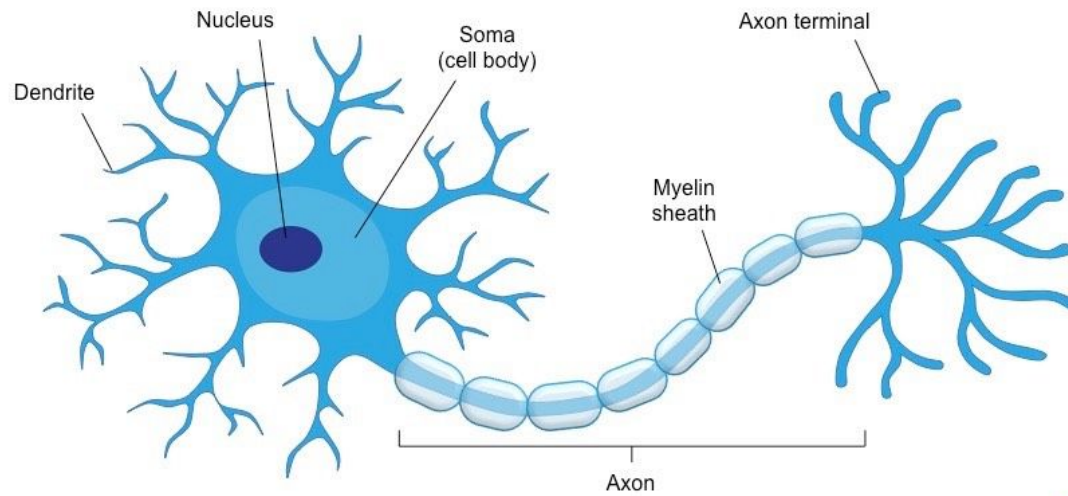


# Redes Neuronales para el análisis y la generación de texto

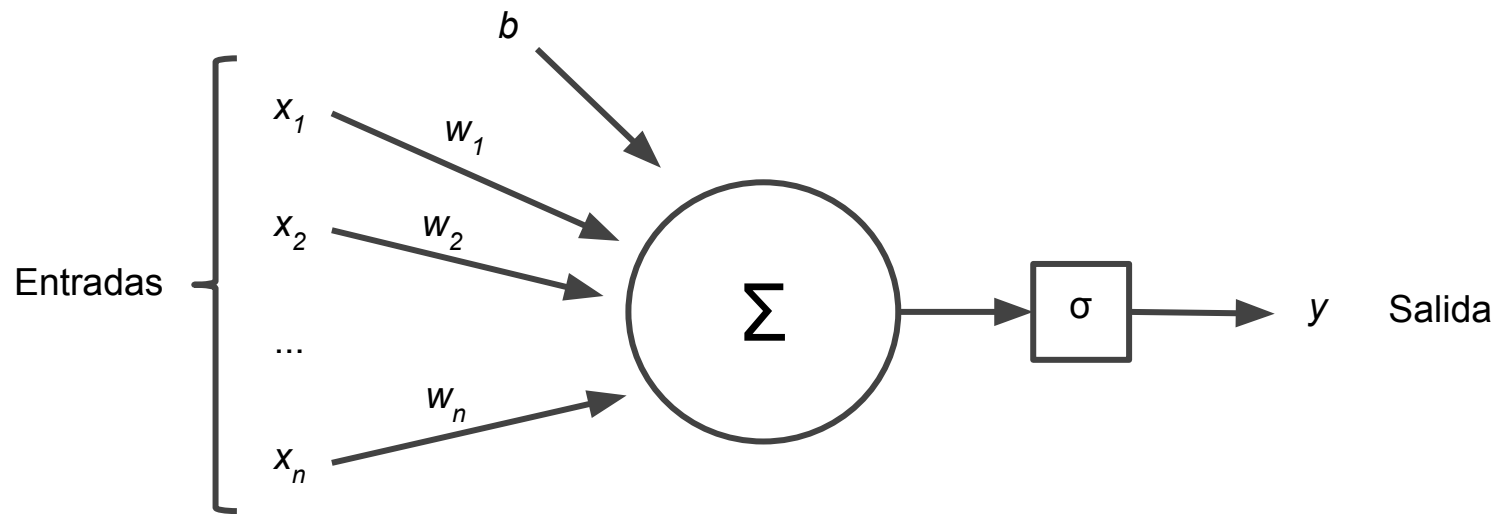
Grupo PLN  
InCo- Fing- UdelaR

ELI - IV Escuela Latinoamericana de Informática  
Octubre 2025 - Valparaíso

# Neurona

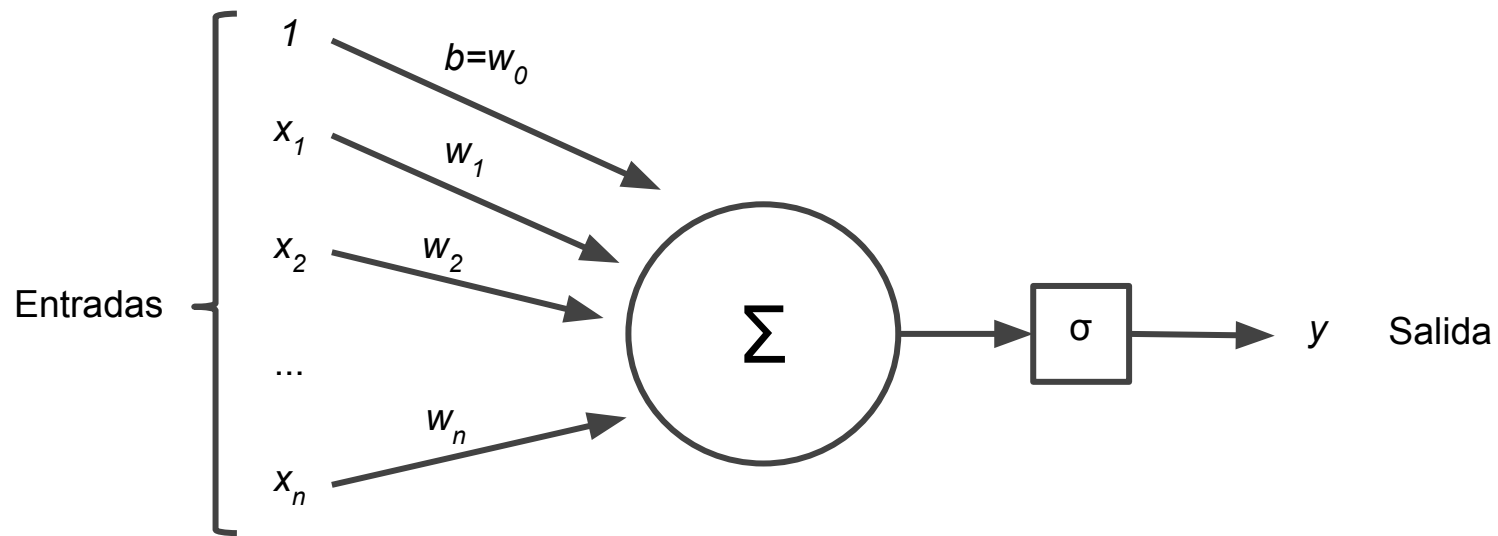


# Neurona Artificial



$$y = \sigma\left(\sum_i x_i w_i + b\right)$$

# Neurona Artificial



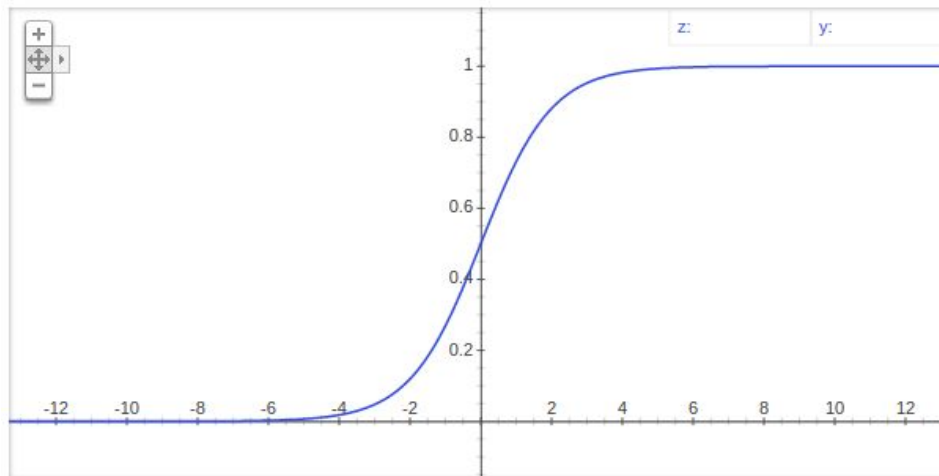
$$\hat{x} = [1, x_1, x_2, \dots, x_n]$$

$$\hat{w} = [w_0, w_1, w_2, \dots, w_n]$$

$$y = \sigma(\hat{x} \cdot \hat{w})$$

# Una Función de Activación

Función sigmoide o logística:  $\sigma(z) = \frac{1}{1 + e^{-z}}$



# Ejemplo: Análisis de Sentimiento

Las entradas de mi red son números, y la salida también

¿Cómo hago para representar mi texto mediante números?

Alternativa 1: Extracción de features (manuales)

Lista de palabras positivas  $P = \{\text{buenísima, buena, gustó, linda, recomiendo}\}$

Lista de palabras negativas  $N = \{\text{horrible, divague, mamarracho}\}$

$x_1$  = cantidad de palabras de la lista  $P$  en el texto

$x_2$  = cantidad de palabras de la lista  $N$  en el texto

$x_3$  = la palabra “no” aparece en el texto

$x_4$  = el signo de exclamación “!” aparece en el texto

$x_5$  = cantidad de palabras en el texto

...

# Ejemplo: Análisis de Sentimiento

*Escenas traídas de los pelos. No la recomiendo.*

$P = \{\text{buenísima, buena, gustó, linda, recomiendo}\}$

$N = \{\text{horrible, divague, mamarracho}\}$

$x_1 = \text{cantidad de palabras de la lista } P \text{ en el texto} = 1$

$x_2 = \text{cantidad de palabras de la lista } N \text{ en el texto} = 0$

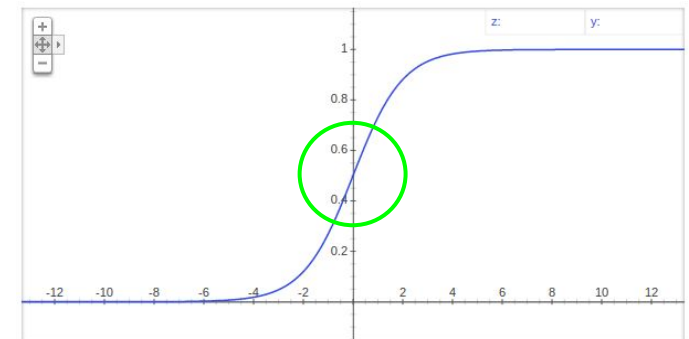
$x_3 = \text{la palabra "no" aparece en el texto} = 1$

$x_4 = \text{el signo de exclamación "!" aparece en el texto} = 0$

$x_5 = \text{cantidad de palabras en el texto} = 8$

$w = [0.5, 0.6, -0.8, 0.3, -0.1]$

$$\frac{1}{1 + e^{-x \cdot w}} = 0.2497$$



# Ejemplo: Análisis de Sentimiento

*Escenas traídas de los pelos. No la recomiendo.*

$P = \{\text{buenísima, buena, gustó, linda, recomiendo}\}$

$N = \{\text{horrible, divague, mamarracho}\}$

$x_1 = \text{cantidad de palabras de la lista } P \text{ en el texto} = 1$

$x_2 = \text{cantidad de palabras de la lista } N \text{ en el texto} = 0$

$x_3 = \text{la palabra "no" aparece en el texto} = 1$

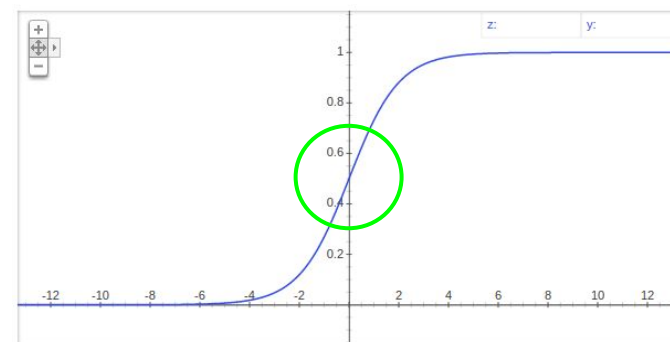
$x_4 = \text{el signo de exclamación "!" aparece en el texto} = 0$

$x_5 = \text{cantidad de palabras en el texto} = 8$

$w = [0.5, 0.6, -0.8, 0.3, -0.1]$

$$\frac{1}{1 + e^{-x \cdot w}} = 0.2497$$

?





# Entrenamiento

¿Cómo aprendo los mejores pesos  $w$ ?

Conjunto de entrenamiento, ejemplos:

$$t^{(1)} \rightarrow y^{(1)}$$

$$t^{(2)} \rightarrow y^{(2)}$$

...

$$t^{(m)} \rightarrow y^{(m)}$$

Siendo cada  $t^{(i)}$  un texto, cada  $y^{(i)}$  vale 0 o 1

Pero de cada ejemplo  $t^{(i)}$  extraigo el vector de features  $x_1^{(i)}, x_2^{(i)}, \dots, x_k^{(i)} = \mathbf{x}^{(i)}$

# Función de pérdida (Loss)

Necesito una forma de relacionar los valores obtenidos con mi red y los valores esperados

Para cierto ejemplo x:

$$\hat{y} = \sigma(x \cdot w)$$

y es el gold standard

Para estos casos en que y puede tomar los valores 0 o 1, se suele usar la entropía cruzada

$$\begin{aligned} L_{CE} &= -\frac{1}{m} \sum_{i=1}^m \left[ y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right] \\ &= -\frac{1}{m} \sum_{i=1}^m \left[ y^{(i)} \log(\sigma(x^{(i)} \cdot w)) + (1 - y^{(i)}) \log(1 - \sigma(x^{(i)} \cdot w)) \right] \end{aligned}$$

# Formalización del problema

Se suele denominar  $\theta$  al conjunto de parámetros de una familia de funciones de aprendizaje automático (en este caso una red neuronal)

En el ejemplo, nuestro  $\theta$  es  $w$

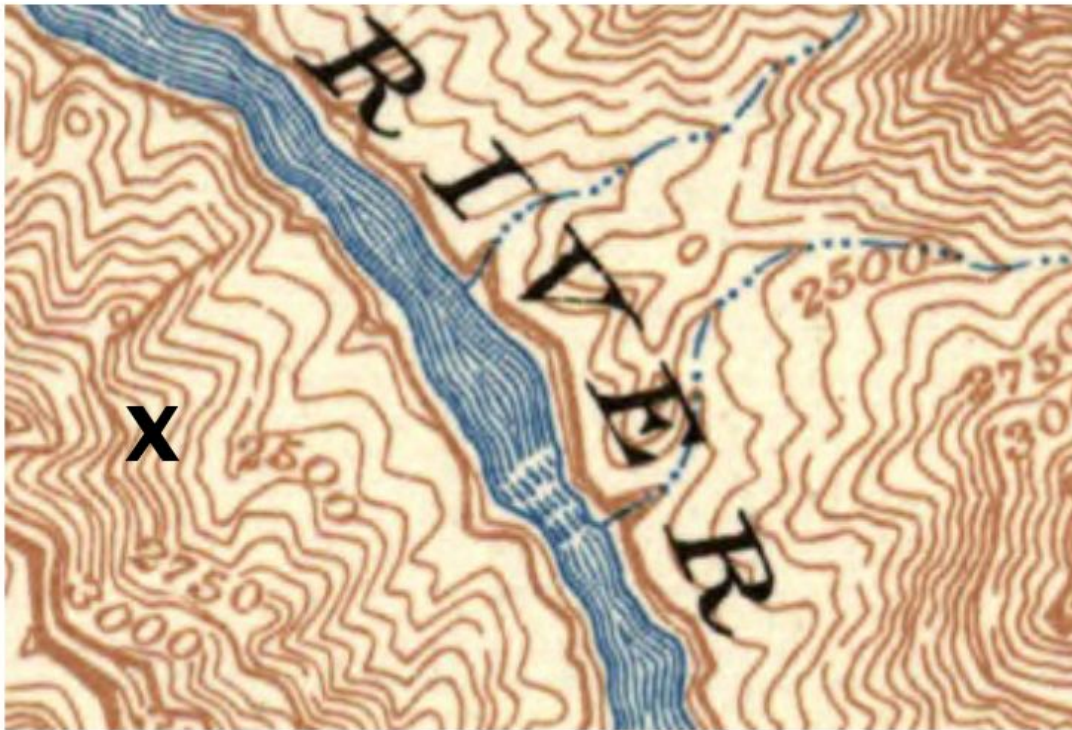
Entonces la función a aprender es  $\hat{y} = f(x; \theta)$

Queremos encontrar el conjunto de parámetros que haga mínimo lo siguiente

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^m L_{\text{CE}}(f(x^{(i)}; \theta), y^{(i)})$$

# ¿Cómo encuentro el mínimo de mi función de pérdida?

Cómo llegar al fondo del cañón del río?

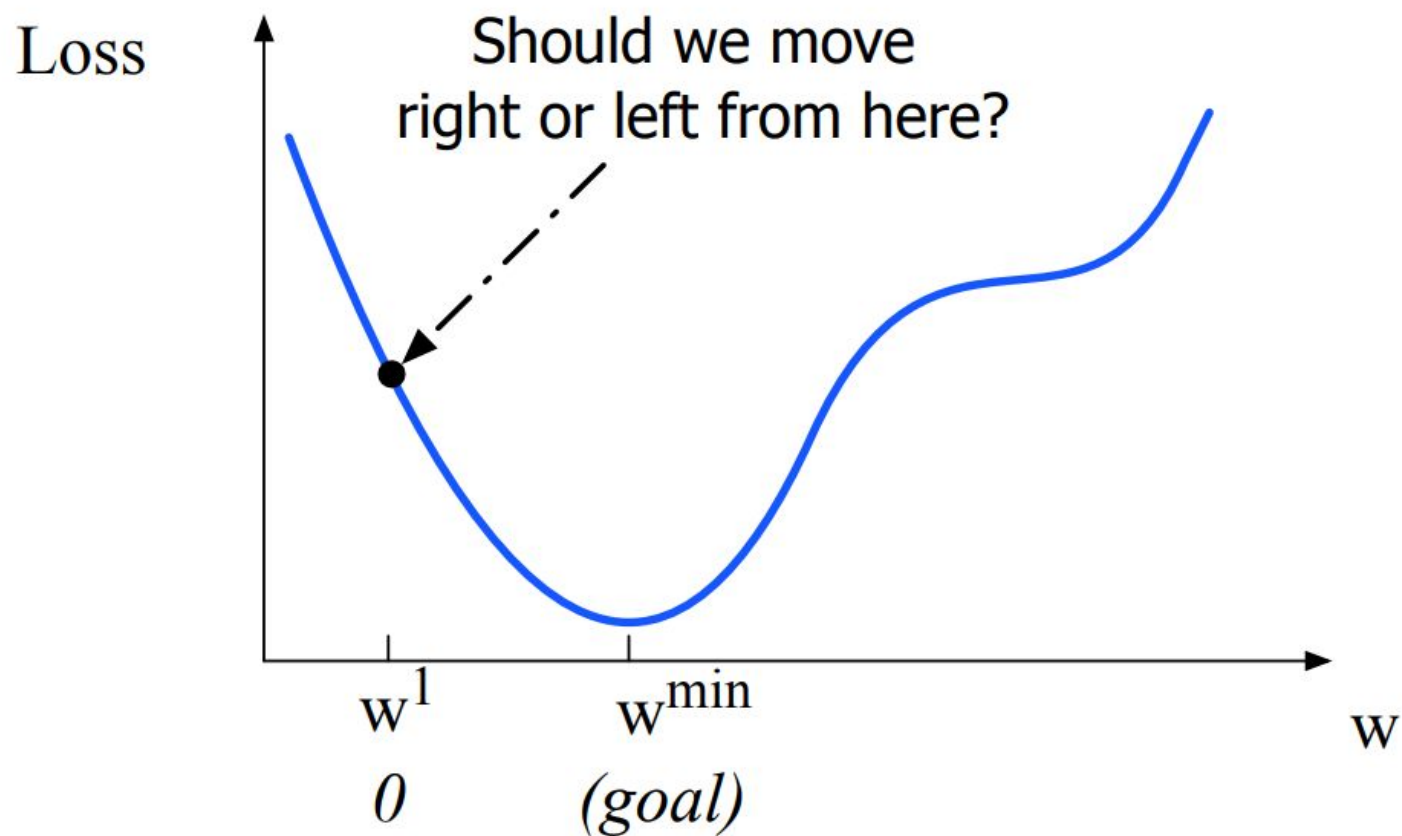


Mirar 360° alrededor de nuestra posición

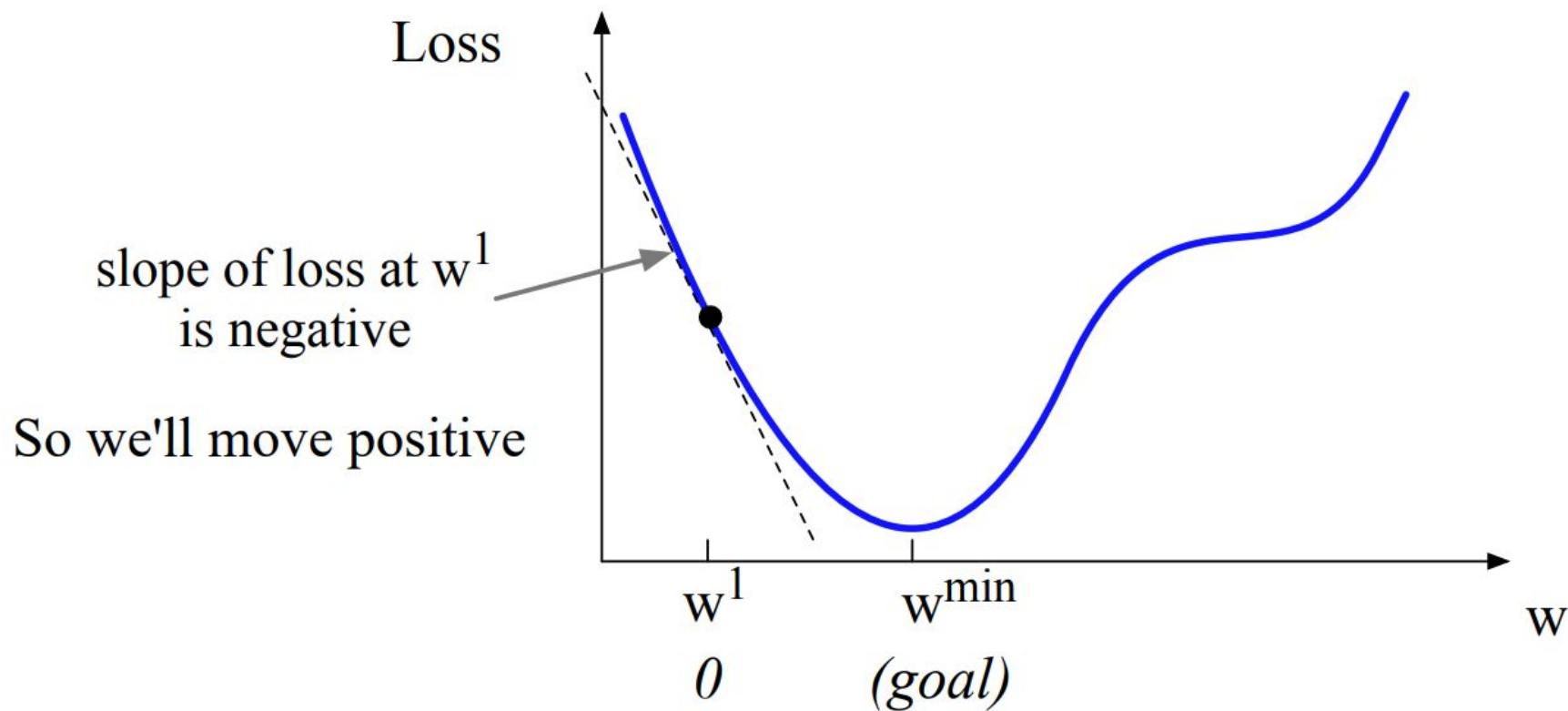
Encontrar la dirección en que la pendiente es más pronunciada hacia abajo

Caminar en esa dirección

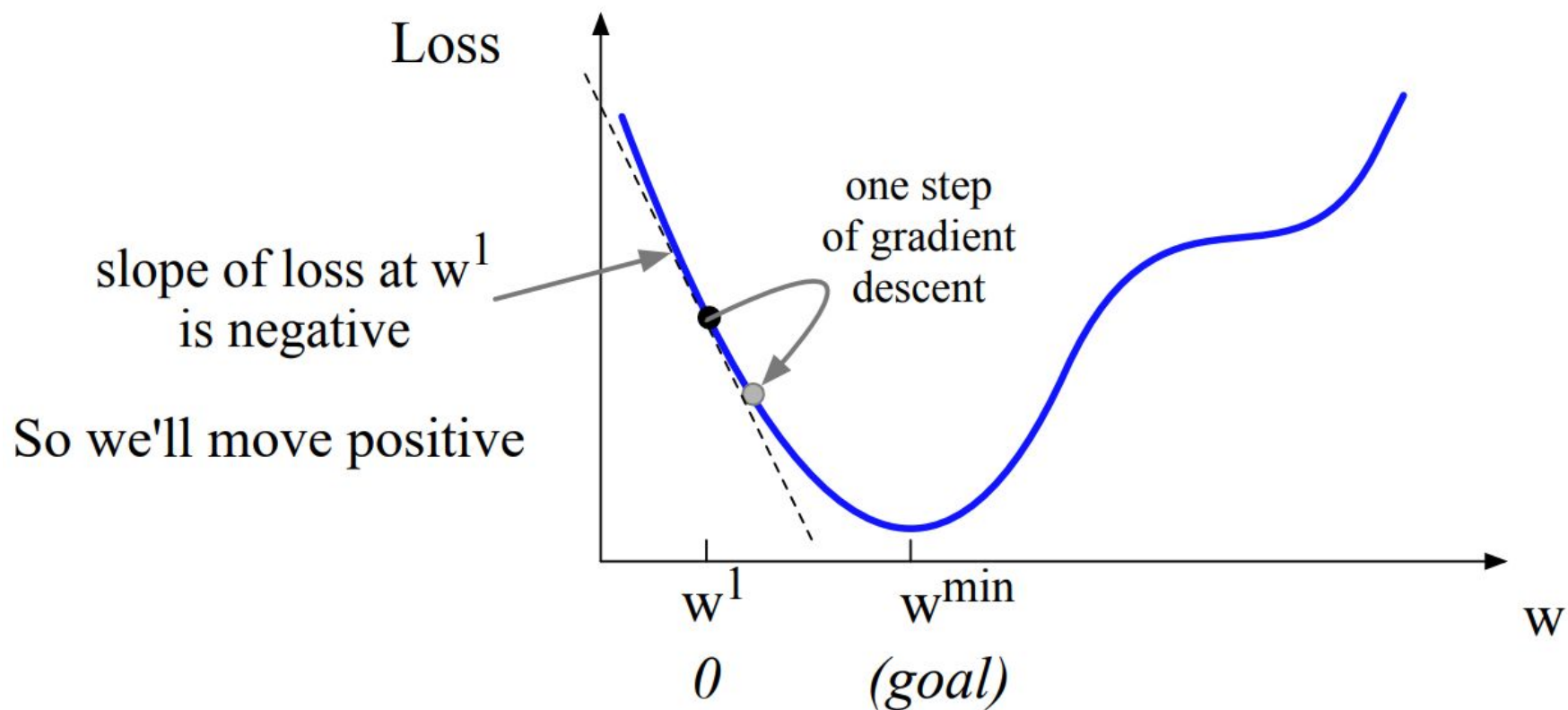
# Visualización en una sola dimensión



# Visualización en una sola dimensión



# Visualización en una sola dimensión



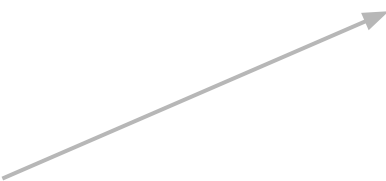
## En varias variables: Gradiente

El gradiente de una función de varias variables es un vector que apunta en la dirección de mayor crecimiento

Descenso por gradiente: Significa encontrar el gradiente de la función de *loss* en el punto actual y moverse en la dirección opuesta

Cuánto nos movemos? Un “paso” que llamaremos  $\eta$

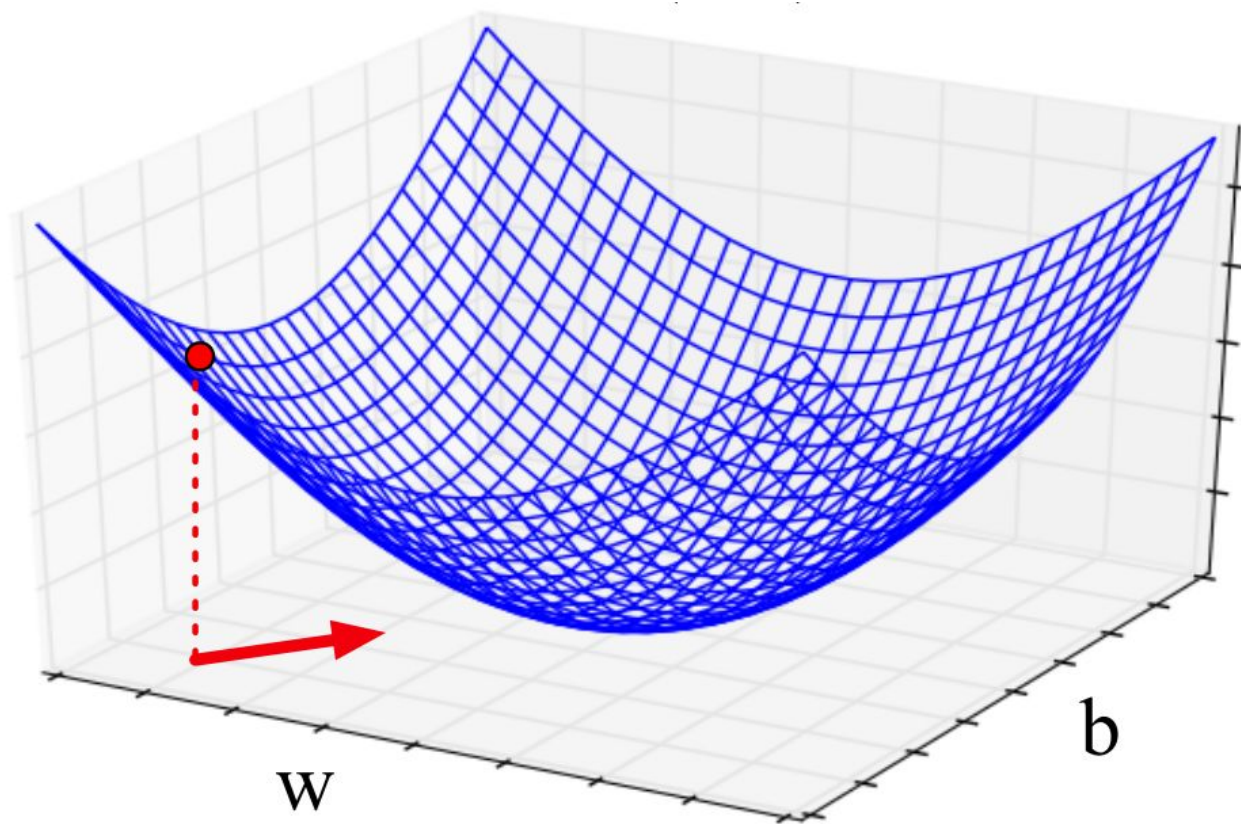
$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$



Learning rate  
(tasa de aprendizaje)



## En varias variables: Gradiente



# Gradiente

En una red vamos a tener muchísimos pesos, incluso en una tan simple como la del ejemplo

$$\nabla_{\theta} L(f(x; \theta), y) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x; \theta), y) \\ \frac{\partial}{\partial w_2} L(f(x; \theta), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x; \theta), y) \end{bmatrix}$$

Fórmula para actualizar  $\theta$  según el gradiente

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x; \theta), y)$$

# Descenso por gradiente estocástico

**function** STOCHASTIC GRADIENT DESCENT( $L()$ ,  $f()$ ,  $x$ ,  $y$ ) **returns**  $\theta$

# where:  $L$  is the loss function

#  $f$  is a function parameterized by  $\theta$

#  $x$  is the set of training inputs  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$

#  $y$  is the set of training outputs (labels)  $y^{(1)}, y^{(2)}, \dots, y^{(m)}$

$\theta \leftarrow 0$

**repeat** til done

For each training tuple  $(x^{(i)}, y^{(i)})$  (in random order)

1. Optional (for reporting): # How are we doing on this tuple?

    Compute  $\hat{y}^{(i)} = f(x^{(i)}; \theta)$  # What is our estimated output  $\hat{y}$ ?

    Compute the loss  $L(\hat{y}^{(i)}, y^{(i)})$  # How far off is  $\hat{y}^{(i)}$  from the true output  $y^{(i)}$ ?

2.  $g \leftarrow \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$  # How should we move  $\theta$  to maximize loss?

3.  $\theta \leftarrow \theta - \eta g$  # Go the other way instead

**return**  $\theta$

# Descenso por gradiente batch versus estocástico

---

Tres formas de entrenar:

- SGD: Presentando un ejemplo a la vez
- Batch training: Poniendo todos los ejemplos en una matriz y calculando el gradiente para todos a la vez
- Un punto medio que funciona mejor: usar mini-batches

Se elige una cantidad de ejemplos (por ejemplo 512 o 1024) y se hace la actualización del gradiente con esos

Luego se elige otro mini-batch hasta agotar el conjunto

# Resumiendo

---

1. Tenemos una función que tomar valores reales como entradas
2. Esta función multiplica cada entrada por un peso, suma los resultados, le agrega un sesgo, y le aplica una función de activación (e.g. una sigmoide), para obtener el resultado
3. Para aprender los pesos, calculamos cuánto se equivoca la función comparando con ejemplos que conocemos, y ajustamos los pesos para que ese error (o pérdida) sea mínimo, utilizando descenso por gradiente.

## Unidad sigmoide - Notación

$$x \in \mathbb{R}^n = a^{(1)} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Nuestra función será...

$$h_{\theta}(x) = a^{(2)} \in \mathbb{R} = g(w_1^{(2)} \cdot a_1^{(1)} + w_2^{(2)} \cdot a_2^{(1)} + \dots + w_n^{(2)} \cdot a_n^{(1)} + b^{(2)})$$

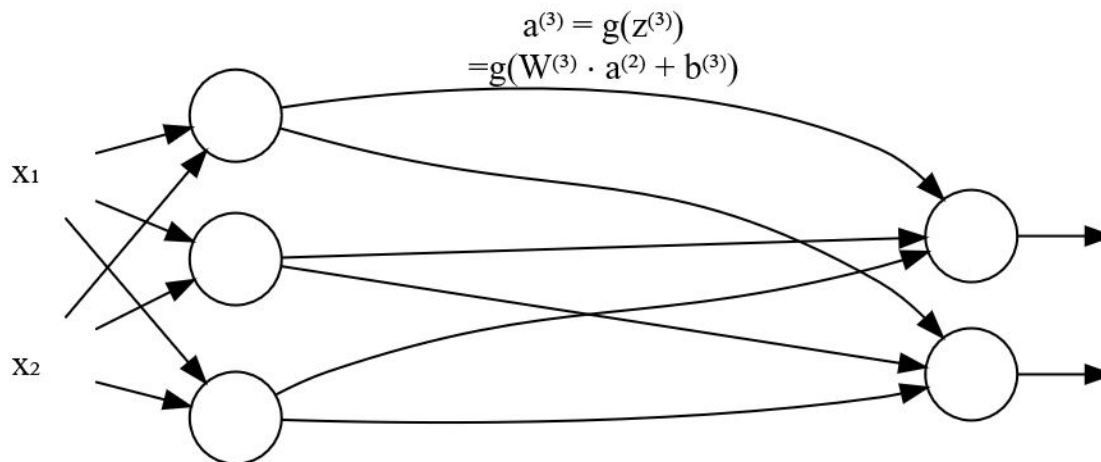
$$h_{\theta}(x) = a^{(2)} \in \mathbb{R} = g(z^{(2)}) = g(w^{(2)} \cdot x + b^{(2)})$$

Siendo  $w^{(2)}$  los parámetros de la capa 2,  $b^{(2)}$  un término de sesgo y  $g(z)$  la función de activación.

# Redes neuronales - Notación

Generalizando el caso anterior, podemos denotar cualquier red neuronal:

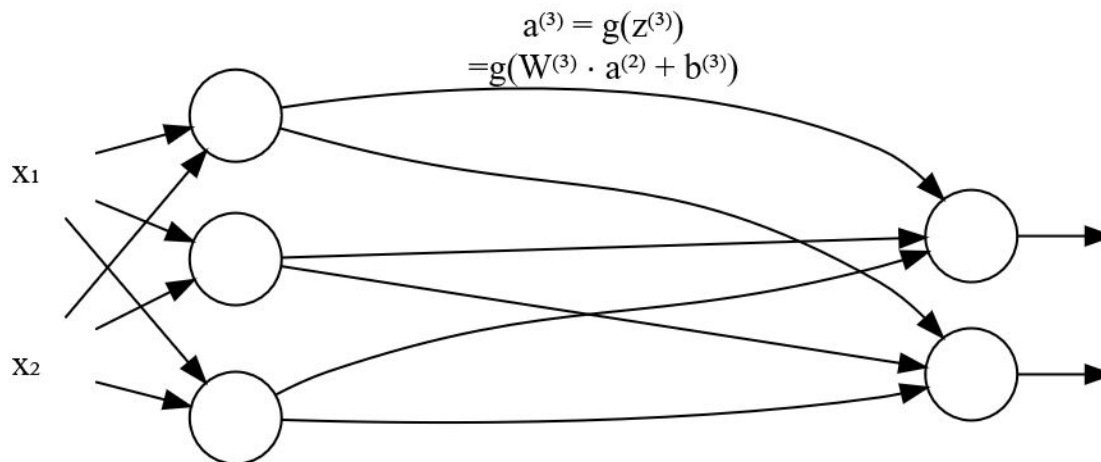
$$a^{(1)} \in \mathbb{R}^n = x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad a^{(j)} \in \mathbb{R}^{s_j} = \begin{bmatrix} a_1^{(j)} \\ \vdots \\ a_{s_j}^{(j)} \end{bmatrix} = g(W^{(j)} \cdot a^{(j-1)} + b^{(j)})$$



# Redes neuronales - Notación

Generalizando el caso anterior, podemos denotar cualquier red neuronal:

$$a^{(1)} \in \mathbb{R}^n = x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad a^{(j)} \in \mathbb{R}^{s_j} = \begin{bmatrix} a_1^{(j)} \\ \vdots \\ a_{s_j}^{(j)} \end{bmatrix} = g(W^{(j)} \cdot a^{(j-1)} + b^{(j)})$$





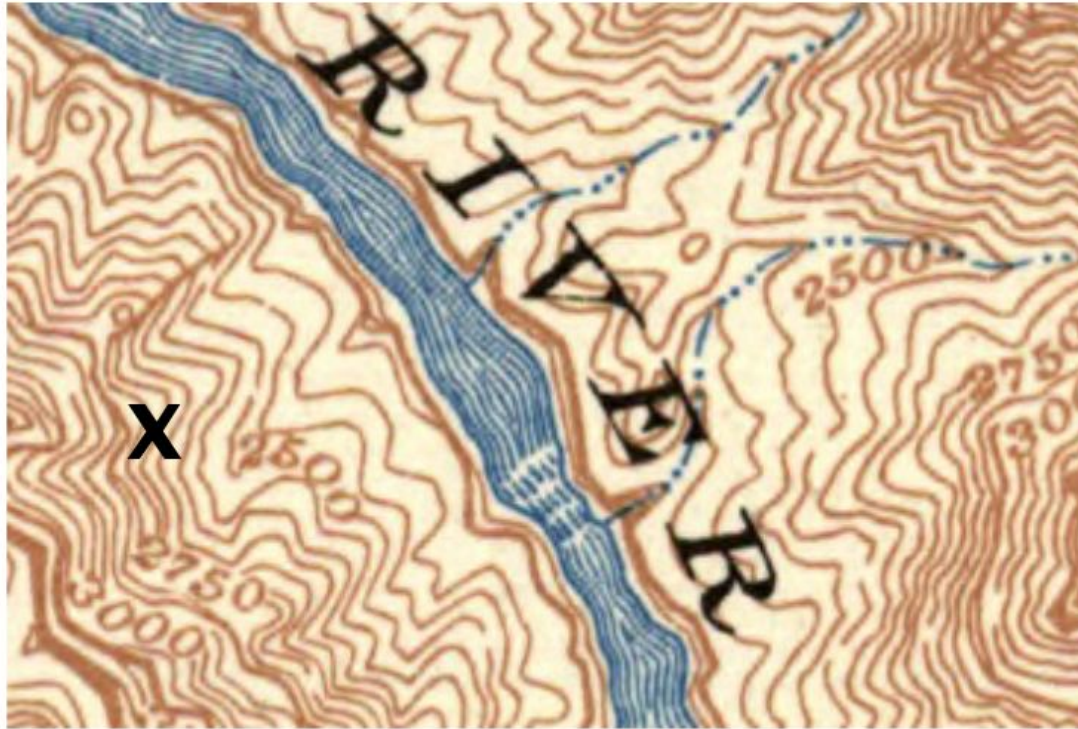
# Redes neuronales - Aprendizaje

---

¿Cómo aprendemos los pesos?

# Redes neuronales - Aprendizaje

¿Cómo aprendemos los pesos? Descenso por gradiente!



# Redes neuronales - Aprendizaje

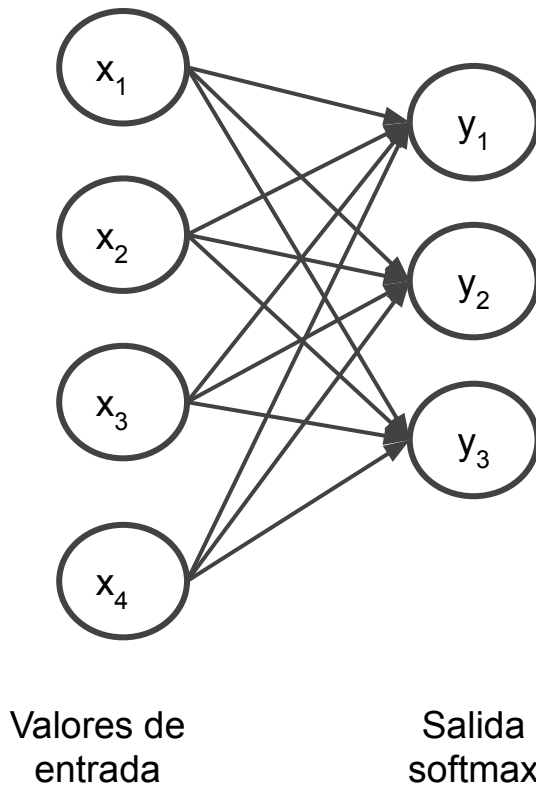
---

Necesitamos una función de pérdida. Las funciones de pérdida tienen algunas características:

1. Solamente dependen de la última capa de la red
2. Pueden calcularse como el promedio de la pérdida de cada ejemplo
3. Por supuesto, son menores cuanto más parecida es la salida a cada ejemplo

Cuál usar va a depender de la salida de nuestra red

# Redes Neuronales - Salida multiclase



Problemas de clasificación discretos, queremos que la salida sea una distribución de probabilidad

Función de activación *softmax*

$$P(j|x) = \frac{e^{y_j}}{\sum_k e^{y_k}}$$

Cada  $y_i$  es combina los valores de las demás para normalizar a una probabilidad

# Clasificación Multiclase

¿Cómo queda la función de pérdida?

Sean  $N$  ejemplos  $y_i$  cada uno con una categoría entre  $k$

Pero para cada  $y_i$  solo una de las clases es la correcta

O sea en cada caso:  $y_{ij} = 1$ , los demás son 0

La entropía cruzada en este caso es:

$$L_{CE} = -\frac{1}{N} \sum_{i=1, y_{ij}=1}^N \log(\hat{y}_{ij})$$

## Redes neuronales - Nos falta algo...

---

¿Cómo calculamos el gradiente de la función de pérdida para cada parámetro de la red?

# Redes Neuronales: Backpropagation

- Las capas apiladas son composiciones de funciones (regla de la cadena)
- Calcular el gradiente implicaría "desplegar" toda la función e ir derivando respecto a cada parámetro. Computacionalmente muy costoso.
- Idea: explotar la arquitectura y partir "de atrás hacia adelante", reutilizando resultados intermedios.
- Dos pasadas para computar el gradiente
  - Forward pass: se ejecuta la red para una entrada
  - Backward pass: se propaga el gradiente desde la capa de salida hacia la capa de entrada

# Backpropagation

---

El grafo de computación permite la diferenciación automática. Las operaciones tensoriales se computan más rápido en hardware específico (Ej. GPUs, TPUs)

Seguimos utilizando SGD con mini-batches

Términos a tener en cuenta:

- Step: un episodio de actualización del gradiente
  - Aplicar la red a un mini-batch
  - Realizar backpropagation y actualizar los pesos
- Epoch: pasar una vez por el corpus de entrenamiento entero
  - Generalmente implica varios steps



# Resumiendo

---

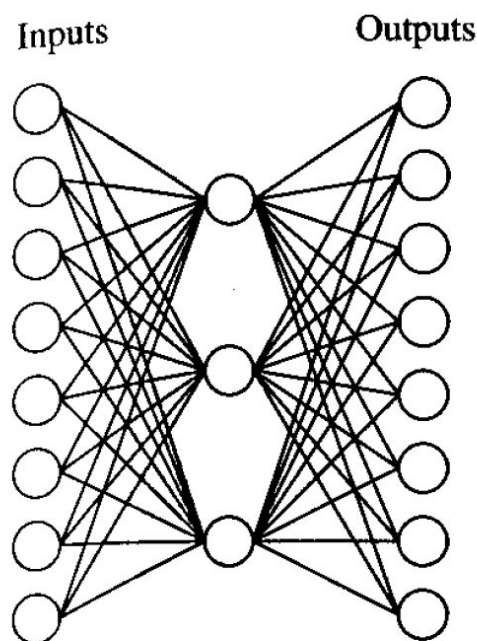
1. Tenemos una función que tomar valores reales como entradas
2. Esta función multiplica cada entrada por un peso, suma los resultados, le agrega un sesgo, y le aplica una función de activación (e.g. una sigmoide), para obtener el resultado...  
**muchas veces, y haciendo que cada capa tome como entrada la salida de la anterior**
3. Para aprender los pesos, calculamos cuánto se equivoca la función comparando con ejemplos que conocemos, y ajustamos los pesos para que ese error (o pérdida) sea mínimo, utilizando descenso por gradiente **(y usando backpropagation para obtener el gradiente)**

# Algunos comentarios...

---

[A Neural Network Playground](#)

# Aprendizaje de Representaciones



Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

**FIGURE 4.7**

**Learned Hidden Layer Representation.** This  $8 \times 3 \times 8$  network was trained to learn the identity function, using the eight training examples shown. After 5000 training epochs, the three hidden unit values encode the eight distinct inputs using the encoding shown on the right. Notice if the encoded values are rounded to zero or one, the result is the standard binary encoding for eight distinct values.

# Aprendizaje de Representaciones

---

*“Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.”*

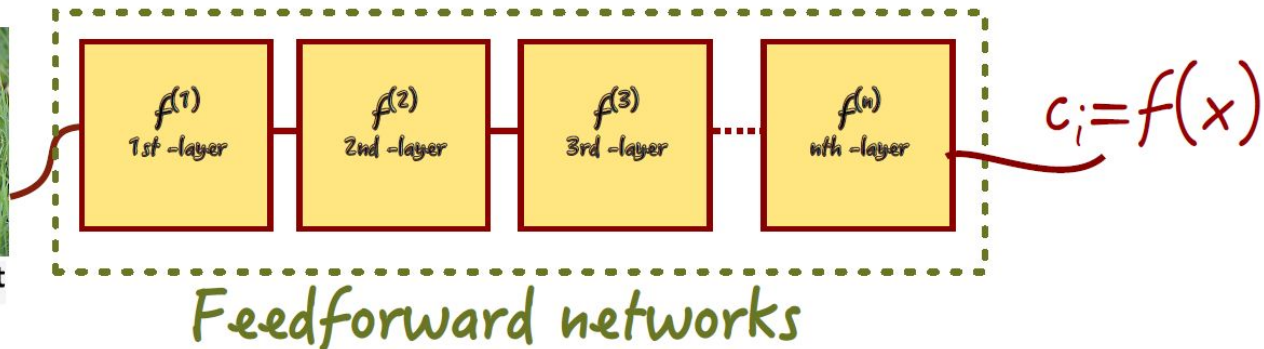
Yann LeCun, Yoshua Bengio and Geoffrey Hinton, Deep Learning, Nature, 2015

# Deep Learning

- Redes con muchas capas no lineales en cascada
- Cada capa usa la salida de la capa anterior como entrada, y obtiene representaciones con mayor nivel de abstracción
- Busca generar una jerarquía de conceptos en las diferentes capas
- Las representaciones de bajo nivel son comunes a las distintas categorías
- Las representaciones de alto nivel son más globales, y más invariantes
- Entrenamiento punta a punta



Grahford, Hidden Cat



# ¿Cómo aplicamos Deep Learning al PLN?

---

## Problemas a resolver

- Cómo representar una palabra
- Cómo representar varias palabras

# Representación de Palabras

## One-hot encoding

- Vector del tamaño del vocabulario (e.g. 10.000 palabras más frecuentes)
- Vale 0 para todas las dimensiones excepto la de la palabra a representar

$D = \{\text{perro, gato, árbol, saltar, correr, ... australopithecus ... sicómoro ...}\}$

$$v_{\text{saltar}} = [0, 0, 0, 1, 0, 0, \dots]$$

$$v_{\text{gato}} = [0, 1, 0, 0, 0, 0, \dots]$$

*Mucho mejor: Word Embeddings! (lo veremos más adelante)*

# Representación de Textos

---

- Representar cada una de las palabras concatenadas
  - Por ejemplo los vectores one-hot (o los embeddings)
- Problema: largo fijo
  - Largo máximo de un texto del corpus (?)
- Pesos de las palabras del principio se actualizan más que los del final
- Pero esta idea puede servir como base para otras (más adelante)



# Representación de Textos

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

Bag of Words

# Bag of Words

Vector de largo fijo (el vocabulario)

Cada componente representa la cantidad de veces que aparece la palabra en el documento (por ejemplo en un review)

Se puede calcular como la suma de los vectores *one-hot* del documento

- Es un método de agregación

Otro método de agregación muy usado: tf-idf

- Ponderar cada componente  $k$  por su idf:  $\log(N/d_k)$

*Mucho mejor: Agregación de Word Embeddings! (lo veremos más adelante)*

# Representación de Textos

---

## Representaciones de agregación:

- Bag-of-words, tf-idf, bag-of-ngrams, promedio de embeddings...

## Problema principal:

- Se pierde el orden de la entrada! (Vamos a tener que mejorar la arquitectura)

*“la vi pero no me gusta”*

*“no la vi pero me gusta”*