Deep Q-Learning

Cristian Muñoz

AI Game Playing

Supervised Learning

- Regression and Classification

Unsupervised Learning

- Autoencoders, GANs e Generative Models

Reinforcement Learning

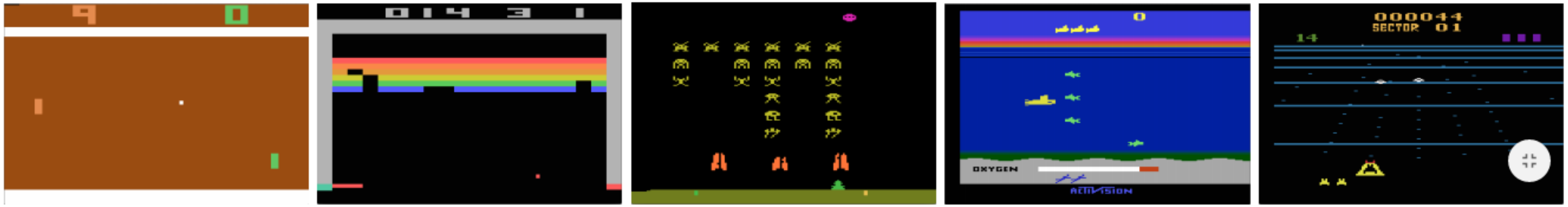
- **Deep reinforcement learning**

Deep Reinforcement Learning

DeepMind (British company) , 2013

“Playing Atari with Deep Reinforcement Learning”, V. Mnih

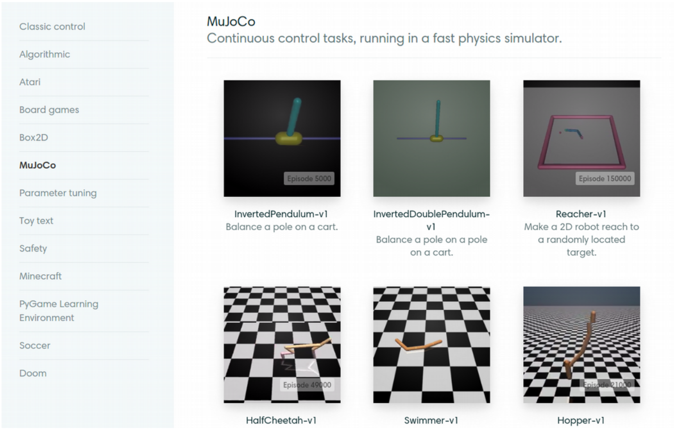
Describe how a CNN could be taught to play Atari 2600 video games by showing it screen pixels and giving it a reward when the score increases.



Modulos Python

OpenAI Gym

https://gym.openai.com/envs

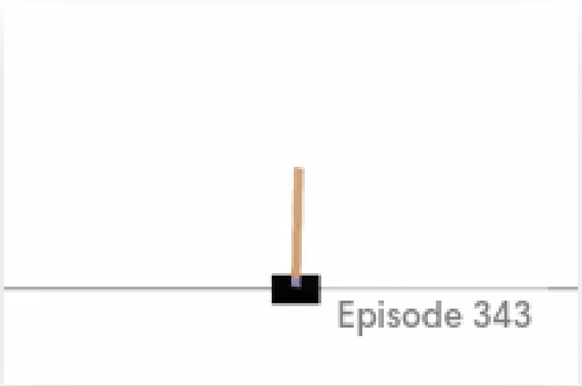


Pygame

https://www.pygame.org/news

Exemplo 1: Pêndulo invertido

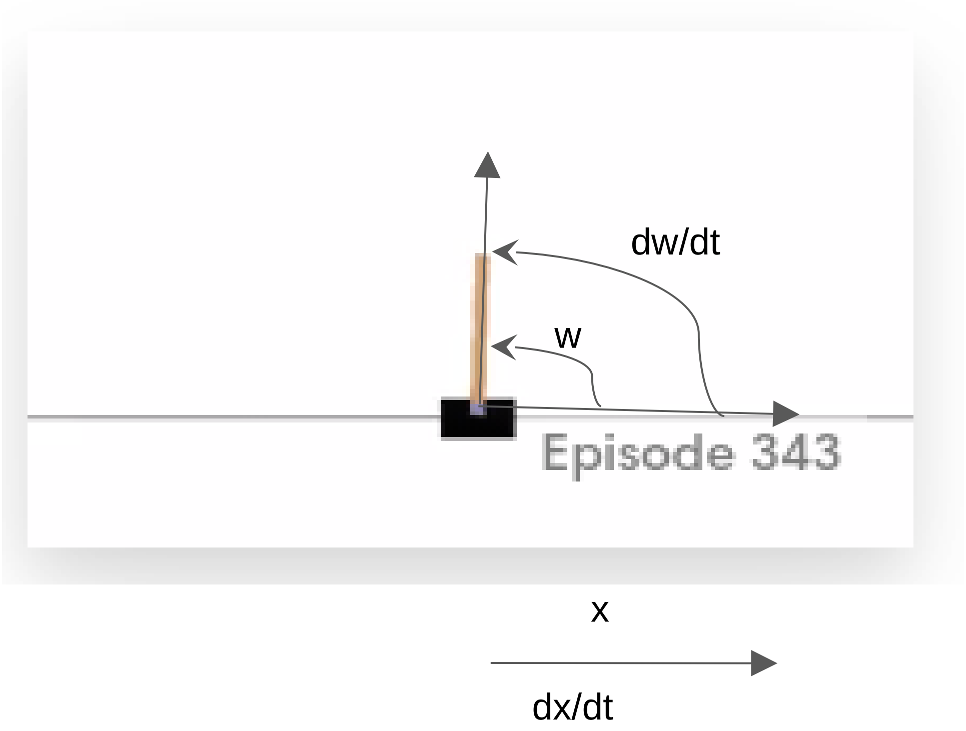
A pole is attached by an un-actuated join to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode end when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.



Parameters

States: x , dx/dt, w , dw/dt

Actions: Force (+1 or -1)



Ver código

Results

Q-function => Neural Network

Dataset???

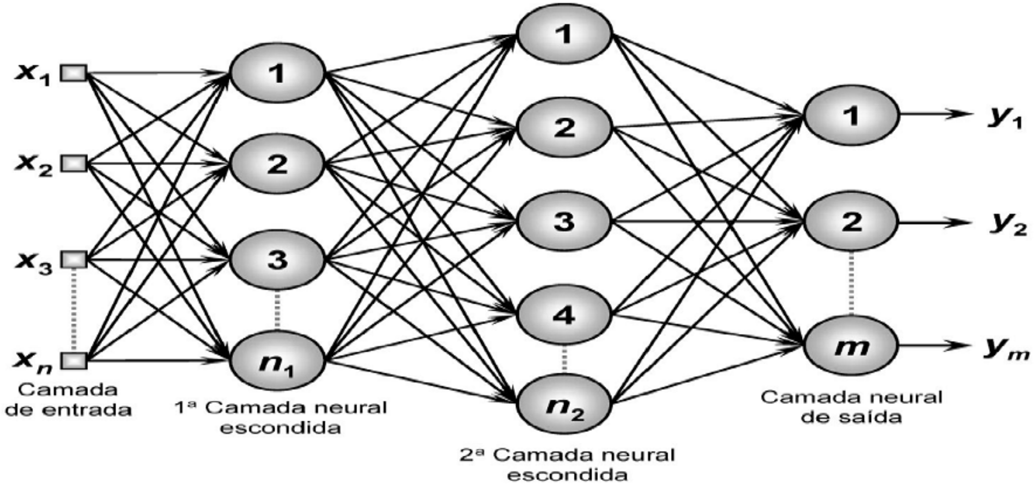
x

dx/dt

States

w

dw/dt



Q-value -- action 1

Q-value -- action 2

Q-value -- action m

Results

Memory = list(s_{t-1}, a_t, r_t, s_t, done?)

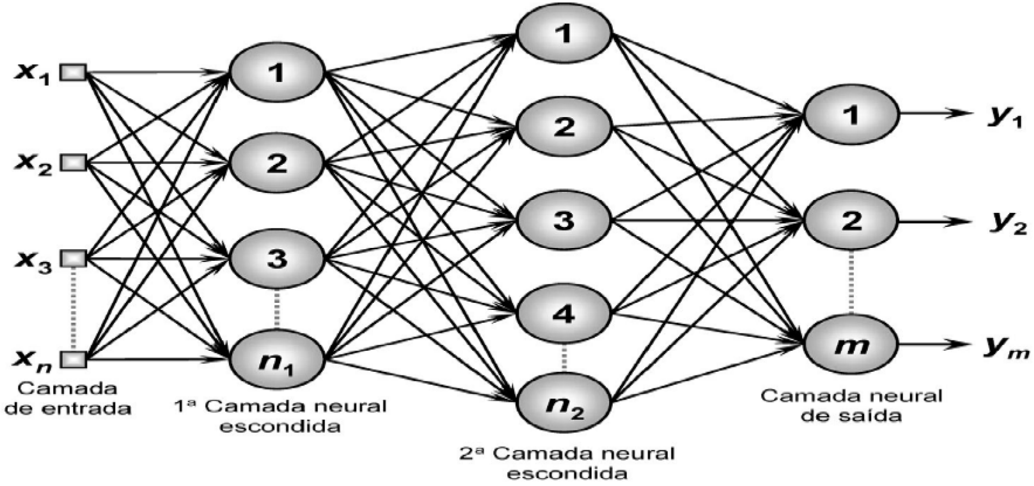
States

x

dx/dt

w

dw/dt



Q-value -- action 1

Q-value -- action 2

Q-value -- action m

Results

Memory = list(s_{t-1}, a_t, r_t, s_t, done?)

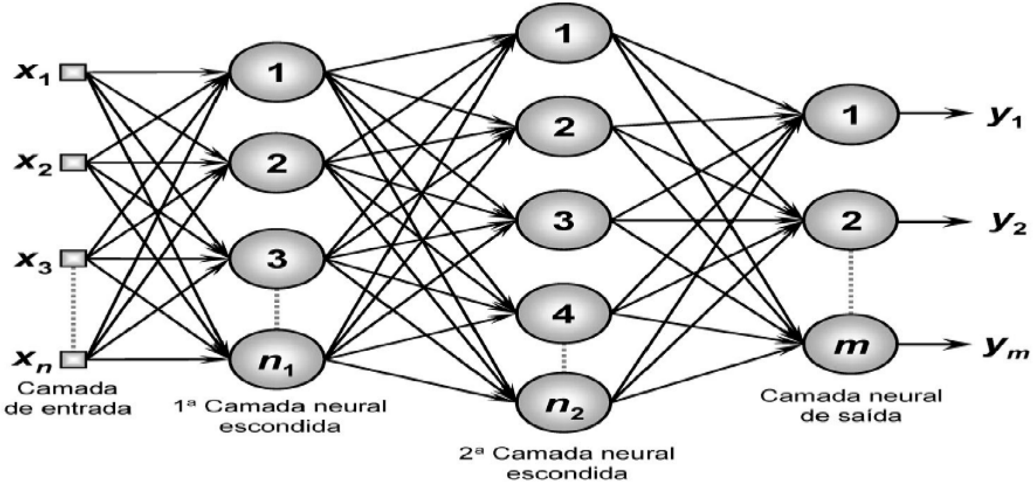
States

x

dx/dt

w

dw/dt



Q-value -- action 1

Q-value -- action 2

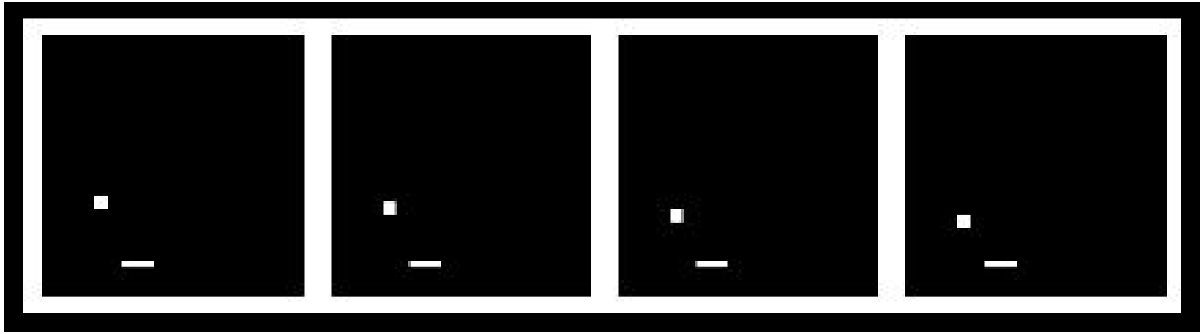
Q-value -- action m

Results

```
Epoch 0233/250 | Loss 0.77897 | score 194.00000 | reward 195.000000
Epoch 0234/250 | Loss 0.74768 | score 194.00000 | reward 195.000000
Epoch 0235/250 | Loss 0.47564 | score 194.00000 | reward 195.000000
Epoch 0236/250 | Loss 0.58068 | score 194.00000 | reward 195.000000
Epoch 0237/250 | Loss 0.35535 | score 194.00000 | reward 195.000000
Epoch 0238/250 | Loss 0.44100 | score 194.00000 | reward 195.000000
Epoch 0239/250 | Loss 0.42452 | score 194.00000 | reward 195.000000
Epoch 0240/250 | Loss 0.29919 | score 194.00000 | reward 195.000000
Epoch 0241/250 | Loss 0.24800 | score 194.00000 | reward 195.000000
Epoch 0242/250 | Loss 0.26820 | score 194.00000 | reward 195.000000
Epoch 0243/250 | Loss 0.28295 | score 194.00000 | reward 195.000000
Epoch 0244/250 | Loss 0.28910 | score 194.00000 | reward 195.000000
Epoch 0245/250 | Loss 0.27983 | score 194.00000 | reward 195.000000
Epoch 0246/250 | Loss 0.23260 | score 194.00000 | reward 195.000000
Epoch 0247/250 | Loss 0.21524 | score 194.00000 | reward 195.000000
Epoch 0248/250 | Loss 0.20771 | score 194.00000 | reward 195.000000
Epoch 0249/250 | Loss 0.17785 | score 194.00000 | reward 195.000000
Epoch 0250/250 | Loss 0.16330 | score 194.00000 | reward 195.000000
--- Tempo total: 381 seconds ---
```

Reinforcement Learning

Objetivo: Build a neural network to play the game of catch. Each game starts with a ball being dropped from a random position from the top of the screen. The objective is to move a paddle at the bottom of the screen using the left and right arrow keys. Ex: Four consecutive screenshots of our catch game:



State: Current game screen image

Reinforcement Learning

Markov decision process (MDP):

The probability of state s_{t+1} depends only on current s_t and action a_t

The environment : **Game**

The agent:

1. **Actions (** a_t **)**: moving paddle left or right.
2. **Reward (** r_t **):** (+ or -)

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, \ldots, s_{n-1}, a_{n-1}, r_n, s_n$$

Maximizing future rewards

The total discounted future reward at a time step t as the sum of the current reward and the total discounted future reward at the next time step:

$$R_t = r_t + \gamma R_{t+1}$$

Q-learning

```
initialize Q-table Q
observe initial state s
repeat
 select and carry out action a
 observe reward r and move to new state s'
 Q(s,a) = Q(s,a) + α(r + γ max_a'(Q(s',a') - Q(s,a)))
 s = s'
until game over
```

Deep Q-Learning

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Deep Q-network as a Q-function

Q-function => Neural Network ……………. (what kind?)

State: 4 consecutive black and white screen image of size (80 , 80).

Número of possible states is $2^{80x80x4}$.

Fortunately, many of these states represent impossible or highly improbable pixel combinations.

CNN have local connectivity, it avoids these impossible or improbable pixel.

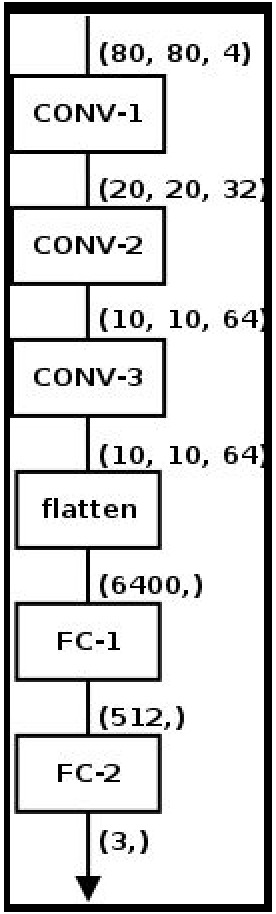
CNN Model

DeepMind paper, also use three layers of convolution followed by two fully connected layers.

There are no polling layers (Make less sensitive to the location of specific objects in the image).

Input = (80 , 80 , 4)

Output_shape = 3 (move left, stay, move right)



Keras deep Q-network for catch

Game described by Eder Santana in his blog post.

We need install Pygame (http://www.pygame.org/download.shtml or http://www.pygame.org/wiki/GettingStarted)

Abrir o arquivo game.py

Keras deep Q-network for catch

```
Epoch 1789/2100 | Loss 0.09356 | Win Count: 638
Epoch 1790/2100 | Loss 0.10564 | Win Count: 638
Epoch 1791/2100 | Loss 0.20233 | Win Count: 639
Epoch 1792/2100 | Loss 0.11323 | Win Count: 640
Epoch 1793/2100 | Loss 0.11274 | Win Count: 641
Epoch 1794/2100 | Loss 0.17127 | Win Count: 641
Epoch 1795/2100 | Loss 0.06934 | Win Count: 641
Epoch 1796/2100 | Loss 0.09507 | Win Count: 642
Epoch 1797/2100 | Loss 0.11883 | Win Count: 643
Epoch 1798/2100 | Loss 0.15115 | Win Count: 644
Epoch 1799/2100 | Loss 0.14886 | Win Count: 645
Epoch 1800/2100 | Loss 0.14325 | Win Count: 646
Epoch 1801/2100 | Loss 0.08702 | Win Count: 647
Epoch 1802/2100 | Loss 0.15176 | Win Count: 648
Epoch 1803/2100 | Loss 0.12056 | Win Count: 649
Epoch 1804/2100 | Loss 0.18569 | Win Count: 650
Epoch 1805/2100 | Loss 0.11933 | Win Count: 651
Epoch 1806/2100 | Loss 0.18903 | Win Count: 651
Epoch 1807/2100 | Loss 0.15308 | Win Count: 651
Epoch 1808/2100 | Loss 0.10187 | Win Count: 652
```