Introduction to Reinforcement Learning

Machine Learning for the Built Environment 2025

Johannes Exenberger
johannes.exenberger@tuwien.ac.at

Lecture Goals

1) Know basics about model predictive control

2) High-level understanding of reinforcement learning (RL)

3) Be able to pick the right tool for a problem (traditional methods vs RL)

4) Be able to apply RL to a problem in the context of building energy systems

Lecture Content

Control Algorithm Basics

Reinforcement Learning Basics

Demo: Q-Learning

Demo: Reinforcement learning for building energy optimization

Control Basics

Methods to drive system into a **desired state** while satisfying a set of **constraints.**

Example: Control indoor temperature to fulfill occupant comfort while minimizing energy consumption.

Algorithms try to **optimize** system behavior

Optimization

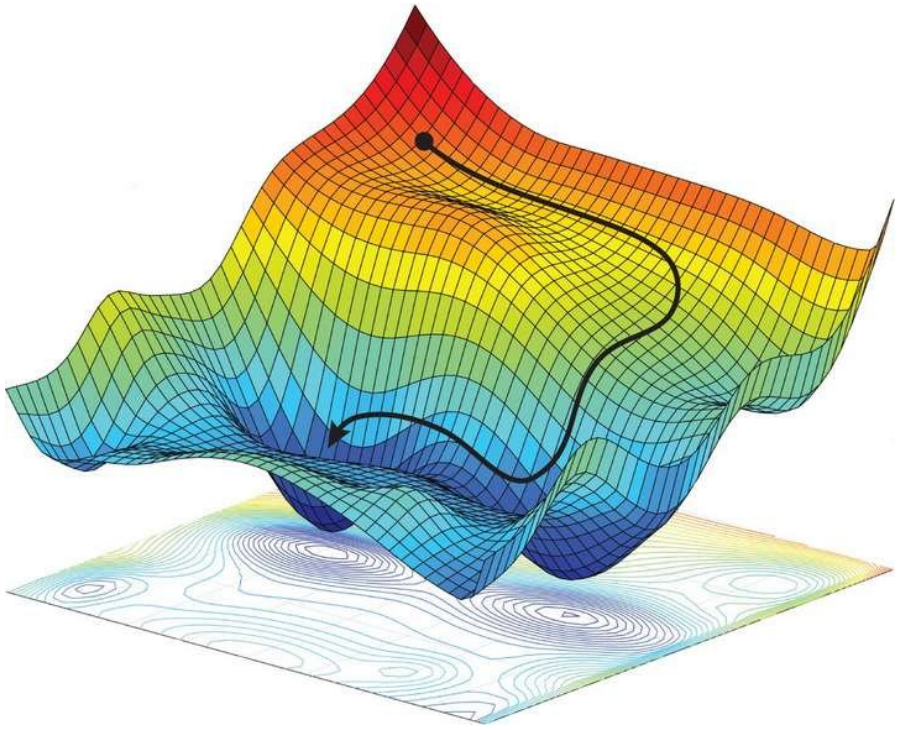
Optimization:

*find the **best solution** to a problem that fulfills certain **requirements** from a set of potential **alternatives***

Minimize or **maximize** a function

Examples:

- Find shortest path from A to B
- Minimize cost of production
- Train machine learning algorithm
- …



https://scienceprog.com/wp-content/uploads/2022/01/optimization_curve.jpg

Optimization is the core of machine learning

In many real-world cases, optimization is **really hard**

Control Basics

There are plenty of control algorithms – two examples:

Rule based control (RBC)

- apply fixed rules (`if else` statements)
- works for small problems that can be completely described
- requires (implicit) knowledge of system dynamics

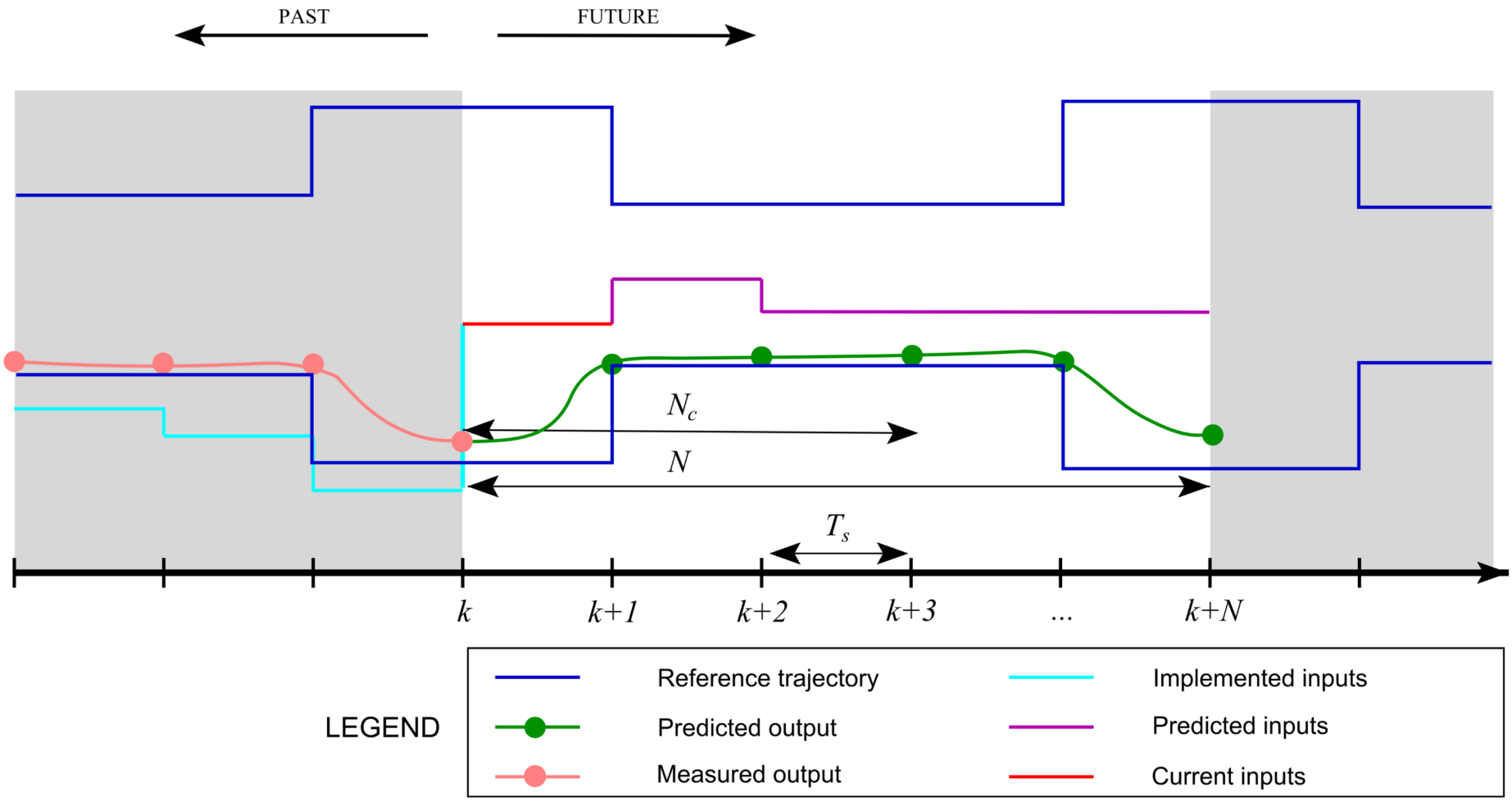
Model predictive control (MPC)

- control algorithm also assesses future impact of action
- simulation-based prediction of future system behavior
- description of system dynamics needed

Both methods can be combined!

Model-Predictive Control

Uses forecasting methods to estimate impact of control action



Jan Drgona et al. All you need to know about model predictive control in buildings. Annual Review in Control, 2020

Model-Predictive Control

Different Paradigms:

White box models	Grey box models	Black box models
Whole system is described by physical equations	Combination of both paradigms	System dynamics are unknown
High accuracy	General system dynamics are modeled, single components are exchanged for black-box models	Purely data-driven models
Computationally expensive	High flexibility to adjust accuracy-speed tradeoff	Neural networks, etc.
		Fast simulation time
		Accuracy/Generalization can be a problem

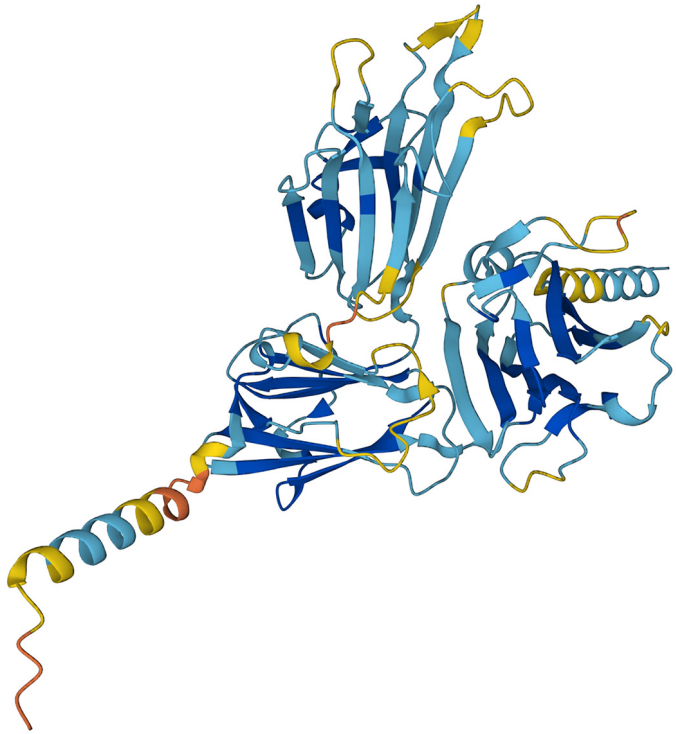
Reinforcement Learning

Like all aspects of machine learning and deep learning, RL has seen immense progress in the last decade

Google Deepmind: AlphaGO

https://aibusiness.com/verticals/deepmind-s-alpha-go-defeat-s-world-s-best-go-player-in-a-three-match-series

Google Deepmind: AlphaFold



https://www.science.org/do/10.1126/science.ade1829/full/_20220729_on_protein_gametocytesurface.jpg

Reinforcement Learning

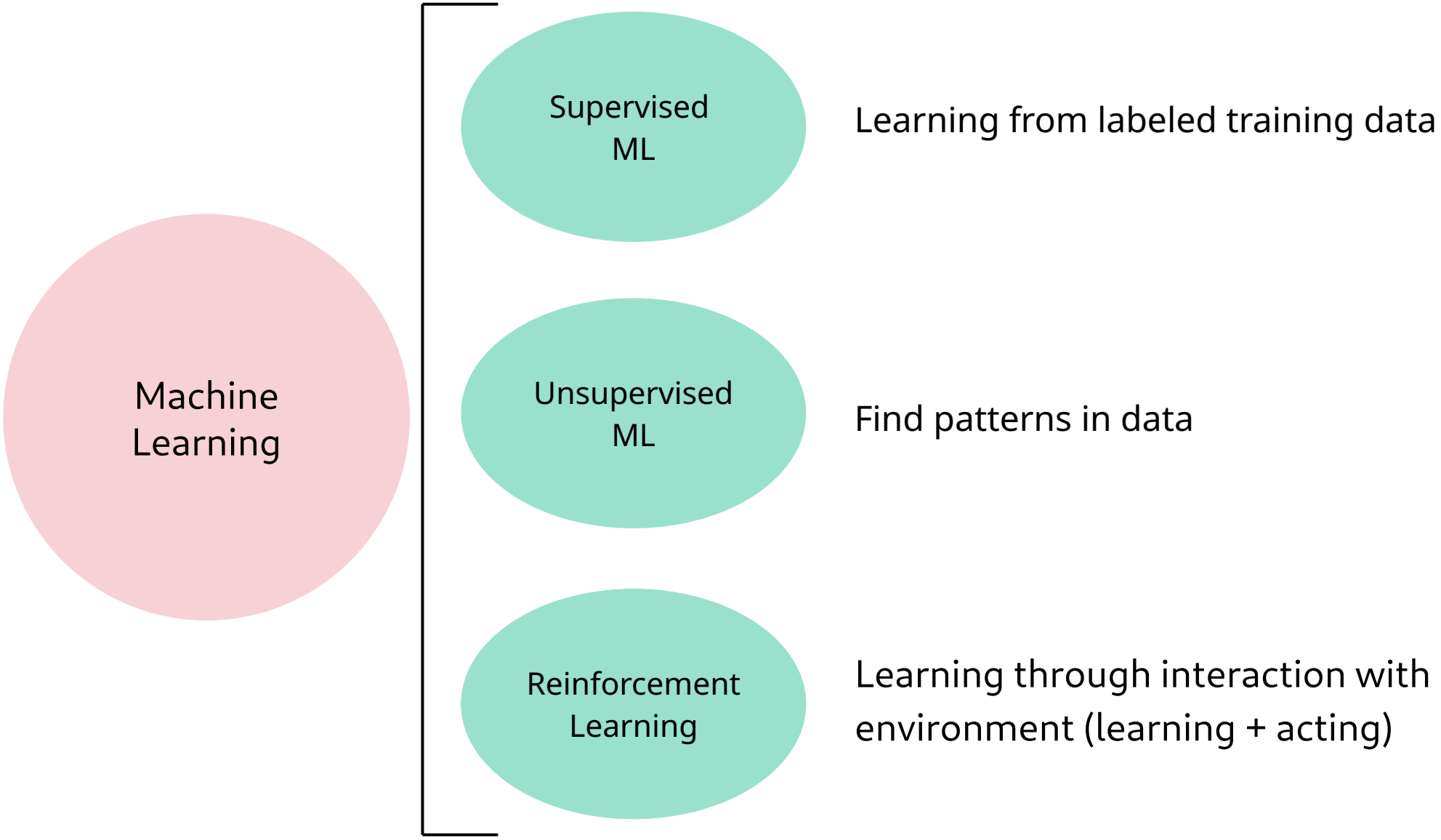
Reinforcement learning (RL) and MPC share many characteristics

RL does not need a model of system – system dynamics are **learned**

If model of system exists, MPC is **more efficient**

→ **RL is especially suited for complex problems with unknown system dynamics!**

Reinforcement Learning



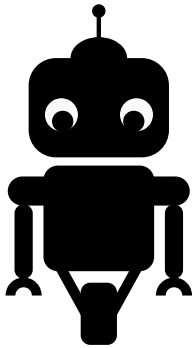
Reinforcement Learning

Agent learns by interacting with environment (trial and error)

Idea: tell agent **what** it should achieve (reward), but not **how** to succeed

Inspired by learning behavior in children

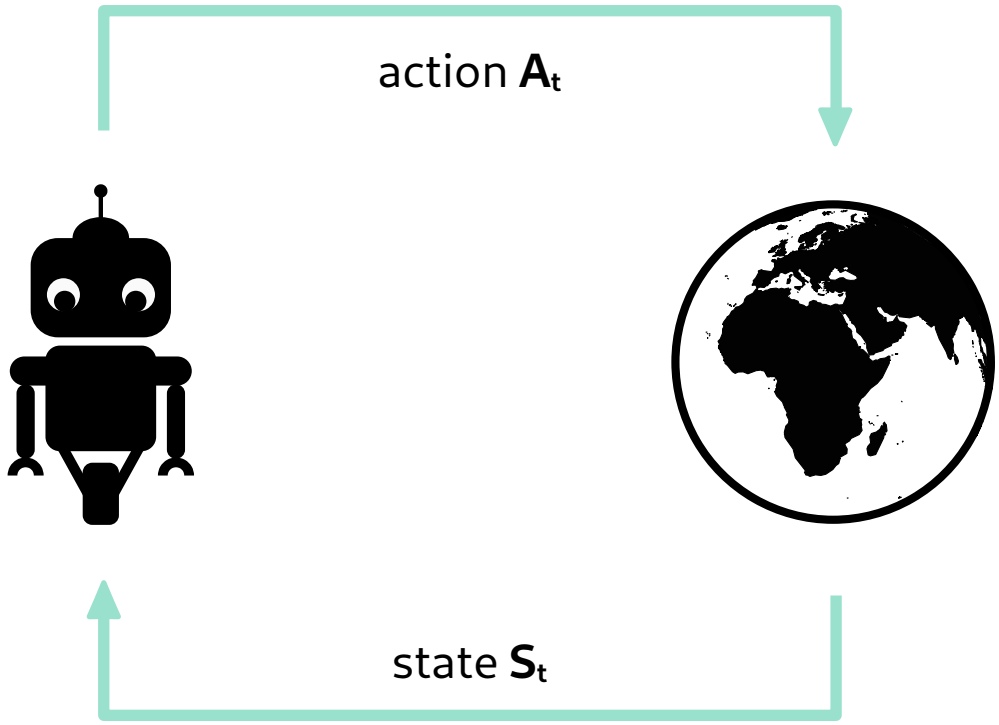
Reinforcement Learning



Reinforcement Learning

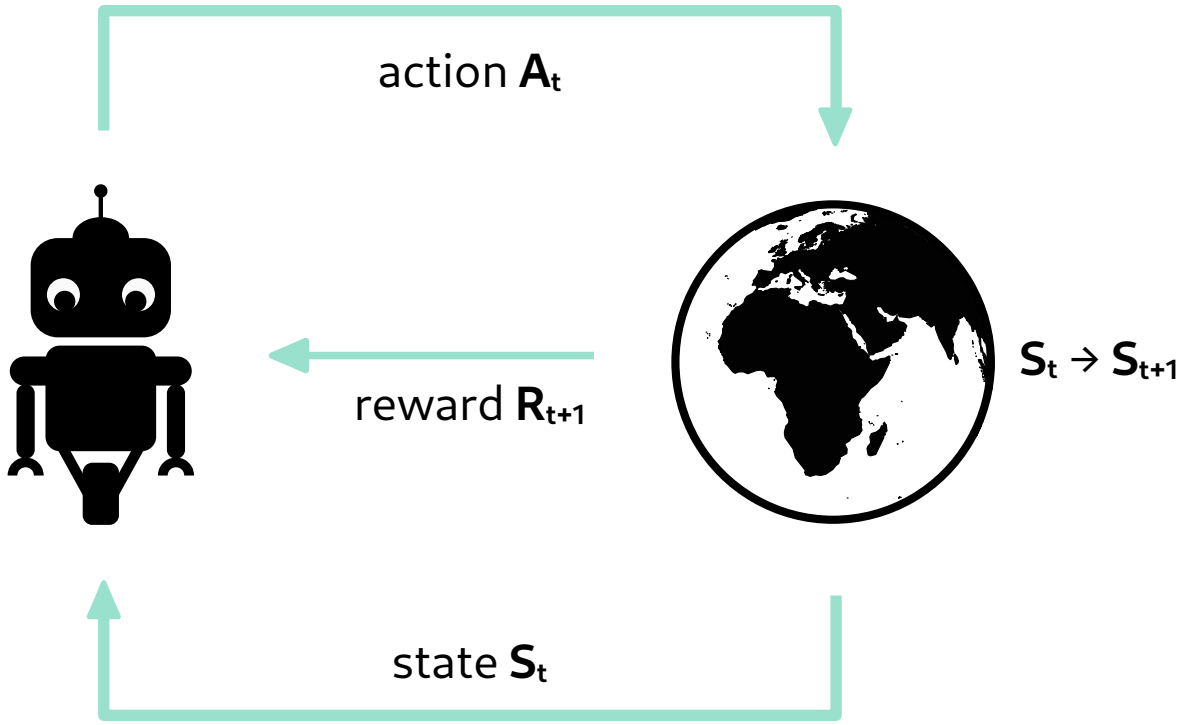
- Agent observes environment state S_t

Reinforcement Learning



- Agent observes environment state S_t
- Agent takes action A_t

Reinforcement Learning



- Agent observes environment state S_t
- Agent takes action A_t
- Environment transitions to new state S_{t+1}
- Agent receives reward R_{t+1}
- Agent observes environment state S_{t+1}
- …

Reinforcement Learning

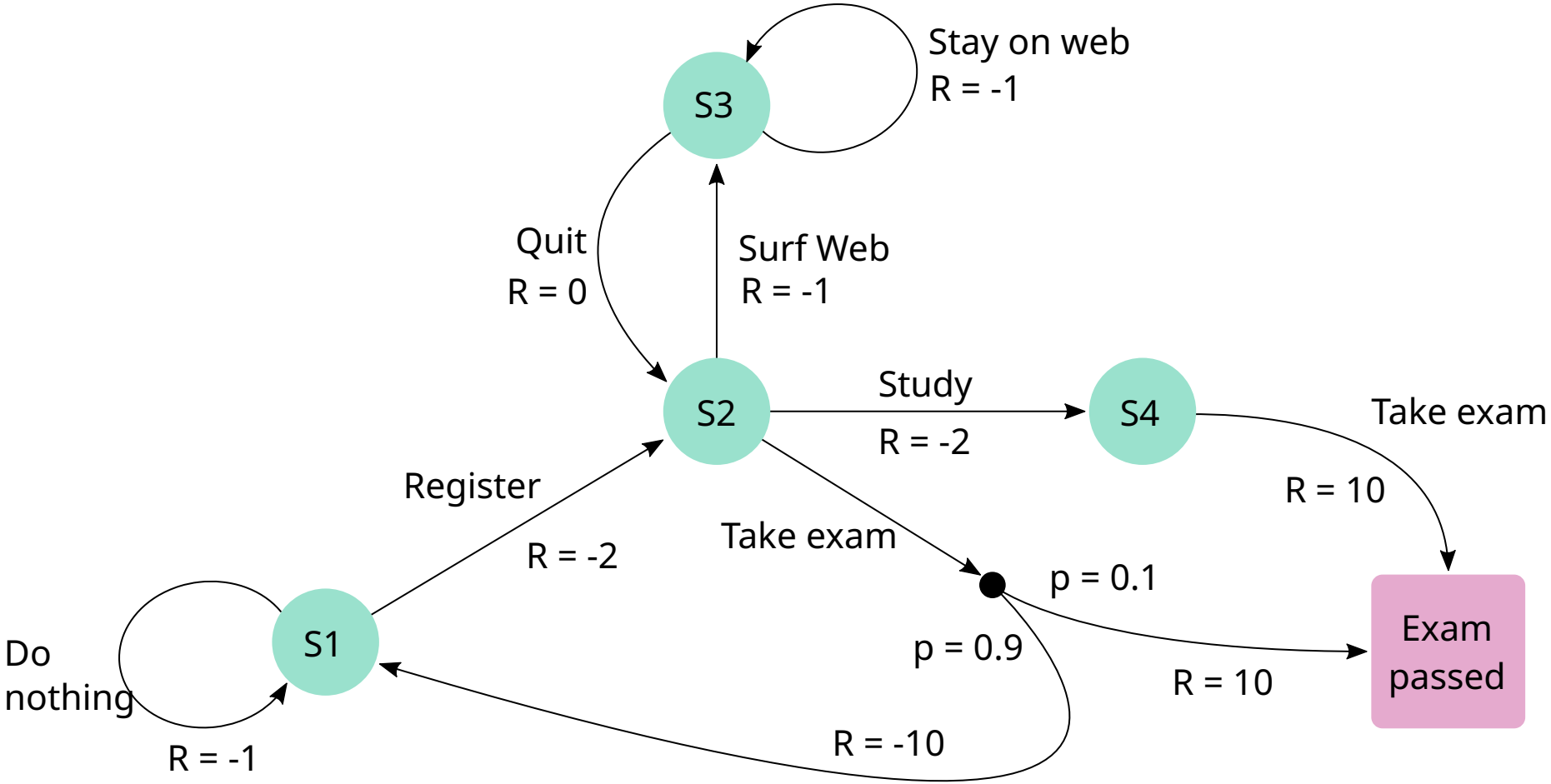
How should the environment be modeled?

Markov decision process: sequence of states with possible actions

State includes enough information to predict outcomes of actions (memoryless)

State transitions: can be deterministic (fixed) or stochastic (random)

Markov Decision Process



State Actions

Behavior of agent is determined by the *policy*
The policy defines the probability of an action a to be chosen in state s

$$\pi(a|s) = p(a|s)$$

Action value function: how much total reward (= return G) can be expected when choosing action a in state s and then following policy pi?

$$q_\pi(s, a) = \mathbb{E}[G_t|s, a] \qquad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

Bellman Equation

Goal: maximize total expected reward (= find *optimal policy*)
Bellman optimality equation:

$$q_*(s,a) = \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[r(s,a) + \gamma \max_{a'} q_*(s',a') \right] \quad \forall s \in \mathcal{S}$$

s	State	$r(s,a)$	Reward function
a	Action	γ	Discount factor
$p(s'\|s,a)$	State transition probability	$\max_{a'} q_*(s',a')$	Maximum possible Q value in state s_{t+1} under the best action a_t

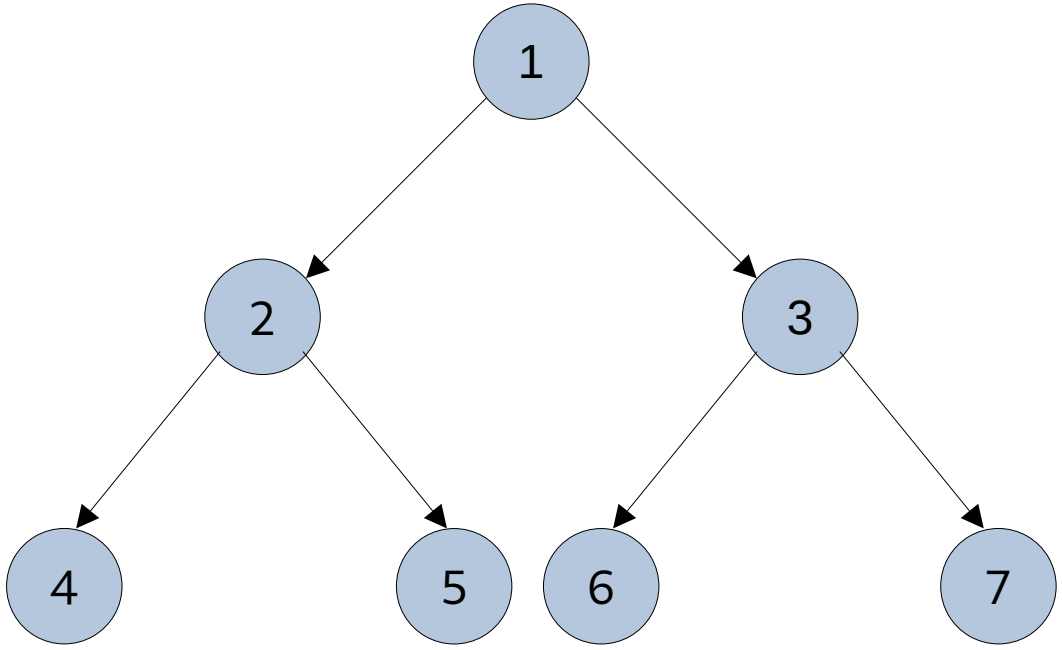
Directly leads to the optimal policy – agent only has to take the action that maximizes q in every step

→ Agent has to explore environment to find optimal actions!

Bellman Equation

Bellman equation is not specific to RL – it is a fundamental equation of **dynamic programming (DP)**

DP: solve problem by recursively solving smaller sub-problems



Example: shortest path problem

Reward Function

The reward function is **crucial** – it is the only way we interact with the agent!

A wrong or poorly designed reward function is often the cause for unsatisfying performance

Agents can take shortcuts: reward function should not be ambiguous!

Reward-hacking example

Exploration – Exploitation Dilemma

Agent wants to maximize reward by *exploiting* known actions, but should also *explore* the environment to find best actions

A common method to address this is the *epsilon-greedy policy:* the agent *exploits* known actions with probability (1 – e) and chooses random action (*exploration*) with probability e

$$a_t = \begin{cases} \max_a Q(s,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

Q-Learning

Most popular basic RL algorithm

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

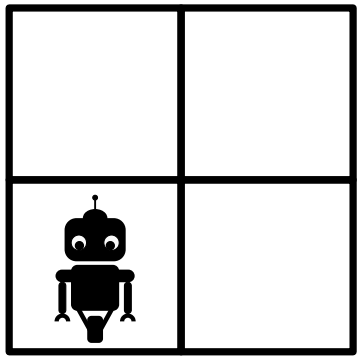
s_t	State
a_t	Action
r_t	Reward
α	Learning rate
γ	Discount factor
$\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$	Maximum possible Q value in state s_{t+1} under the best action a_t

Q-Learning

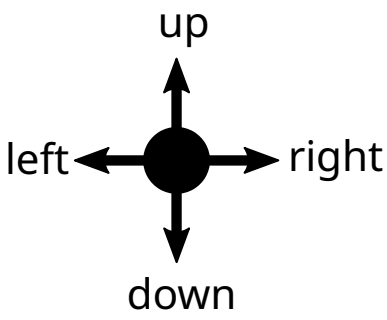
Q-learning is based on discrete state and action spaces

Q-table: table of all state-action pairs

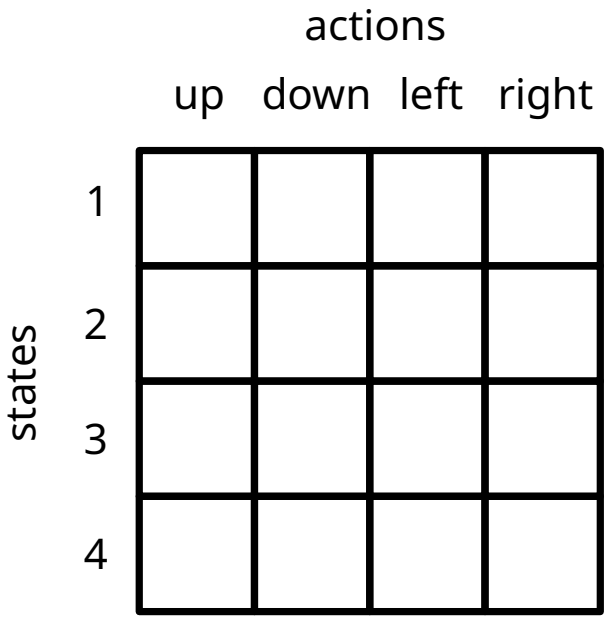
→ **Q-learning best suited for small state-action spaces!**



Environment (4 states)



actions



Q-table

Reinforcement Learning: Further Material

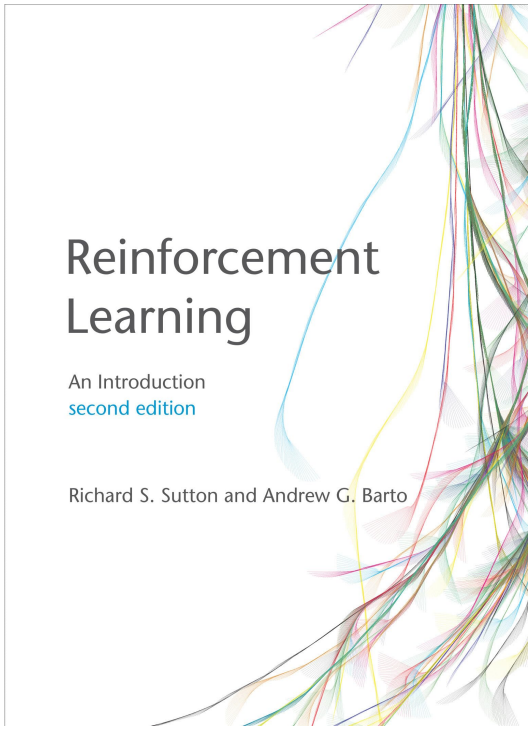
Book:

Sutton & Barto: Reinforcement Learning: An Introduction

Online tutorials/packages:

Open AI: Spinning Up

Stable Baselines package



Example: Taxi Problem

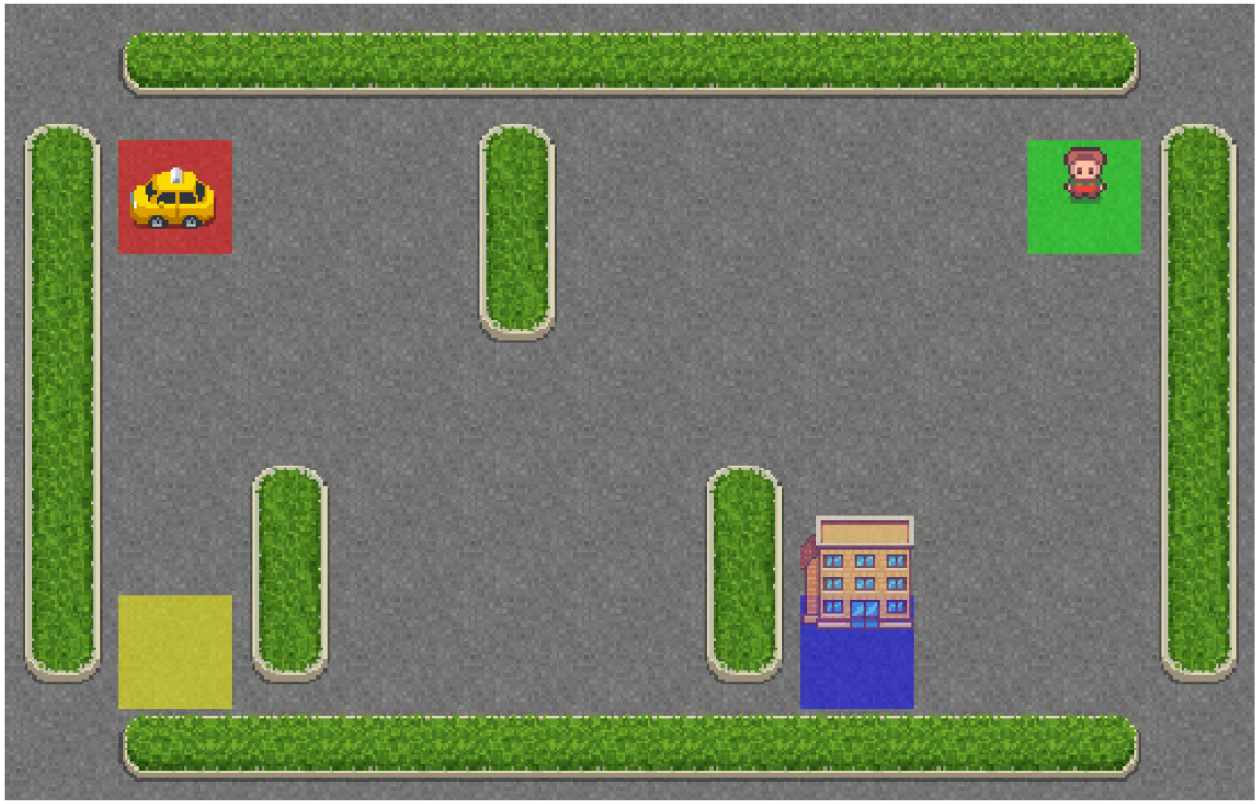
Action space:

Move up, down, left, right
Pickup passenger
Drop-off passenger

State:

Taxi position (25)
Passenger location (5)
Drop off location (4)

} 500 states



Rewards:

-1 per step (if no other rewards)
+ 20 for delivering passenger
-10 for wrong pickup/drop-off actions