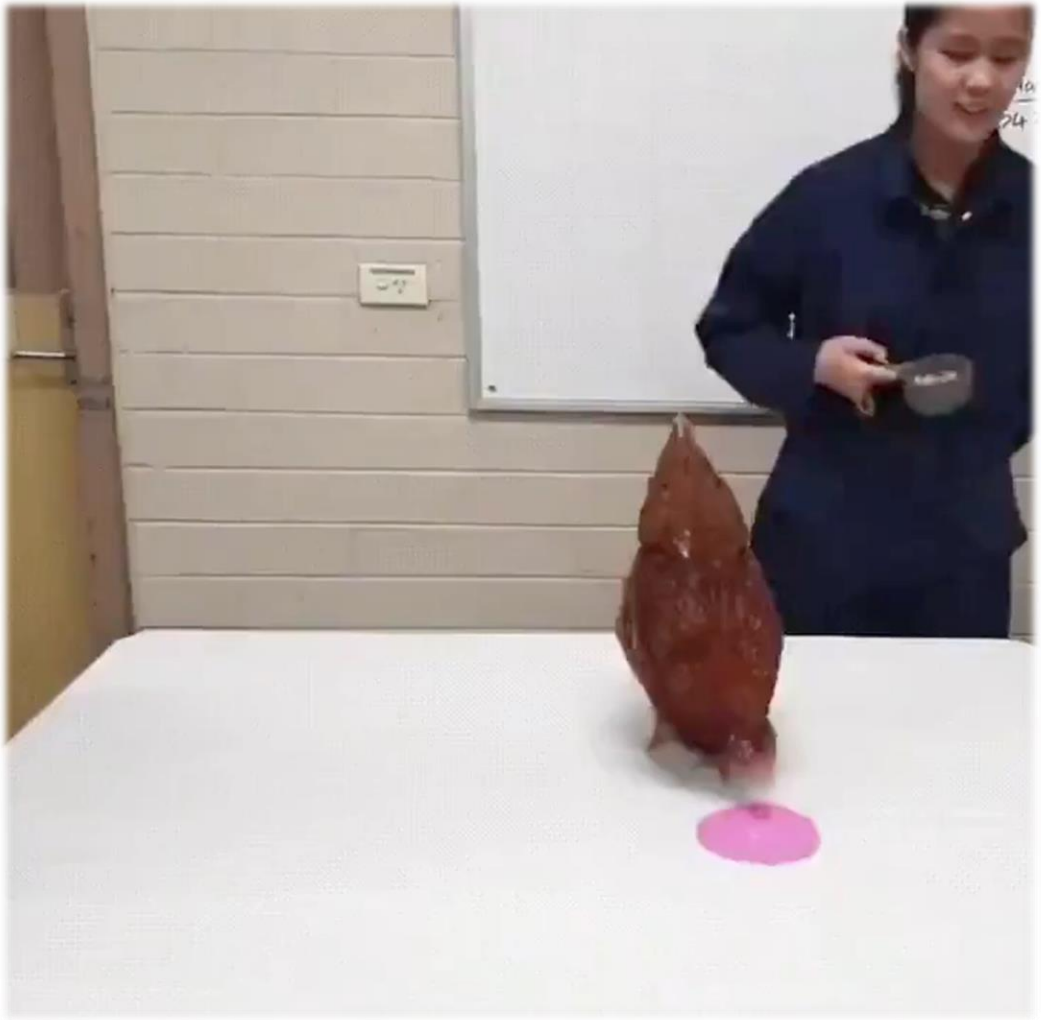
Machine Learning in Accelerators

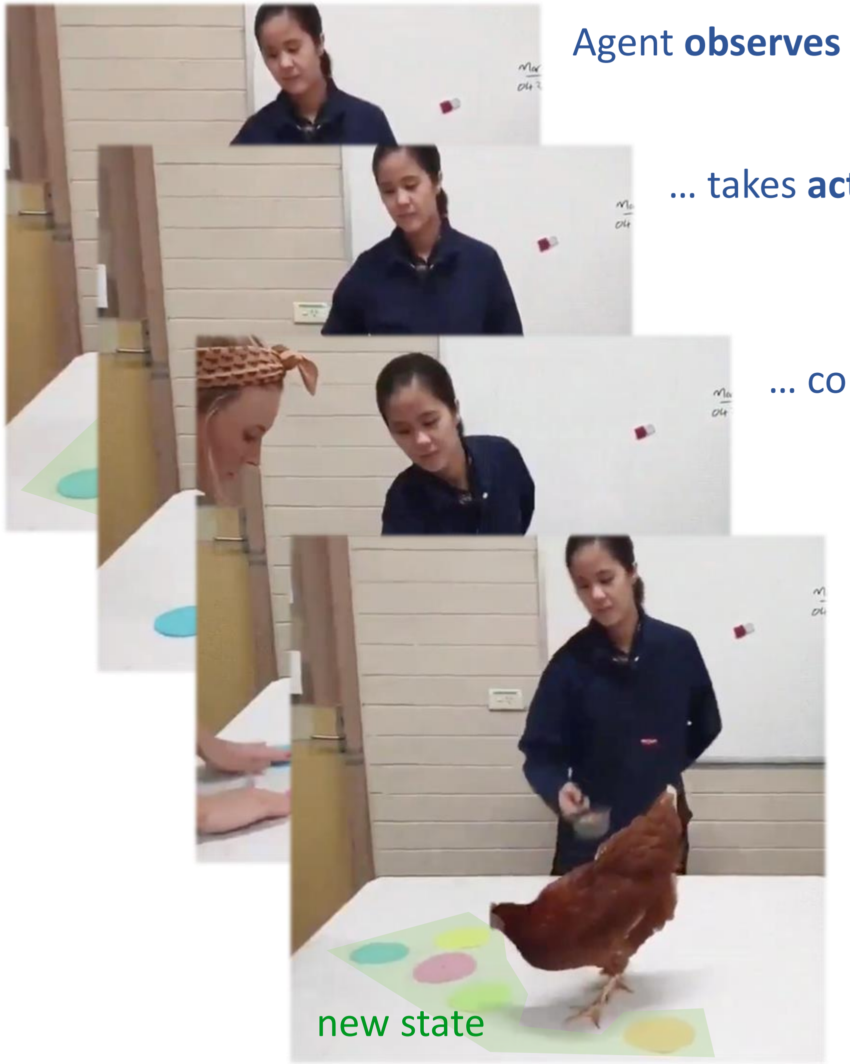
Introduction to Reinforcement Learning

M. Schenk, V. Kain
CERN, Switzerland

Introduction
RL in nature



source



Agent **observes** environment **state**

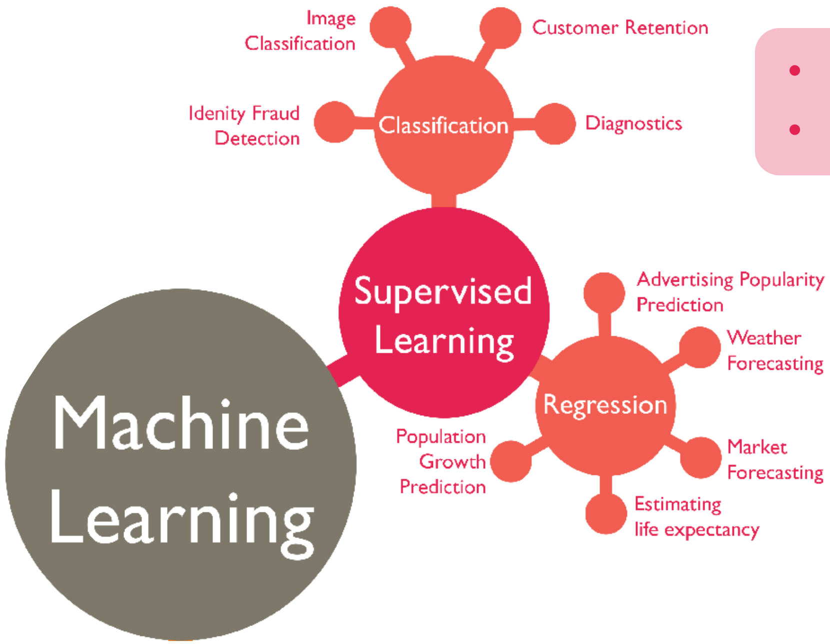
… takes **action**

… collects **reward**

Environment **transitions** to **new state**

Introduction

RL in the machine learning landscape



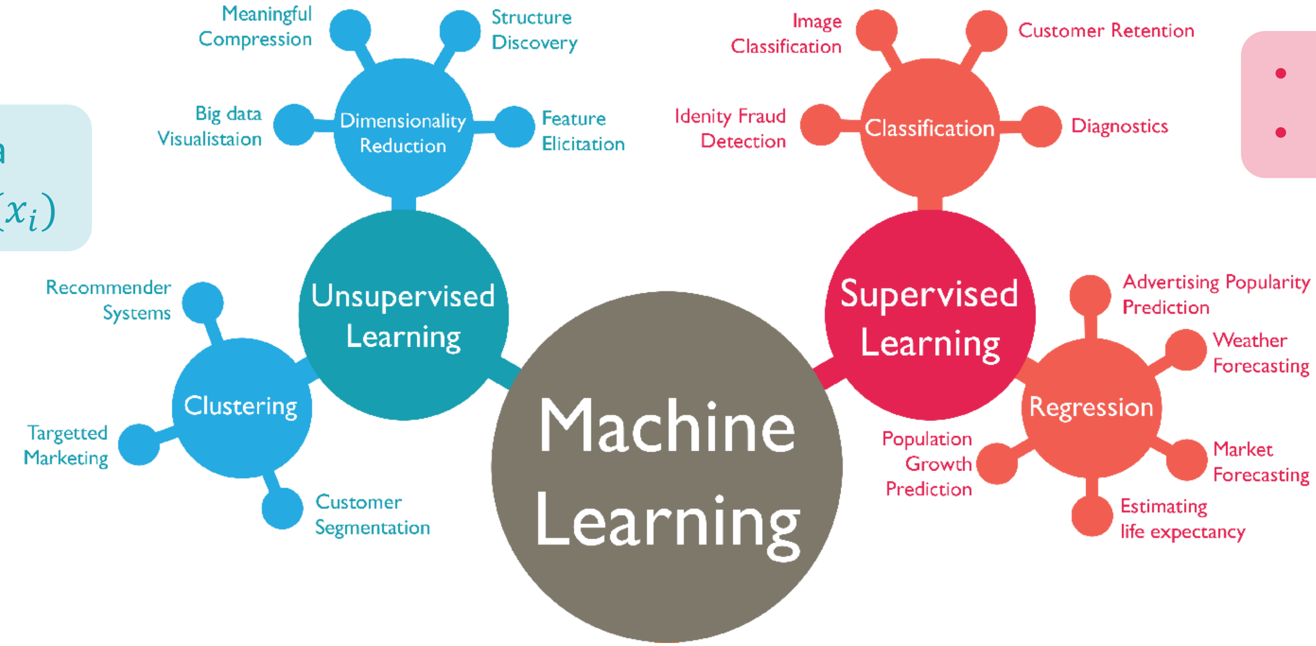
- **Goal:** find mapping $f\colon\ x_i \longmapsto y_i$
- **Data:** labelled samples (x_i, y_i)

source

Introduction

RL in the machine learning landscape

- **Goal:** find structure in data
- **Data:** unlabelled samples (x_i)



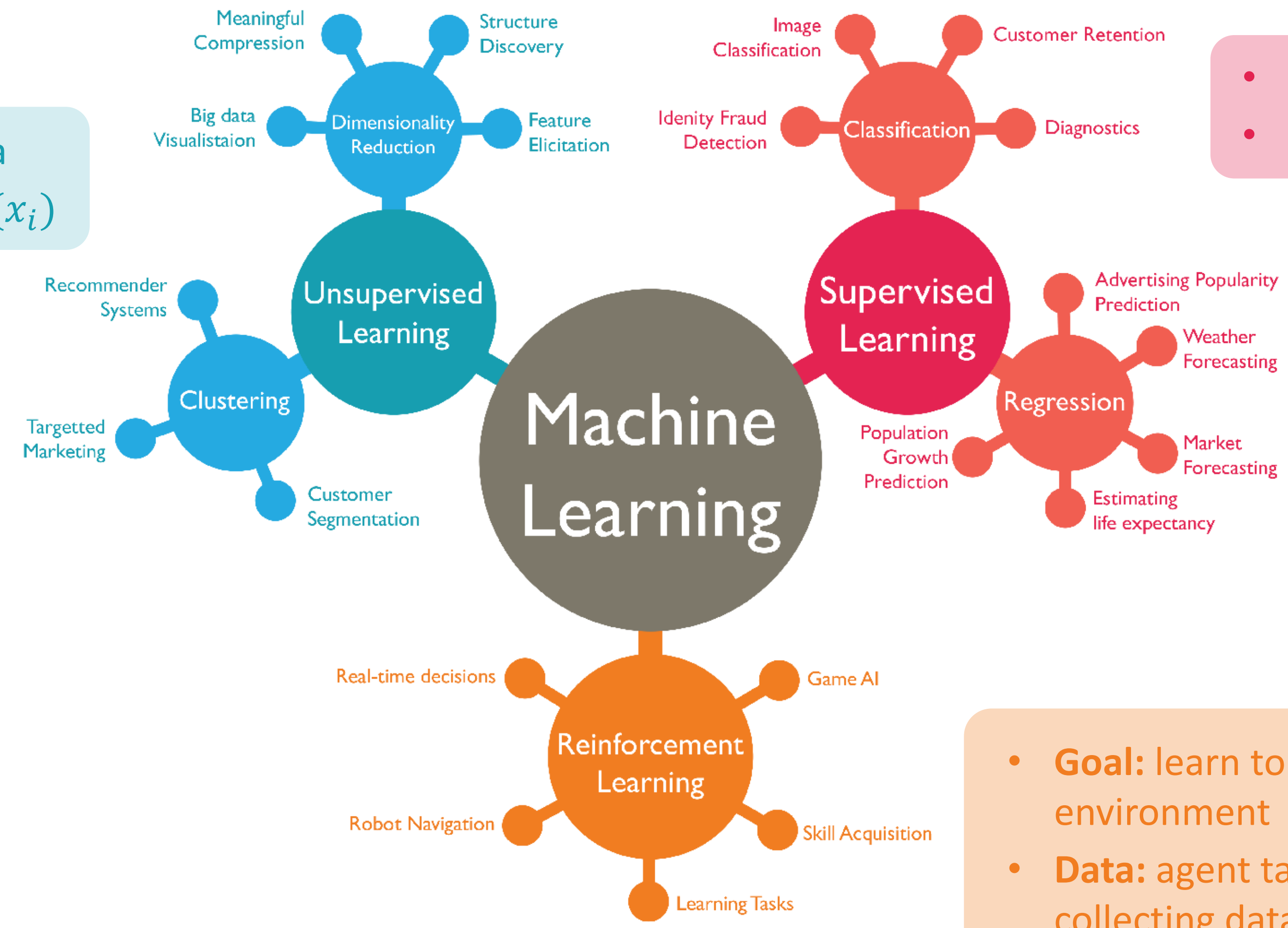
- **Goal:** find mapping $f\colon x_i \longmapsto y_i$
- **Data:** labelled samples (x_i, y_i)

source

Introduction

RL in the machine learning landscape

- **Goal:** find structure in data
- **Data:** unlabelled samples (x_i)



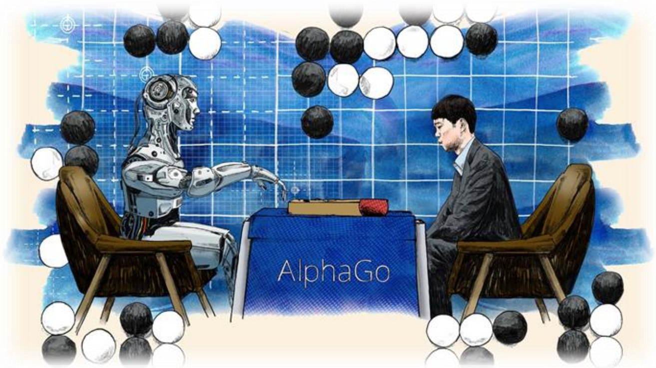
- **Goal:** find mapping $f\colon\ x_i \longmapsto y_i$
- **Data:** labelled samples (x_i, y_i)

- **Goal:** learn to take optimal decisions in an environment
- **Data:** agent takes actions within environment collecting data samples & rewards

source

Introduction

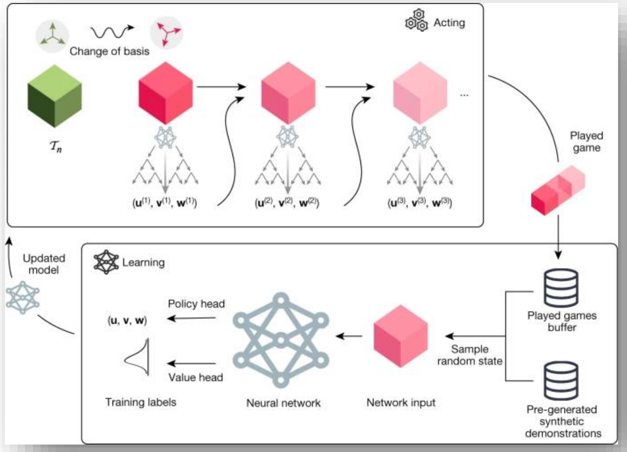
RL: state-of-the-art

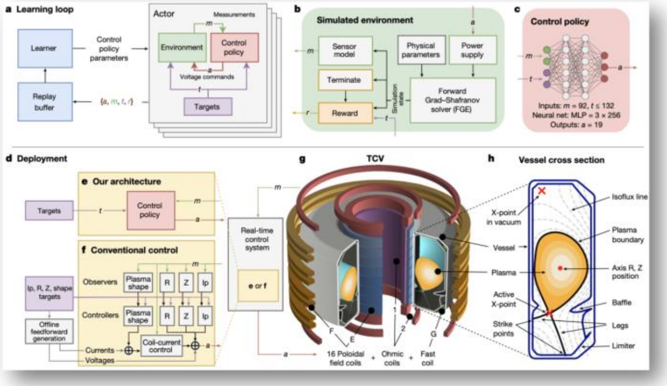


DeepMind, 2016: AlphaGo

DeepMind, 2022: AlphaTensor

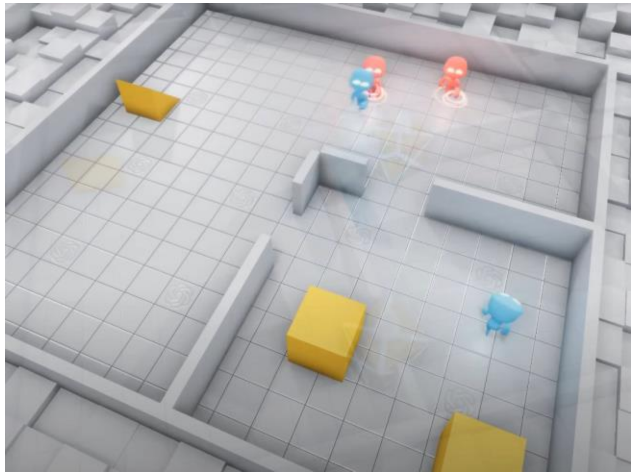
- Improving computational efficiency of **matrix multiplication**
- **RL agent discovered more efficient algorithms** than previously known





DeepMind & SPC-EPFL, 2022: Tokamak control

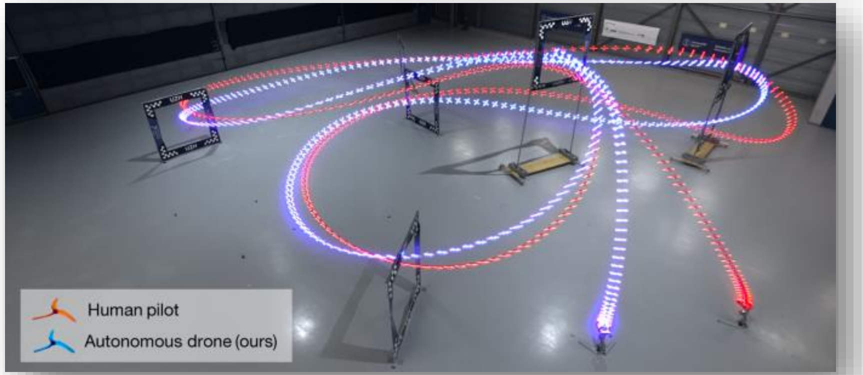
- **Maintaining plasma** within Tokamak
- Requires high-dimensional, high-frequency, closed-loop control
- RL agent as **magnetic controller**



OpenAI, 2019: Hide and seek

UZH & Intel Labs, 2023: Drone racing

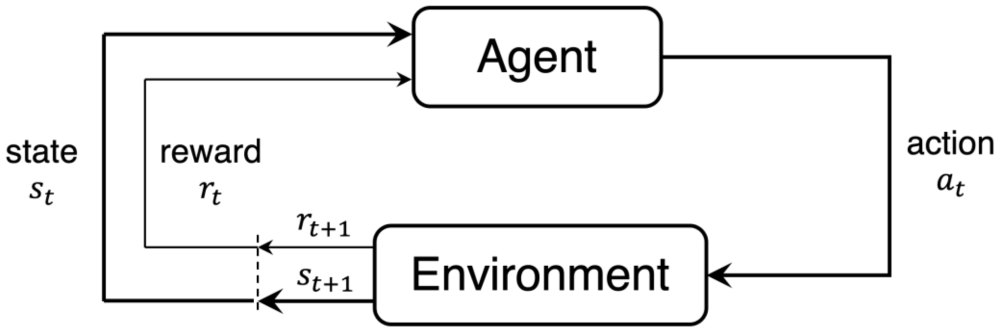
- **RL agent beats human drone racing champions** in real environment
- Training in simulations with mixed-in residual models from real data



and more …

Introduction
RL in a nutshell

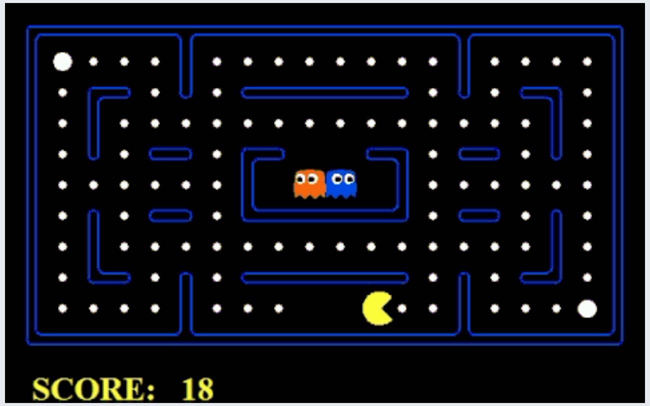
- **Iterative, online, trial-and-error learning**
  - At every **time step** t, agent **observes** environment **state** s_t, **selects** an **action** a_t, and **collects reward** r_{t+1}
  - **Environment transitions** from s_t to s_{t+1} under action a_t



Sutton & Barto

- **Objective**
  - **Learn to act** in a way that **maximises cumulative reward** over time *(= return)*
  - *In other words:* learn an **optimal policy** *(= agent's behaviour)*

Example: Pac-man



Environment: everything you interact with & its dynamics
maze structure, Pac-man, ghosts, food, game rules, etc.

Agent: player (you!)

State: where am I? Where are ghosts, snacks, cookies?

Actions: ↑, ←, ↓, →

Reward: food (+), ghosts, time (−)

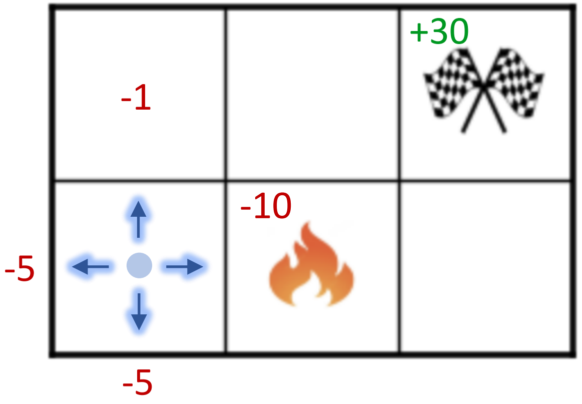
Return: game score (food eaten, lives lost, time elapsed)

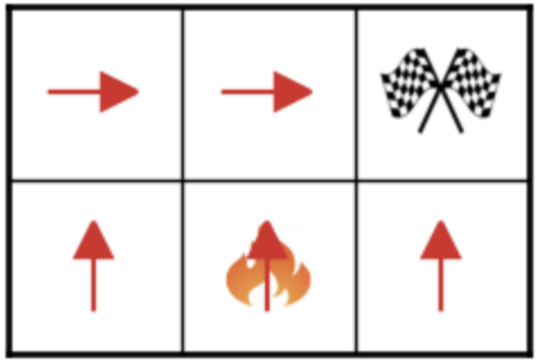
Policy: given current state, should I go ↑, ←, ↓, → ?

Introduction

RL: exercise preview

- **Objective:** navigate around a gridworld maximising return
  (= cumulative reward over time)
- **Four discrete actions:** ↑, ←, ↓, →
- **Reward model**
  - ➢ Stepping into empty field costs -1, bumping into walls -5
  - ➢ Stepping through fire costs -10
  - ➢ Reaching destination gives +30
- Use RL to **learn an optimal policy**
  "what's the best action to pick from any of the fields (= state) ?"





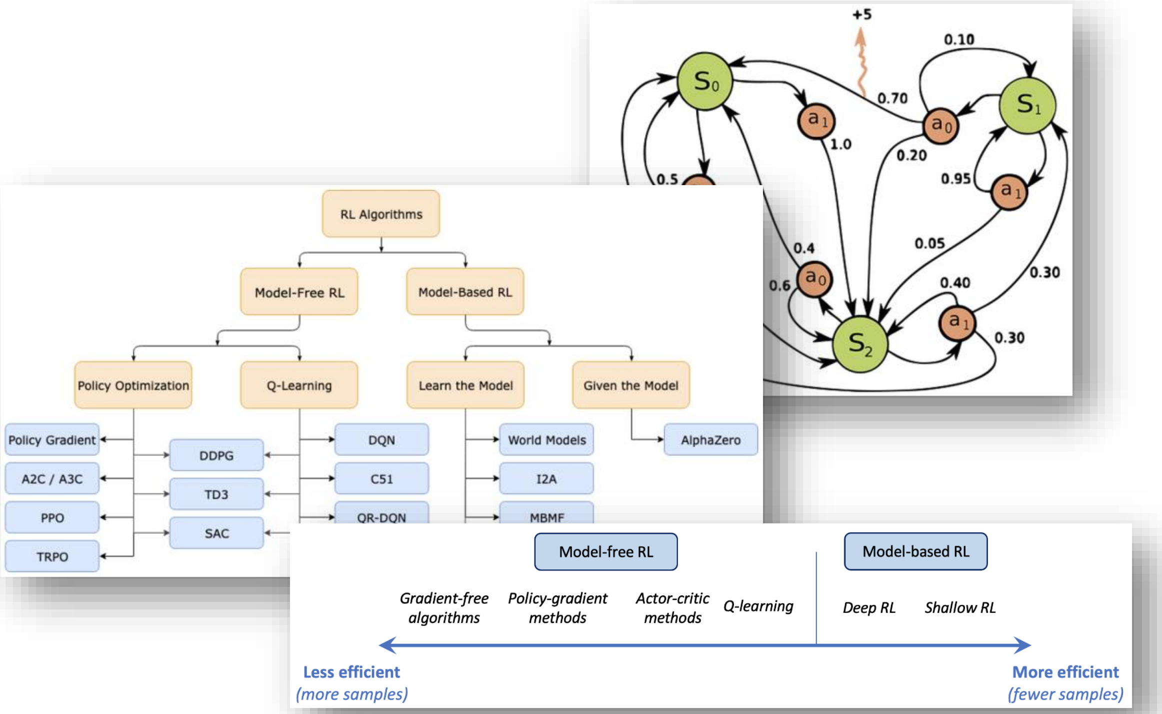
(An) optimal policy

Contents

Introduction

Lecture scope

- **Formalism**†
  - RL terminology
  - Markov decision process
- **Algorithms**
  - Value- and policy-based methods
    foundation for understanding many other RL algorithms
  - Q-learning and actor-critic scheme
  - Discrete and continuous state-action spaces
- **Challenges**



RL is a broad and exciting topic! The goal is to give you an introductory perspective and hopefully spark your interest in exploring it further ☺

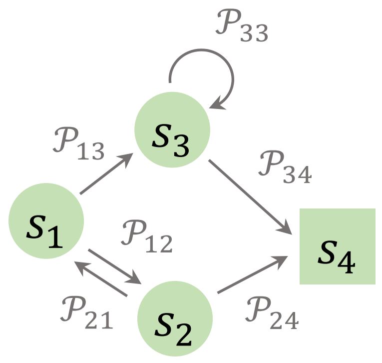
†***Disclaimer:*** *will not be mathematically rigorous or complete*

Contents

Formalism
Overview

State space $\mathcal{S}$
State transitions
Markov property

Markov Process / Chain



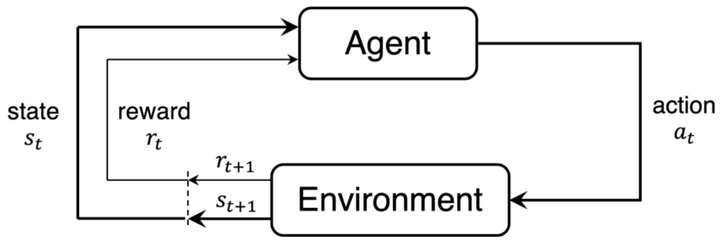
$\mathcal{S} = \{s_1, s_2, s_3, s_4\}$

\+ **Reward mechanism**
$r_{t+1} = R(s_t, s_{t+1})$

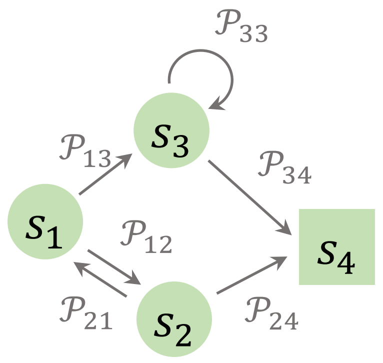
Markov Reward Process

\+ **Decision making**
action space $\mathcal{A}$, agent, policy
$r_{t+1} = R(s_t, s_{t+1}, a_t)$

Markov Decision Process (MDP)



Formalism
Markov Process



$\mathcal{S} = \{s_1, s_2, s_3, s_4\}$

- **Memoryless random process** consisting of
  - **State space** $\mathcal{S}$
    discrete or continuous
  - State **transition probabilities** $\mathcal{P}_{ss'}$
- States possess the **Markov property** *(= memorylessness)*
  - The **future evolution** of the Markov chain **depends only on the information contained in the present state** s_t, but not on the history of past states $s_{t-1}, s_{t-2}, \ldots$
  - $P(s_{t+1} \mid s_t) = P(s_{t+1} \mid s_t, s_{t-1}, \ldots, s_0)$

Formalism

Examples of memoryless states

Chess

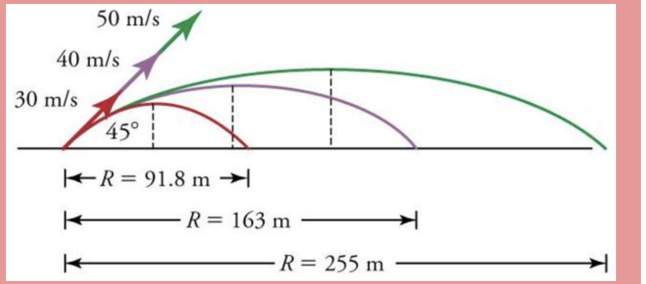
Arrangement of pieces on the board fully defines current state.

There may be many ways to arrive at that particular state, but this is irrelevant for deciding the next move and the future progression of the game.

Flight trajectory of a cannonball

State given by its current position and velocity $s_t = (\vec{x}_t, \vec{v}_t)$ provides enough information to predict the future (in an ideal world …)

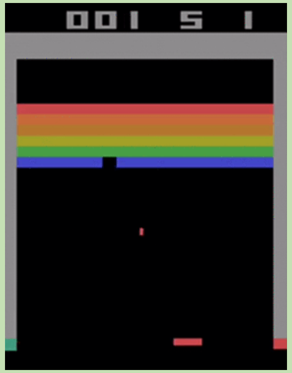
N.B.: $s_t = (\vec{x}_t)$ or $s_t = (\vec{v}_t)$ do not fulfil the Markov property



Atari Breakout

State given by single video frame is not Markov, but a sequence of frames is

E.g.: need to know what direction the ball is moving to forecast future progression of the game and be able to take an optimal decision



Formalism

Markov Process: example

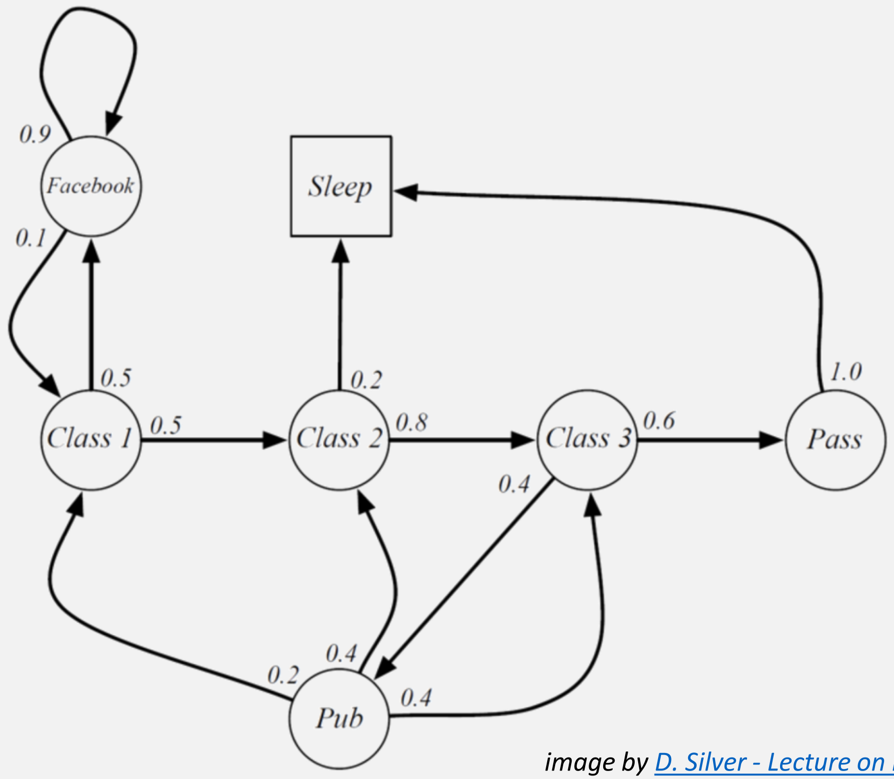


image by [D. Silver - Lecture on RL]

- $\mathcal{S} = \{\text{Class 1}, \text{Class 2}, \text{Class 3}, \text{Facebook}, \text{Pub}, \text{Pass}, \text{Sleep}\}$
- N.B.: “Sleep” is also called a terminal state, because once in it we will never leave it

Formalism

Markov Reward Process

- A Markov Process that has in addition a
  - **Reward function** $r_{t+1} = R(s_t, s_{t+1})$
  - **Discount factor** $\gamma \in [0, 1]$
- **Return**

$$G_t = \sum_{k=0} \gamma^k \, r_{k+t+1}$$

sum of discounted future rewards

- The **discount factor** $\boldsymbol{\gamma}$ controls the relative importance of immediate vs future rewards
  - $\gamma \to 0$: only care about immediate rewards
  - $\gamma \to 1$: care about long-term rewards
- Can be better to give up immediate rewards to collect higher rewards in the long run …
  Example: sacrificing a piece in chess to eventually win the game

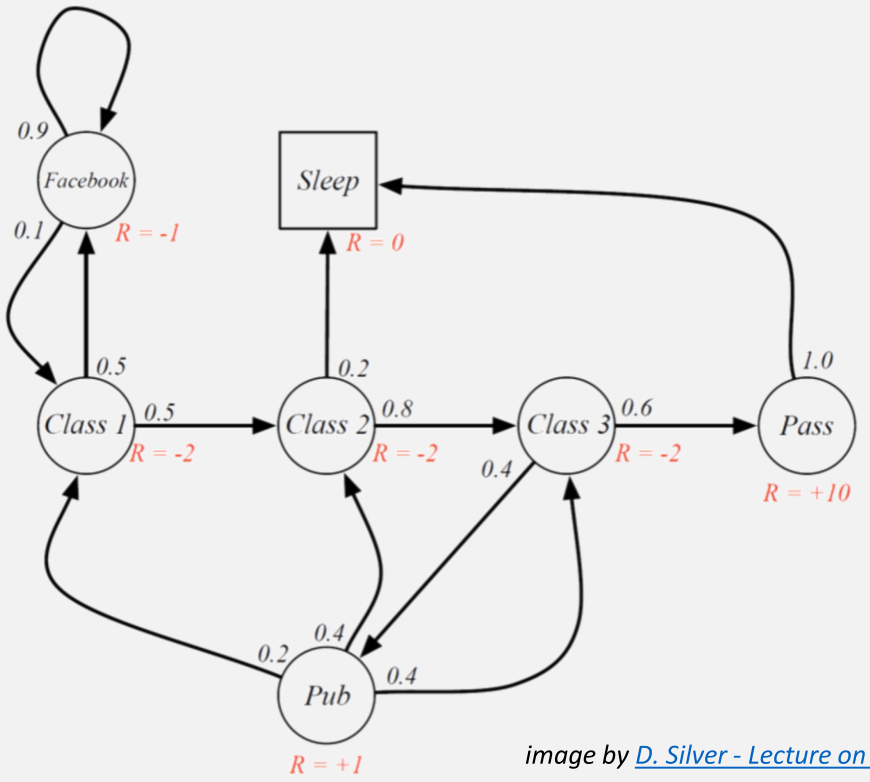


image by D. Silver - Lecture on RL

Class 1 → Class 2 → Class 3 → Pass → Sleep

$G_0 = (-2) + 0.5^1 \cdot (-2) + 0.5^2 \cdot (-2) + 0.5^3 \cdot (+10) = -2.25$

$\gamma = 0.5$

Formalism

Markov Decision Process (MDP)

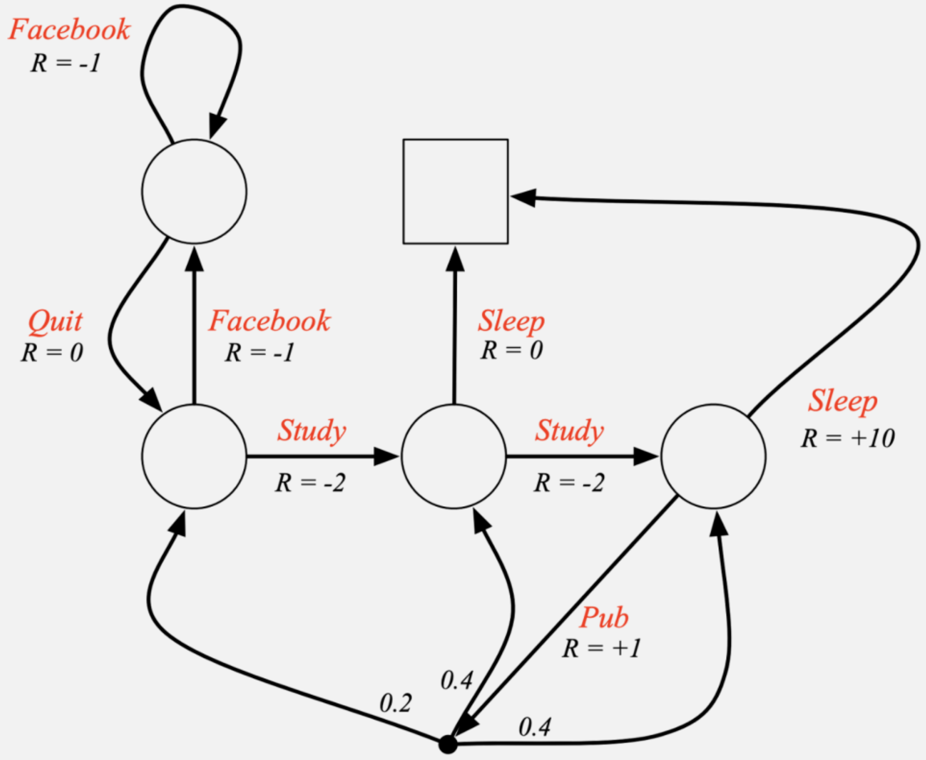


image by *D. Silver - Lecture on RL*

- Extend Markov Reward Process by adding **decision making mechanism**
  - **Action space $\mathcal{A}$**
    discrete or continuous
  - A **decision maker** *(= agent)* acts on the environment following a **policy $\boldsymbol{\pi}$**
  - **Stochastic state transitions** are still allowed
- We define a **trajectory** τ as a **sequence of states, actions, and rewards** over time

$$\tau = (\mathrm{s}_0, \mathrm{a}_0, \mathrm{r}_1, \mathrm{s}_1, \mathrm{a}_1, \mathrm{r}_2, \dots, \mathrm{r}_\mathrm{T}, \mathrm{s}_\mathrm{T})$$

MDPs form the foundation of RL

Formalism

RL objective & policy

- The **policy $\boldsymbol{\pi}$** encodes an agent's decision making or **behaviour**
- **Two formulations** are common
  - **Stochastic policy:** *assigns probabilities to state-action pairs* (s, a)

$$\pi \colon \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$$

$$\pi(a \mid s) = P(A_t = a \mid S_t = s) \text{ with } \textstyle\sum_a \pi(a \mid s) = 1$$

  - **Deterministic policy:** *outputs specific action a_t for given state s_t*

$$\pi \colon \mathcal{S} \rightarrow \mathcal{A}$$

$$a_t = \pi(s_t)$$

- We will also distinguish between
  - **Behaviour policy** π_b: policy **guiding the agent's actions** during **exploration** and **data collection**
  - **Target policy** π_t: policy we **aim for the agent to learn** and optimise towards

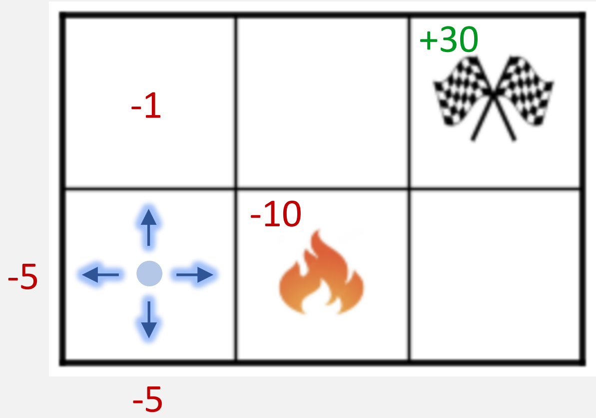
Example: random policy

$$\pi(\uparrow \mid s) = 0.25$$

$$\pi(\downarrow \mid s) = 0.25$$

$$\pi(\leftarrow \mid s) = 0.25$$

$$\pi(\rightarrow \mid s) = 0.25$$



Formalism
RL objective

- **RL objective**
  - **Learn optimal behaviour** in an environment: trained agent should **select best sequence of actions from any state**
  - Also known as the **optimal policy** π^*
  - "Best sequence of actions" means "**the one maximising return**"
- RL is based on the **reward hypothesis**

  "Any goal can be formalised as the outcome of maximising a scalar, cumulative reward"

Interesting thoughts by Sutton and Barto: http://incompleteideas.net/rlai.cs.ualberta.ca/RLAI/rewardhypothesis.html

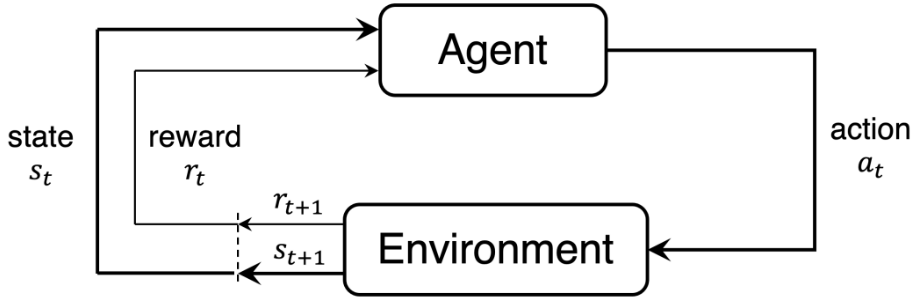
Formalism

Classification of MDPs

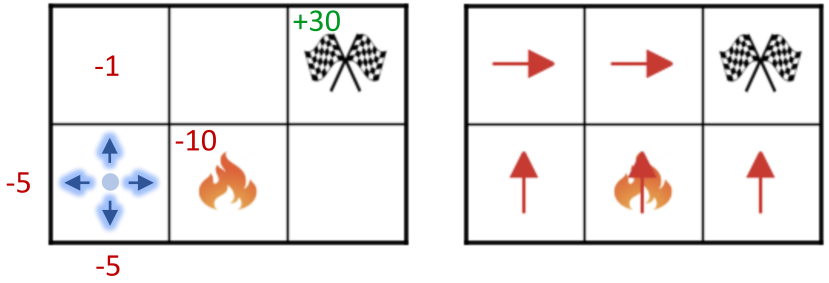
- **Finite MDP:** sets of possible states, actions, and rewards are finite
- **Stochastic vs deterministic MDP**
  - **Stochastic:** outcomes of taking a specific action not deterministic, i.e. starting from state s_t and taking action a_t might not always bring us to the same state s_{t+1}
  - **Deterministic:** outcome of an action is fully predictable
- **Episodic MDP**
  - Each episode ends in a terminal state (or is truncated)
  - Return is the sum of discounted rewards from time t till end of episode
  - Episodes are independent of each other
- **Continuous** *(= infinite horizon)* **MDP**
  - Runs indefinitely with no terminal states
  - Discount factor $\gamma < 1$ is key to avoid infinite returns
- **Partially vs fully observable MDPs**
  - Agent might not see the true, full environment state s_t but only be able to make a partial observation
  - Real-world environments are very often only partially observable

Quick recap

- The goal of RL is to **learn to make optimal decisions** (*take best actions*) in an environment based on some observables (*state*)
- The **quality of a decision** made is quantified by a scalar **reward**
- Through **trial-and-error,** the RL agent **collects data samples** $(s_t, a_t, r_{t+1}, s_{t+1})$ from which it learns **optimal behaviour** (*optimal policy* π^*)
- Formally, this is described by a **Markov decision process** (MDP)
- **Example RL tasks:** playing board or video games, humanoid robots learning to walk, control systems (e.g. tuning accelerator parameters), ...



Sutton & Barto

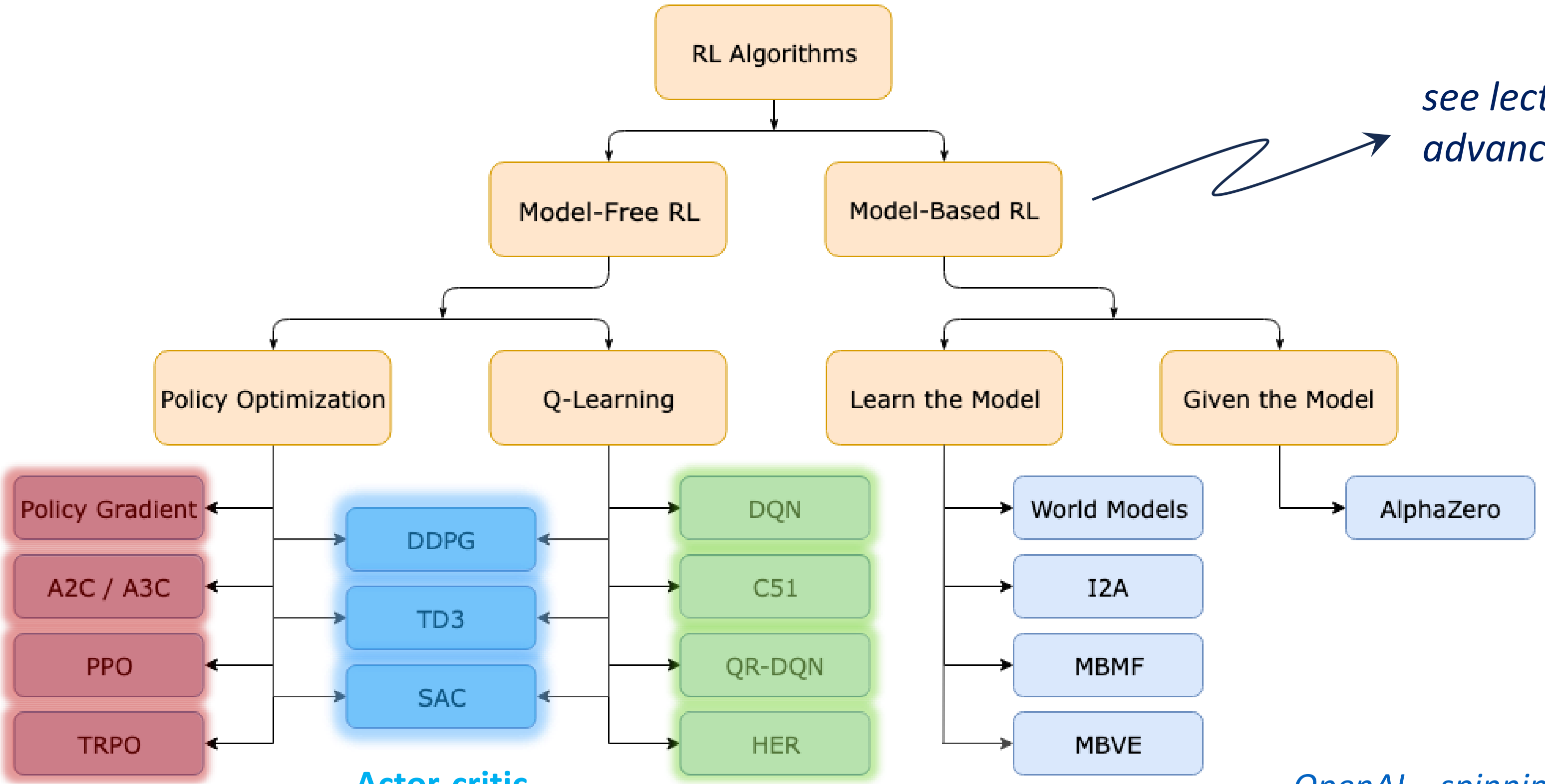


Contents

Algorithms

RL zoo

There are **many ways** to **solve the RL problem** and **finding an optimal policy** in an environment



see lecture on advanced RL

Actor-critic methods

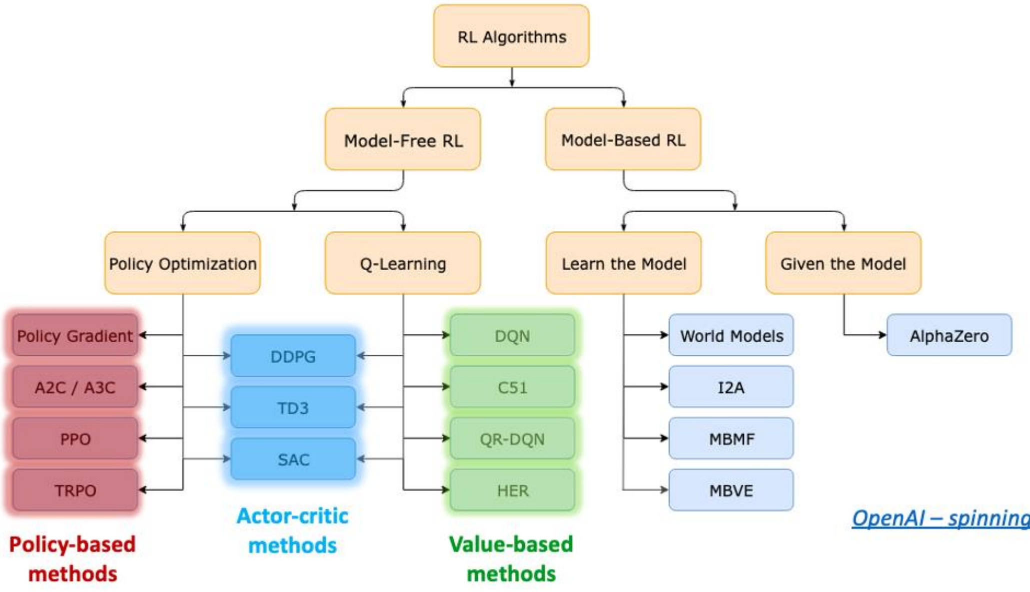
Policy-based methods

Value-based methods

[*OpenAI – spinning up*]

Algorithms
RL taxonomy

- **Value-based methods**
  - Agent **learns a value function** that **estimates expected return**
  - **Policy is indirectly obtained** from value function
  - *E.g.: Deep Q-learning (DQN)*
- **Policy-based methods**
  - Agent **directly optimises** parameters of a **policy** function
  - *E.g.: Proximal Policy Optimisation (PPO)*
- **Actor-critic scheme**
  - Combines value-based and policy-based methods
- **On- vs off-policy methods**
- **Model-free vs model-based algorithms**



[OpenAI – spinning up]

Contents

Value-based methods
Value functions

Return

$$G_t = \sum_{k=0} \gamma^k \, r_{k+t+1}$$

- **Value functions estimate "how good it is"** for the agent …

"… to be in state s given that we follow policy π?"

State-value function

$$V^\pi : \mathcal{S} \rightarrow \mathbb{R}$$

$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s]$$

"… to take action a in state s given that we follow policy π?"

State-action value function

(= "Q-function")

$$Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a]$$

- "Goodness" is measured in terms of **return expected following that policy**
- The **value functions associated** with the (an) **optimal policy** π^* are denoted V^* and Q^*, respectively

$$V^*(s) = \max_{a'} Q^*(s, a')$$

Value-based methods
Q-learning

State-action value function
(= "Q-function")

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t \mid s_t = s, a_t = a]$$

Q-learning

The **goal of Q-learning** is to deduce the **optimal policy** by learning the **optimal state-action value function** $Q^*(s,a)$ first

- **Once** $Q^*(s,a)$ **is known**, it is easy to **read off the best policy** *(= greedy policy)*

$$\pi^*(s) = \arg\max_{a'} Q^*(s,a')$$

i.e.: "in a given state, what is the best action to take to maximise return?"

- **How to learn** $Q^*(s,a)$ **?**

Value-based methods
Bellman optimality equation

$$G_t = \sum_{k=0} \gamma^k\, r_{k+t+1} = r_{t+1} + \gamma \underbrace{\sum_{k=0} \gamma^k\, r_{k+t+2}}_{G_{t+1}}$$

Bellman optimality equation

$s = s_t$
$a = a_t$
$s' = s_{t+1}$

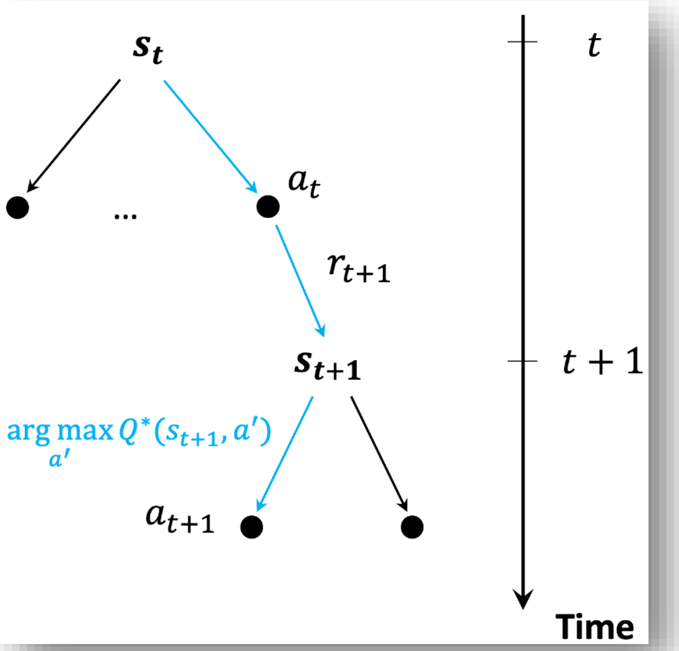
$V^*(s') = \mathbb{E}_{\pi^*}[G_{t+1}]$

$V^*(s') = \max_{a'} Q^*(s', a')$

weighted sum over all possible next states s' under action a

assuming deterministic environment

$$\begin{aligned} Q^*(s,a) &= \mathbb{E}_{\pi^*}[G_t] \\ &= \mathbb{E}_{\pi^*}[r_{t+1} + \gamma G_{t+1}] \\ &= \mathbb{E}_{\pi^*}[r_{t+1} + \gamma \max_{a'} Q^*(s',a')] \\ &= \sum_{s'} \mathcal{P}^a_{ss'}[R(s,s',a) + \gamma \max_{a'} Q^*(s',a')] \\ &= r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \end{aligned}$$



- Bellman splits the trajectory into an **"immediate part"** and **"whatever follows beyond"**
- It allows us to apply the **Temporal Difference (TD) rule** when learning an **estimator** $\hat{Q}^*$ of the optimal Q-function

Value-based methods
Q-learning algorithm

Bellman optimality equation

$$Q^*(s_t, a_t) = r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a')$$

- **At $\boldsymbol{t = 0}$**: initialise $\hat{Q}^*(s, a)$, e.g. random, or all zeros
- **At every time step**
  - Let agent interact with environment to collect $(s_t, a_t, r_{t+1}, s_{t+1})$ following some behaviour policy π_b
    Typically, "ε-greedy": select greedy action with probability 1-ε, random otherwise
  - Update $\hat{Q}^*(s, a)$ based on TD rule using collected agent-environment interactions

This is how the new piece of information comes in –
and is the reason why the new target is a "better guess"

Q-learning update rule

learning rate

TD error

$$\hat{Q}^*(s_t, a_t) \leftarrow \hat{Q}^*(s_t, a_t) + \alpha \, [\underbrace{r_{t+1} + \gamma \max_{a'} \hat{Q}^*(s_{t+1}, a')}_{\text{target (new best guess)}} - \underbrace{\hat{Q}^*(s_t, a_t)}_{\text{old prediction}}]$$

- With **enough iterations** $\hat{Q}^*(s, a)$ will **converge** to the true $Q^*(s, a)$

Value-based methods
Q-learning remarks

Q-learning update rule

$$\hat{Q}^*(s_t,a_t) \leftarrow \hat{Q}^*(s_t,a_t) + \alpha\,[\underbrace{r_{t+1} + \gamma \max_{a'} \hat{Q}^*(s_{t+1},a')}_{\textit{target (new best guess)}} - \underbrace{\hat{Q}^*(s_t,a_t)}_{\textit{old prediction}}]$$

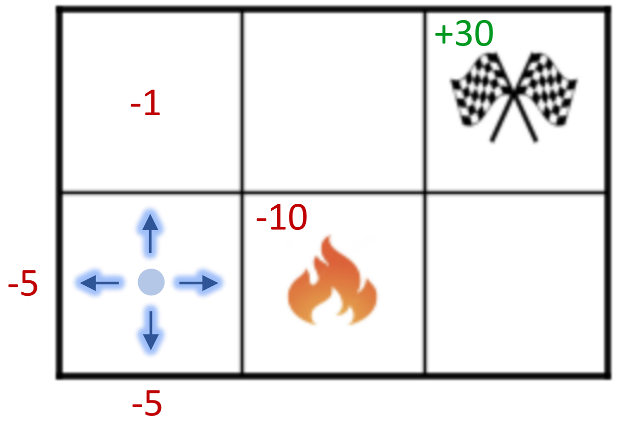
- Q-learning is an **iterative process**
  - Need a way to **track and update Q-values** for each state-action pair at every iteration
  - For **simple** (small, discrete) state-action spaces, we can use a **look-up table**
- Q-learning uses **bootstrapping**
  - Update of $\hat{Q}^*(s,a)$ uses a **target** that is **itself based on an estimate**
    "shooting at a moving target": training can be unstable as target also updates frequently
  - Typically solved using **two separate Q-estimators**:
    "target Q" and "online Q" with periodic synchronisation

by DALL-E

- Q-learning is an **off-policy method**
  - How agent chooses its actions *(= behaviour policy π_b)* to collect samples $(s_t, a_t, r_{t+1}, s_{t+1})$ does not necessarily match the policy, or associated value function, we are trying to learn *(= target policy π_t)*
  - Allows for **experience replay** (recycling previous samples), **improving sample efficiency**
  - More on that later ☺ …

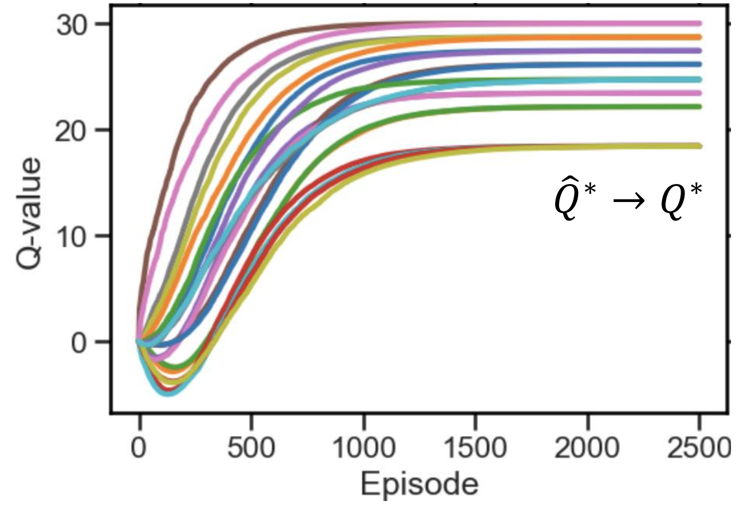
Value-based methods

Q-table example



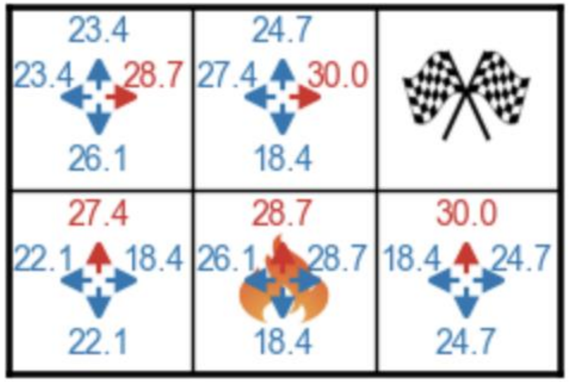
Episode 2400

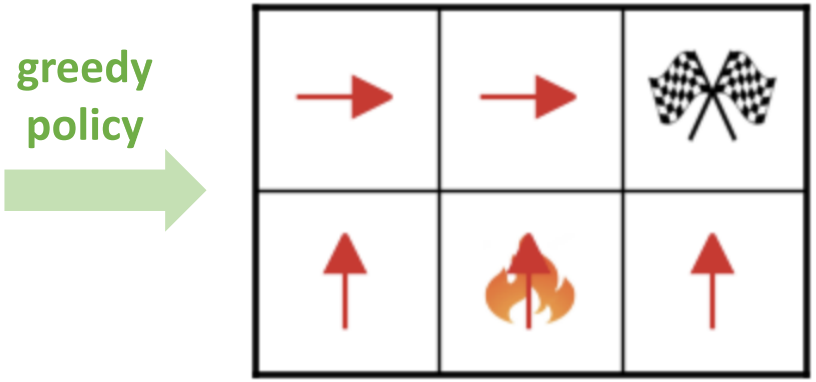
s \ a	up	down	left	right
(0, 0)	27.4	22.1	22.1	18.4
(0, 1)	23.4	26.1	23.4	28.7
(1, 0)	28.7	18.4	26.1	28.7
(1, 1)	24.7	18.4	27.4	30.0
(2, 0)	30.0	24.7	18.4	24.7
(2, 1)	0.0	0.0	0.0	0.0



$$\hat{Q}^*(s_t, a_t) \leftarrow \hat{Q}^*(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} \hat{Q}^*(s_{t+1}, a') - \hat{Q}^*(s_t, a_t)\right]$$

$\hat{Q}^* \approx Q^*$



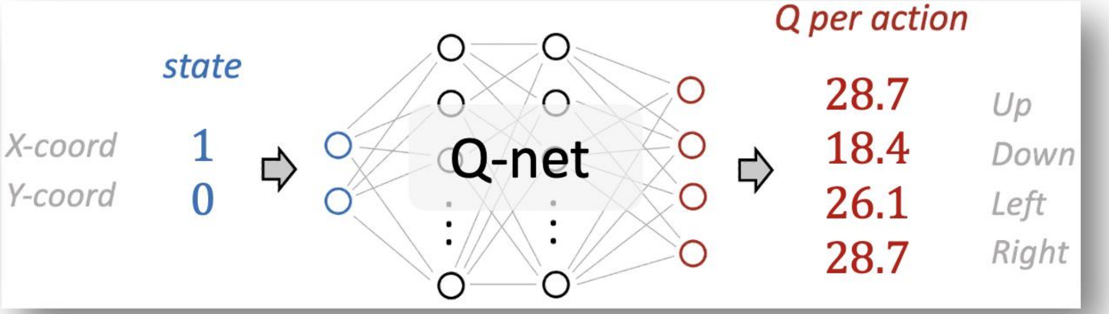


Value-based methods
Deep Q-learning (DQN)

- Q-function of **continuous or very large** $\mathcal{S}$ can no longer be represented by a look-up table
- **Replace table by neural net**: deep Q-learning (DQN)
  - ➢ **Universal function approximator** and great **interpolator** (e.g. for unseen states)
  - ➢ Q-net is **mapping from state to Q-values** of all possible actions
  - ➢ Train network weights using **Q-learning update rule**
- Developed by **DeepMind in 2013** and applied to playing Atari games – many at super-human level (DQN paper)
- **N.B.:** only for **discrete** $\mathcal{A}$ – need one output node per action …

Episode 2400

s \ a	up	down	left	right
(0, 0)	27.4	22.1	22.1	18.4
(0, 1)	23.4	26.1	23.4	28.7
(1, 0)	28.7	18.4	26.1	28.7
(1, 1)	24.7	18.4	27.4	30.0
(2, 0)	30.0	24.7	18.4	24.7
(2, 1)	0.0	0.0	0.0	0.0

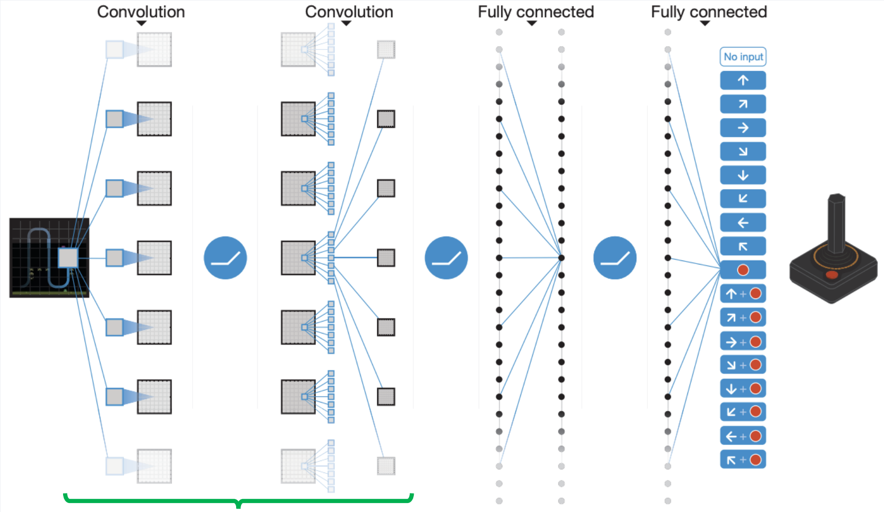


Value-based methods

Example: DeepMind's RL for Atari games

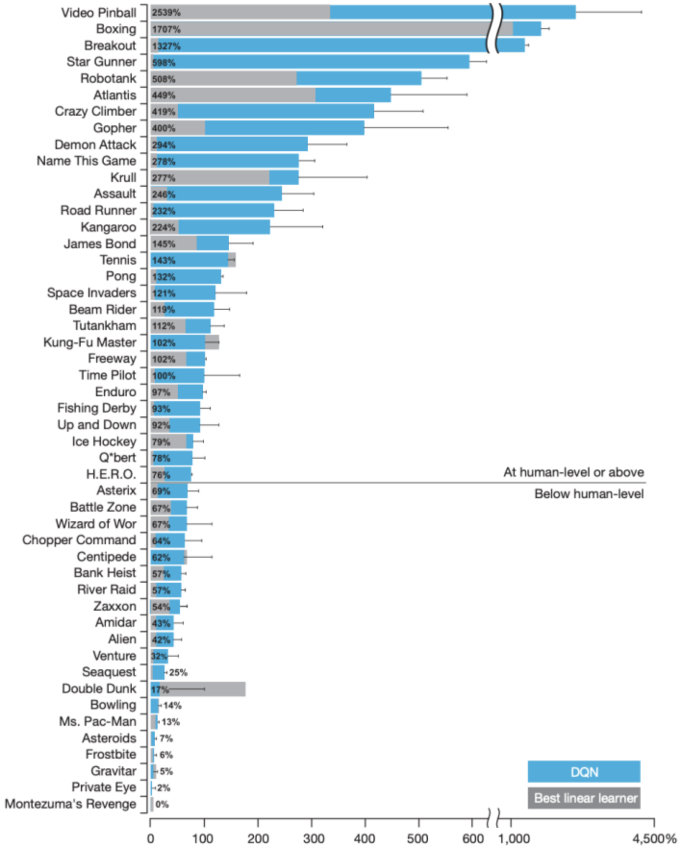
Q-net architecture

state s
sequence of
video frames



“feature extraction”
layers

$\boldsymbol{Q(s, a)}$
for discrete set
of joystick /
button actions



source

Value-based methods

Q-table vs DQN: pros, cons, limitations

Q-table

+ Easy to understand and validate
- Discrete $\mathcal{S}$, $\mathcal{A}$ spaces only
- Relatively small $\mathcal{S}$, $\mathcal{A}$ spaces only

DQN

+ Large, continuous $\mathcal{S}$ possible
+ No need to visit all states during training: neural nets are great interpolators
- Discrete and relatively small $\mathcal{A}$
- Training may be unstable and hard to validate (incl. convergence)

- Many real-world problems require **continuous** $\mathcal{S}$ ***and*** **continuous** $\mathcal{A}$
  $\Rightarrow$ typically use **policy gradient** or **actor-critic** methods
- **Other function approximators:** (quantum) Boltzmann machines, …

Contents

Policy gradient methods

In a nutshell

- Policy represented by **parameterised function** $\pi_\theta(a \mid s)$
  θ: for example weights of a neural network
- Goal is to **directly optimise** θ s.t. π_θ **maximises expected return** over trajectories τ

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[G(\tau)]$$

  - Perform **gradient ascent** for policy parameters θ

$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta J(\theta)$$

  - Gradient can be calculated using the **policy gradient theorem**

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_{t=0} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)]$$

- **N.B.:** policy gradient algorithms typically differ in what they use for Ψ_t
  - It can e.g. be the **return**, a so-called **advantage function**, a **baseline corrected return**, etc.
  - More on that in the advanced lecture

Contents

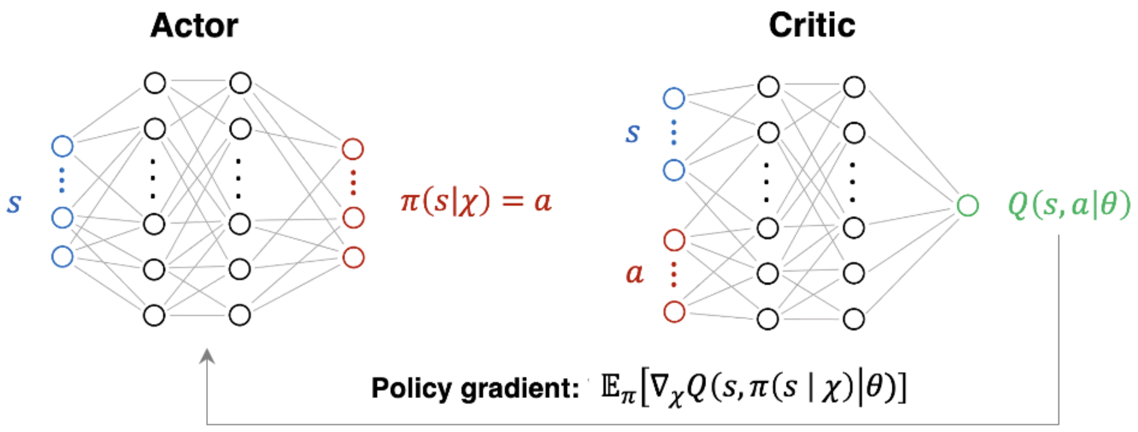
Algorithms

Actor-critic scheme

- Introduce an **actor** *(= policy)* and a **critic** *(= value function estimator)* **combining** a **value-based** with a **policy-gradient** approach
- Typically, actor and critic are **represented by** (fairly small) **neural nets**, trained simultaneously
- Can solve the **continuous state *and* action** problem
- **Various algorithms** exist (DDPG, TD3, SAC, …), e.g. handling **exploration-exploitation** differently, improving **convergence behaviour, …**

Actor *(policy net)*

- Represents the **target policy** π_t to be **learned**
- For each (continuous) **state** s, it proposes a (continuous) **action** a
- Its parameters χ are updated through the **policy gradient**



For given state s, how does the actor have to adjust its parameters χ to propose an action a that results in larger $Q(s, a)$?

Critic *(Q-net)*

- **Learns Q-function** and **evaluates quality** of the (s, a) pair proposed by actor net
- Parameters θ are updated using **TD rule** (like in Q-learning)
- **Feeds back to the actor** via policy gradient

Algorithms
On- vs off-policy methods

- A priori, training an RL agent employs **two policies**
  - **Behaviour policy** π_b: policy the agent follows to **select action at every time step** during data collection
  - **Target policy** π_t: policy we **aim for the agent to learn** and optimise towards
- **RL algorithms can be …**

On-policy

- $\pi_t = \pi_b$
- Agent updates and **learns the same policy** (or value function) that it uses to interact with the environment
- Example: **SARSA**

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

*TD target **based on action** a_{t+1} **that was selected by** π_b and **applied** in the environment*

➡ *Learning Q-function **associated with** π_b*

Off-policy

- $\pi_t \neq \pi_b$
- Agent updates and **learns a different policy** (or value function) than it uses to interact with the environment
- Example: **Q-learning**

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\,[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

*TD target **always** based on current best guess of **greedy action**, **independent of action selected and applied** in the environment by π_b*

➡ *Learning Q-function **associated with a greedy policy** π_t*

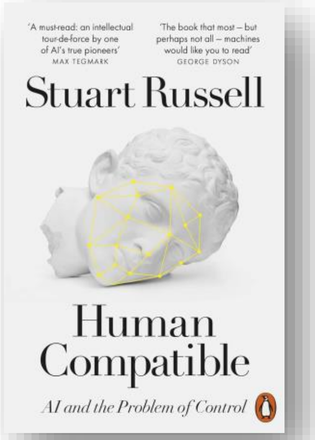
Algorithms
Experience replay

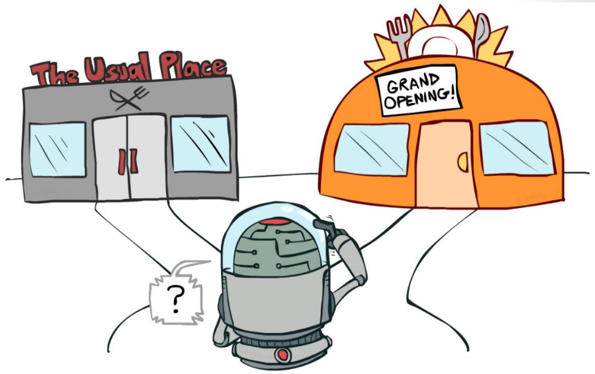
- **Off-policy** algorithms boast **improved sample efficiency**
  - They can learn from agent-environment interactions **collected according to *any policy***
  - **Experience replay**
    - Keep **buffer** of **past interactions** and **update value function on a batch of memories** at every training step
    - Different **sampling methods** exist to select and learn from past experiences **most efficiently**
- **On-policy** algorithms can only learn **"online"**
  - Learning step relies on **samples collected according to *currently valid policy***
  - We have to **discard past experiences** as they were **collected** according to a **different policy**
- **On-policy** methods typically feature **more stable training** than off-policy algorithms

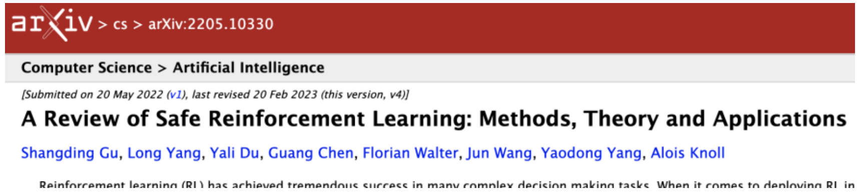
Contents

Challenges
Non-exhaustive list

- **Sample efficiency**
  - **How many agent-environment interactions** are required for training / convergence?
  - **Online training is not always possible:** sim2real & sim2real gap
- **Reward engineering**
  - **Alignment:** getting the objective right
    Making sure the agent does what we want it to do …
  - **Credit assignment problem**
    Which action contributed how to the reward?
- **Exploitation vs exploration dilemma**
- **State definition**
  - Markov property
  - Environments are sometimes only partially observable
- **Non-stationarity**
- **Safety & validation**
  - Particularly a concern during exploration
  - There are ways to add safety to RL agents





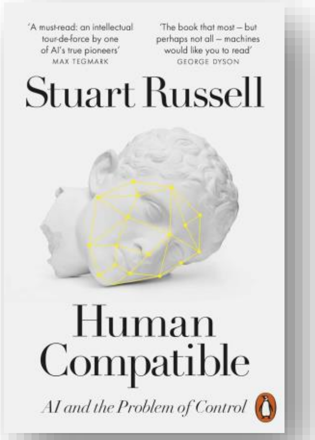


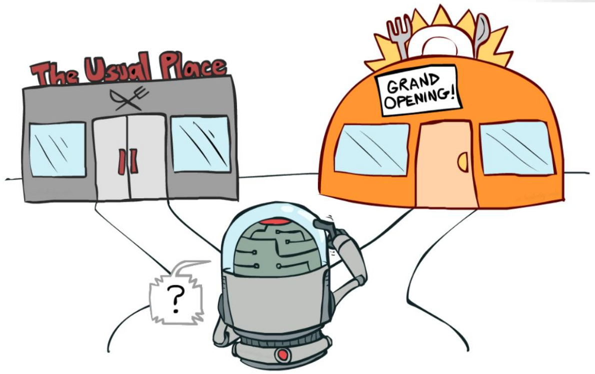
https://arxiv.org/abs/2205.10330

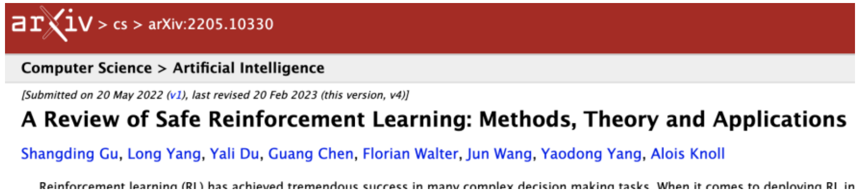
Challenges

Non-exhaustive list

- **Sample efficiency**
  - **How many agent-environment interactions** are required for training / convergence?
  - **Online training is not always possible:** sim2real & sim2real gap
- **Reward engineering**
  - **Alignment:** getting the objective right
    Making sure the agent does what we want it to do …
  - **Credit assignment problem**
    Which action contributed how to the reward?
- **Exploitation vs exploration dilemma**
- **State definition**
  - Markov property
  - Environments are sometimes only partially observable
- **Non-stationarity**
- **Safety & validation**
  - Particularly a concern during exploration
  - There are ways to add safety to RL agents






arXiv > cs > arXiv:2205.10330

Computer Science > Artificial Intelligence

[Submitted on 20 May 2022 (v1), last revised 20 Feb 2023 (this version, v4)]

A Review of Safe Reinforcement Learning: Methods, Theory and Applications

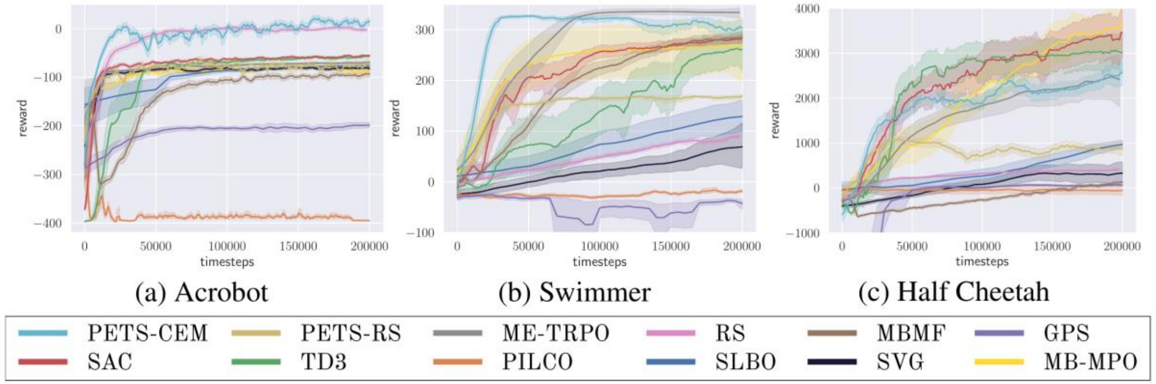
Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, Yaodong Yang, Alois Knoll

https://arxiv.org/abs/2205.10330

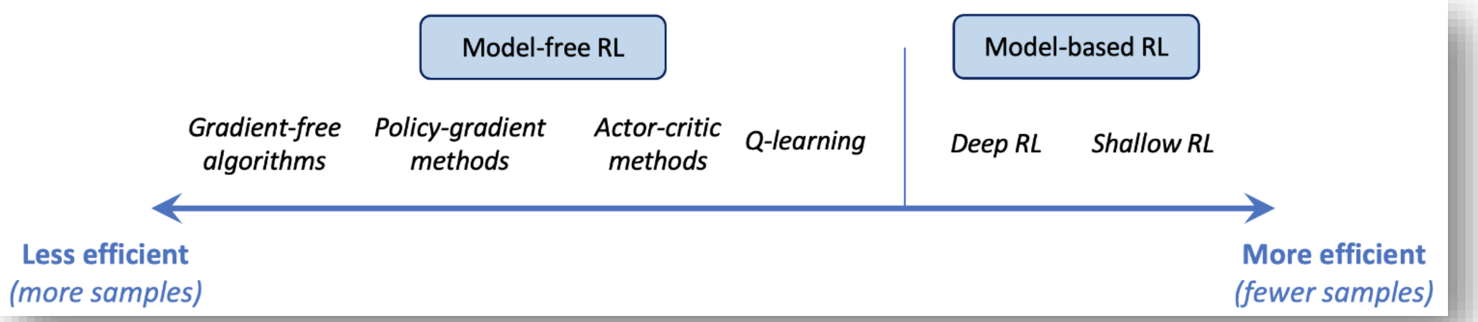
Challenges

Sample efficiency

- **How many agent-environment interactions** are required for the **value function / policy to converge**?
- **Depends heavily on choice of algorithm**
  - Off- vs on-policy algorithms
  - Online vs. offline RL
  - Model-free vs model-based RL
- **Reliable simulations / surrogate models**
  - Train RL agent on model, then deploy in real world (sim2real)
  - Model can be based on simulations, measurements, or both
  - sim2real gap can be a problem



source



Challenges

Exploration-exploitation dilemma

- To learn the **best policy** in an efficient manner, algorithms need to have a good a **balance between**
  - **exploration:** trying out different actions to discover their effects and rewards
  - **exploitation:** picking *best-known* action to maximise return
- **Why?**
  - During training, **best-known action** is typically **not** yet the **true best action**
  - Keep **some degree of exploration** to potentially discover **more rewarding actions** and avoid settling for **suboptimal policy**
  - **Too much exploration** can **slow down training** progress
- Different algorithms use **different techniques to balance out exploration and exploitation**, e.g. ε-greedy with decay, entropy-based methods, …

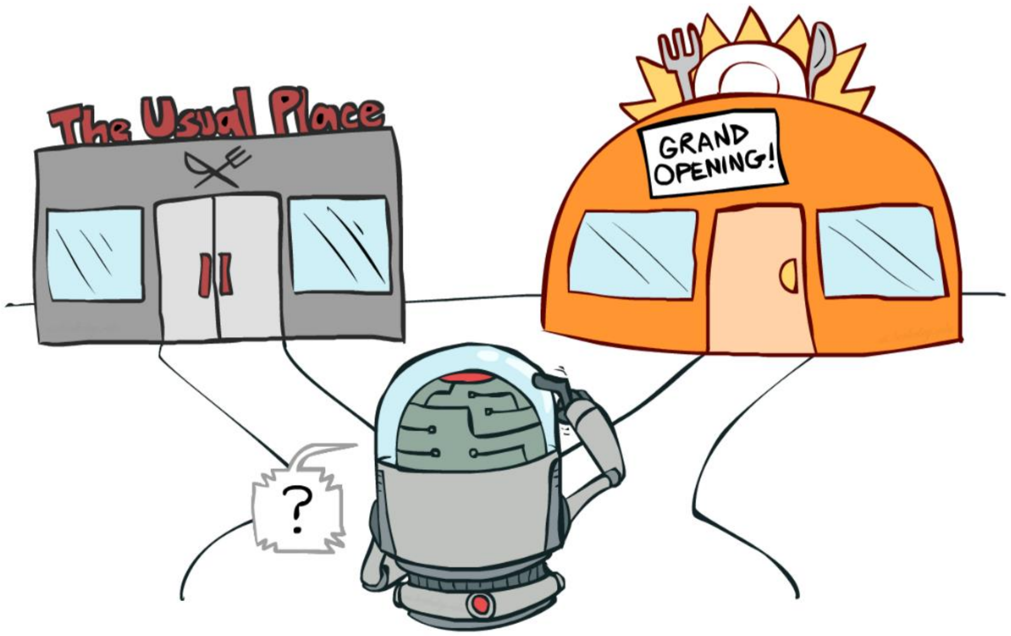


image by Berkeley AI course

Summary

- Reinforcement learning (RL) **solves decision-making problems** and optimises an agent for **best behaviour** *(= optimal policy)* in an environment, i.e. **maximising expected return**
- Formally, RL is based on **Markov Decision Processes** and the **reward hypothesis**
- **RL algorithms employ different techniques**
  - **Value-based methods:** learn a value function that estimates expected return to deduce policy indirectly
  - **Policy-based methods:** optimise parameters of a policy directly for highest expected return
  - **Actor-critic:** combine learning policy and value functions

  Functions to be learned are typically approximated by means of neural nets
- Some algorithms are **suitable only for discrete state-action spaces**
- We distinguish between **on- and off-policy algorithms:** behaviour vs target policy
- RL also faces **many challenges**
  - Balancing **exploration vs exploitation**, **sample efficiency**, dealing with **partially observable systems**, etc.
  - Some will be addressed in the **advanced RL lecture**

Further reading

- R.S. Sutton and A.G. Barto, "Reinforcement learning - an introduction", Book, 2nd edition, 2020.
- S. Levine, Deep Reinforcement Learning, Lecture, UC Berkeley, 2022.
- D. Silver, Reinforcement learning, Lecture, University College London (UCL), 2015.