Karlsruher Institut für Technologie

Machine Learning 1 – Fundamentals

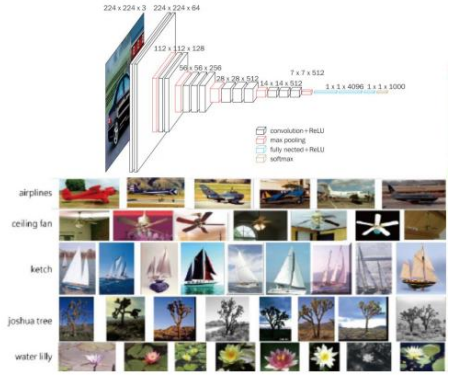
Reinforcement Learning 1

Prof. Dr. J. M. Zöllner, M.Sc. Philipp Stegmaier, M.Sc. Karam Daaboul

Outline

- Motivation and Examples
- Markov Decision Process (MDP)
- Bellman Principle
- Dynamic Programming
  - Policy Iteration
  - Value Iteration
- Summary

Supervised Learning

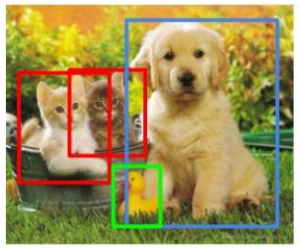


Object classification

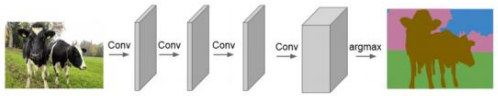
Object localization

Supervised Learning

Data with labels y is given,
Goal: Function approximation
$x \rightarrow y$

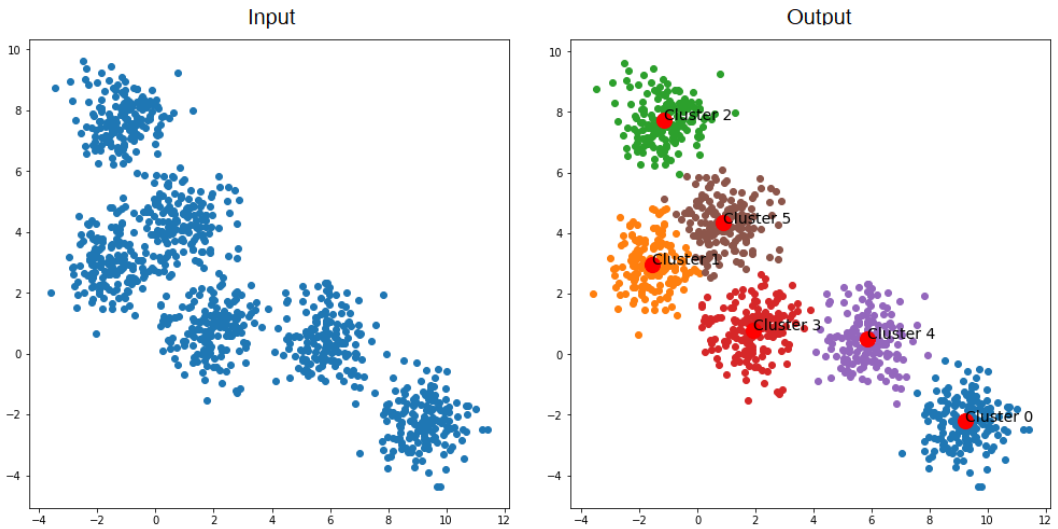


Object detection



Semantic segmentation a.k.a. scene understanding

Unsupervised Learning



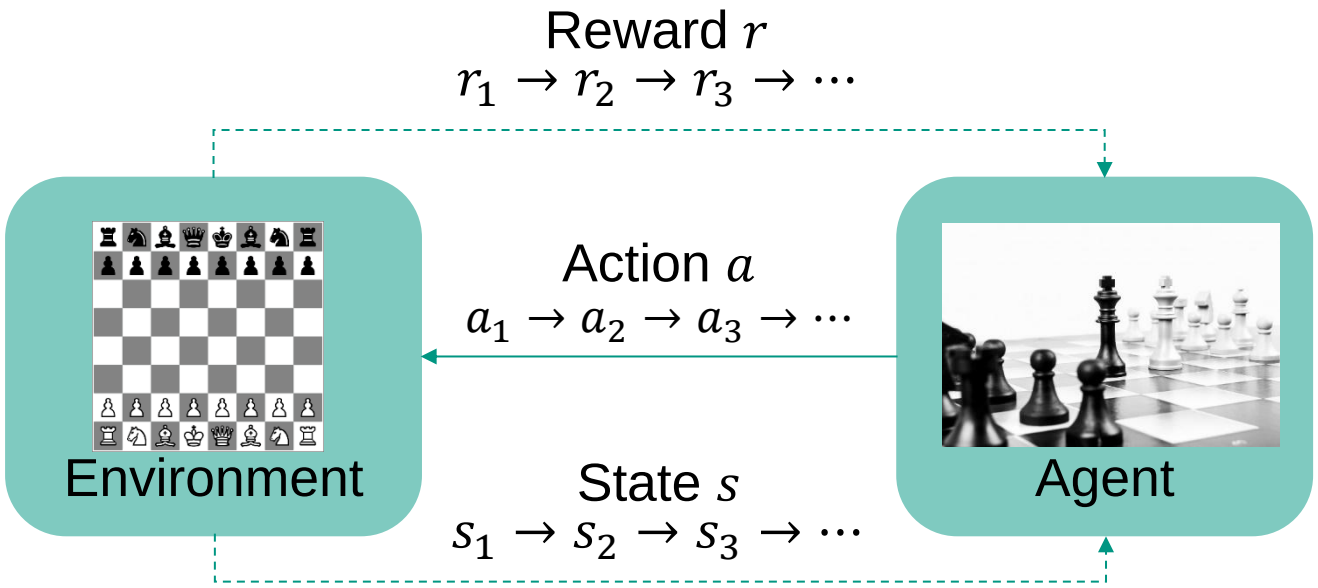
Clustering

Unsupervised Learning

Unlabeled data is given,
Goal: Structure approximation
$x \approx x'$

Reinforcement Learning (RL)

- Sequential decision making:



Reinforcement Learning

Data is generated by interacting with the environment

Goal: Maximize rewards

$$\sum_t \gamma^t \, R(S_t, A_t)$$

What is Reinforcement Learning?

- Humans and animals learn by interacting with the environment and receiving rewards/penalties
- Rewards serve as learning signals for the chosen actions

- **Differences to other types of learning:**
  - Learning process is **active instead of passive** – No fixed dataset, data/experience is collected by interacting with the environment.
  - **No supervision**, just a reward signal
  - **Interactions are often sequential** – Future decisions may depend on previous ones.
  - We can **learn optimal behavior without examples** of optimal behavior.

互动通常是按顺序进行的

Success Stories of RL

- 2013 Atari (DQN) [Deepmind]
- 2014 2D Locomotion (TRPO) [Berkeley]
- 2015 AlphaGo [Deepmind]
- 2016 Real Robot Manipulation (GPS) [Berkeley]
- 2018 Solving the Rubik's Cube (Domain Randomization) [openAI]
- 2019 StarCraft II (multi-agent RL) [Deepmind]

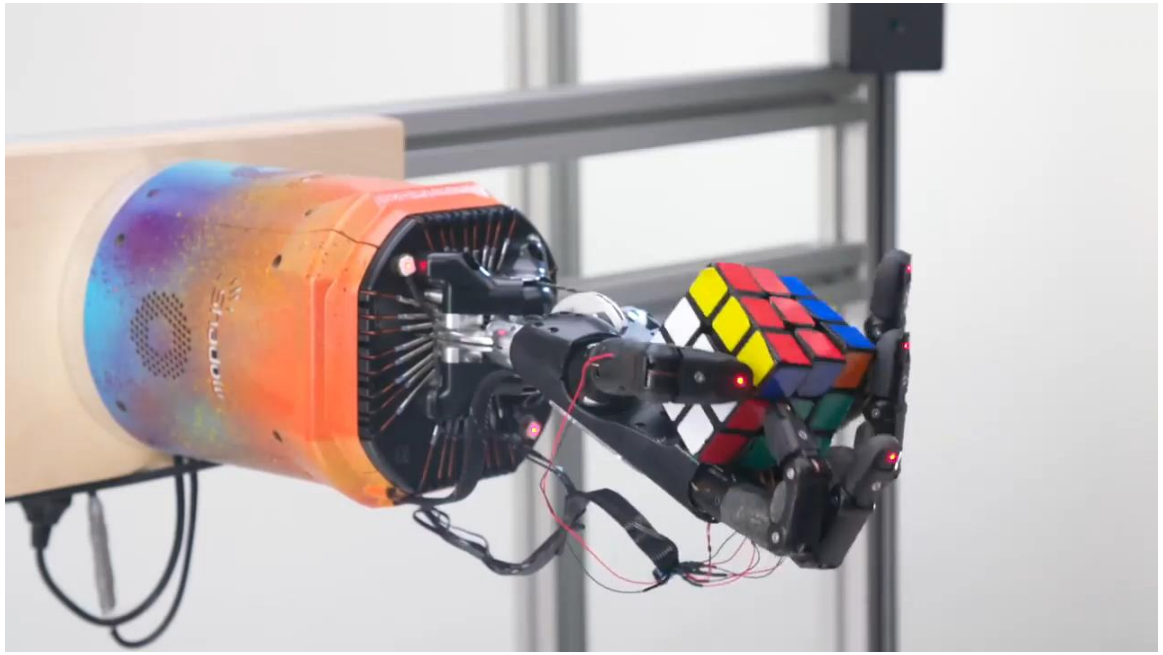
Atari

Google DeepMind's Deep Q-learning playing Atari Breakout!

Alpha GO

Alpha GO

Solving the Rubik's Cube

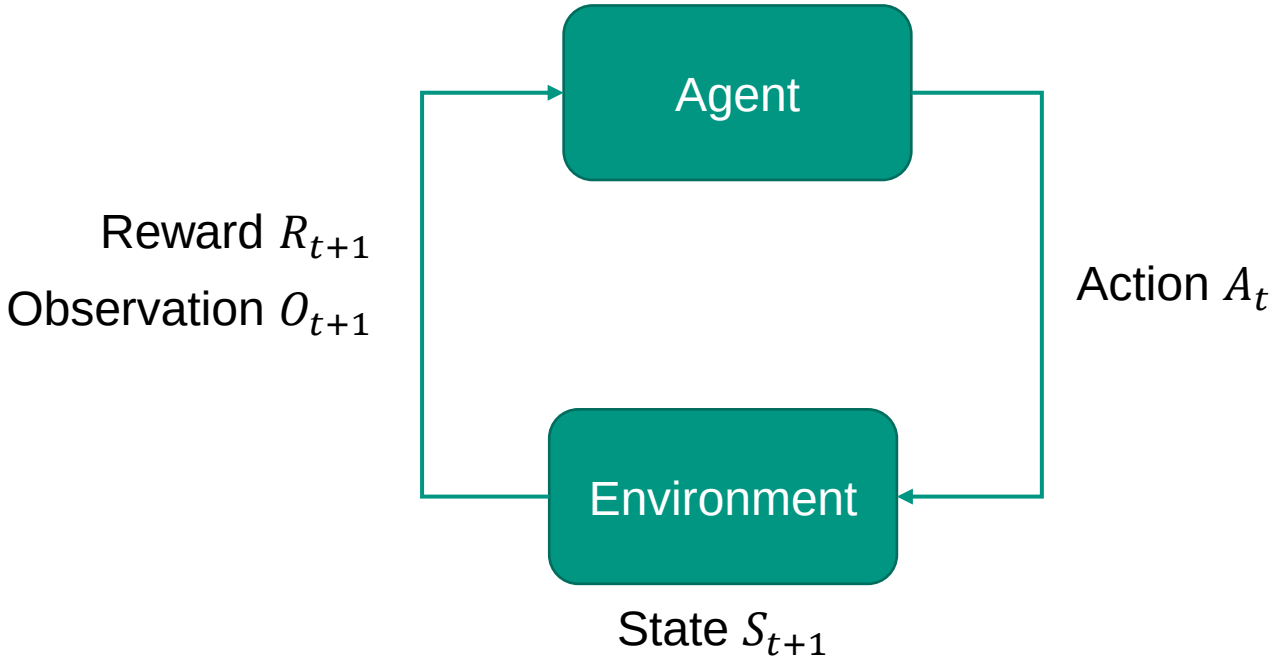


[Solving Rubiks Cube]

Outline

- Motivation and Examples
- Markov Decision Process (MDP)
- Bellman Principle
- Dynamic Programming
  - Policy Iteration
  - Value Iteration
- Summary

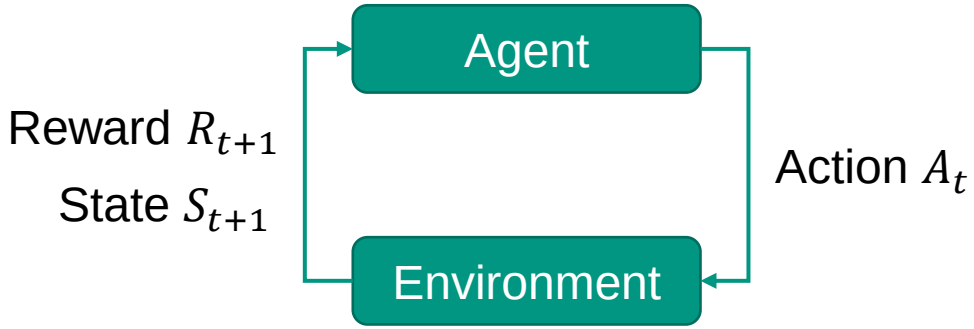
RL: Agent & Environment



- For each time step t:
  - Agent executes action A_t
  - Environment transitions to state S_{t+1} given action A_t
  - Agent receives observation O_{t+1} and reward R_{t+1}

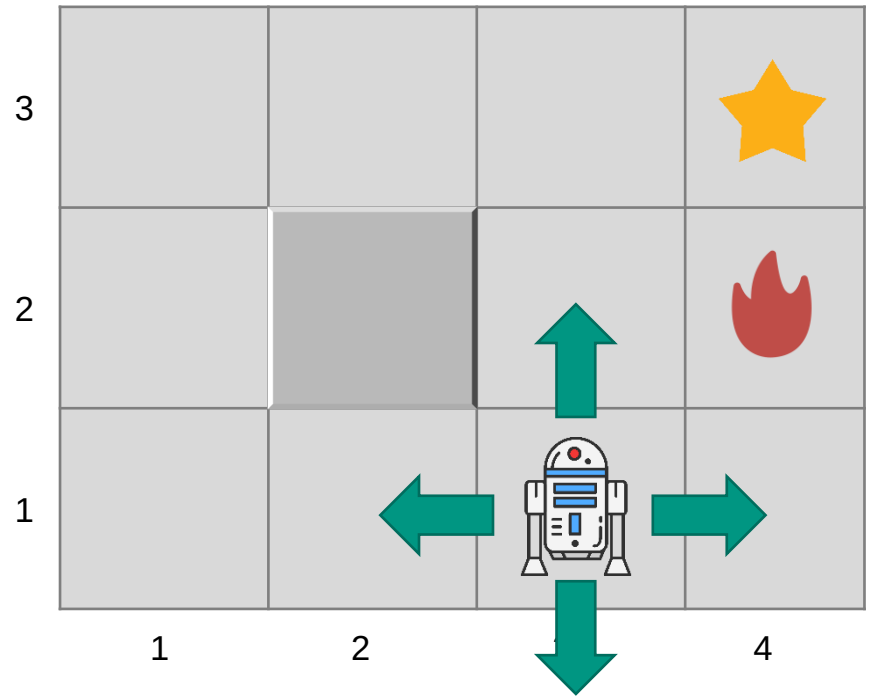
Markov Decision Process (MDP)

- An MDP is a mathematical framework for modeling the interactions between an agent and the environment
- The environment is **fully observable** ➡ $O_{t+1} = S_{t+1}$
- The **Markov property** is satisfied (see later)



Markov Decision Process (MDP)

- Formal description of sequential decision making problems
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{A}$ is a finite set of actions
  - p is a state transition (probability) function
    $p(s'|s,a) = p(S_{t+1} = s' \mid S_t = s, A_t = a)$
  - r is a reward function
    $r(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$
  - $\gamma \in [0,1]$ is a discount factor
    - $\gamma \in [0,1)$ for infinite time horizons



具体结果

Notation

$S_t, A_t, R_{t+1}, S_{t+1}$ are random variables and s, a, r', s' are specific outcomes

States and Observations

State

Observation

- A state s is a complete description of the state of the world.
  - There is no information that remains hidden from the state.
  - The world is completely observable.

- An observation o is a partial description of the state of the world.
  - There is information that remains hidden from the state.
  - The world is often only partially observable.

Markov Property

Markov Property

Definition

The problem satisfies the Markov property if and only if:

$$p(s_{t+1} \mid s_t, a_t) = p(s_{t+1} \mid s_0, a_0, \ldots, s_t, a_t)$$

- The future is independent of the past, given the present

$$\tau_{0:t-1} \rightarrow s_t \rightarrow a_t \rightarrow \tau_{t+1:\infty}$$

- The current state s_t comprises all relevant information from the past
- Once the state s_t is known, the past $\tau_{0:t-1}$ is irrelevant for the decision making and can thus be discarded

Actions

- Different environments enable different types of actions.
- The set of all valid actions in a certain environment is called the **action space**.
- Action spaces can be discrete or continuous.

- In this lecture, we often assume a **finite and discrete** action space.

Reward

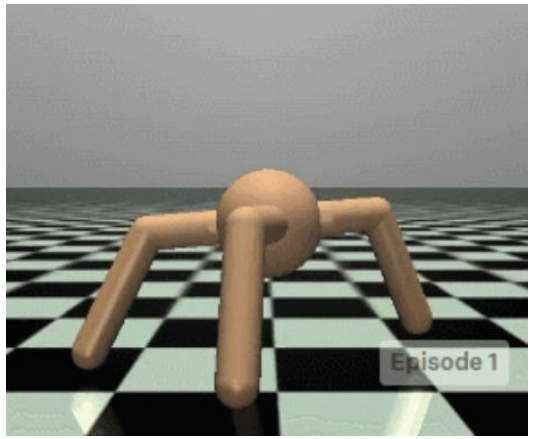
- A **Reward** R_t is a scalar feedback signal
- Describes the utility of the action at time step t

- Basis for reinforcement learning is the **reward hypothesis**

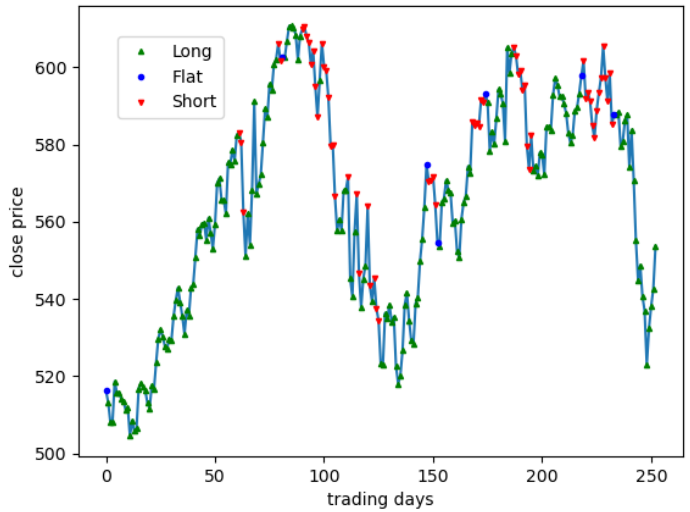
Reward hypothesis

Any goal can be formalized as the outcome of maximizing a cumulative reward!

Example: Reward



Getting a robot to walk



Managing an investment portfolio

Playing video or board games

Return

- **Return** G_t is defined as the cumulative reward with a discount factor $\gamma \in [0,1)$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$$

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

- G_t can be defined **recursively**:

$$G_t = R_{t+1} + \gamma \underbrace{(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots)}_{G_{t+1}} = R_{t+1} + \gamma G_{t+1}$$

Remark

回报是随机的变量，取决于策略和环境。

Returns are random variables that are dependent on the policy and the environment.

Discounting

- Why is the future reward discounted? $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$
  - Animal and human behavior shows preferences for immediate rewards
    - $\gamma = 0$ → only immediate reward is relevant
    - $\gamma \to 1$ → all future rewards are relevant
  - Mathematically practical since it prevents infinite returns for infinite time horizons
    - Used in some convergence proofs
  - Implies uncertainty about the future

片段任务

Episodes – Time Horizon and Terminal States

- So far, we only considered ***continuing tasks*** where the interaction with the environment never stops
  - The time horizon is infinite

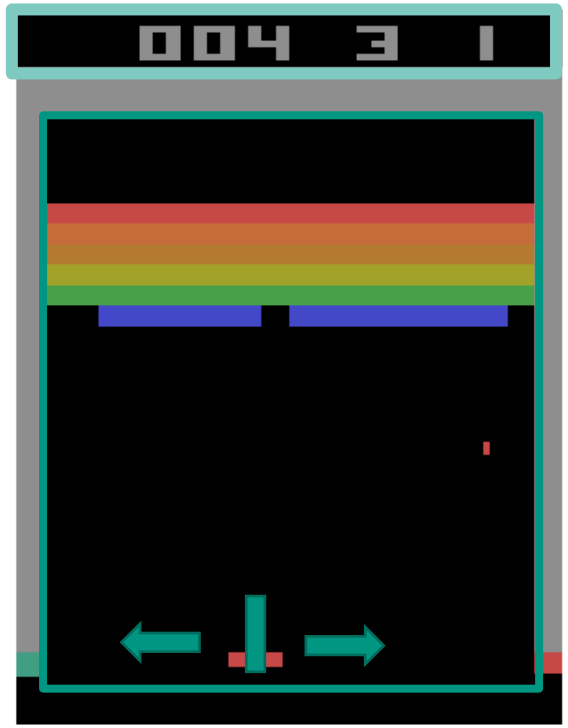
$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$

- Some tasks however end when a terminal state is reached → ***Episodic tasks***
  - Example: A chess game is over when the opponent's king is checkmated
  - The time horizon T is finite

$$G_t = \sum_{k=0}^{T-1} \gamma^k R_{t+1+k}$$

- An **episode** is a **finite sequence** of successive states, actions and rewards
- After an episode, the environment is reset before the next interaction can happen
- Episodic tasks can also be modelled as continuing tasks → Further concepts are only shown for the continuing tasks, but they also apply for the episodic tasks

Example: Atari Breakout



Action a

- Left
- Stay
- Right

Reward r

Observation o

- Entire picture as shown on the left

→ Does this observation satisfy the Markov property?

no,因为单帧图片会缺少方向、速度等信息，
需要对比两帧的变化，所以不满足Markov property

State s

- Position of the bricks
- Position and velocity of the ball
- Position of the agent (paddle)

Model

- A model consists of the **transition function** and the **reward function** and hence specifies the behavior of the environment
- Since the true model can be unknown to the agent, a model $\mathcal{M} = \langle \mathcal{P}, \mathcal{R} \rangle$ can also be estimated

- $\mathcal{P}$ predicts the next state:

$$\mathcal{P}(s'|s,a) \approx p(S_{t+1} = s'|S_t = s, A_t = a)$$

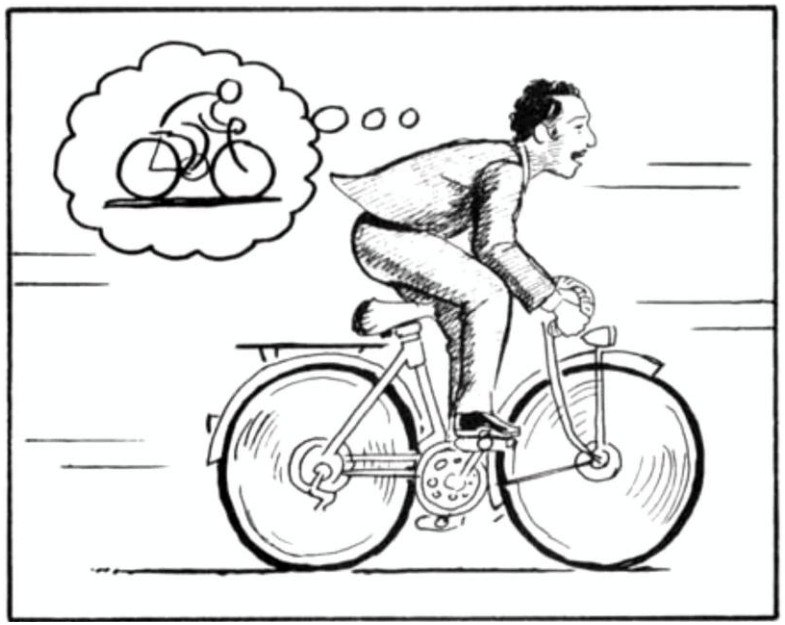
- $\mathcal{R}$ predicts the next reward:

$$\hat{r} = \mathcal{R}(s,a) \approx \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$$

Notation

$\hat{r}$ is an approximation

World Models



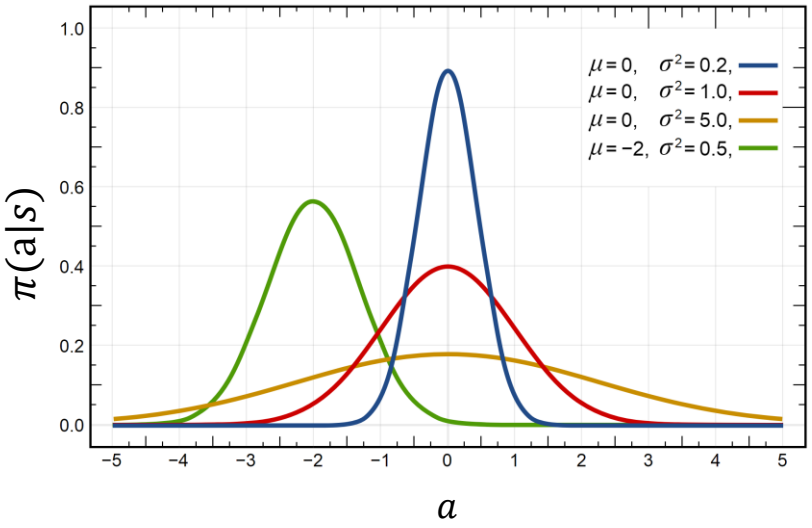
Policy

- A policy π specifies the agent‘s behavior for all states

- Deterministic policy:
  - $\pi: \mathcal{S} \rightarrow \mathcal{A}$ maps each state to an action
  - action $a = \pi(s)$

- Stochastic policy:
  - $\pi(a|s) = p(A_t = a \mid S_t = s)$ is the probability of taking action a in state s
  - action $a \sim \pi(\cdot\,|s)$ is sampled from the probability distribution
  - e.g. Gaussian distribution

Stochastic Policy



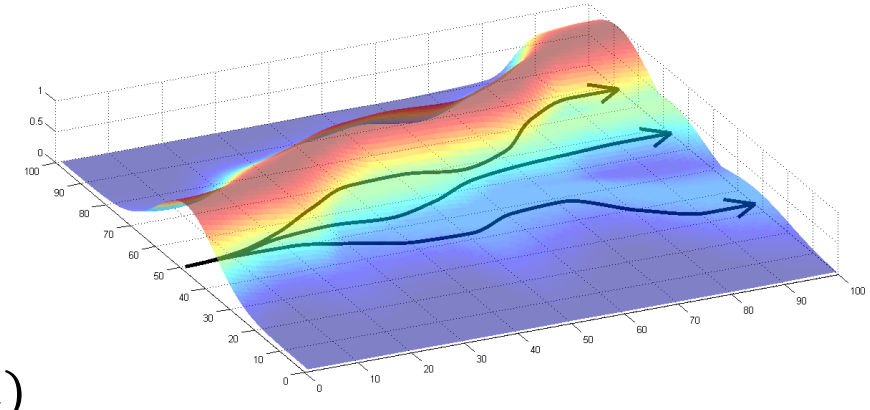
Trajectory

- A trajectory τ is a sequence of states and actions.

$$\tau = (s_0, a_0, s_1, a_1, \dots)$$

- $p(\tau)$ is the probability of trajectory τ by following the policy π

$$p(\tau) = \underbrace{\rho_0(s_0)}_{\text{start state distribution}} \prod_{t=0}^{T-1} \pi(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t)$$



Examples of trajectories sampled from policy π

Source: Deep Reinforcement Learning, Berkeley, 2019

Objective

- Recap: Return G_t is defined as the cumulative reward

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

- **Objective:** Find an **optimal policy** π^* that maximizes the expected return

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim \pi}[G_t]$$

Outline

- Motivation and Examples
- Markov Decision Process (MDP)
- Bellman Principle
- Dynamic Programming
  - Policy Iteration
  - Value Iteration
- Summary

State Value Function (V-Function)

- Assesses how good a **state** is for a given policy
- $v_\pi(s)$ is the expected return from state s by following policy π

$$v_\pi(s) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=0}^{\infty} \gamma^k \, R(S_{t+k}, A_{t+k}) \middle| S_t = s\right]$$

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|a,s) \underbrace{\mathbb{E}_{\tau\sim\pi}\left[\sum_{l=0}^{\infty} \gamma^l \, R(S_{t+1+l}, A_{t+1+l}) \middle| S_{t+1} = s'\right]}_{v_\pi(s')}\right)$$

- Bellman equation:

$$v_\pi(s) = \mathbb{E}_{\tau\sim\pi}[R_{t+1} + \gamma \, v_\pi(S_{t+1}) \mid S_t = s]$$

Action Value Function (Q-Function)

- Assesses how good an **action** is for a given policy
- $q_\pi(s,a)$ is the expected return by taking action a from state s and following policy π afterwards

$$q_\pi(s,a) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=0}^{\infty} \gamma^k \, R(S_{t+k}, A_{t+k}) \middle| S_t = s, A_t = a\right]$$

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|a,s) \sum_{a'\in\mathcal{A}} \pi(a'|s') \underbrace{\mathbb{E}_{\tau\sim\pi}\left[\sum_{l=0}^{\infty} \gamma^l \, R(S_{t+1+l}, A_{t+1+l}) \middle| S_{t+1} = s', A_{t+1} = a'\right]}_{q_\pi(s',a')}$$

- Bellman equation:

$$q_\pi(s,a) = \mathbb{E}_{\tau\sim\pi}[R_{t+1} + \gamma \, q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

Bellman Equations

- **Summary:**
  - Value functions can be defined **recursively**
  - Value functions are expectations of the **immediate reward** plus the value functions at the next state/action

$$v_\pi(s) = \mathbb{E}_{\tau\sim\pi}[\boldsymbol{R_{t+1}} + \gamma\, v_\pi(S_{t+1}) \mid S_t = s]$$

$$q_\pi(s,a) = \mathbb{E}_{\tau\sim\pi}[\boldsymbol{R_{t+1}} + \gamma\, q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

Optimal Policy

- A policy π' is better than or as good as π, if:

$$\pi' \geq \pi \iff v_{\pi'}(s) \geq v_{\pi}(s)\,, \forall s \in S$$

- At least one optimal policy π^* exists with $\pi^* \geq \pi\,, \forall \pi$

- All optimal policies have the same optimal value function $v^*(s)$ and $q^*(s, a)$

Optimal Value Functions

- The optimal state value function is the maximum value across all strategies

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

- The optimal action value function is the maximum value across all strategies

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

- An MDP is solved when the optimal value functions are known, since a **greedy policy** is then also an optimal policy

$$\pi^*(a|s) = \text{greedy}(v^*)$$

$$\pi^*(a|s) = \text{greedy}(q^*)$$

Greedy Policy

- A greedy policy chooses the action that looks **best in the short term** – after one step of lookahead – according to the value functions v_π or q_π for a given policy π.

$$\pi_{new}(a|s) = \text{greedy}(v_\pi) = \begin{cases} 1 & \text{if } a = \arg\max_{a' \in \mathcal{A}} \left(r(s,a') + \sum_{s' \in \mathcal{S}} p(s'|s,a')\, v_\pi(s') \right) \\ 0 & \text{otherwise} \end{cases}$$

$$\pi_{new}(a|s) = \text{greedy}(q_\pi) = \begin{cases} 1 & \text{if } a = \arg\max_{a' \in \mathcal{A}} q_\pi(s,a') \\ 0 & \text{otherwise} \end{cases}$$

- In general, the greedy policy π_{new} is better than or as good as the previous policy π.
- If the greedy policy π_{new} is as good as, but not better than, the previous policy π, then both policies π_{new} and π are optimal policies.

Bellman Optimality Equations

Bellman Optimality Principle

"An optimal policy has the property that whatever the initial state and initial decision is, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

- Bellman Optimality Equations:
  - An optimal policy π^* leads to optimal value functions $v^*(s)$ and $q^*(s,a)$:

$$v^*(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, v^*(s') \right)$$

$$q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \max_{a' \in \mathcal{A}} q^*(s',a')$$

Bellman Equation in Matrix Form

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, v_\pi(s') \right)$$

- The Bellman state value equation $v_\pi(s)$ can be expressed using matrices:

$$\boldsymbol{v}_\pi = \boldsymbol{r}^\pi + \gamma\, \boldsymbol{P}^\pi\, \boldsymbol{v}_\pi$$

- With

$\boldsymbol{v}_\pi \in \mathbb{R}^{|\boldsymbol{S}|}$ $\quad v_i = v_\pi(s_i)$

$\boldsymbol{r}^\pi \in \mathbb{R}^{|\boldsymbol{S}|}$ $\quad r_i = \sum_{a \in \mathcal{A}} \pi(a|s_i)\, r(s_i,a)$

$\boldsymbol{P}^\pi \in \mathbb{R}^{|\boldsymbol{S}| \times |\boldsymbol{S}|}$ $\quad P^\pi_{ij} = p(s_j|s_i) = \sum_{a \in \mathcal{A}} \pi(a|s_i)\, p(s_j|s_i,a)$

Bellman Equation in Matrix Form

- This is a linear equation that can be solved directly:

$$\begin{aligned} \boldsymbol{v}_\pi &= \boldsymbol{r}^\pi + \gamma\, \boldsymbol{P}^\pi\, \boldsymbol{v}_\pi \\ (\boldsymbol{I} - \gamma\, \boldsymbol{P}^\pi)\boldsymbol{v}_\pi &= \boldsymbol{r}^\pi \\ \boldsymbol{v}_\pi &= (\boldsymbol{I} - \gamma\, \boldsymbol{P}^\pi)^{-1}\boldsymbol{r}^\pi \end{aligned}$$

- Challenge:
  - Computational complexity is $O(|\mathcal{S}|^3)$ – only possible for small problems

- There are iterative methods for larger problems
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Solving the Bellman Optimality Equation

- The Bellman **optimality** equation is **non-linear** due to the max operator

$$v^*(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, v^*(s') \right)$$

  - Cannot use the same direct matrix solution as for the Bellman equations

- There exist many iterative solution methods:
  - Using models → **Dynamic programming**
    - Policy Iteration
    - Value Iteration
  - Using samples
    - Monte Carlo
    - Q-Learning
    - …

Outline

- Motivation and Examples
- Markov Decision Process (MDP)
- Bellman Principle
- Dynamic Programming
  - Policy Iteration
  - Value Iteration
- Summary

Dynamic Programming

> *"Dynamic programming refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP)."*
>
> Sutton & Barto 2018

- A perfect model is given → The true transition and reward functions are known

- All these methods consist of two important parts:
  - Policy Evaluation
  - Policy Improvement

Policy Evaluation

- Also referred to as “prediction problem”

- Task:
  - Estimate the value function v_π for a given policy π
- Idea:
  - Use the recursive **Bellman equations** to **iteratively** update the value functions

$$v_\pi(s) = \mathbb{E}_{\tau\sim\pi}[R_{t+1} + \gamma\, v_\pi(S_{t+1}) \mid S_t = s]$$

- Notation:
  - Approximations or estimations of the value function are written in capital letters: $V_\pi \approx v_\pi$
  - Let V_k be the estimate of the k-th iteration

Policy Evaluation

- **Input:** policy π, a small threshold ϵ determining the accuracy of the estimation

Algorithm

Initialize $V_0(s)$ arbitrarily for each $s \in \mathcal{S}$
Repeat until $\Delta \leq \epsilon$:
    $\Delta \leftarrow 0$
    Repeat for each $s \in \mathcal{S}$:

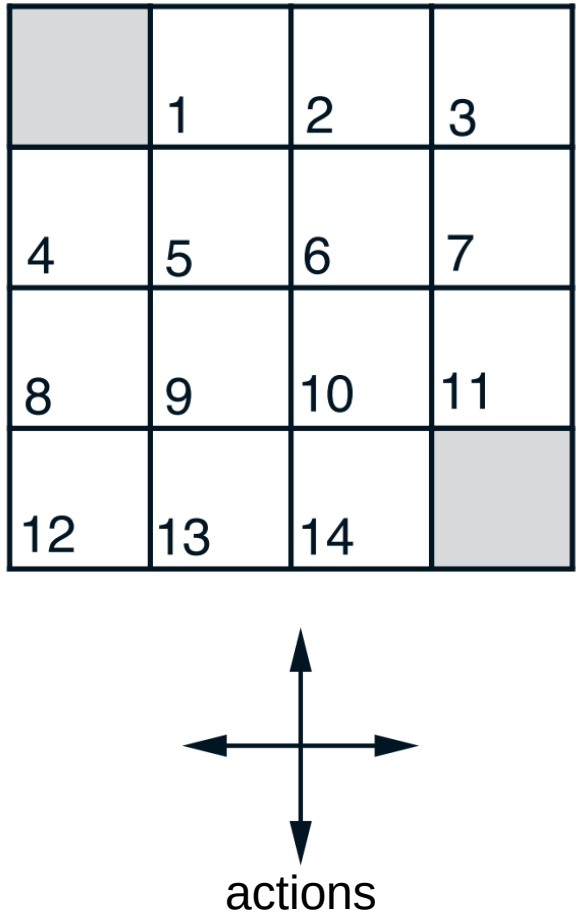
$$V_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, V_k(s') \right)$$

    $\Delta \leftarrow \max(\Delta, |V_{k+1}(s) - V_k(s)|)$
    $k \leftarrow k + 1$

- This algorithm converges under appropriate conditions (e.g., $\gamma < 1$): $\lim_{k \to \infty} V_k(s) = v_\pi(s)$

Example: Policy Evaluation

- Undiscounted, episodic task → $\gamma = 1$
- 16 states
  - gray cells represent terminal states
- 4 actions (up, down, right, left)
- The reward is -1 for all transitions until a terminal state is reached.
- Model is deterministic: $P(s'|s,a) = 1 \quad \exists s' \; \exists s \; \exists a$
  - If there is a wall in front, the agent stays in the current state
- Policy is stochastic:
  → **Random Policy** (all actions equally likely) $a \sim \pi(a|s)$



Example: Policy Evaluation

$V_0(s)$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$V_1(s)$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$V_2(s)$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

- **Example:**

$$V_2(s_1) = \sum_{a \in \mathcal{A}} \pi(a|s_1) \left(r(s_1, a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s_1, a)\, V_1(s') \right)$$

$$V_2(s_1) = 3 * 0.25 * \big(-1 + (-1)\big) + \underbrace{0.25 * (-1 + 0)}_{\text{terminal state is reached}} = -1.75$$

Example: Policy Evaluation

$V_3(s)$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$V_{10}(s)$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$V_\infty(s) = v_\pi(s)$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Policy Improvement

- Also referred to as “control problem”

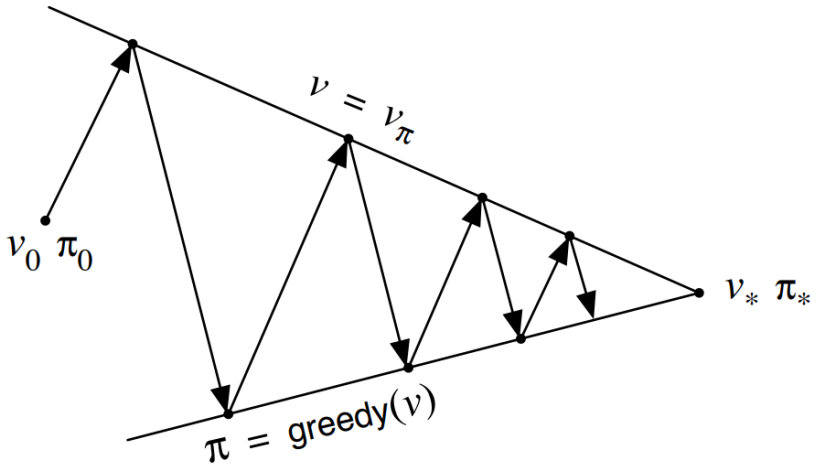
- Task:
  - Improve the policy given the value function V_{π_i} of the policy π_i
- Idea:
  - The new policy π_{i+1} shall be the **greedy policy** that chooses the action maximizing the immediate reward plus the old value function V_{π_i} for the future states.

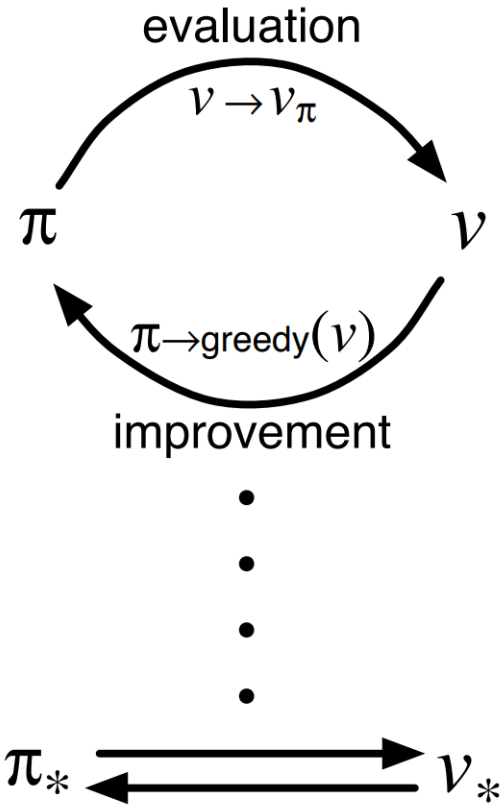
$$\pi_{i+1}(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a' \in \mathcal{A}} \left(r(s,a') + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a')\, V_{\pi_i}(s') \right) \\ 0 & \text{otherwise} \end{cases}$$

Policy Iteration

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} v^*$$

→ converges to the optimal policy and value function

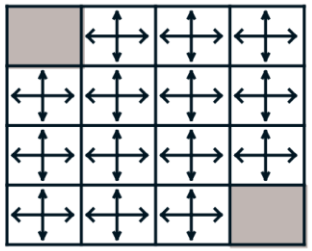




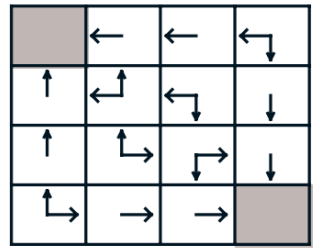
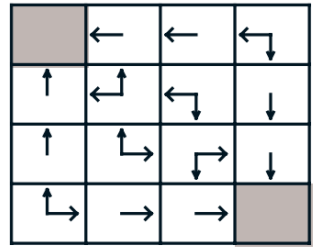
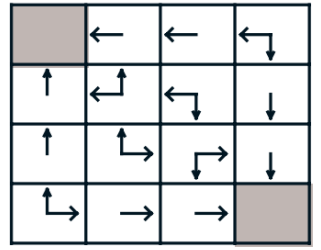
- **Policy Evaluation**: Estimate V_{π_i}
- **Policy Improvement**: Generate $\pi_{i+1} \geq \pi_i$

Example: Policy Iteration

- Must the policy evaluation converge to v_{π_0} before the next policy improvement step?



π_0

	π_0 Evaluation	π_1(Greedy)
$V_3(s)$	0.0 -2.4 -2.9 -3.0 -2.4 -2.9 -3.0 -2.9 -2.9 -3.0 -2.9 -2.4 -3.0 -2.9 -2.4 0.0	
$V_{10}(s)$	0.0 -6.1 -8.4 -9.0 -6.1 -7.7 -8.4 -8.4 -8.4 -8.4 -7.7 -6.1 -9.0 -8.4 -6.1 0.0	
$V_\infty(s)$	0.0 -14. -20. -22. -14. -18. -20. -20. -20. -20. -18. -14. -22. -20. -14. 0.0	

- **No**. k = 3 is sufficient here for determining the greedy policy π_1
  (→ policy π_1 is even the optimal policy)
- Extreme case with k = 1 → **Value Iteration**

Value Iteration

- Combines the truncated policy evaluation steps and the policy improvement
- Uses the idea of the **Bellman optimality equation**

$$v^*(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, v^*(s') \right)$$

- Each iteration k updates both the value function and **implicitly** the policy

$$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, V_k(s') \right)$$

Value Iteration

- **Input:** A small threshold ϵ determining the accuracy of the estimation

Algorithm

Initialize $V_0(s)$ arbitrarily for each $s \in \mathcal{S}$
Repeat until $\Delta \leq \epsilon$:
    $\Delta \leftarrow 0$
    Repeat for each $s \in \mathcal{S}$:

$$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, V_k(s') \right)$$

    $\Delta \leftarrow \max(\Delta, |V_{k+1}(s) - V_k(s)|)$
    $k \leftarrow k + 1$

- This algorithm converges under appropriate conditions (e.g., $\gamma < 1$): $\lim_{k \to \infty} V_k(s) = v_{\pi^*}(s)$

Outlook: Policy and Value Iteration with Q-Values

- So far, we only considered policy and value iteration with state value functions $V_\pi(s)$.
- Both concepts can also be applied with **action value functions** $Q_\pi(s,a)$.

- Some hints on how to accomplish that with **Q-Values**:
  - **Policy Evaluation:**

$$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \left(p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_k(s',a') \right)$$

  - **Policy Improvement:**

$$\pi_{i+1}(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a' \in \mathcal{A}} Q_{\pi_i}(s,a') \\ 0 & \text{otherwise} \end{cases}$$

  - **Value Iteration:**

$$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \left(p(s'|s,a) \max_{a' \in \mathcal{A}} Q_k(s',a') \right)$$

Outline

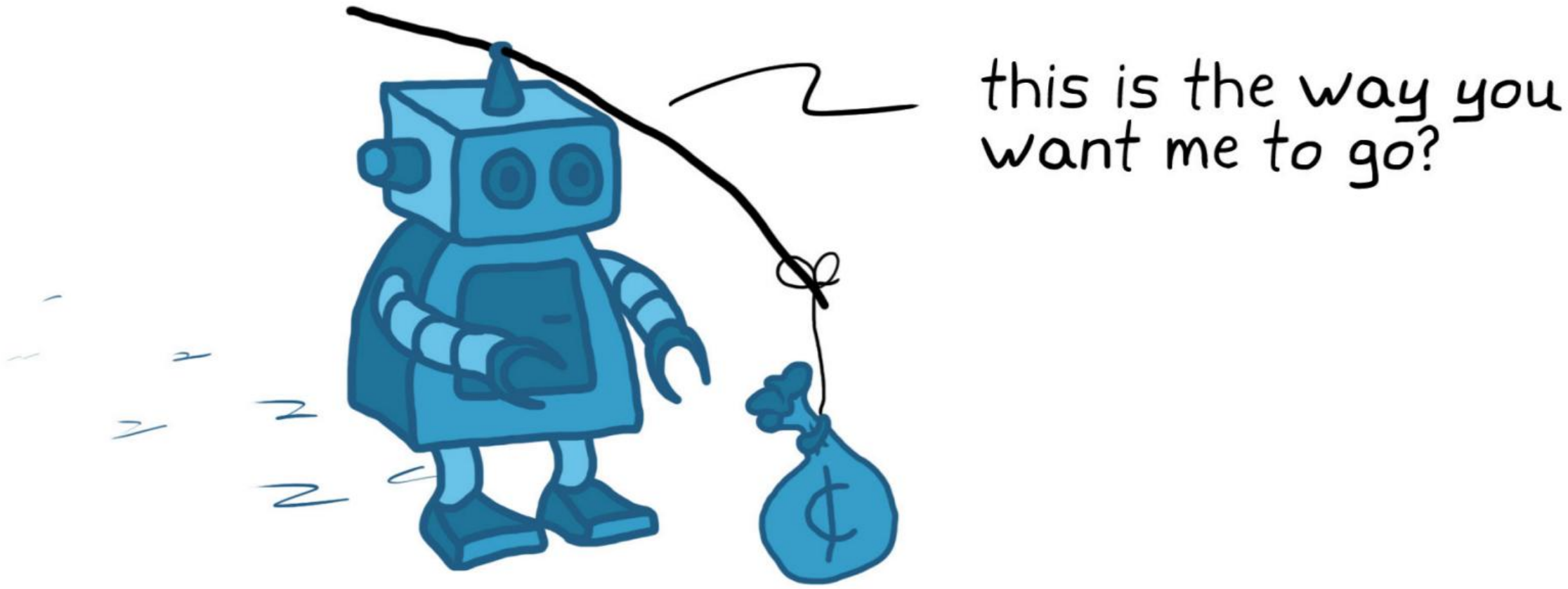
- Motivation and Examples
- Markov Decision Process (MDP)
- Bellman Principle
- Dynamic Programming
  - Policy Iteration
  - Value Iteration
- Summary

Summary

- Reinforcement Learning (RL):
  - Agent interacts with the environment to maximize the expected cumulative reward
- Markov Decision Process (MDP)
- Value functions: Expected returns for a given policy
- Bellman equations: Recursive definition of value functions
- Optimality principles in MDPs:
  - Optimal value functions, Bellman optimality equations and optimal policies
- Two key components in RL:
  - Policy evaluation (prediction) and policy improvement (control)
- Dynamic Programming to compute the optimal value function and policy iteratively if the model is known:
  - Policy Iteration
  - Value Iteration

Literature

- R. Sutton 2018 – “Reinforcement Learning: An Introduction”
- S. Levine 2022 – “Deep Reinforcement Learning” (Berkley Course on RL)
- H. van Hasselt 2021 – „Reinforcement Learning Lecture Series 2021” (DeepMind x UCL)
- P. Abbeel 2017 – “Deep RL Bootcamp” (Berkley Course on RL)
- D. Silver 2015 – “Reinforcement Learning” (UCL Course on RL)
- OpenAI 2018 – “SpinningUp”


this is the way you want me to go?