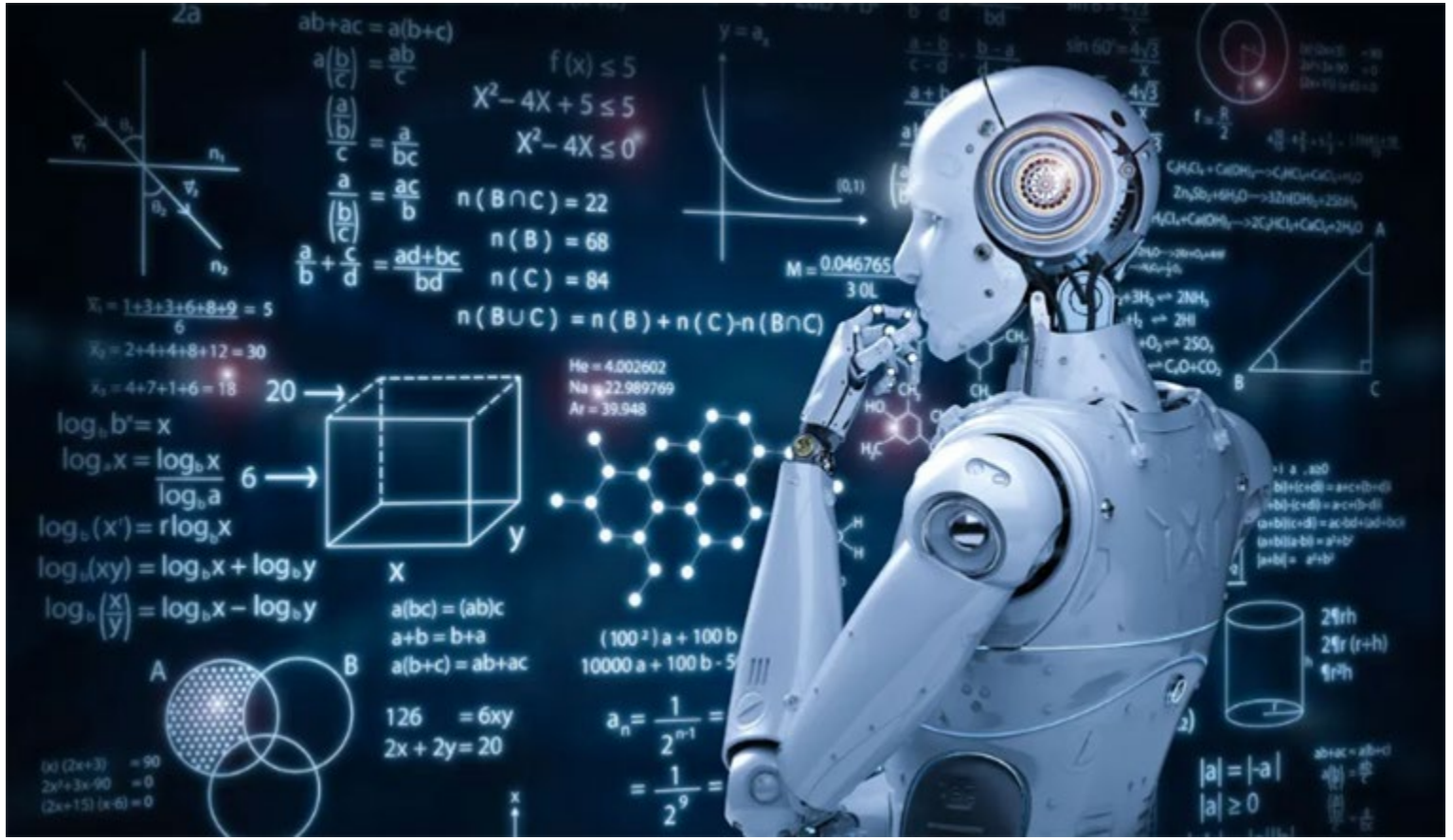


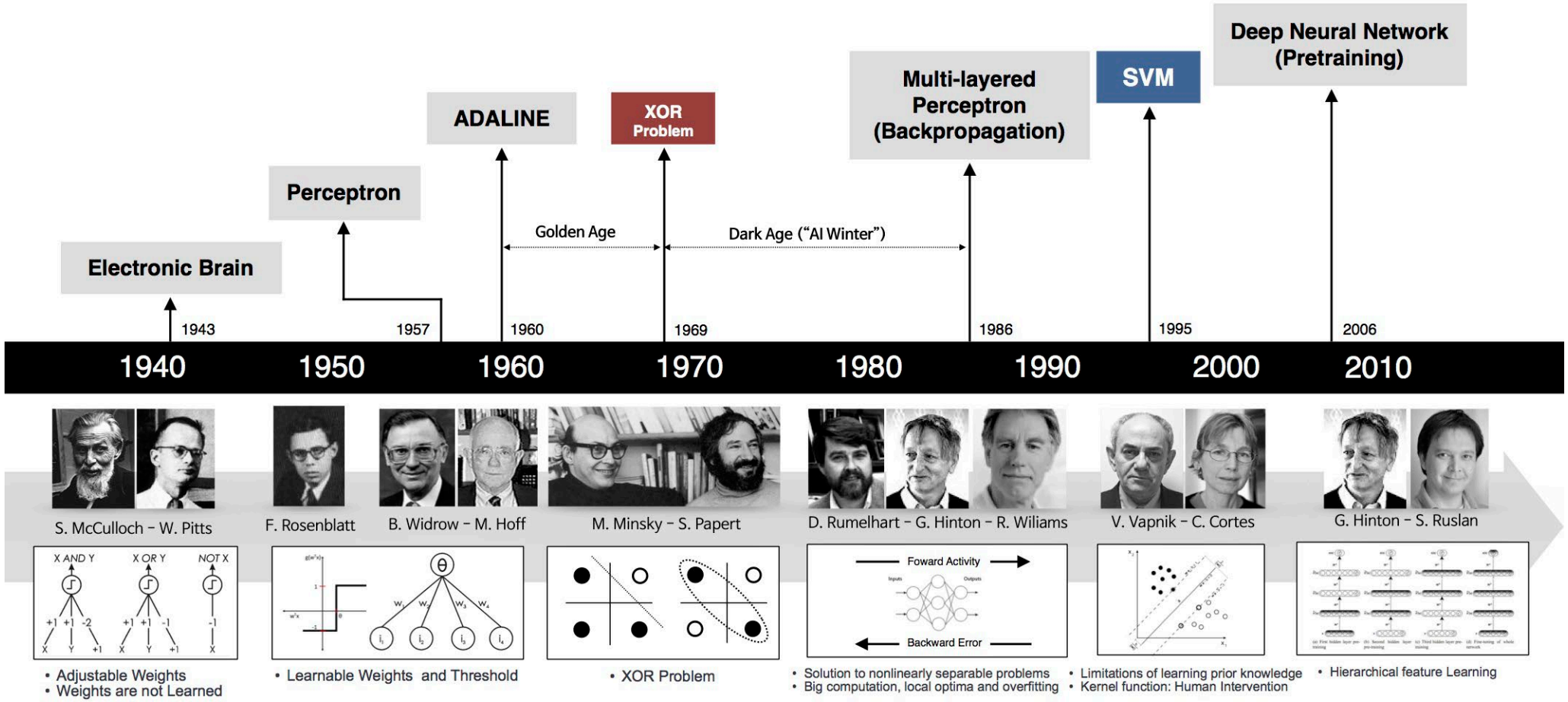
Redes Neuronales Artificiales

Jonnatan Arias Garcia
jonnatan.arias@utp.edu.co
jariasg@uniquindio.edu.co

David Cardenas peña - dcardenasp@utp.edu.co
Hernán Felipe Garcia - hernanf.garcia@udea.edu.co

Contenido

- ❑ Introducción e Historia
- ❑ Redes de propagación hacia adelante (FFNN-Feed Forward Neural Network)
  - ❑ Perceptrón simple (repaso)
  - ❑Modelo matemático
- ❑ Entrenamiento del MLP
  - ❑Propagación hacia atrás
- ❑ Consideraciones y regularización
- ❑ Invarianzas
- ❑ Mejoras
- ❑ Zoo NN

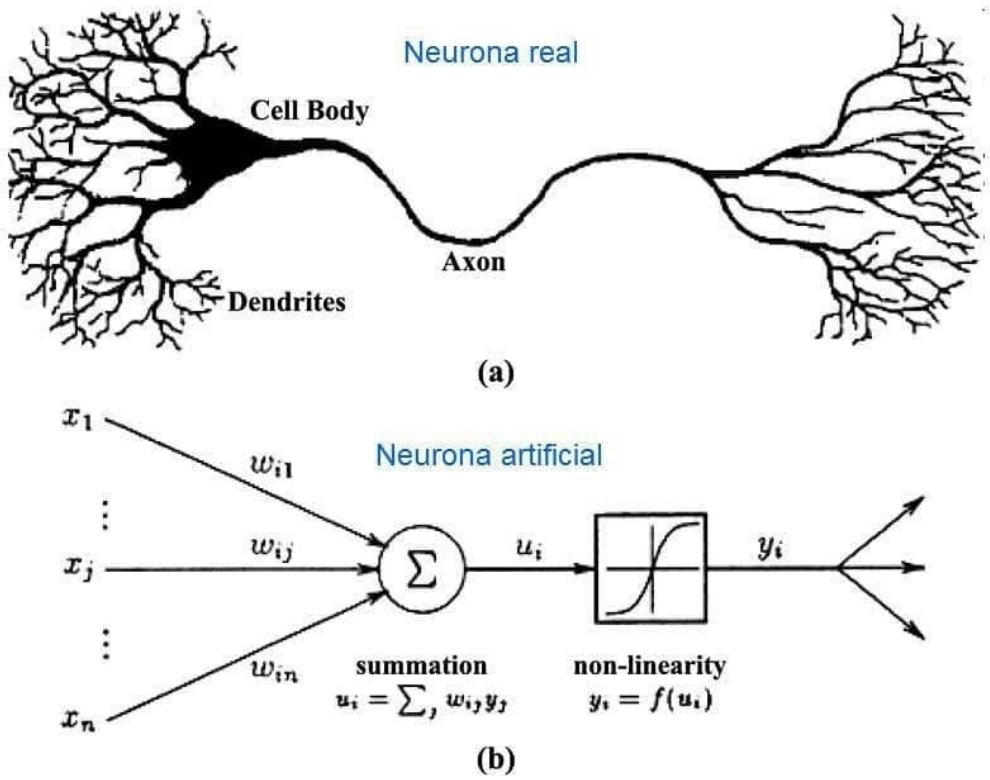


Historia de las Redes Neuronales

Modelo de una neurona

1940
S.McCulloch – W. Pitts

Weights Ajustables
no se aprenden los pesos



- Aproximaron la salida que se quiere modelar como una suma ponderada de las entradas que pasan a través de una función de umbralización

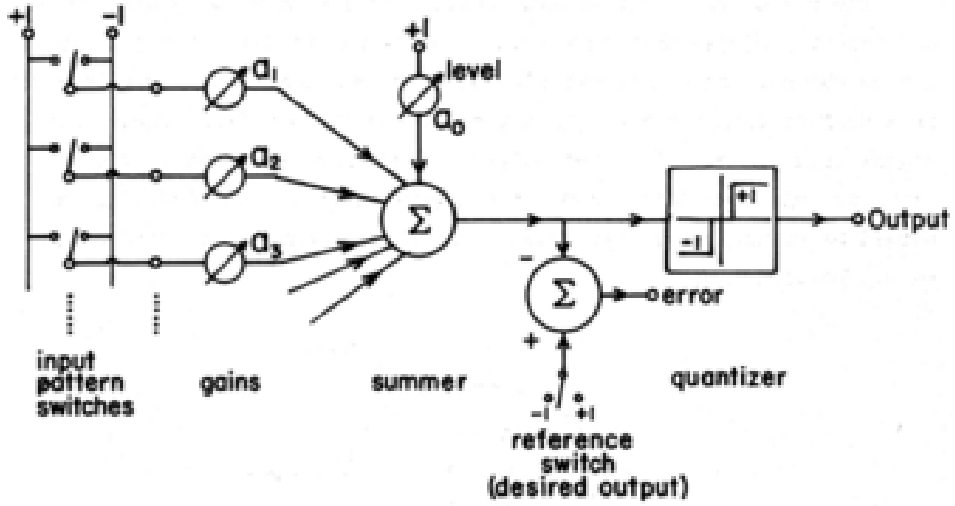
$$y = \mathbb{I}\left(\sum_{\forall i} w_i x_i > \theta\right),$$

para algún umbral θ.

Perceptrón y Adaline

1957 – Frank Rosenblatt invento la estimación de parámetros de la neurona de McCulloh y Pitts

1960 – Widrow & Hoff inventaron el modelo conocido como **adaline** (Adaptive Linear Neuron or later Adaptive Linear Element)



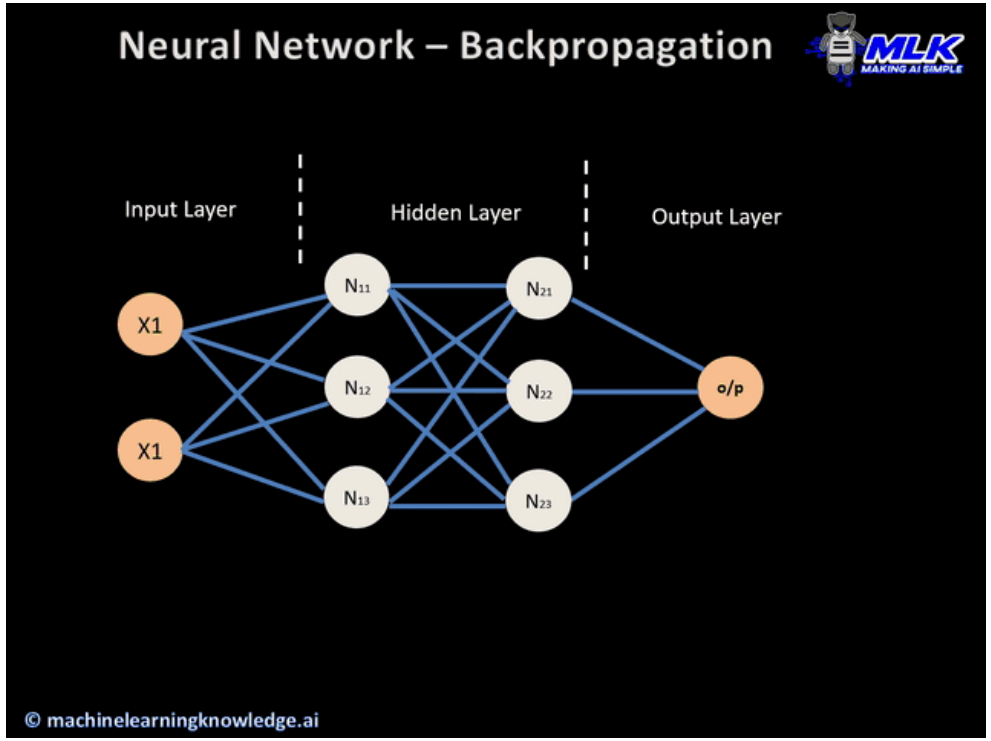
En 1969, Minsky y Papert publicaron un libro conocido como “Perceptrones”, en el que mostraron que un modelo sin capas ocultas era muy limitado, porque no permitía clasificar datos que no fueran linealmente separables.

Lo anterior redujo considerablemente el interés en el tema.

BackPropagation
1986 – Rumelhart, Hintos y Williams

- En 1986, Rumelhart, Hinton y Williams descubrieron el algoritmo de propagación hacia atrás, que permitía ajustar modelos con capas ocultas.
  - El algoritmo backpropagation fue descubierto originalmente por Bryston and Ho, en 1969.
  - Fue descubierto de forma independiente por Werbos en 1974.
  - Rumelhart y sus colaboradores llamaron la atención de la comunidad sobre la existencia del algoritmo.

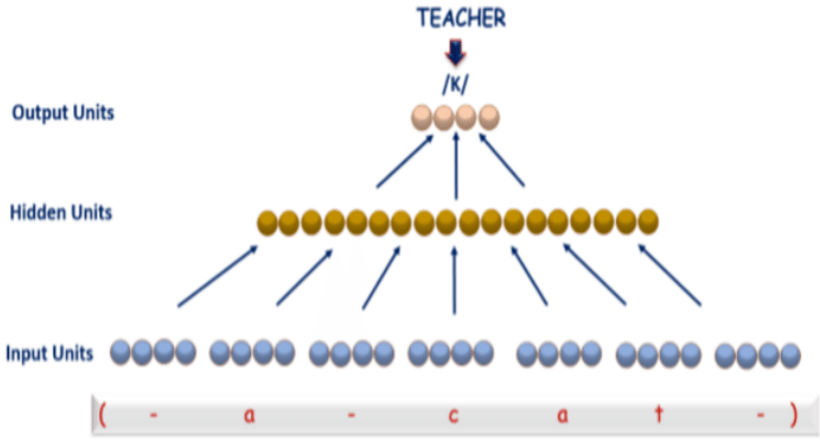
- El descubrimiento del algoritmo de Backpropagation popularizó estos modelos por más de una década.



NETtalk y LeNet

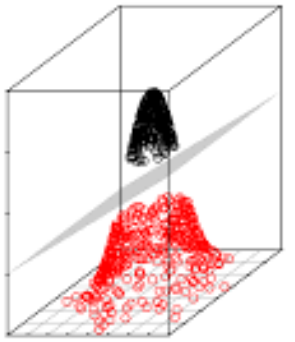
- En 1987, Sejnowski y Rosenberg crearon el sistema NETTalk, que aprendía a mapear palabras escritas en inglés a símbolos fonéticos que podían ser usados para alimentar un sintetizador de voz.

- En 1989, Yann Le Cun y otros crearon el sistema LeNet, que permitía hacer clasificación de dígitos manuscritos a partir de imágenes.



Máquinas de soporte vectorial

- Las máquinas de vectores de soporte fueron inventadas en 1992 (Boser et al. 1992).
- Las SVMs tienen una capacidad de predicción similar a la de las redes neuronales, siendo mucho más fáciles de entrenar (porque la función objetivo es convexa).
- Lo anterior inició una década de investigación sobre SVMs.
- Las SVMs no usan funciones de base adaptativas, por lo cual requieren que un experto humano diseñe la función kernel adecuada.

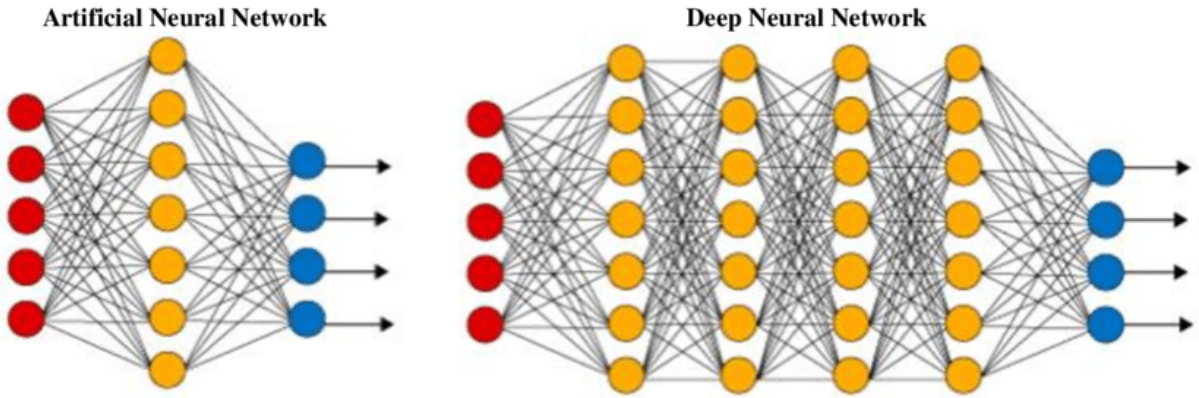


Redes Neuronales Profunda (Deep NN)

- En 2002, Geoff Hinton inventó una técnica de entrenamiento conocida como “contrastive divergence training procedure”.

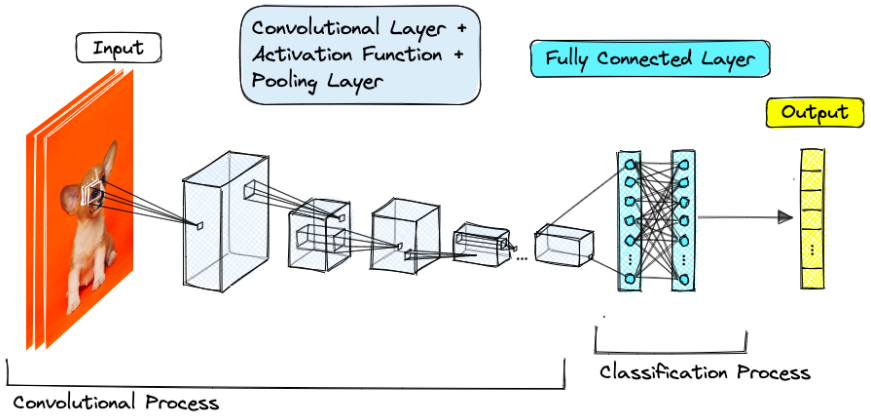
- La importancia de este algoritmo es que permitió, por primera vez, aprender redes profundas entrenando una capa a la vez, en una forma no supervisada.

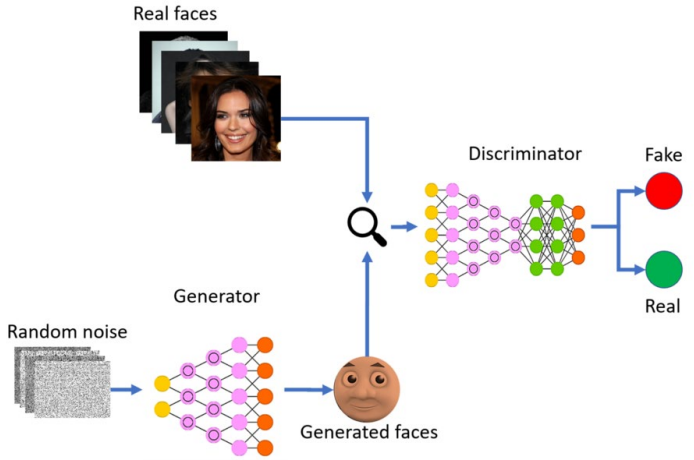
- Este descubrimiento ha generado de nuevo un interés en el área de las redes neuronales.



Post milenio

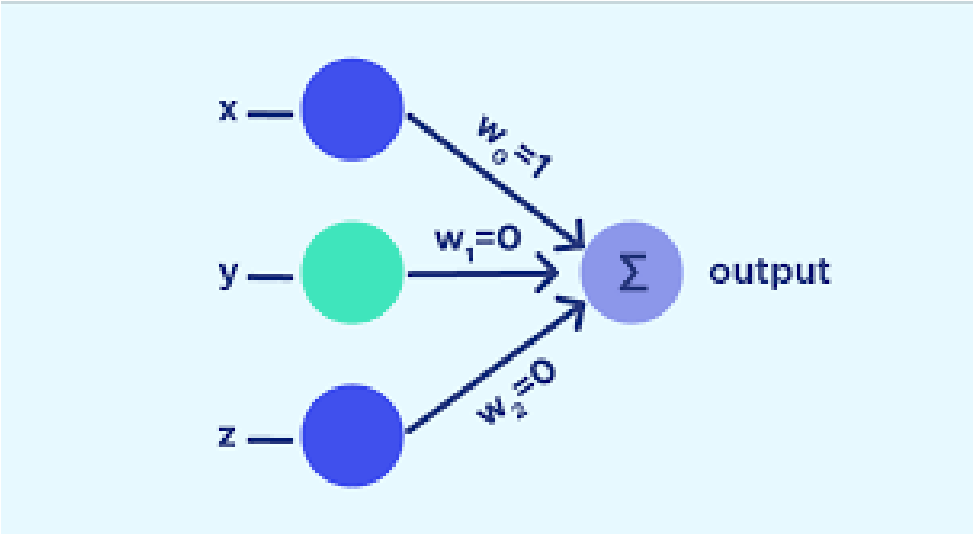
- 2006 - Geoff Hinton, Geoffrey E. y Alex Krizhevsky. Logran las redes neuronales Convolucionales, revolucionando el campo de reconocimiento de imágenes.
- 2012 – AlexNet, Una CNN gana la competencia ImageNet.
- 2014 – Se desarrollan arquitecturas profundas, ResNet
- 2015 – Generative Adversarial Networks (GANs) son propuestas por Goodfellow introduciendo el concepto de generación de datos realistas.
- 2018 BERT – Bidirectional Encoder Representations from Transformers, Destac. en procesamiento de lenguaje
- 2020 Avances continuos en áreas como transformers, Atención y modelos pre entrenados.





(feed forward NN) Propagación Hacia adelante:

Perceptrón



Red Neuronal

- Si unimos varias neuronas, todas alimentadas con la misma entrada x, habrían tantas salidas como neuronas pongamos:

$$\boldsymbol{h} = \begin{bmatrix} f(\boldsymbol{\omega}_1^\top \boldsymbol{x} + b_1) \\ f(\boldsymbol{\omega}_2^\top \boldsymbol{x} + b_2) \\ \vdots \\ f(\boldsymbol{\omega}_M^\top \boldsymbol{x} + b_M) \end{bmatrix}$$

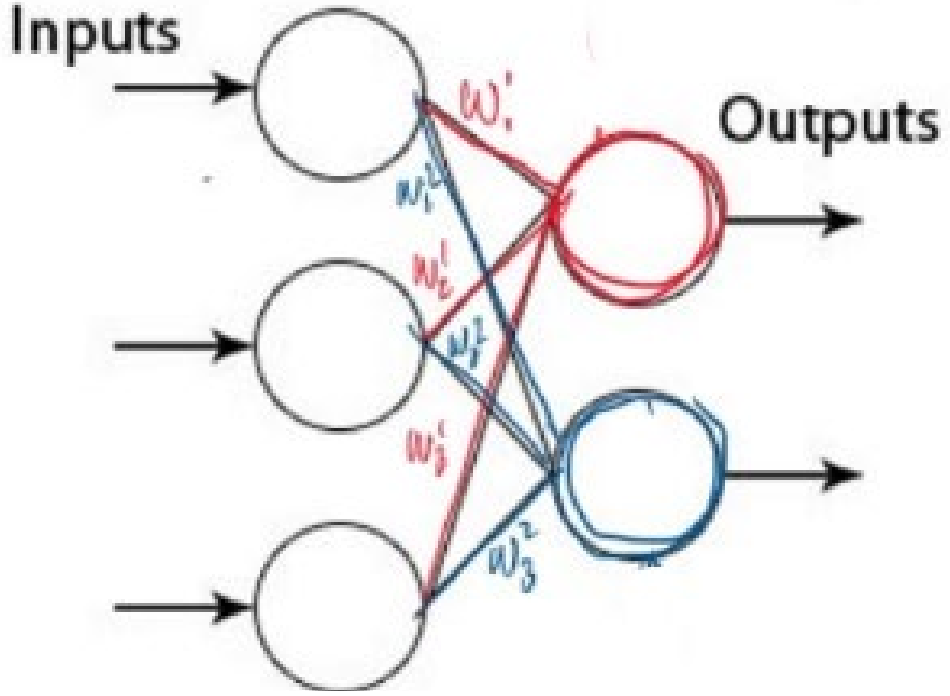
$$\boldsymbol{h} = f(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b})$$

$$\boldsymbol{W} \in R^{D \times M}$$

$$\boldsymbol{h} \in R^M$$

$$\boldsymbol{b} \in R^M$$

$$\boldsymbol{f} \in R^M$$



  - Hay que tener en cuenta que evaluar f en un vector implica evaluarla elemento a elemento en el vector.

Red Neuronal

- Si ahora alimentamos un siguiente grupo de neuronas con la salida de las anteriores obtendremos:

$$h_1 = f(W_1^T x + b_0)$$
$$h_1 = f(a_1)$$
$$a_1 = W_1^T x + b_0$$

$$h_2 = f(W_2^T h_1 + b_1)$$
$$h_2 = f(a_2)$$
$$a_2 = W_2^T h_1 + b_1$$

$W_1 \in R^{D \times M_1}$

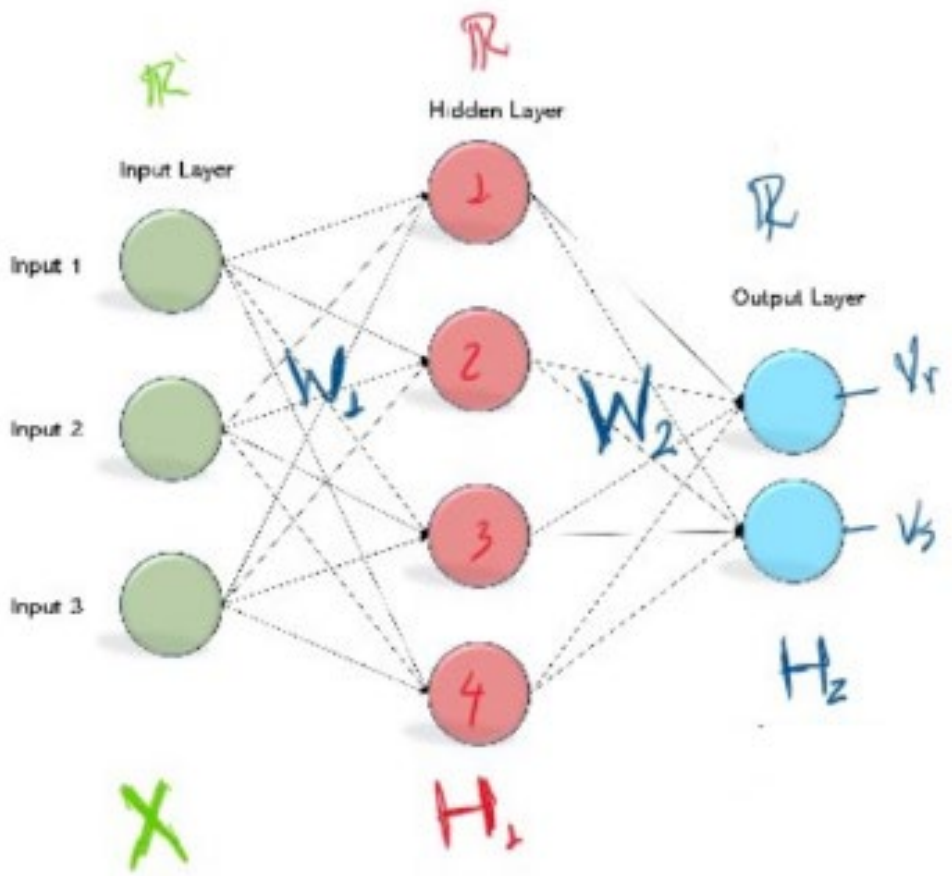
$h_1 \in R^{M_1}$

$b_0 \in R^{M_1}$

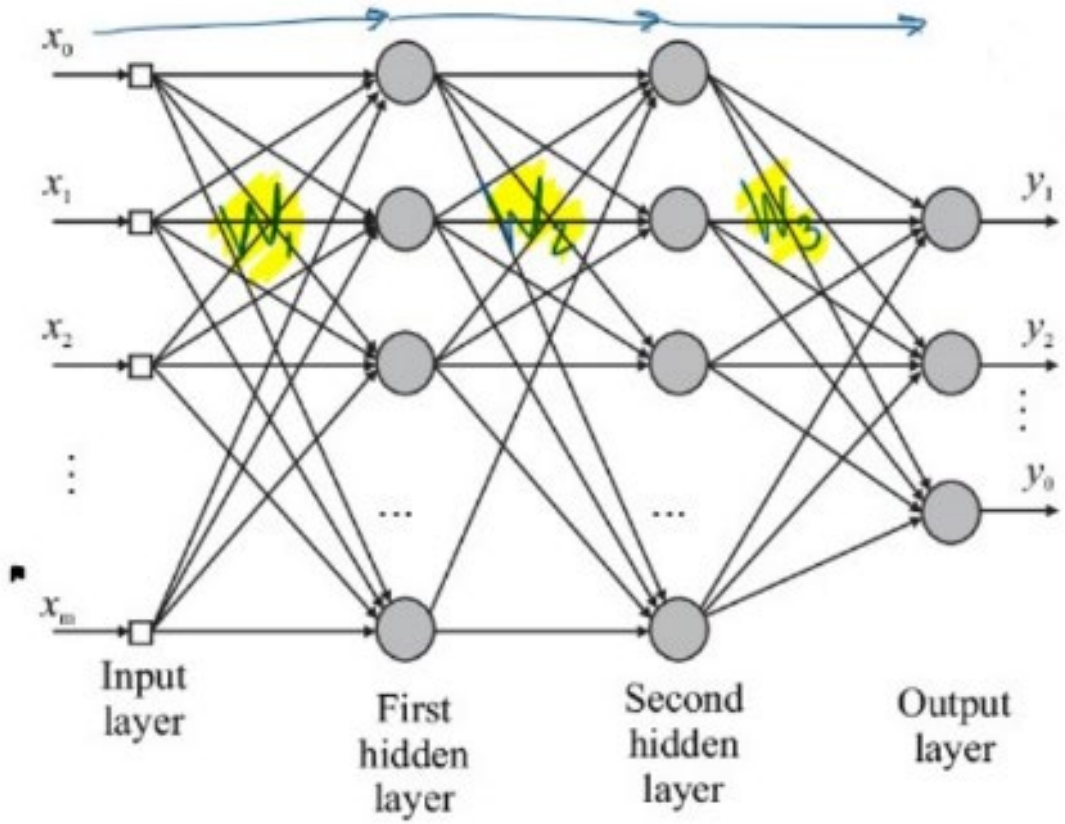
$W_2 \in R^{M_1 \times M_2}$

$h_2 \in R^{M_2}$

$b_1 \in R^{M_2}$



- De forma general, si seguimos agregando *capas* de neuronas alimentadas por una capa anterior obtendremos un modelo que propaga $\boldsymbol{x}$ desde la entrada a la salida $\boldsymbol{y}$ siempre en el mismo sentido, ***hacia adelante***:



  - Capa de entrada: $\boldsymbol{h}_0 = \boldsymbol{x}$
  - Capas internas *"ocultas"*: $\boldsymbol{h}_l = f(\boldsymbol{W}_l^T \boldsymbol{h}_{l-1} + \boldsymbol{b}_{l-1})$
  - Capa de salida: $\boldsymbol{y} = \boldsymbol{h}_L$

    donde $\boldsymbol{W}_l \in R^{M_{l-1} \times M_l}$, $\boldsymbol{h}_l \in R^{M_l}$, $\boldsymbol{b}_{l-1} \in R^{M_l}$, M_l corresponde al número de nodos en la capa $l \in \{1, \ldots, L\}$.

Red Neuronal: Modelo matemático

Combinando estas etapas se tiene

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{k,j}^{(2)} h\left(\sum_{i=1}^{D} w_{j,i}^{(1)} x_i + w_{j,0}^{(1)}\right) + w_{k,0}^{(2)}\right),$$

$$= \sigma\left(\sum_{j=0}^{M} w_{k,j}^{(2)} h\left(\sum_{i=0}^{D} w_{j,i}^{(1)} x_i\right)\right),$$

con $x_0 = 1$.

Los parámetros $\{w_{j,i}\}_{j=1,i=0}^{M,D}$ y $\{w_{k,j}\}_{k=1,j=0}^{K,M}$ se denotan conjuntamente como $\mathbf{w}$, y se organizan como vector.

Red neuronal: función no lineal de $\{x_i\}_{i=1}^{D}$ a $\{y_k\}_{k=1}^{K}$ controlada por $\mathbf{w}$.

- La cantidad de neuronas (nodos) en cada capa y la cantidad de capas constituyen la arquitectura de la capa.

- Las funciones de transferencia de salida f se transforman o no, dependiendo del problema a tratar:

  - Para tareas de regresión se usan funciones lineales $f(a) = a$ y tantas neuronas a la salida como variables a predecir sean necesarias:

$$\boldsymbol{y} = \boldsymbol{W}_L^T \boldsymbol{h}_{L-1} + \boldsymbol{b}_{L-1} = \boldsymbol{W}_L^T \boldsymbol{\varphi}(\boldsymbol{x}) + \boldsymbol{b}_{L-1}; \boldsymbol{y} \in R^P$$

  - Para tareas de clasificación se usan funciones softmax $f(a) = \sigma(a)$ con tantas neuronas a la salida como clases sean consideradas:

$$\boldsymbol{y} = \sigma(\boldsymbol{W}_L^T \boldsymbol{h}_{L-1} + \boldsymbol{b}_{L-1}); \ \boldsymbol{y} \in [0.1]^K$$

$$\sum_{k=1}^{K} y_k = 1$$

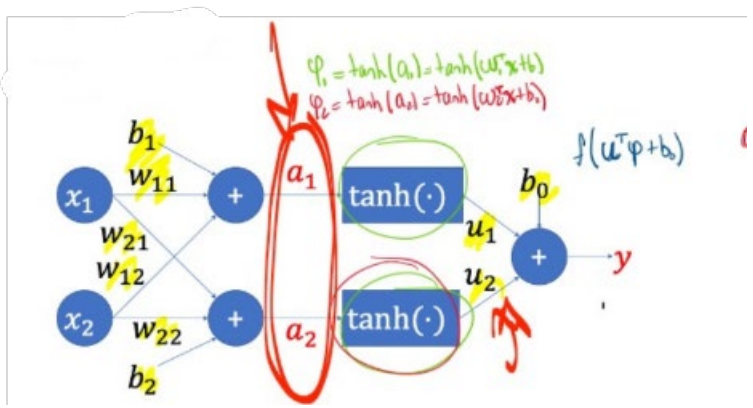
  - El uso de la softmax en clasificación permite que las salidas de la red puedan considerarse como la probabilidad a posteriori de clase:

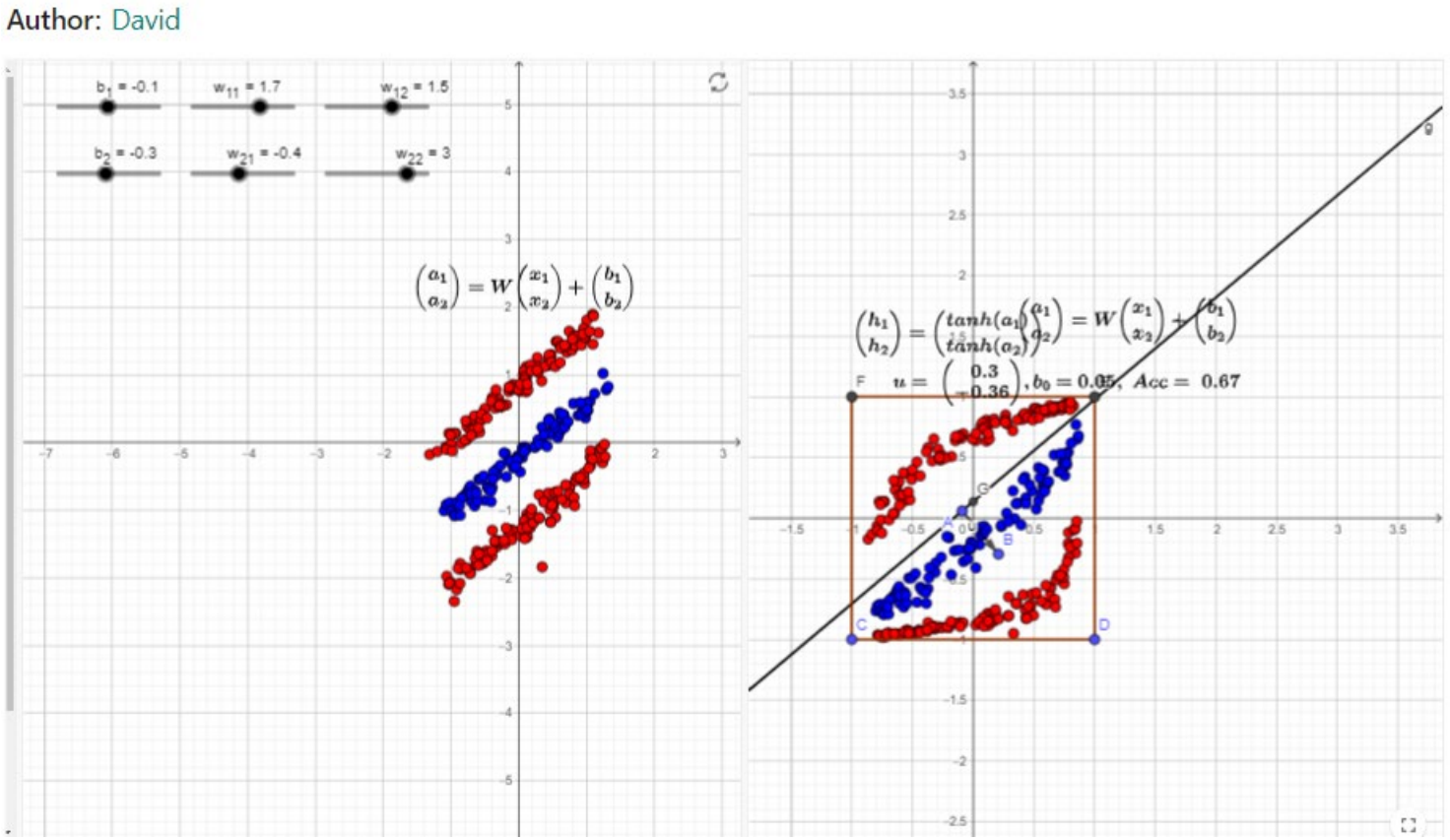
$$y_k = p(c_k|\boldsymbol{x})$$

Ejemplo didáctico .

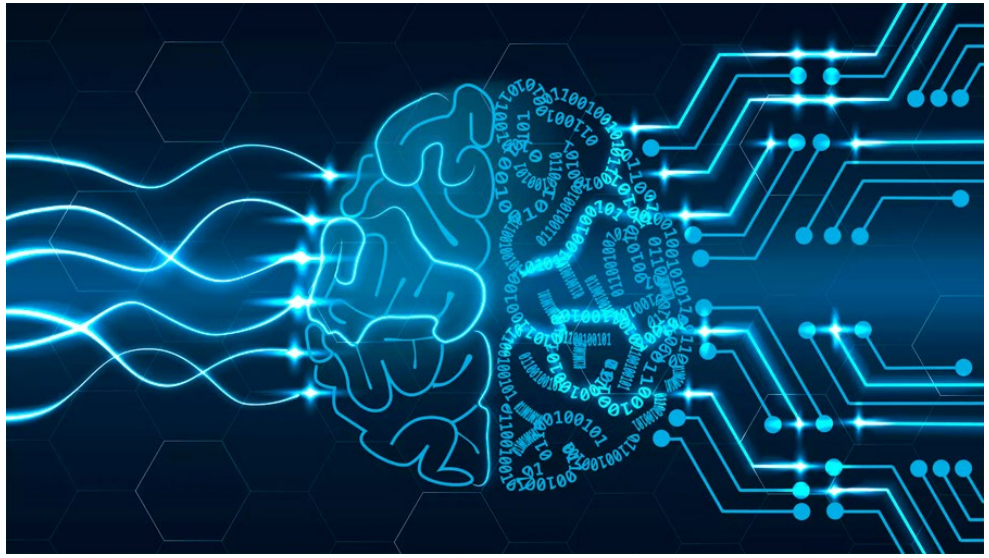
Ajustes para w y b

Con una sola frontera y activación tangencial
https://www.geogebra.org/m/gvkjkrye (de David Cardenas)





Entrenamiento de Redes



Entrenamiento de Redes:
Error y Loss en regresión y clasificación

1. Tarea:

Dado un conjunto de entrenamiento $\{\boldsymbol{x}_n, \boldsymbol{t}_n\}_{n=1}^{N}$, y una arquitectura, $\boldsymbol{y}(\boldsymbol{x}) = f(\boldsymbol{W}_L^{\top} f(\boldsymbol{W}_{L-1}^{\top} \cdots f(\boldsymbol{W}_2^{\top} f(\boldsymbol{W}_1^{\top}\boldsymbol{x} + \boldsymbol{b}_0) + \boldsymbol{b}_1) + \boldsymbol{b}_{L-2}) + \boldsymbol{b}_{L-1})$ minimizar una función de costo J con respecto al conjunto de parámetros $\boldsymbol{\theta} = \{\boldsymbol{W}_l, \boldsymbol{b}_{l-1}\}_{l=0}^{L}$:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{T})$$

- **Regresión simple**. Dado un conjunto de entrenamiento $\{\mathbf{x}_n, t_n\}_{n=1}^{N}$ se minimiza una función de error

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|y(\mathbf{w}, \mathbf{x}_n) - t_n\|^2.$$

MSE ó R2 …

- **Regresión múltiple**. Dado un conjunto de entrenamiento $\{\mathbf{x}_n, \mathbf{t}_n\}_{n=1}^{N}$ se minimiza una función de error

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{w}, \mathbf{x}_n) - \mathbf{t}_n\|^2.$$

$$J(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{T}) = \frac{1}{2}\sum_{n=1}^{N} \left|\left|\boldsymbol{y}_n(\boldsymbol{x}_n|\boldsymbol{\theta}) - \boldsymbol{t}_n\right|\right|^2$$

$$J(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{T}) = \frac{1}{2} tr\{(\boldsymbol{Y}(\boldsymbol{X}|\boldsymbol{\theta}) - \boldsymbol{T})^{\top}(\boldsymbol{Y}(\boldsymbol{X}|\boldsymbol{\theta}) - \boldsymbol{T})\}$$

1. Tarea:

Dado un conjunto de entrenamiento $\{\boldsymbol{x}_n, \boldsymbol{t}_n\}_{n=}^{N}$, y una arquitectura, $\boldsymbol{y}(\boldsymbol{x}) = f(\boldsymbol{W}_L^T f(\boldsymbol{W}_{L-1}^T \cdots f(\boldsymbol{W}_2^T f(\boldsymbol{W}_1^T \boldsymbol{x} + \boldsymbol{b}_0) + \boldsymbol{b}_1) + \boldsymbol{b}_{L-2}) + \boldsymbol{b}_{L-1})$ minimizar una función de costo J con respecto al conjunto de parámetros $\boldsymbol{\theta} = \{\boldsymbol{W}_l, \boldsymbol{b}_{l-1}\}_{l=0}^{L}$:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{T})$$

Clasificación Binaria:

- Se puede demostrar que la función de error a minimizar es

$$E(\mathbf{w}) = -\sum_{n=1}^{N} t_n \ln y_n + (1 - t_n) \ln(1 - y_n),$$

donde $y_n = \sigma(a_n) = 1/(1 + \exp(-a_n))$, y a_n es la activación de la capa de salida.

MSE ó Crossentropy …

Clasificación Múltiple:

- El vector $\mathbf{t}_n$ tiene una codificación 1 de K.
- La función de error a minimizar es

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{n,k} \ln y_k(\mathbf{w}, \mathbf{x}_n),$$

donde

$$y_k(\mathbf{w}, \mathbf{x}_n) = \frac{\exp(a_k(\mathbf{w}, \mathbf{x}_n))}{\sum_j \exp(a_j(\mathbf{w}, \mathbf{x}_n))}.$$

- Para tareas de clasificación, J puede ser el error cuadrático medio con $\boldsymbol{t}_n \in \{0,1\}^K$ en codificación 1 de K (*one hot encoding*) ó la función de cross-entropía (*crossentropy*):

$$J(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{T}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln(y_k(\boldsymbol{x}_n|\boldsymbol{\theta}))$$

$$y_k(\boldsymbol{x}_n|\boldsymbol{\theta}) = \frac{\exp(a_k^{L-1}(\boldsymbol{x}_n|\boldsymbol{\theta}))}{\sum_j \exp(a_j^{L-1}(\boldsymbol{x}_n|\boldsymbol{\theta}))} \quad : \text{softmax}(a)$$

$a_j^{L-1} \in R$ es la entrada a la última función de activación.

Entrenamiento de Redes: Optimización de parámetros Y Back-Propagation

Optimización de parámetros

- El objetivo es encontrar un vector $\mathbf{w}$ que minimice la función $E(\mathbf{w})$.
- Optimización por gradiente descendente

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}),$$

  donde $\eta > 0$ se conoce como la razón de aprendizaje.
- Otros métodos de optimización incluyen gradiente conjugado, o los métodos de cuasi-Newton.

- Asumiendo que los datos son iid, las funciones de error toman la forma:

$$J(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{T}) = \sum_{n=1}^{N} E_n(\boldsymbol{\theta})$$

donde $E_n(\boldsymbol{\theta}) \geq 0$ es el error sobre la n-ésima muestra de la base de datos.

- Por lo tanto, cada muestra contribuye de manera independiente a la derivada de la función de costo.

$$\nabla J(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{T}) = \sum_{n=1}^{N} \nabla E_n(\boldsymbol{\theta})$$

  - Calculemos las derivadas:

$$\begin{matrix} \frac{\partial E_n}{\partial \boldsymbol{W}_1} & \frac{\partial E_n}{\partial \boldsymbol{W}_2} & \cdots & \frac{\partial E_n}{\partial \boldsymbol{W}_{L-1}} & \frac{\partial E_n}{\partial \boldsymbol{W}_L} \\ \frac{\partial E_n}{\partial \boldsymbol{b}_0} & \frac{\partial E_n}{\partial \boldsymbol{b}_1} & \cdots & \frac{\partial E_n}{\partial \boldsymbol{b}_{L-2}} & \frac{\partial E_n}{\partial \boldsymbol{b}_{L-1}} \end{matrix}$$

  - Usando regla de la cadena, podemos calcular las derivadas como:

$$\frac{\partial E_n}{\partial \boldsymbol{W}_l} = \frac{\partial E_n}{\partial \boldsymbol{y}_n(\boldsymbol{x}_n)} \frac{\partial \boldsymbol{y}_n(\boldsymbol{x}_n)}{\partial \boldsymbol{W}_l}$$

$e(y(\theta)) \rightarrow \frac{de}{dy} \cdot \frac{dy}{d\theta}$

- Usando regla de la cadena, podemos calcular las derivadas como:

$$\frac{\partial E_n}{\partial \boldsymbol{W}_l} = \frac{\partial E_n}{\partial \boldsymbol{y}_n(\boldsymbol{x}_n)} \frac{\partial \boldsymbol{y}_n(\boldsymbol{x}_n)}{\partial \boldsymbol{W}_l}$$

$e(y(\theta)) \rightarrow \frac{de}{dy} \cdot \frac{dy}{d\theta}$

- Según la tarea la derivada del error respecto de la salida obtenida se puede calcular como:

Problema	E_n	$\frac{\partial E_n}{\partial \boldsymbol{y}_n(\boldsymbol{x}_n)}$
Regresión simple	$\frac{1}{2}(t_n - y_n)^2$	$y_n - t_n$
Regresión múltiple	$\frac{1}{2}\|\|\boldsymbol{t}_n - \boldsymbol{y}_n\|\|^2$	$\boldsymbol{y}_n - \boldsymbol{t}_n$
Clasificación	$-\sum_{k=1}^{K} t_{nk} \ln(y_k(\boldsymbol{x}_n \mid \boldsymbol{\theta}))$	$-[\frac{t_1}{y_1}, \frac{t_2}{y_2} \cdots \frac{t_k}{y_K}]$

$$\boldsymbol{y}_n(\boldsymbol{x}_n) = f(\boldsymbol{W}_L^\top f(\boldsymbol{W}_{L-1}^\top \cdots f(\boldsymbol{W}_2^\top f(\boldsymbol{W}_1^\top \boldsymbol{x}_n + \boldsymbol{b}_0) + \boldsymbol{b}_1) + \boldsymbol{b}_{L-2}) + \boldsymbol{b}_{L-1})$$

- La derivadas se calculan de salida a entrada. Para la última capa:

$$\boldsymbol{y}_n(\boldsymbol{x}_n) = f_L(\boldsymbol{x}_n) \qquad \boldsymbol{a}_L(\boldsymbol{x}_n) = \boldsymbol{W}_L^\top \boldsymbol{h}_{L-1}(\boldsymbol{x}_n) + \boldsymbol{b}_{L-1}$$

$$\frac{\partial E_n}{\partial \boldsymbol{W}_L} = \frac{\partial E_n}{\partial \boldsymbol{y}_n}\frac{\partial \boldsymbol{y}_n}{\partial \boldsymbol{a}_L}\frac{\partial \boldsymbol{a}_L}{\partial \boldsymbol{W}_L} \quad ?$$

$$\frac{\partial E_n}{\partial \boldsymbol{y}_n} = E_n' \qquad \frac{\partial \boldsymbol{y}_n}{\partial \boldsymbol{a}_L} = f'(\boldsymbol{a}_L) \qquad \frac{\partial \boldsymbol{a}_L}{\partial \boldsymbol{W}_L} = \boldsymbol{h}_{L-1}(\boldsymbol{x}_n)$$

$$\frac{\partial E_n}{\partial \boldsymbol{W}_L} = E_n' f'(\boldsymbol{a}_L)\boldsymbol{h}_{L-1}$$

$$\frac{df(a)}{da} = f'(a)$$

- Para la penúltima capa:

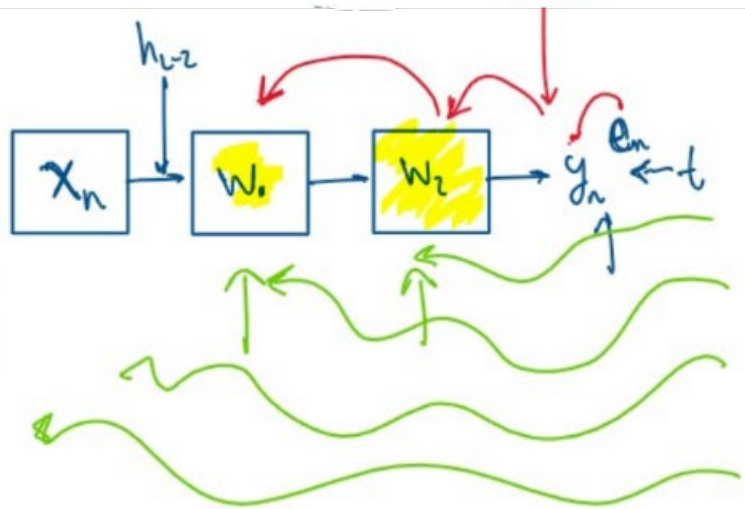
$$\boldsymbol{y}_n(\boldsymbol{x}_n) = f_L(\boldsymbol{x}_n) \qquad \boldsymbol{a}_L(\boldsymbol{x}_n) = \boldsymbol{W}_L^\top \boldsymbol{h}_{L-1}(\boldsymbol{x}_n) + \boldsymbol{b}_{L-1}$$

$$\boldsymbol{h}_{L-1}(\boldsymbol{x}_n) = f_{L-1}(\boldsymbol{x}_n) \qquad \boldsymbol{a}_{L-1}(\boldsymbol{x}_n) = \boldsymbol{W}_{L-1}^\top \boldsymbol{h}_{L-2}(\boldsymbol{x}_n) + \boldsymbol{b}_{L-2}$$

$$\frac{\partial E_n}{\partial \boldsymbol{W}_{L-1}} = \frac{\partial E_n}{\partial \boldsymbol{y}_n}\frac{\partial \boldsymbol{y}_n}{\partial \boldsymbol{a}_L}\frac{\partial \boldsymbol{a}_L}{\partial \boldsymbol{h}_{L-1}}\frac{\partial \boldsymbol{h}_{L-1}}{\partial \boldsymbol{a}_{L-1}}\frac{\partial \boldsymbol{a}_{L-1}}{\partial \boldsymbol{W}_{L-1}} \quad ?$$

$$\frac{\partial \boldsymbol{a}_L}{\partial \boldsymbol{h}_{L-1}} = \boldsymbol{W}_L \qquad \frac{\partial \boldsymbol{h}_{L-1}}{\partial \boldsymbol{a}_{L-1}} = f'(\boldsymbol{a}_{L-1}) \qquad \frac{\partial \boldsymbol{a}_{L-1}}{\partial \boldsymbol{W}_{L-1}} = \boldsymbol{h}_{L-2}$$

$$\frac{\partial E_n}{\partial \boldsymbol{W}_{L-1}} = E_n' f'(\boldsymbol{a}_L)\boldsymbol{W}_L f'(\boldsymbol{a}_{L-1})\boldsymbol{h}_{L-2}$$



- Capa $L-2$:

$$\boldsymbol{y}_n(\boldsymbol{x}_n) = f(\boldsymbol{a}_L(\boldsymbol{x}_n)) \qquad \boldsymbol{a}_L(\boldsymbol{x}_n) = \boldsymbol{W}_L^\top \boldsymbol{h}_{L-1}(\boldsymbol{x}_n) + \boldsymbol{b}_{L-1}$$

$$\boldsymbol{h}_{L-1}(\boldsymbol{x}_n) = f(\boldsymbol{a}_{L-1}(\boldsymbol{x}_n)) \qquad \boldsymbol{a}_{L-1}(\boldsymbol{x}_n) = \boldsymbol{W}_{L-1}^\top \boldsymbol{h}_{L-2}(\boldsymbol{x}_n) + \boldsymbol{b}_{L-2}$$

$$\boldsymbol{h}_{L-2}(\boldsymbol{x}_n) = f(\boldsymbol{a}_{L-2}(\boldsymbol{x}_n)) \qquad \boldsymbol{a}_{L-2}(\boldsymbol{x}_n) = \boldsymbol{W}_{L-2}^\top \boldsymbol{h}_{L-3}(\boldsymbol{x}_n) + \boldsymbol{b}_{L-3}$$

$$\frac{\partial E_n}{\partial \boldsymbol{W}_{L-2}} = \frac{\partial E_n}{\partial \boldsymbol{y}_n} \frac{\partial \boldsymbol{y}_n}{\partial \boldsymbol{a}_L} \frac{\partial \boldsymbol{a}_L}{\partial \boldsymbol{h}_{L-1}} \frac{\partial \boldsymbol{h}_{L-1}}{\partial \boldsymbol{a}_{L-1}} \frac{\partial \boldsymbol{a}_{L-1}}{\partial \boldsymbol{h}_{L-2}} \frac{\partial \boldsymbol{h}_{L-2}}{\partial \boldsymbol{a}_{L-2}} \frac{\partial \boldsymbol{a}_{L-2}}{\partial \boldsymbol{W}_{L-2}}$$

$$\frac{\partial \boldsymbol{a}_{L-1}}{\partial \boldsymbol{h}_{L-2}} = \boldsymbol{W}_{L-1} \qquad \frac{\partial \boldsymbol{h}_{L-2}}{\partial \boldsymbol{a}_{L-2}} = f'(\boldsymbol{a}_{L-2}) \qquad \frac{\partial \boldsymbol{a}_{L-2}}{\partial \boldsymbol{W}_{L-2}} = \boldsymbol{h}_{L-3}$$

$$\frac{\partial E_n}{\partial \boldsymbol{W}_{L-2}} = E_n' f'(\boldsymbol{a}_L) \boldsymbol{W}_L f'(\boldsymbol{a}_{L-1}) \boldsymbol{W}_{L-1} f'(\boldsymbol{a}_{L-2}) \boldsymbol{h}_{L-3}$$

$$\frac{\partial E_n}{\partial \boldsymbol{W}_{L-1}} = E_n' f'(\boldsymbol{a}_L) \boldsymbol{W}_L f'(\boldsymbol{a}_{L-1}) \boldsymbol{h}_{L-2}$$

$$\frac{\partial E_n}{\partial \boldsymbol{W}_L} = E_n' f'(\boldsymbol{a}_L) \boldsymbol{h}_{L-1}$$

Resumen derivadas de la función de error (RM)

- Resumiendo

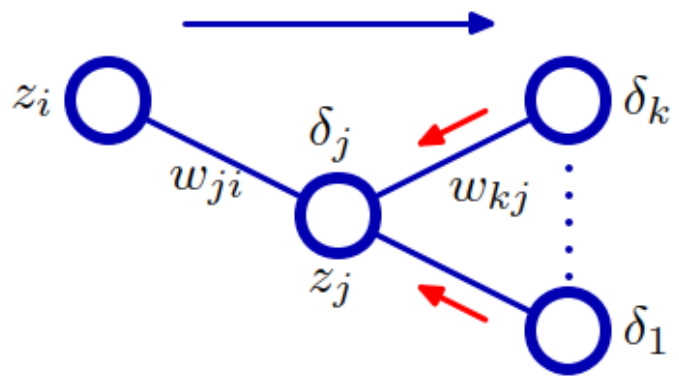
$$\frac{\partial E_n(\mathbf{w})}{\partial w_{k,j}^{(2)}} = (y_{n,k} - t_{n,k}) z_{n,j} = \delta_{n,k} z_{n,j},$$

donde $\delta_{n,k} = y_{n,k} - t_{n,k}$.

- Igualmente,

$$\frac{\partial E_n(\mathbf{w})}{\partial w_{j,i}^{(1)}} = \frac{\partial h\left(a_{n,j}\right)}{\partial a_{n,j}} \sum_{k=1}^{K} (y_{n,k} - t_{n,k}) w_{k,j}^{(2)} x_{n,i} = \delta_{n,j} x_{n,i},$$

donde $\delta_{n,j} = \frac{\partial h\left(a_{n,j}\right)}{\partial a_{n,j}} \sum_{k=1}^{K} \delta_{n,k} w_{k,j}^{(2)}$.



4. Algoritmo general de retropropagación (BP algorithm):

1. Se inicializan los pesos de la red de forma arbitraria $\boldsymbol{W}_l^{(0)}$.
2. Propagar hacia adelante (*feed-forward*) un vector de entrada $\boldsymbol{x}_n$ usando las ecuaciones para encontrar las activaciones de las capas ocultas $\boldsymbol{h}_l$ y de la salida $\boldsymbol{y}_n$.
3. Evaluar el error visto en la salida $\boldsymbol{\delta}_L(\boldsymbol{x}_n, \boldsymbol{t}_n)$.
4. Propagar de adelante hacia atrás (*back-propagate*) $\boldsymbol{\delta}_L$ para obtener el error visto en cada capa oculta $\boldsymbol{\delta}_l = g(\boldsymbol{\delta}_{l+1})$.
5. Calcular las derivadas en cada capa oculta como:

$$\frac{\partial E_n}{\partial \boldsymbol{W}_l} = \boldsymbol{\delta}_l \boldsymbol{h}_{l-1}$$

6. Se actualizan los pesos de las matrices como:

$$\boldsymbol{W}_l^{(\tau+1)} = \boldsymbol{W}_l^{(\tau)} - \eta \sum_{n=1}^{N} \frac{\partial E_n}{\partial \boldsymbol{W}_l^{(\tau)}}$$

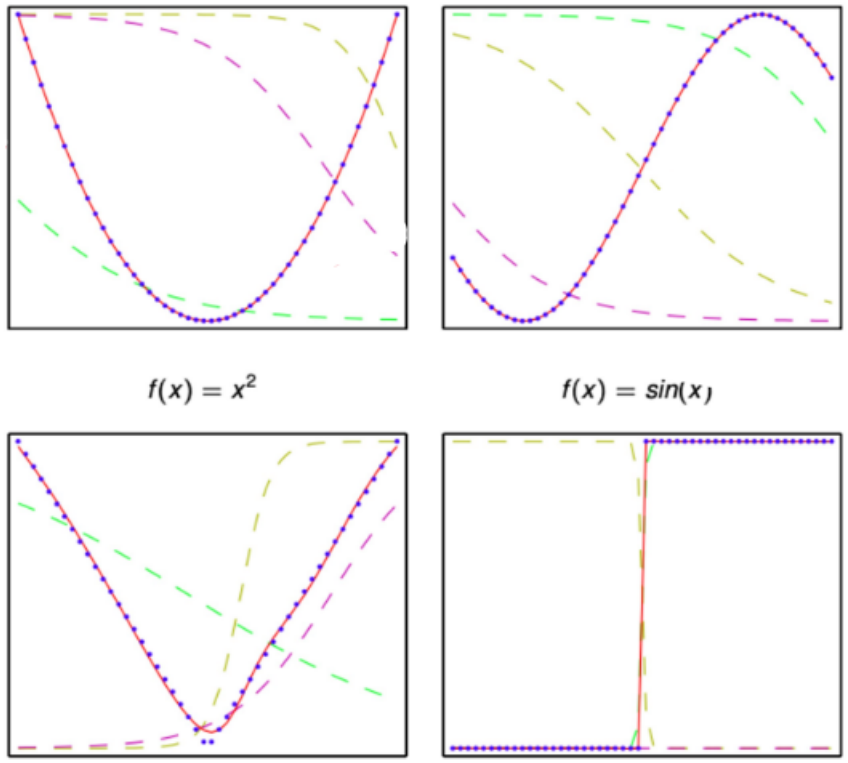
7. Se vuelve a empezar desde el paso 2 hasta que se cumpla algún criterio de convergencia.

Consideraciones:

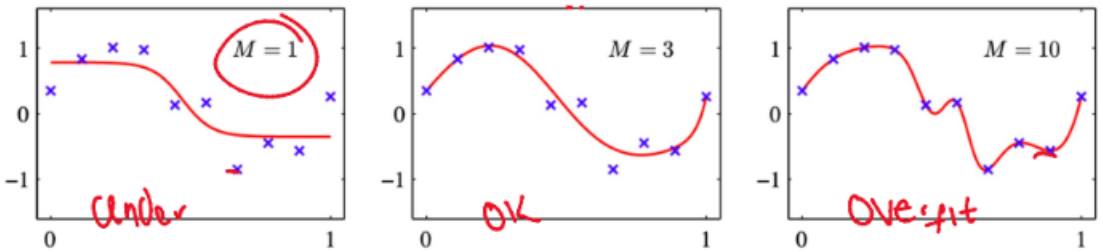
5. Consideraciones sobre el algoritmo:

- Se requieren funciones de transferencia derivables ya que se requiere $f'(a) = \frac{df}{da}$

- Las redes neuronales se consideran aproximadores *universales:*



- El tamaño de la red (número total de neuronas M) funciona como un regularizador inherente.



- Por ser un algoritmo basado en gradiente descendiente puede converger a mínimos locales. La solución actual es entrenar varias veces la misma arquitectura desde diferentes inicializaciones.

- Si la red es suficientemente grande el error en entrenamiento se puede hacer cero, incurriendo en sobreajustes. La solución actual es el entrenamiento con validación cruzada *(k-fold cross-validation)* o con Hold Out y parar de manera temprana (*early stopping*).

- Este algoritmo parte de una arquitectura está dada (nodos y capas). Para encontrar la mejor arquitectura, se exploran diferentes y se selecciona la que tenga el mejor desempeño.

Invarianzas:

Invarianzas (I)

- En diferentes aplicaciones de reconocimiento de patrones, las predicciones deben ser invariantes bajo una o más transformaciones de las variables de entrada.

- Ejemplos de aplicaciones donde esto es importante
  - Clasificación de objetos en imágenes en dos dimensiones: al objeto particular se le debe asignar la misma etiqueta independientemente de su posición en la imagen (invarianza traslacional), o de su tamaño (invarianza de escala).
  - Reconocimiento de voz: cambios pequeños de warping no lineal a lo largo del eje del tiempo, que preserven el orden temporal, no deberían cambiar la interpretación de la señal.

Invarianzas (II)

- Si el número de patrones de entrenamiento es suficiente, la red neuronal podría aprender las invarianzas, así sea de forma aproximada.

- Esto implica incluir en el conjunto de entrenamiento un número suficientemente grande de muestras con ejemplos de las diferentes transformaciones.

- Por ejemplo, para la invarianza por traslación en imágenes, el conjunto de entrenamiento debería incluir muchos ejemplos de objetos en diferentes posiciones.

Invarianzas (III)

Tres formas de incluir invarianzas en redes neuronales (ver Bishop, 2006, secciones 5.5.4 a 5.5.6):

- Propagación de un vector tangente (tangent propagation).

  Se incluye un término de regularización que penaliza los cambios en las salidas del modelo ante transformaciones de la entrada.

- Entrenamiento con datos transformados.

  Se extraen características que sean invariantes ante las transformaciones requeridas.

- Redes convolucionales (convolutional networks).

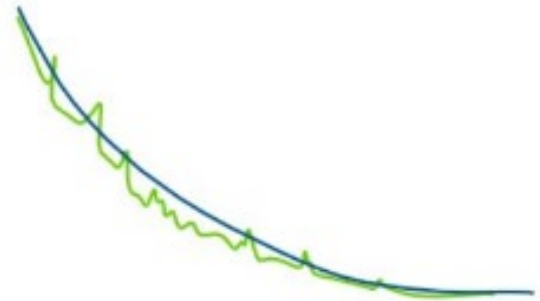
  Las propiedades de invarianza se incluyen en la estructura de la red neuronal.

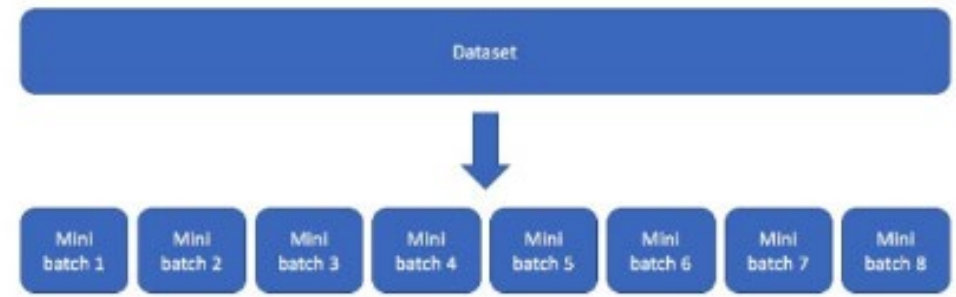
Métodos de mejoras u optimizaciones:

1. Gradiente descendiente estocástico (SGD - Stochastic Gradient Descent)

   - En cada iteración, el gradiente se calcula a partir de un subconjunto de la base de datos (*mini-batches*), $\boldsymbol{X} = \{\boldsymbol{X}_b \in \boldsymbol{R}^{N_b \times D}\}_{b=1}^{B}, \boldsymbol{T} = \{\boldsymbol{T}_b \in \boldsymbol{R}^{N_b \times P}\}_{b=1}^{B}$:

$$\nabla J^{(\tau)}(\boldsymbol{\theta}^{(\tau)} | \boldsymbol{X}_\tau, \boldsymbol{T}_\tau) = \sum_{n=1}^{N_\tau} \nabla E_n(\boldsymbol{\theta}^{(\tau)})$$





   - Los *mini-batches* se pueden armar en cada iteración.

   - Ventajas:

     - Se reduce el costo de calcular el gradiente en cada iteración. Una ventaja mayor cuando los dataset son muy grandes.

     - La aleatorización en la construcción de los *mini-batches* ayuda a evitar los mínimos locales y resulta en convergencias más suaves. *"Le dan un sacudón a la función de costo"*.

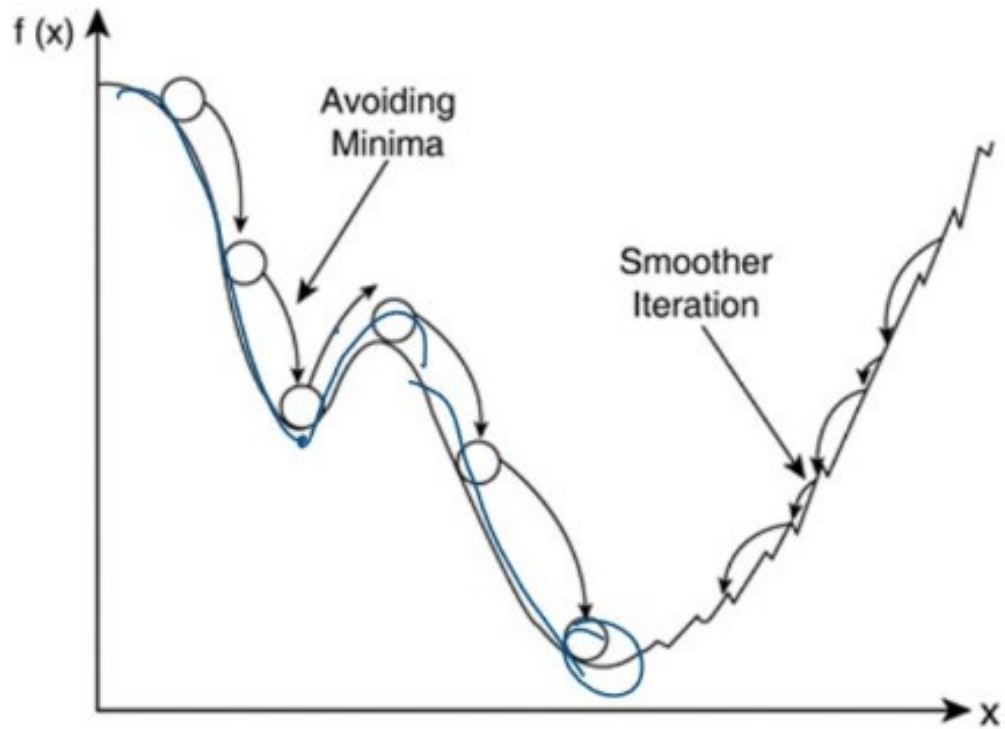
2. BP con momentum:

- El *momentum* hace que el gradiente mantenga una tendencia.
- La actualización anterior influencia la dirección actual del cambio:

$$\boldsymbol{W}_l^{(\tau+1)} = \boldsymbol{W}_l^{(\tau)} - \eta \Delta \boldsymbol{W}_l^{(\tau)}$$

$$\Delta \boldsymbol{W}_l^{(\tau)} = \frac{\partial E_n}{\partial \boldsymbol{W}_l^{(\tau)}} + \beta \Delta \boldsymbol{W}_l^{(\tau-1)}$$

$\beta > 0$ factor de momentum (memoria).



3. BP con decaimiento (Decay BP or regularized)

- En el regresor lineal, agregar $||\omega||^2$ suaviza las salidas de la máquina al "apagar" variables.

- En las redes neuronales, agregar $|w_{ij}^l|^2$ reduce la complejidad de la máquina al "apagar" neuronas:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{T}) + \alpha \sum_{ijl} |w_{ij}^l|^2$$

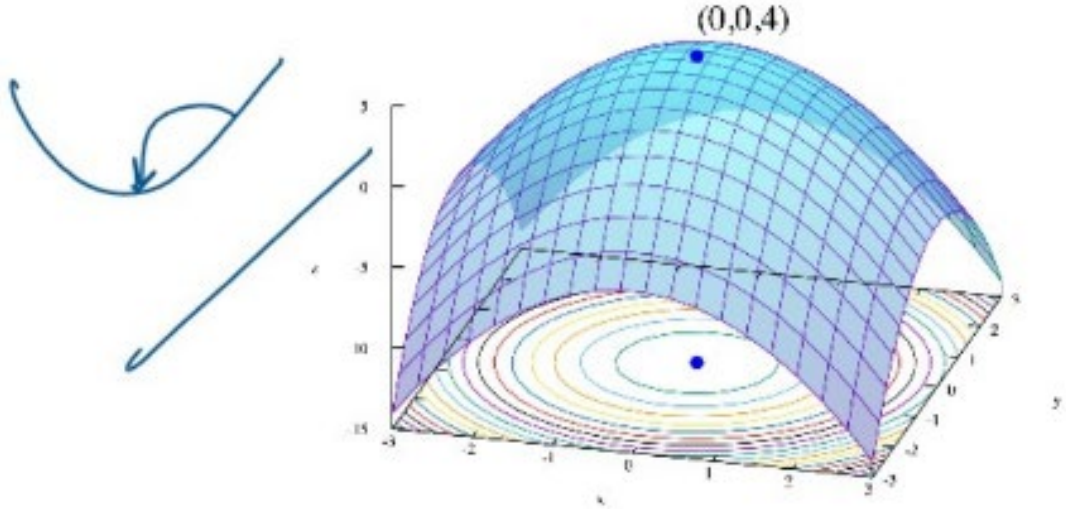
$$\Delta \boldsymbol{W}_l^{(\tau)} = \frac{\partial E_n}{\partial \boldsymbol{W}_l^{(\tau)}} + \alpha \boldsymbol{W}_l^{(\tau)}$$

$\alpha > 0$ parámetro del término de penalización L_2.

- Alternativa para reducir el sobre ajuste.

4. Métodos Quasi-Newton (BFGS, LBFGS):

- Aproxima la función de costo como una función cuadrática:



- La actualización de los pesos incluye la matriz de segundas derivadas, el Hessiano:

$$\boldsymbol{W}_l^{(\tau+1)} = \boldsymbol{W}_l^{(\tau)} - \boldsymbol{H}^{-1}\Delta\boldsymbol{W}_l^{(\tau)}$$

$$\boldsymbol{H}(ijl, i'j'l') = \frac{\partial^2 J}{\partial w_{ij}^{l}\partial w_{i'j'}^{l'}}$$

- En bases de datos pequeñas, los métodos Quasi-Newton suelen converger más rápido y con mejor desempeño.

5. ADAptive Moment estimation (ADAM):

- ADAM es una de las alternativas más recientes (2014) al SGD.

- Combina las ventajas de AdaGrad y RMSProp:
  - **AdaGrad** considera una tasa de aprendizaje a nivel de parámetro (uno por cada peso).
  - **Root Mean Square Propagation** usa una tasa de aprendizaje adaptativa basada en la magnitud de los gradientes recientes, es decir, cuantifica *¿Qué tan rápido cambia cada parámetro?*.

- Regla de actualización:

$$w_{ijl}^{(\tau+1)} = w_{ijl}^{(\tau)} - \eta \frac{m_{ijl}^{(\tau)}}{\sqrt{v_{ijl}^{(\tau)}} + \epsilon}$$

$$m_{ijl}^{(\tau)} = \beta_1 m_{ijl}^{(\tau-1)} + (1-\beta_1)\frac{\partial J}{\partial w_{ij}^{l}}$$

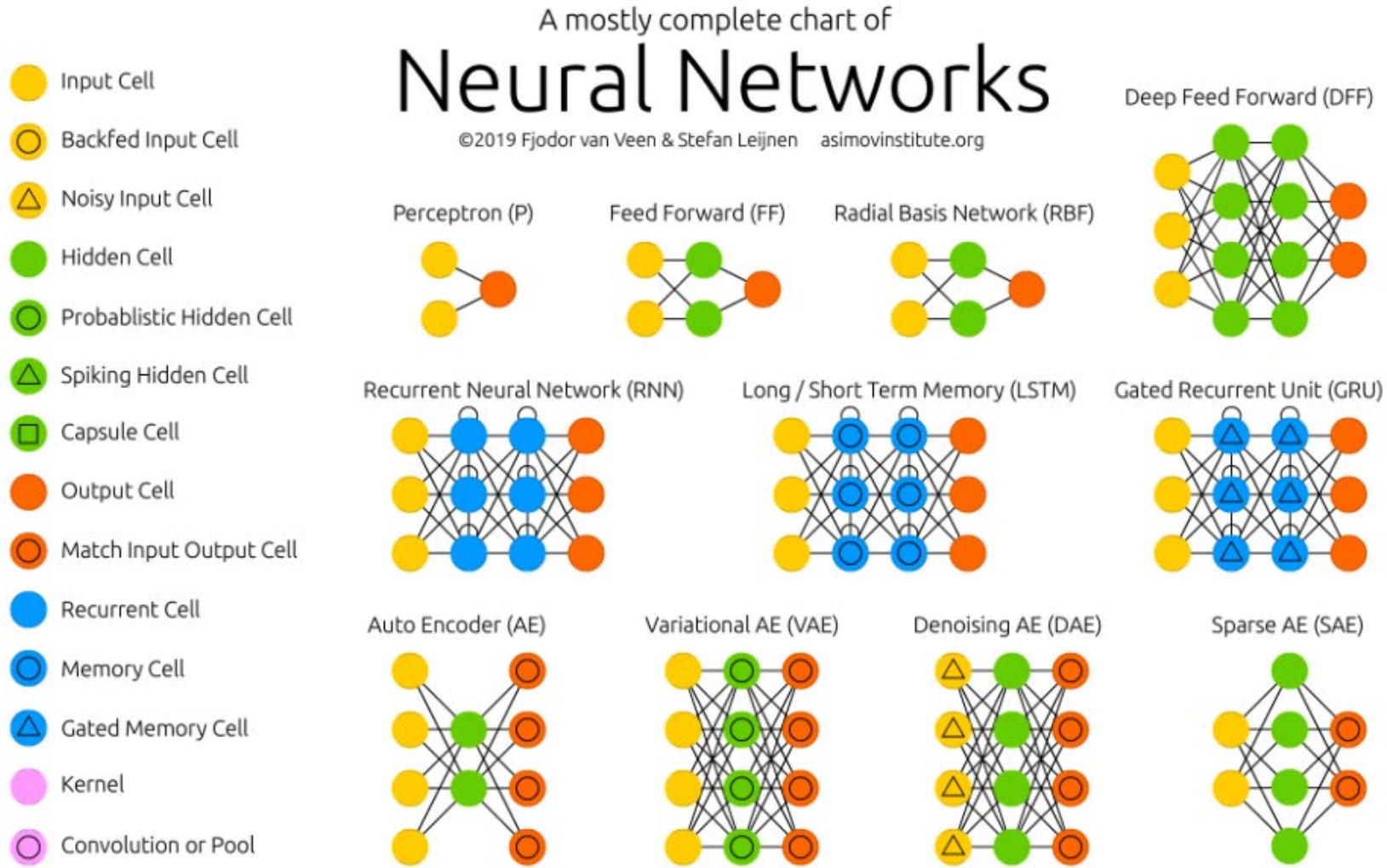
moving average del gradiente

$$v_{ijl}^{(\tau)} = \beta_2 v_{ijl}^{(\tau-1)} + (1-\beta_2)\frac{\partial^2 J}{\partial (w_{ij}^{l})^{(2)}}$$

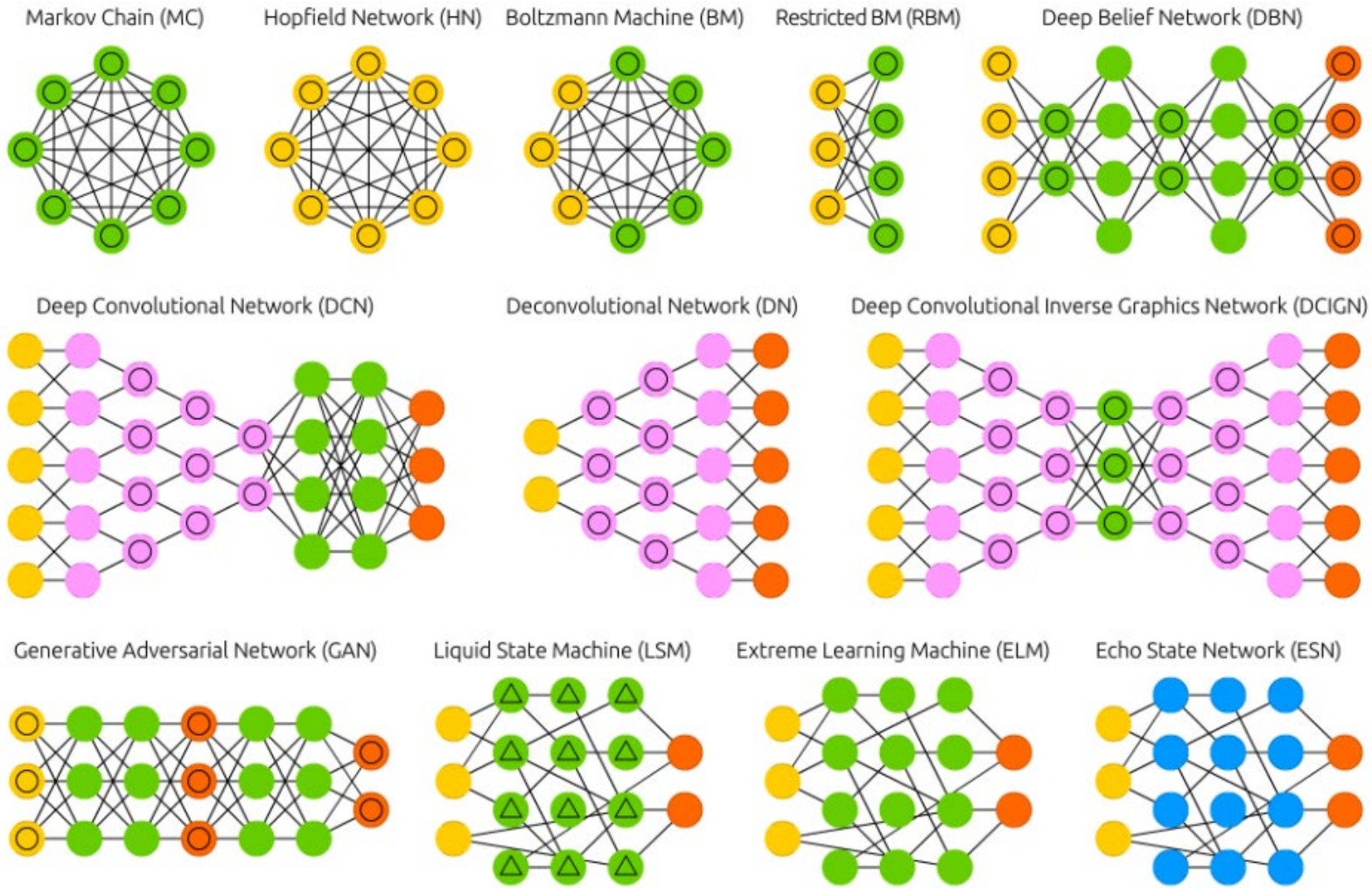
moving average de la segunda derivada

$\beta_1, \beta_2 > 0$ hiper-parámetros que ponderan la memoria de las derivadas anteriores.

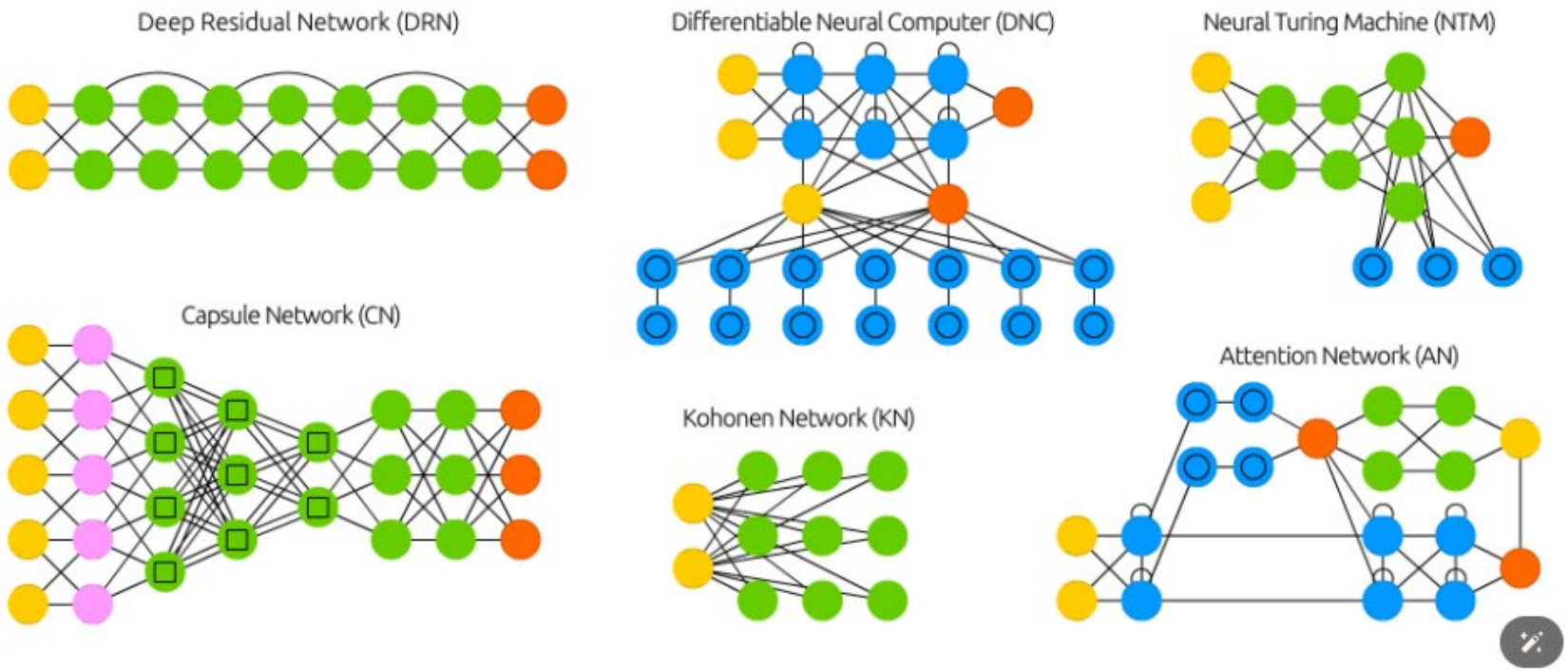
- ADAM funciona bien en grandes bases de datos ($N > 1k$).



https://www.asimovinstitute.org/neural-network-zoo/



https://www.asimovinstitute.org/neural-network-zoo/



https://www.asimovinstitute.org/neural-network-zoo/