Robot Perception and Learning

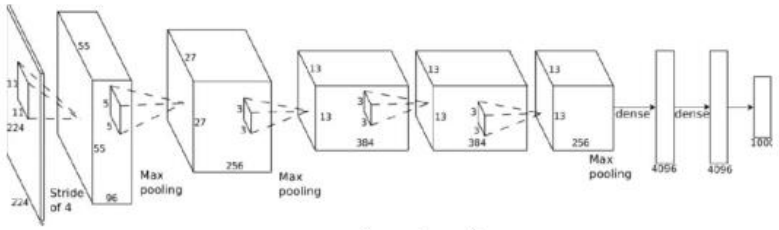
Multi-arm Bandits, Markov Decision Processes, Policy Iteration and Value Iteration

Tsung-Wei Ke

Fall 2025

The Formulation of Imitation Learning is Limited

$\mathbf{o}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

$\mathbf{a}_t$

$o_1^1, a_1^1, o_2^1, a_2^1, \ldots, o_T^1, a_T^1$
$o_1^2, a_1^2, o_2^2, a_2^2, \ldots, o_T^2, a_T^2$
$\vdots$
$o_1^N, a_1^N, o_2^N, a_2^N, \ldots, o_T^N, a_T^N$

Demonstration dataset

sample

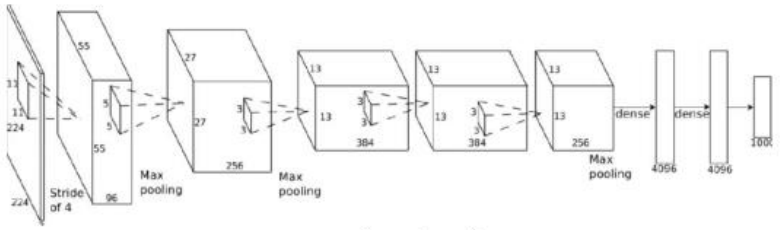
o_t - observation
a_t - action

$\pi_\theta(a_t|o_t)$ - policy

- Models learn only "the mapping" between observations and actions, instead of "the effect" of each action on an observation

Image credit Ali Yahya and S. Levine

The Formulation of Imitation Learning is Limited



$\mathbf{o}_t$ $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ $\mathbf{a}_t$

$o_1^1, a_1^1, o_2^1, a_2^1, \ldots, o_T^1, a_T^1$
$o_1^2, a_1^2, o_2^2, a_2^2, \ldots, o_T^2, a_T^2$
$\vdots$
$o_1^N, a_1^N, o_2^N, a_2^N, \ldots, o_T^N, a_T^N$

Demonstration dataset

sample

o_t - observation
a_t - action

$\pi_\theta(a_t|o_t)$ - policy

- Models learn only "the mapping" between observations and actions, instead of "the effect" of each action on an observation
  - Can we achieve the goal with the current action?
  - How long does it take to achieve the goal?
  - How will the world change?

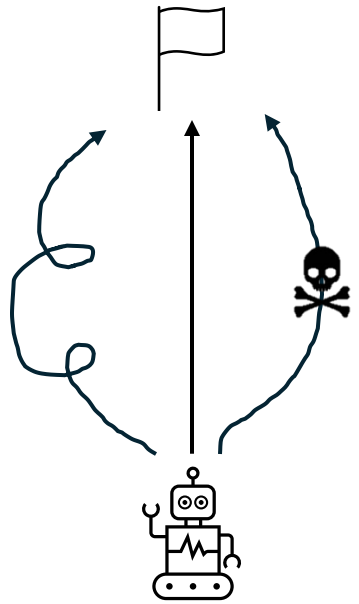
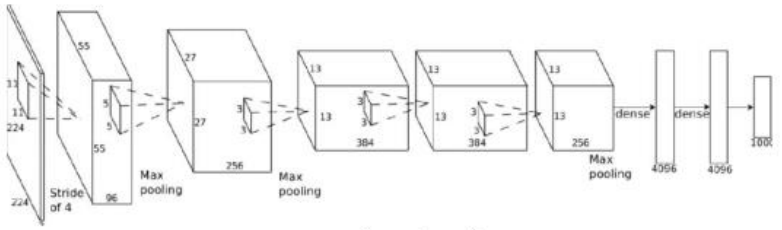


Image credit Ali Yahya and S. Levine

The Formulation of Imitation Learning is Limited



$\mathbf{o}_t$ $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ $\mathbf{a}_t$

$o_1^1, a_1^1, o_2^1, a_2^1, \ldots, o_T^1, a_T^1$
$o_1^2, a_1^2, o_2^2, a_2^2, \ldots, o_T^2, a_T^2$
$\vdots$
$o_1^N, a_1^N, o_2^N, a_2^N, \ldots, o_T^N, a_T^N$

Demonstration dataset

sample

o_t - observation
a_t - action

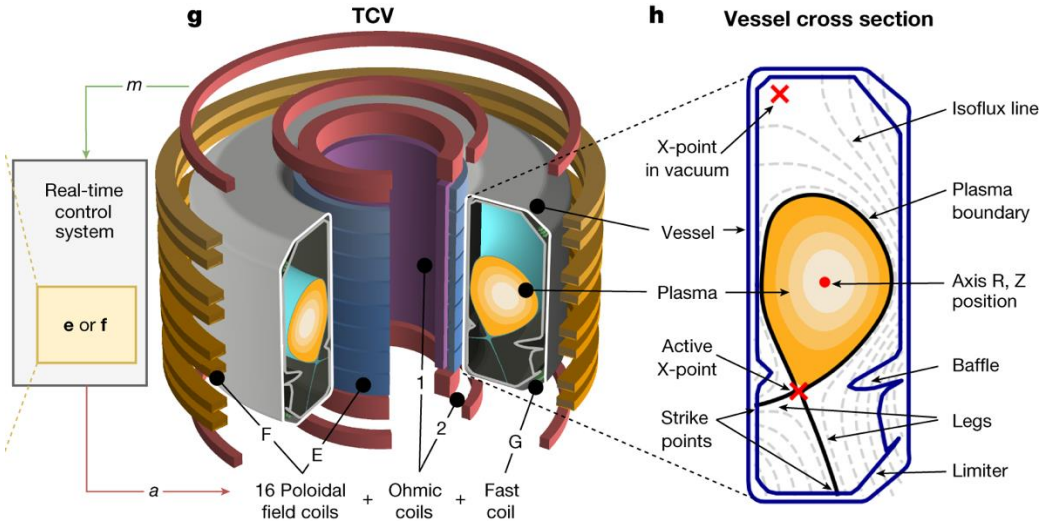
$\pi_\theta(a_t|o_t)$ - policy

- Models learn only "the mapping" between observations and actions, instead of "the effect" of each action on an observation
- Imitation learning assumes the existence of an expert, limiting the application scenario

Image credit Ali Yahya and S. Levine

Can We Have a General Learning Framework for All Different Tasks?

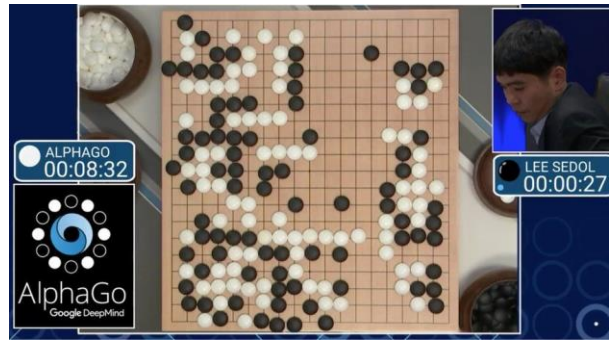
Power Plant Control



Magnetic control of tokamak plasmas through deep reinforcement learning. Degrave et al.

Trading

Gaming



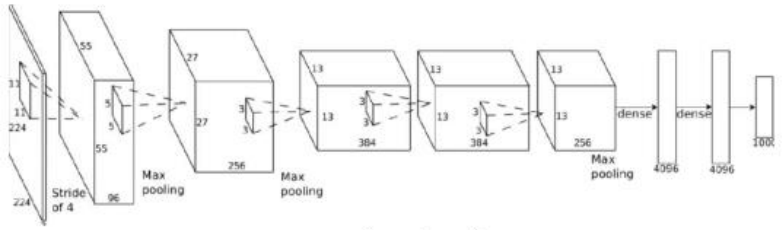
Adaptive VisuoMotor Control

AnyCar to Anywhere: Learning Universal Dynamics Model for Agile and Adaptive Mobility. Xiao et al.

Content Generation

Training Animals

The Formulation of Imitation Learning is Limited



$\mathbf{o}_t$ $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ $\mathbf{a}_t$

$o_1^1, a_1^1, o_2^1, a_2^1, \ldots, o_T^1, a_T^1$
$o_1^2, a_1^2, o_2^2, a_2^2, \ldots, o_T^2, a_T^2$
$\vdots$
$o_1^N, a_1^N, o_2^N, a_2^N, \ldots, o_T^N, a_T^N$

Demonstration dataset

sample

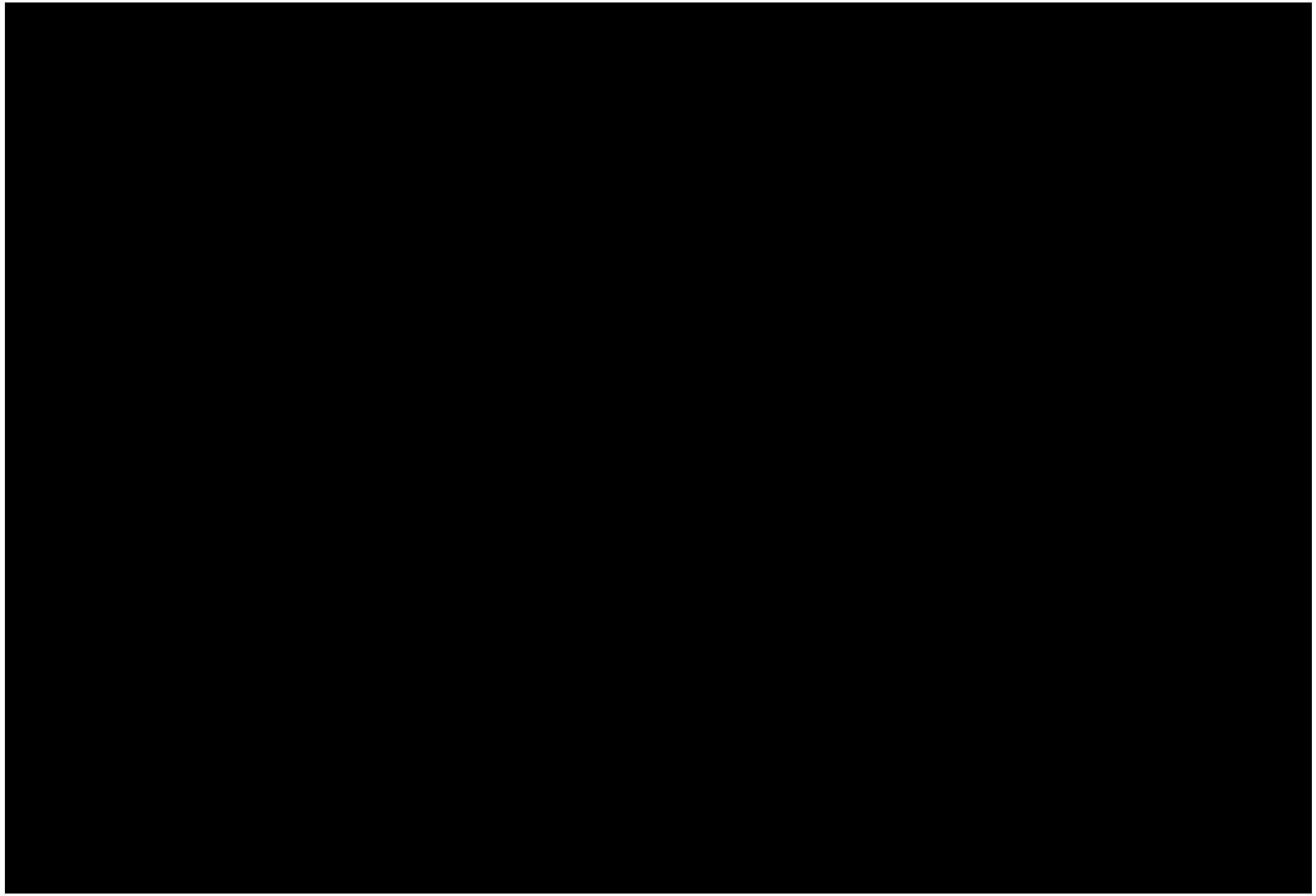
o_t - observation
a_t - action

$\pi_\theta(a_t|o_t)$ - policy

- Models learn only "the mapping" between observations and actions, instead of "the effect" of each action on an observation
- Imitation learning assumes the existence of an expert, limiting the application scenario
- Imitation learning is "at best" as good as the expert

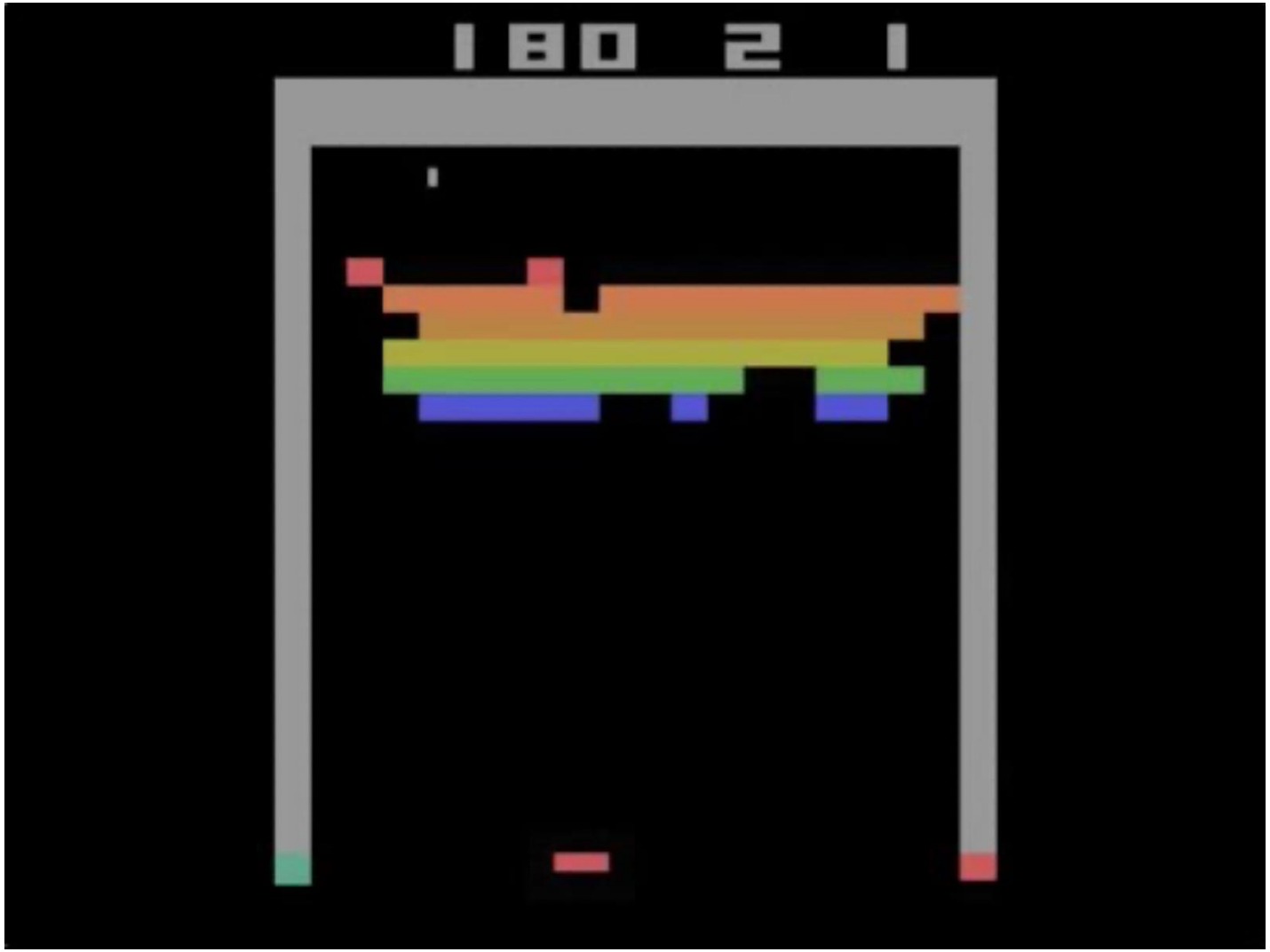
Image credit Ali Yahya and S. Levine

Can We Reformulate the Learning Problem with Rewards not Action Labels?



https://youtu.be/yhvaSEJtOV8?si=jjW_kZP5FT2JX7wk

Can We Reformulate the Learning Problem with Trial-and-Error not Imitation?



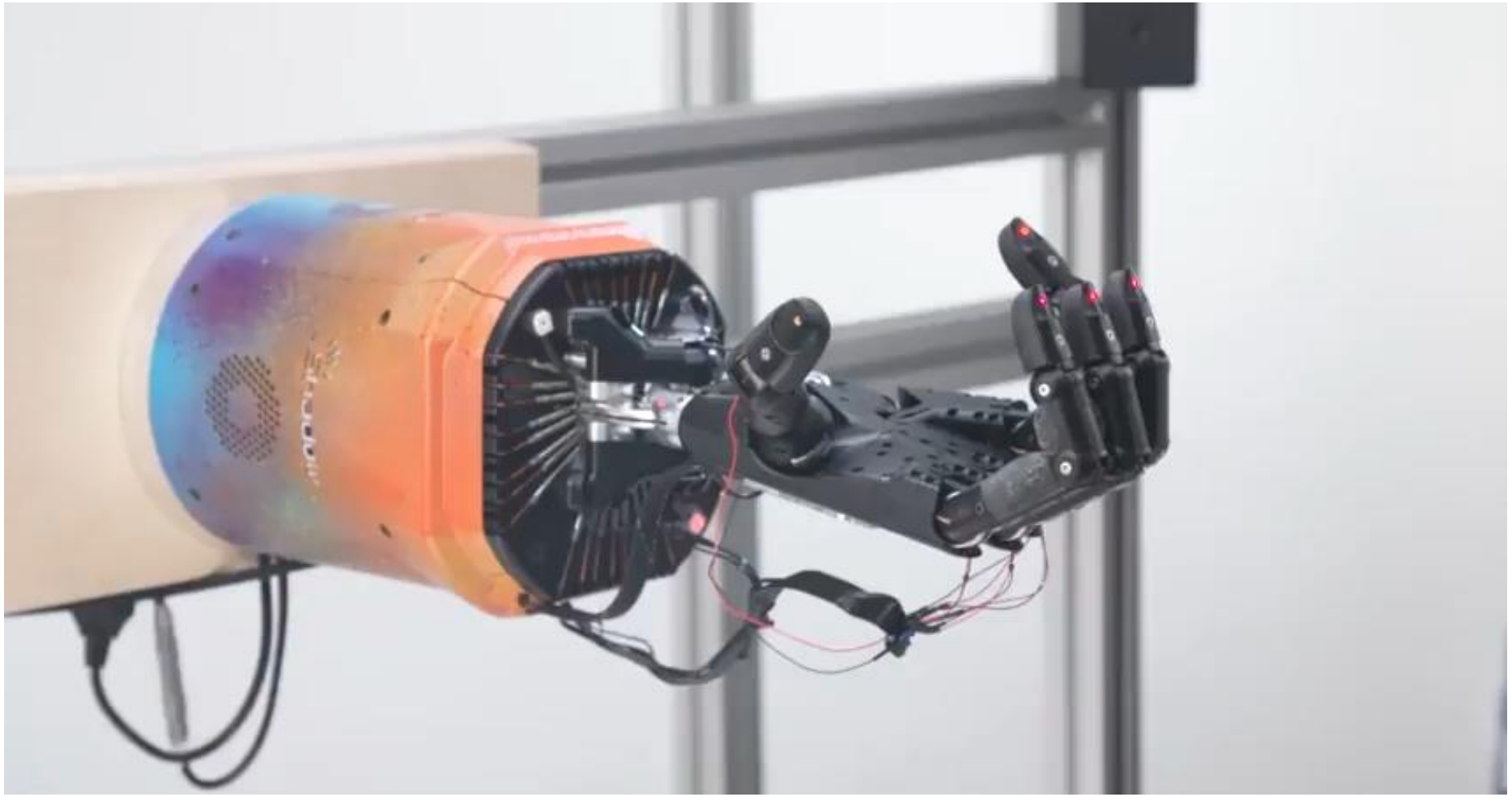
A Learning Framework that Solves Locomotion

RMA: Rapid Motor Adaptation for Legged Robots. Kumar et al.

A Learning Framework that Solves Complex Locomotion

Extreme Parkour with Legged Robots. Chen et al.

A Learning Framework that Solves Rubik's Cube

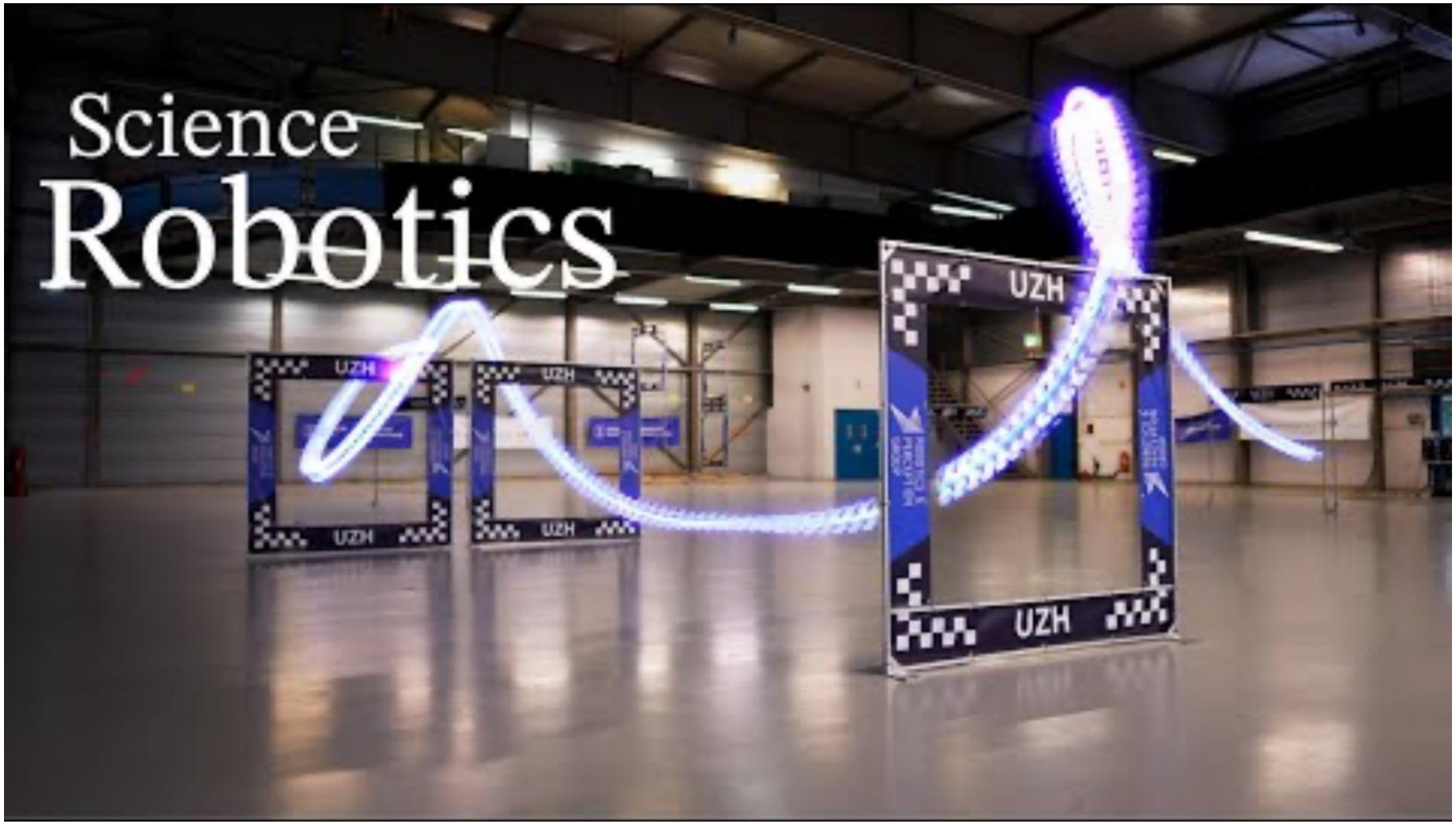


Solving Rubik's Cube with a Robot Hand. OpenAI.

A Learning Framework that Solves Multi-Agent Competition

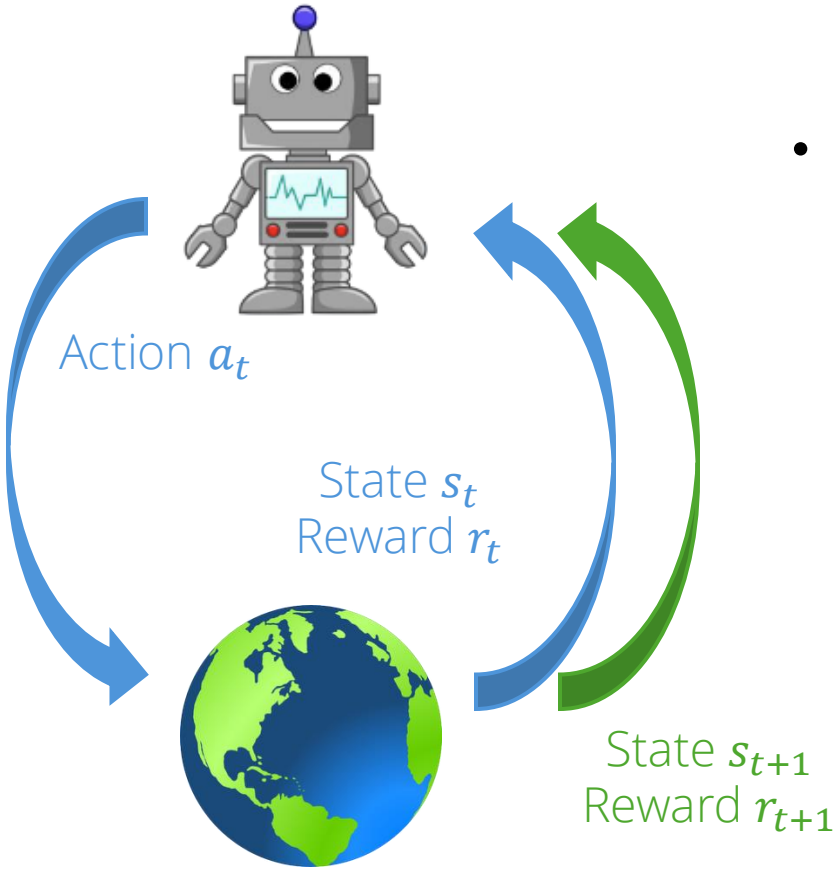
Learning Agile Soccer Skills for a Bipedal Robot with Deep Reinforcement Learning. DeepMind.

A Learning Framework that Surpasses Experts

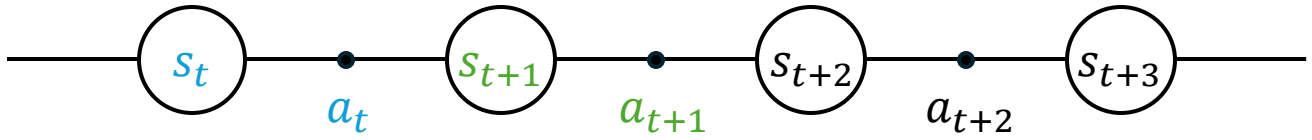


Champion-level drone racing using deep reinforcement learning. Kaufmann et al.

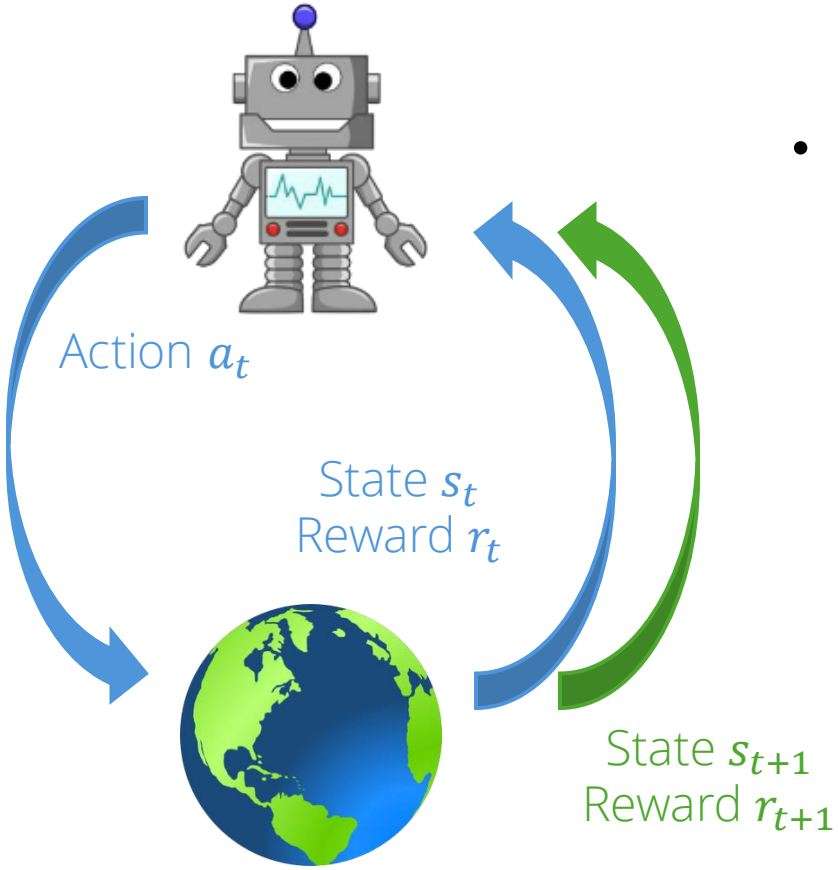
Actions Induce Changes of the Environment



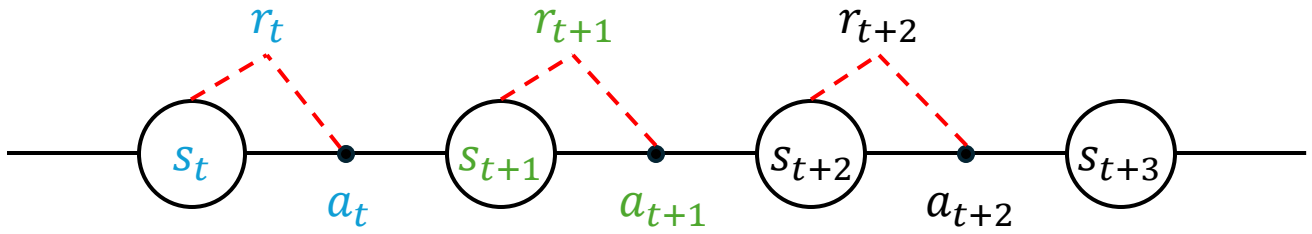
- A trajectory of interaction in the environment



The Reward Function $r(s, a)$ Estimates Goodness of Each State-Action Pair



- A trajectory of interaction in the environment



high reward

low reward

? reward

In fact, Defining Reward Functions is Tricky…

Success ✓

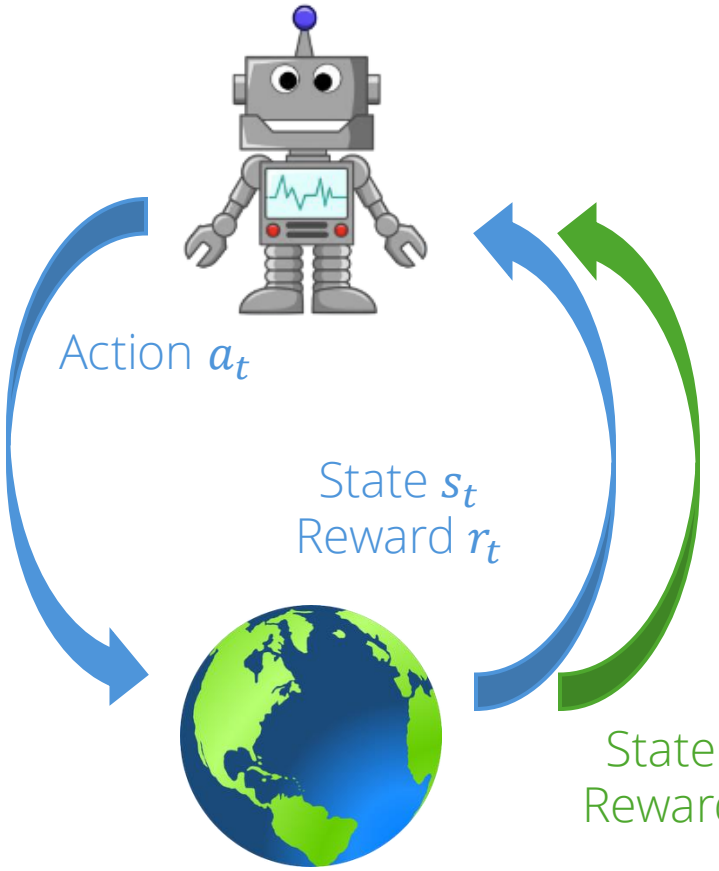
Fail ✗

Leading others ?
Energy efficiency ?
Tire wear ?
Safety ?
Acceleration smoothness ?
Circular motion ?

- The most general form of rewards
  - ➢ Don't need domain knowledge
- The sparsest form of rewards
  - ➢ Credit assignment issue
- Learning with only these rewards is sample inefficient

- The biased form of rewards
  - ➢ Need domain knowledge
- Denser forms of rewards
  - ➢ Guide / facilitate learning with expert knowledge / prior
- Require huge human efforts
  - ➢ LLMs sometimes can help

Reinforcement Learning: Learns to Maximize the Total Reward of an Episode of Interaction



- A trajectory of interaction in the environment

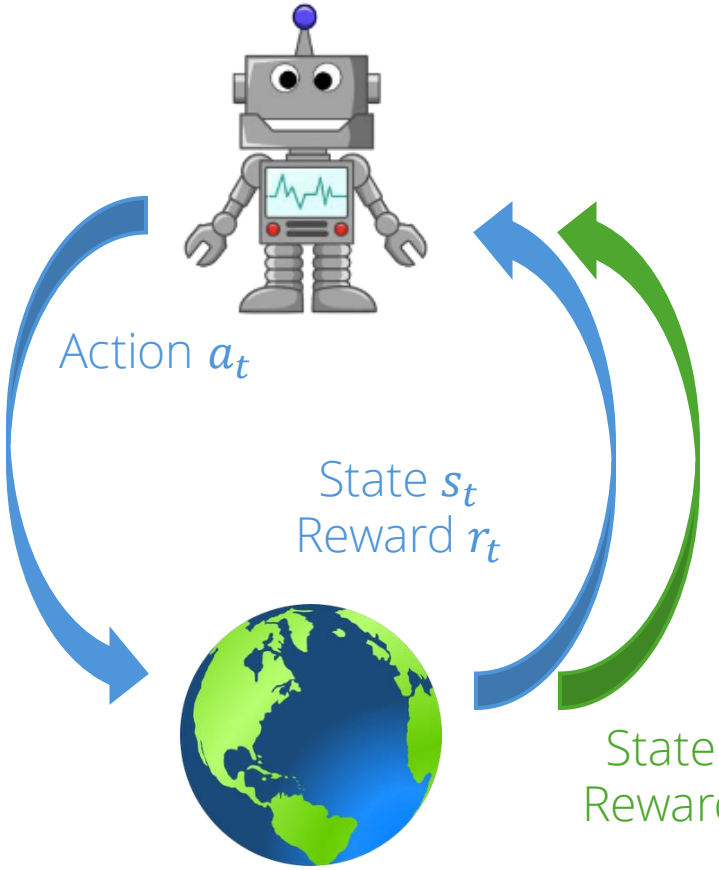
$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

- Maximize the expected value of the cumulative sum of reward

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Slide adapted from S. Levine

Reinforcement Learning: Learns to Maximize the Total Reward of an Episode of Interaction



- A trajectory of interaction in the environment

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

Previous actions decide what future states we will perceive

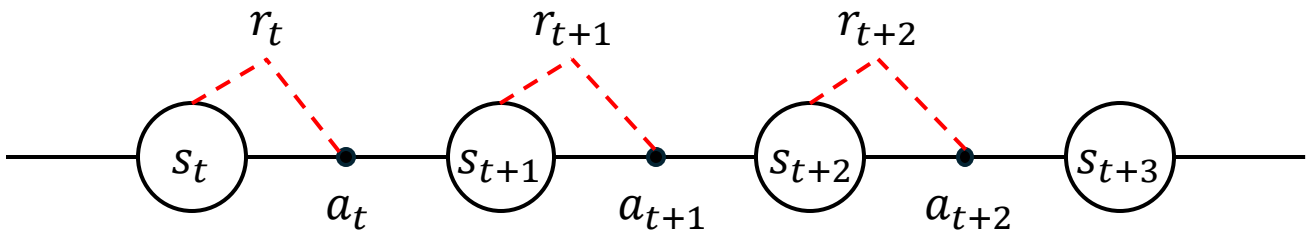
- Maximize the expected value of the cumulative sum of reward

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Slide adapted from S. Levine

A Sequence of Actions are Not Temporally Independent…

- An action induce changes of the environment state
  - ➢ An action may have low reward now, but lead to very high-reward future states
  - ➢ An action may have high reward now, but lead to very low-reward future states

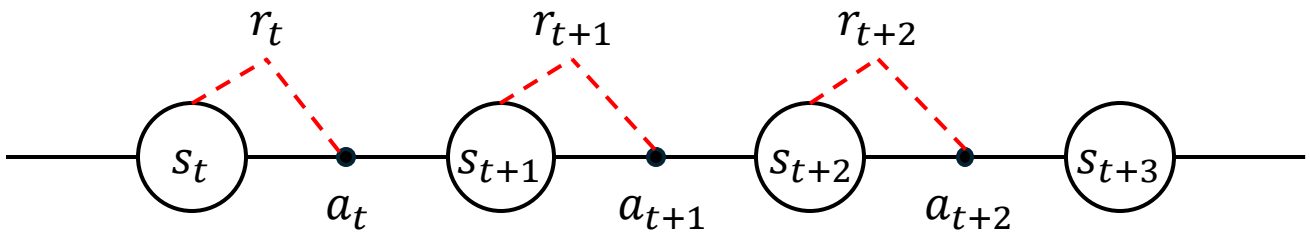


$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Consider an entire episode of the interaction, not each individual action

A Sequence of Actions are Not Temporally Independent…

- An action induce changes of the environment state
  - ➢ An action may have low reward now, but lead to very high-reward future states
  - ➢ An action may have high reward now, but lead to very low-reward future states



Or, we can have better estimation of state-action reward, that consider possible future rewards?

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

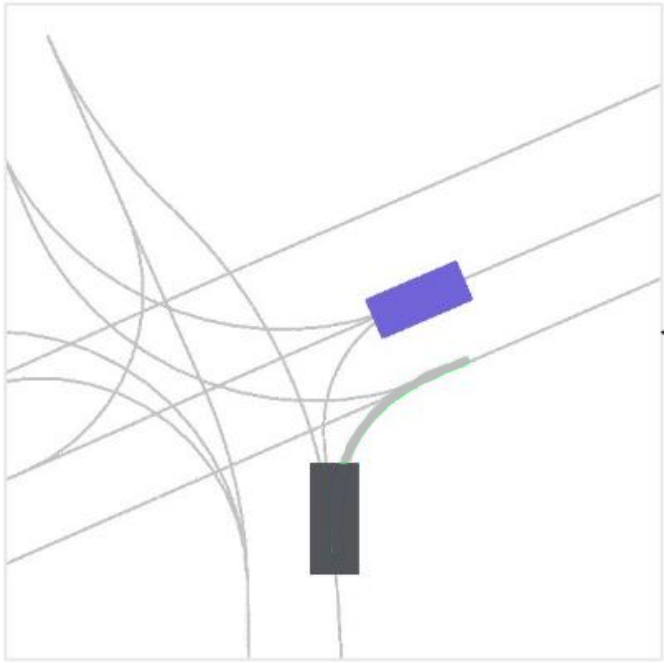
Consider an entire episode of the interaction, not each individual action

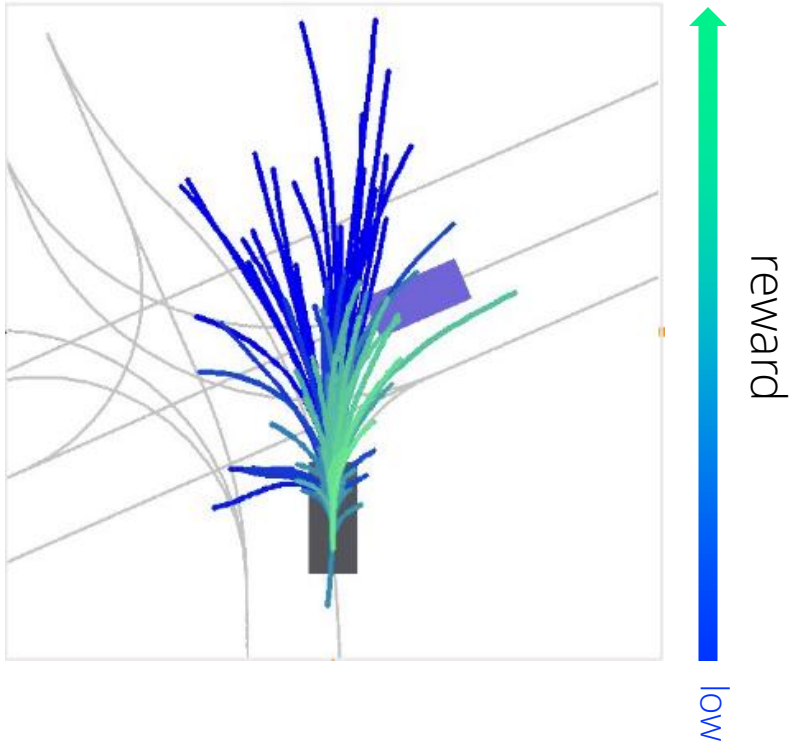
Reinforcement Learning: Trial-and-Error Learning

https://youtu.be/kojH8a7BW04?si=Kc5H1cENuslClgxT

Reinforcement Learning: Trial-and-Error Learning

Task: make a right turn



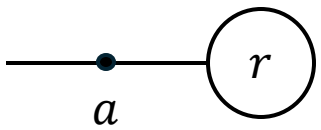


- Explore: discover actions that may have high rewards, but usually are suboptimal
- Exploit: take the current best action

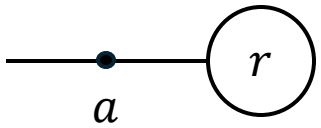
Diffusion-ES: Gradient-free Planning with Diffusion for Autonomous Driving and Zero-Shot Instruction Following. Yang et al.

Let's Consider a Non-Sequential Setup

- Non-sequential setup:
  - Each action results in an **immediate reward**
  - We want to choose actions that maximize our immediate reward in expectation
  - There is no state



Let's Consider a Non-Sequential Setup



- Non-sequential setup:
  - Each action results in an **immediate reward**
  - We want to choose actions that maximize our immediate reward in expectation
  - There is no state

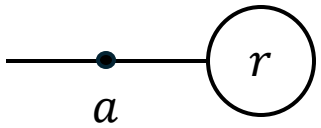
- For example, choosing which restaurant to go
  - **Actions**: the restaurants to choose from
  - **Rewards**: your happiness

- Let's say you can eat outside 100 times, what's the best strategy to maximize your total happiness?
  - **Explore**: discover new restaurants
  - **Exploit**: go to the favorite restaurant

- Without exploration, you may end up going to 7-11 all the time; Without exploitation, you keep trying new but bad restaurants.

Slide adapted from K. Fragkiadaki

Let's Consider a Non-Sequential Setup



- Non-sequential setup:
  - Each action results in an **immediate reward**
  - We want to choose actions that maximize our immediate reward in expectation
  - There is no state

- For example, choosing which restaurant to go
  - **Actions**: the restaurants to choose from
  - **Rewards**: your happiness

- Let's say you can eat outside 100 times, what's the best strategy to maximize your total happiness?
  - **Explore**: discover new restaurants
  - **Exploit**: go to the favorite restaurant

This is the key question in RL

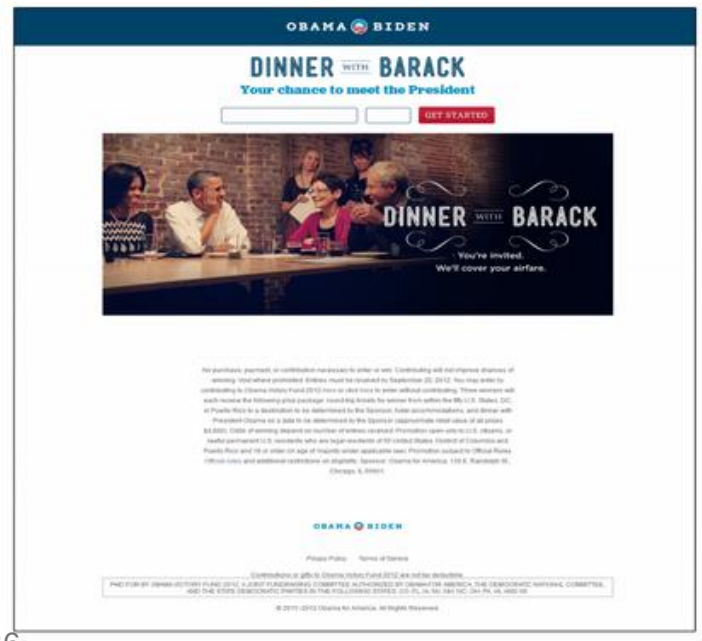
- Without exploration, you may end up going to 7-11 all the time; Without exploitation, you keep trying new but bad restaurants.

Slide adapted from K. Fragkiadaki

Real world motivation: content presentation

We have two variations of content of a webpage, A and B, and we want to decide which one to display to engage more users.

- Two arm bandits: each arm corresponds to a content variation shown to users (not necessarily the same user).
- Reward: 1 if the user clicks, 0 otherwise.
- Mean reward (success probability) for each invitation: the click-through-rate, the percentage of users that click on each ad

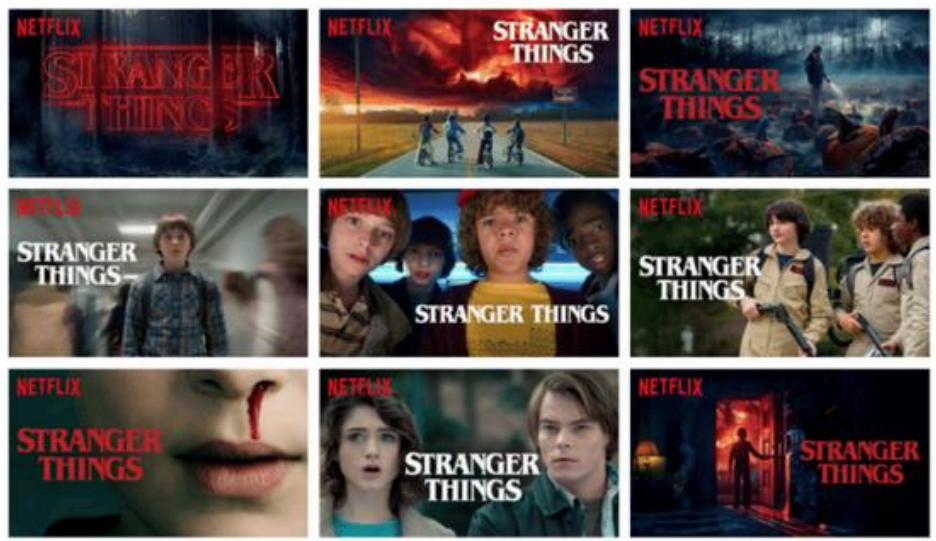


Slide credit K. Fragkiadaki

Real world motivation: NETFLIX artwork

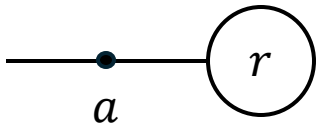
For a particular movie, we want to decide what image to show (to all the NEFLIX users)

- Actions: uploading one of the K images to a user's home screen
- Reward: 1 if the user clicks and watches, 0 otherwise.
- Mean reward (success probability) for each image: the percentage of users that clicked and watched



Netflix Artwork

Let's Consider a Non-Sequential Setup



- Non-sequential setup:
  - Each action results in an **immediate reward**
  - We want to choose actions that maximize our immediate reward in expectation
  - There is no state
- This simplified setup is also called "Bandit Problem"
- This simplified setup helps us to focus on key components / tools in RL

Slide adapted from K. Fragkiadaki

Multi-Armed Bandit Problem

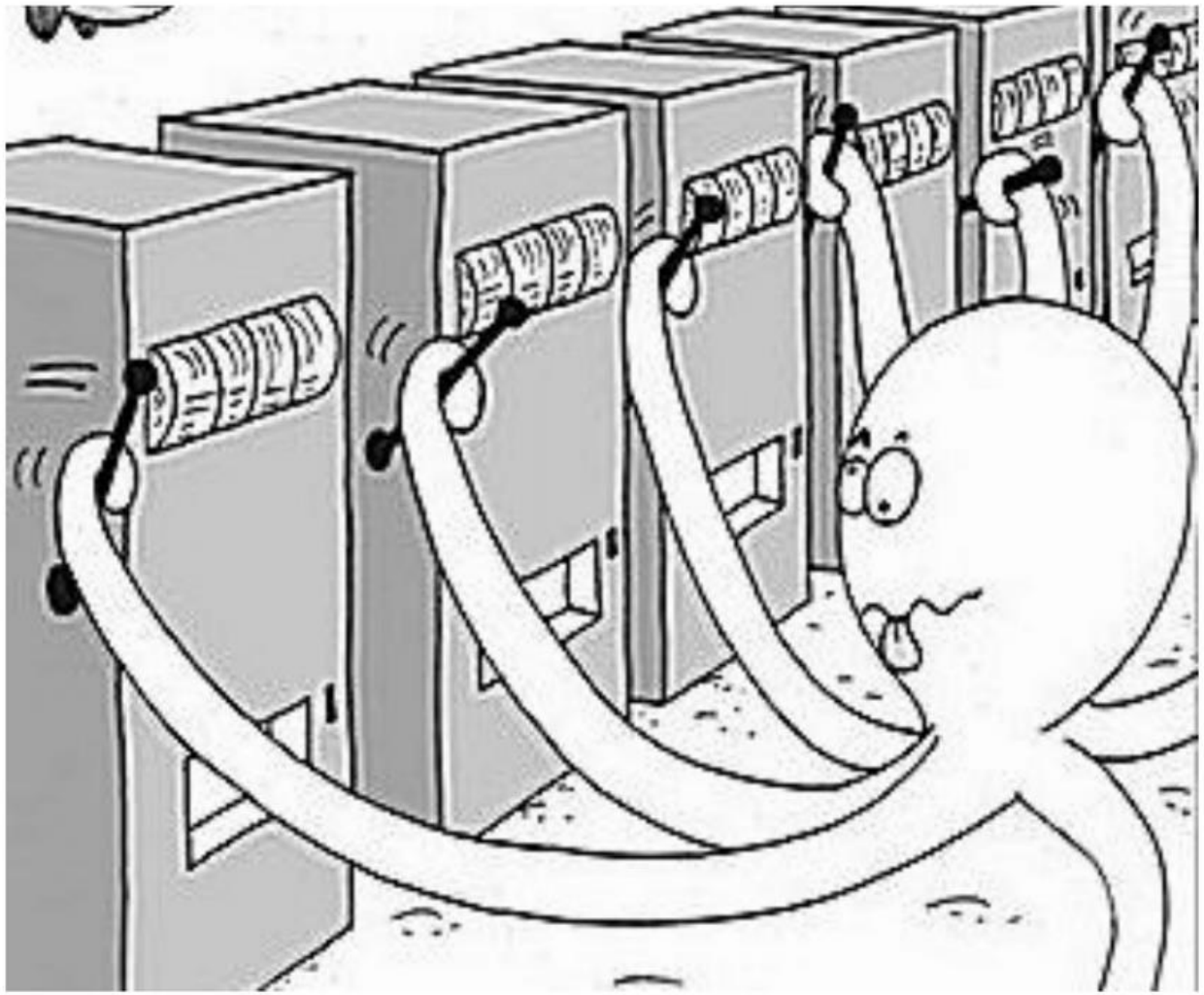


Image credit Microsoft

Multi-Armed Bandit Problem

$r_1 \sim P_1$

$P_1 = \mathcal{N}(\mu_1, \sigma_1)$

$r_2 \sim P_2$

$P_2 = \mathcal{N}(\mu_2, \sigma_2)$

$r_3 \sim P_3$

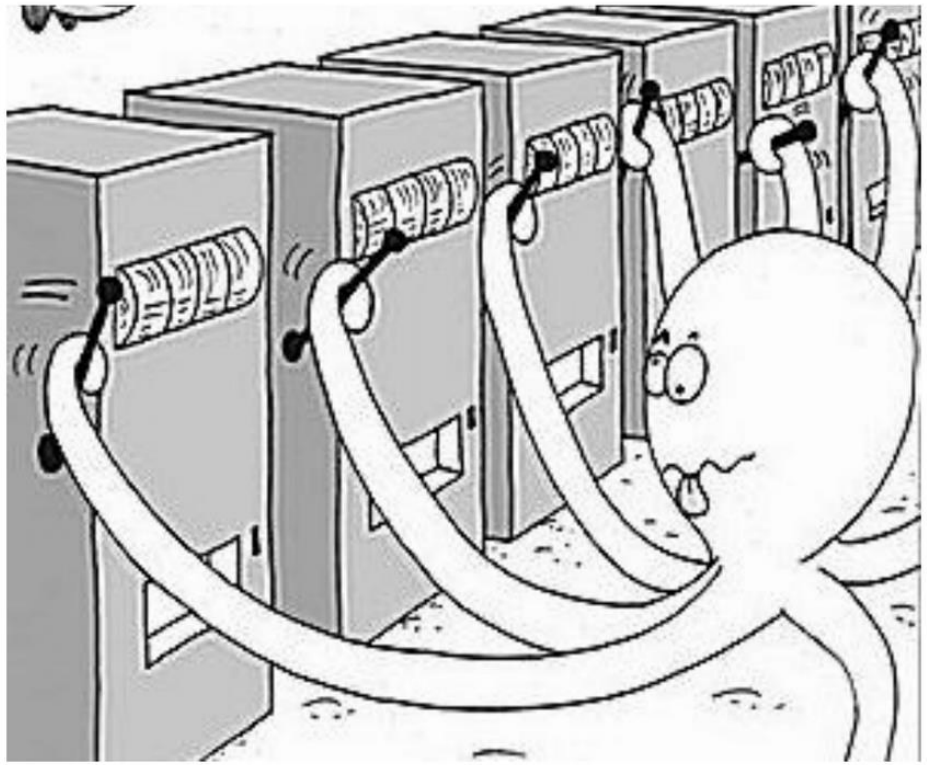
$P_3 = \mathcal{N}(\mu_3, \sigma_3)$

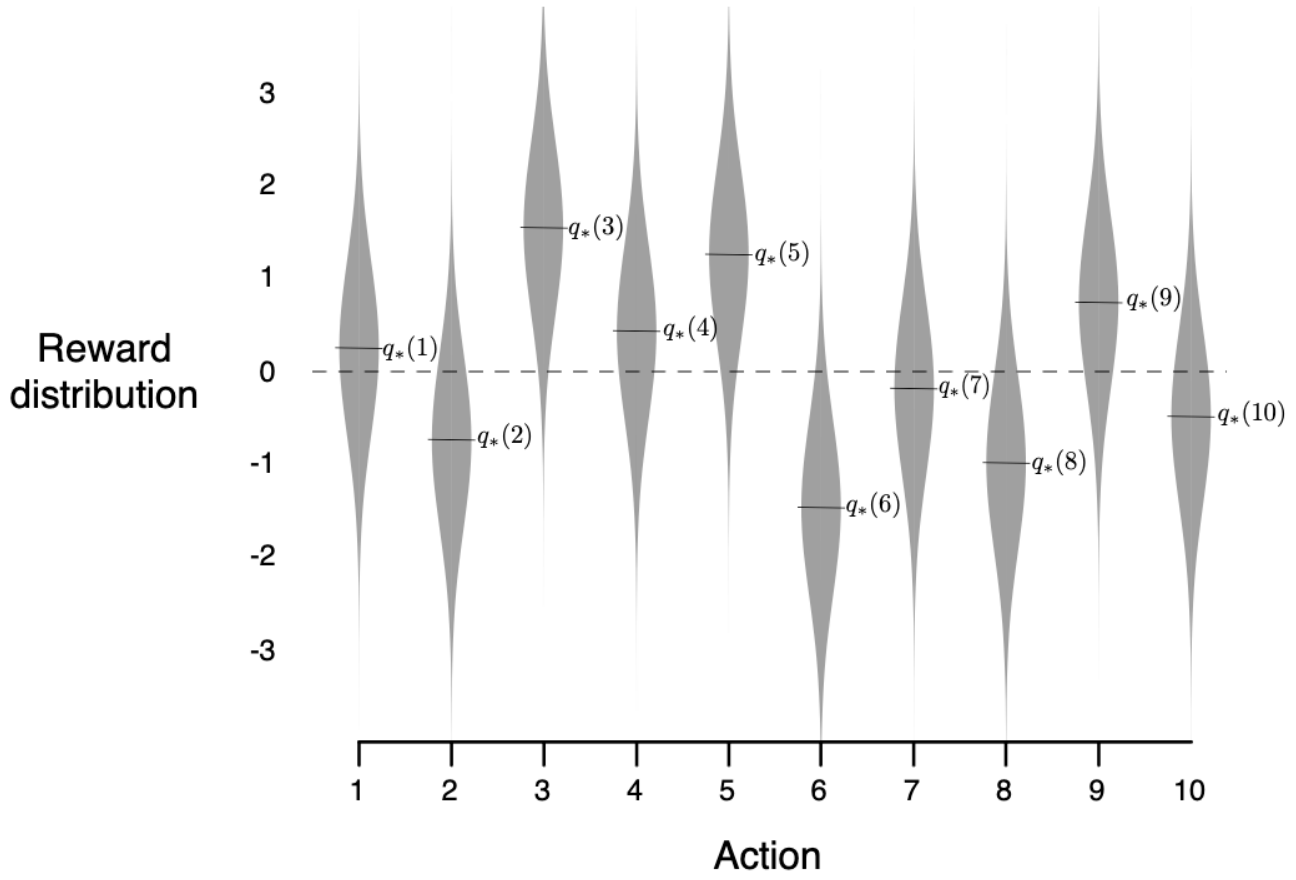
- At timestep **t**
  - The agent plays one of the **K** arms
  - The **k**th arm produces reward $r_{k,t}$ when played
  - The reward $r_{k,t}$ is drawn from a probability distribution P_k with mean μ_k and std σ_k (Note: the distribution could be non-Gaussian).
  - The real reward distribution is unknown

- How to maximize the total reward for playing the bandit machines within a finite or infinite horizon?

Slide adapted from K. Fragkiadaki

What's the Strategy if the Reward Distribution is Known





If the distribution is known, the best strategy is to **exploit**: play the arm with the highest expected reward

Image credit Microsoft

How to Formulate the Behavior of Exploitation?

- Expected reward: $q^*(a_k) = \mathbb{E}[r_t | A_t = a_k]$
- Action-value estimates: $Q_t(a_k)$ ← Characterizes how good an action a_k is
- **Greedy action selection method**: select the action with the highest estimated value:

$$A_t^* = \arg\max_a Q_t(a)$$

  - If $A_t = A_t^*$, you are *exploiting* your current knowledge of the values of the actions
  - If $A_t \neq A_t^*$, you are *exploring*. You improve your estimate of the non-greedy actions

How to Formulate the Behavior of Exploitation?

- Expected reward: $q^*(a_k) = \mathbb{E}[r_t | A_t = a_k]$
- Action-value estimates: $Q_t(a_k)$ ← Characterizes how good an action a_k is
- **Greedy action selection method**: select the action with the highest estimated value:

$$A_t^* = \arg\max_a Q_t(a)$$

  - If $A_t = A_t^*$, you are *exploiting* your current knowledge of the values of the actions
  - If $A_t \neq A_t^*$, you are *exploring*. You improve your estimate of the non-greedy actions

- If $Q_t(a_k) \approx q^*(a_k)$, no need for exploration. Greedy action selection is the best strategy.
- If $Q_t(a_k) \neq q^*(a_k)$, you need exploring different actions to refine your action-value estimation, otherwise, you end up choosing sub-optimal actions

Estimate Values of Actions by Sample-Average Method

- Expected reward: $q^*(a_k) = \mathbb{E}[r_t | A_t = a_k]$
- Action-value estimates: $Q_t(a_k)$
- Sample-average method: Average the rewards actually received:

$$Q_t(a_k) \coloneqq \frac{\text{sum of rewards when } a_k \text{ taken prior to } t}{\text{number of times } a_k \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} r_i \cdot \mathbf{1}_{A_i = a_k}}{\sum_{i=1}^{t-1} \mathbf{1}_{A_i = a_k}}$$

1. When you have enough samples ($\sum_{i=1}^{t-1} \mathbf{1}_{A_i = a_k} \to \infty$): $Q_t(a_k) \approx q^*(a_k)$
2. When you don't have enough samples, $Q_t(a_k)$ differs from $q^*(a_k)$

Estimate Values of Actions by Sample-Average Method

- Expected reward: $q^*(a_k) = \mathbb{E}[r_t | A_t = a_k]$
- Action-value estimates: $Q_t(a_k)$
- **Sample-average method**: Average the rewards actually received:

$$Q_t(a_k) \coloneqq \frac{\text{sum of rewards when } a_k \text{ taken prior to } t}{\text{number of times } a_k \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} r_i \cdot \mathbf{1}_{A_i = a_k}}{\sum_{i=1}^{t-1} \mathbf{1}_{A_i = a_k}}$$

1. When you have enough samples ($\sum_{i=1}^{t-1} \mathbf{1}_{A_i = a_k} \to \infty$): $Q_t(a_k) \approx q^*(a_k)$
2. When you don't have enough samples, $Q_t(a_k)$ differs from $q^*(a_k)$

➢ We need to keep a huge table to store the received rewards

Estimate Values of Actions by Incremental Implementation

- Let Q_n denote the estimate of its action value after being selected $n-1$ times

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

- Let Q_n denote the estimate of its action value after being selected $n-1$ times

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

- Let's start rewriting Q_n:

$$\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}$$

Estimate Values of Actions by Incremental Implementation

- Let Q_n denote the estimate of its action value after being selected $n-1$ times

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

- Let's start rewriting Q_n:

$$Q_{n+1} \quad = \quad Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],$$

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize} \times [\text{Target} - \text{OldEstimate}]$$

Estimate Values of Actions by Incremental Implementation

- Let Q_n denote the estimate of its action value after being selected $n-1$ times

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

- Let's start rewriting Q_n:

$$Q_{n+1} = Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],$$

Error

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize} \times [\text{Target} - \text{OldEstimate}]$$

Doesn't this look familiar? This is a standard form for learning/update rules!

Wait! We Assumed Stationary Reward Distributions What if Reward Distributions are Nonstationary

- A distribution is **stationary** if it is fixed over time, otherwise, it is **nonstationary**
- If a distribution is nonstationary, we should trust recent rewards more than long-past rewards.
- We can re-write the incremental implementation with a step-size parameter $\alpha \in (0,1]$, which denotes how much we emphasize on recent rewards.

$$Q_{n+1} \doteq Q_n + \alpha\Big[R_n - Q_n\Big]$$

Exponential Recency-Weighted Average Method

$$
\begin{aligned}
Q_{n+1} &= Q_n + \alpha\Big[R_n - Q_n\Big] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha)Q_{n-1}\right] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2\alpha R_{n-2} + \\
&\qquad \cdots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i.
\end{aligned}
$$

Exponential Recency-Weighted Average Method

$$
\begin{aligned}
Q_{n+1} &= Q_n + \alpha\Big[R_n - Q_n\Big] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha)Q_{n-1}\right] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2\alpha R_{n-2} + \\
&\qquad \cdots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i .
\end{aligned}
$$

How fast a term is forgotten

The Exploration / Exploitation Dilemma

- Exploitation: Make the best decision given current information
- Exploration: Sacrifice some action budget to gather more information

- Key question: when to explore and when to exploit?

- Idea 1: ε-greedy method, a naïve solution that decide to explore / exploit by throwing a dice with probability ε

Slide adapted from K. Fragkiadaki

ε-Greedy Method

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

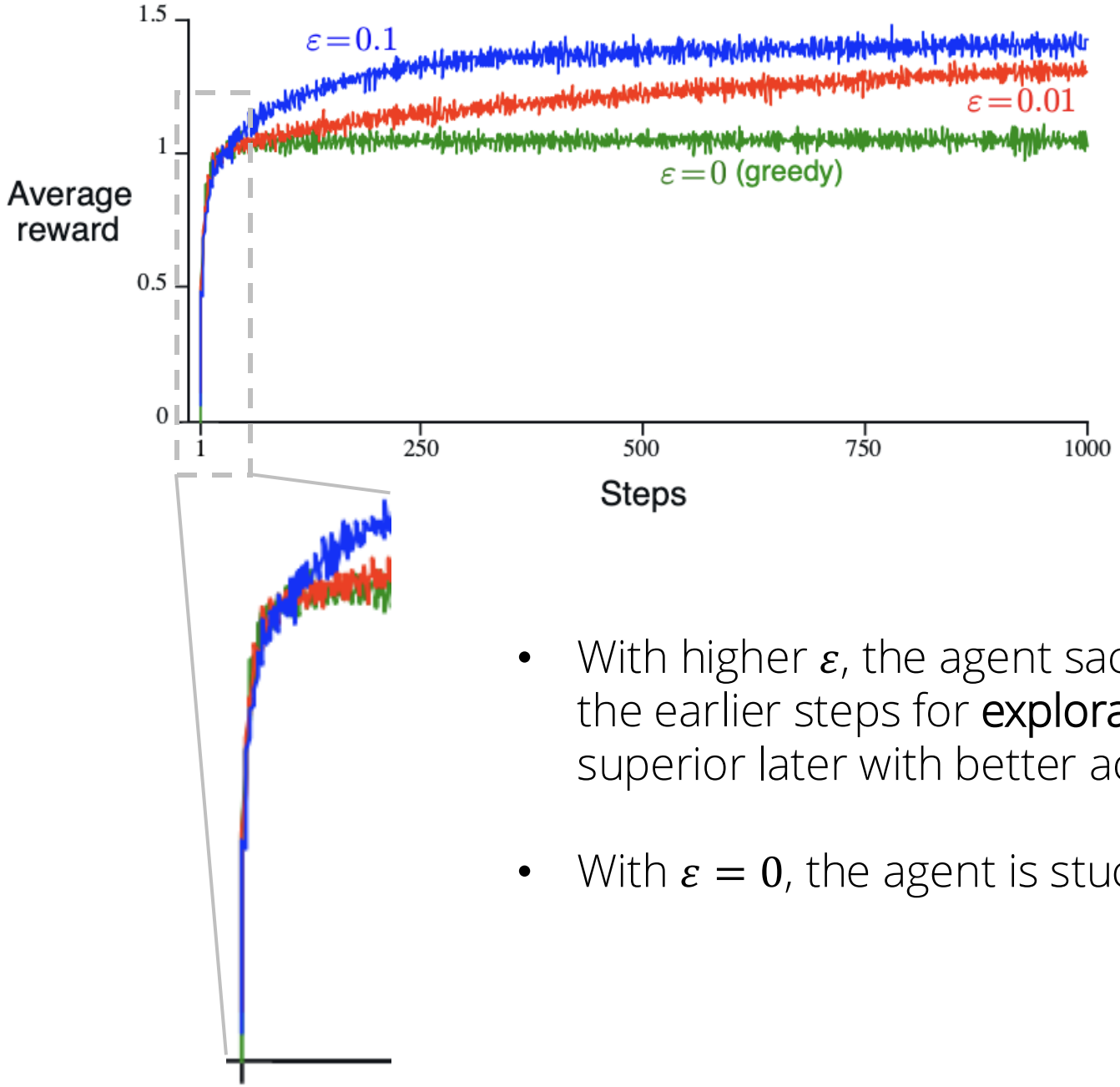
Explore or Exploit?

$$A \leftarrow \begin{cases} \text{argmax}_a\, Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

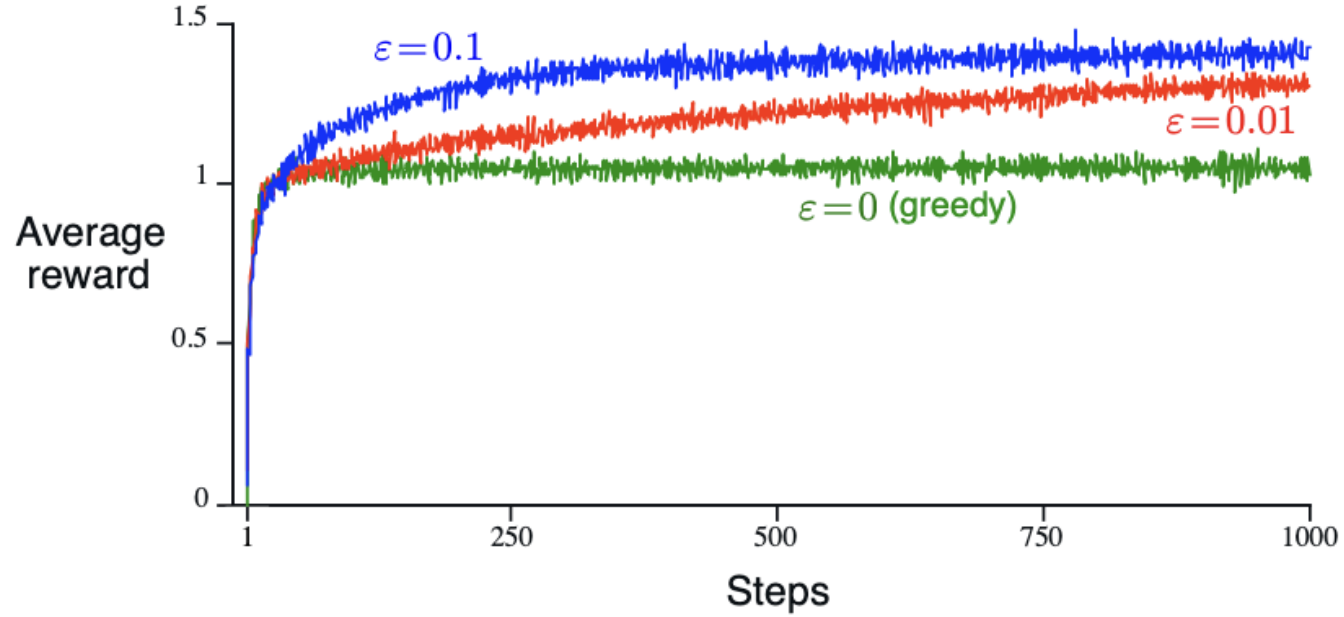
$R \leftarrow bandit(A)$

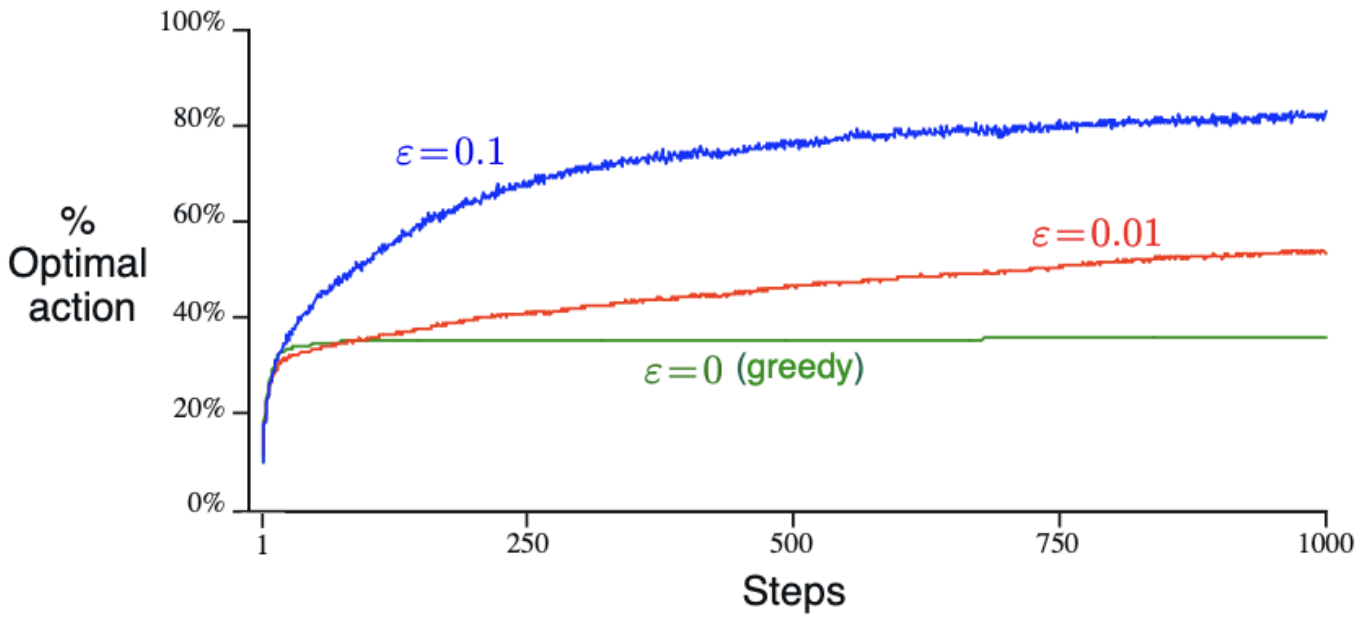
$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$ Update the action-value estimation..



- With higher ε, the agent sacrifice performance in the earlier steps for **exploration**, but it becomes superior later with better action-value estimation
- With $\varepsilon = 0$, the agent is stuck in the local minimum


1.5
1
0.5
0
Average reward
ε = 0.1
ε = 0.01
ε = 0 (greedy)
1
250
500
750
1000
Steps

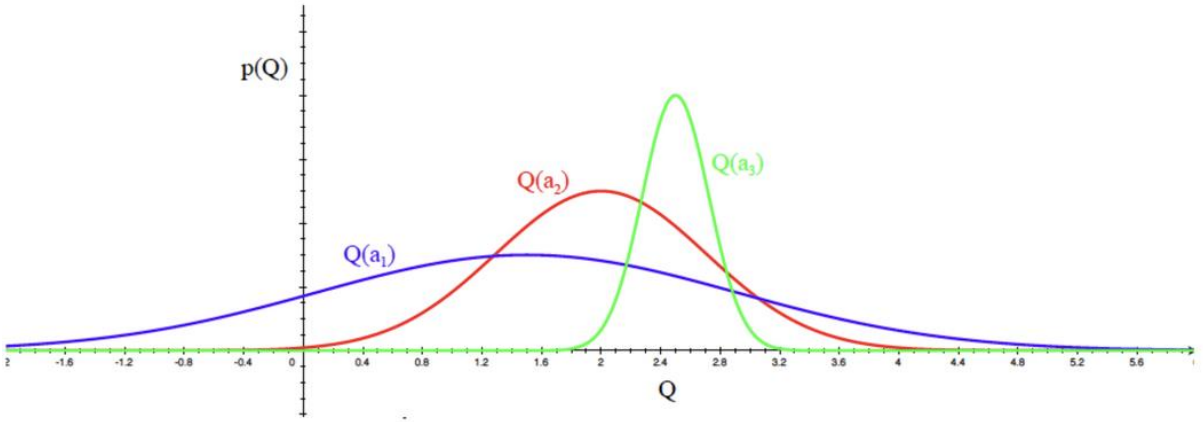

100%
80%
60%
40%
20%
0%
% Optimal action
ε = 0.1
ε = 0.01
ε = 0 (greedy)
1
250
500
750
1000
Steps

The Exploration / Exploitation Dilemma

- Exploitation: Make the best decision given current information
- Exploration: Sacrifice some action budget to gather more information

- Key question: when to explore and when to exploit?

- Idea 1: ε-greedy method, a naïve solution that decide to explore / exploit by throwing a dice with probability ε
- Idea 2: upper confidence bound (UCB) method, the more **uncertain** we are about an action-value, the more important it is to **explore** that action



Upper Confidence Bounds

- Estimate an upper confidence $U_t(a)$ for each action value such that with high probability:

$$q_*(a) \leq Q_t(a) + U_t(a)$$

Estimated mean

Estimated Upper Confidence

- This upper confidence depends on the number of times action a has been selected
  - Small $N_t(a) \Rightarrow$ large $U_t(a)$ (estimated value is uncertain)
  - Large $N_t(a) \Rightarrow$ small $U_t(a)$ (estimated value is accurate)
- Select action maximizing Upper Confidence Bound (UCB)

$$a_t = \text{argmax}_{a \in \mathcal{A}} Q_t(a) + U_t(a)$$

Slide credit K. Fragkiadaki

Upper Confidence Bound (UCB)

- A clever way of reducing exploration over time
- Estimate an upper bound on the true action values
- Select the action with the largest (estimated) upper bound

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$

- c is a hyper-parameter that trades-off explore/exploit
- the confidence bound grows with the total number of actions we have taken t but shrinks with the number of times we have tried this particular action $N_t(a)$. This ensures each action is tried infinitely often but still balances exploration and exploitation.
  - t: how many times I have played any action,
  - $N_t(a)$: how many times I have played action a in t interactions

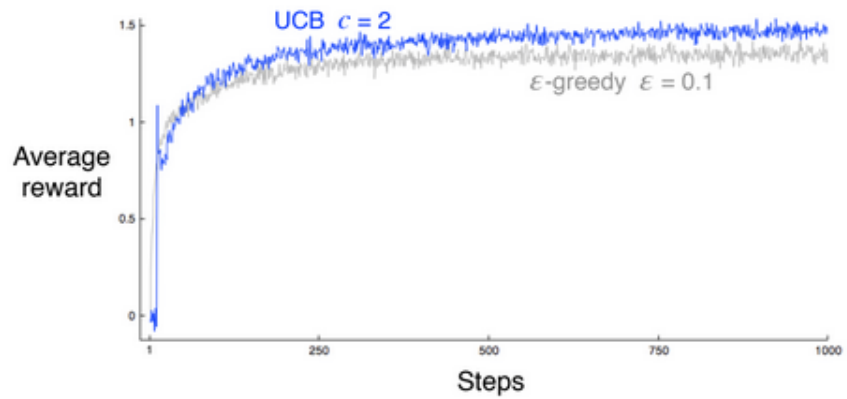
Slide credit K. Fragkiadaki

UCB1: Auer, Cesa-bianchi, and Fischer, Finite-time analysis of the multiarmed bandit problem, 2002

Upper Confidence Bound (UCB)

- A clever way of reducing exploration over time
- Estimate an upper bound on the true action values
- Select the action with the largest (estimated) upper bound

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$



Slide adapted from K. Fragkiadaki

UCB1: Auer, Cesa-bianchi, and Fischer, Finite-time analysis of the multiarmed bandit problem, 2002

The Exploration / Exploitation Dilemma

- Exploitation: Make the best decision given current information
- Exploration: Sacrifice some action budget to gather more information

- Key question: when to explore and when to exploit?

- Idea 1: ε-greedy method, a naïve solution that decide to explore / exploit by throwing a dice with probability ε
- Idea 2: upper confidence bound (UCB) method, the more **uncertain** we are about an action-value, the more important it is to **explore** that action
- Idea 3: gradient bandit method, sample an action based on relative **preference** over other actions

Slide adapted from K. Fragkiadaki

Gradient Bandit Method

- Define $H_t(a_k)$ which characterizes relative preference of action a_k over other actions at time step t

- The probability for sampling an action is based on its preference value:

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a),$$

Gradient Bandit Method

- Define $H_t(a_k)$ which characterizes relative preference of action a_k over other actions at time step t
- The probability for sampling an action is based on its preference value:

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a),$$

- On each step, after selecting action A_t and receiving the reward R_t, the action preferences are updated by

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t,$$

Gradient Bandit Method

- On each step, after selecting action A_t and receiving the reward R_t, the action preferences are updated by

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t,$$

  - $\bar{R}_t$ is called **baseline**, with which an action is compared to
    - a_k is preferred if its reward R_t surpasses the baseline
  - We can define $\bar{R}_t$ as the average of all rewards.

Gradient Bandit Method

- On each step, after selecting action A_t and receiving the reward R_t, the action preferences are updated by

If $R_t > \bar{R}_t$, the preference of a_k should be largely increased if it was not frequently picked

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t,$$

- $\bar{R}_t$ is called **baseline**, with which an action is compared to
  - ➢ a_k is preferred if its reward R_t surpasses the baseline
- We can define $\bar{R}_t$ as the average of all rewards.

Gradient Bandit Method

- On each step, after selecting action A_t and receiving the reward R_t, the action preferences are updated by

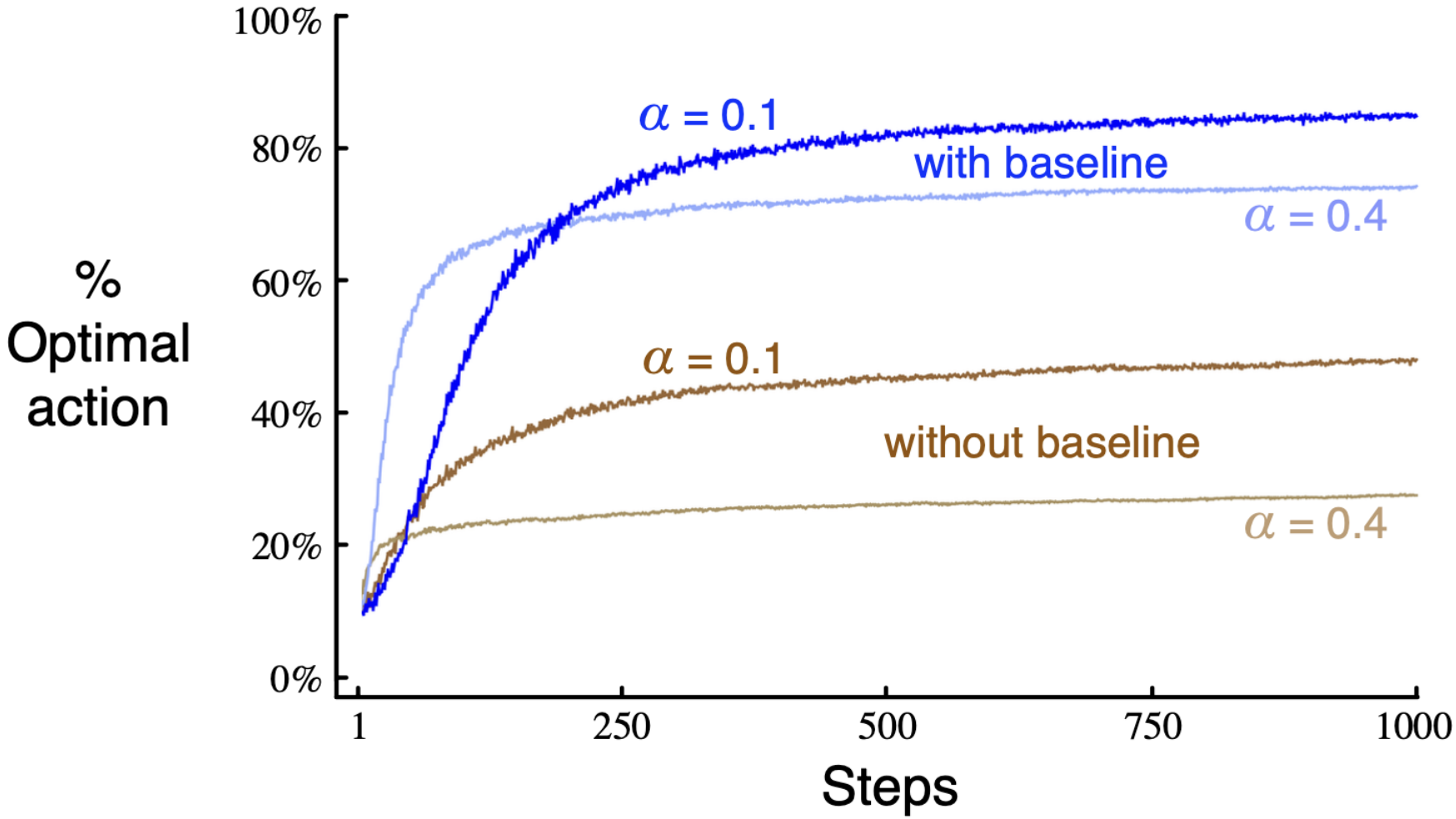
If $R_t > \bar{R}_t$, the preference of a_k should be largely increased if it was not frequently picked

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t,$$

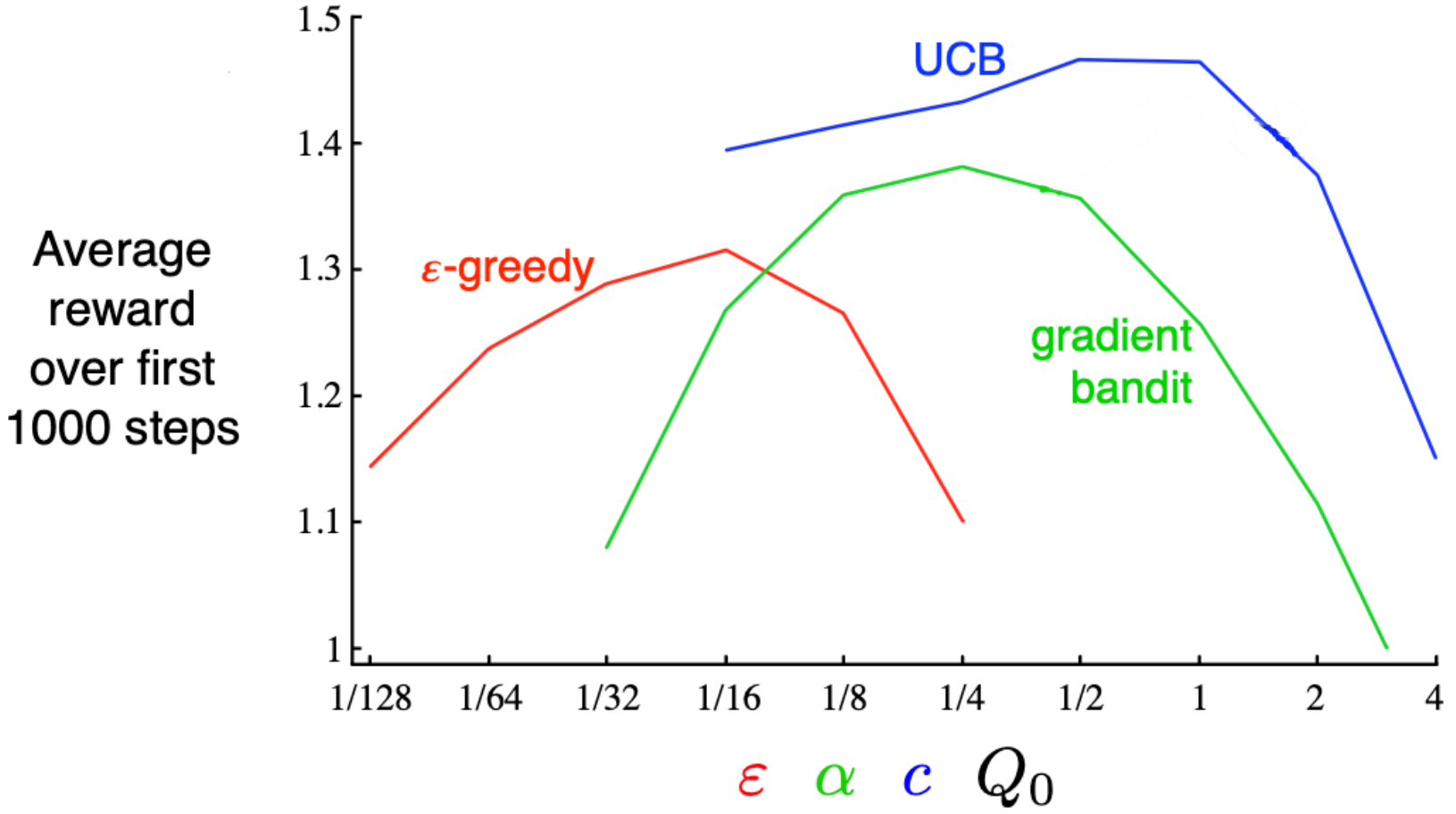
the preference of other actions should be largely decreased if they were frequently picked

  - $\bar{R}_t$ is called **baseline**, with which an action is compared to
    - ➢ a_k is preferred if its reward R_t surpasses the baseline
  - We can define $\bar{R}_t$ as the average of all rewards.

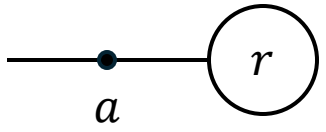
Gradient Bandit Method



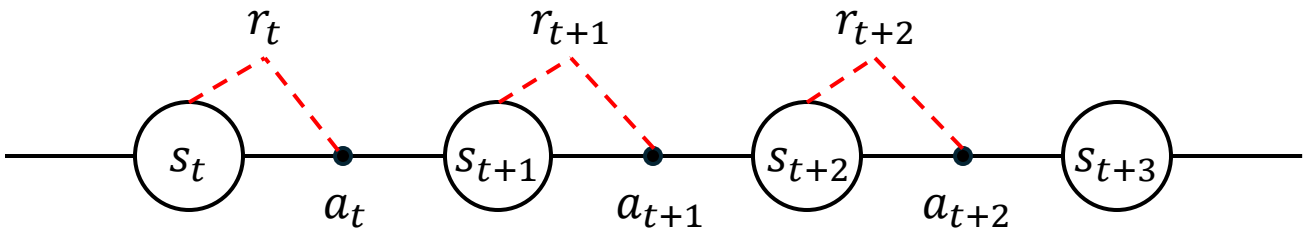
Comparison of Three Methods



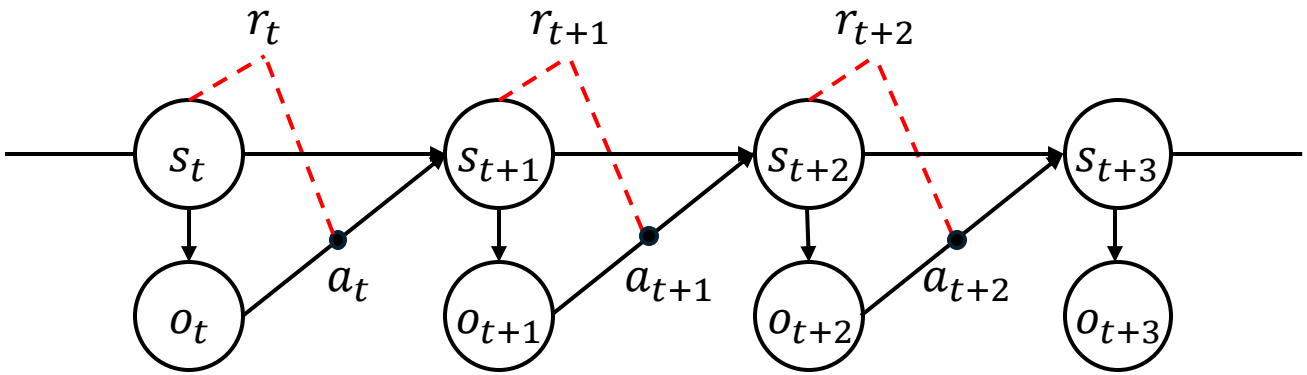
So Far, We've Considered a Non-Sequential Setup



But in most cases, we have "states" and every action has sequential effect!

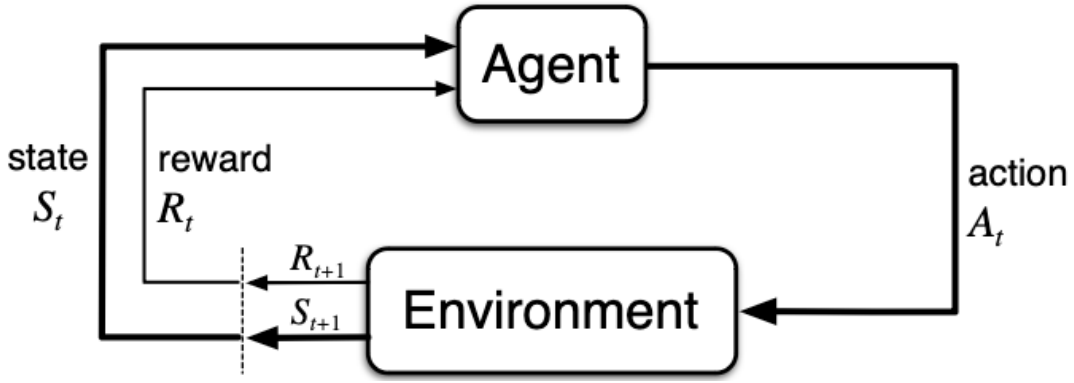


Moreover, we often don't know "states" but have only "observations"…



Slide adapted from K. Fragkiadaki

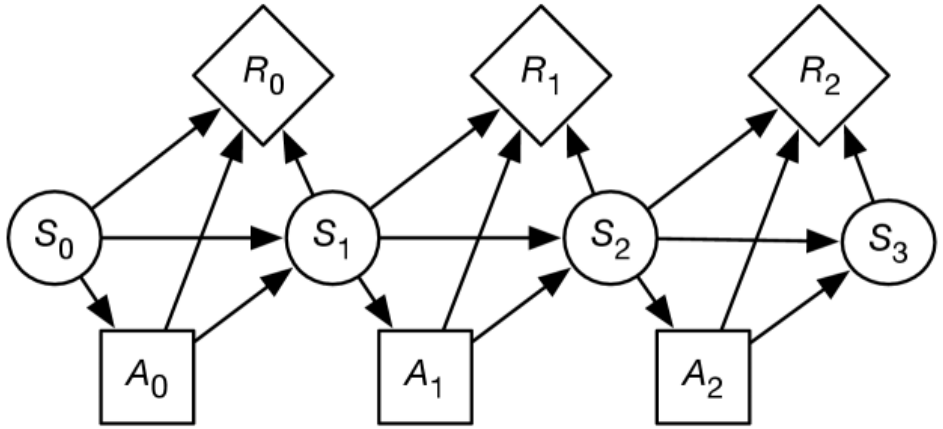
Let's First Define the Following Components



- **Agent**: the decision maker that senses the environment and decide "what actions" to take in the environment
- **Environment**: the world
- **Policy:** a mapping function from states / observations to actions.
- **Reward:** the signal indicates "what" you want a robot to achieve, not "how" you want it to achieve
- **State**: the representations that retain all "essential" information for decision making
- **(World) Model:** the transition function that maps states / observations and actions to future states / observations

MDP vs. POMDP

Markov Decision Process

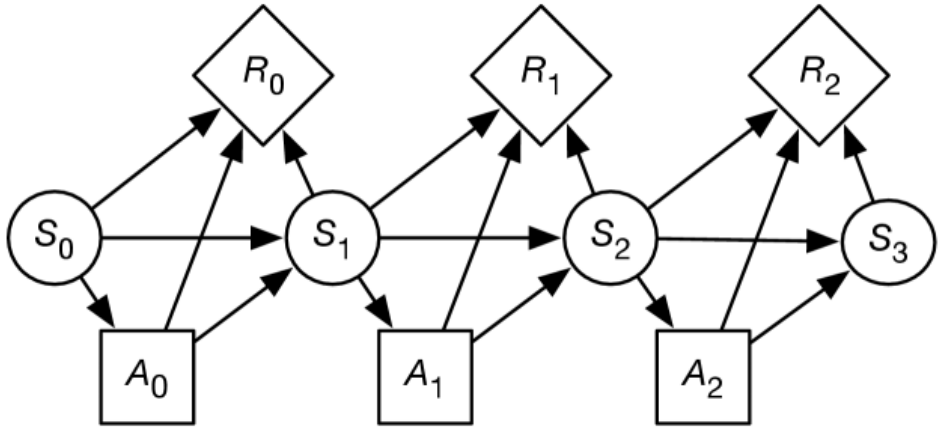


- Markov property:
  - The transition to s_t only depends on the immediately preceding state and action, s_{t-1} and a_{t-1}, not at all on earlier states and actions.
  - The state must include information about all aspects of the past agent–environment interaction that make a difference for the future

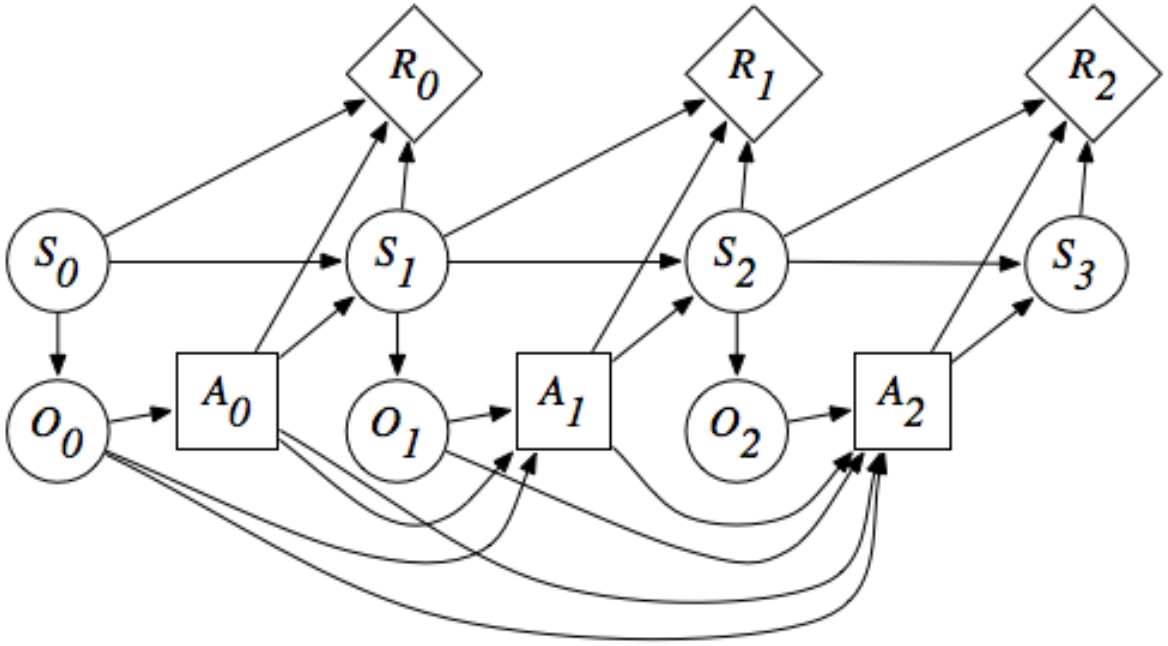
https://artint.info/3e/html/ArtInt3e.Ch12.S5.html

MDP vs. POMDP

Markov Decision Process



Partiall Observable Markov Decision Process

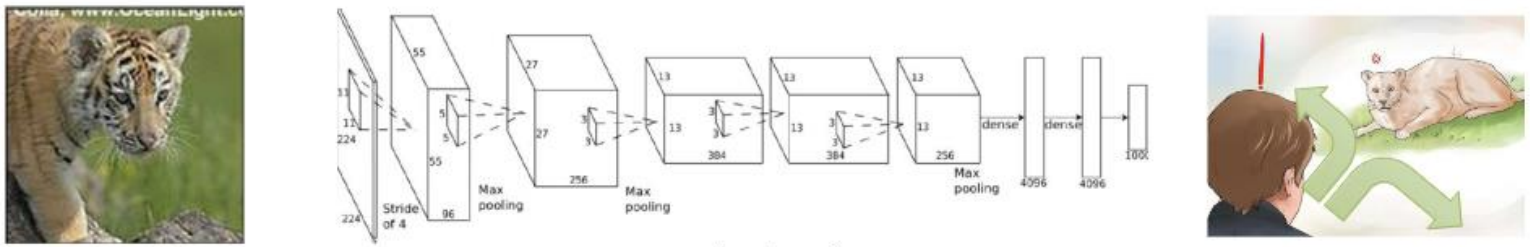


State is a hidden variable. The decision is only conditioned on partial observations

https://artint.info/3e/html/ArtInt3e.Ch12.S5.html

https://www.inf.unibz.it/~franconi/teaching/artint.info/html/ArtInt_230.html

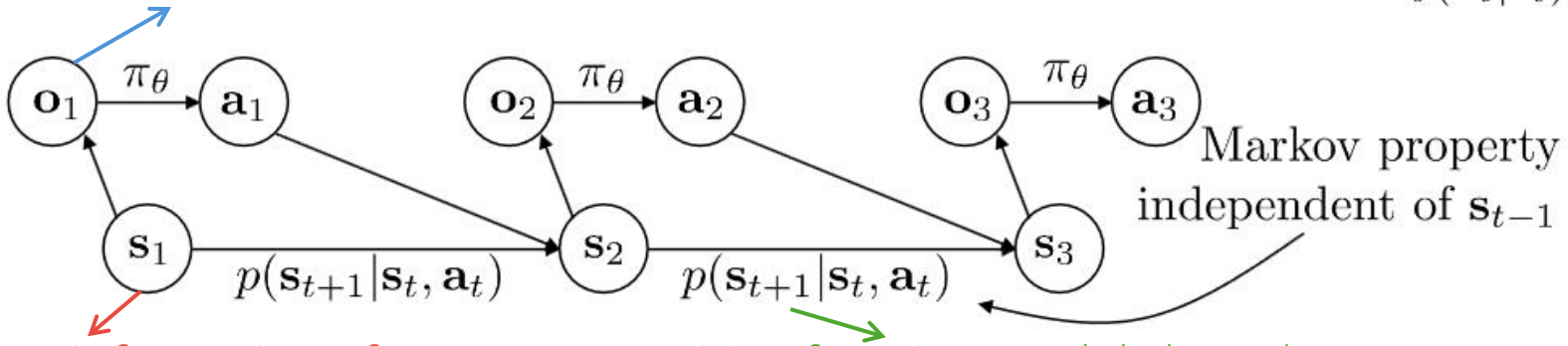
(Partially Observable) Markov Decision Process



$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)

The observed information of the environment



The true information of the environment

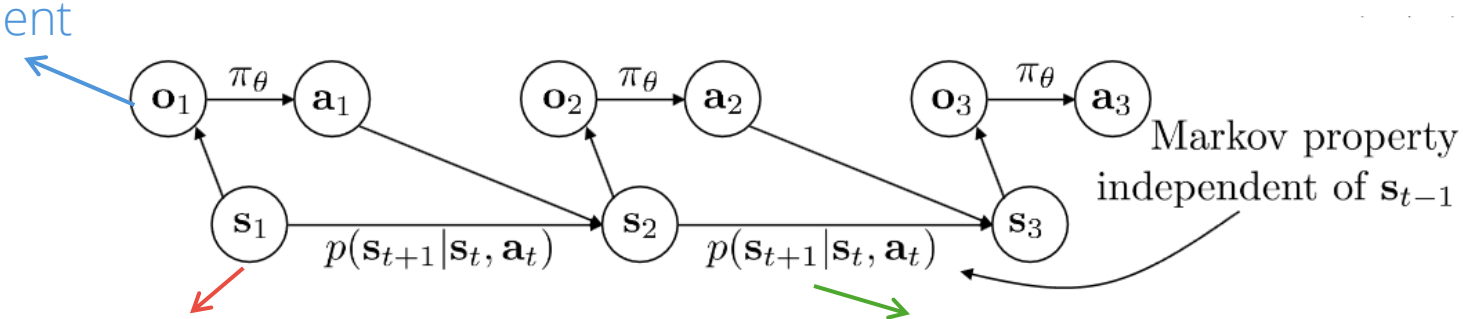
Transition function models how the environment changes with the action

Reward function signals if the goal is achieved:

- $r_t = r(s_{t+1}, s_t, a_t)$
- $r_t = r(s_t, a_t)$

Markov Decision Process

The observed information of the environment



The true information of the environment

Transition function models how the environment changes with the action

Reward function signals if the goal is achieved:

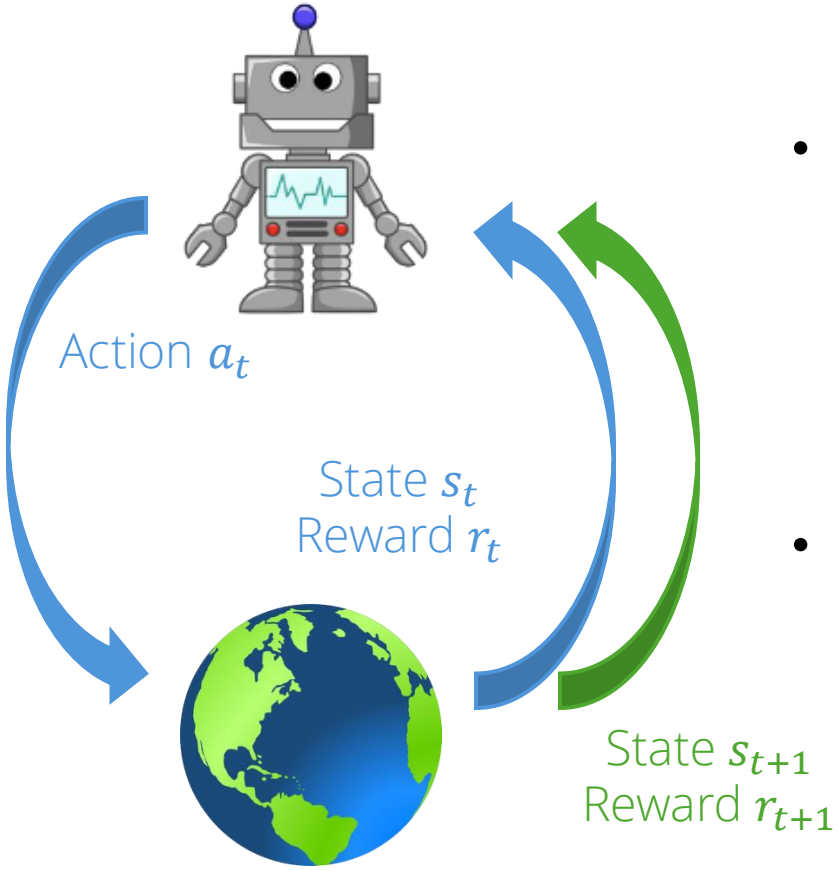
- $r_t = r(s_{t+1}, s_t, a_t)$
- $r_t = r(s_t, a_t)$

Most General Case:
Transition function is **unknown**
Reward function is **unknown**
Observed information is **incomplete** ($o_t \neq s_t$)

Most Specific Case:
Transition function is **known**
Reward function is **known**
Observed information is **complete** ($o_t = s_t$)

Slide adapted from S. Levine

Reinforcement Learning: Learns a Policy that Maximizes the Total Reward of a sequence of actions



- A trajectory of interaction in the environment

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

- Maximize the expected value of the cumulative sum of reward

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

- Policy may be stochastic
- Environment may be stochastic
- Initial state may be randomly sampled
- $E_{\pi_\theta}[r(x)]$ is smooth in θ

Slide adapted from S. Levine

Remember We Talked about Optimal Control

- Constrained Optimal Control Problem:

$$\min_{x,u} \sum_{k=0}^{N-1} c(x_k, u_k) + \mathrm{c_f}\,(x_N)$$

$$s.t\ x_0 = \hat{x}_0$$
$$x_{k+1} = f(x_k, u_k), k = 0, \ldots, N-1$$

- Solution 1: Consider it as a mathematical optimization problem. We can solve it with existing optimization toolboxes (e.g. linear programming, quadratic programming, non-linear programming …)
- Solution 2: Solve it with dynamic programming!

Remember We Talked about Optimal Control

- Constrained Optimal Control Problem:

$$\min_{x,u} \sum_{k=0}^{N-1} c(x_k, u_k) + \mathrm{c_f}\,(x_N)$$

$$s.t\ x_0 = \hat{x}_0$$
$$x_{k+1} = f(x_k, u_k), k = 0, \ldots, N-1$$

The objective of RL:

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

In fact, they are very similar problems with different notations:

- control u ↔ action a
- state x ↔ state s
- cost $c(x, u)$ ↔reward $r(s, a)$
- minimize cost ↔ maximize reward

- Solution 1: Consider it as a mathematical optimization problem. We can solve it with existing optimization toolboxes (e.g. linear programming, quadratic programming, non-linear programming …)
- Solution 2: Solve it with dynamic programming!

Returns G_t in Episodic Tasks

- Episode: A sequence of interactions based on which the reward will be judged at the end
- Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze
- Returns G_t: the cumulative sum of reward, or total reward, starting from time t
- Let $\mathbf{T}$ be the final time step:

$$G_t = R_{t+1} + R_{t+2} + \ ... + R_T$$

Slide adapted from K. Fragkiadaki

Returns G_t in Continuing Tasks

- Continuing tasks: interaction does not have natural episodes, but just goes on and on…just like real life
- Discounted returns G_t: the cumulative sum of reward, weighted by the discount rate γ
- Let $\mathbf{T}$ be the final time step, where $T \rightarrow \infty$:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- Discount rate γ decides the present value of future rewards.
  - If $\gamma = \mathbf{0}$, model is "myopic" concerning only the immediate reward
  - If $\gamma = \mathbf{1}$, model is "farsighted" ignoring the efficiency of achieving the goal

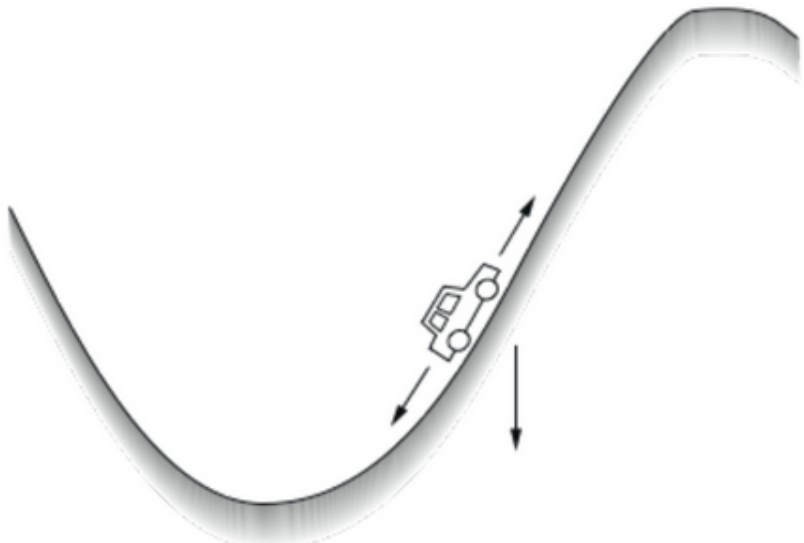
Slide adapted from K. Fragkiadaki

Why discount?

Most Markov reward and decision processes are discounted. Why?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behaviour shows preference for immediate reward
- It is sometimes possible to use *undiscounted* Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate.

Mountain Car



Get to the top of the hill
as quickly as possible.

reward = −1 for each step where **not** at top of hill

$\Rightarrow$ return = − number of steps before reaching top of hill

Return is maximized by minimizing
number of steps to reach the top of the hill.

Slide credit K. Fragkiadaki

Why Discount

- Recursive relationships for discounted returns:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

$$= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots)$$

$$= R_{t+1} + \gamma G_{t+1}$$

Value Functions are Expected Returns

- The state value function $v_\pi(s)$: the expected return starting from state s following policy π

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

- The state value function is dependent on the deployed policy, since different actions lead to varied future states and total rewards.

The Recursive Relationships of Value Functions

- **The state value function** $v_\pi(s)$: the expected return starting from state s following policy π

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = R_{t+1} + \gamma G_{t+1}$$

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s]$$

- The state value function is dependent on the deployed policy, since different actions lead to varied future states and total rewards.

The Recursive Relationships of Value Functions

- **The state value function** $v_\pi(s)$: the expected return starting from state s following policy π

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$$
$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']]$$

- The state value function is dependent on the deployed policy, since different actions lead to varied future states and total rewards.

- The state value function $v_\pi(s)$ estimates the "goodness" of state s when deploying policy π. Simply speaking, how easy it is to achieve the goal from state s.

The Recursive Relationships of Value Functions

- **The state value function** $v_\pi(s)$: the expected return starting from state s following policy π

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

- The state value function is dependent on the deployed policy, since different actions lead to varied future states and total rewards.

Value Functions are Expected Returns

- The action value function $q_\pi(s,a)$: the expected return starting from state s and action a following policy π

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

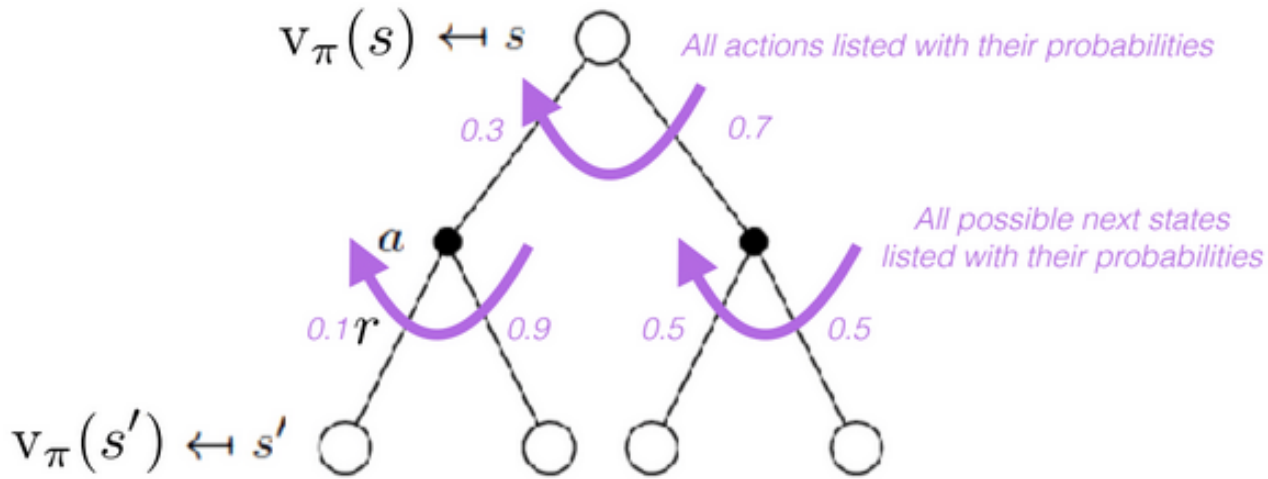
- The action-value function $q_\pi(s,a)$ estimates the "goodness" of action a at state s when deploying policy π. Simply speaking, how easy it is to achieve the goal with action a at state s.

- It's obvious to see

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')] = \sum_a \pi(a|s) q_\pi(s,a)$$

Back-up diagram for value functions

The probabilities of landing on each of the leaves sum to 1

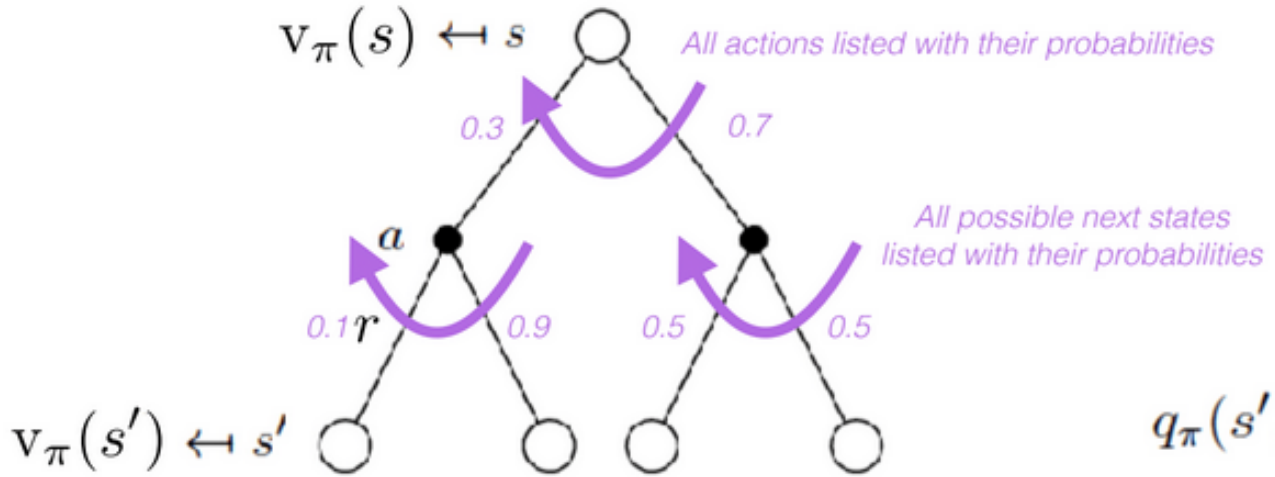


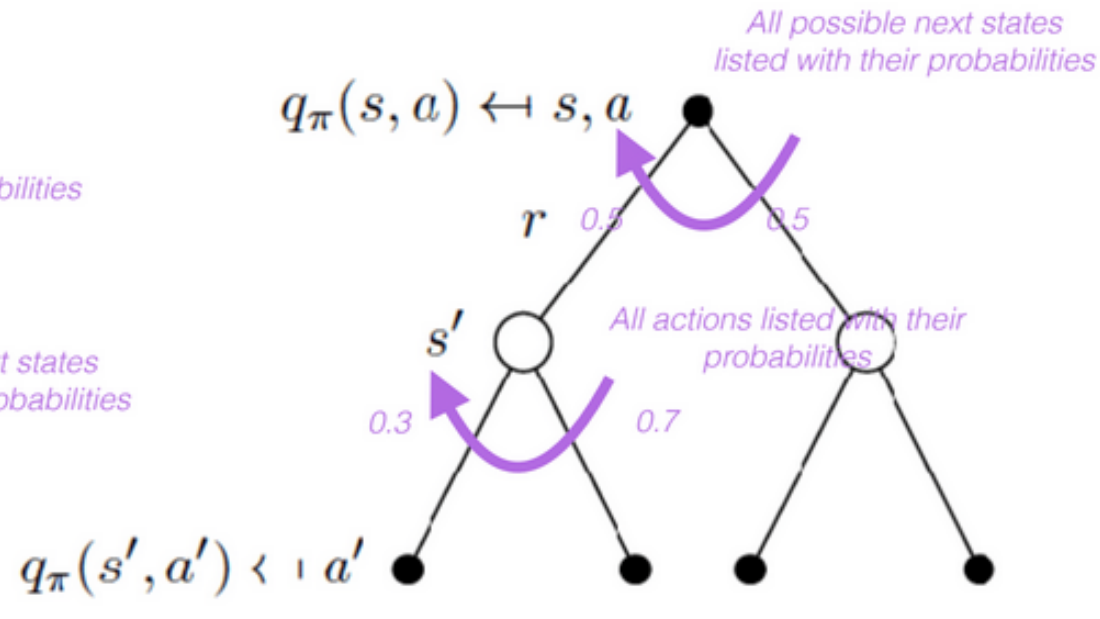
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p\left(s', r | s, a\right) \left[r + \gamma v_\pi(s')\right]$$

Slide credit K. Fragkiadaki

Back-up diagram for value functions

The probabilities of landing on each of the leaves sum to 1

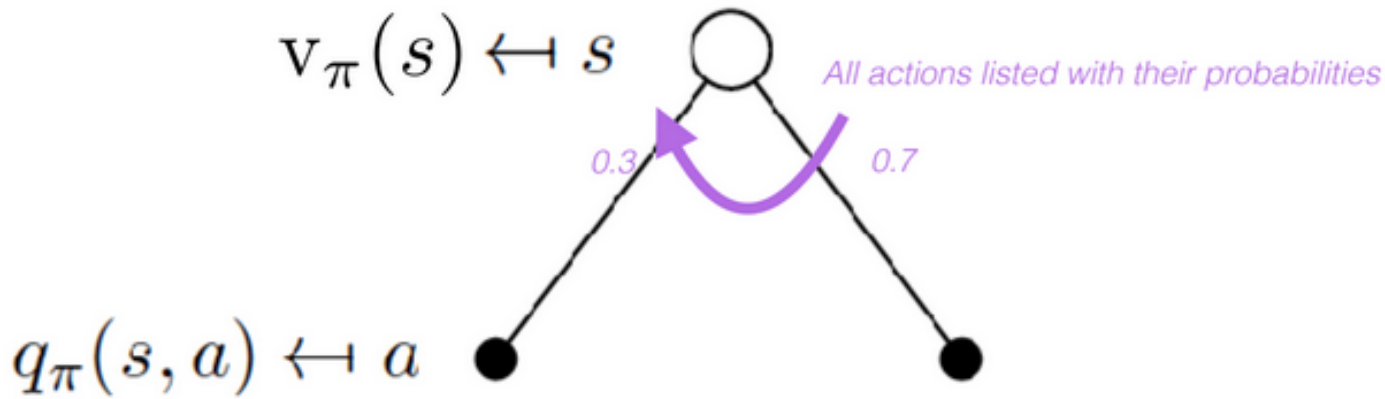




$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p\left(s', r | s, a\right)\left[r + \gamma v_\pi(s')\right]$$

$$q_\pi(s,a) = \sum_{r,s'} p(s', r | s, a)\left(r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right)$$

Relating state and state/action value functions



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) q_\pi(s,a)$$

Optimal Value Functions

- **Definition**: The optimal state-value function $v^*(s)$ is the maximum state-value function over all policies. In other words, the optimal value function specifies the best possible performance in the MDP.

$$v^*(s) = \max_\pi v_\pi(s)$$

- **Definition**: The optimal action-value function $q^*(s, a)$ is the maximum action-value function over all policies

$$q^*(s, a) = \max_\pi q_\pi(s, a)$$

Optimal Value Functions

- **Definition**: The optimal state-value function $v^*(s)$ is the maximum state-value function over all policies. In other words, the optimal value function specifies the best possible performance in the MDP.

$$v^*(s) = \max_\pi v_\pi(s)$$

- **Definition**: The optimal action-value function $q^*(s, a)$ is the maximum action-value function over all policies

$$q^*(s, a) = \max_\pi q_\pi(s, a)$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value function

Value Functions

- Value functions measure the goodness of a particular state or state/action pair: how good is for the agent to be in a particular state or execute a particular action at a particular state, for a given policy.

- Optimal value functions measure the best possible goodness of states or state/action pairs under all possible policies.

	state values	action values
prediction	v_π	q_π
control	v_*	q_*

Slide credit K. Fragkiadaki

Why Value Functions are useful

Value functions capture the knowledge of the agent regarding how good is each state for the goal she is trying to achieve.

"…knowledge is represented as a large number of approximate value functions learned in parallel…"

Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction, *Sutton et al.*

Slide credit K. Fragkiadaki

We Have the Optimal Policy if We Know $q^*(s, a)$

- An optimal policy can be found by maximizing over $q^*(s, a)$:

$$\pi^*(a|s) = \begin{cases} 1, if\ a = \underset{a}{\operatorname{argmax}}\, q^*(s, a). \\ 0, otherwise. \end{cases}$$

We Have the Optimal Policy if We Know $q^*(s, a)$

- An optimal policy can be found by maximising over $q^*(s, a)$:

$$\pi^*(a|s) = \begin{cases} 1, if\ a = \underset{a}{\text{argmax}}\, q^*(s, a). \\ 0, otherwise. \end{cases}$$

- An optimal policy can be found by maximizing over $v^*(s)$ with one-step look ahead:

$$\pi^*(a|s) = \begin{cases} 1, if\ a = \underset{a}{\text{argmax}}(\Sigma_{s',r} p(s', r|s, a))(r + \gamma v_*(s')) \\ 0, otherwise. \end{cases}$$

Bellman Optimality Equation for $\boldsymbol{v}^*$

1. We have $v^*(s) = \max_a q_{\pi^*}(s, a)$

Bellman Optimality Equation for v^*

1. We have $v^*(s) = \max_a q_{\pi^*}(s,a)$

Why?

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

$$\Rightarrow v^*(s) = \sum_{a \in \mathcal{A}} \pi^*(a|s) q_{\pi^*}(s,a) = \max_a q_{\pi^*}(s,a)$$

$$\pi^*(a|s) = \begin{cases} 1, if\ a = \operatorname{argmax}_a q^*(s,a). \\ 0, otherwise. \end{cases}$$

Bellman Optimality Equation for v^*

1. We have $v^*(s) = \max_a q_{\pi^*}(s, a)$

2. We can further write the equation:

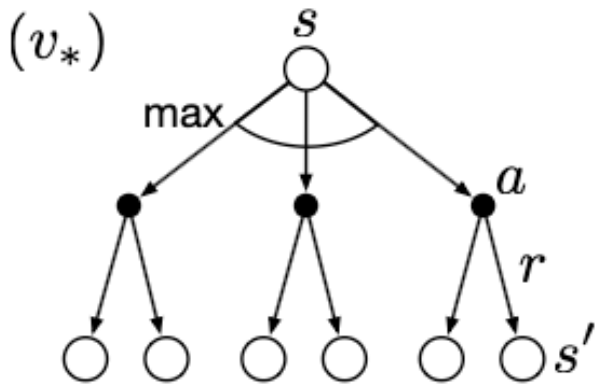
$$
\begin{aligned}
v^*(s) &= \max_a q_{\pi^*}(s, a) \\
&= \max_a \mathbb{E}_{\pi^*}[G_t | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a] \\
&= \max_a \sum_{s', r} p(s', r | s, a)[r + \gamma v^*(s')]
\end{aligned}
$$

Bellman Optimality Equation for v^*

1. We have $v^*(s) = \max_a q_{\pi^*}(s,a)$

2. We can further write the equation:

$$
\begin{aligned}
v^*(s) &= \max_a q_{\pi^*}(s,a) \\
&= \max_a \mathbb{E}_{\pi^*}[G_t | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v^*(s')]
\end{aligned}
$$



This is again the recursive relationship!

Bellman Optimality Equation for q^*

1. We have

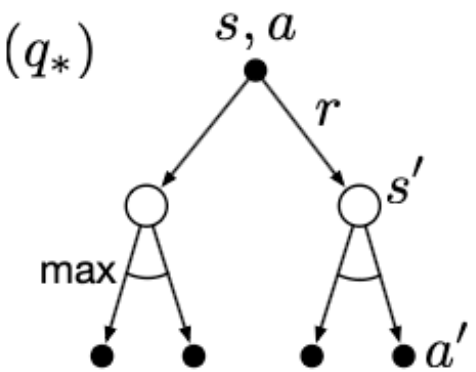
$$
\begin{aligned}
q^*(s,a) &= \mathbb{E}_{\pi^*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\
&= \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a] \\
&= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_{\pi^*}(S_{t+1}, a') \,\middle|\, S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s', r | s, a)[r + \gamma \max_{a'} q^*(s', a')]
\end{aligned}
$$

Bellman Optimality Equation for q^*

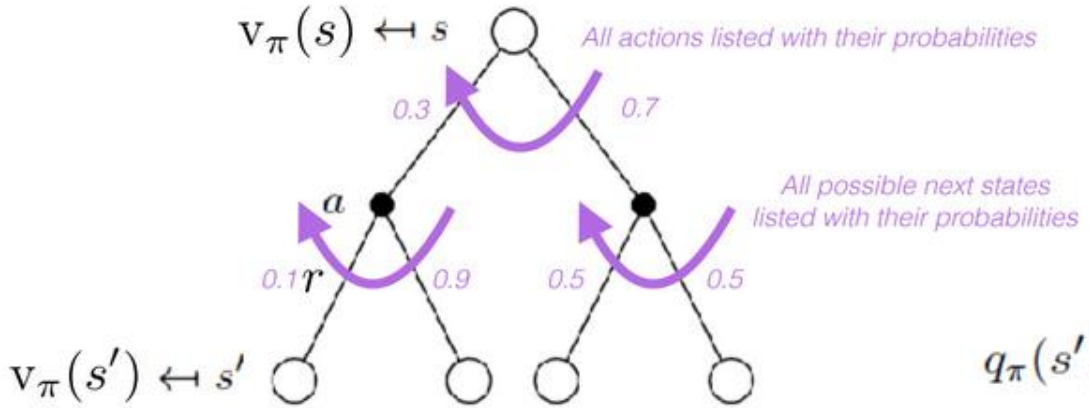
1. We have

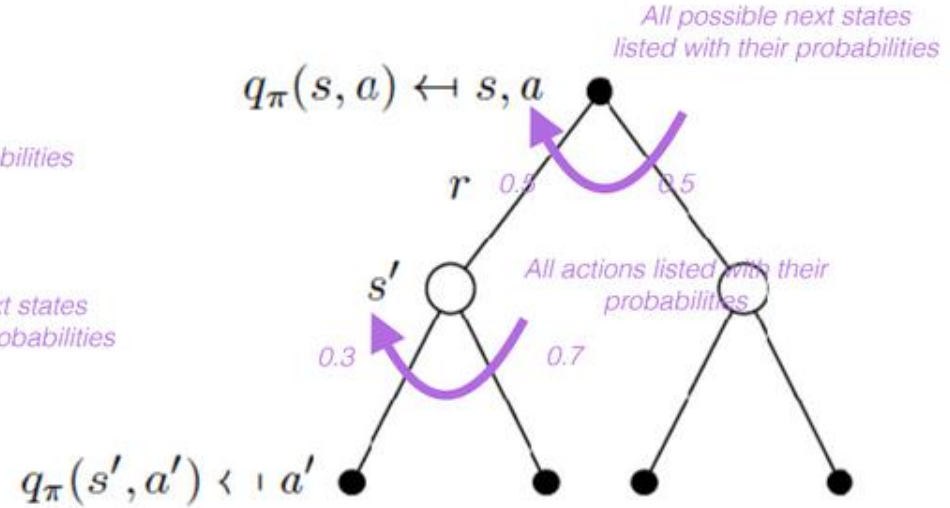
$$
\begin{aligned}
q^*(s,a) &= \mathbb{E}_{\pi^*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\
&= \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a] \\
&= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_{\pi^*}(S_{t+1}, a') \,\middle|\, S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s', r | s, a)[r + \gamma \max_{a'} q^*(s', a')]
\end{aligned}
$$

This is again the recursive relationship!



Back-up Diagrams for Value Functions



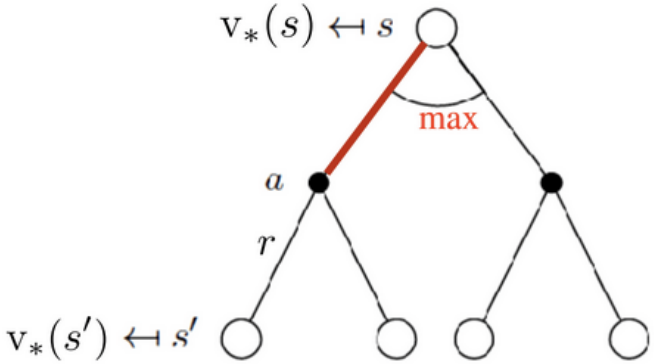


$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p\left(s', r | s, a\right)\left[r + \gamma v_\pi(s')\right]$$

$$q_\pi(s,a) = \sum_{r,s'} p(s', r | s, a)\left(r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')\right)$$

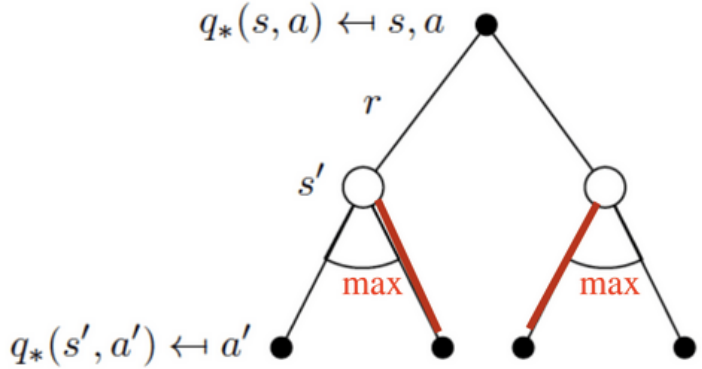
Slide credit K. Fragkiadaki

Back-up Diagrams for Optimal Value Functions



For the Bellman expectation equations we sum over all the leaves, here we choose only the best action branch!

$$v_*(s) = \max_{a \in \mathcal{A}} \left(\sum_{s',r} p(s', r \mid s, a)(r + \gamma v_*(s')) \right)$$

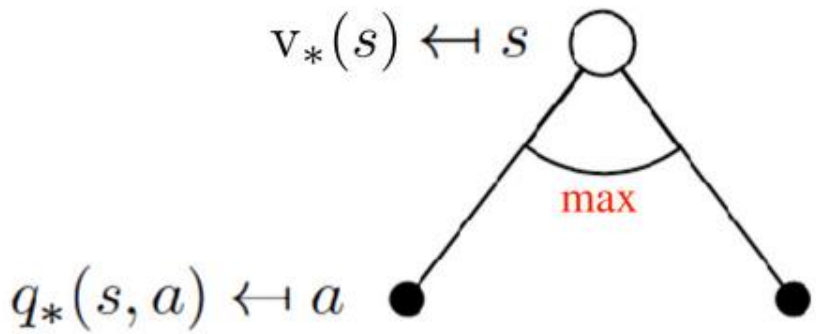


$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a' \in \mathcal{A}} q_*(S_{t+1}, a') \mid S_t = s, A_t = a]$$

$$= \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right]$$

q* is the unique solution of this system of nonlinear equations

Relating Optimal State and Action Value Functions



$$v_*(s') = \max_a q_*(s,a)$$

Slide credit K. Fragkiadaki

How to Calculate Value Functions?

1. Matrix-form solution: solving linear systems of equations

Matrix-Form Solution of Value Functions

- The state value function $v_\pi(s)$:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

- When the policy π is fixed, MDP becomes Markov Reward Process (MRP)

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) \sum_{s'} T(s'|s, a)[r(s, a) + \gamma v_\pi(s')] \\ &= \sum_a \pi(a|s) r(s, a) + \gamma \sum_a \pi(a|s) \sum_{s'} T(s'|s, a) v_\pi(s') \\ &= \mathbb{r}_s^\pi + \gamma \sum_{s'} \mathbb{T}_{s',s}^\pi v_\pi(s') \end{aligned}$$

Matrix Form

The Bellman expectation equation can be written concisely as a system of linear equations

$$v_\pi = r^\pi + \gamma T^\pi v_\pi$$

with direct solution

$$v_\pi = \left(I - \gamma T^\pi\right)^{-1} r^\pi$$

of complexity $\mathcal{O}(|S|^3)$

here T^π is an |S|x|S| matrix, whose (j,k) entry gives $P(s_k \mid s_j, a=\pi(s_j))$

r^π is an |S|-dim vector whose j^{th} entry gives $E[r \mid s_j, a=\pi(s_j)]$

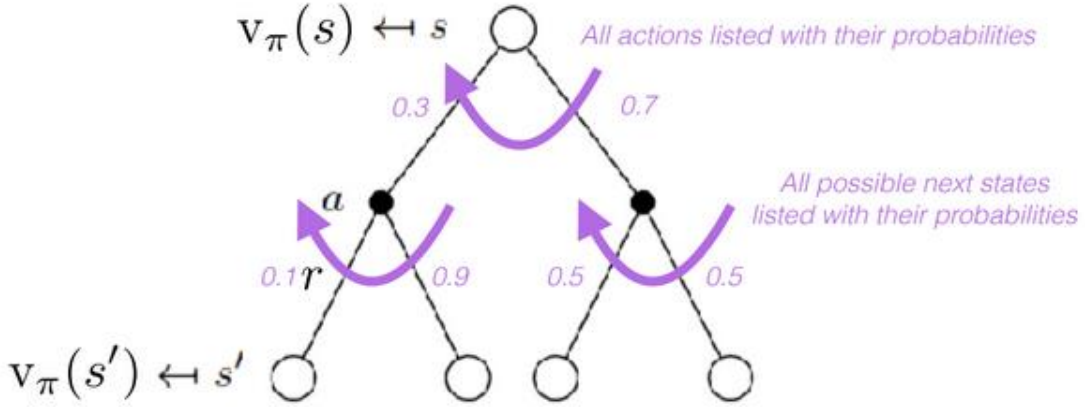
v_π is an |S|-dim vector whose j^{th} entry gives $V_\pi(s_j)$

where |S| is the number of distinct states

Slide credit K. Fragkiadaki

How to Calculate Value Functions?

1. Matrix-form solution: solve linear systems of equations
2. Iterative estimation: utilize the recursive relationships



$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p\left(s', r | s, a\right) \left[r + \gamma v_\pi(s')\right]$$

Iterative Policy Evaluation

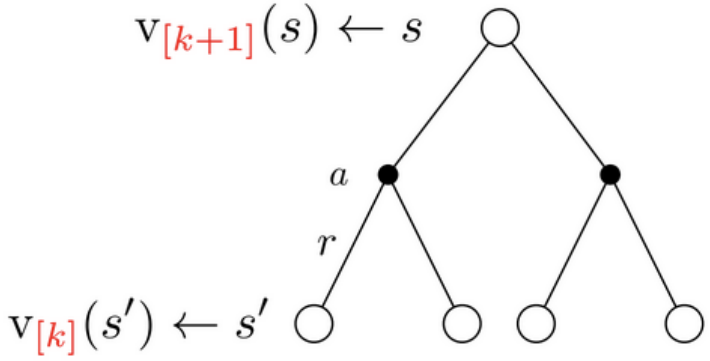
- The state value function $v_\pi(s)$:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

- We can utilize the the recursive relationship to update $v_\pi(s)$

$$for\ k = 1 \cdots \infty :$$

$$v_{k+1}(s) \coloneqq \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$



Iterative Policy Evaluation

- We can utilize the the recursive relationship to update $v_\pi(s)$

$$for\ k = 1 \cdots \infty :$$

$$v_{k+1}(s) \coloneqq \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

- Dynamic programming solution: use a table to keep track of the value function

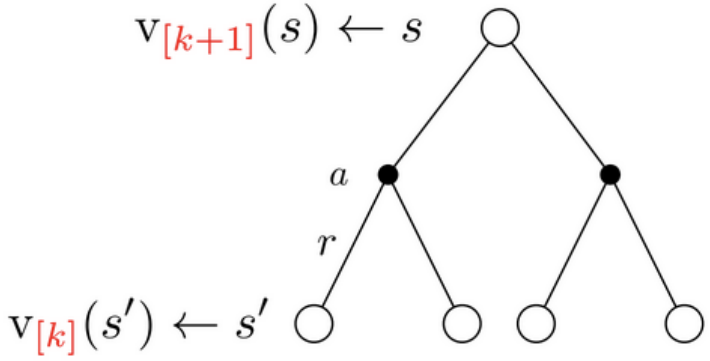
v_k

0.64	0.81	0.9	1
0.58		0.81	0.9
0.52	0		0.81
0	0		0.64

update

v_{k+1}

0.64	0.81	0.9	1
0.58		0.81	0.9
0.52	0.47		0.81
0.47	0		0.64



Estimate Value Function with Dynamic Programming

- Current policy: going up, down, left, right with even probability. The action that would take the agent off the grid would leave the state unchanged.
- Task: Travel from the start to the goal location
- Reward: reach goal = 1, otherwise 0

Reward map

0	0	0	1
0		0	0
0	0		0
0	0		0

v_0

0	0	0	0
0		0	0
0	0		0
0	0		0

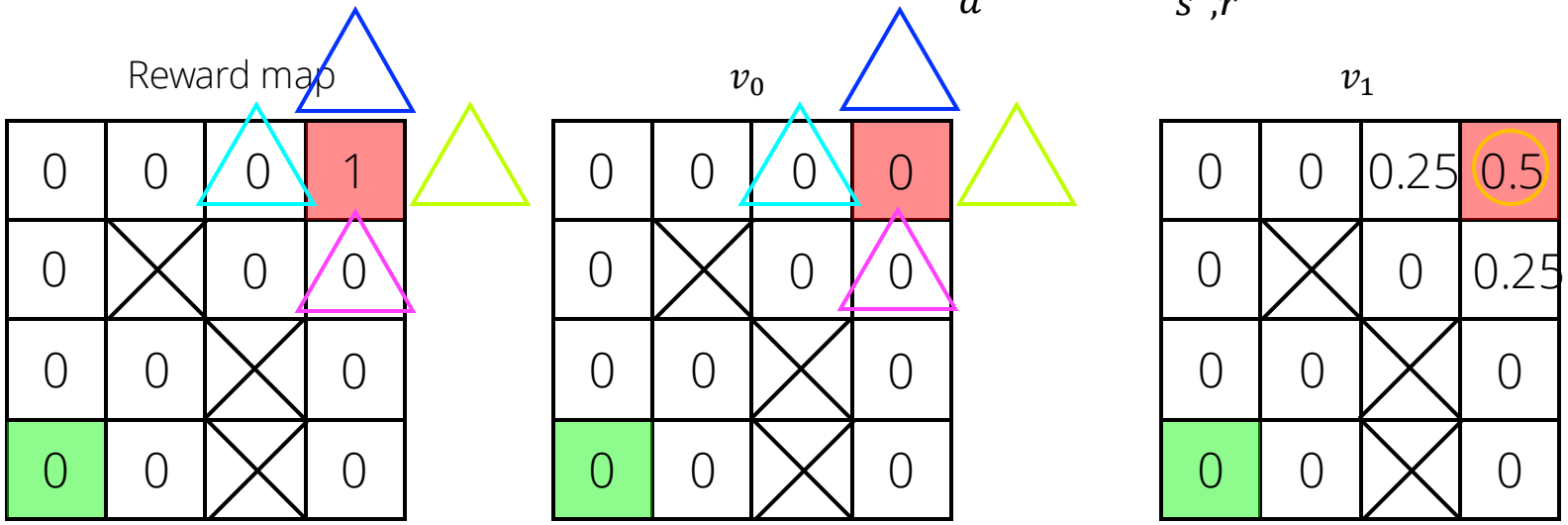
v_1

0	0	0.25	0.5
0		0	0.25
0	0		0
0	0		0

Estimate Value Function with Dynamic Programming

- Backward dynamic programming, let $\gamma = 1.0$ (no discount)

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

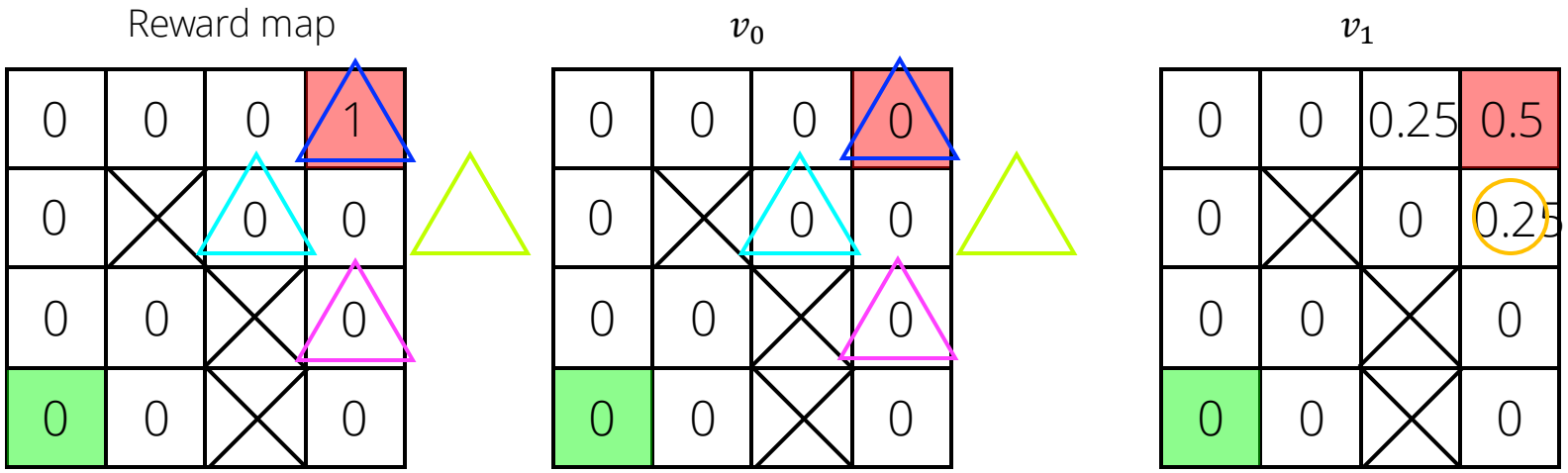


$$0.5 = \frac{1}{4} \times (1 + 1 \times 0) + \frac{1}{4} \times (1 + 1 \times 0) + \frac{1}{4} \times (0 + 1 \times 0) + \frac{1}{4} \times (0 + 1 \times 0)$$

Estimate Value Function with Dynamic Programming

- Backward dynamic programming, let $\gamma = 1.0$ (no discount)

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

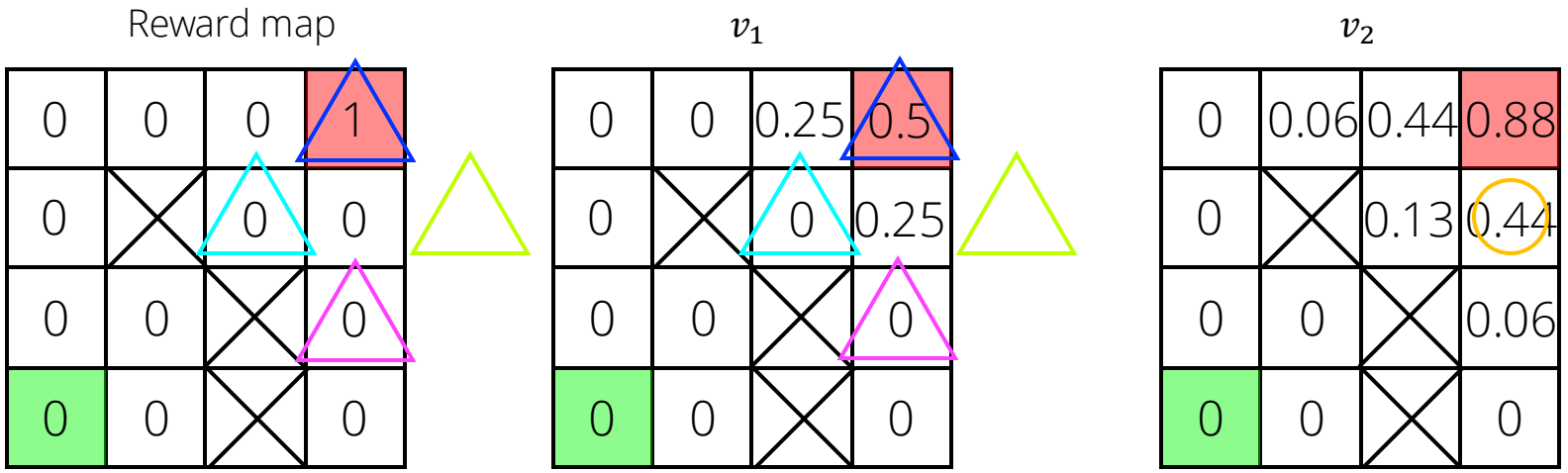


$$0.25 = \frac{1}{4} \times (1 + 1 \times 0) + \frac{1}{4} \times (0 + 1 \times 0) + \frac{1}{4} \times (0 + 1 \times 0) + \frac{1}{4} \times (0 + 1 \times 0)$$

Estimate Value Function with Dynamic Programming

- Backward dynamic programming, let $\gamma = 1.0$ (no discount)

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$



$$0.44 = \frac{1}{4} \times (1 + 1 \times 0.5) + \frac{1}{4} \times (0 + 1 \times 0.25) + \frac{1}{4} \times (0 + 1 \times 0) + \frac{1}{4} \times (0 + 1 \times 0)$$

Iterative Policy Evaluation

$$v_{k+1}(s) \coloneqq \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Can We Improve the Current Policy?

- Let's say we obtain the value function $v_\pi(s)$ based on policy π using dynamic programming, How can we improve the policy?

- Switch to a greedy policy!

$$\pi'(a|s) = \begin{cases} 1, if\ a = \underset{a}{\text{argmax}}(\Sigma_{s',r} p(s', r|s, a))(r + \gamma v_\pi(s')) \\ 0, otherwise. \end{cases}$$

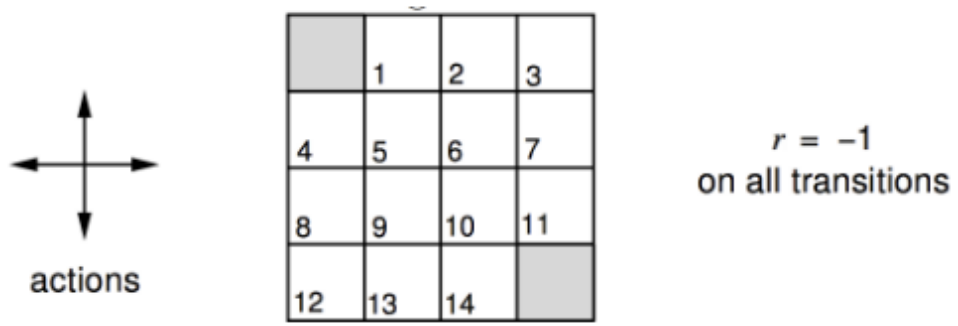
- Why greedy policy π' is better than the original policy π at state s?
  Since a greedy policy is deterministic: $\pi'(s) = \underset{a}{\text{argmax}}(\Sigma_{s',r} p(s', r|s, a))(r + \gamma v_\pi(s'))$

$$\boxed{q_\pi(s|\pi'(s))} = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

The value of selecting action $\pi'(s)$ is higher than following policy π at state s (here we still follow policy π at other states)

$$\geq \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] = \boxed{v_\pi(s)}$$

Evaluating a Random Policy in the Small Gridworld

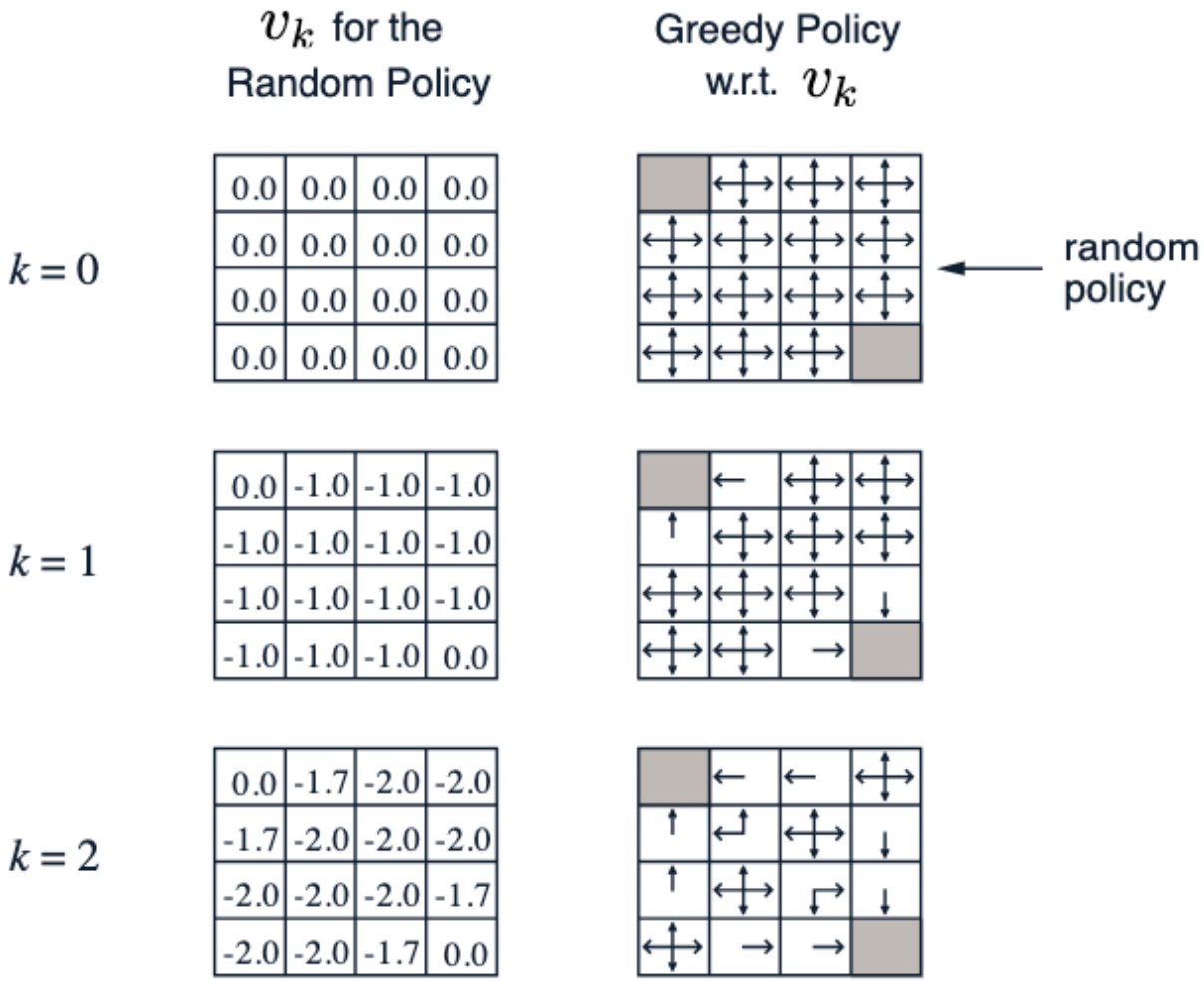


- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, ..., 14$
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

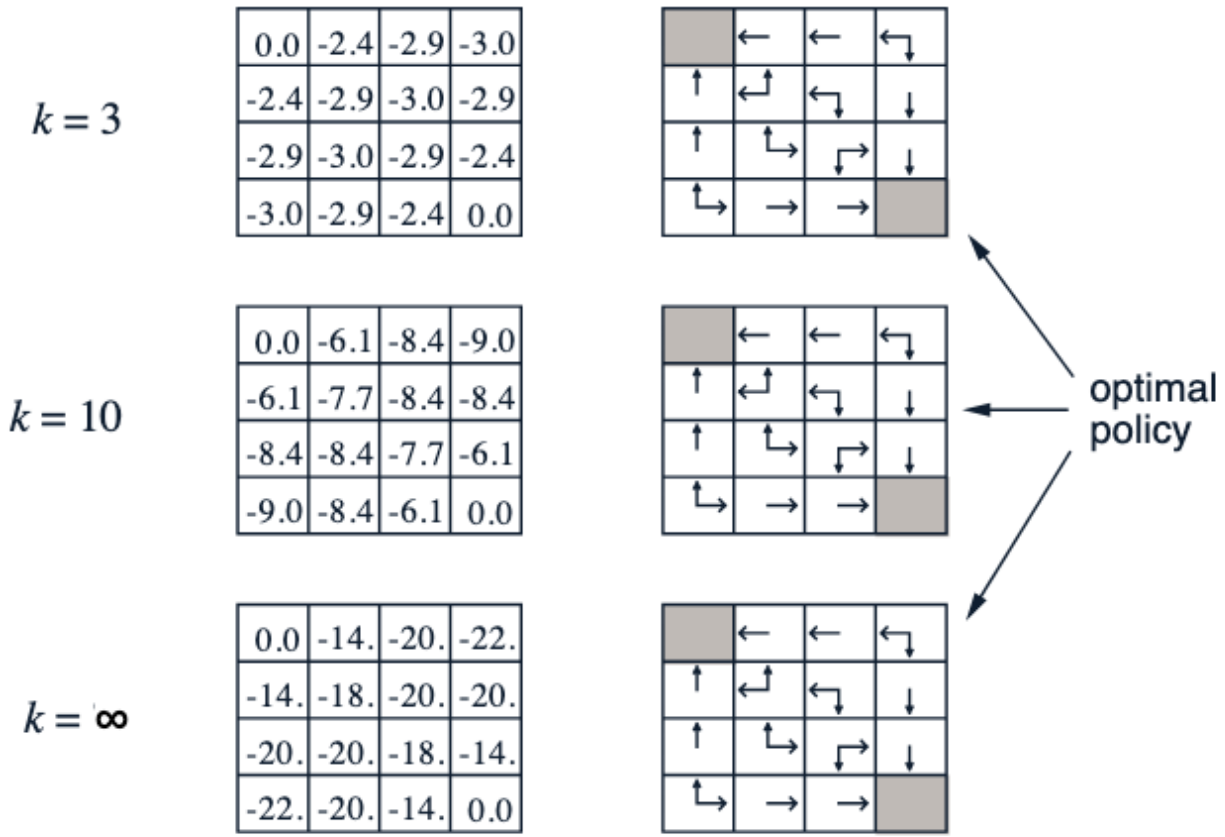
Slide credit D. Silver

Iterative Policy Evaluation in Small Gridworld



Slide credit D. Silver

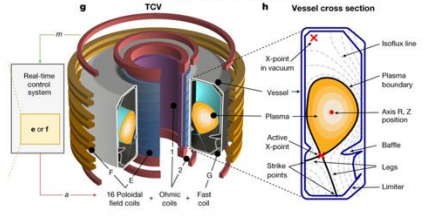
Iterative Policy Evaluation in Small Gridworld (2)



Summary

RL as a general learning framework for different tasks

Power Plant Control



Magnetic control of tokamak plasmas through deep reinforcement learning. Degrave et al.

Trading

Gaming

Adaptive VisuoMotor Control

AnyCar to Anywhere: Learning Universal Dynamics Model for Agile and Adaptive Mobility. Xiao et al.

Content Generation

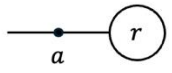
Training Animals

The learning objective of RL

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Multi-armed Bandit Problem

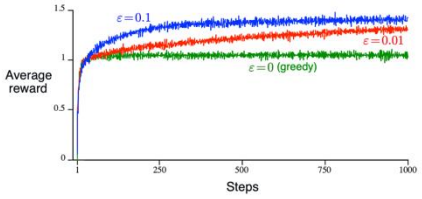


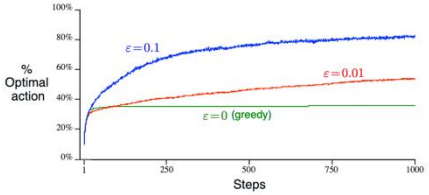
- Expected reward: $q^*(a_k) = \mathbb{E}[r_t | A_t = a_k]$
- Action-value estimates: $Q_t(a_k)$
- Greedy action selection method: select the action with the highest estimated value:

$$A_t{}^* = \arg\max_a Q_t(a)$$

➢ If $A_t = A_t{}^*$, you are *exploiting* your current knowledge of the values of the actions

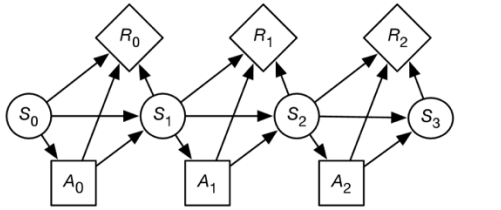
➢ If $A_t \neq A_t{}^*$, you are *exploring*. You improve your estimate of the non-greedy actions

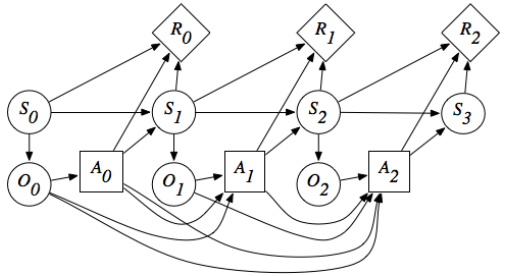


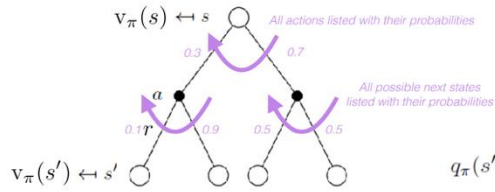


Markov Decision Process

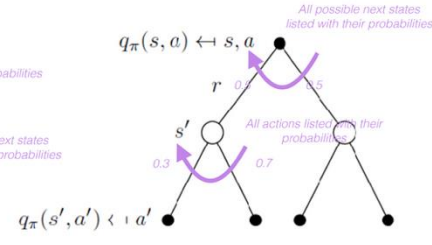
- Discounted returns: $G_t = R_{t+1} + \gamma G_{t+1}$
- The state value function $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$







$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma v_\pi(s')]$$



$$q_\pi(s, a) = \sum_{r,s'} p(s', r|s, a) \left(r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right)$$