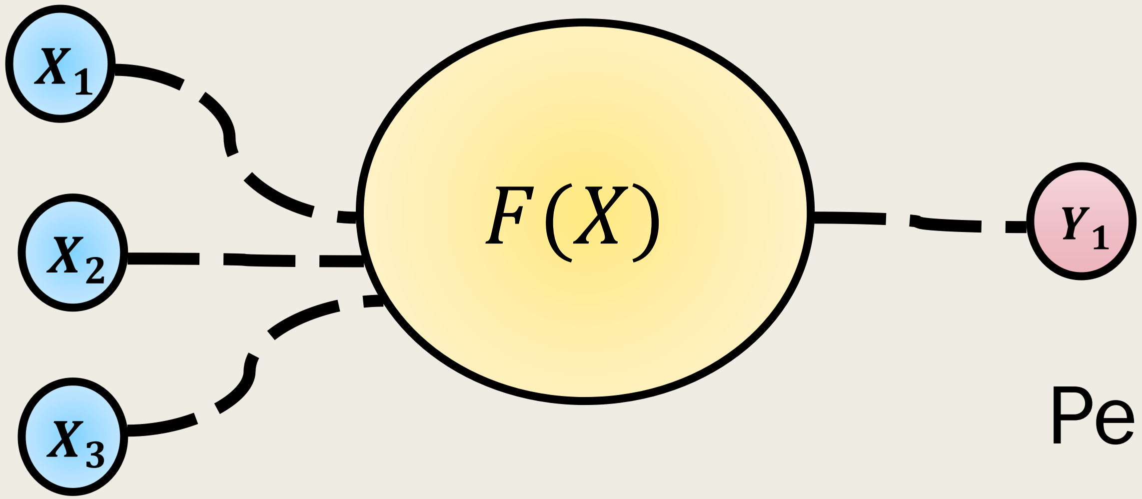
ANALÍTICA AVANZADA DE DATOS: REDES NEURONALES

A. Alejandra Sánchez Manilla
asanchezm.q@gmail.com

NEURONA PERCEPTRÓN

Neurona:

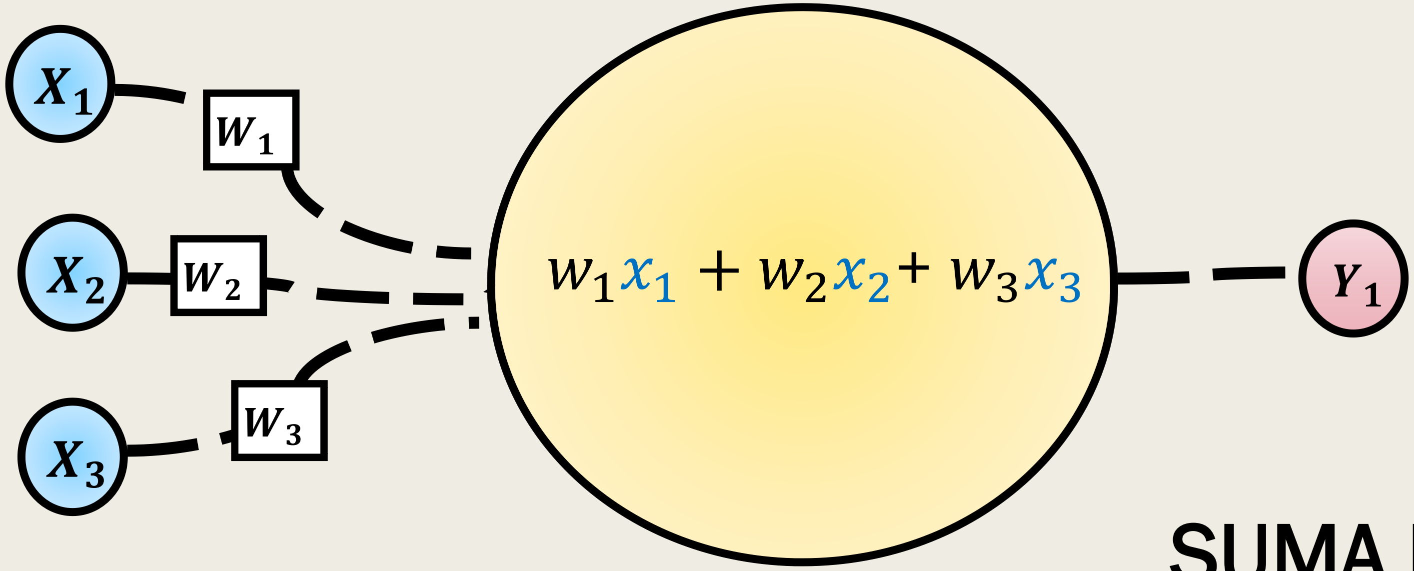
- Unidad básica de procesamiento de las redes neuronales, tiene conexiones de entrada a través de los que reciben estímulos externos, es decir, valores de entrada.
- Con estos valores la neurona realizará un cálculo interno y generará un valor de salida.



Perceptron

Neurona:

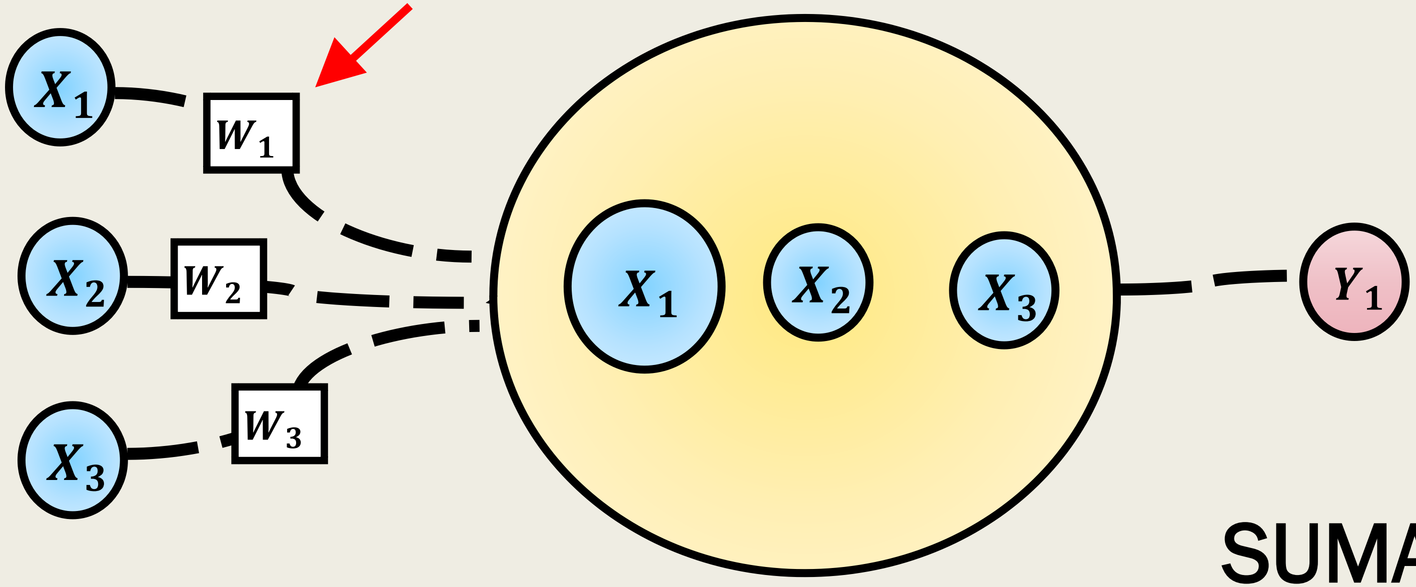
- Internamente, la neurona utiliza todos los valores de entrada para realizar una suma ponderada de ellos.
- La ponderación de cada una de las entradas viene dada por el peso que se le asigna a cada una de las conexiones



SUMA PONDERADA

Neurona:

- Es decir, cada conexión que llega a nuestra neurona tendrá asociado un valor que servirá para definir con qué intensidad cada variable de entrada afecta a la neurona



SUMA PONDERADA

Neurona:

- El perceptrón simple se puede representar como un modelo matemático con varias entradas, un conjunto de pesos y un umbral. Las entradas se multiplican por sus respectivos pesos y se suma el resultado
- Luego, se compara el resultado con el umbral, y si es mayor o igual, la salida es 1, de lo contrario, la salida es 0

Neurona:

Ventajas:

- Es fácil de implementar, adecuado para problemas de clasificación binaria
- El algoritmo de entrenamiento es rápido, lo que permite entrenarlo con grandes cantidades de datos
- Es altamente interpretable, es decir, los pesos y umbrales pueden ser analizados para comprender como el modelo toma decisiones

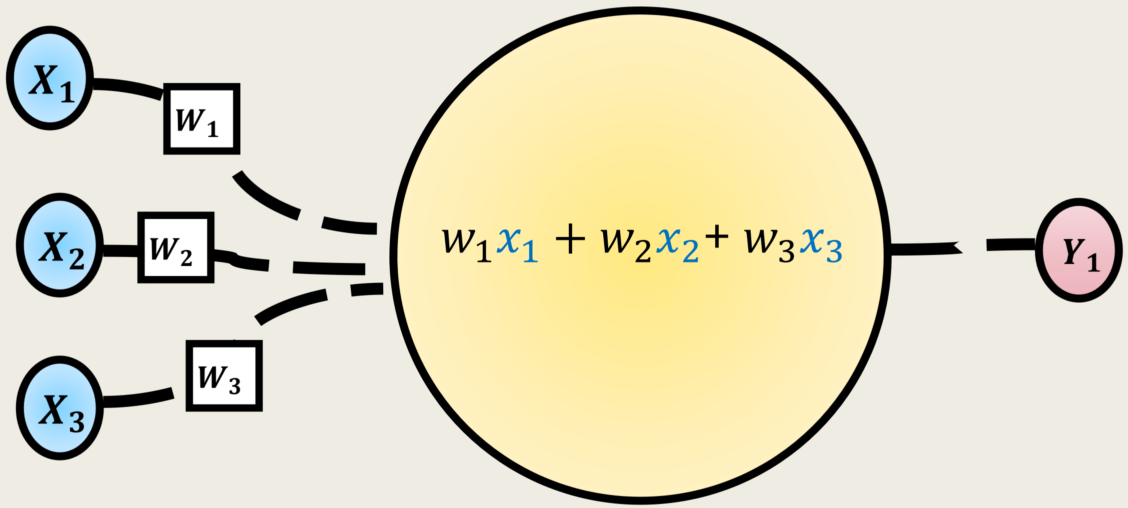
Neurona:

Desventajas:

- Sólo puede resolver problemas de clasificación binaria lineales, lo que limita su capacidad
- El algoritmo de entrenamiento solo converge si los datos son **linealmente separables**
- Es sensible a valores atípicos en los datos, afectando la capacidad del modelo para generalizar

Neurona:

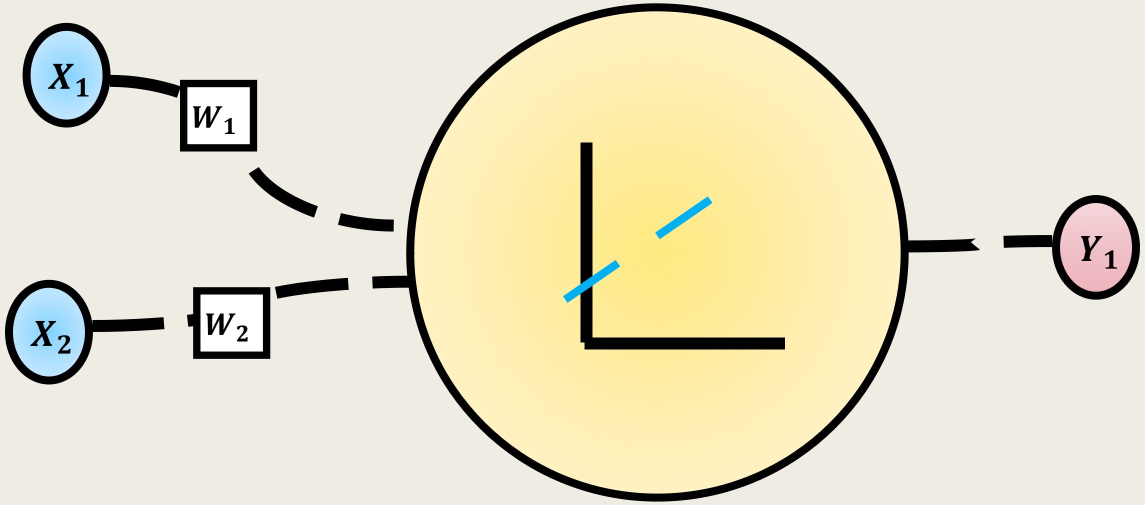
- Como ven, esto es muy parecido a la regresión lineal



$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_3$$

Neurona:

- Otra forma de verlo: internamente es un modelo de regresión lineal

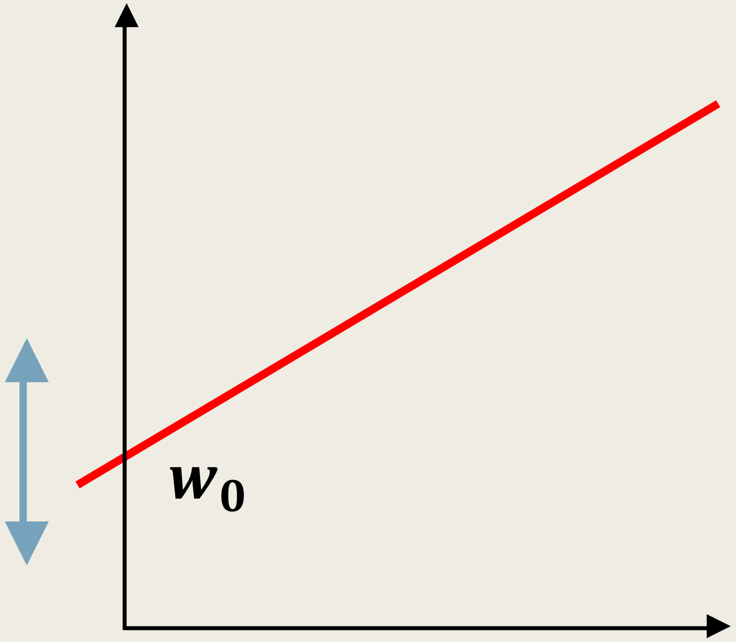


Variables de entrada:
Definen una recta a la que podemos variar la inclinación utilizando nuestros parámetros

$$y = w_1x_1 + w_2x_2 + w_3x_3$$

Neurona:

- En la **regresión lineal** tenemos un término independiente que nos sirve para mover verticalmente a la recta



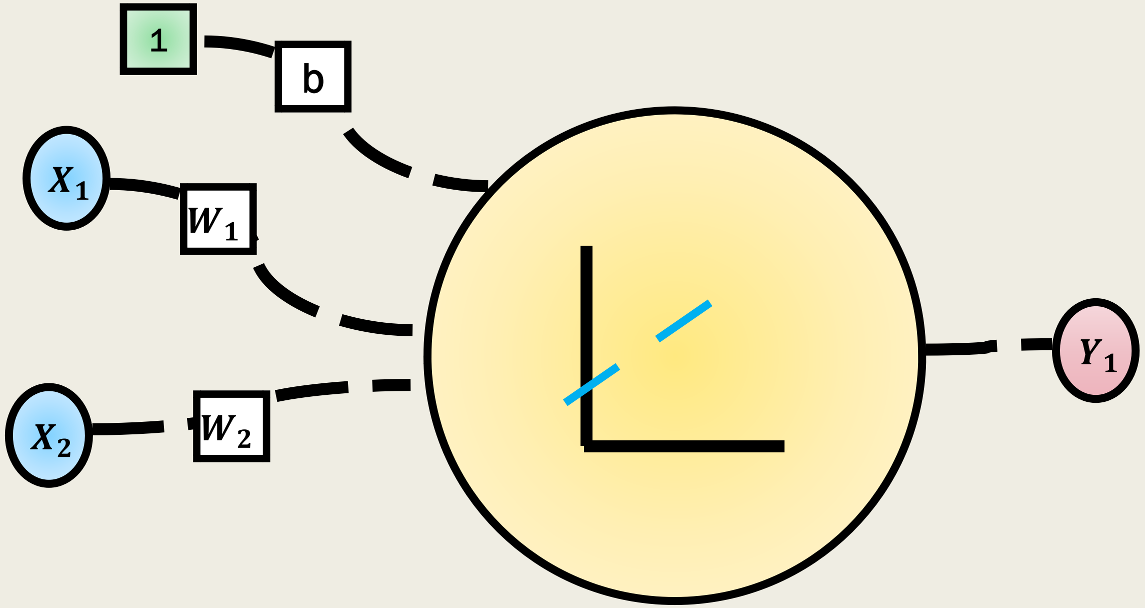
Término independiente

$$y = w_0 + w_1 x$$

Nos sirve para mover verticalmente a la recta

Neurona:

- En la neurona también tendremos este mismo término que nos dará control para mover nuestra función



Se representa como una conexión a la neurona, pero en el que la variable siempre esta asignada a uno

Podemos controlar manipulando el parámetro de sesgo

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

Término de sesgo (BIAS)

Ejemplo:

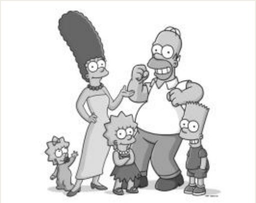
¿Qué se puede hacer en una buena tarde de viernes?

R. Serie favorita y palomitas

Variable Binarias

1 0

SerieF : X_1

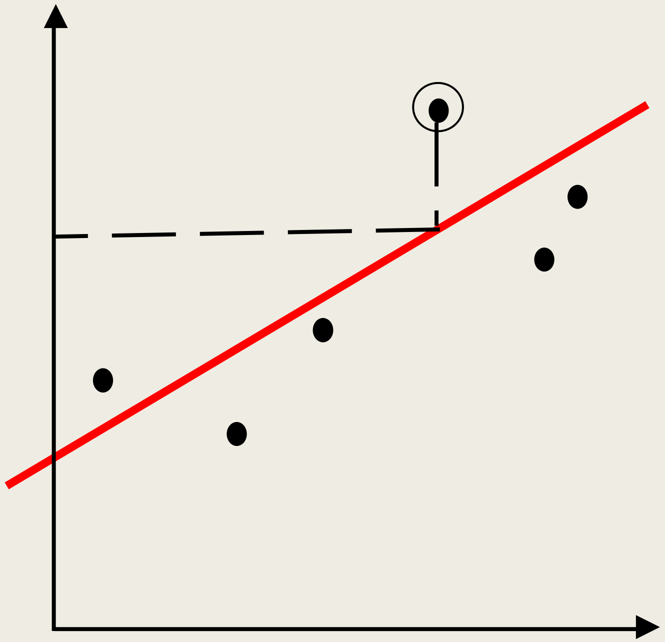
Palomitas : X_2

1 0

B Viernes : Y_1


Neurona:

- Si usamos nuestra neurona como la tenemos ahorita actuaría como el modelo de regresión lineal



Nos da un resultado continuo y no una variable binaria

$$WX \leq UMBRAL \rightarrow Y = 0$$

$$WX > UMBRAL \rightarrow Y = 1$$

$$\boldsymbol{BIAS} = -UMBRAL$$

Asignamos un valor equivalente a lo opuesto al umbral

Neurona:

- Reescribiendo la fórmula de manera más sencilla:

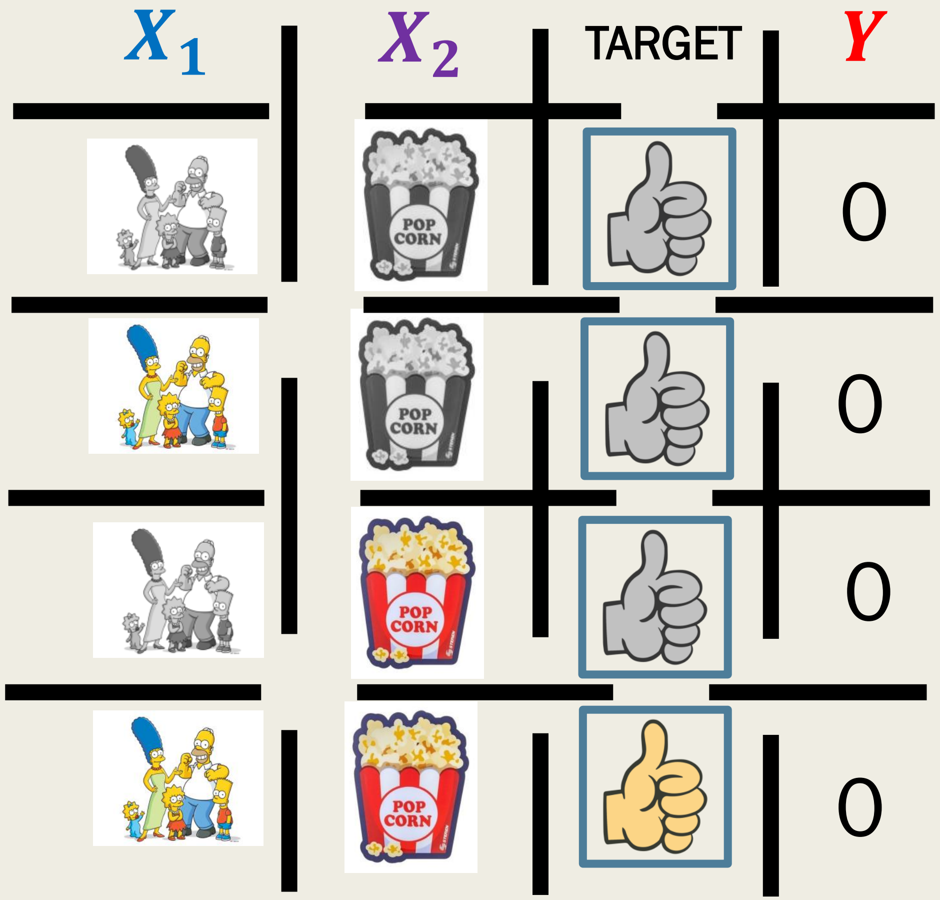
$$BIAS = -UMBRAL$$

$$WX + b \leq 0 \rightarrow Y = 0$$
$$WX + b > 0 \rightarrow Y = 1$$

- Ahora el valor de la salida dependerá de si el resultado de nuestra neurona es mayor o menor que 0

Neurona:

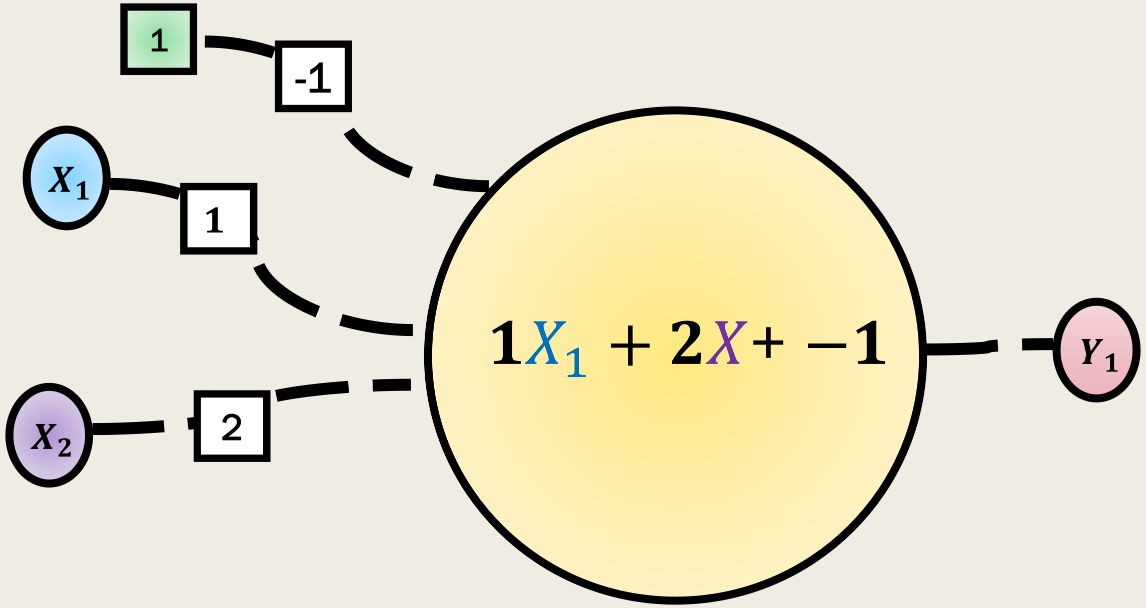
- Todos los resultados posibles de nuestra neurona



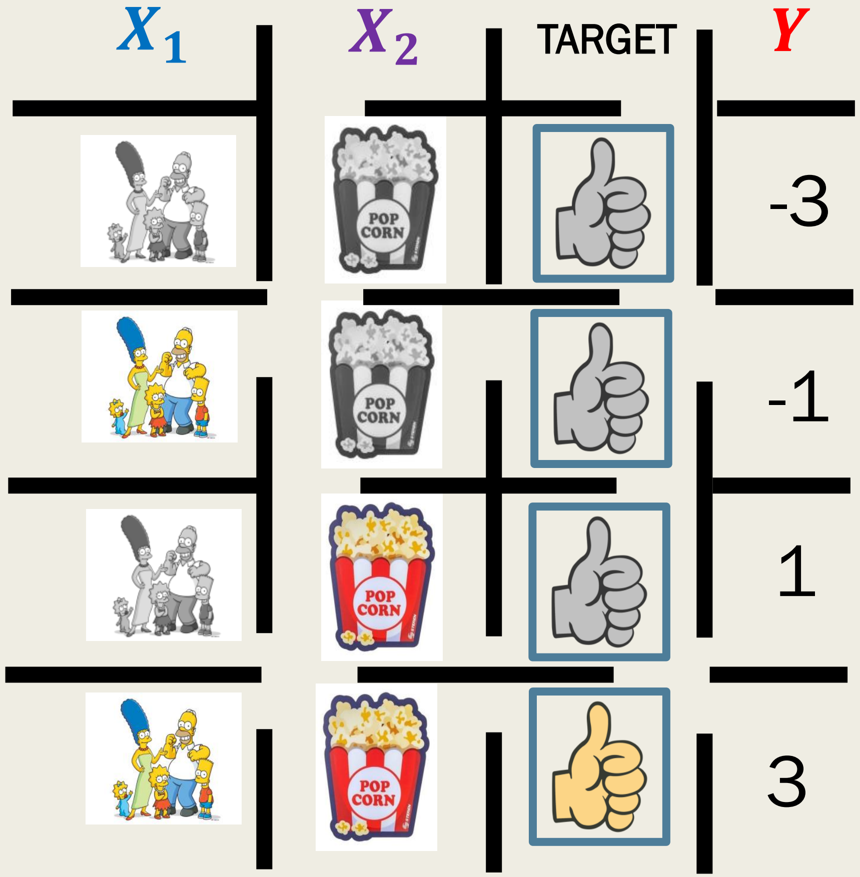
Nuestra tarea será ir variando el valor de nuestros parámetros, tanto los pesos de las conexiones como el sesgo para encontrar la combinación perfecta que modele nuestra tarde de viernes ideal

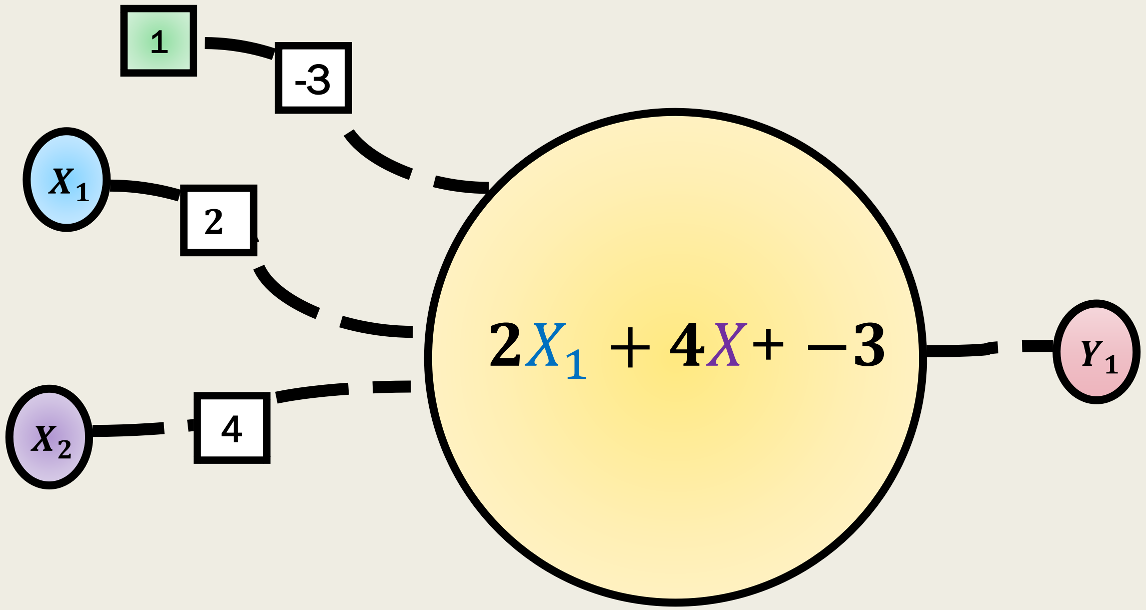
Neurona:

X_1	X_2	TARGET	Y
			-1
			0
			1
			2



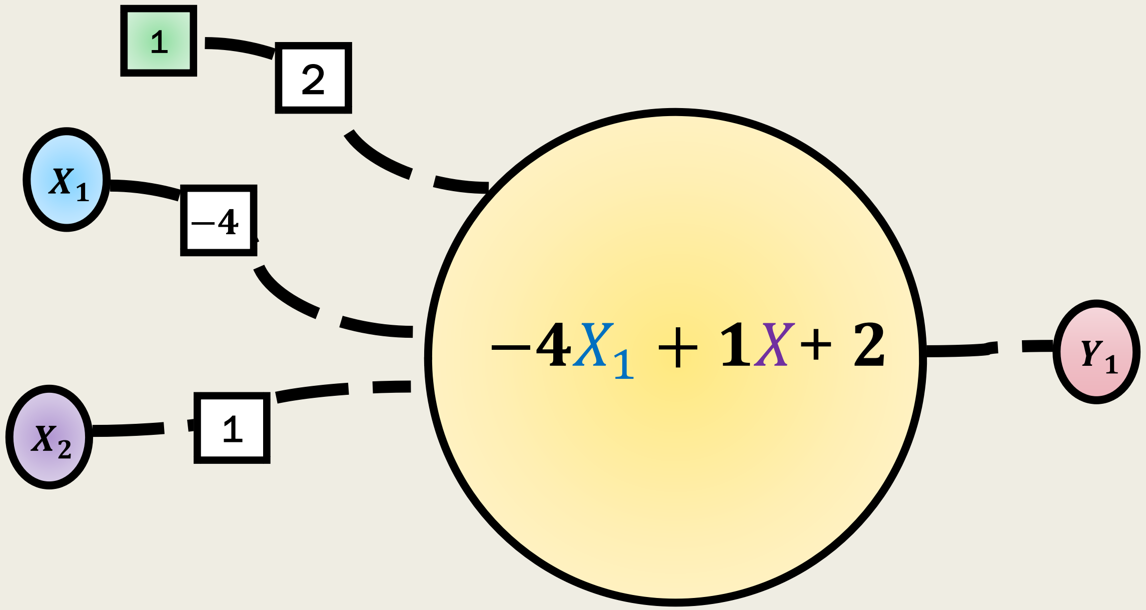
Neurona:





Neurona:

X_1	X_2	TARGET	Y
			2
			-2
			3
			-1

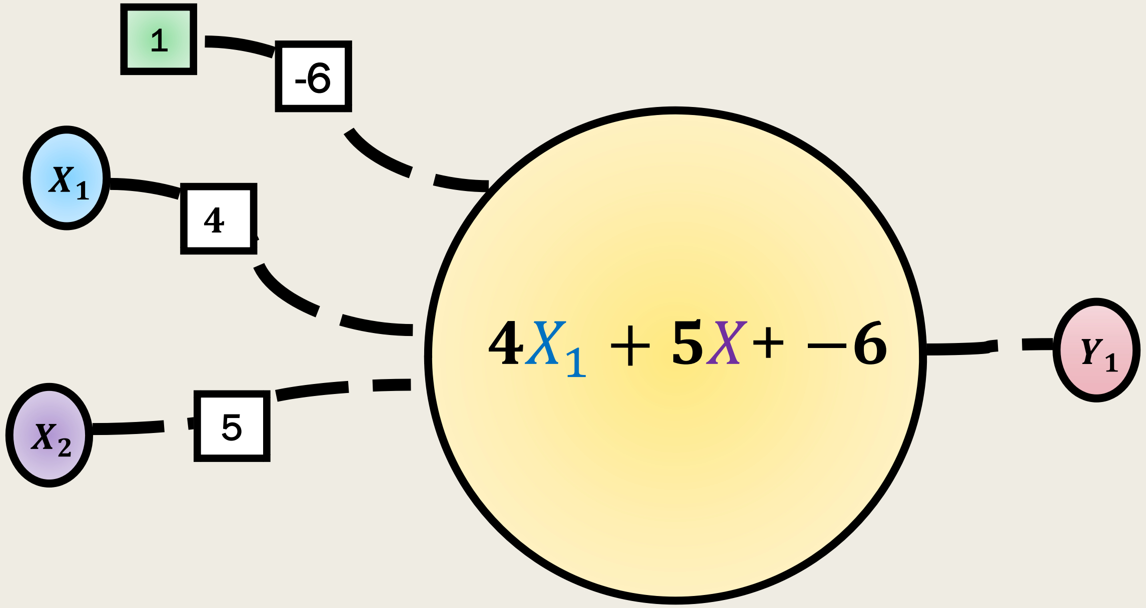


Neurona:

Resultado esperado:

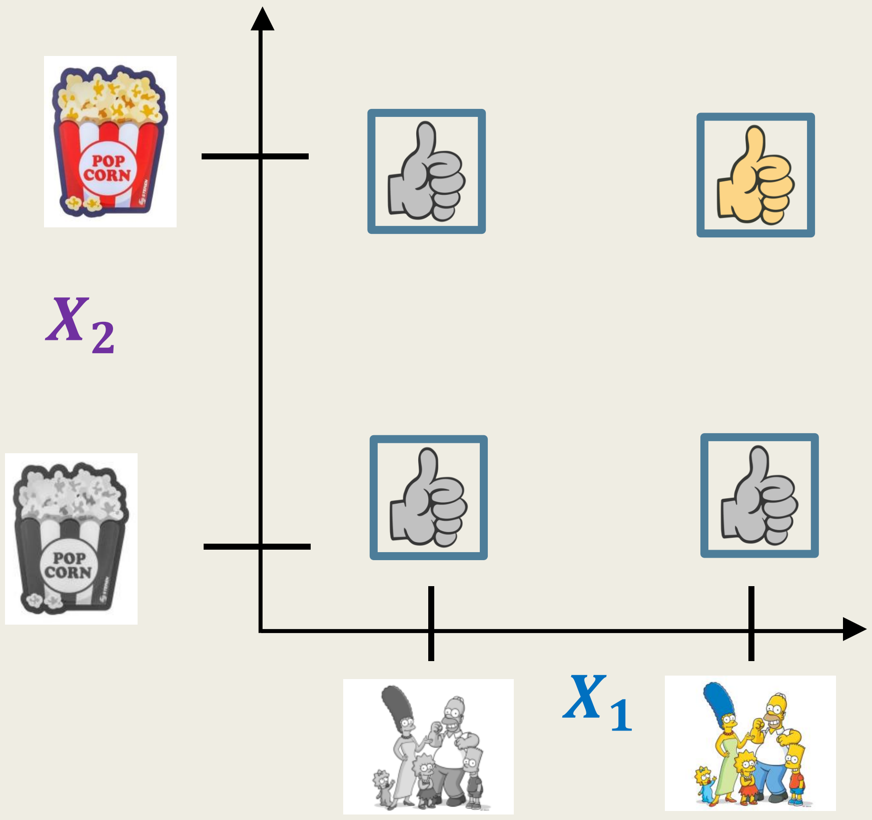
X_1	X_2	TARGET	Y
			-6
			-2
			-1
			3

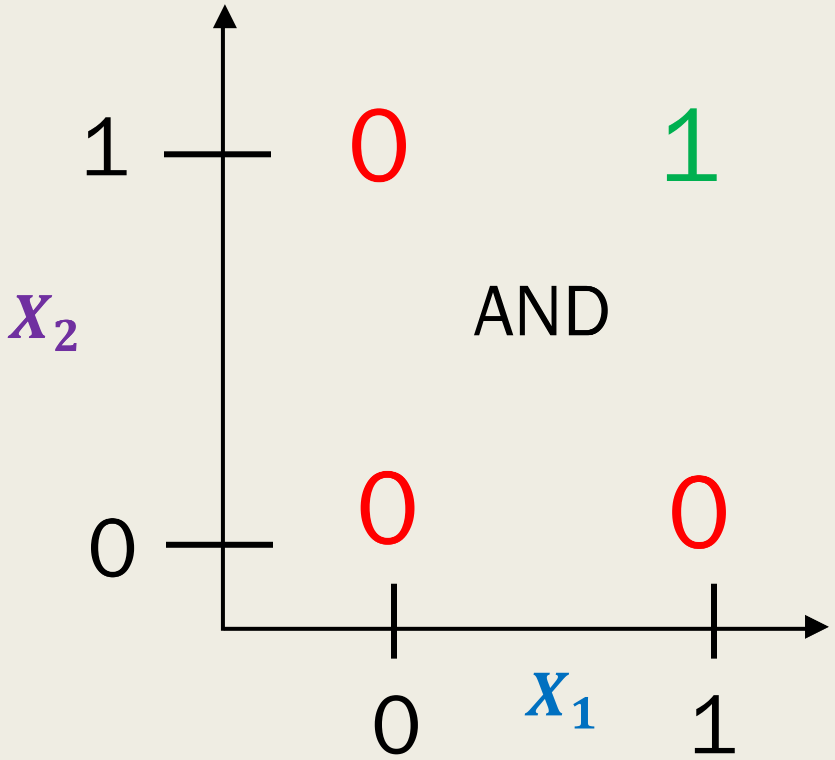
Positiva



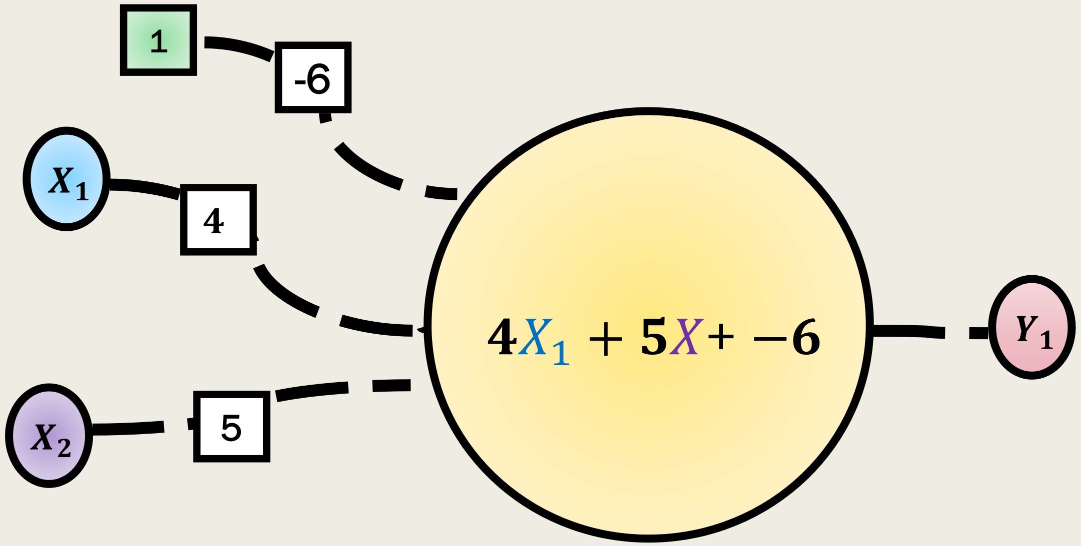
Neurona:

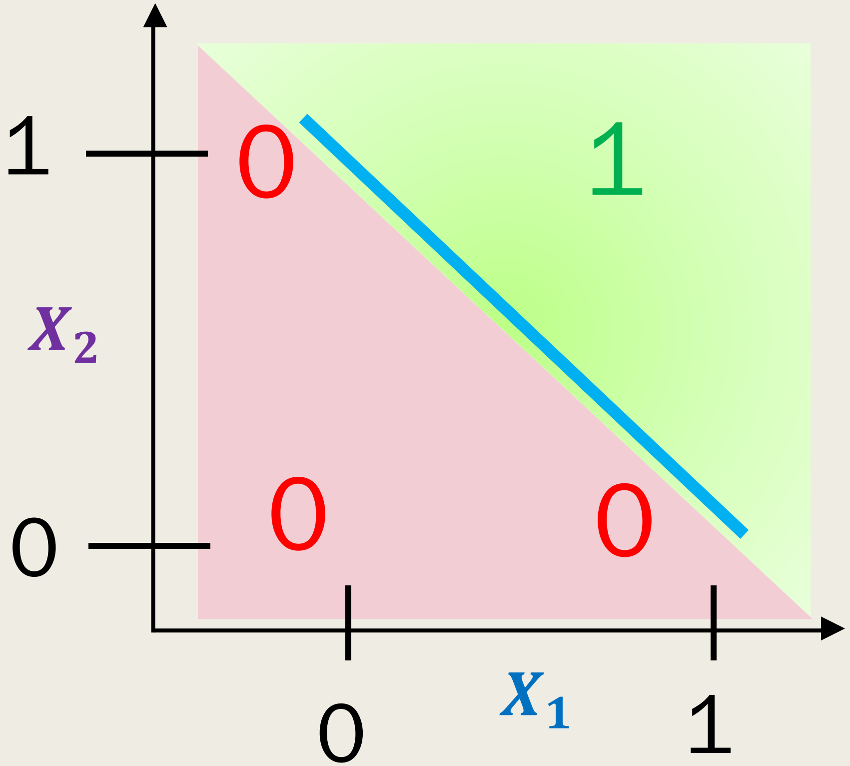
- Representación gráfica:





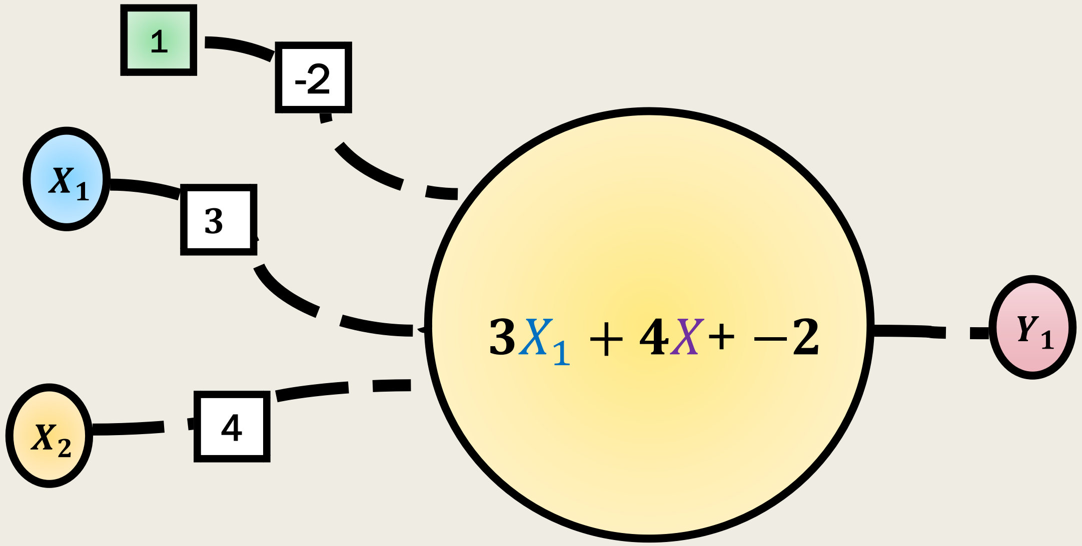
- Dibujamos la recta definida por nuestra neurona:

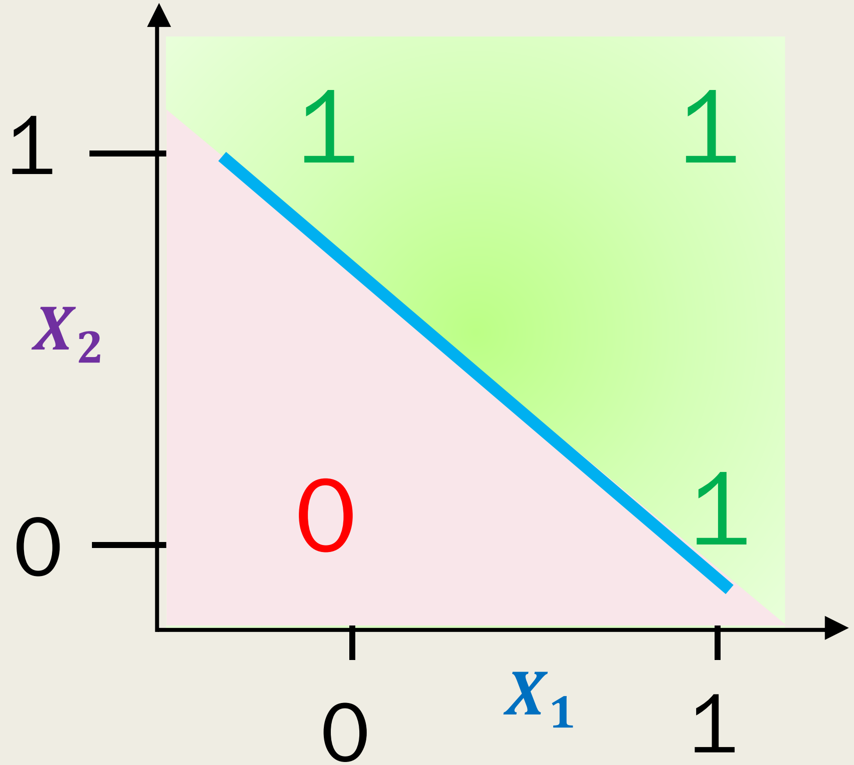




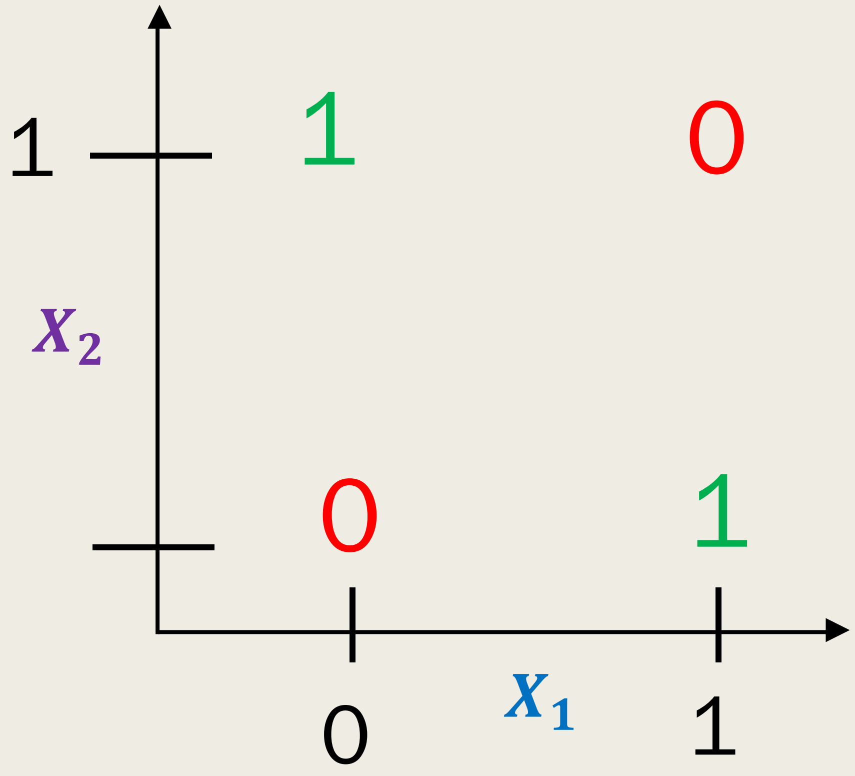
- Separa nuestros puntos de la gráfica en dos grupos diferentes
- Encontrar aquellos valores de nuestros parámetros que tracen una frontera entre las dos clases que queremos clasificar

- Probemos con otras compuertas: **OR**

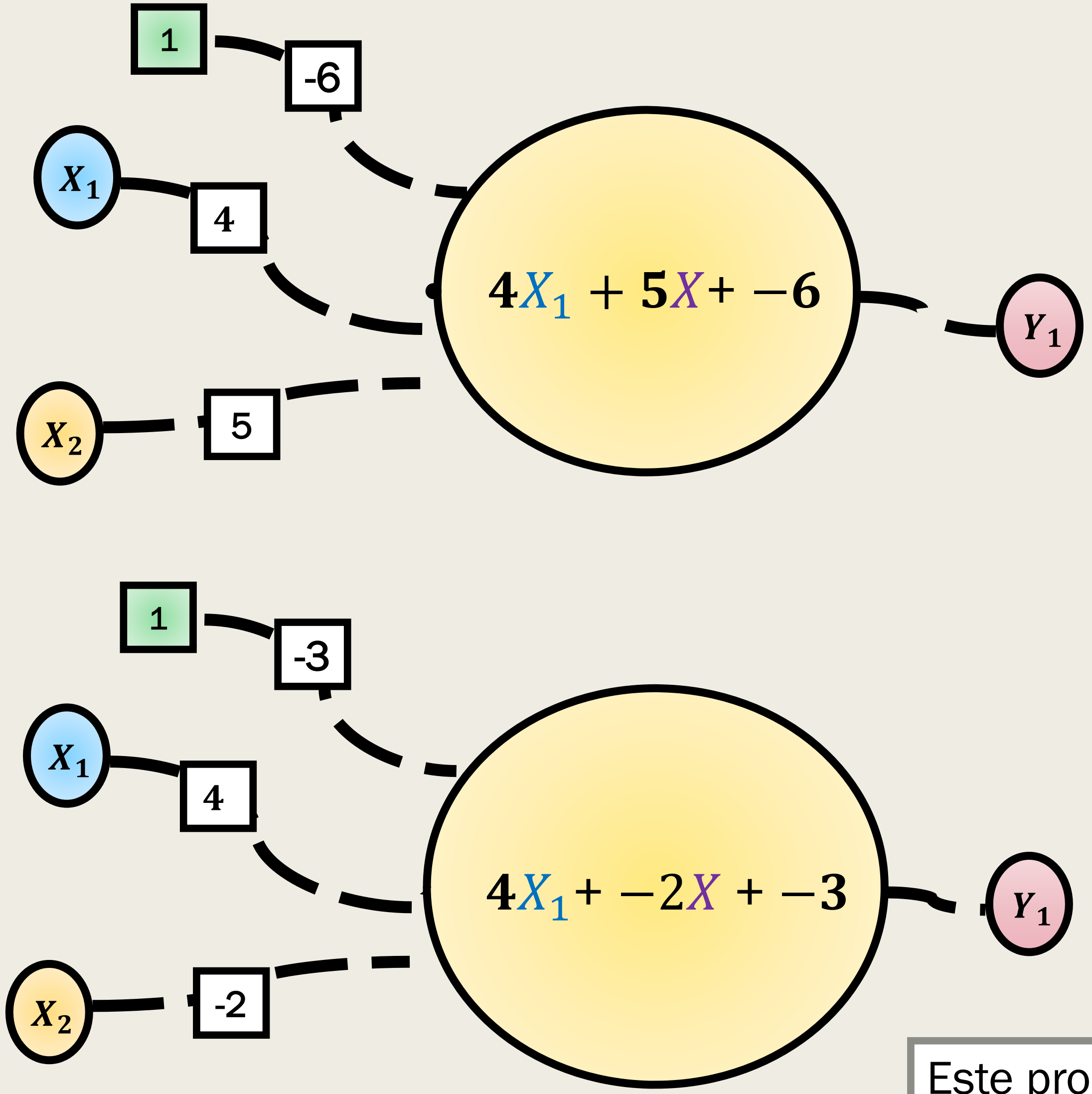


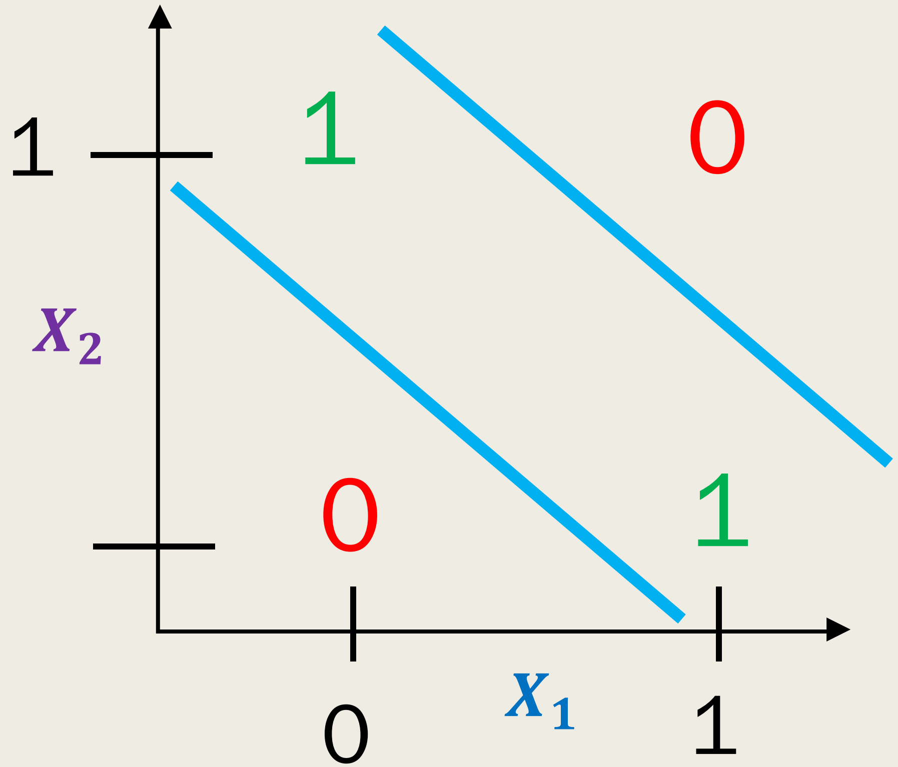


- Probemos con otras compuertas: XOR



- Este problema no tiene solución y es que se trata de una de las limitaciones de utilizar una única neurona para codificar este modelo
- Es imposible separar linealmente ambas clases

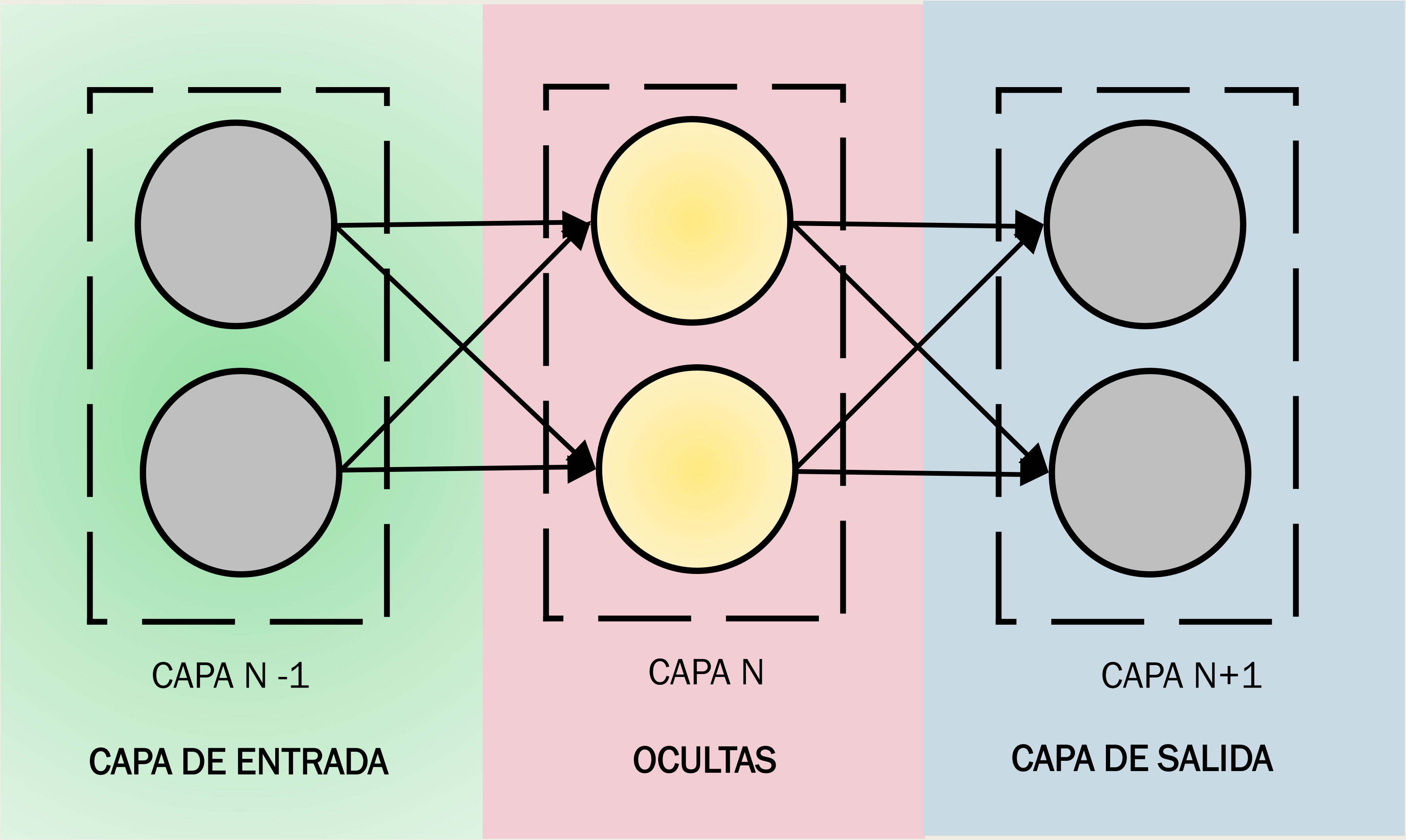




Este problema de la puerta XOR se conoce desde 1969 e ilustra la necesidad de combinar varias neuronas para conseguir modelos más complejos

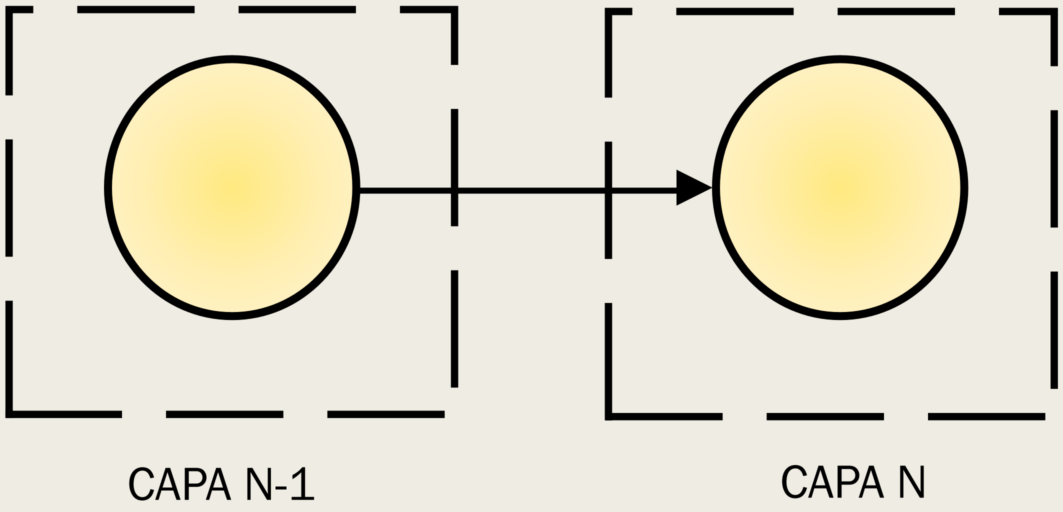
REDES NEURONALES

Redes Neuronales



Redes Neuronales

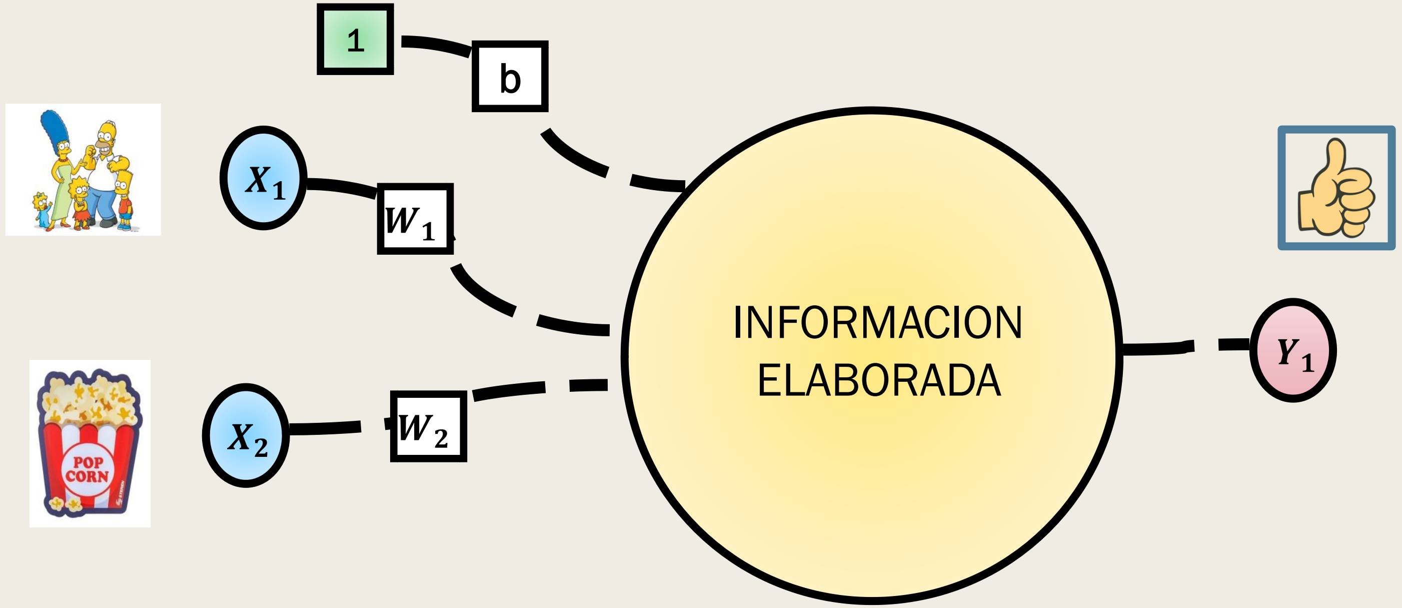
- Cuando colocamos dos neuronas de forma secuencial, una de ellas recibe la información procesada por la neurona anterior



Ventaja?
Conseguimos que la red pueda aprender conocimiento jerarquizado

Redes Neuronales

- Retomando el ejemplo de la tarde de viernes

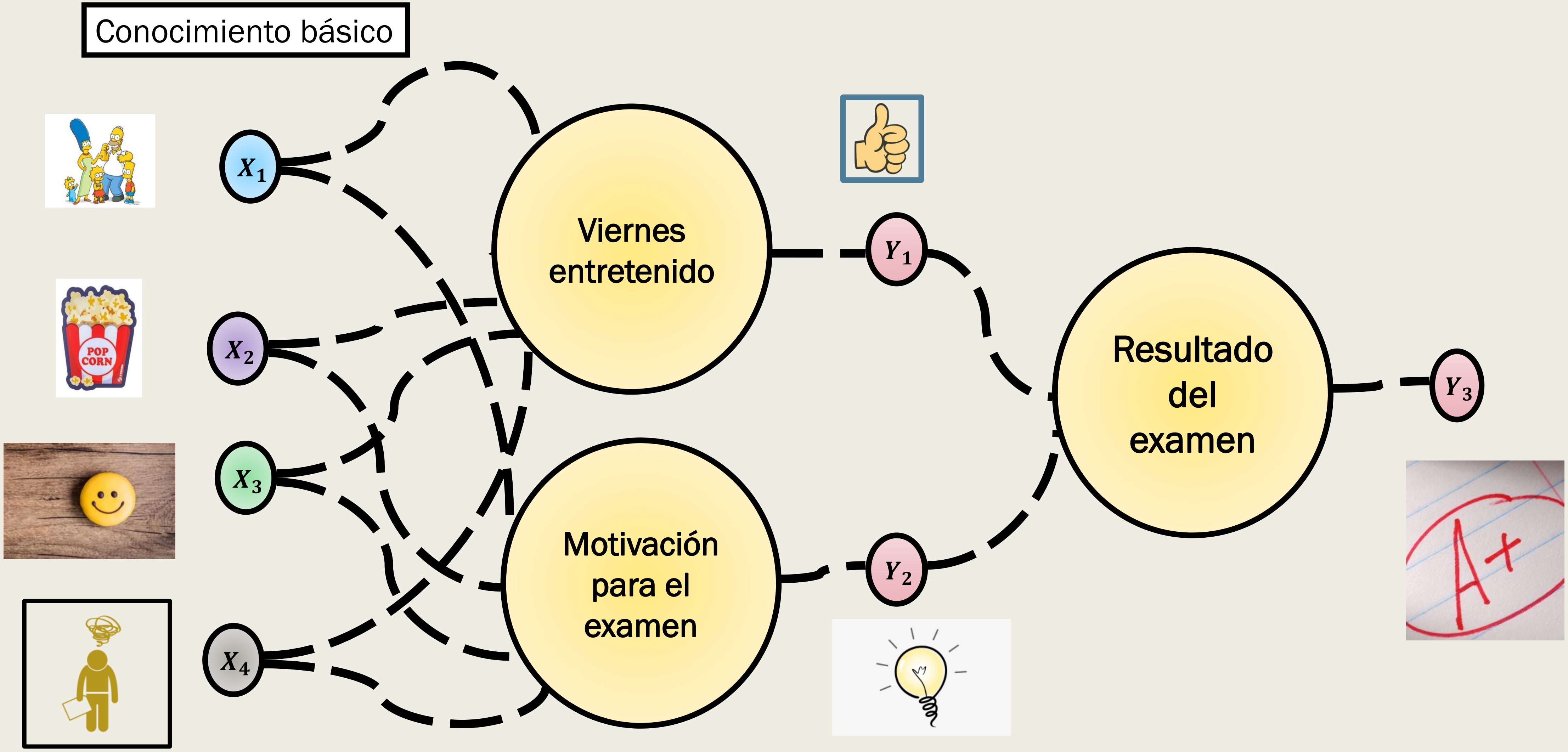


$$y = w_1 x_1 + w_2 x_2 + b$$

Y si utilizamos esta información para elaborar algo más complejo aún??

La calificación de nuestro próximo examen

Redes Neuronales

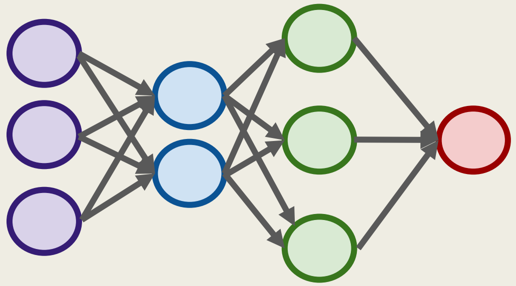


Redes Neuronales

- Mientras más capas, más complejo puede ser el conocimiento que elaboremos
- Esta profundidad en la cantidad de capas es lo que da nombre al ***aprendizaje profundo, el Deep Learning***

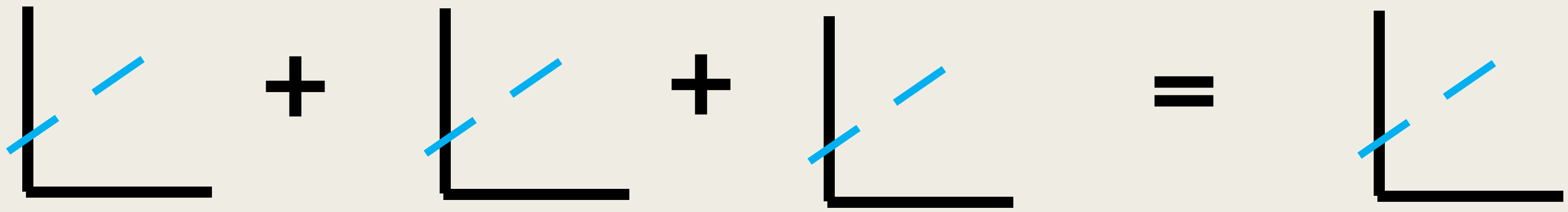
PERO ….

- Queremos conectar múltiples neuronas de forma secuencial y como vimos al final lo que hace cada una de estas es un problema de regresión lineal
- Si lo planteamos matemáticamente, es concatenar diferentes operaciones de regresión lineal

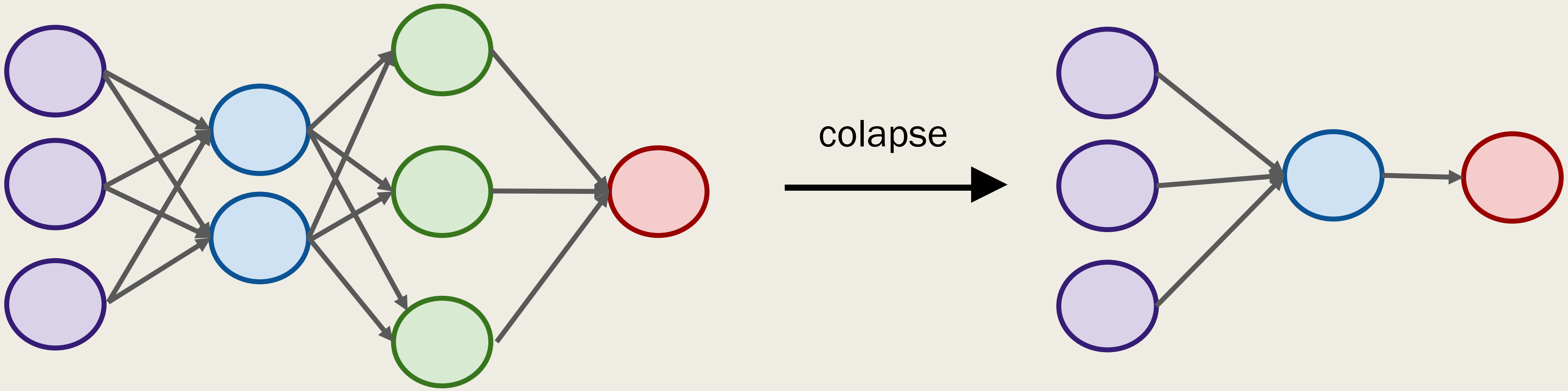


Redes Neuronales

- Matemáticamente se puede comprobar que el efecto de sumar muchas operaciones de regresión lineal equivale solamente haber hecho una única operación, es decir, da como resultado otra lineal recta



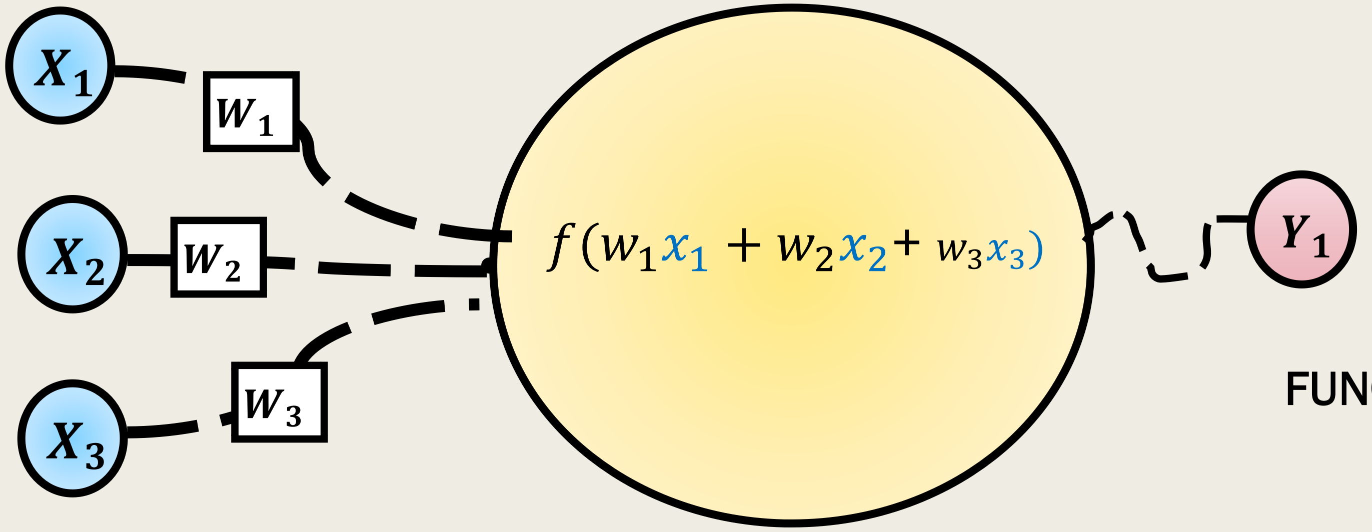
Redes Neuronales



- Necesitamos que la suma de como resultado algo diferente a una línea recta
- Cada línea sufra alguna manipulación no lineal que las distorsione

Funciones de Activación

- La última componente que falta ver en la estructura de la neurona
- Básicamente, si en nuestra neurona calculábamos como valor de salida una suma ponderada, ahora pasaremos nuestra salida por la función de activación
- La función de activación lo que hace es distorsionar nuestro valor de salida, añadiéndole deformaciones no lineales
- Podremos encadenar de forma efectiva el cálculo de varias neuronas



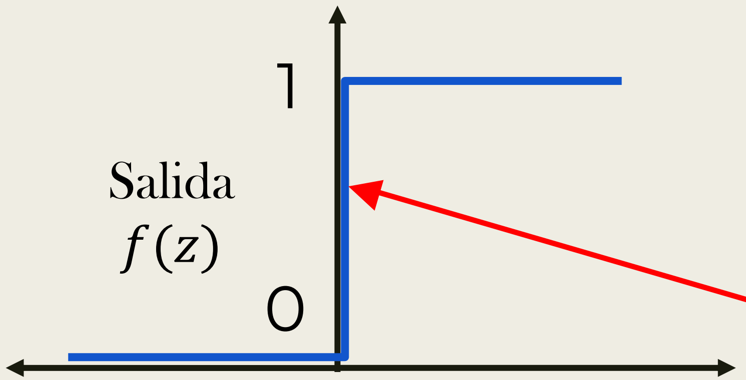
FUNCION DE ACTIVACIÓN : $f(x)$

Funciones de Activación

ESCALONADA:

Para un valor mayor al umbral, la salida es 1
Si es inferior es igual a 0

Se le llama escalonada, porque el cambio es instantáneo y no de forma gradual



Valores intermedios se pierden

$$f(z) = \begin{cases} 0 & si \ \ z < 0 \\ 1 & si \ z \geq 0 \end{cases}$$

Adecuada para clasificación binaria (2 clases).

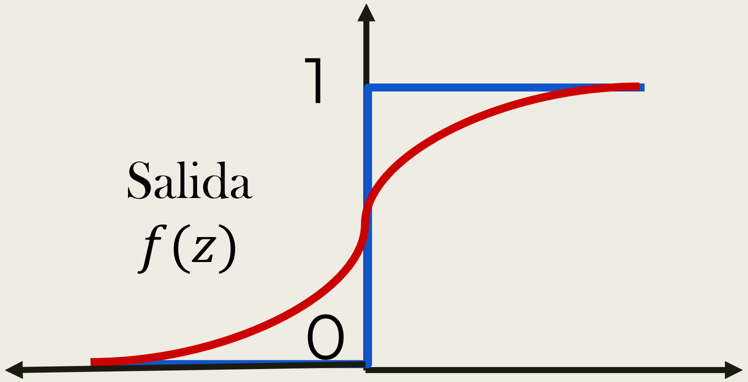
$$Z = \sum_{i=1}^{n} x_i w_i$$

Funciones de Activación

SIGMOIDE:

La distorsión que produce hace que los valores muy grandes se saturen en 1 y los valores muy pequeños se saturen en 0

Con esta función, no solo conseguimos añadir la deformación que estamos buscando, sino que también nos sirve para representar *probabilidades* en el rango de 0 a 1



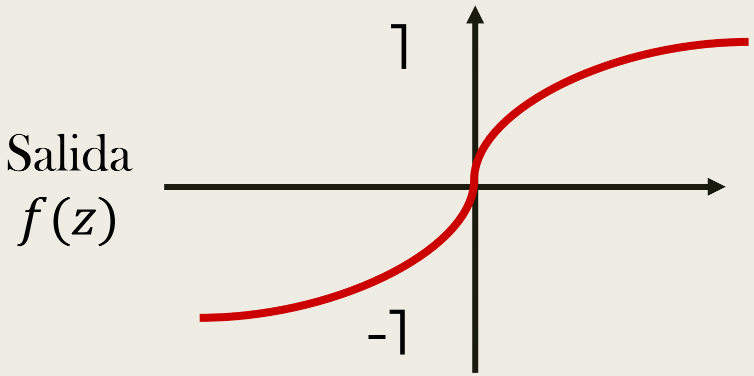
$$\boxed{f(z) = \sigma(z) = \frac{1}{1+e^{-z}}}$$

$$Z = \sum_{i=1}^{n} x_i w_i$$

Funciones de Activación

TANH (Tangente Hiperbólica):

Similar a la sigmoide pero cuyo rango varia de -1 a 1



$$tanh(z) = \frac{2}{1 + e^{-(2z)}} - 1$$

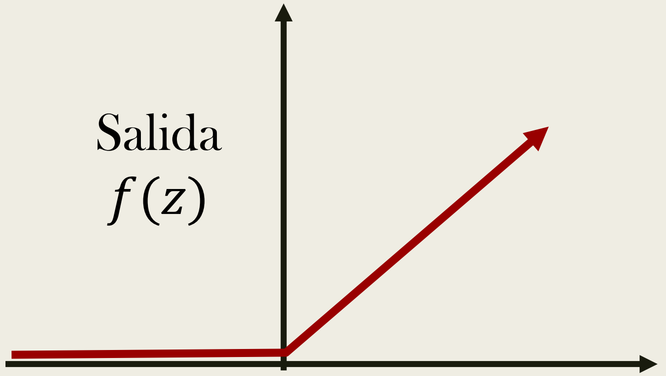
$$Z = \sum_{i=1}^{n} x_i w_i$$

$$f(z) = \tanh(z) = \frac{(e^{z} - e^{-z})}{(e^{z} + e^{-z})}$$

Funciones de Activación

RELU (Rectified Lineal Unit)**:**

Se comporta como una función lineal cuando es positiva y constante a 0 cuando el valor de entrada es negativo

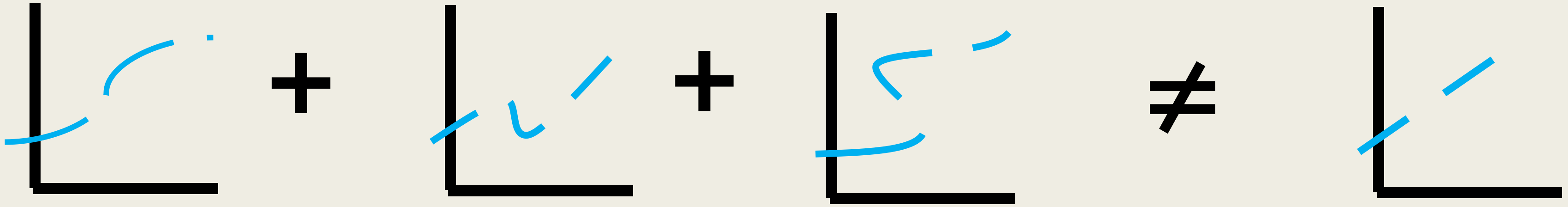


$$z = \sum_{i=1}^{n} x_i w_i$$

$$f(z) = \begin{cases} 0 \;\; si \;\; z \leq 0 \\ z \;\; si \;\; z > 0 \end{cases}$$

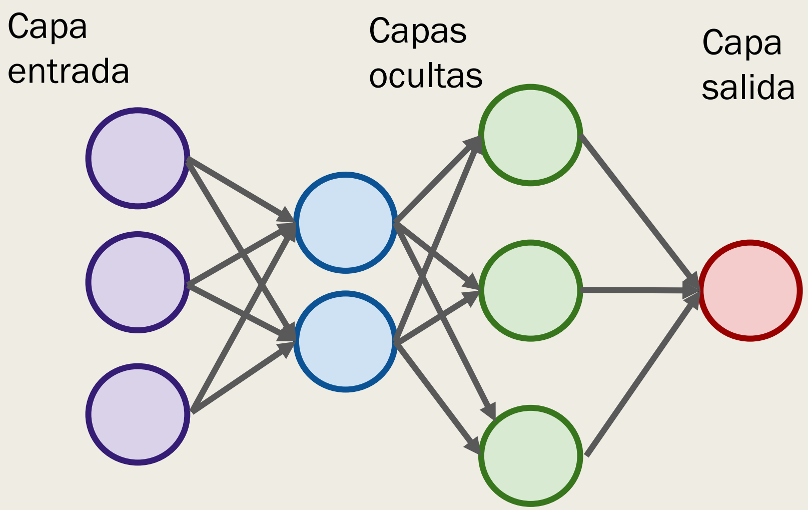
Funciones de Activación

- Cada una de estas funciones, además de aportar la no linealidad que estamos buscando también ofrecen diferentes beneficios de cuando las utilicemos
- Con esto, damos por solucionado el problema de concatenar varias neuronas

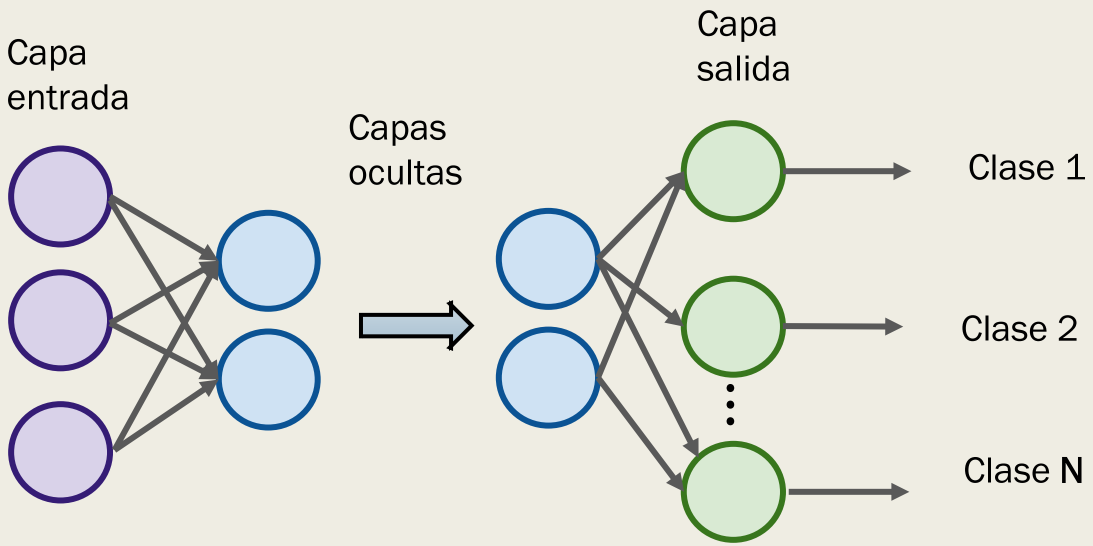


Funciones de Activación

- Estas funciones tienen mucho sentido para problemas de 2 clases, pero *¿Qué pasa con problemas multiclases?*
- En primer lugar, se debe cambiar el diseño de la red



Una sola neurona en la salida solo puede devolver 1 valor



La capa de salida debe tener una neurona por cada clase

Funciones de Activación

- Multiclases no exclusivas (multietiquetas): cada patrón puede estar asociado a varias etiquetas (clases)

Usar la función de activación sigmoide

La salida de cada neurona es un valor entre 0 y 1, que indica la probabilidad de pertenecer a esa clase.
Permite que la salida de cada neurona sea independiente del resto

BACKPROPAGATION

Descenso del gradiente

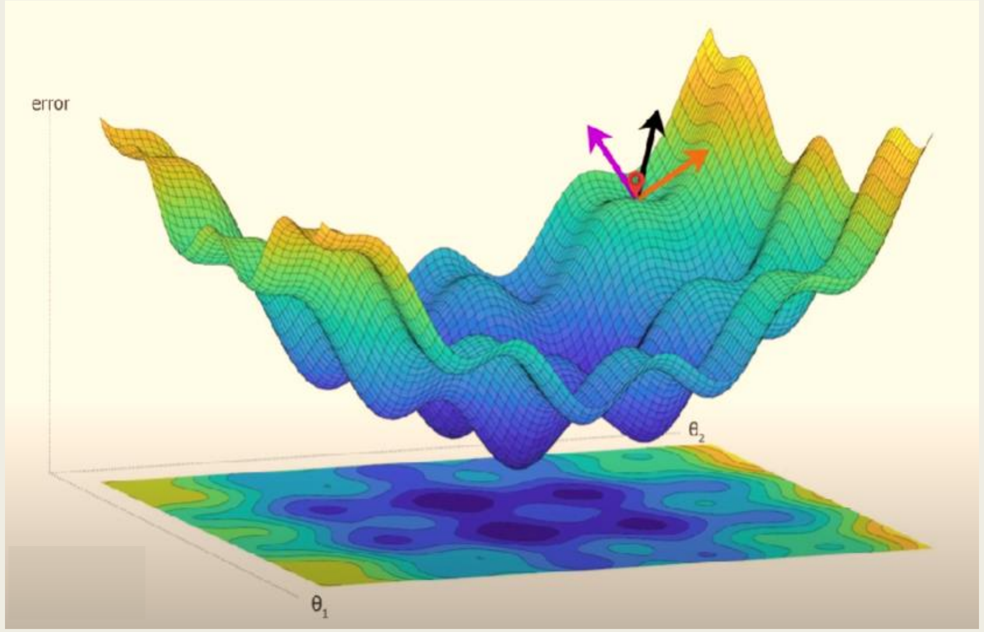
- Con el *descenso del gradiente* conseguimos una estrategia para ajustar los parámetros de nuestro modelo (regresión lineal)

1. Evaluamos el error del modelo en el punto en el que nos encontrábamos y calculamos las derivadas parciales en dicho punto

$$\begin{bmatrix} \frac{\partial error}{\partial \theta_1} \\ \frac{\partial error}{\partial \theta_2} \end{bmatrix} = \nabla f$$

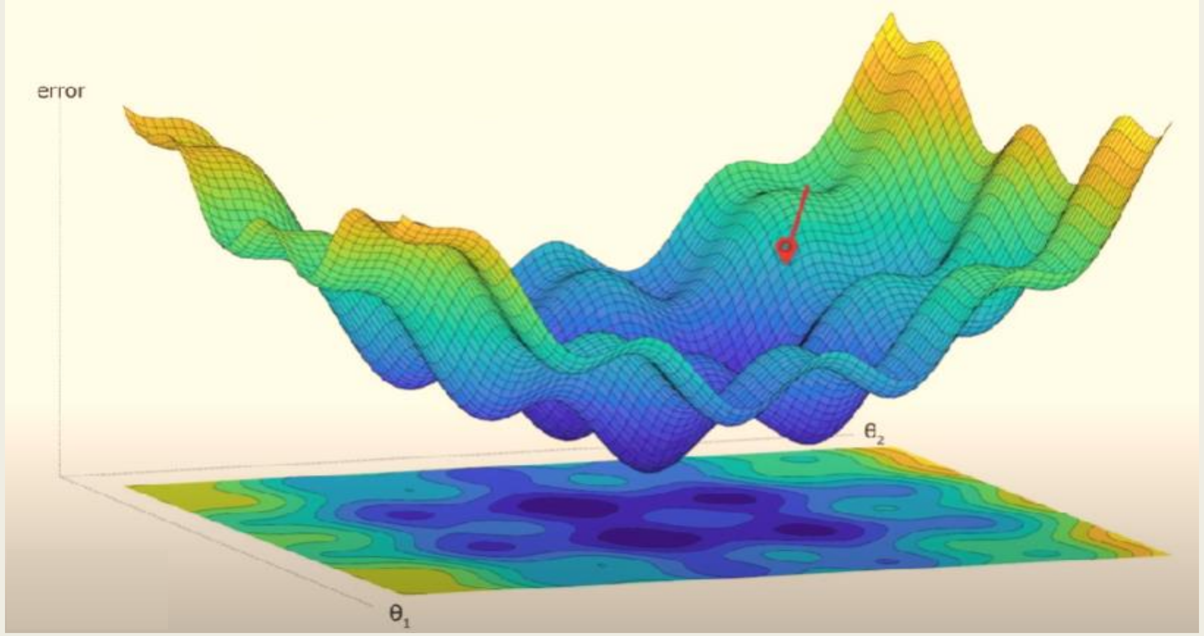
GRADIENTE

2. Con esto obtenemos un vector de direcciones que nos indicaba la pendiente de la función hacia donde el error se incrementaba (gradiente)



Descenso del gradiente

3. Con eso nos movíamos a la dirección contraria, así teníamos una forma por la cual iterativamente podríamos ir reduciendo el error del modelo



$$\theta \; \coloneqq \theta - \nabla f$$

RECUERDA el gradiente es el vector que contiene las pendientes para cada una de las dimensiones de f

Descenso del gradiente

- Cuando trabajamos con regresión lineal, calcular el vector gradiente es muy sencillo porque tenemos sólo dos parámetros que afectan directamente al resultado

$$y = w_0 + w_1 x$$

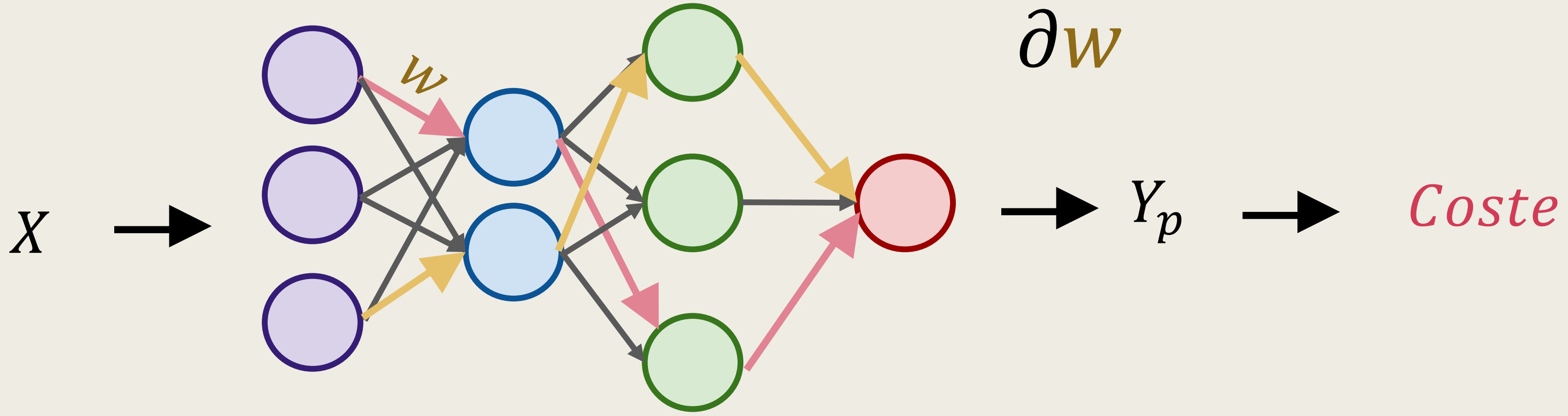
- **¿Cómo varia el coste ante un cambio del parámetro w?**
- *R. Con las derivadas parciales de la función de coste con respecto a cada uno de los parámetros*

$$\frac{\partial C}{\partial w}$$

Descenso del gradiente

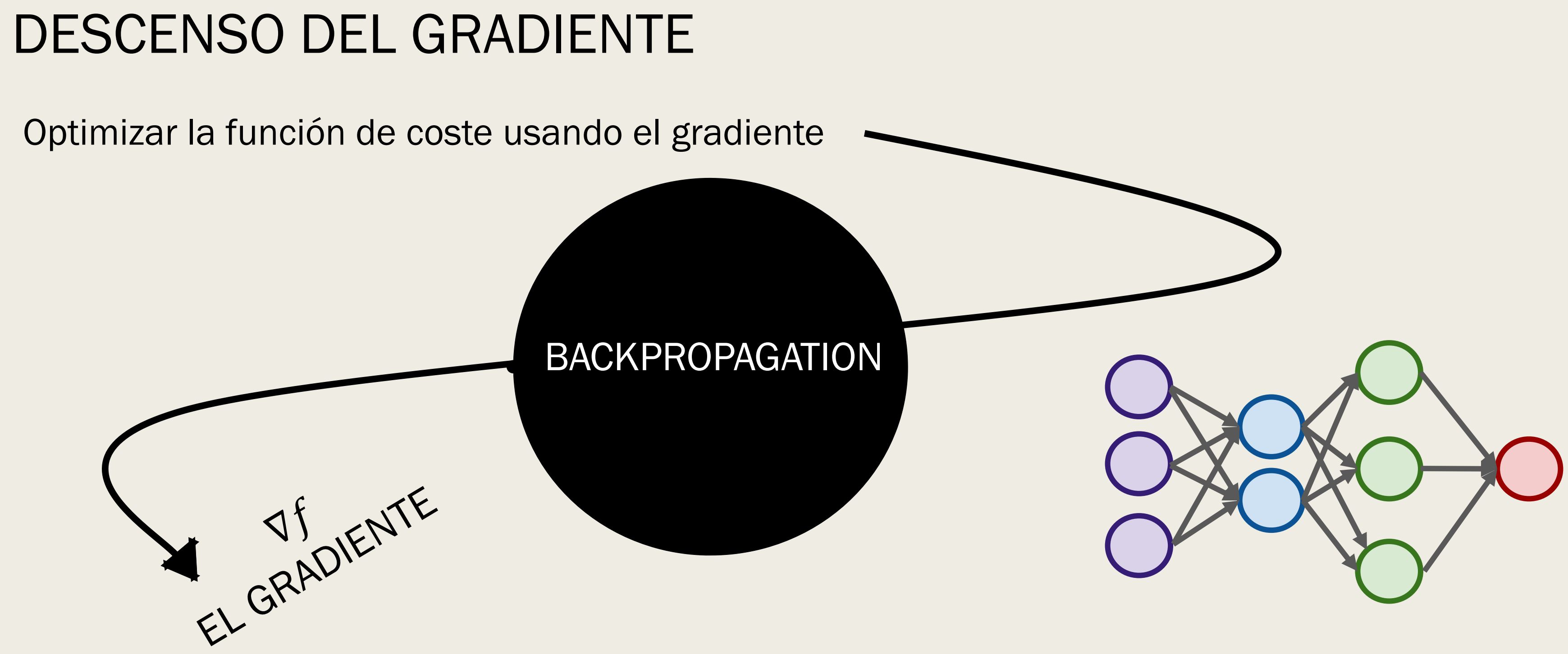
- Pero cuando trabajamos con redes neuronales, el concepto del gradiente es el mismo que acabamos de ver, es decir, cómo varia el coste cuando variamos un parámetros

$$\frac{\partial C}{\partial w}$$



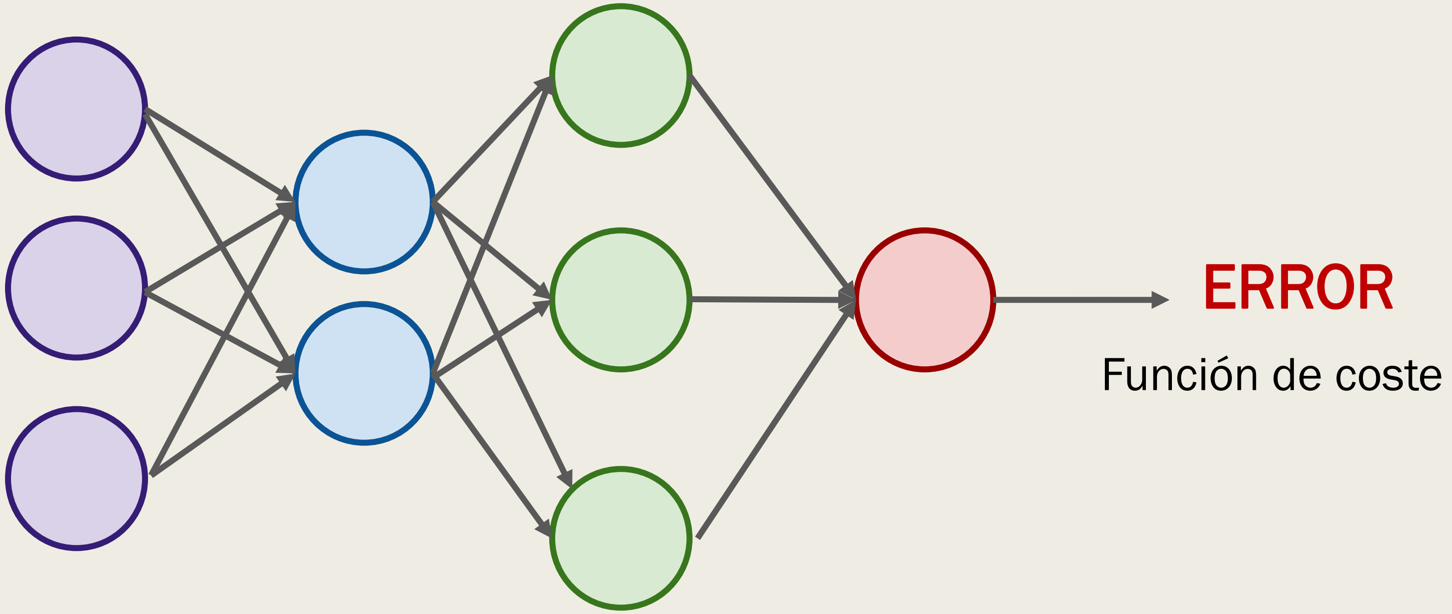
Backpropagation

- Utilizaremos el descenso del gradiente para optimizar nuestra función de coste, haciendo uso de la técnica de **backpropagation** para calcular el vector de gradiente dentro de la complejidad de la arquitectura de la red neuronal



Backpropagation

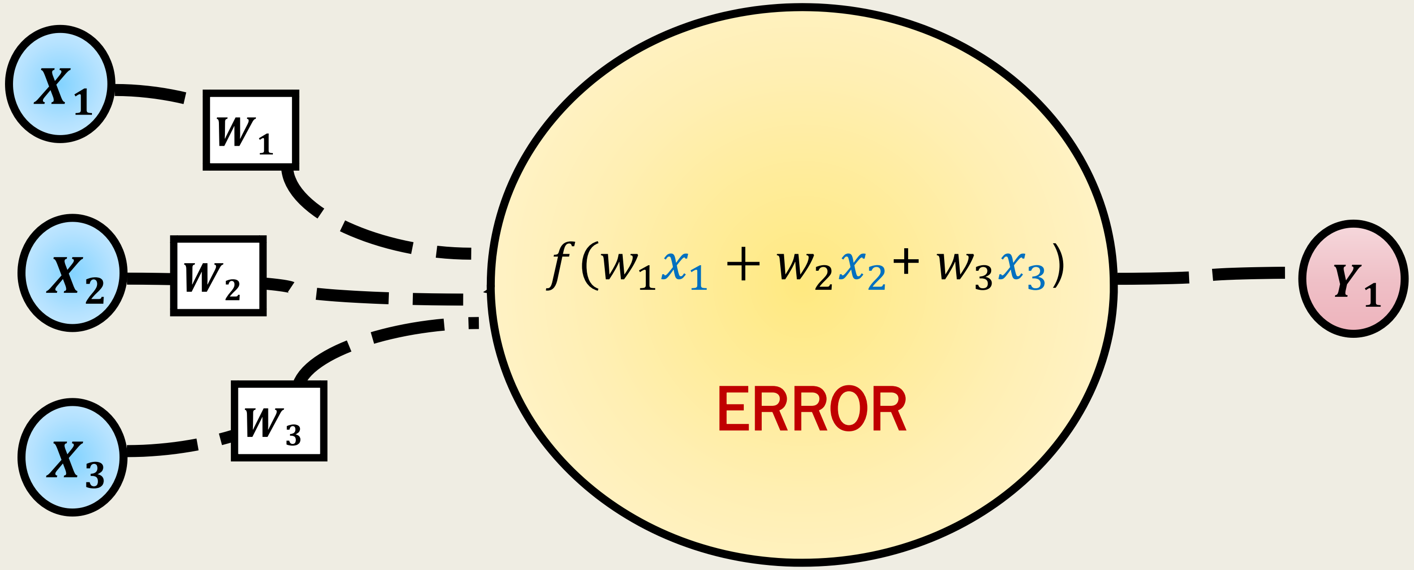
- En esta red neuronal cada nodo es una neurona especializada en una tarea determinada



- Hacemos el análisis de cuánta responsabilidad tiene cada neurona hacia atrás desde la señal de error hacia las primeras capas

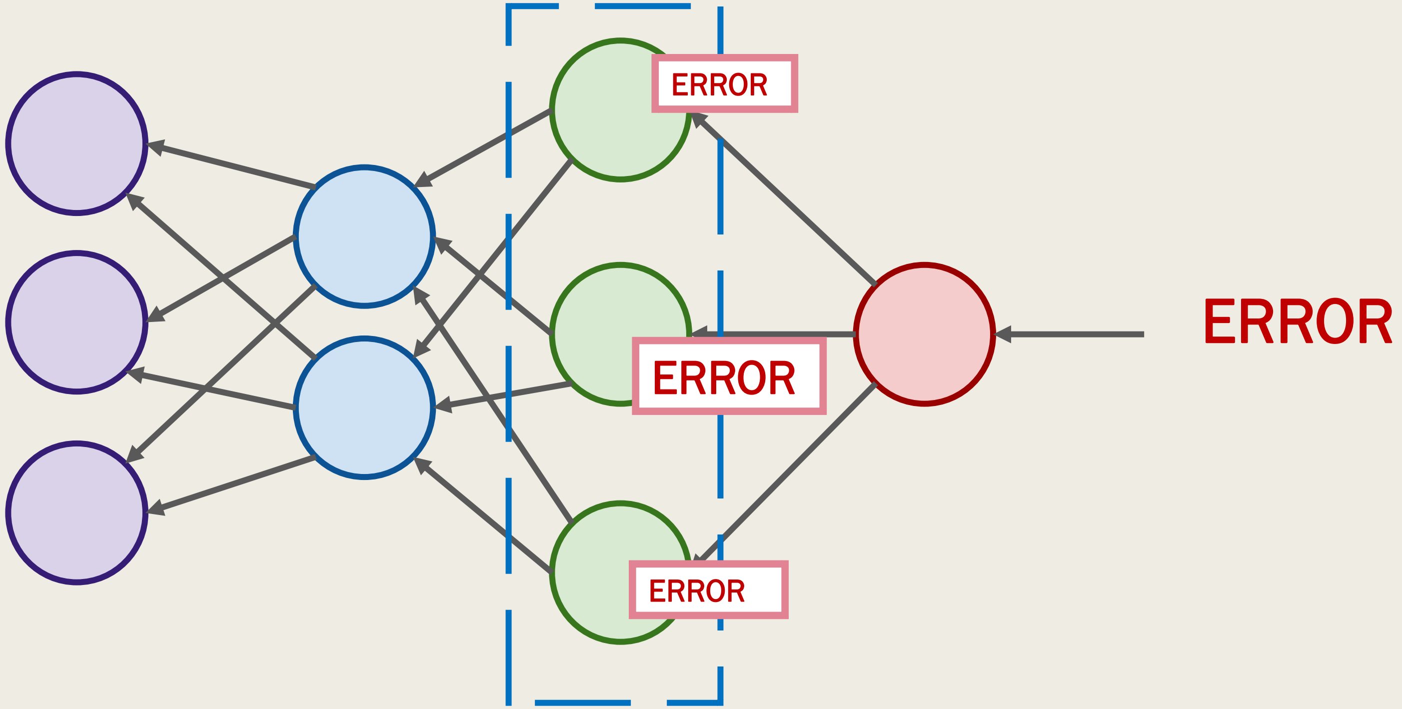
Backpropagation

- Lo hacemos de forma eficiente porque primero analizamos la última capa y en función de cuanto se haya implicado cada neurona podemos asignar un porcentaje del error
- Una vez calculado, ese error lo utilizaremos para calcular cuánto hay que modificar cada parámetro en dicha neurona



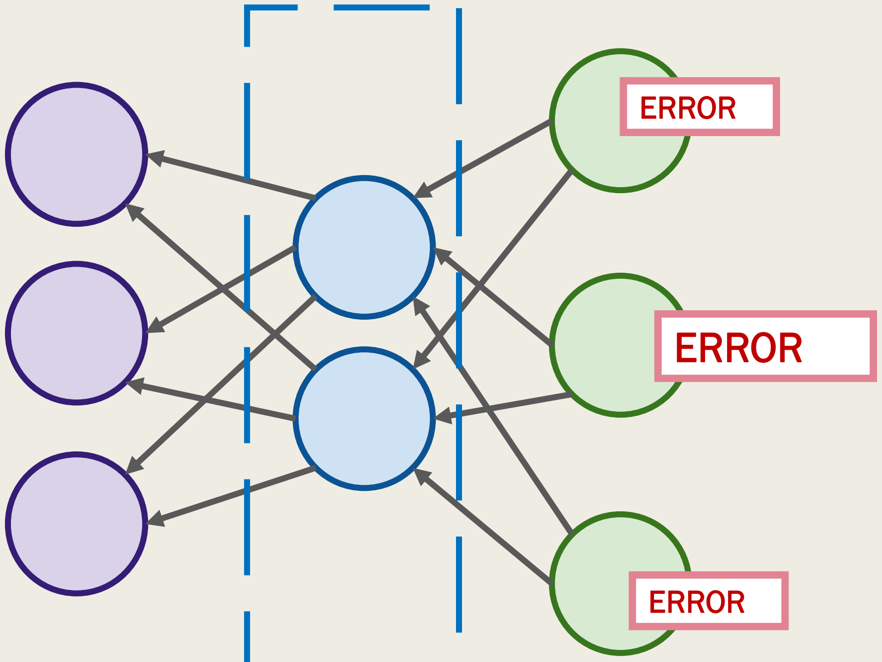
Backpropagation

- Ya que hemos imputado los errores a las neuronas de dicha capa, podemos proceder a repetir el mismo proceso de antes como si este fuera el error final de nuestra red



Backpropagation

- Asumimos que esta es nuestra última capa, así aplicar backpropagation es operar siempre de forma recursiva capa tras capa moviendo el error hacia atrás



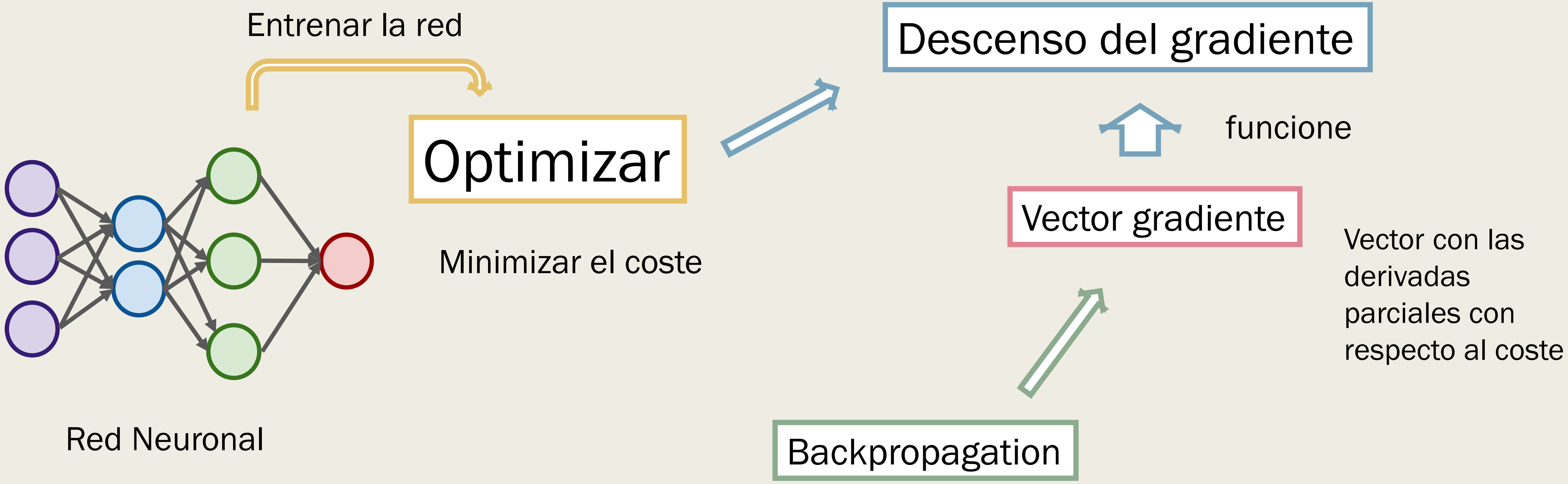
Última capa

Cuando lleguemos a la primera capa habremos obtenido cual es el error para cada neurona y para cada uno de sus parámetros solamente propagando **una única vez** el error hacia atrás

Esos errores son los que usaremos para calcular las derivadas parciales de cada parámetro de la red conformando el **vector gradiente**

Necesita el descenso de gradiente para minimizar el error

- **Backpropagation:** Método para calcular las derivadas parciales de cada uno de los parámetros de nuestra red con respecto al coste



LAS MATEMÁTICAS DEL BACKPROPAGATION

Las matemáticas de Backpropagation

- Tenemos una red neuronal, primeramente tiene sus parámetros inicializados aleatoriamente, por lo tanto la salida también es aleatoria y al comparar la predicción seguramente la función de coste le asignará un error muy elevado

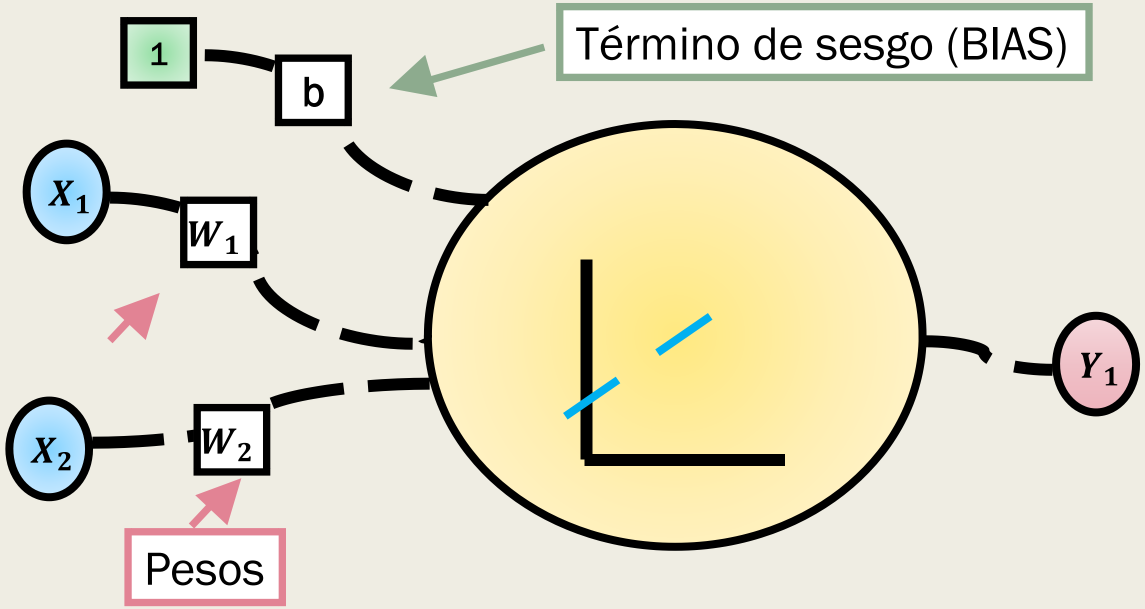
- Con este valor empezaremos a entrenar a la red

Veamos, lo queremos calcular para cada parámetro dentro de la red neuronal es la **derivada parcial del coste respecto a cada uno de los parámetros de la red**

$$\frac{\partial C}{\partial w}$$

Las matemáticas de Backpropagation

Tenemos dos tipos de parámetros:



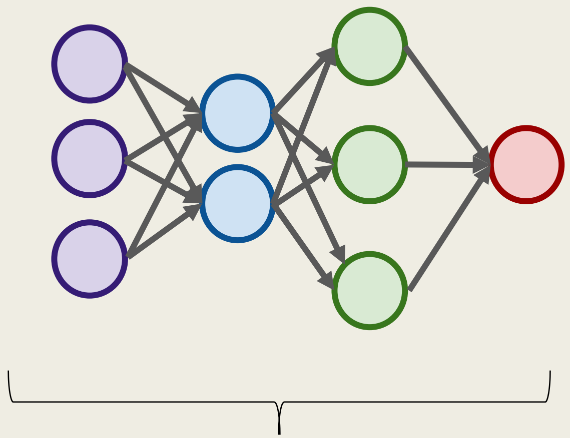
Tendremos que calcular dos tipos de derivadas parciales:

$$\frac{\partial C}{\partial w} \qquad \frac{\partial C}{\partial b}$$

Las matemáticas de Backpropagation

Vamos a comenzar a trabajar hacia atrás, comencemos a calcular la derivada de los parámetros de la última capa

Marcaremos con un superíndice el número de la capa que pertenece el parámetro



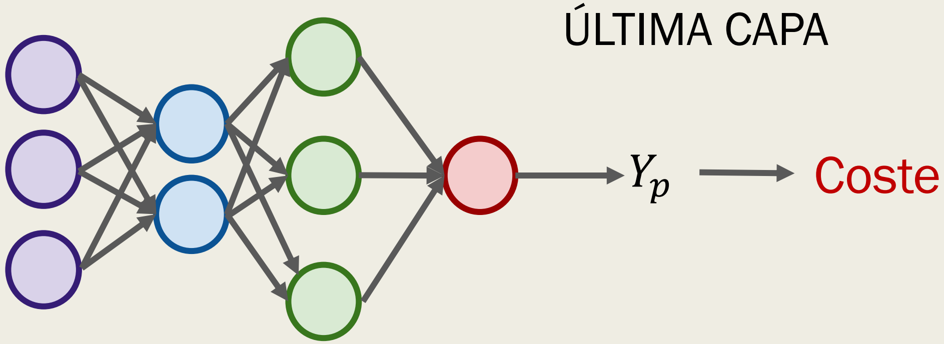
L capas

DERIVADAS DE LA ÚLTIMA CAPA

$$\frac{\partial C}{\partial b^L} \quad \frac{\partial C}{\partial w^L}$$

Las matemáticas de Backpropagation

Para calcular esta derivada, es importante analizar cuál es el camino que conecta el valor del parámetro y el coste final



El coste final no es muy largo pero tiene varios pasos

Las matemáticas de Backpropagation

Si recordamos el funcionamiento de la neurona:

1. El parámetro w participa en la suma ponderada a la que nos vamos a referir como Z^L
2. Luego es pasada por la función de activación $a(Z^L)$
3. El resultado de las activaciones de la neurona en la última capa conformaría el resultado de la red que sería evaluado por la función de coste C para así determinar el error de la red

$$C(a(Z^L)) = \text{ERROR}$$

RESULTADO DE LA SUMA PONDERADA

FUNCIÓN DE ACTIVACION

FUNCION DE COSTE

Las matemáticas de Backpropagation

Cuando el resultado de la función es pasado por otra y por otra se le conoce como: ***composición de funciones***

Utilizamos "*chain rule*" (regla de la cadena)

Esta nos dice, que para calcular la derivada de una composición de funciones, simplemente tenemos que multiplicar cada una de las derivadas intermedias

$\frac{\partial perro}{\partial gato} = 3$ $\frac{\partial gato}{\partial raton} = 10$

$\frac{\partial perro}{\partial raton} = 30$

$$\frac{\partial perro}{\partial raton} = \frac{\partial perro}{\partial gato} \cdot \frac{\partial gato}{\partial raton}$$

TRUCO: Encadenar numerador y denominador

Las matemáticas de Backpropagation

Entonces, nuestra derivada w respecto al coste y b respecto al coste influyen a través de esta composición

$$\frac{\partial C}{\partial w^L} \qquad \frac{\partial C}{\partial b^L} \qquad Z^L = W^L a^{L-1} + b^L \qquad C(a(Z^L))$$

- Si aplicamos la "chain rule" para resolver estas derivadas, necesitamos calcular todas estas derivadas intermedias:

$$\frac{\partial C}{\partial w^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial w^L} \qquad \frac{\partial C}{\partial b^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial b^L}$$

Las matemáticas de Backpropagation

$$\frac{\partial C}{\partial a^L}$$

Derivada de activación con respecto al coste: Estamos hablando de cómo varia el coste de la red cuando variamos un poco el output, la activación de las neuronas en la última capa

Nos pide calcular la derivada de la función de coste con respecto a la salida de la red neuronal (última capa)

FUNCION DE COSTE
ERROR CUADRÁTICO MEDIO

$$C\left(a_j^L\right) = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

La derivada de la función con respecto al output de la red es:

$$\frac{\partial C}{\partial a_j^L} = (a_j^L - y_j)$$

Las matemáticas de Backpropagation

$$\frac{\partial a^L}{\partial z^L}$$

Derivada de activación con respecto z. Esta nos dice como varia el output de la neurona cuando varia la suma ponderada de la neurona

Lo único que separa a z con la activación de la neurona es la función de activación

Nos piden calcular la derivada de la función de activación

FUNCION DE ACTIVACIÓN
SIGMOIDE

$$a_j^L(z^L) = \frac{1}{1+e^{-z^L}}$$

Su derivada es:

$$\frac{\partial a^L}{\partial z^L} = a^L(z^L) \cdot (1 - a^L(z^L))$$

Las matemáticas de Backpropagation

$$\frac{\partial z^L}{\partial w^L} \quad \frac{\partial z^L}{\partial b^L}$$

- Como varia la suma ponderada z con respecto a una variación de los parámetros (w y b)
- Calcular ambas derivadas
- Nos piden calcular la derivada de la función de activación

DERIVANDO LA SUMA PONDERADA

$$z^L = \sum_i a_i^{L-1} w_i^L + b^L$$

$$\frac{\partial z^L}{\partial b^L} = 1$$

La suma ponderada con respecto al término de sesgo es 1, porque el término del sesgo es independiente, por lo tanto la derivada es constante

$$\frac{\partial z^L}{\partial w^L} = a_i^{L-1}$$

La derivada es el valor de entrada a la neurona que conecta a esa conexión donde el parámetro hace referencia (output) de la salida de las neuronas de la capa anterior (CAPA L-1)

Las matemáticas de Backpropagation

Finalmente la solución que estamos buscando para los parámetros de la última capa se calcula (computadora) donde cada una de las derivadas parciales que hemos derivado se multiplican unas con otras

$$\frac{\partial C}{\partial w^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial w^L} \qquad \frac{\partial C}{\partial b^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial b^L}$$

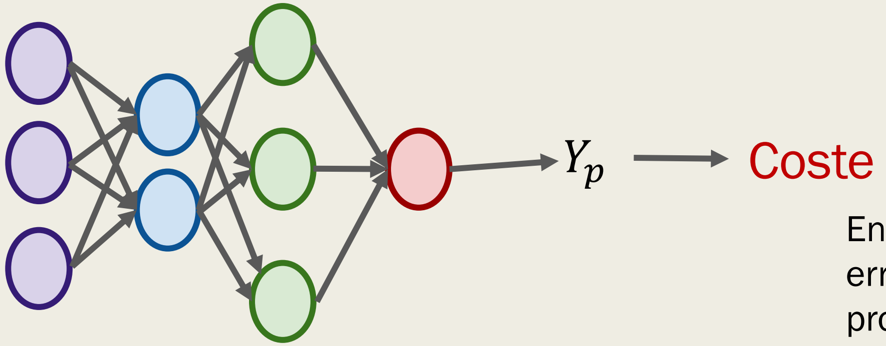
$$\frac{\partial a^L}{\partial z^L} = a^L(z^L) \cdot (1 - a^L(z^L)) \qquad \frac{\partial C}{\partial a_j^L} = (a_j^L - y_j) \qquad \frac{\partial z^L}{\partial b^L} = 1 \qquad \frac{\partial z^L}{\partial w^L} = a_i^{L-1}$$

Las matemáticas de Backpropagation

$$\frac{\partial C}{\partial w^L} = \underbrace{\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}}_{\frac{\partial C}{\partial z^L}} \cdot \frac{\partial z^L}{\partial w^L}$$

Representa cómo varia el error en función del valor de z

Suma ponderada calculada dentro de la neurona



En que grado se modifica el error(el coste) cuando se produce un pequeño cambio en la suma de la neurona

Nos dirá que responsabilidad tiene la neurona en el resultado final y por lo tanto en el error

$$\frac{\partial C}{\partial z^L}$$

- Si la derivada **grande** es que ante un pequeño cambio en el valor de la neurona esta se verá reflejado en el resultado final
- Si la derivada es **pequeña** da igual como variemos el valor de la suma, ya que no afectará al error de la red

$$\frac{\partial C}{\partial z^L}$$

ERROR IMPUTADO A LA NEURONA: δ^L

Las matemáticas de Backpropagation

Simplificando, podemos reestructurar nuestra expresión inicial en función del error de las neuronas de la *CAPA L*

$$\frac{\partial C}{\partial w^L} = \boxed{\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}} \cdot \frac{\partial z^L}{\partial w^L}$$

$$\frac{\partial C}{\partial b^L} = \boxed{\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}} \cdot \frac{\partial z^L}{\partial b^L}$$

$\longrightarrow$

$$\frac{\partial C}{\partial w^L} = \delta^L \cdot \frac{\partial z^L}{\partial w^L}$$

$$\frac{\partial C}{\partial b^L} = \delta^L \cdot \frac{\partial z^L}{\partial b^L}$$

La derivada del coste respecto al término de BIAS es igual al error de las neuronas

$$\frac{\partial C}{\partial b^L} = \delta^L \cdot 1 = \delta^L$$

La derivada del coste respecto w es igual al error de las neuronas multiplicado por la activación de la capa previa

$$\frac{\partial C}{\partial w^L} = \delta^L \cdot a_i^{L-1}$$

Las matemáticas de Backpropagation

Ya hemos deducido tres expresiones diferentes que nos permiten obtener las derivadas parciales que estamos buscando para la última capa

DERIVADA FUNCION DE COSTE

$$\delta^L = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}$$

DERIVADA FUNCION DE ACTIVACION

Nos dice como calcular el error de las neuronas en la última capa

$$\frac{\partial C}{\partial b^L} = \delta^L$$

$$\frac{\partial C}{\partial w^L} = \delta^L \cdot a_i^{L-1}$$

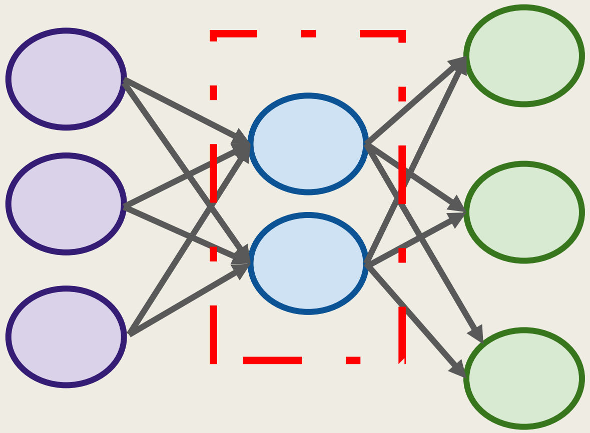
Cada una de las derivadas parciales

¿Tenemos que hacer esto para cada capa?

Las matemáticas de Backpropagation

Aparte de las tres expresiones anteriores, solo necesitamos una más para poder calcular el resto de derivadas de la red

Si ahora queremos calcular los parámetros de la capa anterior, repetimos el mismo razonamiento



CAPA L-1

Las matemáticas de Backpropagation

Aparte de las tres expresiones anteriores, solo necesitamos una más para poder calcular el resto de derivadas de la red

Si ahora queremos calcular los parámetros de la capa anterior, repetimos el mismo razonamiento

APLICAMOS LA CHAIN RULE A ESTA DESCOMPOSICIÓN

$$C(a^L(W^L a^{L-1}(W^{L-1} a^{L-2} + b^{L-1}) + b^L))$$

$$\frac{\partial C}{\partial w^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial w^{L-1}}$$

$$\frac{\partial C}{\partial b^{L-1}} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \frac{\partial z^{L-1}}{\partial b^{L-1}}$$

- Coste
- Activación de la última capa
- La suma ponderada
- La transformación a la capa anterior
- Función de activación

.... Pero como vamos hacia atrás

Las matemáticas de Backpropagation

APLICAMOS LA CHAIN RULE A ESTA DESCOMPOSICIÓN

$$C(a^L(W^L a^{L-1}(W^{L-1}a^{L-2} + b^{L-1}) + b^L))$$

Matriz de parámetros que conecta ambas capas

$$\frac{\partial C}{\partial w^{L-1}} = \underbrace{\frac{\partial C}{\partial a^L}\cdot\frac{\partial a^L}{\partial z^L}}_{\delta^L}\cdot\underbrace{\frac{\partial z^L}{\partial a^{L-1}}}_{W^L}\cdot\frac{\partial a^{L-1}}{\partial z^{L-1}}\cdot\underbrace{\frac{\partial z^{L-1}}{\partial w^{L-1}}}_{a^{L-2}}$$

$$\frac{\partial C}{\partial b^{L-1}} = \frac{\partial C}{\partial a^L}\cdot\frac{\partial a^L}{\partial z^L}\cdot\frac{\partial z^L}{\partial a^{L-1}}\cdot\frac{\partial a^{L-1}}{\partial z^{L-1}}\cdot\underbrace{\frac{\partial z^{L-1}}{\partial b^{L-1}}}_{1}$$

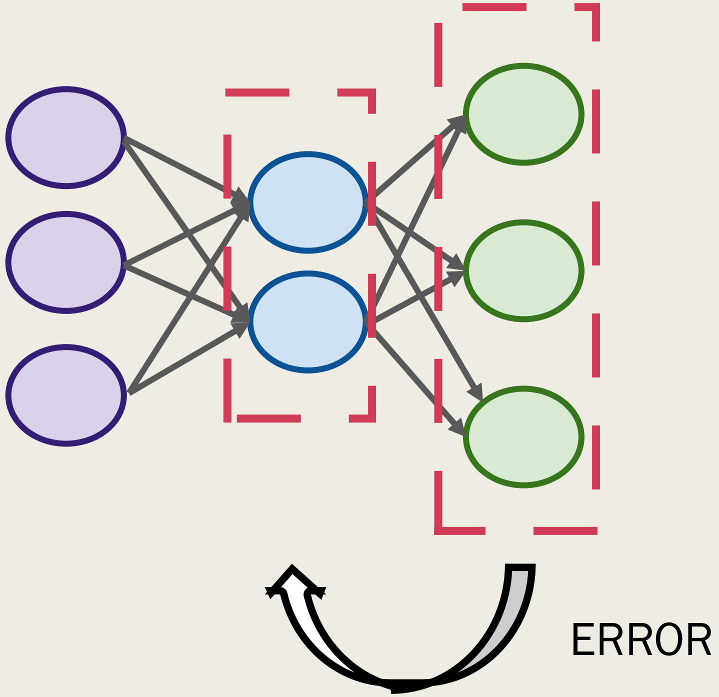
ERROR DE LA CAPA L

DERIVADA DE LA FUNC. ACT

Nos hace falta calcular la derivada que nos habla de **cómo varia la suma ponderada de una capa cuando se varia el output de una neurona en la capa previa**

Las matemáticas de Backpropagation

W^L matriz de parámetros que conecta ambas capas, básicamente lo que hace es mover el error de una capa a la capa anterior distribuyendo el error en función de cuáles son las ponderaciones de las conexiones



Las matemáticas de Backpropagation

Con esto ya tenemos nuevamente una expresión a partir de la cual obtener las derivadas parciales que estamos buscando

$$\frac{\partial C}{\partial w^{L-1}} = \overbrace{\frac{\partial C}{\partial a^{L}} \cdot \frac{\partial a^{L}}{\partial z^{L}}}^{\delta^{L}} \cdot \overbrace{\frac{\partial z^{L}}{\partial a^{L-1}}}^{W^{L}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \overbrace{\frac{\partial z^{L-1}}{\partial w^{L-1}}}^{a^{L-2}}$$

$$\frac{\partial C}{\partial b^{L-1}} = \frac{\partial C}{\partial a^{L}} \cdot \frac{\partial a^{L}}{\partial z^{L}} \cdot \frac{\partial z^{L}}{\partial a^{L-1}} \cdot \frac{\partial a^{L-1}}{\partial z^{L-1}} \cdot \underbrace{\frac{\partial z^{L-1}}{\partial b^{L-1}}}_{1}$$

ERROR DE LA CAPA L

DERIVADA DE LA FUNC. ACT

Este bloque se convierte en esta derivada que vuelve a representar al error de las neuronas en esta capa

$$\frac{\partial C}{\partial z^{L-1}} = \delta^{L-1}$$

Las matemáticas de Backpropagation

Lo que hicimos en esta capa ya es extensible a las otras capas de la red , aplicando la misma lógica:

1. Tomamos el error de la capa anterior

COMPUTO DEL ERROR DE LA ULTIMA CAPA

$$\delta^L = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}$$

2. Lo multiplicamos por la matriz de peso en una transformación que representa la retro propagación de los errores

RETROPROPAGAMOS EL ERROR A LA CAPA ANTERIOR

$$\delta^{l-1} = W^l \delta^l \cdot \frac{\partial a^{l-1}}{\partial z^{l-1}}$$

Las matemáticas de Backpropagation

3. Calculamos las derivadas parciales respecto a los parámetros

CALCULAMOS LAS DERIVADAS DE LA CAPA USANDO EL ERROR

$$\frac{\partial C}{\partial b^{l-1}} = \delta^{l-1} \qquad \frac{\partial C}{\partial w^{l-1}} = \delta^{l-1} a^{l-2}$$

Y así sucesivamente recorriendo todas las capas de la red hasta capa inicial, con esto con un único pase hemos calculado todos los errores y las derivadas parciales de nuestra red haciendo uso de cuatro expresiones

Resumen

Contando cómo tenemos que utilizar el error de la capa anterior para calcular el error en esta capa

Tenemos dos casos diferentes

- Última capa donde el error ya pertenece a la función de coste

$$\delta^L = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}$$

- Resto de capas de nuestra red que dependen de otra capa

$$\delta^{l-1} = W^l \delta^l \cdot \frac{\partial a^{l-1}}{\partial z^{l-1}}$$

Resumen

- Una vez que tenemos estas dos expresiones que nos cuentan cómo podemos calcular el error en la capa actual con respecto al anterior

- Ahora necesitamos dos expresiones que nos cuenten cómo calcular a partir del error en nuestra capa las derivadas parciales para el parámetro de BIAS y el parámetro de pesos

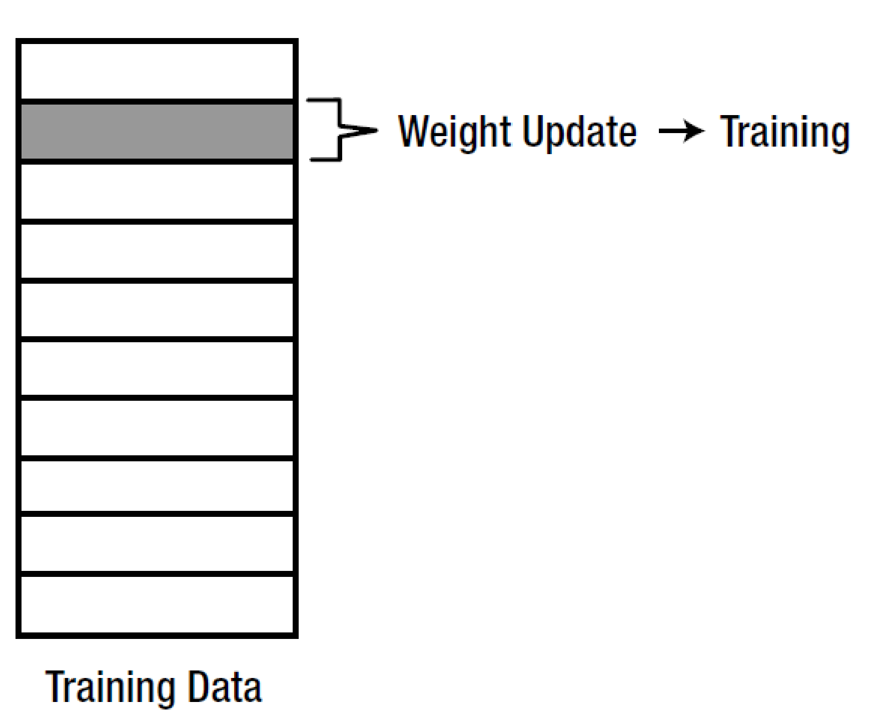
$$\frac{\partial C}{\partial b^{l-1}} = \delta^{l-1} \qquad \frac{\partial C}{\partial w^{l-1}} = \delta^{l-1} a^{l-2}$$

MODOS DE ENTRENAMIENTO DE LAS REDES NEURONALES

Descenso de gradiente estocástico

- En inglés, Stochastic gradient descent (SGD)

Se calcula el error en datos de entrenamiento y de inmediato se ajustan los pesos de la red neuronal



Si tenemos 100 patrones de entrenamiento, haremos 100 ajustes.

Descenso de gradiente estocástico (SGD)

- Conforme se ajustan los pesos por cada punto de entrenamiento, el desempeño de la red neuronal puede alterarse a lo largo del proceso de entrenamiento.

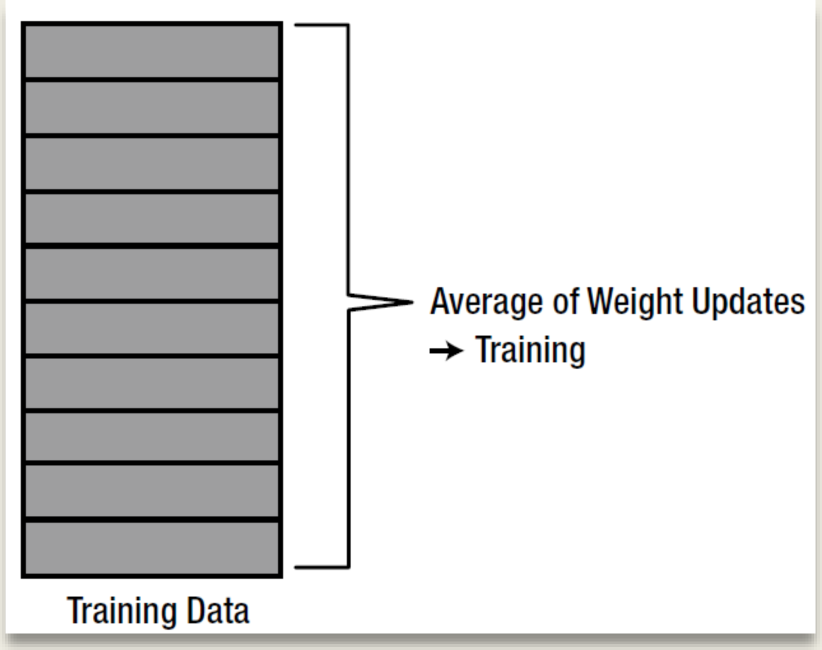
- El término *estocástico* implica el comportamiento aleatorio del proceso de entrenamiento. Por cada patrón, los pesos se ajustarían bajo la regla delta generalizada.

$$\Delta w_{ij} = \alpha \delta_i x_j$$

Batch

La actualización de los pesos se calcula a partir de todos los errores de todos los datos de entrenamiento, el promedio de las actualizaciones de los pesos se utilizará para ajustar los pesos

Este método usará todos los datos de entrenamiento y actualizará los pesos una sola vez.

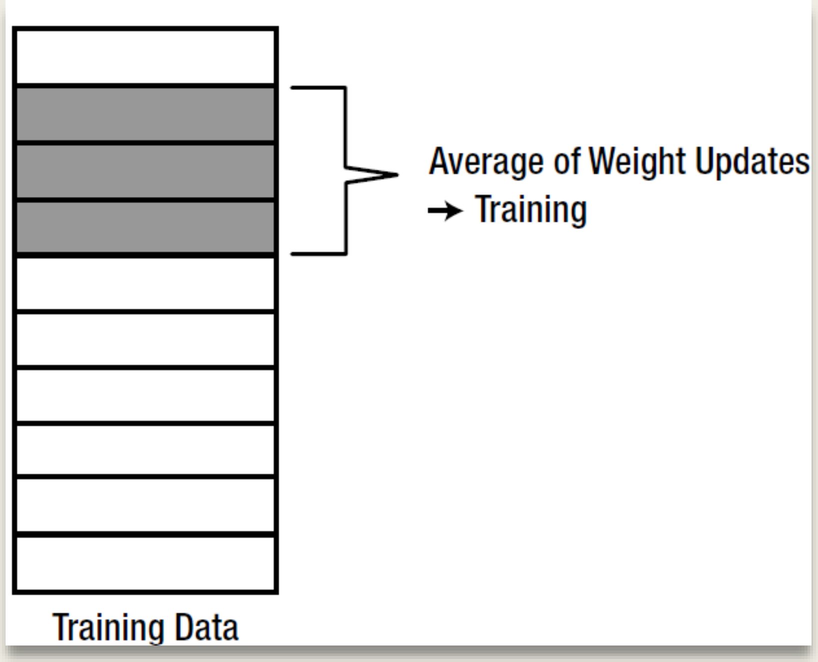


Mini Batch

- Este método es una versión intermedia del SGD y el batch

- Se selecciona una parte del conjunto de entrenamiento y se utiliza con el método batch

- Calcula las actualizaciones de los pesos de los datos seleccionados y entrena la red neuronal con la actualización de pesos promediada.



Mini Batch

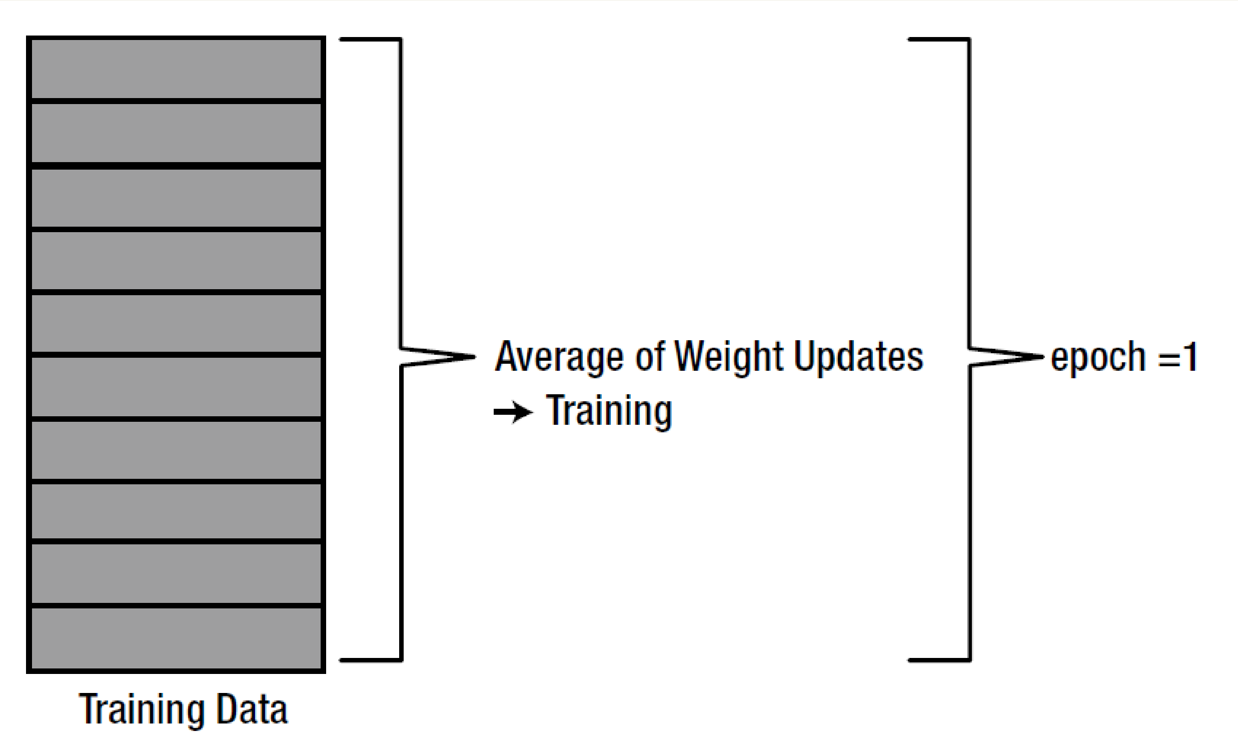
- Si tenemos 100 datos, y aplicamos mini batch de 20; haremos un máximo de 5 ajustes de pesos

- Este método busca aprovechar la velocidad del SGD y la estabilidad del Batch

- Por tal razón es empleado en **Deep Learning**

Épocas o Epochs

- Una época se define como el número de ciclos de entrenamiento completados para todos los datos de aprendizaje

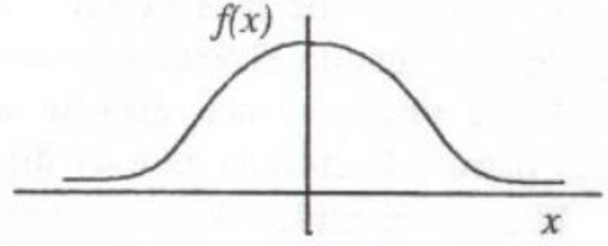


Otros modelos de Redes Neuronales

RBF Network: Redes de Base Radial

- A diferencia de las MLP estas redes solo tienen una capa de entrada, una capa oculta (formada por neuronas con funciones de base radial) y una de salida.

- Las neuronas de la capa oculta en vez de calcular *una suma ponderada de las entradas* y *aplicar una sigmoide*, estas <u>neuronas calculan la distancia euclídea entre el vector de pesos sinápticos</u> (que recibe el nombre en este tipo de redes de centro o centroide) <u>y la entrada</u> (de manera casi análoga a como se hacia con los mapas SOM) <u>y sobre esa distancia se aplica una función de tipo radial con forma gaussiana.</u>



Capa oculta: neurona gaussiana

RBF Network: Redes de Base Radial

- Para el aprendizaje de la capa oculta, hay varios métodos, siendo uno de los más conocidos el algoritmo denominado k-medias (k-means) que es un algoritmo no supervisado de Clustering.

- k es el número de grupos que se desea encontrar, y corresponde con el número de neuronas de la capa oculta, que es un parámetro que hay que decidir de antemano.

- El algoritmo se plantea de la siguiente manera:

1. Calculamos un centroide que representa a una parte del espacio en el que están los datos (clase)

$$\{x_p\}\ donde\ p = 1 \dots n \qquad c_1 = x_1, c_2 = x_2 \dots c_n = x_n$$

RBF Network: Redes de Base Radial

2. En cada iteración, se calculan los dominios, es decir, se reparten las muestras entre los k centros. Esto se hace de la siguiente manera: Dada una muestra x_j se calcula las distancias a cada uno de los centros c_k. La muestra pertenecerá al dominio del centro cuya distancia calculada sea la menor

3. Se calculan los nuevos centros como los promedios de los patrones de aprendizaje pertenecientes a sus dominios. Es como calcular el centro de masas de la distribución de patrones, tomando que todos pesan igual.

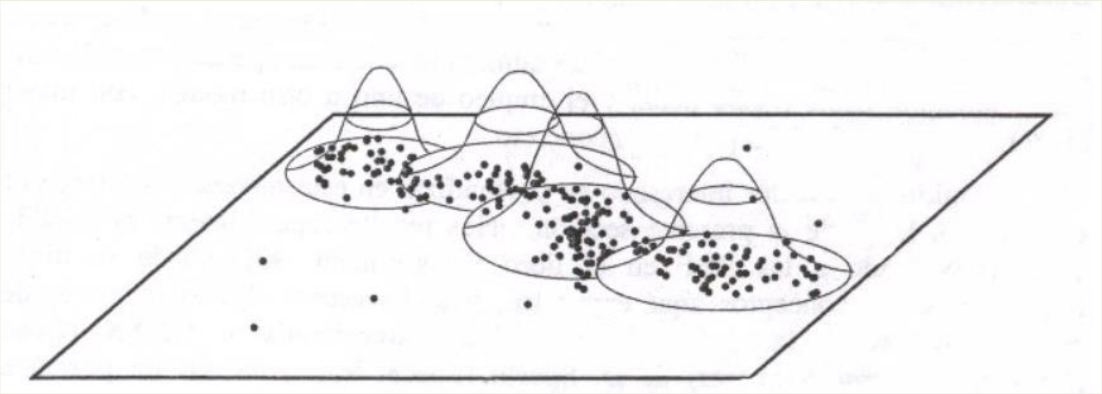
4. Si los valores de los centros varían respecto a la iteración anterior se vuelve al paso 2, si no, es que se alcanzó la convergencia y se finaliza el aprendizaje

RBF Network: Redes de Base Radial

Una vez fijados los valores de los centros, sólo resta ajustar las anchuras de cada neurona

Las anchuras son los parámetros sigma que aparecen en cada una de las funciones gaussianas y reciben ese nombre por su interpretación geométrica

Dan una medida de cuando un muestra activa una neurona oculta para que de una salida significativa normalmente se toma el criterio de que para cada neurona se toma como valor sigma la distancia al centro mas cercano



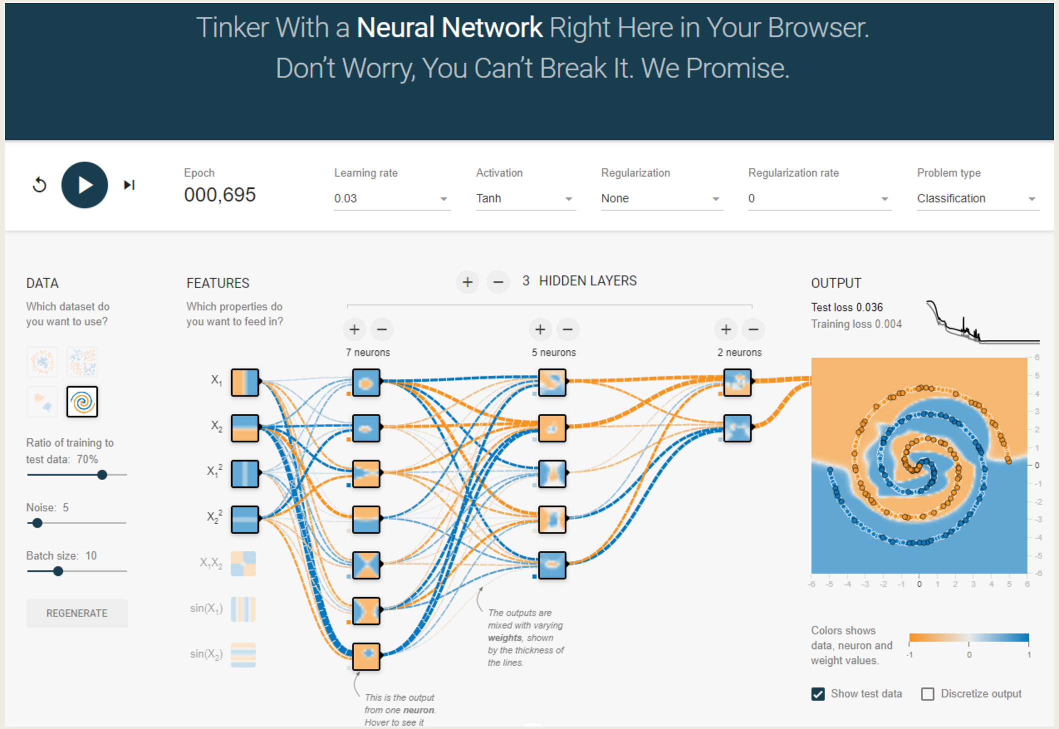
Conclusiones

- El algoritmo de backpropagation derivamos cuatro expresiones que vamos a necesitar para recorrer de atrás hacia adelante nuestra red neuronal para calcular las derivadas parciales que estamos buscando
- Estas expresiones cuentan cómo tenemos que utilizar el error de la capa anterior para calcular el error en esta capa
- El aprendizaje en una red neuronal puede ser visto como el balance (trade-off) entre la minimización del error y la generalización.
- Generalización : capacidad de la red de predecir correctamente la clase de datos que el modelo no conoce.

- No hay garantía:
  - *La red converge a una buena solución (mínimo global).*
  - *La red converge rápidamente.*
  - *La red va a converger algún día.*

DEMO

https://playground.tensorflow.org/



Referencias:

- [1] Leondes, C.T. (2018). Image Processing and Pattern Recognition. California: Academic Press.
- [2] Duda, R.O., Hart, P.E. & Stork, D.G. (2001). Pattern Classification. 2nd edition. Wiley-Interscience.
- [3] Marques de Sá, J:P. (2001). Pattern Recognition: Concepts, Methods and Applications. Berlin: Springer-Verlag.
- [4] Kuncheva, L. (2014). Combining Pattern Classifiers: Methods and Algorithms. 2nd edition. USA: Wiley.
- [5] Witten, I.H., Frank, E. & Hall, M.A. (2011). Data Mining: Practical Machine Learning Tools and Techniques. 3rd edition. USA: Elsevier.
- [6] Murty, N.M. & Devi, V.S. (2011). Pattern Recognition: An Algorithmic Approach. Springer.
- [7] Zaki, M.J. & Meira, W. (2014). Data Mining and Analysis: Fundamental Concepts and Algorithms. Cambridge University Press.

Fecha límite: Viernes 21 de Abril 2023

Tarea 3

1. Utilizando el DEMO diseñar la red neuronal para los dos datasets faltantes
2. Mapa mental del tema de redes neuronales
3. Infografía de aplicaciones de las redes neuronales (un ejemplo por integrante del equipo)