\tau}}$$

where:

- $Q_t(a)$ is the estimated value of action a at time t,

- τ is a temperature parameter that controls the randomness of the selection. A high τ leads to more exploration (more random selection), while a low τ favors exploitation (greedy selection).

The softmax approach allows the agent to explore all actions probabilistically based on their estimated values, which avoids the abrupt transitions between exploration and exploitation seen in ϵ-greedy methods.

5.4.3 Upper Confidence Bound (UCB)

Another popular method for balancing exploration and exploitation is the Upper Confidence Bound (UCB) algorithm. In this approach, the agent selects the action that maximizes not only the current estimated value but also an upper confidence bound that encourages exploration of less frequently chosen actions.

The action a_t at time t is selected as:

$$a_t = \arg\max_{a \in \mathcal{A}} \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

where:

- $Q_t(a)$ is the estimated value of action a at time t,
- $N_t(a)$ is the number of times action a has been selected,
- c is a constant that controls the degree of exploration.

The second term, $\sqrt{\frac{\ln t}{N_t(a)}}$, increases the exploration of actions that have been selected less frequently, while the first term, $Q_t(a)$, drives the agent towards exploiting actions with higher value estimates. This strategy balances exploration and exploitation in a more sophisticated way than ϵ-greedy.

5.5 Actor-Critic Methods

Actor-Critic methods are reinforcement learning algorithms that combine value-based and policy-based approaches. These methods consist of two main components:

- **Actor**: A policy network that selects actions based on the current state, usually parameterized by a set of parameters θ.
- **Critic**: A value function network that evaluates the actions taken by the actor, estimating either the state value $V(s)$ or the action-value function $Q(s, a)$.

The critic provides feedback to the actor by estimating the advantage of the current action, guiding the actor to improve its policy. Actor-Critic methods are typically online algorithms, updating both the policy and value function in real-time as the agent interacts with the environment. These methods can handle large or continuous state and action spaces using function approximation, such as neural networks.

By combining the strengths of value-based and policy-based methods, Actor-Critic methods enable more stable learning and are particularly effective in complex environments.

6 Deep Q-Learning

Deep Q-Learning extends traditional Q-learning by using deep neural networks to approximate the Q-value function, which estimates the expected return for each state-action pair. This allows Deep Q-Learning to handle high-dimensional state spaces, such as images or complex sensory data, where traditional Q-learning would struggle due to the curse of dimensionality. In traditional Q-learning, a Q-table stores the value of each state-action pair. The table grows as the product of the number of states and actions. In high-dimensional environments (e.g., image data or continuous variables), the number of state-action pairs increases dramatically, making it computationally infeasible to store and update Q-values for all possible pairs.

The neural network learns to predict Q-values, and the agent updates its policy based on these predictions, enabling it to perform reinforcement learning in environments with large or continuous state spaces.

6.1 Deep Q-Network (DQN)

The Deep Q-Network (DQN) is a combination of Q-learning and deep learning techniques that approximates the Q-function using a neural network:

$$Q(s, a|\theta) \approx Q^*(s, a)$$

6.1.1 Methodology

1. **Initialize the Q-Network and Target Network**

   - Initialize the Q-network with parameters θ, which approximates the Q-value function:
     $$Q(s, a|\theta)$$
   - Initialize the target network with parameters θ^- (a copy of the Q-network). The target network is updated less frequently than the Q-network to improve stability during training.

2. **Experience Replay**

   - Initialize a replay buffer D to store experiences (state, action, reward, next state) transitions.
   - At each time step, store the experience tuple (s, a, r, s') in the buffer D.

3. **Action Selection (Exploration-Exploitation) and Experience Collection**

   - Select an action a for the current state s using an ϵ-greedy policy with respect to the Q-network:
     $$a = \arg\max_{a' \in \mathcal{A}} Q(s, a'|\theta)$$
     where with probability ϵ, a random action is chosen (exploration), and with probability $1 - \epsilon$, the action that maximizes the Q-value is chosen (exploitation).

- Execute the action, observe the reward r, and next state s', and store the transition (s, a, r, s') in the experience replay buffer D.

4. **Sample a Batch from the Replay Buffer**
   - Randomly sample a mini-batch of transitions (s, a, r, s') from the experience replay buffer D.
5. **Compute the Target Q-value**
   - Compute the target Q-value y for each sampled transition:
   $$y = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'|\theta^-)$$
   where γ is the discount factor, and θ^- are the parameters of the target network.
6. **Update the Q-Network**
   - Update the parameters θ of the Q-network by minimizing the loss function:
   $$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\left(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'|\theta^-) - Q(s, a|\theta)\right)^2\right]$$
   using stochastic gradient descent (SGD).
7. **Periodically Update the Target Network**
   - Every N steps, update the target network parameters θ^- by copying the parameters of the Q-network:
   $$\theta^- \leftarrow \theta$$
8. **Repeat the Process**
   - Repeat the process for each episode, continuously improving the Q-network and its ability to select actions based on the learned Q-values.
9. **Output:**
   - The Q-network parameters θ after training, which approximate the optimal Q-function $Q^*(s, a)$.

6.1.2 Specificities of DQN

1. **Experience Replay**: Storing and randomly sampling transitions to break correlations in the observation sequence
2. **Target Network**: Using a separate network for generating targets to stabilize training

6.2 Double DQN

Double DQN addresses the overestimation bias in DQN by decoupling action selection and evaluation:

$$y_t = r_t + \gamma\, Q(s_{t+1},\, \arg\max_{a \in \mathcal{A}} Q(s_{t+1}, a|\theta_t)\,|\,\theta_t^-)$$

6.3 Dueling DQN

Dueling DQN decomposes the Q-function into state value and advantage functions:

$$Q(s,a|\theta,\alpha,\beta) = V(s|\theta,\beta) + \left(A(s,a|\theta,\alpha) - \frac{1}{|A|} \sum_{a' \in \mathcal{A}} A(s,a'|\theta,\alpha) \right)$$

where $V(s|\theta,\beta)$ is the state value function and $A(s,a|\theta,\alpha)$ is the advantage function.