## horizontal line

SARSA , Q-learning

28.10.2025

# On-Policy Vs Off-Policy

**On-policy learning uses the same policy to make decisions and learn from them, while off-policy learning learns about one policy (target) using data generated by another (behavior) policy. The key difference lies in *whose actions* the agent learns from.**

🧠 What Is Policy in RL?

A **policy** \pi is a strategy that tells an agent what action to take in each state.  
 In reinforcement learning, we want to either:

* **Evaluate** a policy (prediction)
* **Improve** a policy (control)

🔍 On-Policy Learning

**Definition:**

The agent learns from actions taken by the **same policy** it is trying to evaluate or improve.

🔍 Off-Policy Learning

**Definition:**

The agent learns about a **target policy** using data from a **different behavior policy**.

# Model Free Control Vs Model Free Prediction

**Model-free prediction estimates how good a policy is (i.e. expected rewards), while model-free control improves the policy to find the best one — both without knowing the environment’s dynamics.**

🧠 What Is “Model-Free” in RL?

The agent **does not know** the transition probabilities or reward function of the environment. Instead, it learns purely from **experience** — by interacting with the environment and observing outcomes.

**🔍 Model-Free Prediction**

**Goal:  
Estimate the value function V^\pi(s) or Q^\pi(s, a) for a given policy \pi.**

**What it does:**

* Evaluates how good a policy is.
* Learn expected returns from states or actions.
* Doesn’t change the policy — just analyzes it.

✅ Algorithms:

* **Monte Carlo Prediction**
* **TD(0) Prediction**
* **SARSA (when used for evaluation)**

**🔧 Model-Free Control**

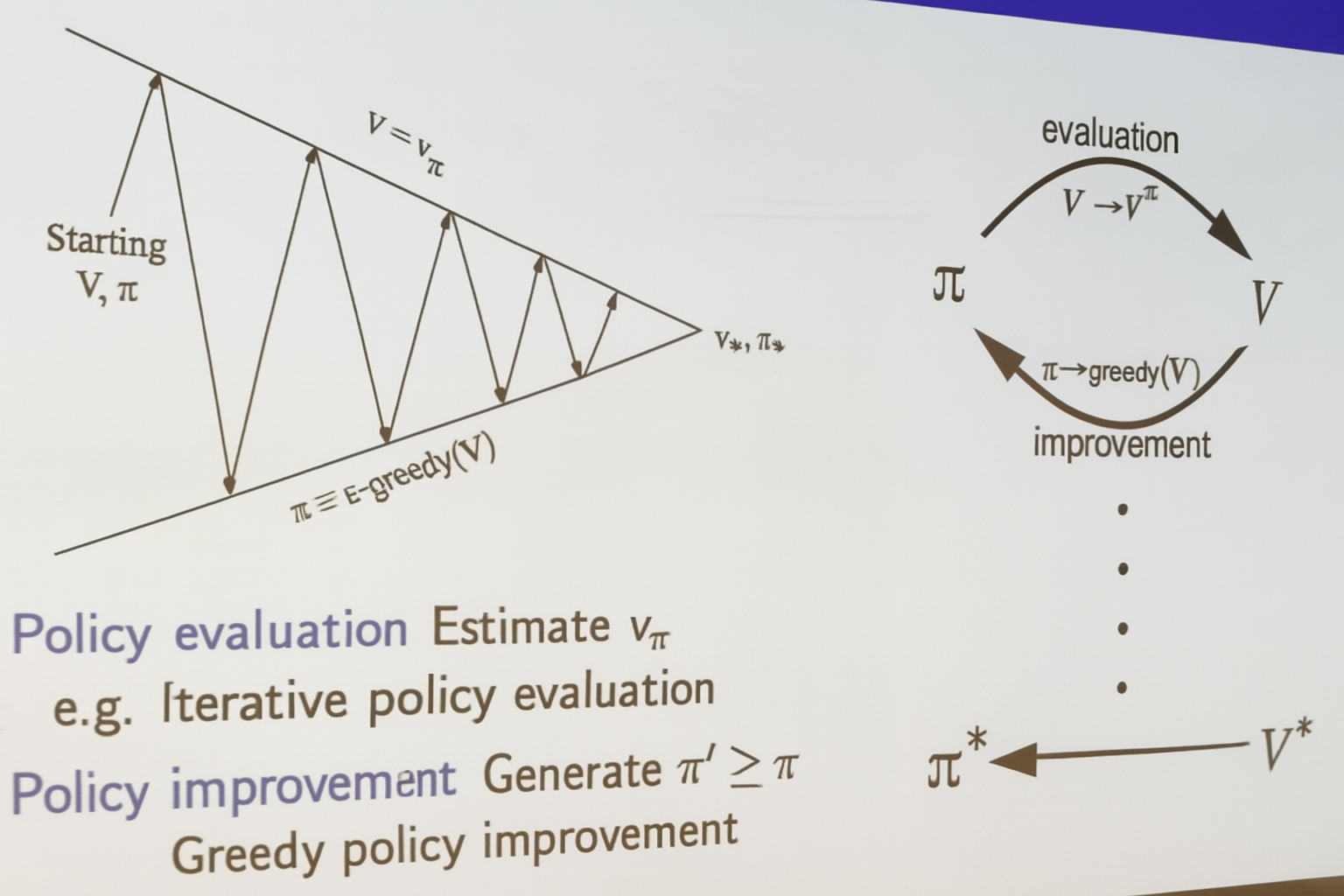
**Goal:  
Find the optimal policy \pi^\* that maximizes expected reward.**

**What it does:**

* Learn both value functions and the best actions.
* Improves the policy over time.
* Uses exploration to discover better strategies.

**✅ Algorithms:**

* **Q-learning**
* **SARSA (when used for control)**
* **Actor-Critic (with bootstrapping)**



What Is Policy Iteration?

Policy iteration is a **two-step loop**:

1. **Policy Evaluation**: Estimate how good the current policy is.
2. **Policy Improvement**: Use that estimate to create a better policy.

Repeat until the policy stops changing — that’s your optimal policy \pi^\*.

Model Free Prediction - Policy evaluation

Model Free Control - Policy improvement

# 🔁 Iterative Policy Evaluation

**Goal:**Estimate the **value function V^\pi(s)** — the expected return from each state when following policy \pi.

🔍 How It Works:

* Start with an initial guess for V(s) (e.g., all zeros).
* Use the Bellman expectation equation to update each state’s value:
* V(s) \leftarrow \sum\_a \pi(a|s) \sum\_{s'} P(s'|s,a) \left[ R(s,a,s') + \gamma V(s') \right]Repeat until values converge (i.e., changes become very small).

# 💡 Greedy Policy Improvement

**Goal:**Use the current value function V(s) to create a **better policy \pi'** by choosing actions that maximize expected return.

🔍 How It Works:

* For each state s, choose the action a that gives the highest expected value:
* \pi'(s) = \arg\max\_a \sum\_{s'} P(s'|s,a) \left[ R(s,a,s') + \gamma V(s') \right]This is called a **greedy policy** — it always picks the best action based on current estimates.

Epsilon-greedy (ε-greedy) exploration is a strategy in reinforcement learning where the agent chooses a random action with probability ε and the best-known action with probability 1−ε. This balances exploration and exploitation.

🎯 Why Use ε-Greedy?

* **Explore** to discover new, potentially better actions.
* **Exploit** known actions that yield high rewards.

The ε-greedy strategy ensures the agent doesn’t get stuck exploiting suboptimal actions too early.

🔍 How It Works

At each decision step:

* With probability **ε**: choose a **random action** (exploration)
* With probability **1−ε**: choose the **best-known action** (exploitation)

Example:

If ε = 0.1:

* 10% of the time → random action
* 90% of the time → greedy action (highest Q-value)

This randomness helps the agent discover better strategies over time.

# GLIE

**GLIE stands for “Greedy in the Limit with Infinite Exploration.” It’s a reinforcement learning strategy that ensures the agent explores enough early on, but eventually behaves greedily to converge to the optimal policy.**

🧠 What Does GLIE Mean?

GLIE is a condition applied to **ε-greedy policies** in reinforcement learning. It guarantees two things:

1. **Infinite Exploration**:  
    Every state-action pair is visited infinitely often.  
    → Ensures the agent doesn’t miss out on discovering better actions.
2. **Greedy in the Limit**:  
    As learning progresses, the policy becomes greedy with respect to the learned value function.  
    → Ensures convergence to the optimal policy.

# What Is Monte Carlo Control?

Monte Carlo control combines:

* **Monte Carlo prediction**: Estimate action-value function Q(s, a) from episodes.
* **Policy improvement**: Use greedy or ε-greedy updates to improve the policy.

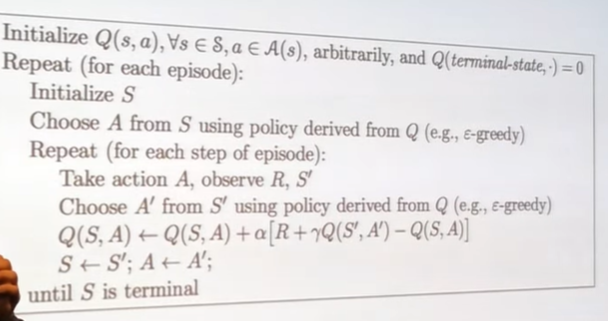
It’s called *model-free* because it learns purely from experience — no transition probabilities or reward models needed.

# 🧠 What Is SARSA?

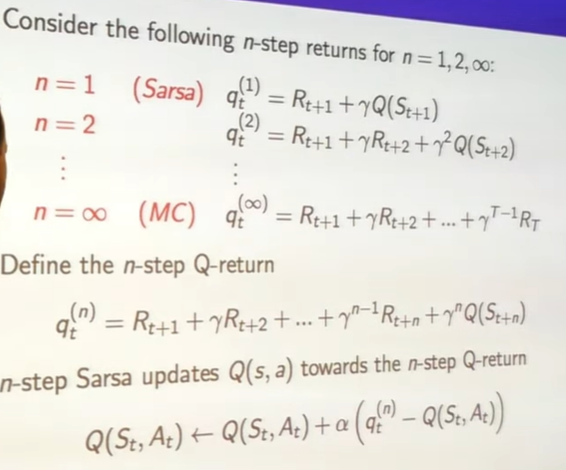
**SARSA** stands for:

**State–Action–Reward–State–Action**

It’s an **on-policy** algorithm, meaning it learns from the actions taken by the current policy — including exploratory ones.



# N-Step SARSA



What Is n-step SARSA?

In standard SARSA, the Q-value update uses the immediate reward and the next Q-value.  
 **n-step SARSA** generalizes this by using **n future rewards** and the Q-value at the n^{th} step.

SARSA becomes equivalent to Monte Carlo (MC) control when the number of steps n \to \infty, because it stops bootstrapping and instead uses the full return from the episode — just like MC.

Forward view SARSA computes updates by looking ahead over future steps, while backward view SARSA uses eligibility traces to assign credit backward from the current time step. Both aim to improve learning efficiency in temporal-difference methods.

# Importance Sampling in Monte Carlo

**Importance sampling is a statistical technique used in off-policy Monte Carlo methods to correct for the mismatch between the behavior policy (used to generate data) and the target policy (being evaluated or improved). It allows learning from episodes generated by a different policy.**

🧠 Why Importance Sampling Is Needed in Off-Policy Monte Carlo

In **off-policy Monte Carlo**, we want to estimate the value of a **target policy** \pi, but the episodes we observe come from a **behavior policy** \mu.  
This creates a mismatch: the actions taken in the episode may not match what \pi would have done.

**Importance sampling corrects this mismatch** by weighting each episode’s return based on how likely it was under the target policy compared to the behavior policy.

# Q-learning

**Q-learning is a model-free reinforcement learning algorithm that learns the optimal action-value function Q(s, a) by interacting with the environment and updating estimates based on rewards and future values. It’s an off-policy method that converges to the optimal policy even when exploring.**

🧠 Core Idea of Q-Learning

Q-learning helps an agent learn *what to do* — which action to take in a given state — to maximize cumulative reward over time.

It does this by learning a **Q-table**:

* Rows = states
* Columns = actions
* Entries = estimated future reward for taking action a in state s

🔁 Q-Learning Update Rule

At each time step t, the agent:

1. Observes current state s\_t
2. Chooses action a\_t (often using ε-greedy)
3. Receives reward r\_{t+1} and next state s\_{t+1}
4. Updates Q-value:

Q(s\_t, a\_t) \leftarrow Q(s\_t, a\_t) + \alpha \left[ r\_{t+1} + \gamma \max\_a Q(s\_{t+1}, a) - Q(s\_t, a\_t) \right]

Off-policy control with Q-learning means learning the optimal policy by updating Q-values using the greedy policy—even though the agent explores using a different behavior policy like ε-greedy. This allows the agent to learn optimal behavior while still exploring.

# SARSA-MAX

SARSA-MAX is another name for the standard Q-learning control algorithm. It’s called “SARSA-MAX” because it resembles SARSA but uses the maximum Q-value for the next state instead of the Q-value from the next action. This makes it an off-policy method.

