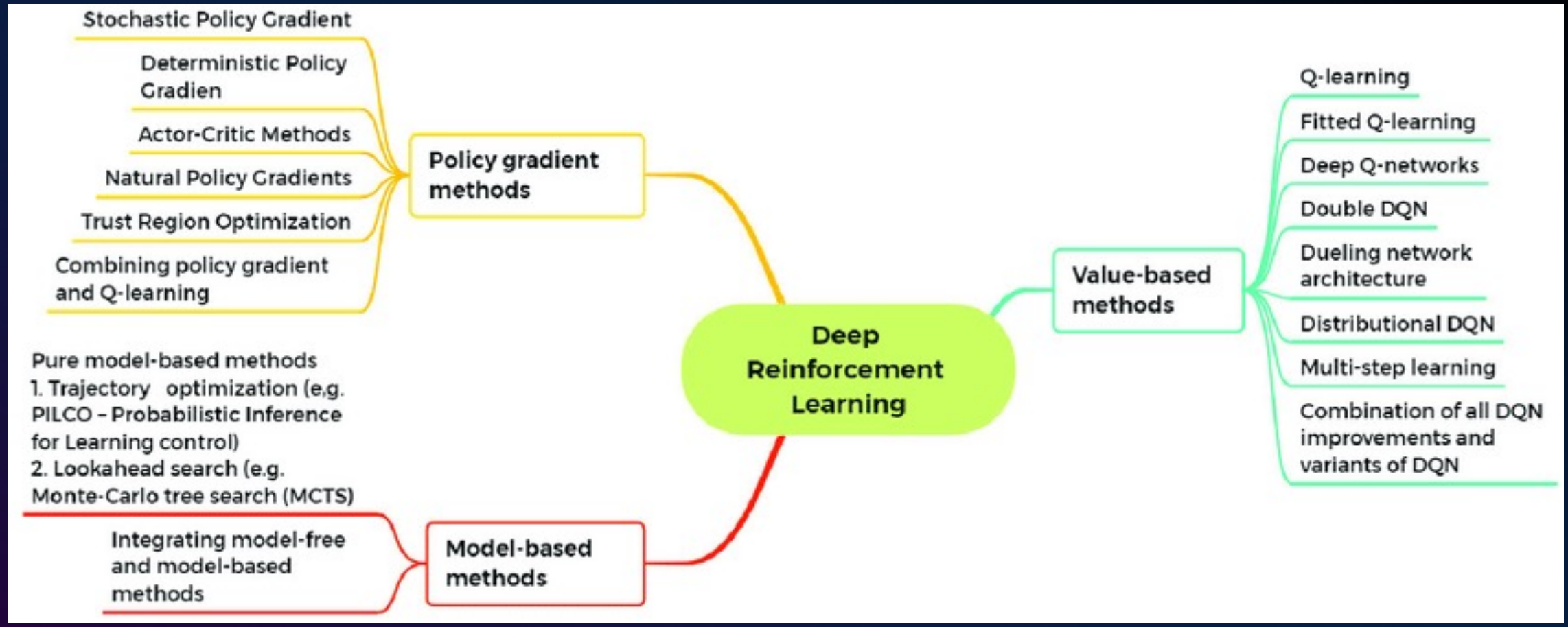
REINFORCEMENT LEARNING

RL – Policy based algorithms

Irshad Chohan

Principal Solutions Architect
AWS India

RL – Learning algorithms



General overview

- **Model-based RL**
  - \+ 'Easy' to learn a model (supervised learning)
  - \+ Learns 'all there is to know' from the data
  - \- Uses compute & capacity on irrelevant details
  - \- Computing policy (=planning) is non-trivial and expensive (in compute)
- **Value-based RL**
  - \+ Easy to generate policy (e.g., $\pi(a|s) = \mathcal{I}(a = \underset{a}{\text{argmax}}\, q(s,a))$)
  - \+ Close to true objective
  - \+ Fairly well-understood, good algorithms exist
  - \- Still not the true objective:
    - May focus capacity on irrelevant details
    - Small value error can lead to larger policy error

Policy Function

- Specifies the agent's behaviour by mapping states to actions
- Learns the optimal policy to maximize reward over time
- Uses techniques like Policy gradient, Deep-Q-Learning.
- NN are used to parameterized the policy directly, outputting probabilities for each possible action.
- Require more space, as policy method evaluate each action through trial-and-error.
- Encourage diversity by optimizing for high policy entropy.

Value Function

- Estimates long-term reward for a given state or state-action
- Helps guide the policy towards higher reward states
- Common techniques involve temporal difference learning and Monte Carlo evaluation
- NN are commonly used to represent state-action value(Q) function.
- Require less environment samples during training. By learning a value function. They can estimate reward impact of unexplored actions.
- Uses epsilon-greedy to balance between exploration vs exploitation.

Advantages and disadvantages of policy-based RL

Advantages:

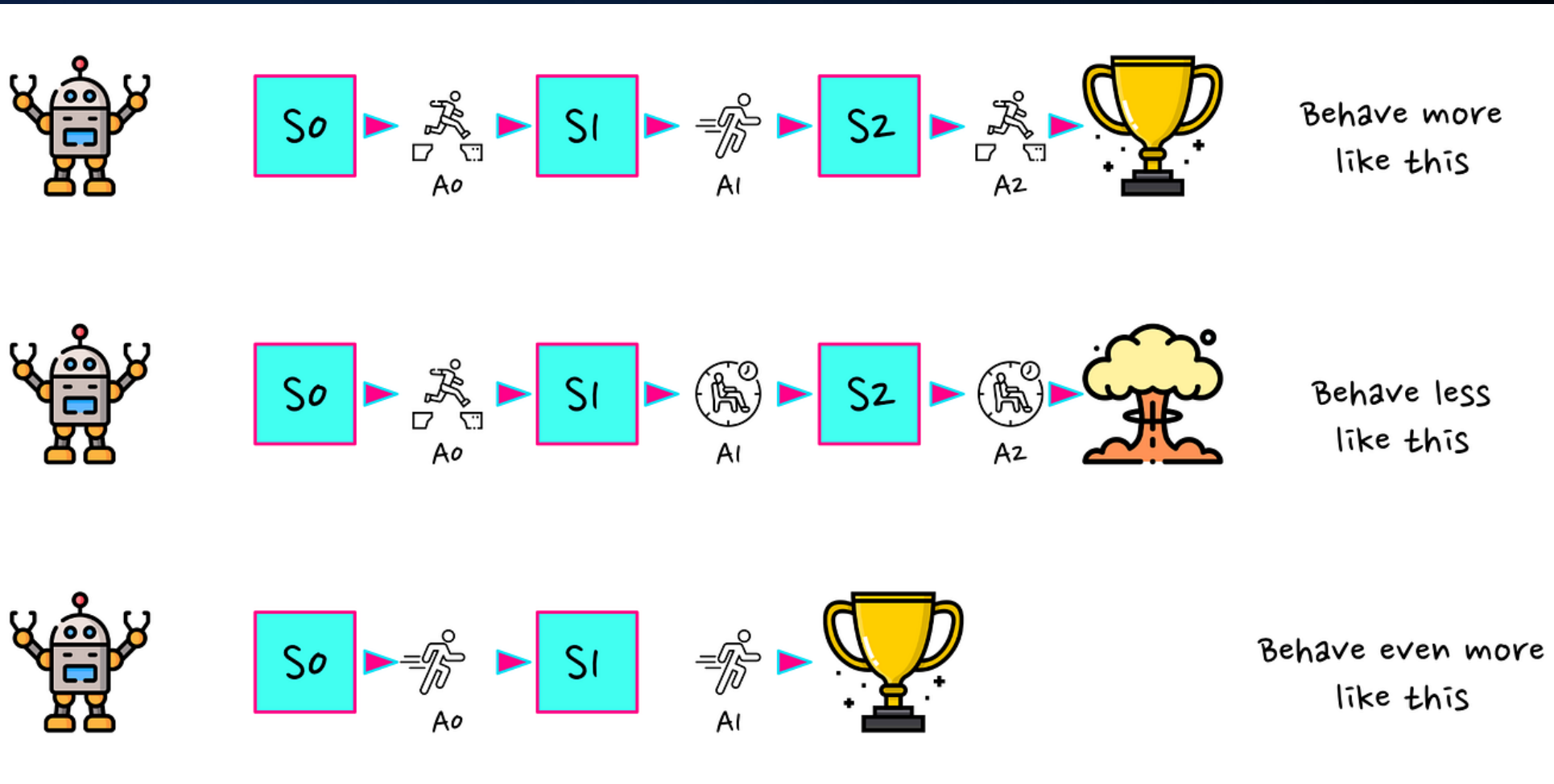
- True objective
- Easy extended to **high-dimensional** or **continuous** action spaces
- Can learn **stochastic** policies
- Sometimes policies are **simple** while values and models are complex
  - E.g., complicated dynamics, but optimal policy is always “move forward”

Disadvantages:

- Could get stuck in local optima
- Obtained knowledge can be **specific**, does not always generalise well
- Does not necessarily extract all useful information from the data (when used in isolation)

Policy gradient algorithms

Policy gradient algorithms attempt to improve the policy directly, by changing the parameters of the policy function, such that the policy produces better results.



Terms used in Policy Gradient model

- The **policy** is the robot's brain that decides what to do in each situation.
- The **objective** is to make the robot as successful as possible at delivering pizzas by maximizing rewards.
- The **policy gradient theorem** helps us adjust the robot's brain by learning from the feedback (rewards) it gets after each delivery.
- **Gradient ascent** is the process of tweaking the robot's brain to improve its performance step by step.
- The **advantage function** refines this process by telling the robot how much better (or worse) a specific decision was compared to others.

Stochastic policies

Why could we need stochastic policies?

- In MDPs, there is always an optimal **deterministic** policy
- But, most problems are **not fully observable**
  - This is the common case, especially with function approximation
  - The optimal policy may then be stochastic
- Search space is smoother for stochastic policies $\Longrightarrow$ we can use gradients
- Provides some 'exploration' during learning

1. Policy (π)

The **policy** (denoted as $\pi(a|s;\theta)$) is a probability distribution over actions, given a state s. The parameters θ (weights) control this probability distribution.

- **Analogy**: Think of the policy like the robot's brain (our pizza-delivering robot), which decides how likely it is to turn left or right based on what it sees (the state). Initially, the robot has a random idea of what to do, but we want to train it to make better choices over time.

In **policy gradient**, the goal is to **improve this policy** so it takes actions that lead to higher rewards.

Policy objective functions

- Goal: given **policy $\pi_\theta(s, a)$**, find best **parameters θ**
- How do we measure the quality of a policy π_θ?
- In episodic environments we can use the **average total return per episode**
- In continuing environments we can use the **average reward per step**

Policy based learning algorithms are optimization problems

2. Objective Function (J(θ))

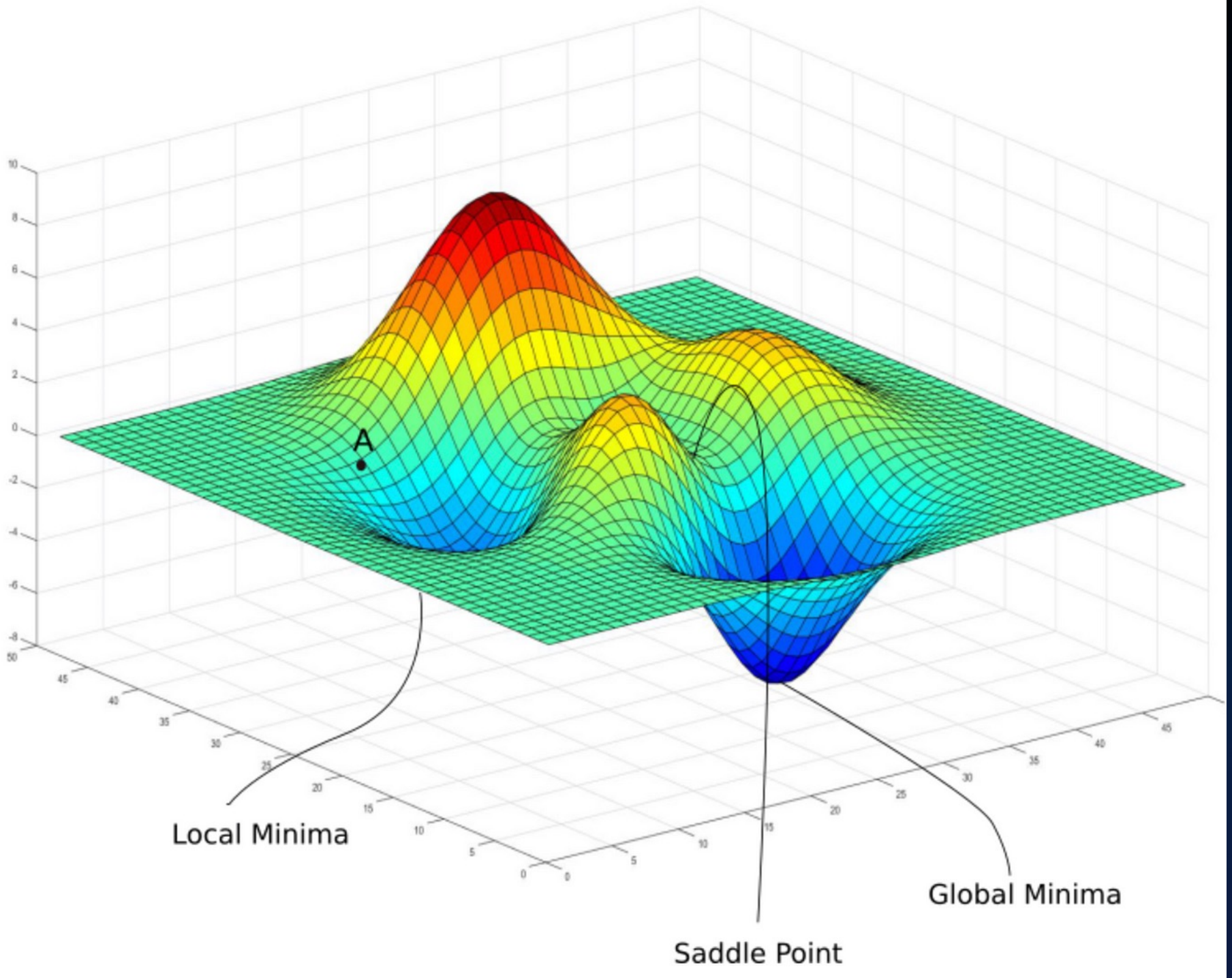
The objective in policy gradient is to **maximize the expected reward** over time. The expected reward depends on the sequence of actions taken, and we want to tweak the policy to increase the chances of taking actions that lead to more rewards.

This is written as:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

Where τ is a trajectory (sequence of states and actions), and $R(\tau)$ is the total reward over that trajectory.

- **Analogy**: In our pizza-delivery robot example, $J(\theta)$ is like measuring how successful the robot is at delivering pizzas on time. The robot wants to maximize its success (rewards), so we need to adjust its brain (policy) to help it learn the best routes.



Let’s understand
how policy optimization
Works, with
the help of
stochastic gradient.

3. Policy Gradient Theorem

Now, how do we improve the policy? The key is to **compute the gradient** of the objective function $J(\theta)$ with respect to the policy parameters θ, so we can adjust the policy to perform better.

The **policy gradient theorem** gives us a way to calculate this gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) \cdot R(\tau) \right]$$

Here's what's going on:

- $\nabla_\theta \log \pi_\theta(a|s)$: This term measures how sensitive the policy is to its parameters θ, i.e., how much the probability of choosing action a changes with respect to the policy's parameters.
- $R(\tau)$: This is the reward received over the entire trajectory.
- **Analogy**: Imagine you are adjusting the robot's brain (policy). $\nabla_\theta \log \pi_\theta(a|s)$ is like figuring out how a slight change in the robot's decision-making (e.g., increasing the likelihood of turning right at an intersection) would affect its overall success (reward). The robot then uses this feedback to tweak its behavior.

4. Updating the Policy: Gradient Ascent

Once we have the gradient, we can update the policy parameters θ using **gradient ascent**. We move in the direction of the gradient to maximize the expected reward.

The update rule is:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Where α is the learning rate.

- **Analogy**: Think of this like teaching the robot to deliver pizzas. After each delivery, you give feedback on how good the delivery was (reward) and tell the robot to slightly tweak its strategy (policy) based on what worked. Over time, the robot keeps adjusting itself to get better at deliveries.

5. Exploration vs. Exploitation

In policy gradient, there's always a trade-off between exploring new actions (exploration) and repeating actions that are known to give good rewards (exploitation).

The **policy** itself naturally encourages both exploration and exploitation since it's a probability distribution. Early in learning, the robot may try random actions to explore different routes. As it gets better, it'll exploit actions that have led to faster deliveries.

- **Analogy**: Early in the pizza-delivery robot's training, it might take random turns to explore the city (exploration). But as it learns, it'll start choosing the routes it knows will likely lead to faster deliveries (exploitation).

6. Advantage Actor-Critic (A2C) Variant

One problem with the basic policy gradient is that it can be inefficient because we're using the total reward for an entire trajectory. To improve this, we often use an **Advantage Function (A(s, a))**, which measures how good an action is compared to the average action at a state.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) \cdot A(s,a) \right]$$

Where:

$$A(s,a) = Q(s,a) - V(s)$$

- **Analogy**: In the pizza robot example, the advantage function tells the robot how much better taking a particular turn was compared to just wandering around. It's like giving the robot more specific feedback: "Turning right at this corner was better than your average decision."

Quiz

What is the primary goal of the policy gradient method in reinforcement learning?

- A) To find the best reward function
- B) To directly optimize the policy without using a value function
- C) To approximate the state value
- D) To minimize the policy's loss function

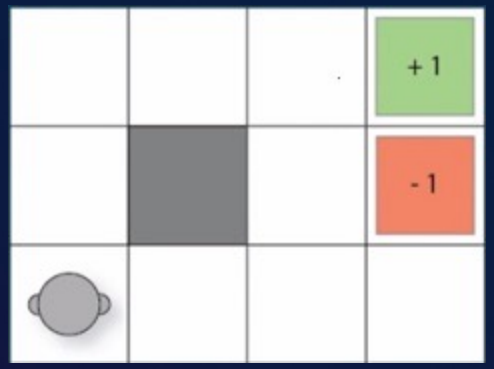
In policy gradient methods, what is the objective function typically optimized?

- A) The total cumulative reward of the policy
- B) The gradient of the state-action value function
- C) The likelihood of selecting actions based on the value function
- D) The expected return of the policy

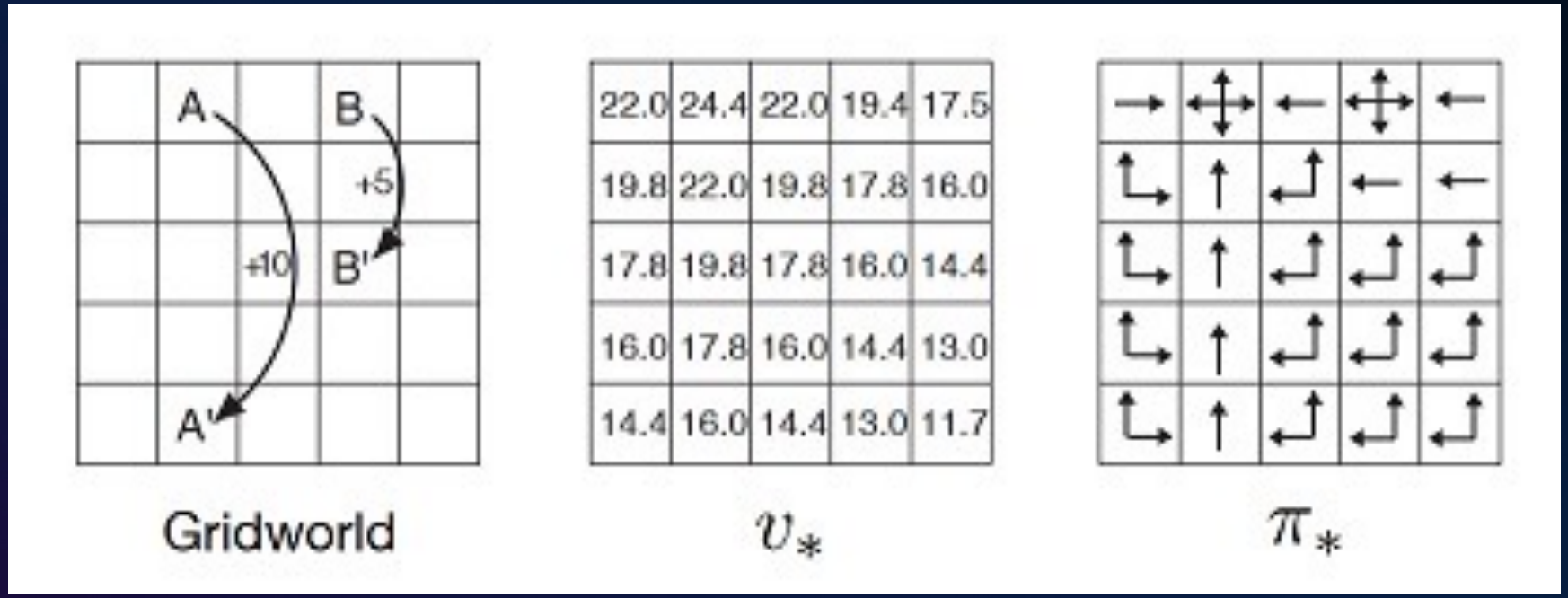
What is a major advantage of policy gradient methods over value-based methods like Q-learning?

- A) Policy gradient methods are easier to implement
- B) They are more stable when approximating continuous action spaces
- C) They do not require reward signals
- D) They are faster to converge in all scenarios

Grid World demo - workshop



Grid World is a game for demonstration. 12 positions, 11 states, 4 actions. Our aim is to find optimal policy.

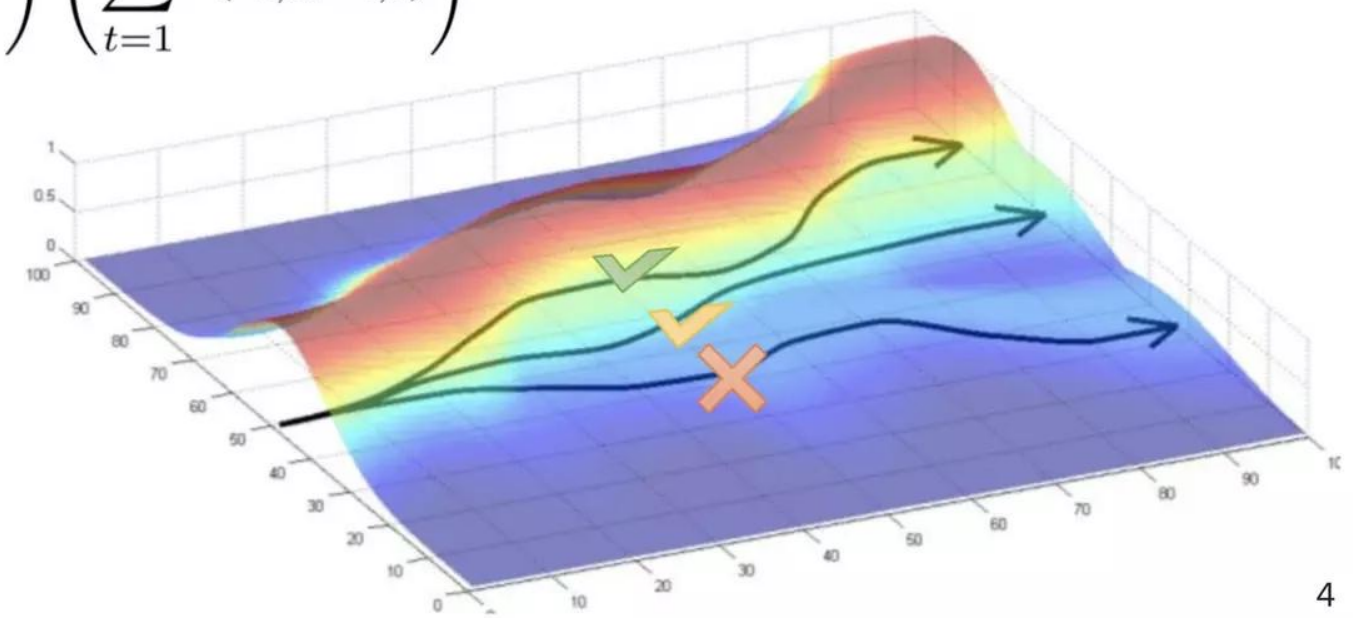


Actor-Critic Algorithm

Recap: policy gradient

REINFORCE algorithm:

1. sample trajectory τ^i from $\pi_\theta(a_t|s_t)$
2. $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$
3. $$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$$



Recap: policy gradient

REINFORCE algorithm:

1. sample trajectory τ^i from $\pi_\theta(a_t|s_t)$
2. $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$
3. $$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$$

Issue: We need to sample whole trajectory to get this term (Monte Carlo)

→ Make policy gradient learn slowly.

Recap: policy gradient

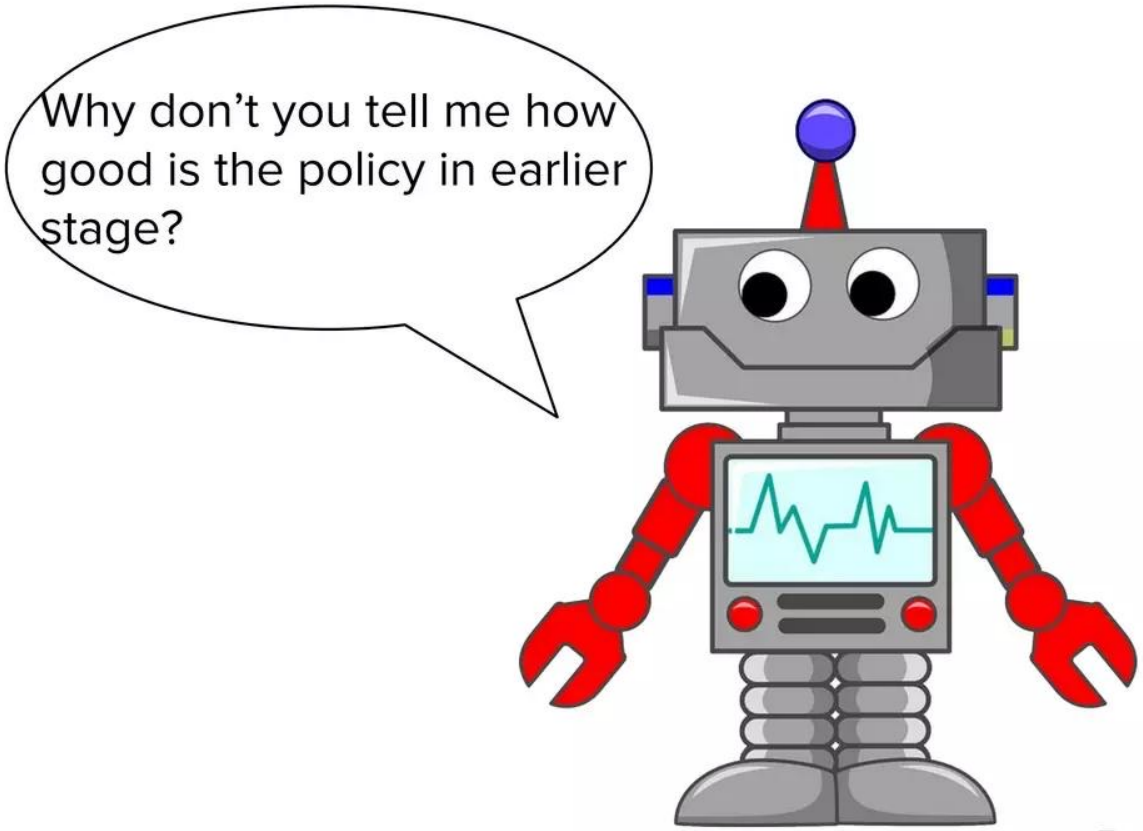
REINFORCE algorithm:

1. sample trajectory τ^i from $\pi_\theta(a_t|s_t)$
2. $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$
3. $$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$$

Can we learn step by step?

Actor-Critic algorithm

In vanilla policy gradient, we only can evaluate our policy when we finish the whole episode.



Actor-Critic algorithm

In vanilla policy gradient, we only can evaluate our policy when we finish the whole episode.

Actor-Critic algorithm

Objective of vanilla policy gradient:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \left[\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right]$$

$$\widehat{Q}(s_{i,t}, a_{i,t})$$

The return in policy gradient with causality in step ***t*** could be replaced by expected action-value function.

If we could find the action-value function in each step, we can improve learning efficiency by TD learning.

Actor-Critic algorithm

Policy gradient with causality:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \left[\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \, \widehat{Q}(s_{i,t}, a_{i,t}) \right]$$

Question: How do we get action-value function Q?

Actor-Critic algorithm

Policy gradient with causality:

Policy Network

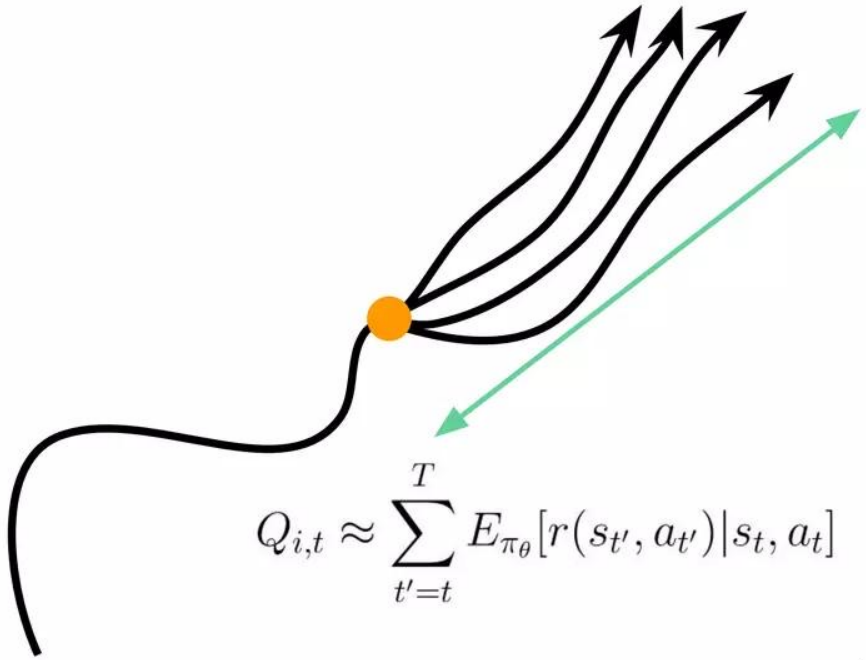
Critic Network

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\left[\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\,\widehat{Q}(s_{i,t},a_{i,t})\right]$$

Question: How do we get action-value function Q?

Using another neural network to approximate value function called Critic. This is so-called Actor-Critic.

By using Critic network, we can update the neural network step by step. However, it will also introduce bias.



Actor-Critic algorithm

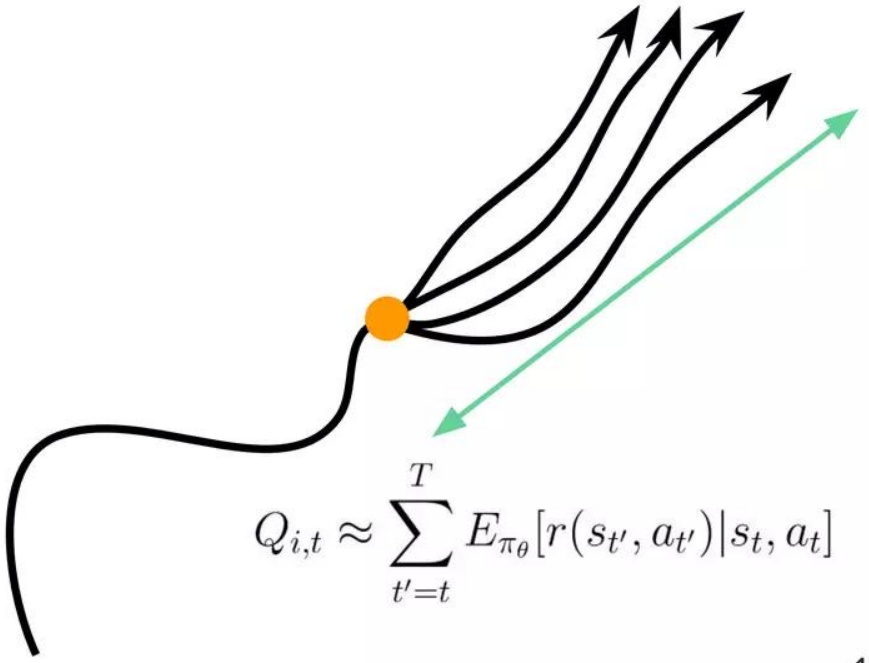
The objective of Actor-Critic:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \left[\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \, \widehat{Q}(s_{i,t}, a_{i,t}) \right]$$

This objective function in this version have lower variance and higher bias than REINFORCE when we learning by TD learning.

Can we also subtract a baseline to reduce the variance?
Yes! we could subtract this term:

$$V(s_t) = E_{a_t \sim \pi_\theta(a_t|s_t)} \left[Q(s_t, a_t) \right]$$



Actor-Critic algorithm

The objective of Actor-Critic with value function baseline:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\big(Q(s_{i,t},a_{i,t}) - V(s_{i,t})\big)$$

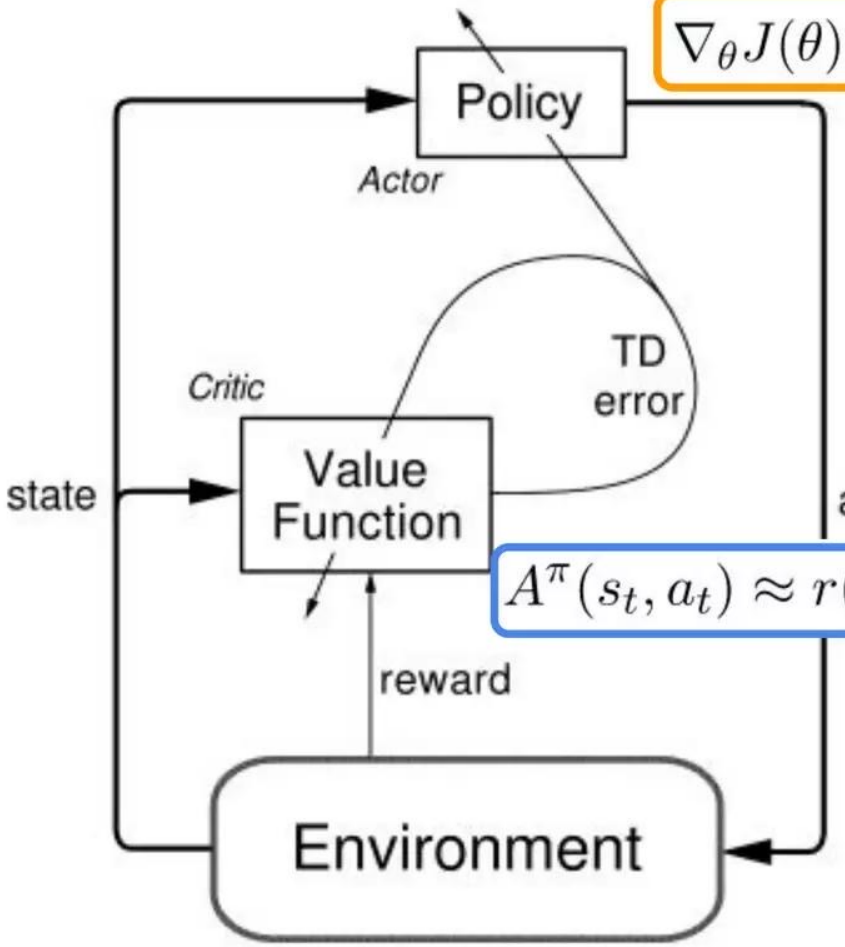
$Q^\pi(s_t,a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'},a_{t'})|s_t,a_t]$: how good the action we take from current state.

$V^\pi(s_t) = E_{a_t\sim\pi_\theta(a_t|s_t)}[Q^\pi(s_t,a_t)]$: The average return when other agent face the same state.

$A^\pi(s_t,a_t) = Q^\pi(s_t,a_t) - V^\pi(s_t)$: We called this **advantage function**, which reflects how good the action we've taken compared to other candidates.

Actor-Critic algorithm

$$\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(a|s) \widehat{A}^\pi(s,a)$$



$$A^\pi(s_t, a_t) \approx r(a_t, s_t) + \gamma \widehat{V}^\pi_\phi(s_{t+1}) - \widehat{V}^\pi_\phi(s_t)$$

Actor-Critic algorithm

Online actor-critic algorithm:

1. Take action, get one-step experience (s, a, s', r)
2. Fit Value function

$$L(\phi) = \frac{1}{2} \sum_i \left\| \widehat{V}_\phi^\pi(s_i) - y_i \right\|^2$$

3. Evaluate advantage function

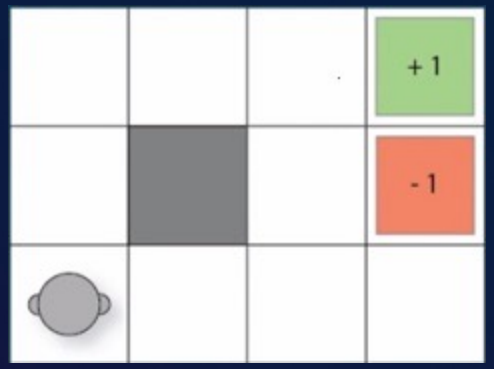
$$A^\pi(s_t, a_t) \approx r(a_t, s_t) + \gamma \widehat{V}_\phi^\pi(s_{t+1}) - \widehat{V}_\phi^\pi(s_t)$$

4. $$\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(a|s) \widehat{A}^\pi(s, a)$$

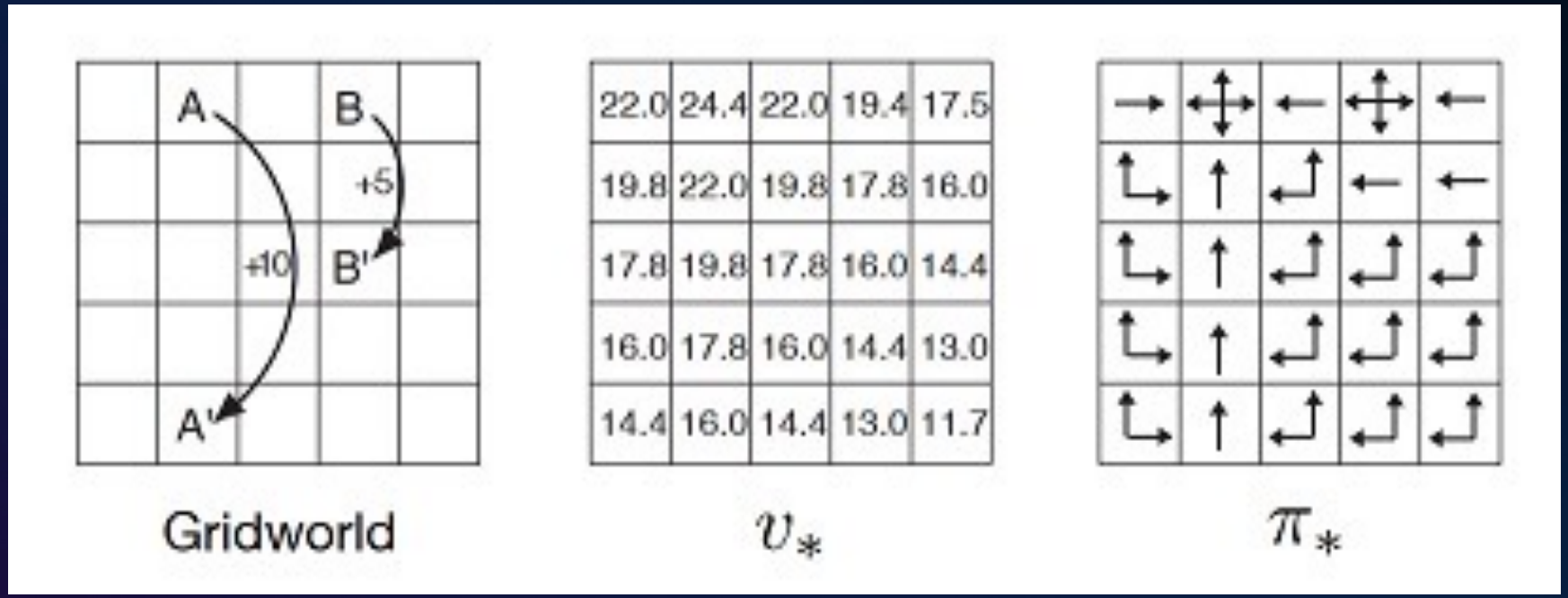
5. $$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Referenced from CS294 in UCB.

Grid World demo - workshop



Grid World is a game for demonstration. 12 positions, 11 states, 4 actions. Our aim is to find optimal policy.



Thank you!