Machine Learning techniques for numerical solvers

Day 4: Reinforcement Learning

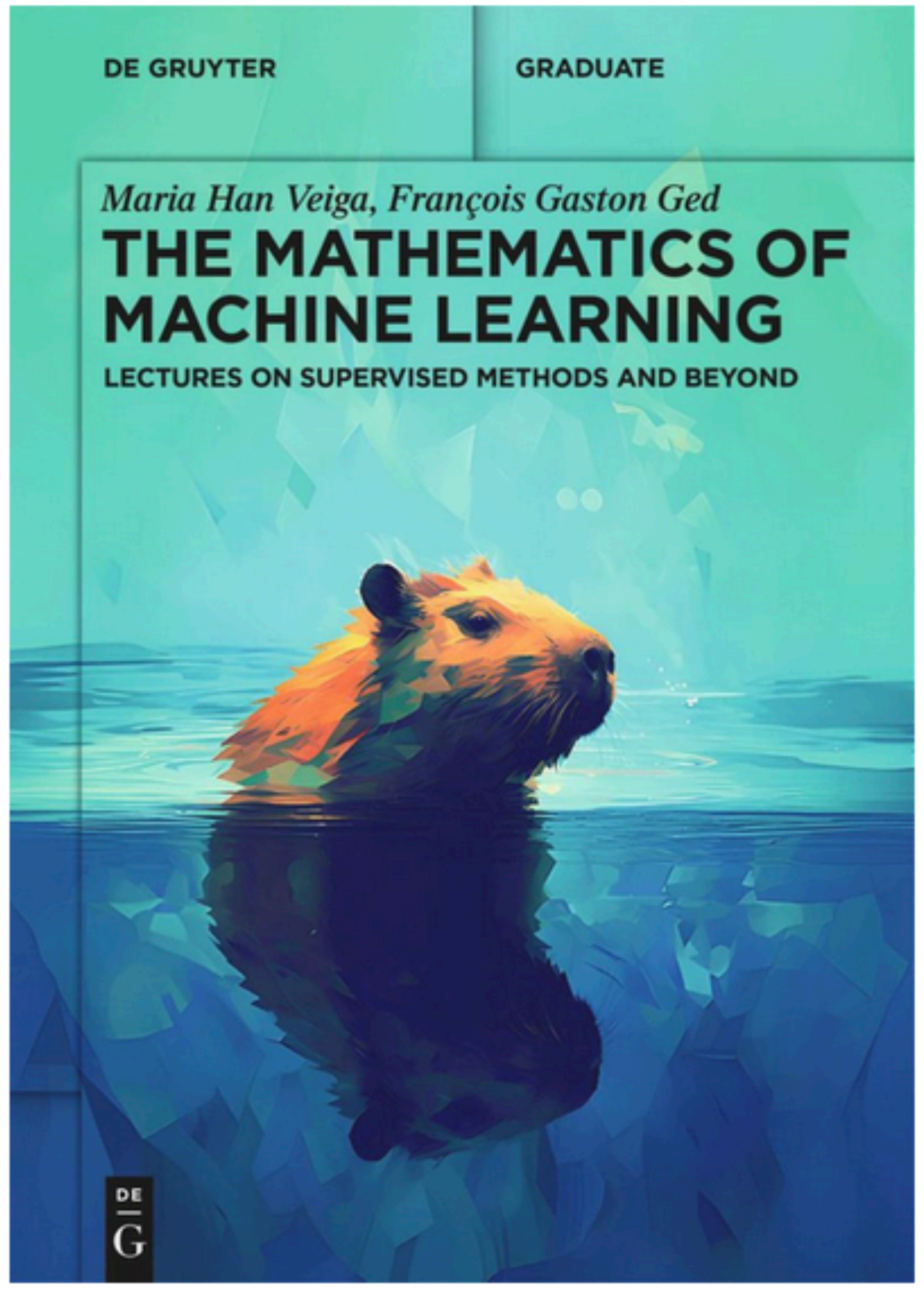
Maria Han Veiga
Mini-course SUSTech
11.03 - 14.03

THE OHIO STATE UNIVERSITY
COLLEGE OF ARTS AND SCIENCES

Schedule

- **Monday:** Introduction to Machine Learning (1h) 17:00-18:00
- **Tuesday:** Computational Framework + Supervised learning: integrating data-driven methods within a numerical solver + Hands-on session (2h) 14:00-16:00
- **Wednesday:** Unsupervised learning: Physics informed neural networks (1h) 11:00-12:00
- **Thursday:** Reinforcement Learning (1h) 11:00-12:00

A quick advertisement



Ahead of Publication Published by De Gruyter 2024

The Mathematics of Machine Learning

Lectures on Supervised Methods and Beyond

Maria Han Veiga and François Gaston Ged

In the series De Gruyter Textbook

Day 4: Reinforcement learning

Outline

- Introduction to Reinforcement Learning (RL)
  - Markov Decision Process
  - Iterative methods
  - Policy gradient method
- RL in numerics

Reinforcement Learning

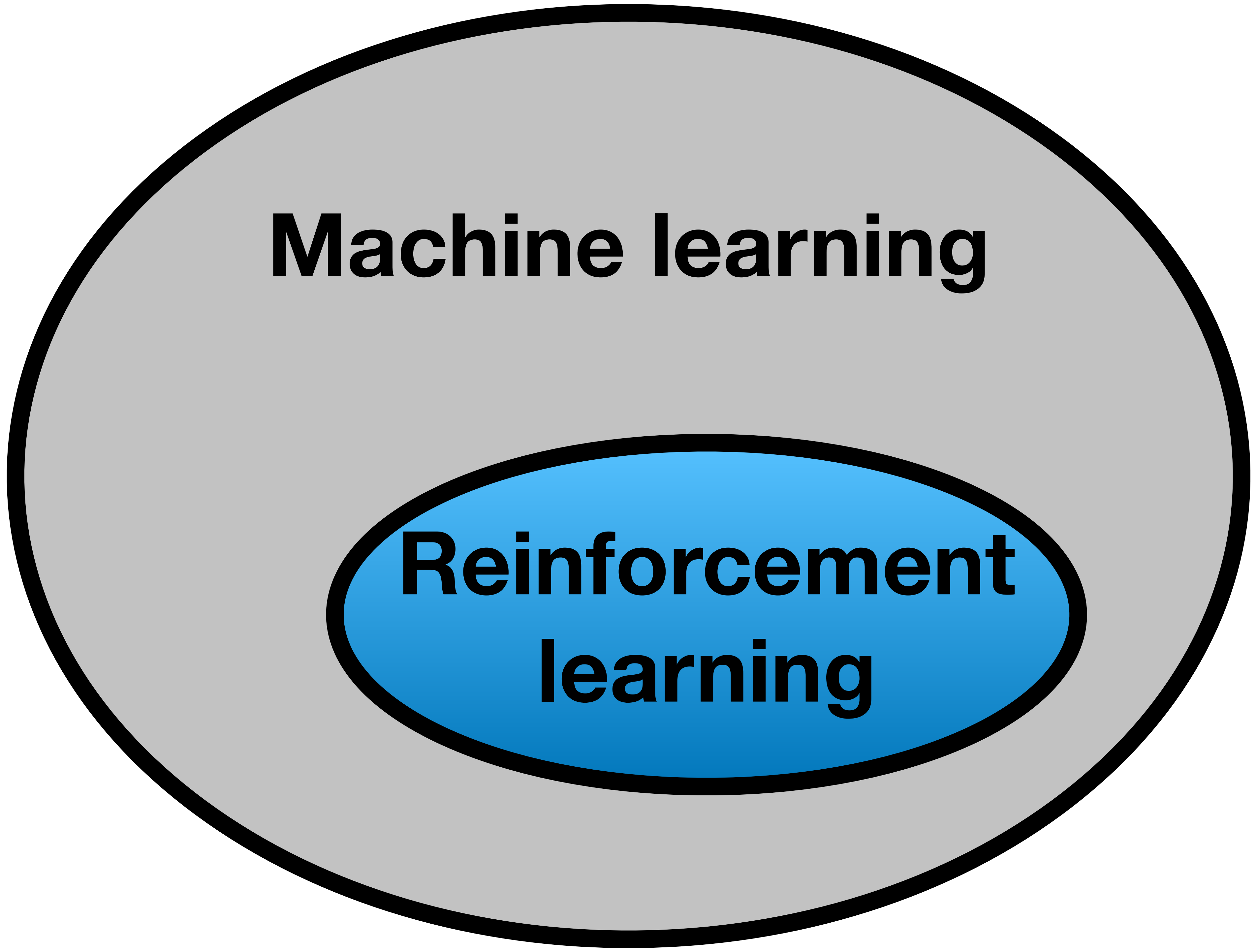
- **Task:** Consider a dynamical system that evolves over time. At a discrete time step n, the state of the system is describe by an element s_n at some state space.

- A control u_n is applied and the system then moves to a new state, $s_{n+1} = f(n, s_n, u_n)$ for some transition map f.

- For a given sequence of states and controls (trajectory) of length N, the performance on this task is measured using a performance criterion $J((s_n, u_n)_{n \leq N})$.

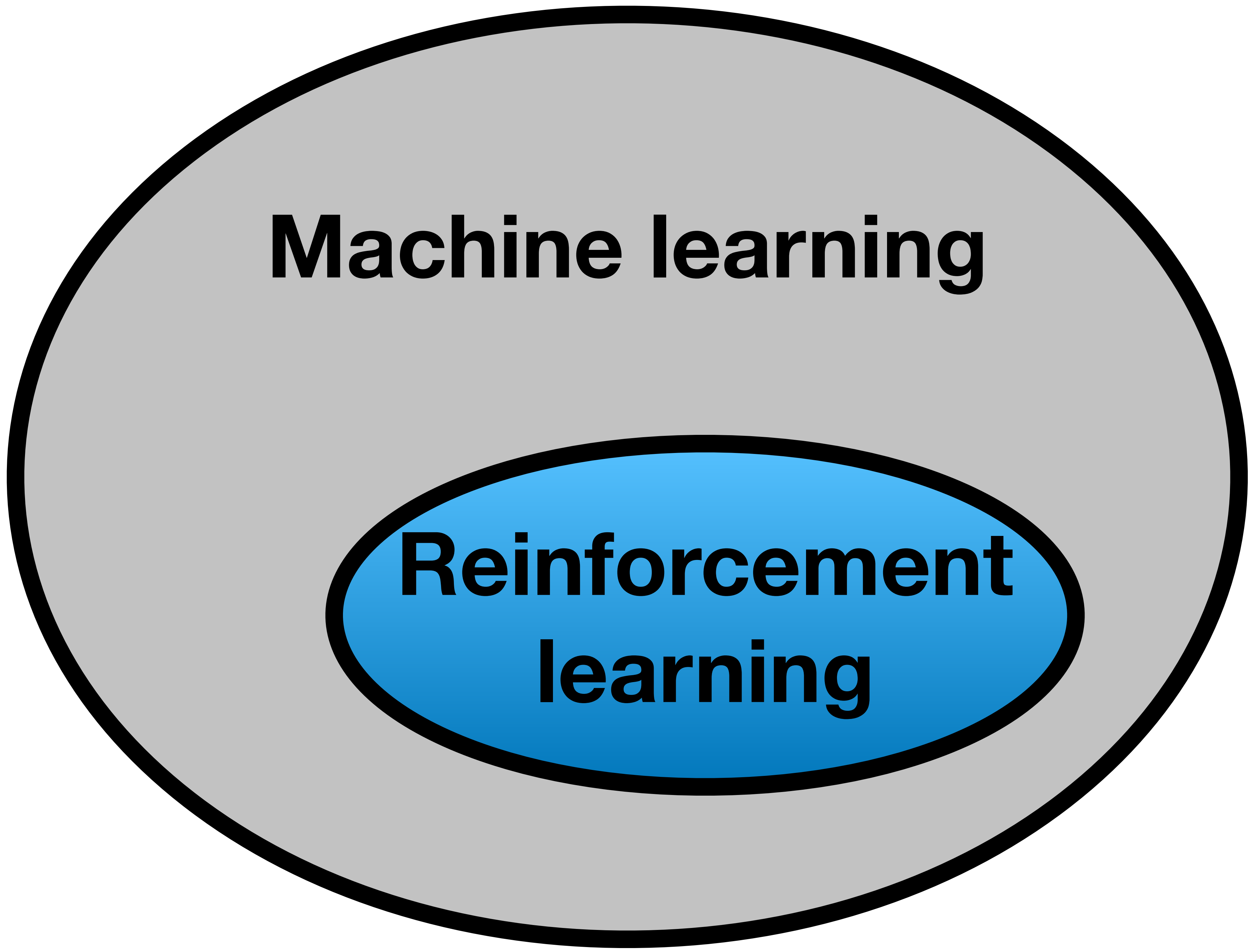
Reinforcement Learning

- Reinforcement Learning (RL) is an approach to solve such problems through trials and errors:
    - a learner (agent) samples a trajectory, collects rewards (based on performance J) and reinforces positively or negatively the chosen actions (control).



Reinforcement Learning

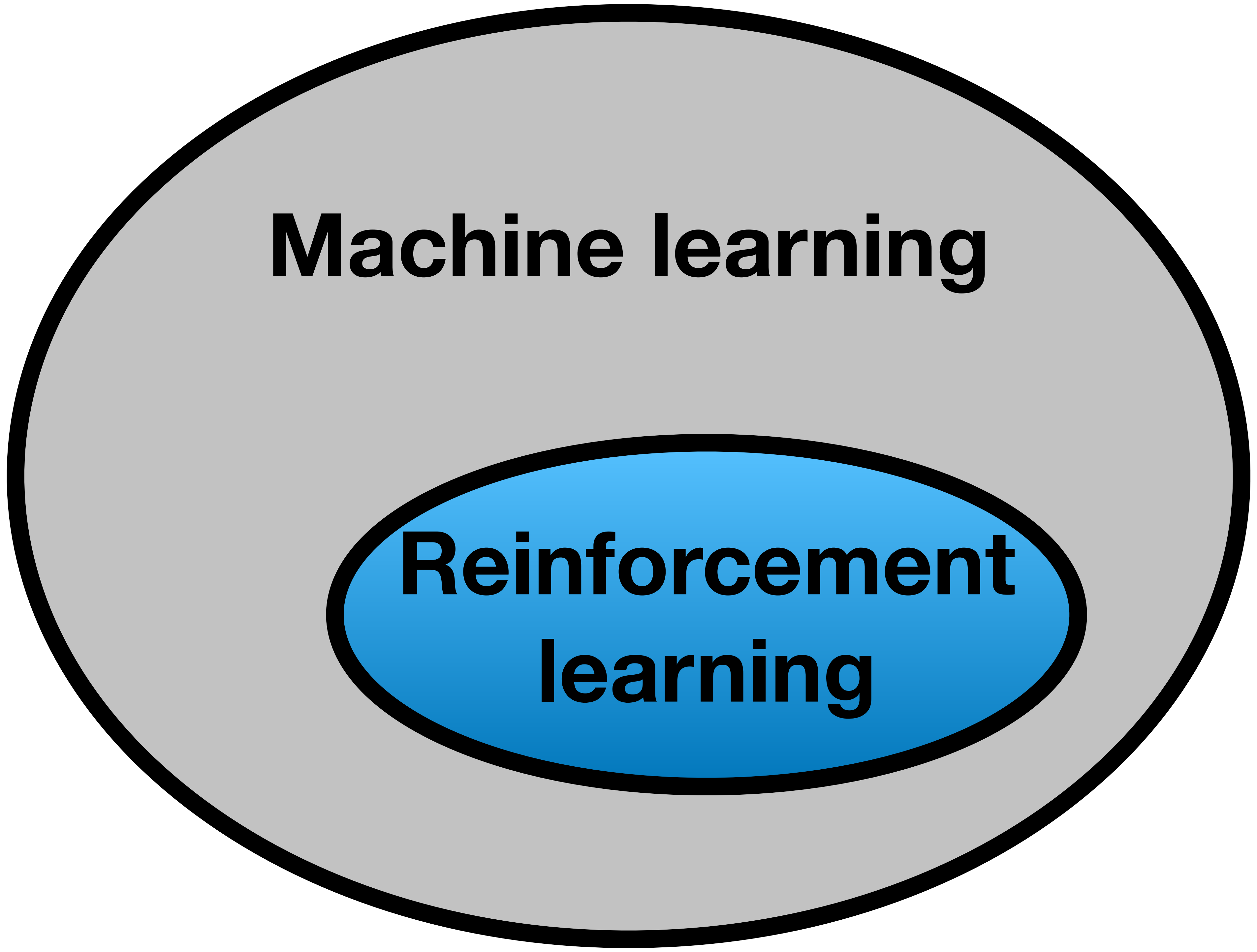
- Reinforcement Learning (RL) is an approach to solve such problems through trials and errors:
    - a learner (agent) samples a trajectory, collects rewards (based on performance J) and reinforces positively or negatively the chosen actions (control).



- **Example:** Playing a chess game. At the end of the game (a sequence of moves), we get a reward, e.g. +1 for winning, -1 for losing or 0 for drawing.

Reinforcement Learning

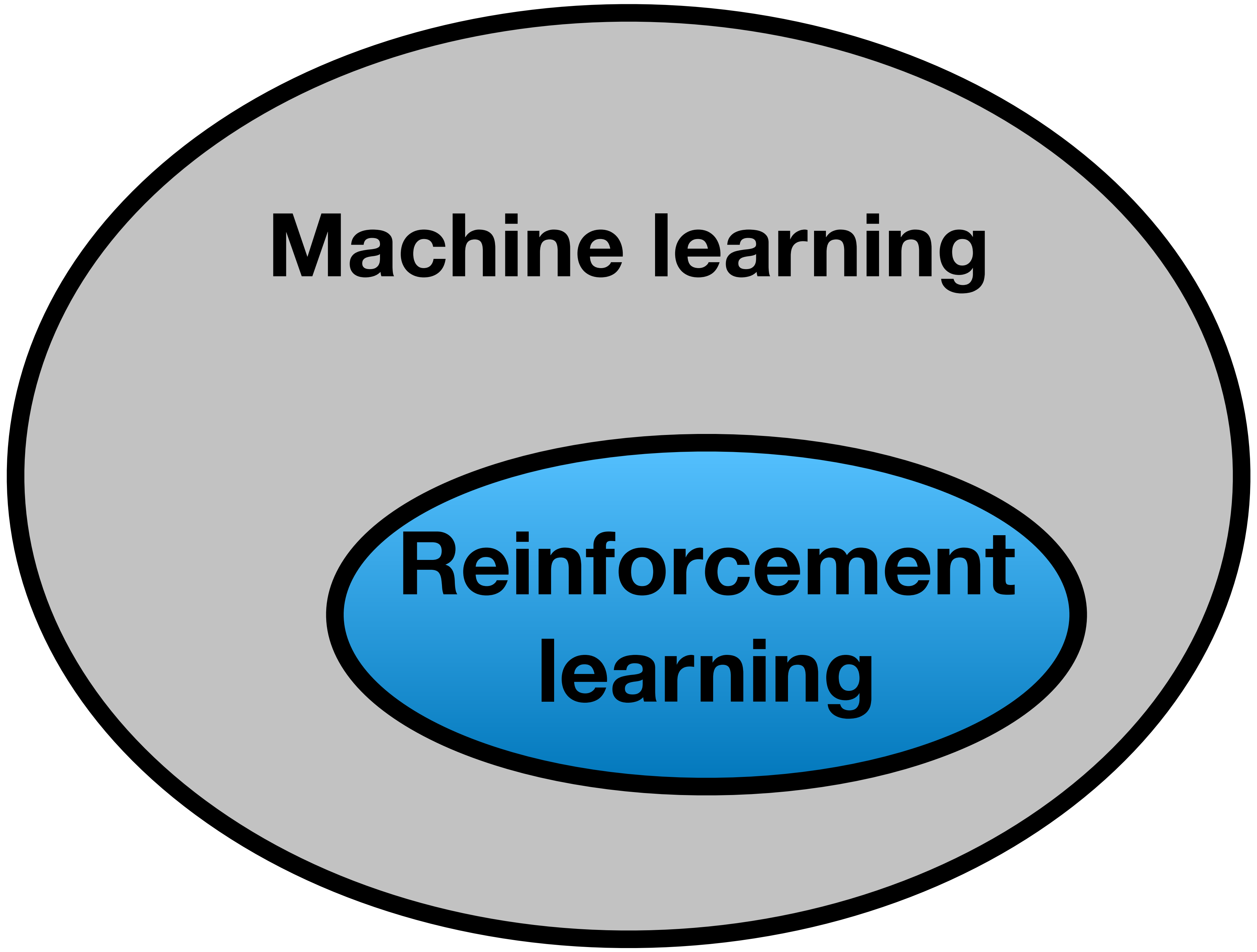
- Reinforcement Learning (RL) is an approach to solve such problems through trials and errors:
  - a learner (agent) samples a trajectory, collects rewards (based on performance J) and reinforces positively or negatively the chosen actions (control).



- **Example:** Playing a chess game. At the end of the game (a sequence of moves), we get a reward, e.g. +1 for winning, -1 for losing or 0 for drawing.
  - **Reward does not have to be differentiable!**

Reinforcement Learning

- Reinforcement Learning (RL) is an approach to solve such problems through trials and errors:
  - a learner (agent) samples a trajectory, collects rewards (based on performance J) and reinforces positively or negatively the chosen actions (control).
- Frame numerical analysis problems as RL tasks: limiting a numerical solution [1], solving PDEs [2], etc.

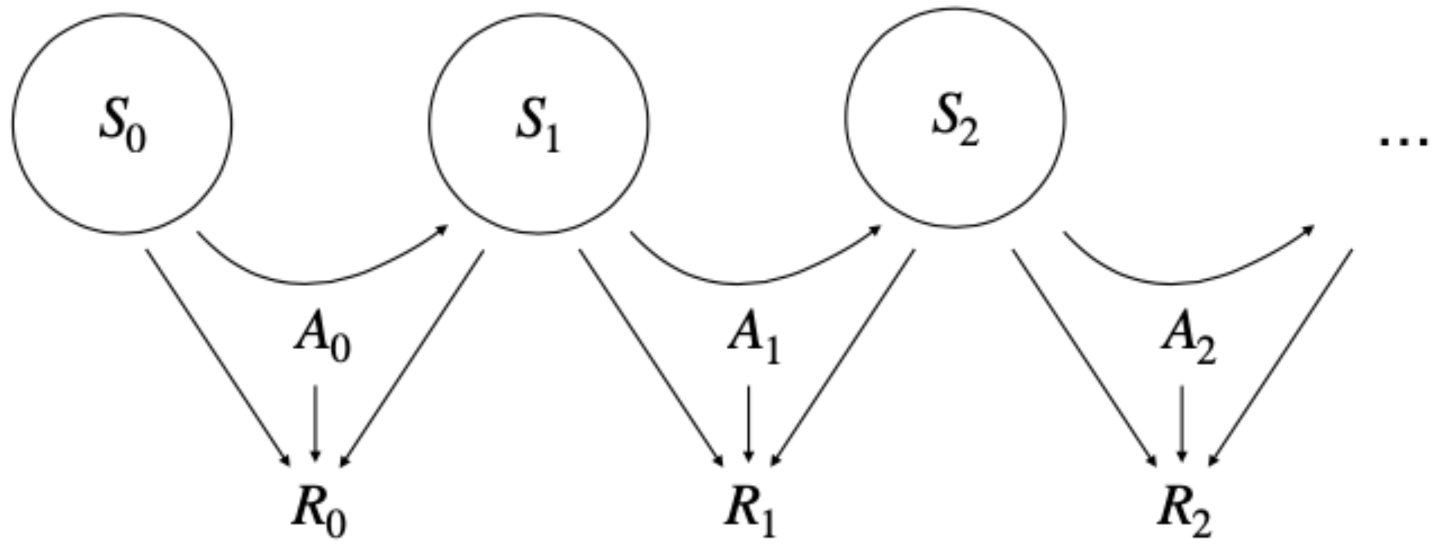


[1] "*A Reinforcement Learning Based Slope Limiter for Second-Order Finite Volume Schemes*", Schwarz et al., PAMM, 2023

[2] "*Solving high-dimensional partial differential equations using deep learning*", Han et al., PNAS, 2018

Reinforcement Learning

- Formally, assume agent interacts with the environment with a sequence of discrete time steps $t = 0,1,2,...$, with a possibly random terminal time T, or continue indefinitely.
- At each time step t, agent has a representation of the environment's sate and its own internal state, $S_t \in \mathcal{S}$. Given that state, it selects an action $A_t \in \mathcal{A}$ and arrives at some other state S_{t+1}.
- In consequence of the tuple (S_t, A_t, S_{t+1}), the agent receives a reward $R_t \in \mathbb{R}$,



Yields a trajectory $S_0, A_0, R_0, S_1, A_1, R_1, \ldots$

Reinforcement Learning

Yields a trajectory $S_0, A_0, R_0, S_1, A_1, R_1, \ldots$

At each tilmestep t, we can define the *return at time* t as the sum of the current reward and all future rewards:

$$G_t := R_t + R_{t+1} + \ldots$$

$$G_t := R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots, \gamma \in (0,1)$$

Reinforcement Learning

Yields a trajectory $S_0, A_0, R_0, S_1, A_1, R_1, \ldots$

At each tilmestep t, we can define the *return at time t* as the sum of the current reward and all future rewards:

$$G_t := R_t + R_{t+1} + \ldots$$

$$G_t := R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots, \gamma \in (0,1)$$

How to choose a good action?

Reinforcement Learning

Definition (Policy): A policy is a mapping $\pi : \mathcal{A} \times \mathcal{S} \to [0,1]$ such that

$$\pi(a \mid s) = P(A_t = a \mid S_t = s)$$

with the condition that $\sum_{a \in \mathcal{A}} \pi(a \mid s) = 1$ for all $s \in \mathcal{S}$.

Reinforcement Learning

Definition (Markov decision process): Let $(\mathcal{S}, \mathcal{A}, p, r)$ be given as:

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- p is the state transition kernel $p(s, a, s')$
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a reward function

Reinforcement Learning

Definition (Markov decision process): Let $(\mathcal{S}, \mathcal{A}, p, r)$ be given as:

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- p is the state transition kernel $p(s, a, s')$
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a reward function

Given a policy π and an initial state distribution ν on $\mathcal{S}$, the MDP is constructed recursively as follows:

- At $t = 0$, the agent is located at $S_0 \sim \nu$
- At state S_t agent takes action $A_t \sim \pi(\,\cdot \mid S_t)$, transits to state $S_{t+1} \sim p(S_t, A_t, \cdot\,)$, collects reward $R_t = r(S_t, A_t, S_{t+1})$, <u>independently from all past random variables.</u>

Reinforcement Learning

Definition (Markov decision process): Let $(\mathcal{S}, \mathcal{A}, p, r)$ be given as:

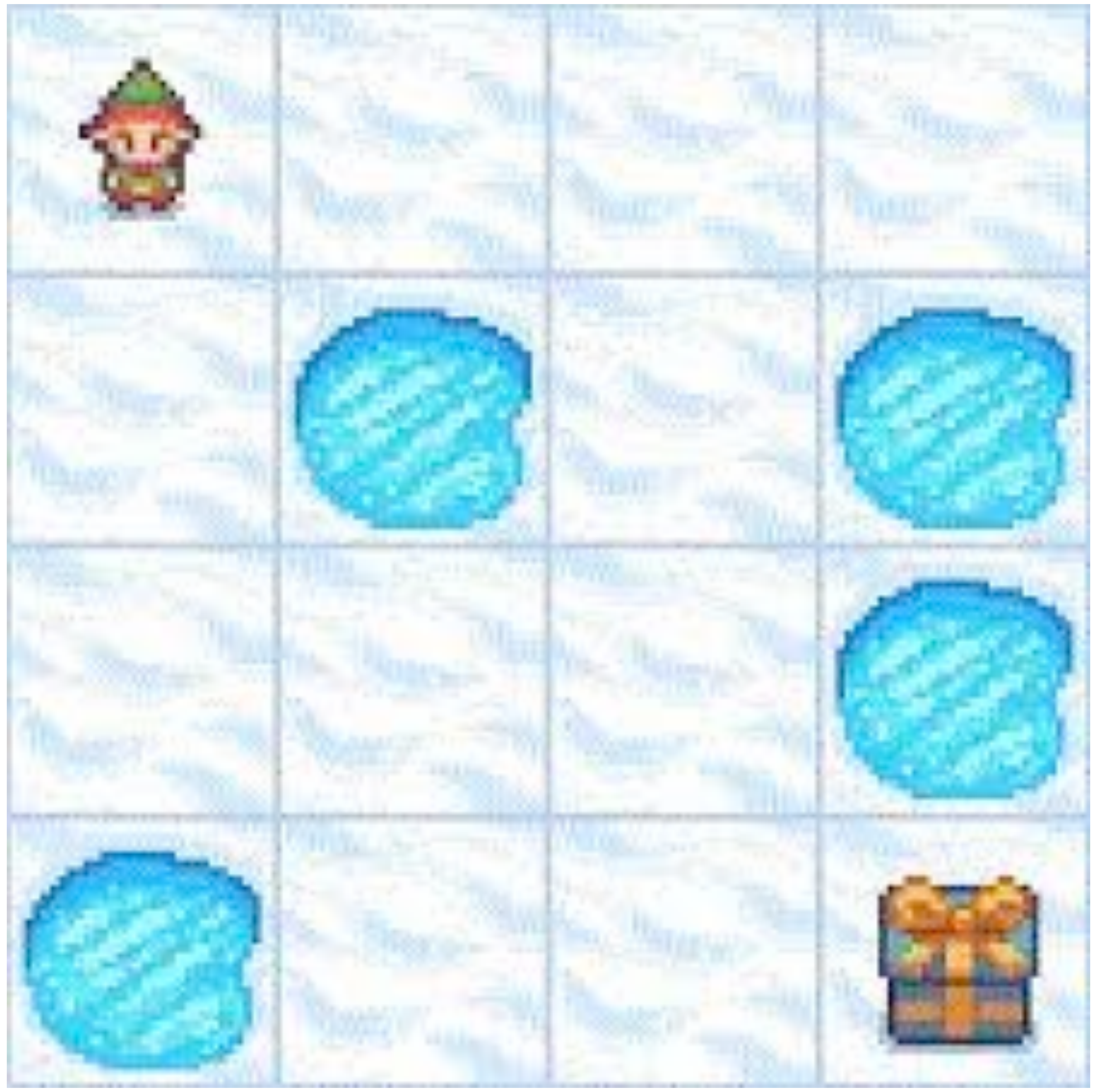
- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- p is the state transition kerne
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a rew

$P((A_{t+k}, S_{t+k+1})_{k\geq 0} \,|\, S_t)$

Given a policy π and an initial state distribution ν on $\mathcal{S}$, the MDP is constructed recursively as follows:

- At $t = 0$, the agent is located at $S_0 \sim \nu$
- At state S_t agent takes action $A_t \sim \pi(\,\cdot\,|\,S_t)$, transits to state $S_{t+1} \sim p(S_t, A_t, \,\cdot\,)$, collects reward $R_t = r(S_t, A_t, S_{t+1})$, <u>independently from all past random variables.</u>

Example



Frozen lake game:

State space: {1,2,3,…,16}

Action space: {up, down, left, right}

Reward:

- Falling into hole: -1
- Finding treasure: 1

p(s,a,s’) deterministic or stochastic

Iterative methods

How to find a good policy?

Iterative methods

How to find a good policy?

What is a *good* state or *good* action?

Iterative methods

How to find a good policy?

What is a *good* state or *good* action?

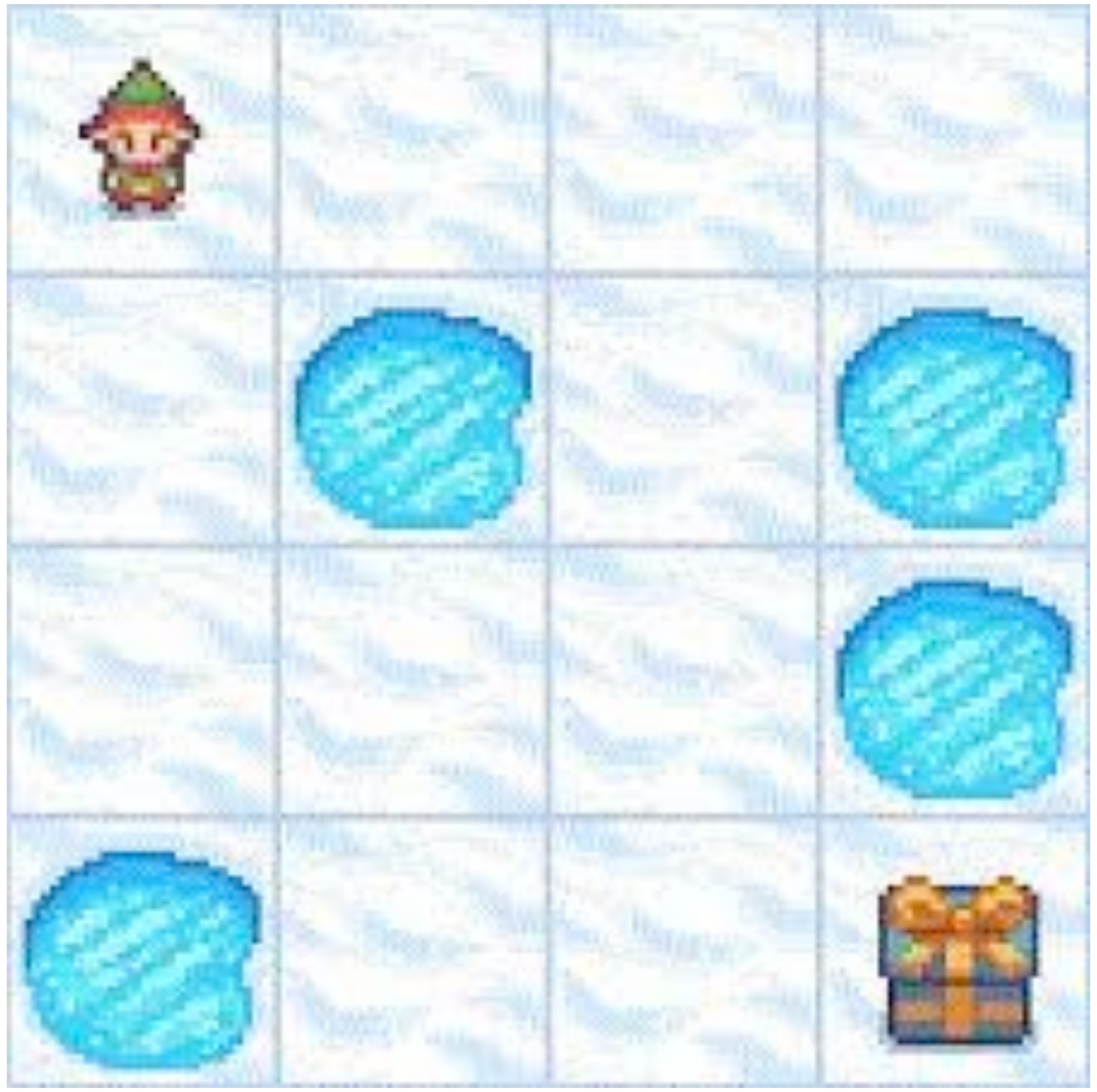
Value function: $v_\pi(s) := \mathbb{E}(G_t \mid S_t = s)$

Expected return of starting at state s and following policy π.

State-action value function: $q_\pi(s, a) := \mathbb{E}(G_t \mid S_t = s, A_t = a)$

Expected return of starting at state s and taking action a, then following policy π.

Example



Frozen lake game:

Policy: always going down

$$v(1) = \gamma^2(-1)$$

$$v(2) = (-1)$$

$$v(3) = 0$$

$$q(1,\text{'right'}) = \gamma(-1)$$

Iterative methods

One class of methods is to estimate v_π and q_π — q-learning, iterative methods.

Iterative methods

One class of methods is to estimate v_π and q_π — q-learning, iterative methods.

Theorem (Consistency condition for v_π). For any policy π and any state s, the following consistency condition holds between value function at s and the value of its possible successor states s'.

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

Iterative methods

- Suppose probability kernel $p(s, a, s')$ and reward r are known
- State space $\mathcal{S} = \{s_1, \ldots, s_n\}$ is finite
- Then, we can write:

$$\vec{v}_\pi = \overrightarrow{R} + \gamma P \vec{v}_\pi,$$

where $\vec{v}_\pi = (v_\pi(s_1), \ldots, v_\pi(s_n))$, $\overrightarrow{R}_i = \sum_{a \in \mathcal{A}} \pi(a \mid s_i) r(a, s_i)$, $P_{i,j} = \sum_{a \in \mathcal{A}} \pi(a \mid s_i) p(s_i, a, s_j)$.

Iterative methods

- Suppose probability kernel $p(s, a, s')$ and reward r are known
- State space $\mathcal{S} = \{s_1, \ldots, s_n\}$ is finite
- Then, we can write:

$$\vec{v}_\pi = \overrightarrow{R} + \gamma P \vec{v}_\pi,$$

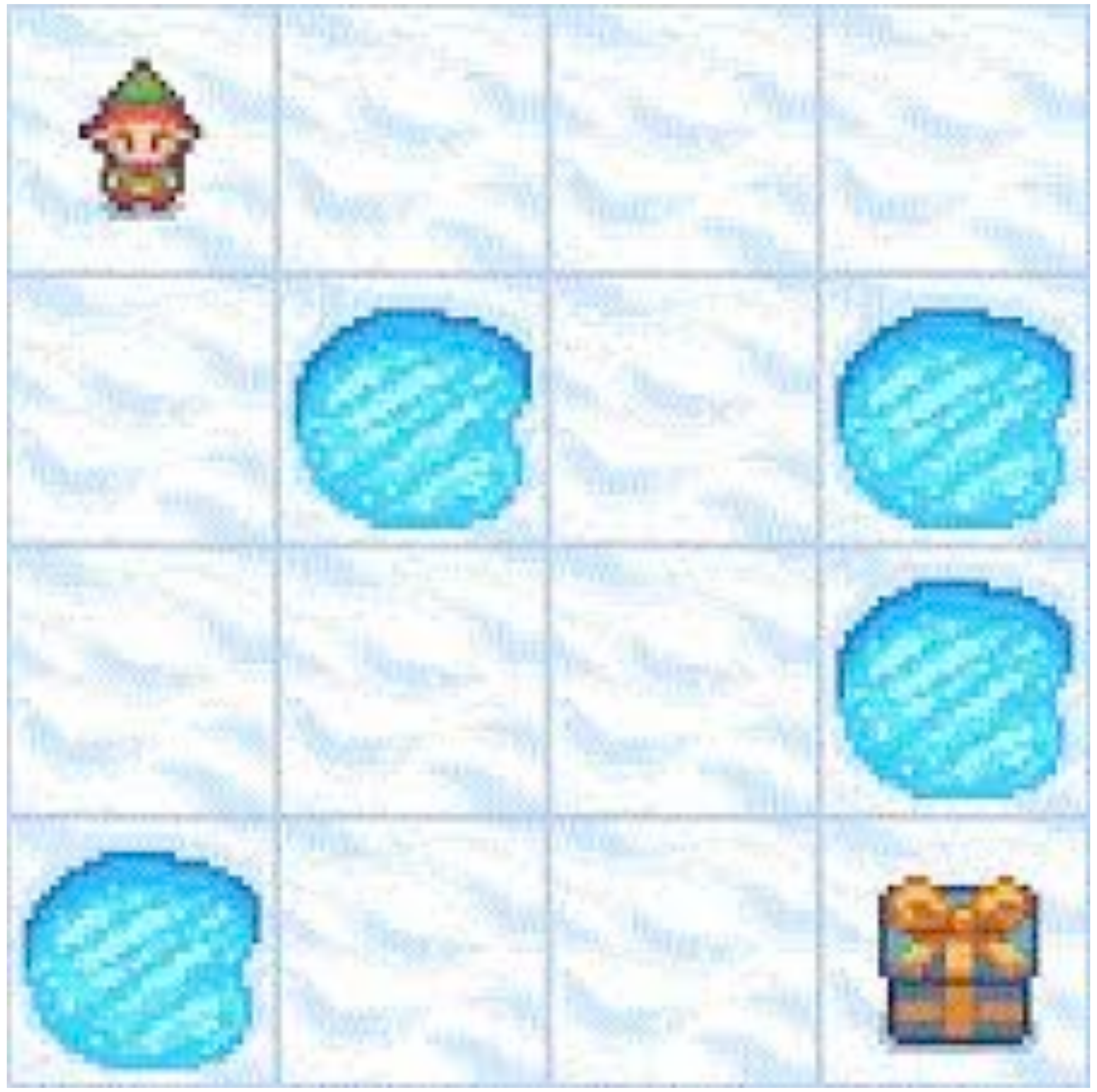
where $\vec{v}_\pi = (v_\pi(s_1), \ldots, v_\pi(s_n))$, $\overrightarrow{R}_i = \sum_{a \in \mathcal{A}} \pi(a \mid s_i) r(a, s_i)$, $P_{i,j} = \sum_{a \in \mathcal{A}} \pi(a \mid s_i) p(s_i, a, s_j)$.

- Solve this system iteratively: $\vec{v}_\pi^{(k+1)} = \overrightarrow{R} + \gamma P \vec{v}_\pi^{(k)}$
- Using Banach's fixed point theorem, the iteration converges to v_π.

Iterative methods

- Suppose probability kernel $p(s, a, s')$ and reward r are known
- State space $\mathcal{S} = \{s_1, \ldots, s_n\}$ is finite
- Given a policy π, we can find v_π.
- To improve policy, select action a at state s, and then follow $v_\pi - q_\pi(s, a)$.
- If $q_\pi(s, a) > v_\pi(s)$ for some $a \in \mathcal{A}$, it means it's better to take action a when we are at state s, which yields a modified policy.

Example



Frozen lake game:

Policy: always going down

$v(15) = 0, \quad q(15,\text{'right'}) = 1$

Modify policy: in state 15, turn right.

Iterative methods

- Suppose probability kernel $p(s, a, s')$ and reward r are known
- State space $\mathcal{S} = \{s_1, \ldots, s_n\}$ is finite
- Given a policy π, we can find v_π.
- To improve policy, select action a at state s, and then follow $v_\pi - q_\pi(s, a)$.
- If $q_\pi(s, a) > v_\pi(s)$ for some $a \in \mathcal{A}$, it means it's better to take action a when we are at state s, which yields a modified policy.

Requires knowledge of transition probability kernel and reward function

Iterative method scales with $|\mathcal{S}|$

Policy gradient method

How to find a good policy when $\mathscr{A}$ and/or $\mathscr{S}$ are very large/infinite?

And we don't know the transition probability kernel and/or reward function?

Definition (Parametric policy): Let policy be a mapping $\pi_\theta : \mathscr{A} \times \mathscr{S} \to [0,1]$ parametrised by parameters $\theta \in \mathbb{R}^p$.

Policy gradient method

How to find a good policy when $\mathscr{A}$ and/or $\mathscr{S}$ are very large/infinite?

And we don't know the transition probability kernel and/or reward function?

Definition (Parametric policy): Let policy be a mapping $\pi_\theta : \mathscr{A} \times \mathscr{S} \to [0,1]$ parametrised by parameters $\theta \in \mathbb{R}^p$.

Maximize the objective function:

$$J(\theta) = \sum_{s \in \mathscr{S}} v_{\pi_\theta}(s)\mu(s)$$

where μ is the initial state distribution.

Policy gradient method

Maximize the objective function:

$$J(\theta) = \sum_{s \in \mathcal{S}} v_{\pi_\theta}(s)\mu(s)$$

Write a gradient ascent on J:

$$\theta_{t+1} = \theta_t + \eta \nabla J(\theta).$$

Policy gradient method

Maximize the objective function:

$$J(\theta) = \sum_{s \in \mathcal{S}} v_{\pi_\theta}(s)\mu(s)$$

Write a gradient ascent on J:

$$\theta_{t+1} = \theta_t + \eta \nabla J(\theta).$$

How to compute $\nabla J(\theta)$?

Policy gradient method

Maximize the objective function:

$$J(\theta) = \sum_{s \in \mathcal{S}} v_{\pi_\theta}(s)\mu(s)$$

Write a gradient ascent on J:

$$\theta_{t+1} = \theta_t + \eta \nabla J(\theta).$$

How to compute $\nabla J(\theta)$?

$$v_\pi(s) = \mathbb{E}_\pi(G_t \,|\, S_t = s) = \mathbb{E}_\pi \left(\sum_{k \geq 0} \gamma^k r(A_k, S_k) \,|\, S_t = s \right)$$

Policy gradient method

Theorem (Policy gradient theorem). The gradient of the objective $J(\theta)$ with respect to θ is:

$$\nabla J(\theta) = \sum_{s_0 \in \mathcal{S}} \mu(s_0) \sum_{s \in \mathcal{S}} \rho_{\pi_\theta}(s_0, s) \sum_{a \in \mathcal{A}} q_{\pi_\theta}(a, s) \nabla \pi_\theta(a \mid s)$$

Policy gradient method

Theorem (Policy gradient theorem). The gradient of the objective $J(\theta)$ with respect to θ is:

$$\nabla J(\theta) = \sum_{s_0 \in \mathcal{S}} \mu(s_0) \sum_{s \in \mathcal{S}} \rho_{\pi_\theta}(s_0, s) \sum_{a \in \mathcal{A}} q_{\pi_\theta}(a, s) \nabla \pi_\theta(a \mid s)$$

The important result is that there is an analytic expression of the gradient of the objective that depends only on the gradient of the policy.

Policy gradient method

Theorem (Policy gradient theorem). The gradient of the objective $J(\theta)$ with respect to θ is:

$$\nabla J(\theta) = \sum_{s_0 \in \mathcal{S}} \mu(s_0) \sum_{s \in \mathcal{S}} \rho_{\pi_\theta}(s_0, s) \sum_{a \in \mathcal{A}} q_{\pi_\theta}(a, s) \nabla \pi_\theta(a \mid s)$$

The important result is that there is an analytic expression of the gradient of the objective that depends only on the gradient of the policy.

In practice, gradient is approximated using a Monte Carlo estimate of $q_\pi(s, a)$

Policy gradient method

$$\nabla J(\theta) = \sum_{s \in \mathcal{S}} \rho_{\pi_\theta}(s_0, s) \sum_{a \in \mathcal{A}} q_{\pi_\theta}(a, s) \nabla \pi_\theta(a \mid s)$$

$$\nabla J(\theta) = \sum_{s \in \mathcal{S}} \rho_{\pi_\theta}(s_0, s) \sum_{a \in \mathcal{A}} q_{\pi_\theta}(a, s) \nabla \pi_\theta(a \mid s) \frac{\pi_\theta(a \mid s)}{\pi_\theta(a \mid s)}$$

$$\nabla J(\theta) = \sum_{s \in \mathcal{S}} \rho_{\pi_\theta}(s_0, s) \sum_{a \in \mathcal{A}} \pi_\theta(a \mid s) q_{\pi_\theta}(a, s) \frac{\nabla \pi_\theta(a \mid s)}{\pi_\theta(a \mid s)}$$

$$\nabla J(\theta) \propto \mathbb{E}_\pi \left(q_{\pi_\theta}(a, s) \nabla \log \pi_\theta(a \mid s) \right)$$

$$\nabla J(\theta) \propto \mathbb{E}_\pi \left(G_t \nabla \log \pi_\theta(A_t \mid S_t) \right)$$

Policy gradient method

$$\nabla J(\theta) \propto \mathbb{E}_\pi \left(G_t \nabla \log \pi_\theta(A_t \mid S_t) \right)$$

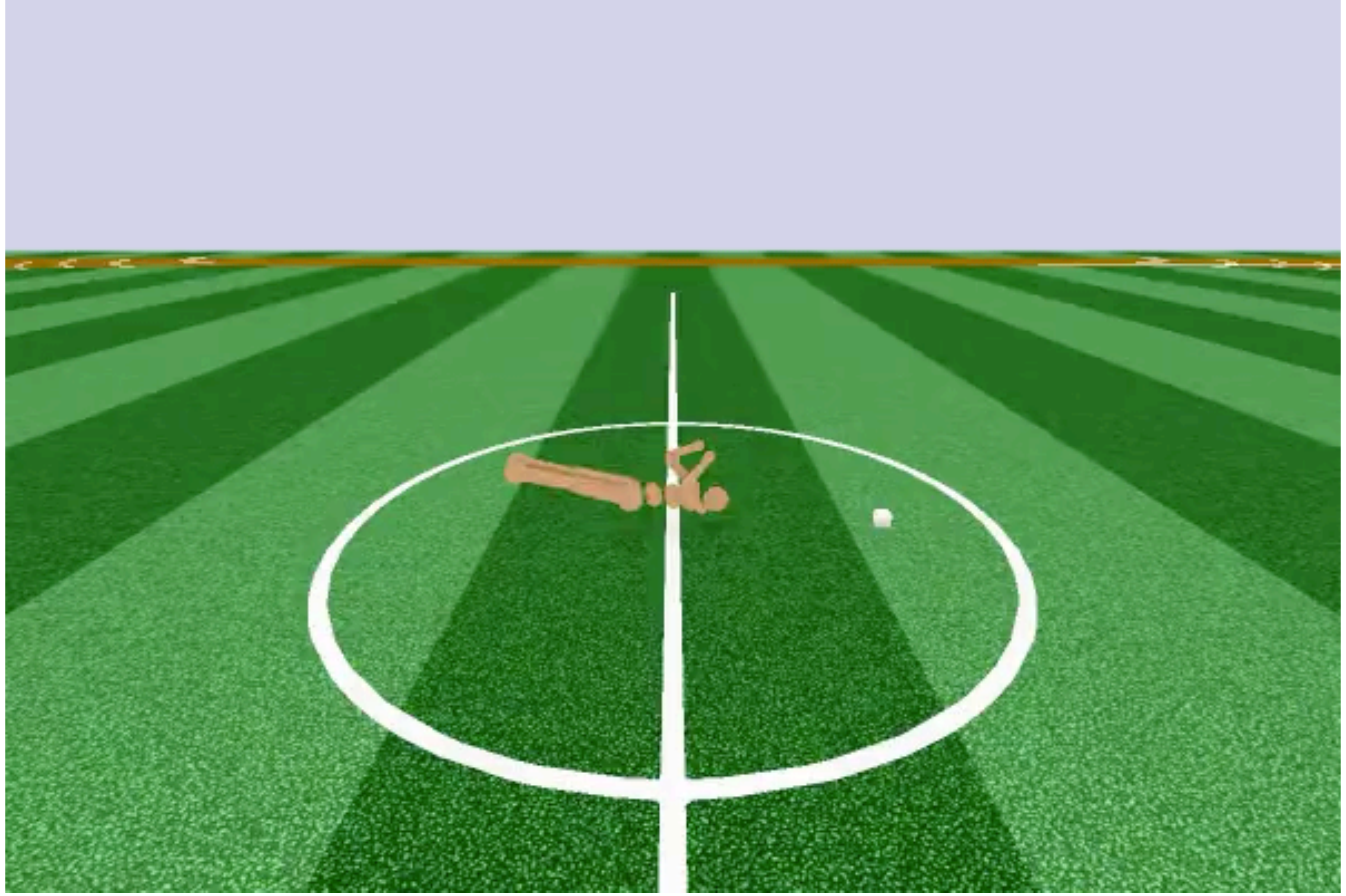
Take policy π, generate a trajectory $S_0, A_0, R_0, S_1, A_1, R_1, \ldots$

For $t = 1,2,...,T$:

Estimate return $G_t := R_t + \gamma R_{t+1} + \ldots$

$\theta = \theta + \eta \gamma^t G_t \nabla \log \pi_\theta(A_t \mid S_t)$

Policy gradient method



Agent learns to get up and seek the pink ball, optimised using Proximal Policy Optimisation (PPO), a type of policy gradient method

Policy gradient method

- Suitable to deal with large state spaces $\mathcal{S}$
- Learns from trajectories
- Suitable for environments where we don't know transition probability kernel
- *Model-free* method (no model of environment)

Policy gradient method

- Suitable to deal with large state spaces $\mathcal{S}$
- Learns from trajectories
- Suitable for environments where we don't know transition probability kernel
- *Model-free* method (no model of environment)

- Despite success, there is not much on the convergence of Policy Gradient methods
- Softmax policies of linear preferences:

$$\pi_\theta(a \,|\, s) = \frac{h_\theta(a, s)}{\sum_{a \in \mathcal{A}} \exp(h_\theta(a, s))}, \; h_\theta(a, s) = \theta \cdot \phi(a, s)$$

- $\nabla J(\theta)$ is L-Lipschitz
- Gradient descent converges to a critical point

Policy gradient method

- Suitable to deal with large state spaces $\mathcal{S}$
- Learns from trajectories
- Suitable for environments where we don't know transition probability kernel
- *Model-free* method (no model of environment)

- Despite success, there is not much on the convergence of Policy Gradient methods
- Softmax policies of linear preferences:

$$\pi_\theta(a \mid s) = \frac{h_\theta(a, s)}{\sum_{a \in \mathcal{A}} \exp(h_\theta(a, s))}, \; h_\theta(a, s) = \theta \cdot \phi(a, s)$$

- $\nabla J(\theta)$ is L-Lipschitz
- Gradient descent converges to a critical point

Can we say more about the critical point?

Advertisement II

Matryoshka Policy Gradient (MPG)

Advertisement II

Matryoshka Policy Gradient (MPG)

Context:
- Fixed horizon problem
- Entropy regularised objective

Advertisement II

Matryoshka Policy Gradient (MPG)

Context:

- Fixed horizon problem
- Entropy regularised objective
- Proof of global convergence for softmax policies with linear features
- Optimal policy for infinite horizon problem can be approximated arbitrarily well by optimal policy for finite horizon
- Criterion for global optimality for neural networks

https://arxiv.org/abs/2303.12785

RL in numerics

Schwarz et al 2023 — Reinforcement learning based limiter

- State s_t is given by $(u^n_{i-1}, u^n_i, u^n_{i+1})$ (with some normalisation)

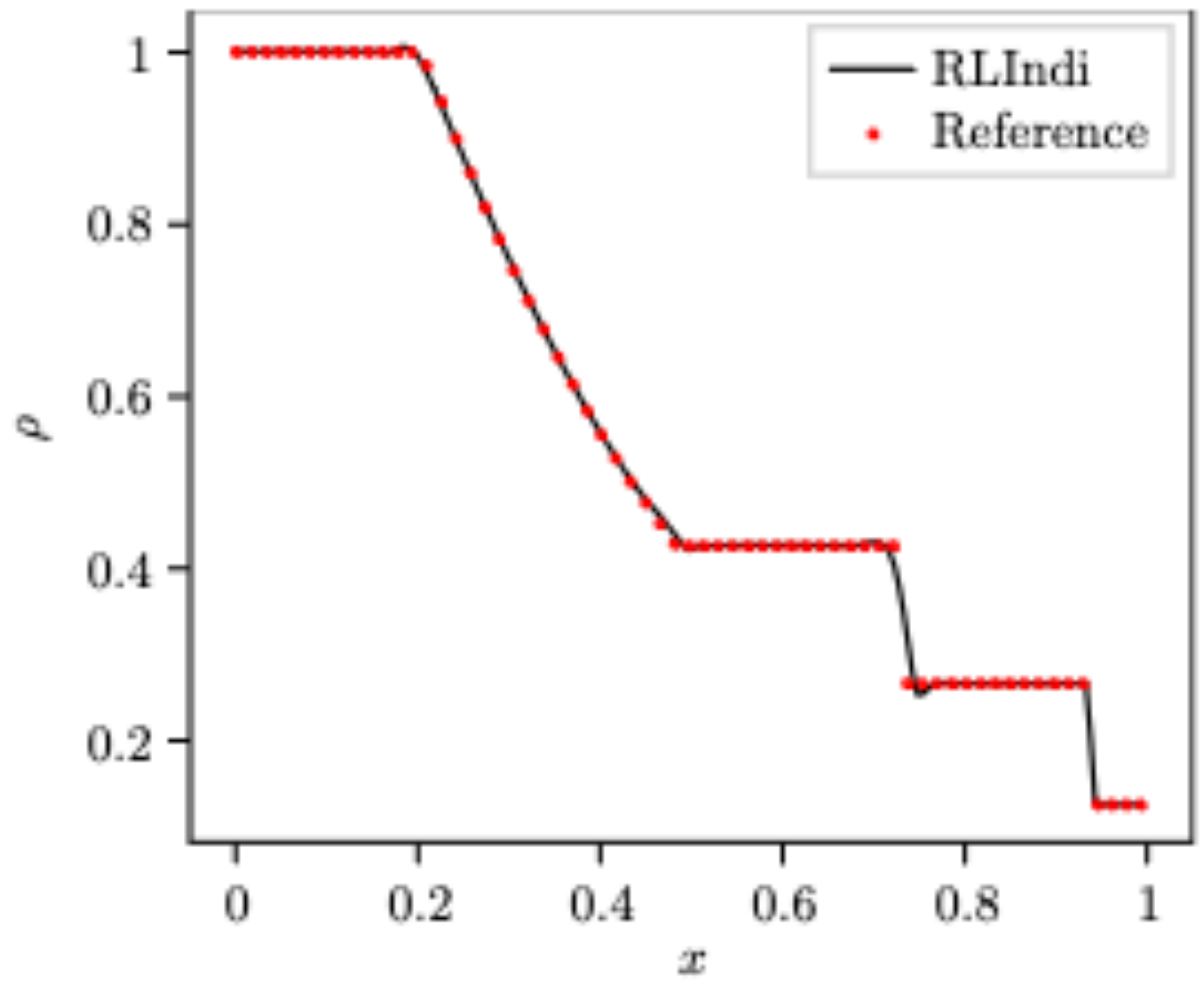
- Action computes the reconstruction slope:

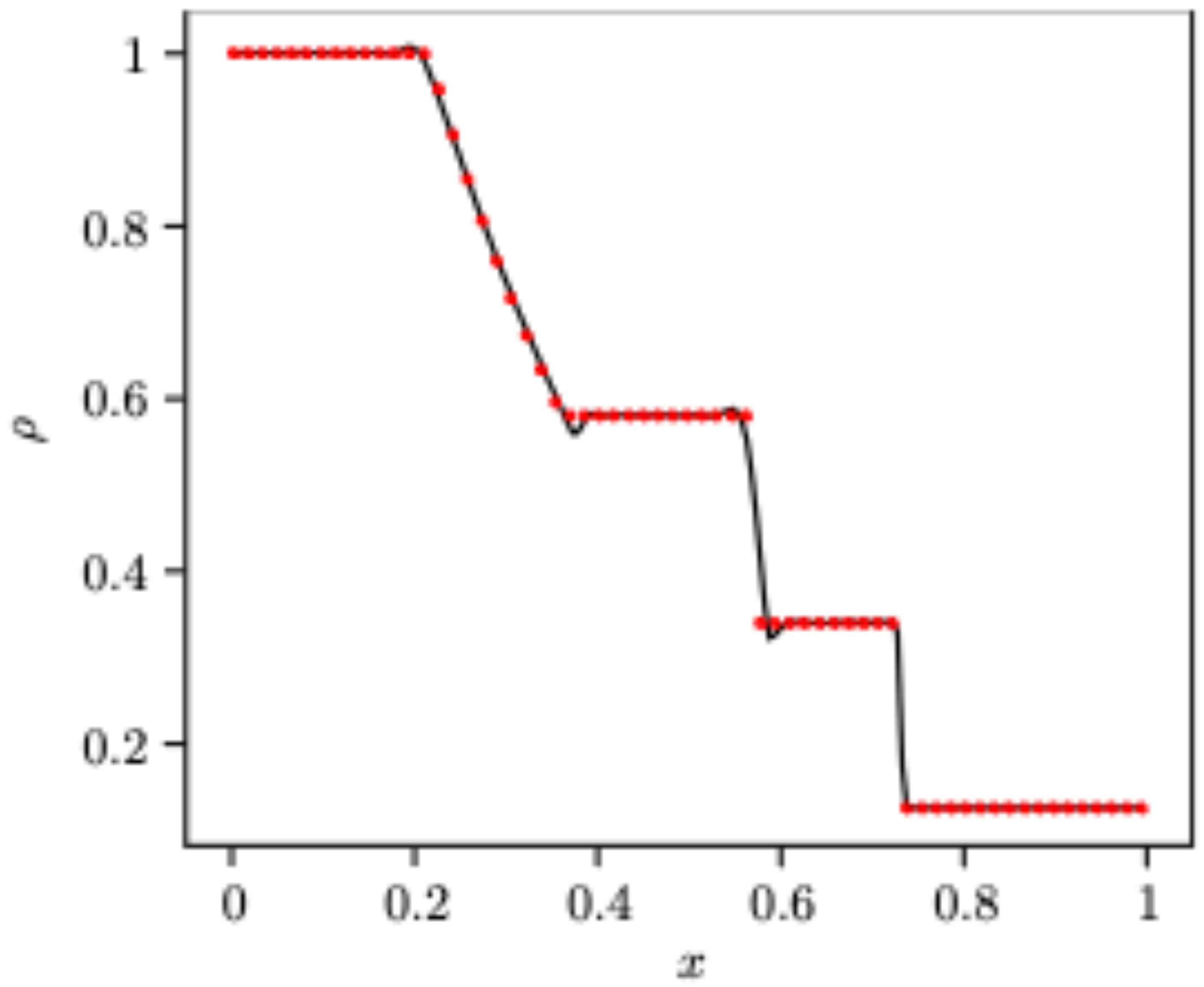
$$\delta u = 0.5(1 - a_i)(u_{i+1} - u_i) + (1 + a_i)(u_i + u_{i-1}), \quad a_i \in [-1,1]$$

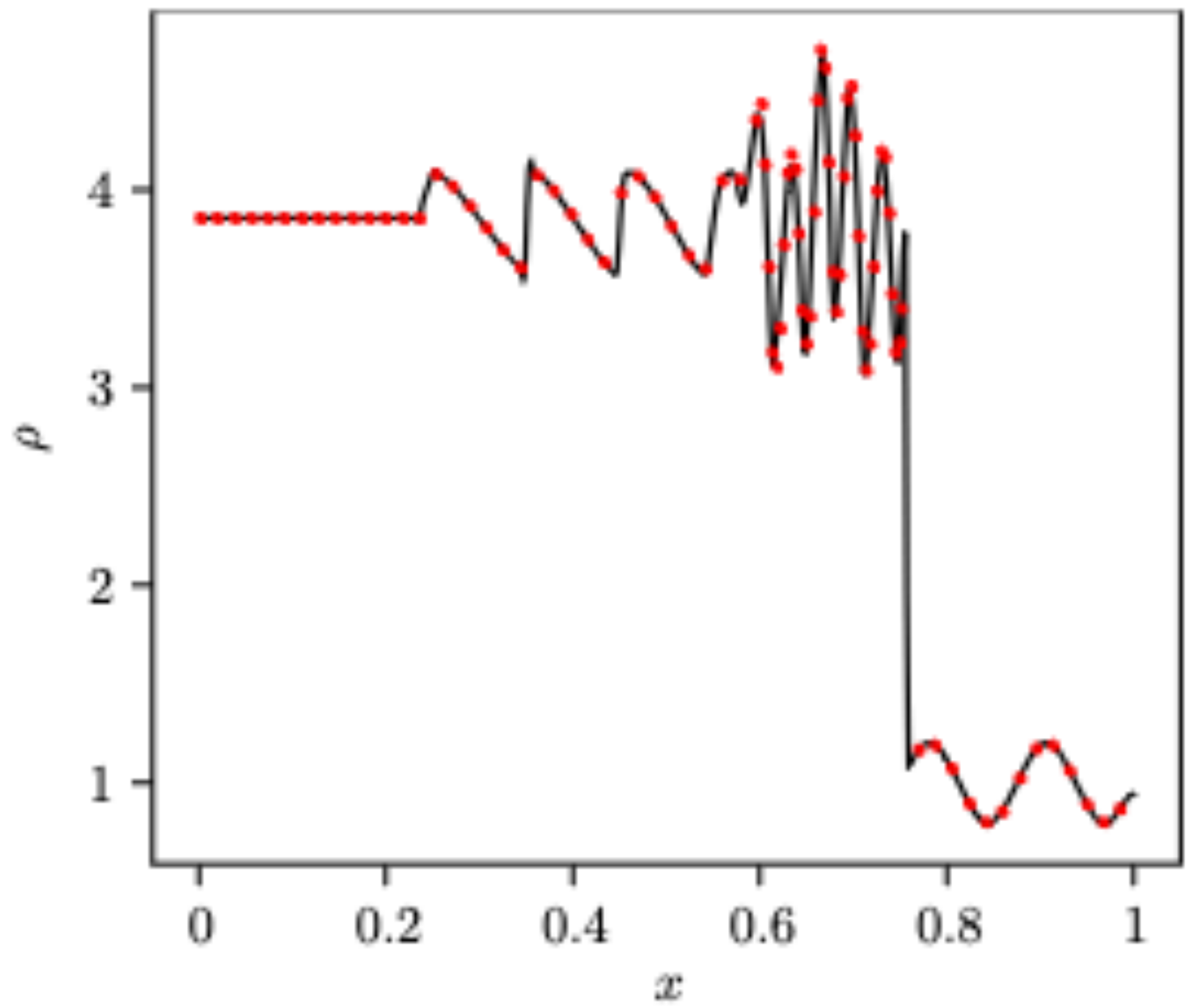
- Reward function $r(s_{t+1}, s_t, a_t)$ measures positivity and most accurate reconstruction before introducing oscillations (oscillator indicator)

- Training using Policy Gradient (Actor-critic)

RL in numerics







Reference N = 5000

RLIndi N = 500

RL in numerics

Kurz et al 2022, Beck et al 2023 — Discretisation-consistent Closure Schemes for Large Eddy Simulation using RL

https://arxiv.org/pdf/2309.06260.pdf

Speculative uses of RL

- Sequential decision making tasks
- Optimal experimental design
- Sequential sampling
- Control problems

DeepMind Alpha-Star plays Starcraft

That's all!

Congratulations, you survived.

Materials: https://github.com/hanveiga/sustech-ml-workshop

E-mail: hanveiga.1@osu.edu