

# Aprendizaje Automático

# Recordatorio de Redes de Neuronas

Francisco Serradilla García

Departamento de Inteligencia Artificial

2 de febrero de 2023

# Definición

- \* Las redes de neuronas artificiales son sistemas computacionales (hardware y software) que imitan las capacidades computacionales de los sistemas biológicos utilizando un gran número de elementos simples interconectados.
- \* Estos elementos simples son las neuronas artificiales, que son emulaciones sencillas de las neuronas biológicas. Cada neurona toma información de otras neuronas, efectúa algunas operaciones muy simples y pasa el resultado a otras neuronas

# Redes más utilizadas

- \* Perceptrón Multicapa
  - Deep learning
- \* Mapas autoorganizados (descienden de los LVQ)
  - Neural gas network
- \* Support Vector Machines (SVM)
  - Kernel networks
- \* Deep Learning
  - CNNs
  - Recurrentes (LSTM, GRU...)
  - Generative Adversarial Networks

# Ventajas de Las RN

- \* Aprendizaje adaptativo: aprenden a partir de un conjunto de experiencias. Por tanto no necesitan experto.
- \* Autoorganización: crean su propia representación de la información que reciben durante el entrenamiento.
- \* Tolerancia a fallos: pueden trabajar con información parcial debido a que codifican la información de modo distribuido y redundante.
- \* Flexibilidad: pueden adaptarse a un nuevo entorno sin necesidad de ser reprogramadas, gracias a su capacidad de aprendizaje

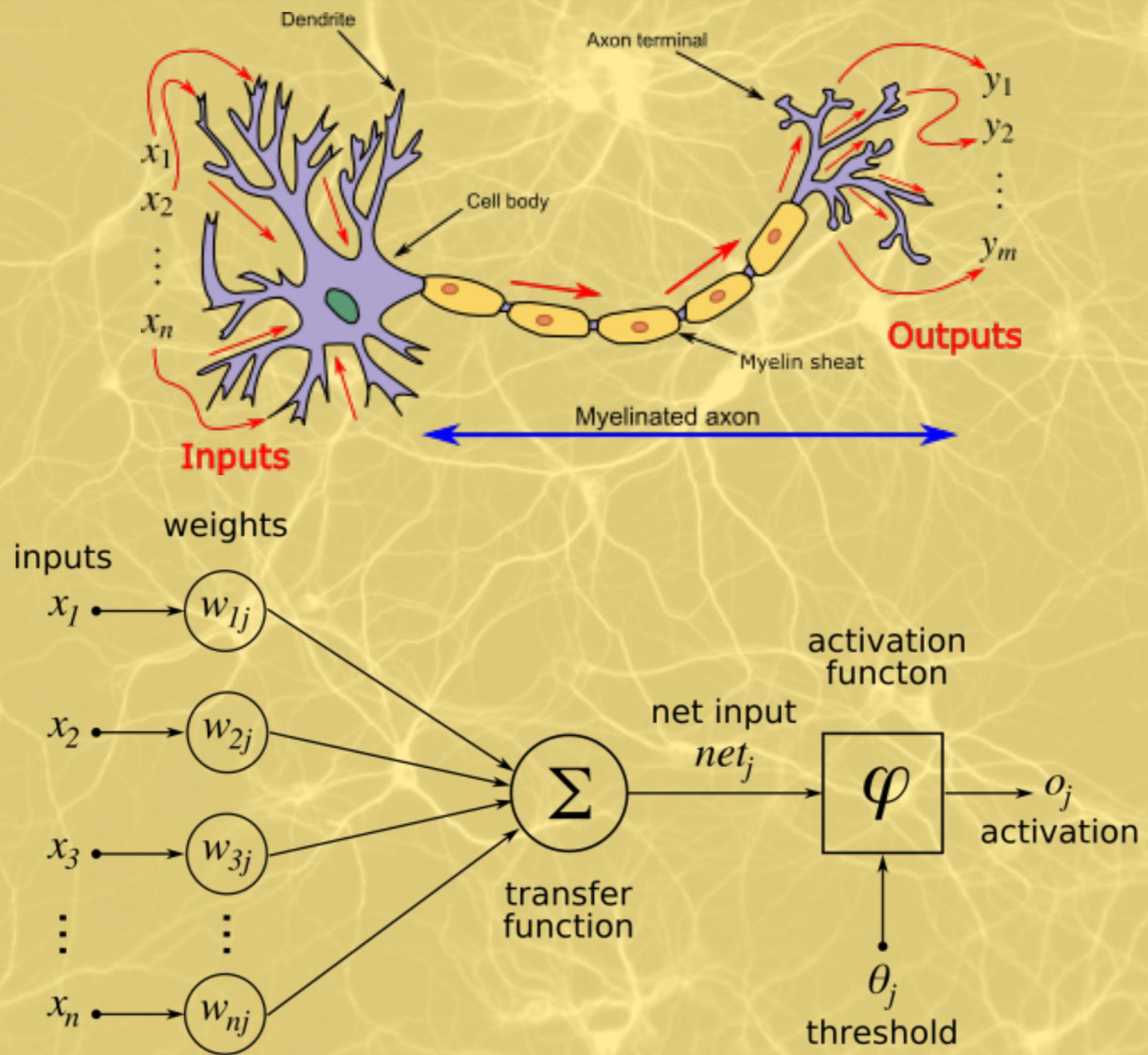
# Ventajas de Las RN (II)

- \* Tiempo real: pueden implementarse en máquinas paralelas, alcanzándose velocidades de operación elevadas
- \* Generalización: una vez que una red aprende, es capaz de clasificar objetos desconocidos. También son capaces de “saber que no saben”
- \* Datos imperfectos: Pueden trabajar con información imprecisa, probabilística, ruidosa o incluso inconsistente

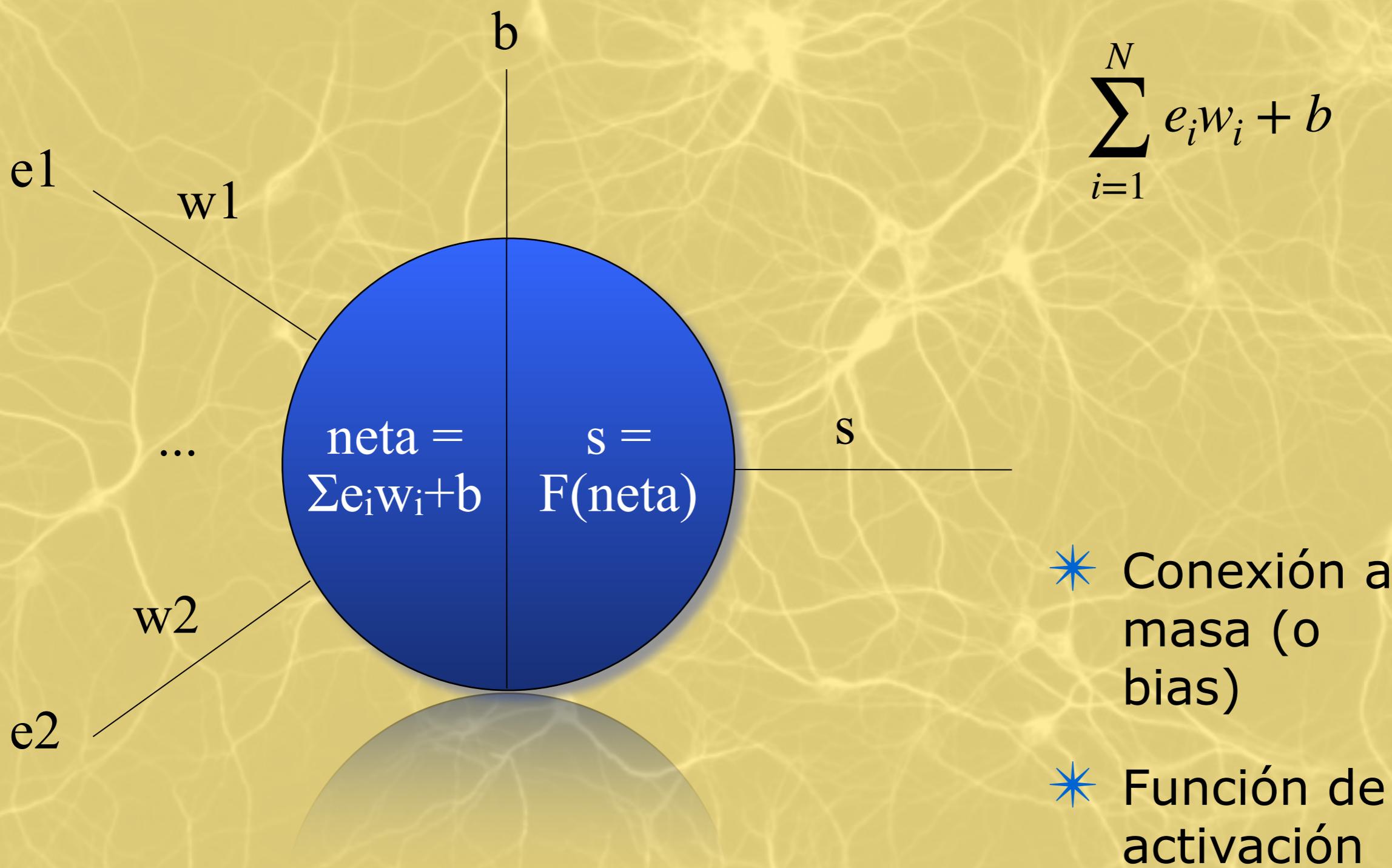
# Inconvenientes

- \* Sobre el diseño
  - Elecciones de arquitectura y parámetros por ensayo y error
  - Aprendizaje lento y a veces difícil: entrenar es un arte
- \* Necesidad de (muchos) datos, o al menos de refuerzo
- \* **Ausencia de explicación de las decisiones**

# La neurona formal



# La neurona formal



# Tipos de funciones de activación

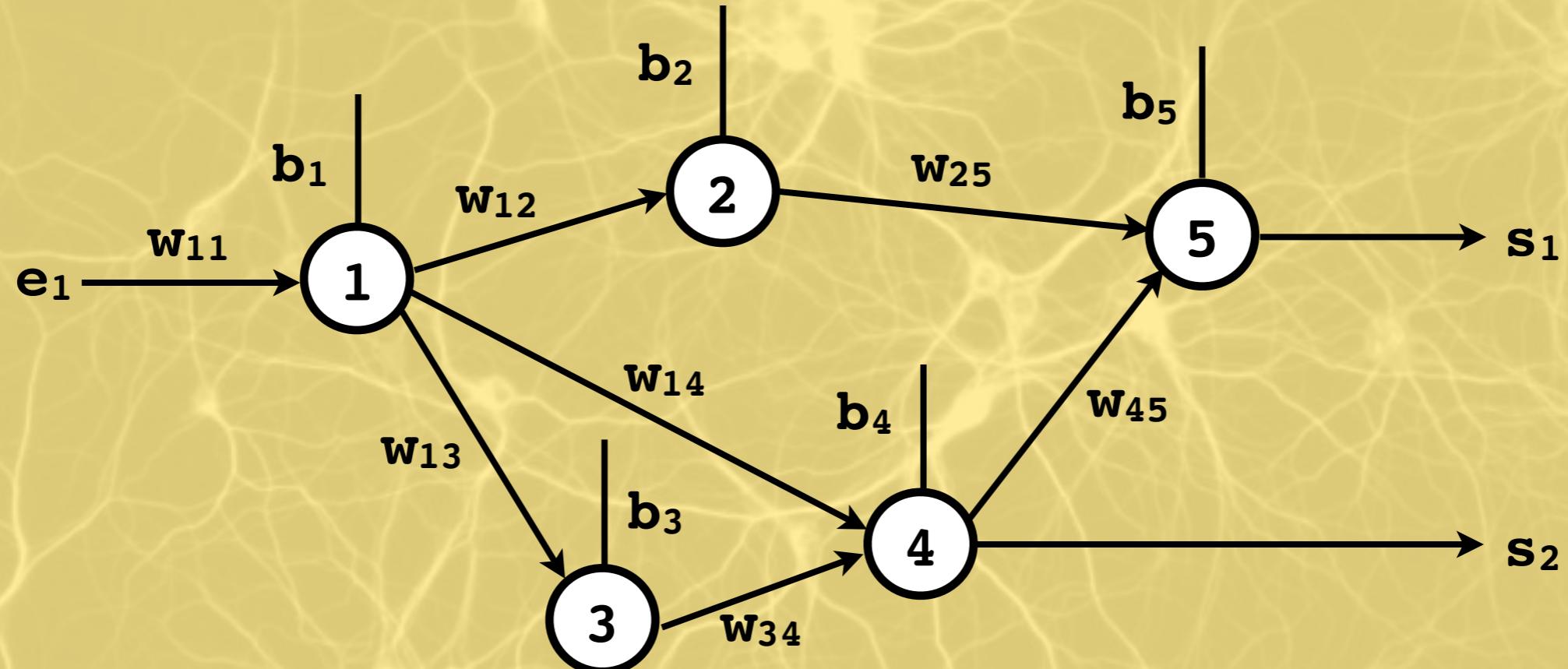
Función de activación	Expresión	Derivada
Escalón	$f(x)=1 \text{ si } x>0;$ $f(x)=0 \text{ si } x\leq 0$	$f'(x)$ no tiene
Lineal	$f(x)=x$	$f'(x)=1$
Sigmoidal	$f(x)=1 / (1 + \exp(-x))$	$f'(x)= f(x) (1 - f(x))$
Tangente hiperbólica	$f(x)=\tanh(x)$	$f'(x)=1 - \tanh^2(x)$
ReLU	$\max(0,x)$	$f'(x)= 1 \text{ si } x>0$

# El cerebro en cifras

- \*  $10^{11}$  neuronas
- \*  $10^{14}$  conexiones
- \* Cada neurona está conectada a otras 1000 – 200000
- \* La tasa máxima de activación es de unos 1000 impulsos por minuto. Activación / desactivación significa mayor o menor tasa de activación.
- \* Las conexiones se autoorganizan en respuesta a sus entradas
- \* El error en alguno de sus elementos no es esencial, pues hay un alto nivel de redundancia

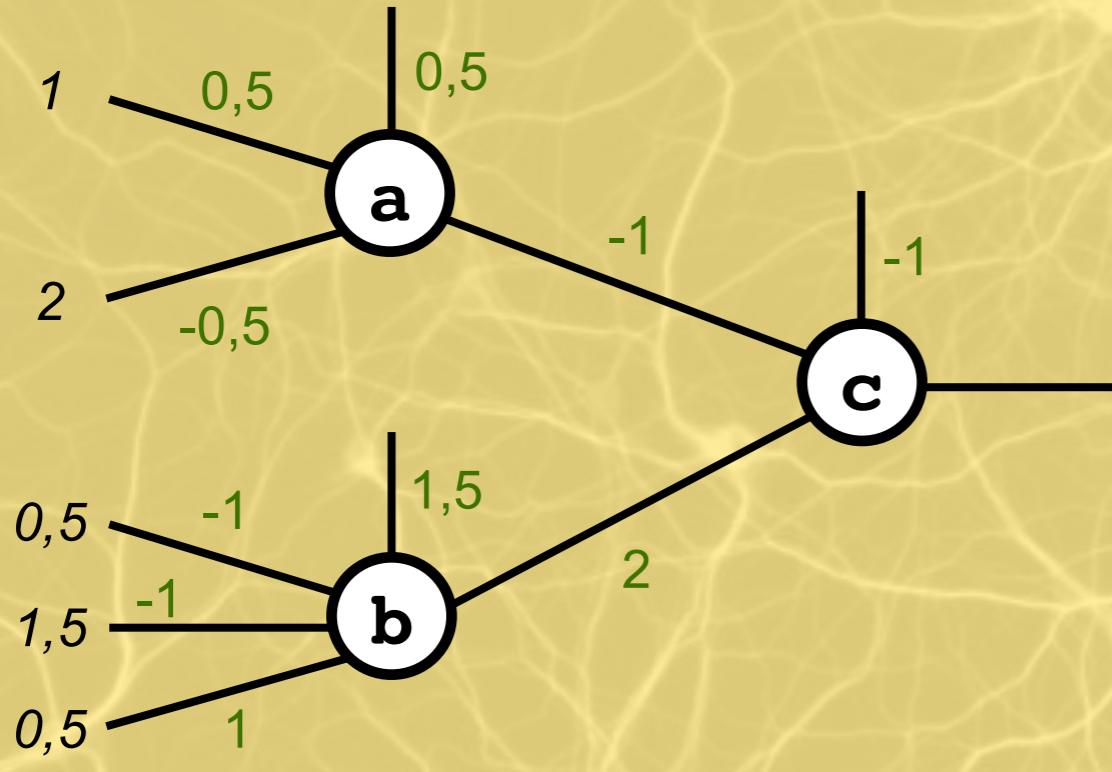


# Interconexión en redes



- \* Pesos
- \* Entradas / salidas / ocultas
- \* Excitación / inhibición

# Ejemplo de propagación en red



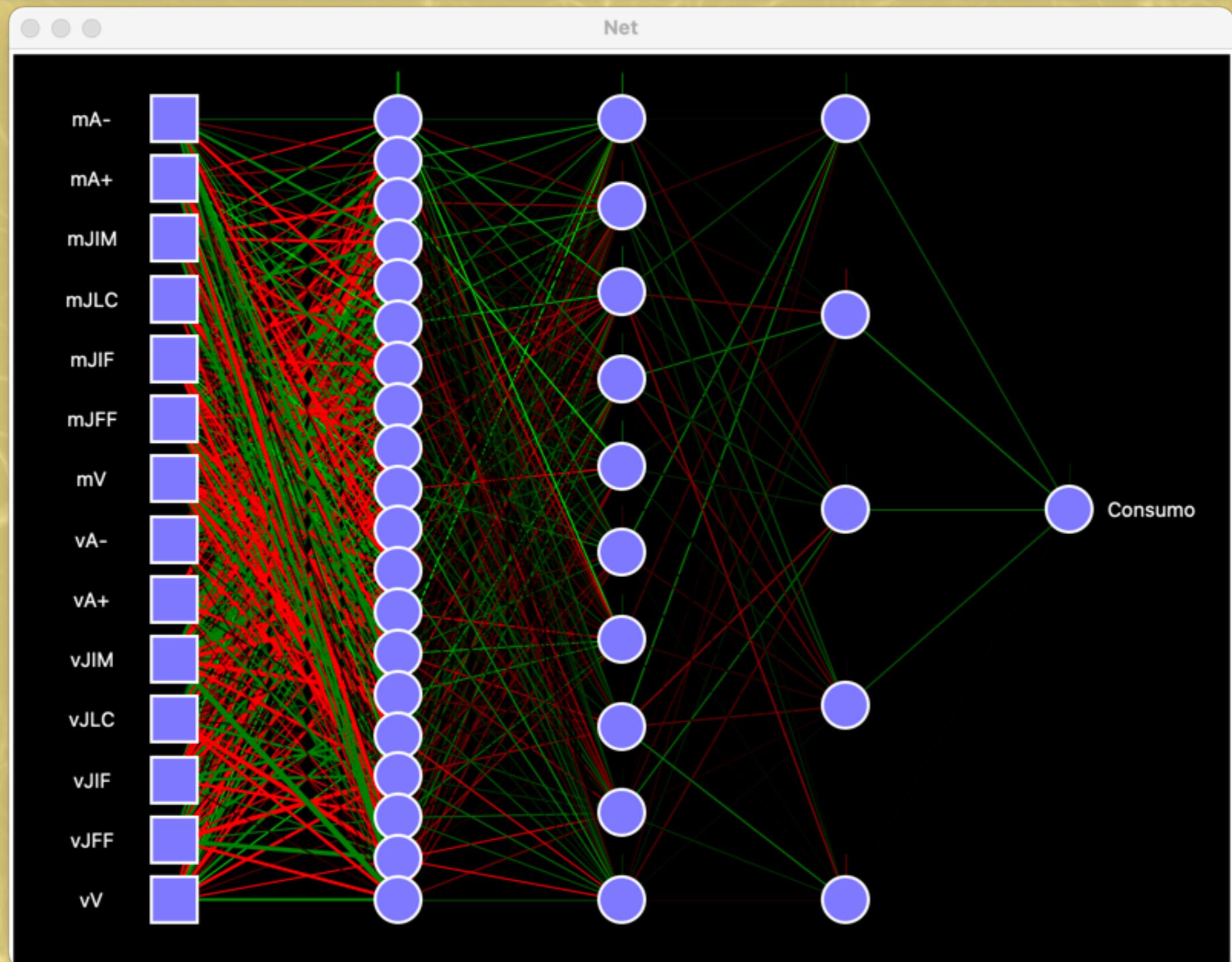
- \* Calcular salida, sabiendo que
  - neurona a tiene una función de activación de tipo escalón
  - neurona b de tipo lineal
  - neurona c de tipo sigmoidal

$$\text{neta}_a = 0,5 + 0,5 - 1 = 0; F(\text{neta}_a) = 0$$

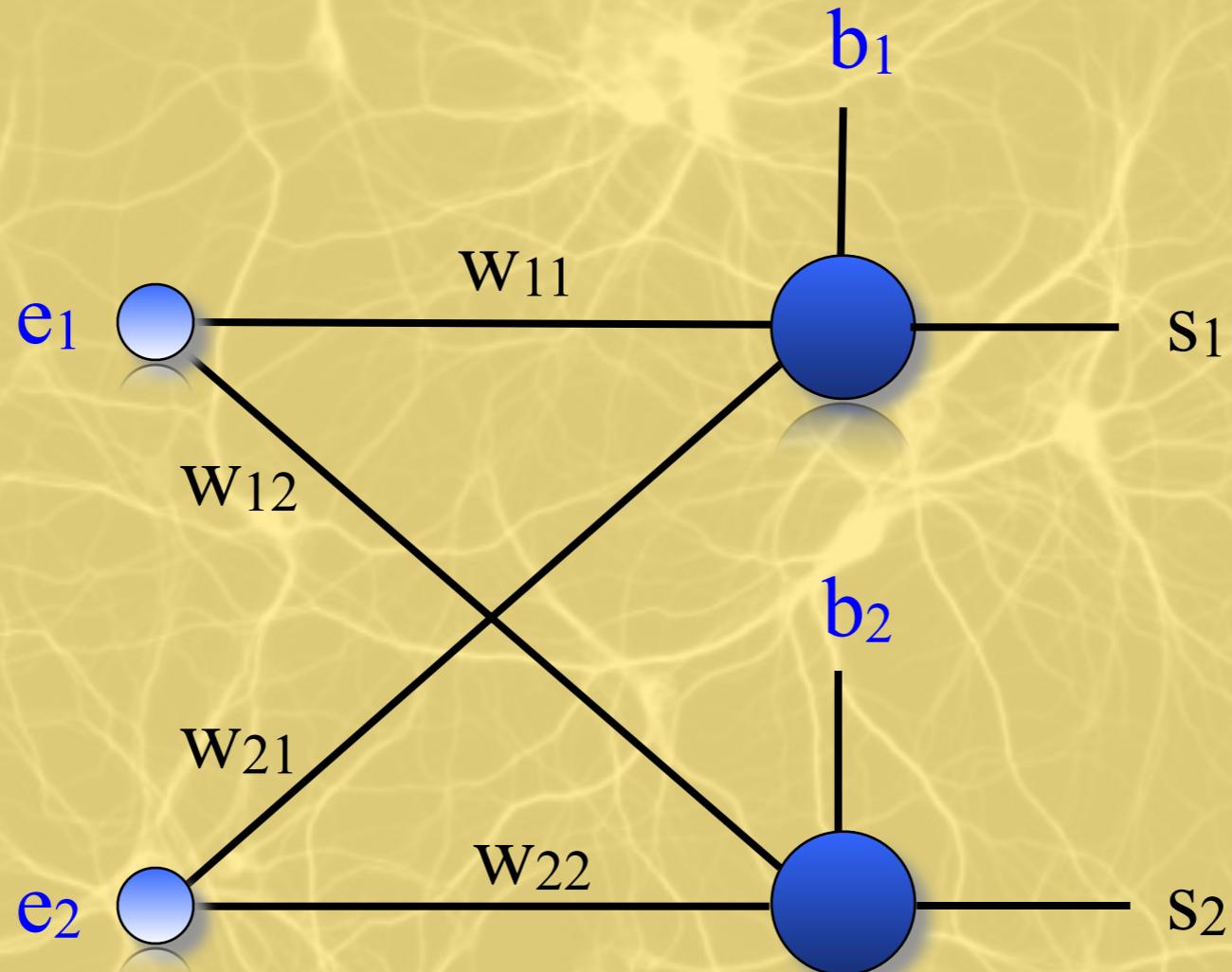
$$\text{neta}_b = 1,5 - 0,5 - 1,5 + 0,5 = 0; F(\text{neta}_b) = 0$$

$$\text{neta}_c = -1 + 0 + 0 = -1; F(\text{neta}_c) = 1 / (1 + \exp(1)) = 0,27$$

# Organización en capas



# Representación matricial



$$neta = [e_1 \ e_2 \ \dots] \begin{bmatrix} w_{11} & w_{12} & \dots \\ w_{21} & w_{22} & \dots \\ \dots & \dots & \dots \end{bmatrix} + [b_1 \ b_2 \ \dots]$$
$$s = F(neta)$$

# Propagación en capas

$$s^{(1)} = F(eW^{(1)} + b^{(1)})$$

$$s^{(2)} = F(s^{(1)}W^{(2)} + b^{(2)})$$

*genericamente :*

$$s^{(k)} = F(s^{(k-1)}W^{(k)} + b^{(k)})$$

*con  $s^{(0)} = e$*

# ¿Dónde reside el conocimiento?

- \* Conocimiento a corto plazo (memoria de trabajo)
  - Neuronas
- \* Conocimiento a largo plazo (lo que el sistema sabe hacer)
  - Conexiones
- \* Aprendizaje
  - Consiste en encontrar las conexiones para un problema determinado

# Aprendizaje

## \* Objetivo

- Conseguir valores adecuados para los pesos  $w_{ij}$  de toda la red
- En general no existe solución analítica al problema
- Necesitamos un conjunto de ejemplos que mostraremos a la red un cierto número de veces
- Se utilizarán procedimientos de aproximación numérica a la solución, con algoritmos iterativos

## \* Al proceso de aprendizaje también se le denomina entrenamiento de la red

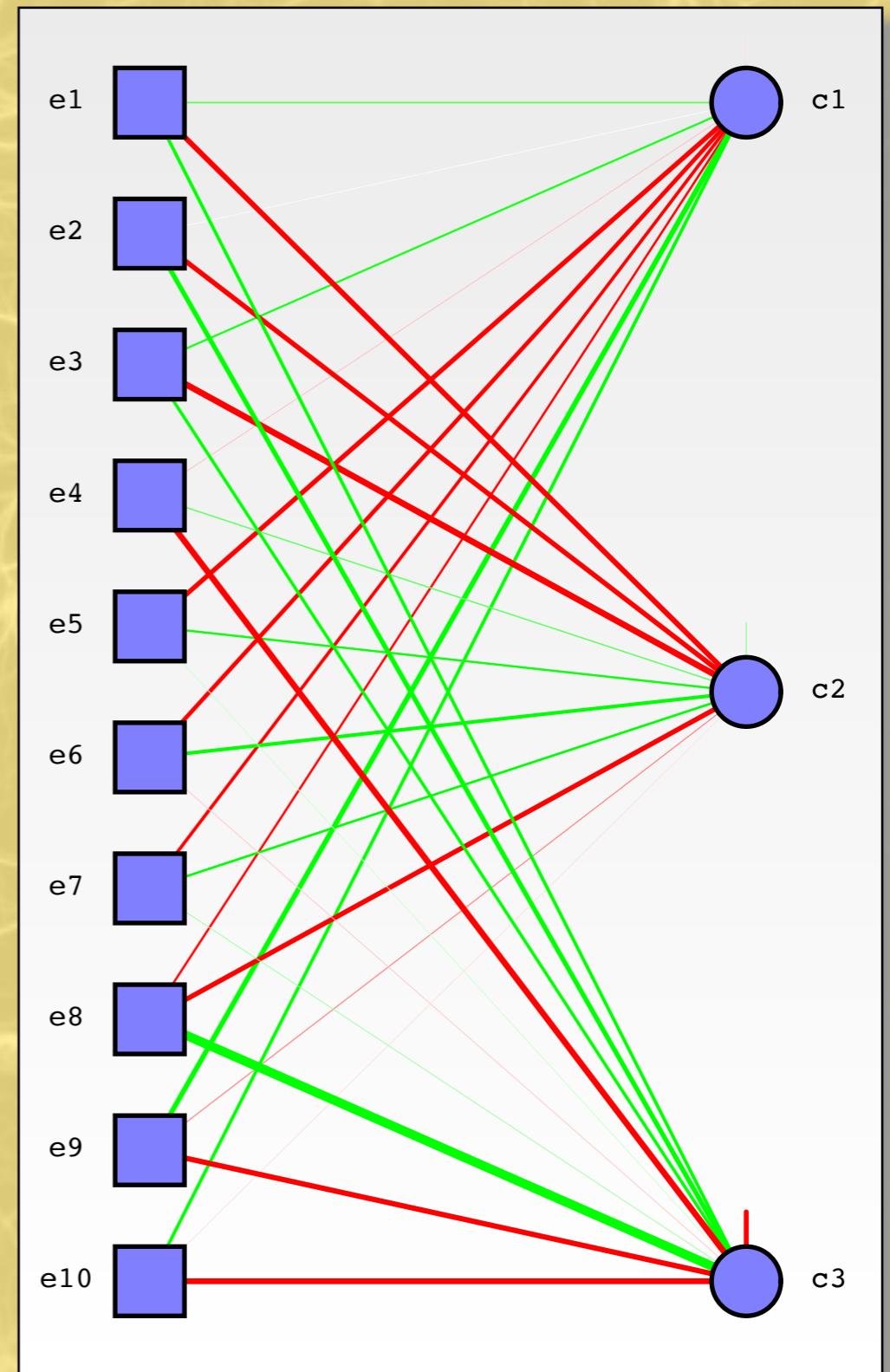
# Técnicas de Entrenamiento

- \* Cada técnica está asociada a un modelo de red y a un tipo de aprendizaje
  - Ley de Hebb → ADALINE
  - Regla delta → perceptrón
  - Regla delta generalizada → perceptrón multicapa
  - Aprendizaje competitivo → mapas autoorganizados
  - ...
- \* En tensorflow estas técnicas se llaman optimizadores
  - SGD, Adam, Adadelta...

# EL Perceptrón

- \* Es una red con una única capa, la de salida
- \* Cada neurona de la capa de salida corresponde a una clase
- \* Todas las entradas se conectan a todas las salidas
- \* Se utiliza función de activación de tipo escalón

$$s = F_{escalon}(eW + b)$$



# Regla de aprendizaje del perceptrón

- \* Intenta modificar las conexiones de modo que se minimice la diferencia entre la salida deseada y la salida actual para cada ejemplo concreto
  - $w_{ij}$  debe modificarse proporcionalmente a



$$\begin{aligned}\Delta w_{ij} &\approx (d_j - s_j) \\ \Delta w_{ij} &\approx e_i (d_j - s_j) \\ \Delta w_{ij} &= \alpha e_i (d_j - s_j) \\ \Delta W &= \alpha e^t (d - s) \\ W[t+1] &= W[t] + \Delta W[t]\end{aligned}$$

# Algoritmo de entrenamiento del perceptrón

- 1) Fijar  $\alpha$
- 2) Inicializar  $W$
- 3) Repetir hasta que se verifique la condición de terminación
  - a) Presentar nuevo ejemplo
  - b) Propagar para obtener la salida
  - c) Calcular  $\Delta W$
  - d) Actualizar  $W$

# Condiciones de terminación

- \* El proceso de entrenamiento puede terminar por
  - A mano
  - Número de epoch
  - Todos los ejemplos bien clasificados
  - Error RMS normalizado menor que cierto valor
  - El error deja de bajar
  - El error en el conjunto de test sube
  - ...

# Ejemplo

- \* Entrenamiento de un perceptrón para que realice el "O" lógico de las entradas, con  $\alpha = 1$

<u>e<sub>1</sub></u>	<u>e<sub>2</sub></u>	<u>d</u>
0	0	0
0	1	1
1	0	1
1	1	1

$$\Delta W = \alpha e^t (d - s)$$

$$\Delta b = \alpha (d - s)$$

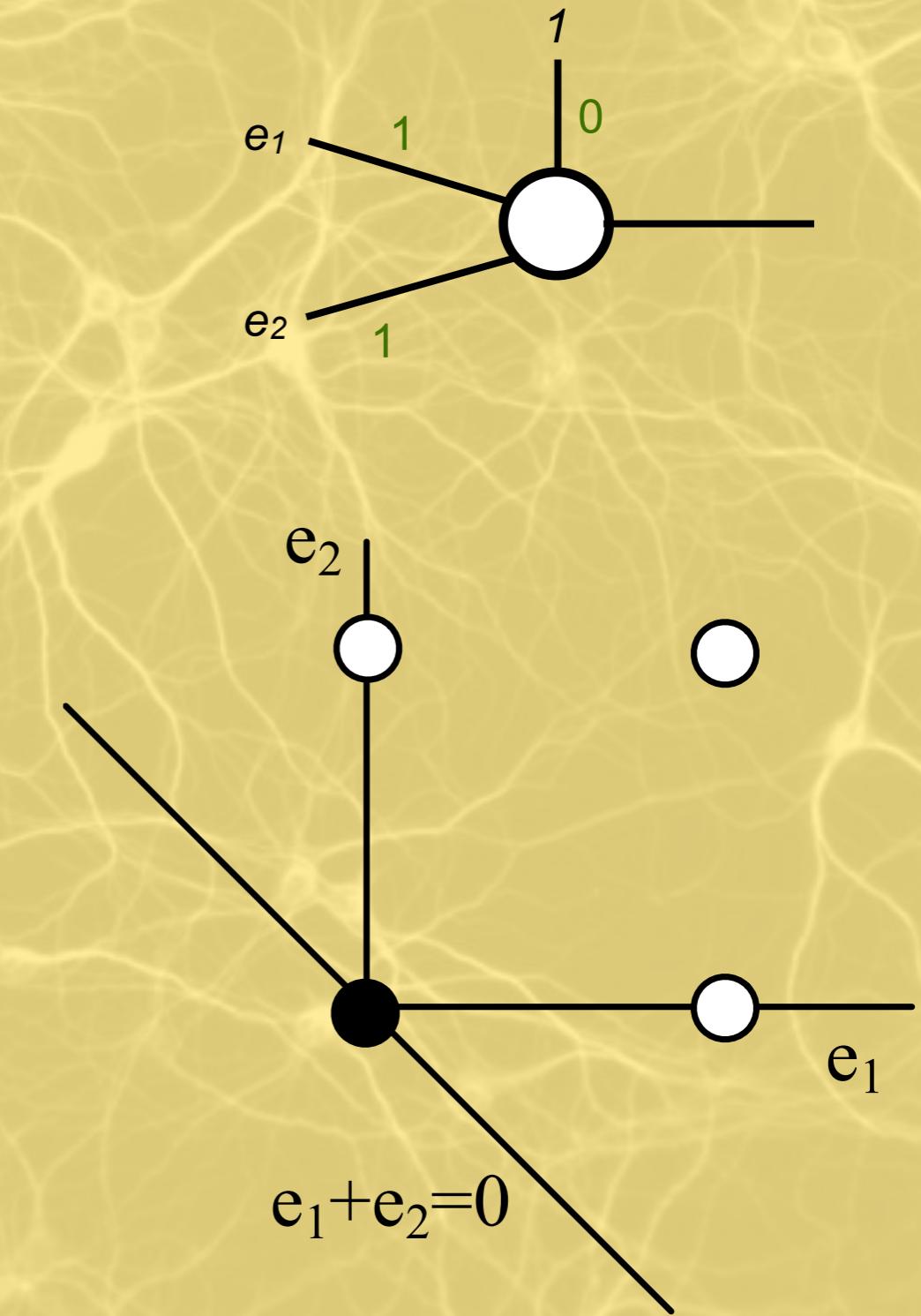
$$W[t+1] = W[t] + \Delta W[t]$$

$$b[t+1] = b[t] + \Delta b[t]$$

t	W(t) <sup>t</sup>	b	e	s	d	(d-s)	$\Delta W(t)^t$	$\Delta b$
0	(0 0)	0	(0 0)	0	0	0	(0 0)	0
1	(0 0)	0	(0 1)	0	1	1	(0 1)	1
2	(0 1)	1	(1 0)	1	1	0	(0 0)	0
3	(0 1)	1	(1 1)	1	1	0	(0 0)	0
4	(0 1)	1	(0 0)	1	0	-1	(0 0)	-1
5	(0 1)	0	(0 1)	1	1	0	(0 0)	0
6	(0 1)	0	(1 0)	0	1	1	(1 0)	1
7	(1 1)	1	(1 1)	1	1	0	(0 0)	0
8	(1 1)	1	(0 0)	1	0	-1	(0 0)	-1
9	(1 1)	0	(0 1)	1	1	0	(0 0)	0
10	(1 1)	0	(1 0)	1	1	0	(0 0)	0
11	(1 1)	0	(1 1)	1	1	0	(0 0)	0
12	(1 1)	0	(0 0)	0	0	0	(0 0)	0

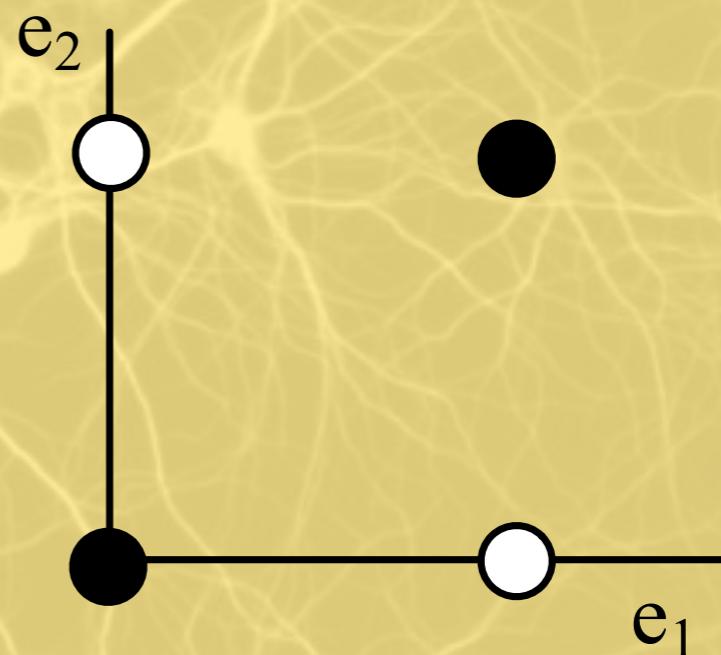
# Interpretación geométrica

- \* Como  $\text{net} = e_1w_1 + e_2w_2 + b$ , el resultado  $W=(1 \ 1)$  y  $b=0$  puede entenderse como la ecuación de una recta  $\text{net}=e_1+e_2$  que permite obtener la clase a la que pertenece el ejemplo
- \* Según la función escalón, si  $e_1+e_2>0$ ,  $F(\text{net})=1$ , y estaríamos en la región correspondiente a  $s=1$ ; en caso contrario nos encontraríamos en la región de  $s=0$



# Capacidades del perceptrón

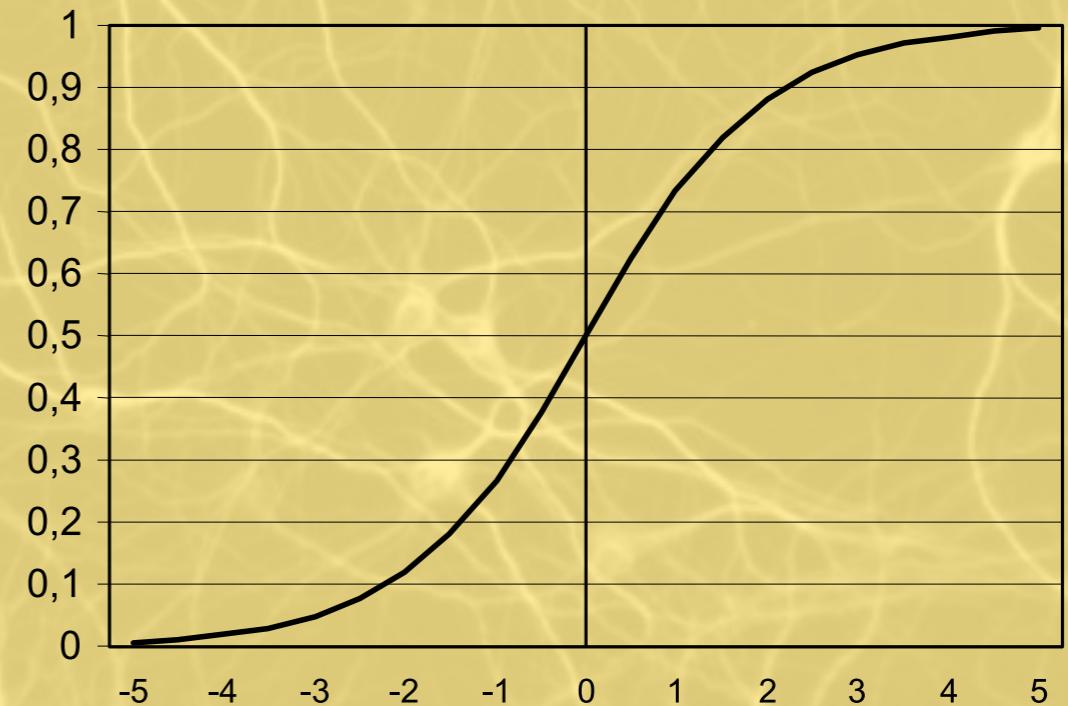
- \* El perceptrón es un clasificador lineal
- \* Por ello es incapaz de aprender funciones no separables linealmente, como la XOR
- \* Para superar estas limitaciones es necesario el uso de redes multicapa



# Perceptrón multicapa

- \* El perceptrón multicapa es una generalización del perceptrón simple con dos particularidades
  - La red consta de dos o más capas
  - Se utiliza una función de activación de tipo sigmoidal, parecida a la función escalón pero derivable

$$s = F_s(\text{neta}) = \frac{1}{1 + e^{-\text{neta}}}$$



# Propagación multicapa

$$s^{(1)} = F_s(eW^{(1)} + b^{(1)})$$

$$s^{(2)} = F_s(s^{(1)}W^{(2)} + b^{(2)})$$

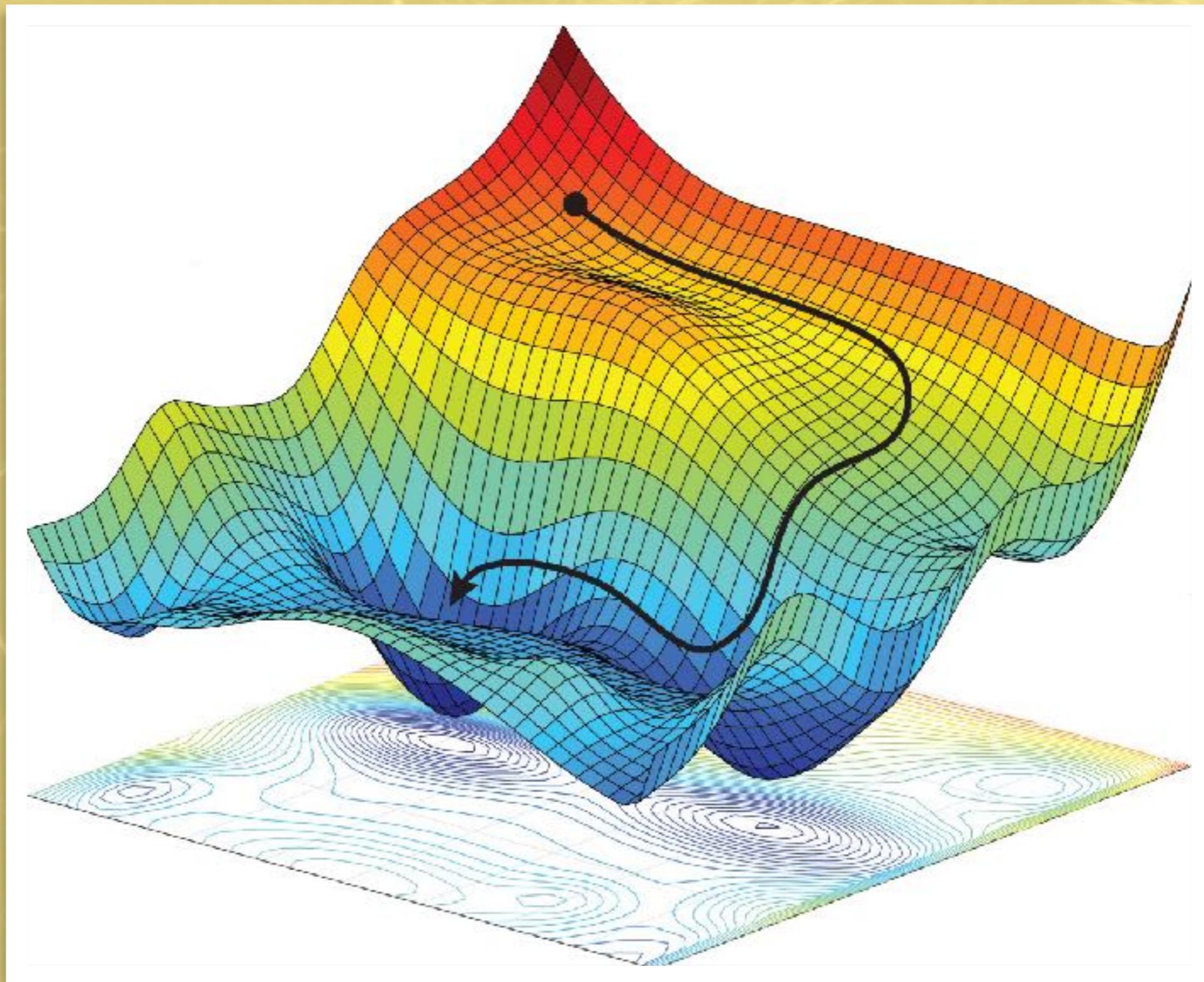
$$s^{(k)} = F_s(s^{(k-1)}W^{(k)} + b^{(k)})$$

$$\text{con } s^{(0)} = e$$

# Entrenamiento del Perceptrón Multicapa

- \* Rumelhart, Hinton y Williams proponen en 1986 el método de retropropagación del gradiente (Back Propagation)
- \* Se basa en calcular el gradiente (expresión de la máxima pendiente) del error en función de los pesos  $w_{ij}$  y modificar dichos pesos en la dirección contraria al gradiente (dirección de decrecimiento)
  - Este proceso se repite iterativamente
- \* De este modo tendemos a minimizar el error
  - Después de un cierto número de pasos alcanzaremos un mínimo local

# Idea general detrás del aprendizaje



# Expresiones del Back Propagation

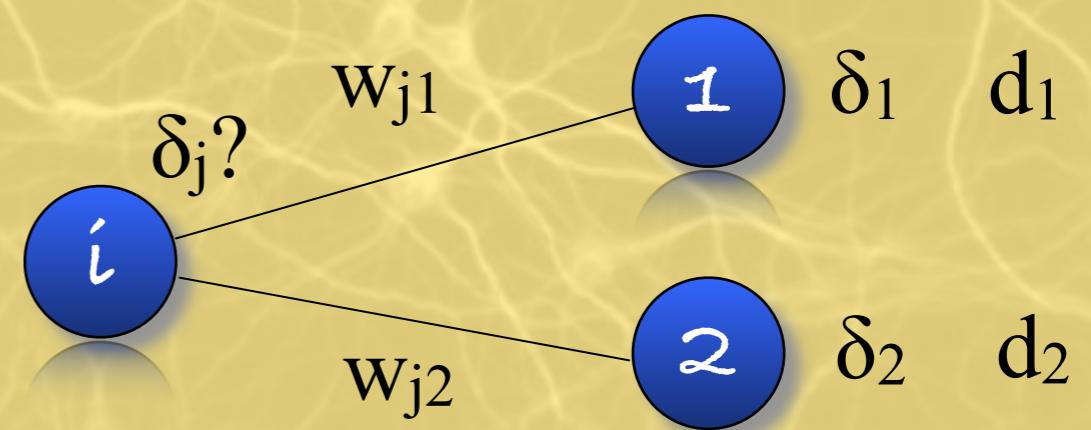
$$\Delta W^{(k)} = \alpha s^{(k-1)t} \delta^{(k)} \quad \Delta b^{(k)} = \alpha \delta^{(k)}$$

$$\delta^{(k)} = (d - s^{(k)}) s^{(k)} ([1] - s^{(k)})$$

- \* En las capas ocultas no conocemos d, la salida deseada
  - Se estima propagando hacia atrás los valores  $\delta$  de la capa siguiente, de ahí el nombre del algoritmo, porque un peso alto tendrá más culpa en el error final

$$\delta^{(k)} = R^{(k)} s^{(k)} ([1] - s^{(k)})$$

$$R^{(k)t} = W^{(k+1)} \delta^{(k+1)t}$$



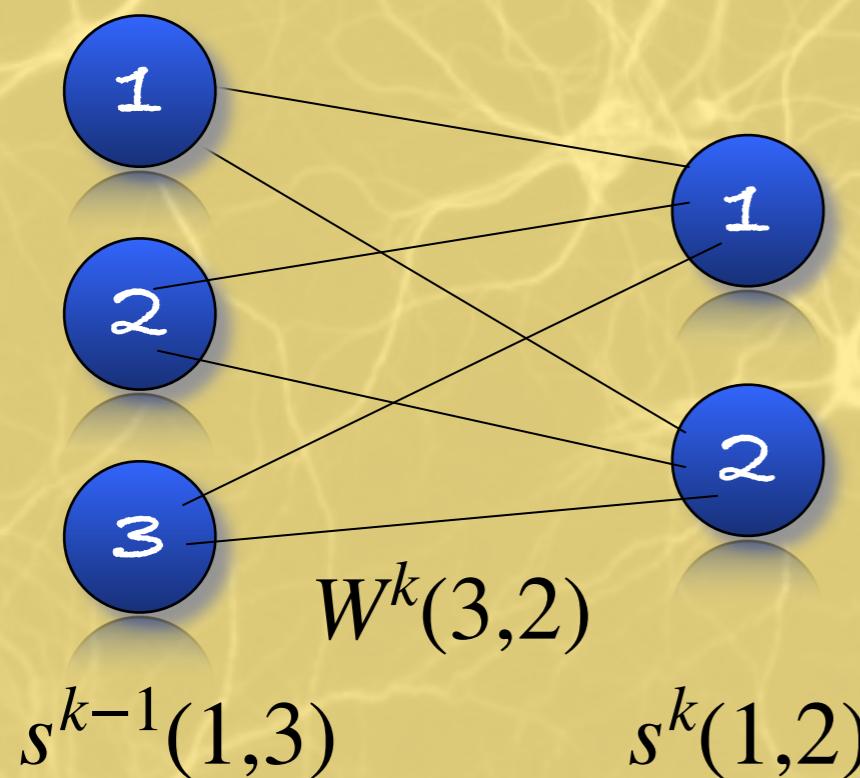
# Dimensiones de las matrices

$$\Delta W^{(k)} = \alpha s^{(k-1)t} \delta^{(k)} \quad \Delta b^{(k)} = \alpha \delta^{(k)}$$

$$\delta^{(k)} = (d - s^{(k)}) s^{(k)} ([1] - s^{(k)})$$

$$\delta^{(k)} = R^{(k)} s^{(k)} ([1] - s^{(k)})$$

$$R^{(k)t} = W^{(k+1)} \delta^{(k+1)t}$$



Dimensiones propagación

$$(1,3) \times (3,2) = (1,2)$$

Dimensiones retropropagación

$$(3,2) \times (2,1) = (3,1)$$

Dimensiones actualización

$$(3,1) \times (1,2) = (3,2)$$

# Optimizadores

- \* En tensorflow disponemos de muchos optimizadores para entrenar las redes
  - Stochastic Gradient Descent
  - RMSprop
  - Adam
  - Momentum
  - AdaGrad
  - AdaDelta
  - ...
- \* Más información
- \* <http://ruder.io/optimizing-gradient-descent/>
  - <https://stackoverflow.com/questions/37214884/how-do-i-choose-an-optimizer-for-my-tensorflow-model>

# Tensorflow flavour

```
# aquí definimos la propagación de la red  
  
s1 = tf.nn.sigmoid(tf.add(tf.matmul(e, W1), b1))  
  
s = tf.nn.sigmoid(tf.add(tf.matmul(s1, W2), b2))  
  
# definimos la función a minimizar (error cuadrático medio)  
  
vRMS = tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(d,s)),  
reduction_indices=0))  
  
RMS = tf.reduce_mean(vRMS)  
  
# definimos el algoritmo de aprendizaje a utilizar (descenso del  
gradiente sobre la función de coste)  
  
train_step = tf.train.RMSPropOptimizer(0.01).minimize(RMS)
```

# Algoritmo de Entrenamiento

- \* 1) Inicializar las matrices de pesos  $W(k)$  y los bias  $b(k)$  con pequeños valores aleatorios (normalmente en el rango  $[-0,5, 0,5]$ )
- \* 2) Repetir hasta que se verifique alguna condición de terminación
  - Tomar la siguiente entrada del conjunto de entrenamiento y propagar hasta obtener la salida de la red, guardando todos los valores intermedios  $s(k)$ . Tomar la salida deseada  $d$  para el ejemplo actual. Si hemos terminado con el conjunto de entrenamiento, tomar de nuevo el primero; cada una de estas pasadas se denomina epoch.
  - Retro-propagar el gradiente desde la salida hasta la primera capa. Este proceso permite obtener todas las matrices de modificación de pesos  $\Delta W(k)$ .
  - Una vez obtenidas todas las matrices  $\Delta W(k)$ , actualizar las matrices de pesos  $W(k)$  con  $W(k)[t+1] = W(k)[t] + \Delta W(k)[t]$

# Modos de entrenamiento y mejoras

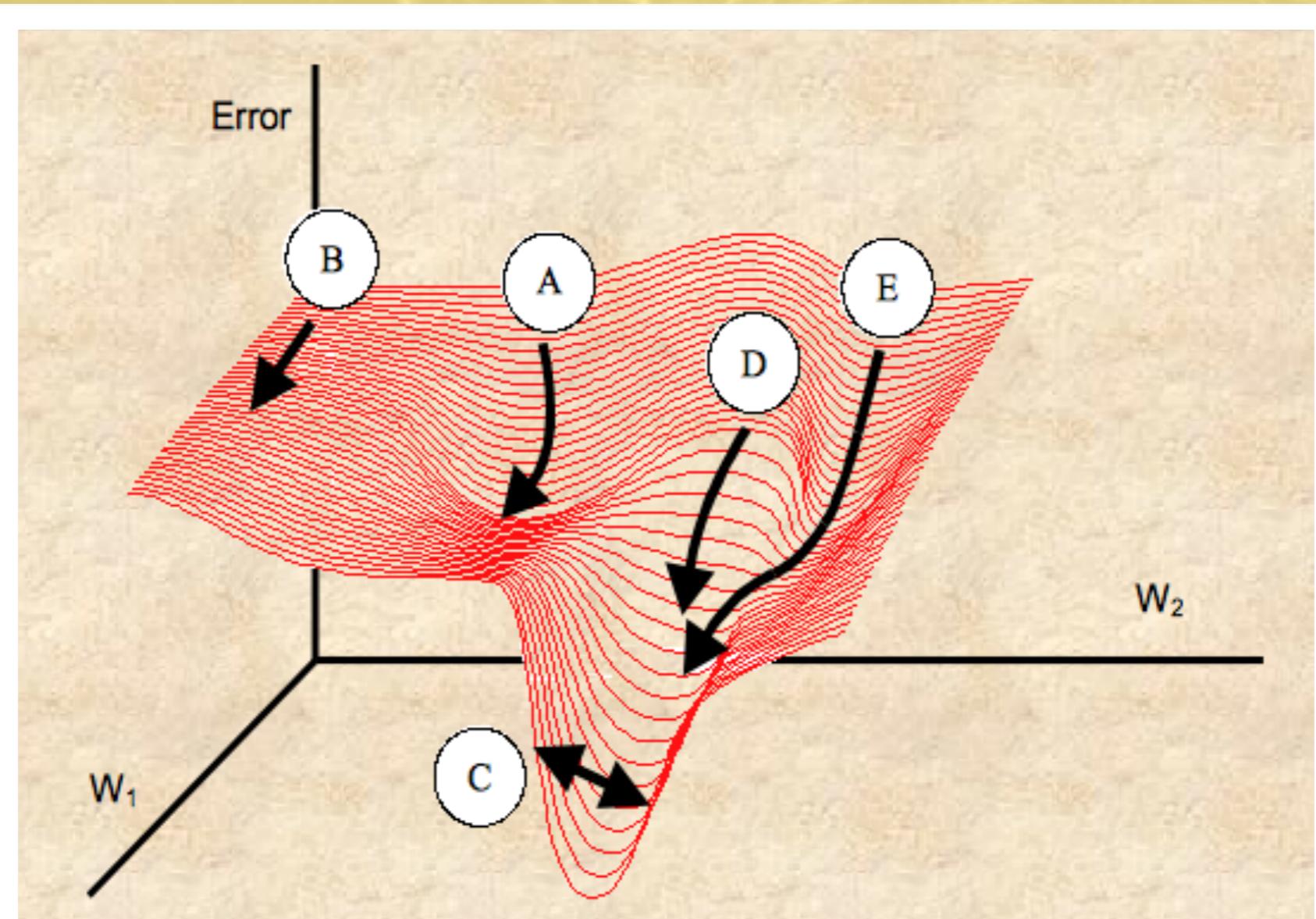
## \* Modos de entrenamiento

- Uno a uno (incremental)
- Por epoch (en batch) → menos potente, pero en algunos algoritmos (y en paralelo) sólo puede utilizarse éste
- Minibatch

## \* Mejoras

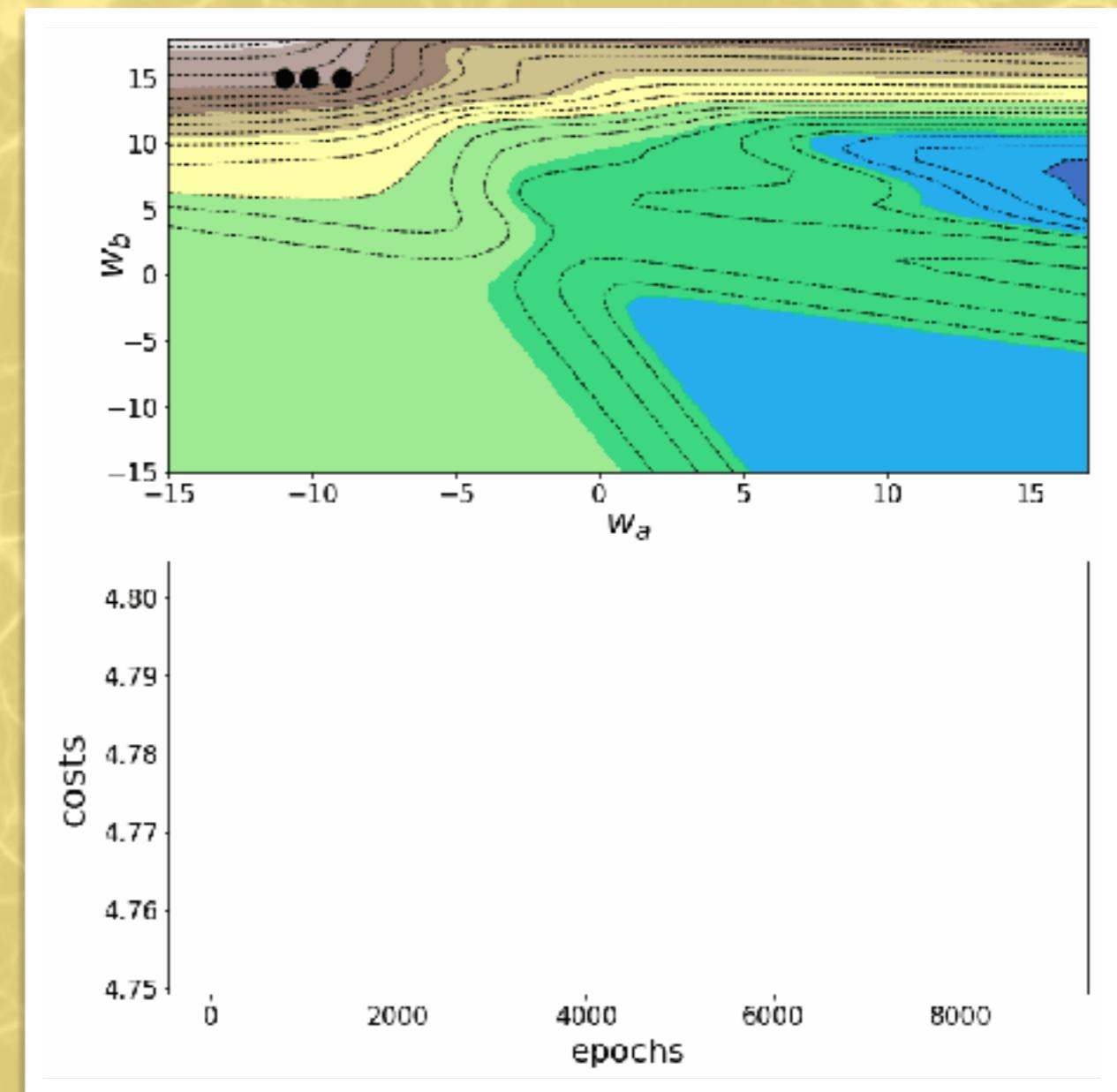
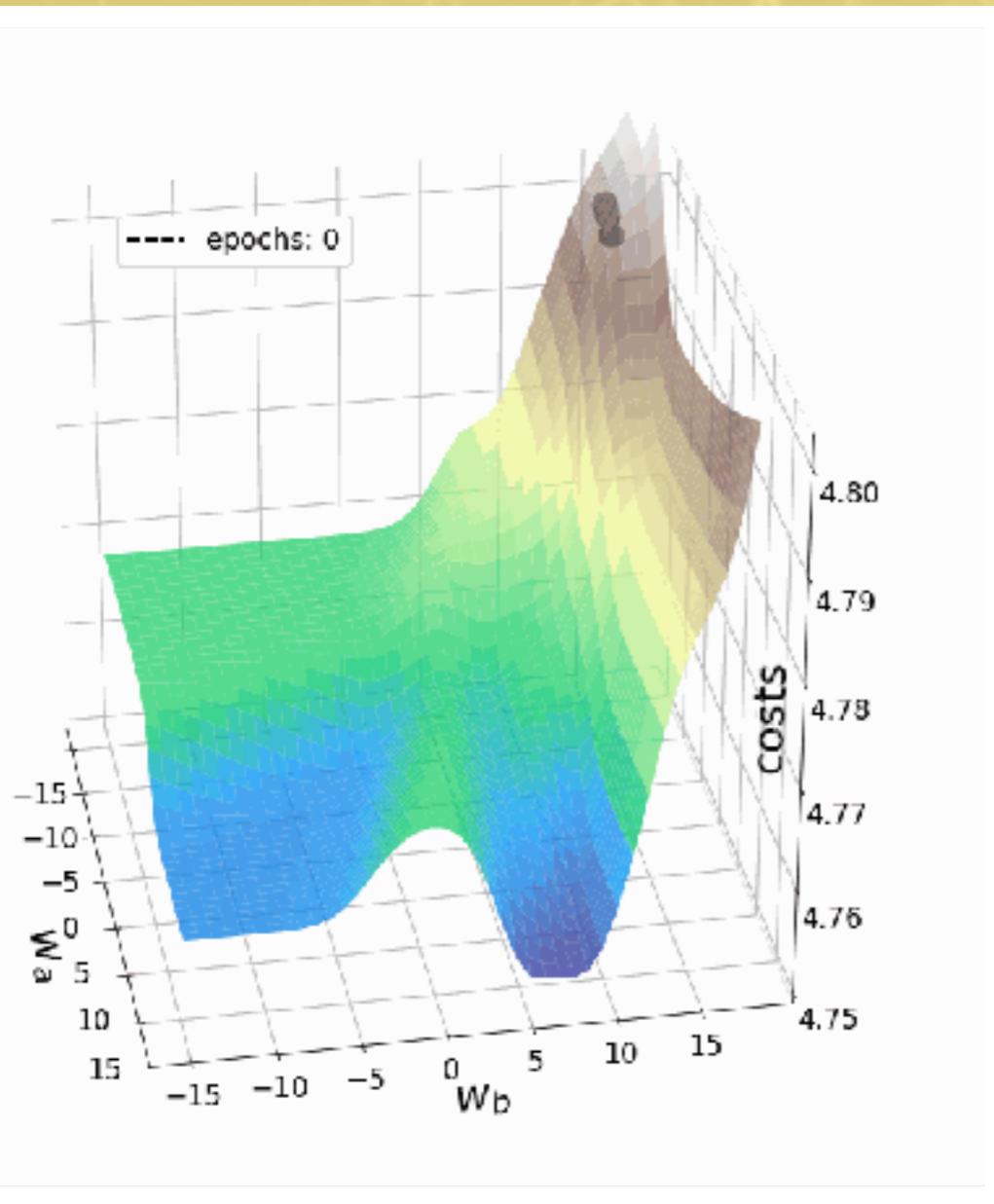
- Momento:  $W[t+1] = W[t] + \Delta W[t] + \beta \Delta W[t-1]$
- RPROP

# Problemas del BP

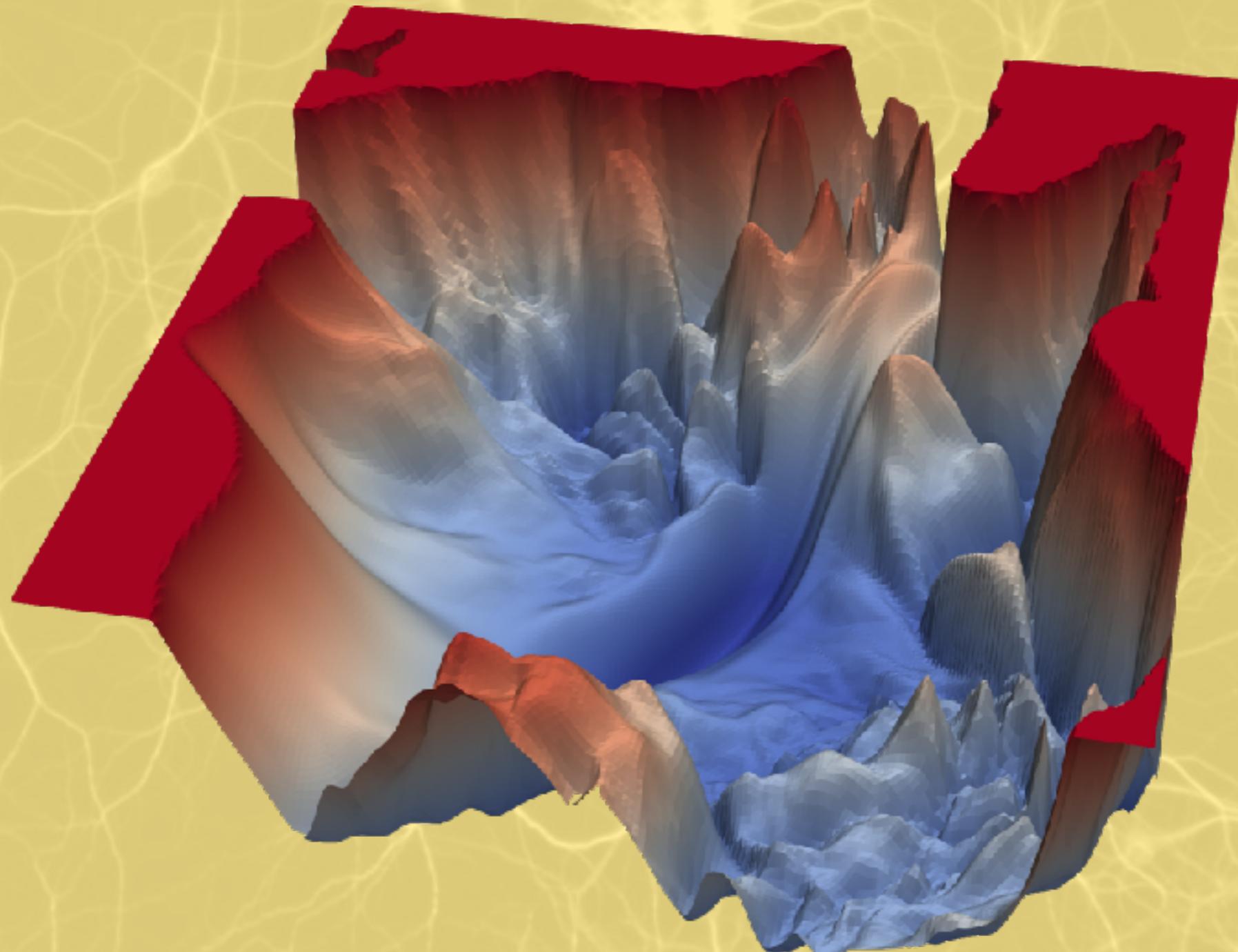


- \* mínimo local (A), meseta (B), oscilaciones (C)
- \* sensibilidad al punto inicial (D y E)

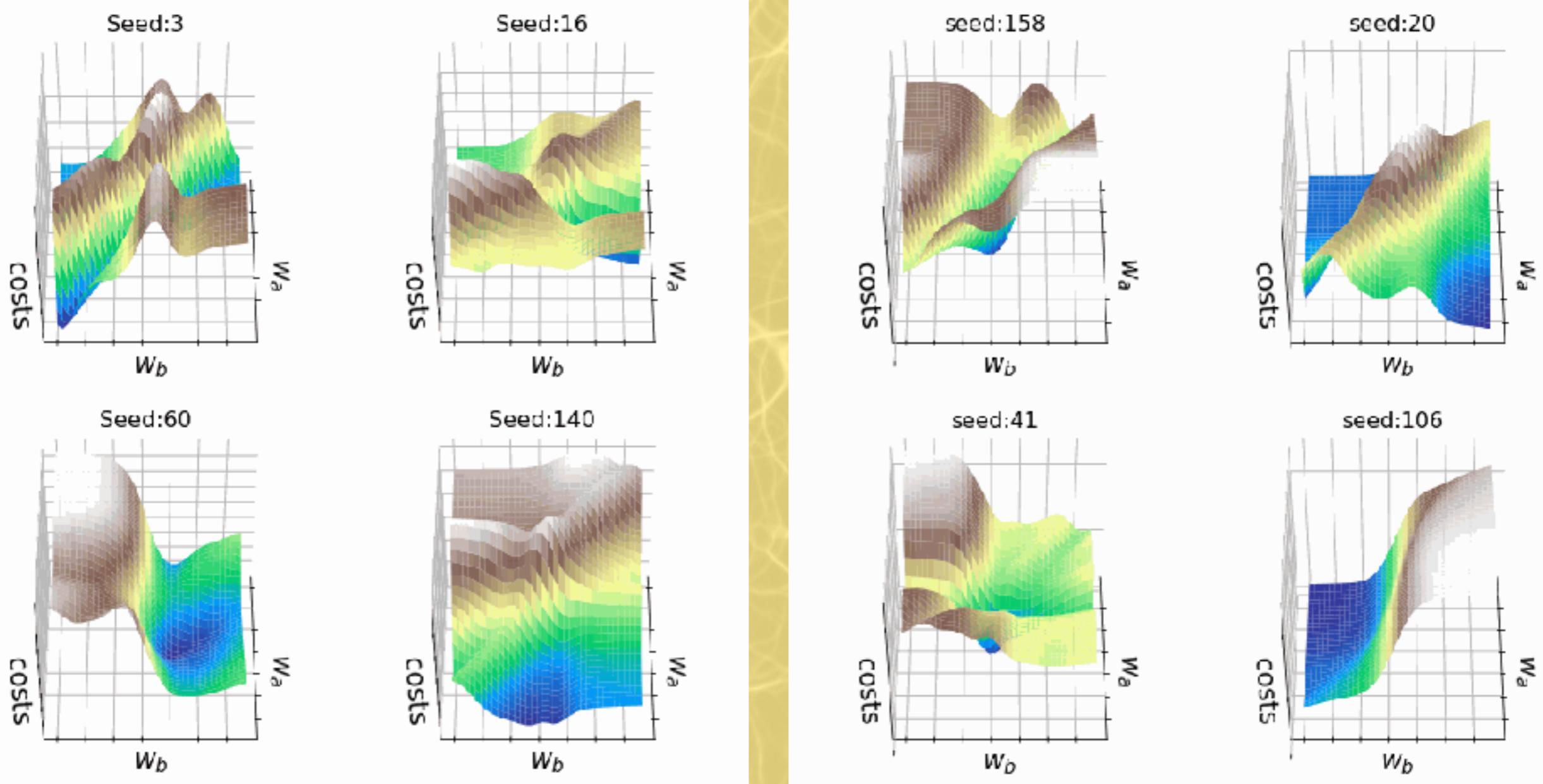
# Problemas del BP



# Espacio de optimización



# Espacio de optimización



# Cálculo del error de la red

## \* Root Mean Square Error

- Se acumula el valor del error cuadrático para cada ejemplo y se divide por el número de ejemplos E, a lo que queda se le hace la raíz cuadrada

## \* Error cuadrático medio global

- En caso de que haya varias salidas, se calcula el RMSE para cada una de ellas y se promedia

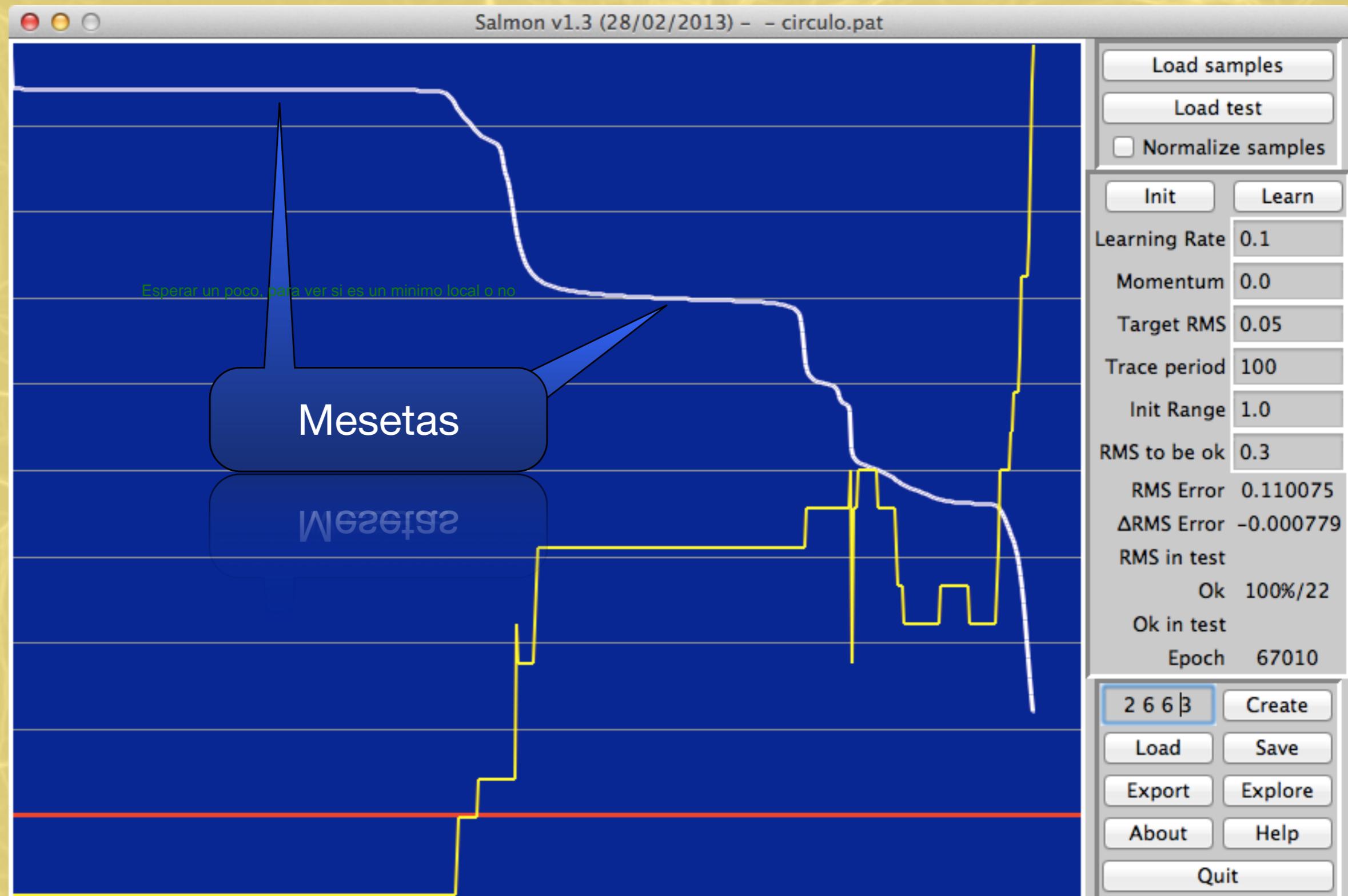
$$RMSE = \sqrt{\frac{1}{E} \sum_{e=1}^E (d_e - s_e)^2}$$

$$GRMSE = \frac{1}{N} \sum_{i=1}^N \sqrt{\frac{1}{E} \sum_{e=1}^E (d_{ei} - s_{ei})^2}$$

# Mínimo Local / red subdimensionada



# Mesetas



# Entrenamiento correcto



# Evaluación del error de un sistema

## \* Tipos de validación

- Con reposición
  - ✗ Excesivamente optimista, nada recomendable
- Cruzada (cross validation)
  - ✗ Puede repetirse N veces promediando el resultado
- Exclusión individual
- Por rotación (K-fold cross validation)

## \* Tipos de error

- Falsos Negativos o Pérdidas: no se predice una clase cuando el individuo es de la clase
- Falsos Positivos o Falsas Alarmas: se predice una clase cuando el individuo no es de la clase

# Error de validación y error de test

- \* Training set: a set of examples used for learning: to fit the parameters of the classifier In the MLP case, we would use the training set to find the “optimal” weights with the back-prop rule
- \* Validation set: a set of examples used to tune the parameters of a classifier In the MLP case, we would use the validation set to find the “optimal” number of hidden units or determine a stopping point for the back-propagation algorithm
- \* Test set: a set of examples used only to assess the performance of a fully-trained classifier In the MLP case, we would use the test to estimate the error rate after we have chosen the final model (MLP size and actual weights) After assessing the final model on the test set, YOU MUST NOT tune the model any further!
- \* Why separate test and validation sets? The error rate estimate of the final model on validation data will be biased (smaller than the true error rate) since the validation set is used to select the final model After assessing the final model on the test set, YOU MUST NOT tune the model any further!
- \* source: Introduction to Pattern Analysis, Ricardo Gutierrez-Osuna, Texas A&M University

# Evaluación del error de un sistema

## \* Casos posibles

- True positives (TP), True negatives (TN), False positives (FP), False negatives (FN)

## \* Medidas de rendimiento

- Accuracy o exactitud

$$\text{X } A = (TP + TN) / (TP + TN + FP + FN)$$

- Precision o tasa de éxito

$$\text{X } P = TP / (TP + FP)$$

- Recall o exhaustividad

$$\text{X } R = TP / (TP + FN)$$

- F1

$$F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}} = 2 \cdot \frac{P \cdot R}{P + R}$$

	Realidad Verdadera	Realidad Falsa
Predicción Verdadera	TP	FP
Predicción Falsa	FN	TN

# Ejercicio

- \* Calcular Accuracy, Precisión, Recall y F1 para la tabla

	Realidad Verdadera	Realidad Falsa
Predicción Verdadera	40	10
Predicción Falsa	20	30

- \* Resultado

- Accuracy

X  $A = (TP + TN) / (TP + TN + FP + FN) = 70 / 100 = 0,7$

- Precision

X  $P = TP / (TP + FP) = 40 / 50 = 0,8$

- Recall

X  $R = TP / (TP + FN) = 40 / 60 = 0,66$

- $F1 = 2 * (0,8 * 0,66) / (0,8 + 0,66) = 0,7233$

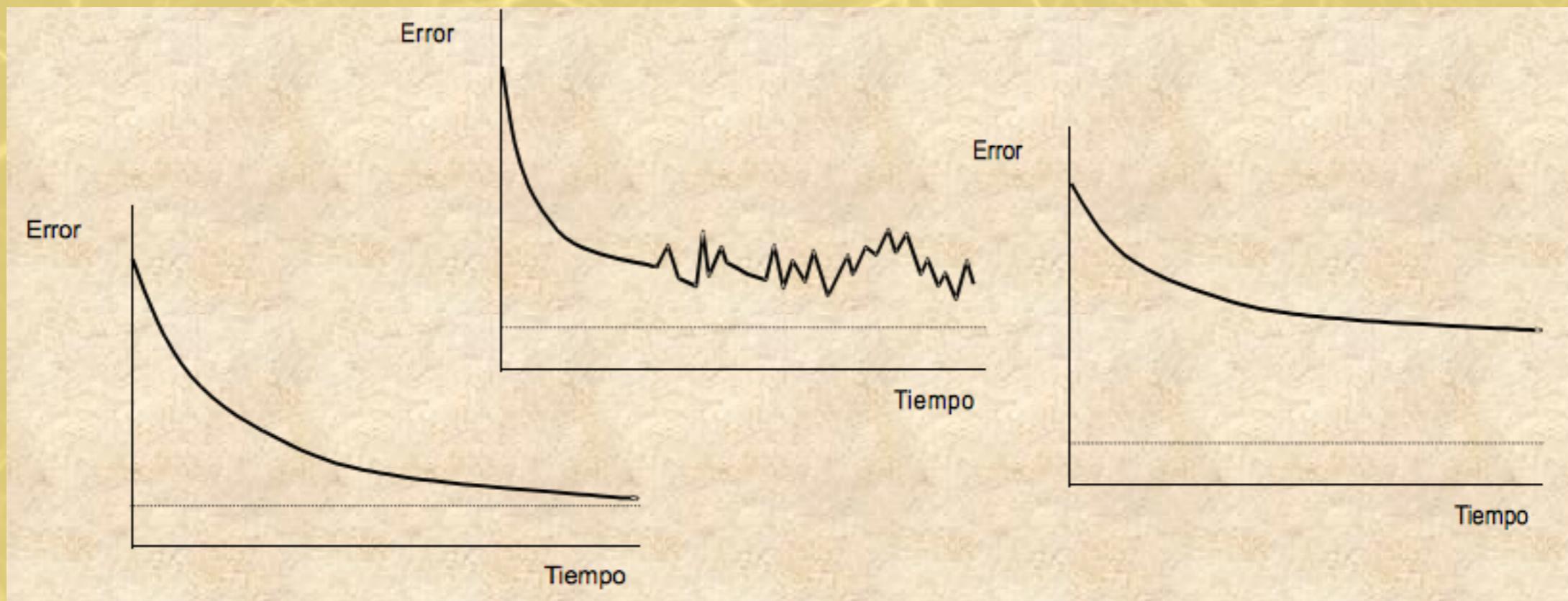
# Evaluación del error (Regresión)

- \* El Error RMS nos da una idea, pero si la varianza de las salidas es baja, un RMS bajo puede no ser significativo, lo que interesa es en cuánto mejora el RMS a la varianza de la variable de salida
  - R<sup>2</sup>, o Varianza Explicada
    - ✖ La varianza de la variable de salida (variable dependiente) se divide en dos: la varianza explicada, que es la que nuestro sistema puede predecir, y la no explicada, que es la que no puede predecir y que origina el error cuadrático (MSE)

$$R^2 = 1 - \frac{var(error)}{var(salida)} = 1 - \frac{MSE}{var(salida)}$$

# Evolución del error al entrenar

Problema	Acción
Mínimo local	Reentrenar
Meseta	Incrementar $\alpha$ . Usar momento
Oscilaciones	Disminuir $\alpha$
No aprende	Puede ser por varios motivos



# ¿Por qué no aprende?

- \* Es una meseta
  - Esperar
- \* Es un mínimo local
  - Reentrenar
- \* La red es demasiado pequeña
  - Aumentar el número de neuronas
- \* Los datos no son representativos
  - Conseguir nuevas variables

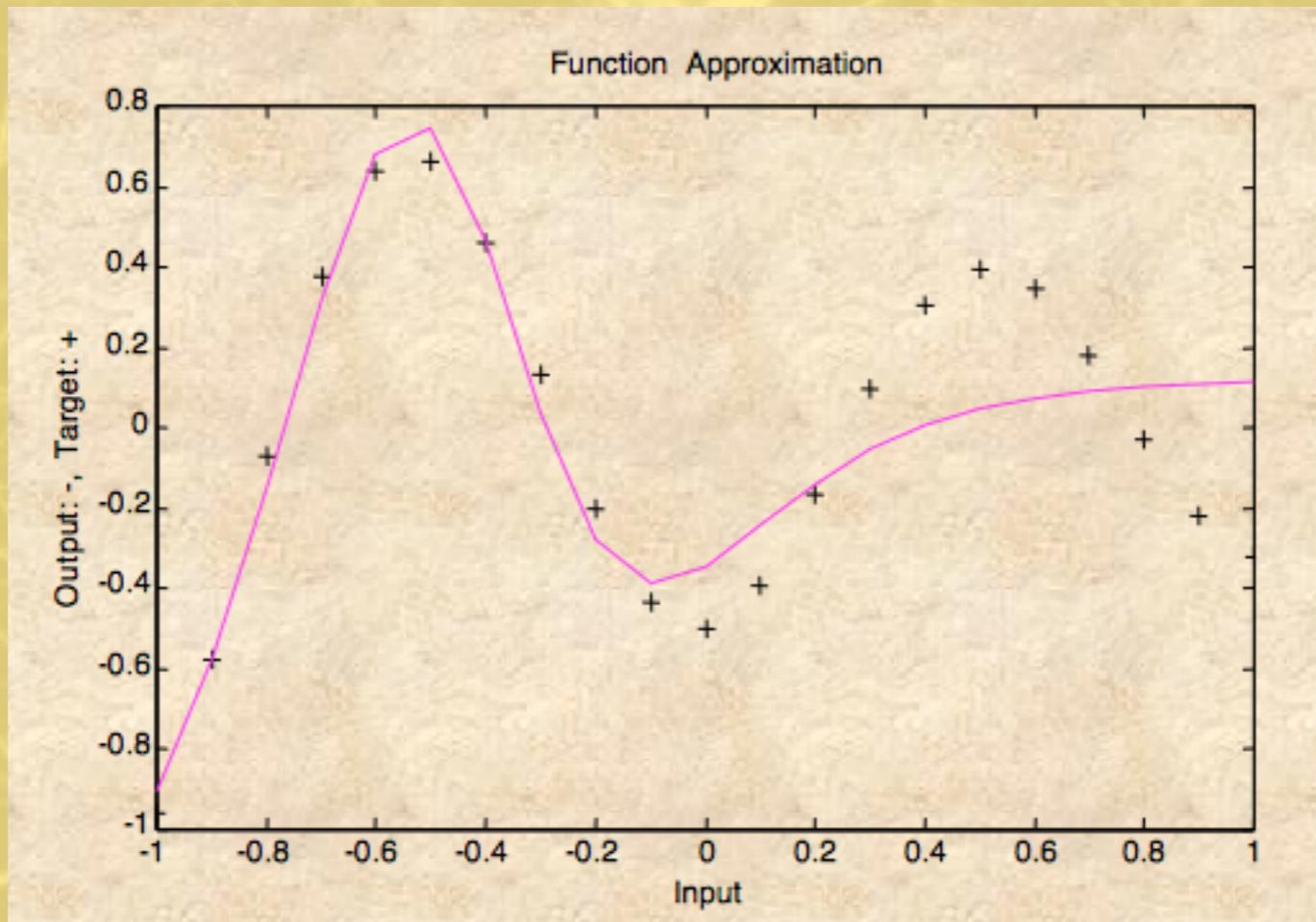
# ¿Por qué lo hace mal en test si entrenó bien?

- \* En este caso sí hay relación entre las entradas y las salidas, puesto que la red aprendió
- \* La red es demasiado grande o se entrenó demasiado (overfitting)
  - Bajar el tamaño de la red
  - Usar métodos de regularización
- \* Tenemos pocos ejemplos
  - Conseguir más ejemplos
  - Incrementar artificialmente el número de ejemplos (data augmentation)
    - ✗ Rotaciones, simetrías
    - ✗ Añadir ruido

una red con muchas neuronas tiende a prenderse los ej de memoria

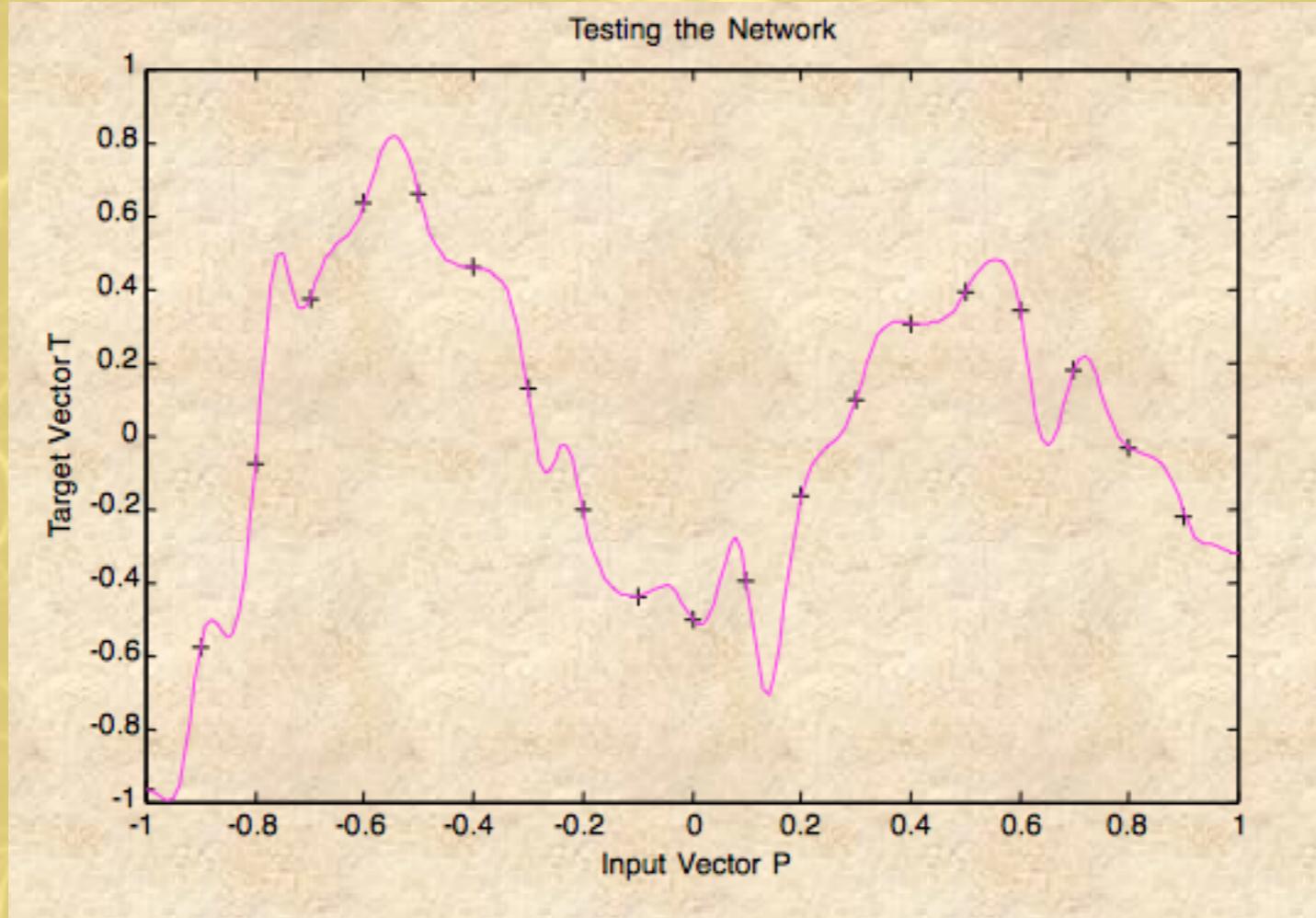
no sirve de mucho

# Underfitting



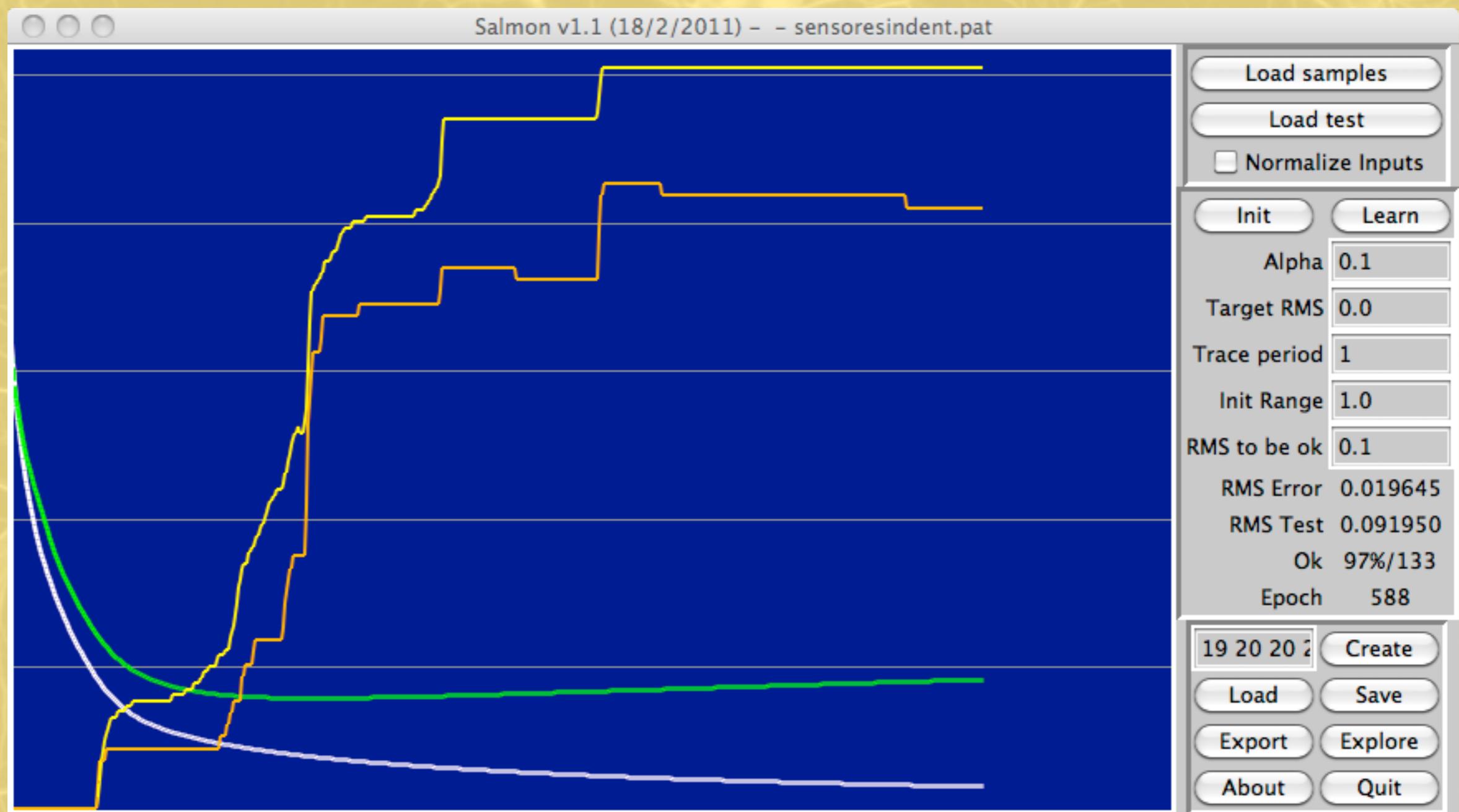
- \* La red no es capaz de aproximarse suficientemente a los datos

# Overfitting

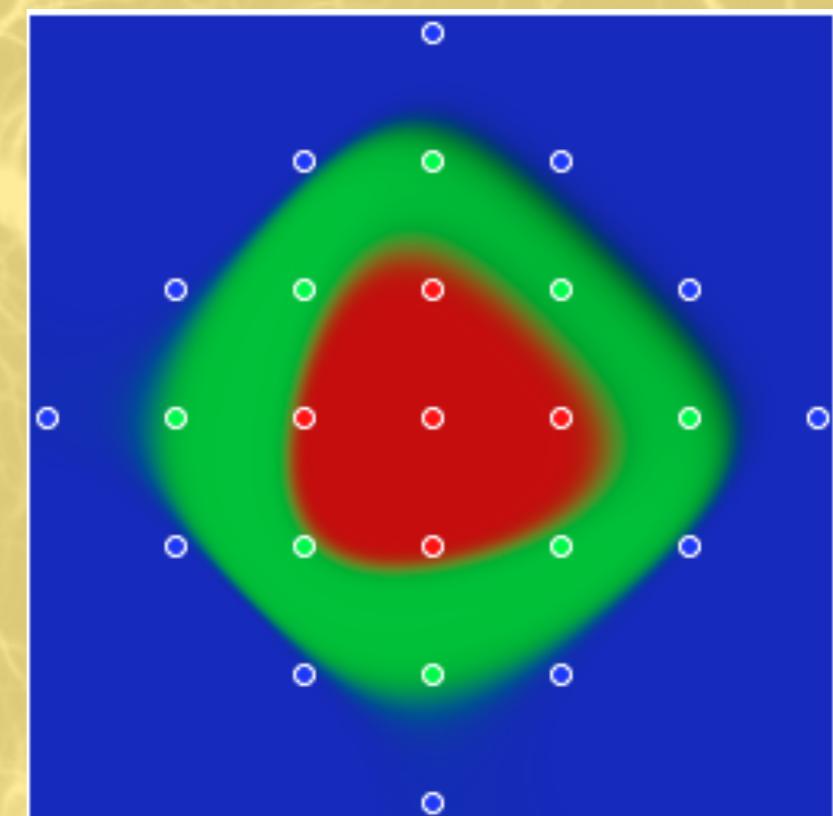
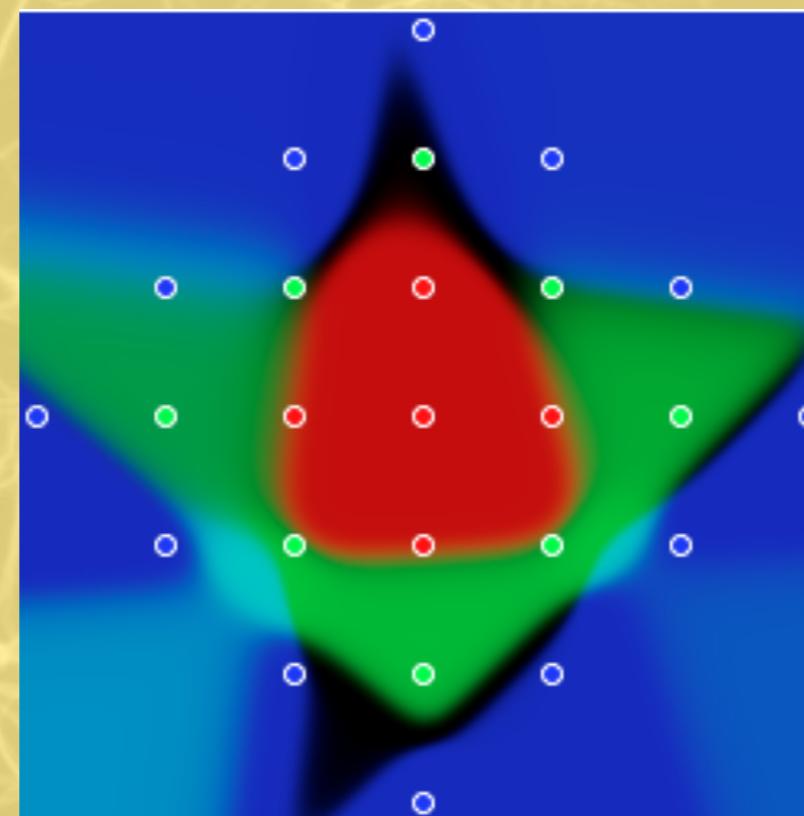
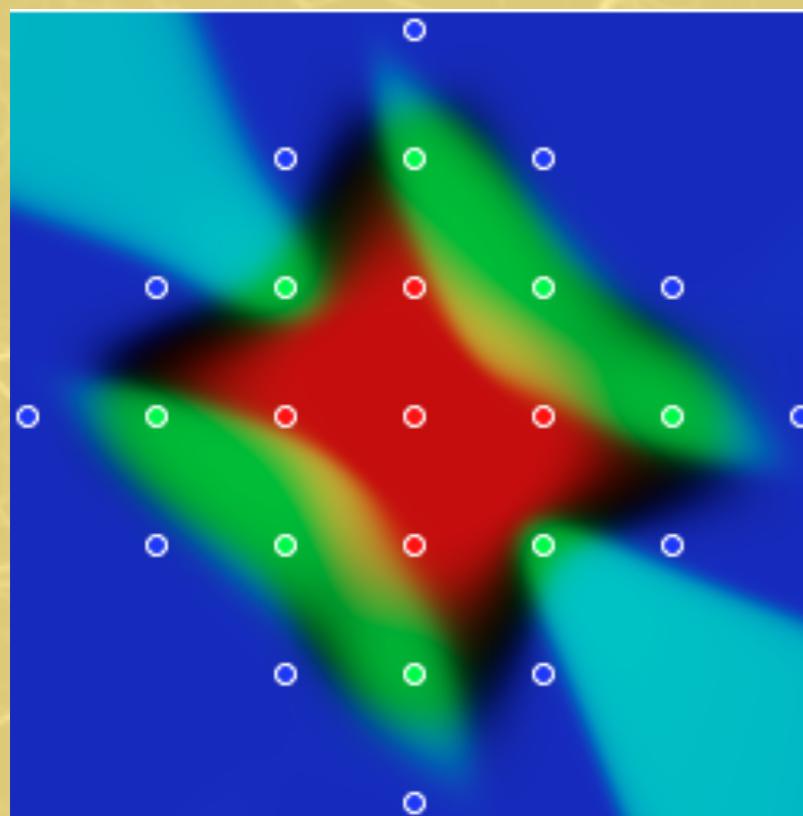


- \* La red interpola todos los datos de entrenamiento, pero no generaliza bien, lo hace mal con los datos de test

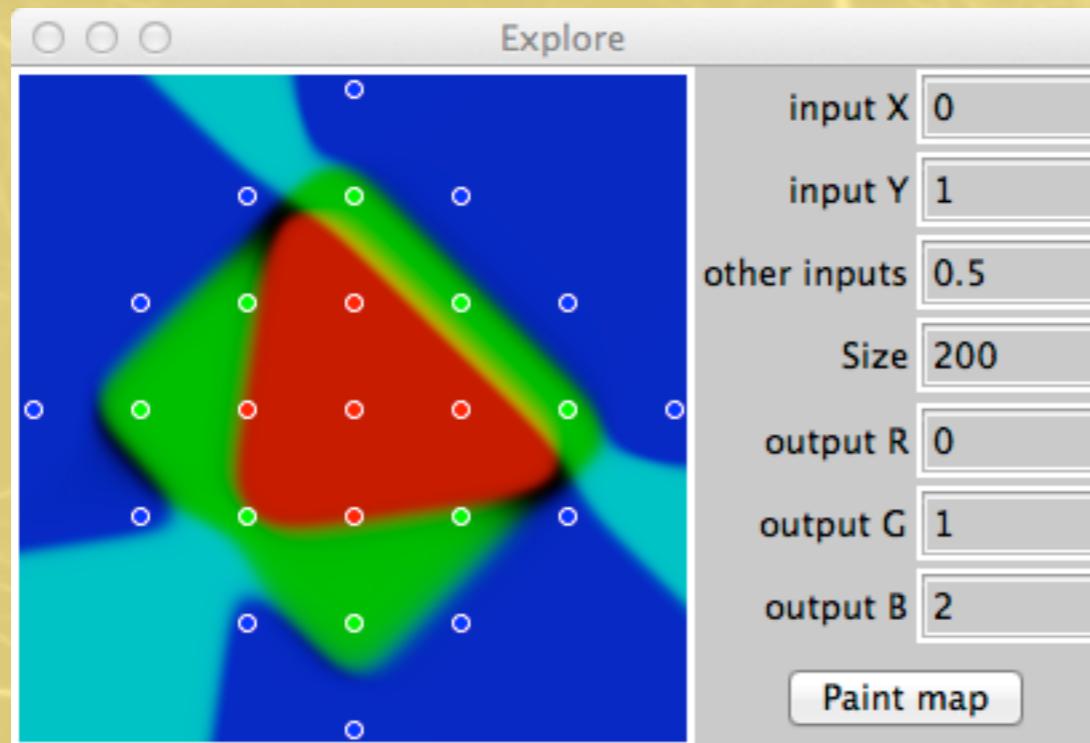
# Generalización y evolución del error de test



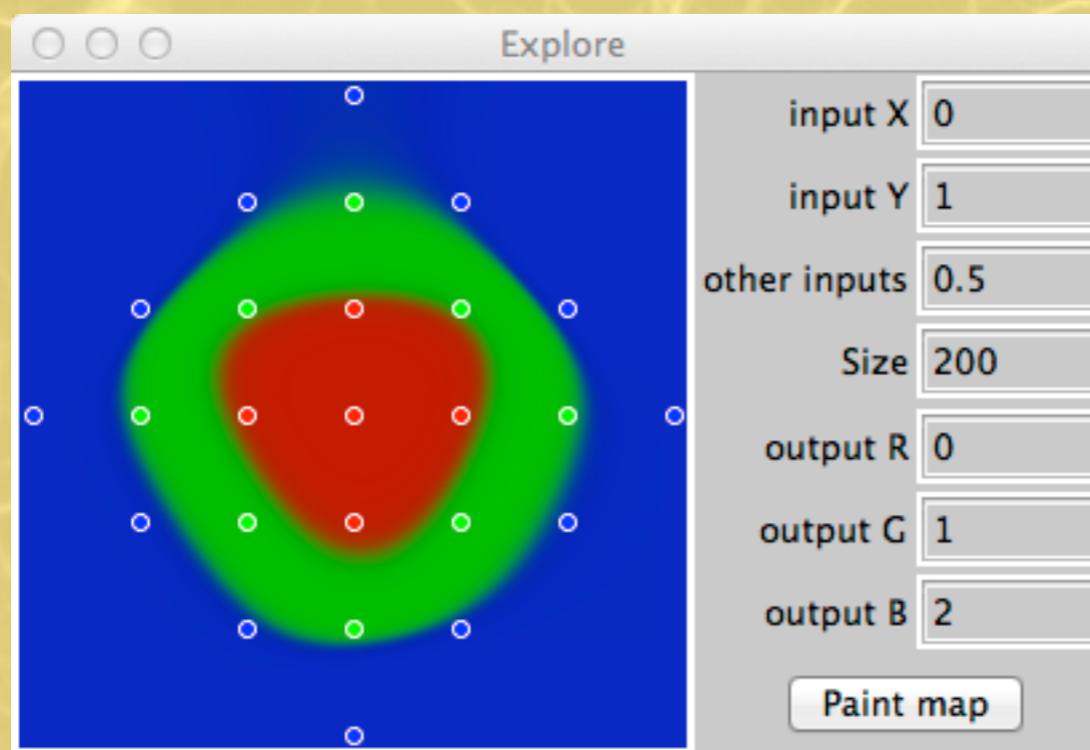
# Underfitting y overfitting



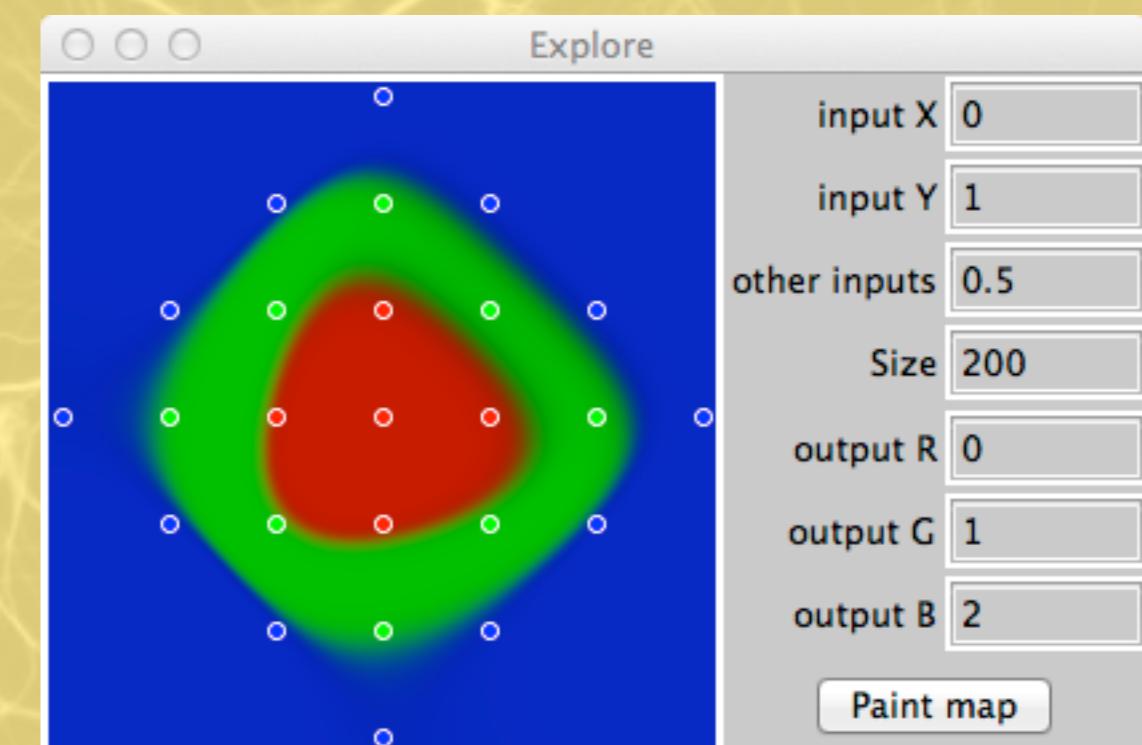
# Generalización y capas ocultas



1 capa oculta



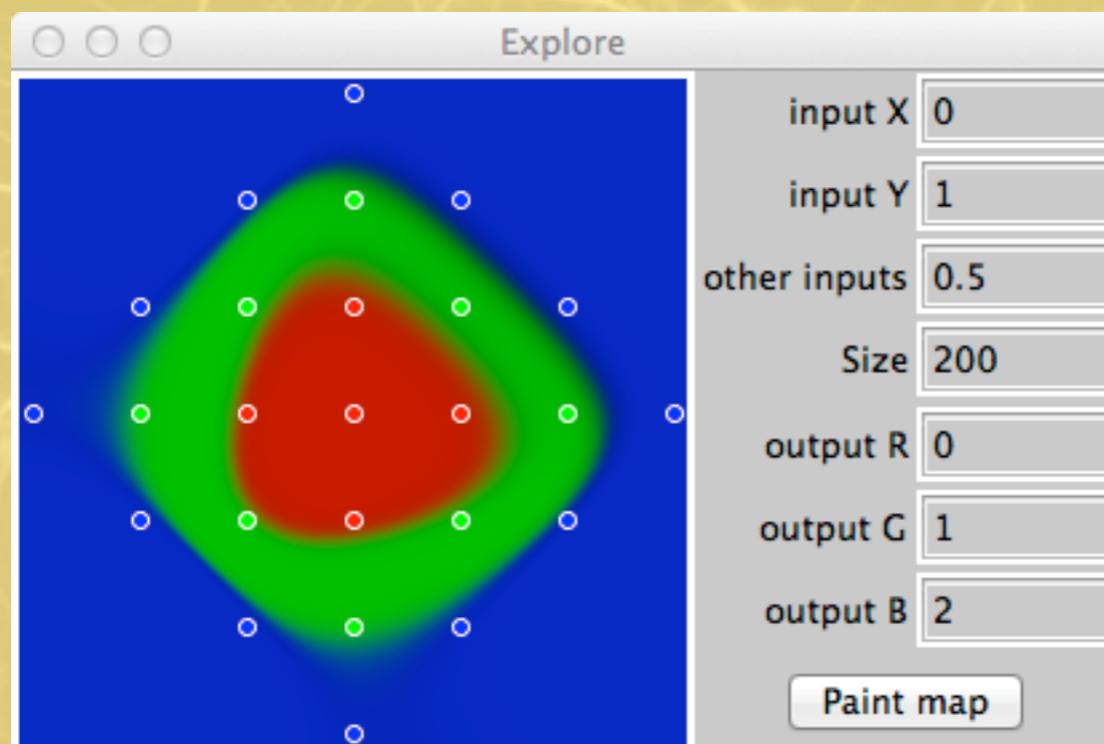
2 capas ocultas



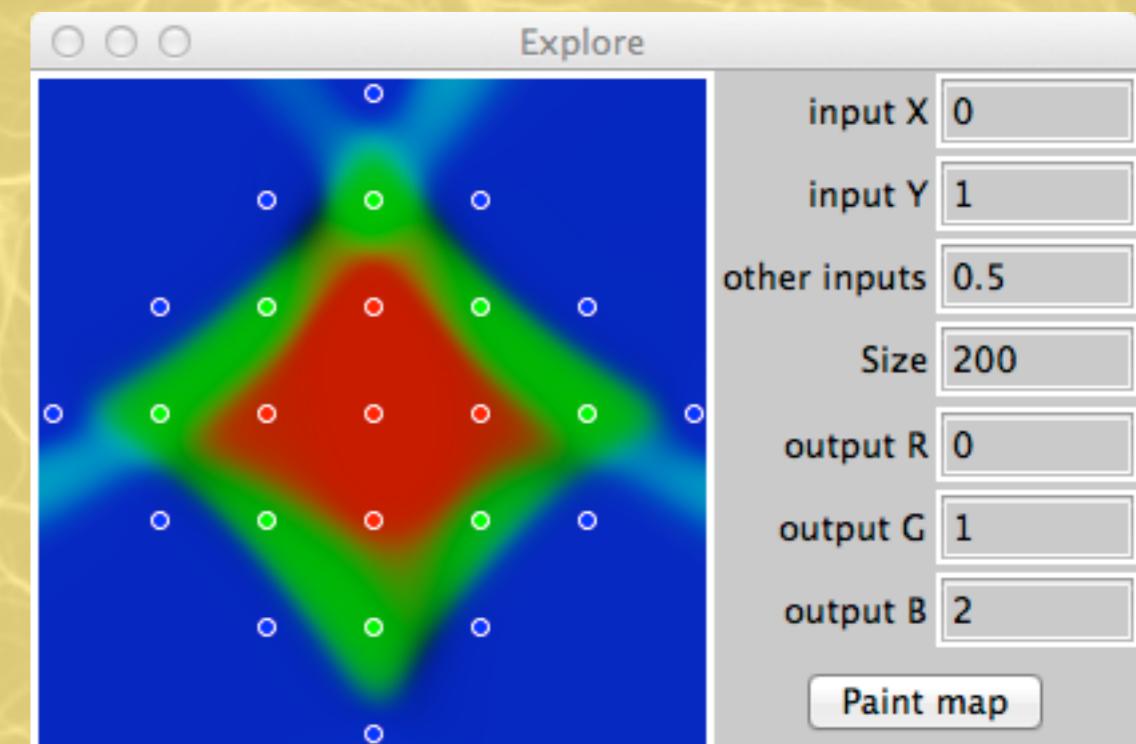
3 capas ocultas

# Generalización y número de neuronas

- \* Con muchas neuronas, la red es más potente, aprende en menos epochs, pero...



Pocas neuronas



Muchas neuronas

# Consideraciones Prácticas

## \* De los datos

- Rango válido de valores de entrada y de salida
- Hay que convertir las variables simbólicas en numéricas
  - ✓ Ordinales
  - ✗ No ordinales (codificación dummy o one-hot)
- Cada clase debe tener un número similar de ejemplos
- Es conveniente que los ejemplos estén aleatorizados
- Análisis de sensibilidad permite evaluar las variables de entrada
- Se pueden usar casos hipotéticos

# Consideraciones Prácticas (II)

## \* De la Arquitectura de la red

- Generalización y memoria. Mínimo número posible neuronas en las capas internas. Regularización
- Elección de número de entradas y salidas es obvio
- Capas ocultas en problemas de clasificación [Gibson & Cowan, 1990. *On the decision regions of multilayer perceptrons*]:
  - ✖ 0 → clasificador lineal
  - ✖ 1 → clasificador no lineal con regiones (disjuntas) convexas simples
  - ✖ 2 → regiones (disjuntas) con forma arbitraria
  - ✖ Añadir más de 2 capas ocultas no añade más potencia de clasificación

# Proceso de diseño de una solución

- \* Obtener datos de entrenamiento
- \* Preparar datos (eliminar datos con errores, normalizar, seleccionar relevantes)
- \* Codificar datos: una red sólo admite datos numéricos como entrada y salida; si no son numéricos...
  - Si son ordinales (tienen un orden implícito) se asignan valores numéricos
  - Si no son ordinales: codificación dummy (one-shot en tensorflow)
    - ✖ Se desglosa en una variable por cada posible valor, siendo 1 cuando se da ese valor y 0 en las demás
- \* Elegir una arquitectura (de más simple a más compleja)

# Proceso de diseño de una solución

- \* Si el error de entrenamiento se estanca (de menos grave a más grave)
  - Meseta, mínimo local
  - Red subdimensionada
  - Variables no caracterizan bien el problema
- \* Si el error de test es mucho más alto que el de entrenamiento
  - Red demasiado potente
  - Pocos ejemplos: añadir ejemplos, reales o artificiales (simetrías, ruido)
  - Ejemplos no caracterizan bien el problema
- \* En la fase de explotación, utilizaremos la red para propagar las nuevas entradas. Las salidas serán las respuestas del sistema

# Metodología

- \* Entrenar un perceptrón sin capa oculta
  - Idea de las posibilidades con métodos clásicos
  - Medida de la linealidad del problema
  - Es un RMS en test a batir con métodos más potentes
- \* Añadir neuronas de 10 en 10 y analizar si mejora el error de test
  - Posiblemente en cierto momento empeorará, aunque mejore el de entrenamiento
- \* Añadir una capa oculta manteniendo un número de conexiones similar al de la fase anterior
- \* Hacer alguna prueba con muchas capas ocultas a ver qué pasa (Deep Learning)

# Expansión del espacio de entrada

## \* Temporales

- Consiste en añadir los valores de las variables en  $t-1$ ,  $t-2$ , etc. o bien directamente sus incrementos  $e(t) - e(t-1)$

## \* Aritméticas

- Consiste en añadir nuevas variables de entrada que son operaciones aritméticas de las reales

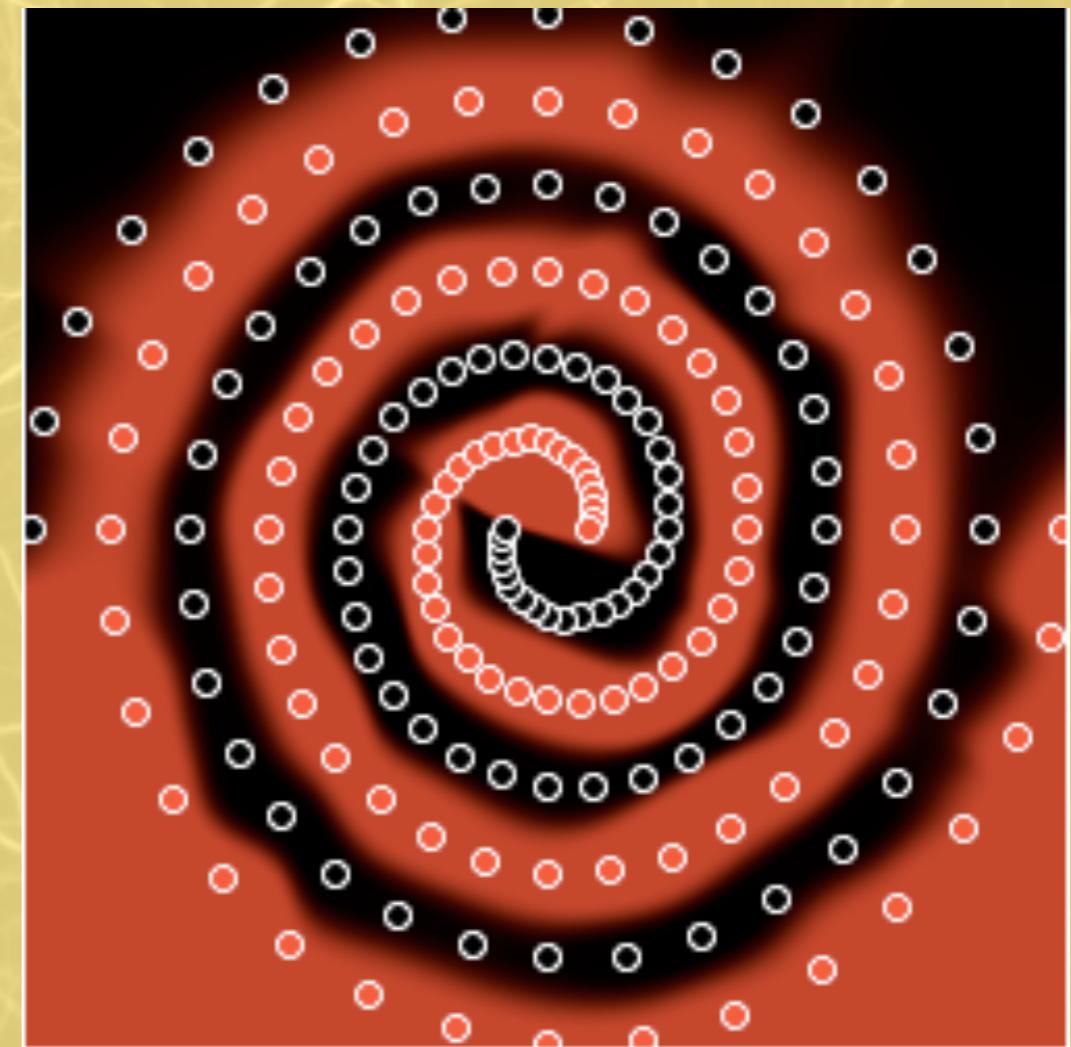
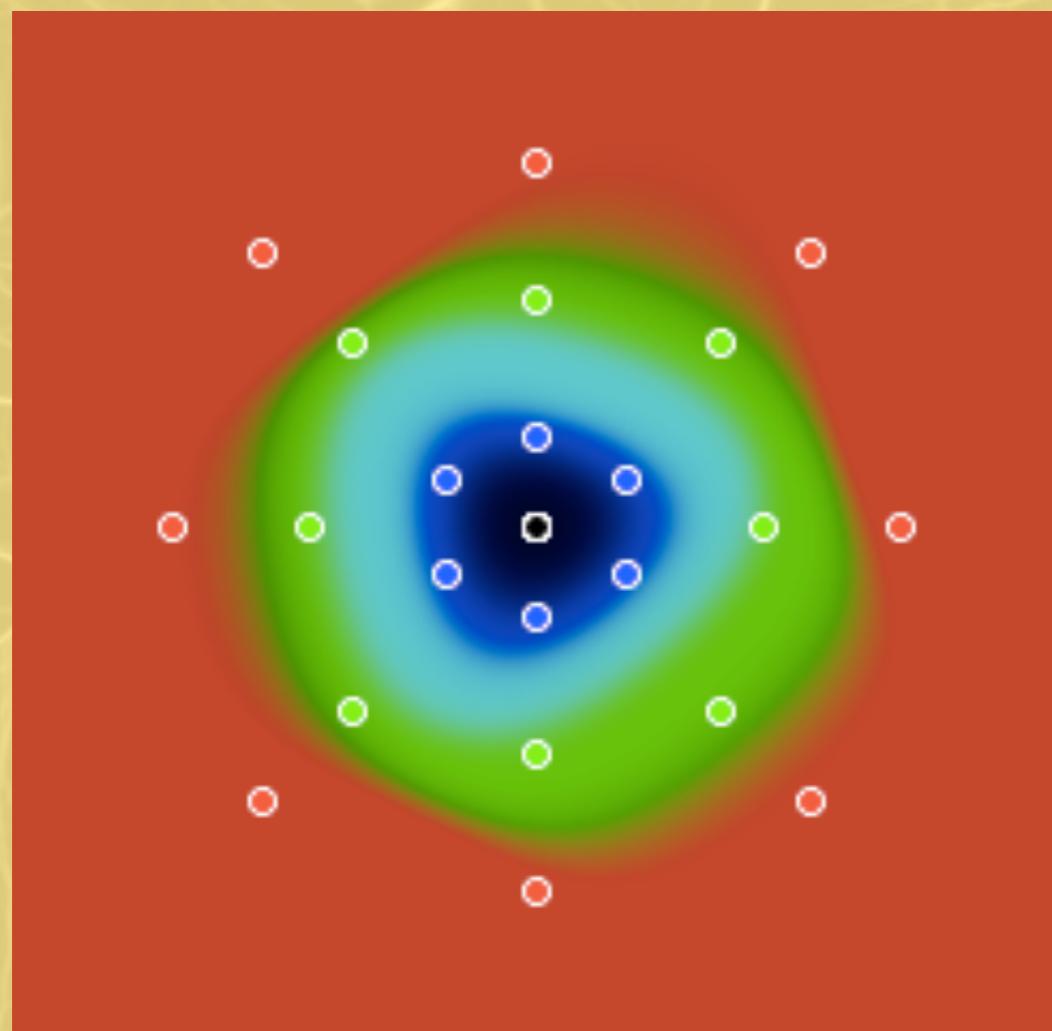
✗  $e_i * e_j$

✗  $\sin(e_i)$

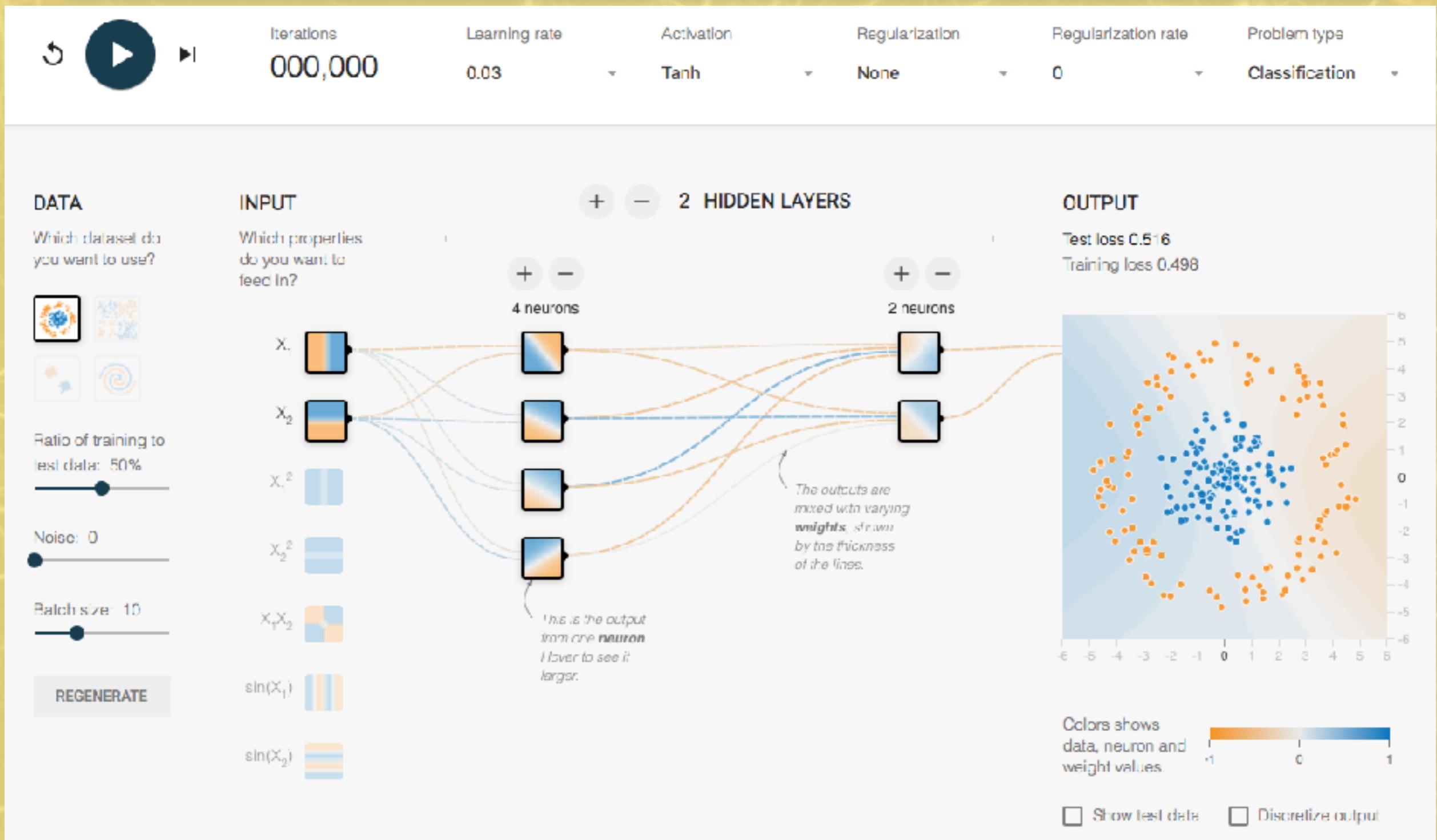
✗  $e_i^2$

✗ etc.

# Regiones complejas



# PlayGround



- \* Sistema demostrativo de la librería TensorFlow de Google
- <http://playground.tensorflow.org>

# Probar

- \* Entrenar la red que sale por defecto, analizar cómo es esa red
- \* Quitar una neurona, reestrenar, quitar otra neurona
- \* Cambiar el ejemplo a uno linealmente separable
  - ¿Debería aprender con 2 neuronas ocultas? ¿Y sin capa oculta?
- \* Cambiar a ejemplo tipo XOR
  - ¿Debería aprender sin capa oculta? ¿Qué arquitectura mínima aprende?
- \* Cambiar a ejemplo espiral
  - ¿Qué arquitectura mínima aprende?

# Deep Learning

- \* Aunque la idea generalizada era que añadir más capas no era útil, algunos investigadores estudiaron esta posibilidad durante el periodo 1990-2010
  - El más conocido es Yann LeCun, que partió del neocognitrón (Fukushima) e intentó dotarlo de aprendizaje
  - A partir de 2008 empiezan a obtenerse éxitos importantes en este campo, debido a varias innovaciones
    - ✖ Neuronas ReLU
    - ✖ Redes de convolución: pesos compartidos, pool layers
    - ✖ Autocorreladores, Deep Belief Networks
    - ✖ Redes LTSM (Long Short Term Memory)
    - ✖ Redes GAN (Generative Adversarial Network)
  - En la actualidad, todos los gigantes tecnológicos están utilizando Deep Learning para diversos propósitos