Reinforcement Learning – Part 2

Deep Reinforcement Learning

Alberto Sardinha
sardinha@inf.puc-rio.br

Outline

- **Value-based RL**
- TD Learning
- SARSA
- Q-learning
- Deep RL

Reinforcement Learning

- **Model-based RL**
  - Learn de reward function and transition probabilities
  - Use planning (e.g., Value Iteration)
  - Extract policy

- **Value-based RL**
  - Learn the value function directly (e.g., Q^*)
  - Extract policy

- **Policy-based RL**
  - Learn the policy directly

Value-based RL

- “Most popular” RL methods
- Include **many** methods
  - E.g., Monte Carlo methods, TD methods, etc.

Outline

- Value-based RL
- **TD Learning**
- SARSA
- Q-learning
- Deep RL

TD Learning

- **Temporal-difference (TD) learning is a model-free RL approach**
  - Learn by **bootstrapping** from the current estimate (just like Dynamic Programming)
  - Learn by **sampling the environment** (just like Monte Carlo methods)

TD Learning

- **What is the difference between model-based RL and model-free RL?**

  - **Model-based RL:** learns the transition probabilities and reward function + planning

  - **Model-free RL**: opposite (e.g., learns the $V^*(s)$ or $Q^*(s, a)$ directly)

TD Learning

- **TD(0) – Estimating $V^{\pi}(s)$**
  - For every new (s_t, r_{t+1}, s_{t+1})
  - Update

$$V(s_t) \leftarrow V(s_t) + \alpha[\underbrace{r_{t+1} + \gamma V(s_{t+1}) - V(s_t)}_{\text{TD Error}}]$$

TD Learning

- ***TD Error***

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

Where

$$V(s_t)$$

is the estimated value of s_t ,

$$r_{t+1} + \gamma V(s_{t+1})$$

is the better estimate

TD Learning

- We also want to use TD prediction for the **control problem** (i.e., finding an optimal policy)

  - **On-policy method** – estimate function (e.g., $Q^{\pi}(s, a)$) for the current behavior policy π

  - **Off-policy method** – estimate function (e.g., $Q^{\pi'}(s, a)$) for a different policy π' than the behavior policy π

Outline

- Value-based RL
- TD Learning
- **SARSA**
- Q-learning
- Deep RL

SARSA

- **SARSA is an on-policy TD Control**
  - For every new $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$
  - Update

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

SARSA

- We start with some estimate Q
- Initialize current state s and choose some action a (e.g., using ϵ-greedy)
- Loop for each step:
  - Take action a and observe next state s' and reward r
  - choose some action a' (e.g., using ϵ-greedy)
  - Update Q estimate according to
    $$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$$
  - $s \leftarrow s'$
  - a $\leftarrow a'$

Outline

- Value-based RL
- TD Learning
- SARSA
- **Q-learning**
- Deep RL

Q-learning

- **Q-learning is an off-policy TD Control**
  - For every new $(s_t, a_t, r_{t+1}, s_{t+1})$
  - Update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \underbrace{\left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right]}_{\textit{TD Error}}$$

Q-learning

- We start with some estimate Q
- Initialize current state s
- Loop for each step:
  - Choose some action a (e.g., using ϵ-greedy)
  - Take action a and observe next state s' and reward r
  - Update Q estimate according to
    $$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$
  - $s \leftarrow s'$

Value-based RL

- Does Q-learning work?

  - Theorem: **As long as every state-action pair is visited infinitely often, Q-learning converges to Q^* w.p.1.**

Let us
revisit
this

Exploration vs. Exploitation

- **How can we visit every state action pair infinitely often?**
  - In practice, “infinitely often” means a “large number of times”

Exploration vs. Exploitation

- The agent needs to **try all actions in all states many times**
- But this means that the agent will keep **doing sub-optimal actions for a long time!**

Exploration vs. Exploitation

On the other hand...

Exploration vs. Exploitation

- We want the agent to start **acting "reasonably" as soon as possible**
  - In practice, this means **using knowledge already available**
  - The agent needs to stop "trying" and start "doing"

Exploration vs. Exploitation

- The agent needs to balance:
  - **Exploration**: trying new actions or actions that have not been selected very often
  - **Exploitation**: using learned knowledge to select actions

Exploration vs. Exploitation tradeoff

Exploration vs. Exploitation

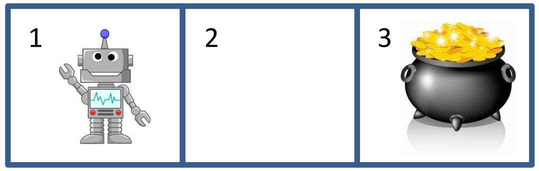
- **Heuristic for exploration vs. exploitation**
  - ε-greedy
    - Agent selects a random action with probability ε (exploration)
    - Agent selects the greedy action (i.e., action with highest Q-value) with probability 1 - ε (exploitation)

$\downarrow$

ε may decay with time

Example

- We have the following MDP:
  - $S = \{1, 2, 3\}$
  - $A = \{left, right\}$
  - $\cancel{\boldsymbol{P(s'|s, a = left) =?}}$
  - $\cancel{\boldsymbol{P(s'|s, a = right) =?}}$
  - $\cancel{\boldsymbol{R(s, a) =?}}$
  - $\gamma = 0.9$



Example

- We start with some estimate Q

$$Q = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

- Initialize current state s

$$s \to 1$$

Example

- First iteration (current state $s \rightarrow 1$)
  - Choose some action a
    - $a \rightarrow left$
  - Take action a and observe next state s' and reward r
    - $s' \rightarrow 1$
    - $r \rightarrow 0$
  - Update Q estimate according to
    - $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    - $Q(1, left) \leftarrow 1 + 0.3[0 + 0.9 \times 1 - 1]$
    - $Q(1, left) \leftarrow 0.97$

Example

- Updated Q

$$Q = \begin{bmatrix} 0.97 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Example

- Second iteration (current state $s \rightarrow 1$)
  - Choose some action a
    - $a \rightarrow right$
  - Take action a and observe next state s' and reward r
    - $s' \rightarrow 2$
    - $r \rightarrow 0$
  - Update Q estimate according to
    - $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    - $Q(1, right) \leftarrow 1 + 0.3[0 + 0.9 \times 1 - 1]$
    - $Q(1, right) \leftarrow 0.97$

Example

- Updated Q

$$Q = \begin{bmatrix} 0.97 & 0.97 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Example

- Third iteration (current state $s \rightarrow 2$)
  - Choose some action a
    - $a \rightarrow left$
  - Take action a and observe next state s' and reward r
    - $s' \rightarrow 1$
    - $r \rightarrow 0$
  - Update Q estimate according to
    - $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    - $Q(2, left) \leftarrow 1 + 0.3[0 + 0.9 \times 0.97 - 1]$
    - $Q(2, left) \leftarrow 0.9619$

Example

- Updated Q

$$Q = \begin{bmatrix} 0.97 & 0.97 \\ 0.9619 & 1 \\ 1 & 1 \end{bmatrix}$$

Example

- Updated Q (after many iterations)

$$Q = \begin{bmatrix} 6.86 & 7.99 \\ 7.22 & 8.91 \\ 9.2 & 10 \end{bmatrix}$$

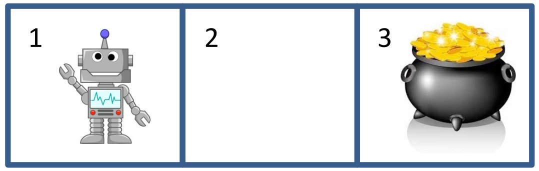
- Recall the Q^* with Value Iteration

$$Q^* = \begin{bmatrix} 6.89 & 7.66 \\ 7.08 & 8.73 \\ 9.07 & 9.95 \end{bmatrix}$$

Example

- After computing Q, we can extract the policy as follows:

$$\pi^*(s) \in \underset{a \in A}{\operatorname{argmax}} \, Q(s, a)$$



$$\pi^* = \begin{bmatrix} right \\ right \\ right \end{bmatrix}$$

$$\pi^* = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Example

```python
import numpy as np
np.set_printoptions(precision=2, suppress=True)

# States
S = ['1', '2', '3']

# Actions
A = ['L', 'R']

# Transition probabilities

L = np.array([[1.0, 0.0, 0.0],
              [0.8, 0.2, 0.0],
              [0.0, 0.8, 0.2]])

R = np.array([[0.2, 0.8, 0.0],
              [0.0, 0.2, 0.8],
              [0.0, 0.0, 1.0]])

P = [L, R]

# Reward function

R = np.array([[0.0, 0.0],
              [0.0, 0.0],
              [1.0, 1.0]])

gamma = 0.9
```

Example

```python
def egreedy(Q,state,eps):

    p = np.random.random()

    if p < eps:
        action = np.random.choice(num_actions)
    else:
        action = np.argmax(Q[state,:])

    return action
```

Example

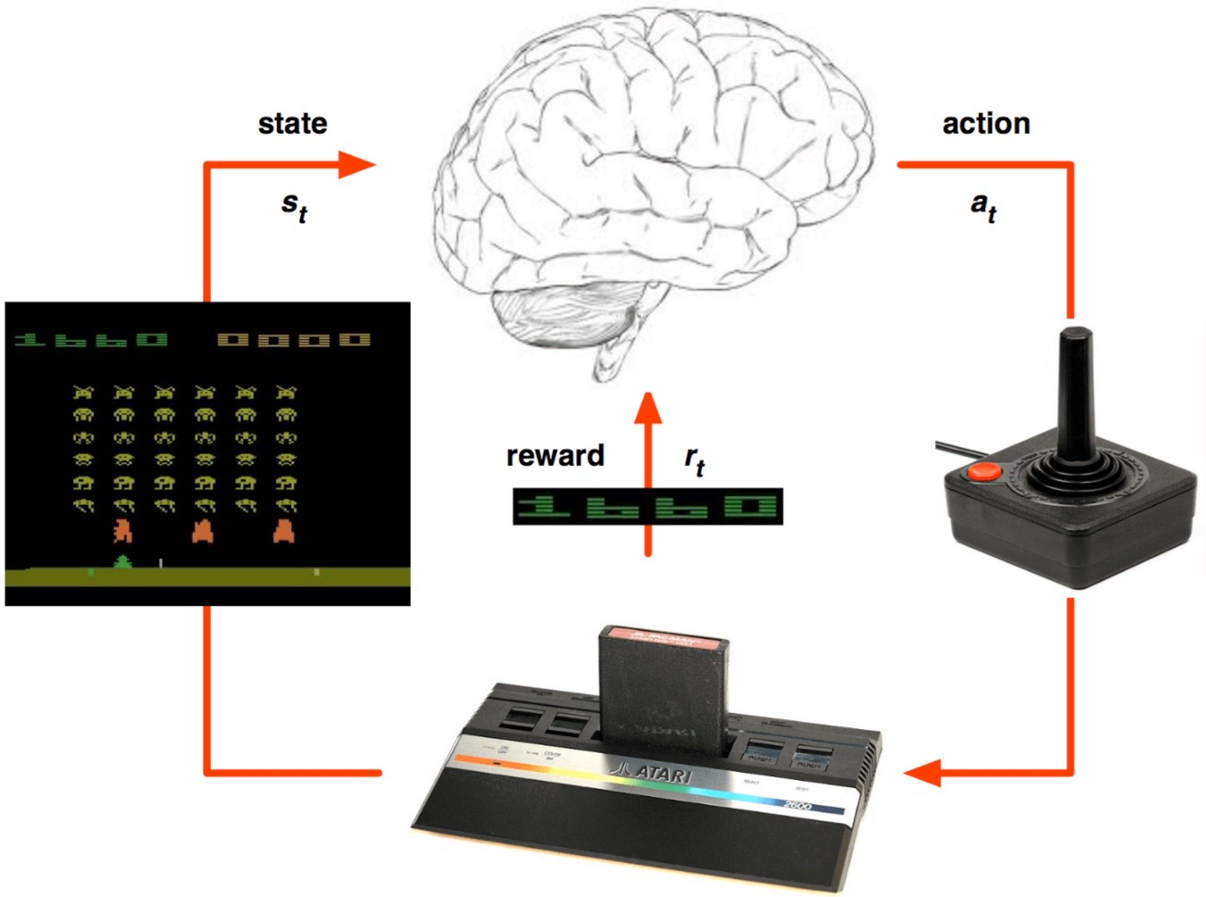
```python
STEPS = 1000000
num_actions = len(A)
num_states = len(S)
ALPHA = 0.3

# Initialize Q-values
Q = np.ones((num_states, num_actions))

# Initialize current state
state = 0

for t in range(STEPS):

    # choose action
    action = egreedy(Q,state,0.05)

    # choose next state
    next_state = np.random.choice(num_states, p=P[action][state, :])

    # obtain reward
    reward = R[state,action]

    # Update Q
    Q[state, action] = Q[state, action] + ALPHA*(reward + gamma*max(Q[next_state, :]) - Q[state, action])

    state = next_state

print(Q)
```

Outline

- Value-based RL
- TD Learning
- SARSA
- Q-learning
- **Deep RL**

Deep Reinforcement Learning

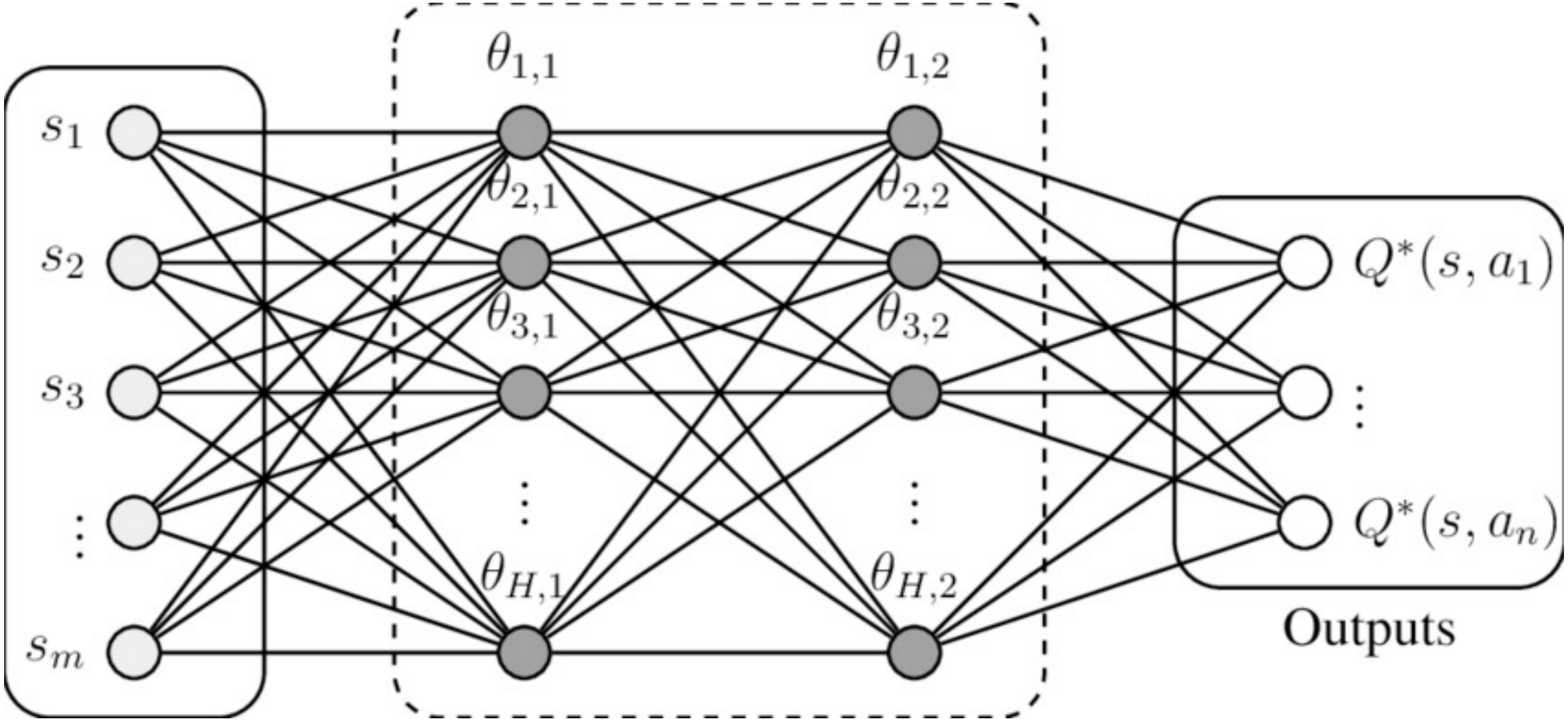
Google DeepMind's Deep Q-network (DQN)



Deep Reinforcement Learning

- Used in domain with **large state space**
- We can **no longer represent Q^* exactly**
- We must resort to some **form of approximation** (e.g., neural network)
- Function approximation **does not retain its convergence guarantees**

Deep Reinforcement Learning



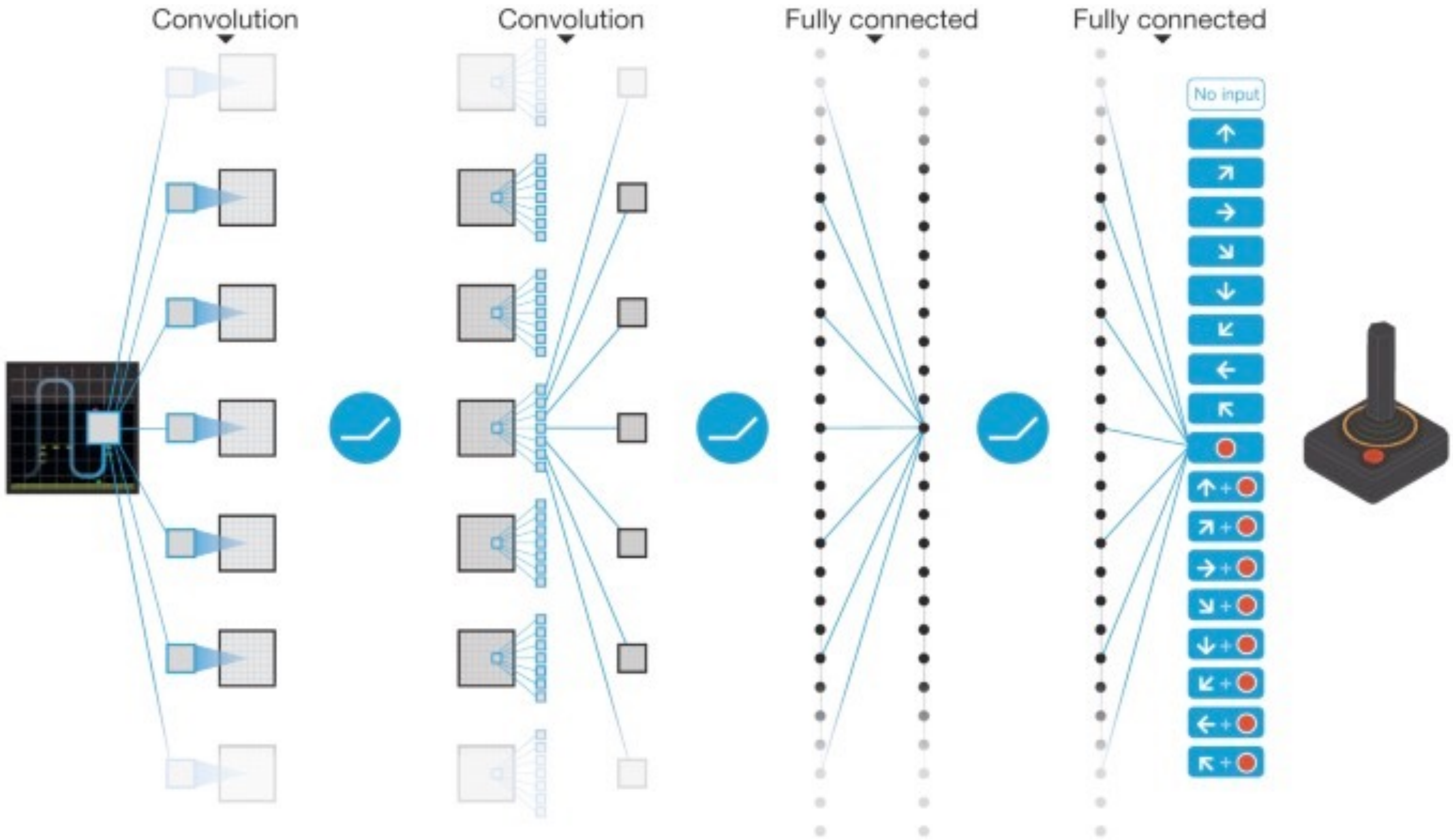
State = vector of features

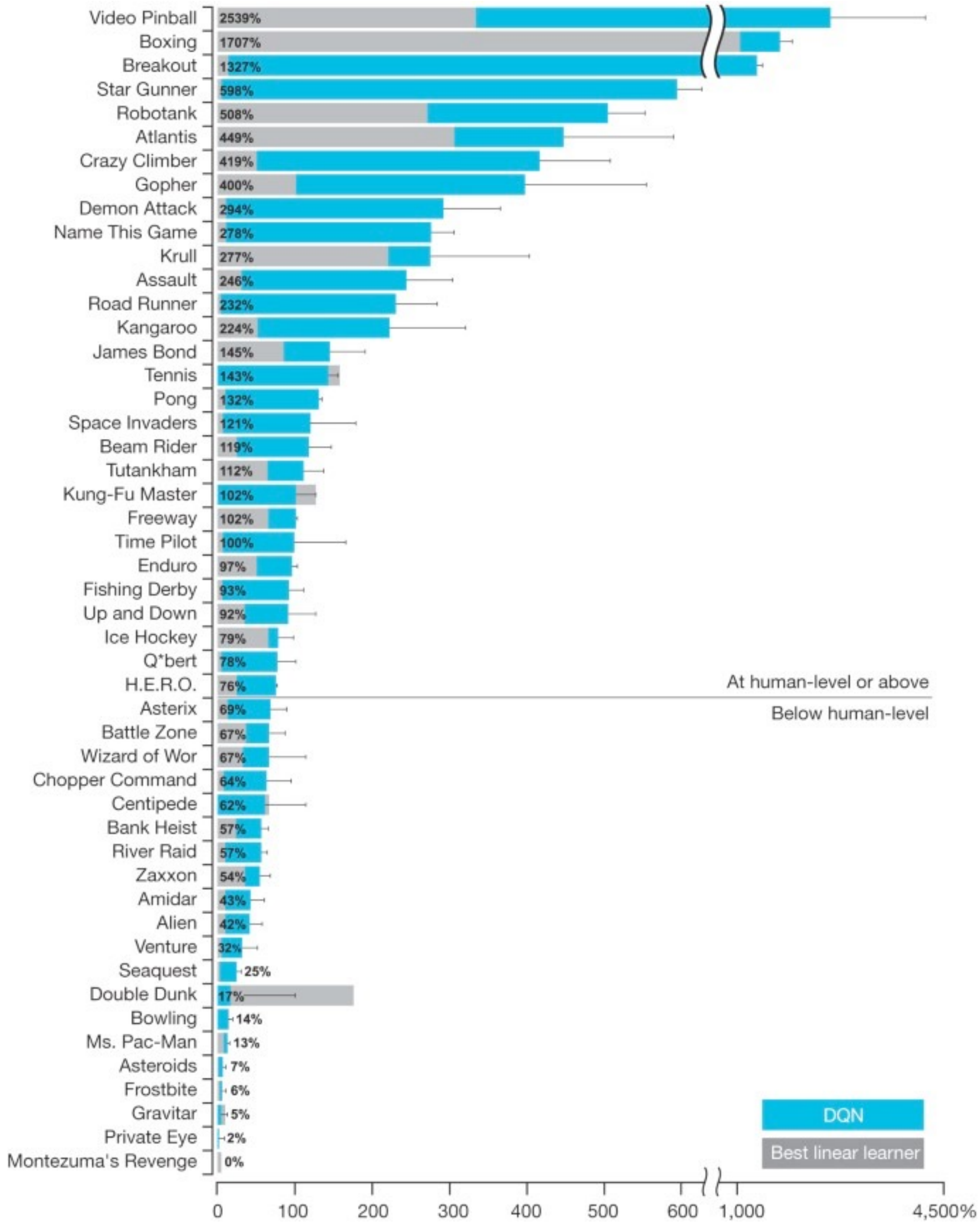
Deep Q-networks

- Introduction to Deep Reinforcement Learning

https://www.youtube.com/watch?v=wrBUkpiRvCA

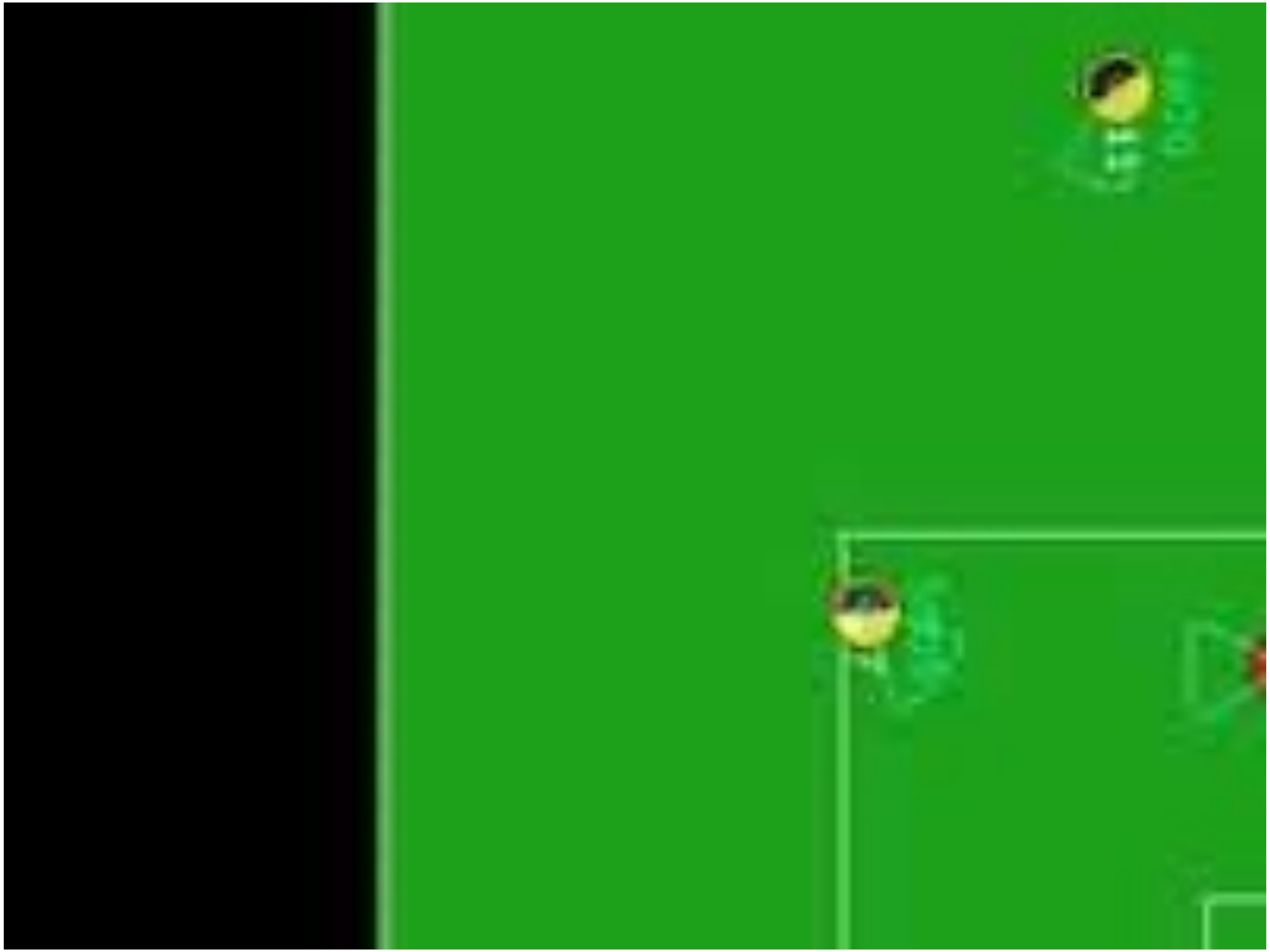
Deep Reinforcement Learning





Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pac-Man 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
At human-level or above
Below human-level
DQN
Best linear learner
0
100
200
300
400
500
600
1,000
4,500%

DQN in Action



Thank You

sardinha@inf.puc-rio.br