Reinforcement Learning

alexandre.carlhammar

January 2026

Reinforcement Learning

Reinforcement Learning is the study of sequential decision making under uncertainty in which an agent interacts with an environment over time and learns how to act so as to maximize long term cumulative reward.

Unlike supervised learning the agent does not receive labeled examples of correct behavior and unlike unsupervised learning there is an explicit performance objective defined through interaction. The data observed by the agent depends on its own actions which creates a feedback loop between learning and control.

The defining characteristics of reinforcement learning are that actions influence future observations rewards may be delayed over many time steps and optimal behavior is defined with respect to long term outcomes rather than immediate reward.

At each discrete time step the agent observes the current state of the environment selects an action receives a scalar reward and transitions to a new state. This interaction continues either indefinitely or until a terminal state is reached.

Objective of Reinforcement Learning

The goal of reinforcement learning is to find a policy that maximizes the expected cumulative reward obtained through interaction with the environment. To formalize this objective we first define the notion of return.

In the infinite horizon discounted setting the return from time step t onward is defined as the discounted sum of all future rewards:

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

The discount factor satisfies $0 \leq \gamma < 1$ and serves three purposes. It ensures convergence of the infinite sum it encodes a preference for rewards received

sooner and it induces an effective planning horizon.

A policy specifies how actions are selected as a function of the current state. Let S_0 denote the initial state drawn from an initial distribution ρ_0. The reinforcement learning objective is then defined as the maximization of expected return:

$$\pi^\star \in \arg\max_\pi E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 \sim \rho_0 \right] \tag{2}$$

The expectation is taken over all randomness induced jointly by the policy and the environment dynamics including stochastic transitions and stochastic rewards.

Markov Decision Processes

To reason rigorously about reinforcement learning problems the environment is modeled as a Markov Decision Process. This model captures both the dynamics of the environment and the effect of the agent's actions.

The Markov Property

The fundamental structural assumption underlying Markov Decision Processes is the Markov property. It states that the future evolution of the system depends on the past only through the current state and action.

Formally the Markov property is expressed as:

$$P(S_{t+1}, R_{t+1} \mid S_0, A_0, \ldots, S_t, A_t) = P(S_{t+1}, R_{t+1} \mid S_t, A_t) \tag{3}$$

This assumption implies that the state is a sufficient statistic for the history and that no additional predictive power is gained by conditioning on earlier states or actions once the current state is known.

Definition of a Markov Decision Process

A Markov Decision Process is defined as a tuple $(\mathcal{S}, \mathcal{A}, P, R, \rho_0, \gamma)$ consisting of the following components.

The state space $\mathcal{S}$ is the set of all possible states of the environment. It may be finite countable or continuous and is assumed to capture all information relevant for decision making.

The action space $\mathcal{A}$ is the set of actions available to the agent. It may be discrete or continuous and in some problems may depend on the current state.

The transition model P specifies the conditional distribution of the next state given the current state and action:

$$P(s' \mid s, a) = P(S_{t+1} = s' \mid S_t = s, A_t = a) \tag{4}$$

This formulation allows for stochastic dynamics. Deterministic dynamics arise as a special case where the distribution assigns probability one to a single next state.

Rewards may be stochastic. In full generality the reward received after a transition is modeled as a random variable whose distribution depends on the current state the action taken and the next state:

$$R(r \mid s, a, s') = P(R_{t+1} = r \mid S_t = s, A_t = a, S_{t+1} = s') \tag{5}$$

In many developments it is sufficient to work with the expected immediate reward associated with a transition. This expected reward is defined as:

$$r(s, a, s') = E[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'] \tag{6}$$

Deterministic rewards arise as the special case in which the reward distribution assigns probability one to a single value.

The initial state distribution ρ_0 specifies how episodes begin:

$$\rho_0(s) = P(S_0 = s) \tag{7}$$

The discount factor γ determines how future rewards are weighted relative to immediate rewards and ensures convergence of infinite horizon returns.

Policies

A policy defines the behavior of the agent. In the most general case a policy is a conditional probability distribution over actions given the current state:

$$\pi(a \mid s) = P(A_t = a \mid S_t = s) \tag{8}$$

Deterministic policies correspond to the special case in which this distribution assigns probability one to a single action for each state.

Partially Observable Markov Decision Processes

In many decision making problems the agent cannot directly observe the true state of the environment. Instead it receives observations that provide partial and possibly noisy information about that state. In such settings the observation alone does not satisfy the Markov property and a standard Markov Decision Process is no longer an adequate model.

A Partially Observable Markov Decision Process extends a Markov Decision Process by distinguishing between the true underlying state and the information available to the agent. The environment still evolves according to a Markov state process but this state is hidden and only indirectly observed.

Formally a POMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, P, r, O, \rho_0, \gamma)$ where $\mathcal{O}$ is the observation space and O is the observation model. The observation model specifies the conditional distribution of observations given the next state and the previous action:

$$O(o \mid s, a) = P(O_{t+1} = o \mid S_{t+1} = s, A_t = a) \tag{9}$$

Because the agent does not observe the true state it cannot condition its policy directly on S_t. Instead optimal decision making requires maintaining a belief

state which summarizes all past observations and actions. The belief state is defined as the posterior distribution over states given the interaction history:

$$b_t(s) = P(S_t = s \mid O_1, A_0, \ldots, O_t) \tag{10}$$

The belief state is a sufficient statistic for the history and evolves according to Bayesian filtering using the transition and observation models. Importantly the belief state itself satisfies the Markov property.

This allows a POMDP to be reformulated as a fully observable Markov Decision Process whose state space is the space of belief distributions. In this belief MDP the dynamics are induced by the Bayesian belief update and rewards are given by their expectation under the belief.

POMDPs therefore do not introduce an additional decision process but instead require a different state representation that restores the Markov property under partial observability. This generalization increases modeling power at the cost of significantly greater computational and statistical complexity.

Value Functions

The reinforcement learning objective is global and long term while decisions are made locally in time. Value functions bridge this gap by assigning numerical predictions of long term return to states and actions under a given policy.

State Value Function

The state value function of a policy quantifies the expected return obtained by starting from a given state and following the policy thereafter:

$$V^{\pi}(s) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} R_{t+1} \mid S_0 = s\right] \tag{11}$$

This expectation accounts for stochastic transitions stochastic rewards and stochastic action selection induced by the policy.

Action Value Function

The action value function refines the state value function by conditioning on the first action taken:

$$Q^{\pi}(s,a) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} R_{t+1} \mid S_0 = s, A_0 = a\right] \tag{12}$$

The action value function assigns a value to each state action pair and directly supports action comparison and policy improvement.

Relationship Between Value Functions

The state value function can be expressed as the expectation of the action value function under the policy's action distribution:

$$V^{\pi}(s) = E_{A\sim\pi(\cdot|s)}\left[Q^{\pi}(s,A)\right] = \sum_{a\in\mathcal{A}} \pi(a \mid s) Q^{\pi}(s,a) \tag{13}$$

Bellman Operators

This section introduces Bellman operators as the central mathematical objects that connect value functions to dynamic programming algorithms. The key results are that certain Bellman operators are contractions under the supremum norm which implies existence and uniqueness of fixed points and convergence of iterative backups to these fixed points.

Standing Assumptions

Unless stated otherwise all results in this section are derived under the following assumptions. The state space and action space are finite or countable. The transition dynamics and reward process may be stochastic. Policies may be stochastic. The horizon is infinite and rewards are discounted by a factor γ satisfying $0 \leq \gamma < 1$.

Under these assumptions value functions are bounded and the space of bounded functions equipped with the supremum norm is a complete metric space. This guarantees that contraction arguments and fixed point results apply.

Value Functions and Their Meaning

We recall that for any policy π the state value function V^π and action value function Q^π are defined as expected discounted returns under π. These functions summarize the long term consequences of decisions and are the central objects that Bellman operators act on.

In control problems we also define the optimal value function $V^\star$ which represents the maximum achievable expected discounted return from each state over all policies:

$$V^\star(s) = \sup_\pi V^\pi(s) \tag{14}$$

Similarly the optimal action value function $Q^\star$ represents the maximum achievable expected discounted return when the agent first takes action a in state s and then behaves optimally thereafter:

$$Q^\star(s,a) = \sup_\pi Q^\pi(s,a) \tag{15}$$

Bellman Expectation Operator and Bellman Optimality Operator

The Bellman expectation operator is an operator that maps any candidate value function V to a new function obtained by taking one step of expected reward

plus discounted value under a fixed policy π. This operator is denoted B_π and is defined by:

$$(B_\pi V)(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma V(s') \right) \tag{16}$$

This definition should be read as follows. Given a current guess V for long term value the operator produces an improved one step consistent estimate by replacing the unknown future return after one transition with the current guess V evaluated at the next state.

The Bellman optimality operator is the analogous operator for control which assumes the agent will choose the action that maximizes the right hand side. This operator is denoted B and is defined by:

$$(BV)(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma V(s') \right) \tag{17}$$

The difference between B_π and B is that B_π evaluates a fixed policy by averaging over actions according to π while B performs a greedy one step optimization over actions.

Bellman Equations as Fixed Point Equations

A Bellman equation is the statement that a value function is exactly consistent with its own one step lookahead. For a fixed policy π the Bellman expectation equation states that V^π is a fixed point of B_π:

$$V^\pi = B_\pi V^\pi \tag{18}$$

For optimal control the Bellman optimality equation states that $V^\star$ is a fixed point of B:

$$V^\star = B V^\star \tag{19}$$

These are not update rules but identities. They characterize the true value functions as fixed points of corresponding Bellman operators.

Bellman Expectation Equations

Value functions satisfy self consistency relations known as Bellman expectation equations. For the state value function this relation is:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma V^\pi(s') \right) \tag{20}$$

The corresponding Bellman equation for the action value function is:

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q^\pi(s', a') \right) \tag{21}$$

These equations are identities that uniquely characterize the value functions of a policy and form the theoretical foundation of dynamic programming temporal difference learning and control methods.

Bellman Backup and Iterative Application

A Bellman backup refers to the operation of replacing the value estimate at a state by the right hand side of the Bellman operator. Given a current estimate V_k one Bellman backup under policy π produces:

$$V_{k+1} = B_\pi V_k \tag{22}$$

A Bellman backup for control produces:

$$V_{k+1} = BV_k \tag{23}$$

Iteratively applying these backups corresponds to repeatedly enforcing one step consistency using the current estimate as a surrogate for future value. The key theoretical question is what these sequences converge to and why.

Norms and the Meaning of Contraction

To make precise statements about convergence we equip the space of bounded value functions with the supremum norm which measures the maximum absolute deviation over all states:

$$\|V\|_\infty = \max_{s \in \mathcal{S}} |V(s)| \tag{24}$$

An operator T is called a contraction with modulus γ under this norm if applying it to two functions brings them closer by at least a factor γ:

$$\|TV - TW\|_\infty \leq \gamma \|V - W\|_\infty \tag{25}$$

The meaning of this inequality is that repeated application of a contraction exponentially shrinks any initial error. If $\gamma < 1$ this implies convergence to a unique fixed point regardless of initialization.

Contraction of the Bellman Expectation Operator

We now show that B_π is a contraction under the supremum norm with modulus γ. Let V and W be any bounded functions. For any state s we have:

$$(B_\pi V)(s) - (B_\pi W)(s) = \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a) \gamma \left(V(s') - W(s')\right) \tag{26}$$

Taking absolute values and using that $\pi(\cdot \mid s)$ and $P(\cdot \mid s, a)$ are probability distributions yields:

$$|(B_\pi V)(s) - (B_\pi W)(s)| \leq \gamma \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a) \, |V(s') - W(s')| \quad (27)$$

Since $|V(s') - W(s')| \leq \|V - W\|_\infty$ for all s' we obtain:

$$|(B_\pi V)(s) - (B_\pi W)(s)| \leq \gamma \|V - W\|_\infty \quad (28)$$

Taking the maximum over s gives the contraction property:

$$\|B_\pi V - B_\pi W\|_\infty \leq \gamma \|V - W\|_\infty \quad (29)$$

Contraction of the Bellman Optimality Operator

The Bellman optimality operator B is also a contraction under the supremum norm with modulus γ. Let V and W be any bounded functions. For any state s define:

$$g_V(a) = \sum_{s'} P(s' \mid s, a) \, (r(s, a, s') + \gamma V(s')) \quad (30)$$

Then $(BV)(s) = \max_a g_V(a)$ and similarly for W. Using the inequality $|\max_a x_a - \max_a y_a| \leq \max_a |x_a - y_a|$ yields:

$$|(BV)(s) - (BW)(s)| \leq \max_{a \in \mathcal{A}} |g_V(a) - g_W(a)| \quad (31)$$

The reward terms cancel and the remaining difference is only through V and W giving:

$$|g_V(a) - g_W(a)| = \left| \sum_{s'} P(s' \mid s, a) \gamma \, (V(s') - W(s')) \right| \quad (32)$$

Applying the same bounding argument as before yields:

$$|g_V(a) - g_W(a)| \leq \gamma \|V - W\|_\infty \quad (33)$$

Therefore for every s we obtain:

$$|(BV)(s) - (BW)(s)| \leq \gamma \|V - W\|_\infty \quad (34)$$

Taking the maximum over s yields:

$$\|BV - BW\|_\infty \leq \gamma \|V - W\|_\infty \quad (35)$$

Fixed Point Existence Uniqueness and Convergence

Because B_π is a contraction on the complete metric space of bounded value functions under the supremum norm it has a unique fixed point. This fixed point is precisely V^π. Moreover repeated Bellman backups converge to that fixed point from any initialization.

Let V_0 be any bounded function and define $V_{k+1} = B_\pi V_k$. Then the contraction property implies the error shrinks geometrically:

$$\|V_k - V^\pi\|_\infty \leq \gamma^k \|V_0 - V^\pi\|_\infty \tag{36}$$

The same reasoning applies to the Bellman optimality operator B. It has a unique fixed point $V^\star$ and value iteration converges to $V^\star$ from any initialization.

Let V_0 be any bounded function and define $V_{k+1} = BV_k$. Then:

$$\|V_k - V^\star\|_\infty \leq \gamma^k \|V_0 - V^\star\|_\infty \tag{37}$$

Dynamic Programming

Policy Evaluation

Policy evaluation is the problem of computing the value function V^π of a fixed policy π. From the Bellman expectation equation this value function is characterized as the unique fixed point of the Bellman expectation operator B_π.

Because B_π is a contraction under the supremum norm repeated application of this operator converges to V^π from any initial bounded function. This leads to the iterative policy evaluation procedure defined by:

$$V_{k+1} = B_\pi V_k = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma V_k(s') \right) \tag{38}$$

Each iteration performs a Bellman backup that enforces one step consistency between immediate rewards and the current estimate of future value. Convergence is guaranteed and the error decreases geometrically with rate γ.

Value Iteration

Value iteration addresses the optimal control problem directly. Instead of evaluating a fixed policy it applies the Bellman optimality operator whose fixed point is the optimal value function $V^\star$.

Starting from an arbitrary bounded initialization value iteration repeatedly applies the Bellman optimality backup:

$$V_{k+1} = B V_k = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma V_k(s') \right) \tag{39}$$

A standard pseudocode description is given below. The update line is exactly the Bellman optimality backup applied synchronously across all states.

Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma V(s') \right)$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    End for
Until $\Delta < \varepsilon$

Because the Bellman optimality operator is also a contraction this sequence converges to the unique fixed point $V^\star$. Each iteration simultaneously performs policy evaluation and policy improvement by assuming that future decisions will be made optimally.

Once convergence is reached an optimal policy can be extracted by acting greedily with respect to the optimal one step lookahead defined by $V^\star$.

The greedy policy extraction step is:

$$\pi^\star(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma V^\star(s') \right) \tag{40}$$

Policy Iteration

Policy iteration separates the tasks of policy evaluation and policy improvement into two distinct steps. It exploits the fact that improving a policy with respect to its own value function yields a policy that is no worse. In this subsection we describe the exact policy iteration algorithm in which policy evaluation is carried out to convergence at each iteration.

Given a current policy π_k the policy evaluation step computes its value function exactly as the unique fixed point of the Bellman expectation operator associated with that policy:

$$V^{\pi_k} = B_{\pi_k} V^{\pi_k} = \sum_{a \in \mathcal{A}} \pi_k(a \mid s) \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma V^{\pi_k}(s') \right) \tag{41}$$

This step assumes that the Bellman expectation operator is applied repeatedly until convergence so that the resulting value function exactly represents the expected discounted return obtained by following π_k.

A standard pseudocode description of exact policy iteration is given below.

Initialize policy $\pi(s)$ arbitrarily for all $s \in \mathcal{S}$
Repeat
    Policy Evaluation
    Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$
    Repeat
        $\Delta \leftarrow 0$
        For each $s \in \mathcal{S}$ do
            $v \leftarrow V(s)$
            $V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma V(s') \right)$
            $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
        End for
    Until $\Delta < \varepsilon$

Policy Improvement
Policy stable $\leftarrow$ true
For each $s \in \mathcal{S}$ do
$a_{\text{old}} \leftarrow \pi(s)$
$\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma V(s')\right)$
If $a_{\text{old}} \neq \pi(s)$ then policy stable $\leftarrow$ false
End for
Until policy stable

Repeating these two steps produces a sequence of policies whose value functions are monotonically non decreasing. In the finite state and action setting exact policy iteration is guaranteed to converge in a finite number of iterations to an optimal policy.

This finite termination property follows from a simple counting argument. In the finite case the number of deterministic stationary policies is finite and equal to $|\mathcal{A}|^{|\mathcal{S}|}$. Since each policy improvement step yields a strictly better policy and previously visited policies cannot be revisited the algorithm must terminate after at most $|\mathcal{A}|^{|\mathcal{S}|}$ policy improvements.

Exact policy iteration can therefore be viewed as alternating between full convergence of the Bellman expectation operator for a fixed policy and a greedy policy improvement step derived from the Bellman optimality principle.

Addendum

Value iteration behaves fundamentally differently. It does not move through a finite set of policies and does not produce a discrete sequence of policy updates. Instead its convergence follows from the contraction property of the Bellman optimality operator which implies geometric decay of the value estimation error. As a result value iteration admits no finite step termination guarantee and its convergence is asymptotic.

Despite these differences in convergence behavior value iteration and exact policy iteration are aligned at the level of optimality. In the finite discounted setting both algorithms converge to the unique optimal value function $V^\star$ and any policy obtained by acting greedily with respect to this value function is optimal. When multiple optimal actions exist in a state the resulting optimal policy need not be unique and different algorithms may converge to different but equally optimal policies.

In continuous state or action spaces the number of policies is infinite and finite termination guarantees no longer apply. In such settings convergence is defined in terms of reaching a policy whose greedy improvement changes the value func-

tion by less than a prescribed tolerance.

Specializations Across Modeling Assumptions

The Bellman equations derived above admit different concrete forms depending on the modeling assumptions imposed on the environment and the policy. In this subsection we collect the most common specializations that arise in practice. These cases are not new algorithms but explicit instantiations of the same Bellman identities under different assumptions on the state space action space transition dynamics reward structure and policy class.

Case 1. Discrete state space and discrete action space with deterministic transitions deterministic rewards and a deterministic policy.

In this setting the next state is given by a deterministic function $s' = f(s, a)$ and the policy selects a single action $\mu(s)$ in each state. No expectations are required and the Bellman equations reduce to simple recursions:

$$V^{\pi}(s) = r(s, \mu(s), f(s, \mu(s))) + \gamma V^{\pi}(f(s, \mu(s))) \tag{42}$$

$$Q^{\pi}(s, a) = r(s, a, f(s, a)) + \gamma Q^{\pi}(f(s, a), \mu(f(s, a))) \tag{43}$$

Case 2. Discrete state space and discrete action space with stochastic transitions and rewards and a deterministic policy.

Here the policy still selects a single action in each state but uncertainty is present in the environment. Expectations over next states and rewards must therefore be retained:

$$V^{\pi}(s) = \sum_{s' \in \mathcal{S}} P(s' \mid s, \mu(s)) \left(r(s, \mu(s), s') + \gamma V^{\pi}(s') \right) \tag{44}$$

$$Q^{\pi}(s, a) = \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma Q^{\pi}(s', \mu(s')) \right) \tag{45}$$

Case 3. Discrete state space and discrete action space with stochastic transitions stochastic rewards and a stochastic policy.

This is the fully general tabular setting. Both the environment and the policy introduce randomness and the Bellman equations involve expectations over actions and next states:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma V^{\pi}(s') \right) \tag{46}$$

$$Q^{\pi}(s, a) = \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \left(r(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a' \mid s') Q^{\pi}(s', a') \right) \tag{47}$$

Case 4. Continuous state space and or continuous action space with stochastic transitions and rewards and a deterministic policy.

In this case sums over states are replaced by integrals with respect to the transition kernel while the expectation over actions collapses due to determinism of the policy:

$$V^{\pi}(s) = \int_{\mathcal{S}} \left(r(s, \mu(s), s') + \gamma V^{\pi}(s') \right) P(ds' \mid s, \mu(s)) \tag{48}$$

$$Q^{\pi}(s, a) = \int_{\mathcal{S}} \left(r(s, a, s') + \gamma Q^{\pi}(s', \mu(s')) \right) P(ds' \mid s, a) \tag{49}$$

Case 5. Continuous state space and or continuous action space with stochastic transitions stochastic rewards and a stochastic policy.

This is the most general formulation. Both the transition model and the policy are probability measures and all expectations are expressed as integrals:

$$V^{\pi}(s) = \int_{\mathcal{A}} \pi(a \mid s) \int_{\mathcal{S}} \left(r(s, a, s') + \gamma V^{\pi}(s') \right) P(ds' \mid s, a) \, da \tag{50}$$

$$Q^{\pi}(s, a) = \int_{\mathcal{S}} \left(r(s, a, s') + \gamma \int_{\mathcal{A}} Q^{\pi}(s', a') \pi(a' \mid s') \, da' \right) P(ds' \mid s, a) \tag{51}$$

In all cases the differences are purely notational and measure theoretic. The Bellman equations always express the same self consistency principle adapted to the modeling assumptions on the state space action space policy and environment dynamics.

Model-Free Policy Evaluation

In many reinforcement learning problems the agent can sample experience by interacting with the environment but does not know the transition dynamics or the reward mechanism in closed form. In such settings the task of policy evaluation becomes the problem of estimating V^π or Q^π using only data generated by interaction while following a fixed policy π.

Model free policy evaluation therefore replaces exact computation with statistical estimation and the central questions become how accurate the estimator is how much data it needs and how efficiently it can be updated as more data arrives.

A recurring idea in model free methods is bootstrapping. Bootstrapping refers to updating an estimate of a value function using other current estimates of the same value function rather than waiting for a complete return to be observed. A canonical bootstrapped target is the one step reward plus discounted estimated value of the next state:

$$\text{bootstrapped target} = R_{t+1} + \gamma V(S_{t+1}) \tag{52}$$

Estimators and Evaluation Criteria

Model free policy evaluation methods construct estimators from data. Abstractly let θ denote an unknown quantity of interest such as $V^\pi(s)$ for a fixed state s. An estimator $\hat{\theta}$ is a random variable computed from observed data and its quality is assessed through statistical and computational criteria.

A basic statistical decomposition uses bias variance and mean squared error. The bias of an estimator is defined as:

$$\text{Bias}_\theta(\hat{\theta}) = E[\hat{\theta}] - \theta \tag{53}$$

The variance of an estimator is defined as:

$$\text{Var}(\hat{\theta}) = E\left[\left(\hat{\theta} - E[\hat{\theta}]\right)^2\right] \tag{54}$$

The mean squared error is defined as:

$$\text{MSE}(\hat{\theta}) = E\left[\left(\hat{\theta} - \theta\right)^2\right] \tag{55}$$

The mean squared error decomposes into variance plus squared bias:

$$\text{MSE}(\hat{\theta}) = \text{Var}(\hat{\theta}) + \text{Bias}_\theta(\hat{\theta})^2 \tag{56}$$

A central asymptotic property is consistency. Let $\hat{\theta}_n$ denote an estimator constructed from n data points. The estimator is consistent if for every $\varepsilon > 0$:

$$\lim_{n\to\infty} P\left(\left|\hat{\theta}_n - \theta\right| > \varepsilon\right) = 0 \tag{57}$$

Unbiasedness and consistency are distinct properties. Unbiasedness requires $E[\hat{\theta}_n] = \theta$ for each n while consistency requires $\hat{\theta}_n$ to converge in probability to θ as n grows.

Beyond statistical accuracy, practical policy evaluation also depends on resource constraints:

1. **Computational complexity** measures how the cost of updating the estimate scales with additional data.
2. **Memory requirements** measure how much of the past data or sufficient statistics must be stored.
3. **Statistical efficiency** describes how quickly estimation error decreases as the amount of data grows and is often summarized empirically by mean squared error as a function of sample size.

Monte Carlo Policy Evaluation

Monte Carlo policy evaluation estimates value functions by treating the return as a random variable and approximating its expectation by empirical averages. The method assumes access to episodes generated by following the policy π and does not require knowing P or R. It can be applied whenever episodes terminate so that complete returns can be computed.

Consider an episodic interaction producing a trajectory $\tau = (S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_T)$ generated by following π. The return from time step t is defined as:

$$G_t = \sum_{k=0}^{T-t} \gamma^k R_{t+k} \tag{58}$$

The state value function is the conditional expectation of this return under trajectories induced by π:

$$V^\pi(s) = E_{\tau\sim\pi}\left[G_t \mid S_t = s\right] \tag{59}$$

The Monte Carlo principle is to estimate this expectation by a sample average over observed returns associated with visits to s. This yields different estimators depending on which visits within an episode are included.

First visit Monte Carlo

First visit Monte Carlo uses at most one return sample per episode for each state. For episode i and time t let $G_{i,t}$ denote the return from that time step. Let $N(s)$ count total first visits to s across episodes and let $G(s)$ accumulate the corresponding returns. The estimator is:

$$\hat{V}^{\pi}(s) = \frac{G(s)}{N(s)} \tag{60}$$

A standard pseudocode description of first visit Monte Carlo is given below:

Initialize $N(s) \leftarrow 0$ and $G(s) \leftarrow 0$ for all $s \in \mathcal{S}$
Loop
    Sample an episode i using policy π producing $S_{i,0}, A_{i,0}, R_{i,1}, \ldots, S_{i,T_i}$
    For each time step $t = 0, 1, \ldots, T_i - 1$ compute $G_{i,t}$
    For each time step $t = 0, 1, \ldots, T_i - 1$ do
        Let $s \leftarrow S_{i,t}$
        If s has not appeared in $\{S_{i,0}, \ldots, S_{i,t-1}\}$ then
            $N(s) \leftarrow N(s) + 1$
            $G(s) \leftarrow G(s) + G_{i,t}$
            $\hat{V}^{\pi}(s) \leftarrow G(s)/N(s)$
        End if
    End for
End loop

Every visit Monte Carlo

Every visit Monte Carlo uses the return from every occurrence of a state within an episode. Let $N(s)$ count total visits to s across all time steps and episodes and let $G(s)$ accumulate all corresponding returns. The estimator is:

$$\hat{V}^{\pi}(s) = \frac{G(s)}{N(s)} \tag{61}$$

A standard pseudocode description of every visit Monte Carlo is given below:

Initialize $N(s) \leftarrow 0$ and $G(s) \leftarrow 0$ for all $s \in \mathcal{S}$
Loop
    Sample an episode i using policy π producing $S_{i,0}, A_{i,0}, R_{i,1}, \ldots, S_{i,T_i}$
    For each time step $t = 0, 1, \ldots, T_i - 1$ compute $G_{i,t}$
    For each time step $t = 0, 1, \ldots, T_i - 1$ do
        Let $s \leftarrow S_{i,t}$
        $N(s) \leftarrow N(s) + 1$
        $G(s) \leftarrow G(s) + G_{i,t}$

$\hat{V}^{\pi}(s) \leftarrow G(s)/N(s)$

End for

End loop

First visit and every visit differ only in which samples are used. First visit uses at most one sample per episode for each state which removes within episode dependence for that state. Every visit uses more samples per episode which can reduce variance in practice but introduces correlations that can make the finite sample estimator biased while preserving consistency under standard conditions.

Incremental Monte Carlo

Incremental Monte Carlo replaces storing $G(s)$ and $N(s)$ with an incremental update that maintains the running mean. Let $\hat{V}_n(s)$ be the estimate after n total samples for state s and let $G_n(s)$ be the nth observed return sample for that state. The running mean identity is:

$$\hat{V}_{n+1}(s) = \frac{n\hat{V}_n(s) + G_{n+1}(s)}{n+1} \tag{62}$$

This identity can be rewritten into an incremental update form:

$$\hat{V}_{n+1}(s) = \hat{V}_n(s) + \frac{1}{n+1}\left(G_{n+1}(s) - \hat{V}_n(s)\right) \tag{63}$$

Using a generic step size α yields a family of incremental Monte Carlo updates applied whenever a return sample $G_{i,t}$ is available for a visited state $S_{i,t}$:

$$V(S_{i,t}) \leftarrow V(S_{i,t}) + \alpha\left(G_{i,t} - V(S_{i,t})\right) \tag{64}$$

A standard pseudocode description of incremental Monte Carlo with step sizes α is given below:

Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$

Loop

Sample an episode i using policy π producing $S_{i,0}, A_{i,0}, R_{i,1}, \ldots, S_{i,T_i}$

For each time step $t = 0, 1, \ldots, T_i - 1$ compute $G_{i,t}$

For each time step $t = 0, 1, \ldots, T_i - 1$ do

Let $s \leftarrow S_{i,t}$

$V(s) \leftarrow V(s) + \alpha\left(G_{i,t} - V(s)\right)$

End for

End loop

Monte Carlo statistical properties

The statistical properties of Monte Carlo estimators follow from classical laws of large numbers under mild conditions. First visit Monte Carlo yields an unbiased estimator of $V^\pi(s)$ because each sampled return associated with the first visit is an unbiased sample of the conditional expectation. As the number of first visits $N(s)$ grows the estimate converges to $V^\pi(s)$.

Every visit Monte Carlo can be biased for finite data because returns from multiple visits within the same episode are generally dependent. Nevertheless under standard ergodicity assumptions it is consistent and often has lower mean squared error in practice due to using more samples.

For incremental Monte Carlo with time varying step sizes $\alpha_n(s)$ a sufficient condition for convergence to $V^\pi(s)$ is that the step sizes satisfy Robbins Monro conditions for each state:

$$\sum_{n=1}^{\infty} \alpha_n(s) = \infty \tag{65}$$

$$\sum_{n=1}^{\infty} \alpha_n(s)^2 < \infty \tag{66}$$

Monte Carlo policy evaluation has two important limitations. It typically exhibits high variance which can require many episodes to obtain accurate estimates. It also requires an episodic setting because a return sample is only available after the episode terminates which prevents immediate online updates within an ongoing episode.

Temporal Difference Learning

Temporal difference learning addresses the policy evaluation problem in settings where the environment dynamics and reward model are unknown and where waiting for full episode termination is undesirable or impossible. It arises naturally in continuing tasks and large scale problems where data arrives sequentially and value estimates must be updated online.

Temporal difference methods occupy a conceptual middle ground between Monte Carlo methods and dynamic programming. Like Monte Carlo they are model free and learn directly from experience. Like dynamic programming they exploit the recursive structure of value functions through bootstrapping. This combination allows temporal difference methods to update value estimates incrementally after each transition while retaining theoretical convergence guarantees under suitable conditions.

Temporal difference learning is the central algorithmic idea that distinguishes reinforcement learning from classical stochastic approximation and control.

Bootstrapping

A defining feature of temporal difference methods is bootstrapping. Bootstrapping refers to the practice of updating an estimate using another estimate rather than a fully observed return.

In policy evaluation this means replacing the unknown expected future return by the current value estimate of the next state. Instead of waiting for the full return G_t temporal difference methods use a one step lookahead target of the form:

$$r_t + \gamma V(s_{t+1}) \tag{67}$$

Bootstrapping reduces variance and enables online learning but introduces bias because the update target depends on an imperfect estimate. The tradeoff between bias and variance introduced by bootstrapping is central to the design and analysis of reinforcement learning algorithms.

Motivation from Monte Carlo and Dynamic Programming

Recall that Monte Carlo policy evaluation estimates the value function by averaging complete returns:

$$V^\pi(s) = E_\pi[G_t \mid S_t = s] \tag{68}$$

In incremental Monte Carlo this leads to updates of the form:

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t)) \tag{69}$$

While unbiased this approach requires waiting until the end of an episode and typically exhibits high variance.

By contrast dynamic programming relies on the Bellman expectation equation:

$$V^\pi(s) = E_\pi[r_t + \gamma V^\pi(S_{t+1}) \mid S_t = s] \tag{70}$$

But dynamic programming requires knowledge of the transition and reward models.

Temporal difference learning combines these ideas by replacing the unknown expectation in the Bellman equation with a single sample transition while replacing the unknown future return with the current value estimate.

TD(0) Learning

TD(0) learning, also called one step temporal difference learning, is the simplest temporal difference method. It estimates the value function of a fixed policy π using sample transitions generated by following that policy.

Given a transition tuple (s_t, a_t, r_t, s_{t+1}) sampled under π, the TD(0) update is:

$$V(s_t) \leftarrow V(s_t) + \alpha \left(r_t + \gamma V(s_{t+1}) - V(s_t)\right) \tag{71}$$

The quantity:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{72}$$

is called the temporal difference error. It measures the inconsistency between the current value estimate and the one step bootstrapped target.

Unlike Monte Carlo methods TD(0) can update immediately after observing a single transition and does not require episodic termination. This makes it applicable to both episodic and continuing tasks.

TD(0) Algorithm

A standard pseudocode description of TD(0) policy evaluation is given below.

Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$
Loop
    Sample transition (s_t, a_t, r_t, s_{t+1}) following policy π
    $V(s_t) \leftarrow V(s_t) + \alpha \left(r_t + \gamma V(s_{t+1}) - V(s_t)\right)$
End Loop

Statistical Properties of TD(0)

Temporal difference learning introduces bias through bootstrapping but often achieves significantly lower variance than Monte Carlo methods. As a result TD(0) typically achieves lower mean squared error for a fixed amount of data.

Under suitable conditions TD(0) is a consistent estimator of V^π. In particular convergence is guaranteed when:

- the policy is fixed
- all states are visited infinitely often
- the learning rate satisfies $\sum_k \alpha_k = \infty$ and $\sum_k \alpha_k^2 < \infty$

In contrast to Monte Carlo methods TD(0) does not produce an unbiased estimator of the value function but is nonetheless consistent.

Comparison with Monte Carlo Policy Evaluation

Monte Carlo and temporal difference methods differ along several important dimensions.

Monte Carlo methods:

- are unbiased
- have high variance
- require episodic termination
- update only after complete returns are observed

Temporal difference methods:

- are biased due to bootstrapping
- have lower variance
- can be applied to continuing tasks
- update immediately after each transition

These differences explain why temporal difference learning is often preferred in large scale or online reinforcement learning problems.

Beyond TD(0)

TD(0) is the simplest member of a broader family of temporal difference methods. Extensions include multi step temporal difference learning TD(λ), eligibility traces, and control algorithms such as SARSA and Q learning. These methods trade off bias and variance by interpolating between Monte Carlo returns and one step bootstrapping and will be developed in subsequent sections.

TD(λ) Learning

TD(λ) learning generalizes TD(0) by updating value estimates toward multi step returns rather than a single one step bootstrapped target. It introduces a parameter $\lambda \in [0, 1]$ that controls how much future rewards influence the current update and thereby interpolates between one step temporal difference learning and Monte Carlo policy evaluation.

The central object underlying TD(λ) is the λ-return. For a time step t the λ-return is defined as a geometrically weighted average of n-step returns:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} \tag{73}$$

where the n-step return is given by:

$$G_t^{(n)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n}) \tag{74}$$

The λ-return smoothly interpolates between TD(0) and Monte Carlo returns. When $\lambda = 0$ the update target reduces to the one step TD target $r_t + \gamma V(s_{t+1})$. When $\lambda = 1$ and episodes terminate the update target coincides with the Monte Carlo return G_t.

TD(λ) policy evaluation updates the value estimate toward the λ-return:

$$V(s_t) \leftarrow V(s_t) + \alpha \left(G_t^\lambda - V(s_t) \right) \tag{75}$$

While this definition is conceptually clear it is not practical to compute G_t^λ explicitly online. Eligibility traces provide an efficient incremental mechanism to implement this update without storing full trajectories.

Eligibility Traces

Eligibility traces are a mechanism that assigns temporary credit to recently visited states so that temporal difference errors can be propagated backward in time. Each state s is associated with an eligibility trace $e_t(s)$ which measures how eligible that state is to receive learning updates at time t.

At each time step eligibility traces decay geometrically and the trace of the current state is incremented:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \\ \gamma\lambda e_{t-1}(s) & \text{otherwise} \end{cases} \tag{76}$$

The temporal difference error is computed as in TD(0):

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{77}$$

The value function is then updated for all states in proportion to their eligibility traces:

$$V(s) \leftarrow V(s) + \alpha\, \delta_t\, e_t(s) \tag{78}$$

Eligibility traces allow a single TD error to update not only the current state but also states visited earlier, with influence decaying according to $\gamma\lambda$. This implements TD(λ) efficiently and incrementally.

TD(λ) with Eligibility Traces Algorithm

A standard pseudocode description of TD(λ) policy evaluation using eligibility traces is given below.

Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$
Initialize $e(s) = 0$ for all $s \in \mathcal{S}$
Loop
    Sample transition (s_t, a_t, r_t, s_{t+1}) following policy π
    $\delta_t \leftarrow r_t + \gamma V(s_{t+1}) - V(s_t)$
    For all $s \in \mathcal{S}$: $e(s) \leftarrow \gamma\lambda e(s)$
    $e(s_t) \leftarrow e(s_t) + 1$
    For all $s \in \mathcal{S}$: $V(s) \leftarrow V(s) + \alpha\, \delta_t\, e(s)$
End Loop

Setting $\lambda = 0$ yields TD(0), since eligibility traces decay immediately and only the current state is updated. Larger values of λ increase the temporal span over which learning updates are propagated, approaching Monte Carlo behavior as λ approaches one.

Practical Comparison of Temporal Difference Methods

While TD(0), TD(λ), and TD(λ) with eligibility traces converge to the same value function under standard assumptions, they differ in practical behavior and resource requirements.

TD(0) performs a single state update per transition and is computationally cheap and easy to implement but typically exhibits higher bias. Increasing λ incorporates longer horizon information which reduces bias and often improves empirical accuracy at the cost of increased variance.

Eligibility traces enable TD(λ) to be implemented fully online by distributing each temporal difference error backward to recently visited states or parameters. This improves learning speed but increases per step computation and memory proportional to the size of the value representation.

In summary TD(0) favors simplicity and low computational cost while TD(λ) with eligibility traces offers improved statistical efficiency and faster propagation of information at higher computational and memory cost.

Model Based Policy Evaluation via Certainty Equivalence

In some settings policy evaluation can be performed without access to the true environment dynamics by explicitly learning a model of the Markov Decision Process from data and then planning in this learned model. This approach is model based and relies on the principle of certainty equivalence.

The core idea is to replace the unknown transition probabilities and reward function by their maximum likelihood estimates computed from observed experience and then to evaluate the policy exactly in the resulting estimated MDP using dynamic programming.

After observing a stream of transitions (s, a, r, s') the maximum likelihood estimates of the transition model and expected reward are given by:

$$\hat{P}(s' \mid s, a) = \frac{1}{N(s, a)} \sum_{k} \mathbf{1}\{s_k = s, a_k = a, s_{k+1} = s'\} \tag{79}$$

$$\hat{r}(s, a) = \frac{1}{N(s, a)} \sum_{k} \mathbf{1}\{s_k = s, a_k = a\} r_k \tag{80}$$

Here $N(s, a)$ denotes the total number of times action a has been taken in state s. These estimates define a fully specified empirical MDP.

Given this estimated MDP the value function of a fixed policy π is computed by solving the Bellman expectation equations using any dynamic programming method such as iterative policy evaluation.

A high level pseudocode description is as follows.

Initialize counts $N(s, a) = 0$, rewards and transitions arbitrarily
Loop
    Observe transition (s, a, r, s')
    Update counts and empirical estimates $\hat{P}$ and $\hat{r}$
    Solve for V^{π} in the estimated MDP using dynamic programming
End Loop

This approach is statistically efficient and consistent under correct Markov assumptions since the maximum likelihood model converges to the true MDP as data increases. However it is computationally expensive because each update requires solving a planning problem whose cost scales polynomially with the size of the state and action spaces.

Policy Improvement

Policy improvement addresses the problem of using value estimates to construct better policies. In model free settings this must be done while data is still being collected and value functions are only approximately known. A central difficulty is balancing the need to exploit current knowledge with the need to explore actions whose consequences are uncertain.

ε-Greedy Policies

A simple and widely used mechanism for balancing exploration and exploitation is the ε-greedy policy. This class of policies is defined with respect to a state action value function $Q(s, a)$ and is particularly suited to discrete action spaces.

Motivation

Purely greedy policies that always select the action with the highest estimated value fail to explore. Actions that appear suboptimal early due to noise or limited data may never be tried again which can prevent learning the optimal policy.

Conversely purely random policies explore thoroughly but perform poorly because they ignore accumulated knowledge. The ε-greedy strategy interpolates between these extremes by acting greedily most of the time while occasionally selecting actions at random to ensure continued exploration.

Definition

Let $\mathcal{A}$ be a finite action space and let $Q(s, a)$ be a given state action value function. An ε-greedy policy with respect to Q is defined by:

$$\pi(a \mid s) = \begin{cases} 1 - \varepsilon + \dfrac{\varepsilon}{|\mathcal{A}|} & \text{if } a \in \arg\max_{a'} Q(s, a') \\ \dfrac{\varepsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases} \tag{81}$$

Equivalently the policy selects a greedy action with probability $1 - \varepsilon$ and selects an action uniformly at random with probability ε.

Policy Improvement Property

Although ε-greedy policies are stochastic the fundamental policy improvement property still holds. If π_i is any policy and π_{i+1} is the ε-greedy policy with respect to Q^{π_i} then the value function improves monotonically:

$$V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s) \quad \forall s \in \mathcal{S} \tag{82}$$

This ensures that interleaving approximate policy evaluation with ε-greedy policy improvement yields a sequence of policies that improve in expectation even though the policy never becomes fully deterministic.

Remarks on Continuous Action Spaces

The standard ε-greedy construction relies on a finite action set and uniform random exploration. In continuous action spaces this definition is no longer directly applicable.

In such settings exploration is typically achieved by adding noise to the greedy action for example through Gaussian perturbations or by sampling from a stochastic policy parameterization. These alternatives play an analogous role to ε-greedy exploration and will be discussed in later sections on continuous control.

Model-Free Policy Learning for Tabular Settings

The objective is no longer to evaluate a fixed policy but to learn an optimal policy directly from interaction with the environment. In the tabular setting this is done by estimating the optimal state–action value function $Q^\star$ and improving the policy with respect to these estimates.

All methods in this section assume a finite state space and a finite action space and rely on explicit tables to store value estimates. This setting allows clean algorithmic descriptions and precise convergence guarantees and serves as the conceptual foundation for more general methods.

Monte Carlo Online Control with On-Policy Improvement

Monte Carlo control extends Monte Carlo policy evaluation from state values V^π to state–action values Q^π. Estimating $Q(s, a)$ is essential for control because it allows direct comparison of actions without requiring a model of the environment.

The key idea is simple. For each visited state–action pair the return following that pair is treated as a sample of the random variable whose expectation defines $Q^\pi(s, a)$. Policy improvement is then performed by acting greedily or ε-greedily with respect to the current Q estimates.

Monte Carlo control extends Monte Carlo policy evaluation to the control setting by estimating state–action values and interleaving policy improvement with evaluation. The goal is to learn an optimal policy directly from experience by

repeatedly estimating Q^π and updating the policy to be greedy or ε-greedy with respect to these estimates.

Let $G_{k,t}$ denote the return observed from time step t onward in the k-th episode. The Monte Carlo estimator treats each observed return following a state–action pair as a sample of the random variable defining $Q^\pi(s,a)$. In first-visit Monte Carlo control, updates are performed only the first time a given pair (s,a) is encountered in an episode.

The update rule is:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \frac{1}{N(s_t,a_t)}\left(G_{k,t} - Q(s_t,a_t)\right) \tag{83}$$

where $N(s,a)$ counts the total number of first visits to (s,a) across episodes.

Policy improvement is performed online by constructing an ε-greedy policy with respect to the current Q estimate. To ensure sufficient exploration early and greedy behavior asymptotically, the exploration rate is decreased over episodes, for example:

$$\varepsilon_k = \frac{1}{k} \tag{84}$$

This yields a Greedy in the Limit of Infinite Exploration policy sequence.

A standard pseudocode description of Monte Carlo online control with on-policy improvement is given below.

Initialize $Q(s,a)$, $N(s,a) = 0$ for all (s,a)
Set $\varepsilon = 1$, $k = 1$
Define initial policy π_k to be ε-greedy w.r.t. Q
Loop
    Sample episode k: $(s_{k,1}, a_{k,1}, r_{k,1}, \ldots, s_{k,T_k})$ using policy π_k
    Compute returns $G_{k,t}$ for all $t = 1, \ldots, T_k$
    For each time step $t = 1, \ldots, T_k$ do
        If $(s_{k,t}, a_{k,t})$ is first visit in episode k then
            $N(s_{k,t}, a_{k,t}) \leftarrow N(s_{k,t}, a_{k,t}) + 1$
            Update $Q(s_{k,t}, a_{k,t})$ using the sample mean update
        End if
    End for
    $k \leftarrow k + 1$, $\varepsilon \leftarrow 1/k$
    Update policy π_k to be ε-greedy w.r.t. current Q
End Loop

Greedy in the Limit of Infinite Exploration

Monte Carlo control relies on the Greedy in the Limit of Infinite Exploration property. A learning process satisfies GLIE if:

$$\lim_{k\to\infty} N_k(s,a) = \infty \quad \forall (s,a) \tag{85}$$

and

$$\lim_{k\to\infty} \pi_k(a \mid s) = \arg\max_a Q(s,a) \tag{86}$$

Under GLIE conditions Monte Carlo control converges with probability one to the optimal action–value function $Q^\star$.

Temporal Difference Methods for Control

Monte Carlo control requires complete episodes and suffers from high variance. Temporal difference methods address these limitations by bootstrapping and updating after each transition. Two central TD control algorithms are Q-learning and SARSA.

Q-Learning

Q-learning is a model-free off-policy control algorithm that directly estimates the optimal action–value function $Q^\star$ while interacting with the environment using an exploratory behavior policy. Unlike on-policy methods, Q-learning learns the value of the optimal policy independently of the policy actually used to generate data.

The central idea is to combine temporal-difference learning with greedy policy improvement. After each transition, the current estimate of Q is updated toward a target that assumes optimal behavior from the next state onward. The update rule is:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left(r_t + \gamma \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t) \right) \tag{87}$$

The bootstrap target uses a maximization over actions in the next state rather than the action actually taken. This is what makes Q-learning off-policy.

Policy improvement is interleaved with learning by defining a behavior policy that is ε-greedy with respect to the current Q estimate. This ensures sufficient exploration while gradually favoring actions with higher estimated value.

A standard pseudocode description of Q-learning with ε-greedy policy improvement is given below.

Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
Set episode counter $k \leftarrow 1$
Set $\varepsilon \leftarrow 1$
Loop over episodes
    Initialize state s_0, time $t \leftarrow 0$
    Define behavior policy π_b to be ε-greedy w.r.t. Q
    Loop until terminal
        Select action $a_t \sim \pi_b(\cdot \mid s_t)$
        Execute a_t, observe reward r_t and next state s_{t+1}
        Update action–value estimate:
            $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\,(r_t + \gamma \max_{a\in\mathcal{A}} Q(s_{t+1},a) - Q(s_t,a_t))$
        Update behavior policy π_b to remain ε-greedy w.r.t. updated Q
        $s_t \leftarrow s_{t+1}$, $t \leftarrow t+1$
    End loop
    $k \leftarrow k+1$, $\varepsilon \leftarrow 1/k$
End loop

Under standard assumptions for tabular settings, including GLIE exploration and learning rates satisfying the Robbins–Monro conditions, Q-learning converges almost surely to the optimal action–value function $Q^\star$. Acting greedily with respect to this limit yields an optimal policy.

SARSA

SARSA is an on-policy temporal difference control algorithm that simultaneously evaluates and improves a policy using experience generated by that same policy. Unlike Q-learning SARSA does not assume greedy behavior in the future but instead learns the action–value function corresponding to the current behavior policy.

Given a transition $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ generated by following an ε-greedy policy with respect to the current Q estimate the SARSA update rule is:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\,(r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)) \tag{88}$$

The bootstrap target uses the value of the actual next action selected by the policy rather than a greedy action. As a result SARSA learns the value of the policy it executes rather than the value of an implicitly optimal policy.

Policy improvement is achieved implicitly by updating the behavior policy to remain ε-greedy with respect to the evolving Q estimates.

A standard pseudocode description of SARSA control is given below.

Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
Set episode counter $k \leftarrow 1$
Set $\varepsilon \leftarrow 1$
Loop over episodes
    Initialize state s_0, time $t \leftarrow 0$
    Define policy π_k to be ε-greedy w.r.t. Q
    Select $a_0 \sim \pi_k(\cdot \mid s_0)$
    Loop until terminal
        Execute a_t, observe reward r_t and next state s_{t+1}
        Select $a_{t+1} \sim \pi_k(\cdot \mid s_{t+1})$
        Update action–value estimate:
            $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\,(r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t))$
        $s_t \leftarrow s_{t+1}$, $a_t \leftarrow a_{t+1}$, $t \leftarrow t+1$
    End loop
    $k \leftarrow k+1$, $\varepsilon \leftarrow 1/k$
End loop

Under GLIE exploration and Robbins–Munro step-size conditions SARSA converges almost surely to the optimal action–value function in finite state and action spaces.

SARSA(λ)

SARSA(λ) extends SARSA by incorporating eligibility traces, allowing credit to be assigned to recently visited state–action pairs. This interpolates between one-step TD learning and Monte Carlo updates and often improves learning speed.

For SARSA(λ) we maintain both an action–value table $Q(s,a)$ and an eligibility trace table $e(s,a)$. At each time step, traces decay and the trace of the currently visited pair is incremented, which can be written compactly using an indicator on the current state–action pair.

The on-policy TD error is:

$$\delta_t = r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t) \tag{89}$$

The accumulating eligibility trace update can be written compactly as:

$$e_t(s,a) = \gamma\lambda\, e_{t-1}(s,a) + \mathbf{1}\{s = s_t,\ a = a_t\} \tag{90}$$

The action–value update is then applied to all pairs:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha_t\, \delta_t\, e_t(s,a) \tag{91}$$

A standard pseudocode description is given below.

Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
Set episode counter $k \leftarrow 1$
Set $\varepsilon \leftarrow 1$
Loop over episodes
    Initialize $e(s,a) \leftarrow 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$
    Initialize state s_0, time $t \leftarrow 0$
    Define policy π_k to be ε-greedy w.r.t. Q
    Choose $a_0 \sim \pi_k(\cdot \mid s_0)$
    Loop until terminal
        Take action a_t, observe reward r_t and next state s_{t+1}
        Choose $a_{t+1} \sim \pi_k(\cdot \mid s_{t+1})$
        $\delta_t \leftarrow r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$
        For all $s \in \mathcal{S}, a \in \mathcal{A}$ do
            $e(s,a) \leftarrow \gamma\lambda e(s,a)$
        End for
        $e(s_t, a_t) \leftarrow e(s_t, a_t) + 1$
        For all $s \in \mathcal{S}, a \in \mathcal{A}$ do
            $Q(s,a) \leftarrow Q(s,a) + \alpha\, \delta_t\, e(s,a)$
        End for
        $s_t \leftarrow s_{t+1}$, $a_t \leftarrow a_{t+1}$, $t \leftarrow t+1$
    End loop
    $k \leftarrow k+1$, $\varepsilon \leftarrow 1/k$
End loop

SARSA versus Q-Learning

The fundamental distinction between SARSA and Q-learning lies in on-policy versus off-policy learning.

SARSA learns the value of the policy actually being executed and therefore accounts for exploration in its updates. Q-learning ignores the exploration policy and learns as if greedy actions will always be taken in the future.

As a result SARSA often exhibits safer behavior under exploration while Q-learning tends to learn more aggressive policies. Both algorithms converge to optimal control in the tabular discounted setting under standard assumptions.

Model-Free Value Function Approximation

In all previous sections value functions and action–value functions were represented explicitly, either as tables or by exact dynamic programming computations. These approaches become infeasible when the state or action spaces are large or continuous. Model-free value function approximation addresses this limitation by representing value functions using a parameterized function class that generalizes across states and actions.

The key idea is to replace tabular representations of V^π or Q^π by a differentiable function $\hat{Q}(s, a; \theta)$ or $\hat{V}(s; \theta)$, parameterized by a finite-dimensional vector θ, and to learn θ directly from data using stochastic optimization.

This enables reinforcement learning algorithms to scale to high-dimensional and continuous domains, but introduces new sources of approximation error and potential instability.

Oracle Perspective and Learning Objective

To motivate value function approximation it is useful to consider an idealized oracle setting. Suppose that for any state–action pair (s, a) an oracle returns the true action–value $Q^\pi(s, a)$. In this case learning Q^π reduces to a supervised learning problem.

Given a parameterized function $\hat{Q}(s, a; \theta)$, the objective is to minimize the mean squared error:

$$J(\theta) = E_{(s,a)\sim\pi}\left[\left(Q^\pi(s, a) - \hat{Q}(s, a; \theta)\right)^2\right] \tag{92}$$

This objective cannot be evaluated directly in practice because Q^π is unknown. Model-free methods replace $Q^\pi(s, a)$ by suitable targets constructed from data.

Stochastic Gradient Descent

Gradient-based optimization is used to minimize the objective $J(\theta)$. The gradient of the mean squared error objective is:

$$\nabla_\theta J(\theta) = -2\, E_{(s,a)\sim\pi}\left[\left(Q^\pi(s, a) - \hat{Q}(s, a; \theta)\right) \nabla_\theta \hat{Q}(s, a; \theta)\right] \tag{93}$$

Since the expectation is unknown, stochastic gradient descent approximates this gradient using samples. For a single sample with target y, the update takes the form:

$$\theta \leftarrow \theta + \alpha \left(y - \hat{Q}(s, a; \theta)\right) \nabla_\theta \hat{Q}(s, a; \theta) \tag{94}$$

Different model-free algorithms correspond to different choices of the target y.

Monte Carlo Value Function Approximation

In Monte Carlo value function approximation the return G_t is used as an unbiased sample of $Q^\pi(s_t, a_t)$. The objective being minimized is:

$$J(\theta) = E_\pi \left[\left(G_t - \hat{Q}(s_t, a_t; \theta) \right)^2 \right] \tag{95}$$

The stochastic gradient update is:

$$\theta \leftarrow \theta + \alpha \left(G_t - \hat{Q}(s_t, a_t; \theta) \right) \nabla_\theta \hat{Q}(s_t, a_t; \theta) \tag{96}$$

Initialize parameters θ
Set episode counter $k = 1$
Loop
    Sample episode k using policy π
    Compute returns $G_{k,t}$ for all time steps
    For each time step t in episode k
        $\theta \leftarrow \theta + \alpha \left(G_{k,t} - \hat{Q}(s_t, a_t; \theta) \right) \nabla_\theta \hat{Q}(s_t, a_t; \theta)$
    End for
    $k \leftarrow k + 1$
End Loop

Monte Carlo value function approximation is unbiased but typically suffers from high variance and requires episodic tasks.

TD(0) Value Function Approximation

Temporal difference value function approximation replaces the return by a bootstrapped target:

$$y_t = r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \theta) \tag{97}$$

The objective minimized is:

$$J(\theta) = E_\pi \left[\left(r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \theta) - \hat{Q}(s_t, a_t; \theta) \right)^2 \right] \tag{98}$$

The update rule is:

$$\theta \leftarrow \theta + \alpha \delta_t \nabla_\theta \hat{Q}(s_t, a_t; \theta) \tag{99}$$

where $\delta_t = r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \theta) - \hat{Q}(s_t, a_t; \theta)$.

Initialize parameters θ
Initialize state s_0
Loop
    Sample $a_t \sim \pi(\cdot \mid s_t)$
    Observe r_t, s_{t+1}
    Sample $a_{t+1} \sim \pi(\cdot \mid s_{t+1})$
    $\delta_t \leftarrow r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \theta) - \hat{Q}(s_t, a_t; \theta)$
    $\theta \leftarrow \theta + \alpha \delta_t \nabla_\theta \hat{Q}(s_t, a_t; \theta)$
    $s_t \leftarrow s_{t+1}$
End Loop

SARSA with Function Approximation

SARSA with function approximation is an on-policy control method that interleaves policy evaluation and policy improvement online. Experience is generated by an ε-greedy policy with respect to the current action–value approximation $\hat{Q}(s, a; \theta)$.

After each transition the parameters are updated by stochastic gradient descent using a bootstrapped target that depends on the next action actually taken by the same policy. Policy improvement is therefore implicit and continuous, since the behavior policy is always defined to be ε_k-greedy with respect to the latest parameter vector, while the exploration rate ε_k is typically annealed across episodes.

With eligibility traces, SARSA(λ) remains well-defined for continuous state and action spaces by maintaining traces in parameter space. Let z_t denote the eligibility trace vector:

$$z_t = \gamma \lambda z_{t-1} + \nabla_\theta \hat{Q}(s_t, a_t; \theta) \tag{100}$$

The temporal difference error and parameter update are:

$$\delta_t = r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \theta) - \hat{Q}(s_t, a_t; \theta) \tag{101}$$

$$\theta \leftarrow \theta + \alpha \, \delta_t \, z_t \tag{102}$$

Setting $\lambda = 0$ recovers standard SARSA with function approximation.

Initialize parameters θ, set episode counter $k \leftarrow 1$
Initialize eligibility trace $z \leftarrow 0$
Set ε_k
Repeat for each episode
    Initialize state s_0, choose $a_0 \sim \varepsilon_k$-greedy$(\hat{Q}(\cdot, \cdot; \theta))$
    Loop for $t = 0, 1, 2, \ldots$ until terminal
        Take a_t, observe r_t, s_{t+1}

Choose $a_{t+1} \sim \varepsilon_k$-greedy$(\hat{Q})$
$\delta_t \leftarrow r_t + \gamma \hat{Q}(s_{t+1}, a_{t+1}; \theta) - \hat{Q}(s_t, a_t; \theta)$
$z \leftarrow \gamma \lambda z + \nabla_\theta \hat{Q}(s_t, a_t; \theta)$
$\theta \leftarrow \theta + \alpha \delta_t z$
$s_t \leftarrow s_{t+1}, a_t \leftarrow a_{t+1}$
End Loop
$k \leftarrow k + 1$, update ε_k
End Repeat

Q-Learning with Function Approximation

Q-learning with function approximation is an off-policy control method that estimates the optimal action–value function while following an exploratory behavior policy. As in SARSA, data is generated by an ε_k-greedy policy with respect to the current approximation, and parameters are updated after every transition.

The crucial difference is that the target assumes greedy optimal behavior at the next state, independently of the action actually taken. Policy improvement is again implicit and continuous, since the behavior policy is always defined with respect to the latest parameters, while ε_k is reduced across episodes.

The bootstrapped target, TD error, and parameter update are:

$$y_t = r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \theta) \tag{103}$$

$$\delta_t = y_t - \hat{Q}(s_t, a_t; \theta) \tag{104}$$

$$\theta \leftarrow \theta + \alpha \, \delta_t \, \nabla_\theta \hat{Q}(s_t, a_t; \theta) \tag{105}$$

Initialize parameters θ, set episode counter $k \leftarrow 1$
Set ε_k
Repeat for each episode
Initialize state s_0
Loop for $t = 0, 1, 2, \ldots$ until terminal
Choose $a_t \sim \varepsilon_k$-greedy$(\hat{Q}(\cdot, \cdot; \theta))$
Take a_t, observe r_t, s_{t+1}
$\delta_t \leftarrow r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \theta) - \hat{Q}(s_t, a_t; \theta)$
$\theta \leftarrow \theta + \alpha \delta_t \nabla_\theta \hat{Q}(s_t, a_t; \theta)$
$s_t \leftarrow s_{t+1}$
End Loop
$k \leftarrow k + 1$, update ε_k
End Repeat

The Deadly Triad

When value function approximation is combined with online policy improvement, stability is no longer guaranteed. In tabular settings Bellman operators are contractions, but with function approximation each update consists of a bootstrapped target followed by a projection back into a restricted function class. This projection can destroy the contraction property that ensures convergence.

Instability arises when the following three ingredients are present simultaneously:

1. **Bootstrapping**, where update targets depend on current value estimates rather than on fully observed returns.

2. **Function approximation**, which introduces approximation and projection error by restricting value functions to a parameterized function class.

3. **Off-policy learning**, where data is generated by a behavior policy that differs from the policy being evaluated or improved.

Bootstrapping causes errors to propagate through future targets, function approximation prevents exact Bellman updates, and off-policy learning breaks the alignment between the data distribution and the target policy. When all three are present, small errors can be amplified rather than corrected. Q-learning with function approximation contains all three elements and is therefore particularly prone to divergence.

In practice, SARSA and Q-learning with function approximation are still used in small-scale problems and as conceptual baselines. Modern deep reinforcement learning methods can be viewed as systematic attempts to control the interaction between these three components. Deep Q-learning introduces architectural and algorithmic mechanisms that decouple data collection from target computation and slow down the propagation of estimation errors, providing a practical resolution to the deadly triad in large-scale settings.