

REDES NEURONALES

Dr. Edwin Rafael Villanueva Talavera

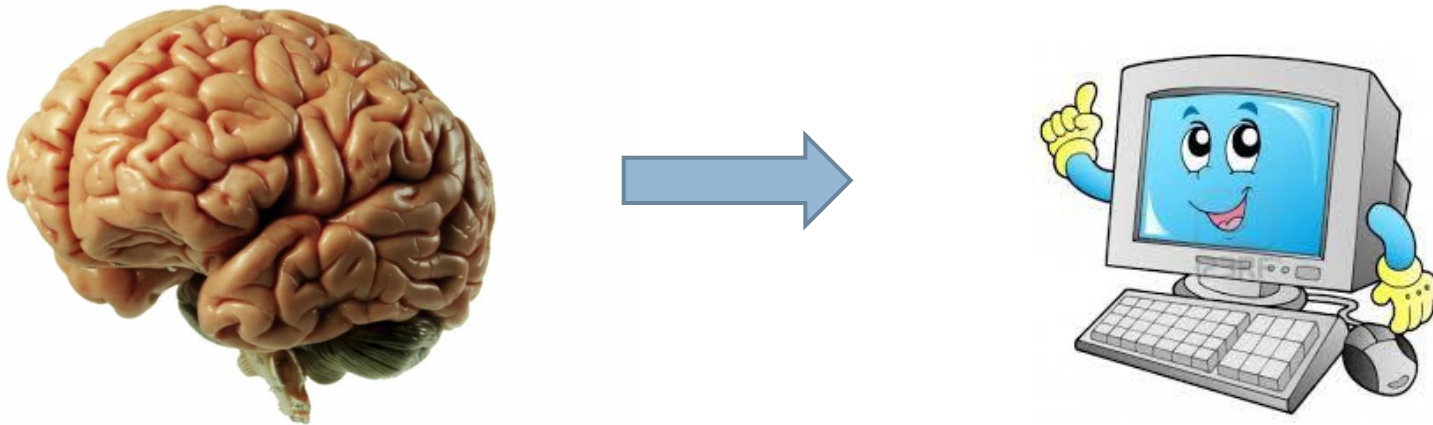
ervillanueva@pucp.edu.pe

Contenido

- Motivacion
- Perceptron
- Redes Perceptron Multicapa
- Entrenamiento
- Aplicacion

Introducción

RNA son resultado del deseo de construir artefactos capaces de exhibir comportamiento inteligente



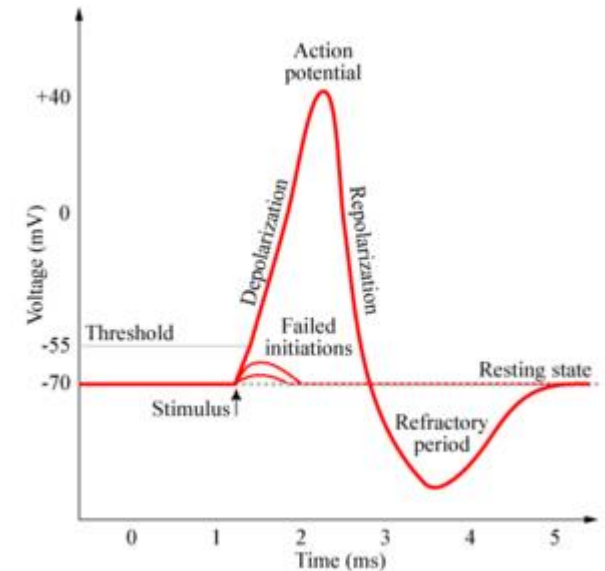
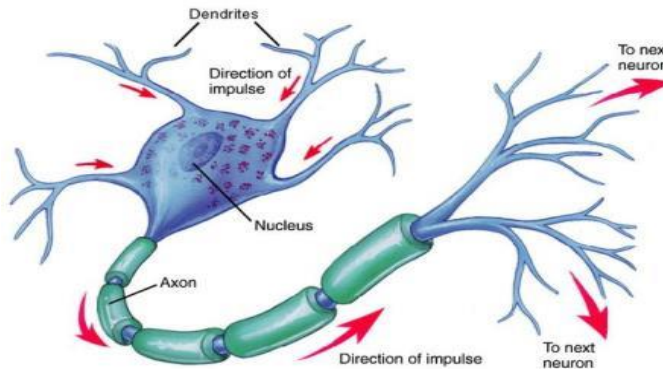
RNA son inspiradas en el sistema nervioso central de animales:

- El cerebro resuelve eficientemente problemas de procesamiento de imágenes, reconocimiento de habla, recuperación de información, aprendizaje basado en ejemplos, etc .

Historia

- McCulloch e Pitts (1943) – Modelo do Neurônio
- Rosenblatt (1958) – Algoritmo do Perceptron
- Minsky y Papert (1969) – Perceptrons
- Rumelhart, Williams, Hinton (1986) – Backpropagation
- Proceedings IEEE, IEEE Trans. Neural Networks

Neurona Biológica

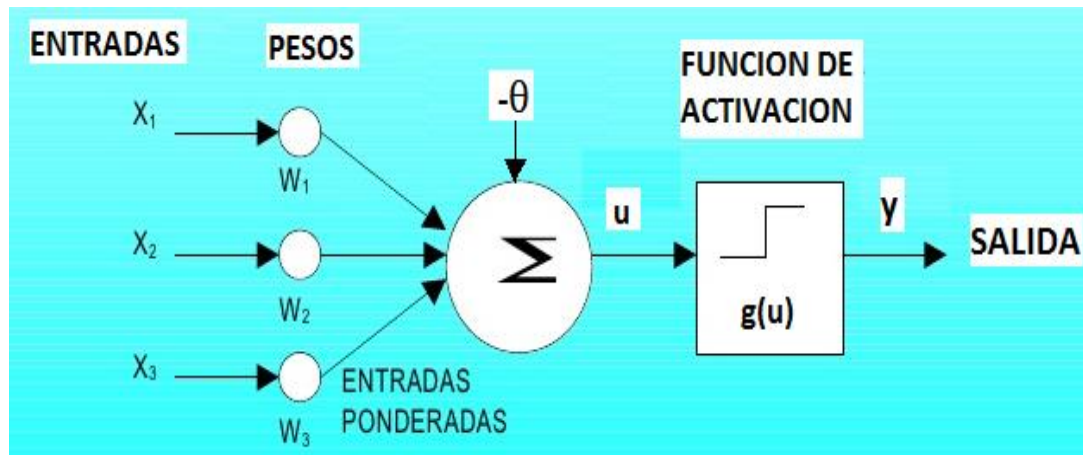


- Potencial eléctrico a través de la membrana de la célula exhibe picos.
- Pico se origina en el cuerpo celular, pasa por el axón, y hace que las terminaciones sinápticas generen neurotransmisores.
- Neurotransmisores pasan a través de las sinapsis hacia las dendritas de otras neuronas.
- Si la entrada total de neurotransmisores hacia una neurona ultrapasa un cierto limite, la neurona se dispara (genera un pico).

Velocidad Neuronal

- Las neuronas biológicas se “activan” y “desactivan” en algunos milisegundos, mientras que el hardware actual hace lo mismo en apenas nanosegundos.
- Sistemas neuronales biológicos realizan tareas cognitivas complejas (visión, reconocimiento de voz) en décimas de segundo.
- Un sistema neuronal usa “paralelismo masivo”.
- El cerebro humano tiene 10^{11} neuronas con una media de 10^4 conexiones por cada neurona.

Neurona Artificial

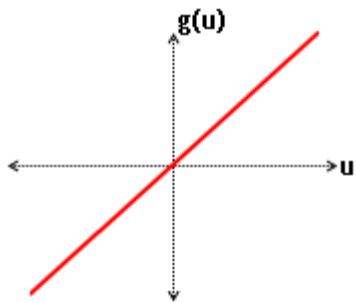


$$u = \left(\sum_{i=1}^N x_i \cdot w_i \right) - \theta$$

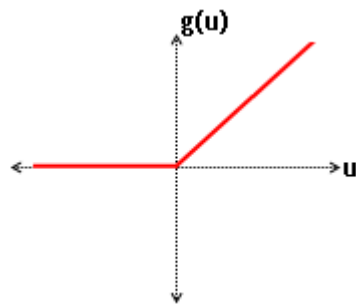
$$y = g(u)$$

Funciones de activación de Neuronas Artificiales

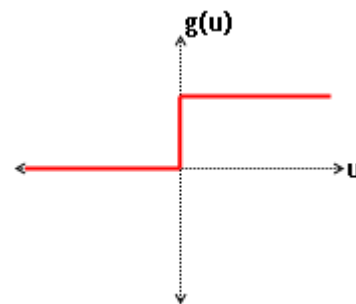
Función lineal



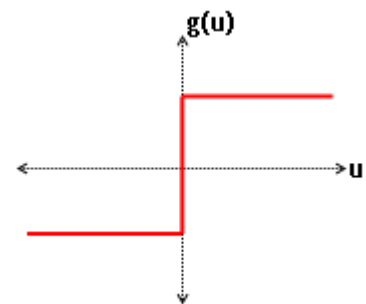
Función RELU



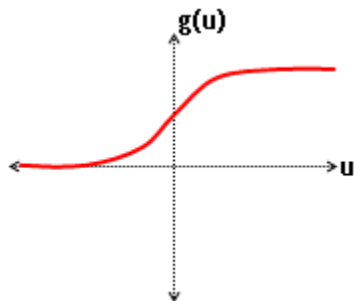
Función escalón



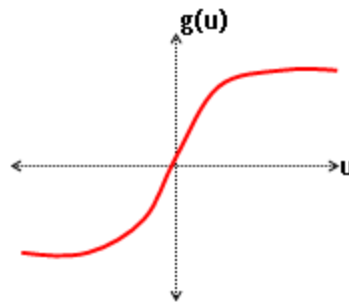
Función signo



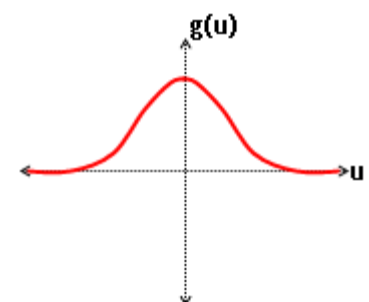
Función sigmoidea



Función tangente hiperbólica

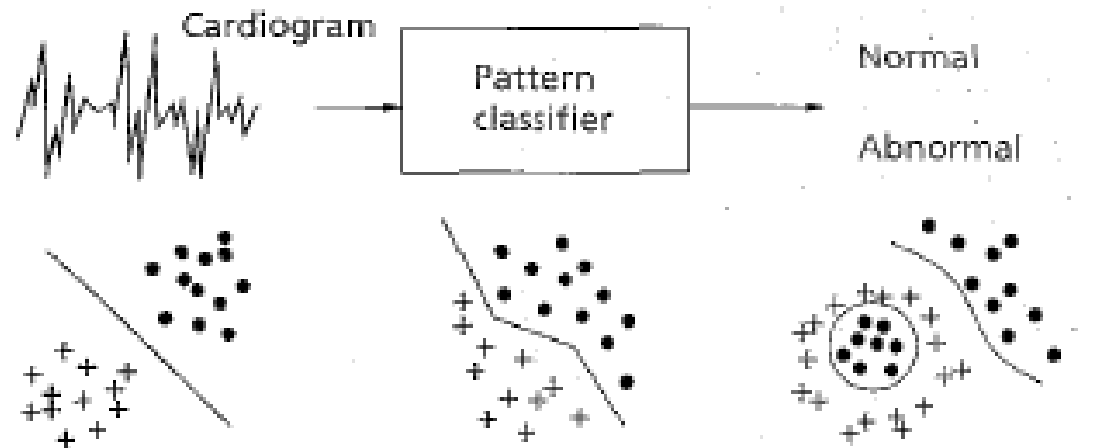


Función gaussiana

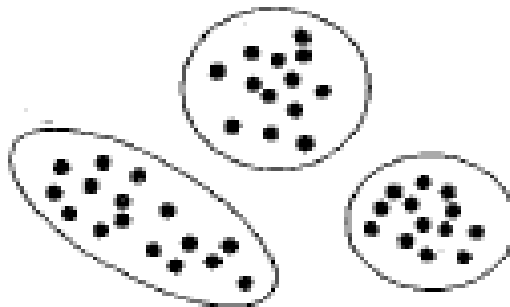


Aplicaciones de RNAs

- **Clasificación de patrones:** reconocimiento de imágenes, voz, retina, texto escrito, etc.

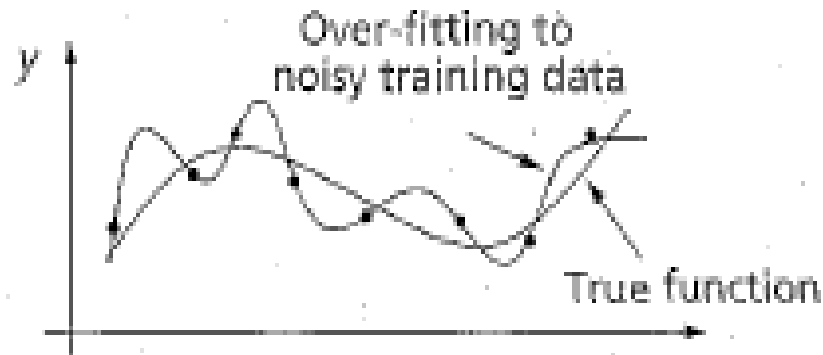


- **Clustering:** explorar similitudes y grupos, compresión de datos, etc.

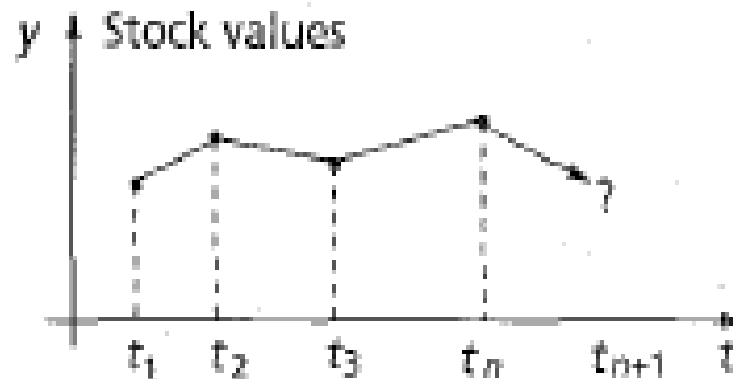


Aplicaciones de RNAs

- **Aproximación de funciones:** modelamiento científico y ingeniería



- **Previsión/Estimación:** Mercado financiero, previsión de clima, etc.

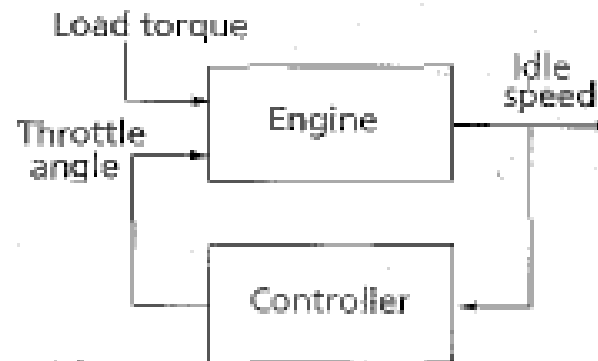


Aplicaciones de RNAs

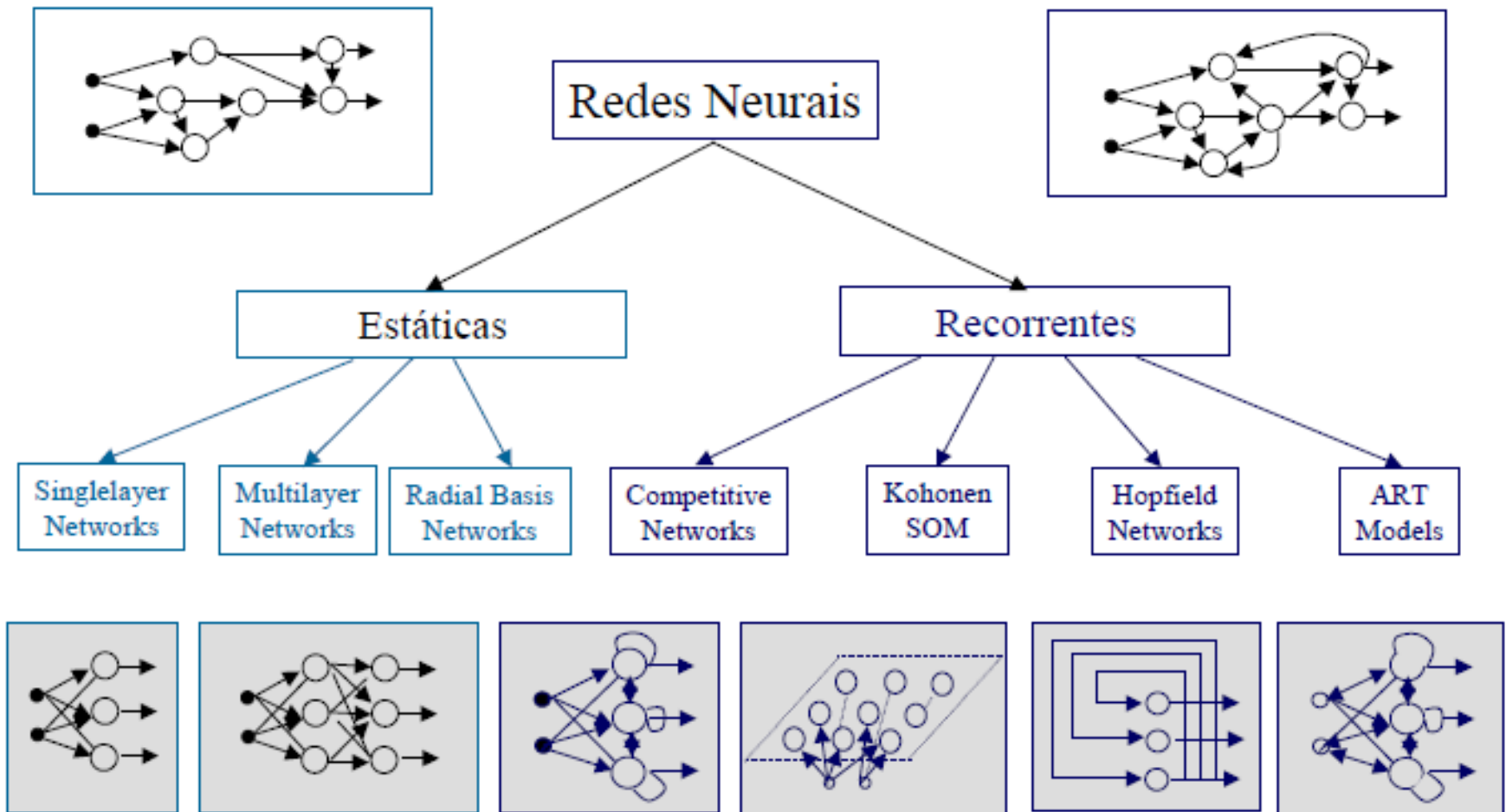
- **Memorias asociativas:** recuperar ítems por contenido, aún cuando la entrada sea distorsionada. (recuperación de imágenes, bases de datos, etc.)



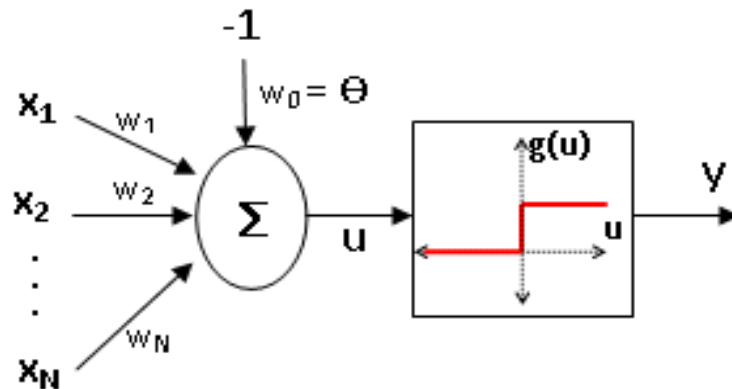
- **Control:** controlar sistemas de ingeniería (control de procesos, robótica, etc)



Tipos de RNA



Perceptron

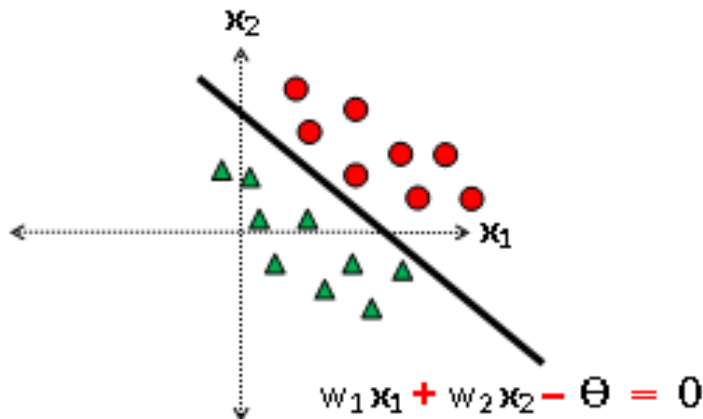


$$u = \left(\sum_{i=1}^N \mathbf{x}_i \cdot w_i \right) - \theta$$

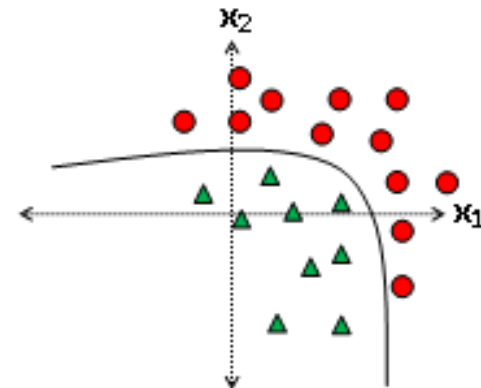
$$y = \begin{cases} 1, & \text{si } u \geq 0 \\ 0, & \text{si } u < 0 \end{cases}$$

□ Clasificación con Perceptron

Separa clases
linealmente separables



Es posible separar clases
linealmente **No** separables



Entrenamiento de un Perceptron

- Utiliza un conjunto de ejemplos de entrenamiento con tuplas $\langle \text{entrada}, \text{salida_desada} \rangle$
- El objetivo es **ajustar los pesos sinápticos** de tal forma que la red neuronal aproxime la salida deseada para cada ejemplo.
- Un algoritmo popular para el ajuste es el **perceptron** realiza actualizaciones iterativamente hasta conseguir los pesos correctos.

Entrenamiento de un Perceptron

▶ datos de entrenamiento

Datos de entrada

	$\mathbf{x}^{(1)}$	$\mathbf{x}^{(2)}$...	$\mathbf{x}^{(n)}$
x_0	-1	-1	...	-1
x_1	0.1	0.3	...	0.5
x_2	0.4	0.7	...	0.2

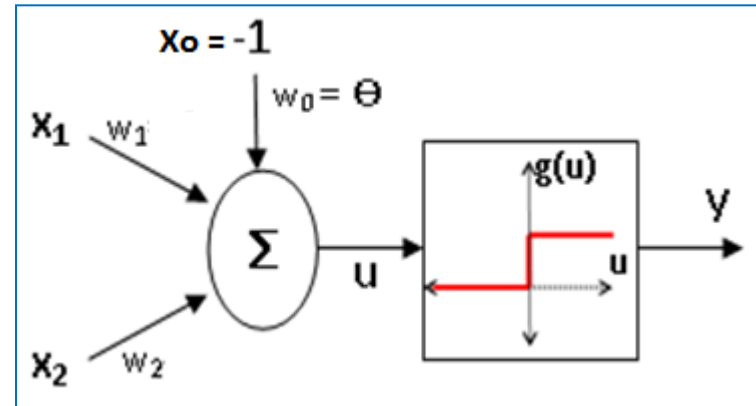
Salidas deseadas (clases)

	$d^{(1)}$	$d^{(2)}$...	$d^{(n)}$
	1	0	...	1

Vector de pesos

$$\mathbf{w} = \begin{bmatrix} \theta \\ w_1 \\ w_2 \end{bmatrix}$$

η = Taza de aprendizaje



▶ Algoritmo de entrenamiento (regla Hebbiana)

Iniciar pesos aleatoriamente

Repetir:

error = "no existe"

Para cada par de entrenamiento $\{\mathbf{x}^{(k)}, d^{(k)}\}$ hacer:

$$u = \mathbf{x}^{(k)T} \cdot \mathbf{w}$$

$$y = g(u)$$

Si $d^{(k)} \neq y$:

$$\text{error} = \text{"existe"}; \quad \mathbf{w} = \mathbf{w} + \eta(d^{(k)} - y)\mathbf{x}^{(k)}$$

Hasta que error = "no existe"

Uso de un Perceptron entrenado

- Presentar un vector de datos \mathbf{x} a clasificar
- Determinar la salida y con los pesos entrenados \mathbf{w} :

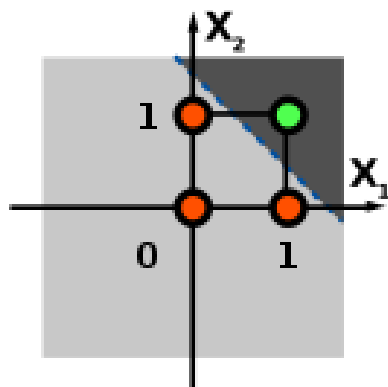
$$u = \mathbf{x}^T \cdot \mathbf{w}$$

$$y = g(u)$$

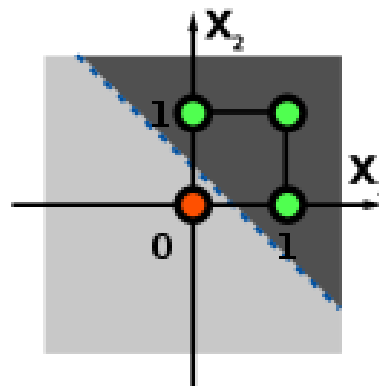
- If $y = 1$
 \mathbf{x} pertenece a la clase A
Else
 \mathbf{x} pertenece a la clase B

Limitaciones del Perceptron

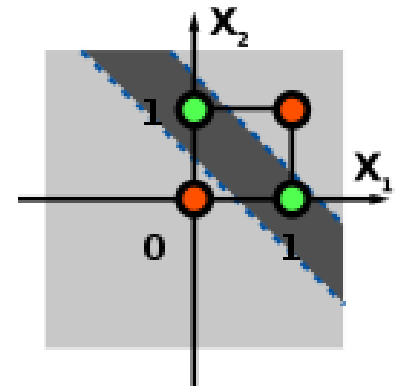
- No puede separar clases que requieran mas de un hiperplano de separación:



AND ($x_1 \cap x_2$)



OR ($x_1 \cup x_2$)

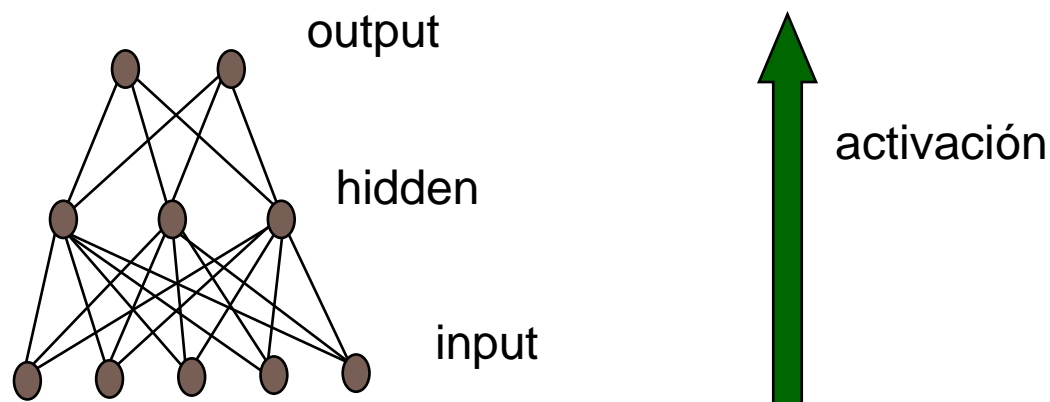


XOR

- Las funciones **AND** y **OR**, solo requieren una línea de separación y por lo tanto pueden ser aprendidas por un perceptrón.
- La función **XOR** no puede ser aprendida por un único Perceptron puesto que requiere al menos de dos líneas para separar las clases (0 y 1).

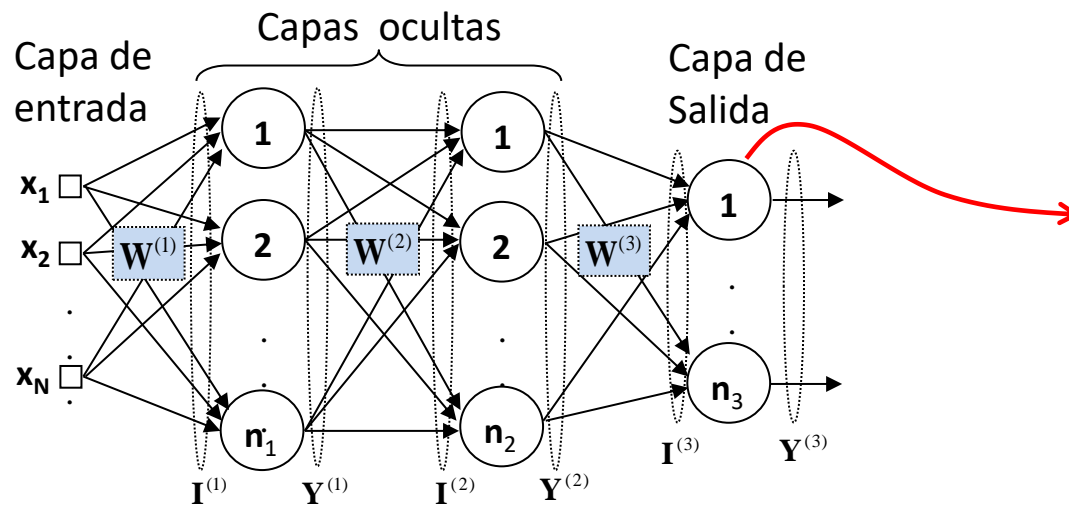
Redes Perceptron Multicapa

- Las redes multicapa pueden representar funciones arbitrarias, sin embargo aprender esas redes era considerado un problema de difícil solución (antes de los 90s).
- Una red multicapa **feed-forward** típica consta de capas de entrada, interna y salida, cada una totalmente conectada a la siguiente, con la activación yendo para adelante.

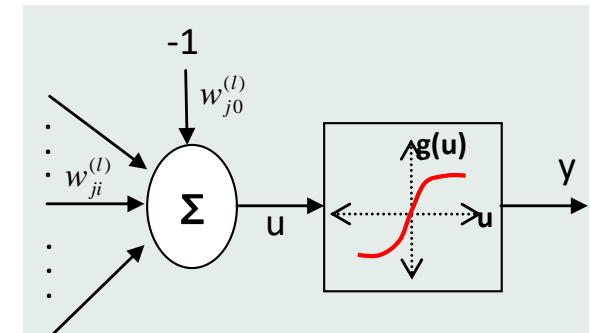


- Los pesos determinan la función calculada.

Redes Perceptron Multicapa



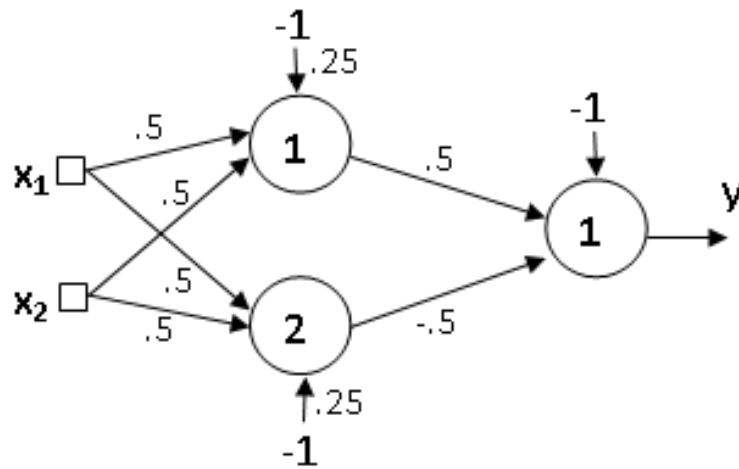
Cada neurona:



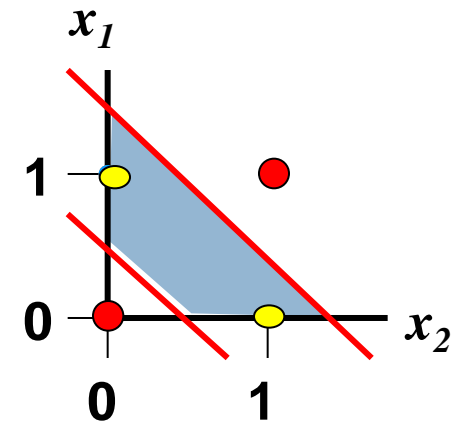
$$I_j^{(1)} = \sum_{i=1}^N w_{ji}^{(1)} \cdot x_i$$
$$Y_j^{(1)} = g(I_j^{(1)})$$

Poder de Representación de Redes Perceptron Multicapa

□ Ejemplo de Red Perceptron como puerta XOR

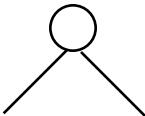
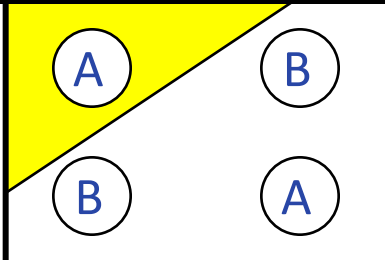
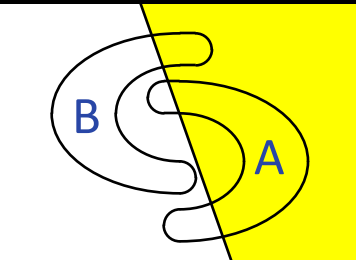
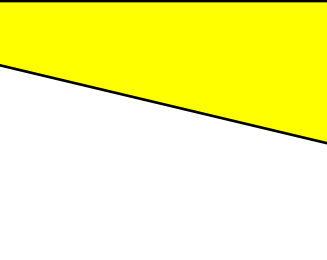
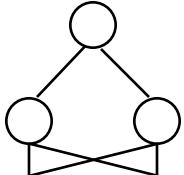
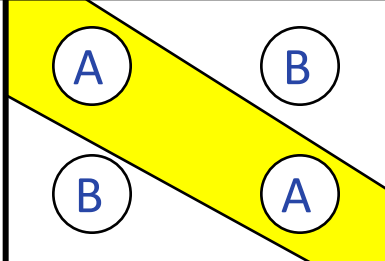
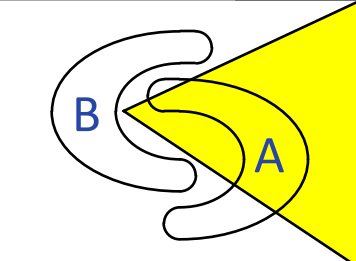
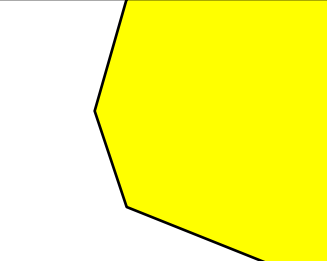
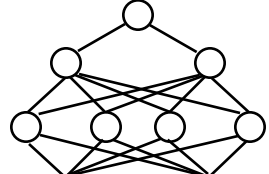
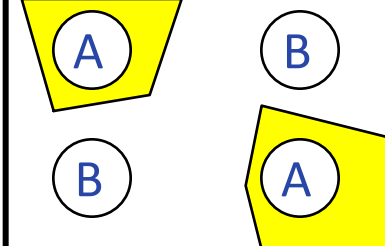
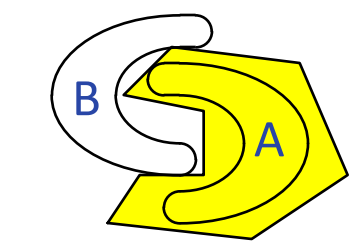
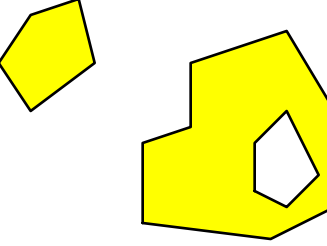


Región de Separación



Poder de Representación de Redes Perceptron Multicapa

□ Estructuras de redes Perceptron para clasificación

Estructura	XOR	Clases No-Convexas	Configuraciones Posibles
1 capa 			
2 capas 			
3 capas 			

Entrenamiento de Redes Perceptron Multicapa

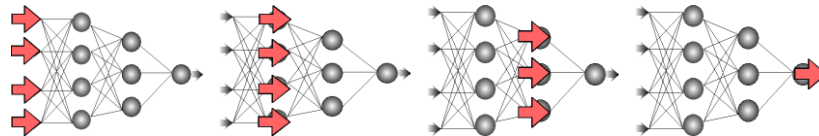
Algoritmo backpropagation

1. Inicializar los pesos de la red (por ejemplo, aleatoriamente)
2. While (not_critério_parada)

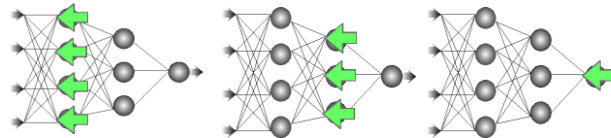
Para cada par entrada-salida $\{\mathbf{x}^{(k)}, \mathbf{d}^{(k)}\}$

Forward pass: Calcular salida Y_k para entrada $X^{(k)}$

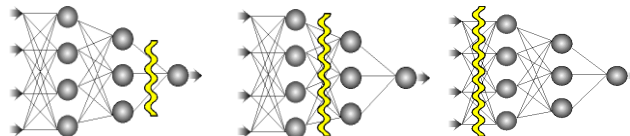
Calcular $e_k = (Y_k - d^{(k)})$, donde $d^{(k)}$ es el target



Backward pass: Calcular $\Delta w_{j,i}$ para cada capa j usando gradientes del error de cada neurona



Actualizar pesos

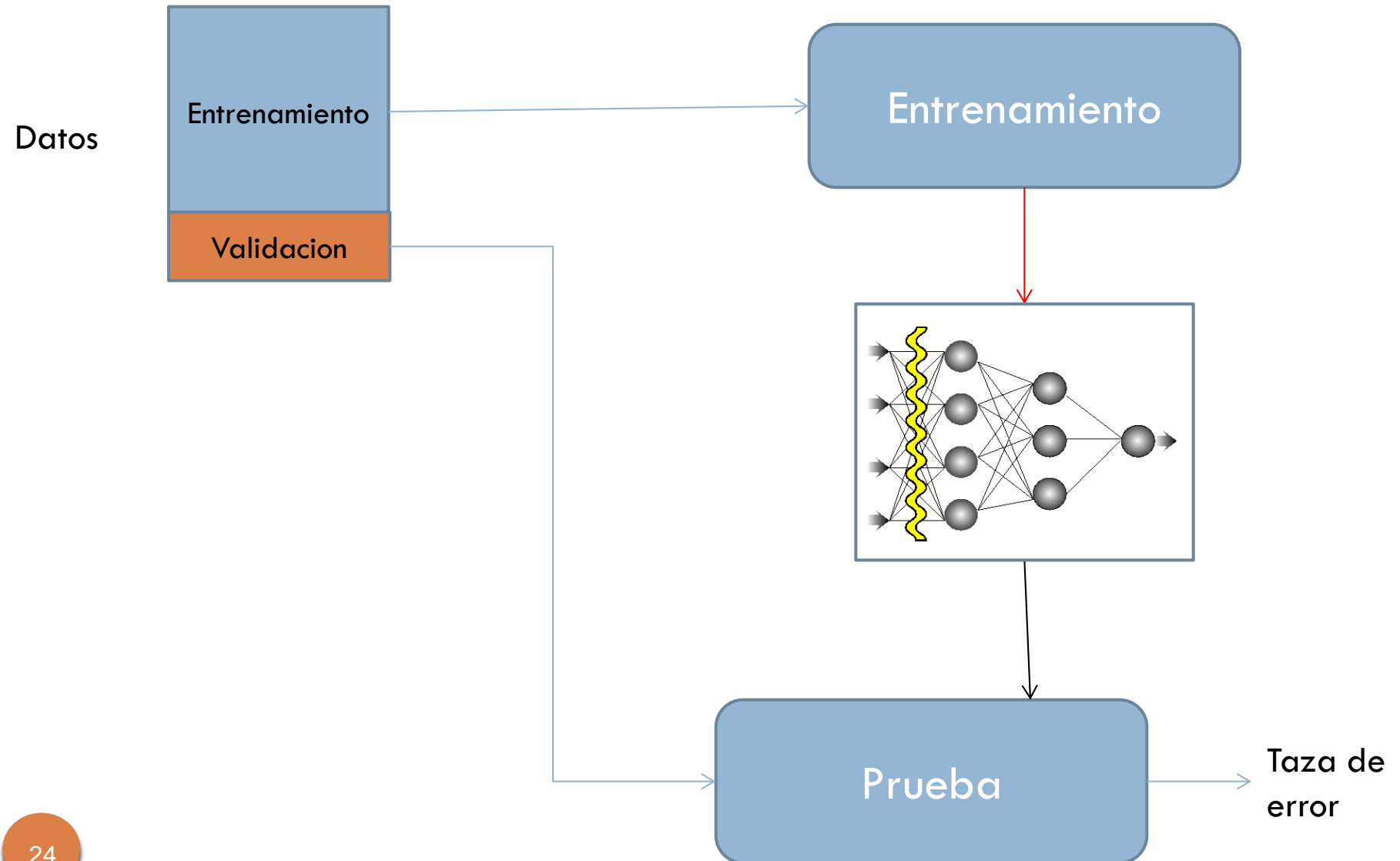


end

Comentarios sobre el algoritmo backpropagation

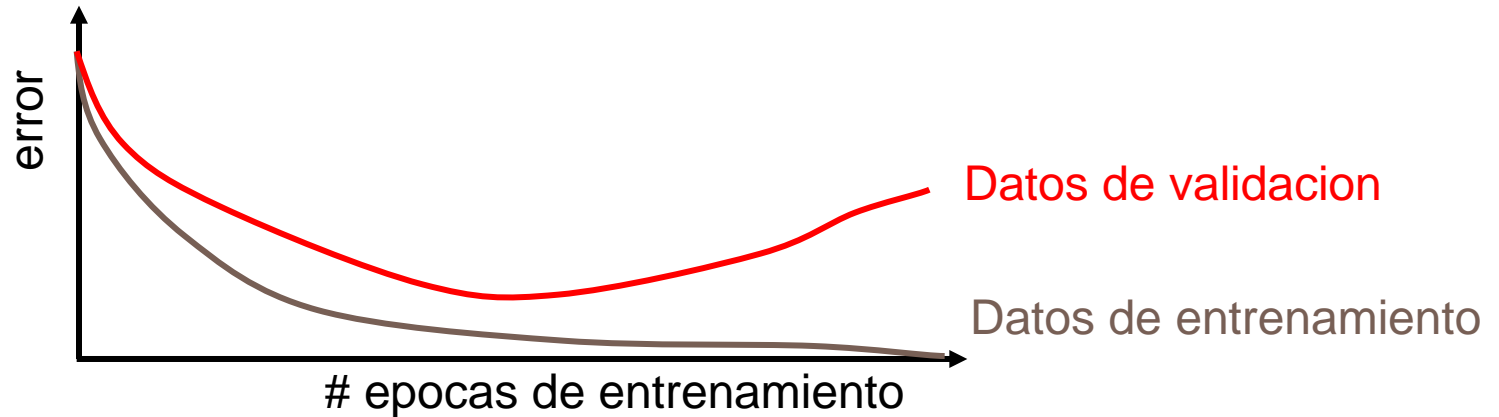
- No garantiza optimalidad – puede convergir para óptimos locales u oscilar indefinidamente.
- Puede ser necesario elevada cantidad de épocas, lo que significa horas de entrenamiento para grandes redes.

Validación de Redes Neuronales



Prevención de Sobreajuste (*Overfitting*)

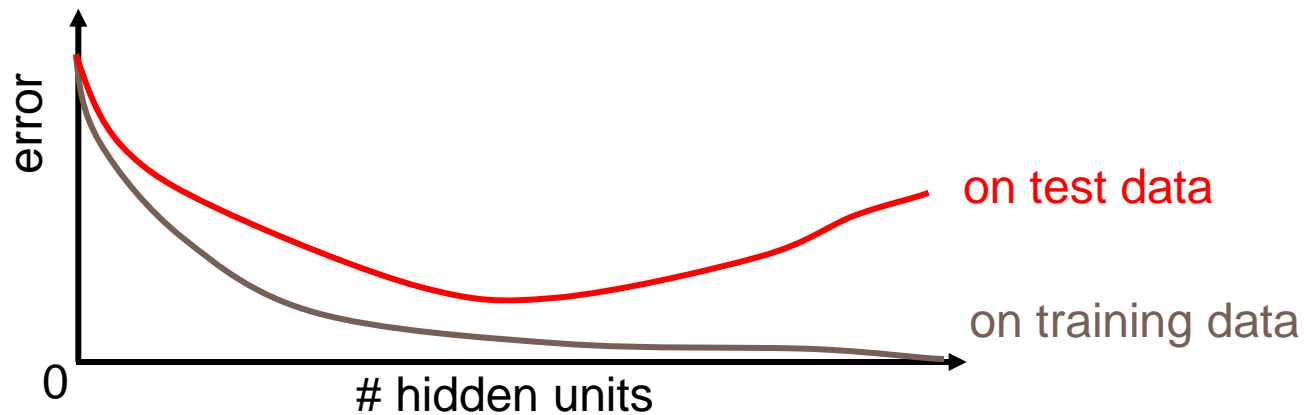
- Entrenar con muchas épocas puede llevar a sobreajuste.



- Una estrategia es usar un conjunto de validación y parar cuando el error comience a aumentar.

Determinando el mejor número de neuronas

- Pocas neuronas pueden impedir que la red se adecue totalmente a los datos.
- Muchas neuronas pueden generar un sobreajuste.



- Usar validación cruzada interna para determinar empíricamente el mejor número de neuronas internas.

Material Complementar

- Demos para jugar con redes neuronales
 - Redes Neuronales en tu browser
(<http://playground.tensorflow.org>)
 - **ConvNetJS**, Convolutional Neural Network demo
(<https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html>)