Deep Reinforcement Learning 1

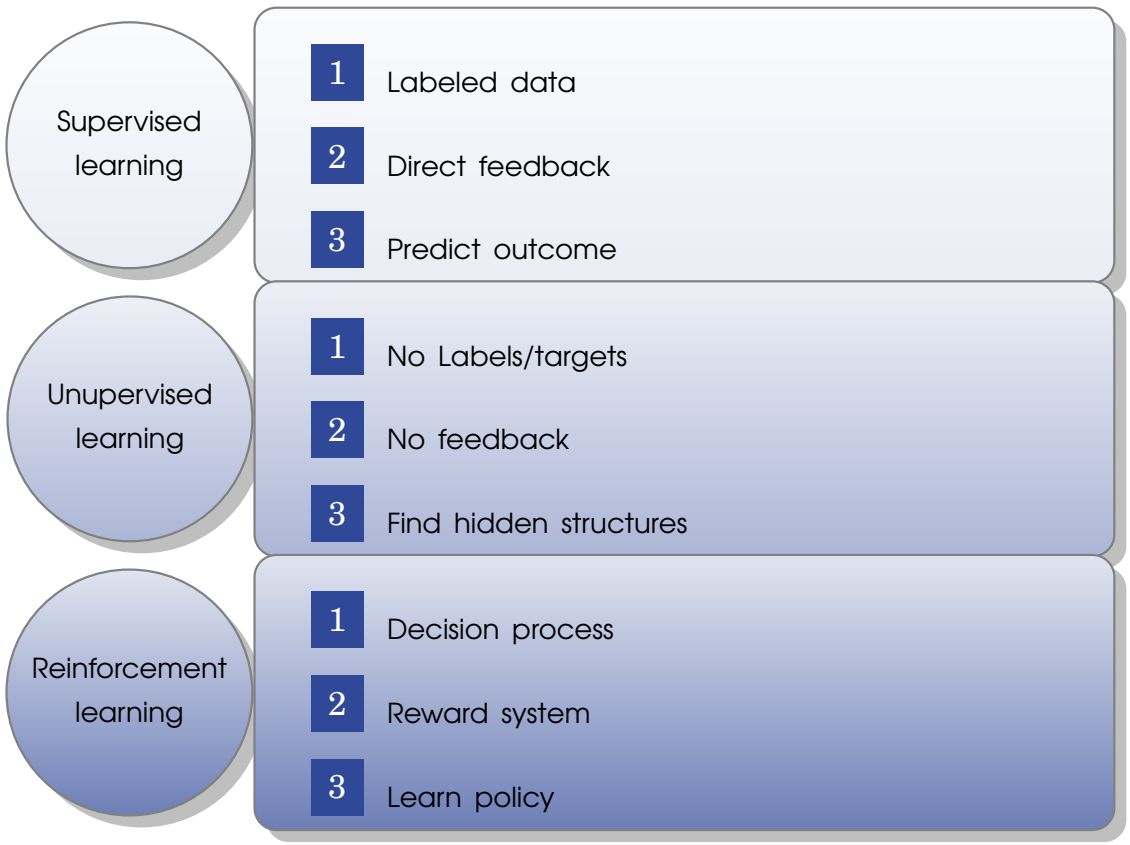
Vincent Barra
LIMOS, UMR 6158 CNRS, Université Clermont Auvergne

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UNIVERSITÉ Clermont Auvergne

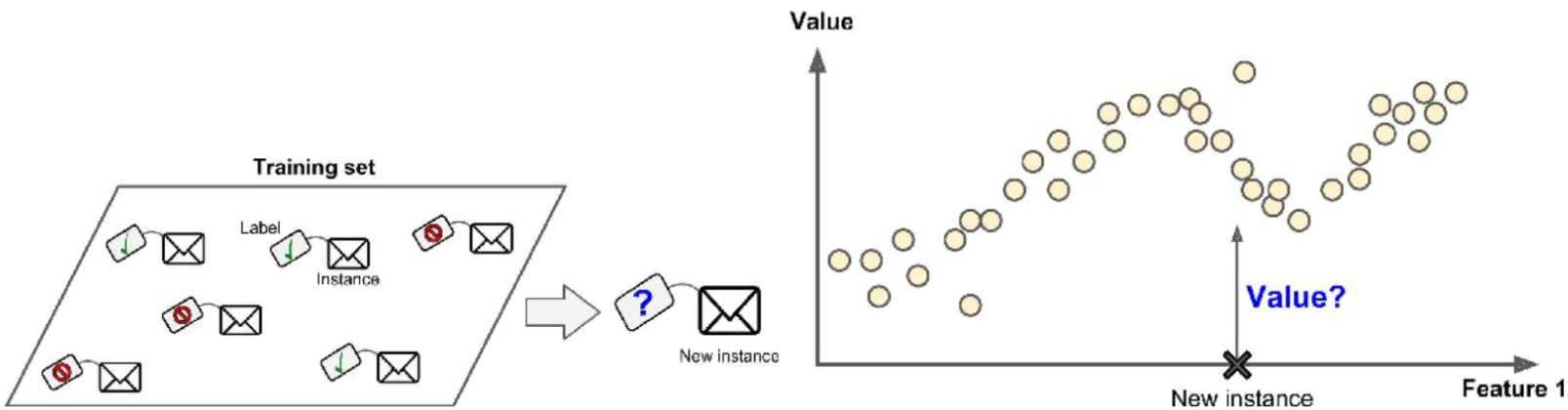
A POSSIBLE TAXONOMY OF MACHINE LEARNING ALGORITHMS



SUPERVISED LEARNING

Supervised learning

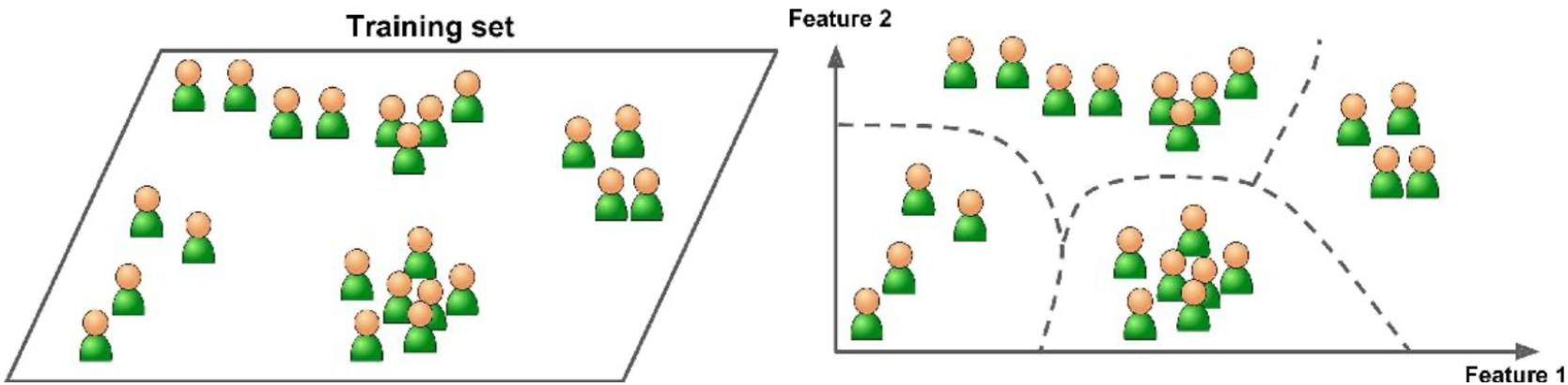
1. Labeled data
2. Direct feedback
3. Predict outcome



UNSUPERVISED LEARNING

Unupervised learning

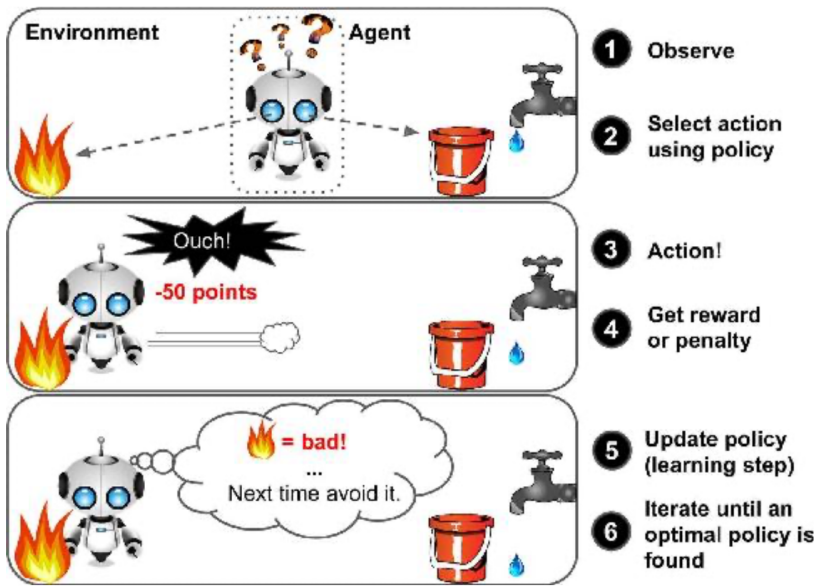
1. No Labels/targets
2. No feedback
3. Find hidden structures



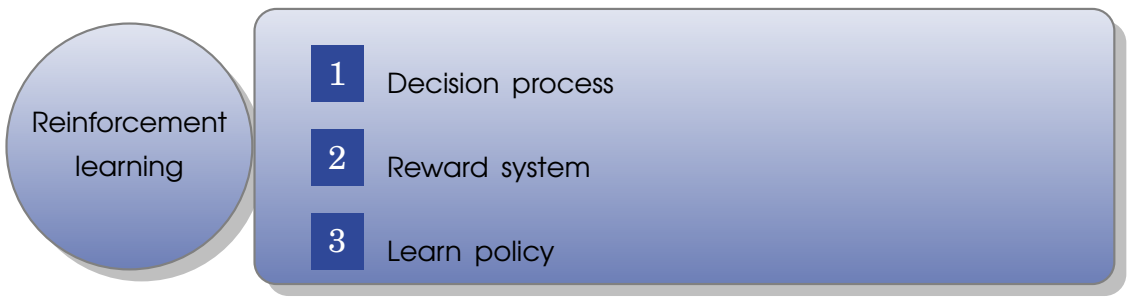
REINFORCEMENT LEARNING

Reinforcement learning

1. Decision process
2. Reward system
3. Learn policy



REINFORCEMENT LEARNING



Topic of the two next lectures

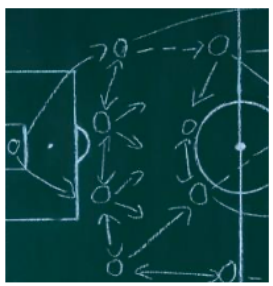
Agent: takes actions

LIMOS
LABORATOIRE D'INFORMATIQUE,
DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UCA
UNIVERSITÉ
Clermont
Auvergne

Environment: the "world"



Environment

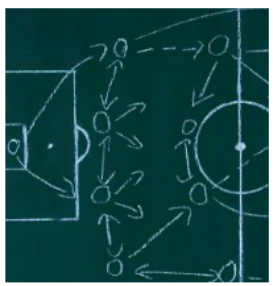
Agent

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UCA UNIVERSITÉ Clermont Auvergne

Action space



Environment

Agent

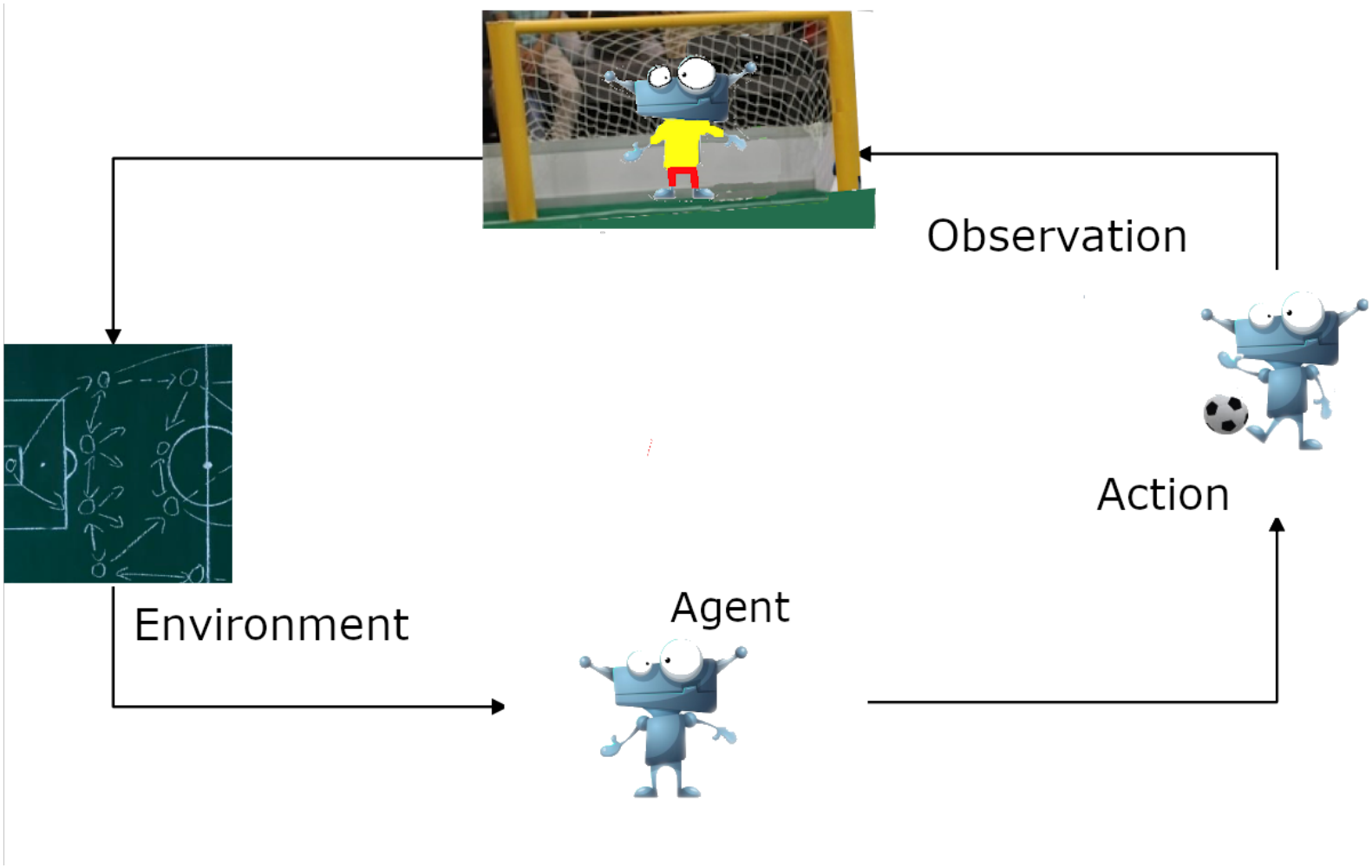
Action

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UCA UNIVERSITÉ Clermont Auvergne

Changes in the environment

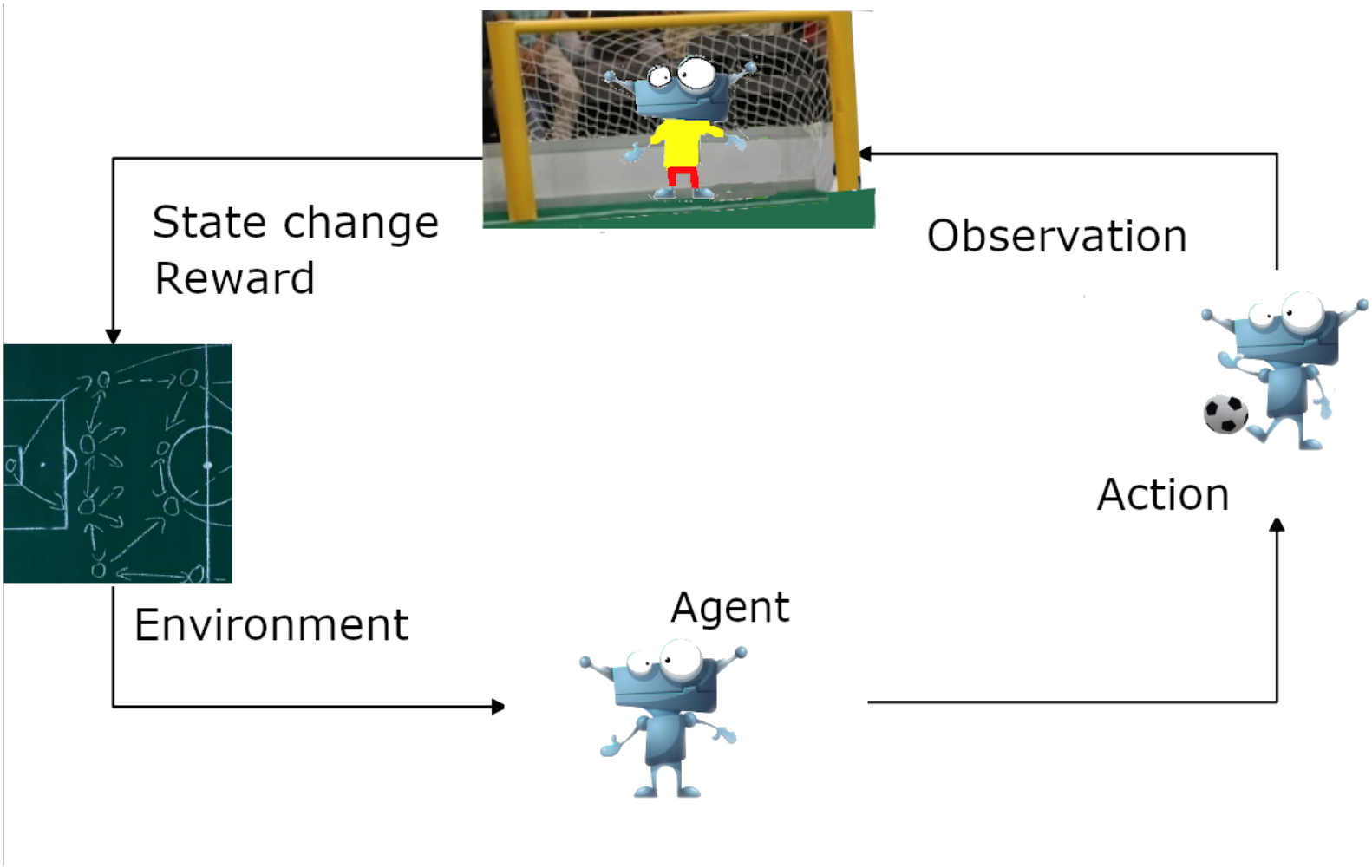


LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UCA UNIVERSITÉ Clermont Auvergne

State and reward

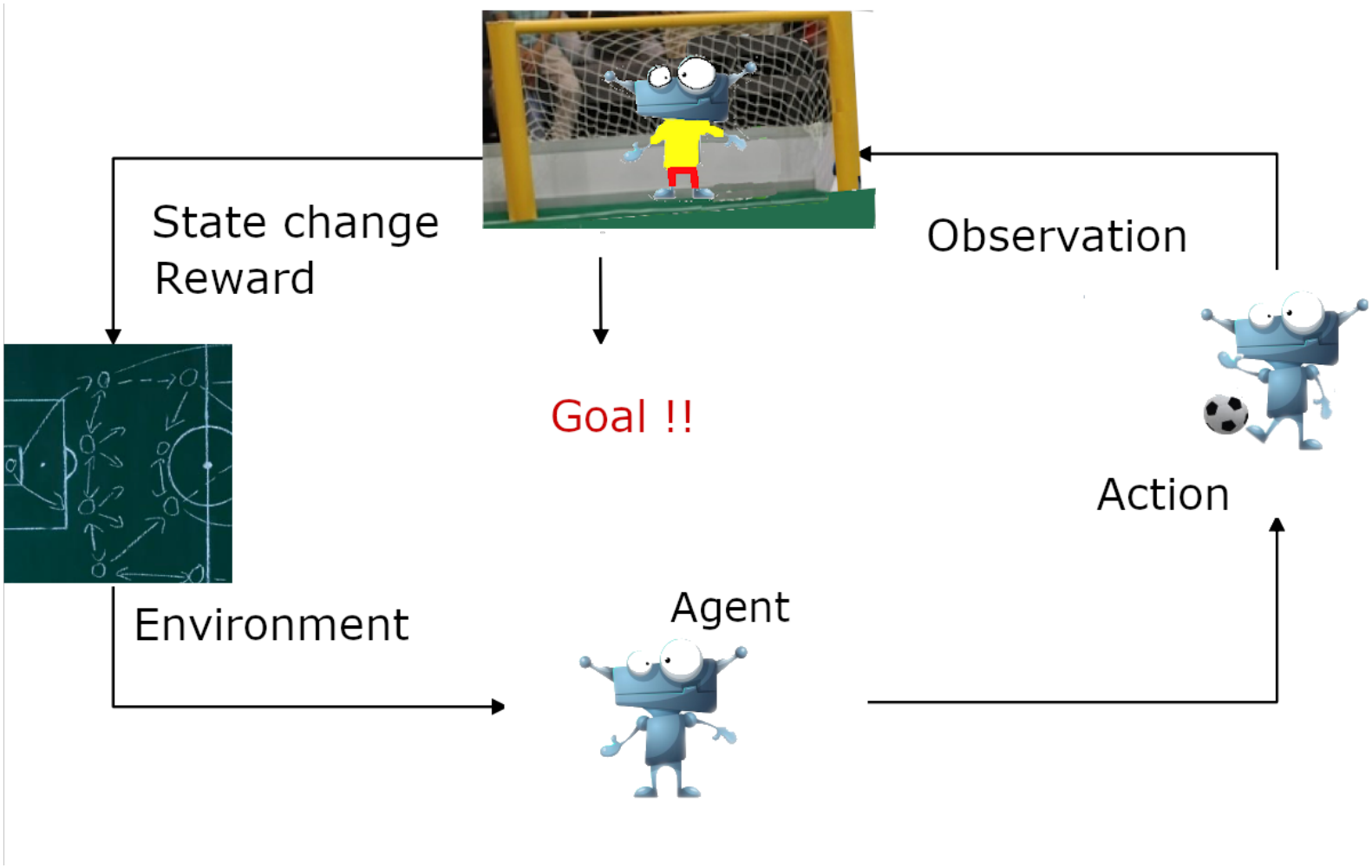


LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UNIVERSITÉ Clermont Auvergne

Objective function



Notations

Space, Action, Reward

For each timestep t:

- $a_t \in A$: action
- $s_t \in S$: state
- $r_t : S \times A \times S \to \mathbb{R}^+$: reward obtained when arriving at s_{t+1} from s_t, taking action a_t

$\forall t$, the agent receives a state representation s_t , selects an action a_t , and receives a reward r_t after which it will find itself in a new state s_{t+1}.

Trajectory: $\tau = (s_0, a_0, r_0, \cdots s_{T-1}, a_{T-1}, r_{T-1}, s_T)$

Notations

Space, Action, Reward

For each timestep t:

- $a_t \in A$: action
- $s_t \in S$: state
- $r_t : S \times A \times S \to \mathbb{R}^+$: reward obtained when arriving at s_{t+1} from s_t, taking action a_t

$\forall t$, the agent receives a state representation s_t , selects an action a_t , and receives a reward r_t after which it will find itself in a new state s_{t+1}.

Trajectory: $\tau = (s_0, a_0, r_0, \cdots s_{T-1}, a_{T-1}, r_{T-1}, s_T)$

Hypothesis

Markovian environment:

- $P(s_{t+1}|s_t, a_t \cdots s_0, a_0) = P(s_{t+1}|s_t, a_t)$
- $P(r_t|s_t, a_t \cdots s_0, a_0) = P(r_t|s_t, a_t)$

Notations

Objective

Total reward: $R_t = \sum_{i=t}^{+\infty} r_i$

Total discounted reward: $R_t = \sum_{i=t}^{+\infty} \gamma^i r_i$

$0 < \gamma < 1$: discount factor, controls the trade-off between immediate and long-term rewards.

Notations

Objective

Total reward: $R_t = \sum_{i=t}^{+\infty} r_i$

Total discounted reward: $R_t = \sum_{i=t}^{+\infty} \gamma^i r_i$

$0 < \gamma < 1$: discount factor, controls the trade-off between immediate and long-term rewards.

Objective: maximise the long-term rewards.

$$R_t = \sum_{i=t}^{+\infty} \gamma^i r_i$$

Value function

$$R_t = \sum_{i=t}^{+\infty} \gamma^i r_i$$

- Estimate how good/bad it is for an agent to be in a particular state
- Goodness of state s_t defined in terms of future rewards that can be expected by from s_t.

Value function

$$R_t = \sum_{i=t}^{+\infty} \gamma^i r_i$$

- Estimate how good/bad it is for an agent to be in a particular state
- Goodness of state s_t defined in terms of future rewards that can be expected by from s_t.

Policy function $\pi : S \rightarrow A$

Value function

$$R_t = \sum_{i=t}^{+\infty} \gamma^i r_i$$

- Estimate how good/bad it is for an agent to be in a particular state
- Goodness of state s_t defined in terms of future rewards that can be expected by from s_t.

Policy function $\pi : S \rightarrow A$

- state-value function $V^{\pi}(s) = \mathbb{E}\left(R_t | s_t = s, \pi\right)$
- state-action function $Q^{\pi}(s_t, a_t) = \mathbb{E}(R_t | s_t, a_t, \pi)$

Value function

- The environment is unknown.
- $V^\pi, Q^\pi \sim$ knowledge representation of an agent, accumulating knowledge through its interactions with the environment, stored by updating its value function.
- If a value function is accurate an agent will know everything he needs to know for interacting with an environment.

Q FUNCTION

Q function

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

Captures the expected reward of a agent being in state s_t at time t, and taking the action a_t.

Question: how to decide actions to take, given a Q function ?

Policy

Policy

- Policy function $\pi : S \rightarrow A$
- Infer the best action to take when being in state $s \in S$

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

Policy

Policy

- Policy function $\pi : S \rightarrow A$
- Infer the best action to take when being in state $s \in S$

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

Value Learning

Find $Q(s, a)$

$$a = arg\max_{a' \in A} Q(s, a')$$

Policy

Policy

- Policy function $\pi : S \rightarrow A$
- Infer the best action to take when being in state $s \in S$

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

Value Learning

Find $Q(s, a)$

$$a = arg\max_{a' \in A} Q(s, a')$$

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Q-LEARNING

In classical reinforcement learning, problem related to Markov Decision Processes (ouf of scope).

Q-learning algorithm

1. build a memory table to store Q-values for all possible combinations of $s \in S$ and $a \in A$
2. as the algorithm evolves, the table is updated...
3. ... until convergence...
4. ... using some tricks (ϵ-greedy policy)

Q-LEARNING

In classical reinforcement learning, problem related to Markov Decision Processes (ouf of scope).

Q-learning algorithm

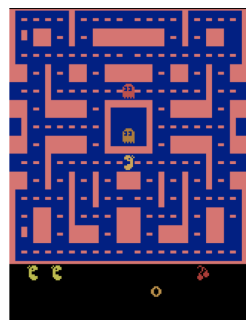
1. build a memory table to store Q-values for all possible combinations of $s \in S$ and $a \in A$
2. as the algorithm evolves, the table is updated...
3. ... until convergence...
4. ... using some tricks (ϵ-greedy policy)

Q-learning does not scale to large problems

If A and/or S are large.

Pacman:

- 240 pellets can be present/absent/eaten
- $2^{240} \approx 10^{72}$ configurations
- + ghost and Pacman positions....



Approximating Q functions

Aim: build an approximation of the Q-function

- Approximate Q-learning: train $Q_{\boldsymbol{W}}$ that will generalize the memory table of the Q-learning algorithm

Approximating Q functions

Aim: build an approximation of the Q-function

- Approximate Q-learning: train $Q_{\boldsymbol{W}}$ that will generalize the memory table of the Q-learning algorithm
- Deep Q-learning: uses a Deep (convolutional) Network as the $Q_{\boldsymbol{W}}$ function

Approximating Q functions

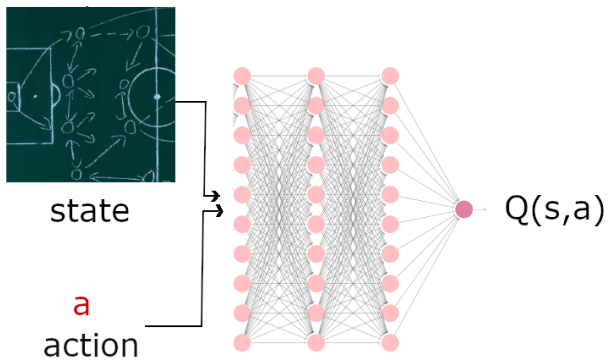
Aim: build an approximation of the Q-function

- Approximate Q-learning: train $Q_{\boldsymbol{W}}$ that will generalize the memory table of the Q-learning algorithm
- Deep Q-learning: uses a Deep (convolutional) Network as the $Q_{\boldsymbol{W}}$ function

V^{π} can be approximated in the same way

Using Deep Networks to model Q functions

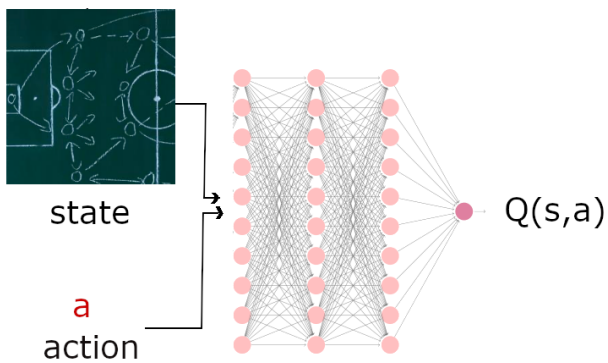
Deep Q-Networks (DQN)



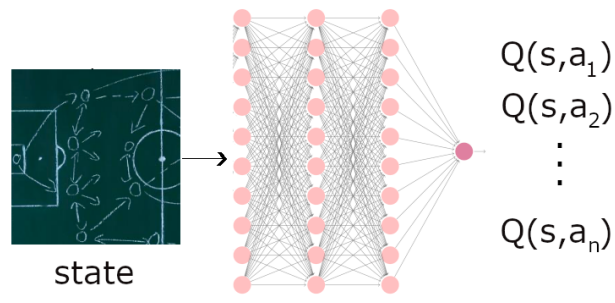
$$S \times A \rightarrow Q(s, a)$$

Using Deep Networks to model Q functions

Deep Q-Networks (DQN)



$$S \times A \rightarrow Q(s,a)$$



$$S \rightarrow Q(s,a)$$

Training

Strategy

- take all the best actions

$\Rightarrow$ Maximize expected reward

$\Rightarrow$ Train the agent

Training

Strategy

- take all the best actions

$\Rightarrow$ Maximize expected reward

$\Rightarrow$ Train the agent

Let put it in equation

Training

Strategy

- take all the best actions

$\Rightarrow$ Maximize expected reward

$\Rightarrow$ Train the agent

$$\underbrace{(r + \gamma \max_{a'} Q(s, a'))}_{\text{target}}$$

Training

Strategy

- take all the best actions

$\Rightarrow$ Maximize expected reward

$\Rightarrow$ Train the agent

$$\underbrace{(r + \gamma \max_{a'} Q(s, a'))}_{\text{target}} \underbrace{Q(s, a)}_{\text{predicted}}$$

Training

Strategy

- take all the best actions

$\Rightarrow$ Maximize expected reward

$\Rightarrow$ Train the agent

$$\underbrace{(r + \gamma \max_{a'} Q(s, a'))}_{\text{target}} \quad \underbrace{- Q(s, a)}_{\text{predicted}}$$

Training

Strategy

- take all the best actions

$\Rightarrow$ Maximize expected reward

$\Rightarrow$ Train the agent

$$\mathcal{L} = \mathbb{E}[||\underbrace{(r + \gamma \max_{a'} Q(s, a'))}_{\text{target}} - \underbrace{Q(s, a)}_{\text{predicted}} ||^2] \quad Q\text{-loss}$$

Training

Strategy

- take all the best actions

$\Rightarrow$ Maximize expected reward

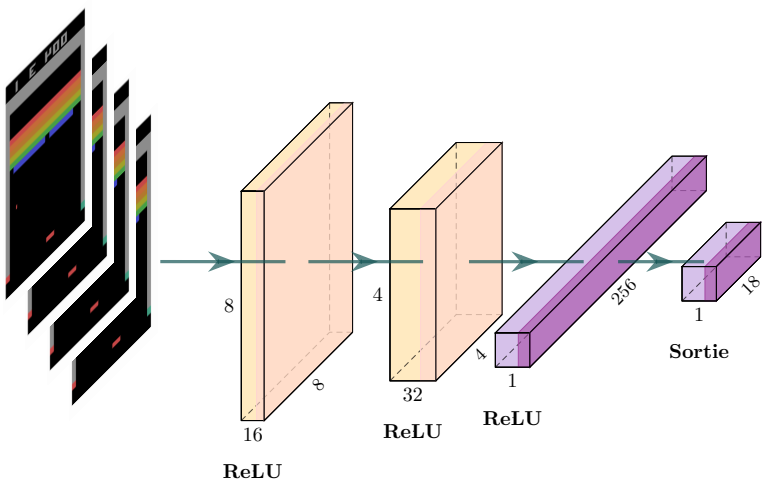
$\Rightarrow$ Train the agent

$$\mathcal{L} = \mathbb{E}[||\underbrace{(r + \gamma \max_{a'} Q(s, a'))}_{\text{target}} - \underbrace{Q(s, a)}_{\text{predicted}} ||^2] \quad Q\text{-loss}$$

And so

1. Use a (deep) neural network to learn Q function and use it to infer the optimal policy $a_t = \pi^*(s_t)$.
2. Send action back to the environment: $s_t \xrightarrow{a_t} s_{t+1}$

Example - DQN



Deep Q-Network (DQN) on 49 ATARI games (Mnih et al., 2015).

- Input $4 \times 84 \times 84$ images
- Conv(16)-ReLU
- Conv(32)-ReLU
- FCC(256)
- Output: FCC(18): joystick positions

Example - DQN

Add-ons

- Experience replay: previous agent experiences (s_t, a_t, s_{t+1}, r_t) are stored in a replay memory and randomly drawned during training

$\Rightarrow$ suppress the correlation between training data

- Use of a target network $Q_{\boldsymbol{W}^-}$, keeping its own parameters and frequently updating them

$\Rightarrow$ reduce the correlations between Q -values and the target $r + \gamma \max_{a'} Q(s, a')$

$$\mathcal{L} = \mathbb{E}\left[\|(r + \gamma \max_{a'} Q_{\boldsymbol{W}_t^-}(s, a')) - Q_{\boldsymbol{W}_t}(s, a)\|^2 \right]$$

Example - DQN

Episode = a complete run of game; S: target network update step; T: length of a game : x_t input at time t ;

ϕ: preprocessing function

begin

Input : Image of the game; score, T, S

Output : Q value

Initialize Experience replay memory D

Initialize Q with random weights $\boldsymbol{W}$

Initialize target $\hat{Q}$ with weights $\boldsymbol{W}^- = \boldsymbol{W}$

foreach *episode* **do**

Initialize a sequence $s_1 = \{x_1\}, \phi_1 = \phi(s_1)$

for t=1 *to* T **do**

Select an action $a_t = \begin{cases} \text{randomly} & \text{with probability } \epsilon \\ Arg\max_a Q(\phi(s_t), a, \boldsymbol{\theta}) & \text{otherwise} \end{cases}$

Execute a_t and observe r_t and x_{t+1}

$s_{t+1} = s_t, a_t, x_{t+1}$

$\phi_{t+1} = \phi(s_{t+1})$

$D \leftarrow D \cup (\phi_t, a_t, r_t, \phi_{t+1})$;

`/* Experience replay */`

Sample a batch $(\phi_j, a_j, r_j, \phi_{j+1})$ from D ;

$y_j = \begin{cases} r_j & \text{if episode finishes in } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a', \boldsymbol{\theta}^-) & \text{otherwise} \end{cases}$

Gradient descent step w.r.t. $\boldsymbol{\theta}$ on $(y_j - Q(\phi_j, a_j, \boldsymbol{\theta}))^2$

`/* Update target network */`

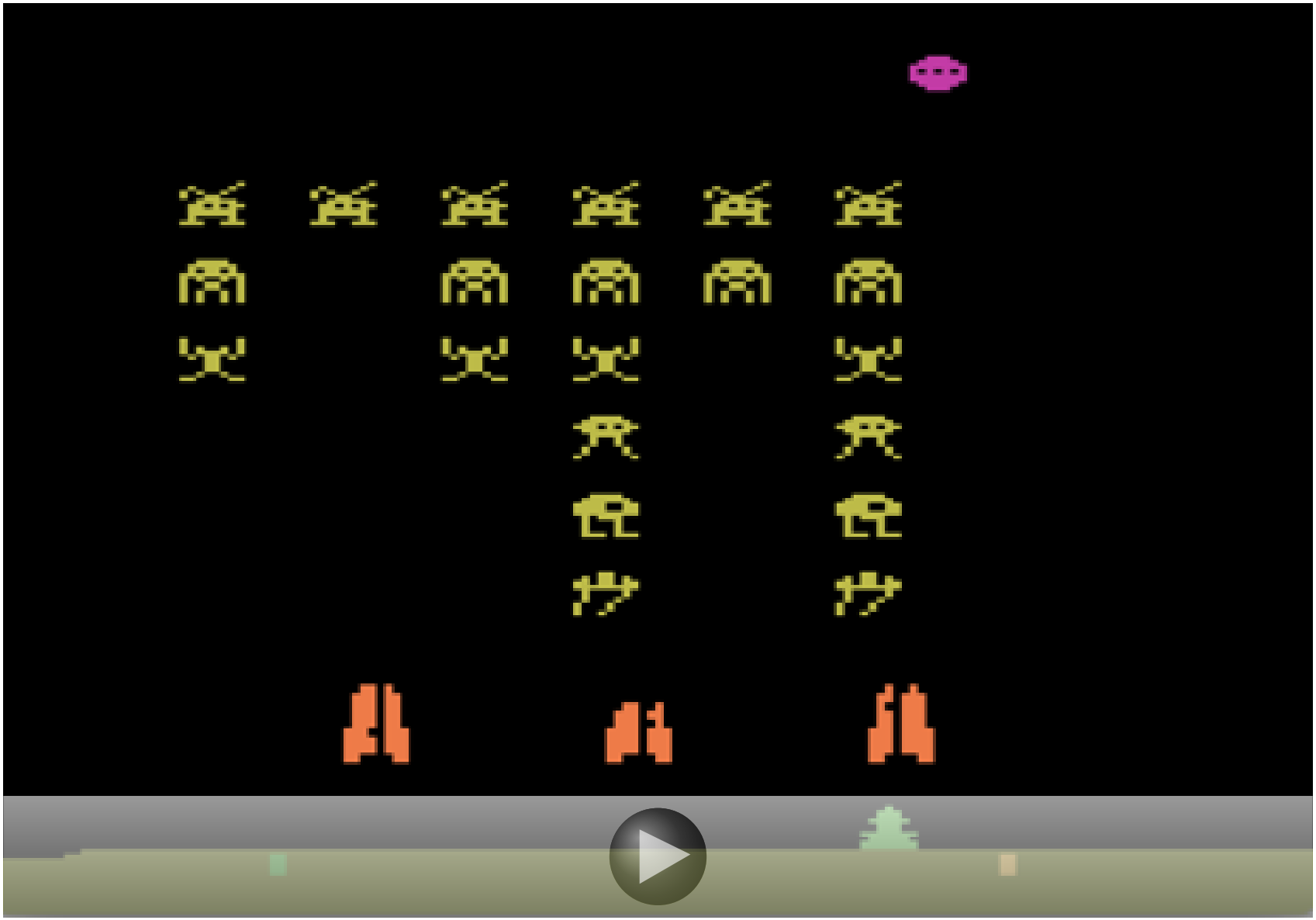
Each S step do $\boldsymbol{\theta}^- = \boldsymbol{\theta}$

end

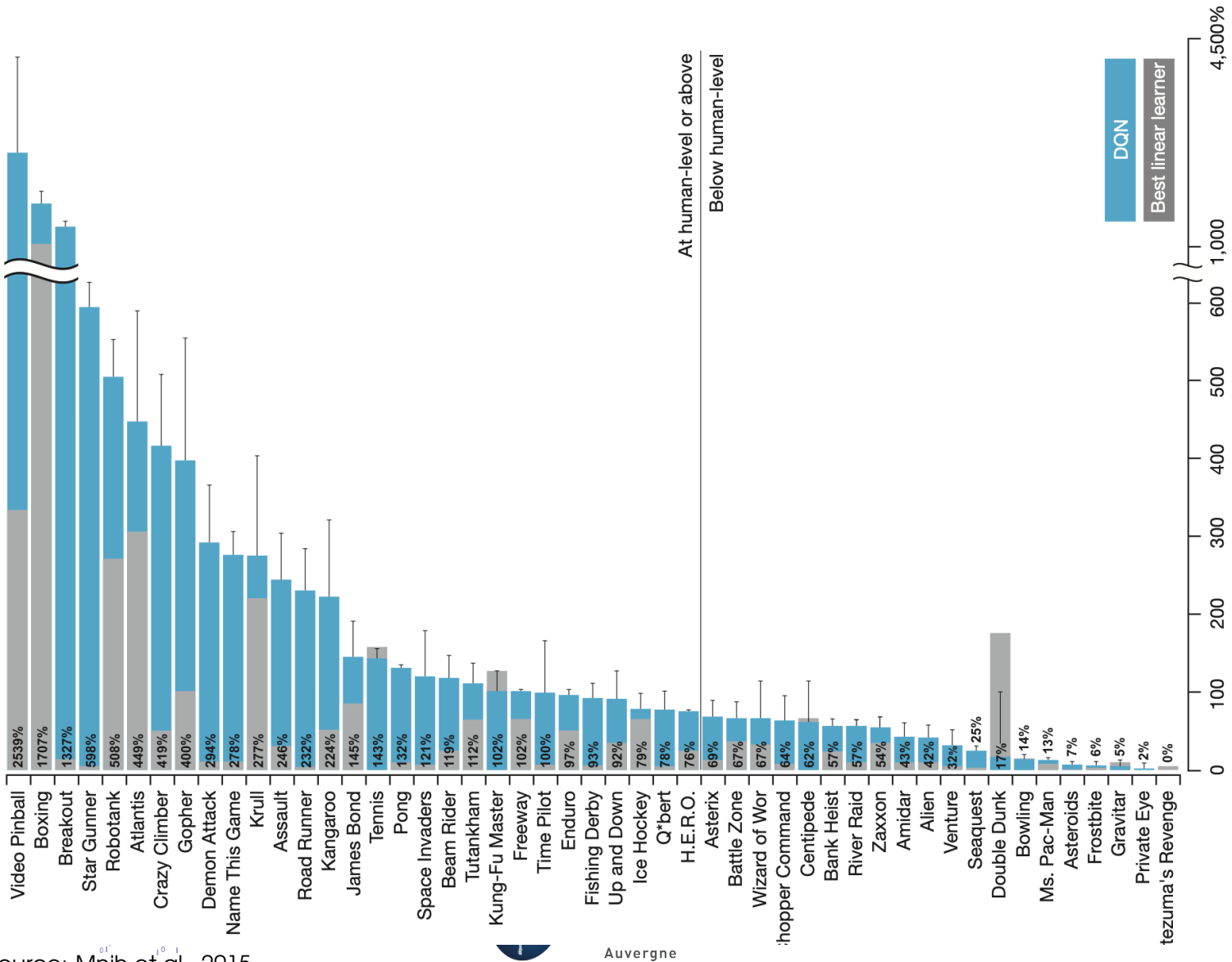
end

end

Example - DQN



Example - DQN

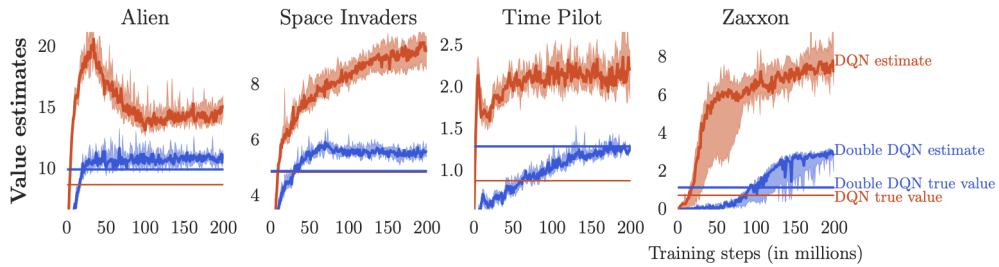


source: Mnih et al., 2015

Improvments

Improvments

- Double DQN (2016)



- Q Learning has been shown to overestimate its targets, because it uses a single estimator for estimating the Q values and choosing the maximum over actions.
- Solution: to overcome this, Double Q Learning uses a double estimator technique:
  - The target network is used for estimation
  - The online network for choosing the greedy action

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

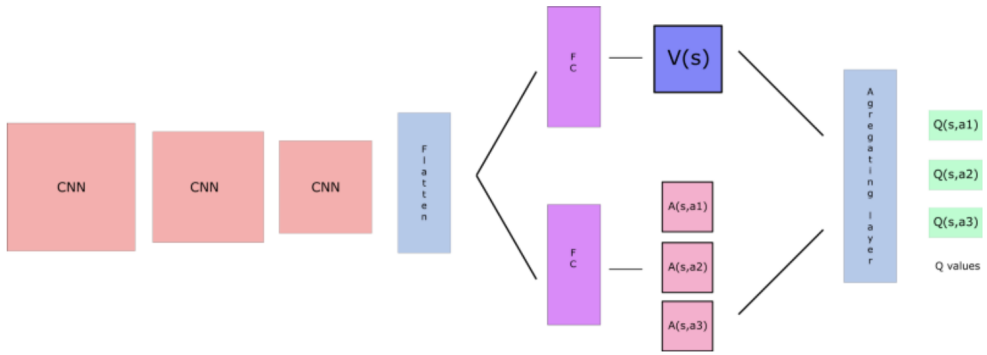
cnrs

UNIVERSITÉ Clermont Auvergne

Improvments

Improvments

- Double DQN (2016)
- Dueling DQN (2016)



Dueling architecture:

- Decompose the Q value into a state-dependent part and a state-action dependent part
- This allows to share the state-dependent estimations (good for actions that are less chosen), focuses in estimating the state-action dependent part (good for relative ranking of the actions).

Pros and cons

Flexibility

- Q determines π^* in a deterministic way

$\Rightarrow$ cannot manage stochastic policies

Complexity

- OK if $|A|$ small, and A discrete
- KO if A is continuous

Pros and cons

Flexibility

- Q determines π^* in a deterministic way

$\Rightarrow$ cannot manage stochastic policies

Complexity

- OK if $|A|$ small, and A discrete
- KO if A is continuous

See next lecture

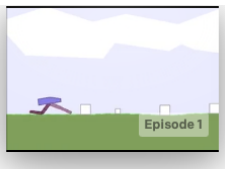
Policy gradient methods

Gym

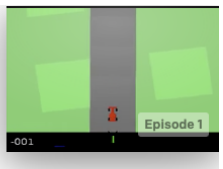
Gym: toolkit for developing and comparing reinforcement learning algorithms

Environments

BipedalWalker-v2
Train a bipedal robot to walk.



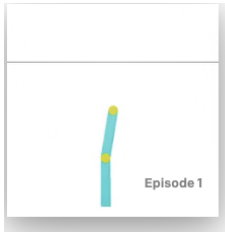
BipedalWalkerHardcore-v2
Train a bipedal robot to walk over rough terrain.



CarRacing-v0
Race a car around a track.

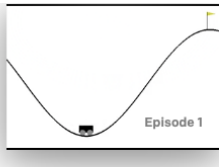
LunarLander-v2
Navigate a lander to its landing pad.

LunarLanderContinuous-v2
Navigate a lander to its landing pad.

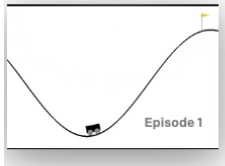


Acrobot-v1
Swing up a two-link robot.

CartPole-v1
Balance a pole on a cart.



MountainCar-v0
Drive up a big hill.



MountainCarContinuous-v0
Drive up a big hill with continuous control.

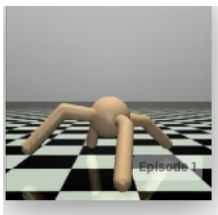
Pendulum-v0
Swing up a pendulum.

LIMOS
LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

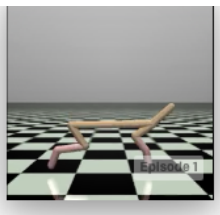
cnrs

UNIVERSITÉ Clermont Auvergne

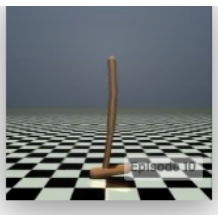
Environments



Ant-v2
Make a 3D four-legged robot walk.



HalfCheetah-v2
Make a 2D cheetah robot run.



Hopper-v2
Make a 2D robot hop.

Humanoid-v2
Make a 3D two-legged robot walk.

HumanoidStandup-v2
Make a 3D two-legged robot standup.

InvertedDoublePendulum-v2
Balance a pole on a pole on a cart.

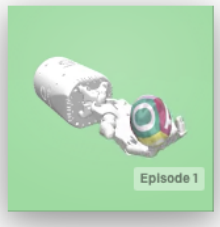
FetchPickAndPlace-v1
Lift a block into the air.

FetchPush-v1
Push a block to a goal position.

FetchReach-v1
Move Fetch to a goal position.

FetchSlide-v1
Slide a puck to a goal position.

HandManipulateBlock-v0
Orient a block using a robot hand.



HandManipulateEgg-v0
Orient an egg using a robot hand.

HandManipulatePen-v0
Orient a pen using a robot hand.

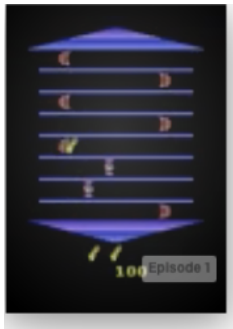
HandReach-v0
Reach a pose using a robot hand.

LIMOS
LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

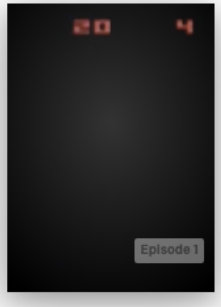
cnrs

UNIVERSITÉ Clermont Auvergne

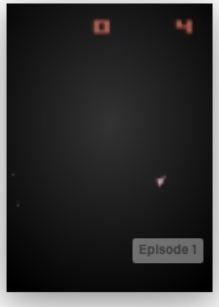
Environments



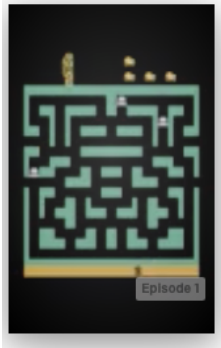
Asterix-v0
Maximize score in the game Asterix, with screen images as input



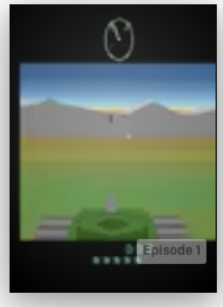
Asteroids-ram-v0
Maximize score in the game Asteroids, with RAM as input



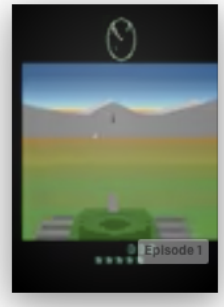
Asteroids-v0
Maximize score in the game Asteroids, with screen images as input



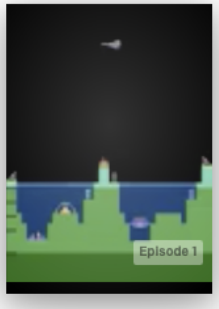
BankHeist-v0
Maximize score in the game BankHeist, with screen images as input



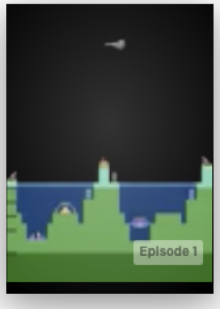
BattleZone-ram-v0
Maximize score in the game BattleZone, with RAM as input



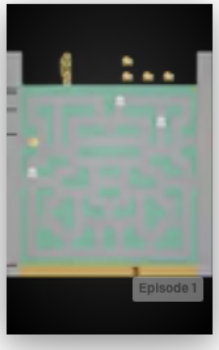
BattleZone-v0
Maximize score in the game BattleZone, with screen images as input



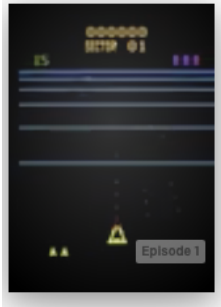
Atlantis-ram-v0
Maximize score in the game Atlantis, with RAM as input



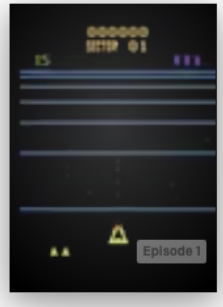
Atlantis-v0
Maximize score in the game Atlantis, with screen images as input



BankHeist-ram-v0
Maximize score in the game BankHeist, with RAM as input



BeamRider-ram-v0
Maximize score in the game BeamRider, with RAM as input



BeamRider-v0
Maximize score in the game BeamRider, with screen images as input

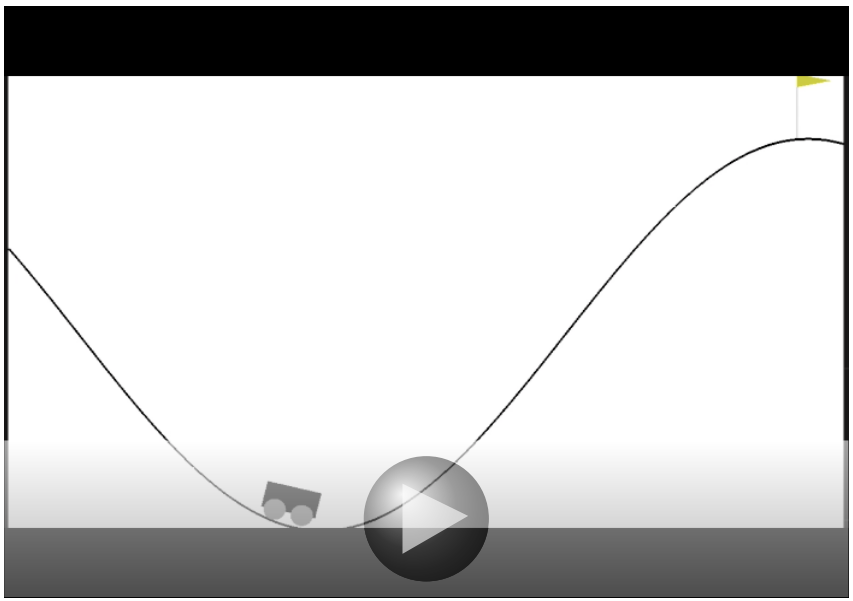
Berzerk-ram-v0
Maximize score in the game Berzerk, with RAM as input

LIMOS
LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UNIVERSITÉ Clermont Auvergne

Code



```python
import gym
# Game to play
env = gym.make("MountainCar-v0")

# initial observation of the environment
observation = env.reset()

# A
print(env.action_space)

# S
print(env.observation_space)

for t in range(1000):

  env.render()

  # The agent chooses an action a(t)
  action = env.action_space.sample()

  # return s(t+1), r(t+1), time to reset, debugging info
  observation, reward, done, info = env.step(action)

  if done:
    # episode terminated
    observation = env.reset()
env.close()
```