AI in Mathematics
Lecture 11
Reinforcement Learning

Bar-Ilan University
Nebius Academy | Stevens Institute of Technology

June 10, 2025

About This Course

~~1 week: Intro~~

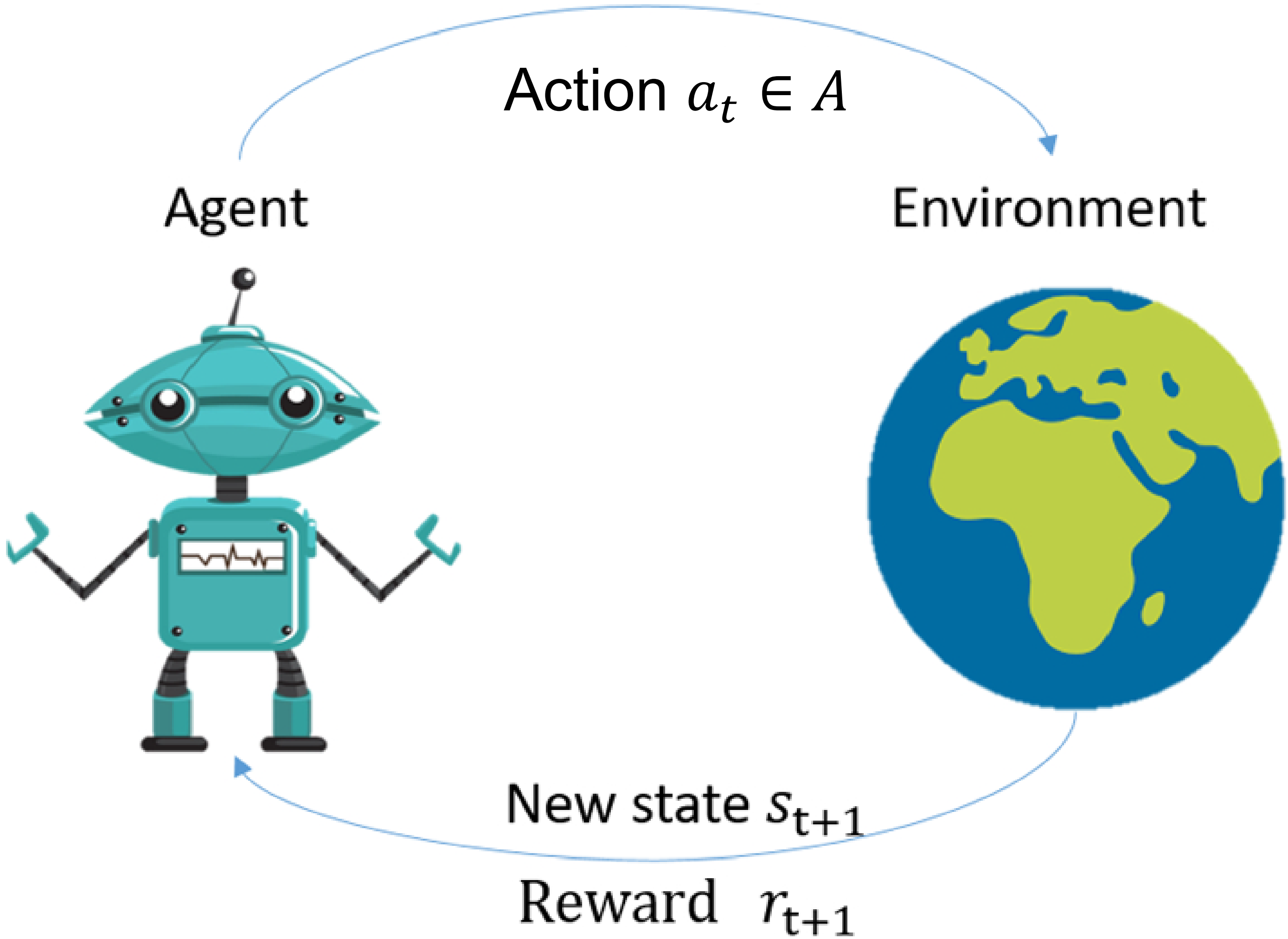
~~2 weeks: Classic ML~~

~~2 weeks: Deep Learning in Mathematics~~

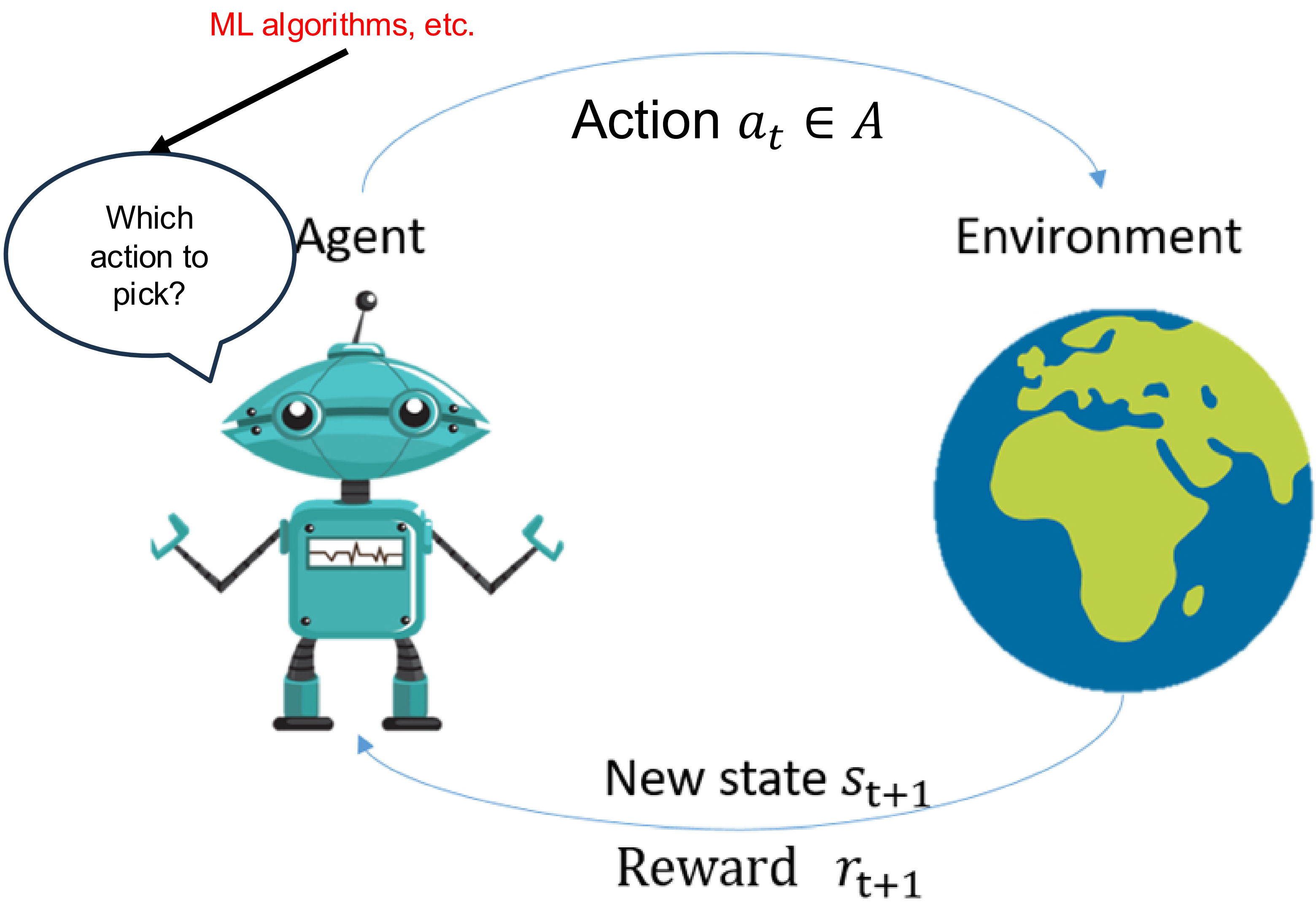
~~4 weeks: Math as an NLP problem (LLMs etc.)~~

4 weeks: Reinforcement Learning (RL) in Math

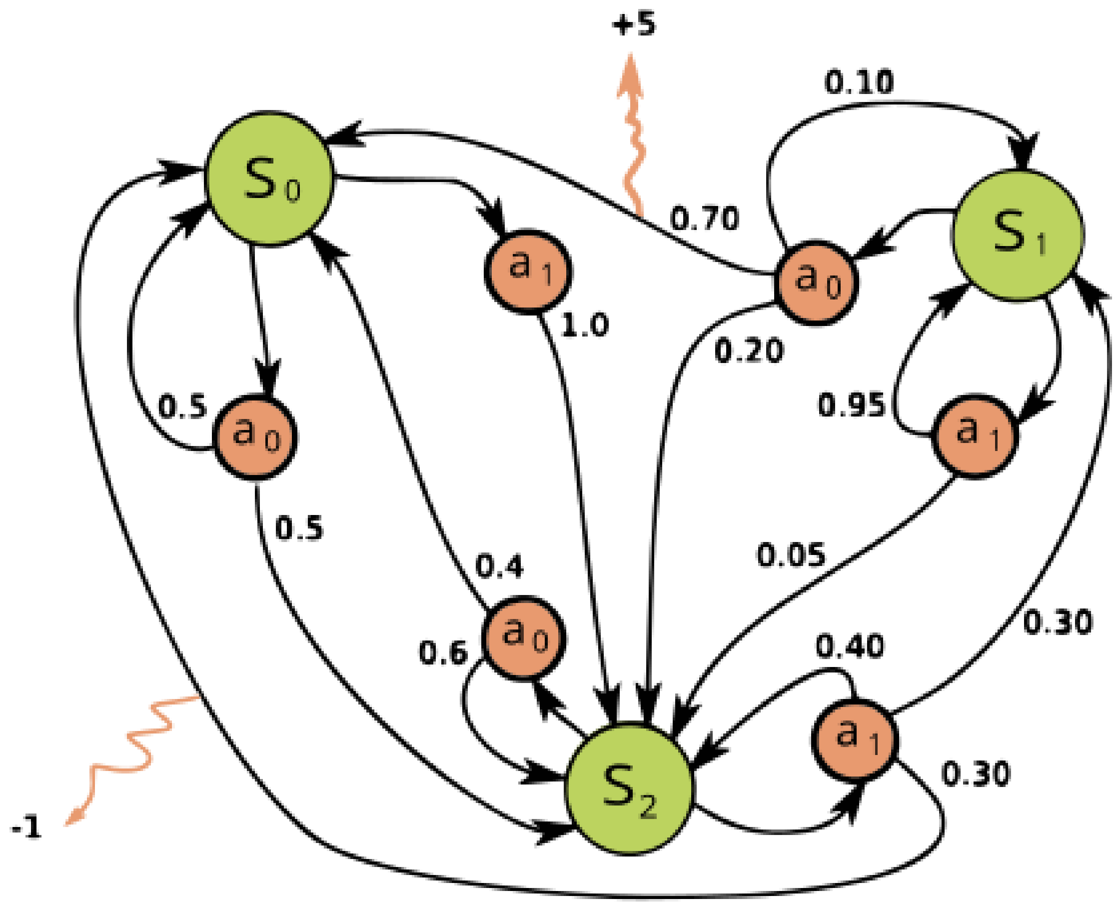
Reinforcement Learning



Reinforcement Learning

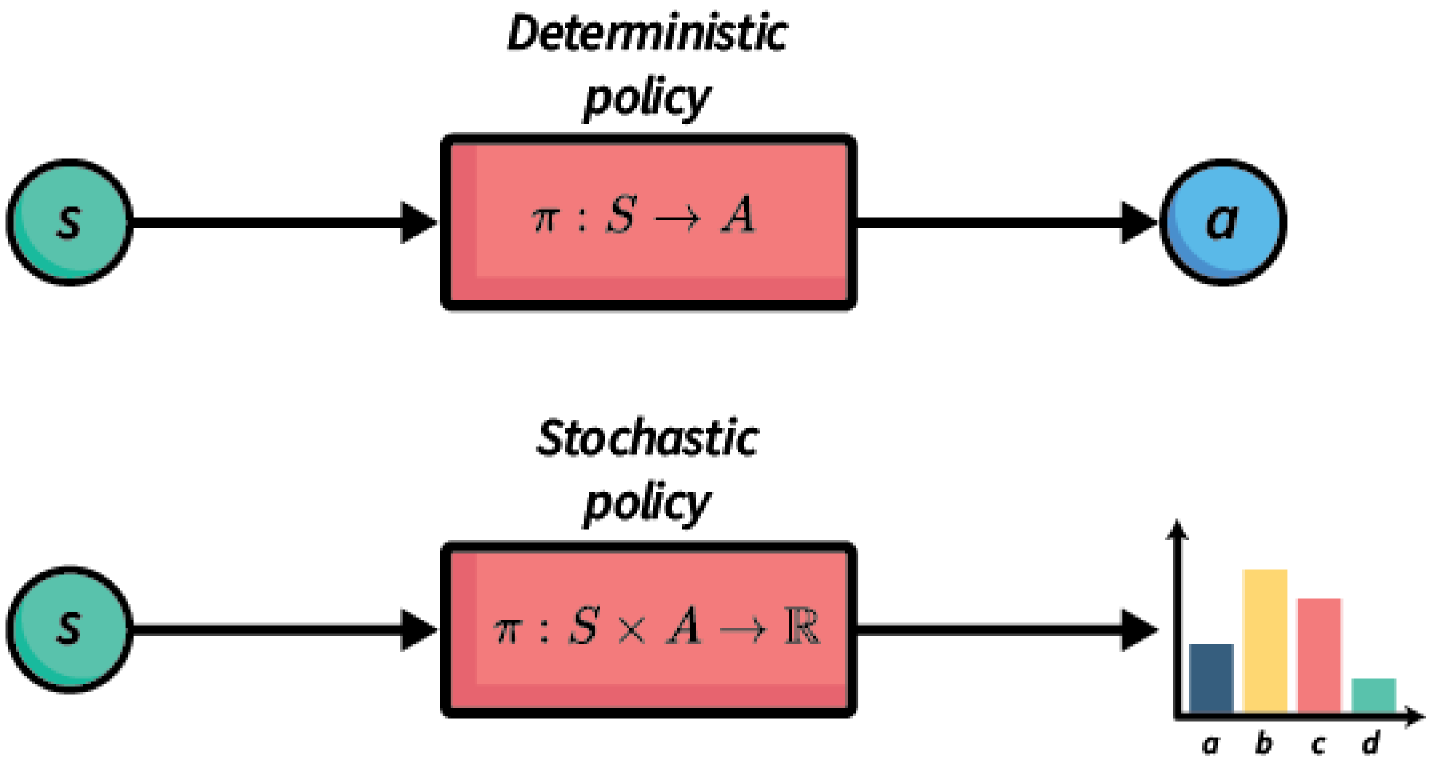


Markov Decision Process



Policy

A **policy** is the agent's strategy for choosing actions.



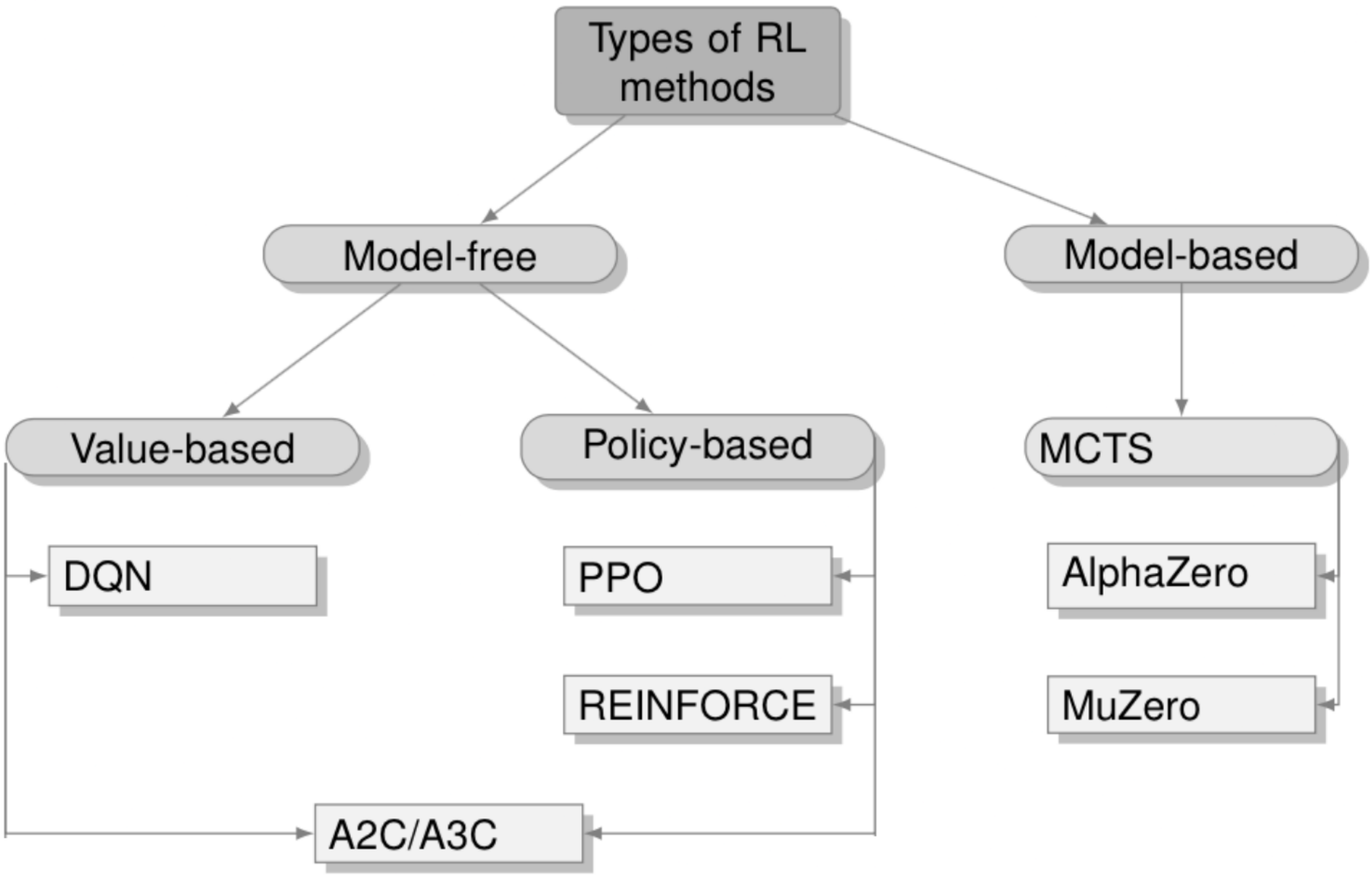
Policy

A **policy** is the agent's strategy for choosing actions.

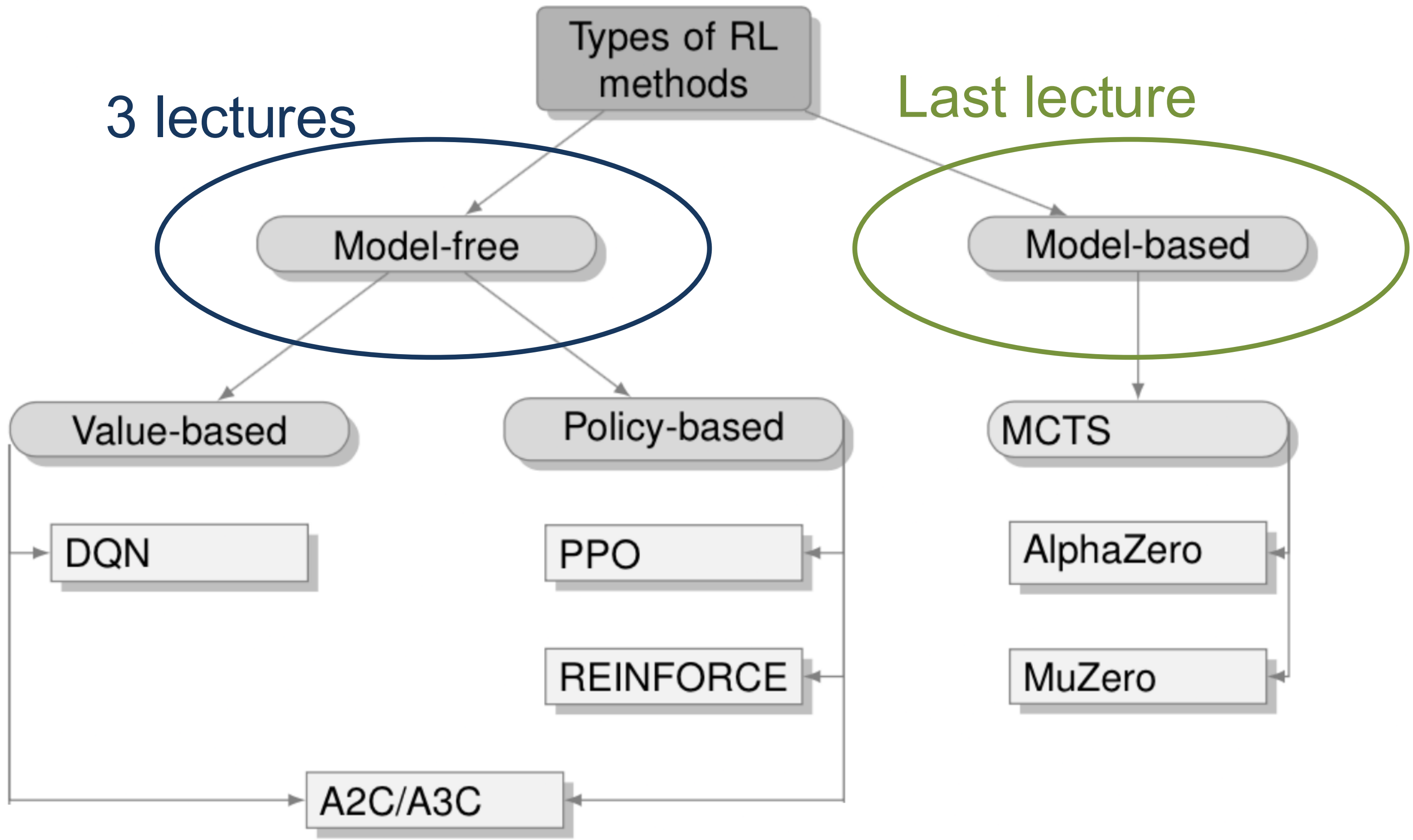
Deterministic policy:
$\pi(s) = \ a \ -$ always choose action a in state s.

Stochastic policy:
$\pi(a \mid s) = P(a_t = a \mid s_t = s) -$ the probability of taking an action a in state s.

RL Algorithms



RL Algorithms



Best Policy

The best policy looks as follows:

$$\pi^\star = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{n} R(s_t, a_t)\right],$$

$$a_t = \pi(s_t)$$

What are the problems with this formula?

Best Policy

$$\pi^\star = \operatorname*{argmax}_{\pi} \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} R(s_t, a_t)\right], a_t = \pi(s_t)$$

$R(s_t, a_t)$ is a notation for $\mathbb{E}_{s_{t+1}}[R(s_t, a_t, s_{t+1})]$

$\mathbb{E}[\sum_{t=0}^{\infty} R(s_t, a_t)]$ can be infinite.

Methods to Penalize Agent

Per-step penalty (negative reward per time step)

Time limit truncation (implicit penalty)

Discount factor trick ($\gamma < 1$).

Best Policy

$$\pi^\star = \operatorname*{argmax}_{\pi} \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right],$$
$$a_t = \pi(s_t)$$

$$\mathcal{J}(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)].$$

So we can apply gradient descent on $\mathcal{J}(\pi_\theta)$ to find $\pi^\star$.

Policy Gradient

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta \mathcal{J}(\pi_{\theta_k})$$

We need to calculate $\nabla_\theta \mathcal{J}(\pi_\theta)$ first.

$$\mathcal{J}(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

$$= \int R(\tau) p_\theta(\tau) d\tau$$

We need to understand how to compute the probability $p_\theta(\tau)$.

Policy Gradient

$$\mathcal{J}(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right] = \int R(\tau) p_\theta(\tau) d\tau$$

$$p_\theta(\tau) = Pr(s_0) \cdot \prod_{t=0}^{\infty} \pi_\theta(a_t | s_t) \cdot Pr(s_{t+1} | s_t, a_t)$$

So we can not simply calculate it or its gradient.
We need to invent some other way.

Policy Gradient

$$\nabla_\theta \mathcal{J}(\pi_\theta) = \int R(\tau) \nabla_\theta p_\theta(\tau) d\tau$$

$$= \int R(\tau) \nabla_\theta p_\theta(\tau) \frac{p_\theta(\tau)}{p_\theta(\tau)} d\tau$$

$$= \int R(\tau) \nabla_\theta \ln p_\theta(\tau)\ p_\theta(\tau) d\tau$$

$$= \mathbb{E}_\tau [R(\tau) \nabla_\theta \ln p_\theta(\tau)]$$

Policy Gradient

Why is it better?

$$\nabla_\theta \ln p_\theta(\tau) =$$

$$= \nabla_\theta \ln \left[p(s_0) \cdot \prod_{t=0}^{\infty} \pi_\theta(a_t|s_t) p(s_{t+1}|\, s_t, a_t) \right]$$

$$= \nabla_\theta \sum_{t=0}^{\infty} \ln \pi_\theta(a_t|s_t)$$

Policy Gradient

$$\nabla_\theta \mathcal{J}(\pi_\theta) = \mathbb{E}_\tau[R(\tau)\nabla_\theta \ln p_\theta(\tau)] =$$

$$= \mathbb{E}_\tau\left[R(\tau)\nabla_\theta \sum_{t=0}^{\infty} \ln \pi_\theta(a_t|s_t)\right]$$

$$= \mathbb{E}_\tau\left[\sum_t \nabla_\theta \ln \pi_\theta(a_t|s_t) \sum_{t' \geq t} \gamma^{t'-t} R(s_{t'}, a_{t'})\right]$$

$$Q(s_t, a_t) = \sum_{t' \geq t} \gamma^{t'-t} R(s_{t'}, a_{t'})$$

Policy Gradient

$$\nabla_\theta \mathcal{J}(\pi_\theta)$$

$$= \mathbb{E}_\tau \left[\sum_t \nabla_\theta \ln \pi_\theta(a_t|s_t)\, Q(s_t, a_t) \right]$$

$$= \mathbb{E}_{(s,\, a) \sim\, d^{\pi_\theta}(s)\, \pi_\theta(s\, |\, a)} [\nabla_\theta \ln \pi_\theta(a|s)\, Q(s, a)]$$

Where the last expectation is over the stationary distribution of statesdefined by our policy.

Policy Gradient

$$\nabla_\theta \mathcal{J}(\pi_\theta) = \mathbb{E}_{(s,a)\sim\, d^{\pi_\theta}(s)\,\pi_\theta(s\,|\,a)}[\nabla_\theta \ln \pi_\theta(a|s)\; Q(s,a)]$$

This is the policy gradient theorem.

It expresses the gradient of expected return with respect to the policy parameters as an expectation over state-action pairs.

REINFORCE

$$\nabla_\theta \mathcal{J}(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_{\theta_k} \ln \pi_{\theta_k}(a|s)\, Q(s,a)\right]$$

In practice, REINFORCE algorithm approximates this gradient by trajectories sampled from the policy.

We get $Q(s,a)$ from a trajectory we have seen.

REINFORCE

$$\nabla_\theta \mathcal{J}(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_{\theta_k} \ln \pi_{\theta_k}(a|s)\, Q(s,a)\right]$$

We need full trajectories to be able to compute $Q(s,a)$.

This method suffers from high variance, as the gradient estimate $\nabla_\theta \mathcal{J}(\pi_\theta)$ may fluctuate substantially across different episodes.

Variance problem

$$\nabla_\theta \mathcal{J}(\pi_\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \ln \pi_\theta(a|s)\,(Q(s,a) - b(s))]$$

$b(s)$ is some function of the state that allows us to reduce variance.

One good solution is to use $b(s) = V^{\pi_\theta}(s) = \mathbb{E}_a Q(s,a)$. It is exactly what A2C will do.

So $V^{\pi_\theta}(s)$ represents how good or promising this state is on average, under the future path defined by π.

A2C

$$\nabla_\theta \mathcal{J}(\pi_\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \ln \pi_\theta(a|s)\, A(s,a)]$$

where the *advantage* function is defied as

$A(s,a) = \ Q(s,a) - V^{\pi_\theta}(s), \ V^{\pi_\theta}(s) = \mathbb{E}_a Q(s,a).$

To approximate A, we will use

$$A_t(s,a) = \ r_t \ + \gamma\, V(s_{t+1}) - \ V(s_t)$$

This is a **one-step temporal difference (TD) estimate** of the advantage — efficient and requires minimal storage.

1. The **value function** V(s) is learned using a neural network.
2. The **advantage estimate** A_t uses only the **immediate reward** and **next state value** — no need to store full trajectories.

A2C

Collect Trajectories

1. Interact with the environment using the current policy π_θ.
2. Collect (s_t, a_t, r_t, s_{t+1}) for multiple time steps (usually in parallel environments).

Use a neural network (critic) to predict $V_\phi(s) \approx V^\pi(s)$.

Use 1-step TD estimate:

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

Update Policy (Actor):

1. $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot A_t$

Update Critic using loss function:

$$L_{critic} = (r_t + \gamma V(s_{t+1}) - V(s_t))\text{^}2$$

A2C (Detailed)

Collect Trajectories
(Shared Step)

- Interact with the environment using the current policy π_θ.
- Collect transitions (s_t, a_t, r_t, s_{t+1}) over multiple steps (e.g., from parallel environments)
- Use a neural network (the critic) to predict $V_\phi(s) \approx V^\pi(s)$.
- Compute 1-step TD Advantage estimate:

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

What Actually Happens:

We **collect a batch** of transitions (s_t, a_t, r_t, s_{t+1}) from multiple environments or steps.

Then compute:

- **Batch of value predictions** $V(s_t), V(s_{t+1})$
- **Batch of advantages** $A_t = r_t + \gamma V(s_{t+1}) - V(s_t)$

These batches are then used to perform **gradient updates**

A2C (Detailed)

Actor Update

Policy Improvement
The actor updates the policy parameters θ to maximize expected return.

Gradient ascent step:
$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \ln \pi_\theta(\, a_t \mid \, s_t\,) \cdot \, A_t$$

Critic Update

The critic learns to estimate the **state value** $V^\pi(s) \approx \, V_\phi(s)$

Loss function:
$$L_{critic} = \Big(\, r_t \, + \gamma V \phi(s_{t+1}) - \, V_\phi(s_t) \Big)^2$$
Gradient descent step:
$$\phi \leftarrow \phi \, - \eta \cdot \nabla_\phi \, L_{critic}$$

In both cases we can use stochastic gradient descent to update parameters.