Deep reinforcement learning

Oswaldo Andrés Ordóñez Bolaños

Tabla de contenido

“La búsqueda de la inteligencia artificial siempre se ha entremezclado con otra lucha, más filosófica, más romántica, menos tangible.

La comprensión de la inteligencia humana”

Y como aprendemos.

Aprendizaje automático

Aprendizaje supervisado: El maestro son los **ejemplos**.

- Hay algunos ejemplos etiquetados, se aprende patrones basados en esos ejemplos.

Aprendizaje no supervisado: **No** hay maestro.

- Aprender desde ejemplos sin etiqueta.

Aprendizaje reforzado: El maestro es la **experiencia**.
“Aprendiendo con un critico”

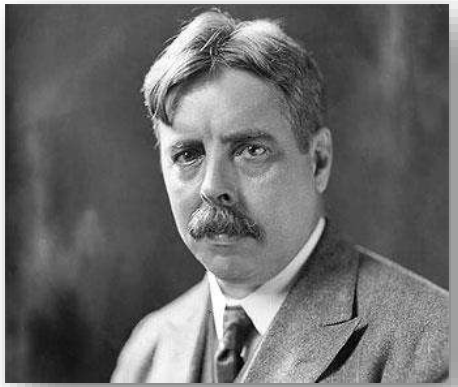
- Hay un mundo, se aprende patrones explorándolo.

Intelligent Machinery, 1948

Turing

sistema de placer-dolor.

Mundo de prueba y error, 1989



Thorndike realizo el experimento del gato en la jaula y propuso la ley del efecto que establece que es probable que se repita cualquier comportamiento seguido de consecuencias agradables.

¿ Cuál es la naturaleza del hombre ?

El hombre es Hedonista

The Hedonistic Neuron — A Theory of Memory, Learning, and Intelligence, 1982.

*Se estaban perdiendo aspectos esenciales de la **conducta adaptativa** a medida que los investigadores del aprendizaje se centraron en el **aprendizaje supervisado**, ignorando aspectos hedónicos del comportamiento,*
*controlar el entorno hacia **fines deseados** y lejos de **fines no deseados**.*

Harry Klopf, suponer que los humanos, después de haber asegurado la homeostasis (la búsqueda de un bien), buscan maximizar el placer, no estabilizarlo.

Playing Atari 2600 with deep reinforcement learning

Deep mind, 2013

El modelo es una **red neuronal convolucional**, entrenada con una variante de **Q-learning**, cuya entrada son píxeles sin procesar y cuya salida es una función de valor que estima recompensas futuras.

Learning Dexterity

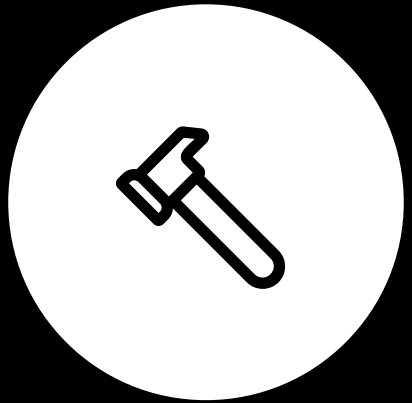
Open AI, 2018

Mano robótica parecida a un humano para manipular objetos físicos con una destreza sin precedentes (24 grados de libertad) entrenada con Deep reinforcement learning.

Dota 2 with Large Scale Deep Reinforcement Learning

Open AI, 2019

OpenAI Five aprovecha las técnicas de **aprendizaje por refuerzo** existentes, para aprender de lotes de aproximadamente 2 millones de fotogramas cada 2 segundos.



Herramientas

Librerias: TF-Agents, Stable-Baseline 2.0, TensorFlow

MineRL, Unity Ml.agents, OpenAI retro, GYM

La gran pintura

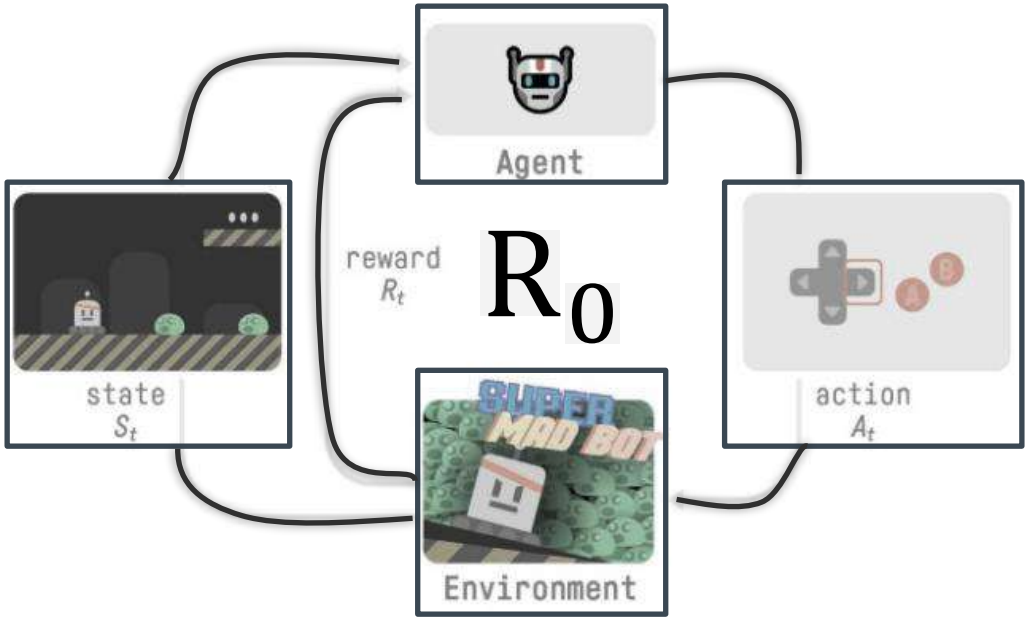
El hermano pequeño y un videojuego

El aprendizaje por refuerzo es un marco de trabajo para resolver **tareas de control** (también llamadas problemas de decisión) mediante la construcción de **agentes** que aprenden del entorno al interactuar con él a través de **prueba y error** y recibir recompensas **(positivas o negativas)** como retroalimentación única.

Markov Decision Processes (MDPs)

Aprendizaje reforzado

S_1 S_0 R_0 A_0



Nuestro agente **solo necesita el estado actual** para tomar una decisión sobre qué acción tomar y no la historia de todos los estados y acciones que tomó antes.

Marco de trabajo
algunos conceptos

Estado (s)

Descripción completa del estado del mundo (no hay información oculta). Entorno completamente observado.

Observación (O)

Descripción **parcial** del estado. Entorno parcialmente observado.

Espacio discreto

El número de acciones posibles son **finitas**.

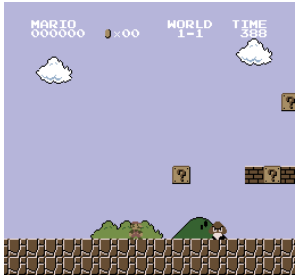
Action Space

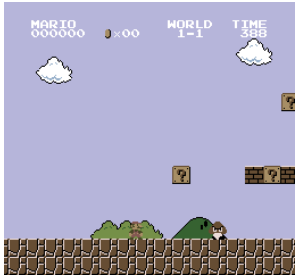
El espacio Acción es el conjunto de todas las acciones posibles en un entorno.

Espacio Continuo

el número de acciones posibles son **infinitas**.

Marco de trabajo
algunos conceptos





Recompensas y descuentos

La recompensa es fundamental para el RL porque es la única **información para el agente**. Gracias a ella, nuestro agente sabe si la acción realizada fue buena o no.

$$G_T = R(\tau) = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + \cdots$$

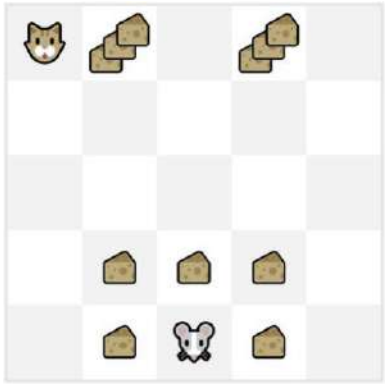
Trayectoria: *Secuencia de estados y acciones.*

Return: *Recompensas acumuladas.* ***Rendimiento esperado***

El objetivo del agente es **maximizar las recompensas acumuladas**

No solo las recompensas inmediatas

Recompensas y descuentos



$$G_T = R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots$$

Recompensas acumuladas esperadas con descuento

Expected discounted return of rewards.

Gamma: *taza de descuento.*

Recompensas y descuentos

Las recompensas que llegan antes son más probables (Mayor gamma, menor descuento) porque que son más predecibles que la recompensa futura a largo plazo.

$$G_T = R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots$$

Recompensas acumuladas esperadas con descuento

Expected discounted return of rewards.

Gamma: *taza de descuento.*

El objetivo del agente es **maximizar las recompensas esperadas acumuladas con descuento**

Tipos de tareas

- Tareas episódicas

En la tarea hay un punto de **inicio** y un punto **final** (un estado terminal). Esto crea un episodio: una lista de estados, acciones, recompensas y nuevos estados.

- Tareas continuas

Tareas que continúan para **siempre** (sin estado terminal). El agente tiene que aprender a elegir las mejores acciones y simultáneamente interactuar con el entorno.

Compensación de exploración y explotación

- Exploración

La exploración consiste en explorar el medio ambiente probando acciones aleatorias para encontrar más información sobre el medio ambiente.

- Explotación

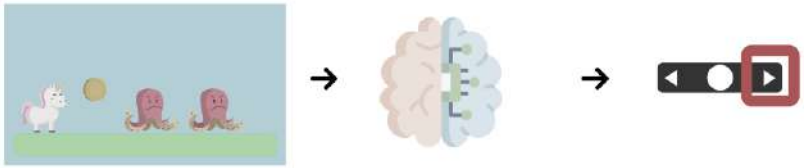
Exploitation is exploiting known information to maximize the reward.

Compensación de exploración y explotación

¿Qué tan bueno es un estado o una acción?

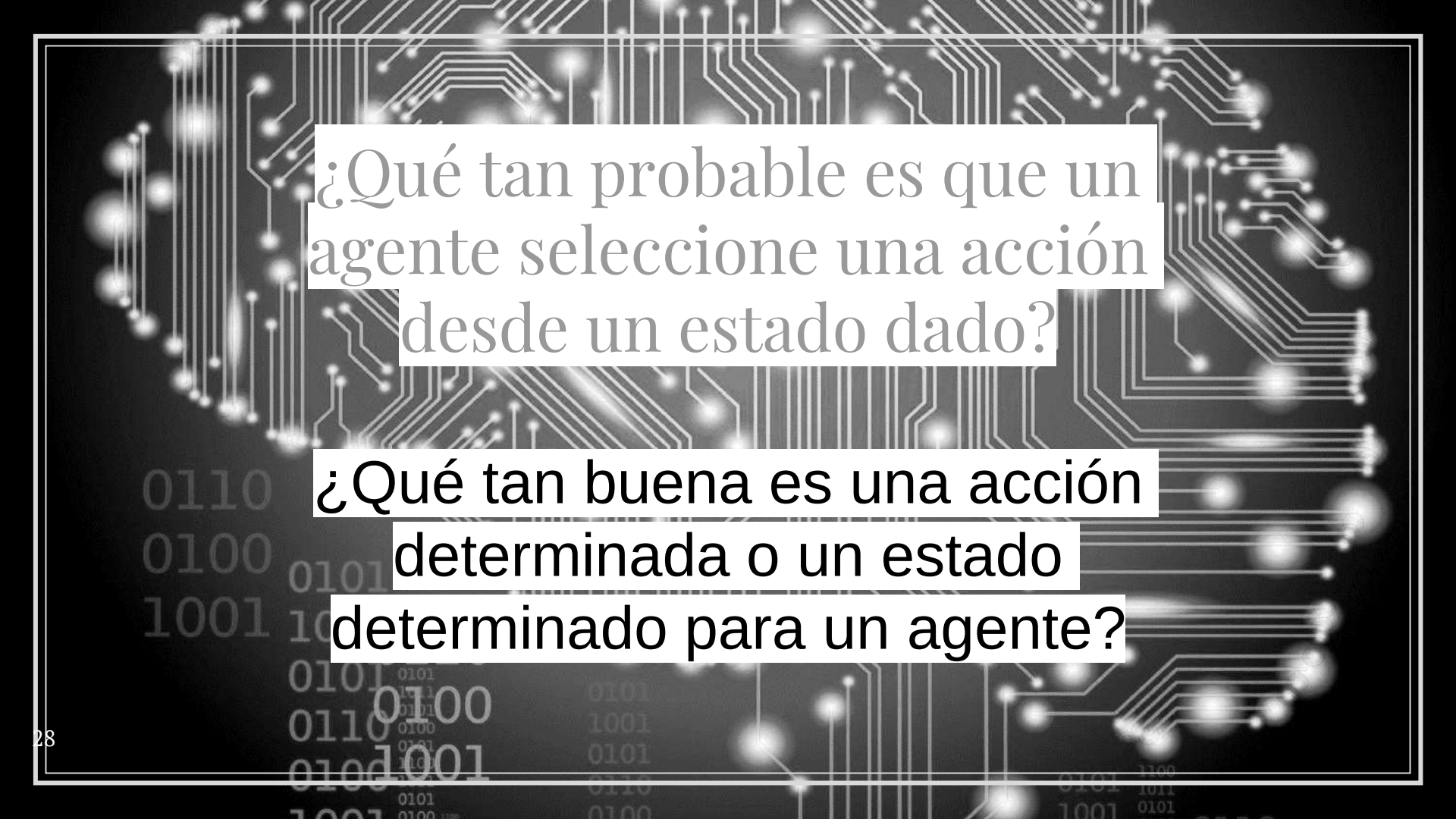
La política (policy) π: el cerebro del agente

Función que nos dice qué **acción** tomar dado el **estado** en el que nos encontramos. Entonces define el **comportamiento del agente** en un momento dado.



State → π(State) → Action

$$\pi(a|s)$$


¿Qué tan probable es que un agente seleccione una acción desde un estado dado?
¿Qué tan buena es una acción determinada o un estado determinado para un agente?

Métodos basados en políticas

$$a = \pi(s)$$

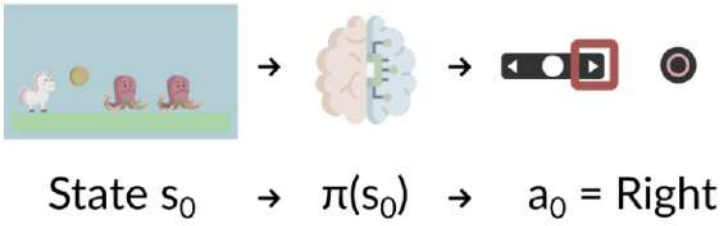
Se aprende **directamente** una función de política **entrenada**

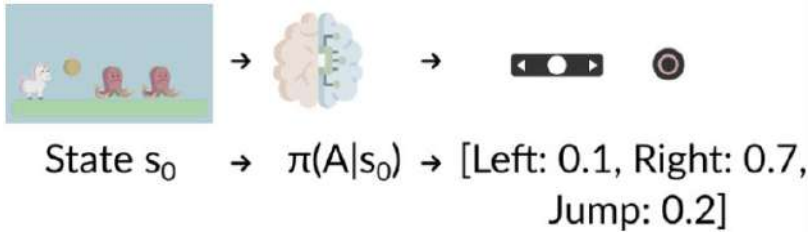
Determinista

Una política en un estado dado siempre devolverá la misma acción.

Estocástico

Generar una distribución de probabilidad sobre las acciones.





Métodos basados en valores

En los métodos basados en valor, en lugar de entrenar una función de política, entrenamos una **función de valor (en términos de el rendimiento esperado)** que tan bueno es para el agente estar en el estado dado, y dependiendo de ello, la política actúa.

Función de valor de estado

Qué tan bueno es un **estado** dado para un agente que sigue la política π .

$$V_\pi(s) = E_\pi[G_t|S_t = s]$$

Agente inicia en el estado s

Rendimiento esperado

Función de valor de acción

Qué tan bueno es para el agente tomar una **acción** determinada desde un **estado** dado mientras sigue la política π.

$$Q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a]$$

Agente inicia en el estado s y elige acción a

Rendimiento esperado

Optimalidad

En términos de rendimiento, se considera que una **política** es mejor o igual que otra política si el **rendimiento esperado** de la primera es mayor o igual al rendimiento esperado de la otra.

Maximizar la recompensa que el agente espera acumular en su recorrido.

Función de valor de estado <u>óptimo</u>

$$V_*(s) = \max_\pi V_\pi(s)$$

Función de valor de acción <u>óptima</u>

$$Q_*(s,a) = \max_\pi Q_\pi(s,a)$$

Ecuación de Bellman

Bellman en 1959 demostró que la manera de **maximizar** la recompensa acumulada que el agente **espera** recibir al escoger un determinado par estado-acción en un instante **t**, es decir la **política óptima**, viene dada por la elección del mayor valor $Q_*(s,a)$ de entre todas las acciones del siguiente estado en **t+1,** multiplicado por el factor de descuento γ**,** más su recompensa asociada.

En resumen, la idea de la ecuación de **Bellman** es que en lugar de calcular cada valor como la **suma del rendimiento esperado,** se realiza su equivalente que es la suma de la recompensa inmediata + el valor descontado del estado que sigue.

$$Q_*(s,a) = E_\pi[r_{t+1} + \gamma max Q_*(s,a)]$$

Recompensa asociada al siguiente estado

Recompensa máxima acumulada

¿ Como entrenar nuestra función de valor o nuestra función de política ?

Monte Carlo

Se actualiza la función de valor de un **episodio completo** y por lo tanto, se usa la **devolución con descuento** exacta real de este episodio.

Temporal Difference Learning

Se actualiza la función de valor **de un solo paso**, por lo que reemplaza $\boldsymbol{G_T}$ con un retorno estimado llamado **TD target**.

$$V(S_t) \leftarrow V(S_t) + \alpha[\boldsymbol{G_T} - V(S_t)]$$

Anterior estimación del valor de estado t

$$V(S_t) \leftarrow V(S_t) + \alpha[r_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

α = learning rate

TD target

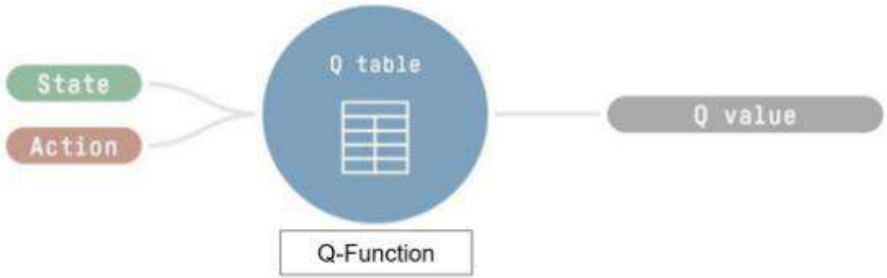
TD error

¿ Como entrenar nuestra función de valor o nuestra función de política ?

MC usa el retorno exacto $\boldsymbol{G_T}$ para actualizar el valor, mientras que **TD** usa la **Ecuación de Bellman** para estimar el valor y luego actualiza el valor estimado con el valor objetivo.

Q - learning

Método basado en **valores fuera de la política** que utiliza un enfoque **TD** para entrenar su función de valor de acción



Dado un **estado y una acción**, nuestra **Q-function** genera un valor de acción de estado (también llamado **Q-value**)

Q - table

Tabla donde cada celda corresponde a un valor de acción-estado. La **Q-table** es la memoria o la hoja de trucos de nuestra **Q-function**.

Q - learning

- Entrena la **función Q**, una función de valor de acción que contiene como memoria interna una **tabla Q** que contiene todos los valores del par estado-acción.
- Dado un estado y una acción, nuestra **función Q** buscará en su **tabla Q** el valor correspondiente.
- Cuando se realiza el entrenamiento, tenemos una **función Q óptima**, por lo que una tabla **Q óptima**.
- Y si tenemos una función **Q óptima**, tenemos una **política óptima**, ya que sabemos para cada estado, cuál es la mejor acción a tomar.

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

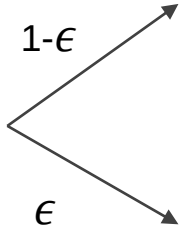
Q – learning
Ratón y el queso

	←	→	↑	↓
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

Se inicializa la Q-table

Q – learning
Ratón y el queso

$\epsilon - greedy$ *politica*



Exploración *(Selecciona la acción greedy)*

Esta tasa que
nuestro a ugar de
explotarlo

```
if random_num > epsilon:
# Elegir acción via explotación
else:
# Elegir acción via exploración
```

Explotación *(Selecciona una acción aleatorio)*

Se elije una ***acción*** *usando la estrategia de* ***Epsilon Greedy***

Q – learning
Ratón y el queso

Dada la acción $\boldsymbol{A_t}$ *ejecutada por la politica,*
observamos $\boldsymbol{R_{t+1}, S_{t+1}}$

↓

Actualizamos $\boldsymbol{Q = (S_t, A_t)}$ ***con TD***

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[r_{t+1} + \gamma maxQ(S_{t+1}, a) - Q(S_t, A_t)]$$

Tasa de aprendizaje

TD target
Actualizar política

Actuación y actualización

- *Fuera de política (Off-policy):*

Utiliza una política **diferente** de actuación y actualización.

- En política (On-policy):

Utiliza la **misma** política de actuación y actualización

Q-Learning con Taxi-v3

El objetivo es capacitar a **un agente de taxi** para que navegue en esta ciudad para transportar a sus pasajeros desde el punto **A** al punto **B**.

```
env = gym.make("Taxi-v3")
env.render()
```

```
+---------+
|R: | : :G|
| : | : : |
| :▒: : : |
| | : | : |
|Y| : |B: |
+---------+
```

- Ambiente **5x5**
- El taxi aparecerá de manera aleatoria
- El pasajero aparecerá de manera aleatoria en (R, B, G, Y) y desea ir a algunos de los lugares.
- **6** acciones (N,S,W,E,Recoger, dejar)

Sistema de recompensas:

- **-1** por cada tiempo de paso
- **+20** por entregar con éxito al pasajero
- **-10** por acciones ilegales (recoger o dejar al pasajero en el exterior del destino).

Q-Learning con Taxi-v3

Crear la Q-table e inicializarla

```
state_space = env.observation_space.n
action_space = env.action_space.n
# Crear nuestra Q-tabla con state_size filas and action_size columnas (500x6)
Q = np.zeros((state_space, action_space))
print(Q)
print(Q.shape)
```

```
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 ...
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
(500, 6)
```

Q-Learning con Taxi-v3

Se define parámetros

```
total_episodes = 25000        # Numero total de episodios
total_test_episodes = 50    # Numero total de episodios test
max_steps = 200               # Maximo de pasos por episodio

learning_rate = 0.01          # Taza de aprendizaje
gamma = 0.99                  # Taza de descuento

# Parametros de exploración / explotación
epsilon = 1.0                 # Taza de exploración epsilon
max_epsilon = 1.0             # Probabilidad de exploración al empezar
min_epsilon = 0.001           # Probabilidad minima de exploración
decay_rate = 0.01             # Tasa de decaimiento exponencial para exploración
```

Q-Learning con Taxi-v3

Se define la estrategia de Epsilon Greedy

```
def epsilon_greedy_policy(Q, state):
  if(random.uniform(0,1) > epsilon):
    # explotación
    action = np.argmax(Q[state])
  else:
    # exploración
    action = env.action_space.sample()

  return action
```

Q-Learning con Taxi-v3

Se implementa el algoritmo de Q-learning y entrenamos el agente

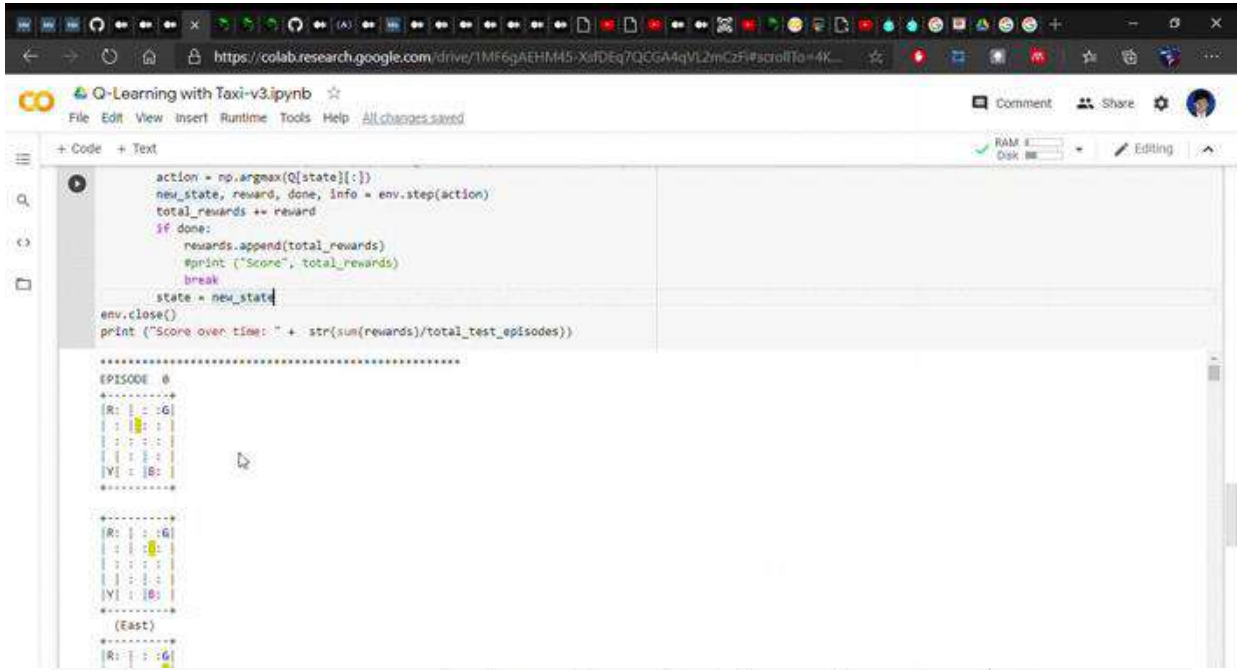
```
for episode in range(total_episodes):
    # Reinicio ambiente
    state = env.reset()
    step = 0
    done = False
    # Reducimos para cada episodio el epsilon (Se necesita más explotación cada vez)
    epsilon = min_epsilon + (max_epsilon - min_epsilon)*np.exp(-decay_rate*episode)
        for step in range(max_steps):
        # Se inicia con la politica epsilon greedy
        action = epsilon_greedy_policy(Q, state)
        # Dependiendo de la interacción de la acción con el ambiente, tenemos una nuevo estado y una recompensa
        new_state, reward, done, info = env.step(action)

        # Actualizamos Q(s,a):= Q(s,a) + lr [R(s,a) + gamma * max Q(s',a') - Q(s,a)]
        Q[state][action] = Q[state][action] + learning_rate * (reward + gamma *
                                    np.max(Q[new_state]) - Q[state][action])
        # Se analiza si ya termino el episodio
        if done == True:
            break

        state = new_state
```

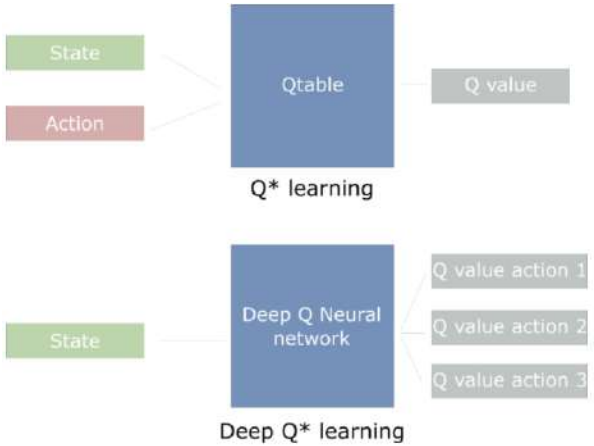
Q-Learning con Taxi-v3

Taxi autónomo



Deep Q - learning

En el caso de que nuestro ambiente sea mucho más **complejo** (millones de estados), la mejor idea es crear una **red neuronal** que aproxime, dando un estado, los diferentes **Q-values** para cada acción.



¿ Como funciona DQL ?
Pasos principales

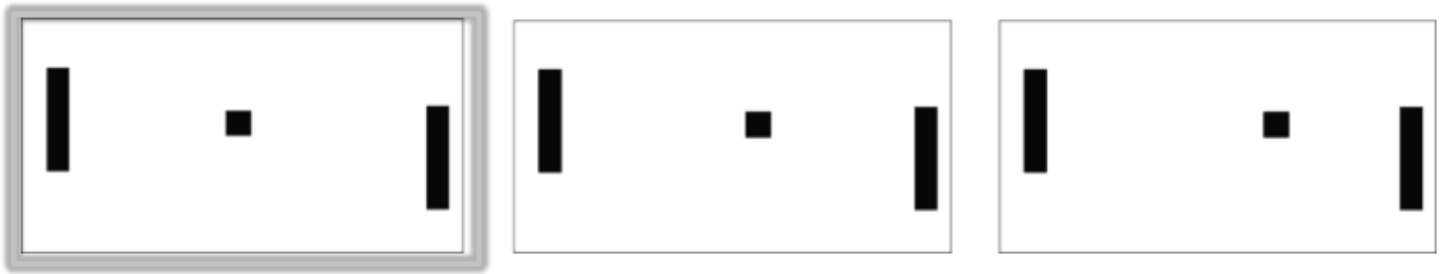
La red neuronal toma una pila de fotogramas como entrada. Estos pasan a través de su red y generan un vector de **Q-values** para cada acción posible en el estado dado. Se toma el mejor **valor de Q** para encontrar nuestra acción más optima.

Pre-procesamiento

¿ Que variables aportan información al problema ?

*Blanco y negro

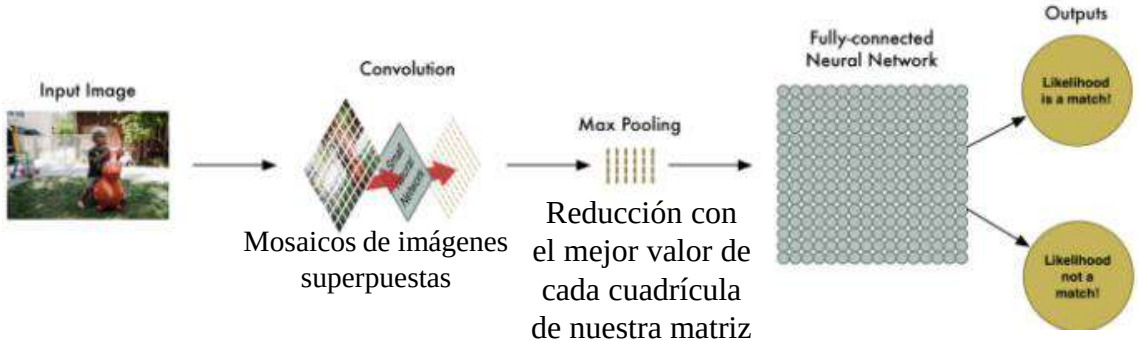
El problema de la limitación temporal



- Redes neuronales **recurrentes** (Redes con bucles que permiten que la información persista) *El problema de las dependencias a largo plazo*
- Redes de memoria a corto plazo (“LSTMs Networks”)
- Marcos apilados

Uso de redes de convolución

Se usa una capa completamente **conectada** con la función de activación **ELU** y una capa de salida (una capa completamente conectada con una función de activación lineal) que produce la estimación del **Q-value** para cada acción.



Mosaicos de imágenes superpuestas

Reducción con el mejor valor de cada cuadrícula de nuestra matriz

Experiencia de repetición

Hacer un uso más eficiente de la experiencia observada

- Evitar olvidar experiencias anteriores

Para evitar olvidar experiencias anteriores significativas, se debe crear un **"búfer de reproducción"**. **Este es** como una carpeta donde cada hoja es una **tupla de experiencia** que se alimenta interactuando con el ambiente. Para luego tomar una hoja al azar para alimentar la **red neuronal.**

- Reducir las correlaciones entre experiencias.

Tomando muestras del **búfer de reproducción** al azar se puede romper la correlación. Esto evita que los valores de acción oscilen o diverjan catastróficamente.

Una clase de aprendizaje supervisado.

Algoritmo Deep Q- learning

En los métodos basados en valor, en lugar de entrenar una función de política, entrenamos una **función de valor (en términos de el rendimiento esperado)** que tan bueno es para el agente estar en el estado dado, y dependiendo de ello, la política actúa.

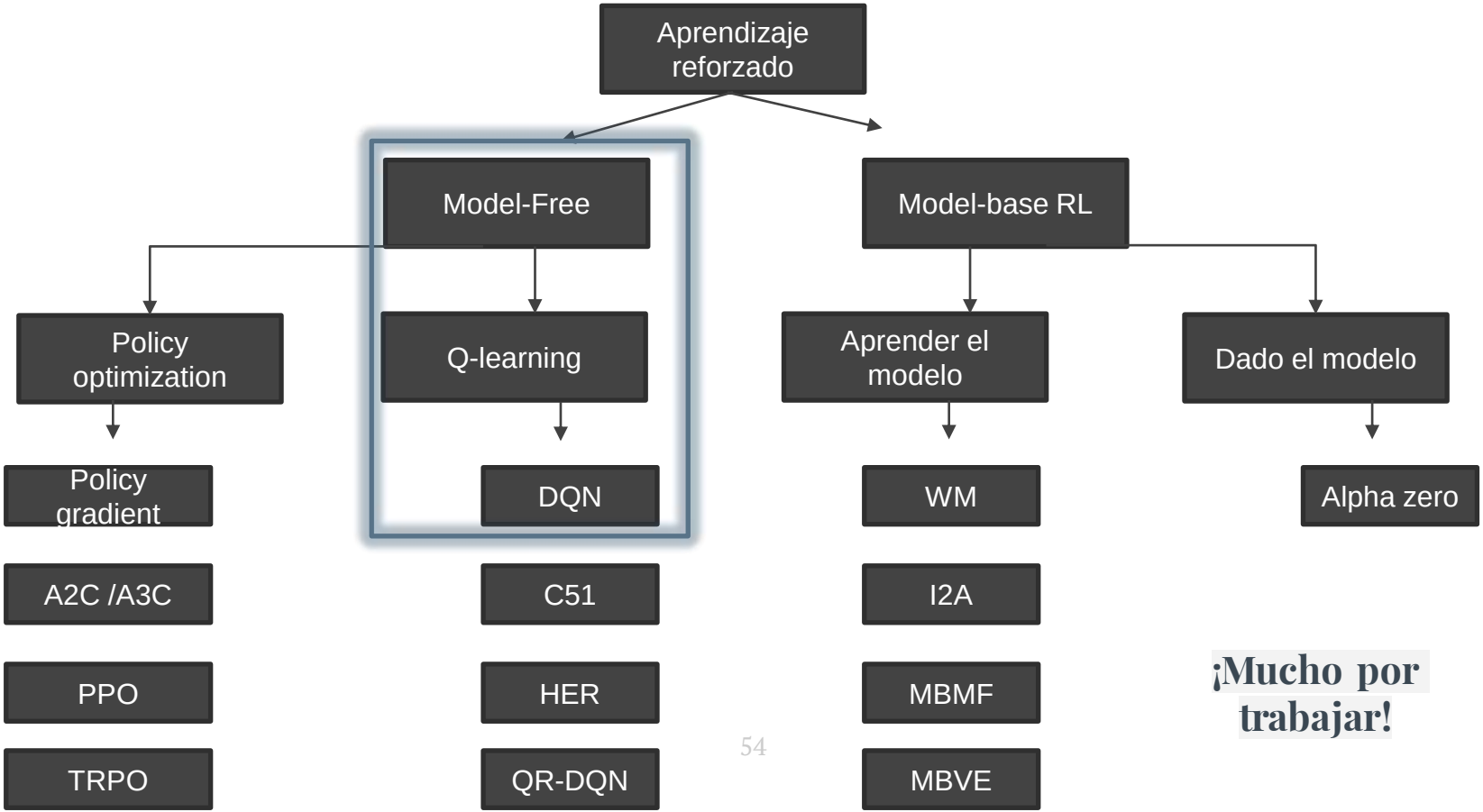
Queremos **actualizar los pesos** de nuestras redes neuronales para reducir el error.

$$\Delta w = \alpha [r_{t+1} + \gamma max \mathrm{Q}(s,a,w) - \mathrm{Q}(s,a,w)] \Delta w \mathrm{Q}(s,a,w)$$

Máximo Q-value para el siguiente estado

Actual predicho Q-value

Gradiente de nuestro actual Q-value predicho



¡Mucho por trabajar!

Bibliografia

- https://towardsdatascience.com/reinforcement-learning-the-naturalist-the-hedonist-and-the-disciplined-fc335ac9289c
- https://www.freecodecamp.org/news/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8/
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

Cursos

- [x] Deep Reinforcement Learning Course – Thomas simonini
- [x] Reinforcement Learning - Goal Oriented Intelligence – deeplizard

¡Gracias!

¿ Preguntas ?

Oswaldo Andrés Ordóñez Bolaños
Oswaldordonez@unicauca.edu.co