Reinforcement Learning

Prototyping with Deep Learning

Learning outcomes

After this lesson you will be able to:

- Understand the basics of Reinforcement Learning
- Understand the Q-Learning algorithm and the basics of Deep Q-Learning (DQN)
- Know some of the popular Deep Reinforcement Learning algorithms and their applications

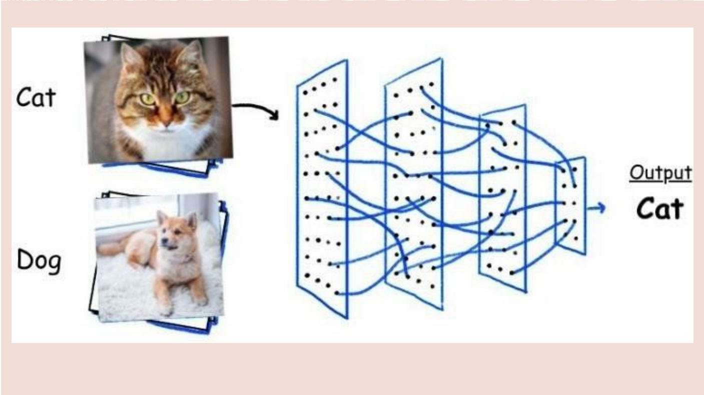
Recap: Classes of learning Problems

Supervised Learning

Data: *(x,y)*

Goal: Learn function to map

$x \longrightarrow y$

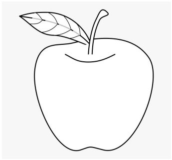


Unsupervised Learning

Data: *x*

x is data, No labels

Goal: Learn Learn underlying structure



This thing is like the other thing

What is Reinforcement Learning?

Problems involving an **agent** interacting with an **environment** which provides (numeric) reward signals.

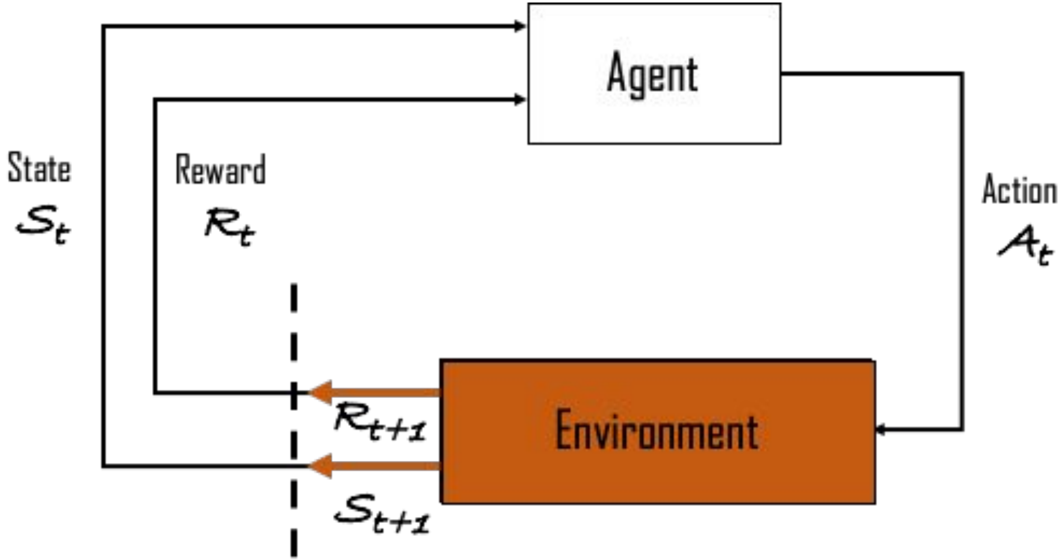
Data: State-action pairs

Goal: Learn how to take actions in order to maximize reward

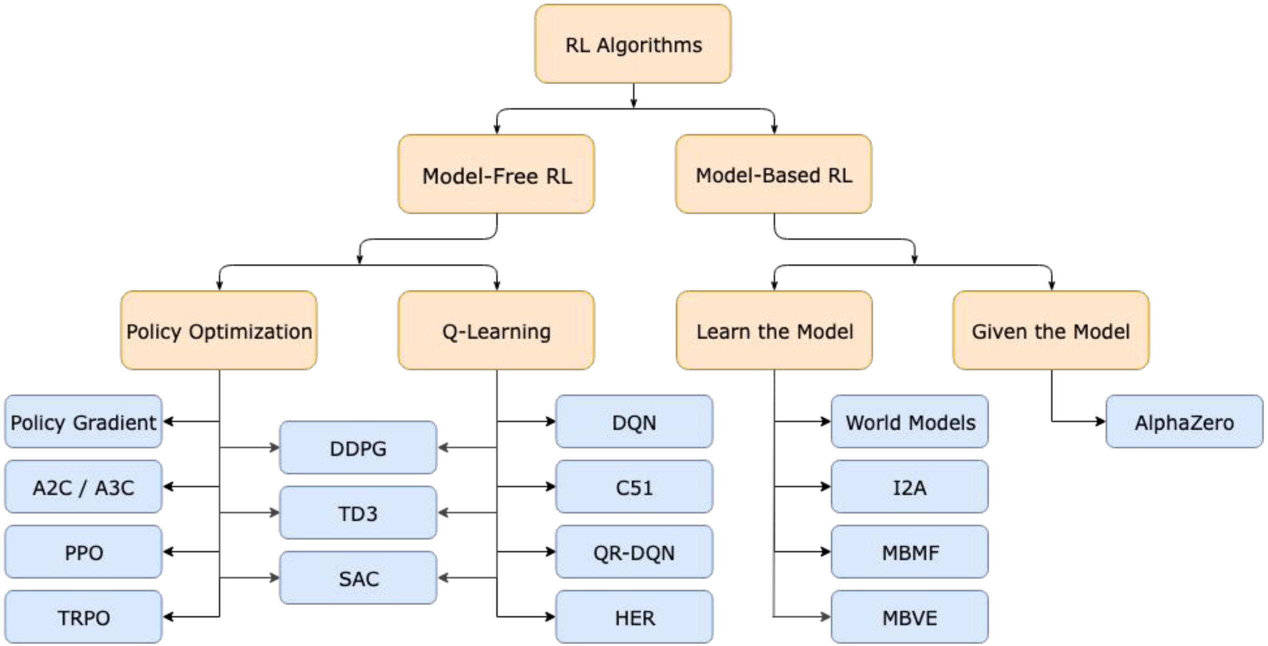
Eat this thing because it will keep you alive.

Classical RL

Agents interact with their environment through a sequence of **observations**, **actions** and **rewards**.

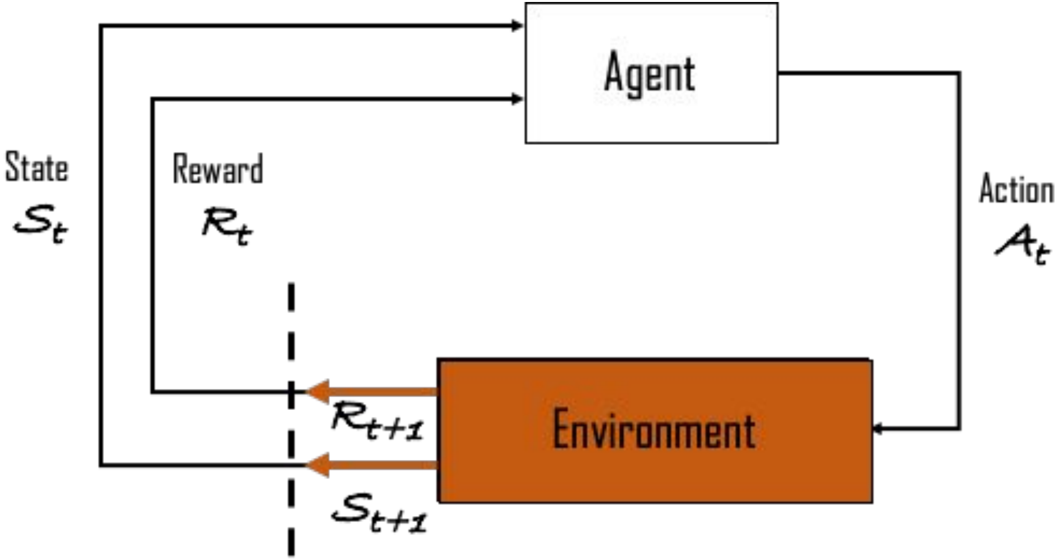


RL taxonomy



https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below

Classical RL



Reward: feedback that measures the success of failure of the agent's action

Total Reward (Return)

$$\sum_{i=t}^{\infty} r_i = r_t + r_{t+1} \dots + r_{t+n} + \cdots$$

Classical RL

Total Reward (Return) → $$\sum_{i=t}^{\infty} r_i = r_t + r_{t+1} \dots + r_{t+n} + \cdots$$

Discounted Total Reward (Return) $$R_t = \sum_{i=t}^{\infty} \gamma^i r_i = \gamma^t r_t + \gamma^{t+1} r_{t+1} \dots + \gamma^{t+n} r_{t+n} + \cdots$$

$\boldsymbol{\gamma}$**:** discount factor ; $0 < \gamma < 1$, designed to make future rewards worth less

Policy

What is the probability that an agent will select a specific action in a given state?

- If an agent follows policy Π at time t, then $\Pi(a|s)$ is the probability that $A_t=a$ if $S_t=s$.
- This means that, at time t, under policy Π, the probability of taking action a in state s is $\Pi(a|s)$.

Note that, for each state $s \in S$, Π is a probability distribution over $a \in A(s)$

How do we measure the quality of actions?

Q-function

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

Q-function captures the **expected total future reward** an agent in *state* ***s*** $_t$ *can receive by executing a certain action* ***a*** $_t$

How useful is a given action in gaining some future reward.

- *Good actions result high Q value*
- *Bad actions result low Q value*

Q-function

- The optimal Q-value function Q* is the maximum expected cumulative reward achievable from a given (state, action) pair

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geqslant 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

- The maximum sum of rewards r_t discounted by γ at each time step t, achievable by a policy ***Π*** = p(a|s).
- The agent operates based on a policy ***Π*** to approximate Q-values (state-action pairs) that maximize a future reward.

This is done by enforcing the **Bellman equation:**

Reinforcement Learning

Bellman equation:

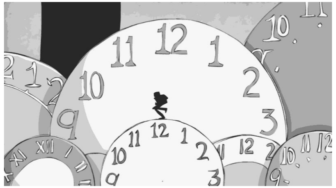
$$Q^*(s,a) = \mathbb{E}_{s' \sim \varepsilon}\left[r + \gamma \max_{a'} \underline{Q^*(s',a')} \mid s,a \right]$$

Given any state-action pair (s, a) the maximum cumulative reward achieved is the sum of the reward for that pair **r** plus the value of the next state we end up with, **s′**.

The value at state **s′** is going to be the maximum over actions a′ at $Q^*(s',a')$.

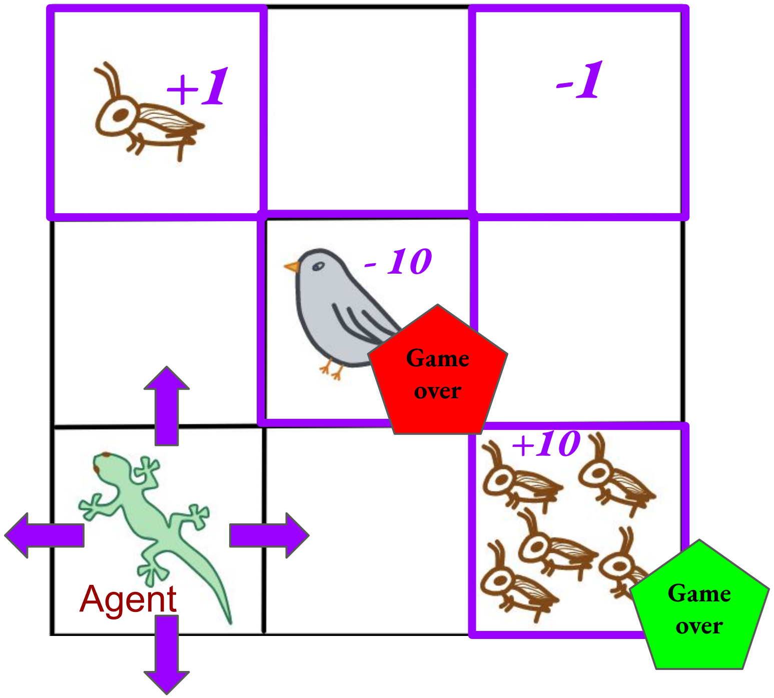
- The objective of **RL** is to find an optimal **policy** in a sense that the expected return over all successive time steps is the **maximum** achievable.
- This can be done by learning the **optimal Q-values** for each state-action pairs.

Q-Learning



Q-Learning

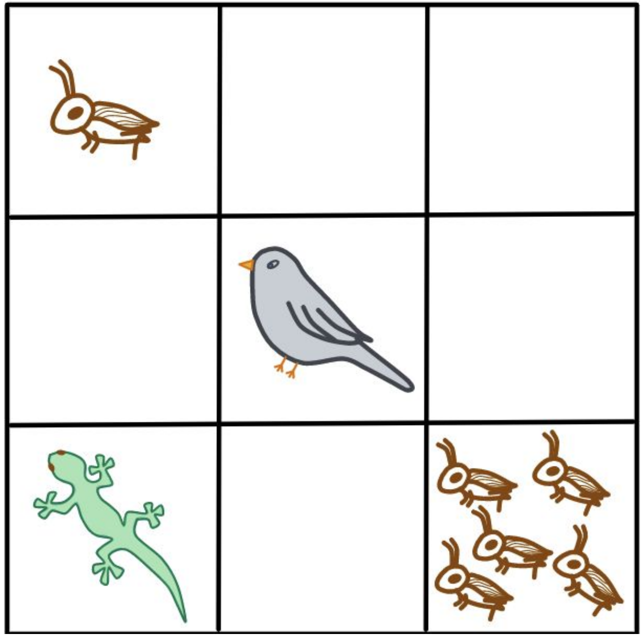
The lizard wants to eat as many crickets as possible in the least amount of time without stumbling across a bird, which will, itself, eat the lizard.



- States determined by individual tiles and where the agent is at a given time

Q-Learning

At the beginning the lizard has no idea of how good any action is at any given state



Actions

States	Left	Right	Up	Down
1 cricket	0	0	0	0
Empty 1	0	0	0	0
Empty 2	0	0	0	0
Empty 3	0	0	0	0
Bird	0	0	0	0
Empty 4	0	0	0	0
Empty 5	0	0	0	0
Empty 6	0	~~0~~ +10	0	0
5 crickets	0	0	0	0

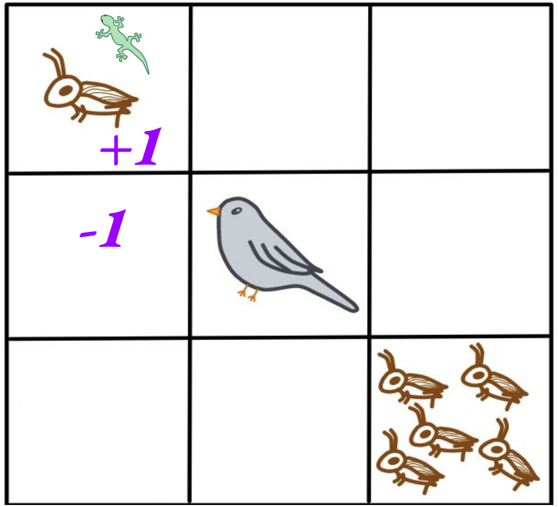
Q-table

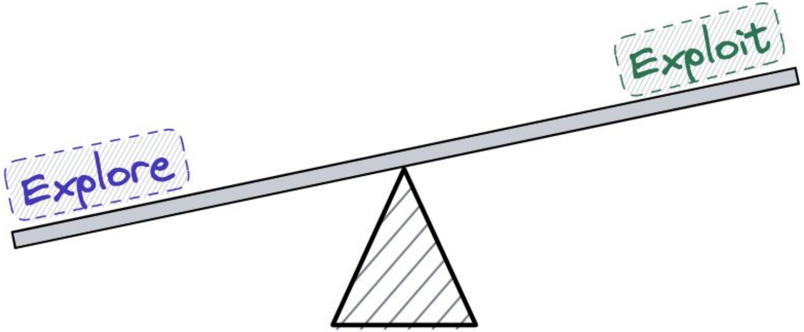
Q-Learning (Exploration Vs Exploitation)

Exploration: the act of exploring the environment in order to find out information about it.

Exploitation: making use of the information that is already known about the environment in order to maximize the return. **Q-table look up**.

```
if random_num > epsilon:
# choose action via exploitation
else:
# choose action via exploration
```





Greedy Epsilon strategy

Q-Learning (Updating Q-table)

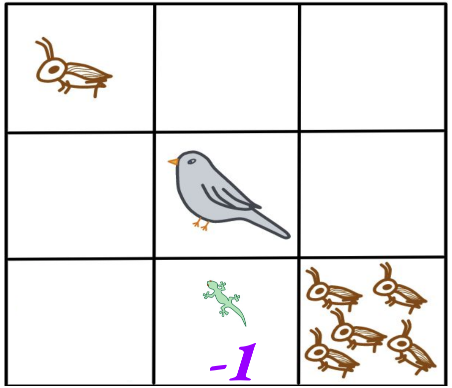
UNIVERSITÉ DU LUXEMBOURG

- To update the Q-value We use the bellman equation
- We want to make the *Q(s,a) for any state action pair as close as possible to the right hand side of the Bellman equation.*
- The formula to calculate new Q-values:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Q-Learning (Updating Q-table)

UNIVERSITÉ DU LUXEMBOURG



discount rate $\gamma = 0.99$

learning rate $\alpha = 0.7$

Actions

States	Left	Right	Up	Down
1 cricket	0	0	0	0
Empty 1	0	0	0	0
Empty 2	0	0	0	0
Empty 3	0	0	0	0
Bird	0	0	0	0
Empty 4	0	0	0	0
Empty 5	0	~~0~~ -0.7	0	0
Empty 6	0	0	0	0
5 crickets	0	0	0	0

$$Q^{new}(s,a) = (1-\alpha)\underbrace{Q(s,a)}_{\text{old value}} + \alpha\overbrace{\left(R_{t+1} + \gamma \max_{a'} Q(s',a')\right)}^{\text{new value}}$$

$$= (1-0.7)(0) + 0.7\left(-1 + 0.99\left(\boxed{\max_{a'} Q(s',a')}\right)\right)$$

$$= (1-0.7)(0) + 0.7(-1 + 0.99(0))$$
$$= 0 + 0.7(-1)$$
$$= -0.7$$

= max(Q(empty6, left), Q(empty6, right), Q(empty6, up), Q(empty6, down))

= max(0,0,0,0)

= 0

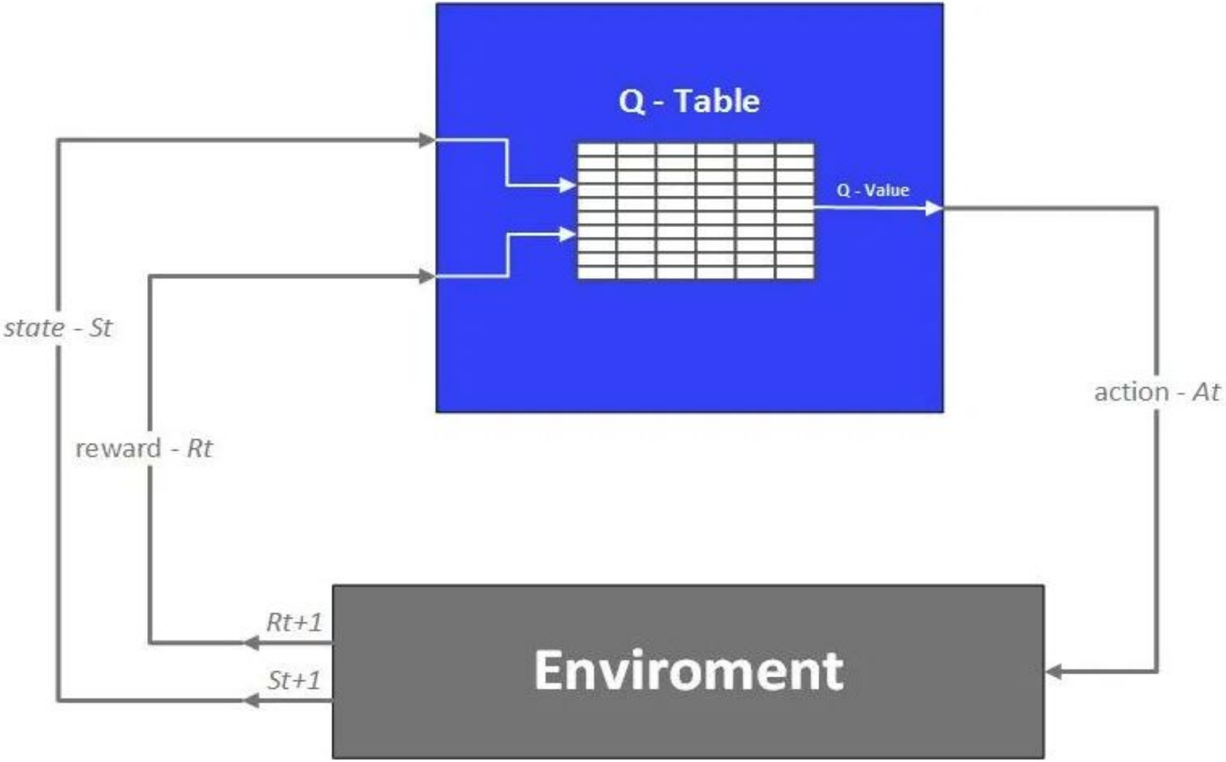
Reinforcement Learning

Learning: Refine Q table by approximating the optimal Q-values

Q-Table		Actions				
		Action 1	Action 2	...	Action n-1	Action n
States	State 1	0.789112	0.745642	...	0.212485	0.256545
	State 2	5.123455	5.11565	...	5.156545	4.155612
	...	...	...	...	...	...
	State n-1	2.156454	2.15567	...	2.144423	2.454658
	State n	6.156212	6.154556	...	6.145441	6.444444

- Initially the agent will do a bit of *exploration*. It selects random values. Over time when it reaches rewards it will slowly learn to exploit those rewards for future action.

Reinforcement Learning



Reinforcement Learning

Atari

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

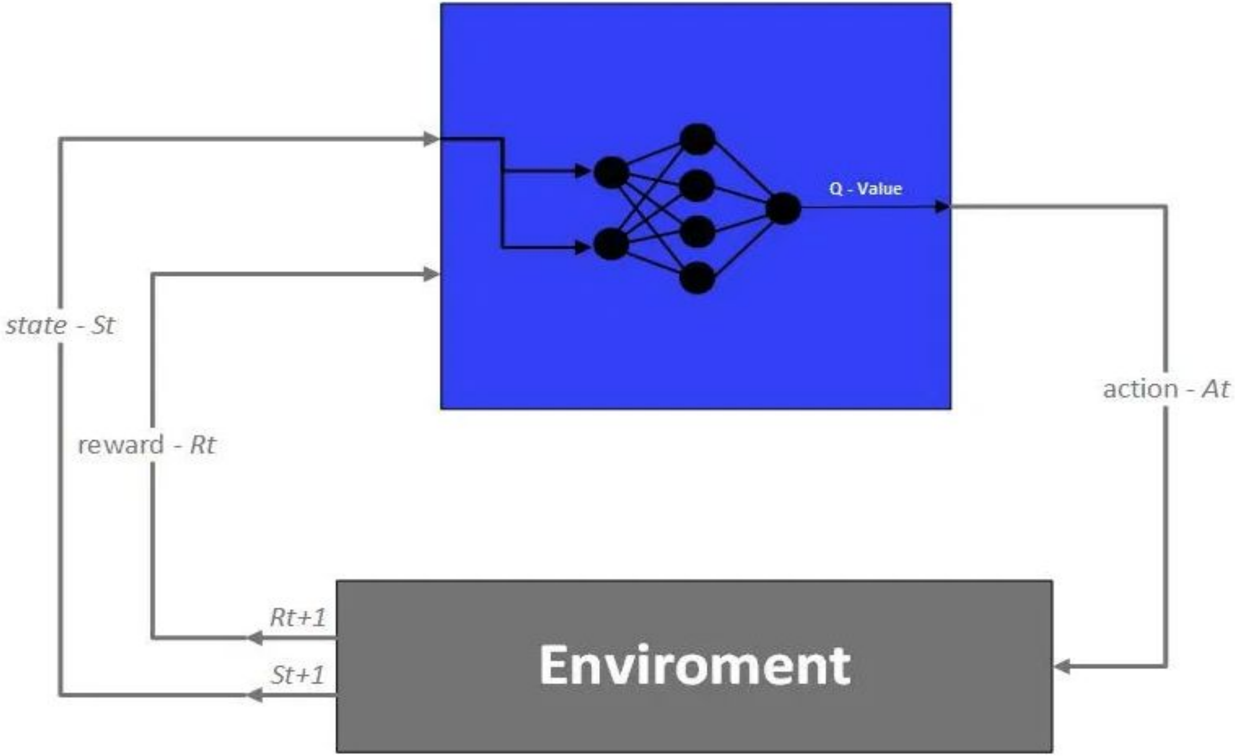
What is the problem with this?

Not scalable: we must compute Q(s, a) for every state-action pair. Computationally expensive to compute for the entire state space, perhaps infeasible.

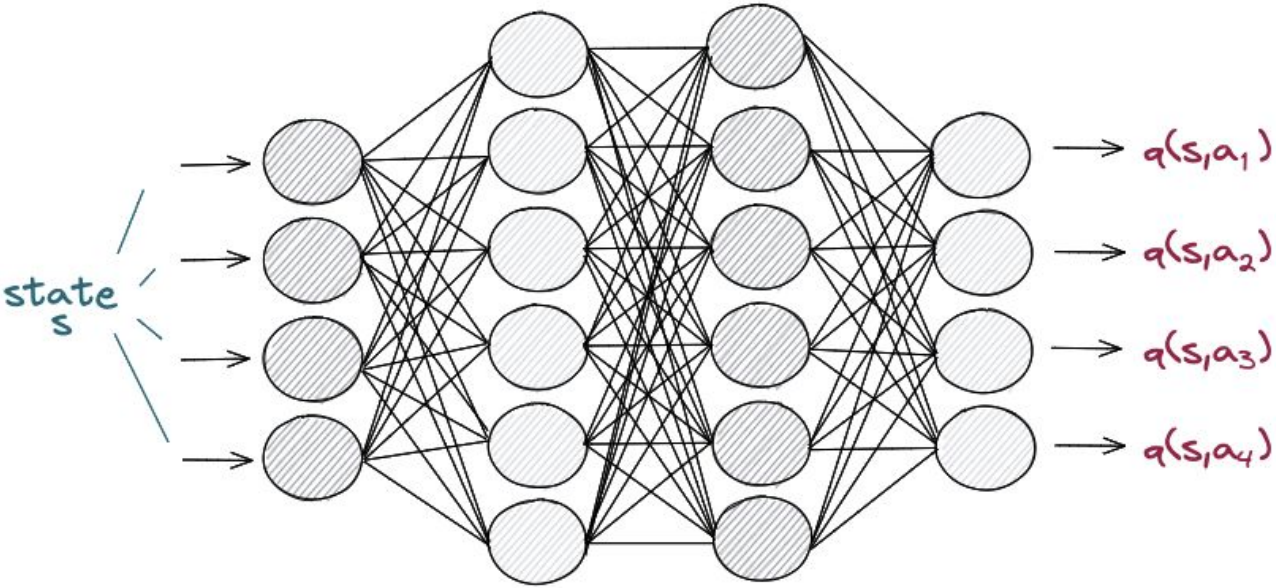
Solution: Use function approximator to estimate the value of Q(s, a),

- Neural Network → **Deep Q-learning**

Deep Reinforcement Learning



Deep Reinforcement Learning



Deep Reinforcement Learning

Forward pass: loss function tries to Minimise the error of the bellman equation.

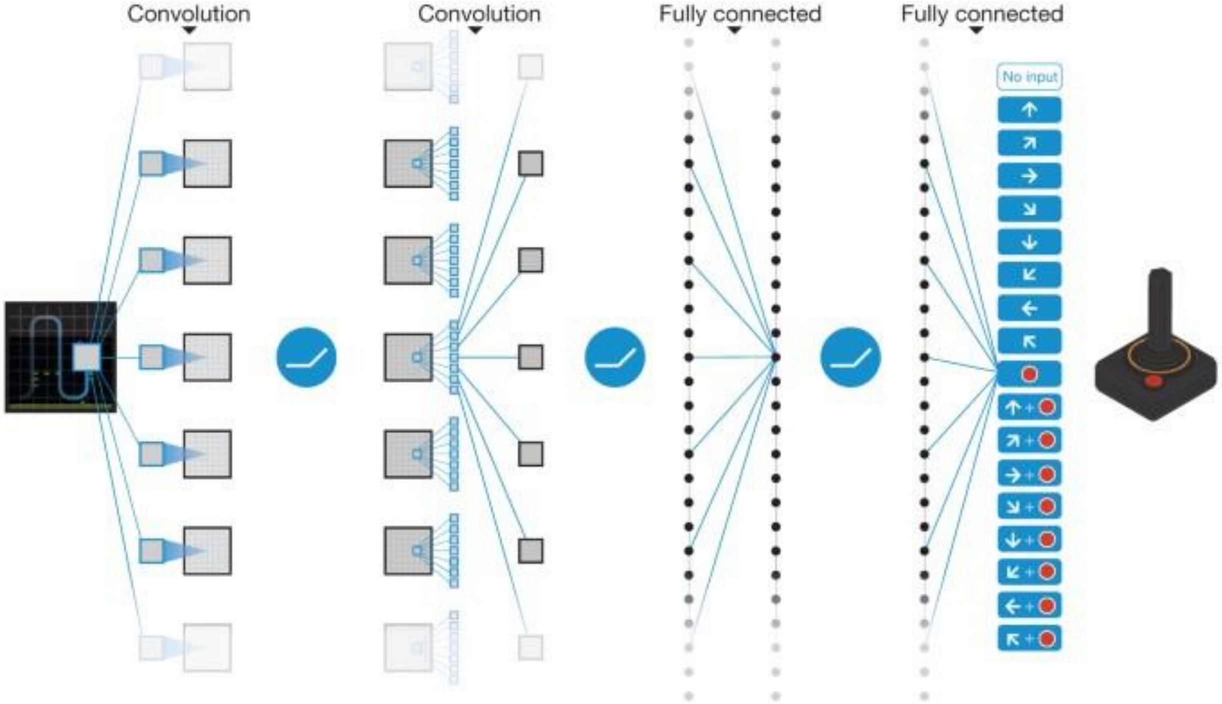
Backward pass: gradient update with respect to the Q-function parameters θ.

$$Q(s,a;\theta) \approx Q^*(s,a)$$

The optimal policy $\Pi^*(s) = argmax\, Q(s,a)$

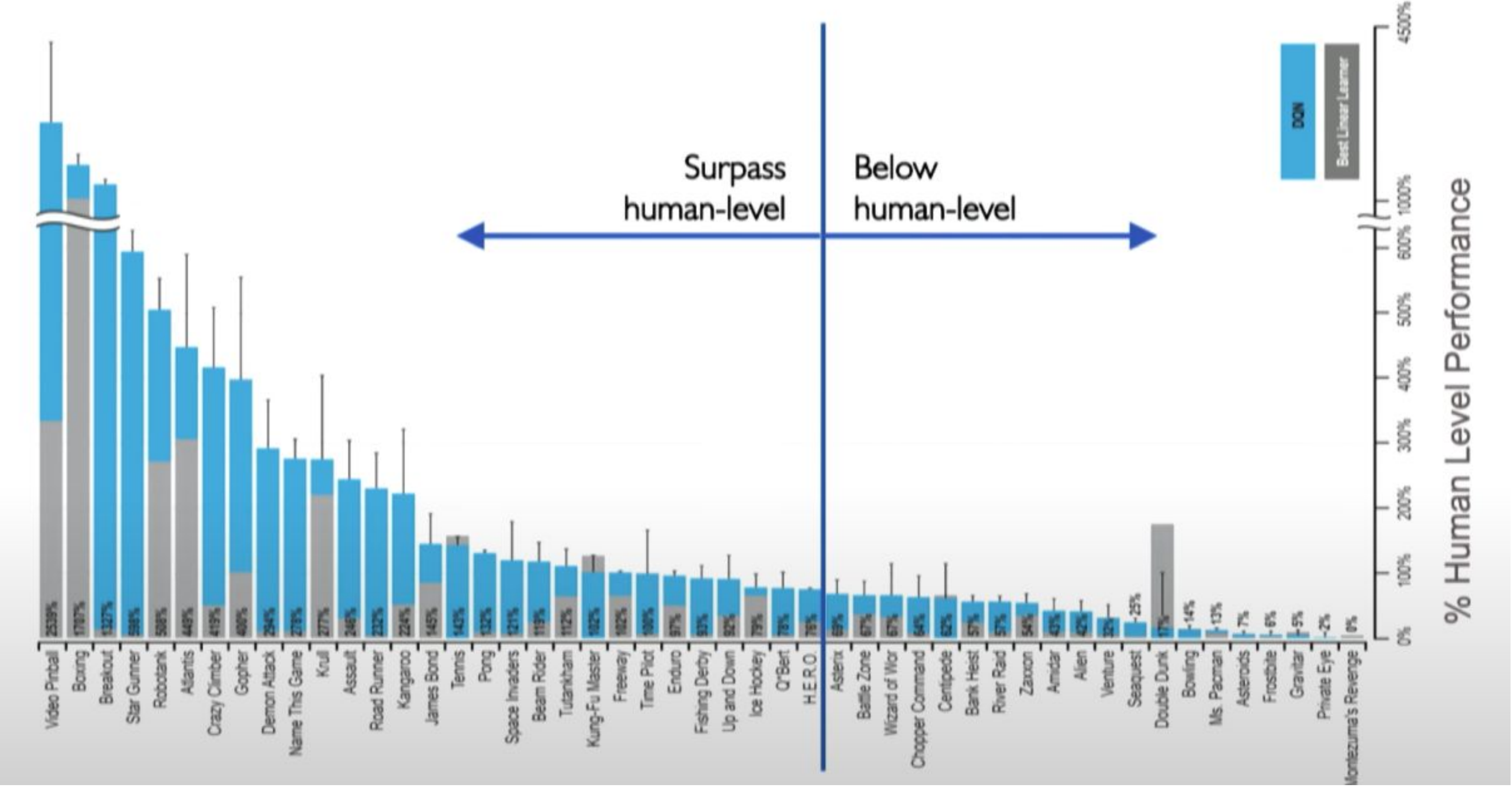
Deep Reinforcement Learning

- DeepMind [Human-level control through deep reinforcement learning]()



Deep Reinforcement Learning

Deep Reinforcement Learning



Deep Reinforcement Learning

UNIVERSITÉ DU LUXEMBOURG

Limitations of Q-learning

- Can not handle **continuous action spaces**: limited to scenarios where we can define the action space in **discrete** and **small** pieces

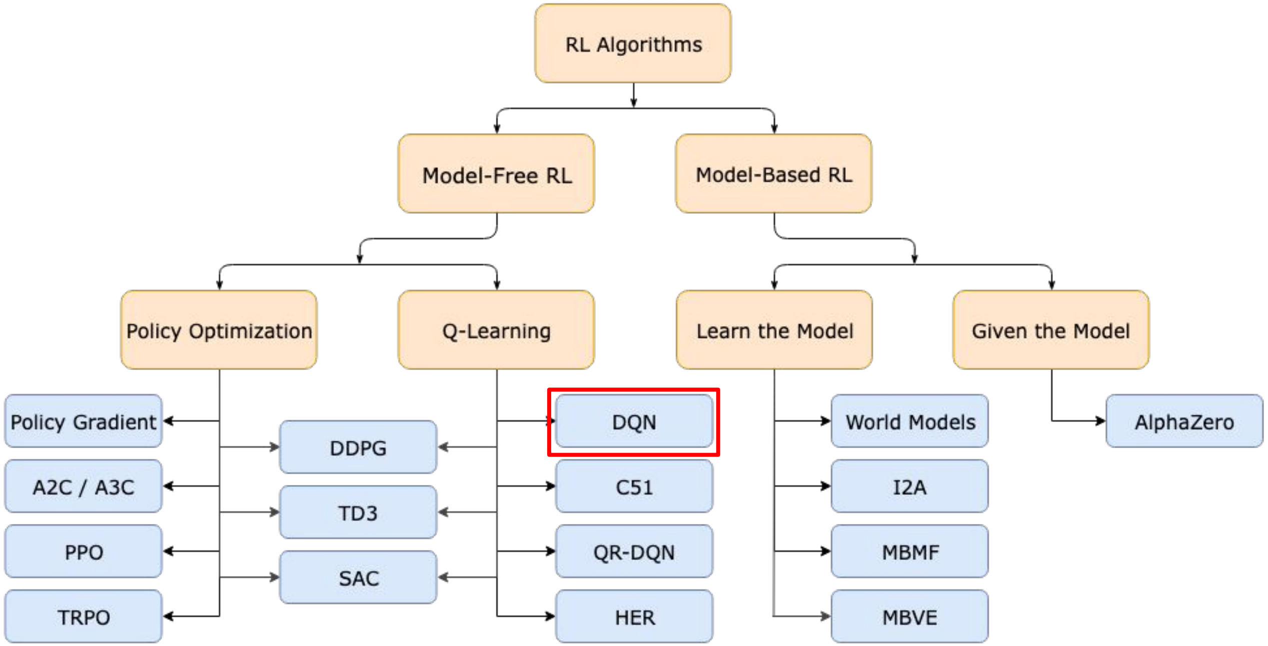
UNIVERSITÉ DU LUXEMBOURG

Limitations of Q-learning

- Policy is deterministically computed from the Q function by maximising the reward : can not learn stochastic policies.

$$\Pi^*(s) = argmax\, Q(s,a)$$

Deep Reinforcement Learning



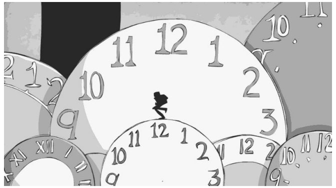
Deep Reinforcement Learning

Value Learning

- Find $Q(s, a)$
- $\boldsymbol{a} = \operatorname{argmax} Q(s,a)$

Policy Learning

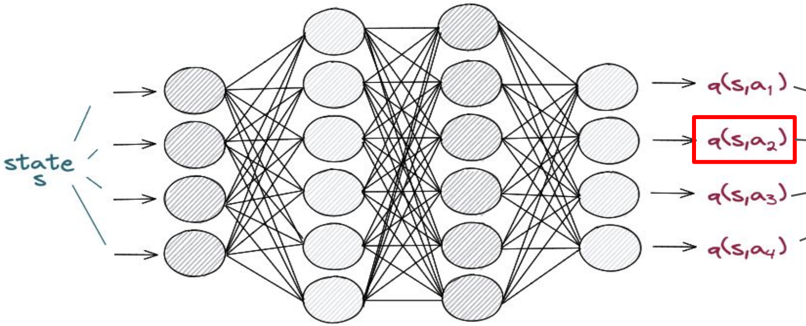
- Find $\boldsymbol{\pi}(s)$
- Sample $\boldsymbol{a} \sim \boldsymbol{\pi}(s)$



Learning a policy can be much simpler

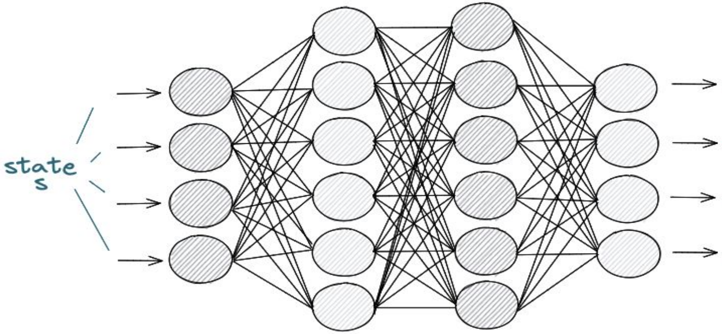
Deep Reinforcement Learning

DQN: Approximate Q-function then use it to infer the optimal policy, $\boldsymbol{\pi}(s)$.



$$\Pi(s) = argmax\, Q(s,a)$$
$$= \boldsymbol{a}_2$$

Policy gradient: directly optimise the policy, $\boldsymbol{\pi}(s)$.



$p(\boldsymbol{a}_{1,}|s) = 0.1$

$p(\boldsymbol{a}_{2,}|s) = 0.8$

$p(\boldsymbol{a}_{3,}|s) = 0.1$

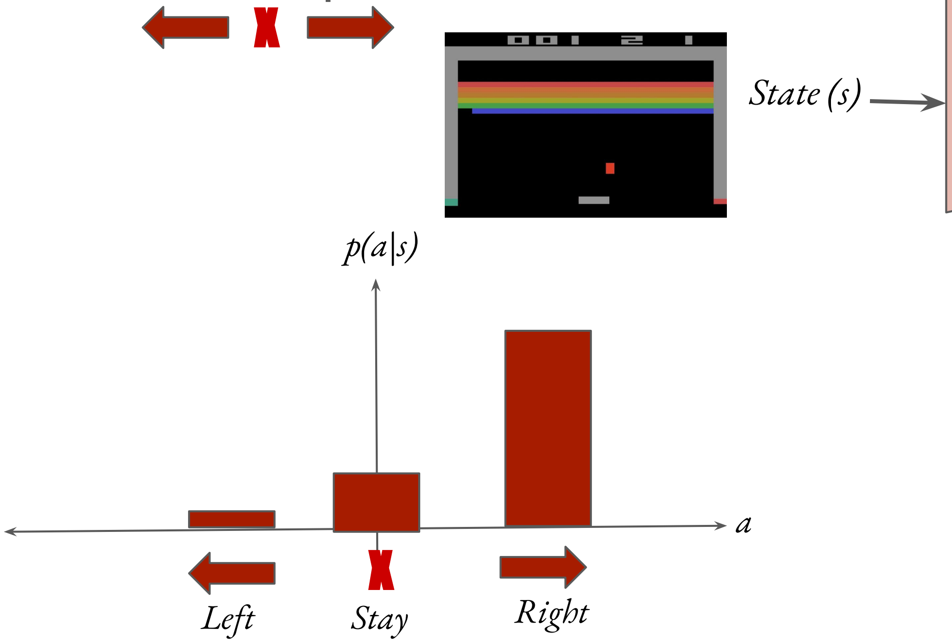
$p(\boldsymbol{a}_{4,}|s) = 0$

$$\Pi(s) \sim p(a|s)$$
$$= \boldsymbol{a}_2$$

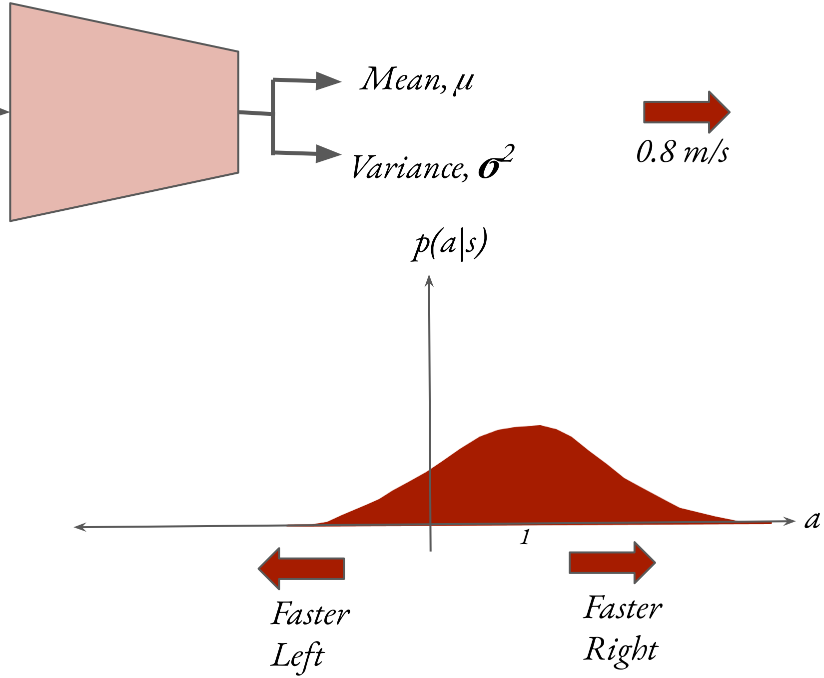
Deep Reinforcement Learning

Policy gradient: Handle **continuous action** spaces.

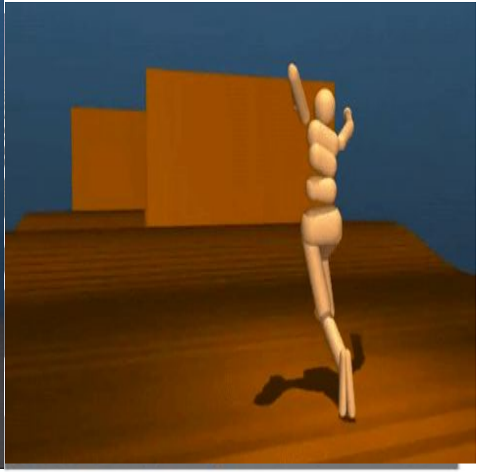
Discrete action space



Continuous action space



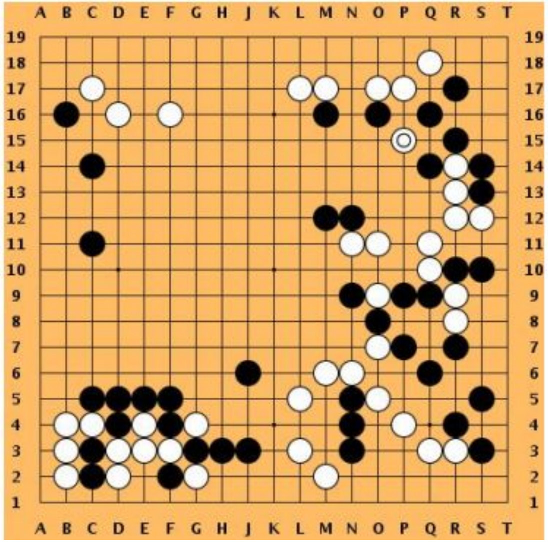
Deep Reinforcement Learning



Objective: Make the robot move forward

State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

Deep Reinforcement Learning



[AlphaGo versus Lee Sedol]

Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

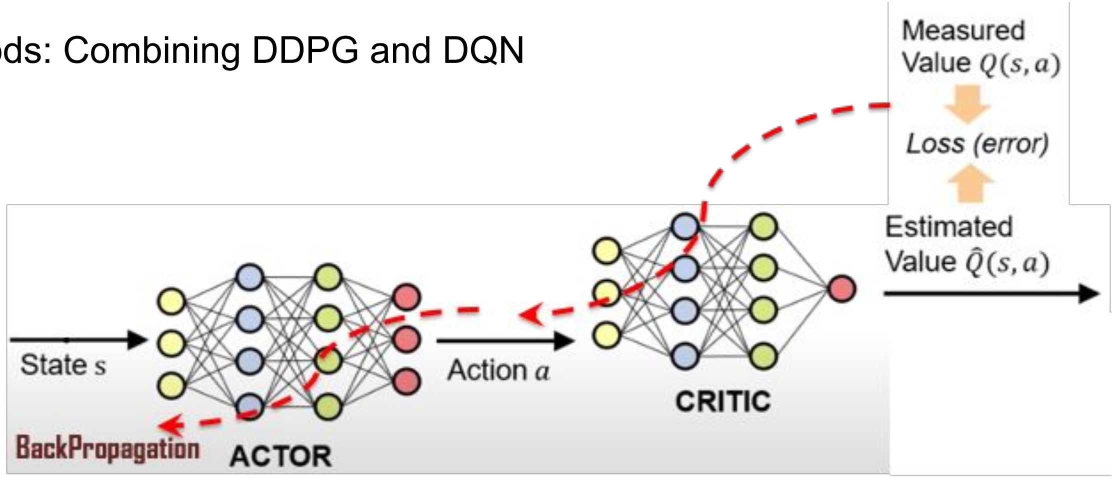
Deep Reinforcement Learning

Continuous control with deep reinforcement learning

Actor critic method works well when we have both an **infinite input space** and infinite output space

Deep Reinforcement Learning

Actor-Critic Methods: Combining DDPG and DQN

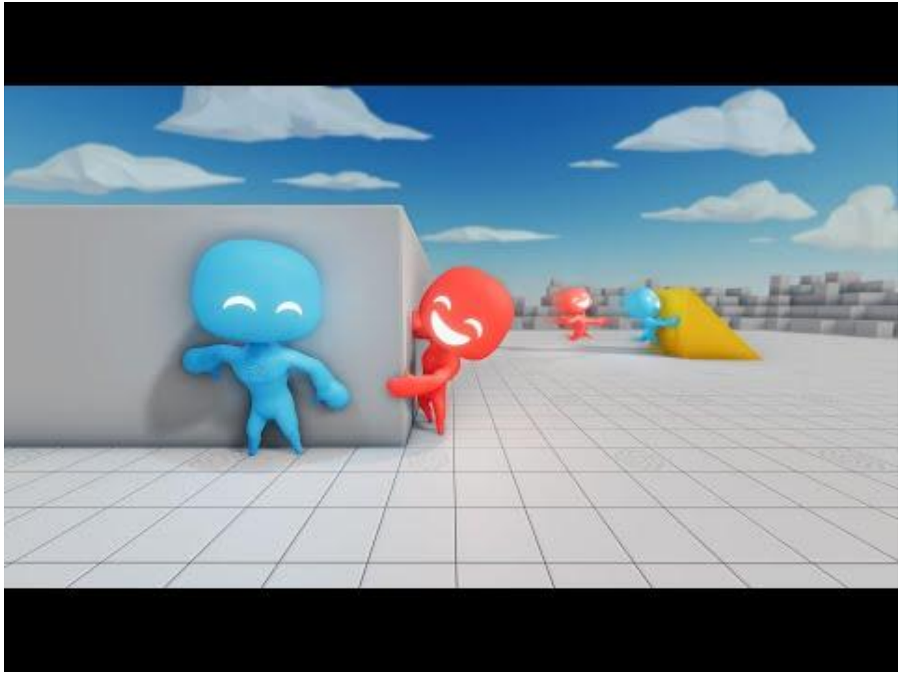


Actor approximates policy p(a|s)

Critic approximates Q-values

- Quite natural in the human's world **Child as an actor** and **parent as a Critic** .

Deep Reinforcement Learning



- [Multi-Agent Actor-Critic for Mixed Cooperative Competitive Environments]
- [Emergent Tool Use From Multi-Agent Autocurricula]

UNIVERSITÉ DU LUXEMBOURG

Hands-on

- **OpenAI gym:** a toolkit for developing and comparing reinforcement learning algorithms. https://gym.openai.com/
- **Stable Baselines3 (SB3):** a set of reliable implementations of reinforcement learning algorithms in PyTorch. https://stable-baselines3.readthedocs.io/en/master/