# Aprendizaje por refuerzo

## Sesión 3 – Algoritmos base: Deep Q-network

## Curso 21/22

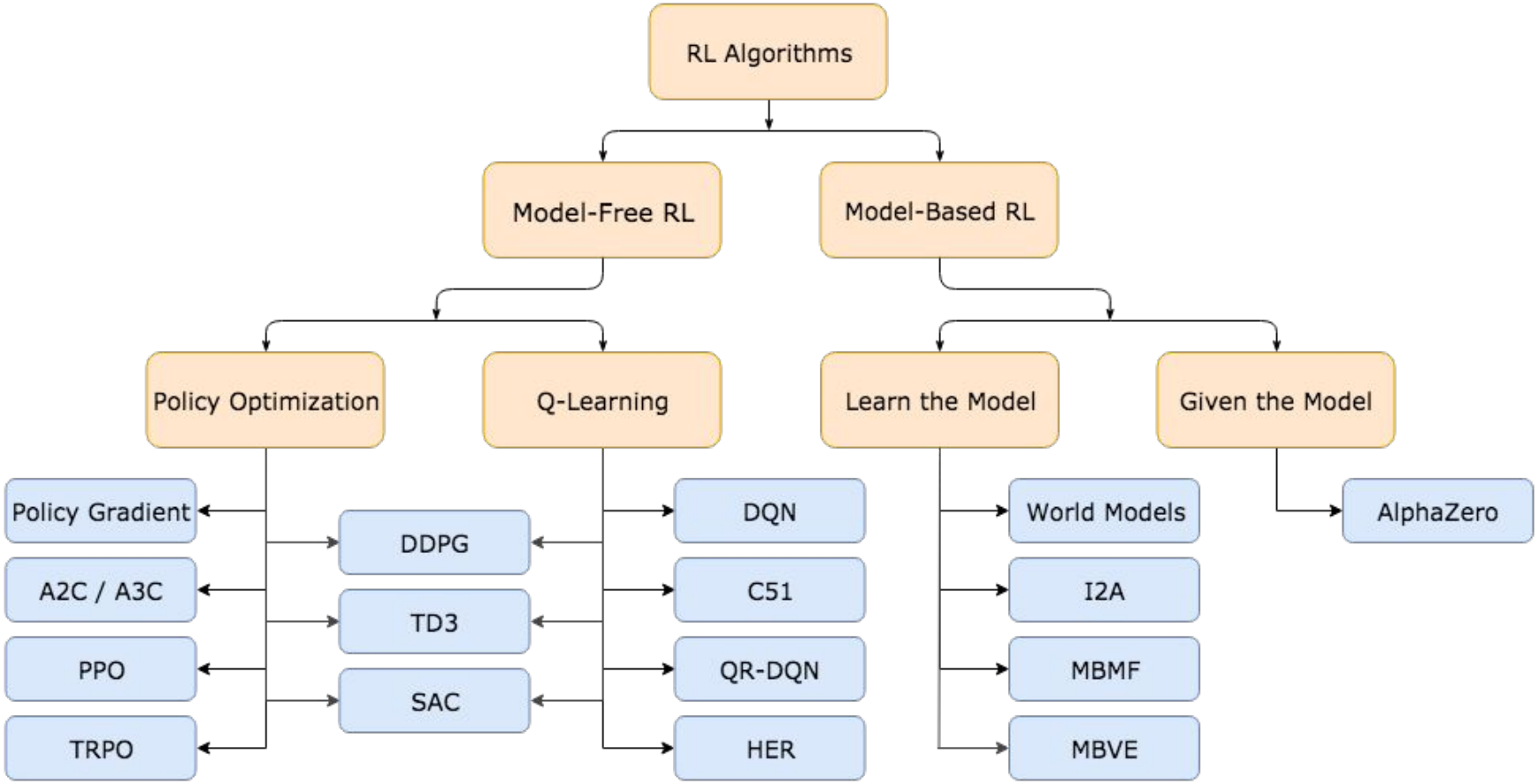

Universidad Internacional de Valencia

De:
Planeta Formación y Universidades

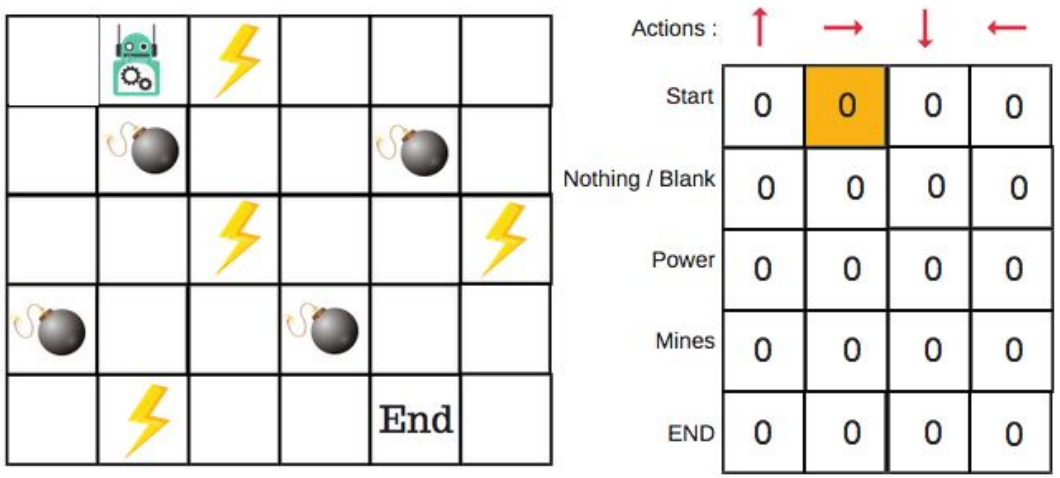
# Índice

# Índice

# Definición Q-learning



# Definición Q-learning




| Actions : | ↑ | → | ↓ | ← |
|---|---|---|---|---|
| Start | 0 | 0 | 0 | 0 |
| Nothing / Blank | 0 | 0 | 0 | 0 |
| Power | 0 | 0 | 0 | 0 |
| Mines | 0 | 0 | 0 | 0 |
| END | 0 | 0 | 0 | 0 |

Cuando hablamos de DQN, tenemos que comenzar hablando de Q-learning.

Q-learning es un algoritmo de aprendizaje por refuerzo que se basa en el aprendizaje a partir de diferencias temporales. Matemáticamente, su enfoque es como una cadena de Markov, en el que la transición entre estados sólo depende de la información que tenemos en el estado actual.

Por ello, la teoría nos dice que existe una función capaz de modelar qué acción es la mejor en cada estado.

# Definición Q-learning

Usando esta función Q podríamos encontrar la estrategia óptima para nuestro agente. Los parámetros de la función Q van a ser pares estado-acción. Es común usar una tabla que relacione estados con acciones.

Esta función nos devolverá, para cada par estado-acción, "el valor esperado de la suma de las recompensas futuras". De esta forma podemos ir midiendo en cada estado cuál es la acción que más nos conviene tomar para maximizar la recompensa futura.

# Índice

# Conceptos importantes

$$Q^{\pi}(s,a) = E[R_t]$$

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

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

Para encontrar la estrategia óptima, este algoritmo se basa en el proceso de exploración-explotación que vimos en la sesión anterior.

La estrategia irá tomando la acción que maximiza el valor de la recompensa esperada.

Respecto a la recompensa esperada, es común usar un ***discount factor*** para estimar la importancia que tendrán los valores futuros.

# Conceptos importantes

Pero, ¿dónde queda la idea de diseñar como cadena de Markov?

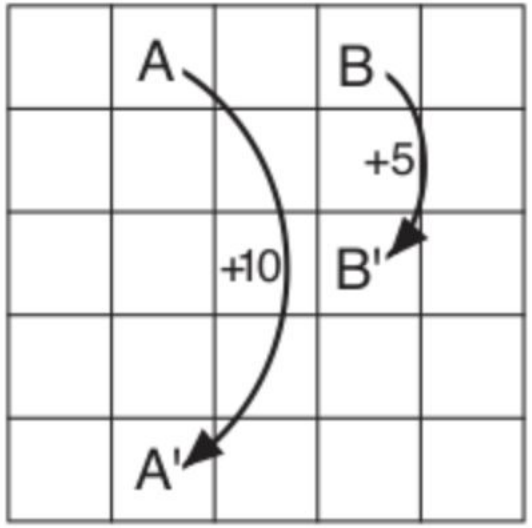
Para ello usaremos la ecuación de Bellman. Como trabajaremos con la recompensa esperada a futuro, necesitamos alguna forma de modelar este comportamiento.

"*La ecuación de Bellman proporciona una definición recursiva con la que podremos encontrar la función óptima Q. La fórmula Q*(s, a) es igual a la suma de la recompensa inmediata, después de ejecutar la función A en el estado S, y la recompensa futura esperada después de la transición al siguiente estado S*".

$$Q^{\pi^*}(s,a) = r + \gamma \max_{a'} Q(s',a')$$

# Índice

# Ejemplo Q-learning: *gridworld*




Actions

- Ejemplo *gridworld* de *Sutton & Barto, Capítulo 3*

- Las celdas corresponden con los estados del entorno.

- Cuatro acciones disponibles. Todas tienen la misma probabilidad de ejecutarse (1/4)

- La recompensa para cada acción-estado es:
  - +10 para la transición de A a A' (que es la única acción que se puede ejecutar en A)
  - +5 para la transición de B a B' (que es la única acción que se puede ejecutar en B')
  - -1 si el movimiento se sale del *grid*
  - 0 en cualquier otro caso

# Ejemplo Q-learning: *gridworld*

```
array([[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.]])
```

Comenzamos en la posición `[0,0]`. Para cada acción posible:

- Si nos movemos arriba:

`0.25 * (-1 + 0.9 * 0) = -0.25`

- Si nos movemos abajo:

`0.25 * (0 + 0.9 * 0) = 0`

- Si nos movemos a la derecha:

`0.25 * (0 + 0.9 * 0) = 0`

- Si nos movemos a la izquierda:

`0.25 * (-1 + 0.9 * 0) = -0.25`

El valor esperado en este estado es ~ `-0.5`

# Ejemplo Q-learning: *gridworld*

```
array([[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.]])
```

Una vez hemos calculado el valor en `[0,0]`, si nos movemos a `[1,0]`:

- Si nos movemos arriba:

`0.25 * (0 + 0.9 * -0.5) = -0.1125`

- Si nos movemos abajo:

`0.25 * (0 + 0.9 * 0) = 0`

- Si nos movemos a la derecha:

`0.25 * (0 + 0.9 * 0) = 0`

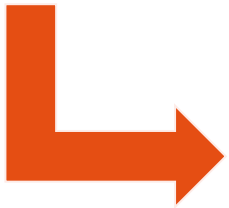
- Si nos movemos a la izquierda:

`0.25 * (-1 + 0.9 * 0) = -0.25`

El valor esperado en este estado es ~ `-0.4`

# Ejemplo Q-learning: *gridworld*

```
array([[-0.5, 10. ,  2. ,  5. ,  0.6],
       [-0.4,  2.2,  0.9,  1.3,  0.2],
       [-0.3,  0.4,  0.3,  0.4, -0.1],
       [-0.3,  0. ,  0.1,  0.1, -0.3],
       [-0.6, -0.4, -0.3, -0.3, -0.6]])
```



```
array([[ 1.4,  9.7,  3.7,  5.3,  1. ],
       [ 0.4,  2.6,  1.8,  1.7,  0.4],
       [-0.2,  0.6,  0.6,  0.5, -0.1],
       [-0.5, -0. ,  0.1,  0. , -0.5],
       [-1. , -0.6, -0.5, -0.6, -1. ]])
```


*Q-values* finales para conocer la estrategia óptima.

```
array([[ 3.4,  8.9,  4.5,  5.4,  1.6],
       [ 1.6,  3.1,  2.4,  2. ,  0.6],
       [ 0.2,  0.9,  0.8,  0.5, -0.3],
       [-0.8, -0.3, -0.2, -0.5, -1.1],
       [-1.7, -1.2, -1.1, -1.3, -1.9]])
```

# Índice

# Deep Q-network

Al ver Q-learning, hemos ido aproximando los *q_values* durante las iteraciones de nuestro proceso de aprendizaje. Nuestro agente ha ido aprendiendo una estrategia a partir de la tabla que se va creando.

Esto ha sido posible porque el problema no era demasiado complejo en términos de dimensionalidad (datos y atributos). En la realidad, la complejidad hace que esta forma de aprendizaje sea intratable.

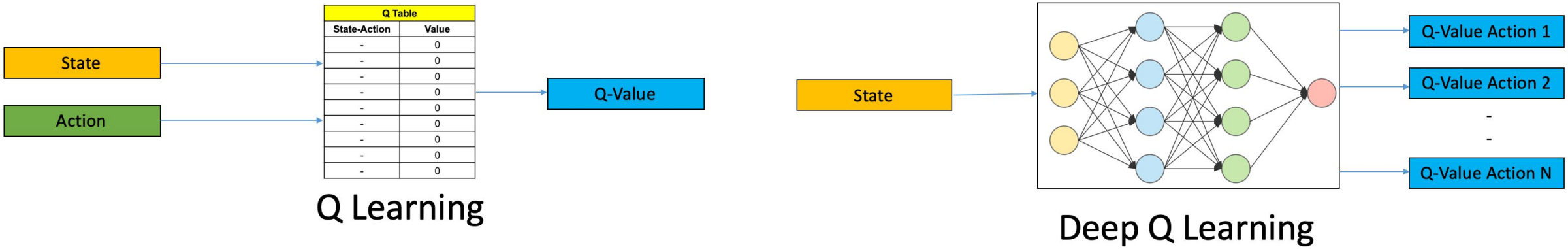
Por ello, en vez de una tabla de *q_values* usaremos una red neuronal como función aproximadora . En vez de aprender los *q_values*, aprenderemos los parámetros de nuestro modelo. Normalmente, y debido a que la mayoría de simulaciones trabajan con la pantalla directamente, el tipo de red neuronal que usaremos serán redes convolucionales.

# Deep Q-network

El uso de redes neuronales como función aproximadora conlleva más decisiones por nuestra parte a la hora del desarrollo de nuestras soluciones.

Ya comentamos en sesiones pasadas que a todos los hiperparámetros necesarios de los algoritmos de aprendizaje por refuerzo hay que sumarle los hiperparámetros de los modelos basados en Deep Learning. Esto hace que el proceso de aprendizaje sea empírico, en el que la prueba y error de nuestras hipótesis es fundamental.

# Deep Q-network




https://cdn.analyticsvidhya.com/wp-content/uploads/2019/04/Screenshot-2019-04-16-at-5.46.01-PM.png

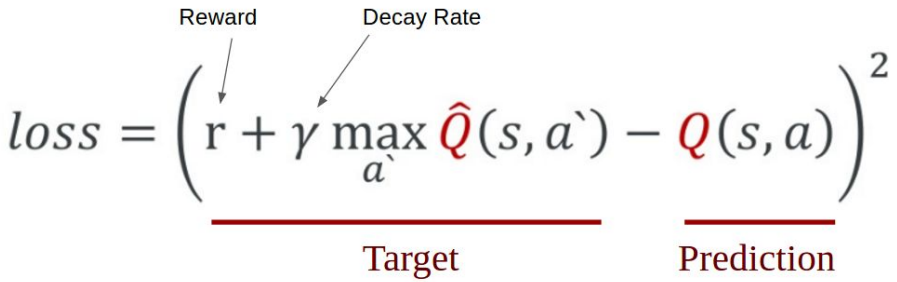
# Índice

# Proceso de aprendizaje

Como queremos aproximar la función Q con un modelo de Deep Learning, necesitamos una función de coste que mida cómo lo estamos haciendo.

Podemos usar la ecuación de Bellman para, iterativamente, aproximar la función Q usando aprendizaje basado en diferencia temporal.

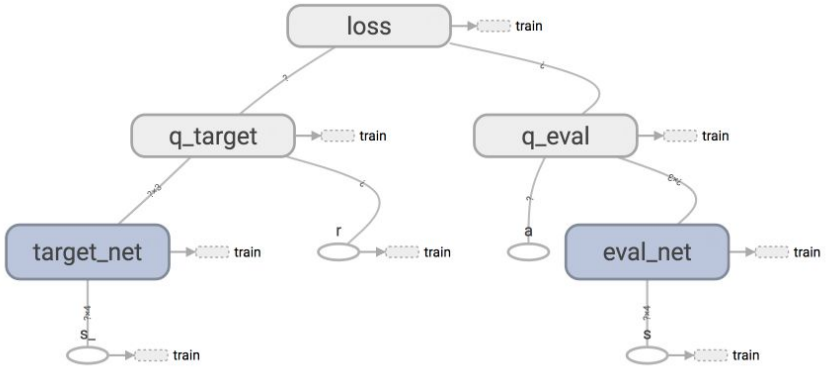
Concretamente, en cada instante de tiempo buscaremos minimizar el error que se produce entre Q(s, a) (término que se predice) y la ecuación de Bellman (término objetivo).




https://keon.io/images/deep-q-learning/deep-q-learning.png

# Proceso de aprendizaje

Respecto al proceso de aprendizaje y a las DQN, hay otros dos elementos que son fundamentales para asegurar la convergencia de nuestro modelo. Estos elementos son la *target network* y el uso de una *secuencia de frames*.





http://neuro.cs.ut.ee/wp-content/uploads/2015/12/breakout_deepmind.png

# Índice

# Algoritmo DQN

**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 $\epsilon$ 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**

# Índice

# Conclusiones

- Hemos estudiado el primero de los algoritmos base que veremos en la asignatura, DQN. Este algoritmos pertenece a la familia de métodos *off-policy.*

- Está basado en Q-learning, donde el objetivo es estimar la recompensa esperada para cada par estado/acción.

- DQN combina Q-learning con arquitecturas de Deep Learning, para poder abarcar problemas con espacios de dimensiones muy grandes

- Algunas variaciones, necesarias para la convergencia de la solución, en la versión final del algoritmo de DQN son el uso de una target network y de secuencias de frames como datos de entrada del modelo del agente.

# Bibliografía recomendada


- *Human level control through Deep Reinforcement learning, Google Deepmind*
*https://deepmind.com/research/publications/2019/human-level-control-through-deep-reinforcement-learning*

- *An Introduction to Q-learning: Reinforcement Learning, Sayak Paul, Floydhub*
*https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/*

Universidad Internacional de Valencia

universidadviu.com

De:
Planeta Formación y Universidades