Policy Gradient Methods

In value-based methods, we learn a value function.

* The idea is that an optimal value function leads to an optimal policy.
* Our objective is to minimize the loss between the predicted and target value to approximate the true action-value function.
* We have a policy, but it is implicit since it is generated directly from the value function. For instance, in Q-Learning, we used an (epsilon-)greedy policy.

On the other hand, in policy-based methods, we directly learn to approximate the policy without having to learn a value function.

* The idea is to parameterize the policy. For instance, using a neural network, this policy will output a probability distribution over actions (stochastic policy).
* Our objective then is to maximize the performance of the parameterized policy using gradient ascent.
* To do that, we control the parameter θ that will affect the distribution of actions over a state.

Consequently, thanks to policy-based methods, we can directly optimize our policy to output a probability distribution over actions (a∣s) that leads to the best cumulative return. To do that, we define an objective function J(θ), that is, the expected cumulative reward, and we want to find the value θ that maximizes this objective function.

The difference between policy-based and policy-gradient methods:

The difference between these two methods lies on how we optimize the parameter θ:

* In **policy-based** methods, we search directly for the optimal policy. We can optimize the parameter θ indirectly by maximizing the local approximation of the objective function with techniques like hill climbing, simulated annealing, or evolution strategies.
* In **policy-gradient** methods, because it is a subclass of the policy-based methods, we search directly for the optimal policy. But we optimize the parameter θ directly by performing the gradient ascent on the performance of the objective function J(θ).

The advantages of policy-gradient methods:

1. **The simplicity of integration**: We can estimate the policy directly without storing additional data (action values).
2. **Policy-gradient methods can learn a stochastic policy**: Policy-gradient methods can learn a stochastic policy while value functions can’t.

This has two consequences:

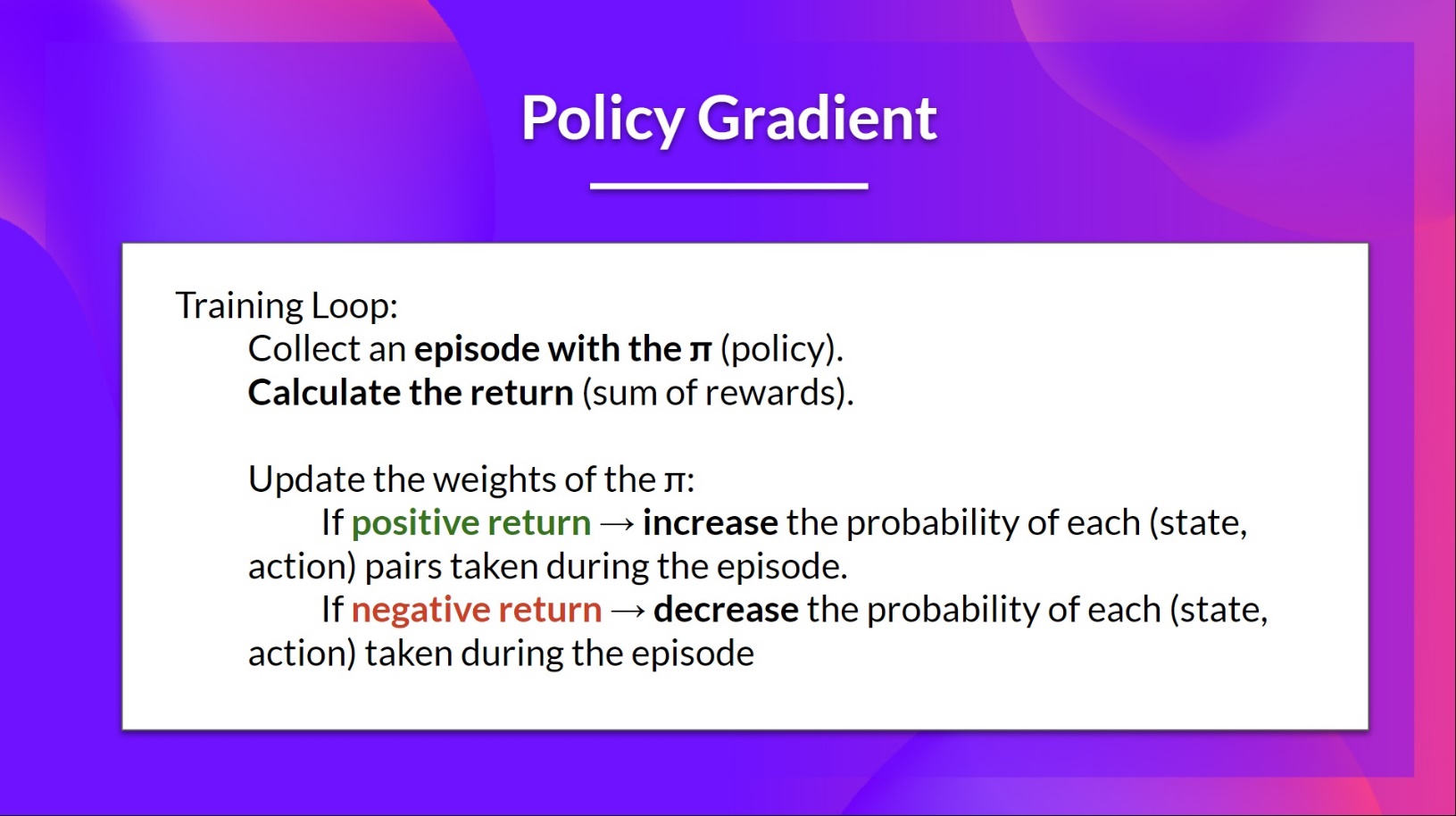
* We don’t need to implement an exploration/exploitation trade-off by hand. Since we output a probability distribution over actions, the agent explores the state space without always taking the same trajectory.
* We also get rid of the problem of perceptual aliasing. Perceptual aliasing is when two states seem (or are) the same but need different actions.

1. **Policy-gradient methods are more effective in high-dimensional action spaces and continuous actions spaces**: Example: A self-driving car, at each state, you can have a (near) infinite choice of actions (turning the wheel at 15°, 17.2°, 19,4°, honking, etc.)
2. **Policy-gradient methods have better convergence properties**: In policy-gradient methods, stochastic policy action preferences (probability of taking action) change smoothly over time.

The disadvantages of policy-gradient methods:

1. Frequently, policy-gradient methods converge to a local maximum instead of a global optimum.
2. Policy-gradient goes slower, step by step: it can take longer to train (inefficient).
3. Policy-gradient can have high variance. We’ll see in the actor-critic unit why, and how we can solve this problem.

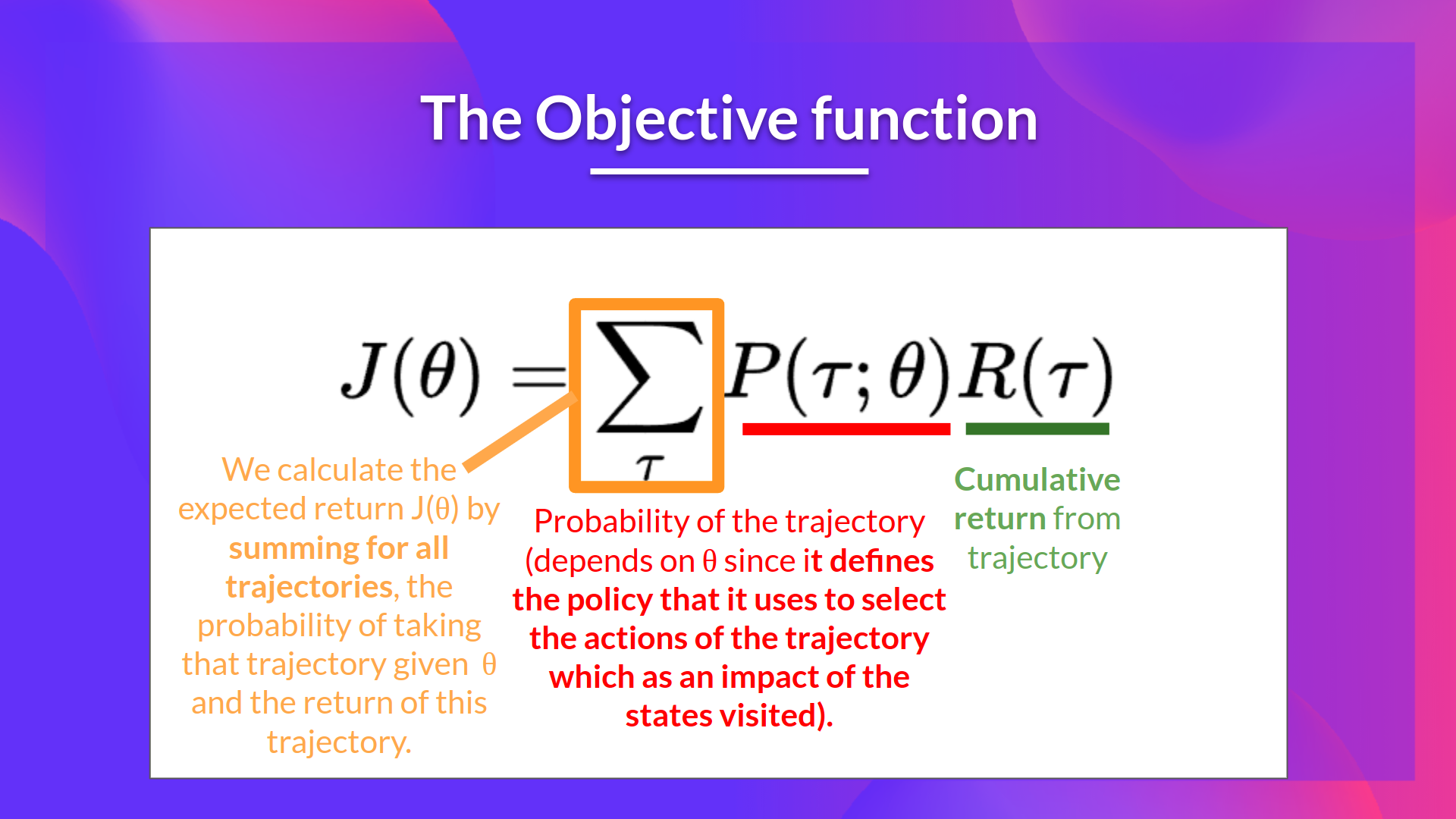
The Policy-Gradient Algorithm:



The idea is that we’re going to let the agent interact during an episode. And if we win the episode, we consider that each action taken was good and must be more sampled in the future since they lead to win.

So, for each state-action pair, we want to increase the P(a∣s): the probability of taking that action at that state. Or decrease if we lost.

Objective Function:



The Reinforce algorithm (Monte Carlo Reinforce):

The Reinforce algorithm, also called Monte-Carlo policy-gradient, is a policy-gradient algorithm that uses an estimated return from an entire episode to update the policy parameter θ.

