# Introucción a Redes Neuronales

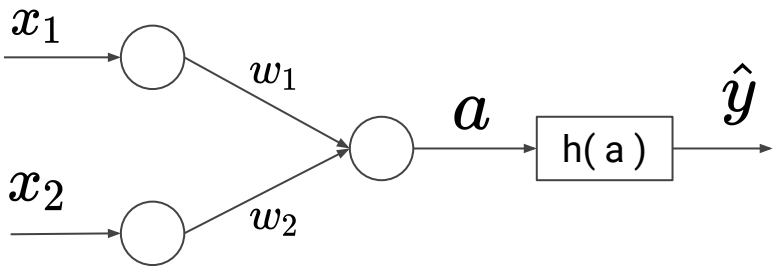
I302 - Aprendizaje Automático y Aprendizaje Profundo

Roberto Bunge

Universidad de San Andrés

# Pereceptrón Simple

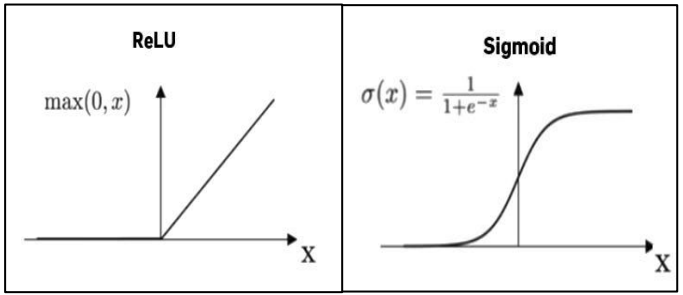
- Supongamos dos entradas (features) y una salida (target):




$$a = w_0 + w_1 x_1 + w_2 x_2 = [w_0 \ w_1 \ w_2][1 \ x_1 \ x_2]^T$$

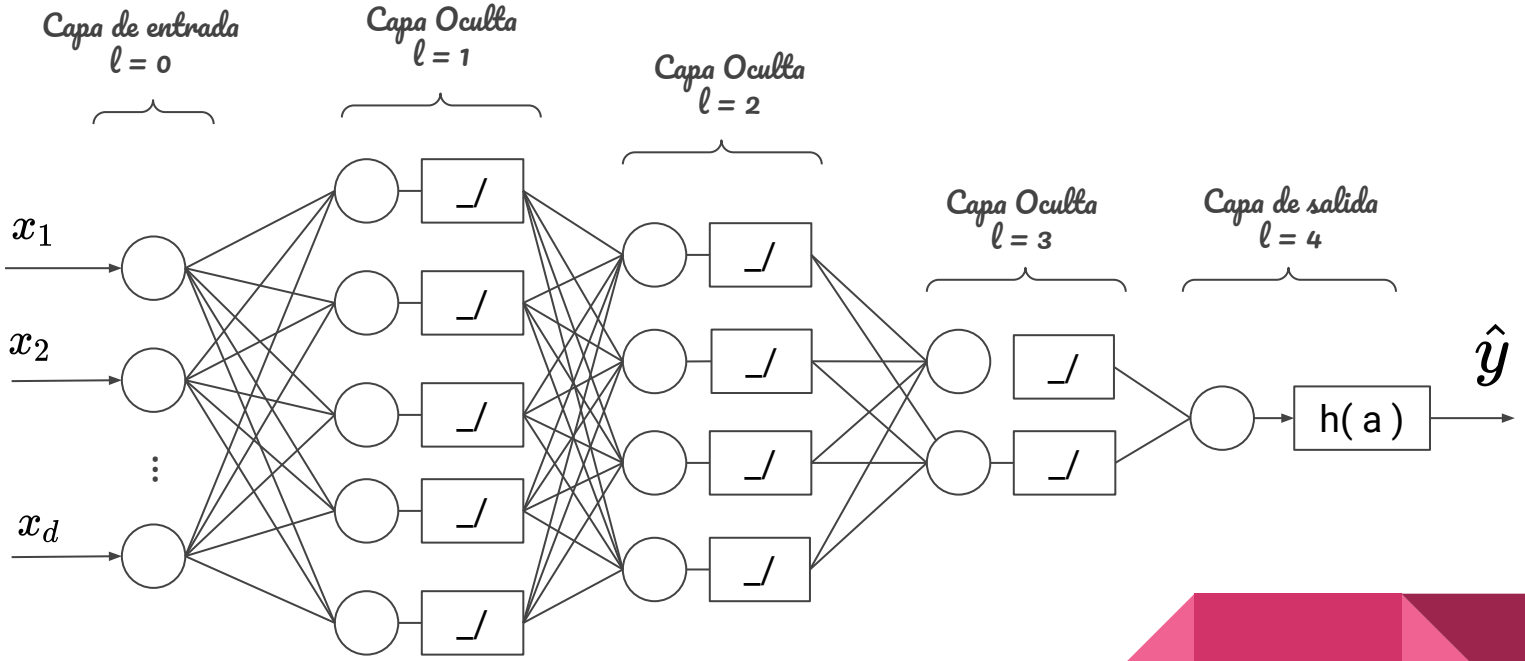
$$\hat{y} = h(a)$$

Ejemplos funciones de activación:



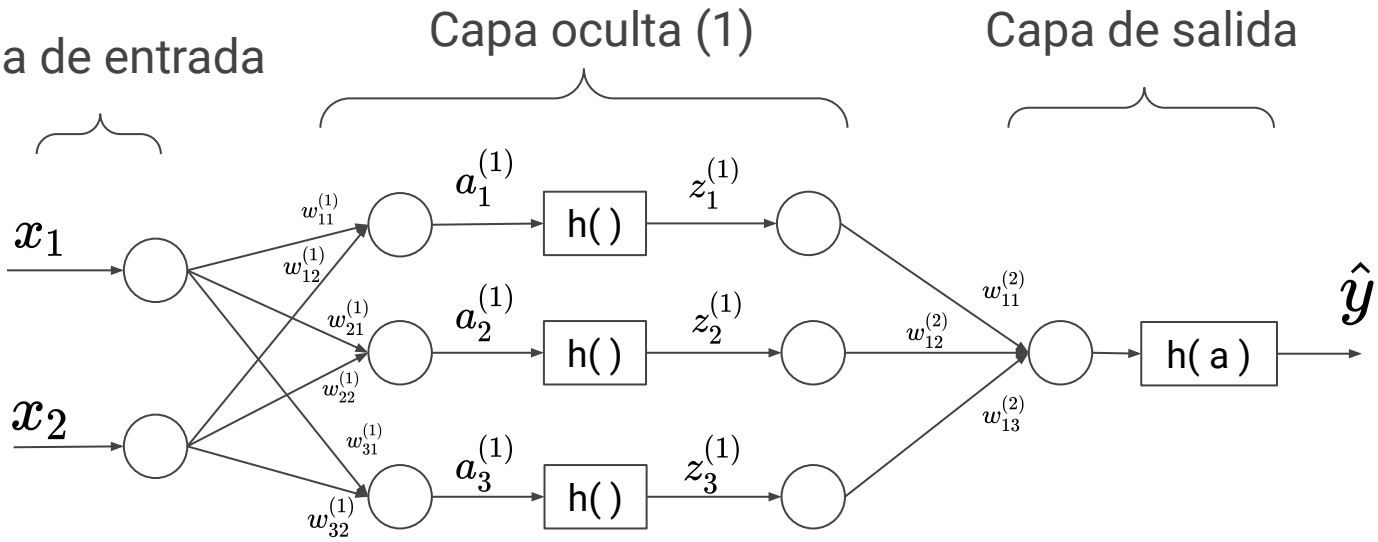

- Regresión: salida no tiene activación o ReLu
  - Regresión lineal = perceptrón simple sin activación de salida
- Clasificación: salida con activación sigmoide (o softmax)

# Perceptrón Multi-capa (Multi layer perceptron, MLP)




$M^{(l)}$ : cantidad de unidades ocultas en la capa $l$

# Notación MLP



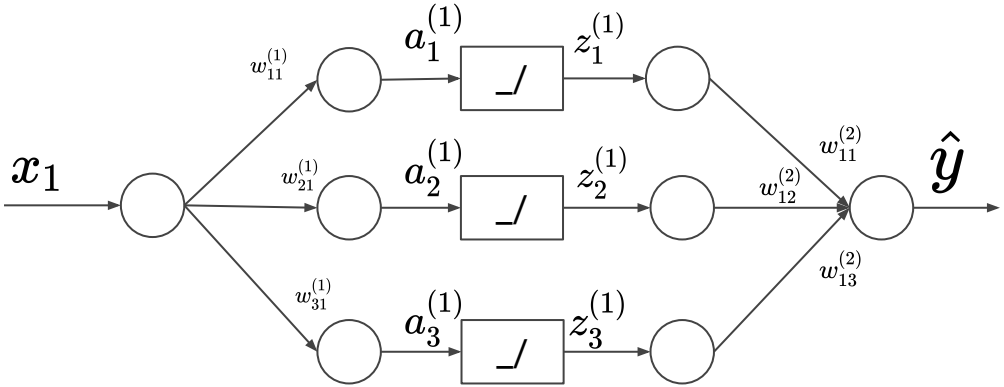

$w_{ij}^{(l)}$ : peso que se aplica a la salida del nodo j de la capa (l-1) y se suma al nodo i de la capa (l)

$a_j^{(l)}$ : señal de **pre-activación** del nodo j de la capa (l)

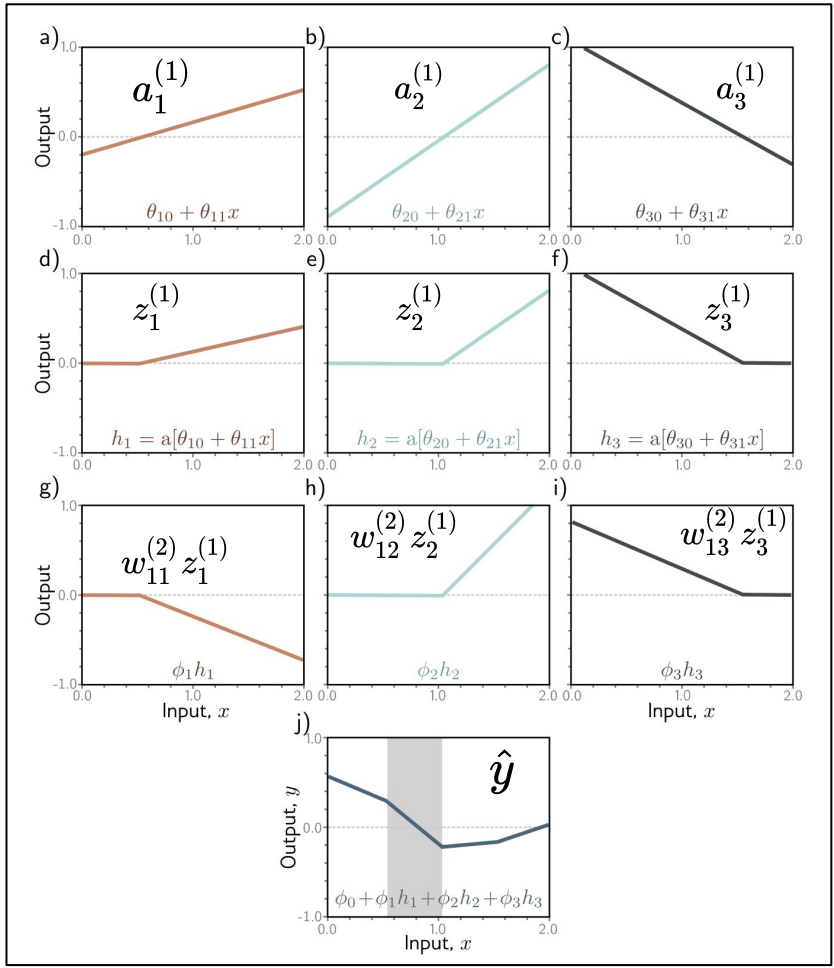
$z_j^{(l)}$ : señal de **activación** (salida) del nodo j de la capa (l)

# Red con tres unidades ocultas

- MLP con una entrada y una capa oculta con 3 unidades ocultas



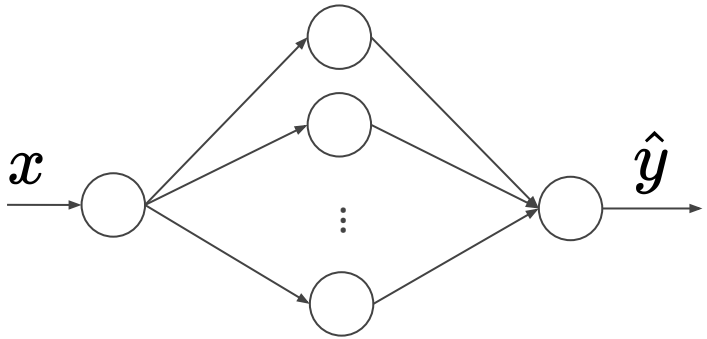

- La salida es lineal por tramos:
  - Por cada unidad oculta se agrega un "codo"




* Grafico tomado de "Understanding Deep Learning", Prince (2023).

# Teoremas de Aproximación Universal

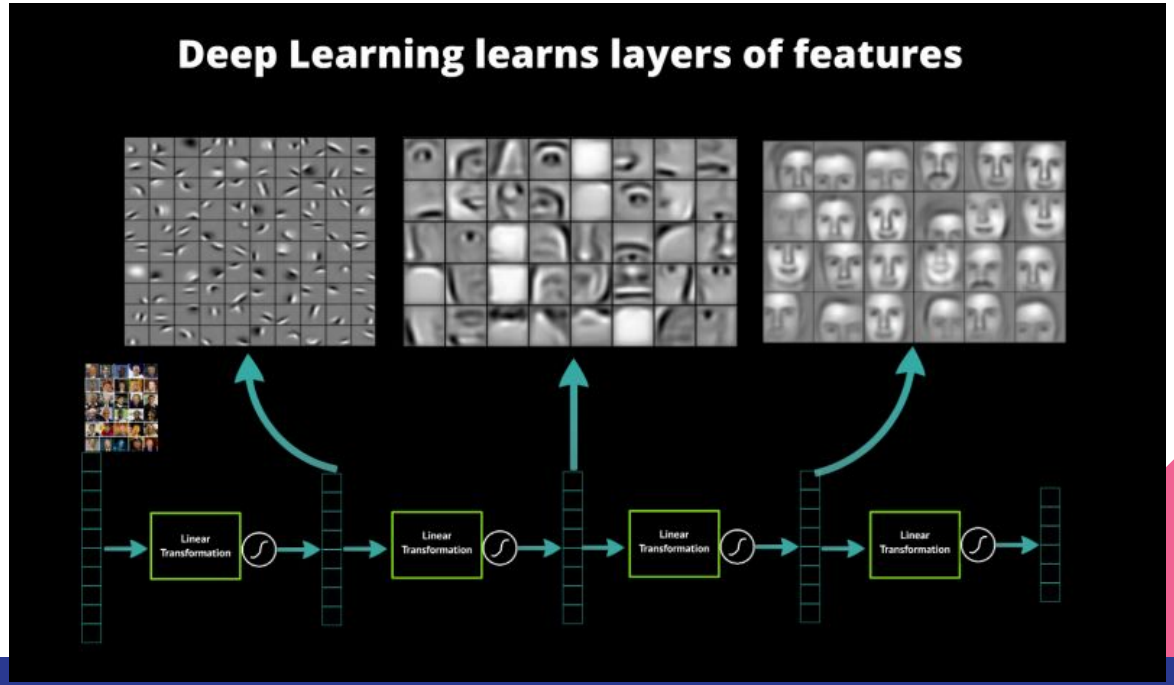
- Podemos aproximar cualquier función con infinita precisión con un MLP con una sola capa oculta, si tenemos suficientes unidades ocultas en dicha capa




- A medida que agregamos mas unidades ocultas, la expresividad del modelo aumenta (pero la cantidad de parámetros también aumenta!)

# Features Internos

- A medida que agregamos mas capas ocultas el MLP puede aprender features de orden superior



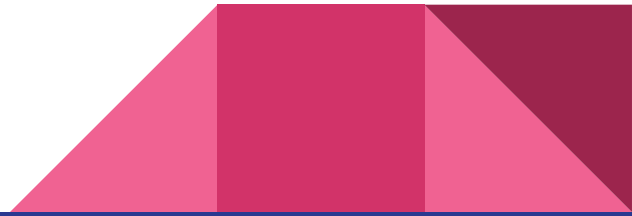
# Aprender (optimizar) los pesos en MLP

- Cascada de productos internos y no-linealidades
  - Problema de optimzación no-convexo ⇒ Habran multiples minimos locales
  - No puedo encontrar mínimos analíticamente ⇒ Tengo que optimizar numéicamente
- Si hay muchas neuronas ⇒ Metodo de Newton es demasiado costoso

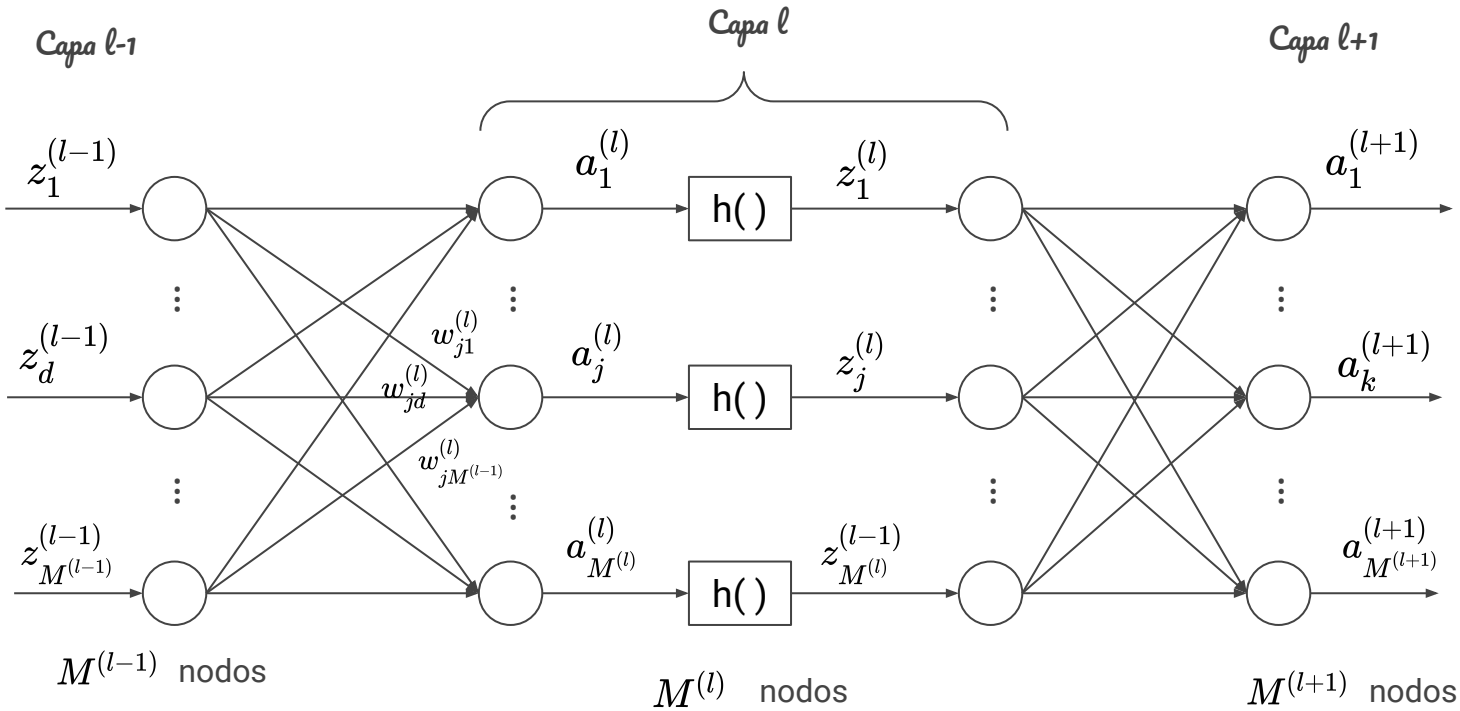
  ⇒ Nos limitamos a Gradiente Descendiente
- En cualquier caso, vamos a necesitar computar el gradiente

# Computo de Gradiente

1. Escribir L(w) explicitamente y computar gradiente analiticamente
    a. Es artesanal, proclive a error
    b. Toma mucho trabajo humano
2. Diferenciación simbólica
    a. Puede dar expresiones MUY grandes
    b. No puede manejar loops (recurrencias, etc.)
3. Diferenciación numérica (diferencia finitas)
    a. Error numerico
    b. Costo computacional, O(D^2)
4. Aprovechar recursividad
    a. Backpropagation
    b. Automatic Differentiation

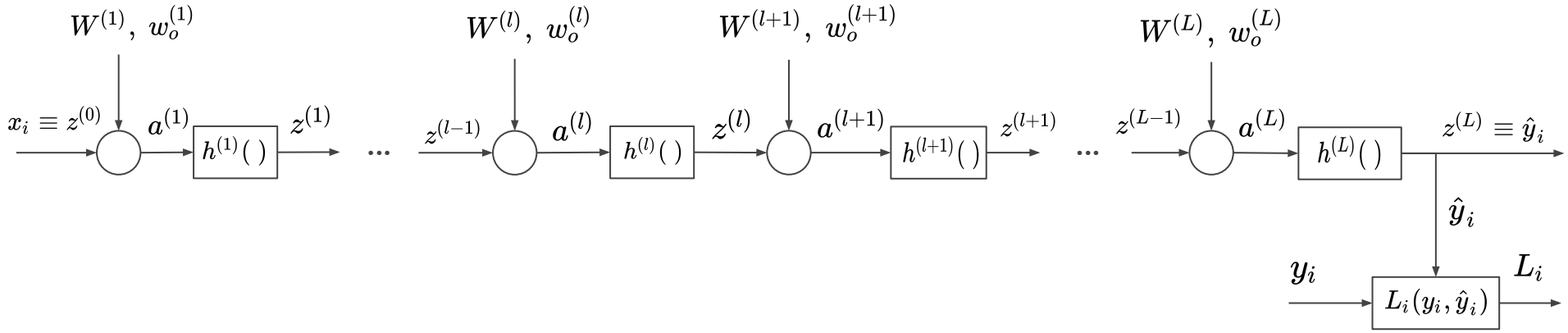


# Notación de señales de MLP




NOTA: cada unidad oculta también tiene un bias $w_{j0}^{(l)}$

# Representación Vectorial de MLP




Relación entre señales de una capa y la siguiente:

$$
\begin{aligned}
a^{(l)} &= W^{(l)} z^{(l-1)} + w_0^{(l)} \\
z^{(l)} &= h^{(l)}(a^{(l)}) \\
a^{(l+1)} &= W^{(l+1)} z^{(l)} + w_0^{(l+1)}
\end{aligned}
$$

$$
W^{(l)} = \begin{bmatrix} w_{11}^{(l)} & \cdots & w_{1M^{(l-1)}}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{M^{(l)}1}^{(l)} & \cdots & w_{M^{(l)}M^{(l-1)}}^{(l)} \end{bmatrix} \in \mathbb{R}^{M^{(l)} \times M^{(l-1)}}
$$

$$
w_0^{(l)} = \begin{bmatrix} w_{10}^{(l)} \\ \vdots \\ w_{M^{(l)}0}^{(l)} \end{bmatrix}, \quad a^{(l)} = \begin{bmatrix} a_1^{(l)} \\ \vdots \\ a_{M^{(l)}}^{(l)} \end{bmatrix}, \quad z^{(l)} = \begin{bmatrix} z_1^{(l)} \\ \vdots \\ z_{M^{(l)}}^{(l)} \end{bmatrix} \in \mathbb{R}^{M^{(l)} \times 1}
$$

# Backpropagation (I)

$$\mathcal{D} = \{(x_1, y_1), ..., (x_i, y_i), ..., (x_N, y_N)\}$$

$$L(w) = \sum_{i \in \mathcal{D}} L_i(w)$$

$$w^* = \min_w L(w)$$

$$\nabla_w L(w) = \sum_{i \in D} \nabla_w L_i(w)$$

# Backpropagation (II)

$$\frac{\partial f(x)}{\partial x} \equiv \nabla_x f(x)$$

Podemos computar el gradiente de $L_i(w)$ con respecto a los pesos $W^{(l)}$ de una capa $l$, aplicando la regla de la cadena:

$$\frac{\partial L_i}{\partial W^{(l)}} = \frac{\partial L_i}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial W^{(l)}} = \frac{\partial L_i}{\partial a^{(l)}} z^{(l-1)^T}$$

$$\frac{\partial L_i}{\partial w_0^{(l)}} = \frac{\partial L_i}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial w_0^{(l)}} = \frac{\partial L_i}{\partial a^{(l)}} I = \frac{\partial L_i}{\partial a^{(l)}}$$

$$\frac{\partial L_i}{\partial a^{(l)}} = \frac{\partial z^{(l)}}{\partial a^{(l)}} \frac{\partial L_i}{\partial z^{(l)}} = \text{diag}(h'^{(l)}(a^{(l)})) \frac{\partial L_i}{\partial z^{(l)}}$$

$$\frac{\partial L_i}{\partial z^{(l)}} = \frac{\partial a^{(l+1)}}{\partial z^{(l)}} \frac{\partial L_i}{\partial a^{(l+1)}} = W^{(l+1)^T} \frac{\partial L_i}{\partial a^{(l+1)}}$$

Definiendo $\delta^{(l)} \triangleq \frac{\partial L_i}{\partial a^{(l)}}$, y juntando las ecuaciones anteriores tenemos:

$$\delta^{(l)} = \text{diag}(h'^{(l)}(a^l)) W^{(l+1)^T} \delta^{l+1}$$

$$\frac{\partial L_i}{\partial W^{(l)}} = \delta^{(l)} z^{(l-1)^T}$$

$$\frac{\partial L_i}{\partial w_0^{(l)}} = \delta^{(l)}$$

# Backpropagation (III)

Para la capa de salida podemos computar $\delta^{(L)}$ volviendo a la definicion de $\delta^{(l)}$ y aplicando la regla de la cadena sobre $L_i$ con respecto a $\hat{y}_i$ y $a^{(L)}$ (donde $\hat{y}_i = \hat{y}(x_i, w)$):

$$\delta^{(L)} \triangleq \frac{\partial L_i}{\partial a^{(L)}} = \frac{\partial \hat{y}_i}{\partial a^{(L)}} \frac{\partial L_i}{\partial \hat{y}_i} = \text{diag}(h'^{(L)}(a^L)) \frac{\partial L_i}{\partial \hat{y}_i} \tag{12}$$

Teniendo $\delta^{(L)}$, podemos computar $\frac{\partial L_i}{\partial W^{(L)}}$ y $\frac{\partial L_i}{\partial w_0^{(L)}}$ aplicando la formula de back-propagation:

$$\frac{\partial L_i}{\partial W^{(L)}} = \delta^{(L)} z^{(L-1)^T} \tag{13}$$

$$\frac{\partial L_i}{\partial w_0^{(L)}} = \delta^{(L)} \tag{14}$$

# Computo mas eficiente

En la práctica, se puede hacer una mejora notando que la mutlplicacion matricial entre una matriz diagonal $diag(a)$ y vector columna $b$ se puede computar mas eficientemente mediante el producto elemento a elemento de la diagonal y el vector columna:

$$diag(a)b = a \odot b$$

# Algoritmo Backpropagation

**Algorithm 1** Algoritmo backpropagation

INPUTS: $x_i, W^{(:)}, w_o^{(:)}$

**# Forward-pass**

$z^{(0)} = x_i$

**for** $l = 1$ **to** $L$ **do**

$a^{(l)} = W^{(l)} z^{(l-1)} + w_0^{(l)}$

$z^{(l)} = h^{(l)}(a^{(l)})$

**end for**

$\hat{y}_i = z^{(L)}$

$L_i = L_i(y_i, \hat{y}_i)$

**# Backward-pass**

$\delta^{(L)} = h'^{(L)}(a^l) \odot \frac{\partial L_i}{\partial \hat{y}_i}$

$\frac{\partial L_i}{\partial W^{(L)}} = \delta^{(L)} {z^{(L-1)}}^T$

**for** $l = L - 1$ **to** $1$ **do**

$\delta^{(l)} = h'^{(l)}(a^{(l)}) \odot {W^{(l+1)}}^T \delta^{(l+1)}$

$\frac{\partial L_i}{\partial W^{(l)}} = \delta^{(l)} {z^{(l-1)}}^T$

$\frac{\partial L_i}{\partial w_0^{(l)}} = \delta^{(l)}$

**end for**

OUTPUTS: $\hat{y}_i, L_i, \frac{\partial L_i}{\partial W^{(:)}}, \frac{\partial L_i}{\partial w_0^{(:)}}$