Karlsruher Institut für Technologie

Machine Learning 1 – Fundamentals

Reinforcement Learning 2

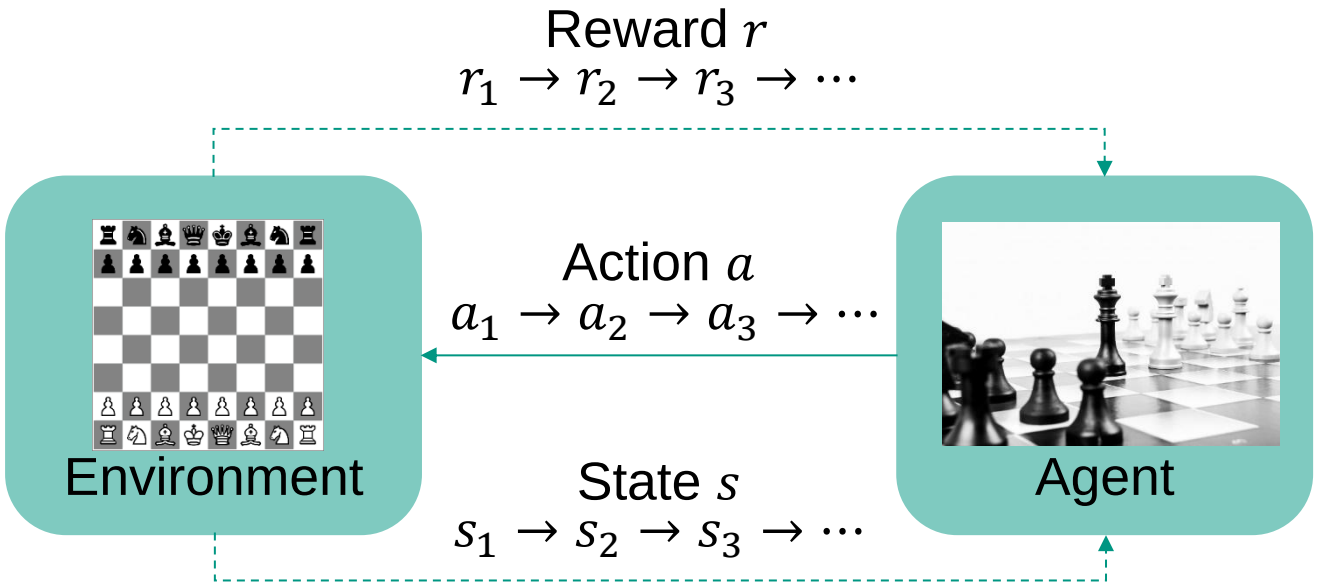
Prof. Dr. J. M. Zöllner, M.Sc. Philipp Stegmaier, M.Sc. Karam Daaboul

Outline

- Recap of RL Basics
- Taxonomy of Reinforcement Learning
- Value-Based RL
  - Monte-Carlo Methods
  - Temporal-Difference (TD) Learning
  - n-step Bootstrapping
  - Deep Q-Network (DQN)
- Summary

Reinforcement Learning (RL)

- Sequential decision making:



Reward hypothesis

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

Reinforcement Learning

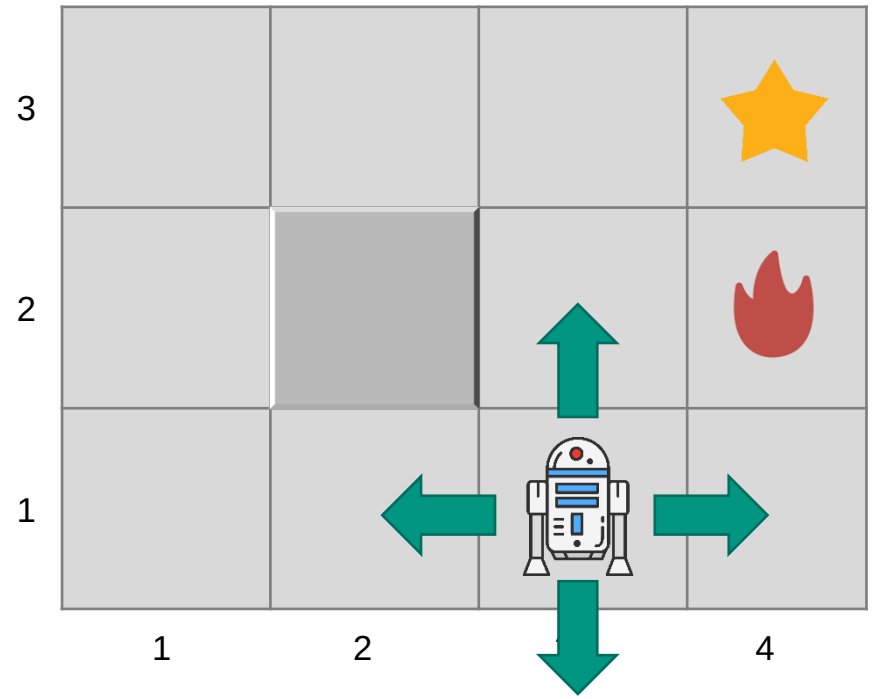
Data is generated by interacting with the environment

Goal: Maximize rewards

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

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



Notation

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

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:

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

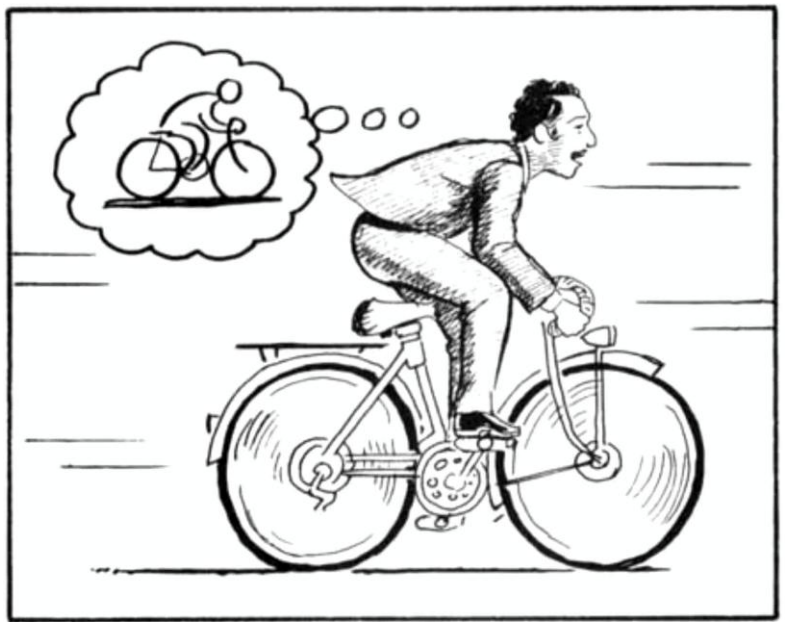
- $\mathcal{R}$ predicts the next reward:

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

Notation

$\hat{r}$ is an approximation

World Models



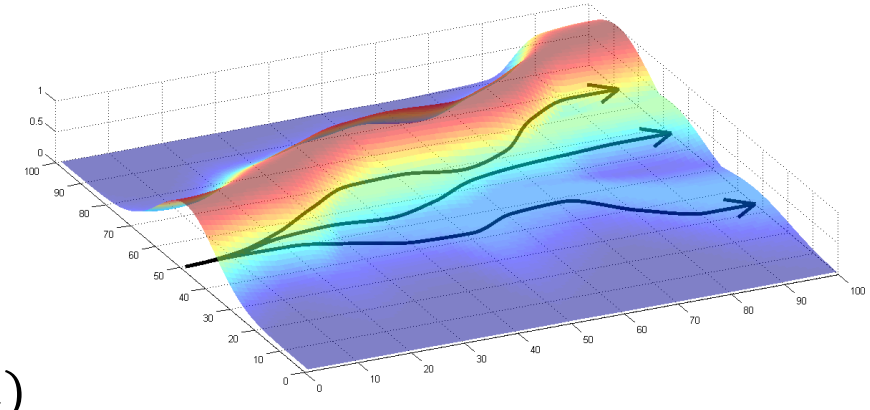
Trajectory

- 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

- 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

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

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

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

- 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

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

- 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 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 $q^*(s,a)$:

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

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

Outline

- Recap of RL Basics
- Taxonomy of Reinforcement Learning
- Value-Based RL
  - Monte-Carlo Methods
  - Temporal-Difference (TD) Learning
  - n-step Bootstrapping
  - Deep Q-Network (DQN)
- Summary

Reinforcement Learning

- In this lecture, we will have our first look at “real” reinforcement learning.

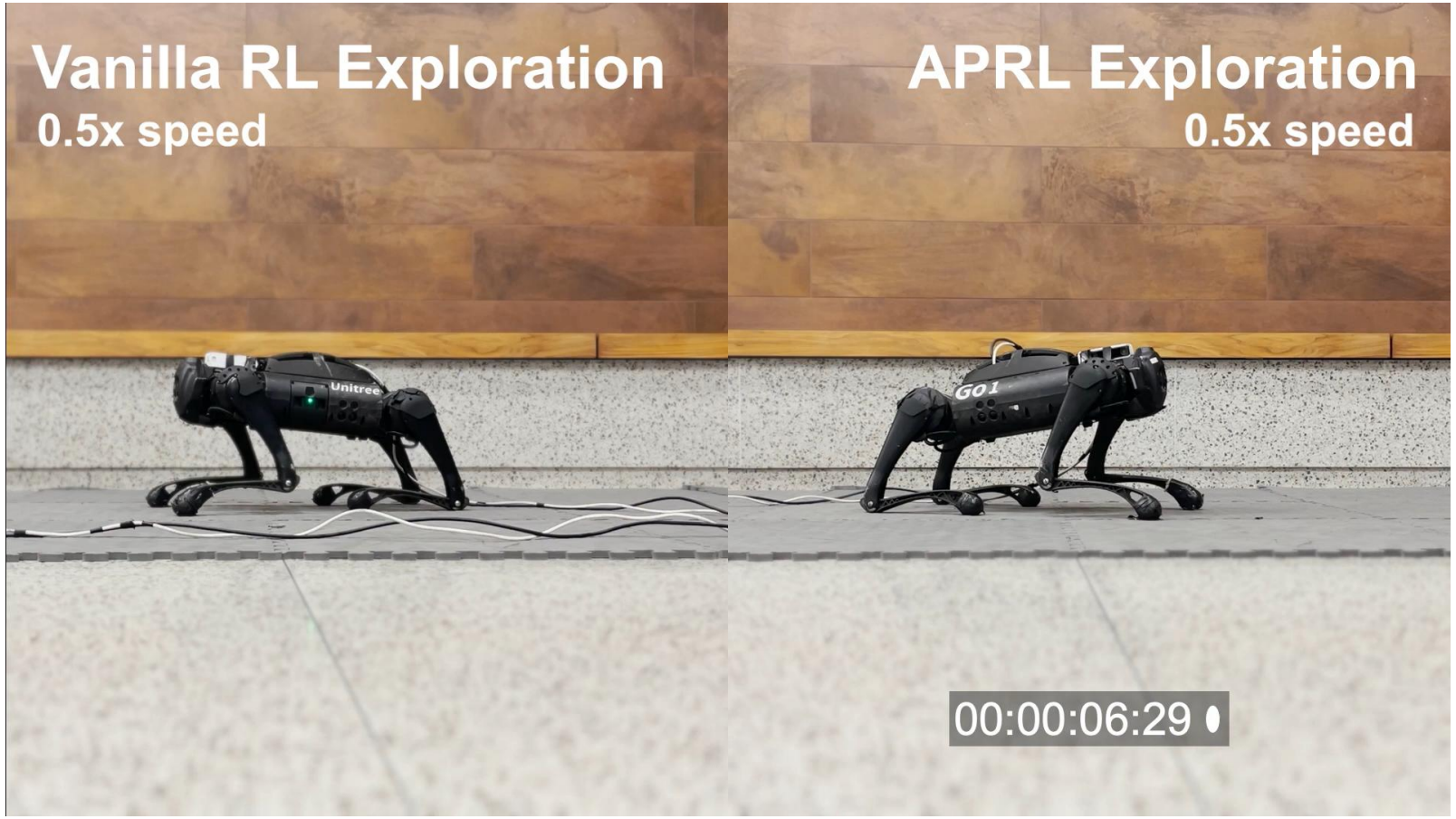
- Last lecture, we covered **dynamic programming**:
  - The model consisting of the reward function $r(s, a)$ and the transition function $p(s'|s, a)$ were known.
  - The value functions were updated iteratively given the model.

- However in **“real” reinforcement learning**, the true model of the environment is often not known by the agent.
  - The agent interacts with the environment to gather **experience samples**.
  - From this experience, the agent learns a policy, value functions or a model.

Exploration

- Reinforcement learning is **trial-and-error learning**
- **Exploration:**
  - Extending the current knowledge by choosing actions that can appear to be sup-optimal but may potentially lead to higher returns
- **Exploitation:**
  - Choosing the best actions based on the current knowledge
- There exists an "**exploration-exploitation tradeoff**":
  → Agent must balance between exploration and exploitation so that all actions are sufficiently explored, but the return is also maximized.

Example: Exploration



Smith et al. (2024), “Grow Your Limits: Continuous Improvement with Real-World RL for Robotic Locomotion”

Epsilon-Greedy Policy

- A greedy policy can get stuck on a suboptimal action forever since it only uses exploitation
- A solution for the exploration-exploitation tradeoff is the **ϵ-greedy** policy:
  - With probability $1 - \epsilon$ select greedy action: $a = \arg\max_{\tilde{a} \in \mathcal{A}} Q_\pi(s, \tilde{a})$
  - With probability ϵ select a random action
  - Equivalently:

$$\pi_{new}(a|s) = \epsilon\text{−greedy}(Q_\pi) = \begin{cases} (1-\epsilon) + \epsilon/|\mathcal{A}| & \text{if } a = \arg\max_{\tilde{a} \in \mathcal{A}} Q_\pi(s, \tilde{a}) \\ \epsilon/|\mathcal{A}| & \text{otherwise} \end{cases}$$

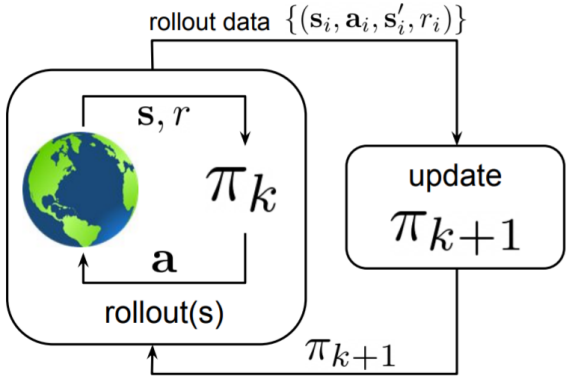
On- / Off-Policy RL

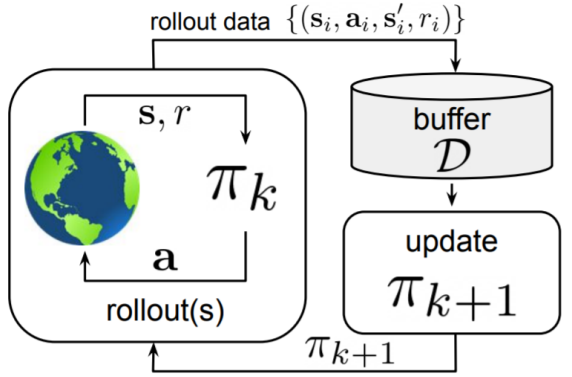
- **Problem:** How can we evaluate or improve the optimal policy while behaving according to an exploratory policy?

- **On-policy methods:**
  - evaluate or improve the policy that is used for the experience collection
  - learn action values not for the optimal policy, but for a near-optimal policy that still explores → e.g. the ϵ-greedy policy

- **Off-policy methods:**
  - evaluate or improve a **target policy** that is different from the **behavior policy** used for the experience collection
  - Only the behavior policy must be exploratory

On- / Off-Policy RL





On-Policy RL

1. Gather data $\{(s_i, a_i, r_i', s_i')\}$ with π_{k}
2. Update π_{k}

Off-Policy RL

1. Gather data $\{(s_i, a_i, r_i', s_i')\}$ with π_{k} and store them in a replay buffer D
2. D is composed of samples from $\pi_0, \pi_1, \ldots, \pi_{\mathrm{k}}$
3. D is used to train a new policy $\pi_{\mathrm{k+1}}$
4. $k \leftarrow k + 1$

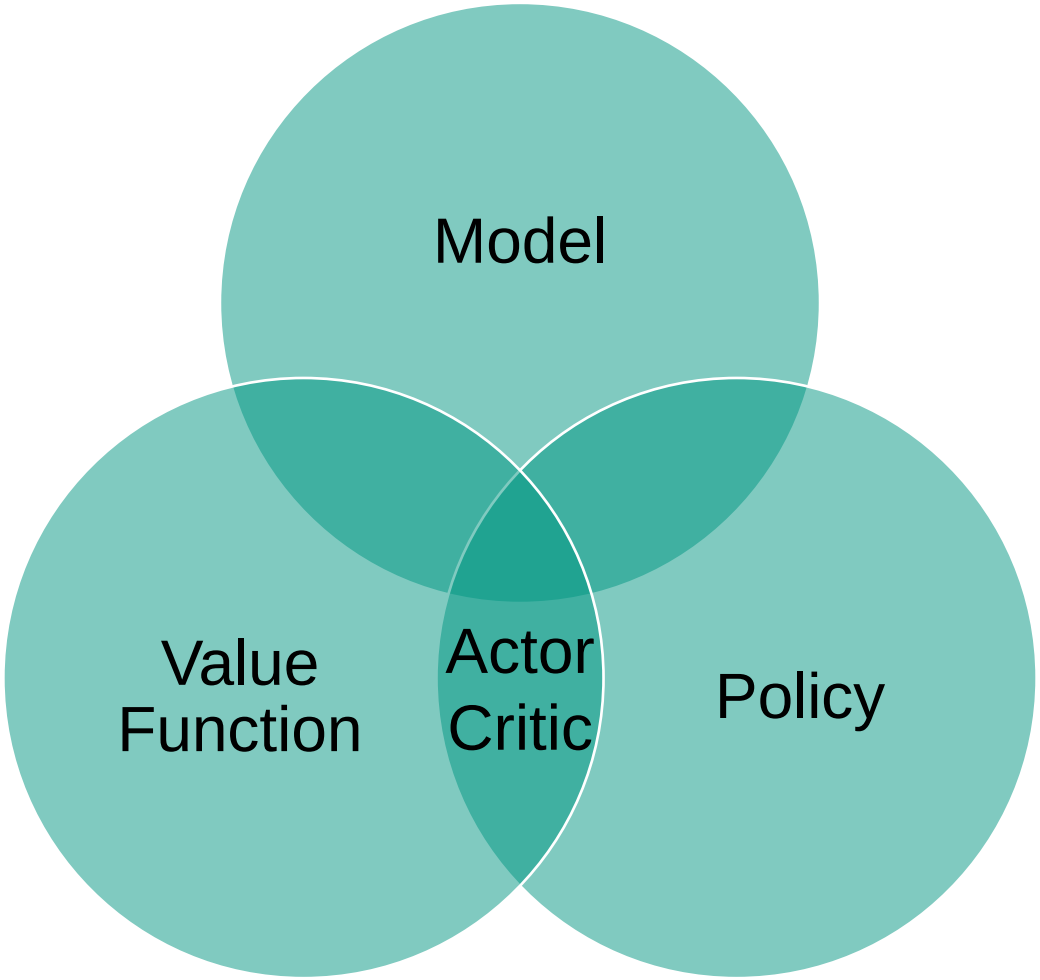
Source: Offline RL: Tutorial, Review, and Perspectives on Open Problems 2020

Taxonomy of Reinforcement Learning

- **Value-based**
  - No Policy (Implicit)
  - Value Function
- **Policy-based**
  - Policy
  - No Value Function
- **Actor-Critic**
  - Policy
  - Value Function

- **Model-free**
  - No explicit model necessary
- **Model-based**
  - Model is learned (or given)

Taxonomy of Reinforcement Learning



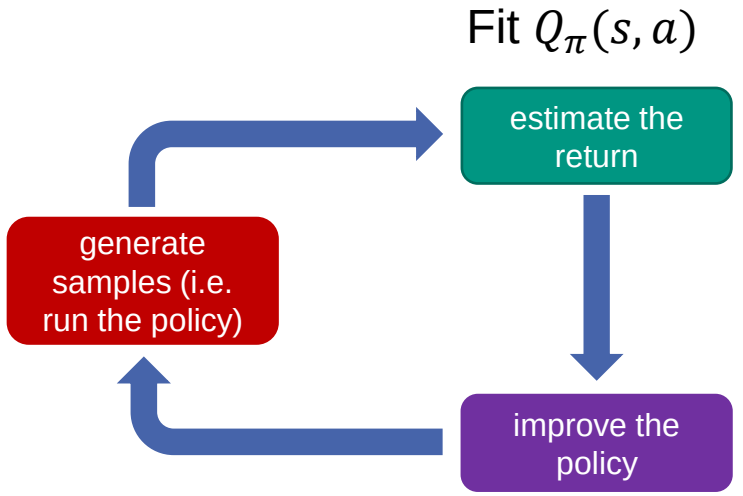
Outline

- Recap of RL Basics
- Taxonomy of Reinforcement Learning
- Value-Based RL
  - Monte-Carlo Methods
  - Temporal-Difference (TD) Learning
  - n-step Bootstrapping
  - Deep Q-Learning Network (DQN)
- Summary

Value-Based RL

- **Main idea:**
  - Don‘t learn a policy explicitly
  - Just learn a value function, e.g. $Q_\pi(s, a)$
  - Derive an optimal policy from this learned value function

- **Difference to dynamic programming:**
  - Generate sample trajectories by interacting with the environment according to the policy
  - The value function is learned with the generated samples and not computed with the model



$$a^* = \arg\max_{a \in \mathcal{A}} Q_\pi(s, a)$$

Outline

- Recap of RL Basics
- Taxonomy of Reinforcement Learning
- Value-Based RL
  - Monte-Carlo Methods
  - Temporal-Difference (TD) Learning
  - n-step Bootstrapping
  - Deep Q-Network (DQN)
- Summary

Monte-Carlo Methods

- Monte-Carlo methods estimate an expectation through a **mean of samples**
- In general:

$$\mathbb{E}_{p(x)}[f(x)] = \int f(x)p(x)dx \approx \frac{1}{N}\sum_{x_i \sim p(x)} f(x_i)$$

- In the context of RL:
  - The true value function $q_\pi(s,a)$ is an expectation and can hence be estimated with trajectory samples

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

Monte-Carlo Evaluation

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

- Each sampled trajectory i must be an **episode** with a finite time horizon T_i
- In total, N relevant trajectories are sampled

- The true value function $q_\pi(s,a)$ can then be estimated with

$$Q_\pi(s,a) = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=0}^{T_i-1} \gamma^k \, r_{t_i+1+k}$$

  under the **condition** that all initial rewards r_{t_i+1} must be the result of choosing action a in the initial state s

Example: Monte-Carlo Evaluation

- $\mathcal{S} = \{s^{(1)}, s^{(2)}, s^{(3)}, s^{(4)}\}$ and $\mathcal{A} = \{a^{(1)}, a^{(2)}\}$ and $\gamma = 1$

- The following trajectories with rewards were sampled:
  1. $(s^{(2)}, a^{(1)}, +1, s^{(1)}, a^{(1)}, +2, s^{(3)}, a^{(2)}, +1, s^{(4)})$
  2. $(s^{(1)}, a^{(1)}, +3, s^{(3)}, a^{(1)}, +1, s^{(1)}, a^{(1)}, +1, s^{(4)})$
  3. $(s^{(3)}, a^{(2)}, +1, s^{(1)}, a^{(1)}, -2, s^{(4)})$

Notation: $(s_0, a_0, r_1, s_1, a_1, r_2, s_2, \dots, s_{T_i})$

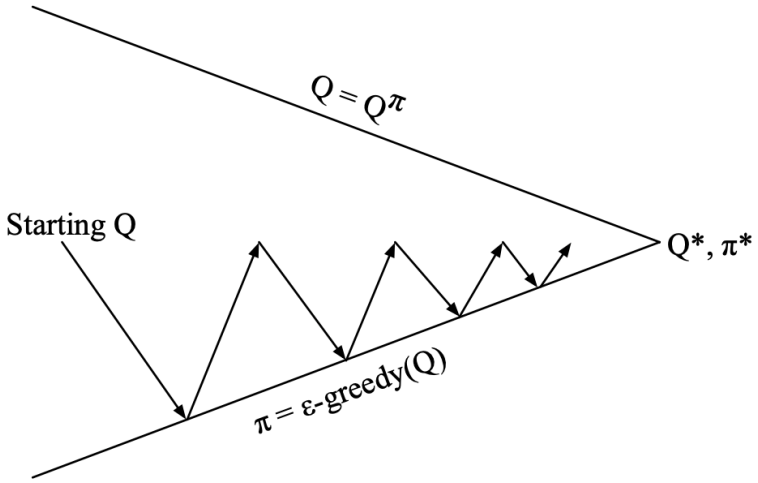
- **First visit Monte-Carlo:** Average return following first visit to state-action pair
  - Estimate $Q_\pi(s^{(1)}, a^{(1)})$

$$Q_\pi(s^{(1)}, a^{(1)}) = \frac{\overbrace{(+2+1)}^{1.} + \overbrace{(+3+1+1)}^{2.} + \overbrace{(-2)}^{3.}}{3} = 2$$

Monte-Carlo Policy Iteration

- The general idea of **policy iteration** can be applied:

$$\pi_0 \xrightarrow{E} Q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} Q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots$$



- **Policy Evaluation**: Estimate Q_{π_i} with Monte-Carlo sampling
- **Policy Improvement**: Generate new policy $\pi_{i+1} \geq \pi_i$ with ϵ-greedy

Outlook: GLIE

Definition **Greedy in the Limit with Infinite Exploration** (GLIE)

- All state-action pairs are explored infinitely many times
- The policy converges to a greedy policy

- For example, ϵ-greedy with $\epsilon_i = \frac{1}{i}$ → ϵ_i decreases each iteration and converges to 0

Theorem

GLIE model-free control converges to the optimal action value function q^*

Limitations of Monte-Carlo Methods

- Sampled returns are very noisy! → **high variance**
  - Stochasticity induced by policy and dynamics
  - Sum of many steps and every step induces noise
- Many samples are necessary for a good estimate → **sample inefficient**
- We have to wait until the episode is over

Outline

- Recap of RL Basics
- Taxonomy of Reinforcement Learning
- Value-Based RL
  - Monte-Carlo Methods
  - Temporal-Difference (TD) Learning
  - n-step Bootstrapping
  - Deep Q-Learning Network (DQN)
- Summary

Temporal-Difference (TD) Learning

- TD Learning is a combination of Monte-Carlo (MC) ideas and Dynamic Programming (DP) ideas
  - can learn directly from raw experience without a model of the environment's dynamics like Monte-Carlo methods

    → **Model-free by sampling**

  - updates estimates based on other learned estimates like dynamic programming

    → **Bootstrapping**

TD Evaluation

- Recall the recursive Bellman equation

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

- TD learning uses this idea, but approximates the expectation with samples
- For a transition $(s_i, a_i, r_i', s_i', a_i')$ sampled according to policy π, a **target** T_i can be defined:

$$T_i = r_i' + \gamma\, Q_\pi(s_i', a_i')$$

- We do not estimate Q_π as an average of targets, but as a **moving average** with α specifying the impact of the new target T_i compared to the old estimate Q_π

$$Q_\pi(s_i, a_i) \leftarrow (1-\alpha)\, Q_\pi(s_i, a_i) + \alpha\, \underbrace{[r_i' + \gamma\, Q_\pi(s_i', a_i')]}_{T_i}$$

TD Evaluation

$$Q_\pi(s_i, a_i) \leftarrow (1-\alpha)\, Q_\pi(s_i, a_i) + \alpha\, [r_i' + \gamma\, Q_\pi(s_i', a_i')]$$

- This equation can be transformed to

$$Q_\pi(s_i, a_i) \leftarrow Q_\pi(s_i, a_i) + \alpha [r_i' + \gamma\, Q_\pi(s_i', a_i') - Q_\pi(s_i, a_i)]$$

- Note that the term in brackets is a sort of error, measuring the difference between the target and the current estimate.
- This difference is called **Temporal Difference (TD) error** δ_i

$$\delta_i = \underbrace{r_i' + \gamma\, Q_\pi(s_i', a_i')}_{T_i} - Q_\pi(s_i, a_i)$$

- The update rule can thus be interpreted as minimizing the TD error with a “step size” α

SARSA - (State, Action, Reward, State, Action)

- On-policy TD control 更新目标与当下状态有关，模型只能在执行当前策略的情况下逐步改善

Algorithm

Initialize $Q_\pi(s, a)$ arbitrarily for each $s \in \mathcal{S}$ and $a \in \mathcal{A}$

Repeat for each episode:

  Initialize state s

  Choose action a from s using a policy derived from Q_π (e.g., ϵ-greedy)

  Repeat for each step of the episode:

    Take action a and observe reward r' and next state s'

    Choose action a' from s' using a policy derived from Q_π (e.g., **ϵ-greedy**)

    $Q_\pi(s, a) \leftarrow Q_\pi(s, a) + \alpha[r' + \gamma\, Q_\pi(s', a') - Q_\pi(s, a)]$

    $s \leftarrow s'$, $a \leftarrow a'$

- The algorithm is **on-policy** since the target depends on the next action a' that must be chosen based on the current policy π.

Q-Learning

action value

- Q-Learning is a sample-based version of Q-Value Iteration
- Recall the recursive **Bellman optimality equation**:

$$q^*(s,a) = \mathbb{E}_{\tau\sim\pi}\left[R_{t+1} + \gamma \max_{a'\in\mathcal{A}} q^*(S_{t+1}, a') \mid S_t = s, A_t = a\right]$$

- Q-Learning uses this idea, but approximates the expectation with samples
- For a transition (s_i, a_i, r_i', s_i') sampled according to policy π, a **target** T_i can be defined:

$$T_i = r_i' + \gamma \max_{a'\in\mathcal{A}} Q_\pi(s_i', a')$$

Q-Learning

- The Q-value can be updated by reducing the **Temporal Difference (TD) error** δ_i

$$Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha\Big[\underbrace{r' + \gamma \max_{a' \in \mathcal{A}} Q_\pi(s',a')}_{T_i} - Q_\pi(s,a)\Big]$$

δ_i

- Remark: The target and the TD error are defined slightly differently in different contexts (cf. TD evaluation)

Q-Learning

- Off-policy TD control

Algorithm

Initialize $Q_\pi(s, a)$ arbitrarily for each $s \in \mathcal{S}$ and $a \in \mathcal{A}$

Repeat for each episode:

  Initialize state s

  Repeat for each step of the episode:

    Choose action a from s using a policy derived from Q_π (e.g., ϵ-greedy)

    Take action a and observe reward r' and next state s'

    $Q_\pi(s, a) \leftarrow Q_\pi(s, a) + \alpha\left[r' + \gamma \max_{a' \in \mathcal{A}} Q_\pi(s', a') - Q_\pi(s, a)\right]$

    $s \leftarrow s'$

- The algorithm is **off-policy** since the target only depends on the maximum of the old estimates and not directly on the current policy π.

Pros and Cons of TD-Learning

- **Pros:**
  - TD is model-free (no knowledge of MDP) and learns directly from experience
  - TD can learn from incomplete episodes by bootstrapping
  - TD can learn during each episode
  - Low variance of estimates since only one time step is sampled in contrast to Monte-Carlo methods

- **Cons:**
  - High bias of estimates is possible since the estimates strongly depend on old estimates which can be inaccurate

Bootstrapping und Sampling

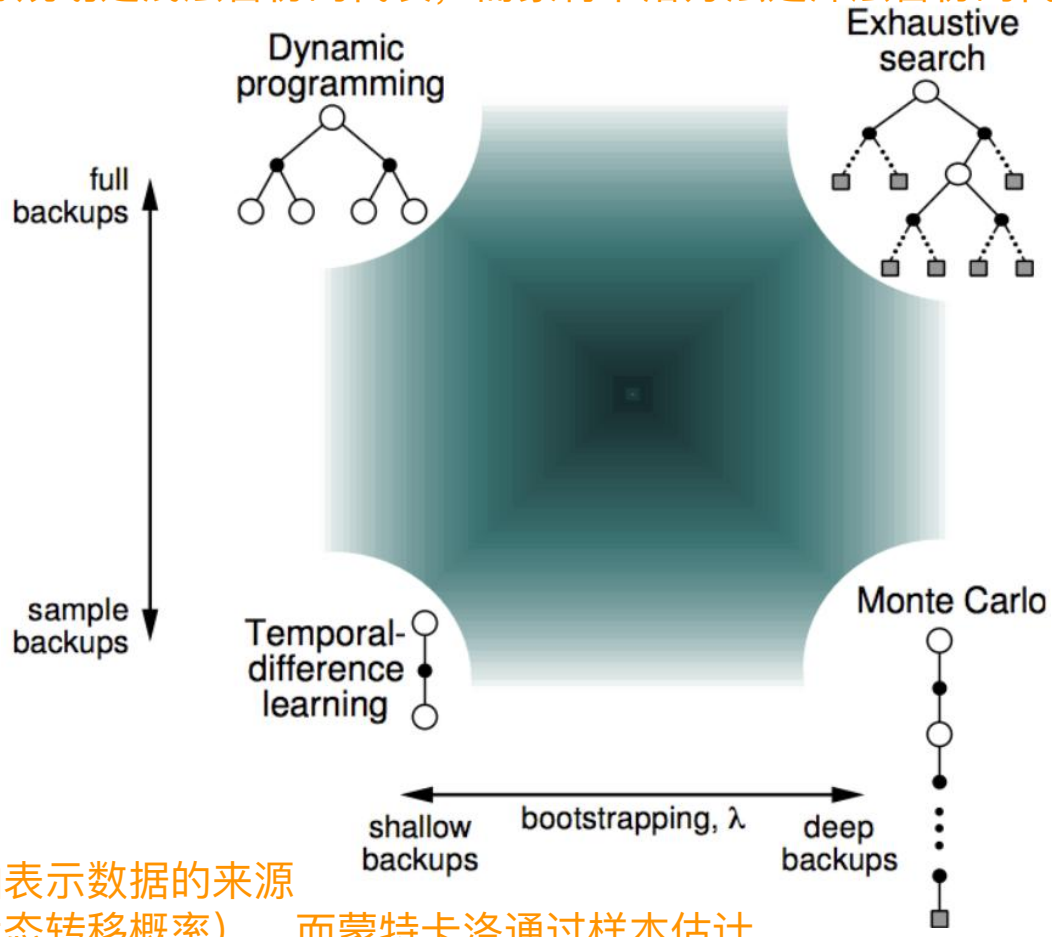
Bootstrapping (shallow backups)
Update involves an estimate

- Dynamic Programming
- Temporal-Difference Learning

Sampling (sample backups)
Update samples an expectation

- Monte-Carlo Methods
- Temporal-Difference Learning

横轴表示备份的深度
动态规划是浅层备份的代表，而蒙特卡洛方法是深层备份的代表



纵轴表示数据的来源
动态规划需要完全的信息（如状态转移概率），而蒙特卡洛通过样本估计。

Silver 2015

Outline

- Recap of RL Basics
- Taxonomy of Reinforcement Learning
- Value-Based RL
  - Monte-Carlo Methods
  - Temporal-Difference (TD) Learning
  - n-step Bootstrapping
  - Deep Q-Network (DQN)
- Summary

n-step Bootstrapping

- Combines Monte-Carlo sampling and temporal-difference

$$\sum_{k=0}^{T-1} \gamma^k \, r_{t+1+k} \qquad\qquad r_{t+1} + \gamma \, Q_\pi(s_{t+1}, a_{t+1})$$

- The **n-step estimate** contains n sampled rewards plus the value function for the successive future

$$\sum_{k=0}^{n-1} \gamma^k \, r_{t+1+k} + \gamma^n \, Q_\pi(s_n, a_n)$$

- This n-step estimate can also be used as a **target** for the previously described algorithms.

n-step Bootstrapping

- Monte-Carlo: $n \to \infty$
- Temporal-Difference: $n = 1$

$$\sum_{k=0}^{n-1} \gamma^k \, r_{t+1+k} + \gamma^n \, Q_\pi(s_n, a_n)$$

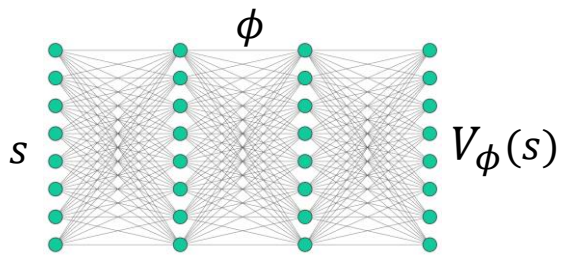
- The parameter n **balances the advantages and disadvantages** of Monte-Carlo and Temporal-Difference:
  - $n \uparrow \Rightarrow$ No bias and high variance
  - $n \downarrow \Rightarrow$ High bias and low variance

Outline

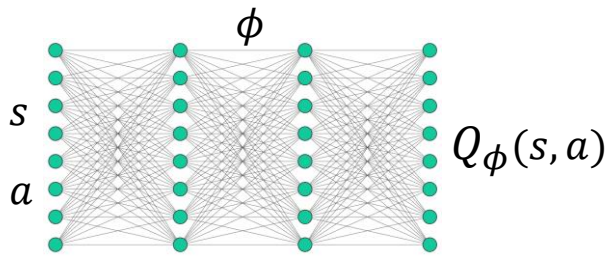
- Recap of RL Basics
- Taxonomy of Reinforcement Learning
- Value-Based RL
  - Monte-Carlo Methods
  - Temporal-Difference (TD) Learning
  - n-step Bootstrapping
  - Deep Q-Network (DQN)
- Summary

RL with Function Approximation – Deep RL

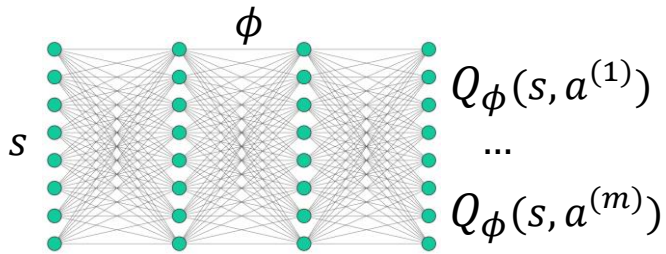
- So far, we only considered the **tabular setting** where all values are stored in a table



- **Problem:**
  - RL is often applied in more complex problems
    - Game Go: 10^{170} states
    - Controlling a robot: continuous state and action spaces
  - Impossible to store all values in a table

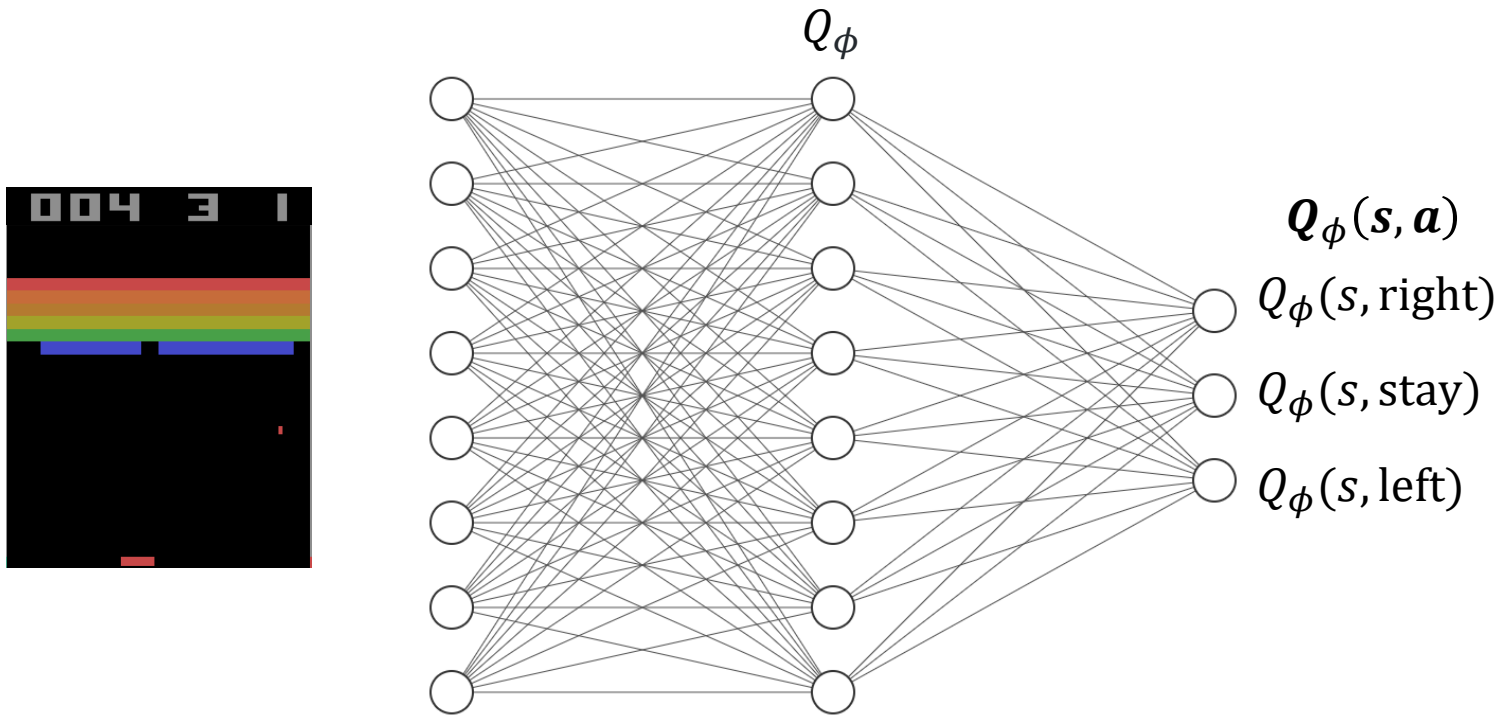


- **Solution:**
  - The value functions are **approximated** and can hence **interpolate** between different inputs
  - We call it **Deep RL** if **neural nets** are used for the function approximation in Reinforcement Learning



Deep Q-Learning

- Q-Function is approximated by a neural net with parameters ϕ



Deep Q-Learning

- **Q-Learning** algorithm as previously described but with a **neural net** for the approximation
- For a transition (s_i, a_i, r_i', s_i'), define the **target** as $T_i = r_i' + \gamma \max_{a' \in \mathcal{A}} Q_\phi(s_i', a')$
- **Idea**: Minimize the **Temporal Difference (TD) error** between Q_ϕ and the targets T_i
- Gather transitions and minimize the **loss function** by **gradient descent**:

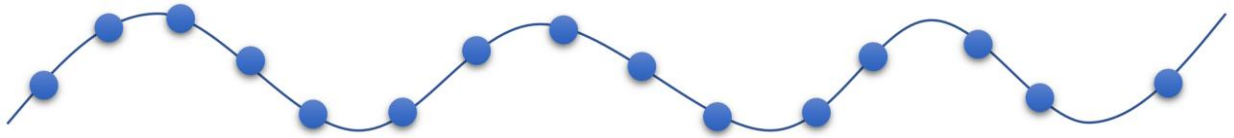
$$\mathcal{L}(\phi) = \sum_i \frac{1}{2} \underbrace{\left\| T_i - Q_\phi(s_i, a_i) \right\|^2}_{\text{TD error } \delta_i}$$

Problems of Deep Q-Learning

- We cover 3 main problems of Deep Q-Learning in this lecture:
  - Catastrophic forgetting
  - Changing target values
  - Overestimation of Q-values

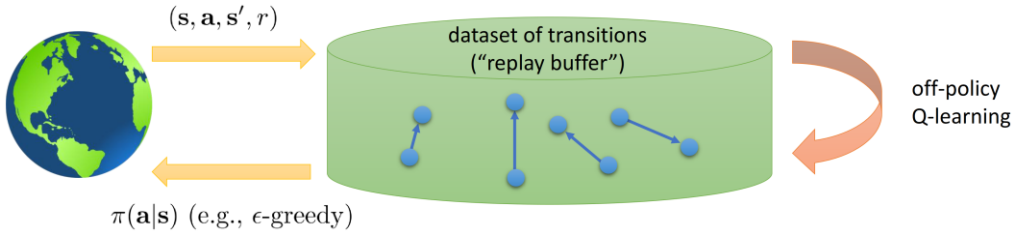
Problem: Catastrophic Forgetting

- Sequential states are strongly correlated



- The Q-values will be updated many times for similar current states, but the Q-values for states experienced in the past will be forgotten → catastrophic forgetting

- **<u>Solution:</u> Replay Buffers** 经验回放
  - Store the transitions in a replay buffer
  - Sample uncorrelated transitions from the replay buffer for the off-policy Q-Learning → “experience replay”



Levine 2022

Outlook in APPENDIX:
“prioritized experience replay”

在优先回放中，会根据样本的重要性（如 TD-error）分配更高的采样概率，从而更快收敛。

Deep Q-Learning with Replay Buffer

Algorithm

1. Collect dataset $\{(s_i, a_i, r_i', s_i')\}$ using some policy and add it to replay buffer D

$K \times$

2. Sample a batch (s_i, a_i, r_i', s_i') from D
3. Compute targets $T_i \leftarrow r_i' + \gamma \max_{a' \in \mathcal{A}} Q_\phi(s_i', a')$
4. $\phi \leftarrow \phi - \alpha \nabla_\phi \left(\sum_i \frac{1}{2} \left\| T_i - Q_\phi(s_i, a_i) \right\|^2 \right)$

- Remarks:
  - With the TD error $\delta_i = T_i - Q_\phi(s_i, a_i)$, the gradient can be simplified:

$$\nabla_\phi \left(\sum_i \frac{1}{2} \left\| T_i - Q_\phi(s_i, a_i) \right\|^2 \right) = \sum_i \delta_i \left(-\nabla_\phi Q_\phi(s_i, a_i) \right)$$

  - The update rule can hence also be formulated as $\phi \leftarrow \phi + \alpha \sum_i \delta_i \nabla_\phi Q_\phi(s_i, a_i)$

Problem: Changing Target Values

- The target value T_i is dependent on the parameters ϕ

$$T_i = r_i' + \gamma \max_{a' \in \mathcal{A}} Q_\phi(s_i', a')$$

- When the parameters ϕ are updated, the target values also change

→ This leads to an **instable training**

- **<u>Solution:</u> Target networks**

Target Network

- The target is not computed with the Q-values of the current net, but with a **less frequently updated copy** of the net
- The target network uses the parameters $\boldsymbol{\phi'}$: $T_i \leftarrow r_i' + \gamma \max_{a' \in \mathcal{A}} Q_{\boldsymbol{\phi'}}(s_i', a')$
- Periodic update of the target network parameters: $\boldsymbol{\phi'} \leftarrow \phi$
- More stable training:
  - Breaks the correlation of Q-value und target
  - Results in a more static distribution of labels

延迟更新，静态标签分布，减少梯度更新震荡，简介解耦目标和当前网络

Q-Learning with Replay Buffer and Target Network

Algorithm

1. Save target network parameters $\boldsymbol{\phi'} \leftarrow \phi$

$N \times$

2. Collect dataset $\{(s_i, a_i, r_i', s_i')\}$ using some policy and add it to replay buffer D

$K \times$

3. Sample a batch (s_i, a_i, r_i', s_i') from D

4. Compute targets $T_i \leftarrow r_i' + \gamma \max_{a' \in \mathcal{A}} Q_{\boldsymbol{\phi'}}(s_i', a')$

5. $\phi \leftarrow \phi - \alpha \nabla_\phi \left(\sum_i \frac{1}{2} \left\| T_i - Q_\phi(s_i, a_i) \right\|^2 \right)$

Deep Q-Network (DQN)

- „Classic“ deep Q-Learning
- Deep Q-Learning with replay buffer and target network, and experience collection after each update ($\boldsymbol{K = 1}$)

这里k=1，相当于实时更新，相当于并没有使用到“target network“（其核心为：延迟更新）

Algorithm

1. Take some action a_i, observe (s_i, a_i, r_i', s_i') and add it to replay buffer D
2. Sample a batch (s_i, a_i, r_i', s_i') from D uniformly
3. Compute targets $T_i \leftarrow r_i' + \gamma \max_{a' \in \mathcal{A}} Q_{\phi'}(s_i', a')$
4. $\phi \leftarrow \phi - \alpha \nabla_\phi \left(\sum_i \frac{1}{2} \left\| T_i - Q_\phi(s_i, a_i) \right\|^2 \right)$
5. Every N steps: Update target network parameters $\phi' \leftarrow \phi$

Problem: Overestimation of Q-Values

- $Q_\phi(s_i, a_i)$ typically overestimates the true expected return
- Problem is the max operator in the target: $T_i = r_i' + \gamma \max_{a' \in \mathcal{A}} Q_{\phi'}(s_i', a')$
- In general, the following inequality holds for two random variables X_1 and X_2:

$$\mathrm{E}[\max(X_1, X_2)] \geq \max(\mathrm{E}[X_1], \mathrm{E}[X_2])$$

- Since $Q_{\phi'}(s_i', a')$ is a noisy estimate (like a random variable and not an expectation), $\max_{a' \in \mathcal{A}} Q_{\phi'}(s_i', a')$ overestimates the next value
- **<u>Solution:</u> Double Q-Learning**

Double Q-Learning

- Note the equality:

$$\max_{a' \in \mathcal{A}} Q_{\phi'}(s'_i, a') = Q_{\phi'}\left(s'_i, \arg\max_{a' \in \mathcal{A}} Q_{\phi'}(s'_i, a')\right)$$

value according to $Q_{\phi'}$

action selected according to $Q_{\phi'}$

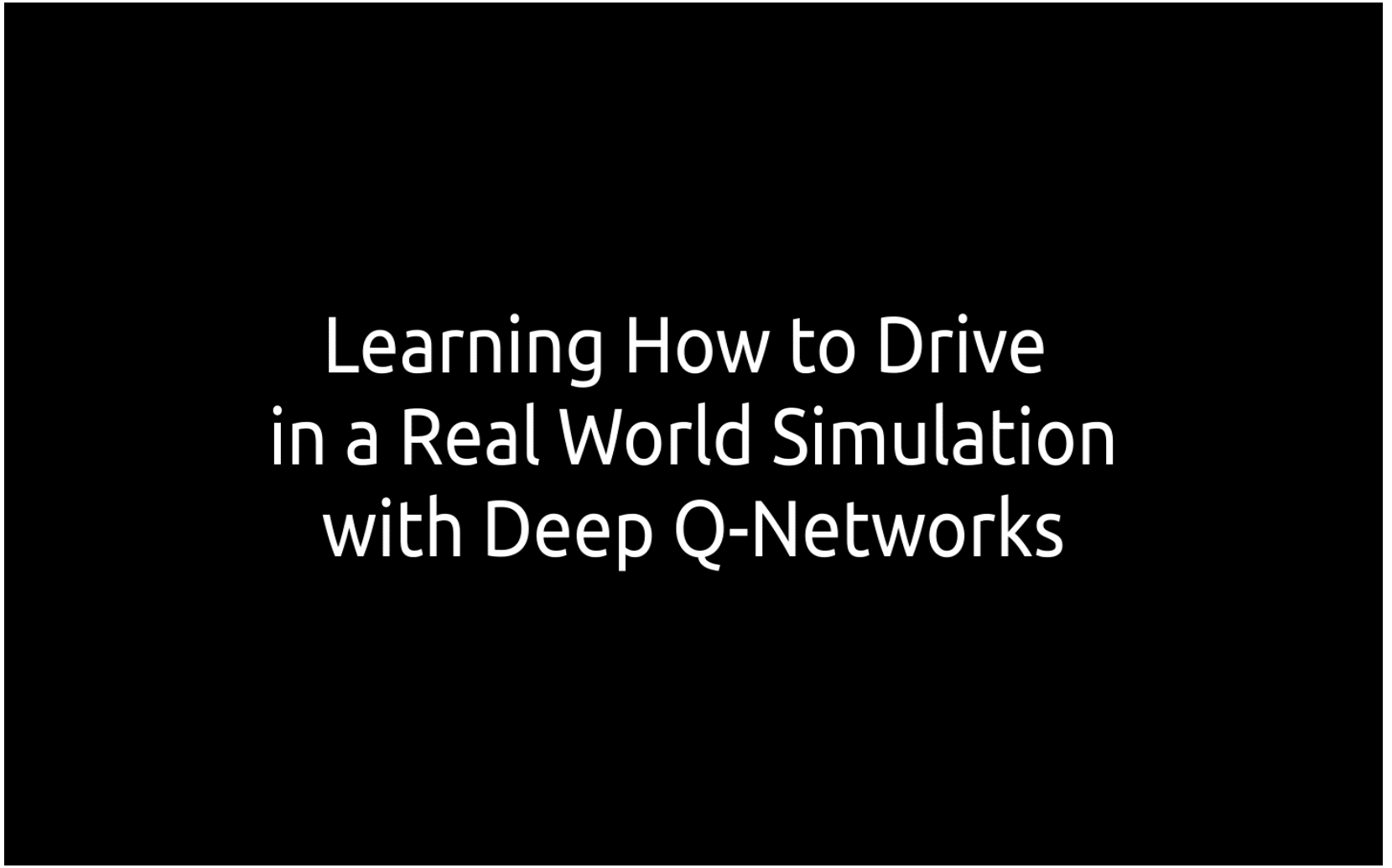
- If the noise in these components is decorrelated, the problem disappears!

- **Idea of "Double" Q-Learning:**
  - Use two different networks for action selection and value evaluation

- In practice, we can use the target network $Q_{\phi'}$ only for the **evaluation** and the current network Q_ϕ for the **action selection**:

$$T_i = r'_i + \gamma\, Q_{\phi'}\left(s'_i, \arg\max_{a' \in \mathcal{A}} Q_\phi(s'_i, a')\right)$$

Example: DQN for Autonomous Driving



Wolf et al. 2017

Outline

- Recap of RL Basics
- Taxonomy of Reinforcement Learning
- Value-Based RL
  - Monte-Carlo Methods
  - Temporal-Difference (TD) Learning
  - n-step Bootstrapping
  - Deep Q-Network (DQN)
- Summary

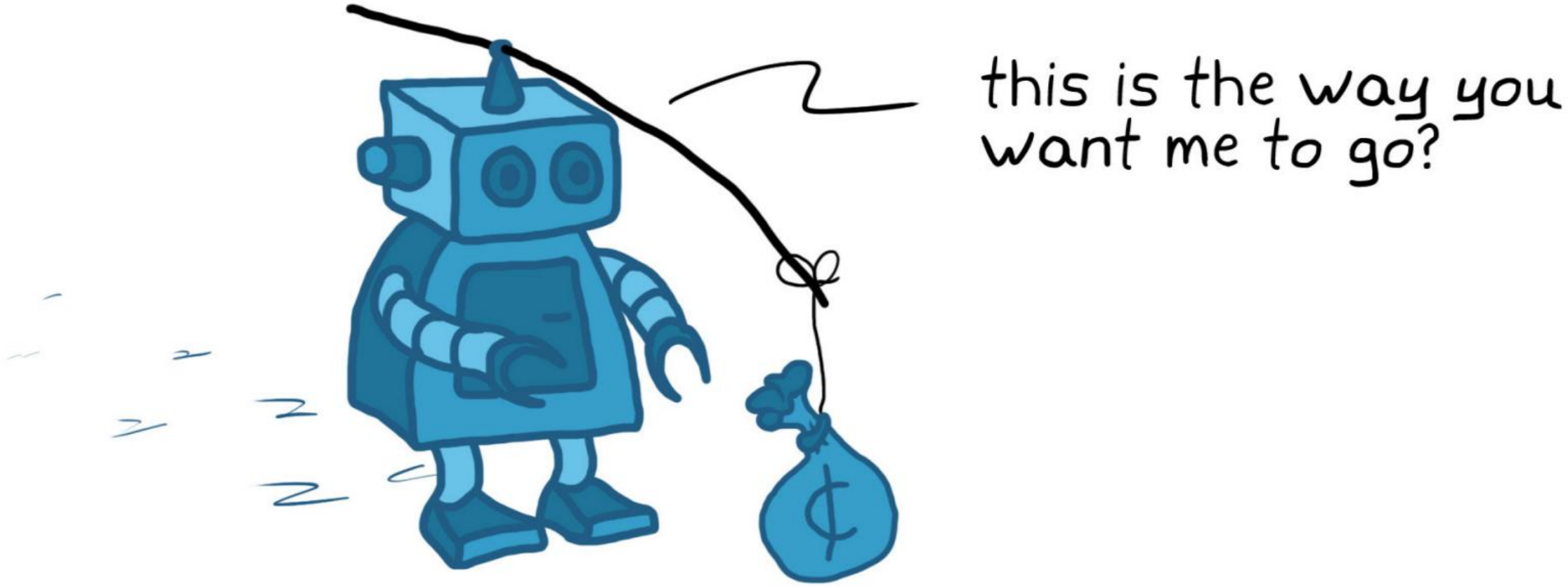
Summary

- Reinforcement Learning:
  - Model is unknown and the agent learns by interacting with the environment → sampling
- Exploration necessary: ϵ-greedy policy and exploration-exploitation tradeoff
- Value-based RL: Learn the value function and derive the policy from it
- Monte-Carlo Methods: Estimate by sampling
- Temporal-Difference (TD) Learning: Learn by bootstrapping
  - Reduce the TD error between value functions of successive states according to the Bellman equations
  - SARSA: on-policy TD control
  - Q-Learning: off-policy TD control
- n-step bootstrapping: Combination of Monte-Carlo and Temporal-Difference
- Deep Q-Learning: Use neural nets for function approximations
  - Catastrophic forgetting → replay buffer
  - Changing target values → target network

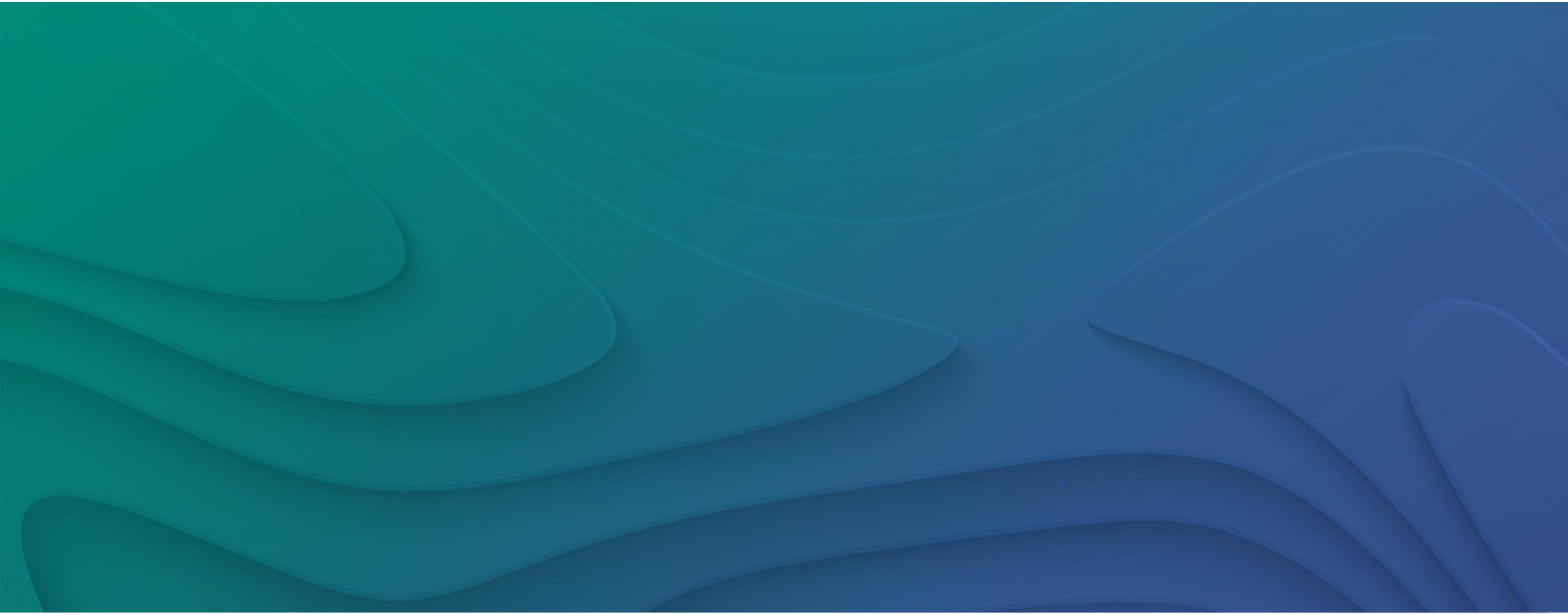
  } Deep Q-Network (DQN)
  - Overestimation of Q-values → double Q-Learning

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?

APPENDIX



Outlook: Prioritized Experience Replay

- Idea for faster training:
  - Sample transitions that promise high expected learning progress more frequently
  - The greater the TD-error for a transition, the more the model can learn from this transition

- The priority p_i of a transition i is the magnitude of the TD-error δ_i + a small number $\epsilon > 0$

$$p_i = |\delta_i| + \epsilon$$

- The probability of being sampled is ensured to be monotonic in a transition's priority, while guaranteeing a non-zero probability even for the lowest-priority transition

- The probability of sampling transition i with hyperparameter α is $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$

Outlook: Prioritized Experience Replay

- Prioritized replay introduces bias because it changes the distribution
- This bias can be corrected by using importance-sampling (IS) weights: $\mathrm{w_i} = \left(\frac{1}{N}\frac{1}{P(i)}\right)^{\beta}$
- The non-uniform probabilities $P(i)$ can fully be compensated if $\beta = 1$
- For stability reasons, the weights are always normalized by $1/\max_j w_j$
- The update rule contains now additional normalized IS-weights:

$$\phi \leftarrow \phi + \alpha \sum_i \frac{w_i}{\max_j w_j} \delta_i \nabla_\phi Q_\phi(s_i, a_i)$$