## horizontal line

Policy Gradient Methods

31.10.2025

# Policy Based RL

**Policy-based reinforcement learning directly learns a policy that maps states to actions, without estimating value functions. It’s especially useful in high-dimensional or continuous action spaces where value-based methods struggle.**

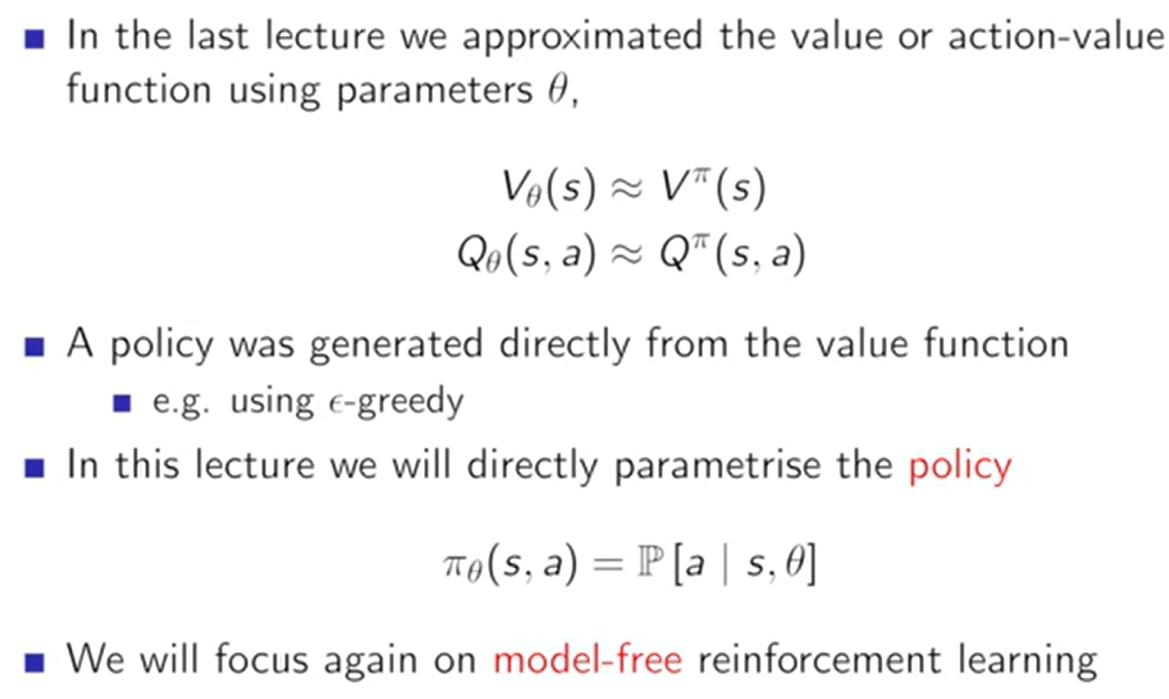
🧠 What Is a Policy?

In RL, a **policy** \pi (a|s) defines the probability of taking action in state s.  
Policy-based methods aim to **learn this mapping directly**, rather than indirectly through value functions.

🔁 How Policy-Based RL Works —> Instead of learning Q(s,a) or V(s), we learn:

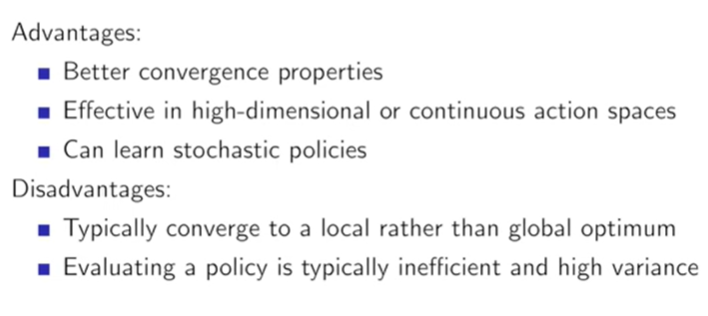
It builds a **policy function**:

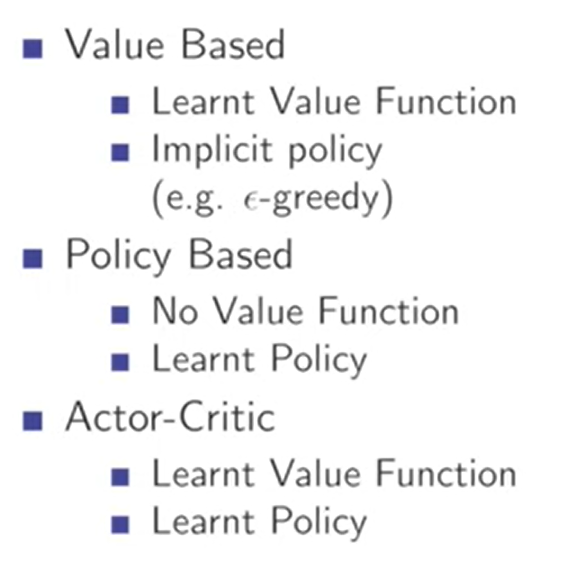
\pi \_{\theta }(a|s)

* \theta : parameters of the policy (e.g., weights of a neural network)
* The agent updates \theta to maximize expected return:

This is done using **gradient ascent**: \theta \leftarrow \theta +\alpha \cdot \nabla \_{\theta }J(\theta )

* This tells the agent what action to take in state s



🎯 What Is Value-Based RL?

Value-based RL focuses on learning **how good it is to be in a state** or **take an action in a state** — so the agent can choose the best actions.

It learns a **value function**:

* V(s): value of being in state s
* Q(s,a): value of taking action a in state s

Then it uses these values to **derive a policy**:

\pi (s)=\arg \max \_aQ(s,a)

🧠 How Does It Work?

The agent:

1. Interacts with the environment
2. Observes transitions (s,a,r,s')
3. Updates its value estimates using **bootstrapping** (TD learning)

# Policy Objective Functions

**Policy objective functions define what a reinforcement learning agent is trying to optimize when learning a policy. The most common objective is to maximize expected cumulative reward, but there are several formulations depending on the algorithm and setting.**

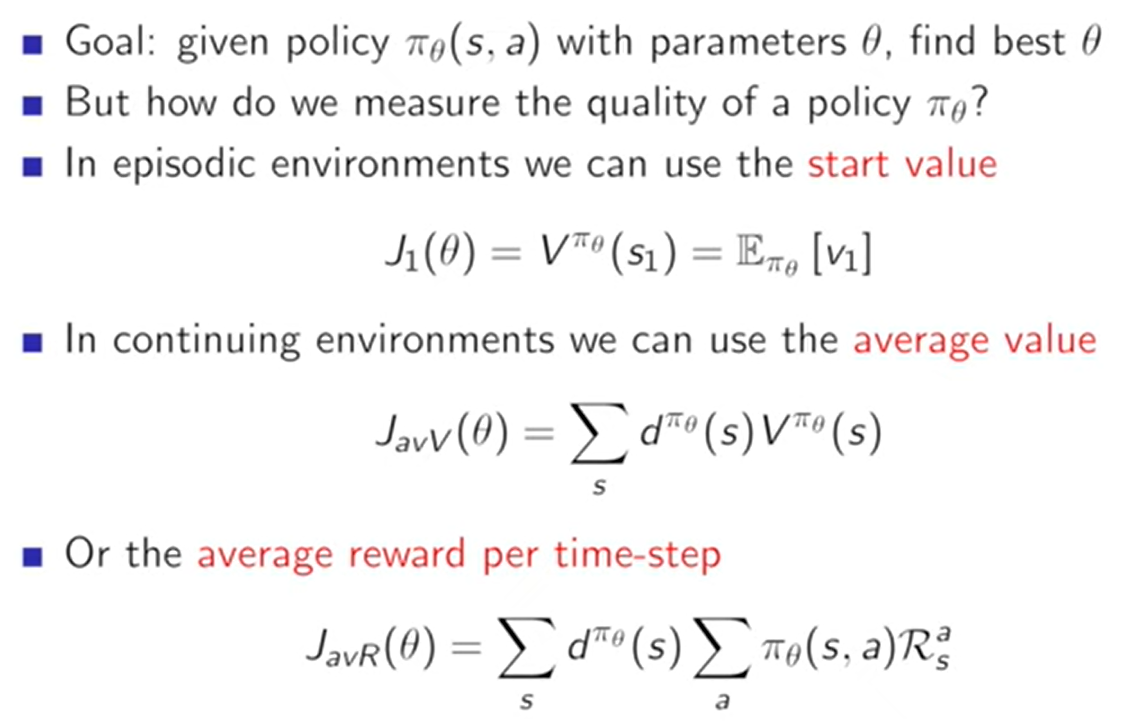
🎯 Core Policy Objective

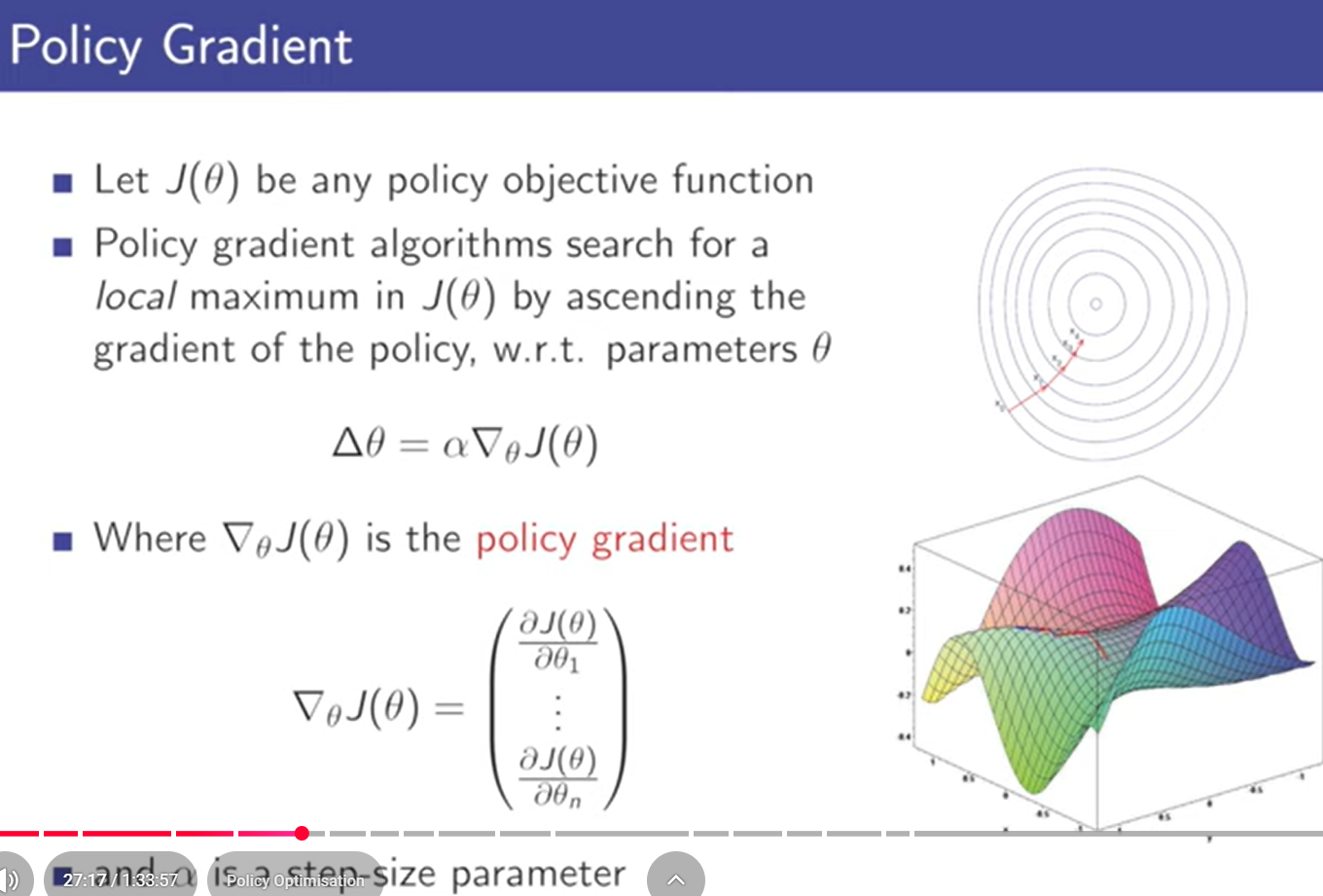
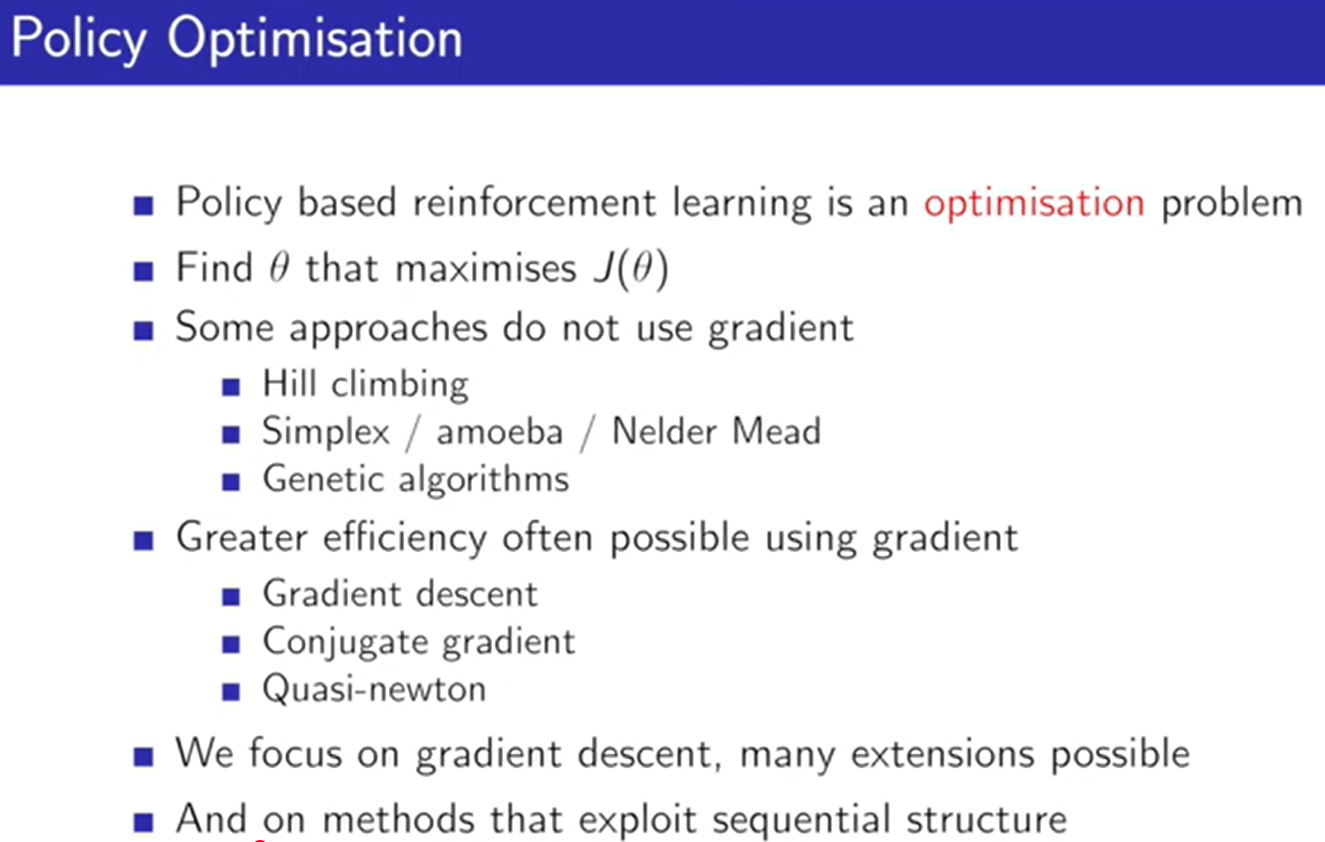
The standard goal in policy-based RL is to **maximize expected return**:

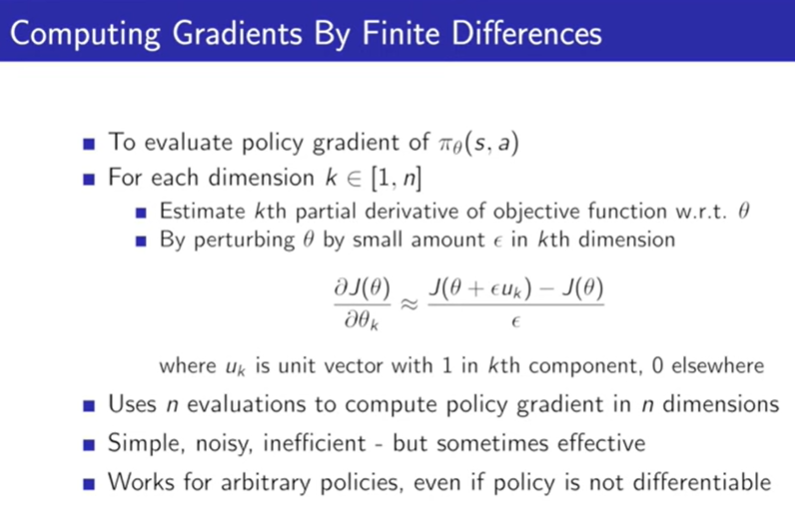
J(\theta )=\mathbb{E\_{\mathnormal{\pi \_{\theta }}}}\left[ \sum \_{t=0}^{\infty }\gamma ^tr\_t\right]

* \pi \_{\theta }: policy parameterized by \theta
* r\_t: reward at time t
* \gamma : discount factor

This objective guides how the policy parameters \theta are updated using gradient ascent.





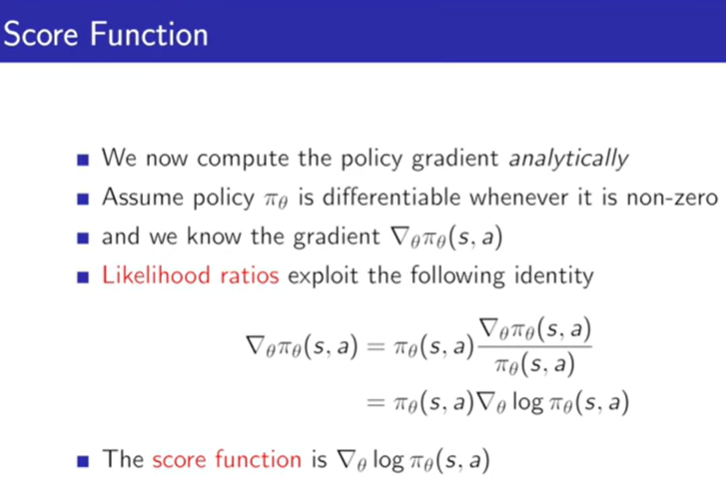
****

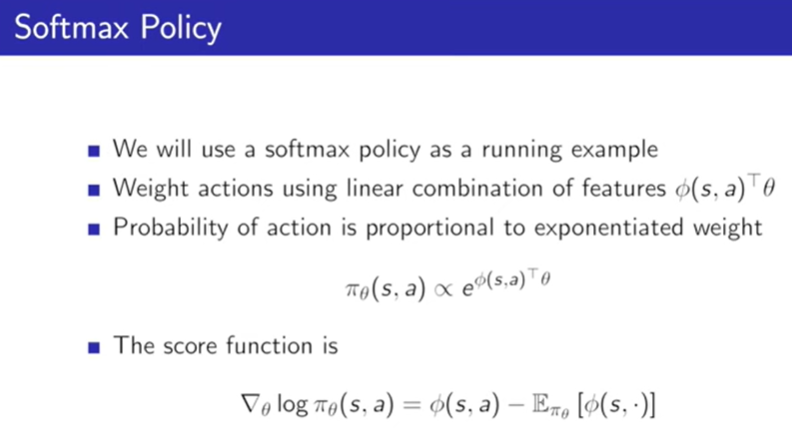
**Finite differences** is a numerical method to estimate the gradient (derivative) of a function by measuring how much the function changes when you slightly change its input.

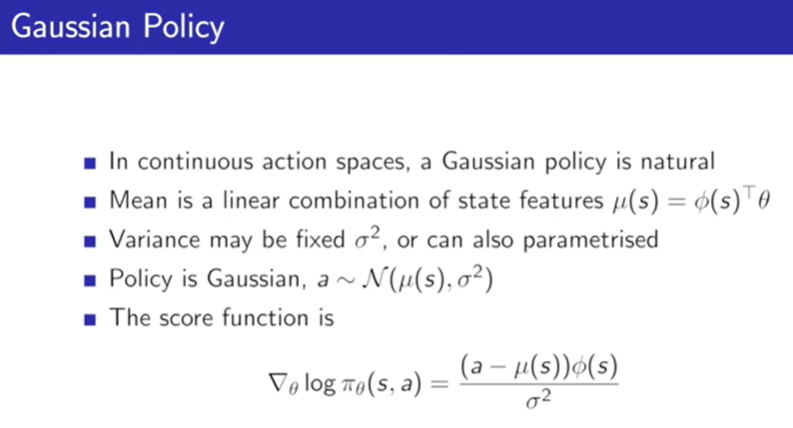
Instead of using calculus, you use:

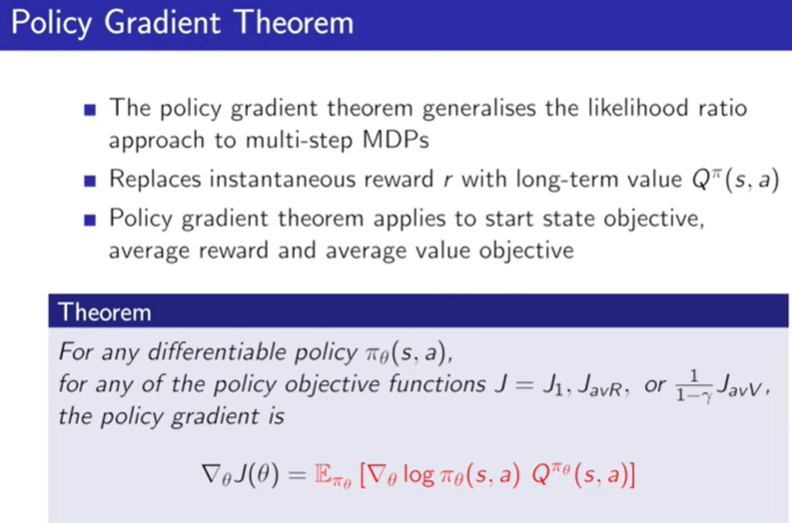
\frac{\partial f(x)}{\partial x}\approx \frac{f(x+\epsilon )-f(x)}{\epsilon }

* f(x): your function (e.g., reward, loss)
* \epsilon : a small number (e.g., 0.01)
* This gives you an **approximate slope** at point x

**The score function is the gradient of the log-probability of a policy with respect to its parameters. It’s used in policy gradient methods to estimate how changing the policy affects expected rewards.**

**A Softmax policy is a stochastic action selection strategy in reinforcement learning where actions are chosen probabilistically based on their estimated values. Higher-value actions are more likely, but all actions have a non-zero chance of being selected.**

**A Gaussian policy is a type of stochastic policy used in reinforcement learning for continuous action spaces, where actions are sampled from a Gaussian (normal) distribution whose mean and variance are learned by the agent.**

**The Policy Gradient Theorem provides a formula for computing the gradient of the expected return with respect to the parameters of a stochastic policy, enabling direct optimization of the policy in reinforcement learning.**

**🧠 What Does It Say?**

If your policy is parameterized by \theta , the goal is to maximize:

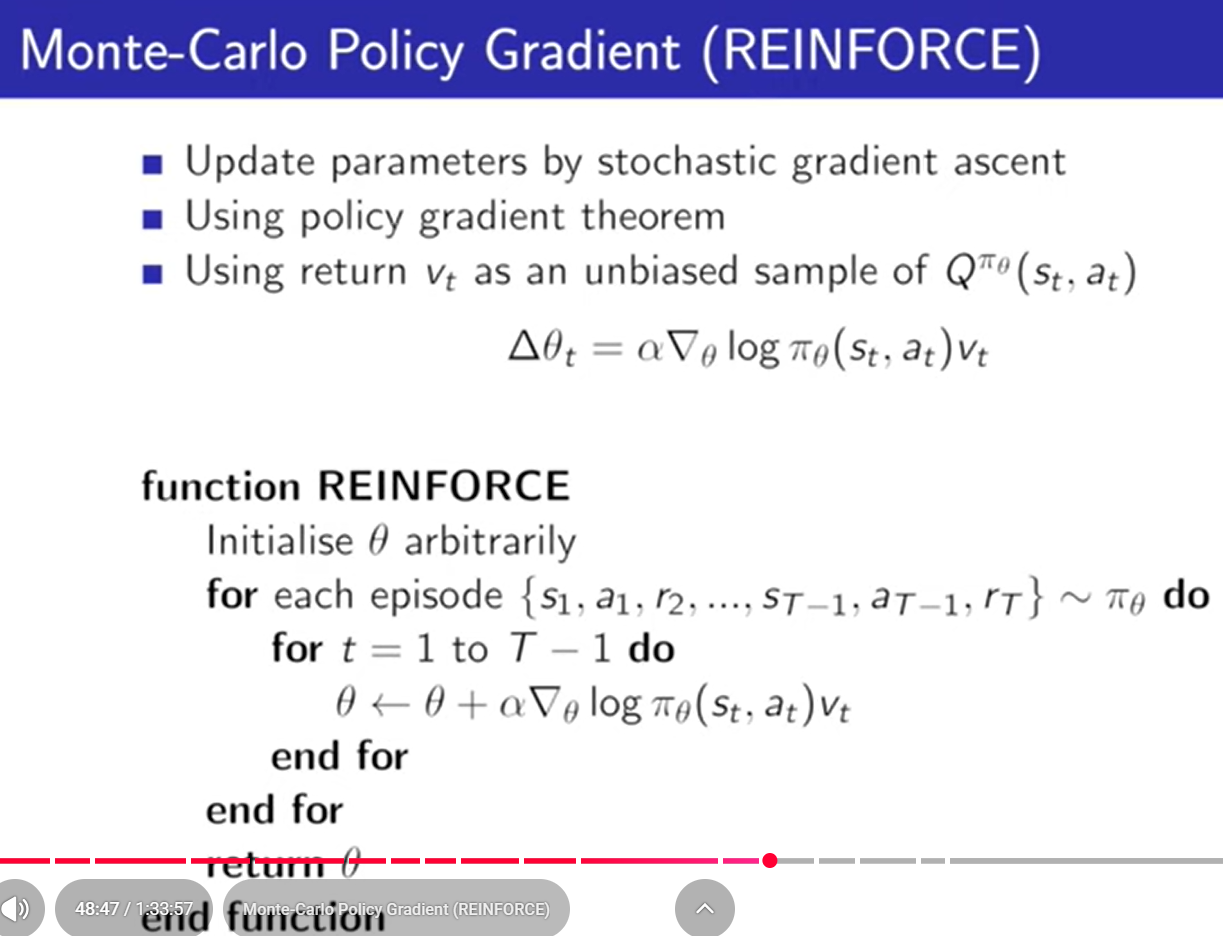
J(\theta )=\mathbb{E\_{\mathnormal{\pi \_{\theta }}}}\left[ \sum \_tr\_t\right]

The Policy Gradient Theorem gives the gradient:

\nabla \_{\theta }J(\theta )=\mathbb{E\_{\mathnormal{\pi \_{\theta }}}}\left[ \nabla \_{\theta }\log \pi \_{\theta }(a\_t|s\_t)\cdot Q^{\pi }(s\_t,a\_t)\right]

This tells us:

To improve the policy, increase the probability of actions that lead to high returns.

**🎯 What Is REINFORCE?**

**REINFORCE is a foundational policy gradient algorithm that uses Monte Carlo sampling to estimate gradients and improve a stochastic policy.**

It directly optimizes the expected return:

J(\theta )=\mathbb{E\_{\mathnormal{\pi \_{\theta }}}}\left[ \sum \_tr\_t\right]

by adjusting the policy parameters \theta using sampled trajectories.

**🧠 Core Update Rule**

After running an episode, REINFORCE updates the policy using:

\theta \leftarrow \theta +\alpha \cdot \nabla \_{\theta }\log \pi \_{\theta }(a\_t|s\_t)\cdot G\_t

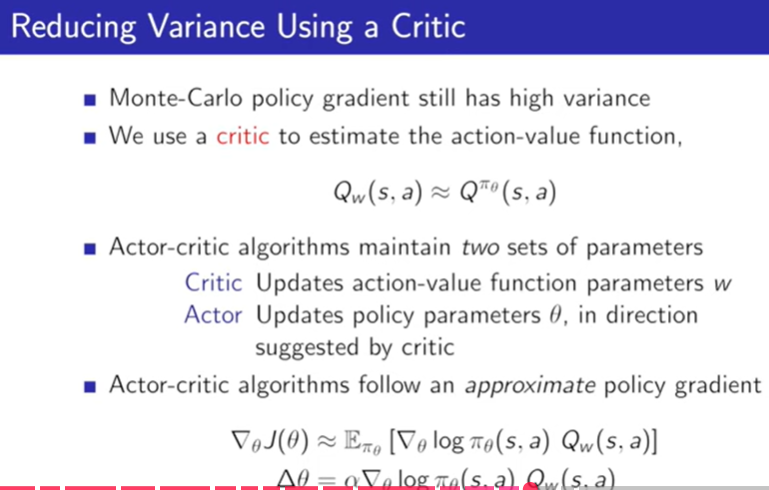
* \alpha : learning rate
* \pi \_{\theta }(a\_t|s\_t): probability of taking action a\_t in state s\_t
* G\_t: return from time t onward (i.e., \sum \_{k=t}^Tr\_k)

This is derived from the policy gradient theorem, using the score function trick.

🔁 Monte Carlo Sampling

* REINFORCE uses **full episodes** to compute returns G\_t
* No bootstrapping or value function — just raw sampled rewards
* This makes it **unbiased**, but **high variance**

# 2.

****

**🎯 The Problem: High Variance in REINFORCE**

In Monte Carlo Policy Gradient (REINFORCE), we estimate the gradient as:

\nabla \_{\theta }J(\theta )=\mathbb{E\_{\mathnormal{\pi \_{\theta }}}}\left[ \nabla \_{\theta }\log \pi \_{\theta }(a\_t|s\_t)\cdot G\_t\right]

* G\_t: total return from time t
* This estimator is unbiased but has high variance, especially in long episodes or sparse reward settings.

**🧠 The Solution: Use a Critic (Baseline)**

We reduce variance by subtracting a baseline from the return:

\nabla \_{\theta }J(\theta )=\mathbb{E\_{\mathnormal{\pi \_{\theta }}}}\left[ \nabla \_{\theta }\log \pi \_{\theta }(a\_t|s\_t)\cdot (G\_t-b(s\_t))\right]

**🔹 What is b(s\_t)?**

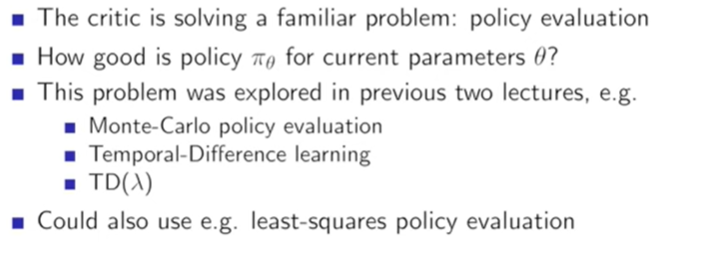
* **A baseline — ideally the value function V^{\pi }(s\_t)**
* **It doesn’t introduce bias, but reduces variance**

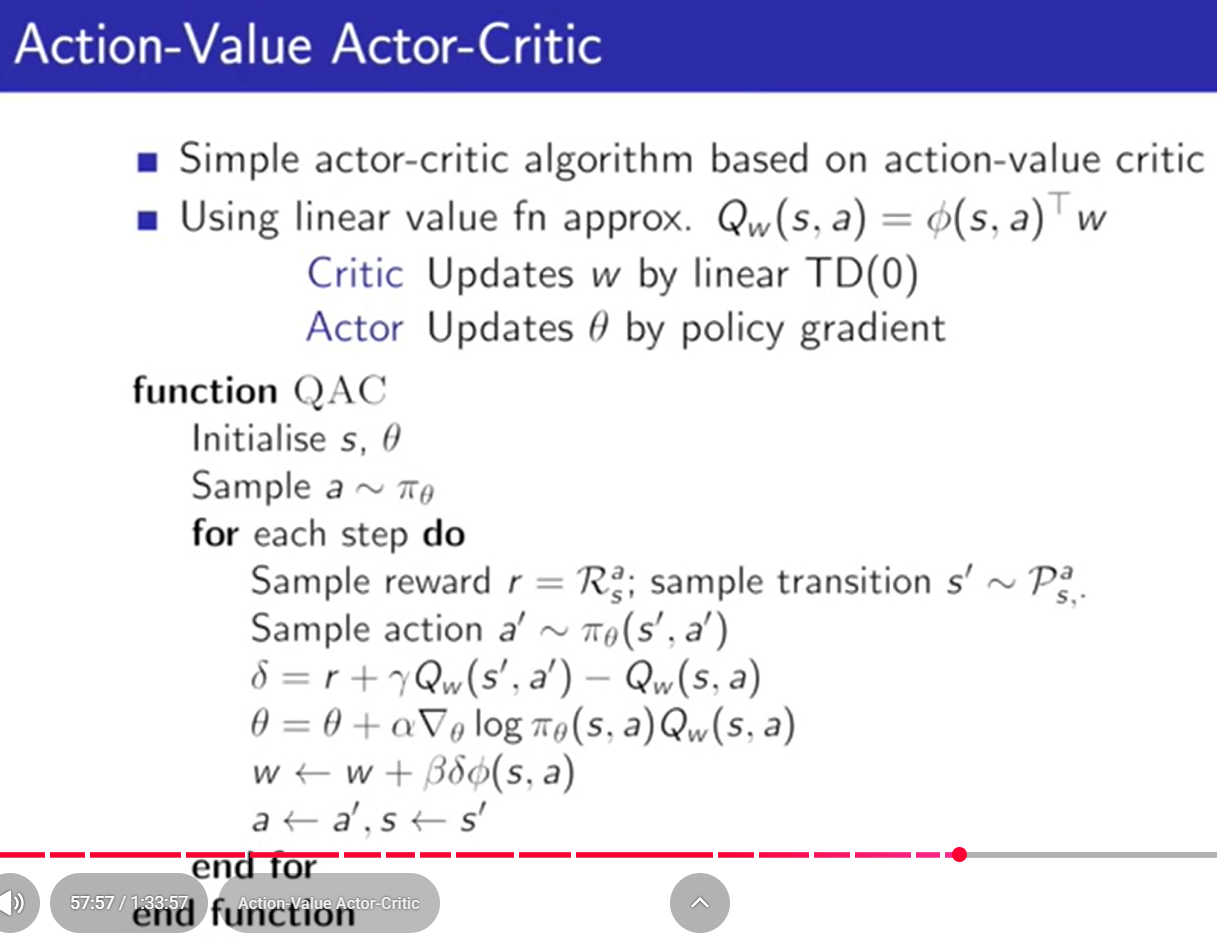
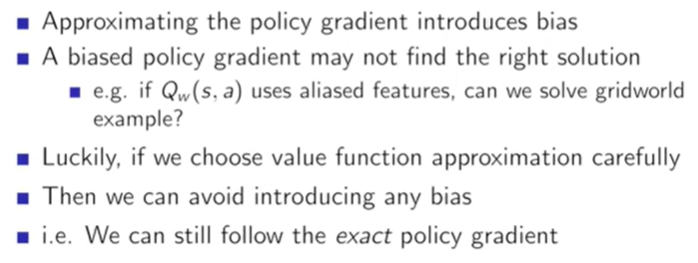
This leads to the Advantage Actor-Critic formulation:

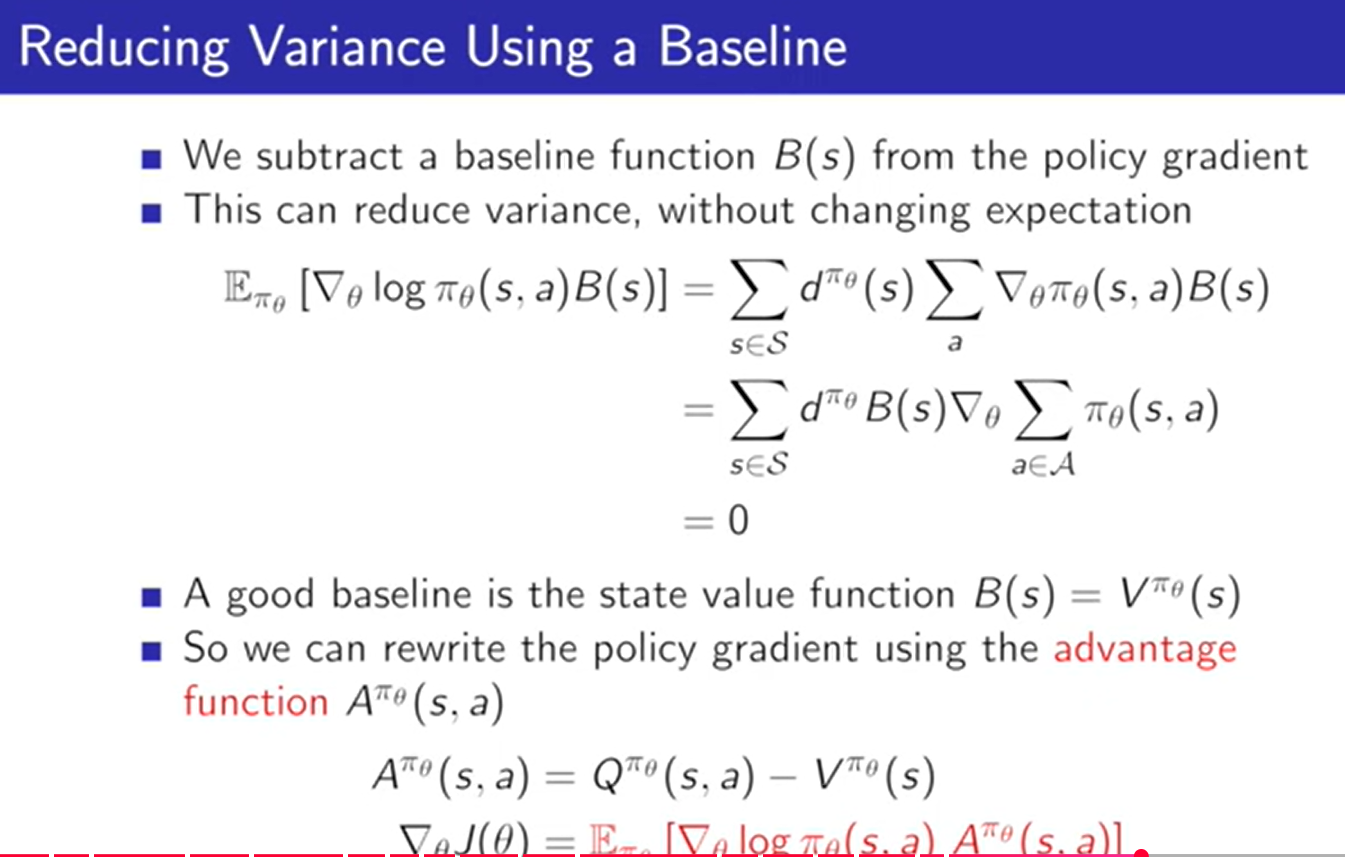
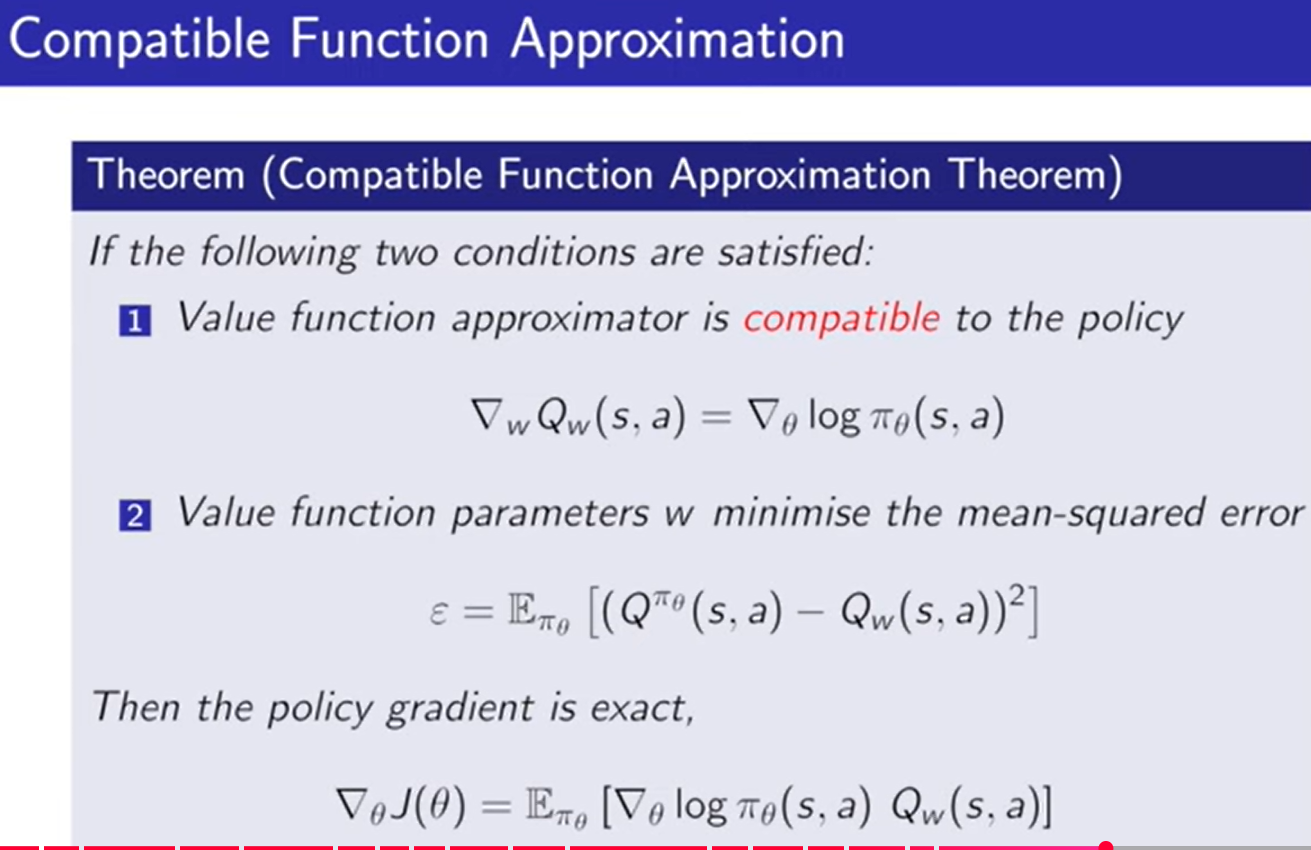
\nabla \_{\theta }J(\theta )=\mathbb{E\_{\mathnormal{\pi \_{\theta }}}}\left[ \nabla \_{\theta }\log \pi \_{\theta }(a\_t|s\_t)\cdot A^{\pi }(s\_t,a\_t)\right]

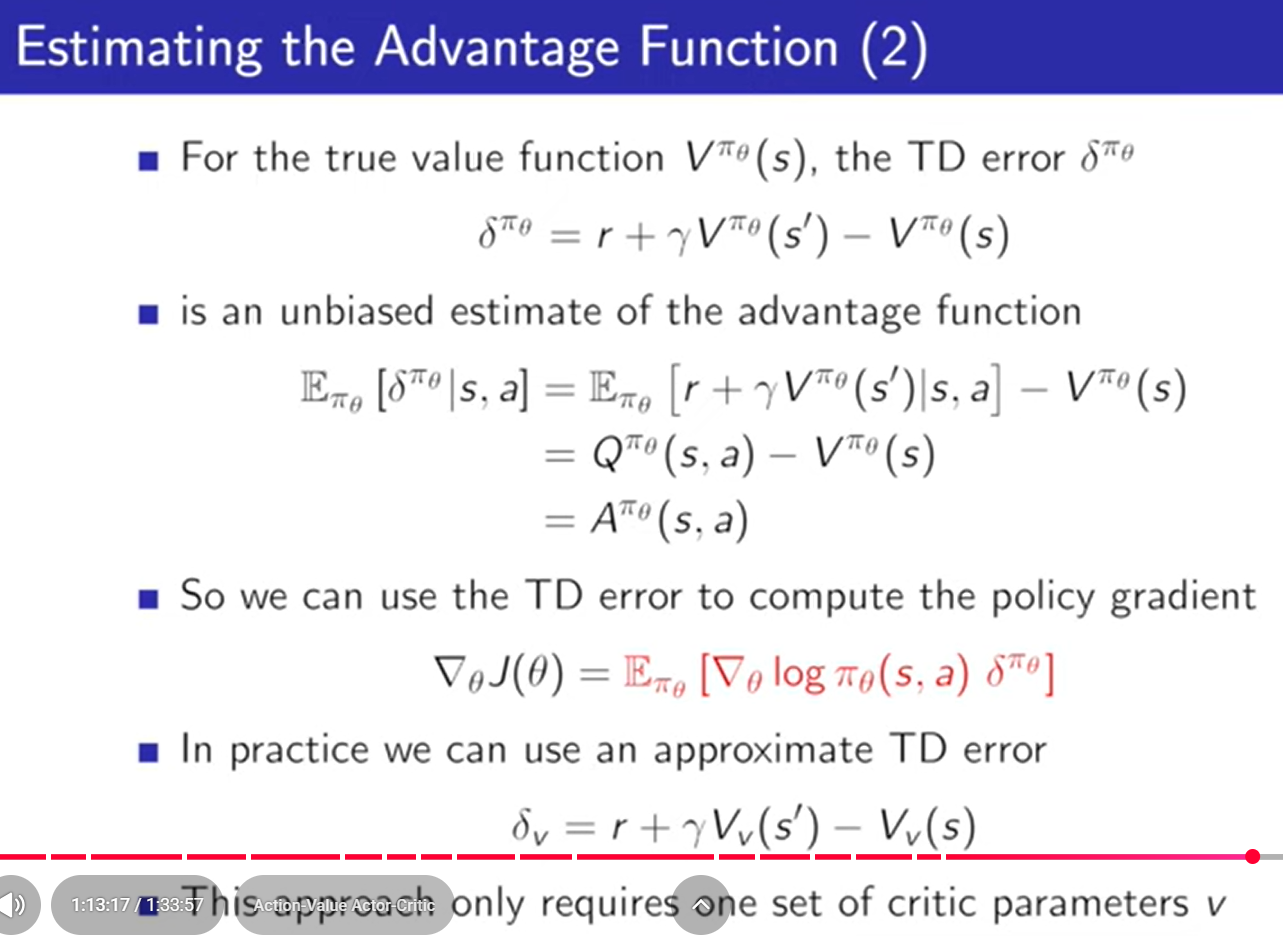
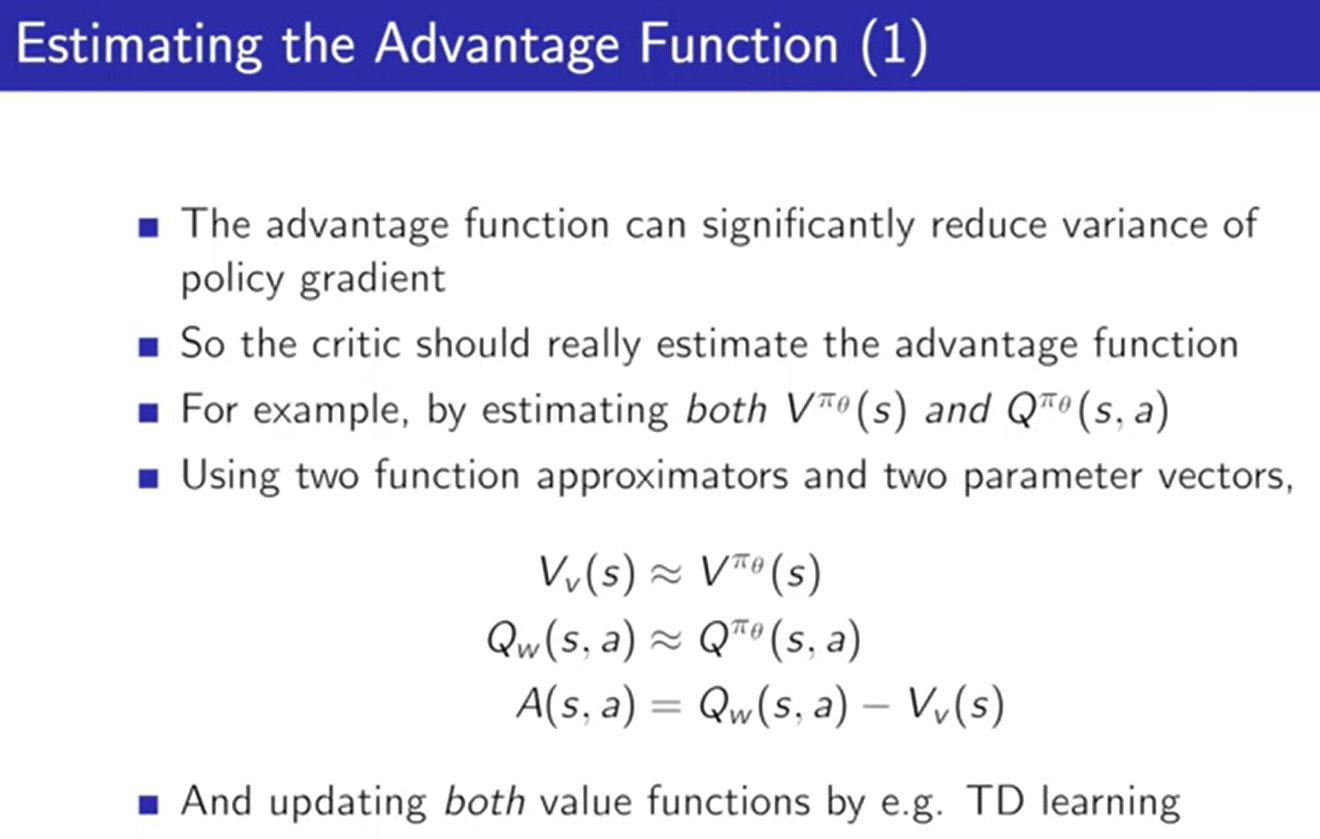
where:

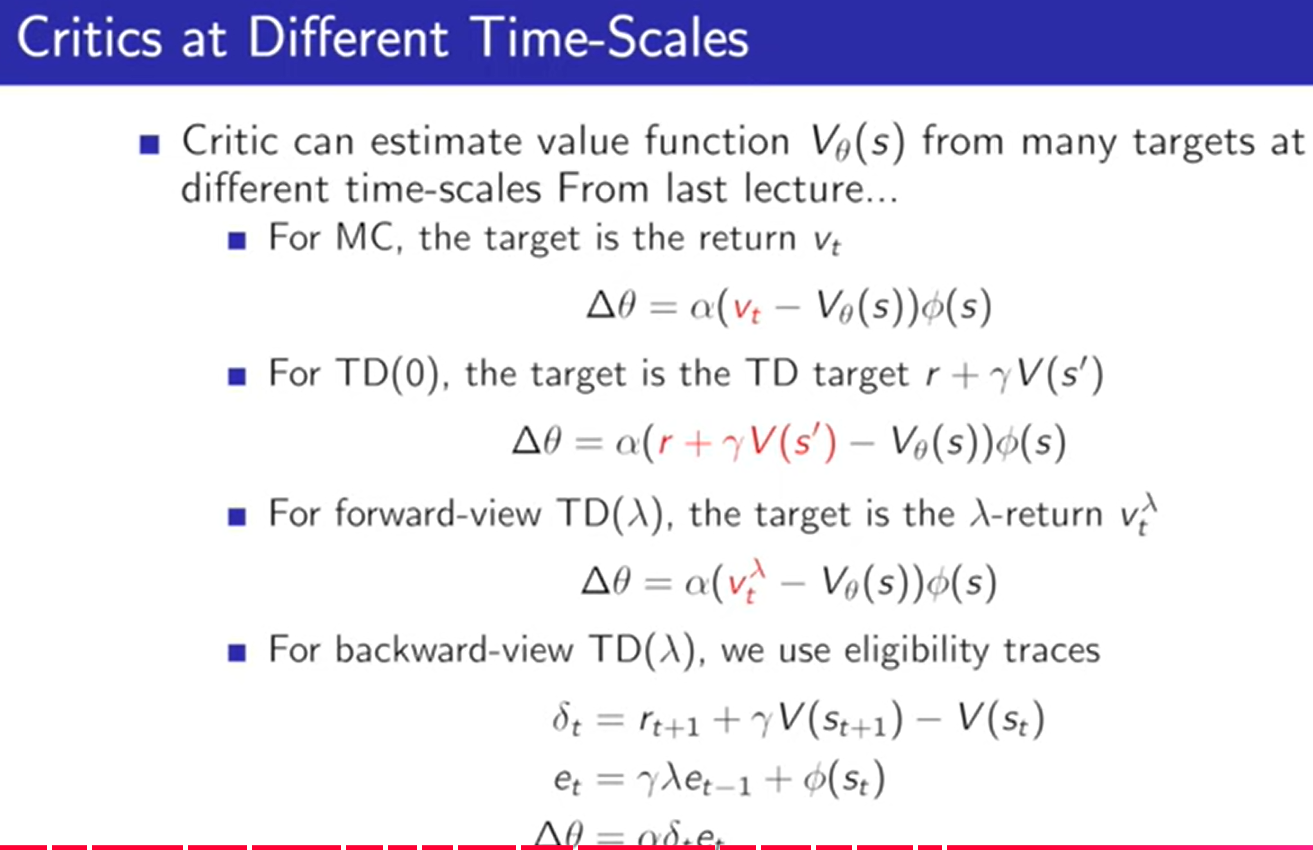
A^{\pi }(s\_t,a\_t)=Q^{\pi }(s\_t,a\_t)-V^{\pi }(s\_t)

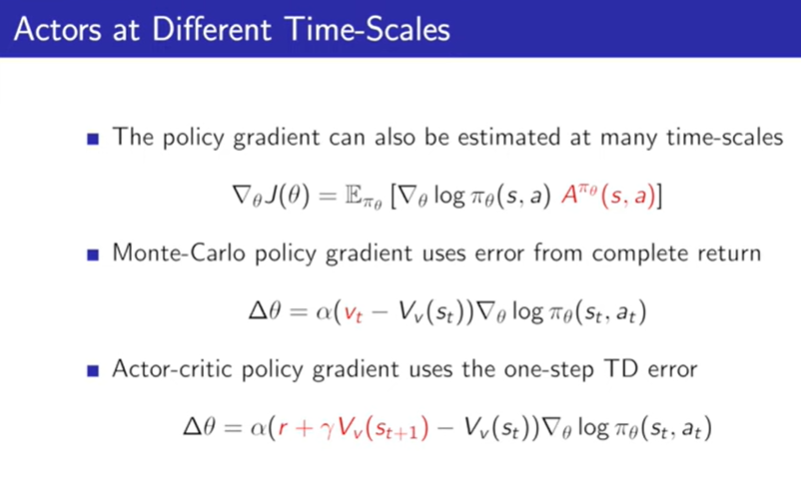










****

**Natural Policy Gradient (NPG) and Natural Actor-Critic (NAC) are advanced policy optimization methods that improve learning stability and efficiency by adjusting the direction of policy updates using the geometry of the policy space. They use the Fisher Information Matrix to scale gradients more intelligently than standard policy gradient methods.**

**🧠 Natural Policy Gradient (NPG)**

**🔹** Core Idea:

Instead of updating the policy using the vanilla gradient:

\theta \leftarrow \theta +\alpha \cdot \nabla \_{\theta }J(\theta )

NPG uses a natural gradient:

\theta \leftarrow \theta +\alpha \cdot F^{-1}\nabla \_{\theta }J(\theta )

* F: Fisher Information Matrix — captures the curvature of the policy space
* This update respects the geometry of the parameter space, leading to more stable and efficient learning

**🎭 Natural Actor-Critic (NAC)**

**🔹 What It Adds:**

NAC combines NPG with a critic that estimates value functions:

* The actor uses natural gradients to update the policy
* The critic estimates V^{\pi }(s) or Q^{\pi }(s,a) to guide the actor

🔧 How It Works:

* The critic uses compatible function approximation to ensure that the estimated advantage aligns with the policy gradient
* The actor update becomes:

\theta \leftarrow \theta +\alpha \cdot F^{-1}\cdot \mathbb{E}[\nabla \_{\theta }\log \pi \_{\theta }(a|s)\cdot A(s,a)]

