



Universidad
Francisco de Vitoria
UFV Madrid

Inteligencia Artificial II

Tema 4:
Aprendizaje Supervisado:
Modelos No lineales



Objetivos del tema

- Ubicación
 - Bloque II: **COMPUTACION NEURONAL**
 - Tema 2: *Aprendizaje no supervisado: Aprendizaje competitivo*
 - Tema 3: *Aprendizaje supervisado: Modelos lineales*
 - Tema 4: *Aprendizaje supervisado: Modelos no lineales*
- Objetivos
 - **Arquitectura y parámetros** del Perceptrón Multicapa
 - Entender el **procesamiento de un PMC** y la definición y formulación matemática asociada a la **Regla Delta Generalizada**
 - **Problemas del entrenamiento** de un modelo supervisado: sobreentrenamiento/generalización
 - Comprender el significado de los **parámetros de la RDG**
 - **Arquitectura** de las redes RBF



1. El Perceptrón Multicapa
 1. Historia
 2. Arquitectura
 3. Procesamiento
 4. ¿Podemos resolver cualquier problema?
 5. Aprendizaje
2. Regla Delta Generalizada
 1. Definición del error
 2. Ajuste de los pesos: backpropagation
 3. Entrenamiento
 4. Problemas con el algoritmo
3. Redes de Función de Base Radial
 1. Arquitectura
 2. Procesamiento
 3. Aprendizaje



- 1. El Perceptrón Multicapa**
 - 1. Historia**
 - 2. Arquitectura**
 - 3. Procesamiento**
 - 4. ¿Podemos resolver cualquier problema?**
 - 5. Aprendizaje**
- 2. Regla Delta Generalizada**
 - 1. Definición del error**
 - 2. Ajuste de los pesos: backpropagation**
 - 3. Entrenamiento**
 - 4. Problemas con el algoritmo**
- 3. Redes de Función de Base Radial**
 - 1. Arquitectura**
 - 2. Procesamiento**
 - 3. Aprendizaje**

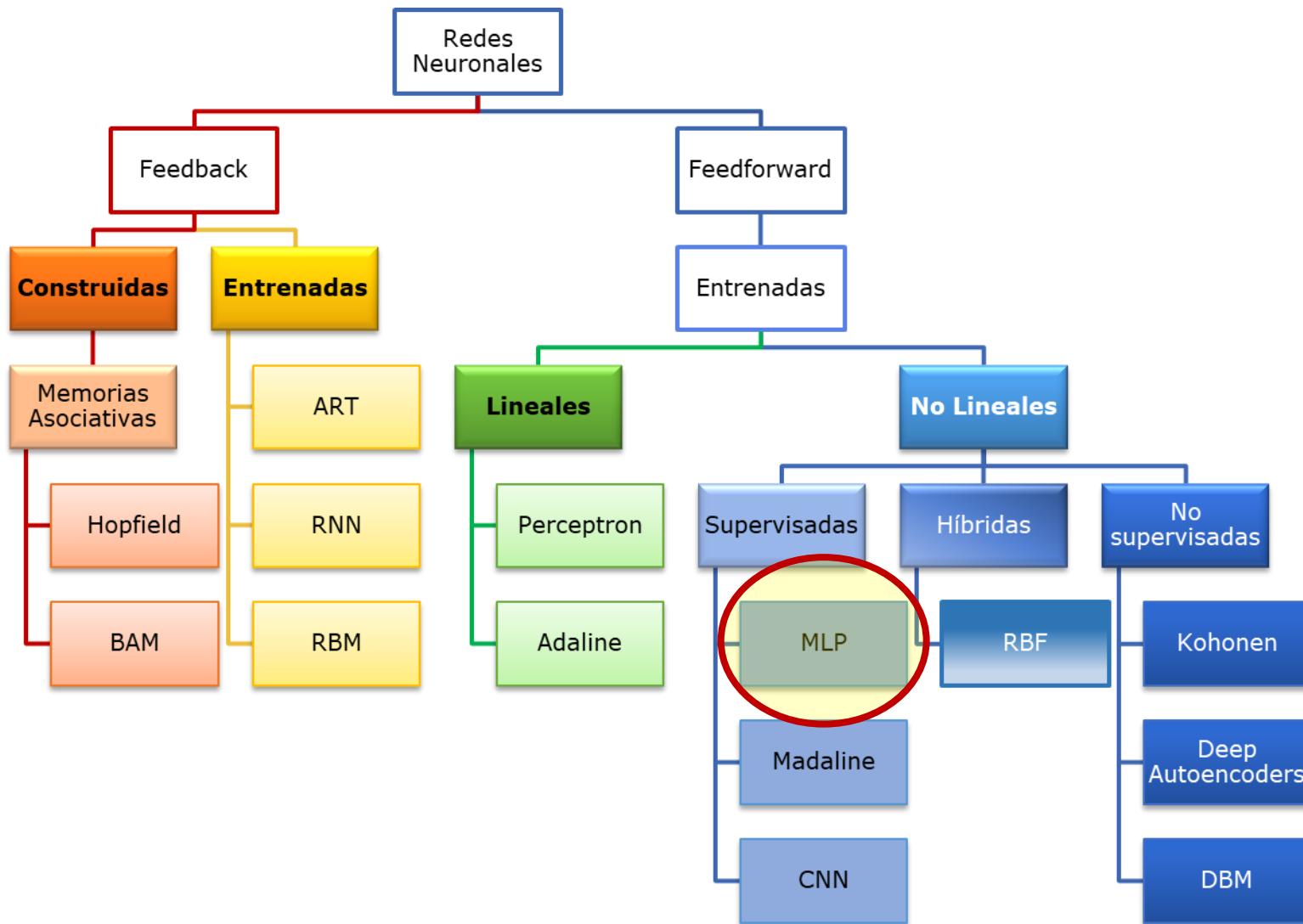


1. El Perceptrón Multicapa

- Redes neuronales
 - Modelo feedforward
 - No lineales
 - Entrenadas
 - De aprendizaje supervisado
 - [Perceptrón Multicapa](#)
 - Redes modulares
 - Reducen la complejidad y el tamaño de las redes utilizadas
 - División del problema en subtareas y construcción de una red topológicamente más simple para cada subtarea
 - Otras.... (Deep Learning)



1. El Perceptrón Multicapa



1.1 Historia



- [1986] Rumelhart, Hinton y Williams
 - Publican el algoritmo de aprendizaje back-propagation (Regla Delta Generalizada)
 - Lo aplican a una mejora del perceptrón de Rosenblatt, llamado **Perceptrón Multicapa**, combinación de varias capas de neuronas tipo perceptrón pero DERIVABLES NO LINEALES
 - Cada neurona tiene un bias que funciona de forma parecida la umbral del perceptrón, también ajustable.



David Rumelhart



Geoffrey Hinton

Variación del modelo Adaline de Widrow con aprendizaje por minimización del MSE mediante Regla Delta

1.1 Historia

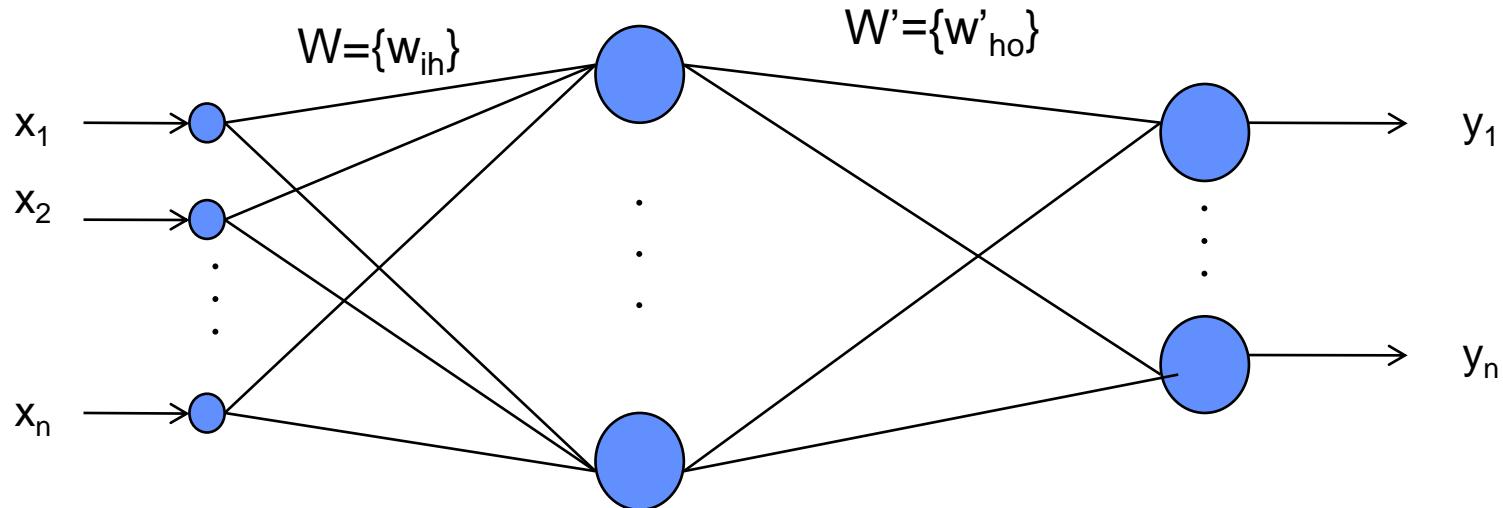


- Aplicaciones
 - **Clasificación** supervisada de patrones espaciales
 - **Reconocimiento de señales**, radar, sonar, electromagnéticas
 - **Monitorización en tiempo real**
 - **Control adaptativo** sobre múltiples variables en tiempo real
 - **Predicción de series temporales**
- Problemas en los que es asumible que se necesite previamente un tiempo largo de entrenamiento de la red...
... Y en los que se requieren tiempos cortos para evaluar una nueva instancia
- Ventajas del modelo PMC
 - Capacidad de generalización
 - Soluciona la mayoría de problemas a los que se aplica
 - Una vez entrenada, produce en runtime resultados muy rápidos



1.2 Arquitectura

- Red neuronal con al menos 3 capas:
 - Entrada
 - Salida
 - Uno o más ocultas (intermedias)
- Cada neurona en cada capa está conectada a todas las neuronas de la siguiente capa.



1.2 Arquitectura



- Cada neurona procesa la información recibida y la respuesta se propaga actuando como entrada para todas las neuronas de la siguiente capa
- Combinando unidades en distintas capas (y siempre que F sea no lineal) aumenta la capacidad expresiva de la red (el número de funciones/hiperespacios no lineales que puede implementar)
- Matriz de pesos: cada capa está completamente conectada con la siguiente
 - W_{ih} → matriz de pesos capa de entrada - capa oculta
 - W_{ho} → matriz de pesos capa oculta – capa de salida
- Las conexiones se ajustan durante el entrenamiento
 - Presentación sucesiva y reiterada de pares de vectores en las capas de entrada y salida
 - La red crea un modelo a base de ajustar la matriz de pesos en función de los pares de vectores de entrenamiento



1.2 Arquitectura

Problemas de arquitectura

■ ¿Cuántas capas ocultas?

- En una capa oculta no hay forma de saber cual debería ser la salida adecuada de las neuronas de esta capa → No se puede calcular el ECM
- Teóricamente puede haber cualquier número de capas ocultas
- Esta decisión si tiene influencia en la fase de entrenamiento
- En principio, una única capa oculta valdría para cualquier modelo multicapa

Teorema de Aproximación Universal



1.2 Arquitectura

■ ¿Cuántas neuronas en cada capa?

- Obvio el número de neuronas en las capas de entrada y salida
- Número de neuronas totales
 - Balance entre capacidad y velocidad de aprendizaje
 - Menor que el número de patrones de entrenamiento
 - Lo contrario hace que cada neurona de la capa intermedia memorice un patrón de entrenamiento en lugar de generalizar a partir de casos individuales
- En la mayoría de las situaciones, no hay forma de calcular el mejor número de neuronas en la capa oculta sin entrenar varias redes y estimar el error para cada una.
 - Pocas neuronas → el error de entrenamiento muy grande debido al *underfitting* (no se entrena a la red lo suficiente)
 - Demasiadas neuronas → error de entrenamiento bajo pero error alto de generalización debido al *overfitting* (se entrena demasiado a la red)



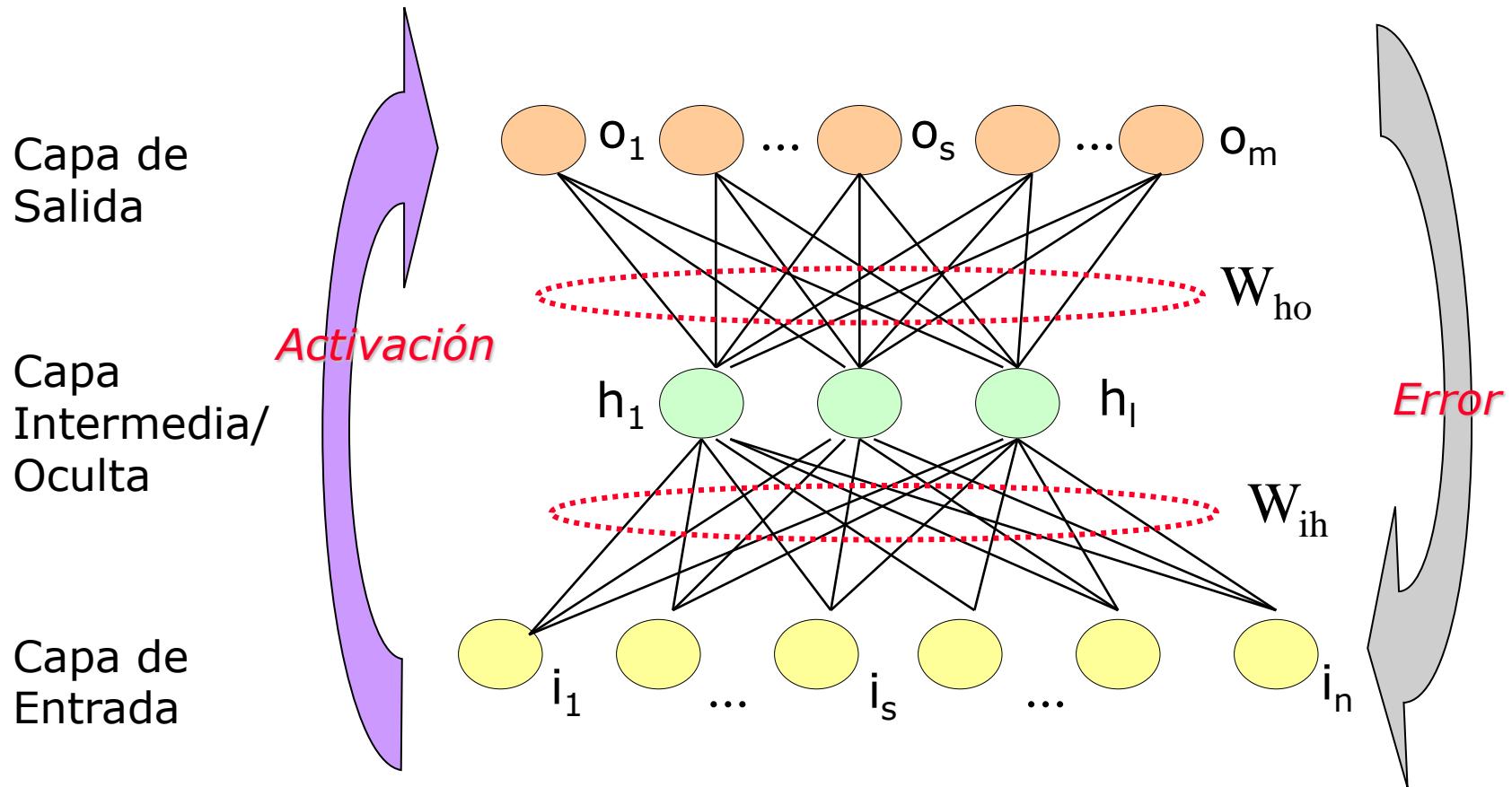
1.2 Arquitectura

- Empíricamente se ha comprobado que el número de elementos en las capas ocultas debe de ser

$$\frac{M}{2N} < n < \frac{2M}{N}$$

- Donde
 - M es el número de ejemplos del conjunto de entrenamiento
 - N es el número de neuronas de la capa de entrada
 - n es el número de neuronas en la capa intermedia

1.3 Procesamiento



1.3 Procesamiento



Flujo de datos Forward/Ejecución

- Propaga la entrada de la red a través de las capas hasta obtener la salida aplicando la función de transferencia
- El modelo es continuo y no-lineal debido a la función de transferencia utilizada:

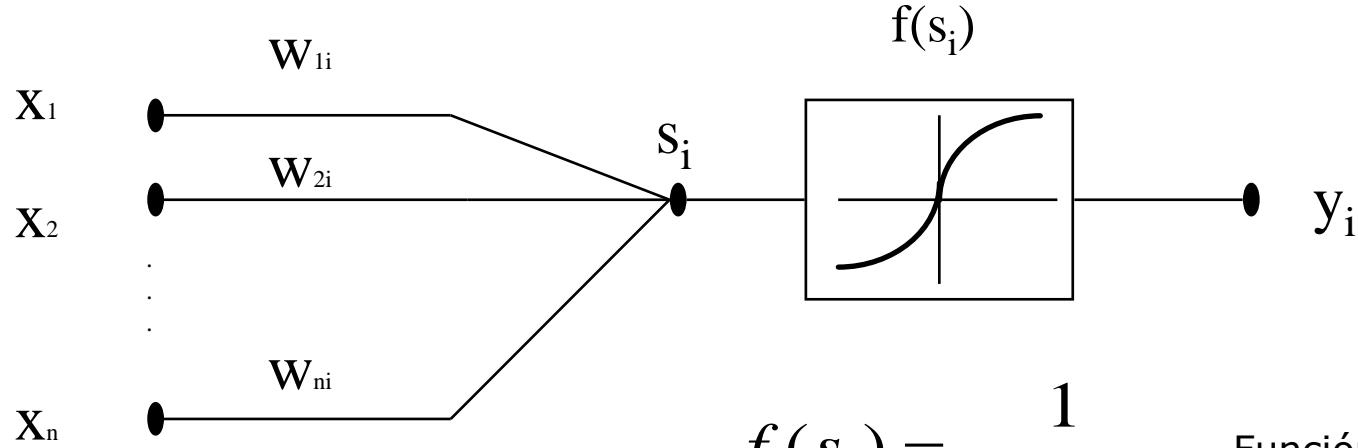
$$R^N \xrightarrow{f} R^M$$

- Sigmoidea/Tangente hiperbólica u otras no lineales.
- Salida en el intervalo $(0, +1)$ o $(-1, +1)$ (o no: ReLU)
- Continua y derivable en todos los puntos (necesario para poder usar la Regla Delta Generalizada)
- Cuanto mayor sea la pendiente (el gradiente de la función)
 - Más rápidamente la neurona alcanzará valores próximos a 0 o 1
 - Más difícil será cambiar su respuesta si está equivocada

1.3 Procesamiento



Suma de entradas $S_i = \sum_{j=1}^n w_{ij}x_j + b$



$$f(s_i) = \frac{1}{1 + e^{-s_i}} \quad \text{Función Sísmoidea}$$

$$f(s_i) = \frac{1 - e^{-s_i}}{1 + e^{-s_i}} \quad \text{Función Tangente Hiperbólica}$$



1.3 Procesamiento

- Propagación de la entrada

1. Se presenta el p -ésimo vector $\overrightarrow{x^p}$ a la capa de entrada que lo propaga hacia la capa oculta. La salida i de cada neurona de la **capa de entrada** es el mismo valor que su entrada:

$$i_k^p = x_k^p \rightarrow k = 1, 2, \dots, n$$

2. Cada neurona de la **capa oculta** recibe el patrón completo desde la capa de entrada. Su salida h se calcula aplicando la Función de Transferencia F :

$$h_k^p = f(S) = f\left(\sum_{j=1}^n w_{jk} i_j^p\right) \rightarrow k = 1, 2, \dots, l$$

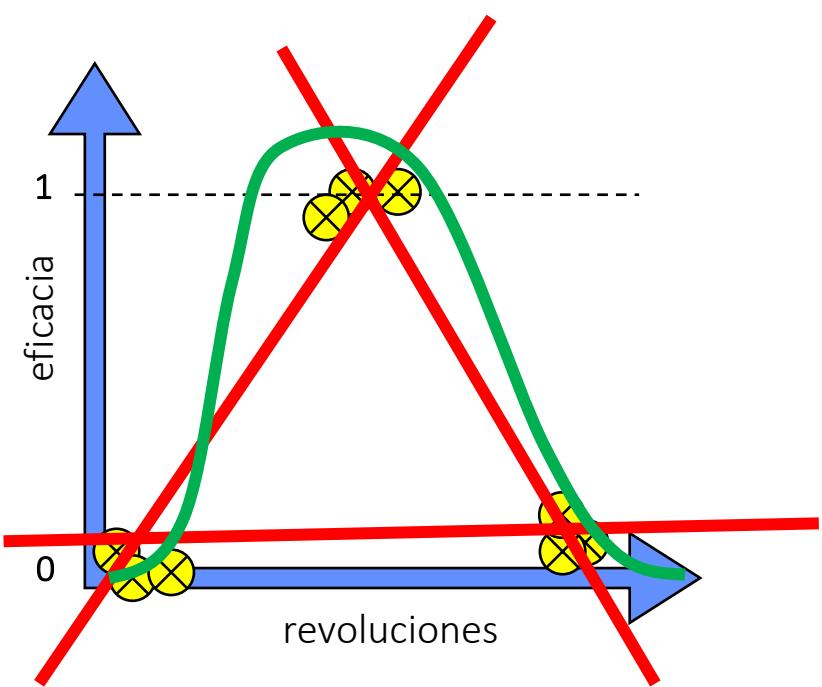
3. Cada neurona de la **capa de salida** recibe la salida completa de la capa oculta y la procesa aplicando F , proporcionando la respuesta de la red o :

$$o_k^p = f(S) = f\left(\sum_{j=1}^l w_{jk} h_j^p\right) \rightarrow k = 1, 2, \dots, m$$



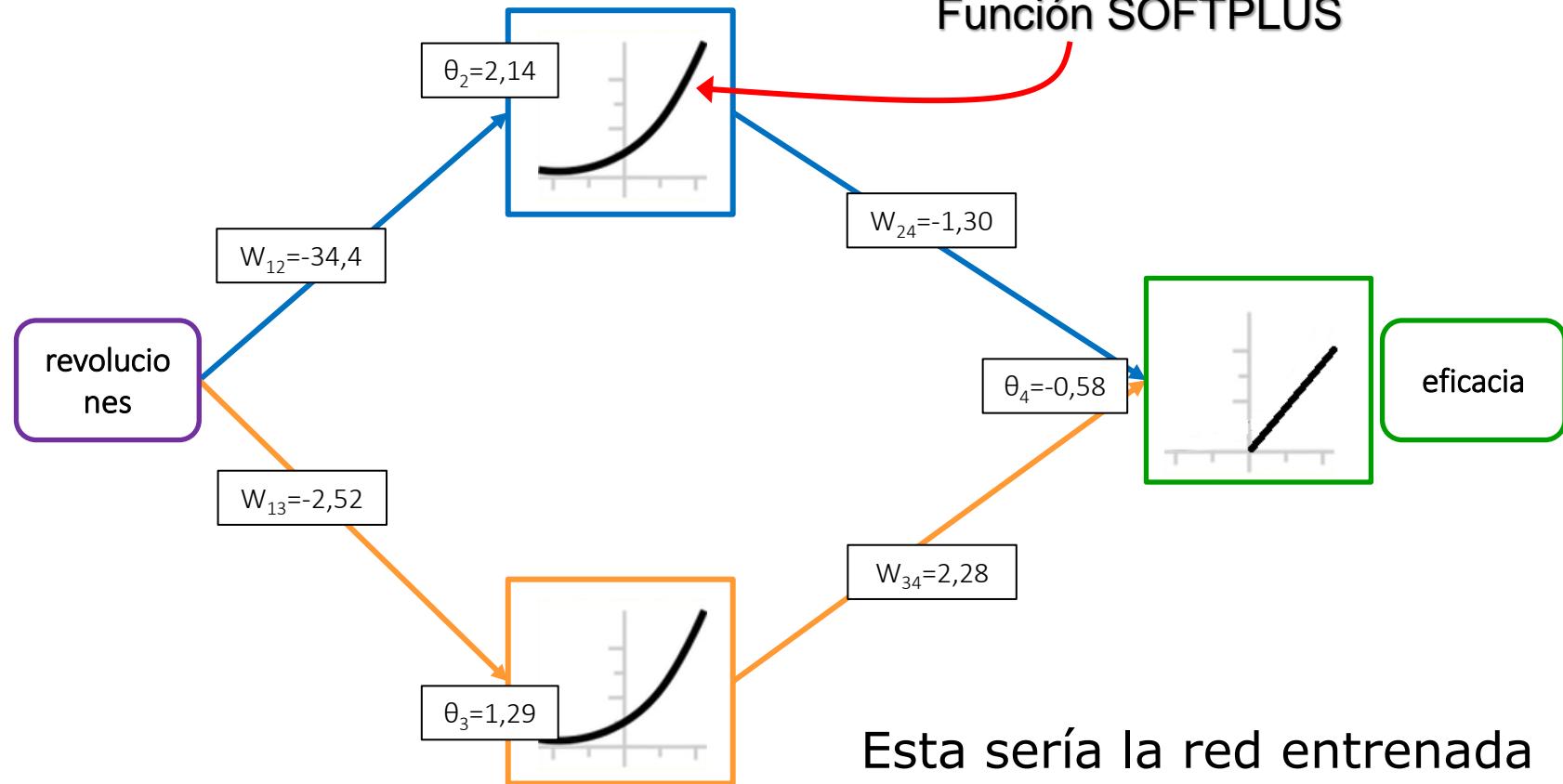
1.3 Procesamiento

- Eficacia del motor (0...1)vs Revoluciones (%): Problema de predicción/regresión

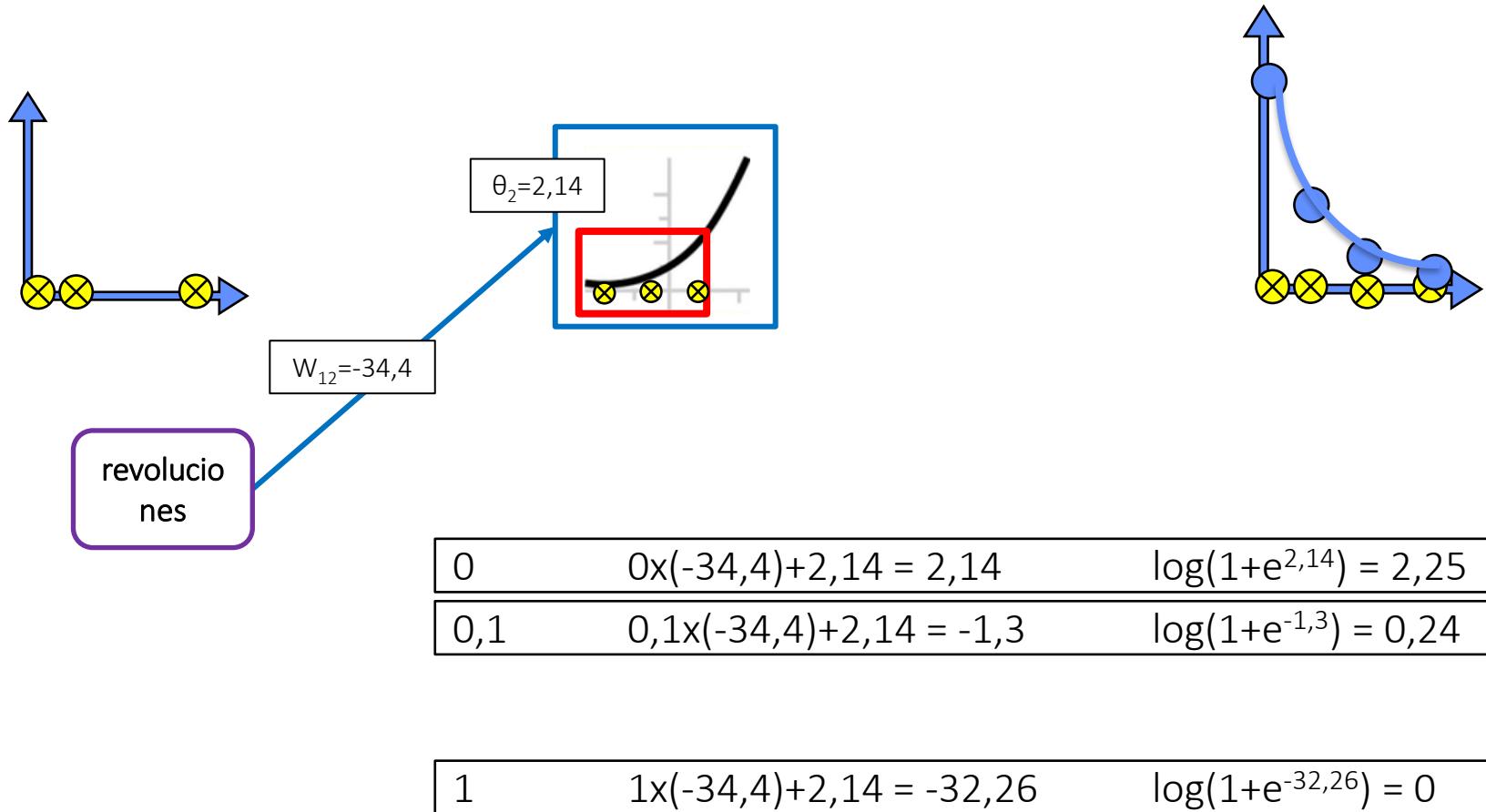


No hay forma de ajustar una recta que prediga todos los valores. Ha de ser una curva

1.3 Procesamiento



1.3 Procesamiento

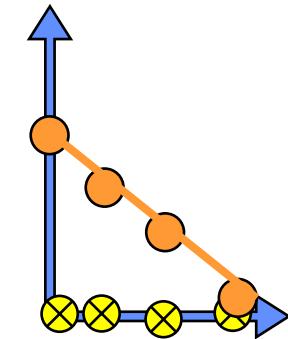
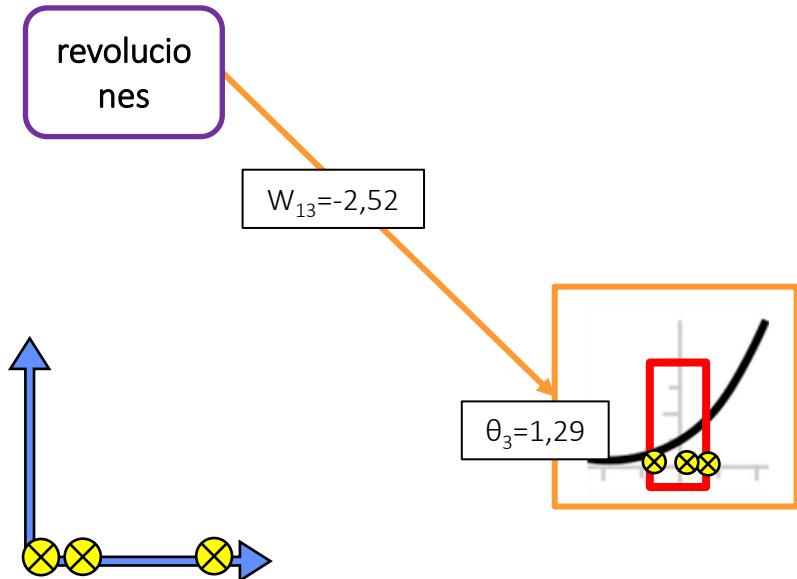


1.3 Procesamiento

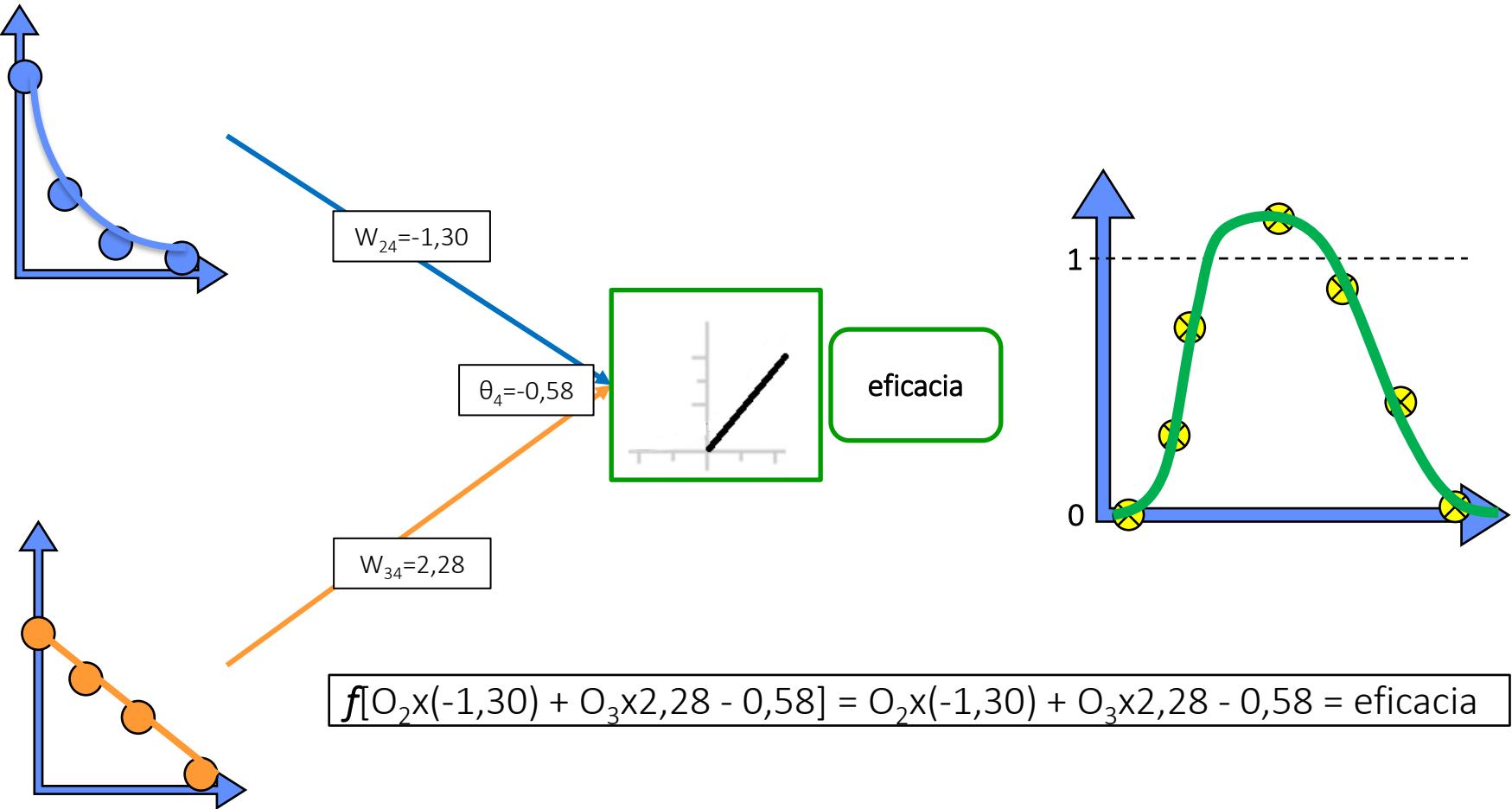


0	$0 \times (-2,52) + 1,29 = 1,29$	$\log(1+e^{1,29}) = 1,53$
0,1	$0,1 \times (-2,52) + 1,29 = 1,04$	$\log(1+e^{1,04}) = 0,24$

1	$1 \times (-2,52) + 1,29 = -1,23$	$\log(1+e^{-1,23}) = 0,11$
---	-----------------------------------	----------------------------



1.3 Procesamiento



1.3 Procesamiento





1.4 ¿Podemos resolver cualquier problema?

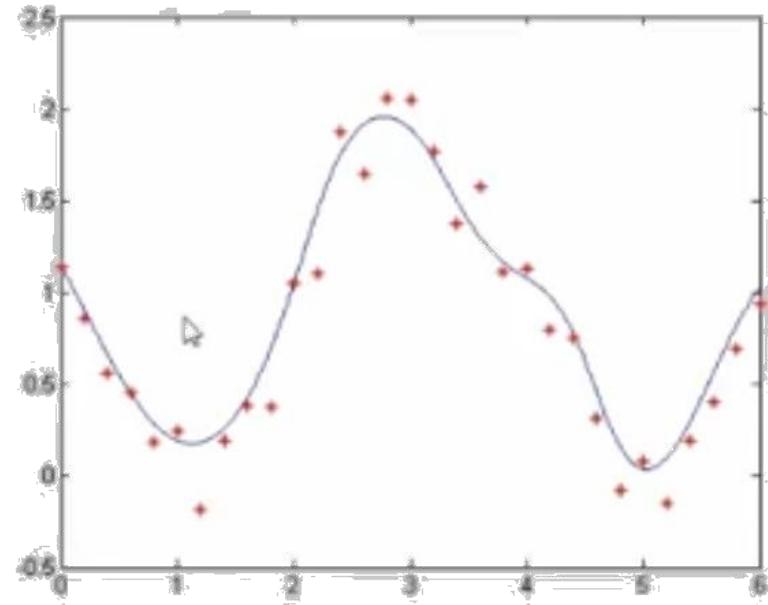
■ Teorema de Aproximación Universal

- *Cualquier función continua $R^N \xrightarrow{f} R^M$ definida sobre un conjunto compacto de datos puede aproximarse con precisión arbitraria mediante una red neuronal de una sola capa oculta.*
- Formulado en 1989 por George Cybenko para funciones de activación sigmoidales y demostrado por Kurt Hornik en 1991 para todas las funciones de activación.
- Una sola capa modeliza una proyección no lineal entre el espacio de entradas y el de salidas siempre y cuando las funciones de activación de las neuronas ocultas sean no lineales.
- La última capa puede ser lineal, ya que solo combina las anteriores.



1.4 ¿Podemos resolver cualquier problema?

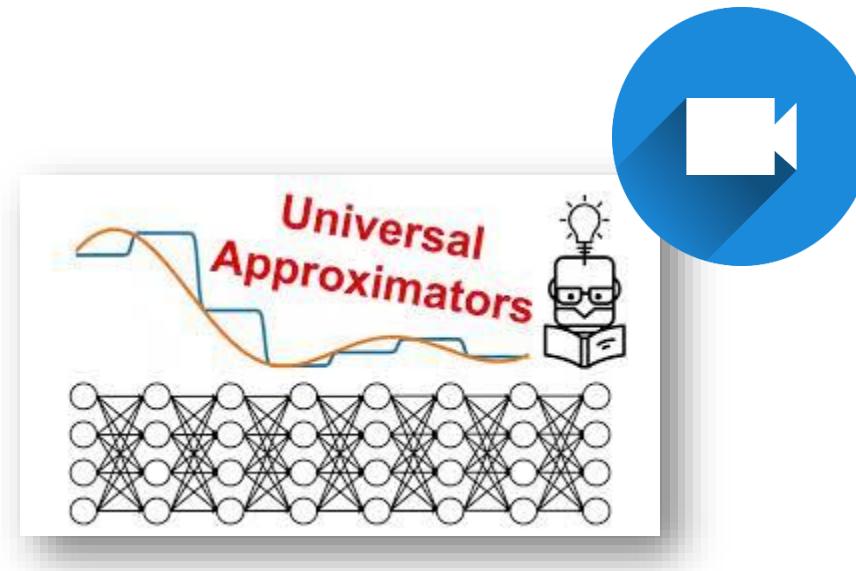
- El teorema establece que las redes multicapa no añaden capacidad a menos que la función de activación de las capas sea no lineal.
- Si aceptamos que la mayoría de las clases de problemas pueden reducirse a funciones, una red neuronal puede, en teoría, resolver cualquier problema pero
 - la capa puede ser demasiado grande y no aprender ni generalizar correctamente.
 - si la función da saltos o tiene grandes lagunas, no podremos aproximarla.



1.4 ¿Podemos resolver cualquier problema?



Combinando lo visto sobre procesamiento
con el Teorema de Aproximación Universal...





Flujo de datos Backward/Aprendizaje

- Propaga el error hacia las capas interiores ajustando a la vez los pesos de las conexiones hasta que la salida de la red sea **lo más próxima posible** a la salida deseada
- Aplica el algoritmo de aprendizaje a toda la red produciendo el ajuste de **todos los pesos** a la vez
- Ley de Aprendizaje:
 - **Regla Delta Generalizada o Backpropagation**
 - Técnica que usa el descenso de gradiente consistente en la generalización a varias capas de la Regla Delta del modelo ADALINE, que fue el primero en utilizarla
 - Con capas ocultas, la Regla Delta no sirve para el entrenamiento
 - Se conoce la salida deseada para cada patrón en la última capa de la estructura)
 - **Pero no se conoce** la salida deseada para las neuronas de las capas ocultas.



AVANZADO

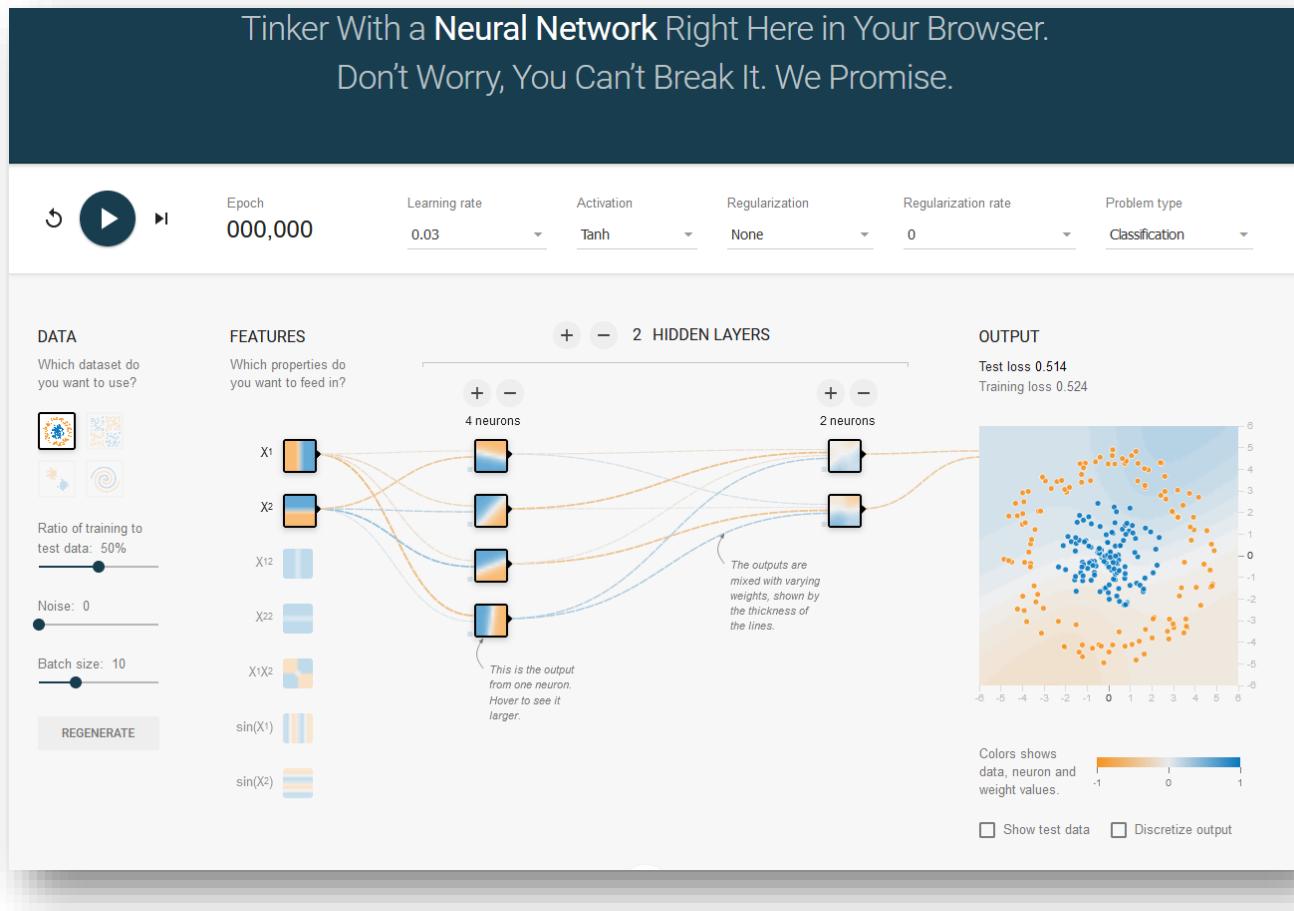
Jugando con Redes Neuronales - Parte 2.5 | DotCSV



1.5 Aprendizaje



<https://playground.tensorflow.org/>





- 1. El Perceptrón Multicapa**
 - 1. Historia**
 - 2. Arquitectura**
 - 3. Procesamiento**
 - 4. ¿Podemos resolver cualquier problema?**
 - 5. Aprendizaje**
- 2. Regla Delta Generalizada**
 - 1. Definición del error**
 - 2. Ajuste de los pesos: backpropagation**
 - 3. Entrenamiento**
 - 4. Problemas con el algoritmo**
- 3. Redes de Función de Base Radial**
 - 1. Arquitectura**
 - 2. Procesamiento**
 - 3. Aprendizaje**



2. Regla Delta Generalizada

- ¿Qué es la **Regla Delta Generalizada** o **Backpropagation**?
Método de obtención del vector gradiente en Redes Neuronales de múltiples capas y funciones de transferencia no lineales y diferenciables para poder aplicar a cada capa el descenso de gradiente.
- Utiliza la superficie de error asociada a la red para buscar el estado estable de mínimo error descendiendo por el gradiente de esa superficie
 - Propaga el error cometido por la red hacia las capas ocultas para usarlo en la actualización de sus matrices de pesos
 - En cada capa
 - usa el valor del error para calcular el vector gradiente del error respecto a los pesos de cada neurona
 - modifica los pesos proporcionalmente a lo que hemos descendido por el vector gradiente de la función de error



2.1 Definición del error

- Conjunto de entrenamiento

$$C = \{(\overrightarrow{X^p}, \overrightarrow{Y^p}) \mid \overrightarrow{X^p} \in R^N, \overrightarrow{Y^p} \in R^M, p = 1, 2, \dots, c\}$$

- Para el vector de entrada $\overrightarrow{x^p}$ se compara la salida real obtenida con la esperada calculando el **Error Cuadrático** para todas las neuronas de la capa de salida:

$$E^p = \frac{1}{2} \sum_{k=1}^m (d_k^p - y_k^p)^2$$

El factor $1/2$ se usa por conveniencia en el cálculo de la derivada en operaciones posteriores

- Después se obtiene la media de los errores o **Error Cuadrático Medio MSE (\bar{E})** para todos los patrones del conjunto de entrenamiento

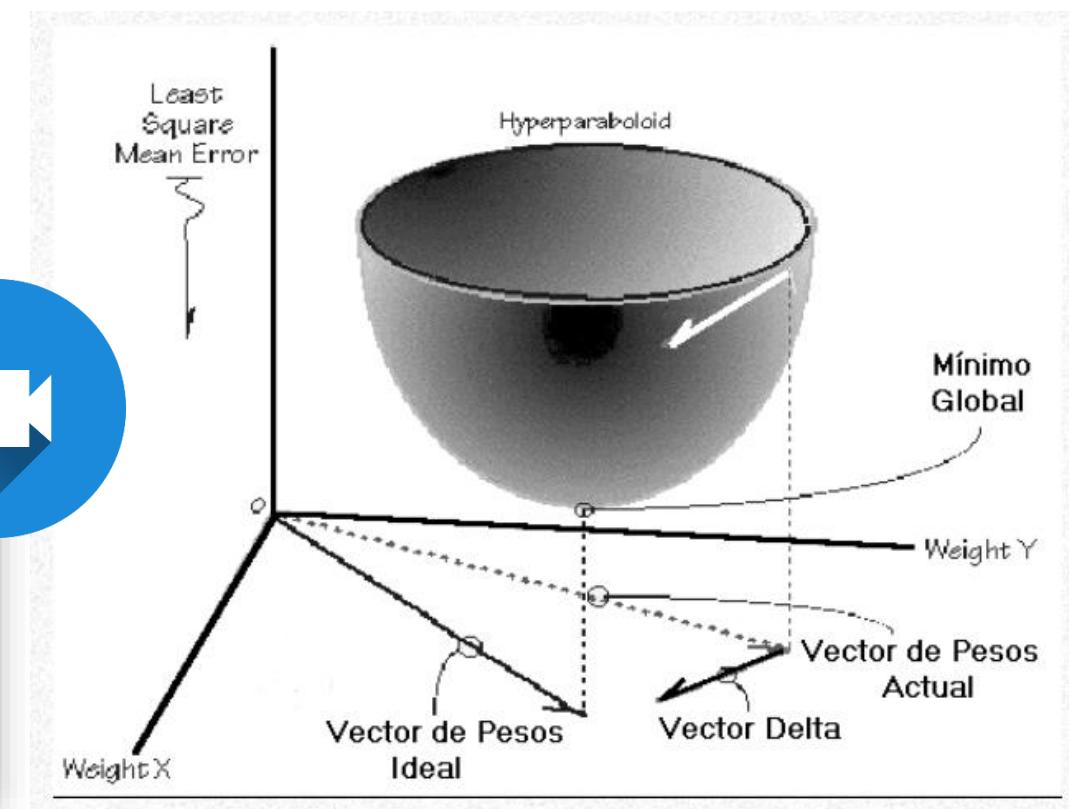
$$\bar{E} = \frac{1}{2c} \sum_{p=1}^c \sum_{k=1}^m (d_k^p - y_k^p)^2$$



2.1 Definición del error

- Donde

- Y_k es la salida real en t de la neurona k
- d_k es la salida esperada de la neurona k
- m son las neuronas de la capa de salida
- c son los casos del conjunto de entrenamiento
- p es p-ésimo patrón de entrada



2.2 Ajuste de los pesos: Backpropagation



- Corrección de las matrices de pesos propagando el error desde la capa de salida hasta la de entrada en dos fases
- Basado en minimización del **MSE** (similar a la Regla Delta)
 - Los pesos de la matriz se modifican...

$$\vec{W}_{t+1} = \vec{W}_t + \Delta \vec{W}_t$$

- proporcionalmente en dirección opuesta al gradiente $\nabla E(\vec{W}_t)$

$$\Delta \vec{W}_t = -\eta \nabla E(\vec{W}_t)$$

— η es el factor que marca esa proporcionalidad

- y se obtiene la expresión:

$$\vec{W}_{t+1} = \vec{W}_t - \eta \nabla E(\vec{W}_t)$$



2.2 Ajuste de los pesos: Backpropagation

- Como E depende de \vec{W}_t a través del producto escalar $S = \sum_{i=1}^n w_i x_i$ podemos aplicar la regla de la cadena a la definición de gradiente

$$\nabla E^p(\vec{W}_t) = \frac{\partial E^p}{\partial \vec{W}_t} = \frac{\partial E^p}{\partial S^p} \frac{\partial S^p}{\partial \vec{W}_t} \quad (1)$$

2.2 Ajuste de los pesos: Backpropagation



Fase 1: *Actualización de la matriz de pesos de la capa de salida* (pesos que unen la capa oculta con la capa de salida)

- Para la neurona O_o de la capa de salida y la neurona h_h de la capa oculta la *expresión (1)* se convierte en

$$\frac{\partial E^p}{\partial w_{ho}^t} = \frac{\partial E^p}{\partial S_o^p} \frac{\partial S_o^p}{\partial w_{ho}^t}$$

- Llamamos *sensibilidad* de la neurona de salida o -ésima al patrón p , que mide como el error depende de esa neurona,

$$\delta_o^p = - \frac{\partial E^p}{\partial S_o^p}$$



2.2 Ajuste de los pesos: Backpropagation

- $\left[\frac{\partial S_o^p}{\partial w_{ho}^t} \right] S_o^p$ es la entrada a la neurona de la capa de salida.
 - El peso W_{ho} solo existe en la neurona h_h por lo que las demás derivadas parciales son 0 y el resultado es justamente la salida de la neurona oculta de la que parte la conexión

$$\frac{\partial S_o^p}{\partial w_{ho}^t} = \frac{\partial}{\partial w_{ho}^t} \left(\sum_{j=1}^l w_{jo} h_j^p \right) = f'(S) = h_h^p$$

- $\left[\frac{\partial E^p}{\partial S_o^p} \right]$ Es la sensibilidad, un término no lineal:

$$\delta_o^p = -\frac{\partial E^p}{\partial S_o^p} = -\frac{\partial}{\partial S_o^p} \frac{1}{2} (d_o^p - f_o(S_o^p))^2 = (d_o^p - o_o^p) f_o'(S_o^p)$$



2.2 Ajuste de los pesos: Backpropagation

- Con lo que ΔW_{ho} sería

$$\Delta W_{t_{ho}} = -\eta \nabla E^p(\overrightarrow{W_{t_{ho}}}) = -\eta \frac{\partial E^p}{\partial S_o^p} \frac{\partial S_o^p}{\partial w_{ho}^t} = \eta \delta_o^p h_h^p$$

- La ley para modificar el peso que une una neurona h de la capa oculta con una neurona o de la capa de salida es:

$$w_{ho}^{t+1} = w_{ho}^t + \Delta W_{t_{ho}}$$

$$w_{ho}^{t+1} = w_{ho}^t + \eta \delta_o^p h_h^p$$

2.2 Ajuste de los pesos: Backpropagation



- Donde
 - h_h^p es la salida (activación) de la neurona de la capa oculta que a su vez es entrada a la neurona de la capa de salida
 - δ_o^p es la sensibilidad de la neurona de la capa de salida
 - η (*coeficiente de aprendizaje*)
 - Controla cuánto se desplazan los pesos en dirección negativa al gradiente
 - Influye en la velocidad de convergencia del algoritmo y en su estabilidad (problema de mínimos locales)
 - **Valores altos de η** favorecen a una convergencia más rápida en las proximidades del mínimo pero existe el riesgo de saltar y oscilar alrededor de él
 - **Valores bajos de η** hacen la convergencia más lenta pero evitan el problema de saltar el mínimo

2.2 Ajuste de los pesos: Backpropagation



Fase 2: *Actualización de la matriz de pesos de la capa oculta* (pesos que unen la capa de entrada con la capa oculta)

- En las neuronas de las capas ocultas no podemos hacer lo mismo
 - *No se conoce de antemano la salida correcta de esta capa*
 - Intuitivamente,
 - Cada neurona h es “responsable” del error de las neuronas a las que envía su salida en la medida que marca el peso de la conexión entre ellas
 - Cada neurona de salida o distribuye hacia atrás su valor de δ a todas las neuronas ocultas que se conectan a ella.
 - Idea: ir hacia atrás, calculando la contribución al error de la red de una neurona de la capa oculta a partir del error de las neuronas de la capa de salida
 - La matriz de pesos W_{ih} es actualizada con una fórmula similar a la obtenida para la matriz de pesos de salida W_{ho}

2.2 Ajuste de los pesos: Backpropagation



- Para la neurona i , de la capa de entrada y la neurona h_h de la capa oculta la expresión (1) se convierte en

$$\frac{\partial E^p}{\partial w_{ih}^t} = \frac{\partial E^p}{\partial S_h^p} \frac{\partial S_h^p}{\partial w_{ih}^t}$$

- Llamamos *sensibilidad* de la neurona oculta h -ésima al patrón p , que mide como el error depende de esa neurona, a

$$\delta_h^p = - \frac{\partial E^p}{\partial S_h^p}$$



2.2 Ajuste de los pesos: Backpropagation

- $\left[\frac{\partial E^p}{\partial S_h^p} \right]$ Al no ser una neurona de salida, su sensibilidad está en función del error que cometan todas las neuronas de salida (que reciben como entrada la salida de h_h)
 - La salida de cada neurona de la capa oculta influye en TODAS las salidas de la red por lo que la sensibilidad tiene más términos

$$\delta_h^p = -\frac{\partial E^p}{\partial S_h^p} = f_h'(S_h^p) \sum_{k=1}^m W_{hk} \delta_k^p$$

- $\left[\frac{\partial S_h^p}{\partial w_{ih}^t} \right]$ S_h^p es la entrada a la neurona de la capa oculta.
 - El peso W_{ih} solo influye en la neurona O_o por lo que las demás derivadas parciales son 0 y el resultado es justamente la salida de la neurona de la capa de entrada, que vimos que coincide con la entrada X_i^p

$$\frac{\partial S_h^p}{\partial w_{ih}^t} = f'(S) = x_i^p$$



2.2 Ajuste de los pesos: Backpropagation

- Con lo que ΔW_{ih} sería

$$\Delta W_{ih} = -\eta \nabla E^p(\vec{W}_t) = -\eta \frac{\partial E^p}{\partial S_h^p} \frac{\partial S_h^p}{\partial w_{ih}^t} = \eta \delta_h^p x_h^p$$

- La ley para modificar el peso que une una neurona *i* de la capa de entrada con una neurona *h* de la capa oculta es:

$$w_{ih}^{t+1} = w_{ih}^t + \Delta W_{ih}$$

$$w_{ih}^{t+1} = w_{ih}^t + \eta \delta_h^p x_i^p$$

La actualización de cada peso de la capa oculta depende de todos los errores de la capa de salida, por lo que el error se retropropaga hasta la capa de entrada

2.2 Ajuste de los pesos: Backpropagation



- Las ecuaciones correspondientes a las dos matrices son:

- Matriz de pesos oculta-salida

$$w_{ho}^{t+1} = w_{ho}^t + \eta \delta_o^p h_h^p$$

- Matriz de pesos entrada-oculta

$$w_{ih}^{t+1} = w_{ih}^t + \eta \delta_h^p x_i^p$$

- Estas dos expresiones constituyen la *REGLA DELTA*

2.2 Ajuste de los pesos: Backpropagation



- Aunque pueda parecer la misma formula de actualización de pesos, no es así, ya que
 - Los valores de η son distintos para cada matriz de pesos, y se ajustan por separado para dar los resultados óptimos.
 - Las funciones de activación pueden ser distintas en cada capa.
 - En la fórmula de ajuste de los pesos de la capa de salida, el valor que se utiliza como entrada (en el término de corrección) es la salida de las neuronas de la capa intermedia
 - En la fórmula de ajuste de los pesos de la capa oculta, el valor que se utiliza como entrada (en el término de corrección) es la entrada a la red
 - La sensibilidad de una neurona de la capa oculta es distinta de la sensibilidad de una neurona de la capa de salida.

2.2 Ajuste de los pesos: Backpropagation



Momento

- Mejora de la convergencia del algoritmo de backpropagation
 - Ayuda a mantener la dirección del descenso de gradiente al guardar la memoria de lo que pasó en el instante $t-1$
 - Inercia del camino recorrido descendiendo el gradiente, evita oscilaciones, mejora la convergencia
 - Propuesta por Rumelhart (1986)

$$\mu(w^t - w^{t-1})$$

- μ (*momento*): parámetro que controla la importancia del término de inercia
 - Valor alto < 1 (típicamente = 0.95)
 - Debe reducirse a medida que avanza el entrenamiento
 - Se calcula/aplica una vez que todo el conjunto de entrenamiento ha pasado por la red



2.2 Ajuste de los pesos: Backpropagation

- Las ecuaciones anteriores quedarían

- Matriz de pesos oculta-salida

$$w_{ho}^{t+1} = w_{ho}^t + \eta \delta_o^p h_h^p + \mu (w_{ho}^t - w_{ho}^{t-1})$$

- Matriz de pesos entrada-oculta

$$w_{ih}^{t+1} = w_{ih}^t + \eta \delta_h^p x_i^p + \mu (w_{ih}^t - w_{ih}^{t-1})$$

- Estas dos expresiones constituyen la *REGLA DELTA GENERALIZADA*

2.2 Ajuste de los pesos: Backpropagation



- Para las capas ocultas, si la Función de Activación de la neurona O es sigmoidea

$$f_o(S_o^p) = \frac{1}{1 + e^{-S_o^p}}$$

$$f'_o(S_o^p) = f_o(1 - f_o) = o_o^p(1 - o_o^p)$$

- Y por lo tanto

$$\delta = (d_o^p - y_o^p)y_o^p(1 - y_o^p)$$

- Con lo que la Regla Delta Generalizada quedaría

$$w_{ho}^{t+1} = w_{ho}^t + \eta(d_o^p - y_o^p)y_o^p(1 - y_o^p)h_h^p + \mu(w_{ho}^t - w_{ho}^{t-1})$$

2.2 Ajuste de los pesos: Backpropagation



- Para la capa de salida, si la Función de Activación de la neurona O es lineal

$$f_o(S_o^p) = S_o^p$$
$$f'_o(S_o^p) = 1$$

- Y por lo tanto

$$\delta = (d_o^p - y_o^p)$$

- Con lo que la Regla Delta Generalizada quedaría

$$w_{ho}^{t+1} = w_{ho}^t + \eta(d_o^p - y_o^p)h_h^p + \mu(w_{ho}^t - w_{ho}^{t-1})$$

Que coincide (excepto por el término del momento) con la Regla Delta de Widrow para el modelo ADALINE

2.2 Ajuste de los pesos: Backpropagation



AVANZADO

Descenso de Gradiente y Backpropagation | 3Blue1Brown

Neural Networks

Neural networks

3Blue1Brown



2.3 Entrenamiento



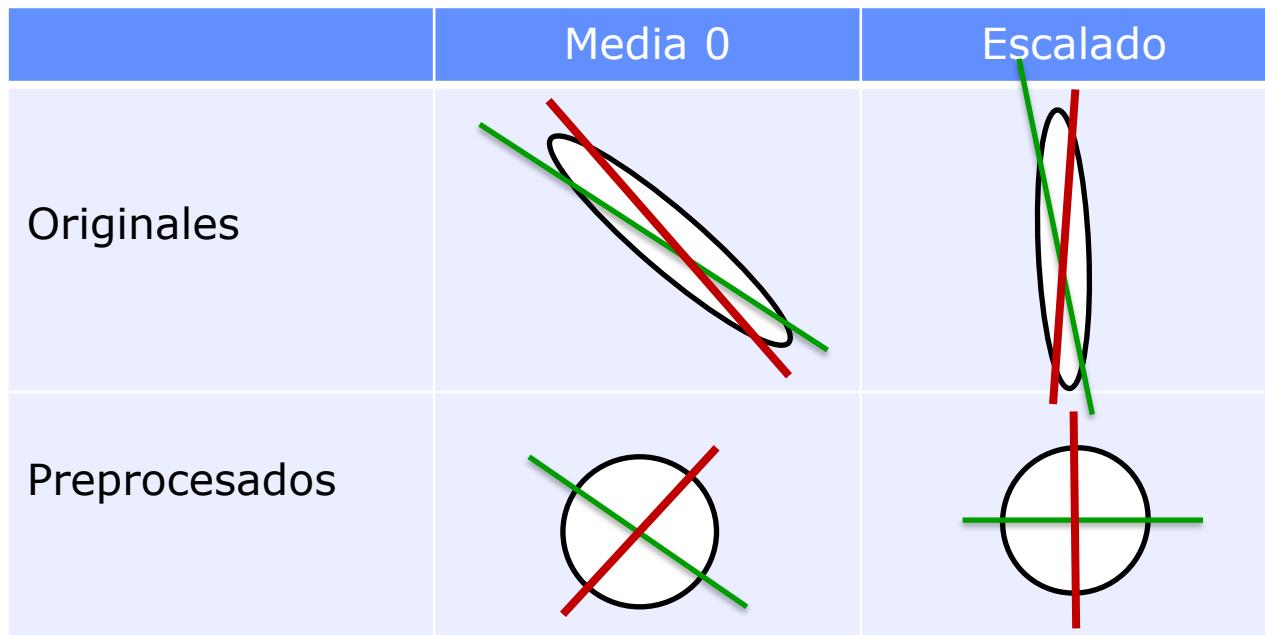
Preprocesamiento

- Las entradas se definen en función del tipo de problema
 - *Tipo de datos*: numéricos/simbólicos cuantitativos/cualitativos
 - *Carácter numérico*: real, entero
 - *Naturaleza* de los datos numéricos: discretos, continuos
 - *Rango de los datos*: acotados, no acotados
- Suele ser necesario un preprocesamiento de las entradas
 - Parametrización de variables cualitativas
 - Discretización de variables continuas
 - Normalización al intervalo (0,1) o similar (z-score)
 - Acotación del rango (escalado)
 - Eliminar correlaciones
 - Combinación de parámetros de entrada

2.3 Entrenamiento



- Normalización z-score
 - La **media** de cada variable de entrada en el conjunto de entrenamiento debería ser cercana a cero.
 - La **escala** de las variables de entrada debería ajustarse para que sus covarianzas sean similares.



2.3 Entrenamiento



Organización de los datos de entrenamiento

- El conjunto de entrenamiento determina lo que la red va a aprender. Su tamaño
 - Debe de cubrir todo el espacio estudiado, seleccionando los patrones que proporcionan mayor información para el entrenamiento de la red.
 - 2 ó 3 veces el número de pesos
 - Regla de Baum (1989): $N_{\text{patrones}} = N_{\text{pesos}} / \text{Error}_{\text{deseado}}$
 - La utilización de muchos datos hace excesivamente lento el entrenamiento
 - Hay que reservar datos para los conjuntos de
 - Prueba/Test: casos concretos y difíciles (aprox. 5% de los datos)
 - Validación (aprox. 20% del resto de datos)
- No entrenar a la red de forma consecutiva con grupos de vectores que proporcionen la misma salida

2.3 Entrenamiento

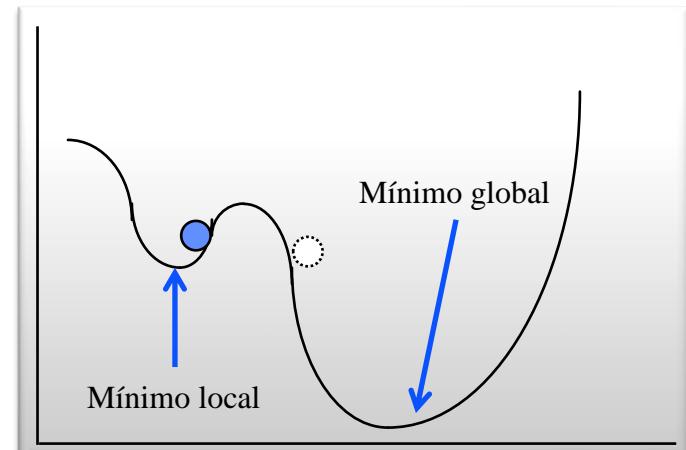


Criterio de parada

- Número de iteraciones prefijadas
- Cuando el error sobre el conjunto de entrenamiento se estabiliza por debajo de una cota prefijada
 - Cuando el error se estabiliza es porque los parámetros de la red no cambian apenas de una iteración a otra
 - Se ha alcanzado el mínimo en la función de error

$$\frac{\partial E}{\partial w} \approx 0$$

- Esto también ocurre en los mínimos locales (trampa)
 - El método detecta que cualquier pequeño cambio en los pesos incrementa el error
 - No es capaz de determinar en qué dirección deben moverse los pesos para que el error vuelva a decrecer.



2.3 Entrenamiento



Iteración

1. Inicializar pesos con valores aleatorios
Distribución normal $N(0,1)$ con media 0 y desviación 1.
2. Presentar el vector de entrada X^p
3. Calcular la salida de las neuronas de la capa oculta
4. Calcular la salida de la red (neuronas de la capa de salida)
5. Repetir desde 2 para todo el conjunto de entrenamiento
6. Calcular el Error
7. Aplicar la Regla Delta Generalizada
 1. Calcular la sensibilidad de las neuronas de la capa de salida
 2. Calcular la sensibilidad de las neuronas de la capa oculta
 3. Actualizar los pesos de la matriz de la capa oculta
 4. Actualizar los pesos de la matriz de la capa de salida
8. Repetir desde 2 si no se da la condición de parada

2.3 Entrenamiento

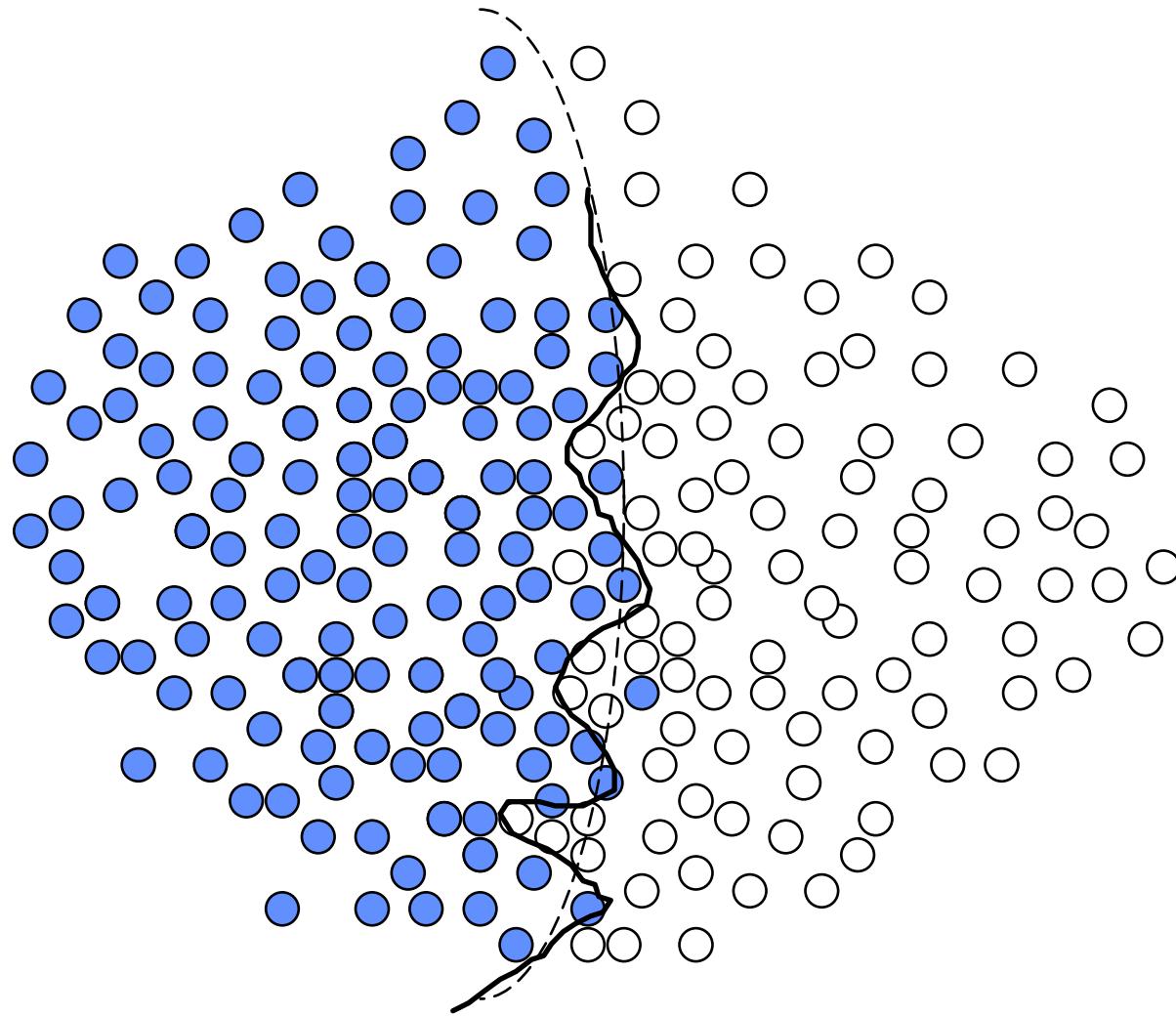


Posibles situaciones al concluir el entrenamiento

- **Entrenamiento correcto:** La red crea un modelo a partir de los datos de entrenamiento (**generaliza**)
 - La generalización garantiza
 - Que la respuesta para cualquier elemento del conjunto de entrenamiento será correcta
 - Que puede resolver un patrón que no haya visto antes pero **NO** garantiza que el error sea mínimo
 - Para generalizar
 - Obvia datos irrelevantes
 - Resalta similitudes generales (características de las muestras)



2.3 Entrenamiento



2.3 Entrenamiento



- **Subentrenamiento:** Un entrenamiento insuficiente de la red hace que ésta no generalice
- **Sobre-entrenamiento:** Un exceso de entrenamiento hace que la red **memorice** exclusivamente los vectores de entrenamiento y tampoco generalice (*modeliza el ruido*)
 - También por un exceso de neuronas en la capa oculta
 - Para asegurar la generalización se puede
 - Aumentar el conjunto de entrenamiento (Baum 1989)
 - Modificar la pendiente de la sigmoide
 - Introducir ruido en los patrones
 - Suprimir neuronas
 - reduciendo entradas (análisis de los datos)
 - de la capa oculta
 - Emplear **validación cruzada** (*Método Jack Knife*)



2.3 Entrenamiento



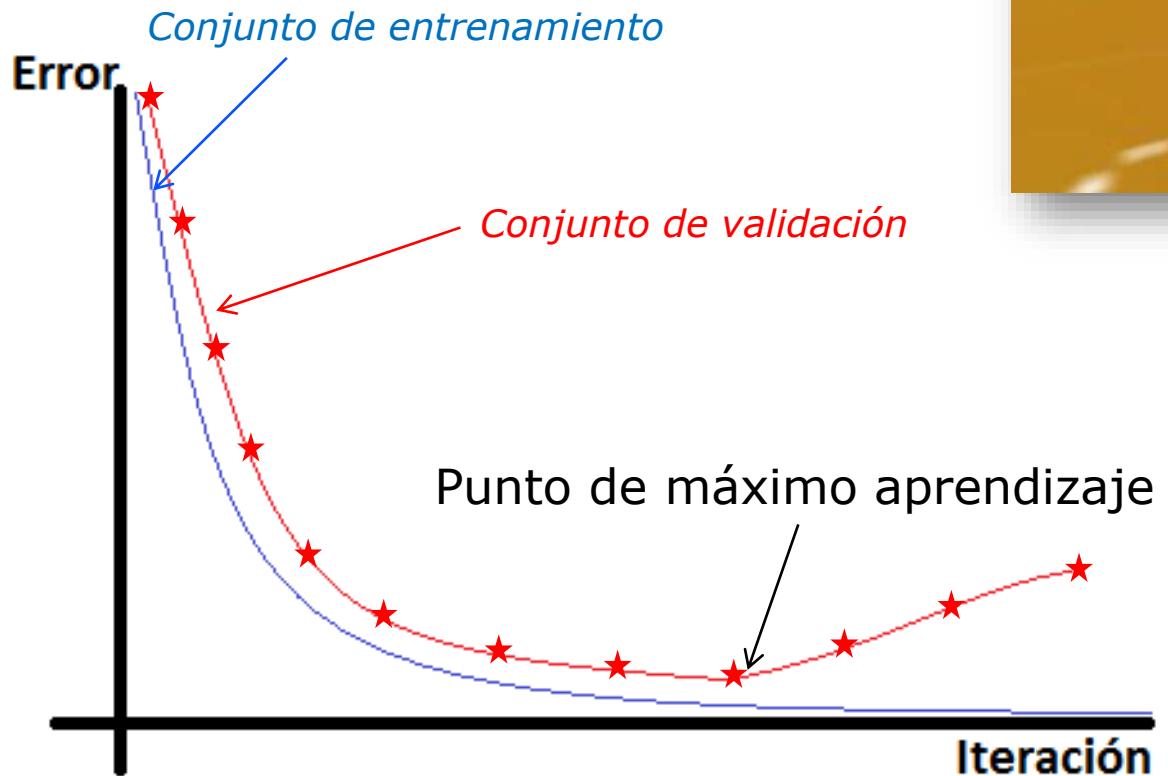
Validación cruzada simple (Cover, 1969)

- Método estadístico de estimación del error de generalización basado en usar un conjunto de datos de validación elegido aleatoriamente del dataset (20% de los datos disponibles) que no se usarán en el entrenamiento.
 - Mide la capacidad de la red de responder ante patrones que no han sido utilizados en el entrenamiento
- Cada cierto número de ciclos con el **conjunto de entrenamiento**, se presentan a la red los **patrones de validación**, evaluando el error cometido por la red
 - $E_{\text{validación}}$ sigue disminuyendo → continuar
 - $E_{\text{validación}}$ aumenta respecto $E_{\text{entrenamiento}}$ → parar

2.3 Entrenamiento



- Ejemplo de proceso de validación



2.3 Entrenamiento



Validación cruzada de k iteraciones (*K-fold cross-validation*)

- Los datos de muestra se dividen en K subconjuntos. Uno de los subconjuntos se utiliza como datos de validación y el resto ($K-1$) como datos de entrenamiento.
- El proceso se repite k iteraciones, con cada uno de los subconjuntos de datos de validación.
- Se generan k valores de error. El promedio es el valor final.

$$\bar{E} = \frac{1}{k} \sum_{i=1}^k E_i$$



2.4 Problemas con el algoritmo

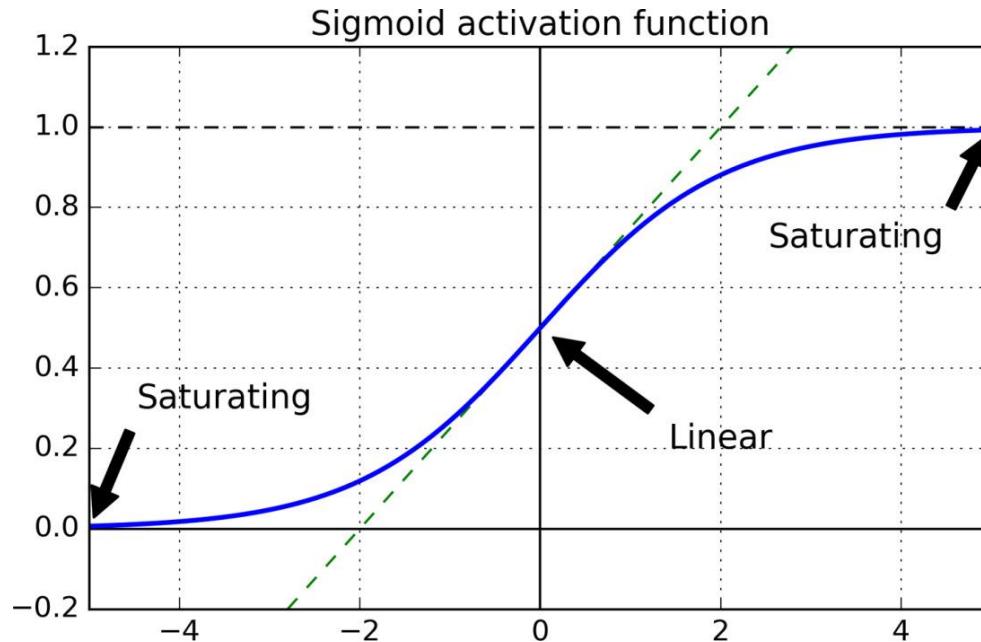


- Tiempo de computación necesario para el entrenamiento.
 - Depende del tamaño de la red y del número de pesos.
- Existencia de mínimos locales en la función.
 - Depende del problema.
- Conjunto de entrenamiento.
 - Depende del tamaño de la red y del problema.
 - Debe de cubrir todo el espacio muestral.
 - Su tamaño aumenta de forma exponencial al número de neuronas
- Tamaño de la red
 - Depende del problema
 - Solución al problema de los mínimos locales pero el aumento de capas internas hace que crezcan los problemas anteriores
 - No existen criterios generales para establecer el número de neuronas, de capas ni el valor de los parámetros
- Sobre-entrenamiento

2.4 Problemas con el algoritmo



- Parálisis de la red
 - Depende de la función de activación
 - Para el caso de la sigmoide/tngch,
 - si la entrada a una neurona toma valores muy altos → saturación
 - En la zona donde la derivada de la función de activación es cercana a 0, no hay gradiente del error que propagar hacia atrás
 - Cuanto más tiempo de entrenamiento transcurre, más lento es el aprendizaje





2.4 Problemas con el algoritmo

- Posibles soluciones
 - Variaciones al modelo básico
 - Conexión entre capas no adyacentes:
 - Modelo de red feedforward
 - Conecta directamente algunas entradas con la salida
 - Redes recurrentes:
 - Modelo de red feedforward
 - especialmente utilizadas en predicción de series temporales
 - Técnicas optimizadoras del entrenamiento
 - Otras técnicas de entrenamiento

2.4 Problemas con el algoritmo



Técnicas optimizadoras del entrenamiento

- Extracción de características (datos de entrada)
 - Obtención de nueva información a partir de expertos
 - Elección de datos significativos o confusos, especialmente en la frontera de las regiones del espacio
 - Objetivos:
 - Disminuir el número de entradas
 - Facilita la separación de clases
 - Reduce el número de datos sin perder precisión
 - Acelera el proceso de entrenamiento
 - Metodos
 - Class-Mean
 - Karhunen-Loève

2.4 Problemas con el algoritmo



- Entrenamiento Adaptativo (Adaptive Learning Rate)
 - Aplicable a datos no estacionarios, cuando su variación es razonablemente lenta
 - Objetivos:
 - Ajuste a datos no estacionarios
 - Si $\overline{\Delta E}$ mantiene el mismo signo (positivo o negativo) a lo largo de varios ciclos, aumentar el valor de η
 - Si $\overline{\Delta E}$ va alternando (positivo \leftrightarrow negativo) a lo largo de varios ciclos, disminuir el valor de η
- Simulated Annealing:
 - Técnica basada en la incorporación de ruido aleatorio al sistema
 - Evita posibles mínimos locales de la superficie de error

2.4 Problemas con el algoritmo



Otras técnicas de entrenamiento

- Algoritmos genéticos:
 - Optimización de la función $1/E$ mediante generación evolutiva de poblaciones en las que se modifica, mediante operadores genéticos:
 - *La matriz de pesos*: cada individuo parte de una matriz aleatoria y aquellas redes con menor error se reproducen en la siguiente generación de la población
 - *La estructura del PCM*: cada individuo tiene un número de neuronas, capas y conexiones aleatorias y aquellos con menor error se reproducen



- 1. El Perceptrón Multicapa**
 1. Historia
 2. Arquitectura
 3. Procesamiento
 4. ¿Podemos resolver cualquier problema?
 5. Aprendizaje
- 2. Regla Delta Generalizada**
 1. Definición del error
 2. Ajuste de los pesos: backpropagation
 3. Entrenamiento
 4. Problemas con el algoritmo
- 3. Redes de Función de Base Radial**
 1. Arquitectura
 2. Procesamiento
 3. Aprendizaje

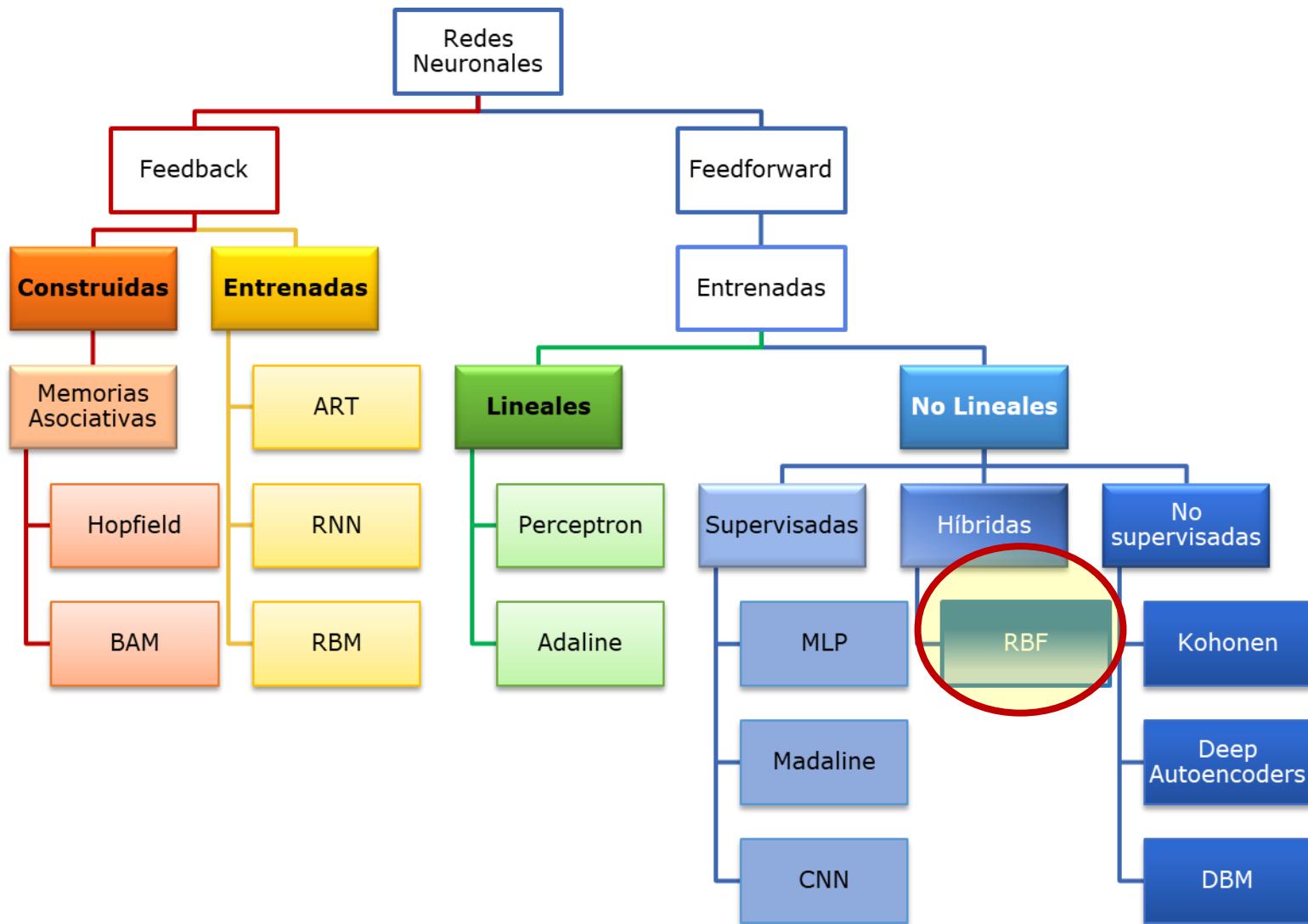


3. Redes de Función de Base Radial

- [1989] Moody y Poggio: Modelo híbrido que incorpora aprendizaje supervisado y no supervisado
 - Redes multicapa con conexiones hacia adelante
 - Modelo no lineal
 - Feedforward
 - Entrenada
 - Única capa oculta
- Se han aplicado a gran variedad de problemas
 - Análisis de series temporales
 - Procesamiento de imágenes
 - Reconocimiento automático del habla
 - Diagnósticos médicos, etc



3. Redes de Función de Base Radial

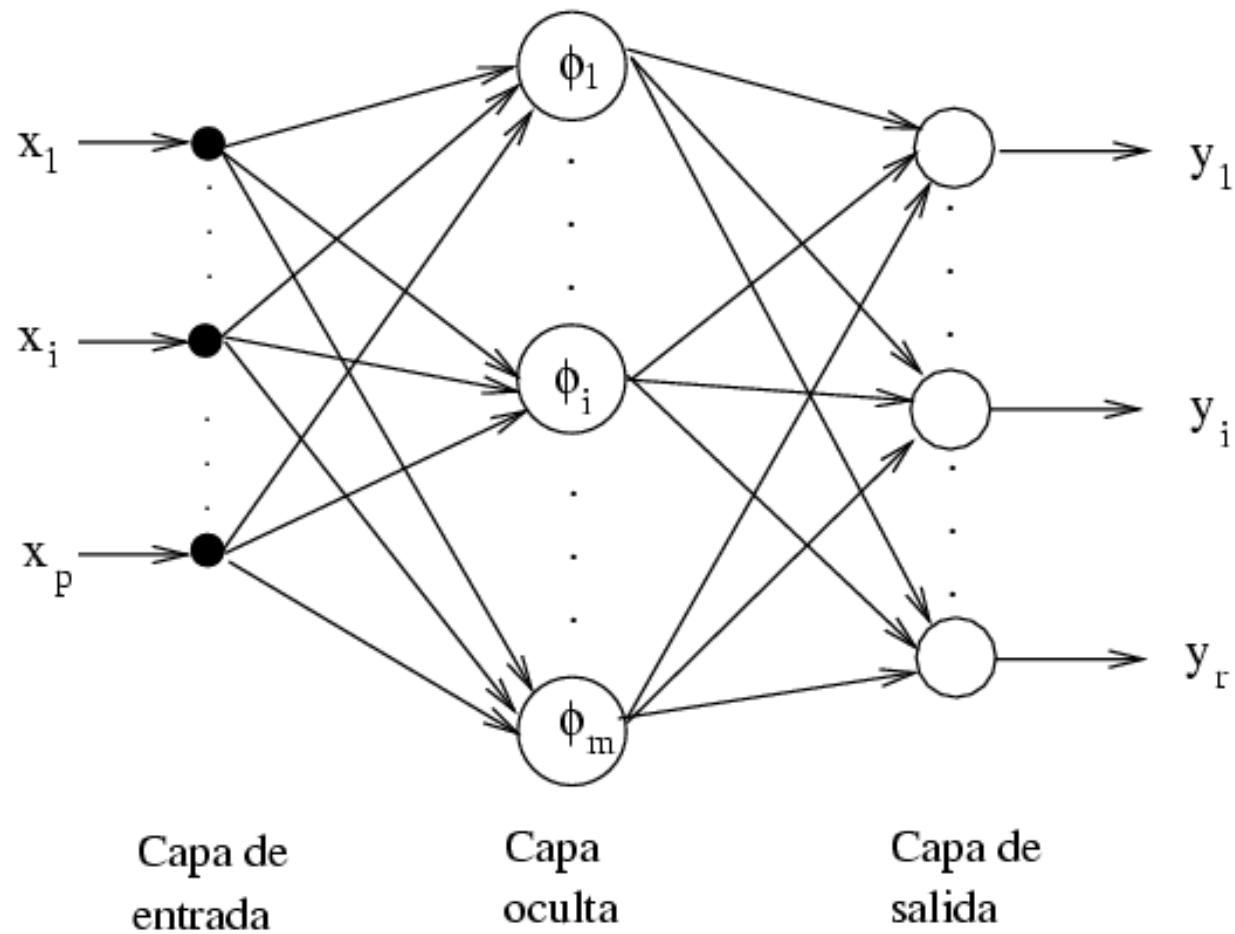




3. Redes de Función de Base Radial

- Son aproximadores universales (como el MLP)
 - Demostrado formalmente por Park y Sandberg (1991)
 - Tiempo de entrenamiento mucho más reducido
- Las funciones de base radial (RBF) definen hiperesferas o hiperelipses que dividen el espacio de entrada en regiones excluyentes
 - Cada neurona RBF construye una aproximación local no lineal en una determinada región del espacio de entrada
 - Las Redes RBF construyen aproximaciones que son combinaciones lineales de múltiples funciones locales no lineales

3.1 Arquitectura



3.1 Arquitectura



- Tres capas de neuronas
 - Capa de entrada
 - Transmiten las señales de entrada a las neuronas ocultas sin realizar procesamiento. Las conexiones de la capa de entrada a la capa oculta no llevan pesos asociados
 - Capa oculta (una sola)
 - Operan en base a la distancia que separa el vector de entrada del *centroide* de la neurona ()
 - **CARÁCTER LOCAL:** Cada neurona oculta responde únicamente a entradas cerca de su centroide y se activa en una región del espacio de entrada distinta de las demás.
 - **CARÁCTER NO LINEAL:** A la distancia vector de entrada – centroide se le aplica una función de base radial como función de activación (generalmente la **función gaussiana**)
 - Capa de salida
 - Las neuronas de salida realizan una **combinación lineal** de las activaciones de las neuronas ocultas

3.2 Procesamiento



- Aplicación de \mathcal{R}^n en \mathcal{R}^m con neuronas ocultas

1. Se presenta el patrón de entrada: $\vec{X}^p = (x_1^p, x_2^p, \dots, x_n^p)$
2. Las neuronas de la capa oculta se activan en función de la distancia del vector de entrada con su centroide

$$\phi_i^p = \phi\left(\frac{\|X^p - C_i\|}{d_i}\right) \text{ para } i = 1, 2, \dots, l$$

- Donde
 - Φ : es una Función de Base Radial
 - C_i : es el vector centroide de cada neurona de la capa oculta
 - d_i : es la desviación σ de la función Φ (*anchura de la campana*)
 - $\|\cdot\|$: es la distancia euclídea entre cada centroide y el vector X^p

$$r = \|X^p - C_i\| = \sqrt{\sum_{j=1}^n (x_j^p - c_{ij})^2}$$

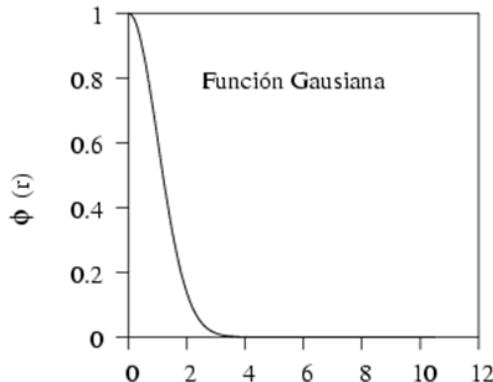
3.2 Procesamiento



- Funciones de Base Radial más habituales:

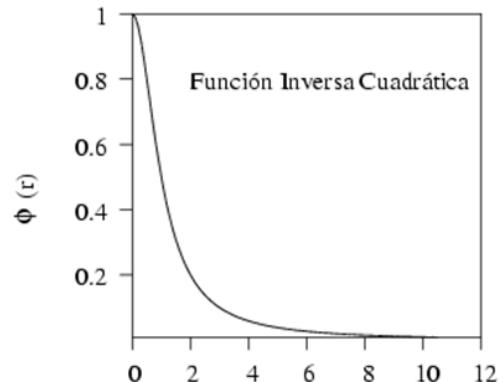
- Gausiana

$$\phi(r) = \exp\left(\frac{-r^2}{2d^2}\right)$$



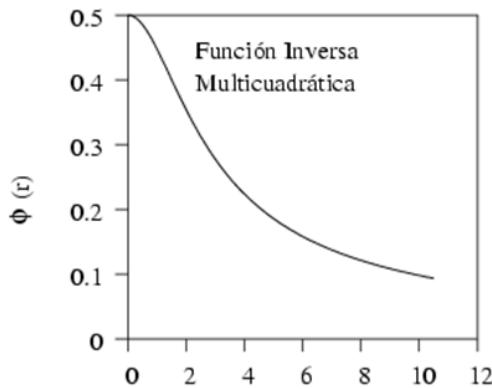
- Inversa cuadrática

$$\phi(r) = \frac{1}{d^2 + r^2}$$



- Inversa multicuadrática

$$\phi(r) = \frac{1}{\sqrt{d^2 + r^2}}$$

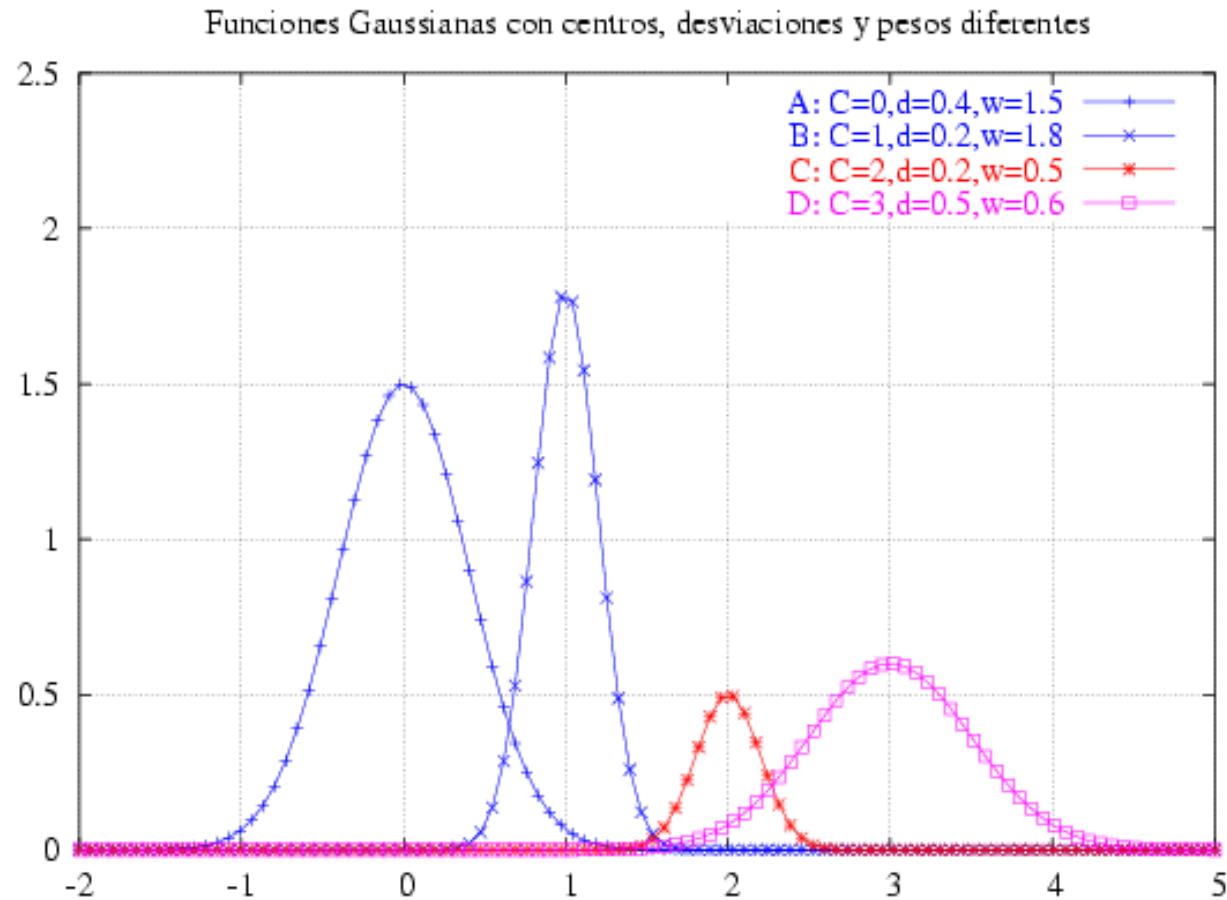


3.2 Procesamiento



- La más utilizada es la función gaussiana

$$\phi_i(n) = e^{-\frac{\|X(n)-C_i\|^2}{2d_i^2}}$$



3.2 Procesamiento



3. La salida de la red son las activaciones lineales de las neuronas de la capa de salida y_o^p

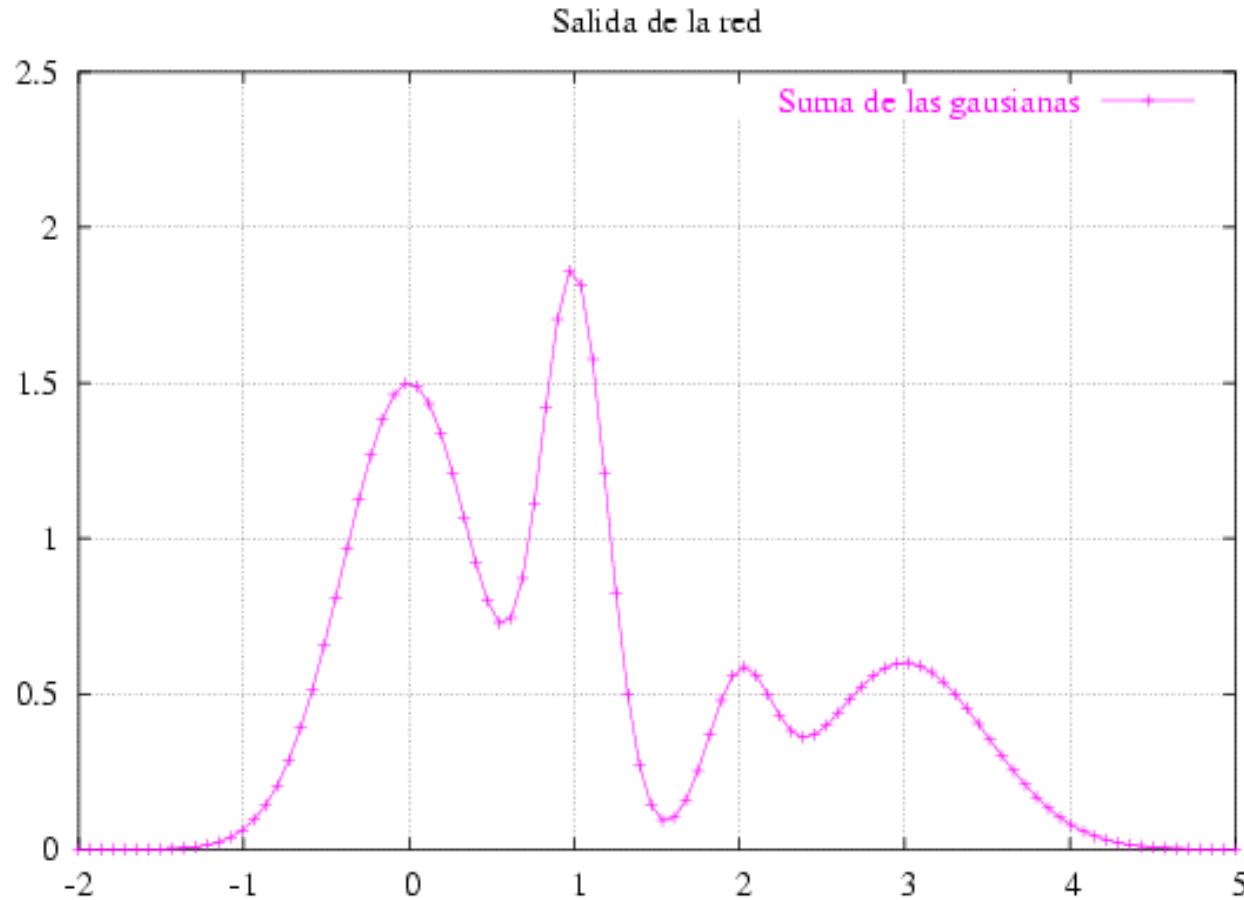
$$y_o^p = \sum_{h=1}^l w_{ho} \phi_h^p \text{ para } o = 1, 2, \dots, m$$

- Donde
 - w_{ho} : peso de la conexión neurona oculta h – neurona de salida o
 - ϕ_h^p : activación de la neurona oculta h con el patrón p



3.2 Procesamiento

- La salida de la red es una combinación lineal de las gaussianas implementadas en la capa oculta



3.3 Aprendizaje



- Como las capas de la red realizan tareas diferentes, se separa la obtención de los parámetros de la capa oculta y los de la capa de salida empleando un **aprendizaje híbrido**
 - **Fase no supervisada:** neuronas de la capa oculta
 - *Centroides* (centro de la RBF de cada clase) y
 - *Desviaciones* (amplitud de cada clase)
 - Proceso guiado por una optimización en el espacio de entrada
 - Deben ser determinados con el objetivo de clasificar el espacio de entrada en diferentes clases
 - **Fase supervisada:** neuronas de la capa de salida
 - *Pesos*: que unen la capa oculta con la capa de salida
 - En base a las salidas que se desea obtener

3.3 Aprendizaje



Fase no supervisada

Determinación de los Centroides

- Algoritmo de clasificación no supervisado que permita dividir el espacio de patrones de entrada en clases
- El número de clases es el número de neuronas ocultas en la red de base radial
- Método más utilizado: algoritmo de K-medias (J. McQueen, 1967)
 - Algoritmo mediante el cual el espacio de patrones de entrada se divide en K clases o regiones
 - El representante de cada una de estas clases, C_i , será el centro de la neurona oculta i.
 - Dichos centros se determinan con el objetivo de minimizar las distancias euclídeas entre los patrones de entrada y el centro más cercano



3.3 Aprendizaje

- Equivalente a la regla de aprendizaje competitivo de Kohonen (sin vecindario)

$$J = \sum_{i=1}^K \sum_{n=1}^N M_{in} \|X(n) - C_i\|$$

- Donde
 - N es el número de patrones
 - $\|\cdot\|$ es la distancia euclídea
 - $X(n)$ es el patrón de entrada n
 - M_{in} es la función de pertenencia, que vale 1 si el centro C_i es el más cercano al patrón $X(n)$, y 0 en otro caso



3.3 Aprendizaje

Determinación de las Desviaciones (σ)

- Las amplitudes se calculan de manera que cada neurona oculta se active en una región del espacio de entrada y el solapamiento de las zonas de activación de una neurona a otra sea lo mínimo posible
- Método heurístico
- Una opción bastante efectiva es determinar la amplitud de la función de base radial como la media geométrica de la distancia del centro a sus dos vecinos más cercanos:

$$d_i = \sqrt{\|C_i - C_t\| \|C_i - C_s\|}$$

siendo C_t y C_s los dos centros más cercanos al centro C_i .

3.3 Aprendizaje



Fase supervisada

Determinación de los Pesos

- El objetivo es minimizar las diferencias entre las salidas de la red y las salidas deseadas minimizando la función error computada en la salida de la red
- Como la salida de la red depende linealmente de los pesos, puede utilizarse el método de mínimos cuadrados que en este caso es equivalente a la Regla Delta.