Aircraft Control with Deep Reinforcement Learning

Gordon Rennie

http://www.flightgear.org

Content

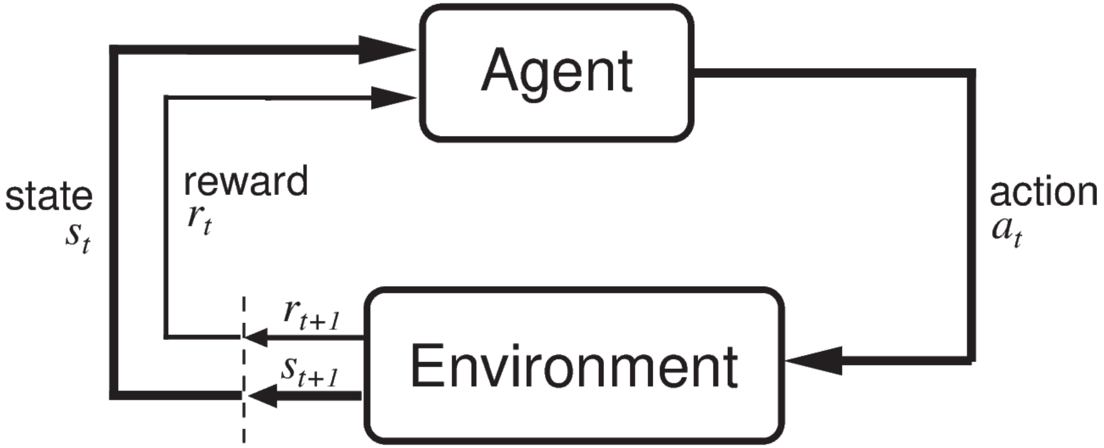
1. **Reinforcement Learning:**
   what is it?
2. **Deep** reinforcement learning:
   improving performance with deep neural nets
3. **Aircraft control** with distributed deep RL:
   an example application

Bonus:

- **Distributed** deep reinforcement learning:
  scaling up learning with parallel workers

Reinforcement Learning

Based around the agent-environment interaction:



Episode **trajectory**:

$$s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T$$

(Sutton and Barto, 1998)

Reinforcement Learning

Agents draw actions from their **policy**, π:

$$\pi(s) = a$$

The agent will learn to **maximise total reward**:

$$R_0 = \sum_{t=0}^{T} r_t$$

How good is a given state? We define an **action value function**:

$$Q(s_t, a_t) = \mathbb{E}[R_t | s = s_t, a = a_t, \pi]$$

‘given I am in state s_t at time t, and I select action a_t then follow my policy π, what future cumulative reward can I expect?’

Reinforcement Learning

Each action value recursively depends on the value of the next state:

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

$$= r_t + r_{t+1} + Q(s_{t+2}, a_{t+2})$$

We want to learn to maximise reward, with the **optimal policy**:

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

Problem: we don't know the true value function Q^*

Reinforcement Learning

Solution: estimate it from experience!

- start with an arbitrary Q
- follow policy π (select greedy a), receive experience s_t, a_t, r_t
- update value of Q recursively:

$$Q(s_t, a_t) = (1 - \alpha)\underbrace{Q(s_t, a_t)}_{\text{old estimate}} + \alpha\underbrace{(r_t + \arg\max_a Q(s_{t+1}, a))}_{\text{new estimate}}$$

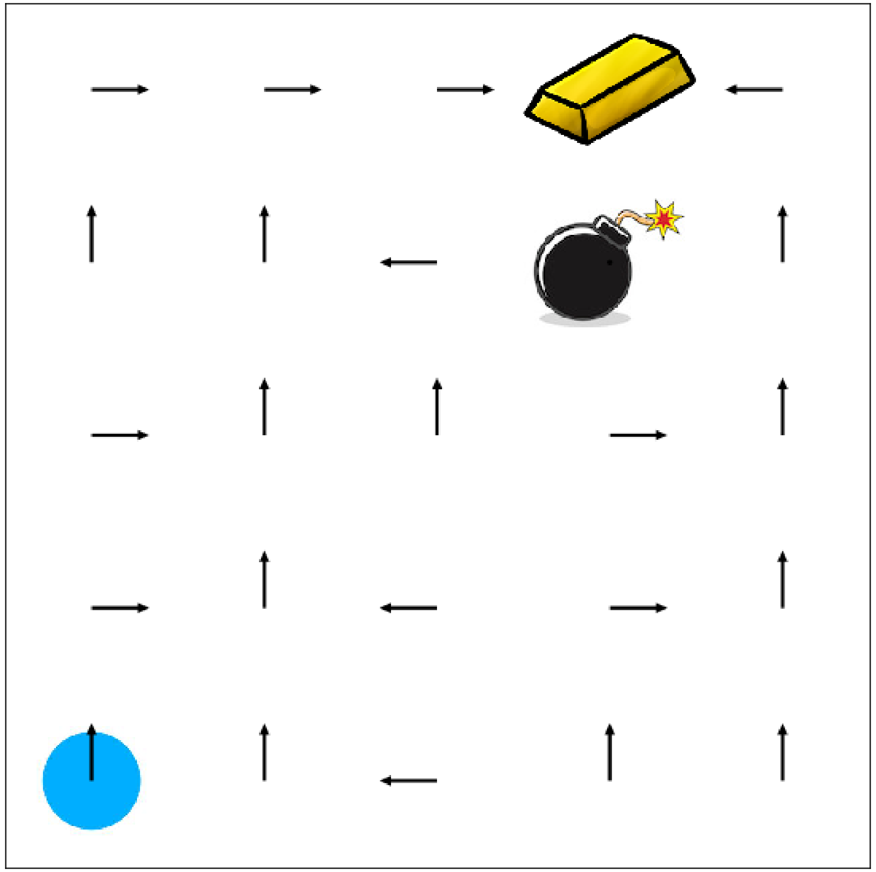
- estimate of Q improves → policy improves →
  estimate of Q improves → policy improves → …

Reinforcement Learning

Solving a toy block-world problem:

- Positive reward on gold
- Negative reward on bomb
- Small negative reward each step

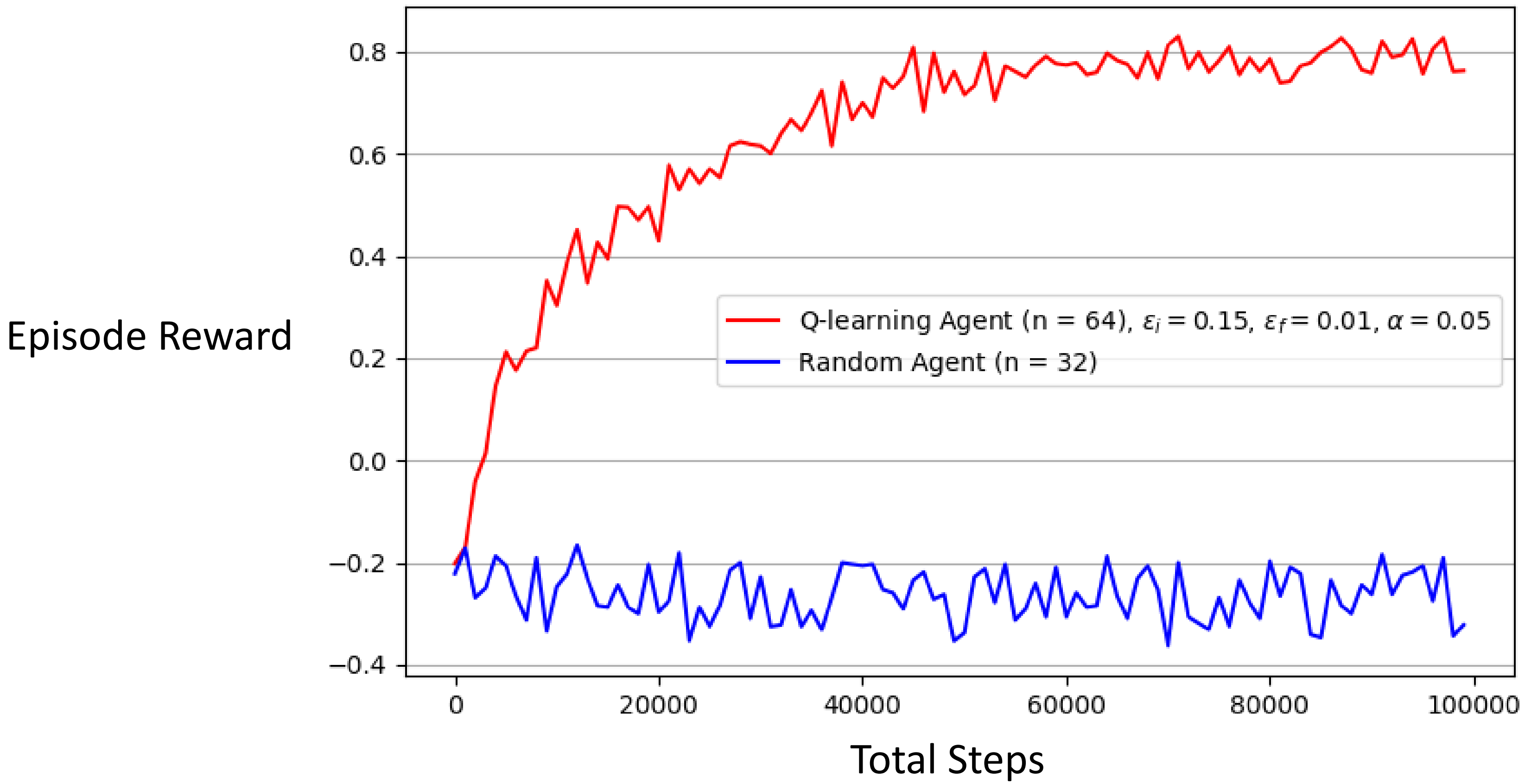
Visualisation shows highest value actions in each state.

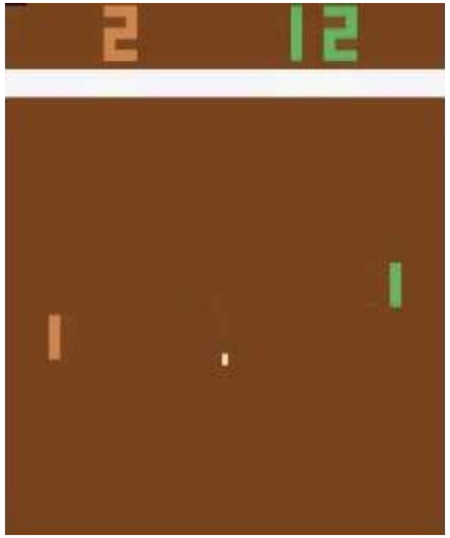


(Lichtenberg, 2018)

Reinforcement Learning

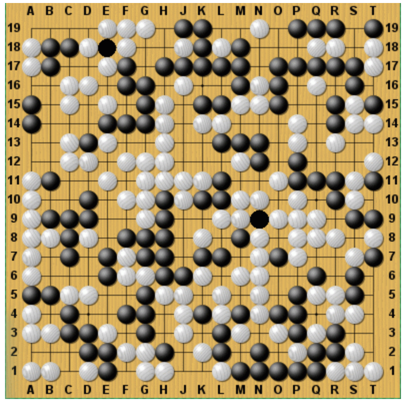
We track agent performance with a **learning curve:**

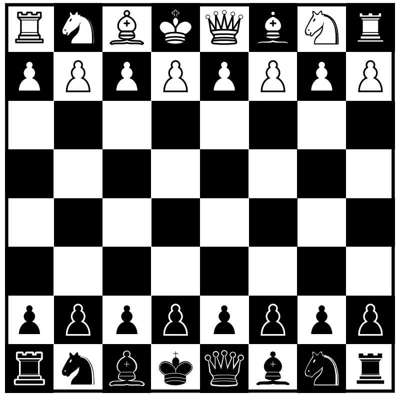


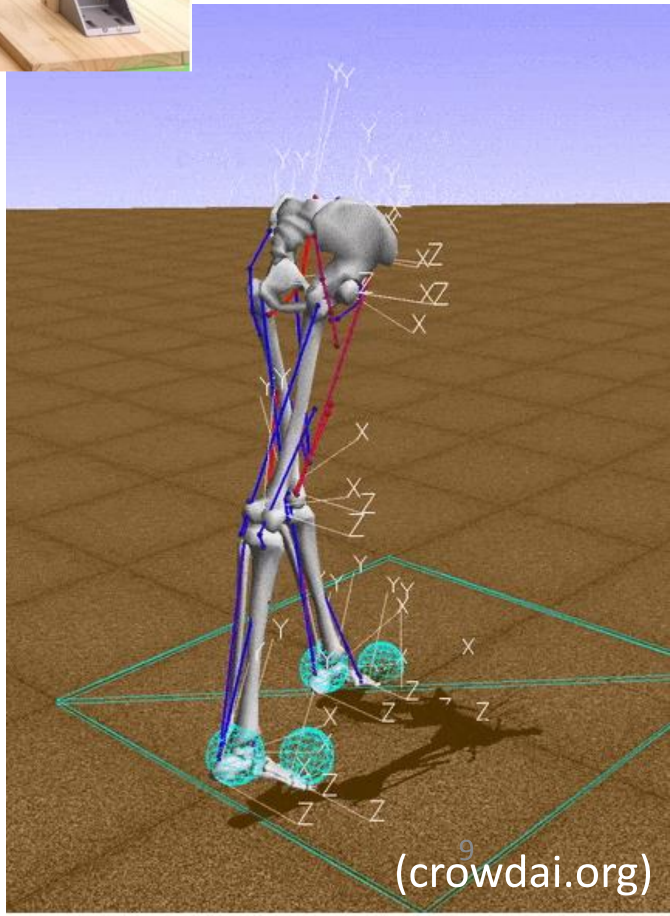


(Gu et al, 2016)

Environment







(crowdai.org)

Reinforcement Learning

Problem 1: approach scales poorly to problems with high complexity (many states and actions).

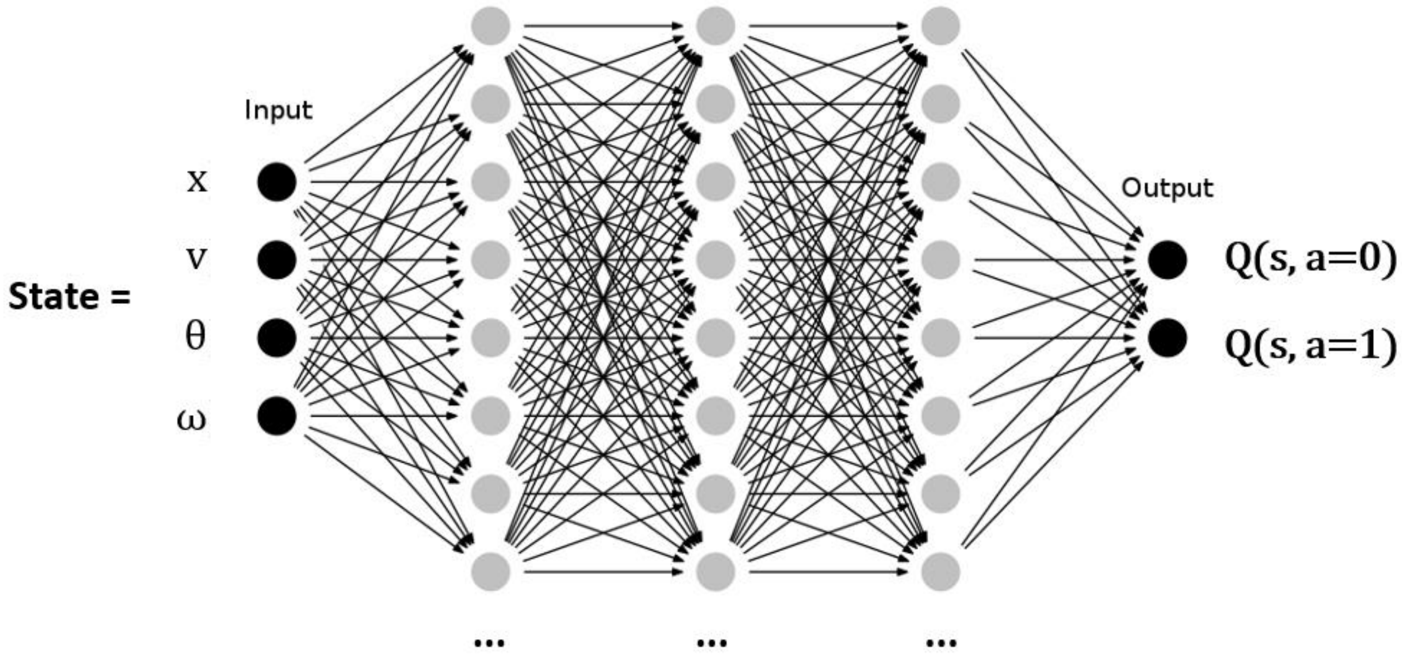
(a) we cannot experience every state/action to update values

(b) we would run out of memory even if we could!

Solution: learn a function which calculates value from an input (s, a)

We have a function approximation / supervised learning problem.

Deep Reinforcement Learning



Deep Reinforcement Learning

Train NN to match new estimates of $Q(s_t, a_t)$ (**bootstrapping**)

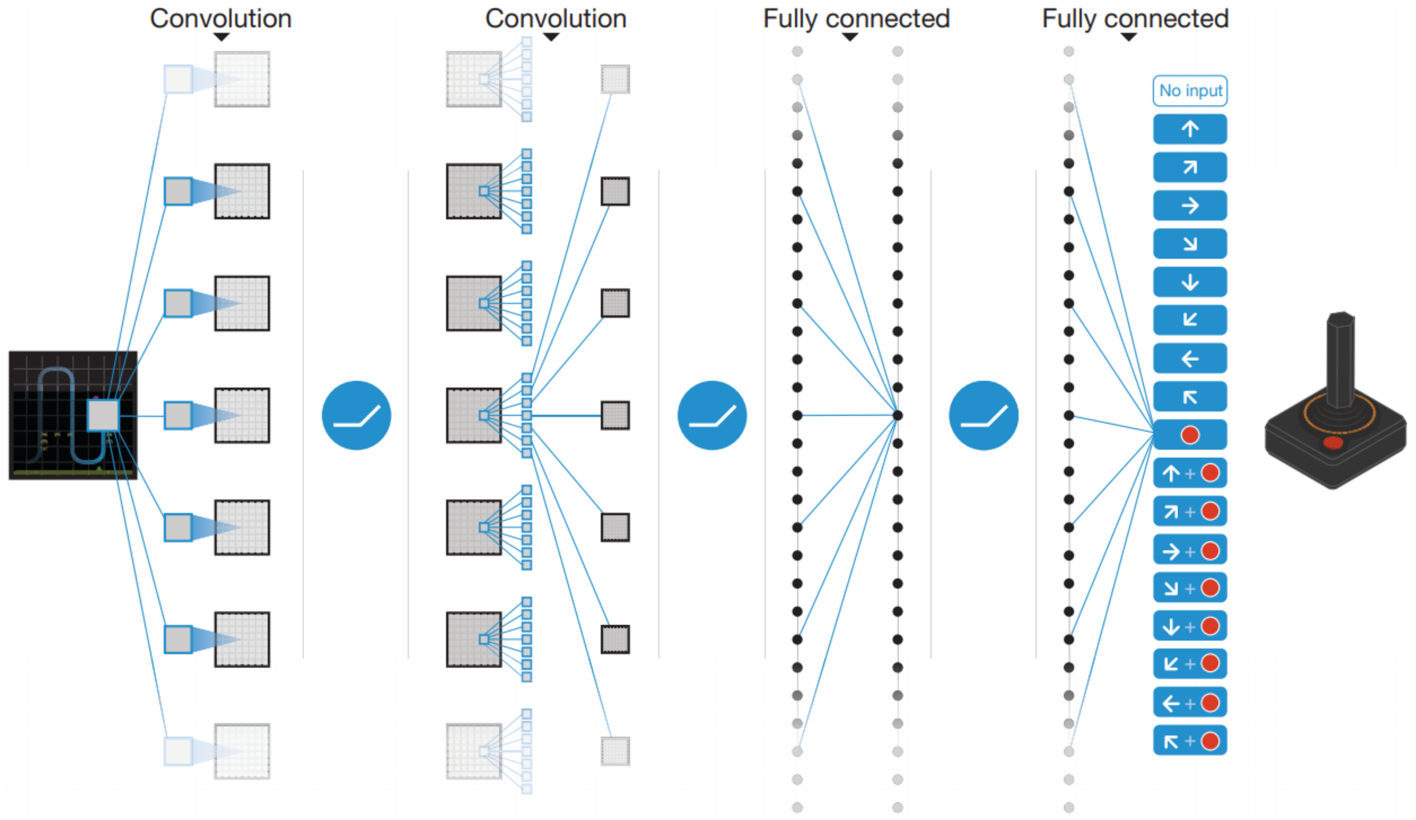
- Generic supervised learning:

$$\mathcal{L} = \sum (y - y_i)^2$$

- Deep reinforcement learning:

$$\mathcal{L} = \sum \Big(\underbrace{r_t + \arg\max_a Q(s_{t+1}, a)}_{\text{new estimate}} - \underbrace{Q(s_t, a_t)}_{\text{old estimate}} \Big)^2$$

Deep Reinforcement Learning



(DeepMind, 2015)

Deep Reinforcement Learning

What about environments with continuous actions?

Calculating $\arg\max_{a} Q(s_t, a)$ becomes an expensive search.

We typically turn to **policy-based** methods (vs. value-based methods):

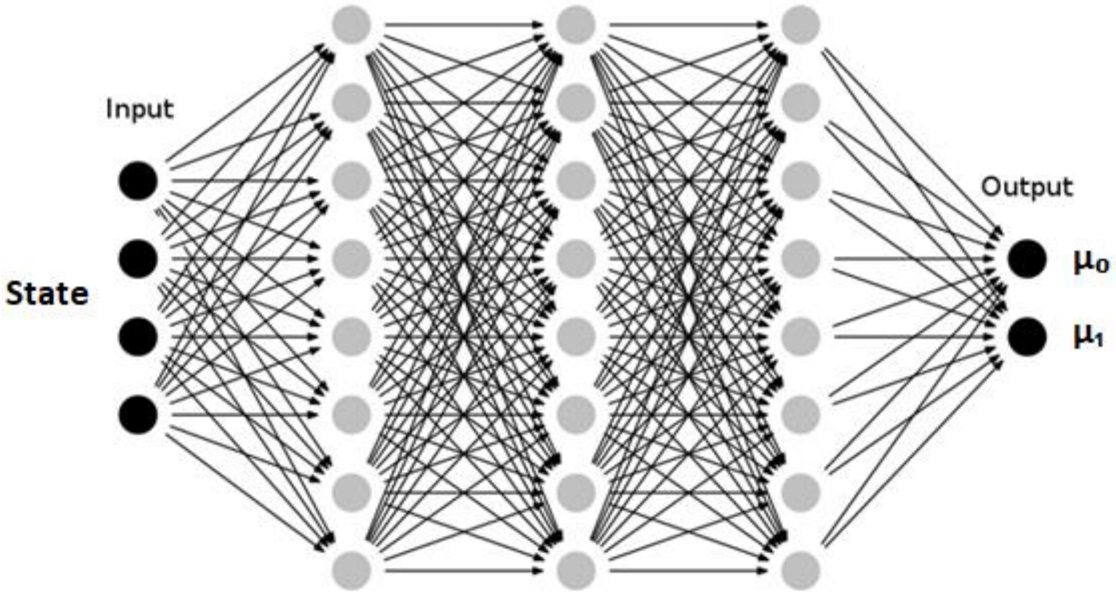
- Implement stochastic policy $\pi_\theta(a|s)$ parameterised by NN weights θ
- Treat it as an optimisation problem: use gradient ascent to find θ which maximises reward

Deep Reinforcement Learning

Policy-based methods (vs. value-based methods):

- Implement stochastic policy $\pi_\theta(a|s)$ parameterised by NN weights θ
- Treat it as an optimisation problem: use gradient ascent to find θ which maximises reward
- We have well developed policy in this

area

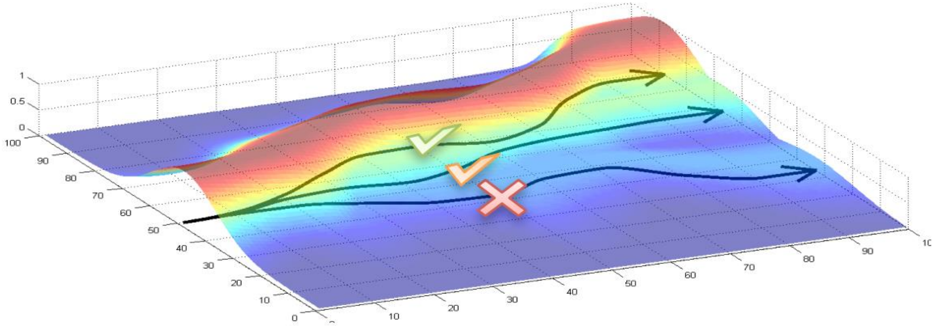


Deep Reinforcement Learning

$$\nabla_\theta \underbrace{J(\theta)}_{\substack{\text{objective function} \\ \text{(e.g. sum of reward)}}} = \mathbb{E}[\underbrace{\nabla_\theta \log \pi_\theta(s,a)}_{\substack{\text{gradient of NN to do} \\ \text{more of } a \text{ in state } s}} \underbrace{Q(s,a)}_{\substack{\text{how good doing} \\ a \text{ in state } s \text{ is}}}]$$

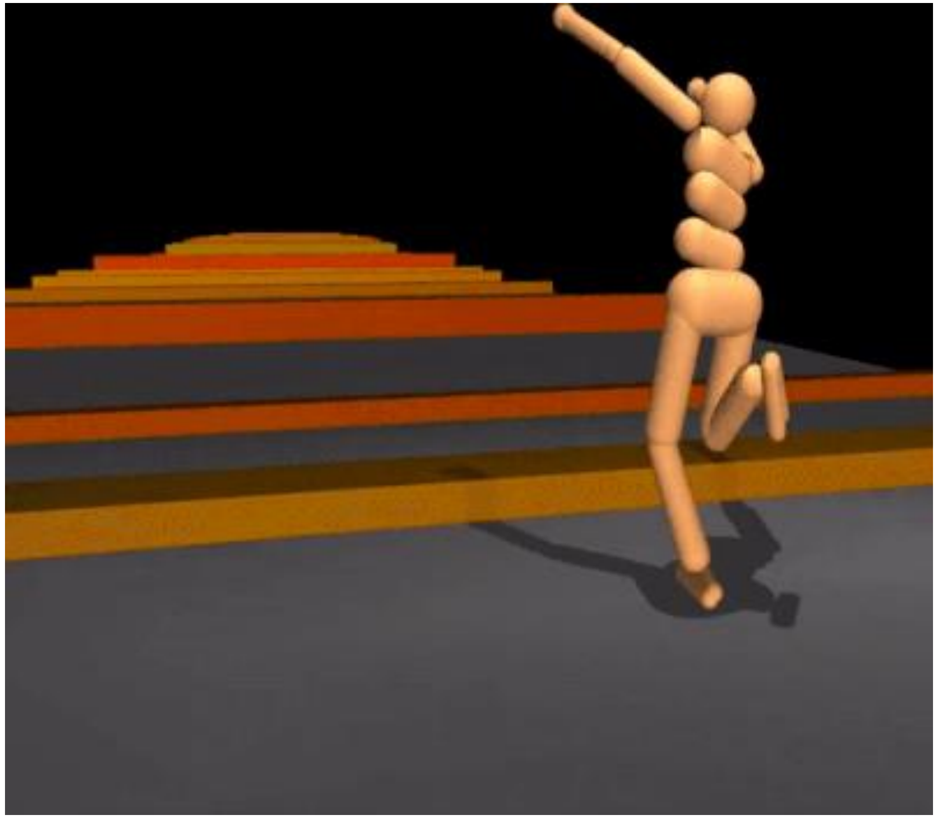
- Directly optimises agent's policy by encouraging good actions more
- Various improvements on this base algorithm, e.g. limiting the size of the policy change (better stability), reducing the variance of the term by subtracting baseline (better sample efficiency)
- Algorithms include A2C, PPO, ACER, TRPO,

DPG, …

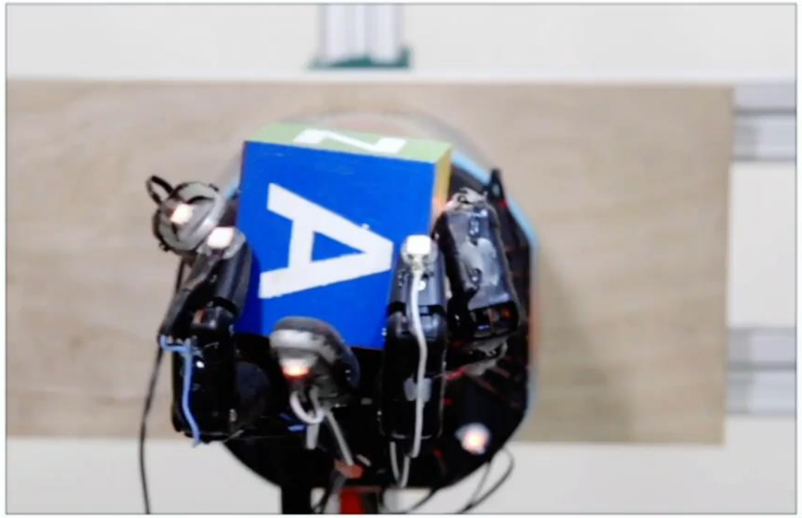


Policy Gradient Results

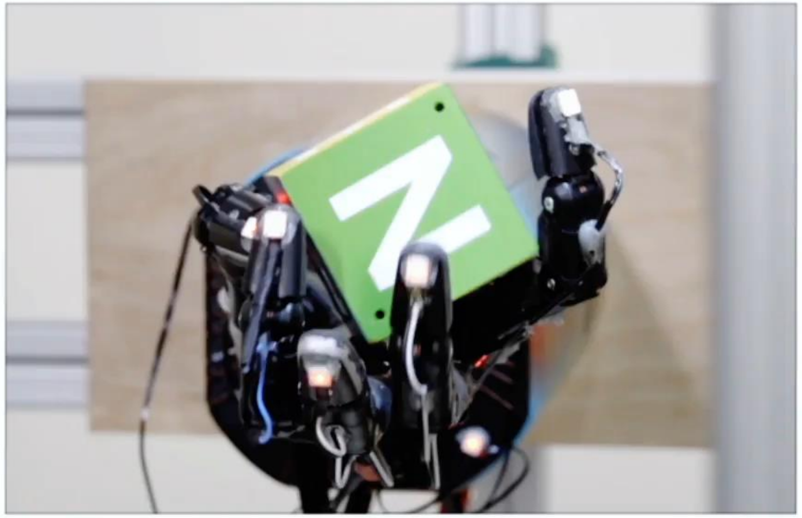
- Performs well in high-dimensional control tasks, e.g. robotics



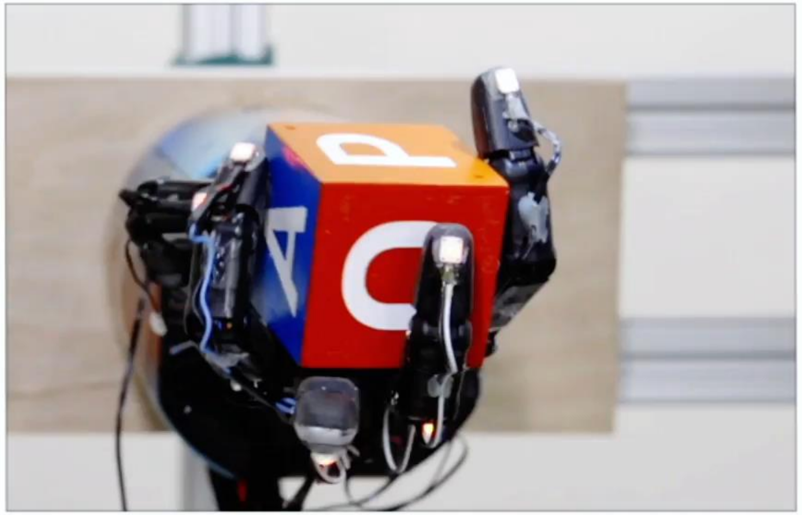
Policy Gradient Results



FINGER PIVOTING



SLIDING

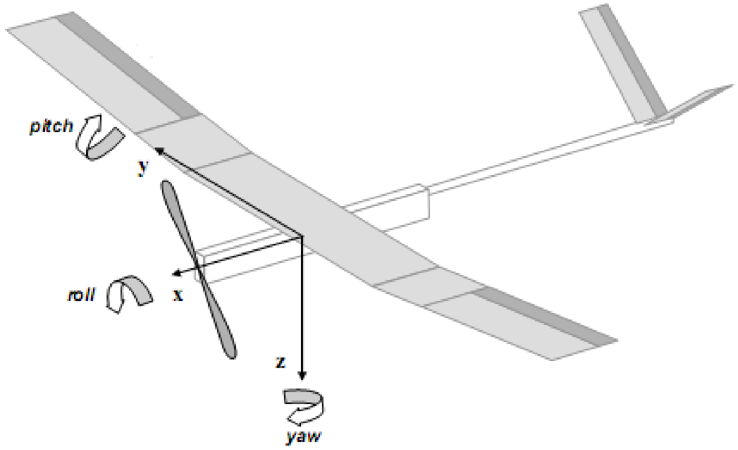


FINGER GAITING

Aircraft Control with DDRL

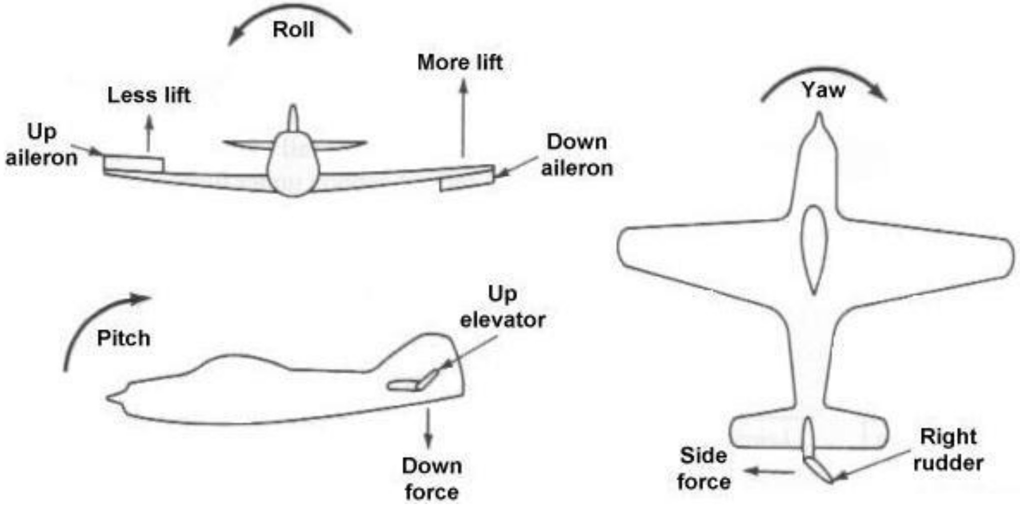
Aircraft Control with DDRL

State



$$s = \begin{cases} h & \text{altitude} \\ \phi, \theta, & \text{roll, pitch} \\ \dot{x}, \dot{y}, \dot{z}, & \text{linear velocity} \\ \dot{\phi}, \dot{\theta}, \dot{\omega}, & \text{angular velocity} \\ \delta_a, \delta_e, \delta_r & \text{control positions} \\ e_h, e_\psi & \text{error to target} \end{cases}$$

Actions



$$a = \begin{cases} c_a & \text{ailerons} \\ c_e & \text{elevator} \\ c_r & \text{rudder} \end{cases}$$

(images from Kimathi et al, 2017)

Aircraft Control with DDRL

We want to control the aircraft's altitude and heading (direction of flight). We encode this in the environment's **reward function**

- **Heading:** up to 0.5 reward each timestep, when $e_h = 0$
- **Altitude**: up to 0.5 reward each timestep, when $e_\psi = 0$

Optimal behaviour: maintain the heading and altitude with zero error.

(images from Kimathi et al, 2017)

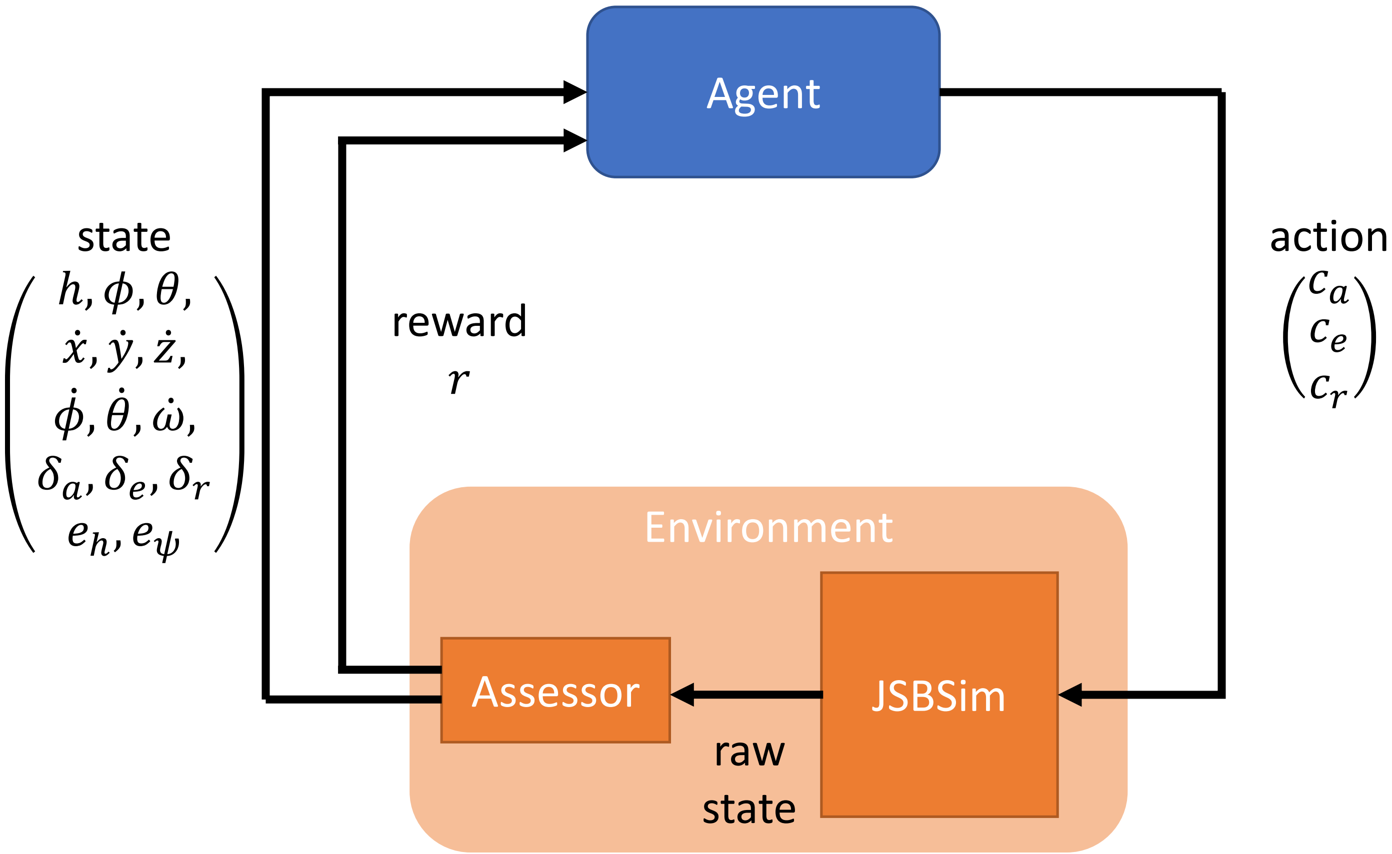
Aircraft Control with DDRL

Environment

JSBSim, an open-source flight dynamics model used in the FlightGear simulator.

http://www.flightgear.org

Aircraft Control with DDRL



Aircraft Control with DDRL

Aircraft Control with DDRL

Needed lots of hyperparameter tuning

RL is difficult and often unstable!

Future work: automate hyperparameter optimisation

Meta-learning is an active research area

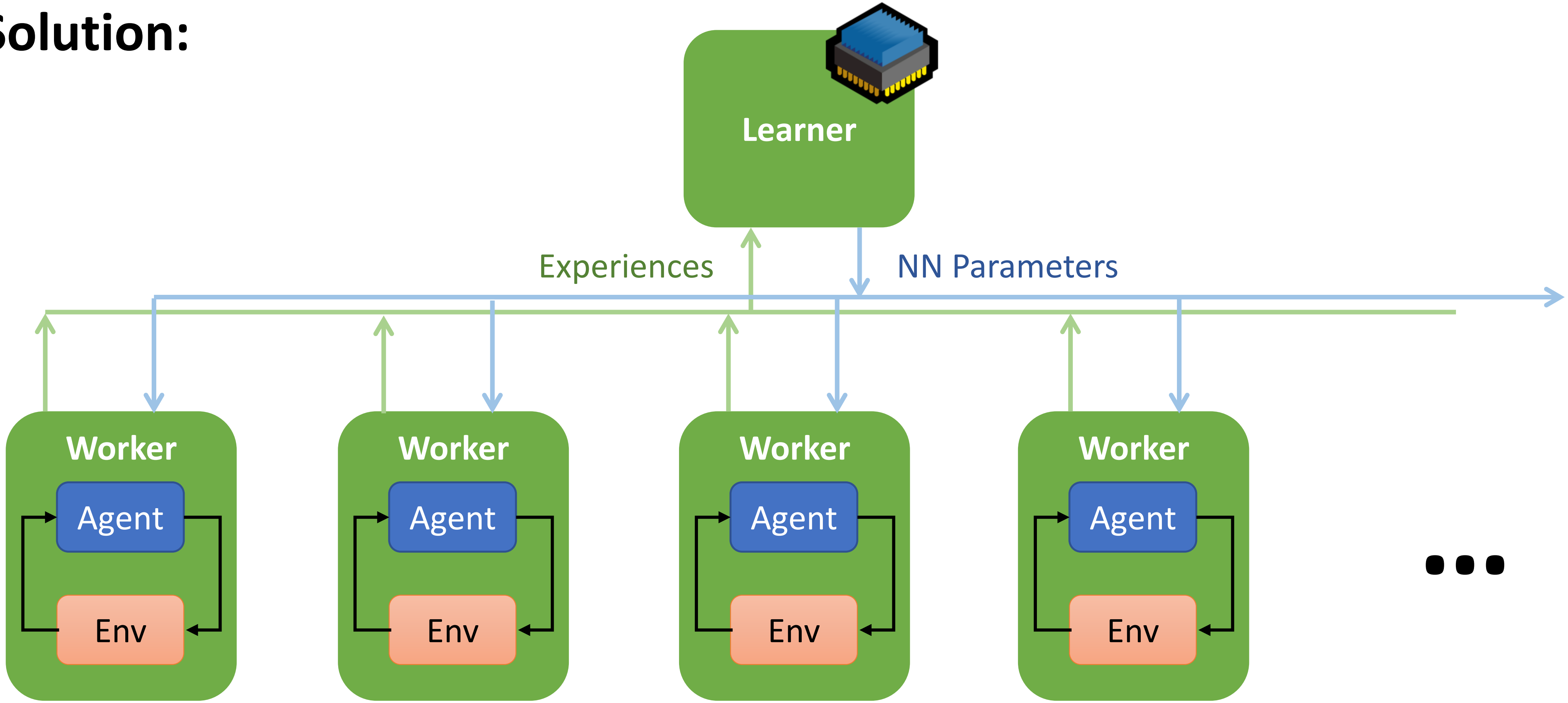
Hyperparameter	Optimised Value
Network hidden layers	2
Network nodes per layer	64
Network activation function	tanh
Network optimiser	Adam
Network learning rate	3×10^{-4}
Network optimisation steps	10
Baseline hidden layers	3
Baseline learning rate	1×10^{-3}
Baseline optimisation steps	3
Batch size [episodes]	1
Sub-sampling fraction	0.30
GAE parameter (λ)	0.99
Clipping parameter (ϵ)	0.10
Entropy coefficient	1×10^{-3}
Discount factor (γ)	1.0
States normalisation	`running_standardise`

(images from Kimathi et al, 2017)

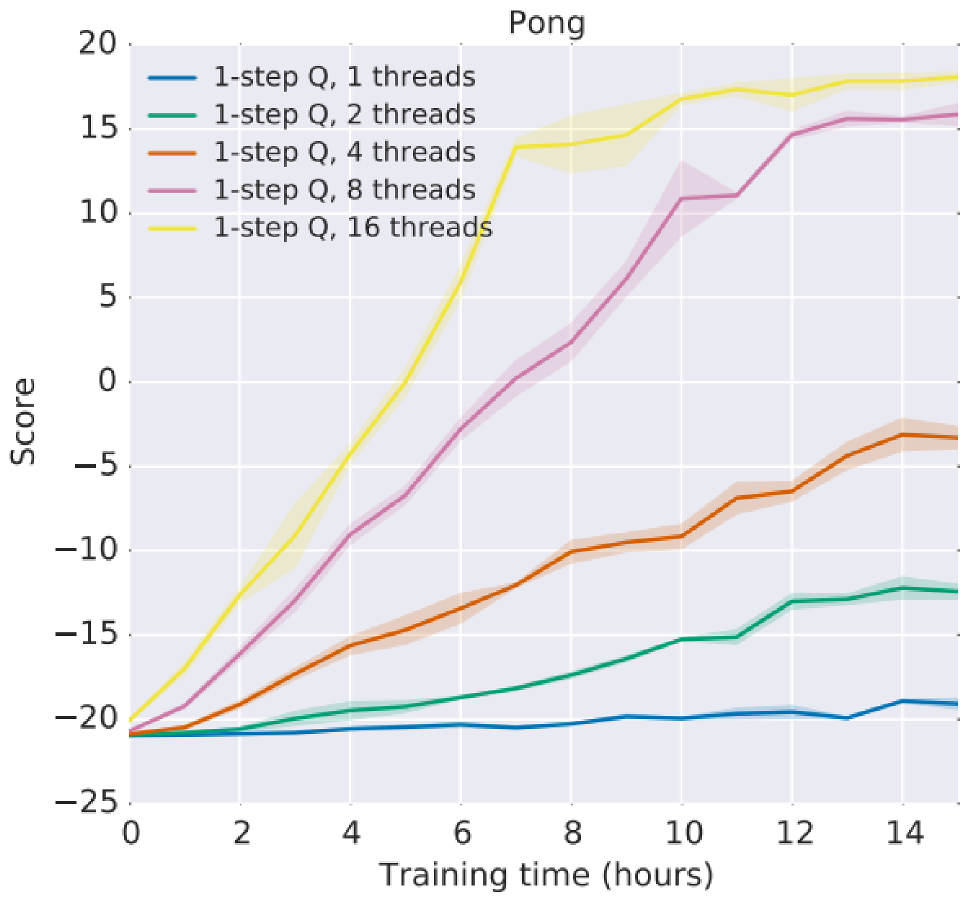
Distributed Deep Reinforcement Learning

Problem: Low sample efficiency → days of wall-clock time to learn

Solution:

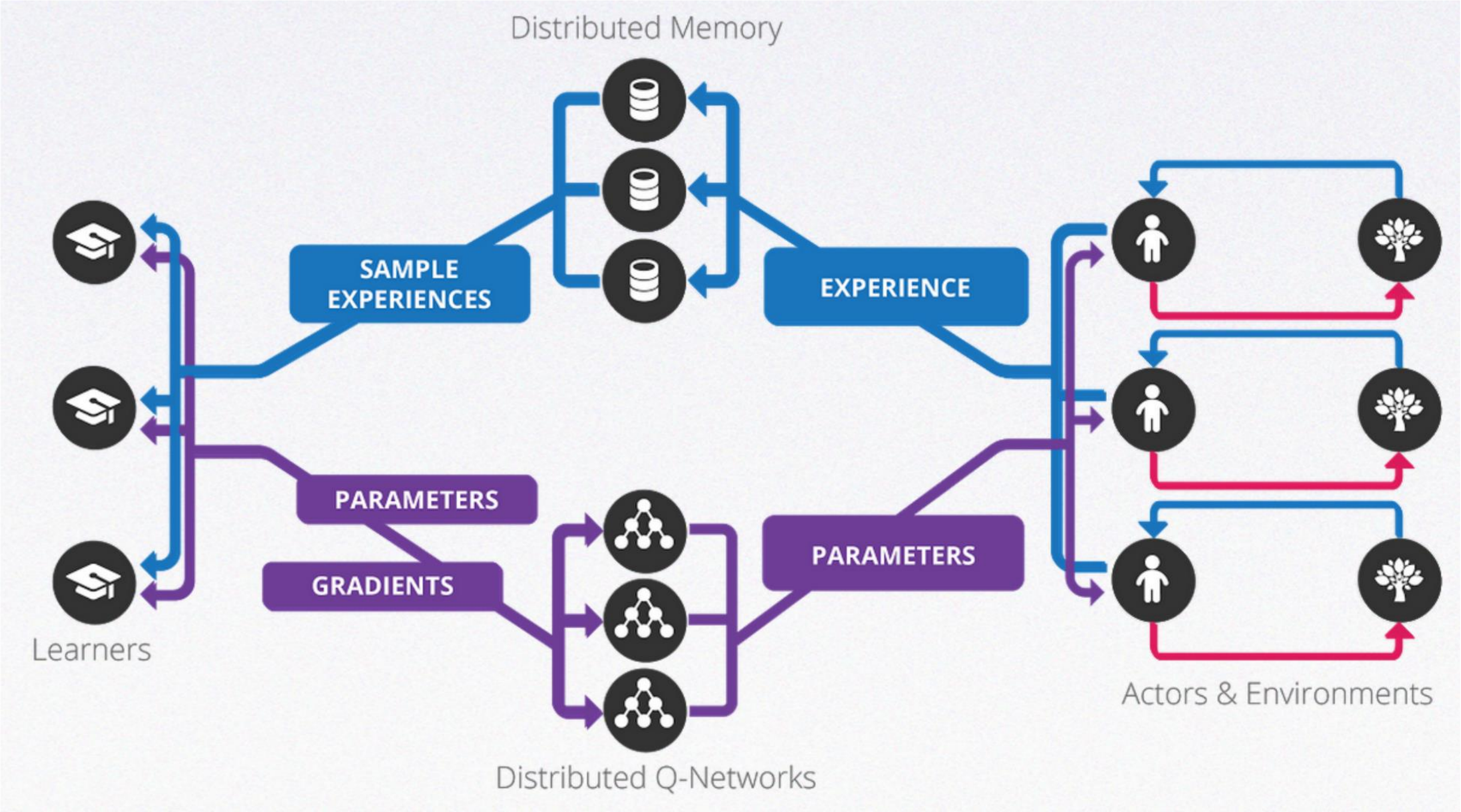


Distributed Deep Reinforcement Learning



(Mnih et al, 2016)

Distributed Deep Reinforcement Learning



(Nair et al, 2015 in Silver, 2016)

OpenAI Five: Deep RL at Scale

Recap

- Reinforcement learning is a **general-purpose framework** for learning how to act in an arbitrary environment
- **Deep neural networks** have improved the ability of agents to generalise in complex challenges
- We've discussed the two main classes of RL algorithms: **value-based** and **policy-based**
- Distributed implementations **scale** well and allow us to tackle difficult problems faster