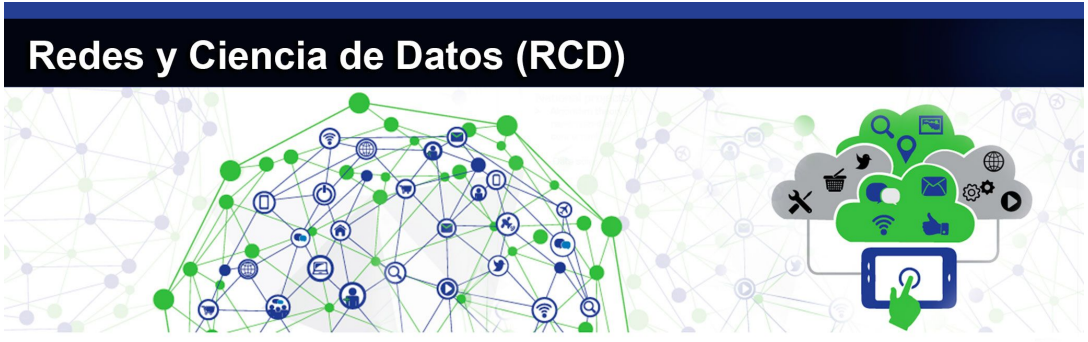
Centro de Investigación en Computación
Instituto Politécnico Nacional

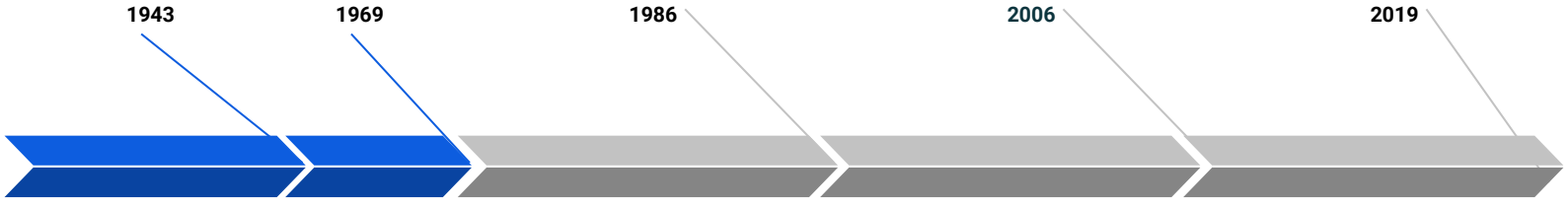
Redes Neuronales Artificiales



Antecedentes

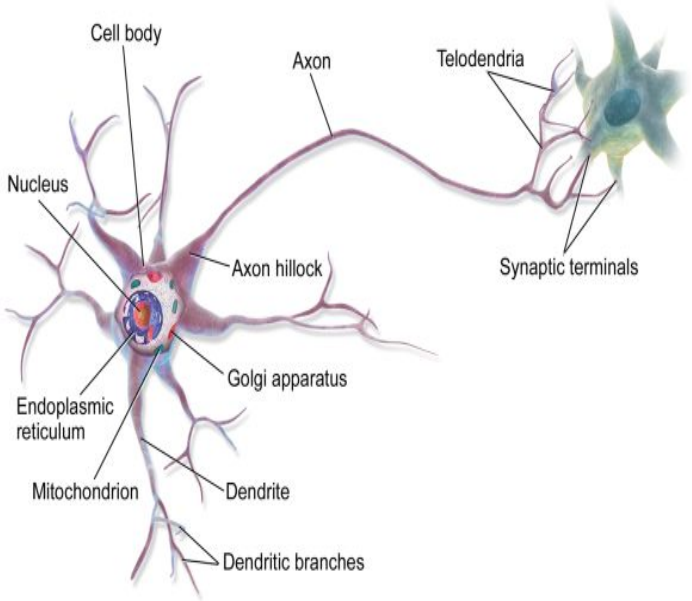
- Fueron propuestas por primer vez en 1943 por Warren McCulloch y Walter Pitts en el artículo “A logical calculus of ideas immanent in nervous activity”. Pero en 1969 entraron en desuso.
- En 1986 con el algoritmo de BackPropagation se revive el interés en el conexionismo. Pero debido a su lentitud y al desarrollo de otros algoritmos, pasan a un segundo plano.
- En 2006 Geoffrey Hinton et al introducen el término Deep Learning en un artículo que logró tener una precisión de más del 98% en el dataset de MNist.

Línea de tiempo de las Redes Neuronales Artificiales

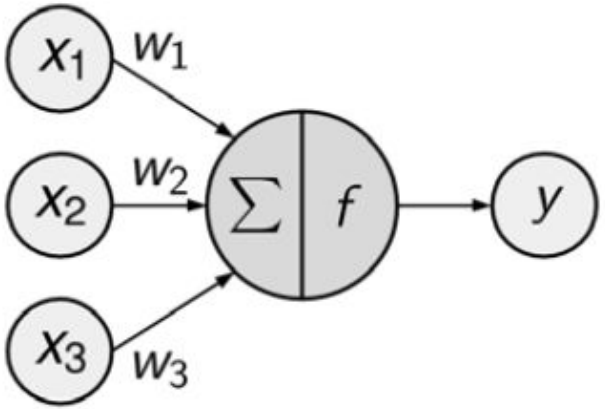


- **Datasets más grandes**
- **Mayor poder de cómputo**
- **Mejorar en los algoritmos***

Modelo biológico



Modelo artificial



Elementos de una red neuronal

- Arquitectura de la red
- Función de activación
- Función de error
- Algoritmo de entrenamiento

Perceptrón

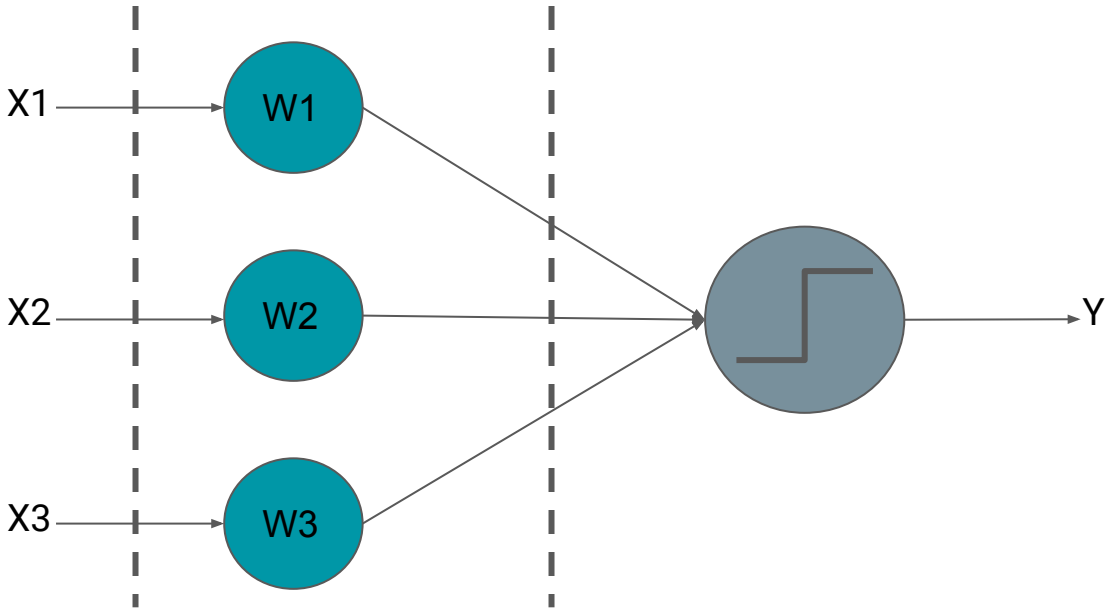
Problema: ¿Aprobar crédito?

Información del aplicante:

Edad	23
Género	Masculino
Salario Anual	$300,000
Años de residencia	1 año
Años en trabajo	1 año
Deuda actual	150,000
...	...

¿Crédito aprobado?

Perceptrón



Perceptrón

Para entrada $\mathrm{X} = (x_1, \cdots, x_d)$

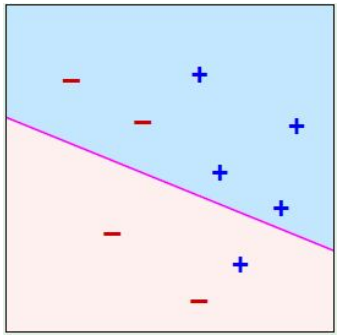
Se aprueba el crédito si: $\sum_{i=1}^{d} w_i x_i > umbral$

Se niega el crédito si: $\sum_{i=1}^{d} w_i x_i < umbral$

$z = w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n = X^T W$

Fórmula lineal:

$$h(x)=sign((\sum_{i=1}^{d} w_i x_i)-b)$$

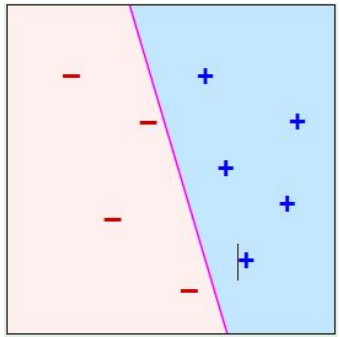


Representación vectorial:

$$h(x)=sign(W^T X+b)$$

Regla de actualización:

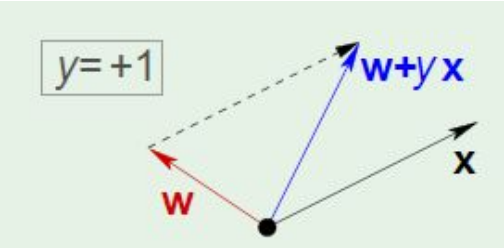
$$w_{i,j}^{(siguiente\ paso)}=w_{i,j}+\eta(y_j-\hat{y}_j)x_i$$

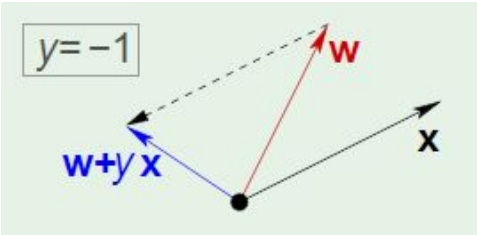


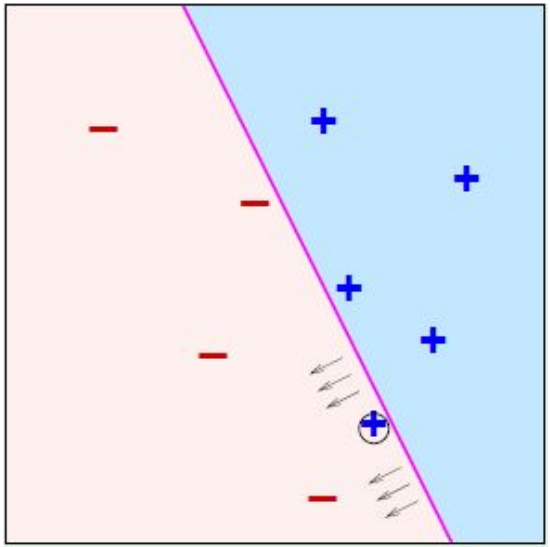
Linealmente separables

Algoritmo del perceptrón

Regla de actualización: $w_{i,j}^{(siguiente\ paso)} = w_{i,j} + \eta\left(y_j - \hat{y}_j\right)x_i$





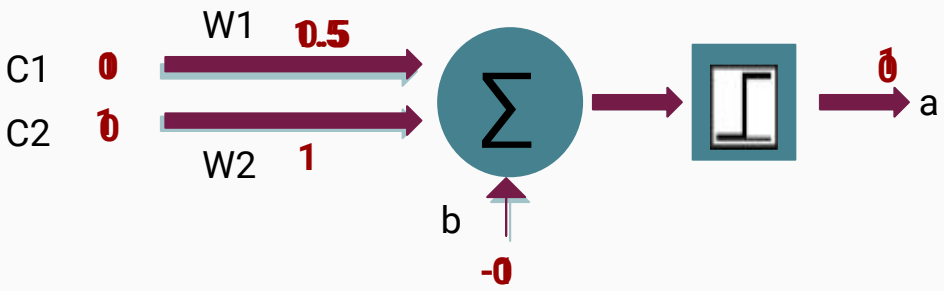


Entradas

Neuronas $[W_1 \quad W_2]$

Forma matricial

$$a = f(WP + b)$$



C1	C2	t
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

¿ $a = t$?

$$a = f\,act.\left([1.5 \quad 1]\begin{bmatrix}1\\1\end{bmatrix} - 1\right)$$

- Actualizamos W y b de la red con el target
- Realizamos la propagación hacia adelante (*feedforward*)

$$e = t - a$$

$$W_{new} = W_{old} + eP^T$$

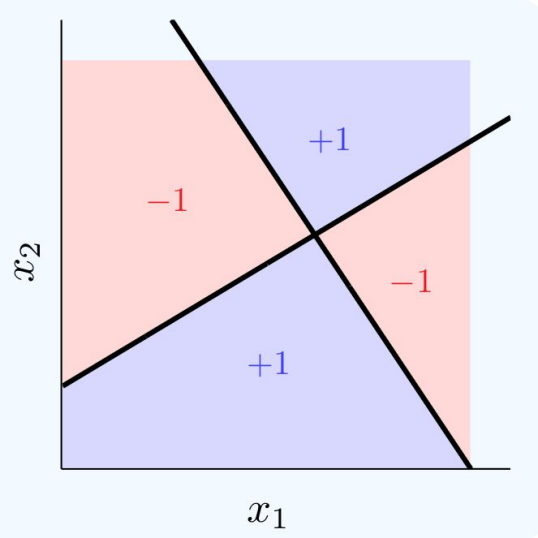
$$W_{new} = [1.5 \quad 1] + [-1][0 \quad 0] = [1.5 \quad 1]$$

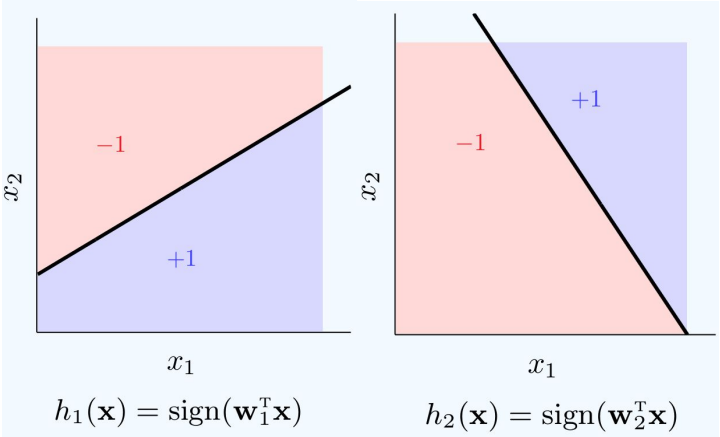
$$b_{new} = b_{old} + e$$

$$b_{new} = -1 + 0 = -1$$

Perceptrón multicapa

Compuerta XOR

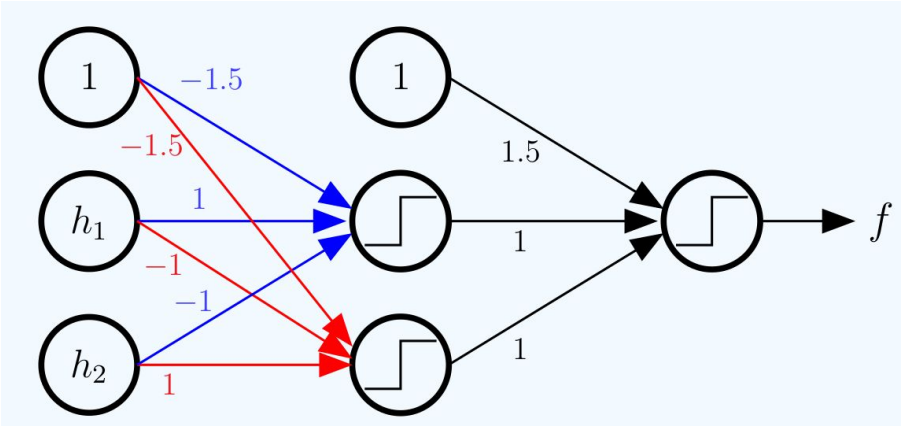




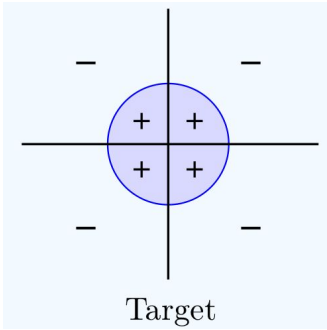
$$f = h_1 \bar{h}_2 + \bar{h}_1 h_2$$

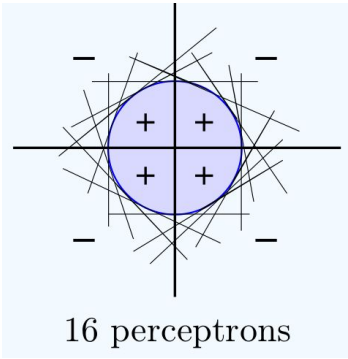
MLP para función XOR

$$f = h_1 \bar{h}_2 + \bar{h}_1 h_2$$

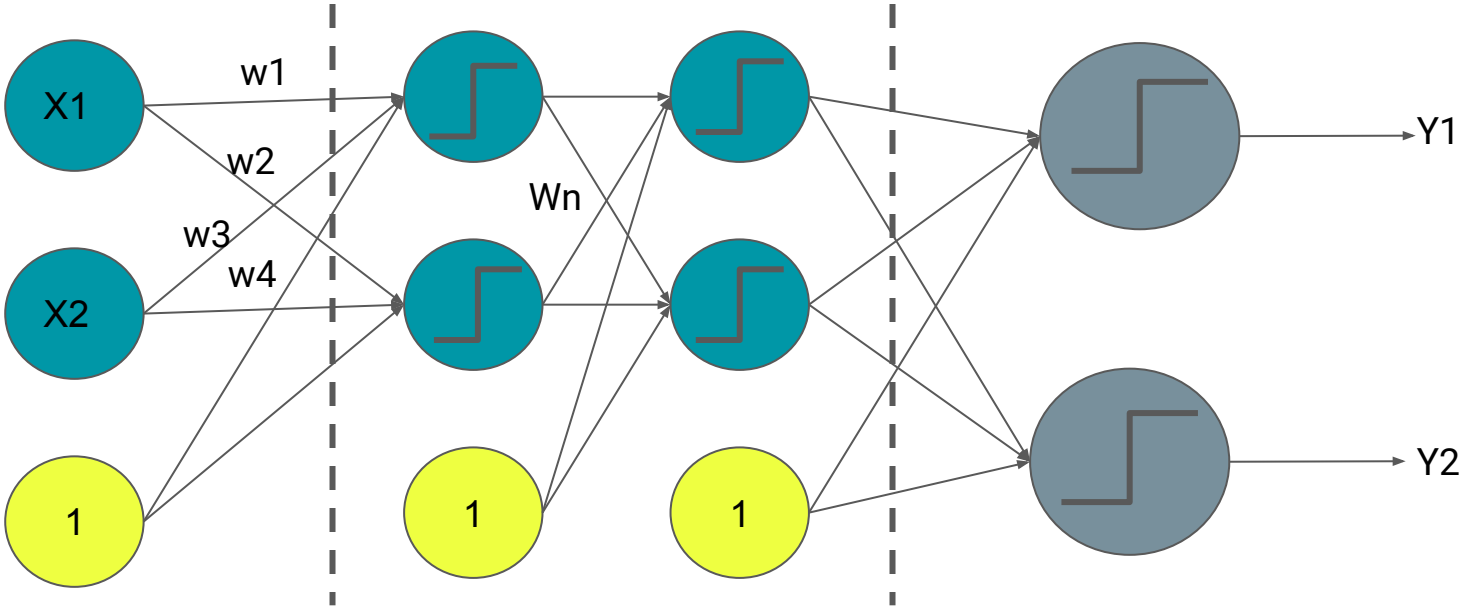


MLP para función no lineal





Perceptrón Multicapa



Funciones de activación

- Introducir propiedades no lineales a la red (Non linearities)
- Funciones lineales (fáciles de resolver)
- Funciones no lineales

¡Redes Neuronales son FUNCIONES DE APROXIMACIÓN UNIVERSAL!

- ¿Qué pasa si no usamos una f.activación no lineal?

R = Se comporta como una sola red de una capa

- Buscamos que sean diferenciables

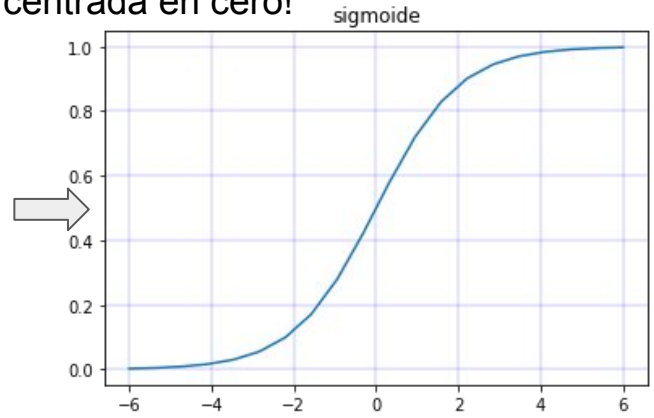
ALGORITMO DE BACKPROPAGATION

GRADIENTE DESCENDENTE

Funciones de activación

Sigmoide

No centrada en cero!

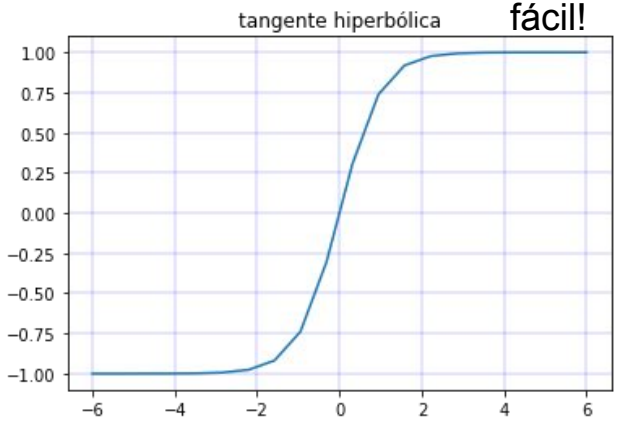


$$f(x)=\frac{1}{1+e^{x}}$$

Se desvanece el gradiente :(

Tangente Hiperbólica

¡Optimización más fácil!



También se desvanece :(

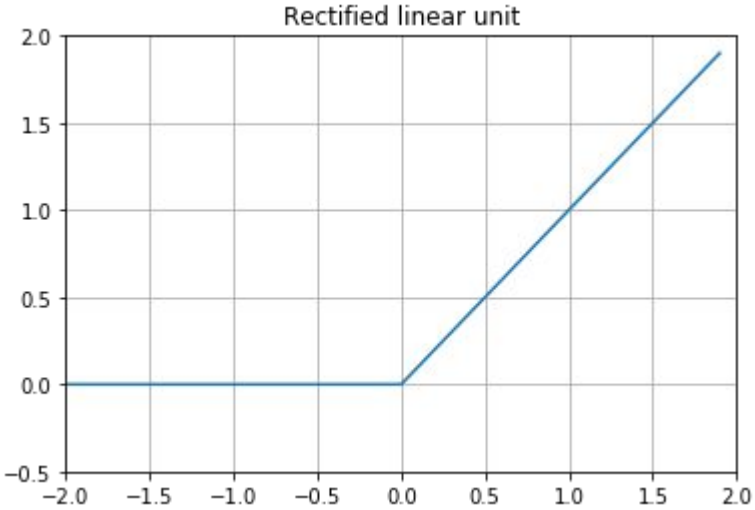
$$f(x)=\frac{2}{1+e^{-2x}}-1$$

- Buen desempeño para redes recurrentes

Funciones de activación

ReLU

¡Sólo para capas ocultas!

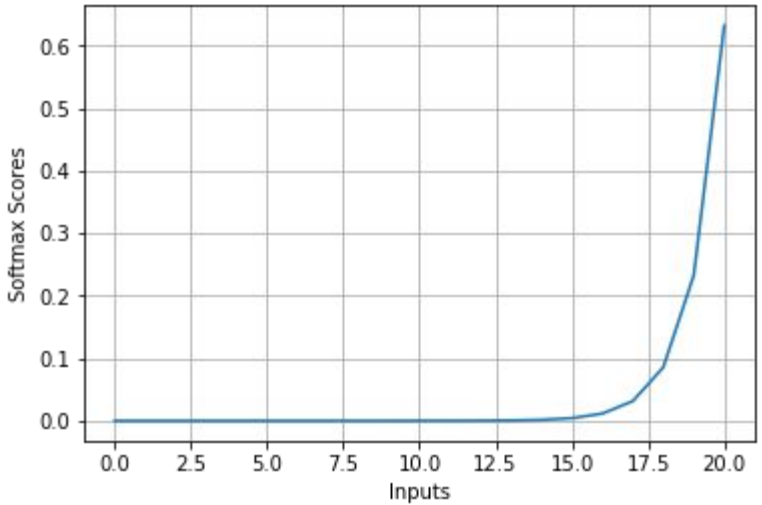


$$f(x) = \max(0, x) = \begin{cases} 0 \; for \;\; x < 0 \\ x \; for \;\; x \geq 0 \end{cases}$$

¡Aprende más rápido!

Softmax

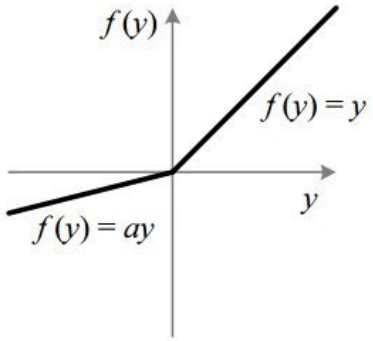
Usado para clasificación



$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Otras funciones de activación

- Leaky RELU, incluye una pequeña pendiente negativa

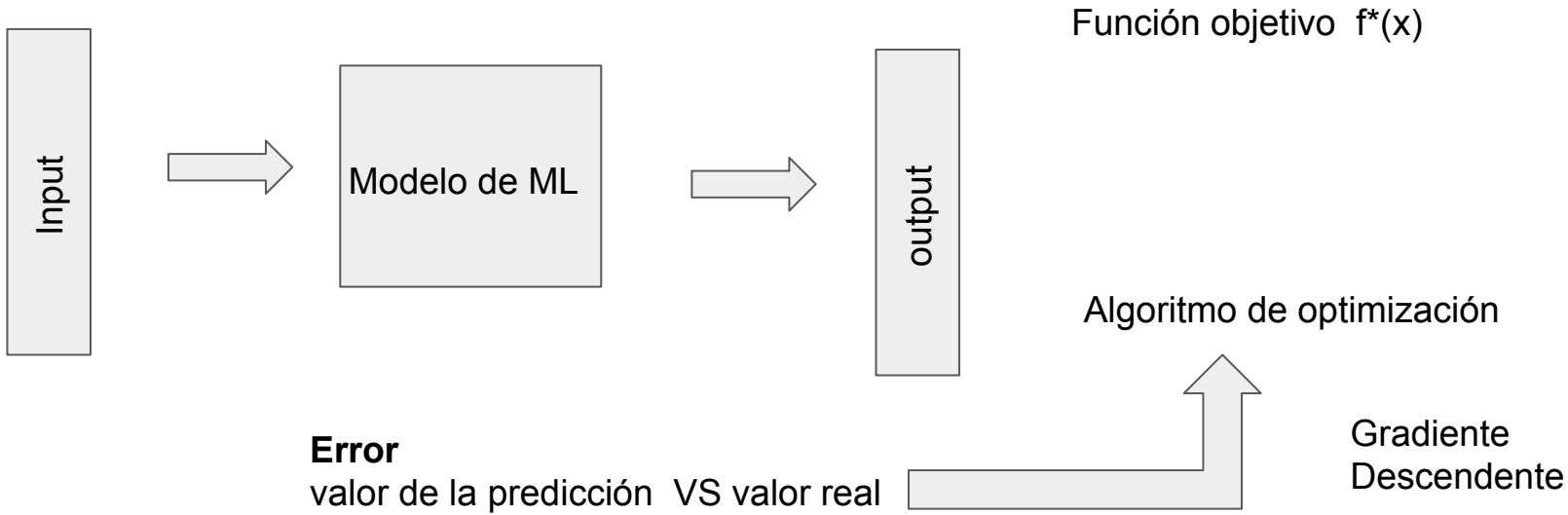


- **Maxout**: Generalización entre RELU y Leaky RELU
  - Dobla el número de parámetros

- Smooth RELU
- ELU
- SELU
- ...

¡Área de investigación!

Funciones de pérdida (loss functions)



Loss Function!

Funciones de pérdida

¿Qué función de pérdida elegir?

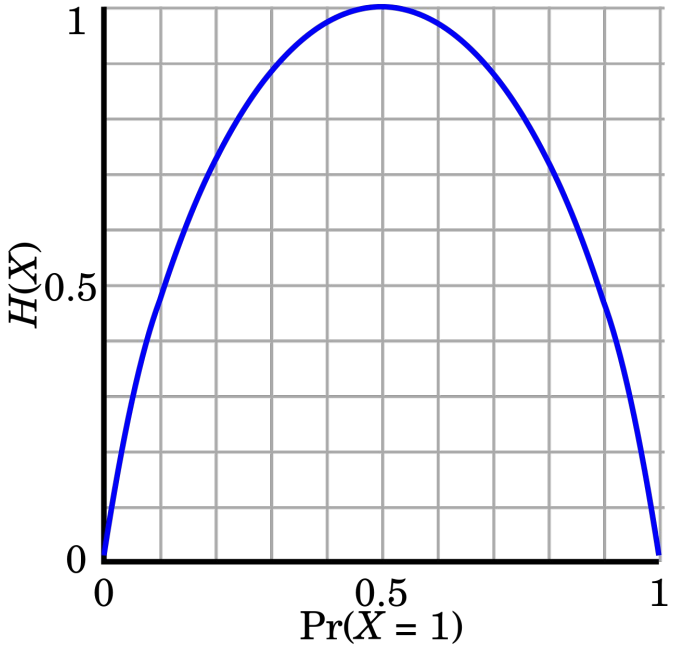
- No es universal
- La presencia de valores atípicos
- El algoritmo de ML
- La eficiencia del algoritmo de GD
- La precisión de las predicciones

CLASIFICACIÓN

REGRESIÓN

Entropía de la información

- **Entropía**: Mide la cantidad de información promedio que obtenemos de una muestra obtenida de una distribución de probabilidad P.
- Indica que tan impredecible es nuestra distribución, **entre más variación tengamos en nuestros datos, mayor la entropía.**



Entropía cruzada

Función entre la verdadera distribución de probabilidad P y las distribución predicha Q

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

Si la distribución Q = P, la entropía cruzada es igual a la entropía

Si P!=Q la entropía cruzada será mayor que la entropía, esa diferencia es la llamada entropía relativa(algunos bits)

Cross Entropy-Clasificación

- Mide el desempeño de nuestro clasificador cuya salida es una distribución de probabilidad
- log-loss incrementa si las distribuciones divergen.
- Modelo ideal:
  - Cross Entropy = 0
- Usa el -log para determinar métrica de comparaciones

Otras funciones para Clasificación

- Función Hinge*
- Square loss
- Logistic loss
- Expontial loss

*Elegir entre precisión y velocidad vs CE

Función de pérdida - REGRESIÓN

MSE Mean Squared Error

Mide la cantidad promedio que los valores calculados por el modelo difieren de los valores reales.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

- Función cuadrática
- Mínimo global

MAE Mean Absolute Error

Mide el promedio sobre la muestra de entrenamiento

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_i|$$

Función de pérdida- REGRESIÓN

MSE

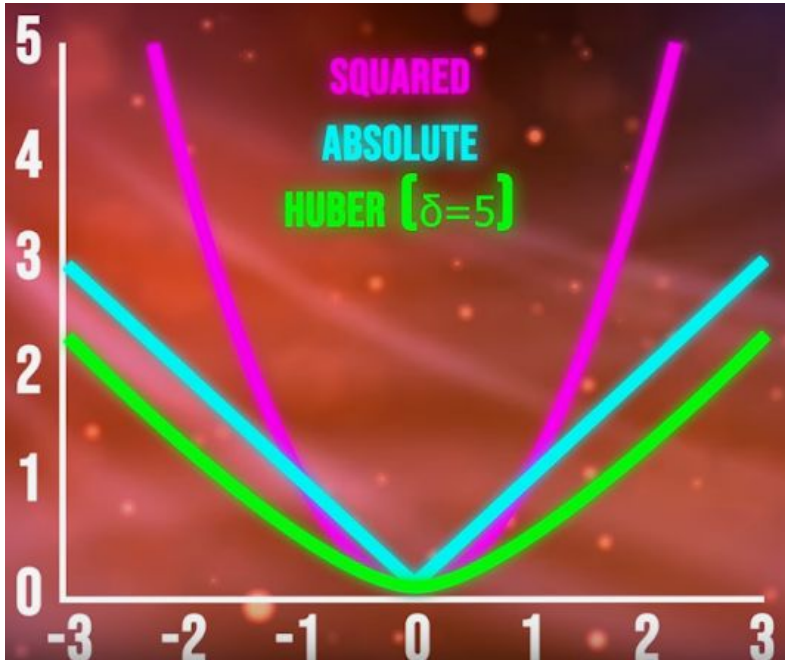
- Penaliza desviaciones grandes

MAE

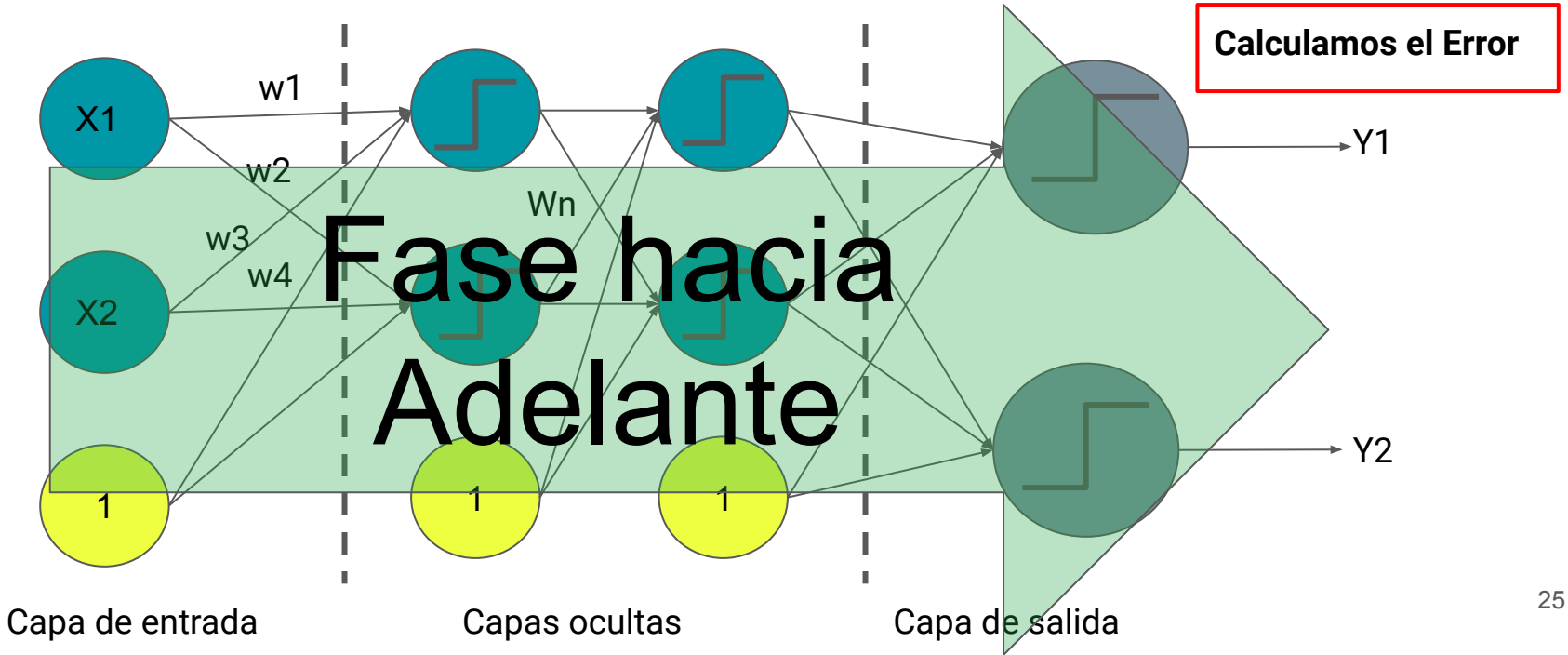
- Más robusta a valores atípicos
- Asigna valores sin ponderación

HUBER

- Similar a MAE y MSE
  - Cuadrática para valores pequeños
  - Lineal para valores grandes



BackPropagation

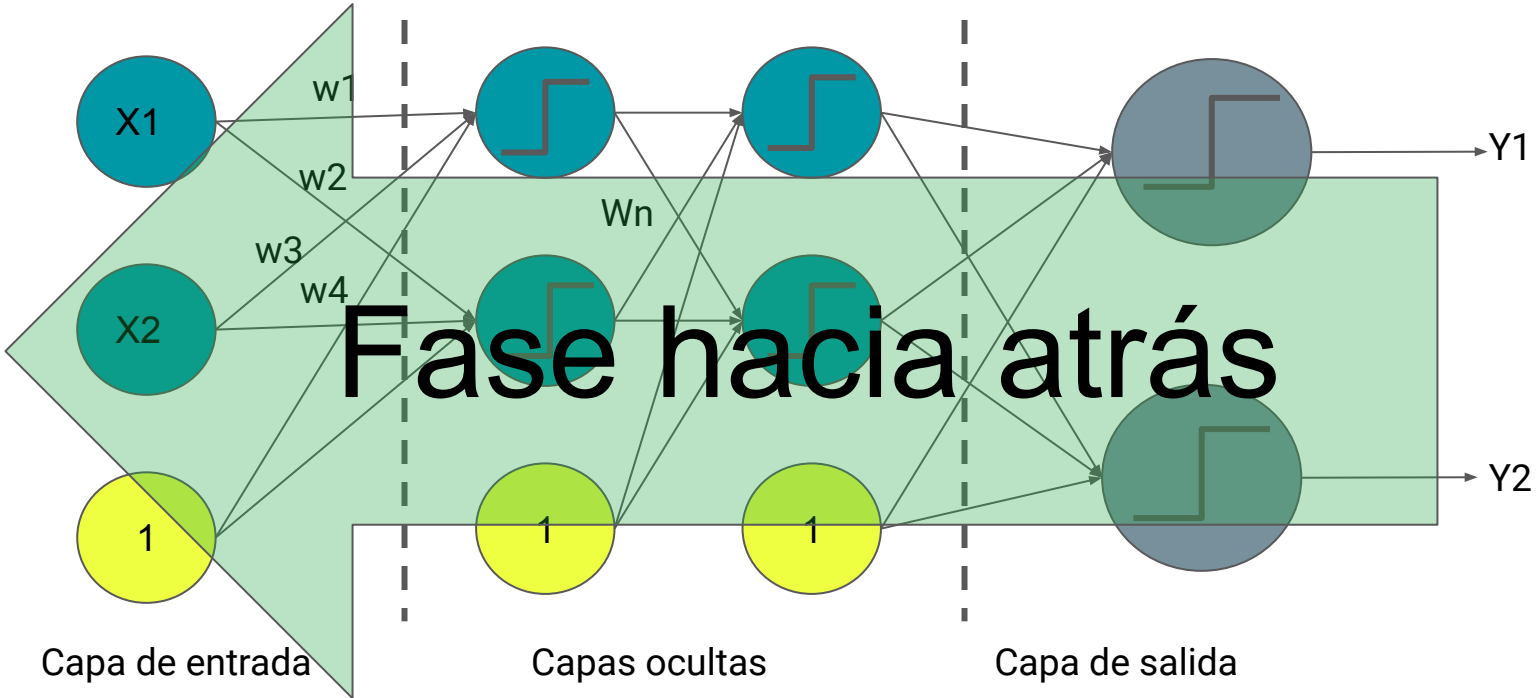


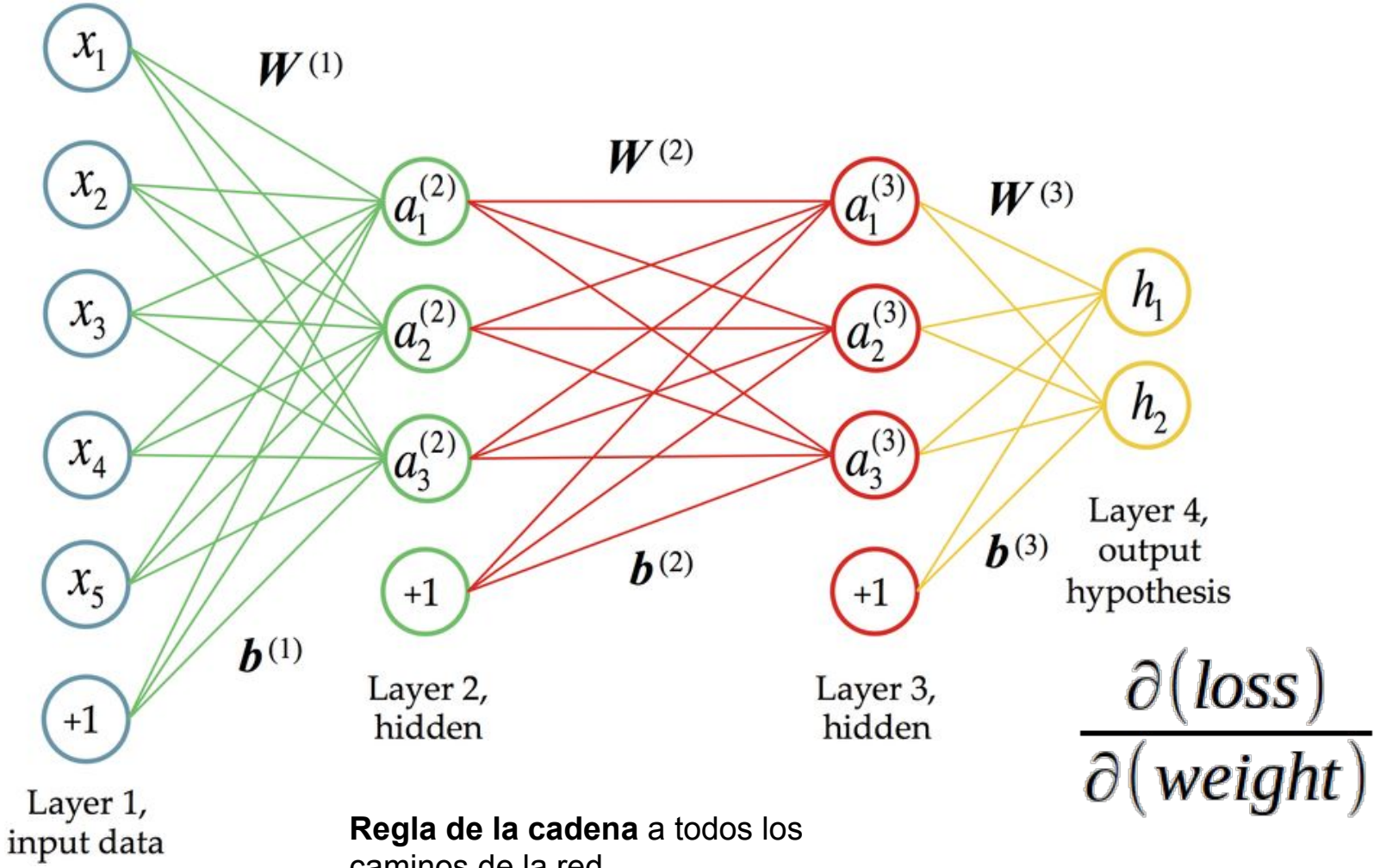
BackPropagation

$$W_{new} = W_{old} - \nabla f(w_{old}) \times learning\ rate$$

$$\nabla f(w_{old}) = \frac{\partial(loss)}{\partial(weight)}$$

Actualizamos los pesos





Regla de la cadena a todos los caminos de la red

Algoritmo de Backpropagation

1. Tomar los datos como :Batch, mini-batches (épocas)
2. Pasar los datos por la red hasta la capa de salida, fase de forward hasta obtener el resultado.
3. Medir el error de la salida de la neurona (loss function)
4. Calcular el error de la capa anterior en cada conexión en la red (regla de la cadena) hasta la capa de entrada.
    a. La medida del error se calcula con el gradiente de los pesos, propagando el error al revés por toda la red
5. El algoritmo ejecuta Gradiente Descendiente para ajustar todos los pesos en la red

Back propagation

Algoritmo de optimización:

Actualizar parámetros para minimizar el error

- Gradiente Descendente Estocástico (SGD)
- Mini- Batch Gradiente Descendiente *
- Momentum
- ADAM (Adaptive Moment Estimation)