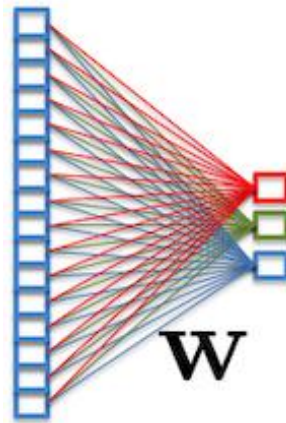
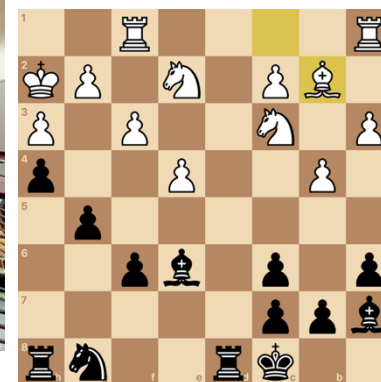
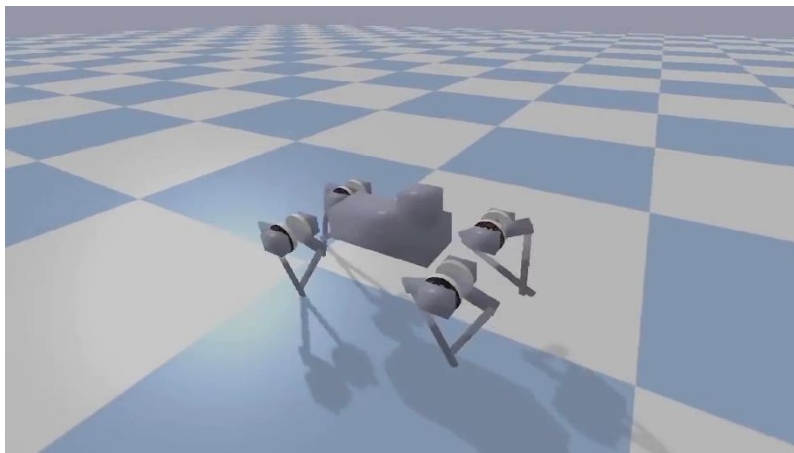


# AI and Data Analytics

## 5.1 Reinforcement Learning

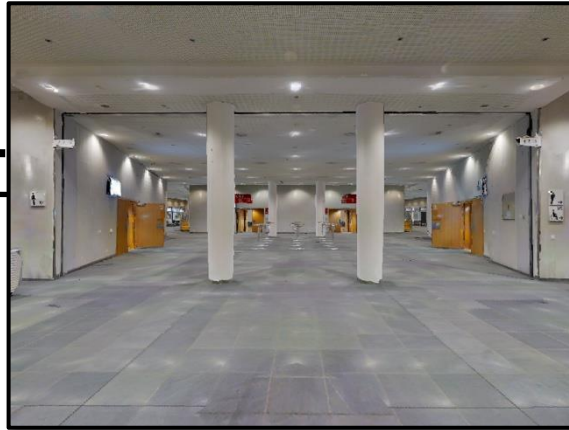


# Learning to control



# Reinforcement learning

Environment

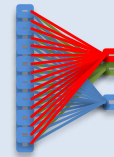


State or  
observation  
 $s_t$

Reward  
 $r_t$

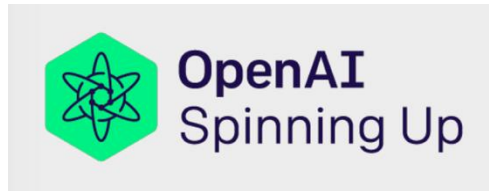
Action  
 $a_t$

Agent



$\pi_\theta$

# Literature



This presentation is largely inspired by the open-IA Tutorial on RL:

[https://spinningup.openai.com/en/latest/spinningup/rl\\_intro.html](https://spinningup.openai.com/en/latest/spinningup/rl_intro.html)

A lengthier introduction into Reinforcement learning:

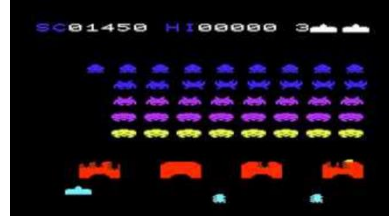


Dimitri P. Bertsekas, "Reinforcement Learning and Optimal Control », 2019.

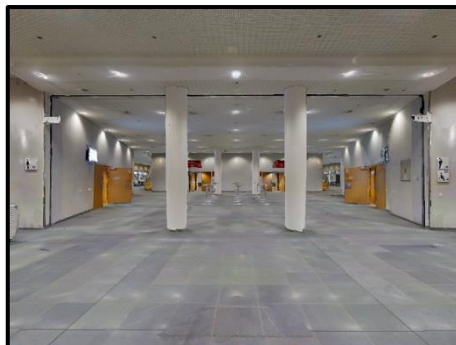
# States

An environment is in a given state  $s_t$  at any instant  $t$ .

**Fully observable problems** — the state is observable to the agent at any time, e.g. board games (chess, Go), 2D computer games with visible full screen (Pacman).



**Partially observable problems** — only an observation is returned to the agent at each instant, not the full state.  
Examples: robot navigation; 3D computer games with ego-centric view.



# Actions

At each time  $t$ , the agent takes an action  $a_t$  from an action space  $\mathcal{A}$ .

## Discrete action spaces

- Chess:  $\mathcal{A} = \{a1, Qf3, Ra1, bf3, \dots\}$ .
- Go:  $\mathcal{A} = [1 \dots 19] \times [1 \dots 19]$ .
- Dialogue:  $\mathcal{A} = \text{words or characters}$ .

## Continuous action spaces

- Robotic arm control:  $\mathcal{A} = \text{motor commands}$
- Drone (UAV) control:  $\mathcal{A} = \text{motor commands}$

# Policies

The policy  $\pi$  of an agent is a function which

- chooses an action given that the agent is in a current state;
- or, in the stochastic case, it assigns a probability to an action:

$$\pi(a_t | s_t)$$

# The reward function

After taking action  $a_t$  in state  $s_t$ , the agent is rewarded with a scalar reward  $r_t$ :

$$r_t = R(s_t, a_t, s_{t+1})$$

or, simpler:

$$r_t = R(s_t, a_t)$$

We are most often interested in the accumulation of the reward over time using a horizon  $T$ :

$$R(\tau) = \sum_{t=0}^T \gamma^t r_t$$

We discount rewards which are further away in the future by a factor  $\gamma$ .

# Markov Decision Processes

Mathematically, the situation we just presented is formalized as a Markov Decision Process (MDP)

$$M = \{\mathcal{S}, \mathcal{A}, P, R, p_0\}$$

where

- $\mathcal{S}$  is the set of states.
- $\mathcal{A}$  is the action space.
- $P(s_{t+1}|s_t, a_t)$  is a probability distribution governing state transitions.
- $R(s_t, a_t, s_{t+1})$  is the reward function.
- $p_0$  is the distribution of the initial state  $s_0$ .

MDPs, like Markov Chains, satisfy the **Markov property**: the outcome depends only on the current state, and not on the full history.

# Trajectories

A trajectory  $\tau$  is a sequence of states  $s_t$  and actions  $a_t$  taken by an agent following a policy. It can be assigned a probability:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t) \pi(a_t|s_t)$$

Given a trajectory, this probability only depends on the environment ( $P$  of the MDP).

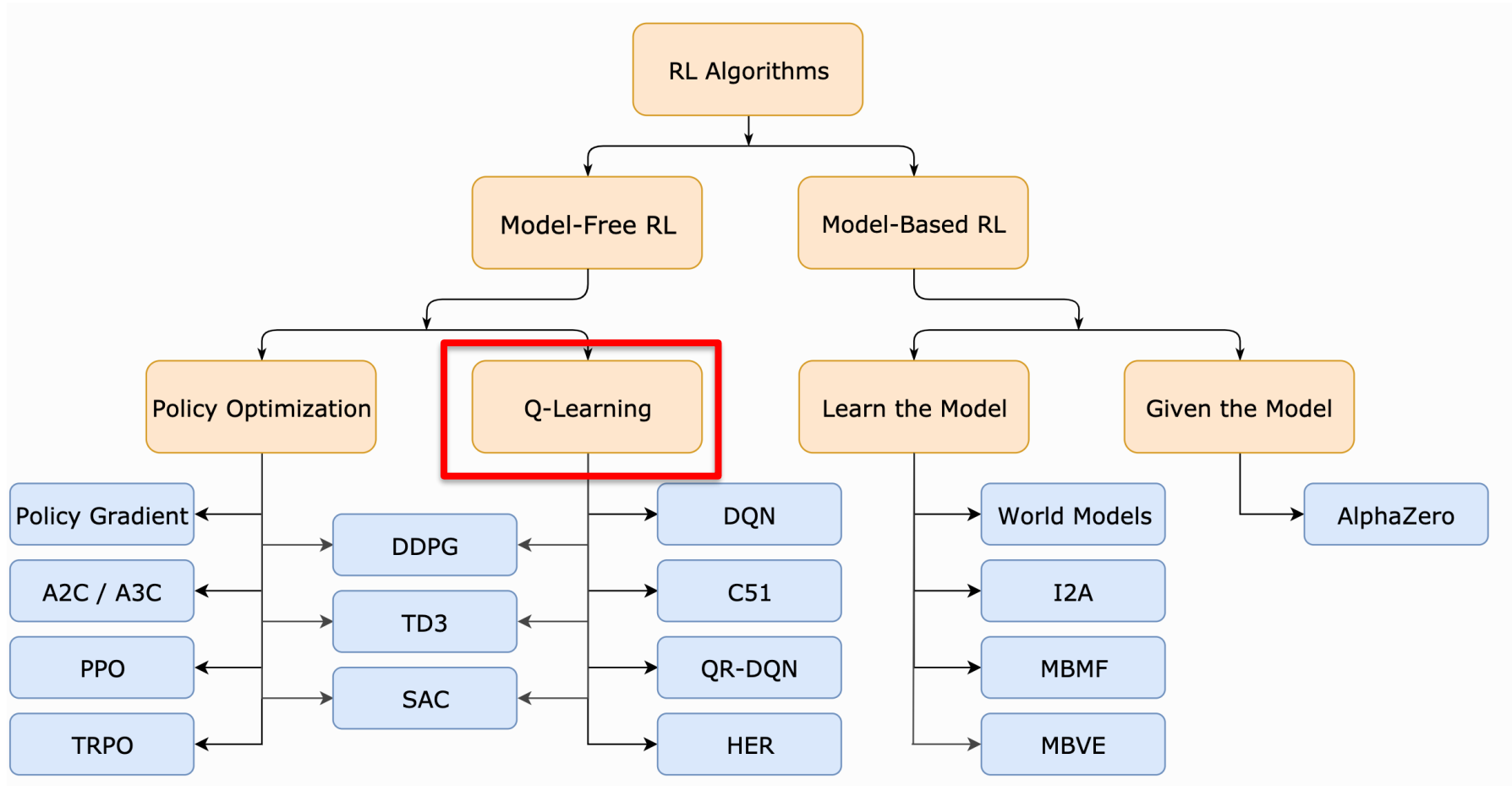
# What is the objective?

Our goal is to learn a policy  $\pi_\theta$  which optimizes the cumulative reward:

$$\begin{aligned} J(\pi_\theta) &= \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)] \\ &= \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^{T-1} r_{t+1} \mid \tau \right] \end{aligned}$$

$\Rightarrow$  we want to take actions, which are not only of high reward immediately, but of long term **value**!

# Types of reinforcement learning algorithms



# Value functions

The **Value** of state  $s_t$ , given a policy  $\pi_\theta$ , is given as the expected cumulative return when following the given policy from this state:

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = s]$$

The optimal value function is obtained with the optimal policy:

$$V^*(s) = \max_{\pi} \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = s]$$

The **Action-value function**:

$$Q^\pi(s, a) = \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = s, a_0 = a]$$

And it's optimal version:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}_{\tau \sim \pi} [R(\tau) | s_0 = s, a_0 = a]$$

# Q-Learning

Q-Learning does not directly learn a policy  $\pi$ , but attempts in estimating the optimal Q-function  $Q^*(s, a)$ .

The optimal policy is then the one which selects the optimal action  $a^*$  in state  $s$  as follows:

$$a^*(s) = \arg \max_a Q^*(s, a)$$

$\Rightarrow$  we need to estimate  $Q^*(s, a)$ .

We only have access to the immediate reward  $r_t$  at each instant.

# Q-Learning

For a given policy  $\pi$ , we can estimate the Q-function with the **Bellman equation**:

$$Q^\pi(s, a) = \mathbb{E}_{s' \sim P} \left[ r(s, a) + \gamma \mathbb{E}_{a' \sim \pi} [Q^\pi(s', a')] \right]$$

But we are interested in the Bellmann equation for the **optimal Q-function**:

$$Q^*(s, a) = \mathbb{E}_{s' \sim P} \left[ r(s, a) + \gamma \max_{a'} Q^*(s', a') \right]$$

# Q-Learning: value iteration

We update the current estimate of  $Q^*(s, a)$  at each time step:  
The agent is in state  $s_t$ , performs an action  $a$ , gets a new state  $s'$  and a reward  $r(s, a)$ .

**Discrete state and action spaces:** we can keep the Q-function in a table and update it:

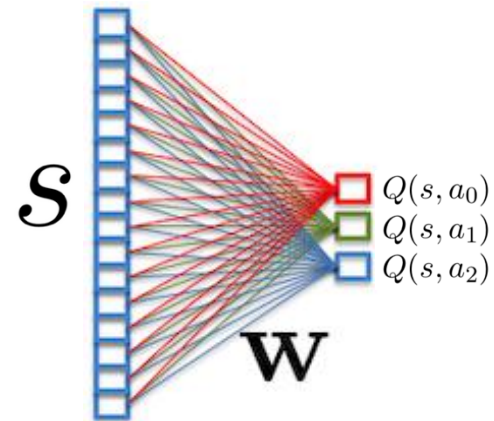
$$Q(s, a)^{[t+1]} = (1-\nu)Q^*(s, a)^{[t]} + \nu \left[ r(s, a) + \gamma \max_{a'} Q(s', a') \right]$$

where  $\nu$  is a learning rate.

# Q-Learning: value iteration

**Continuous state spaces:** we approximate the Q-function with a neural network:

$$f(s, \theta) = \mathbf{q} = \begin{bmatrix} Q(s, a_0) \\ Q(s, a_1) \\ Q(s, a_2) \\ \vdots \\ Q(s, a_N) \end{bmatrix}$$



We minimize a loss between the old value and the new one:

$$\mathcal{L} \left( Q(s, a), r(s, a) + \gamma \max_{a'} Q(s', a') \right)$$

and learn by gradient descent.

# Exploration vs. exploitation

In order to learn a reliable and useful value function, the agent needs to explore the state space and can't always take the action which is **currently** estimated as optimal  $\Rightarrow$  Exploration/exploitation trade-off.

$\epsilon$ -greedy strategy:

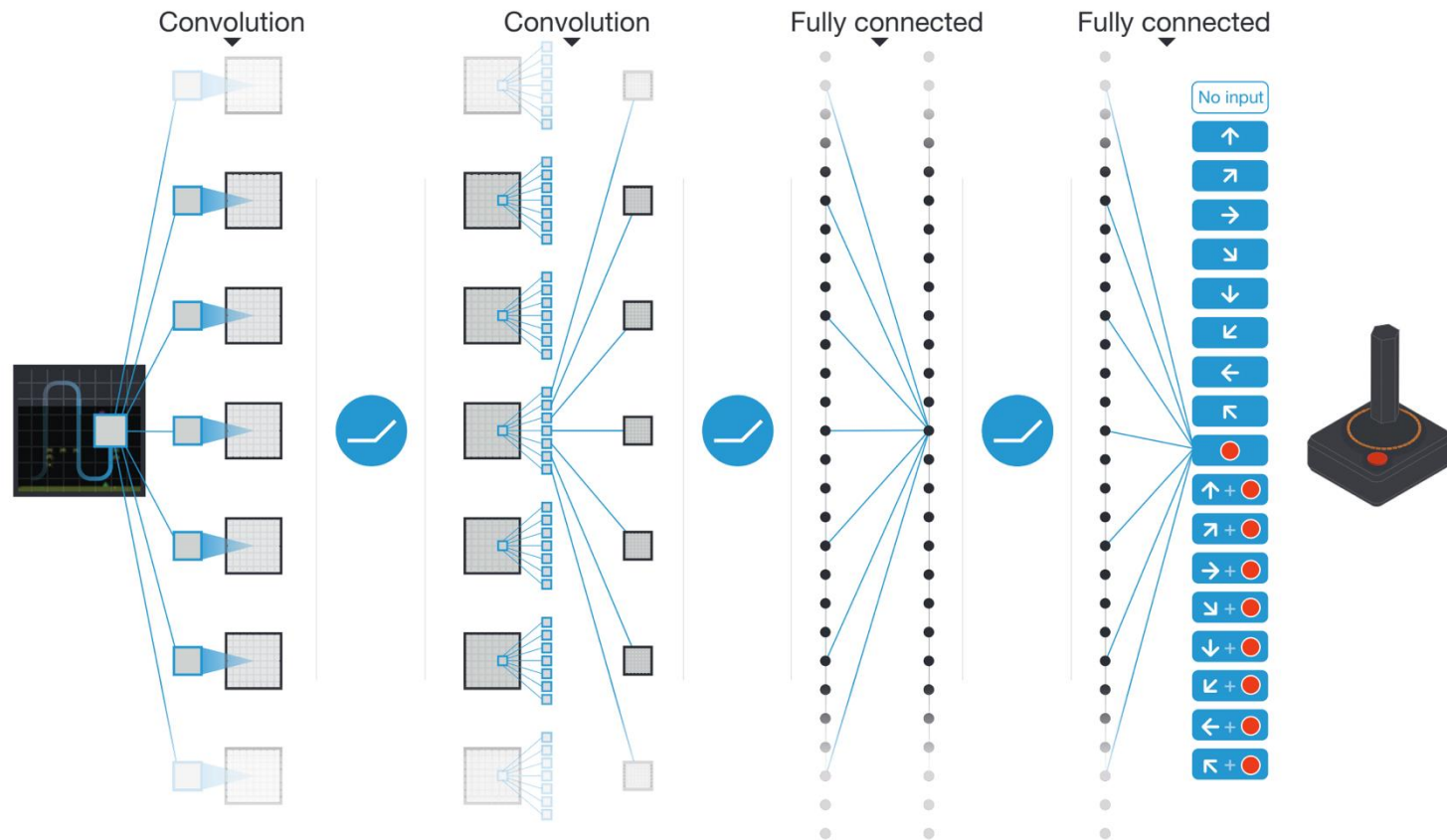
1. Draw a random number  $r \in [0, 1]$ .
2. Choose action  $a$  as:

$$a = \begin{cases} \text{random} & \text{if } r < \epsilon \\ \arg \max_a Q(s, a) & \text{if } r \geq \epsilon \end{cases}$$

Higher  $\epsilon$  chooses more exploration.

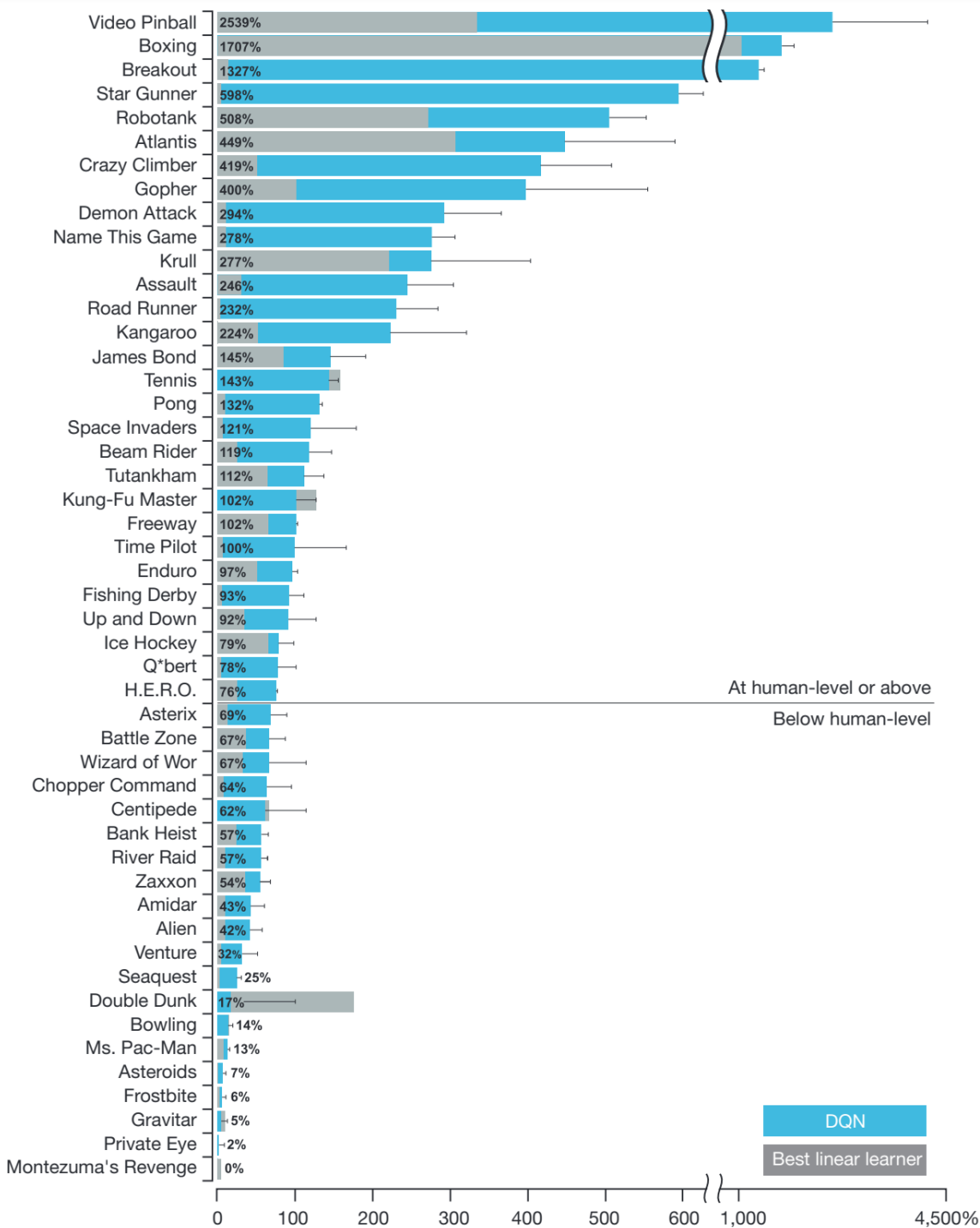
# Deep Q Learning

Deepmind's work on ATARI games in 2015 launched work in Deep Reinforcement learning.



[Mnih et al., Human Level Control Through Deep Reinforcement Learning, Nature, 2015]

# **Google DeepMind's Deep Q-learning**



At human-level or above

Below human-level

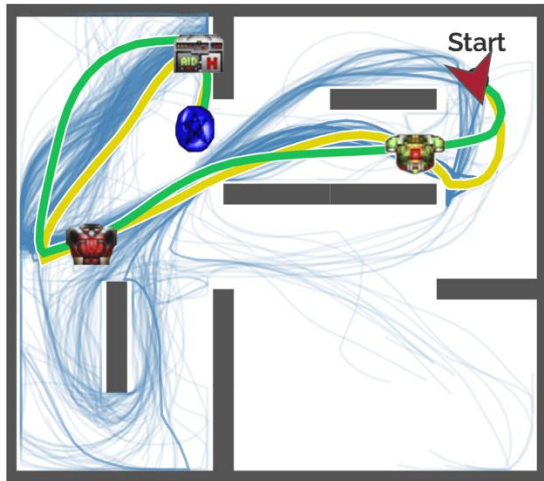
DQN

Best linear learner

[Mnih et al., Human Level Control Through Deep Reinforcement Learning, Nature, 2015]

# What if we Reduce the Memory of an Artificial Doom Player?

We built a Doom player AI 🚀 using Deep Reinforcement learning. While playing, it builds and updates an inner representation (memory) of what it sees from the game. This memory represents what the AI knows about the game, and is the root of each decision. Reducing the size of the memory 📦 could help the player learning to complete its task and thus lower its training time and energy consumption footprint. In this scenario, the player has to gather items in a specific order: Green Armor 🛡️ → Red Armor 🛡️ → Health Pack 📦 → Soul-sphere 🌌, with the shortest path possible.



- Human
- Full Memory
- Current



Input

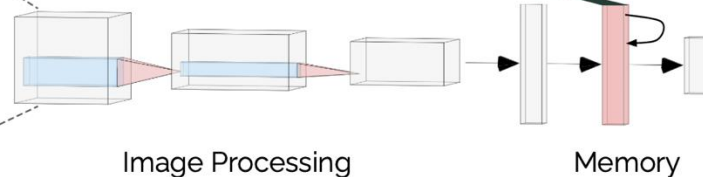
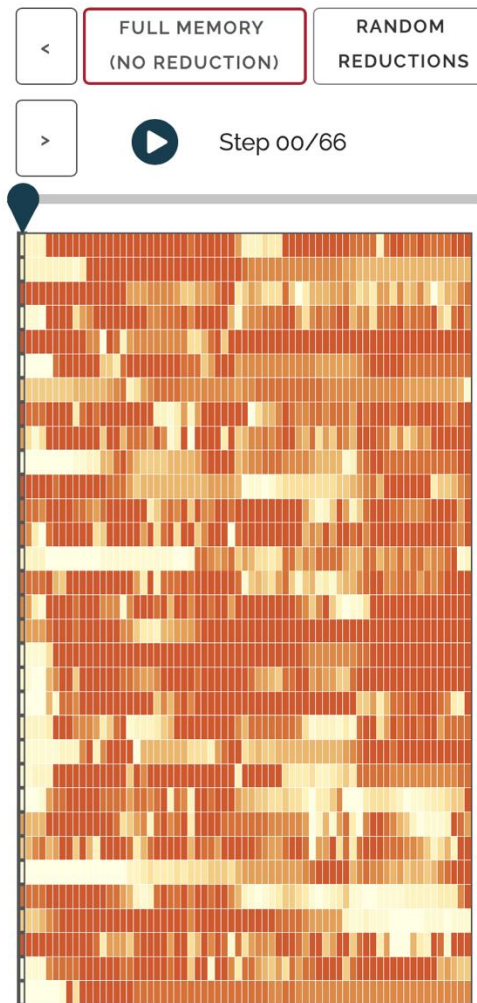


Image Processing

Memory

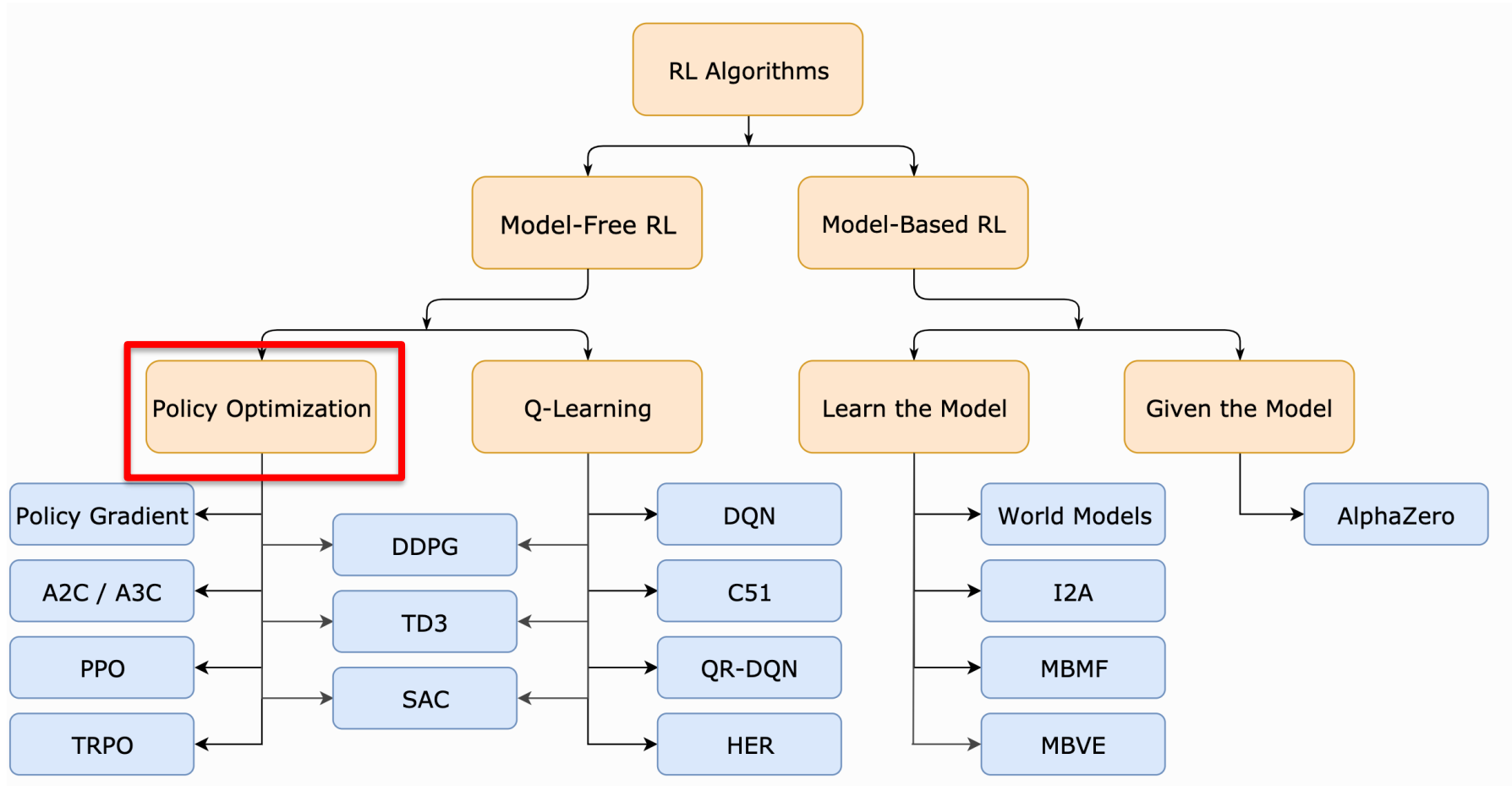
- ← ■
  - ■
  - ↑ ■
  - ↓ ■
  - ↖ ■
  - ↗ ■
- Output



<https://theo-jaunet.github.io/MemoryReduction/>

# Appendix: Policy gradient

# Types of reinforcement learning algorithms



# Policy Gradient

Policy gradient directly optimizes the expected cumulative reward over a parametrized policy function  $\pi_\theta$ :

$$\begin{aligned} J(\pi_\theta) &= \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)] \\ &= \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^{T-1} r_{t+1} | \tau \right] \end{aligned}$$

We would like to learn the policy network  $\pi_\theta$  with gradient descent:

$$\theta_{k+1} = \theta_k + \nu \nabla_\theta J(\pi_\theta)$$

How do we calculate the gradient  $\nabla_\theta J(\pi_\theta)$ ?

# Policy Gradient

$$\begin{aligned}\nabla_{\theta} J(\pi_{\theta}) &= \nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] \\&= \nabla_{\theta} \int_{\tau} P(\tau|\theta) R(\tau) \\&= \int_{\tau} \nabla_{\theta} P(\tau|\theta) R(\tau) \\&= \int_{\tau} P(\tau|\theta) \nabla_{\theta} \log P(\tau|\theta) R(\tau) \\&= \mathbb{E}_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log P(\tau|\theta) R(\tau)] \\ \nabla_{\theta} J(\pi_{\theta}) &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[ \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t|s_t) R(\tau) \right]\end{aligned}$$