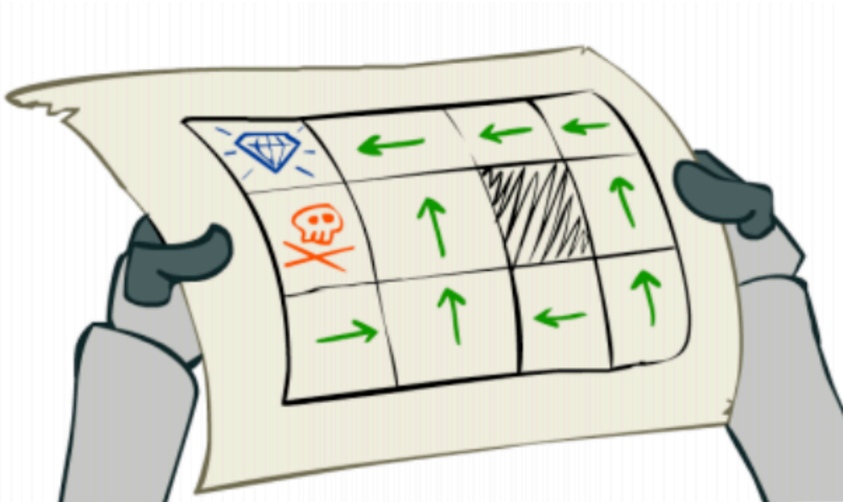
Deep Reinforcement Learning

Matteo Prata — 29 November 2023

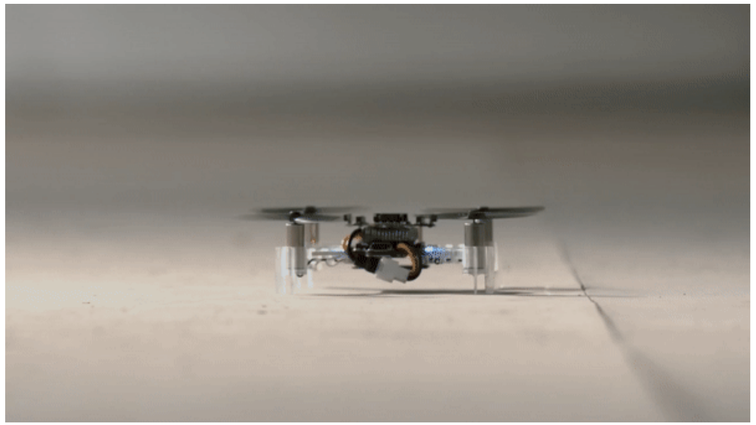


Deep RL Applications

- Controlling robots
- Protocols optimization
- Financial stock markets trading
- Playing games

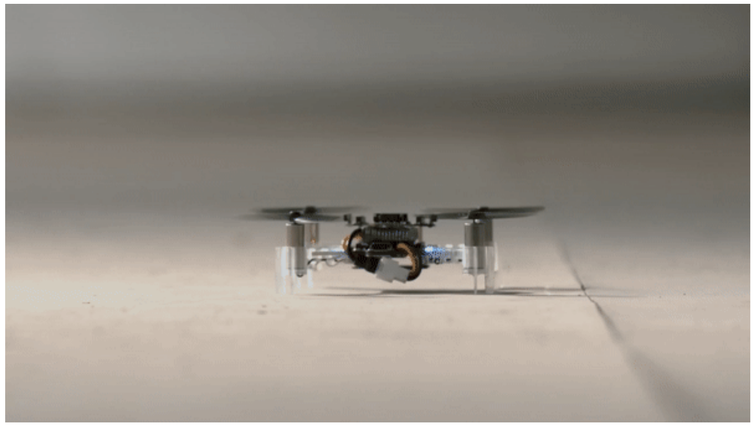
Deep RL Applications

- Controlling robots
- Protocols optimization
- Financial stock markets trading
- Playing games



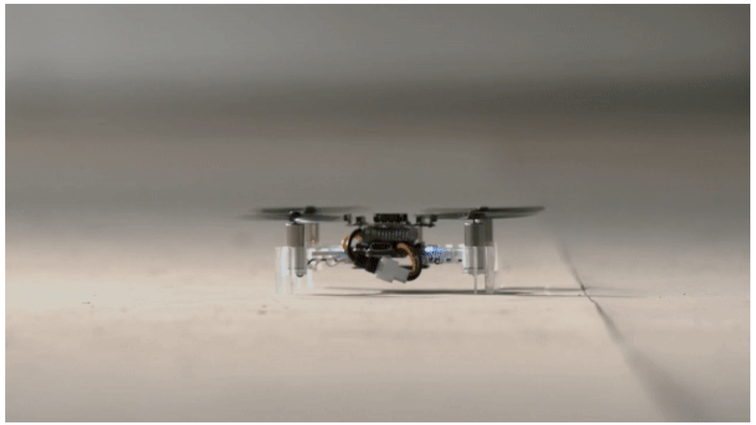
Deep RL Applications

- Controlling robots
- Protocols optimization
- Financial stock markets trading
- Playing games



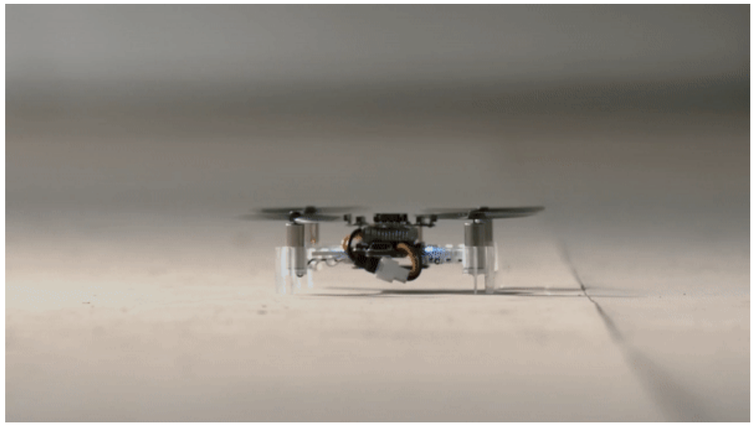
Deep RL Applications

- Controlling robots
- Protocols optimization
- Financial stock markets trading
- Playing games



Deep RL Applications

- Controlling robots
- Protocols optimization
- Financial stock markets trading
- Playing games



Machine Learning

- RL is a branch of **Machine Learning** (siblings: Supervised, Unsupervised Learning)

supervised learning
(regression)

	x (size m^2)	$\hat{y}$ (house price $)
$x^{(0)}$	116	250k
$x^{(1)}$	200	350k
$x^{(2)}$	232	400k
$x^{(n)}$	...	...

house price $

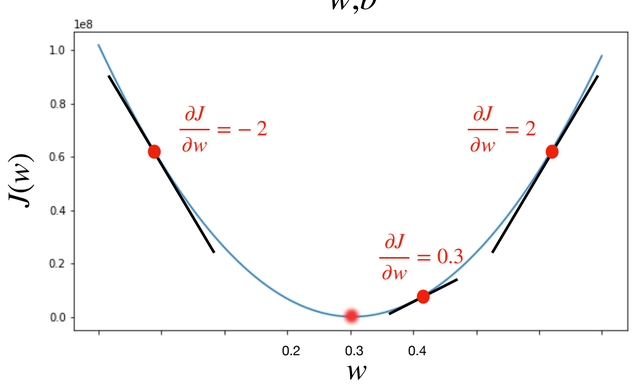
size m^2

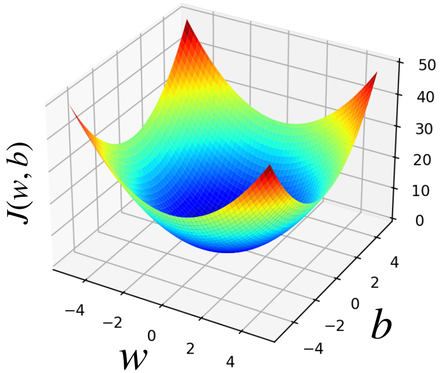
slope intercept

$$f_{w,b}(x^{(i)}) = wx^{(i)} + b \quad \leftarrow \text{model}$$

$$J(w,b) = \frac{1}{n}\sum_{i=0}^{n}\left(f_{w,b}(x^{(i)}) - \hat{y}^{(i)}\right)^2 \quad \leftarrow \text{cost}$$

$$\min_{w,b} J(w,b)$$





- you are given x data points (e.g., size) and $\hat{y}$ outcome (e.g., house price)
- **learn function** to fit the data and generalize over new data points
- minimize error between the model output and the data (e.g., MSE loss) using **gradient descent**

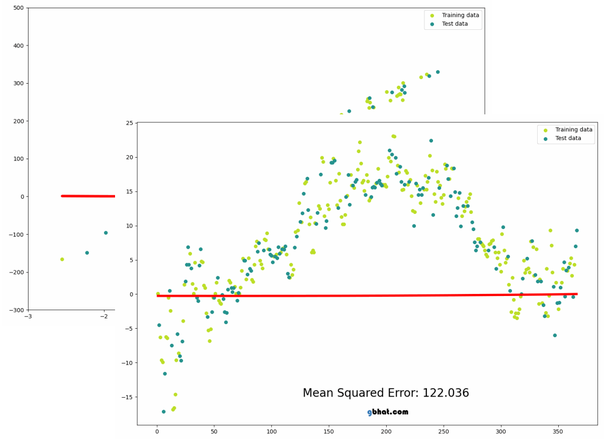
iteratively update weights proportionally to the gradients of the cost function

$$w \leftarrow w - \alpha\frac{\partial}{\partial w}J(w,b) \qquad b \leftarrow b - \alpha\frac{\partial}{\partial b}J(w,b)$$

Machine Learning

- RL is a branch of **Machine Learning** (siblings: Supervised, Unsupervised)

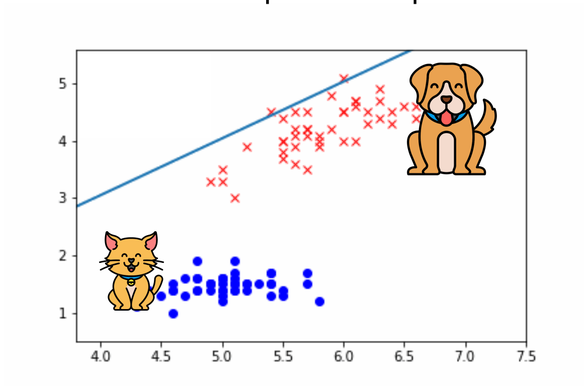
supervised learning
(regression, linear/polynomial)



- expands to multiple features (e.g. number of bedrooms, distance to city center, surface)
- can capture complex trend like polynomials

supervised learning
(classification)

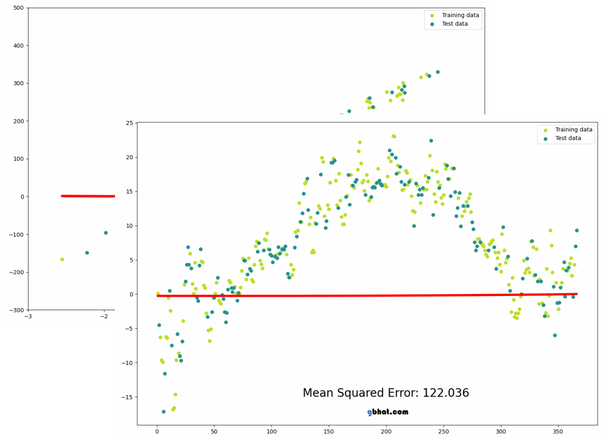
x_0 (ears length)	x_1 (fur length)	$\hat{y}$ (animal type)
20 cm	10 cm	
25 cm	5 cm	
5 cm	2 cm	



Machine Learning

- RL is a branch of **Machine Learning** (siblings: Supervised, Unsupervised)

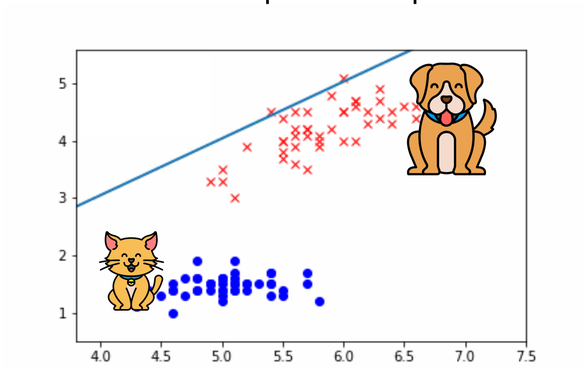
supervised learning
(regression, linear/polynomial)



- expands to multiple features (e.g. number of bedrooms, distance to city center, surface)
- can capture complex trend like polynomials

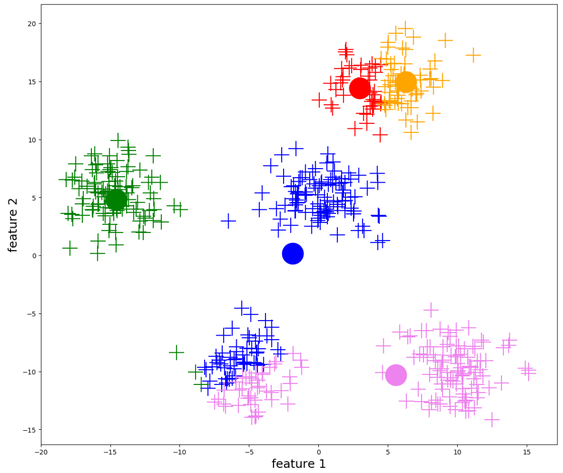
supervised learning
(classification)

x_0 (ears length)	x_1 (fur length)	$\hat{y}$ (animal type)
20 cm	10 cm	
25 cm	5 cm	
5 cm	2 cm	



- classify animal (cat, dog) based on features like ear length, fur length
- learn a **decision boundary**

unsupervised learning
(clustering)

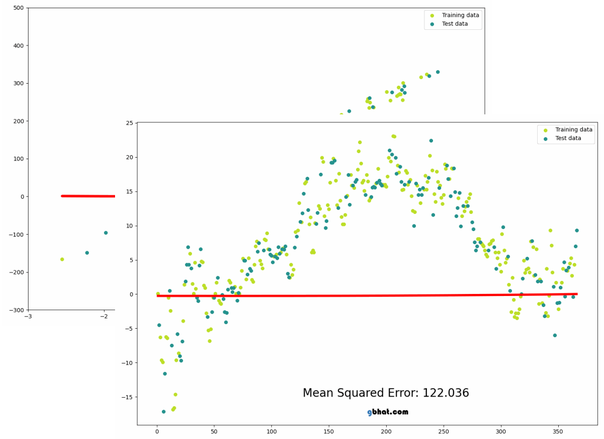


- unknown mapping
- learn clusters by data point similarity
- anomaly detection, dimensionality reduction

Machine Learning

- RL is a branch of **Machine Learning** (siblings: Supervised, Unsupervised)

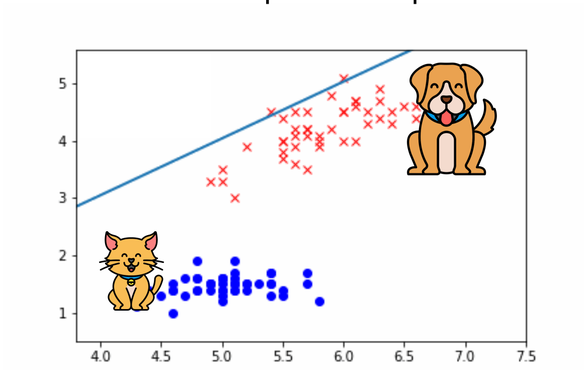
supervised learning
(regression, linear/polynomial)



- expands to multiple features (e.g. number of bedrooms, distance to city center, surface)
- can capture complex trend like polynomials

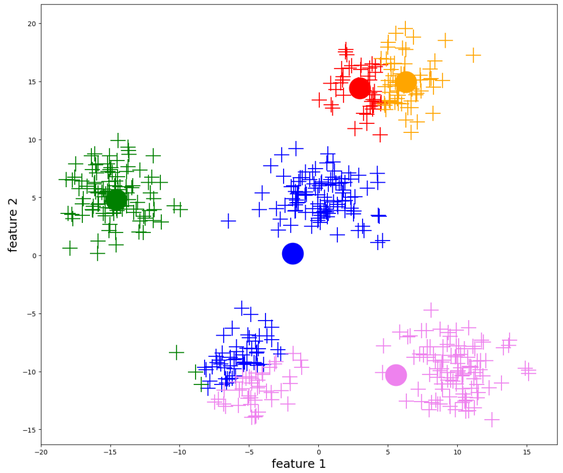
supervised learning
(classification)

x_0 (ears length)	x_1 (fur length)	$\hat{y}$ (animal type)
20 cm	10 cm	
25 cm	5 cm	
5 cm	2 cm	



- classify animal (cat, dog) based on features like ear length, fur length
- learn a **decision boundary**

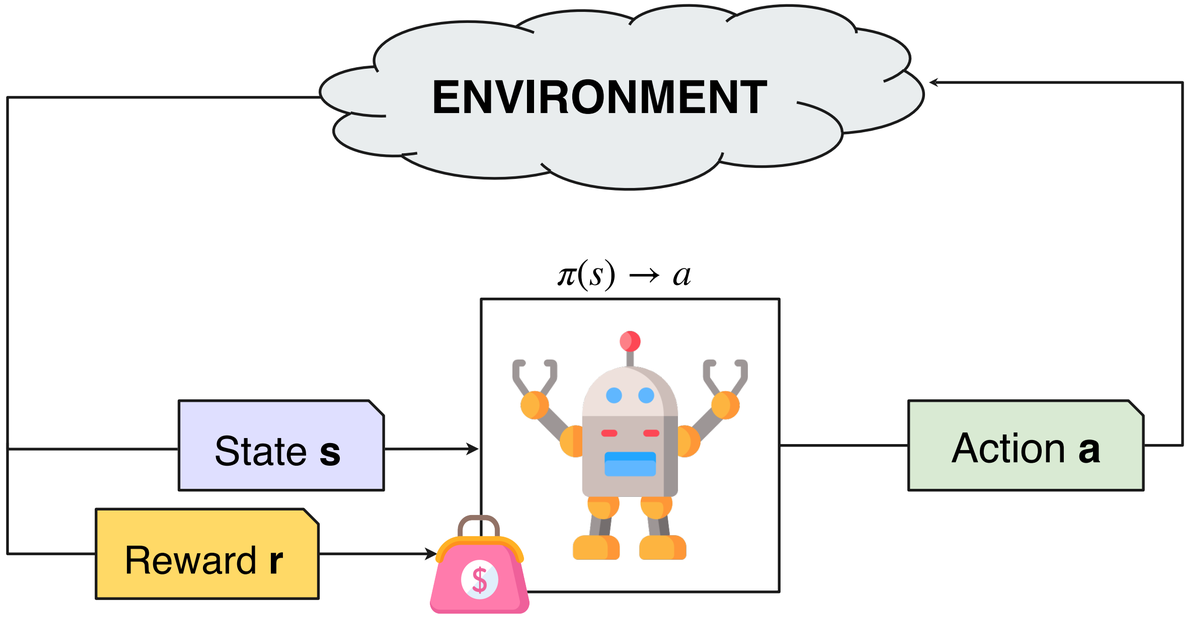
unsupervised learning
(clustering)



- unknown mapping
- learn clusters by data point similarity
- anomaly detection, dimensionality reduction

RL Loop

- An **agent** wants to learn a function, a **policy** (how to act) in an **environment** to maximize the **cumulative reward** obtained from the environment

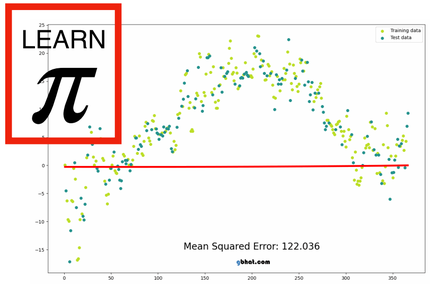


Sense
Think
Act

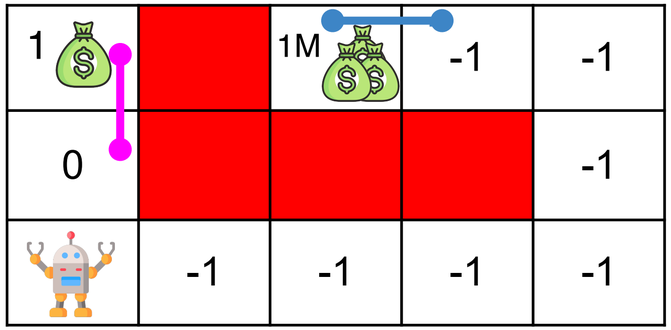
(s_t, a_t, s_{t+1}, r_t)

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

Experience tuples



- learn a policy $\pi(s)$ to maximize cumulative reward during agent lifetime = **episode**
- DeepRL learns π using **experience tuples** and a **loss** as in supervised learning



Cumulative reward

$$\sum_{i=0}^{\infty} r_i$$

P1: 0, 1, 0, 1, 0, 1, … → ∞

P2: -1, -1, -1, -1, -1, -1, -1, -1, 1M, -1, 1M … → ∞

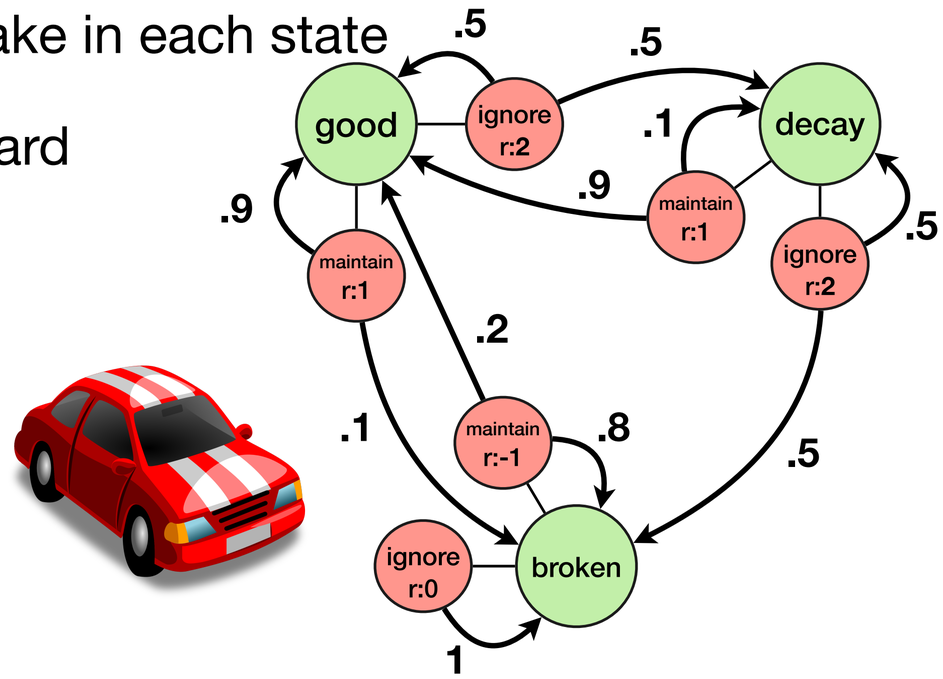
Discounted reward

$$\sum_{i=0}^{\infty} \gamma^i \, r_i = (-1)\,\gamma + (-1)\,\gamma^2 + \cdots + 1M\,\gamma^8 + (-1)\,\gamma^9 + 1M\,\gamma^{10} + \cdots$$

RL Model — MDPs

- A Markov Decision Process is a 4 tuple **MDP** (S, T, A, R) 🌟
  - S : set of **states**
  - A : set of **actions**
  - $T : \{S \times S \times A\} \rightarrow [0,1]$: **transition probability** function $T(s', s, a) = P(s' \mid s, a)$
  - $R : \{S \times A\} \rightarrow \mathbb{R}$: **reward function**
- The **policy** is a function $\pi : S \rightarrow A$ specifying what action to take in each state
- Learn π^* that maximises <u>expected</u> <u>cumulative</u> <u>discounted</u> reward

known only in <u>model based RL</u>
value/policy iteration algorithms



Q-learning

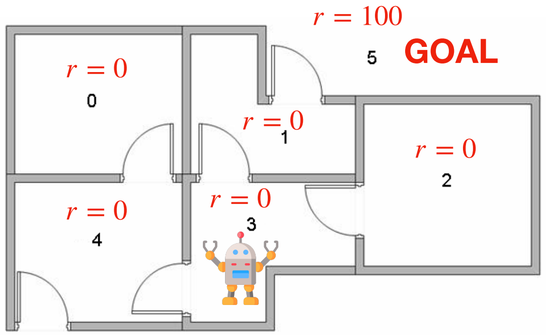
- **set** $Q(s,a) = 0 \; \forall s \in S, a \in A$
- **while** true:
  - from state s (random at the start of the epoch) let the process evolve, until s is end, as:
    - ✳ choose action *initially random* with decaying probability, then $a = \arg\max_{a' \in A} Q(s,a')$
    - ✳ take action and observe s', r
    - ✳ $Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a' \in A} Q(s',a'))$
    - ✳ **set** $s = s'$
  - **exit** when converged

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

actions

states

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 0 & 80 & 0 \\ 1 & 0 & 0 & 0 & 64 & 0 & 100 \\ 2 & 0 & 0 & 0 & 64 & 0 & 0 \\ 3 & 0 & 80 & 51 & 0 & 80 & 0 \\ 4 & 64 & 0 & 0 & 64 & 0 & 100 \\ 5 & 0 & 80 & 0 & 0 & 80 & 100 \end{array} \begin{array}{l} \\ \\ \\ s' \\ \\ s \\ \\ \end{array}$$



$$Q(3,1) = 0.9 \cdot 80 + 0.1(0 + 0.9 \cdot 100) = 81$$

Q-learning Limitations

state: camera/radar signals
actions: steering angle
rewards: distance borders

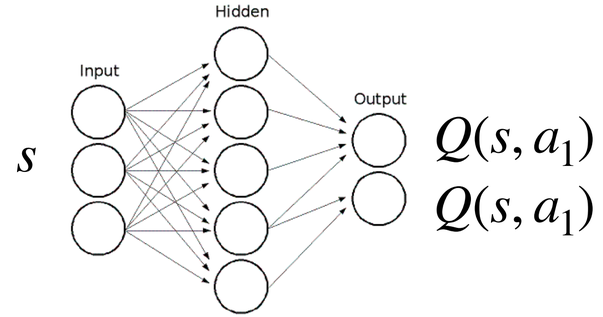
state: raw pixels
actions: move right / left
rewards: score

state: price and volume of each of the orders
actions: buy and sell
rewards: expected profit

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

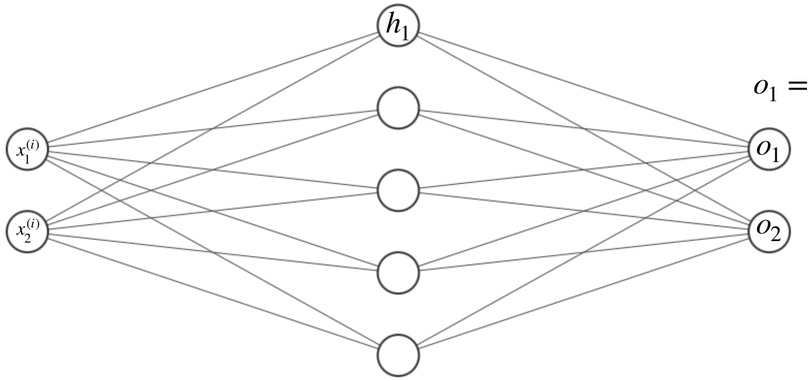
Q-TABLES do not scale!

Deep Q-Learning you train a NN rather than a Q-Table



Neural Networks

$$h_1 = \sigma(x_1^{(i)} w_{1,1}^{(0)} + x_2^{(i)} w_{2,1}^{(0)} + b^{(0)})$$



$$o_1 = \sigma(h_1 w_{1,1}^{(1)} + h_2 w_{2,1}^{(1)} + \cdots)$$

probability of 

probability of 

input layer | hidden layer | output layer

x_0 (ears length)	x_1 (fur length)	$\hat{y}$ (animal type)
20 cm	10 cm	
25 cm	5 cm	
5 cm	2 cm	

before training $f_{\vec{w}}\begin{pmatrix}20\\10\end{pmatrix} \rightarrow \begin{matrix}0.5\\0.5\end{matrix}$

after training $f_{\vec{w}}\begin{pmatrix}20\\10\end{pmatrix} \rightarrow \begin{matrix}1\\0\end{matrix}$

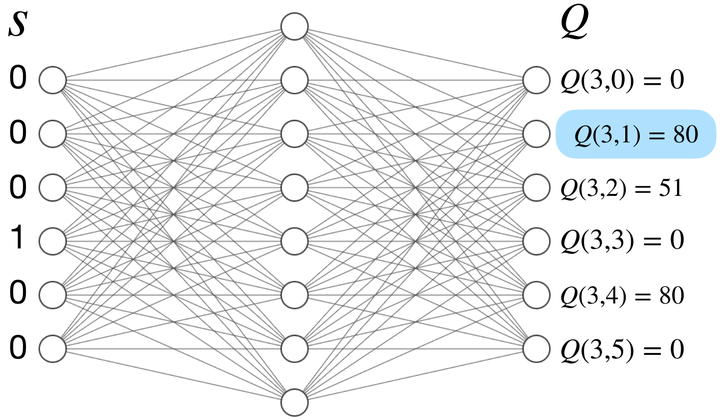
Use **gradient descent** to update the weights taking into account the dependencies of the neurons, in the graph.

Use the **chain rule** as the output neurons are function of functions.

$$J(\vec{w}) = \frac{1}{n}\sum_{i=0}^{n}\left(f_{\vec{w}}(x^{(i)}) - \hat{y}^{(i)}\right)^2$$

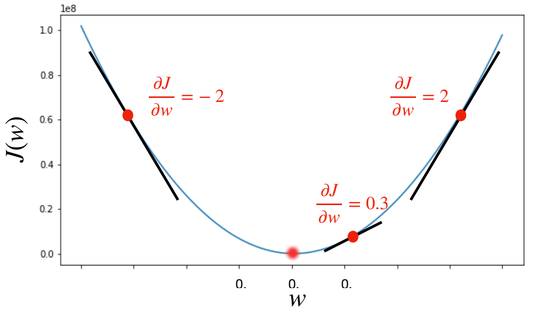
$x^{(i)}$: $\begin{matrix}20\\10\end{matrix}$ $\quad$ $f_{\vec{w}}(x^{(i)})$: $\begin{matrix}0.5\\0.5\end{matrix}$ $\quad$ $\hat{y}^{(i)}$: $\begin{matrix}1\\0\end{matrix}$ 

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array} s$$

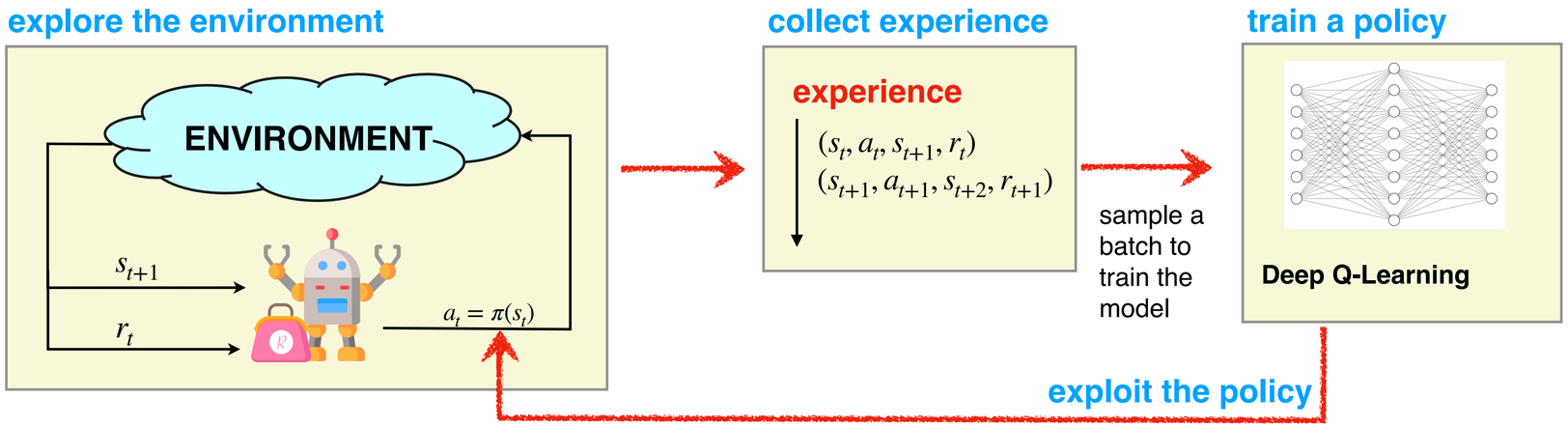


Q-function approximation

$$\min_{\vec{w}} J(\vec{w})$$



Deep RL Training



step 1 — Act

- agent **acts** in the environments following the policy $f_{\overrightarrow{w}}(s)$ or going random
- **store** experience tuples (s, a, s', r)

step 2 — Train

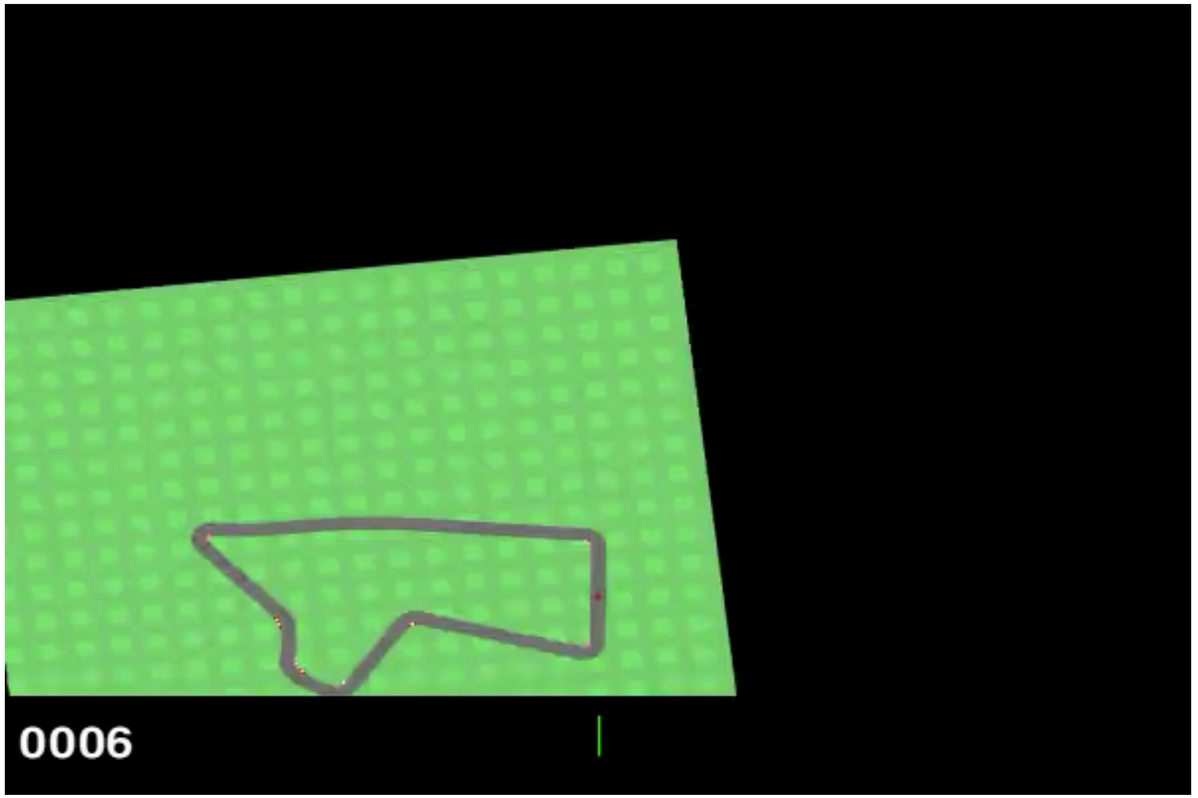
- select a **batch** of tuples, and iterate over each one at a time (s, a, s', r)
- query the q-values for the state s, $f_{\overrightarrow{w}}(s)$ then for $f_{\overrightarrow{w}}(s')$
- predicted q-values $f_{\overrightarrow{w}}(s)$ should converge to $r + \gamma \max_{a'} f_{\overrightarrow{w}}(s')[a']$

$$L(\overrightarrow{w}) = \Big(f_{\overrightarrow{w}}(s) - \Big(r + \gamma \max_{a'} f_{\overrightarrow{w}}(s')[a'] \Big) \Big)^2$$

predicted target

HANDS ON!

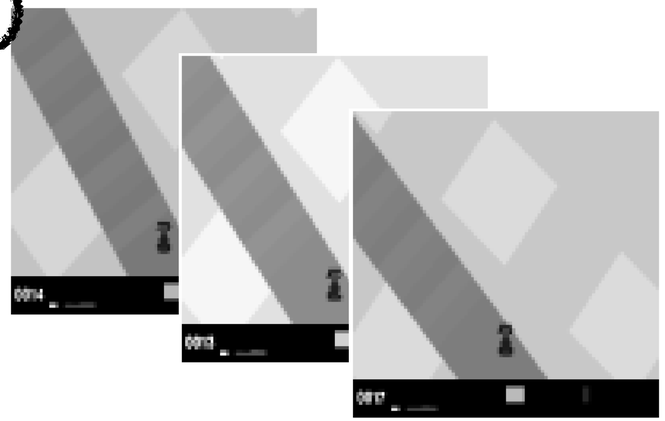
https://github.com/matteoprata/DeepRL-Class



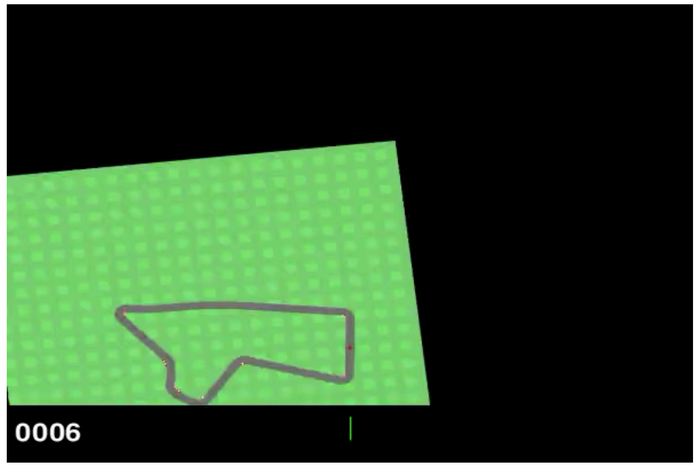
🌟 🍴

Car Racing App

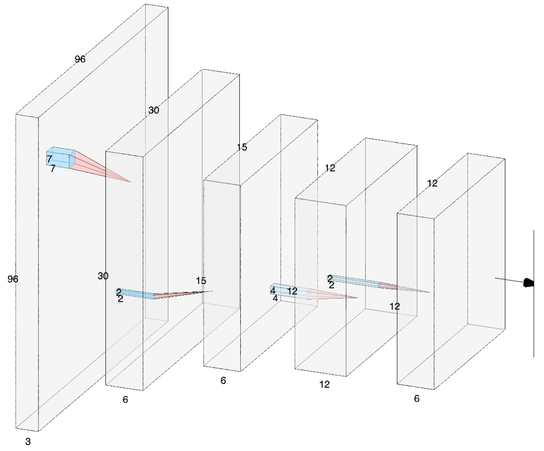
1



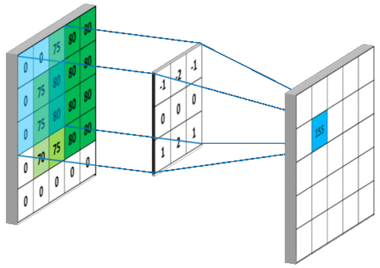
- **State** s is **gray scale** images (no RGB). 3 channels represent sequence of frames of the env *3x96x96*



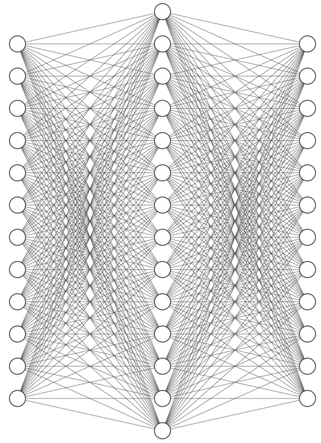
2



- **Extract features** from the image with a Convolutional NN (CNN) get a vector of 432 signals, like a *processed state*
- Help to further reduce dimensionality 3x96x96 ≈ 27,600 *vs* 432
- Similar states have similar extracted features, signals



3



12 actions
(gas, steering angle, break)

reward
inversely proportional to time off-track

- A NN predicts 12 *q-values* for each of the actions

Optimal policy $\pi^*(s) = \arg\max_{a \in A} f_{\overrightarrow{w}}(s)$