Reinforcement learning

Weizhe Yuan

11/20/2023

Outline

- **Part 1**: Basic Concepts
- **Part 2**: Value-based Methods
  - Q-Learning
  - Deep Q-Learning
- **Part 3**: Policy-based Methods
  - Policy Gradient Methods
  - Actor Critic Methods
  - Proximal Policy Optimization

Part 1:
Basic Concepts

NYU

What is Reinforcement Learning

- The idea behind **Reinforcement Learning (RL)** is that **an agent** (an AI) will learn from the environment by **interacting with it** (through trial and error) and **receiving rewards** (negative or positive) as feedback for performing actions.

What is Reinforcement Learning

- Learning from interactions with the environment comes from our natural experiences.

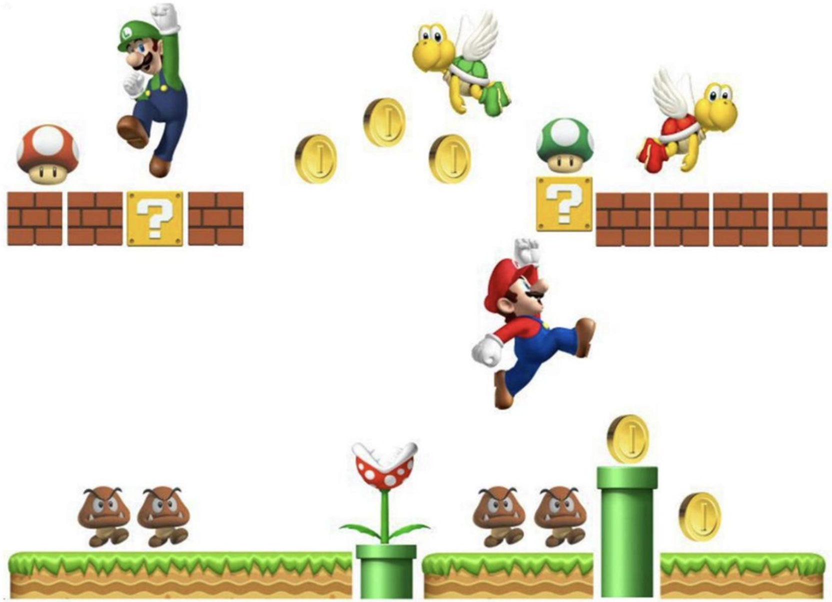
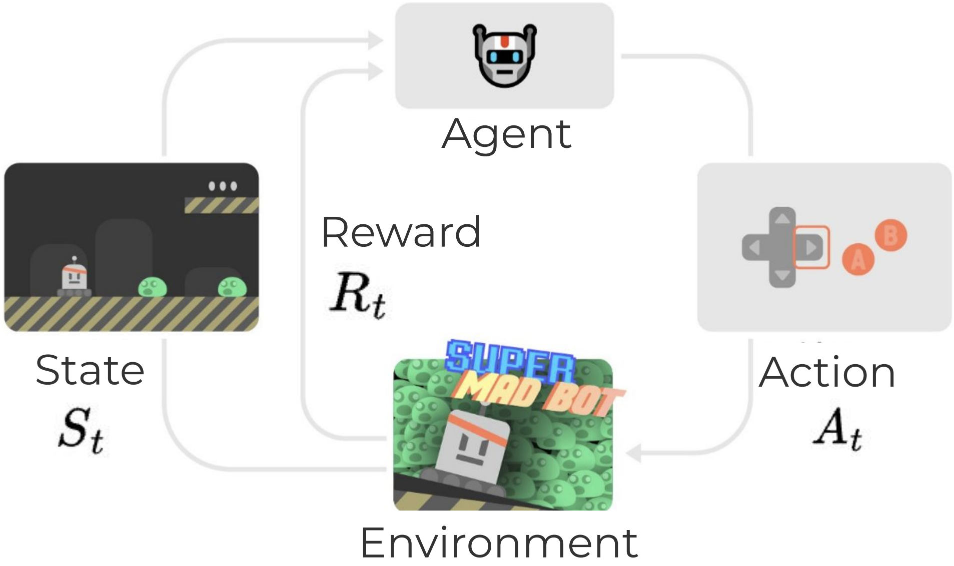


Image from:

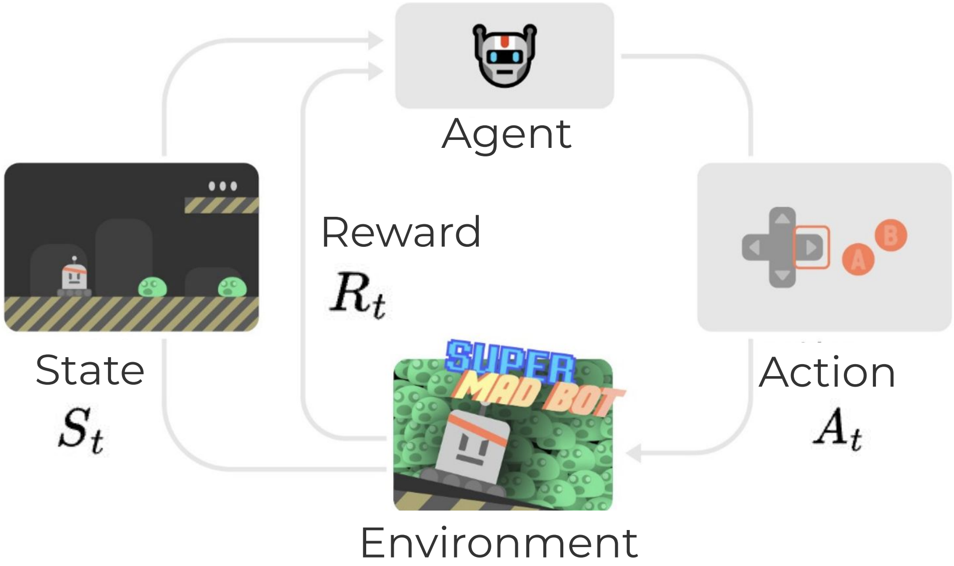
The RL Process (Loop)



- **Agent**: The decision-maker in a system
- **Environment**: The setting or context where the agent operates.
- **State**: The current situation or condition of the environment.
- **Action**: The choices or moves an agent can make in response to a state.
- **Reward**: Feedback from the environment indicating the success of an action in achieving a goal.

Image from: https://huggingface.co/learn/deep-rl-course

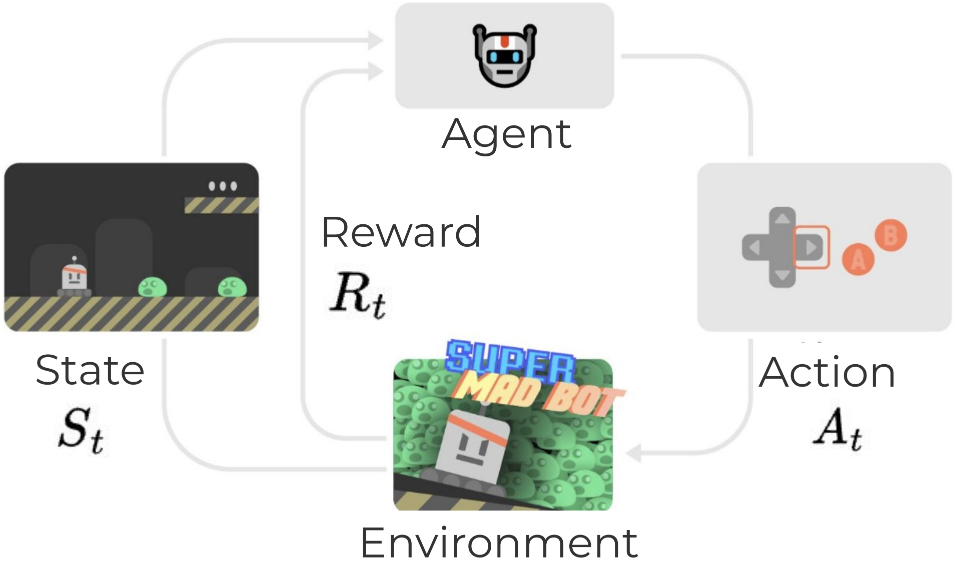
The RL Process (Loop)



A Walkthrough

Image from: https://huggingface.co/learn/deep-rl-course

The RL Process (Loop)

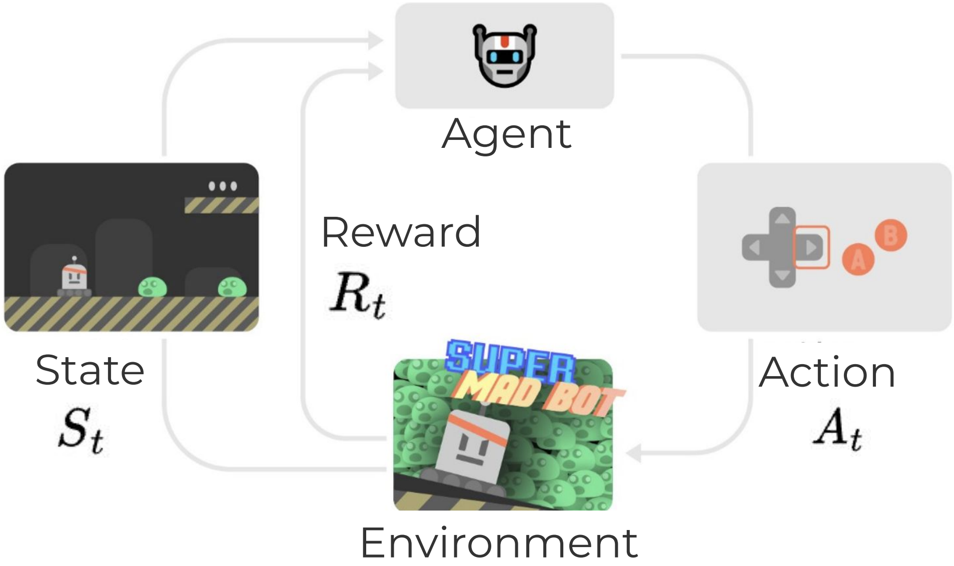


- Our Agent receives state S_0 from the Environment
  - we receive the first frame of our game (Environment).
- Based on that state S_0 the Agent takes action A_0
  - our Agent will move to the right.
- The environment goes to a new state S_1
  - new frame
- The environment gives some reward R_1 to the Agent
  - we're not dead (Reward +1)

NYU

Image from: https://huggingface.co/learn/deep-rl-course

The RL Process (Loop)



- Outputs
  - A sequence of state, action, reward and next state.

$$\underbrace{\boxed{S_0}}_{\text{State}}, \underbrace{\boxed{A_0}}_{\text{Action}}, \underbrace{\boxed{R_1}}_{\text{Reward}}, \underbrace{\boxed{S_1}}_{\text{Next State}}$$

NYU

Image from: https://huggingface.co/learn/deep-rl-course

Goal of the Agent

- The agent's goal is to maximize its cumulative reward, **called the expected return**.

Rewards and the discounting

- The **cumulative reward** at each time step after t can be written as:

$$R(\boxed{\tau}) = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + \dots$$

Return: cumulative reward

Trajectory (read Tau)
Sequence of states and actions

- However, in reality, the rewards that **come sooner** are more likely to happen since they are more predictable than the long-term future reward.

NYU

Rewards and the discounting

- Our **discounted expected cumulative reward** is:

$$R(\boxed{\tau}) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots$$

Return: cumulative reward

Gamma: discount rate

Between 0 and 1

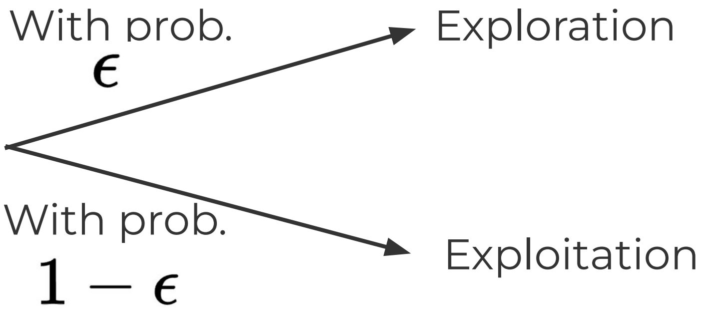
Trajectory (read Tau)
Sequence of states and actions

Type of tasks

- We can have two types of tasks: **episodic** and **continuing**.
- **Episodic task**
  - In this case, we have a starting point and an ending point (a terminal state). This creates an episode: a list of States, Actions, Rewards, and new States.
- **Continuing tasks**
  - These are tasks that continue forever (no terminal state). In this case, the agent must learn how to choose the best actions and simultaneously interact with the environment.

The Exploration/Exploitation trade-off

- **Exploration** is exploring the environment by trying random actions in order to find more information about the environment.
  - E.g., Go to a new restaurant
- **Exploitation** is exploiting known information to maximize the reward.
  - E.g., Pick a known good restaurant
- **epsilon-greedy strategy**



NYU

Two main approaches for solving RL problems

- The **Policy π**: the agent's brain
  - The Policy π is the brain of our Agent, it's the function that tells us what action to take given the state we are in. So it defines the agent's behavior at a given time.
- **Our goal is to find the optimal policy π***
- **Two ways**
  - **Directly (Policy-based methods)**: teaching the agent to learn which action to take
  - **Indirectly (Value-based methods)**: teach the agent to learn which state is more valuable

Can also combine them!

NYU

Part 2:
Value-based Methods

Value-based Methods

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.

The link between value and policy

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

State Value Function

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function**

$$\underbrace{V_\pi(s)}_{\text{Value of state s}} = \underbrace{E_{\boxed{\pi}}}_{\text{Expected return}}[G_t | \underbrace{S_t = s}_{\text{If the agent starts at state s}}]$$

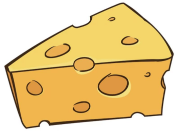
Use the policy to choose its actions for all time steps

NYU

State Value Function

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function (an example)**

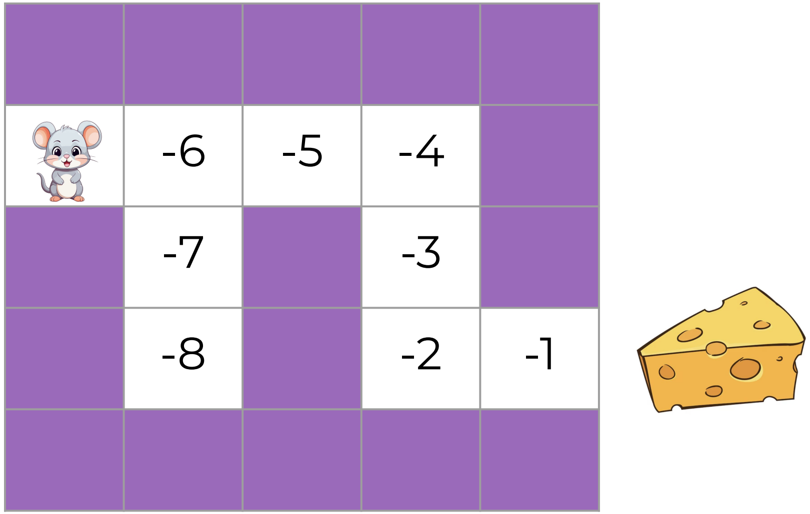
-7	-6	-5	-4	
	-7		-3	
	-8		-2	-1



NYU

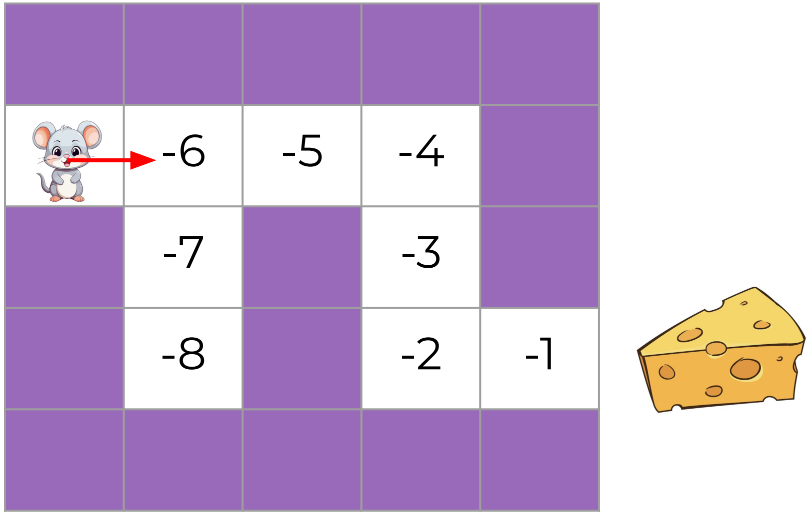
State Value Function

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function (an example)**



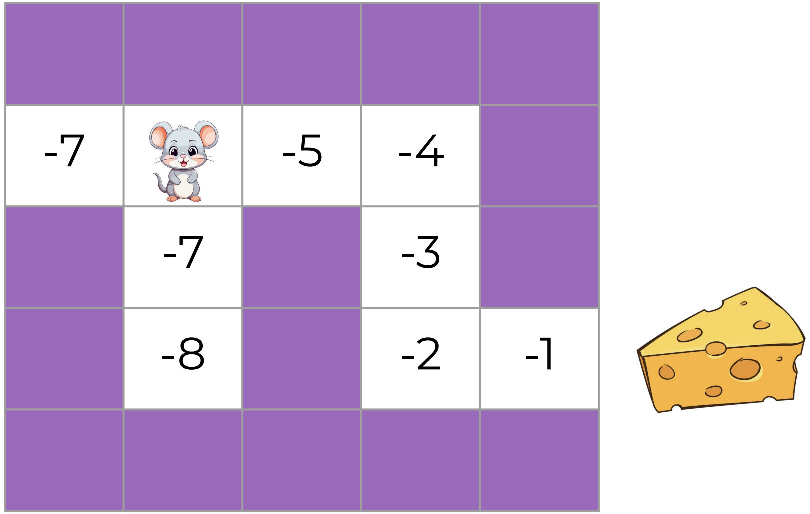
State Value Function

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function (an example)**



State Value Function

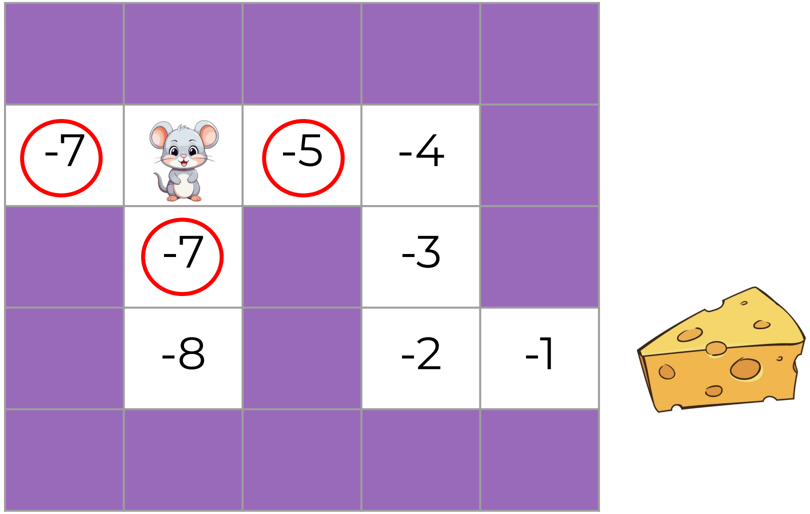
- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function (an example)**



NYU

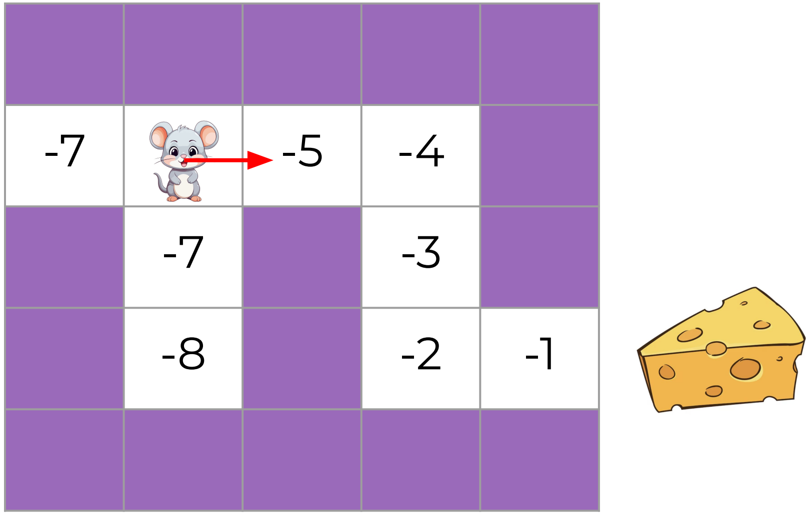
State Value Function

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function (an example)**



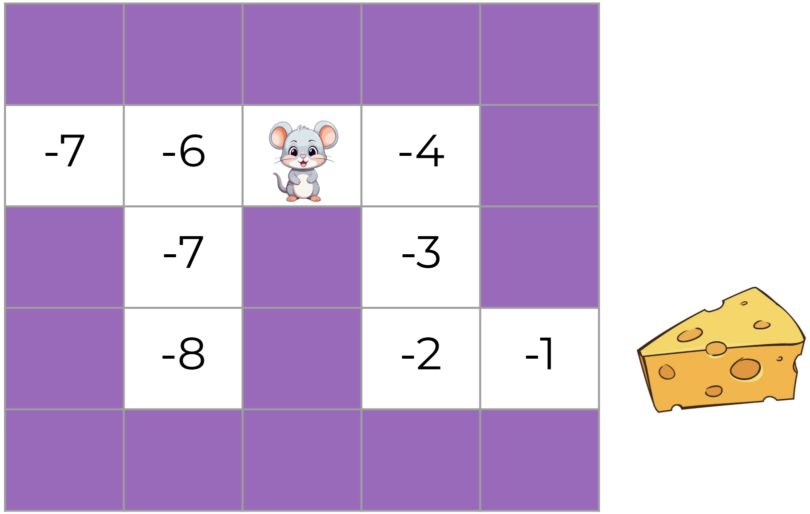
State Value Function

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function (an example)**



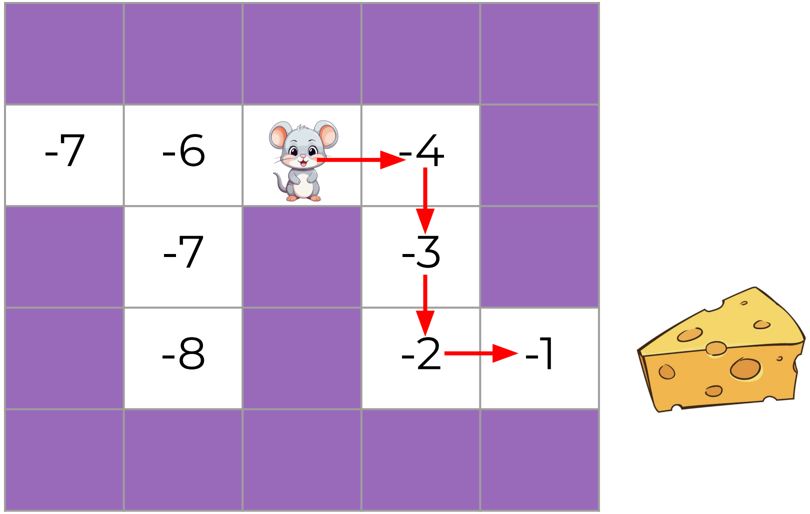
State Value Function

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function (an example)**



State Value Function

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function (an example)**



NYU

Action Value Methods

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function**
- **Action Value Function**

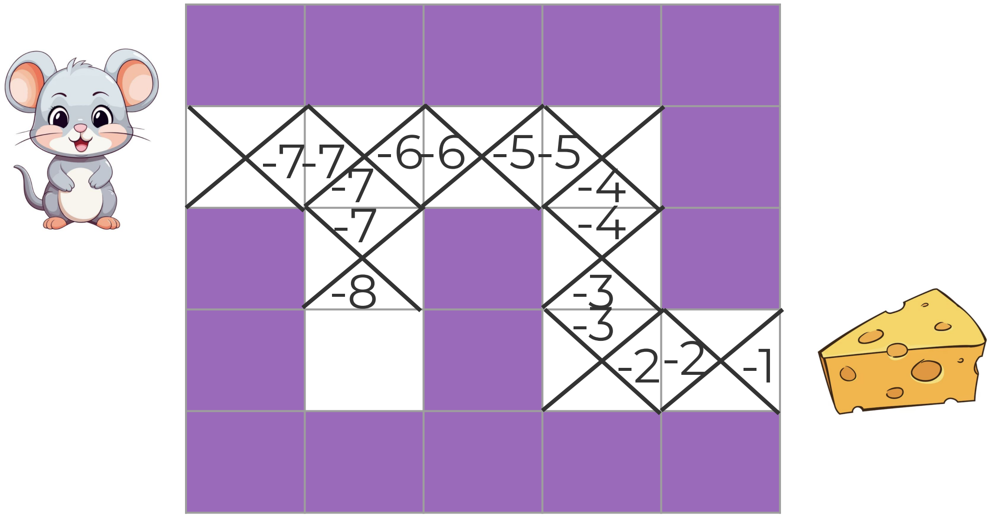
$$\underbrace{Q_\pi(s, a)}_{\text{Value of state action pair (s, a)}} = \underbrace{E_{\boxed{\pi}}}_{\text{Expected return}}[G_t | \underbrace{S_t = s}_{\text{If the agent starts at state s}}, \underbrace{A_t = a}_{\text{and chooses action a}}]$$

Use the policy to choose its actions for all time steps

NYU

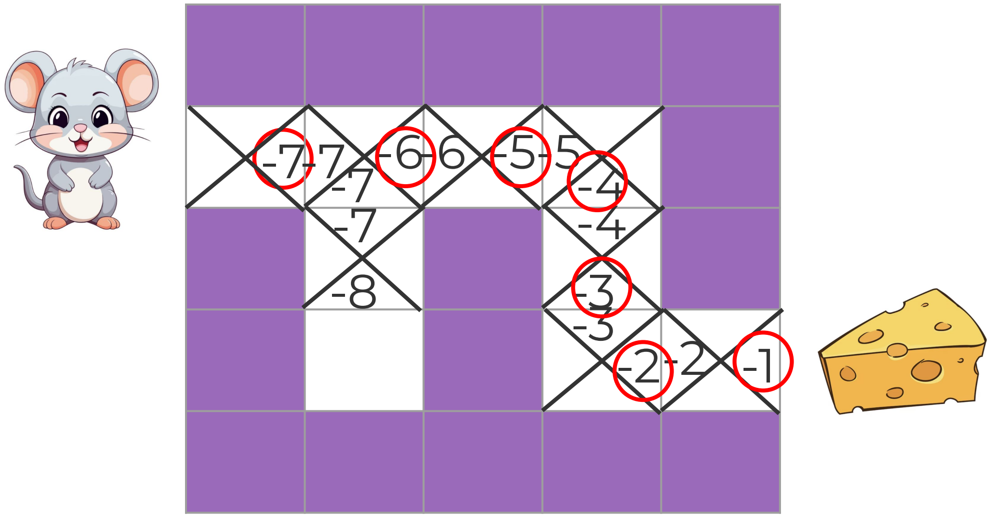
Action Value Methods

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function**
- **Action Value Function (an example)**



Action Value Methods

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function**
- **Action Value Function (an example)**



Value-based Methods

- Train a **value function** that outputs the value of a **state or a state-action pair**. Given this value function, our policy will take an action.
- **State Value Function**
- **Action Value Function**

How to calculate the value function? 🤔

Value Calculation

- If we calculate $V(S_t)$ (the value of a state), we need to calculate the return starting at that state and then follow the policy forever after.

$$R(\boxed{\tau}) = r_{t+1} + \underline{\gamma} r_{t+2} + \underline{\gamma^2} r_{t+3} + \underline{\gamma^3} r_{t+4} + \ldots$$

Return: cumulative reward

Gamma: discount rate

Between 0 and 1

Trajectory (read Tau)
Sequence of states and actions

The Bellman Equation

- The Bellman equation simplifies our state value or state-action value calculation

Use the policy to choose its actions for all time steps

$$\underbrace{V_\pi(s)}_{\text{Value of state s}} = \underbrace{E_{\boxed{\pi}}[R_{t+1}}_{\text{Expected value of immediate reward}} + \underbrace{\gamma * V_\pi(S_{t+1})}_{\text{discounted value of next\_state}} \underbrace{|S_t = s]}_{\text{If the agent starts at state s}}$$

Two Learning Strategies

- Monte Carlo
- Temporal Difference Learning

Monte Carlo

- Idea: **learning at the end of the episode**
- Wait until the end of the episode, calculate G_t (return) and uses it as a target for updating $V(S_t)$

$$\underbrace{V(S_t)}_{\text{New (estimated) value of state t}} \leftarrow \underbrace{V(S_t)}_{\text{Former (estimated) value of state t}} + \underbrace{\alpha}_{\text{Learning rate}}[\underbrace{G_t}_{\text{Return}} - \underbrace{V(S_t)}_{\text{Former (estimated) value of state t}}]$$

Temporal Difference Learning

- Idea: **learning at each step**
- Wait for only one interaction (one step) S_{t+1} to form a TD target and update $V(S_t)$ using R_{t+1} and $\gamma * V(S_{t+1})$

$$\underbrace{V(S_t)}_{\text{New (estimated) value of state t}} \leftarrow \underbrace{V(S_t)}_{\text{Former (estimated) value of state t}} + \underbrace{\alpha}_{\text{Learning rate}}[\underbrace{R_{t+1}}_{\text{Reward}} + \underbrace{\gamma V(S_{t+1})}_{\text{Discounted value of next state}} - \underbrace{V(S_t)}_{\text{Former (estimated) value of state t}}]$$

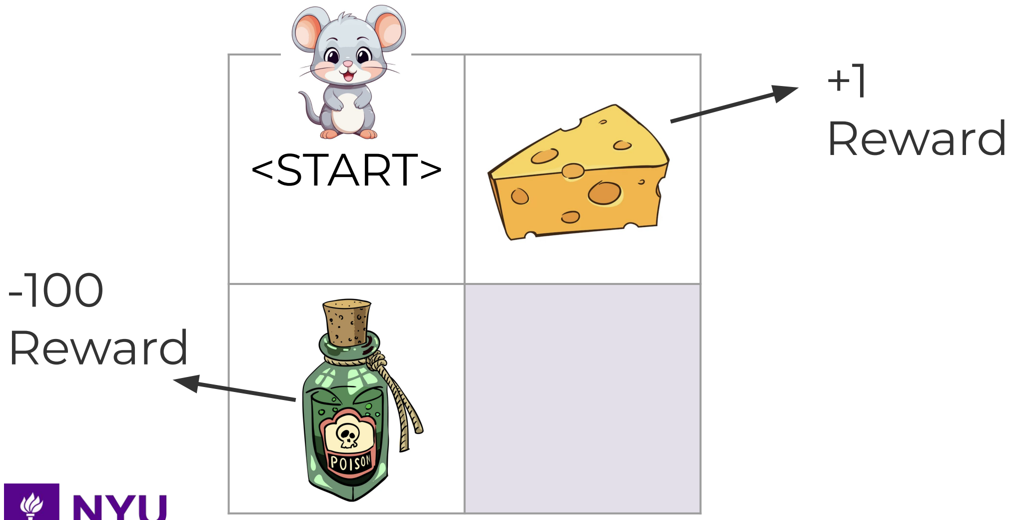
NYU

Introduction to Q-Learning

- Q-Learning is an **off-policy** value-based method that uses a **TD approach** to train its **action-value function**
  - The **Q** comes from “the Quality” (the value) of that action at that state.
- **Off-policy & On-policy**
  - **Off-policy**: Using a different policy for acting (inference) and updating (training)
  - **On-policy**: Using the same policy for acting and updating

Introduction to Q-Learning

- The Q-Learning algorithms

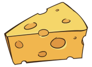


Introduction to Q-Learning

- The Q-Learning algorithms
  - **Step 1: Initialize the Q table**

Actions

States

	←	→	↑	↓
<START>	0	0	0	0
	0	0	0	0
	0	0	0	0

NYU

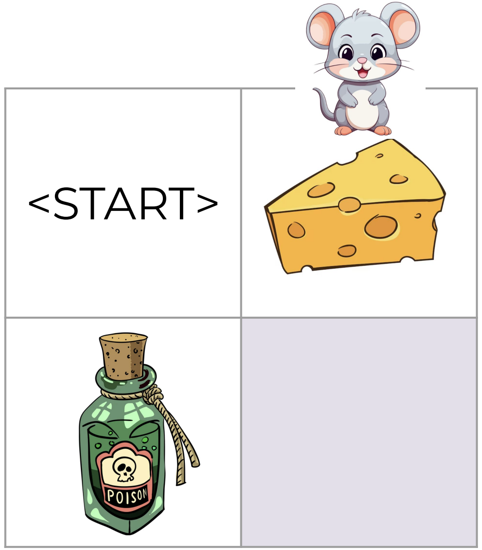
Introduction to Q-Learning

- The Q-Learning algorithms
  - **Step 2: Choose an action using the epsilon-greedy strategy**

NYU

Introduction to Q-Learning

- The Q-Learning algorithms
  - **Step 3: Perform action At, get reward Rt+1 and next state St+1**



Get +1 Reward

Introduction to Q-Learning

- The Q-Learning algorithms
  - **Step 4: Update Q(St, At)**

Temporal Difference Update

$$\boxed{V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]}$$

$$\downarrow$$

$$\boxed{Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]}$$

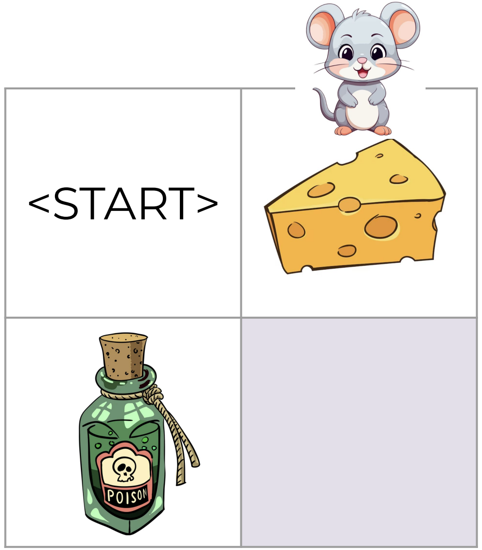
Q-Learning update formula

Introduction to Q-Learning

- The Q-Learning algorithms
  - **Step 4: Update Q(St, At)**

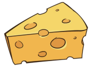
Learning rate   discount factor

New value: $0 + \boxed{0.1}\,(+1 + \boxed{0.99} * 0 - 0) = \mathbf{0.1}$



Actions

States

	←	→	↑	↓
<START>	0	0	0	0
	0	0	0	0
	0	0	0	0

Introduction to Q-Learning

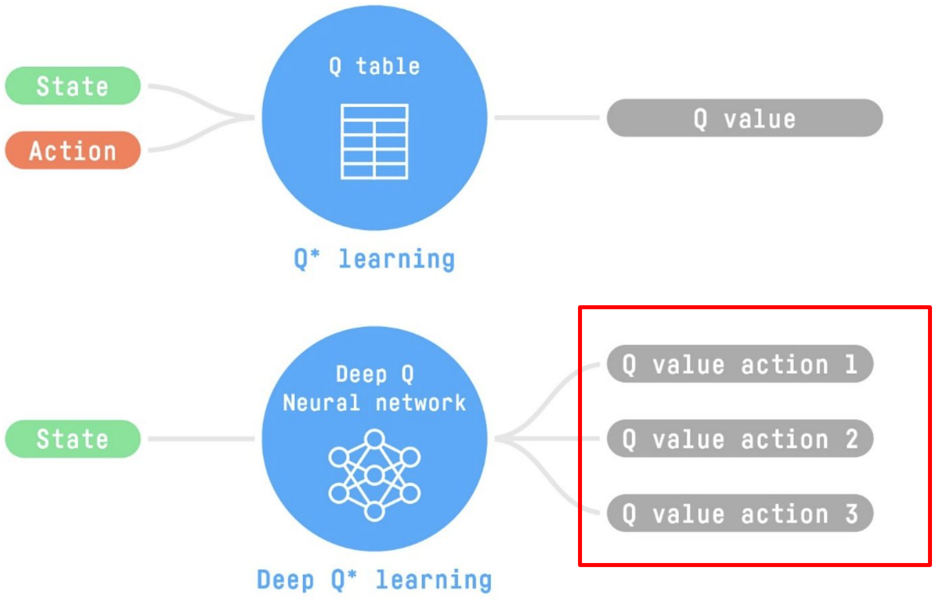
- The Q-Learning algorithms
  - **Go back to Step 2 (choose an action) and repeat**

From Q-Learning to Deep Q-Learning

- **However**, producing and updating a Q-table can become ineffective in large state space environments

From Q-Learning to Deep Q-Learning

- **Idea**: Use a Deep Neural Network to represent the Q function



A linear layer

Image from: https://huggingface.co/learn/deep-rl-course

Part 3:
Policy-based Methods

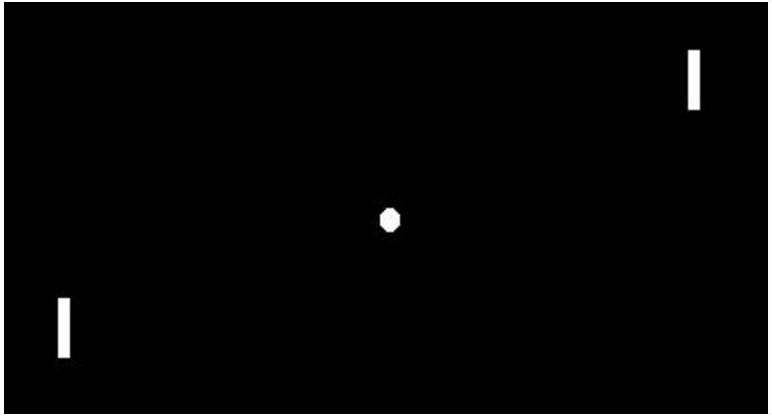
Policy-based Methods

- **Directly** learn to approximate the gold **policy** (typically a NN) without having to learn a value function.
- Compare to value-based methods
  - Policy-gradient methods can learn a stochastic policy

Introduction to Policy Gradient Method

Introduction to Policy Gradient Method

- **The game of Pong**
  - Either a +1 reward if the ball went past the opponent, a -1 reward if we missed the ball, or 0 otherwise.



Goal: earn more rewards

Gif from: https://karpathy.github.io/2016/05/31/rl/

Introduction to Policy Gradient Method

- **Policy gradient:** Run a policy for a while. See what actions led to high rewards. Increase their probability.

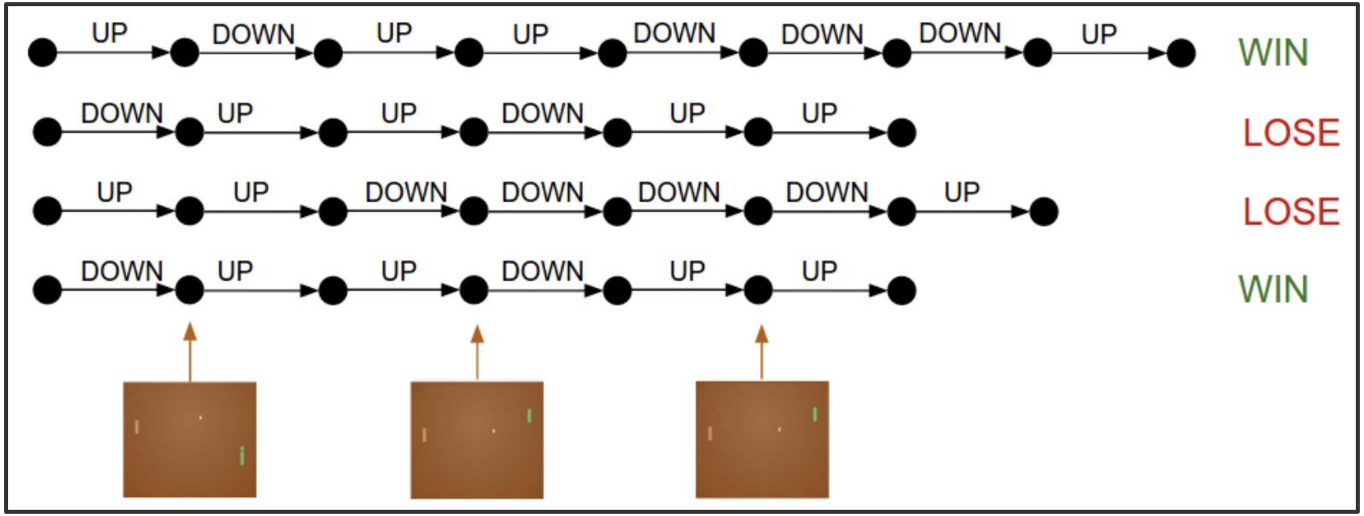


Image from: https://karpathy.github.io/2016/05/31/rl/

NYU

Introduction to Policy Gradient Method

- **Policy gradient:** Run a policy for a while. See what actions led to high rewards. Increase their probability.

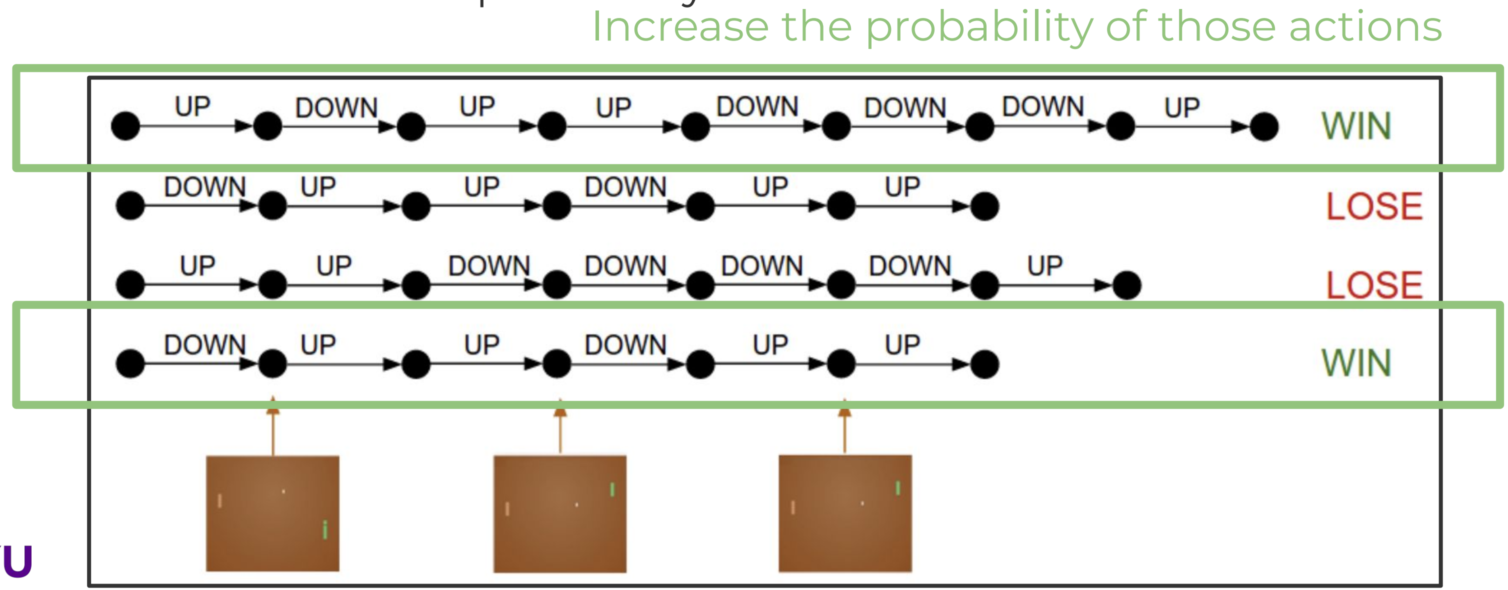


Image from: https://karpathy.github.io/2016/05/31/rl/

Introduction to Policy Gradient Method

- **Policy gradient:** Run a policy for a while. See what actions led to high rewards. Increase their probability.

Decrease the probability of those actions

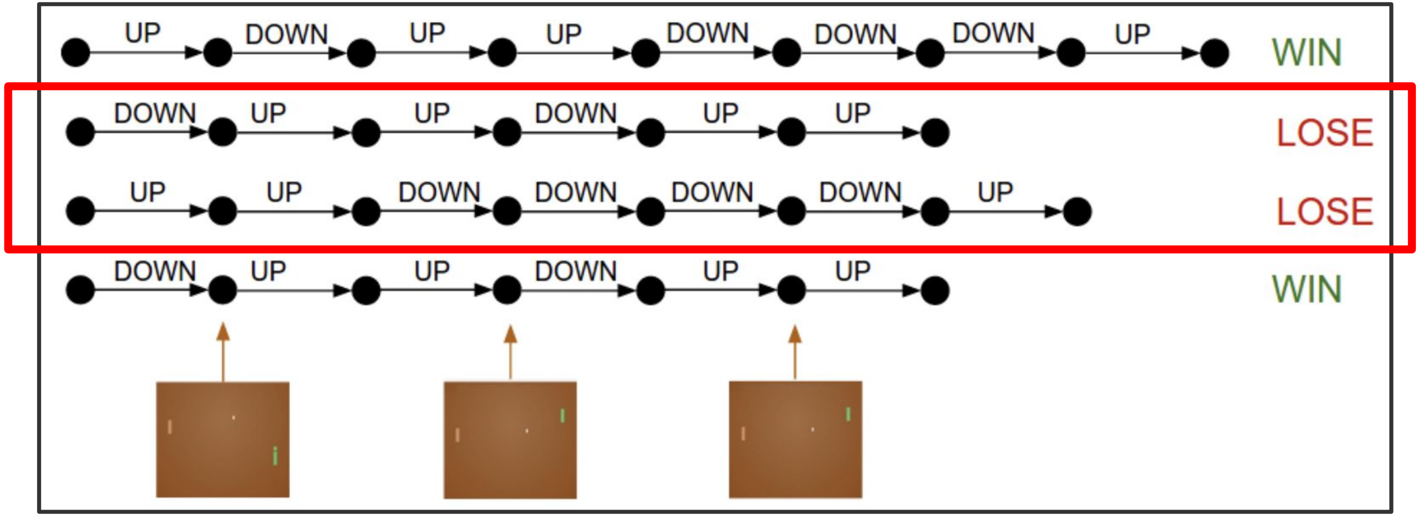


Image from: https://karpathy.github.io/2016/05/31/rl/

Introduction to Policy Gradient Method

- **Policy Gradient Training loop**
  - Collect an episode with the π (policy)
  - Calculate the return (sum of rewards)
  - Update the weights of the π
    - If **positive return** → increase the probability of each (state, action) pairs taken during the episode
    - If **negative return** → decrease the probability of each (state, action) taken during the episode

Introduction to Policy Gradient Method

- We have a policy π parameterized by θ

$$\pi_\theta(s) = \mathbb{P}[A|s;\theta]$$

The policy given a state outputs **a distribution over actions** at that state

Introduction to Policy Gradient Method

- The objective function: expected cumulative reward

$$J(\theta) = E_{\tau \sim \pi}[R(\tau)]$$

$$R(\boxed{\tau}) = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + \dots$$

Return: cumulative reward

Trajectory (read Tau)
Sequence of states and actions

Introduction to Policy Gradient Method

- The objective function: expected cumulative reward

$$J(\theta) = E_{\tau \sim \pi}[R(\tau)]$$

$$= \sum_{\tau} \boxed{P(\tau;\theta)} R(\tau)$$

$$P(\tau;\theta) = [\prod_{t=0} \underbrace{P(s_{t+1}|s_t, a_t)}_{\text{Environment Dynamics}} \underbrace{\pi_\theta(a_t|s_t)}_{\text{Action Probability}}]$$

NYU

Introduction to Policy Gradient Method

- **The Policy Gradient Theorem**
  - Derivation: see here

For any differentiable policy and for any policy objective function, the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)]$$

Introduction to Policy Gradient Method

- **The Reinforce algorithm (Monte Carlo Reinforce)**
  - **In a loop:**
    - Use the policy π_θ to collect an episode τ
    - Use the episode to estimate the gradient

$$\nabla_\theta J(\theta) \boxed{\approx} \tilde{g} = \sum_{t=0} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)$$

Estimation of the gradient, given we are only using one trajectory

NYU

Introduction to Policy Gradient Method

- **The Reinforce algorithm (Monte Carlo Reinforce)**
  - **In a loop:**
    - Use the policy π_θ to collect an episode τ
    - Use the episode to estimate the gradient
    - Update the weights of the policy using **gradient ascent** (since we are maximizing the objective)

Introduction to Policy Gradient Method

- **The Reinforce algorithm (Monte Carlo Reinforce)**
  - **In a loop:**
    - Use the policy π_θ to collect an episode τ
    - Use the episode to estimate the gradient
    - Update the weights of the policy using **gradient ascent** (since we are maximizing the objective)

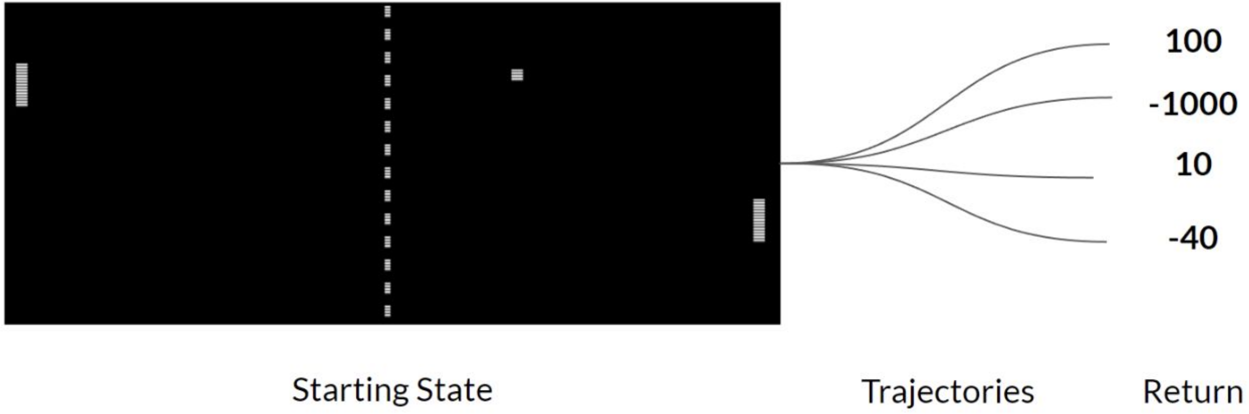
Can also collect multiple trajectories to estimate the gradient

$$\nabla_\theta J(\theta) \approx \tilde{g} = \underbrace{\frac{1}{m} \sum_{i=1}^{m}}_{\text{Multiple trajectories}} \sum_{t=0} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) R(\tau^{(i)})$$

NYU

Introduction to Policy Gradient Method

- The Reinforce algorithm (Monte Carlo Reinforce)
  - However....



Introduction to Policy Gradient Method

- **The Reinforce algorithm (Monte Carlo Reinforce)**
  - **However….**
  - **One solution**: using a large number of trajectories to provide a good estimation of the return
  - Need other ways…

Too computationally expensive!

Introduction to Actor Critic Methods

- Recap: **The policy gradient theorem**

For any differentiable policy and for any policy objective function, the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)]$$
$$= \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T-1} \boxed{G_t} \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)]$$

Derivation see here

Cumulative future rewards starting from t

Can use a Q value function to estimate this !

NYU

Introduction to Actor Critic Methods

- Idea: **Combine** value-based method and policy-based method
  - We learn two function approximations
    - A **policy** π_θ that controls how our agent acts
    - A **value function** $Q_w(s, a)$ to assist the policy update by measuring how good the action taken is

Introduction to Actor Critic Methods

- Learning Process
  - At each timestep t, we get the current state S_t
  - We pass S_t to our policy network (actor) π_θ and get an action A_t
  - We pass S_t, A_t to our value network (critic) $Q_w(s, a)$ and get the value of taking that action at that state $Q_w(S_t, A_t)$
  - We enter a new state S_{t+1} and receive reward R_{t+1}
  - Then actor updates its policy parameters using the Q value

$$\Delta\theta = \alpha \nabla_\theta (\log \pi_\theta(S_t, A_t)) Q_w(S_t, A_t)$$

NYU

Introduction to Actor Critic Methods

- Learning Process
  - At each timestep t, we get the current state S_t
  - We pass S_t to our policy network (actor) π_θ and get an action A_t
  - We pass S_t , A_t to our value network (critic) $Q_w(s,a)$ and get the value of taking that action at that state $Q_w(S_t, A_t)$
  - We enter a new state S_{t+1} and receive reward R_{t+1}
  - Then actor updates its policy parameters using the Q value
  - The actor then produces the next action A_{t+1} to take at S_{t+1}
  - The critic then updates its value parameters using TD update

$$\Delta w = \underbrace{\beta}_{\text{Use a different Learning rate}}\underbrace{(R_{t+1} + \gamma Q_w(S_{t+1}, A_{t+1}) - Q_w(S_t, A_t))}_{\text{TD Error}}\underbrace{\nabla Q_w(S_t, A_t)}_{\text{Gradient of our value function}}$$

Introduction to Actor Critic Methods

- To further stabilize policy learning
  - Use **advantage function** as critic instead of the action value function

$$A(s,a) = \underbrace{Q(s,a)}_{\text{Q value for action a at state s}} - \underbrace{V(s)}_{\text{Average value of that state}} = \underbrace{r + \gamma V(s\prime) - V(s)}_{\text{TD Error}}$$

Can use the TD error as a good estimator of the advantage function

Introduction to Proximal Policy Optimization (PPO)

- (Side Note) This is **one of the most popular** method that we use to align LLMs nowadays

Introduction to PPO

- An architecture that improves our agent's training stability by **avoiding policy updates that are too large**
- **Reasons**
  - Empirically, smaller policy updates during training are more likely to converge to an optimal solution
  - A too-big step in a policy update can result in falling "off the cliff" (getting a bad policy) and taking a long time or even having no possibility to recover

Introduction to PPO

- Recap: The Policy Objective Function

$$L^{PG}(\theta) = \underbrace{\tilde{\mathbb{E}}_t}_{\text{Empirical average over a finite batch of samples}}[\log \pi_\theta(a_t|s_t)\underbrace{\tilde{A}_t}_{\text{Estimator of the advantage function at timestep t}}]$$

Empirical average over a finite batch of samples

Estimator of the advantage function at timestep t

NYU

Introduction to PPO

- Recap: The Policy Objective Function
- **Problem: step size**
    - Too small, the training process will be too slow
    - Too high, there will be too much variability in the training

Introduction to PPO

- Recap: The Policy Objective Function
- Problem: step size
- Introduce the **clipped surrogate objective function**

$$L^{CLIP}(\theta) = \tilde{\mathbb{E}}_t[\min(r_t(\theta)\tilde{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\tilde{A}_t)]$$

The ratio function

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$$

- If $r_t(\theta) > 1$ the action a_t at state s_t is more likely in the current policy than the old policy
- If $r_t(\theta)$ is between 0 and 1, the action is less likely for the current policy than for the old one

NYU

Introduction to PPO

- Recap: The Policy Objective Function
- Problem: step size
- Introduce the **clipped surrogate objective function**

$$L^{CLIP}(\theta) = \tilde{\mathbb{E}}_t[\min(r_t(\theta)\tilde{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\tilde{A}_t)]$$

The ratio function

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$$

- This ratio can replace the log probability we use in the policy objective function

NYU

Introduction to PPO

- Recap: The Policy Objective Function
- Problem: step size
- Introduce the **clipped surrogate objective function**

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \boxed{clip(r_t(\theta), 1-\epsilon, 1+\epsilon)}\hat{A}_t)\right]$$

Ensure that we do not have a too large policy update because the current policy can't be too different from the older one

NYU

Introduction to PPO

- Recap: The Policy Objective Function
- Problem: step size
- Introduce the **clipped surrogate objective function**

$$L^{CLIP}(\theta) = \tilde{\mathbb{E}}_t[\min(r_t(\theta)\tilde{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\tilde{A}_t)]$$

For more details, see the paper [Proximal Policy Optimization Algorithms]()

NYU

Questions?