

DESARROLLO DE JUEGOS CON INTELIGENCIA ARTIFICIAL

Índice

BLOQUE II

0. Introducción

- 1. ¿Qué es el aprendizaje automático?**
- 2. Redes neuronales**
- 3. Árboles de decisión**
- 4. Predicción con n-gramas**
- 5. Aprendizaje por refuerzo (Q-learning)**

Javier Castro Magro

Tema 0. – Introducción

1. APRENDIZAJE AUTOMÁTICO

El Aprendizaje Automático o Aprendizaje Máquina, en inglés Machine Learning (ML), es un subconjunto de técnicas dentro de la Inteligencia Artificial.

Inteligencia Artificial



Aprender, en este contexto, es extraer patrones o modelos/algoritmos a partir de los datos proporcionados a modo de ejemplos.

Aprender NO ES memorizar datos.

2. PLANTEAMIENTO GENERAL EN ML

En ML se dispone de un conjunto de ejemplos de aquello que se quiere aprender denominado **Conjunto de datos (data set/dataset)**.

Cada ejemplo consta de varios **atributos** que lo describen.

Los atributos pueden ser:

– **Numéricos:** Enteros finitos o infinitos, y reales. Por ej. {-1,1} ó [-1,1]

A veces se distinguen entre **continuos y discretos**

– **Categóricos:** Por ej. {Rojo, Verde, Azul} ó {Sí, No}

Se pueden mapear a conjuntos de enteros finitos, o sea a “discretos”.

En definitiva, el conjunto de entrenamiento se puede ver como una tabla donde **cada fila es un ejemplo y cada columna un atributo.**

En ocasiones contaremos con una columna más, que contiene la **etiqueta** para cada ejemplo.

Aprender significa encontrar el **modelo (algoritmo)** que **mejor** capture las propiedades y características de estos datos y/o la asociación entre estas propiedades y las etiquetas.

Conjunto de datos / data set

	atributo 1	atributo 2	...	atributo M	etiqueta
ejemplo 1
ejemplo 2
...
ejemplo N

2.1 TIPOS DE ML

Según la tarea a realizar y/o los datos de que se dispone, las técnicas de ML se pueden clasificar en:

- **Aprendizaje supervisado:** Cada ejemplo proporcionado lleva una etiqueta asociada. El objetivo del algoritmo es asignar a un nuevo ejemplo, no etiquetado, la etiqueta correcta.

Etiquetas discretas = Problema de **clasificación**.

Etiquetas continuas = Problema de **regresión**.

- **Aprendizaje no supervisado:** Solo tenemos ejemplos, no etiquetas. Se pretende agrupar los ejemplos en grupos o clusters y también decidir a qué grupo pertenece un nuevo ejemplo.

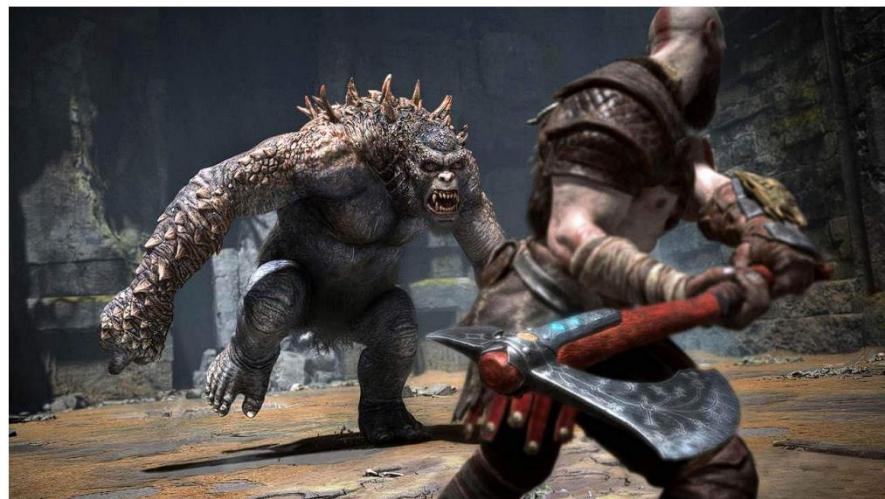
- **Aprendizaje por refuerzo:** Hay información parcial que me dice como de bien estoy aprendiendo. Es una aproximación “débil” de aprendizaje supervisado, más apropiada para vj., que consiste en proporcionar alguna realimentación positiva al sistema cada vez que se hace una acción buena.

APLICACIÓN EN VIDEOJUEGOS

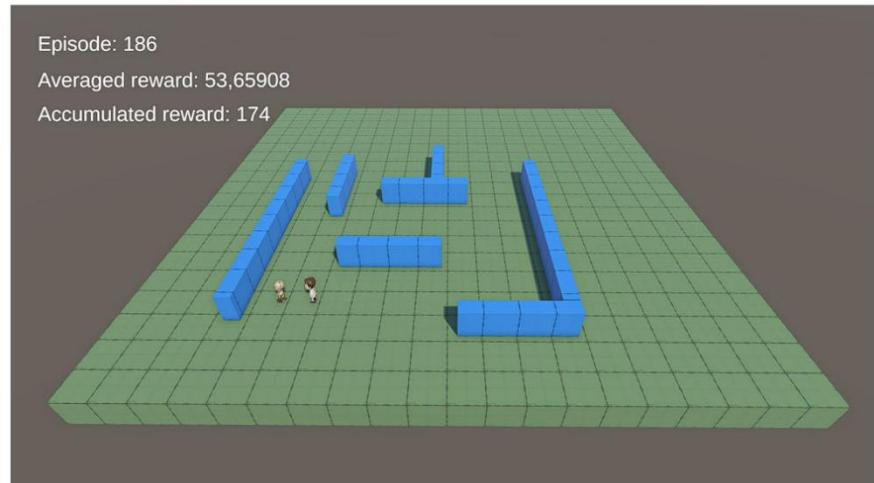
Toma de decisión (Optimización numérica)



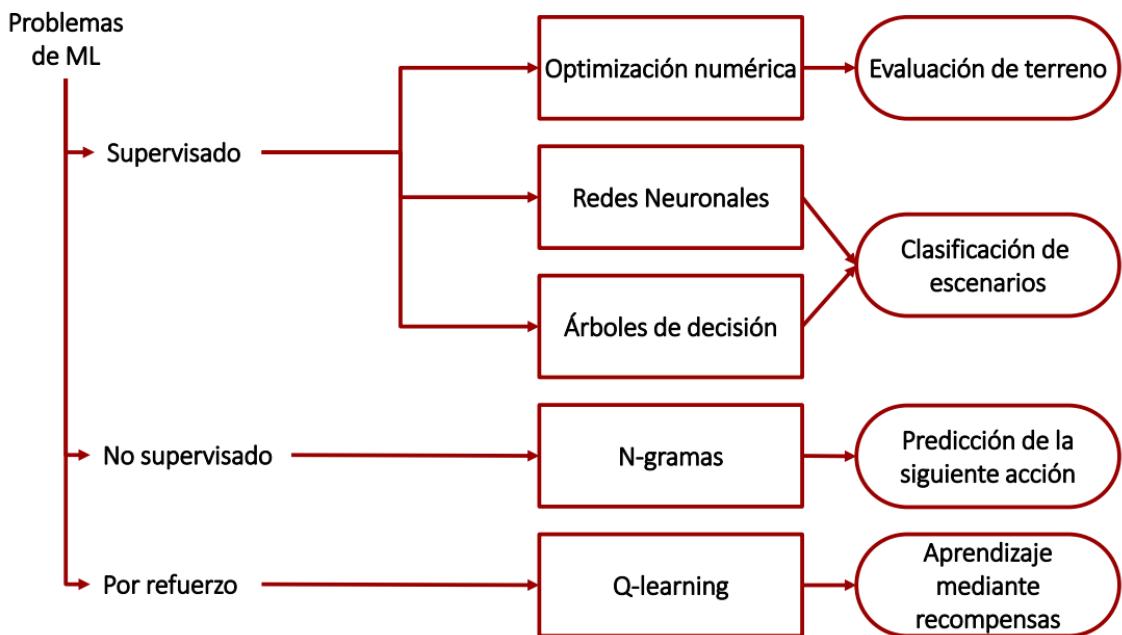
Balanceo de la dificultad (Redes neuronales)



Agentes NPC inteligentes (Aprendizaje por refuerzo)

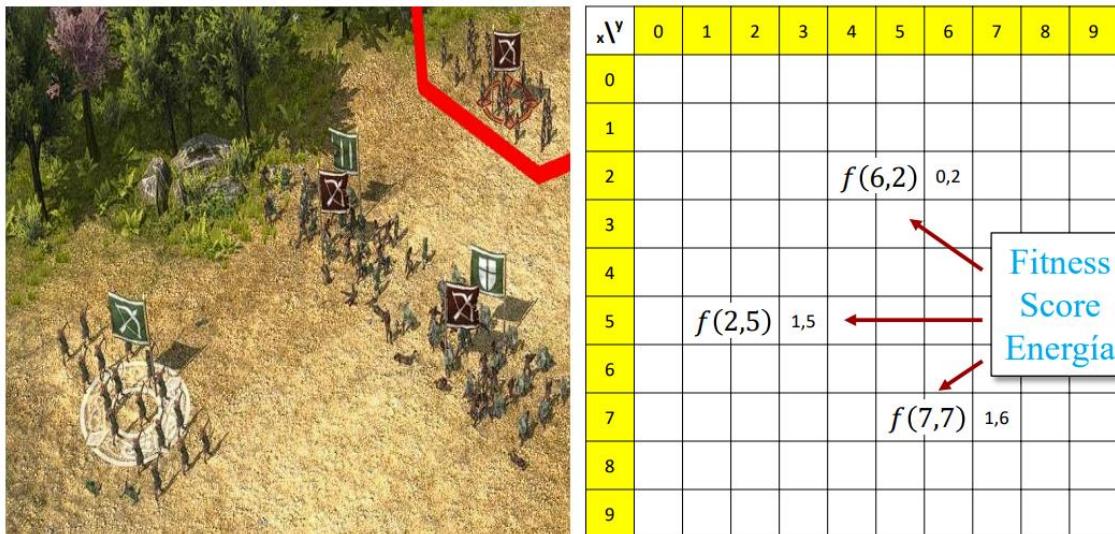


3. MÉTODOS



Tema 1. – ¿Qué es el aprendizaje automático?

1. SUPERVISADO – OPTIMIZACIÓN NÚMERICA – EVALUACIÓN DE TERRENO



Disponemos de una función $f(x, y) = \text{daño causado}$

¿Dónde disparamos?

La optimización es una rama de las matemáticas.

Si el problema (encontrar el mejor valor de **ff**, el fitness), **es amable** se podrá resolver **analíticamente**, de manera exacta. Es decir, **llegaremos a la solución óptima**, la mejor posible.

Cuando **NO** es posible, es por alguna de estas **razones**:

- La **función** a optimizar es **muy difícil/imposible de tratar analíticamente**.
- La **función** a optimizar **cambia con el tiempo**.

La **solución** es **resolverlo numéricamente**.

- Es decir, aplicamos un **algoritmo que sucesivamente evalúa la función** y va guardando el mejor resultado hasta una condición de parada.
- **No se encuentra la solución óptima**, pero sí una **bastante buena** y en mucho menos tiempo.

ALGORITMO DE ESCALADA BÁSICO (HILL CLIMBING)

Se trata de un **proceso iterativo** en el que se evalúa el **fitness** alrededor del valor actual de los parámetros y **se elige el camino por el que se sube más**.

Punto inicial: Siempre se arranca con un **punto inicial aleatorio**.

Paso: El **paso** es la distancia a la que se encuentran los puntos vecinos del punto en el que nos encontramos en la iteración actual.

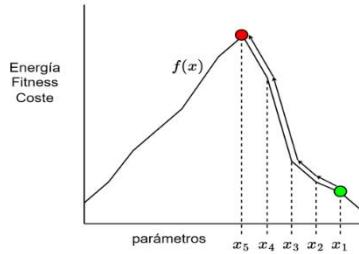
BestNeighbour devuelve el **mejor** vecino del punto dado. En el Hill Climbing básico es el vecino con mayor fitness.

Optimización mediante Hill Climbing

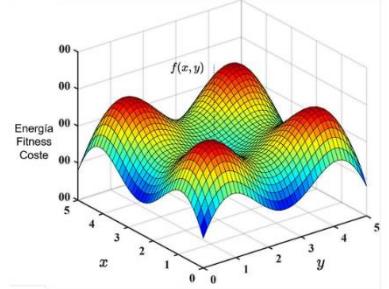
```

1: procedure HILLCLIMBING
2:   current  $\leftarrow$  randomStartPoint
3:   energy =  $\infty$ 
4:   while energy > 0 do
5:     V  $\leftarrow$  BestNeighbour current
6:     energy  $\leftarrow$  fitness V – fitness current
7:     if energy > 0 then
8:       current  $\leftarrow$  V
9:     end if
10:   end while
11:   return current
12: end procedure
```

- Con 1 parámetro



- Con 2 parámetros



	0	1	2	3	4	5	6	7	8	9
0										
1									0	
2						0,2	0,2	0		
3						1,1	1,1	0,1		
4				3,3	2,4	2,2	1,3			
5					2,4	2,2	1,3			
6						2,6	2,7	1,8		
7						2,8	3,1	2,2		
8							3			
9										

Hemos llegado a un buen valor de daño: 3,1

¡¡Pero había uno mejor: 3,3!!

En 3,1 hemos encontrado un **máximo local (subóptimo u óptimo local)**

SUBÓPTIMOS

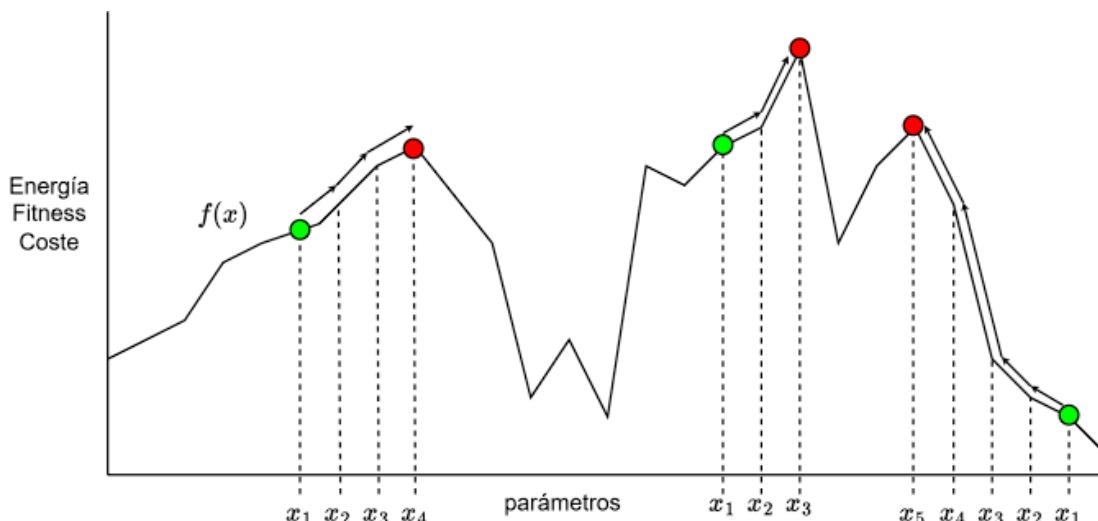
Hill Climbing encuentra un buen valor máximo, pero **puede no encontrar el mejor valor; puede encontrar un subóptimo u óptimo local.**

ARRANQUE MÚLTIPLE

Trata de evitar los **óptimos locales** ejecutando el algoritmo varias veces, y eligiendo cada una de ellas un punto inicial diferente elegido al azar. Nos quedaremos con el punto devuelto por la ejecución que devuelva el mayor fitness.

Esta técnica se denomina **arranque múltiple**.

- Si tenemos varias veces el mismo resultado, podemos suponer que es el global.

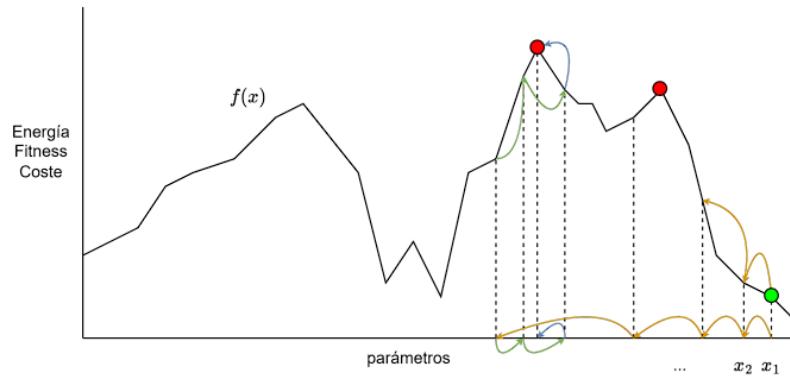


PASO ADAPTATIVO

Un algoritmo más elaborado consiste en hacer que el paso sea X veces el valor del paso anterior. Por ejemplo, $s = 1.3$. De este modo, avanzamos cada vez más rápido en la escalada mientras mantengamos un sentido ascendente.

El valor de ss es otro control más que se añade al algoritmo.

Si el siguiente paso nos lleva a un Fitness menor que el anterior el paso se resetea, es decir vuelve a su valor inicial.



ALGORITMO DE ESCALADA CON MOMENTUM

Otro modo de superar óptimos locales es incorporar cierta **inercia/momentum**.

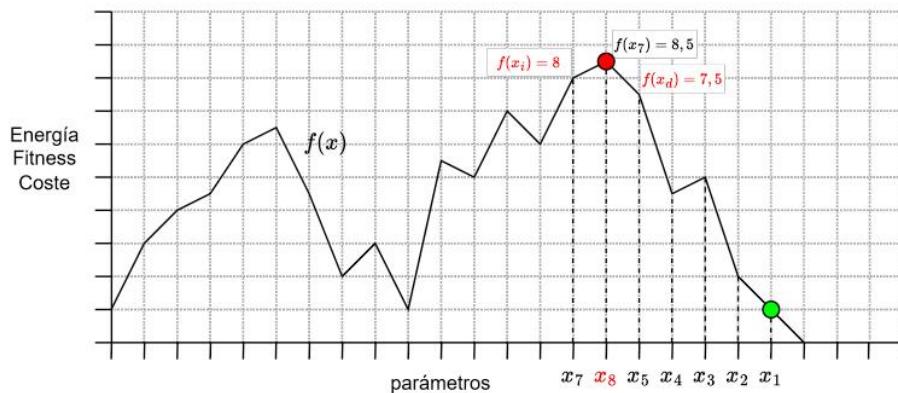
El algoritmo avanza igual, buscando el camino de mayor subida.

Al terminar un paso se calcula un **porcentaje** de cuanto ha mejorado el Fitness en dicha iteración (incremento en el Fitness)

En la iteración siguiente, ese porcentaje se suma **SÓLO** al incremento en el Fitness que se produzca **en la misma dirección** que el paso anterior.

De este modo se favorece el avance en la dirección que traímos y podemos superar óptimos locales.

Otro control más para el algoritmo: el porcentaje de inercia o **momentum**.



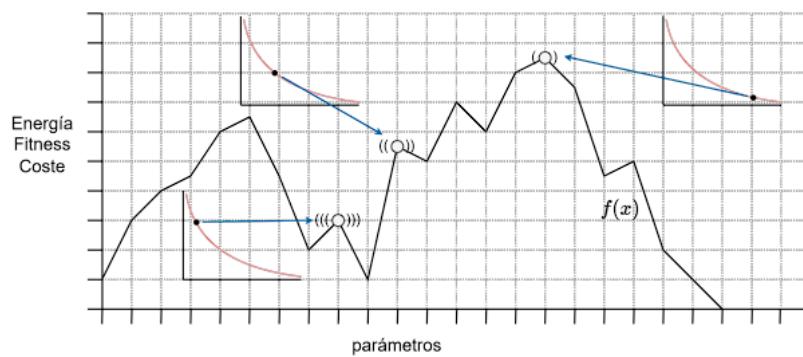
Iteración	Fitness	Momentum	Fitness	Incremento	Incremento	Fitness actual	Fitness	Incremento	Incremento	Mejor	Mejor
	anterior	30%	Izq.	Fit. Izq.	Fit.Izq + Mom		Der.	Fit. Der	Fit.Der + Mom	Inc.Fit	Direccion
1	0	0,0000	2	1,00	1,00	1	0	-1,00	-1,00	1,00	Izq
2	1	0,3000	5	3,00	3,30	2	1	-1,00	-1,00	3,30	Izq
3	2	0,9900	4,5	-0,50	0,49	5	2	-3,00	-3,00	0,49	Izq
4	5	0,1470	7,5	3,00	3,15	4,5	5	0,50	0,50	3,15	Izq
5	4,5	0,9441	8,5	1,00	1,94	7,5	4,5	-3,00	-3,00	1,94	Izq
6	7,5	0,5832	8	-0,50	0,08	8,5	7,5	-1,00	-1,00	0,08	Izq
7	8,5	0,0250	6	-2,00	-1,98	8	8,5	0,50	0,50	0,50	Der
8	8,5	0,1500	8	-0,50	-0,50	8,5	7,5	-1,00	-1,00	-0,50	STOP

TEMPLADO SIMULADO (SIMULATED ANNEALING)

El templado simulado es una técnica de fabricación de aleaciones que consiste en aumentar su temperatura para agitar las partículas y después enfriar despacio para que estas ocupen estados de energía mínima. De este modo se logran aleaciones sin fallos internos (burbujas, fisuras..)

La optimización con **templado simulado** se inspira en esta técnica para salir del máximo local, en base a una probabilidad que es mayor al principio y menor según se avanza en el tiempo.

Este algoritmo añade un nuevo control: la **curva de enfriamiento**.



NOTACIÓN PARA SIMULATED ANNEALING

Notación

- $k \in \{1, 2, 3, \dots\}$ Iteración del algoritmo.
- X_k : Conjunto de parámetros alcanzado en la iteración k .
- V_k : Un vecino de X_k evaluado en la iteración k .
- $f(k)$: Función de temperatura, que debe ser descendiente a 0 según crece k .
- $T_k = f(k)$: Temperatura en la iteración k calculada con la función f .
- \square : Incremento en el Fitness (Energía, Score, ...) cuando se evalúa un vecino.
- $n \in [0, 1]$: un número aleatorio distribuido entre 0 y 1 que expresa la probabilidad de avanzar en la dirección de ese vecino

ALGORITMO DEL TEMPLADO SIMULADO PROBABILÍSTICO

RandomNeighbour devuelve un vecino aleatorio del punto dado.

p representa la probabilidad de continuar en esa dirección

$$p = e^{\frac{-|f(V) - f(X)|}{T_k}}$$

$n \in [0,1)$ evalúa esa probabilidad

Optimización mediante Simulated annealing

```

1: procedure SIMMULATEDANNEALING
2:   current  $\leftarrow$  randomStartPoint
3:   for each  $T_k$  in  $[T, 0]$  do
4:      $V \leftarrow$  RandomNeighbour current
5:     energy  $\leftarrow$  fitness  $V$  – fitness current
6:     if energy  $> 0$  then
7:       current  $\leftarrow V$ 
8:     else
9:        $p \leftarrow e^{-\frac{|energy|}{T_k}}$ 
10:       $n \leftarrow$  random  $[0, 1)$ 
11:      if  $p \geq n$  then
12:        current  $\leftarrow V$ 
13:      end if
14:    end if
15:   end for
16:   return current
17: end procedure

```

$x \setminus y$	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1,9	1,4	0,1	0	0	0	0
2	0	0	0	2,2	3,2	0,2	0,2	0	0	0
3	0	0	0,1	1,2	3,3	1,1	1,1	0,1	0	0
4	0	0	0,3	1,1	3,2	2,4	2,2	1,3	1,1	0
5	0	0	0,6	0,5	1,5	2,6	2,7	1,8	1,2	0
6	0	0,5	1,1	1,7	0,3	2,8	3,1	2,2	1,1	0
7	0	1,8	2,1	2,1	0,1	3,2	3	1,6	1,9	0
8	0	1,5	1,3	2,9	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

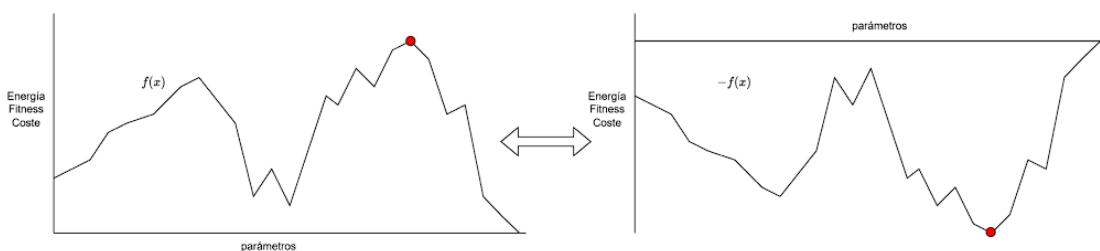
k	T	Fitness actual		Fitness V_k		De	p	n	Aceptar
		V	k	V	k				
1	10	0,6	E	0,5	-0,1	0,9900	0,3	Sí	
2	8	0,5	N	1,1			0,8	Sí	
3	6	1,1	E	3,2			0,1	Sí	
4	4	3,2	S	1,5	-1,7	0,6538	0,7	NO	
5	2	3,2	N	3,3			0,3	Sí	
6	0	3,3	O	1,2	-2,1	0,0000	0,9	NO	

MAXIMIZAR = OPTIMIZAR = MINIMIZAR

El punto donde se alcanza el máximo de una función es el mismo que donde se alcanza el mínimo de la función con el signo cambiado.

Hasta ahora, hemos **optimizado** mediante **maximización** (buscando el mayor fitness posible). Si queremos **minimizar**, solo tenemos que usar la misma función de fitness, en valor negativo.

$$x^* = \arg \max f(x) = \arg \min -f(x)$$



Tema 2. – Redes neuronales

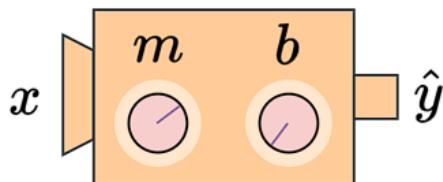
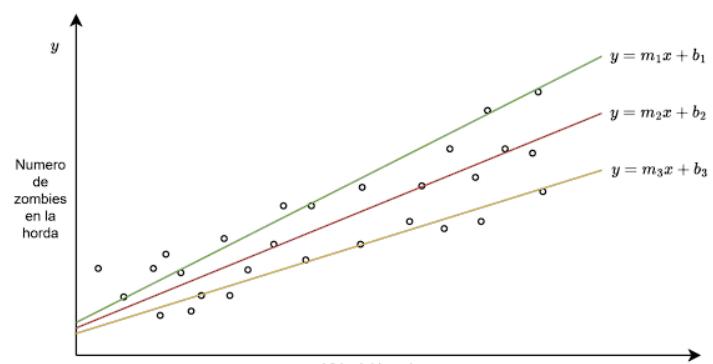
2. SUPERVISADO – REDES NEURONALES – CLASIFICACIÓN DE ESCENARIOS

REGRESIÓN LINEAL: EL PROBLEMA

Balanceo de la dificultad:

Disponemos de datos de partidas donde los jugadores han superado la horda, con el nivel de vida del jugador en ese momento.

Queremos calcular/predecir una cantidad de zombies en la horda que sea superable para un jugador con un determinado nivel de vida.



CLASIFICACIÓN LINEAL: EL PROBLEMA

IA para decisión estratégica

La IA controla nuestro personaje, decide su estrategia

Dos escuadrones, 5 soldados por escuadrón

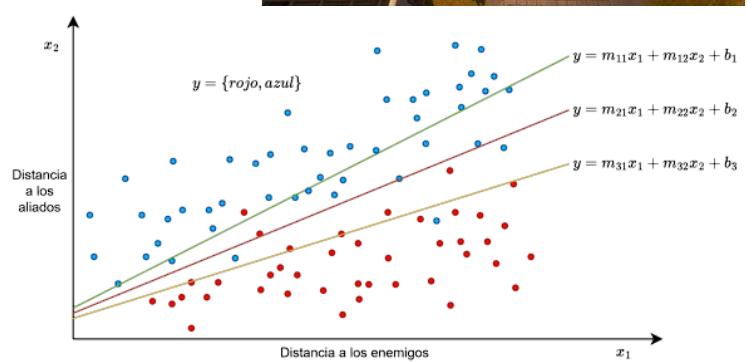
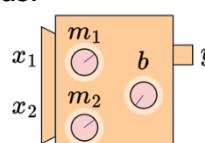


Datos:

- Distancia a los enemigos
- Distancia a los aliados

Posibles acciones/etiquetas:

- Huir
- Luchar



NEURONAS BIOLÓGICAS Y ARTIFICIALES

Las neuronas son células del cerebro encargadas de transmitir impulsos nerviosos.

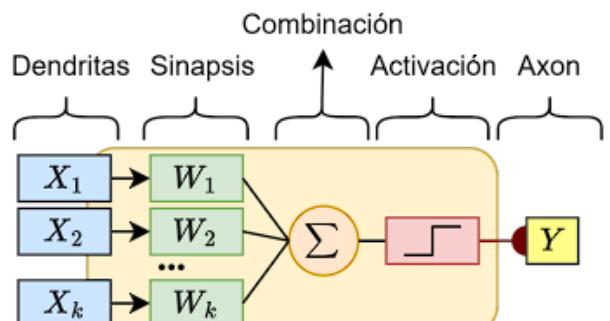
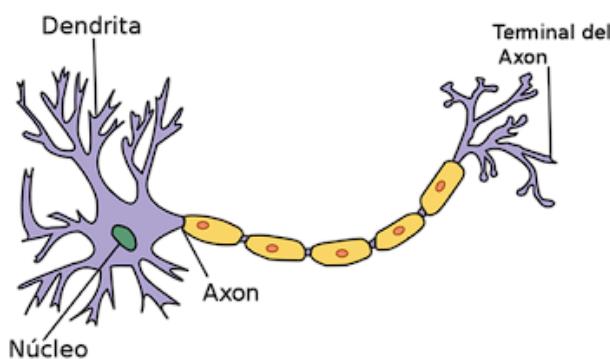
Morfología:

- **Dendritas:** terminaciones por las que **recibe impulsos**
- **Núcleo:** zona central de la célula, donde se **realizan sus funciones**
- **Axón:** terminación por la que **envía impulsos**
- **Sinapsis:** La **unión axón-dendrita** cuando se está **transmitiendo el impulso**

Funcionamiento:

- Los impulsos llegan al núcleo a través de las sinapsis.
- Cada sinapsis tiene una **fuerza diferente**.
- La neurona combina los impulsos recibidos y emite uno a través del axón.
- Si ese impulso supera un **umbral de activación** entonces se transmite, en caso contrario NO se transmite.

Las neuronas artificiales son un modelo de computación, inspirado en el funcionamiento de las neuronas biológicas.



PERCEPTRÓN: MODELO DE UNA NEURONA ARTIFICIAL

El modelo de 1 neurona fue el primero propuesto y se llamó **Perceptrón**

Una neurona artificial realiza una **combinación lineal** de las entradas.

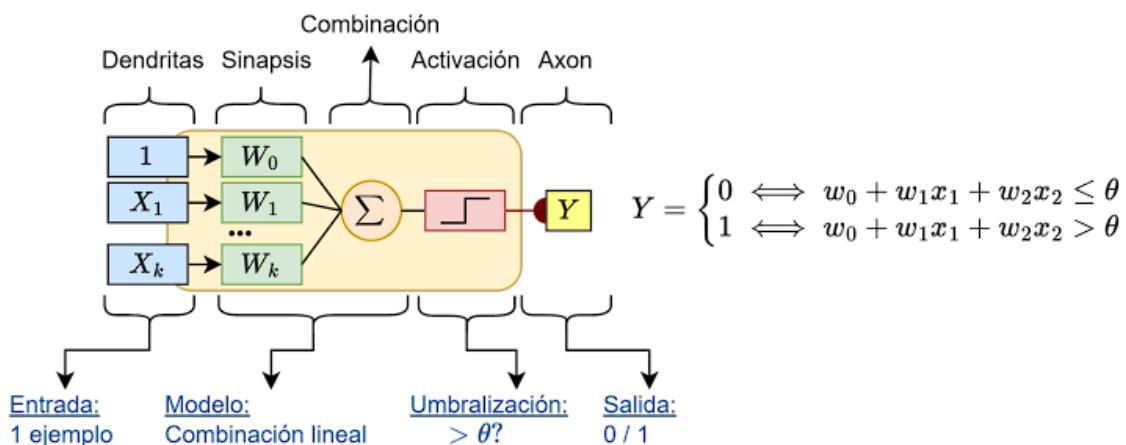
Una combinación lineal es una suma ponderada de las variables.

Las variables son los atributos que describen cada ejemplo

Los **pesos** representan la **contribución de cada atributo**

Además, se añade un **término independiente**, es decir un valor w_0 como offset, bias o sesgo.

Si resultado de la combinación lineal supera un cierto umbral θ se transmite un 1 En caso contrario se transmite un 0



MECANISMO DE APRENDIZAJE

Las redes neuronales son principalmente un método de clasificación.

Cada ejemplo proporcionado va acompañado de una **etiqueta**.

En el modo más simple habrá dos etiquetas, que se pueden codificar como 0 y 1.

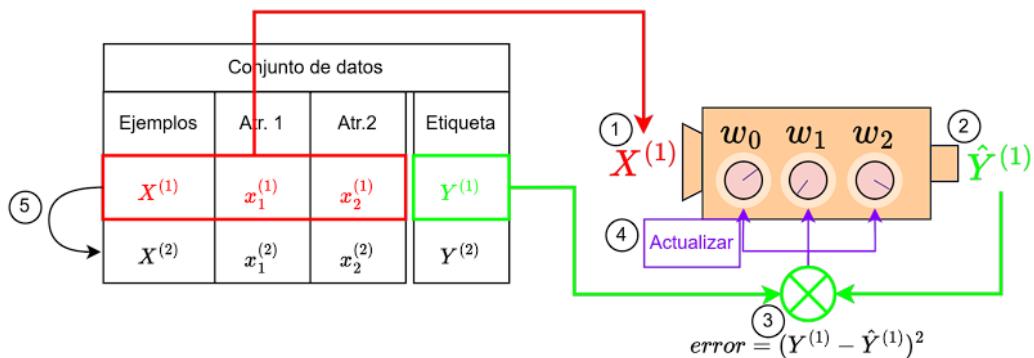
Aprender consiste en:

1. Alimentar la neurona con **un ejemplo**
2. **Calcular la salida**, es decir la **etiqueta estimada** por la neurona para ese ejemplo.

3. Comparar la etiqueta estimada con la etiqueta verdadera y calcular el error como la diferencia, al cuadrado para que sea positivo siempre.

4. Se actualiza el valor de los pesos con una regla de aprendizaje

5. Iterar con los siguientes ejemplos.



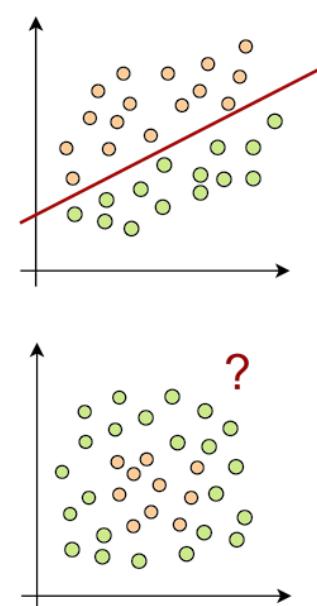
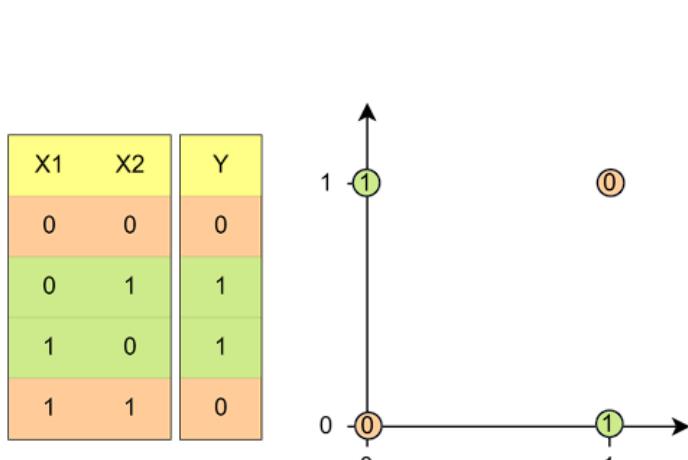
LIMITACIONES

Su utilidad fundamental es aprender a clasificar ejemplos de 2 clases que se puedan separar por una recta, un plano o en general un hiperplano.

Este tipo de problemas se denomina **linealmente separables**

Muchos problemas no son linealmente separables.

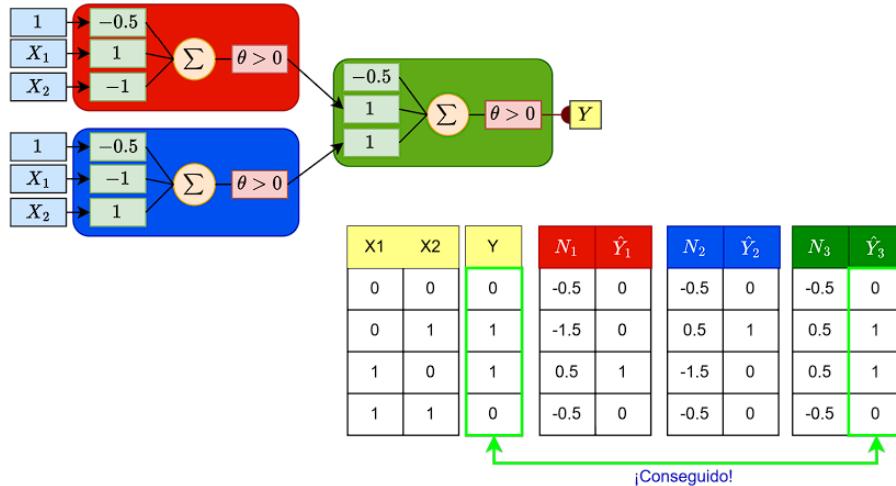
En concreto hay uno muy sencillo ante el cual el **Perceptrón** fracasa:



REDES DE NEURONAS

Al igual que en la naturaleza, las neuronas son más eficaces si forman circuitos o redes, donde la salida de una neurona es una entrada de otra.

Por ejemplo, el caso anterior se resuelve con una red de dos capas y 3 neuronas



ARQUITECTURA MLP (MULTILAYER PERCEPTRON)

Capa de entrada

- 1 neurona sin pesos por cada atributo.
- Simplemente sirve como puerto de entrada.

Capa(s) oculta

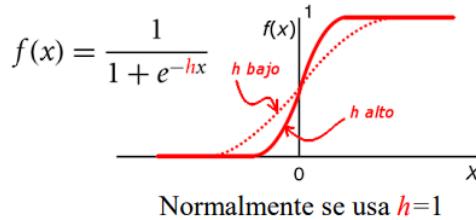
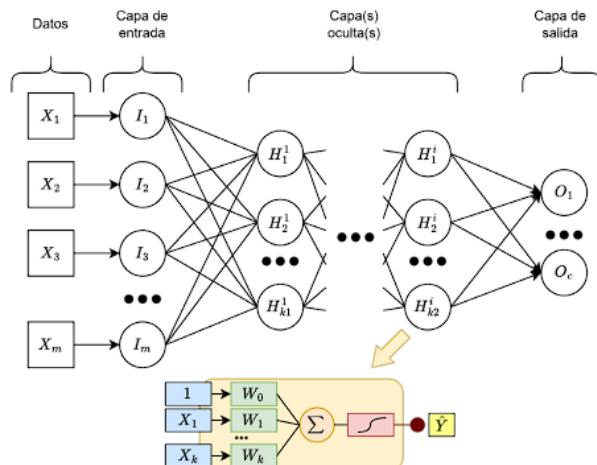
- Un cierto número de neuronas.
- Cada neurona está conectada a todas las neuronas de la capa anterior.

Capa de salida

- 1 neurona por cada clase, conectada a todas las neuronas de la capa anterior.

Función de activación

- La binarización es sencilla de entender, pero **impide** aprender de manera eficiente
- Se utiliza la sigmoide porque es **matemáticamente conveniente**



EJEMPLO

IA para decisión estratégica

- La IA controla nuestro personaje, decide su estrategia
- Dos escuadrones, 5 soldados por escuadrón
- Datos:

Distancia a los enemigos, distancia a los aliados, salud de los aliados, munición de los aliados, salud propia, munición propia.

- Posibles acciones/etiquetas:

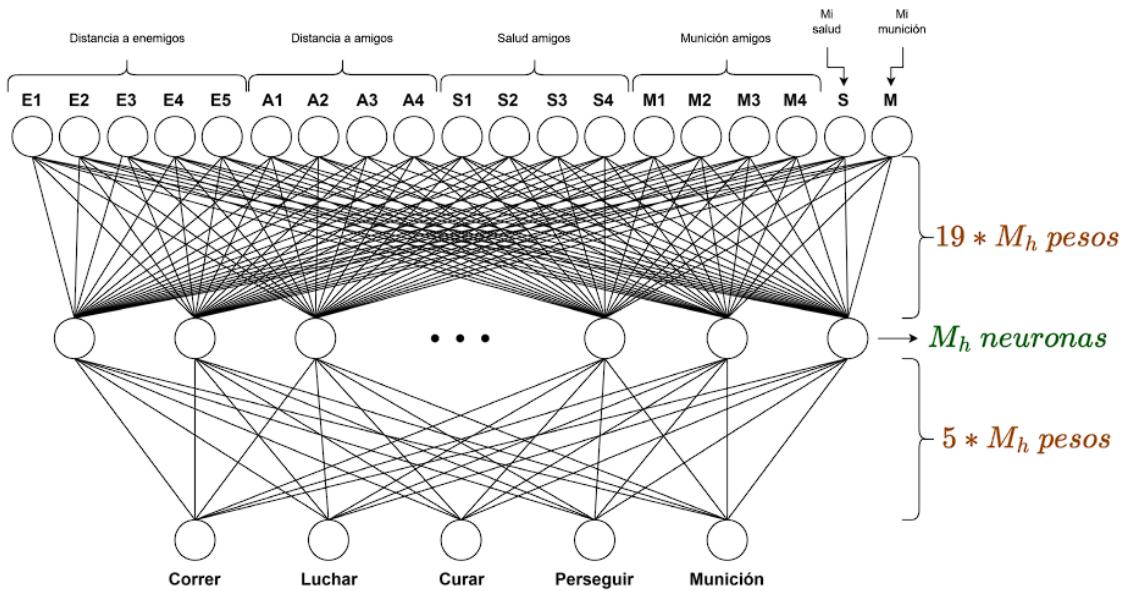
Correr, luchar, curar amigo, perseguir enemigo, buscar munición.



Data set	Distancia a enemigos					Distancia a amigos				Salud amigos				Munición amigos				Acciones						
	E1	E2	E3	E4	E5	A1	A2	A3	A4	S1	S2	S3	S4	M1	M2	M3	M4	S	M	Correr	Luchar	Curar	Perseguir	Munición
1	17	18	29	11	30	10	19	14	5	9	9	4	8	22	39	14	8	7	40	1	0	0	0	0
n	10	8	9	6	17	19	1	14	14	8	7	7	3	39	1	17	21	3	3	0	1	0	0	0

Características del modelo

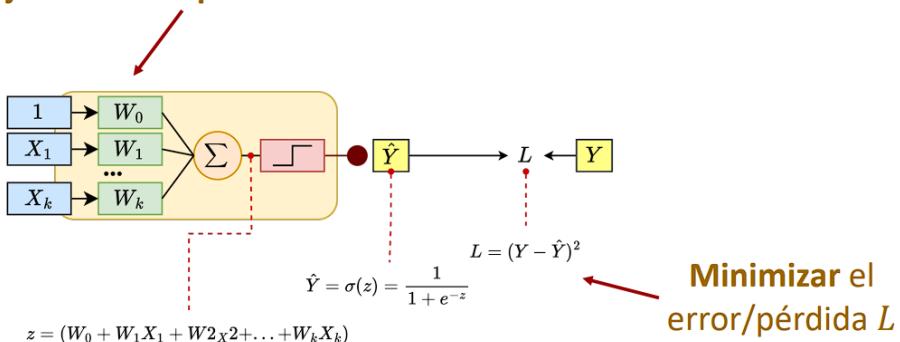
- **19 atributos → 19 columnas = 19 neuronas de entrada**
- **5 clases excluyentes → 5 neuronas de salida**
- **1 capa oculta con un número indeterminado M_h de neuronas**
- **Número total de pesos: $19M_h + M_h + 5M_h + 5$**



REDES NEURONALES: APRENDIZAJE

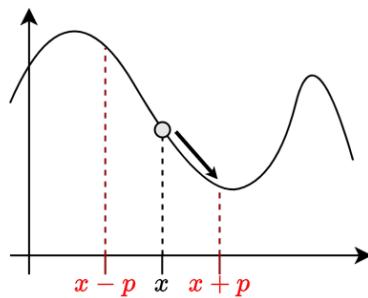
Aprender es optimizar:

Ajustando los pesos W^*

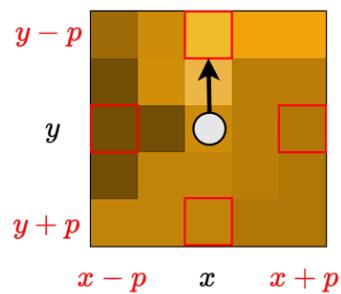


Aprender es optimizar:

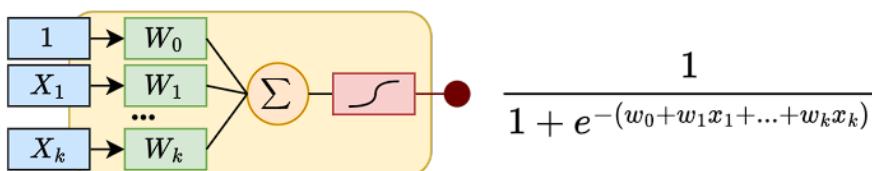
1 parámetro: 2^1 vecinos



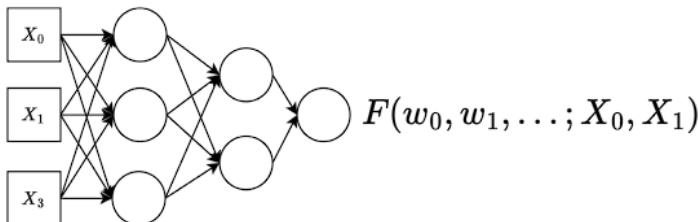
2 parámetros: 2^2 vecinos



1 neurona es realmente una función paramétrica de las entradas.



1 red es realmente una función paramétrica (más compleja) de las entradas.



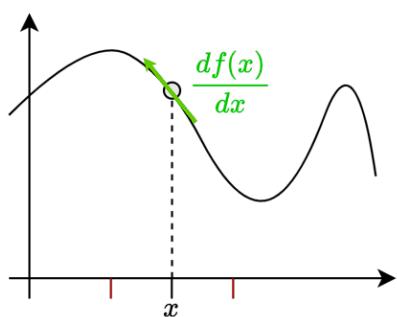
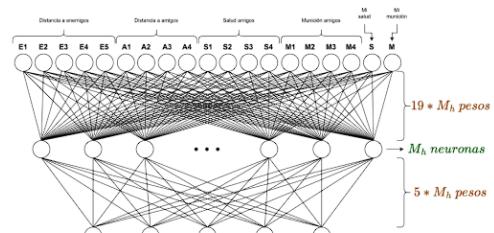
Se podría encontrar los W^* que minimicen el error L , pero ¿a qué “precio”?

– Una red como la del ejemplo tiene 23 pesos/parámetros: $2^{23} = 8,388,608$ vecinos

¿a qué “precio”?

- $19N_h + N_h + 5N_h + 5$ pesos (si $N_h = 8$ habría 213 pesos)

$$\text{!!!!} 2^{213} = 1.32^{64} \text{ vecinos!!!!}$$



- **Derivando $f(x)$** podemos averiguar en qué dirección y con qué intensidad crece la función, sin necesidad de evaluar sus vecinos. Podemos optimizar en la dirección de esta derivada:

DESCENSO DEL GRADIENTE

¿Cuánto hay que cambiar **cada W** para que L tenga la mayor bajada?

$$w_1' = w_1 - algo$$

$$w_1' = w_1 - \frac{dL(w; x)}{dw}$$

Variación del error con respecto a los parámetros
Para ampliar, ver [Derivada del coste](#)

$$w_1' = w_1 - -(Y - \hat{Y})(1 - \hat{Y})(\hat{Y})x$$

$$w_1' = w_1 + \alpha(Y - \hat{Y})(1 - \hat{Y})(\hat{Y})x$$

No se aplica todo, sólo un porcentaje $\alpha = (0,1)$, la *ratio de aprendizaje* o *learning rate*

REGLA DE APRENDIZAJE

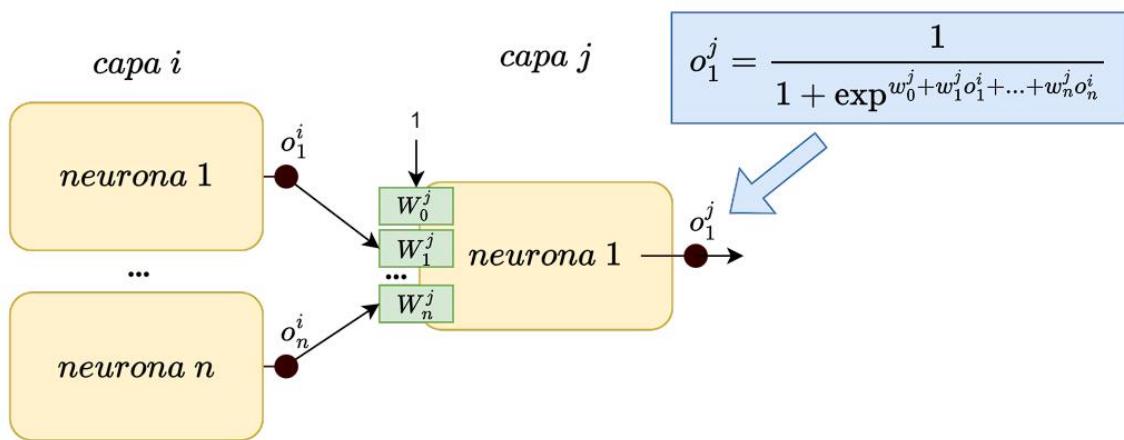
Back Propagation

- Método desarrollado a partir de la función de activación sigmoide.
- Cada ejemplo introducido ajusta los pesos de la red.
- Para ello:

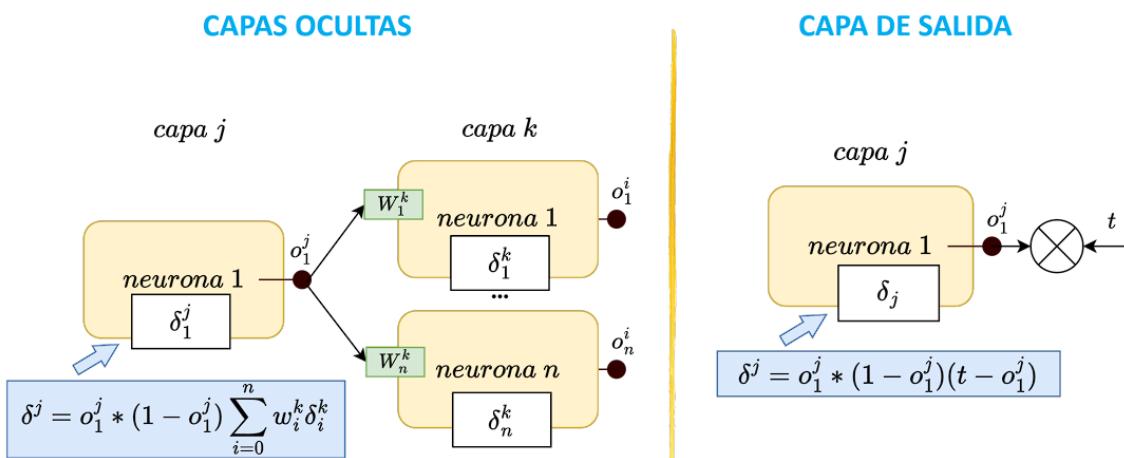


1. Se propaga hacia delante (**feed forward**) la entrada y se almacenan los resultados parciales de cada neurona hasta llegar a las de la capa de salida.
2. Se calcula el error (δ) producido por la red, comparando la salida de esta con la etiqueta que tenemos en los datos. Este error se propaga (**back propagation**) a **todas las neuronas** desde la capa de salida hasta la de entrada.
3. Se actualizan los pesos (**gradient descent**) añadiéndoles, al peso actual, el error correspondiente a su neurona por la entrada de ese peso multiplicado por una ratio de aprendizaje (learning rate).

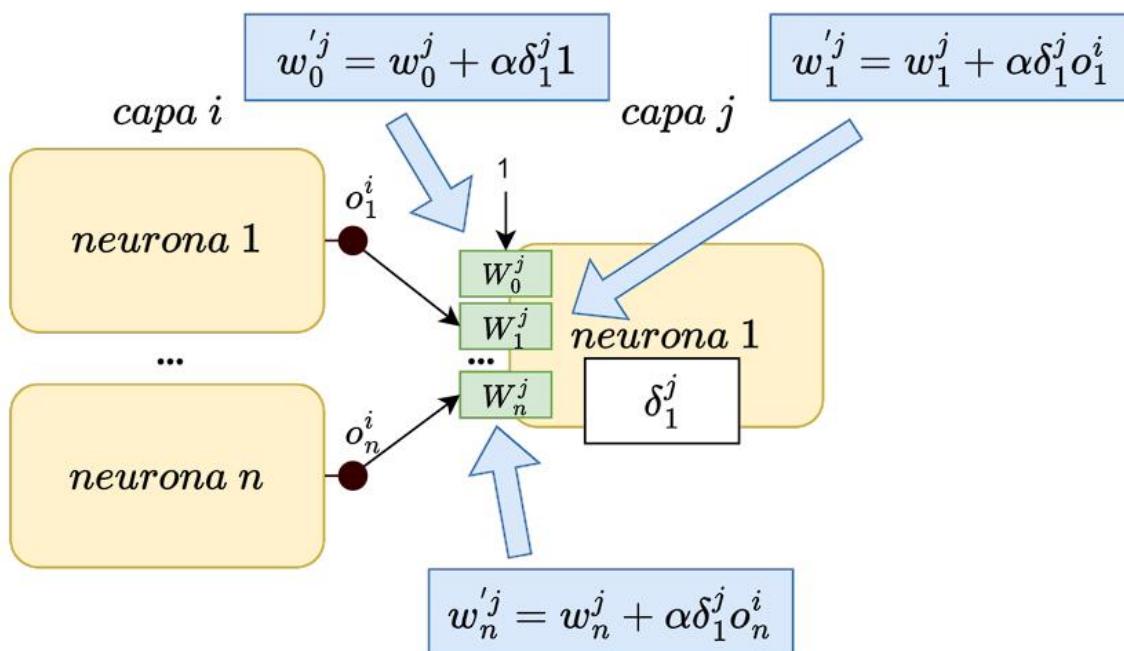
FEED FORWARD



BACK PROPAGATION



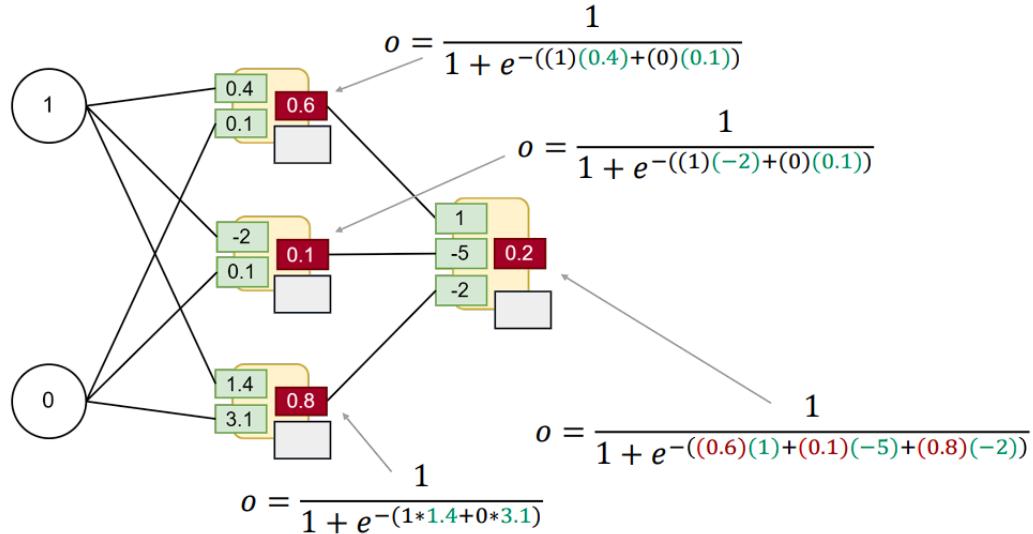
GRADIENT DESCENT



APRENDIZAJE = AJUSTE DE LOS PESOS

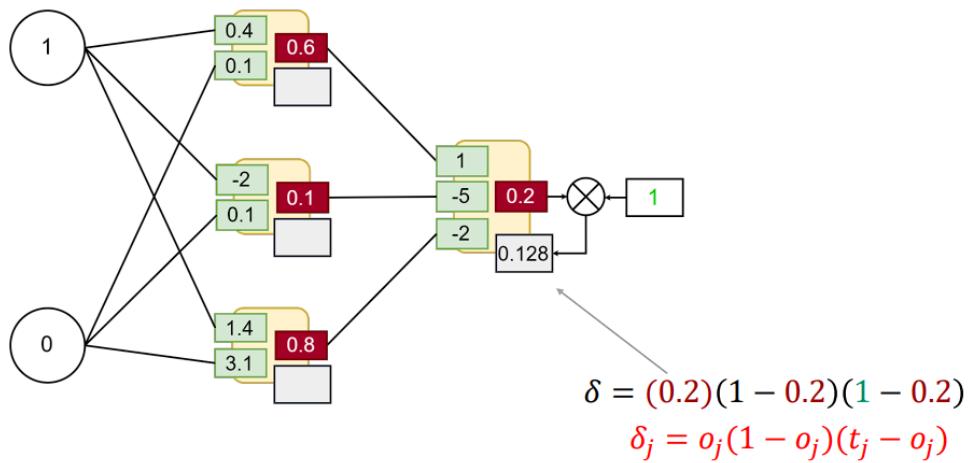
1. Etapa de alimentación hacia delante (*feed-forward*)

1. La red recibe un ejemplo etiquetado
2. Cada neurona de cada capa calcula su **salida**, la salida de la última capa es la predicción.

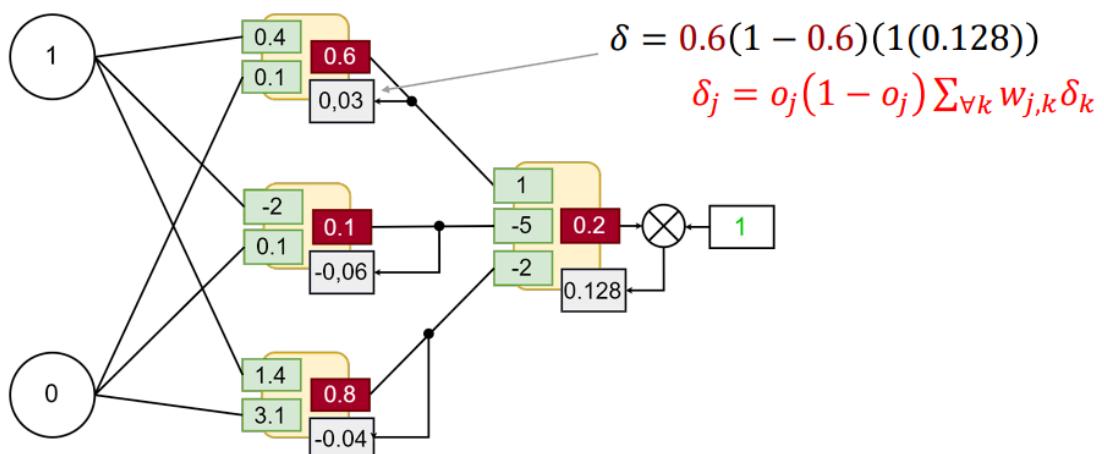
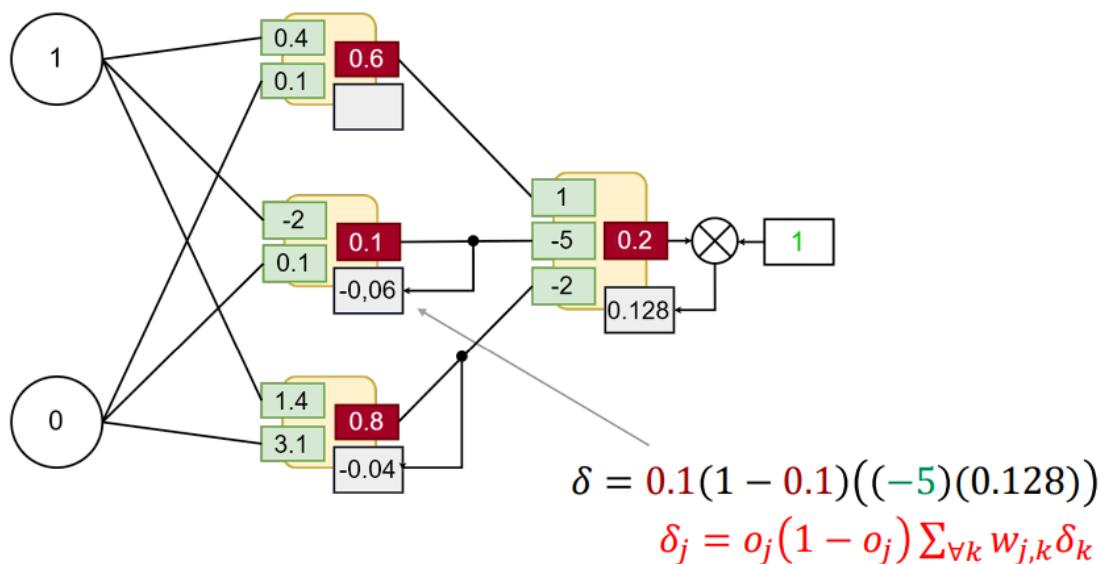
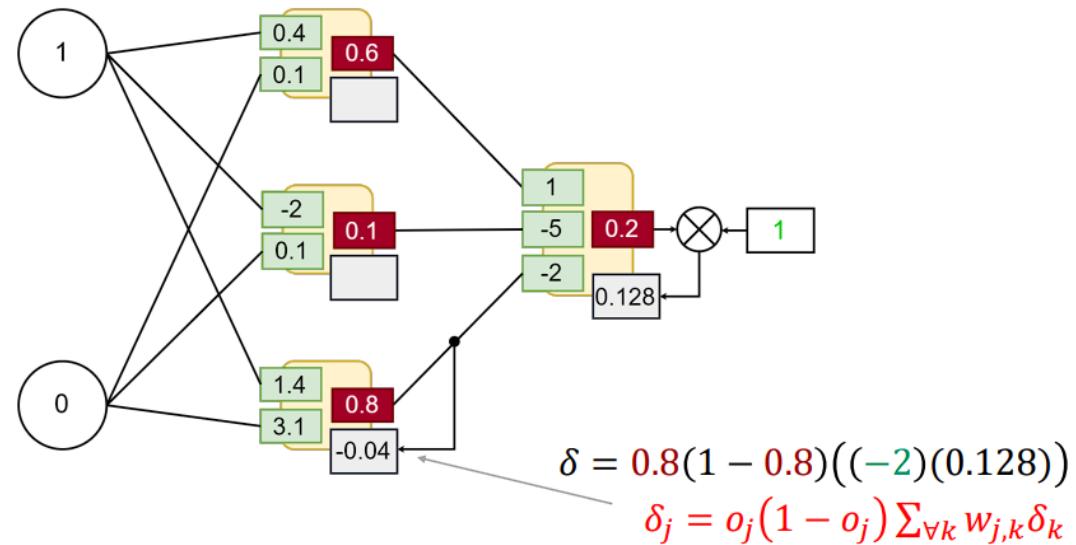


2. Etapa de propagación hacia atrás (*back propagation*)

1. La red compara su salida con la etiqueta
2. Se calcula la **tasa de error** de las neuronas de la capa de salida en función del error cometido por la red.



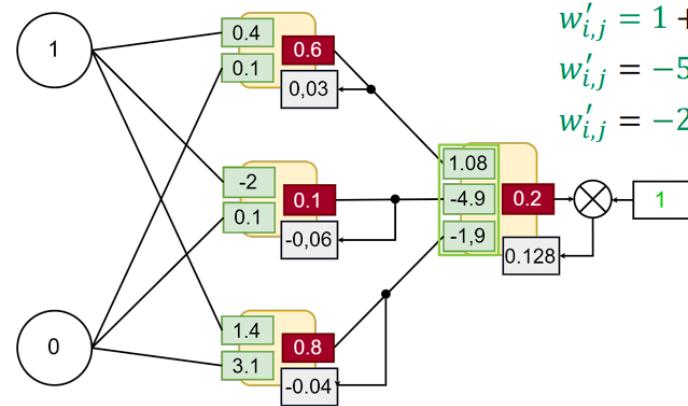
3. Este error se propaga (**back propagation**) a todas las neuronas hasta la capa de entrada.



3. Actualización de los pesos

- Se actualizan los pesos (**gradient descent**) añadiéndoles al peso actual el error correspondiente a su neurona por la entrada de ese peso y multiplicado por la ratio de aprendizaje (learning rate)

X1	X2	Y
1	0	1

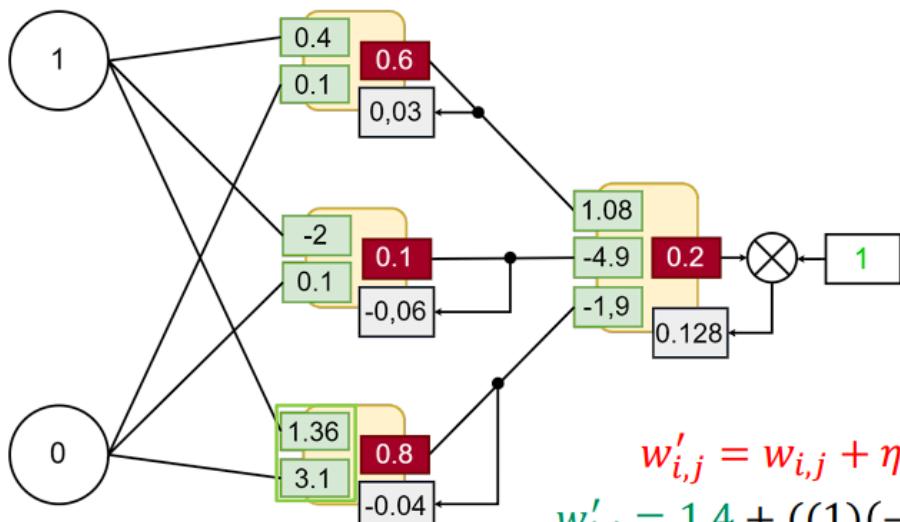


$$w'_{i,j} = w_{i,j} + \eta \delta_j o_i$$

$$w'_{i,j} = 1 + ((1)(0.128)(0.6))$$

$$w'_{i,j} = -5 + ((1)(0.128)(0.1))$$

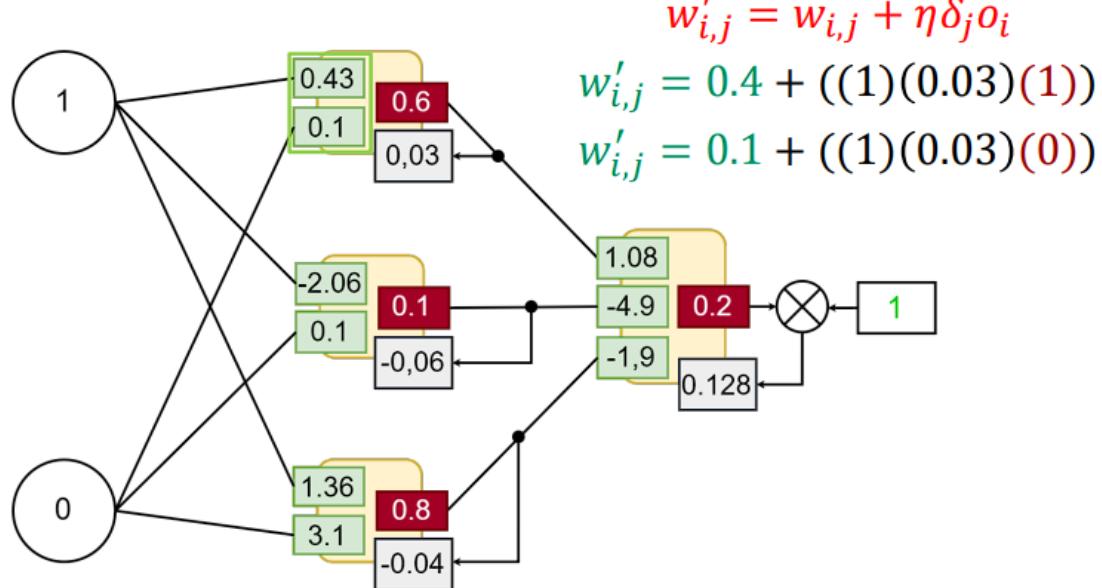
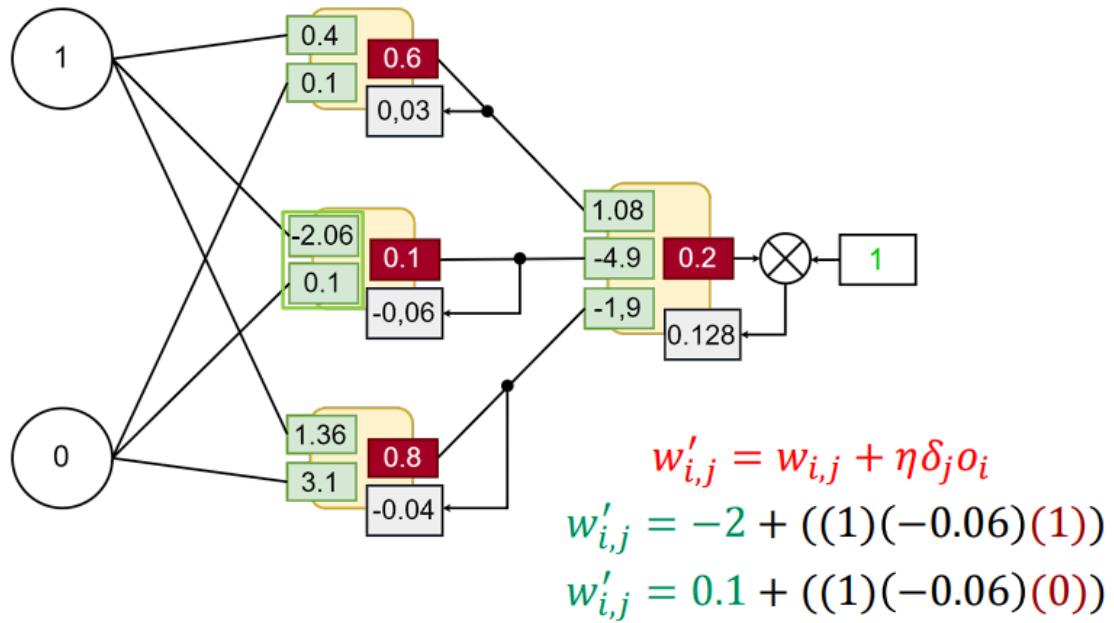
$$w'_{i,j} = -2 + ((1)(0.128)(0.8))$$



$$w'_{i,j} = w_{i,j} + \eta \delta_j o_i$$

$$w'_{i,j} = 1.4 + ((1)(-0.04)(1))$$

$$w'_{i,j} = 3.1 + ((1)(-0.04)(0))$$



VALIDACIÓN DEL MODELO

¿Cómo sabemos si nuestra red neuronal ha aprendido?

Para saber si nuestra red neuronal, nuestro modelo, ha aprendido, tenemos que ponerlo a prueba.

¡Queremos que el modelo aprenda, no memorice!

Por tanto, necesitamos ejemplos que no hayan sido empleados para entrenar el modelo, necesitamos el **conjunto de datos de test**.

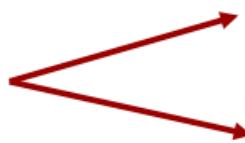
Dividimos el conjunto de datos:

- 90% para entrenar – 10% para testear (ejemplo)

Data set	E1	E2	E3	E4	E5	A1	A2	A3	A4	S1	S2	S3	S4	M1	M2	M3	M4	S	M	Correr	Luchar	Curar	Perseguir	Muntón
1	17	18	29	11	30	10	19	14	5	9	9	4	8	22	39	14	8	7	40	1	0	0	0	0
n	10	8	9	6	17	19	1	14	14	8	7	7	3	39	1	17	21	3	3	0	1	0	0	0

Train set

Data set	E1	E2	E3	E4	E5	A1	A2	A3	A4	S1	S2	S3	S4	M1	M2	M3	M4	S	M	Correr	Luchar	Curar	Perseguir	Muntón
1	17	18	29	11	30	10	19	14	5	9	9	4	8	22	39	14	8	7	40	1	0	0	0	0
n	10	8	9	6	17	19	1	14	14	8	7	7	3	39	1	17	21	3	3	0	1	0	0	0



Data set	E1	E2	E3	E4	E5	A1	A2	A3	A4	S1	S2	S3	S4	M1	M2	M3	M4	S	M	Correr	Luchar	Curar	Perseguir	Muntón
1	17	18	29	11	30	10	19	14	5	9	9	4	8	22	39	14	8	7	40	1	0	0	0	0
n	10	8	9	6	17	19	1	14	14	8	7	7	3	39	1	17	21	3	3	0	1	0	0	0

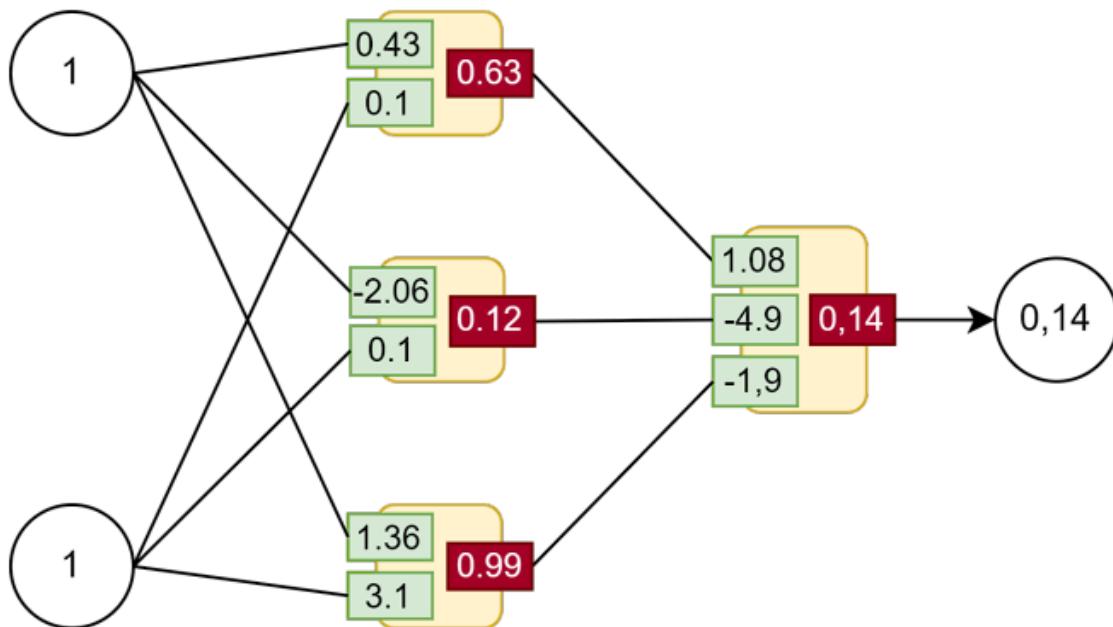
Test set

Usamos el conjunto de datos de test para **medir el error** de nuestro modelo.

INFERENCIA

Una vez entrenada la red, para usarla tan solo tenemos que introducir los datos de entrada y obtener los datos de las neuronas de la capa de salida. Estos datos representan la respuesta/predicción de la red para los datos de entrada.

X1	X2	Y
1	1	0



Derivada del Coste con respecto a los parámetros

¿Cuánto hay que cambiar **cada w** para que el J tenga la mayor bajada?

$$\frac{\text{Variación de Coste}}{\text{Variación de } w_1} = \frac{\text{Variación del Coste}}{\text{Variación de } \hat{Y}} \cdot \frac{\text{Variación de } \hat{Y}}{\text{Variación de } \sigma} \cdot \frac{\text{Variación de } \sigma}{\text{Variación de } w_1}$$

$$\frac{d}{d\hat{Y}} J = \frac{d}{d\hat{Y}} \frac{1}{2} (Y - \hat{Y})^2 = -(Y - \hat{Y}) \quad \text{Error}$$

$$\frac{d}{d\sigma} \hat{Y} = \frac{d}{d\sigma} \frac{1}{1 + e^{-\sigma}} = \frac{e^{-\sigma}}{(1 + e^{-\sigma})^2} = \frac{e^{-\sigma}}{1 + e^{-\sigma}} \cdot \frac{1}{1 + e^{-\sigma}} = (1 - \hat{Y})(\hat{Y}) \quad \text{Salida}$$

$$\frac{d}{dw_1} \sigma = \frac{d}{dw_1} (w_0) + (w_1 x_1) + (w_2 x_2) = x_1 \quad \text{Entrada}$$

Para w_2 los dos primeros serían los mismos y el tercero x_2 , etc.

Tema 3. – Árboles de decisión

3. SUPERVISADO – ÁRBOLES DE DECISIÓN – CLASIFICACIÓN DE ESCENARIOS

ÁRBOLES DE DECISIÓN

Cada **ramificación** (nodo intermedio) representa un **aspecto** del mundo del juego **que debe ser considerado.**

Cada **arista** representa la **decisión tomada** en el nodo donde arranca.

Cada **hoja** (nodo final) representa una **acción** que finalmente se decide.



APRENDIZAJE DE ÁRBOLES DE DECISIÓN

Los árboles de decisión son un método de clasificación.

Por tanto, se trata de **aprendizaje supervisado**.

De modo que se debe **proporcionar un conjunto de ejemplos etiquetados** que en el caso de árboles de decisión es de **observaciones + acciones**.

Por ejemplo:

Player Health	Distance	Behind Cover	Action
High	Close	0	Move
Low	Far	1	Attack
Low	Close	0	Attack
High	Far	0	Move
Low	Far	1	Move

El objetivo es **dividir esta tabla** de modo que **las acciones se agrupen de la manera más eficiente**.

La medida de **eficacia** son la **entropía** y la **ganancia de información**.

Player Health	Enemy Health	Distance	Player Buff	Enemy Buff	Behind Cover	Action
High	Mid	Far	HealBoost	Empty	0	Move
Low	Mid	Far	AttackBoost	DefenseBoost	1	Attack
Mid	Low	Far	DefenseBoost	Empty	0	Move
High	Mid	Far	HealBoost	AttackBoost	0	Heal
...						
High	Mid	Far	HealBoost	Empty	0	Move
Low	Low	Far	Empty	Empty	1	Move

ENTROPÍA

La entropía es una medida de la incertidumbre/información que hay en un conjunto de datos.

$$E(S) = - \sum_{i=1}^C p_i \log_2 p_i$$

Donde:

- $E(S)$ = Entropía del conjunto de datos S
- C = Número de clases diferentes en S
- p_i = Probabilidad/proorción de la clase i respecto del total

Player Health
High
Low
Mid
High
High
Low

$$E(PHealth) = -\left(\frac{3}{6} \log_2 \frac{3}{6} + \frac{2}{6} \log_2 \frac{2}{6} + \frac{1}{6} \log_2 \frac{1}{6}\right) = 1,4591$$

Mas incertidumbre $1,4591 \leftrightarrow 0,6500$ Más información

$$E(Distance) = -\left(\frac{5}{6} \log_2 \frac{5}{6} + \frac{1}{6} \log_2 \frac{1}{6}\right) = 0,65$$

Distance
Far
Far
Far
Far
Close
Far

ENTROPÍA CONDICIONADA

La entropía condicionada es una medida de la incertidumbre/información que hay en un conjunto de datos dado otro conjunto de datos.

Donde:

- $E(S_1|S_2)$ = Entropía de S_1 en función de S_2
- V = Número de clases diferentes en S_2
- $P(S_2 = v_i)$ = Probabilidad del valor v_i en S_2
- $E(S_1|S_2 = v_i)$ = Entropía de S_1 cuando S_2 vale v_i

$$E(S_1|S_2) = \sum_{i=1}^V P(S_2 = v_i) E(S_1|S_2 = v_i)$$

Player Health	Action	Player Health	Action
High	Move	High	Move
Low	Attack	High	Heal
High	Move	High	Attack
High	Heal	High	Move
High	Attack	Low	Attack
Low	Move	Low	Move

↗

$$P(PH = High)E(Action|PH = High) = \frac{4}{6} * \left(-\left(\frac{2}{4}\log_2 \frac{2}{4} + \frac{1}{4}\log_2 \frac{1}{4} + \frac{1}{4}\log_2 \frac{1}{4}\right)\right) = 1$$

$$P(PH = Low)E(Action|PH = Low) = \frac{2}{6} * \left(-\left(\frac{1}{2}\log_2 \frac{1}{2} + \frac{1}{2}\log_2 \frac{1}{2}\right)\right) = 0,33$$

$$E(Action|PHealth) = 1 + 0,33 = 1,33$$

GANANCIA DE INFORMACIÓN

La ganancia de información es una medida de cuánto varía la entropía de un conjunto de datos cuando lo ponemos en función de otro relacionado.

$$G(S_1|S_2) = E(S_1) - E(S_1|S_2)$$

Donde:

- $G(S_1|S_2)$ = Ganancia de información de S_1 en función de S_2
- $E(S_1)$ = Entropía de S_1
- $E(S_1|S_2)$ = Entropía de S_1 en función de S_2

Action
Move
Attack
Move
Heal
Attack
Move

$$E(Action) = -\left(\frac{3}{6}\log_2 \frac{3}{6} + \frac{2}{6}\log_2 \frac{2}{6} + \frac{1}{2}\log_2 \frac{1}{2}\right) = 1,45$$

$$G(Action|PHealth) = 1,45 - 1,33 = 0,1166$$

$$E(Action|PHealth) = 1,33 \text{ (diapositiva anterior)}$$

Player Health	Action
High	Move
Low	Attack
High	Move
High	Heal
High	Attack
Low	Move

ALGORITMO ID3

El siguiente algoritmo fue originalmente diseñado para observaciones y acciones binarias (dos clases), de ahí su nombre (ID = Iterative Dichotomizer), aunque posteriormente se generalizó para soportar atributos multi-clase y numéricos.

Intuición:

1. Calcular la ganancia de energía de la etiqueta con respecto a cada uno de los atributos.
2. Elegir el atributo con la mayor ganancia de energía y generamos un nodo en el árbol.
3. Por cada uno de los valores (clases) de ese atributo, creamos una rama que sale del nodo creado en 2.
4. Para cada rama:

- Generamos una **nueva tabla** seleccionando en la original las filas con el valor asociado a la rama.
- **Descartamos el atributo seleccionado actualmente.**
- **Repetimos el proceso, generando un nuevo árbol que se creará desde esta rama.**

sameLabel: verdadero si todos los ejemplos tienen la misma etiqueta

createLeafNode: crea un nodo hoja con la etiqueta especificada

mostFrequentLabel: etiqueta más frecuente en los datos

maxGain: atributo que presenta la ganancia máxima para los ejemplos

createNode: crea un nuevo nodo con el atributo recibido

possibleValues: posibles valores para el atributo***

valueSamples: ejemplos con el valor recibido en el atributo especificado

addChildNode: añade **newNode** a **node** con una arista etiquetada como **value**

Algoritmo ID3

```

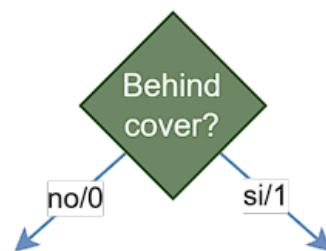
1: procedure ID3(samples, attributes)
2:   if sameLabel samples OR empty attributes then
3:     node ← createLeafNode mostFrequentLabel samples
4:   else
5:     bestAttribute ← maxGain samples, attributes
6:     node ← createNode bestAttribute
7:     for value ∈ possibleValues bestAttribute, samples do
8:       valueSamples ← valueSamples bestAttribute, value, samples
9:       newAttributes ← attributes - bestAttribute
10:      if empty valueSamples then
11:        newNode ← createLeafNode mostFrequentLabel samples
12:      else
13:        newNode ← ID3 valueSamples, newAttributes
14:      end if
15:      addChildNode node, value, newNode
16:    end for
17:  end if
18:  return node
19: end procedure

```

DATOS CATEGÓRICOS

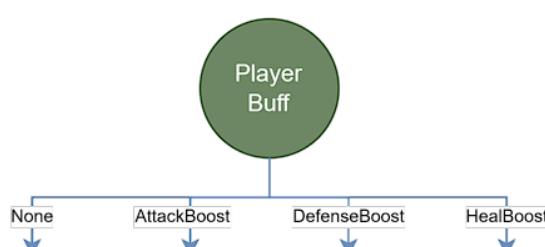
Binarios: dos posibles valores, separación en 2 ramas.

Behind Cover	Class
Yes	1
No	0



Multiclas: múltiples posibles valores, separación en tantas ramas como posibles clases.

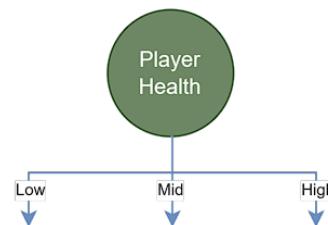
Buff
AttackBoost
DefenseBoost
HealBoost
Empty



DATOS NUMÉRICOS

Convertir en categóricos (manual)

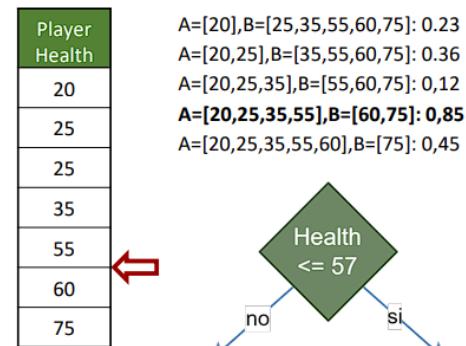
Health	Class
[0,33)	Low
[33,66)	Mid
[66,100)	High



Aprender binarización (2 clases):

Hacer splits: $[v_0, v_t], [v_{t+1}, v_n], v_i \in V = \{v_0, v_1, v_2, \dots, v_n\}$
donde V son los valores **ordenados** del atributo numérico.

Calcular la ganancia de entropía para cada Split.
Elegir el split que mayor entropía ofrezca.



EJEMPLO DE ALGORITMO ID3

Player Health	Enemy Health	Distance	Player Buff	Enemy Buff	Behind Cover	Action
High	Mid	Far	HealBoost	Empty	0	Move
Low	Mid	Close	AttackBoost	DefenseBoost	1	Attack
Mid	Low	Far	DefenseBoost	Empty	0	Move
High	Mid	Close	HealBoost	AttackBoost	0	Heal
High	High	Close	AttackBoost	HealBoost	1	Attack
Low	High	Far	DefenseBoost	HealBoost	0	Move
Mid	Mid	Close	AttackBoost	DefenseBoost	1	Attack
High	Low	Far	DefenseBoost	Empty	0	Move
Low	High	Far	AttackBoost	DefenseBoost	1	Move
Mid	Low	Close	AttackBoost	Empty	0	Move
High	Low	Close	DefenseBoost	HealBoost	1	Defend
High	Low	Far	Empty	Empty	0	Move
Mid	High	Close	HealBoost	DefenseBoost	1	Heal
Low	Low	Close	Empty	Empty	1	Attack
Low	Low	Far	AttackBoost	DefenseBoost	0	Move
Mid	High	Far	Empty	Empty	0	Move
Low	Mid	Close	DefenseBoost	HealBoost	1	Attack
Low	Mid	Close	AttackBoost	DefenseBoost	0	Attack
High	Low	Far	AttackBoost	HealBoost	1	Move
Low	High	Far	DefenseBoost	HealBoost	1	Move
High	High	Close	AttackBoost	HealBoost	0	Attack
Low	Low	Close	AttackBoost	Empty	0	Attack
Mid	High	Close	DefenseBoost	AttackBoost	1	Defend
High	Mid	Far	HealBoost	Empty	0	Move
Low	Low	Far	Empty	Empty	1	Move

Health	Category
[0,33)	Low
[33,66)	Mid
[66,100)	High

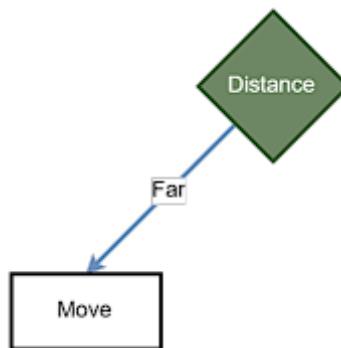
Action
Move
Attack
Heal
Defend

Distance	Class
<10m	Close
= 10m	Far

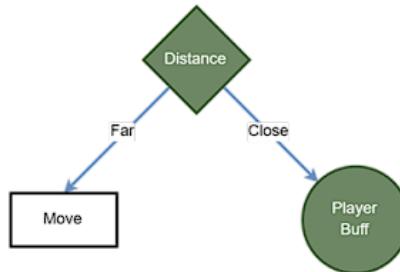
Behind Cover	Class
Yes	1
No	0

Buff
AttackBoost
DefenseBoost
HealBoost
Empty

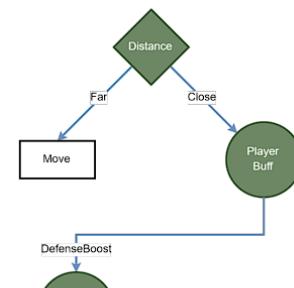
Category	Gain
Distance	0.7953988829
Player Buff	0.5353834205
Enemy Buff	0.4749924203
Enemy Health	0.1908562984
Player Health	0.1726446547
Behind Cover	0.1503710021



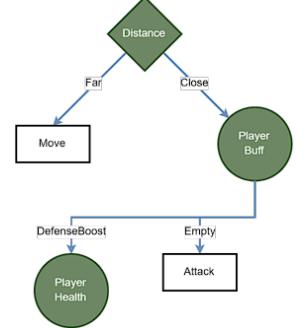
Category	Gain
Player Buff	1.0160861064
Enemy Buff	0.6815849106
Player Health	0.4696704874
Enemy Health	0.34558519893
Behind Cover	0.2200464491



Player Health	Enemy Health	Player Buff	Enemy Buff	Behind Cover	Action
High	Low	DefenseBoost	HealBoost	1	Defend
Low	Mid	DefenseBoost	HealBoost	1	Attack
Mid	High	DefenseBoost	AttackBoost	1	Defend



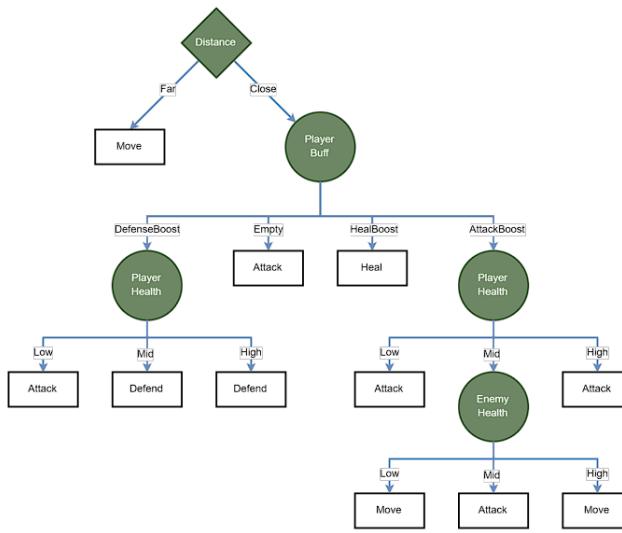
Category	Gain
Player Health	0.9182958341
Enemy Health	0.9182958341
Enemy Buff	0.2516291674
Behind Cover	0



Player Health	Enemy Health	Player Buff	Enemy Buff	Behind Cover	Action
Low	Low	Empty	Empty	1	Attack

RESULTADO FINAL

<https://youtu.be/yeBogcHnZ2A> VÍDEO DE INTERNET CON UN EJEMPLO



ALGORITMO ID4

En vj. es muy raro disponer de un conjunto de observaciones-acciones de un tamaño suficiente como para aprender el árbol de decisión. Especialmente si queremos que aprenda a partir de la interacción con el consumidor.

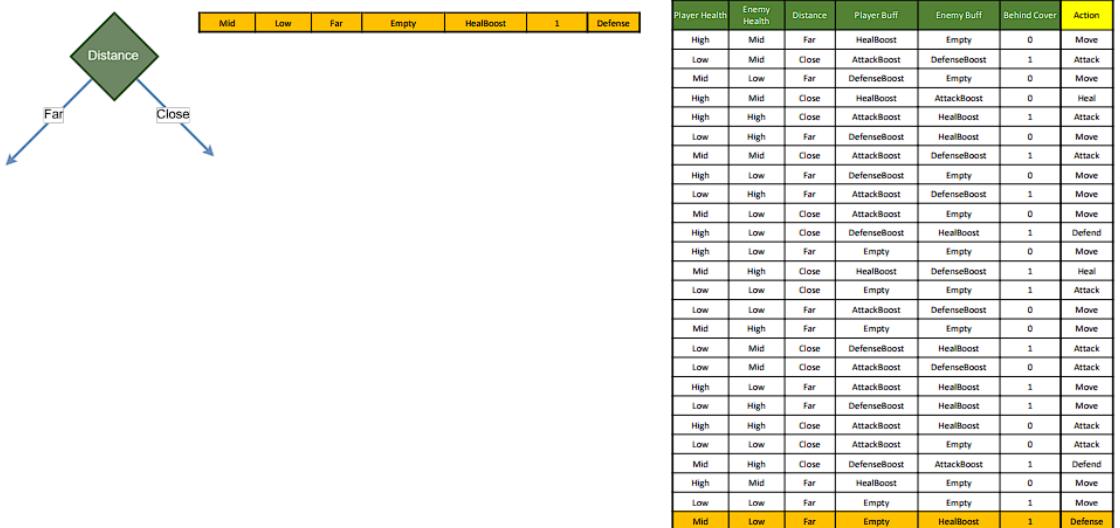
Hace falta un algoritmo que aprenda y **se actualice según llegan nuevos ejemplos**, es decir un algoritmo **on-line** o **incremental**.

El ID4 es la versión incremental del ID3.

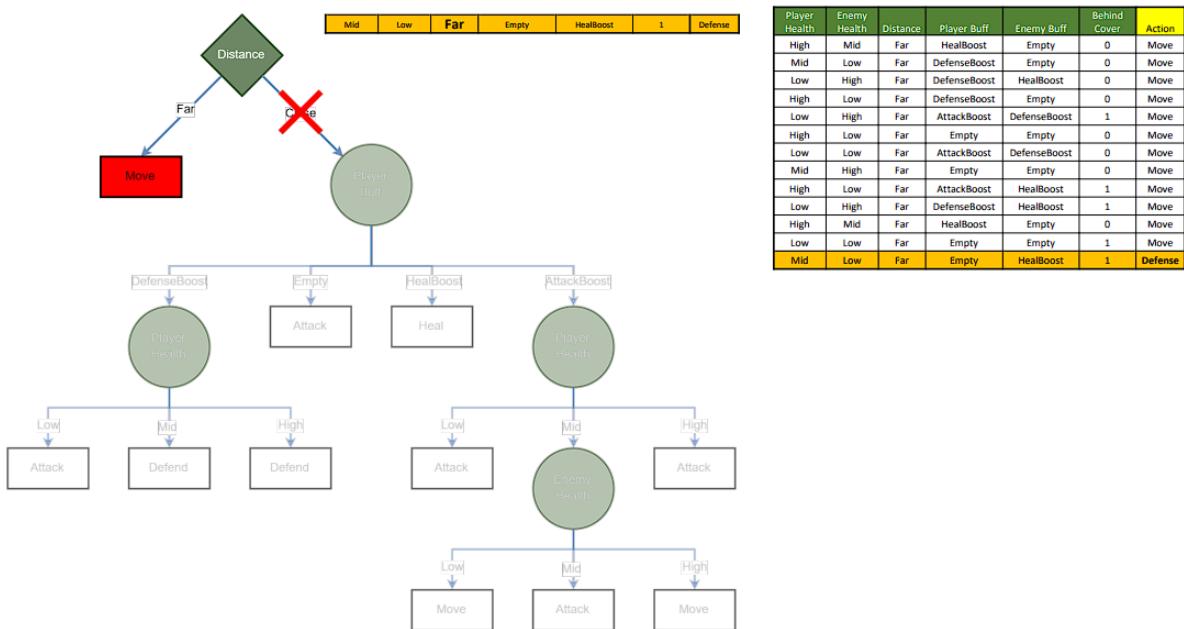
Ideas clave:

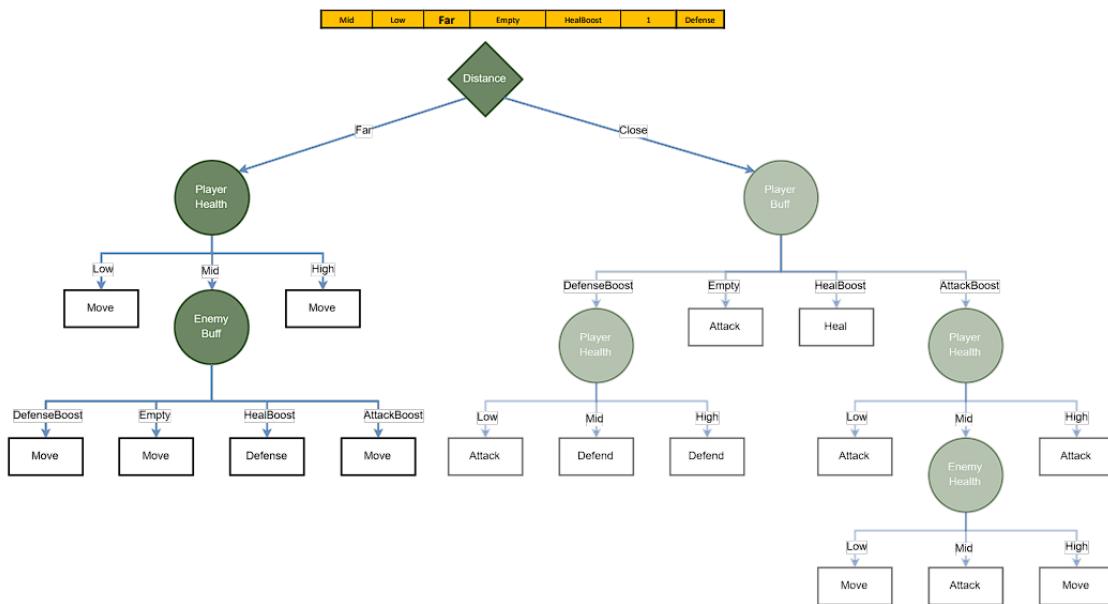
1. Cuando llega un nuevo ejemplo, se recalculan los nodos desde el nodo raíz, manteniendo el mismo árbol hasta que encuentra una **discrepancia**.
2. Cuando se encuentra una **discrepancia**, se **reconstruye** el árbol a partir de ella con el nuevo ejemplo **utilizando ID3**.

EJEMPLO

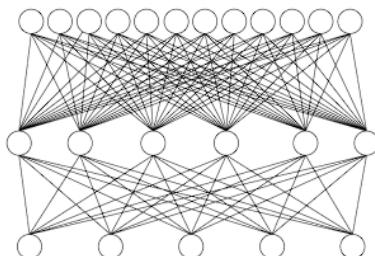
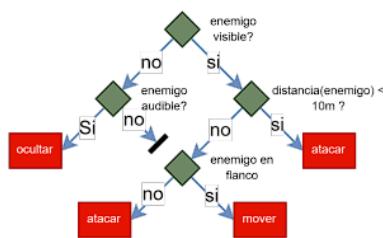


Category	Gain
Distance	0.8043782181
Player Buff	0.6111528063
Enemy Buff	0.5246662649
Enemy Health	0.23276802041
Player Health	0.24190134019
Behind Cover	0.18415910206





ÁRBOLES DE DECISIÓN VS REDES NEURONALES



Árboles de decisión

- Trabajan bien con datos estructurados
- Rápidos de entrenar, rápidos en inferencia
- Difíciles de actualizar con nueva información
- Fácilmente interpretables
- Menor tendencia al overfitting
- Pocos **hiperparámetros**

Redes neuronales

- Trabajan bien con todo tipo de datos, estructurados o no
- Más lentos para entrenar e inferir
- Fácil actualización con nuevos datos
- Difíciles de interpretar
- Mayor tendencia al overfitting
- Muchos **hiperparámetros**

Tema 4. – Predicción con n-gramas

4. NO SUPERVISADO – N-GRAMAS – PREDICCIÓN DE LA SIGUIENTE ACCIÓN

EJEMPLO, PREDICCIÓN DE ACCIONES

Es muy frecuente que un jugador repita un cierto patrón cuando realiza una secuencia de acciones.

Una forma sencilla de dotar a la IA del contrincante de la capacidad de predecir lo que vamos a hacer consiste en:

– **Recordar** acciones anteriores ...

– ... y buscar la **acción más probable** que seguirá en la secuencia actual



Secuencia de acciones

A	B	A	C	C	B	A	C	A	C	B	C	A	B	B	A	C	A	C	B	B	A	C	?
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Acción	Categoría
Saltar	A
Atacar	B
Defender	C

¿Podemos predecir el siguiente movimiento del oponente?

La probabilidad de un suceso en una secuencia de sucesos, se define el número de veces

que ha ocurrido ese suceso dividido el número de sucesos totales.

$$p(e) = \frac{\text{# veces que ocurre el suceso } e}{\text{# de sucesos totales}}$$

¿En la secuencia anterior, cual es la acción más probable?

– Las probabilidades son: $A = \frac{16}{40} = 0,4$, $B = \frac{14}{40} = 0,35$ y $C = \frac{14}{40} = 0,35$

Según esto, el jugador va a Saltar (A) en la siguiente acción.

N-GRAMAS

Los n-gramas son buenos para encontrar patrones dentro de una secuencia de datos.

Los n-gramas tienen un orden n que indica cuantos elementos incluye el patrón, y almacenan la frecuencia (número de ocurrencias) de ese patrón.

1-grama (unígrama)

Patrón	Frecuencia
A	16
B	14
C	14

2-grama (bigrama)

Patrón	Frecuencia
AA	4
AB	3
AC	9
BA	7
BB	3
BC	2
CA	4
CB	6
CC	1

3-grama (trígrama)

Patrón	Frecuencia
AAA	1
AAB	2
AAC	1
ABA	2
ABB	1
ACA	2
ACB	5
ACC	1
BAA	2
BAC	5
BBA	3
BCA	2

PREDICCIÓN CON N-GRAMAS

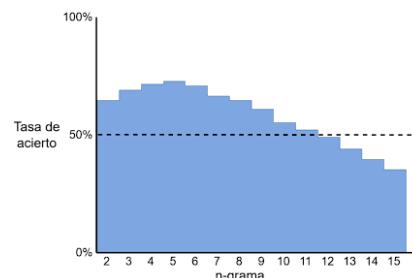
¿Cuál será la siguiente acción?

El **número de acciones** que se toman en cuenta para hacer la predicción se denomina **tamaño de la ventana ($n-1$)**.

	1-grama	2-grama	3-grama
Ventana	Ninguna	C	AC
Patrones	A,B,C	CA, CB, CC	ACA,ACB,ACC
Mejor	A: 16	CB: 6	ACB: 5
Predicción	A	B	B

RENDIMIENTO DEL PREDICTOR N-GRAMA

Se ha comprobado que los mejores predictores n-gramas son aquellos con $n = 3, 4$ o 5 .



Generalmente hacer muy grande n provoca que la tabla de ocurrencias esté muy vacía y no aporte información.

– Para n y p posibilidades por acción habrá pn filas.

- Por ejemplo, la tabla de un 11-grama donde cada acción puede ser A, B,C tendrá 3^{11} filas (117.147 filas).

PREDICTOR N-GRAMA JERÁRQUICO

El problema de este método es que para predecir primero hay que construir la tabla de ocurrencias.

Por tanto, la IA no funciona hasta que se completa (o no funciona bien)

Una solución es correr varios predictores en paralelo, cada uno de ellos con un N distinto.

Cuando se pide una predicción se acude al N-grama de mayor N

Si tiene datos suficientes se toma este resultado,

Si no se pasa al siguiente N-grama, el de (N-1).

Tema 5. – Aprendizaje por refuerzo (Q-learning)

5. POR REFUERZO – Q-LEARNING – APRENDER EN BASE A RECOMPENSAS

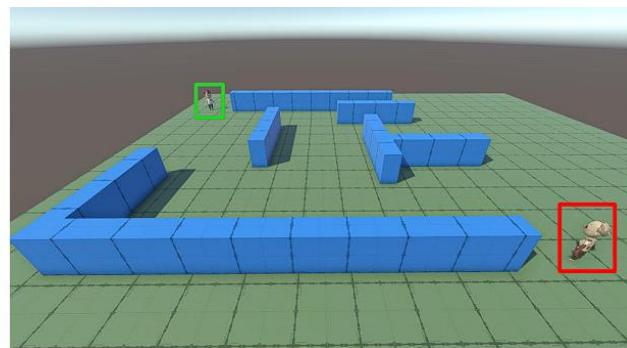
IDEA PRINCIPAL

El **agente inteligente** trata de evitar ser atrapado por el **jugador**.

- El **agente inteligente** conoce el **estado actual** del mundo.
- El **agente inteligente** puede realizar una serie de **acciones** en cada momento.
- Cada acción realizada llevará al **agente inteligente** a un **nuevo estado**.
- ¿Cómo puede **aprender** el **agente inteligente** la **mejor acción a realizar** en cada estado?

Daremos **recompensas** al **agente inteligente** cuando su acción lo lleve a un **estado mejor**, y lo **penalizaremos** cuando le lleve a un **estado peor**.

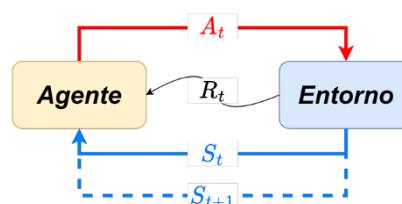
APRENDIZAJE POR REFUERZO



ELEMENTOS DEL PROBLEMA

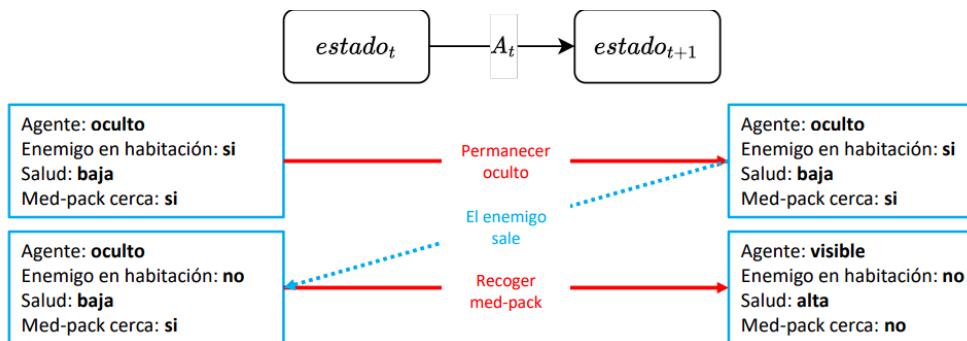
Para resolver un problema con aprendizaje por refuerzo lo formulamos como la interacción de un entorno y un agente.

- En un instante actual t ,
- el **agente** observa el **estado actual del entorno** S_t y decide realizar una **acción** A_t ,
- el **entorno** reacciona a la acción A_t y cambia al **estado siguiente** S_{t+1} ,
- el **agente** recibe una **recompensa** R_t del **entorno** según la **acción** realizada A_t y lo buena que es la transición de estados $S_t \rightarrow S_{t+1}$.



REPRESENTACIÓN DEL ENTORNO

El **entorno** en un instante en el tiempo define el **estado** en que el agente (y el resto del mundo) se encuentran en ese instante del tiempo. Cuando el agente realiza una **acción**, el **entorno** cambia y por lo tanto cambia también el **estado** que lo representa.



Nosotros elegimos aquello que conforma el estado.

Si algo NO se incluye en el estado, no se posible aprender de ello.

El entorno suele ser estocástico.

LA TABLA Q

La **Tabla Q** contiene el conocimiento del agente inteligente. Sus valores representan la **recompensa** total obtenida al ejecutar cada una de las **acciones** (columnas) en cada uno de los **estados** (filas). A este valor se le denomina valor ***Q*** (*s, a*) donde *s* es el estado y *a* es la acción.

APRENDER supone establecer los valores ***Q*** de la tabla en función de la recompensa obtenida al ejecutar cada una de las acciones en cada uno de los estados posibles.

INFERRIR consiste en localizar el estado actual en la tabla y realizar la acción con mayor valor ***Q***.

	Permanecer oculto	Recoger med-pack	Avanzar posición
S1	-10	5	10
S2	4	100	20
S3	-20	-10	10
S4	90	20	30

Debemos categorizar los diferentes estados.

Normalmente cada estado posible se codifica/enumera para indexar la tabla.

El número de filas corresponderá con todos los posibles estados s para los que queramos aprender cual es la mejor acción a . La tabla tendrá $s \times aa$ celdas. **El tamaño de la tabla tiene un alto impacto en el aprendizaje.**

The diagram illustrates the process of creating a Q-table. It starts with four categorical feature tables (Visible, Enemigo en habitación, Salud, Med-pack cerca) which map to a numerical state space (S1 to S24). This numerical state space then maps to a Q-table with columns for actions A1, A2, and A3.

Visible	Categorizado	Enemigo en habitación	Categorizado	Salud	Categorizado	Med-pack cerca	Categorizado	Estado
Sí	1	0	0	0	0	0	0	S1
No	0	0	0	0	1	0	1	S2
		0	0	1	0	0	1	S3
		0	0	1	1	1	1	S4
		0	0	2	0	0	0	S5
		0	0	2	1	1	2	S6
		0	1	0	0	0	0	S7
		0	1	0	1	0	1	S8
		0	1	1	0	0	0	S9
		0	1	1	1	1	1	S10
		0	1	2	0	0	0	S11
		0	1	2	1	1	2	S12
		0	0	0	0	0	0	S13
		1	0	0	0	1	0	S14
		1	0	1	0	0	1	S15
		1	0	1	1	1	1	S16
		1	0	2	0	0	0	S17
		1	0	2	1	1	2	S18
		1	1	0	0	0	0	S19
		1	1	0	0	1	0	S20
		1	1	1	0	0	1	S21
		1	1	1	1	1	1	S22
		1	1	2	0	0	2	S23
		1	1	2	1	1	2	S24

	A1	A2	A3
S1	0	61	33
S2	87	36	46
S3	-4	53	21
S4	1	-7	95
S5	93	12	39
S6	63	98	-8
S7	95	55	8
S8	95	8	60
S9	3	16	67
S10	17	44	-1
S11	74	27	23
S12	44	37	74
S13	85	97	97
S14	27	41	7
S15	34	88	67
S16	51	32	91
S17	61	71	49
S18	84	61	70
S19	49	1	82
S20	37	99	4
S21	25	5	71
S22	-1	56	4
S23	78	59	7
S24	44	-6	11

APRENDIZAJE DEL AGENTE

Simulamos el entorno para el agente, y dejamos que este realice acciones, calculando la recompensa de las acciones y actualizando la Tabla Q . Repetimos este proceso en bucle durante suficientes simulaciones (**episodios**) para que el agente inteligente gane experiencia.

El proceso de aprendizaje tiene 3 componentes:

- **Estrategia de exploración.** El agente puede probar acciones nuevas o repetir acciones conocidas
- **Función de refuerzo.** Nos informa de la calidad de nuestras acciones Las recompensas (refuerzos positivos) tienen dos funciones:
 - Alcanzar la meta
 - Valorar la bondad intrínseca del estado en el que estamos.

Los refuerzos negativos, en caso de que se quieran utilizar, se reciben cuando

- Perdemos el juego o muere nuestro avatar.
- Cuando realizamos una acción no deseada

- **Regla de aprendizaje.** Para actualizar los parámetros del agente según avanza el entrenamiento

ALGORITMO Q-LEARNING

Parámetros:

- α : **learning rate**, cuanto aprende nuestro algoritmo con cada nueva simulación
- γ : **discount factor**, que parte de la recompensa futura tomamos
- ϵ : **exploration rate**, cuanto exploramos nuevas opciones

Funciones:

- **randomState** devuelve un estado aleatorio
- **selectAction** selecciona una acción a realizar, según el **exploration rate**
- **updateWithReward** actualiza el valor actual de $Q(s, a)$
- **terminal** verdadero si se ha alcanzado el final del episodio

Algoritmo Q-Learning greedy

```

1: procedure Q-LEARNING
2:    $\alpha \leftarrow \text{learning rate}$ 
3:    $\gamma \leftarrow \text{discount factor}$ 
4:    $\epsilon \leftarrow \text{exploration rate}$ 
5:   for each episode do
6:      $s \leftarrow \text{randomState}$ 
7:     repeat
8:        $a \leftarrow \text{selectAction possibleActions}, \epsilon$ 
9:       updateWithReward  $Q(s, a), \alpha, \gamma$ 
10:      until terminal  $s$ 
11:    end for
12:  end procedure

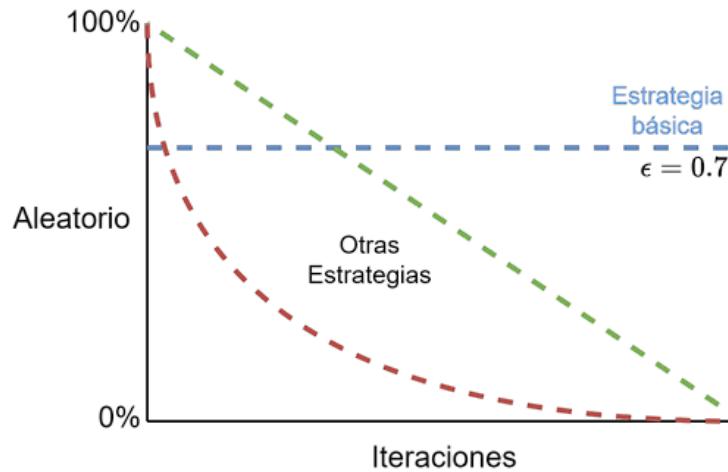
```

ESTRATEGIA DE EXPLORACIÓN

La estrategia de exploración es la política seguida para seleccionar las acciones que se realizan en cada estado.

- La estrategia básica es parcialmente aleatoria, basada en ϵ ; es decir:
 - A veces ($\text{random} \leq \epsilon$) elegimos **una acción aleatoria**
 - A veces ($\text{random} > \epsilon$) elegimos la acción con mayor valor **Q para el estado actual**

- Una estrategia más elaborada consiste en controlar el grado de aleatoriedad según avanza el tiempo.



REGLA DE APRENDIZAJE

$$Q'^{(s,a)} = (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{\forall a'} Q(s', a'))$$

Significado:

- $Q'(s, a)$ es el nuevo valor Q en el estado s realizando la acción a
- $(1 - \alpha)Q(s, a)$ es la fracción del valor Q anterior que mantenemos
- $\alpha(r + \gamma \max_{\forall a'} Q(s', a'))$ es la fracción que “aprendemos”, que se calcula con:
 - r es la **recompensa** recibida por esta acción
 - $\gamma \max_{\forall a'} Q(s', a')$ es una fracción de la recompensa que obtendremos en la **siguiente acción**. Mejorará la recompensa en función de cuanto nos acerca a la meta esta acción.

RECOMPENSA

Cómo elegir las recompensas

El diseño de la recompensa es una pieza clave del aprendizaje por refuerzo

En general:

NO es recomendable dar recompensas parciales, o sea pequeños “premios” por hacer cosas no malas.

El algoritmo se suele quedar estancado, acumulando esas recompensas pequeñas.

Sí es recomendable dar grandes recompensas positivas cuando se alcanza el objetivo y recompensas negativas cuando se llega a un punto de no retorno. Si hay más de un estado meta, se puede dar la misma recompensa (es decir no hace falta dividirla entre dos)

- *Ejemplo*

- Dar recompensa 100 a todo par (s, a) de la tabla Q que nos lleva a la meta deseada.
- Dar recompensa 0 a todo par (s, a) que simplemente nos lleva a una transición a otro estado.
- Dar castigo (recompensa negativa) -1 a todo par que nos lleva a una situación dañina e irreversible, como muerte; o bien a pares (s, a) que son imposibles, por ejemplo, aquellos que implican atravesar paredes.

EJEMPLO

Queremos que un agente aprenda por refuerzo a moverse por una casa **para salir de ella**.

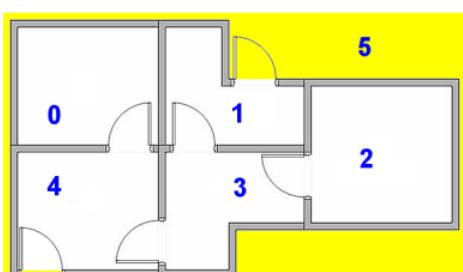
El plano de la casa se muestra a la derecha, donde la zona amarilla es el objetivo a lograr.

Podemos asociar cada hab. a un estado

Las acciones posibles para el agente en cada habitación son ir la **Norte, Sur, Este y Oeste**

Utilizamos la siguiente **política de recompensas**:

- Si logro llegar al exterior = 100
- Si no logro llegar al exterior = 0
- Si intento atravesar una pared = -1



Controles del algoritmo:

- $\alpha = 0.3, \gamma = 0.8$
- $\epsilon = 1.0 \rightarrow$ Exploración 100% siempre

Tabla Q inicializada a 0 en todas sus celdas.

	N	S	E	O
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0

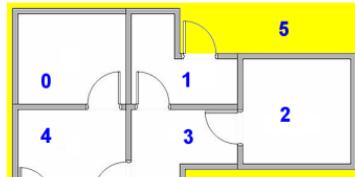
Episodio 1 – Paso 1

Estado inicial: **1** (aleatorio)

Acción: **N** (aleatoria) – Estado final: **5**

Recompensa: **100**

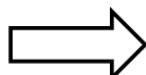
Mejor futuro: **0** ($\max_{s=5} Q$)



$$\alpha = 0.3, \gamma = 0.8, \epsilon = 1.0$$

$$Q(1, N) = (0.7)(0) + (0.3)[100 + (0.8)(0)] = 30.0$$

	N	S	E	O
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0



	N	S	E	O
0	0	0	0	0
1	30.0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0

Fin del episodio 1

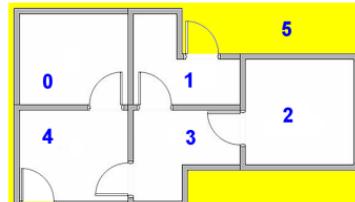
Episodio 2 – Paso 1

Estado inicial: **3** (aleatorio)

Acción: **N** (aleatoria) – Estado final: **1**

Recompensa: **0**

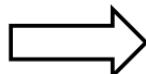
Mejor futuro: **30.0** ($\max_{s=1} Q$)



$$\alpha = 0.3, \gamma = 0.8, \epsilon = 1.0$$

$$Q(3, N) = (0.7)(0) + (0.3)[0 + (0.8)(30.0)] = 7.2$$

	N	S	E	O
0	0	0	0	0
1	30.0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0



	N	S	E	O
0	0	0	0	0
1	30.0	0	0	0
2	0	0	0	0
3	7.2	0	0	0
4	0	0	0	0
5	0	0	0	0

Episodio 2 – paso 2

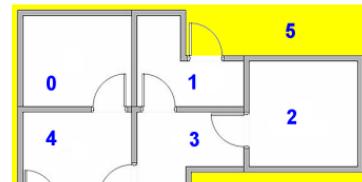
Episodio 2 – Paso 2

Estado inicial: 1 (paso anterior)

Acción: S (aleatoria) – Estado final: 3

Recompensa: 0

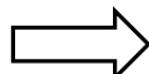
Mejor futuro: 7.2 (max $Q_{s=3}$)



$$\alpha = 0.3, \gamma = 0.8, \epsilon = 1.0$$

$$Q(1, S) = (0.7)(0) + (0.3)[0 + (0.8)(7.2)] = 1.7$$

	N	S	E	O
0	0	0	0	0
1	30.0	0	0	0
2	0	0	0	0
3	7.2	0	0	0
4	0	0	0	0
5	0	0	0	0



	N	S	E	O
0	0	0	0	0
1	30.0	1.7	0	0
2	0	0	0	0
3	7.2	0	0	0
4	0	0	0	0
5	0	0	0	0

Episodio 2 – paso 3

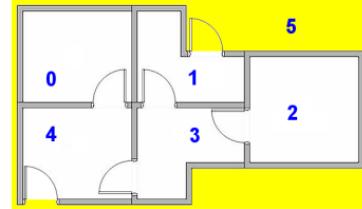
Episodio 2 – Paso 3

Estado inicial: 3 (paso anterior)

Acción: O (aleatoria) – Estado final: 4

Recompensa: 0

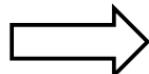
Mejor futuro: 0 (max $Q_{s=4}$)



$$\alpha = 0.3, \gamma = 0.8, \epsilon = 1.0$$

$$Q(3, O) = (0.7)(0) + (0.3)[0 + (0.8)(0)] = 0$$

	N	S	E	O
0	0	0	0	0
1	30.0	1.7	0	0
2	0	0	0	0
3	7.2	0	0	0
4	0	0	0	0
5	0	0	0	0



	N	S	E	O
0	0	0	0	0
1	30.0	1.7	0	0
2	0	0	0	0
3	7.2	0	0	0
4	0	0	0	0
5	0	0	0	0

Episodio 2 – paso 4

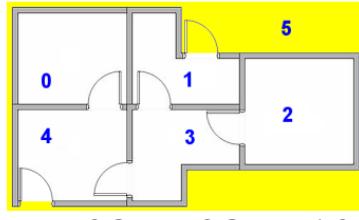
Episodio 2 – Paso 4

Estado inicial: **4** (paso anterior)

Acción: **S** (aleatoria) – Estado final: **5**

Recompensa: **100**

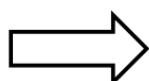
Mejor futuro: **0** ($\max_{s=4} Q$)



$$\alpha = 0.3, \gamma = 0.8, \epsilon = 1.0$$

$$Q(4, S) = (0.7)(0) + (0.3)[100 + (0.8)(0)] = 30.0$$

	N	S	E	O
0	0	0	0	0
1	30.0	1.7	0	0
2	0	0	0	0
3	7.2	0	0	0
4	0	0	0	0
5	0	0	0	0



	N	S	E	O
0	0	0	0	0
1	30.0	1.7	0	0
2	0	0	0	0
3	7.2	0	0	0
4	0	30.0	0	0
5	0	0	0	0

Fin del episodio 2

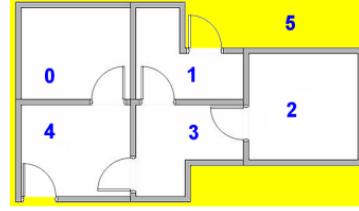
Episodio 3 – Paso 1

Estado inicial: **2** (aleatorio)

Acción: **O** (aleatoria) – Estado final: **3**

Recompensa: **0**

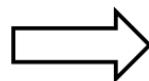
Mejor futuro: **7.2** ($\max_{s=3} Q$)



$$\alpha = 0.3, \gamma = 0.8, \epsilon = 1.0$$

$$Q(2, O) = (0.7)(0) + (0.3)[0 + (0.8)(7.2)] = 1.7$$

	N	S	E	O
0	0	0	0	0
1	30.0	1.7	0	0
2	0	0	0	0
3	7.2	0	0	0
4	0	30.0	0	0
5	0	0	0	0



	N	S	E	O
0	0	0	0	0
1	30.0	1.7	0	0
2	0	0	0	1.7
3	7.2	0	0	0
4	0	30.0	0	0
5	0	0	0	0

Episodio 3 – paso 2

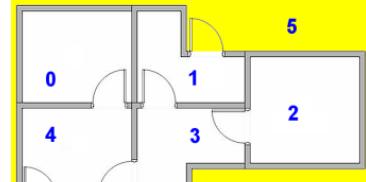
Episodio 3 – Paso 2

Estado inicial: **3** (paso anterior)

Acción: **N** (aleatoria) – Estado final: **1**

Recompensa: **0**

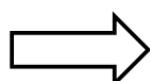
Mejor futuro: **30** ($\max_{s=1} Q$)



$$\alpha = 0.3, \gamma = 0.8, \epsilon = 1.0$$

$$Q(3, N) = (0.7)(7.2) + (0.3)[0 + (0.8)(30)] = 12.3$$

	N	S	E	O
0	0	0	0	0
1	30.0	1.7	0	0
2	0	0	0	1.7
3	7.2	0	0	0
4	0	30.0	0	0
5	0	0	0	0



	N	S	E	O
0	0	0	0	0
1	30.0	1.7	0	0
2	0	0	0	1.7
3	12.3	0	0	0
4	0	30.0	0	0
5	0	0	0	0

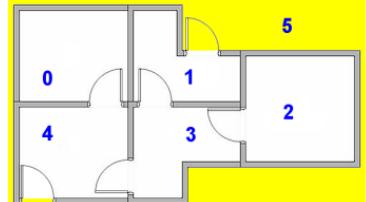
Episodio 3 – paso 3

Estado inicial: **1** (paso anterior)

Acción: **N** (aleatoria) – Estado final: **5**

Recompensa: **100**

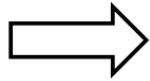
Mejor futuro: **0** ($\max_{s=5} Q$)



$$\alpha = 0.3, \gamma = 0.8, \epsilon = 1.0$$

$$Q(1, N) = (0.7)(30) + (0.3)[100 + (0.8)(0)] = 51.0$$

	N	S	E	O
0	0	0	0	0
1	30.0	1.7	0	0
2	0	0	0	1.7
3	12.3	0	0	0
4	0	30.0	0	0
5	0	0	0	0



	N	S	E	O
0	0	0	0	0
1	51.0	1.7	0	0
2	0	0	0	1.7
3	12.3	0	0	0
4	0	30.0	0	0
5	0	0	0	0

Fin del episodio 3

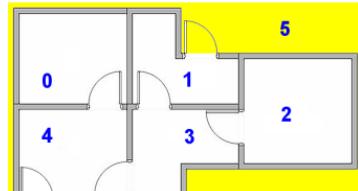
Episodio 4 – Paso 1

Estado inicial: **0** (aleatorio)

Acción: **S** (aleatoria) – Estado final: **4**

Recompensa: **0**

Mejor futuro: **30** ($\max_{s=4} Q$)



$$\alpha = 0.3, \gamma = 0.8, \epsilon = 1.0$$

$$Q(0, S) = (0.7)(0) + (0.3)[0 + (0.8)(30)] = 7.2$$

	N	S	E	O
0	0	0	0	0
1	51.0	1.7	0	0
2	0	0	0	1.7
3	12.3	0	0	0
4	0	30.0	0	0
5	0	0	0	0



	N	S	E	O
0	0	7.2	0	0
1	51.0	1.7	0	0
2	0	0	0	1.7
3	12.3	0	0	0
4	0	30.0	0	0
5	0	0	0	0

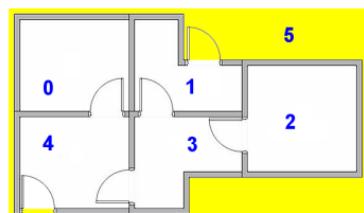
Episodio 4 – paso 2

Estado inicial: **4** (paso anterior)

Acción: **S** (aleatoria) – Estado final: **5**

Recompensa: **100**

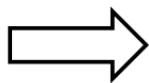
Mejor futuro: **0** ($\max_{s=5} Q$)



$$\alpha = 0.3, \gamma = 0.8, \epsilon = 1.0$$

$$Q(0, S) = (0.7)(30) + (0.3)[100 + (0.8)(0)] = 51.0$$

	N	S	E	O
0	0	0	0	0
1	51.0	1.7	0	0
2	0	0	0	1.7
3	12.3	0	0	0
4	0	30.0	0	0
5	0	0	0	0



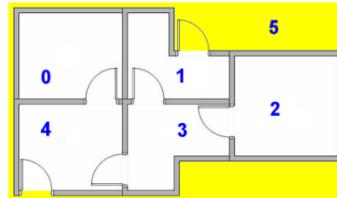
	N	S	E	O
0	0	7.2	0	0
1	51.0	1.7	0	0
2	0	0	0	1.7
3	12.3	0	0	0
4	0	51.0	0	0
5	0	0	0	0

Fin del episodio 4

Resultado

Tabla Q tras 4 episodios

	N	S	E	O
0	0	7.2	0	0
1	51.0	1.7	0	0
2	0	0	0	1.7
3	12.3	0	0	0
4	0	51.0	0	0
5	0	0	0	0



Recorridos aprendidos			
0	4	5	
1	5		
2	3	1	5
3	1	5	
4	5		
5			

Ajustes del algoritmo

Control α : Ritmo de aprendizaje $\in (0,1)$

- Influencia de $Q(s, a)$.
 - ✓ Si $\alpha = 0$ los valores Q no cambian \Rightarrow No aprendemos nada
 - ✓ Si $\alpha = 1$ no tenemos en cuenta el valor Q almacenado \Rightarrow Tiramos a la basura la *experiencia*

Control γ : Ratio de descuento $\in (0,1)$

- Influencia de $Q(s', a')$.
 - ✓ Si $\gamma = 0$ solo añadimos la recompensa
 - ✓ Si $\gamma = 1$ significa que el próximo estado es igual de importante que la recompensa.
 - ✓ Los valores altos favorecen a las secuencias largas, pero tarda más en aprender.

Control ϵ : Aleatoriedad de la exploración $\in (0,1)$

- Controla con qué frecuencia la acción elegida es aleatoria, frente a la mejor posible.
 - ✓ Controla la posibilidad de aprender nuevas cosas.
 - ✓ Si, por ejemplo, $\epsilon = 0.3$, 3 de cada 10 acciones serán aleatorias.
 - ✓ Cuando estamos en aprendizaje **on-line** es preferible valores bajos, por ejemplo $\rho = 0.1$.

Control ν : Factor para conectar acciones seguidas $\in (0,1)$

- Nuevo**
- ✓ Si $\nu = 0$ el algoritmo utiliza el estado al que ha llegado para continuar
 - ✓ Si $\nu = 1$ cada iteración comienza en un estado aleatorio