CI5438. Inteligencia Artificial II
Clase 6: Redes Neuronales
Cap 20.5 Russel & Norvig
Cap 4 Mitchell

Ivette C. Martínez

Universidad Simón Bolívar

14 de Octubre de 2009

Modelos Conexionistas

Consideremos los seres humanos:

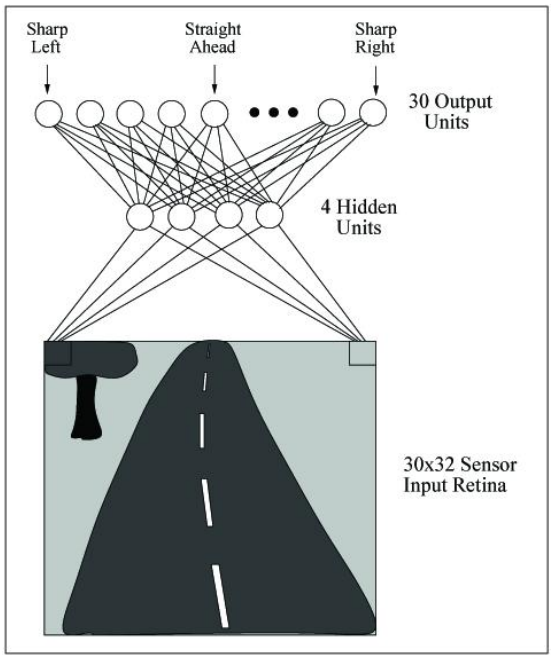
- Tiempo de activación de las neuronas $\approx 0{,}001$ segundos
- Número de neuronas $\approx 10^{10}$
- Conexiones por neurona $\approx 10^{4-5}$
- Tiempo para reconoces una escena $\approx 0{,}1$ segundos
- 100 pasos de inferencia no parecen ser suficientes

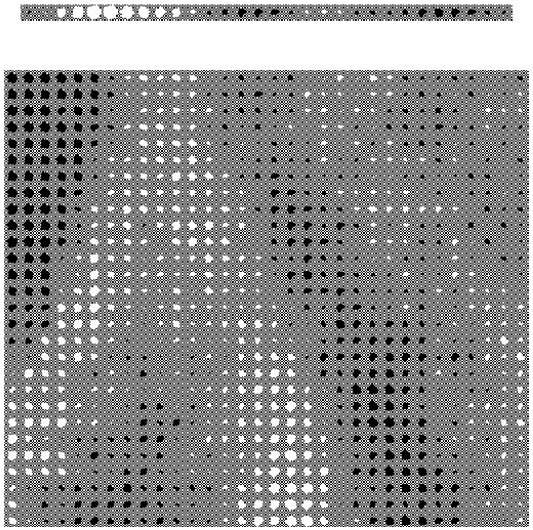
$\longrightarrow$ Muchos cómputos paralelos

Propiedades de las redes neurales artificiales:

- Muchas unidades de activación de umbral, como "neuronas".
- Muchas interconecciones pesadas entre las unidades
- Procesamiento distribuido, altamente paralelo
- Énfasis en la entonación automática de pesos.

ALVINN


Sharp
Left
Straight
Ahead
Sharp
Right
30 Output
Units
4 Hidden
Units
30x32 Sensor
Input Retina



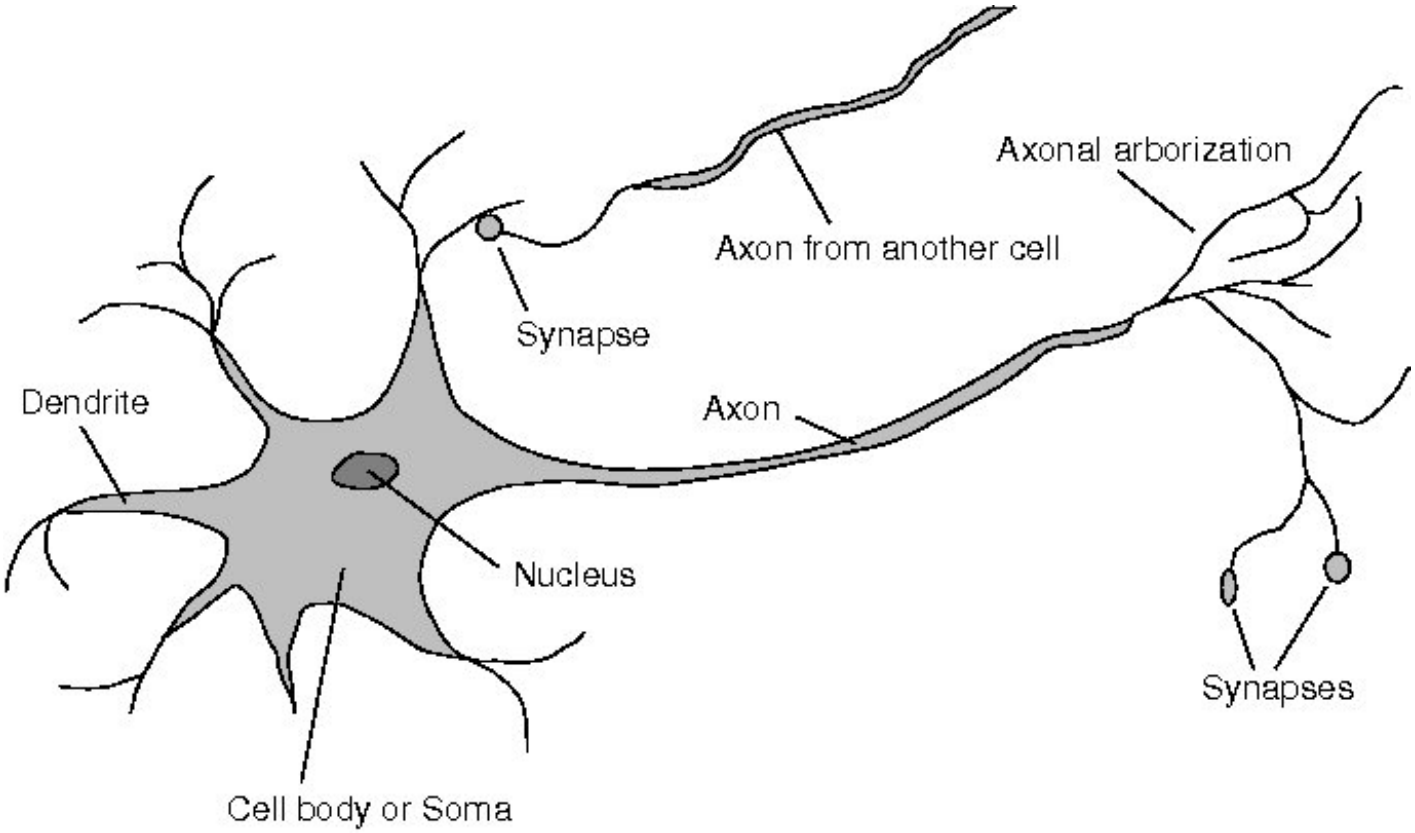
Cuándo considerar Redes Neurales

- La entrada es de muchas dimensiones, de valores discretos o reales (e.j. Entradas de un sensor)
- La salida es de valores discretos o contínuos
- La salida es un vector de valores
- Los datos son posiblemente ruidosos
- La forma de la fución objetivo es desconocida
- La legibilidad de los resultados por parte de los humanos no tiene importancia

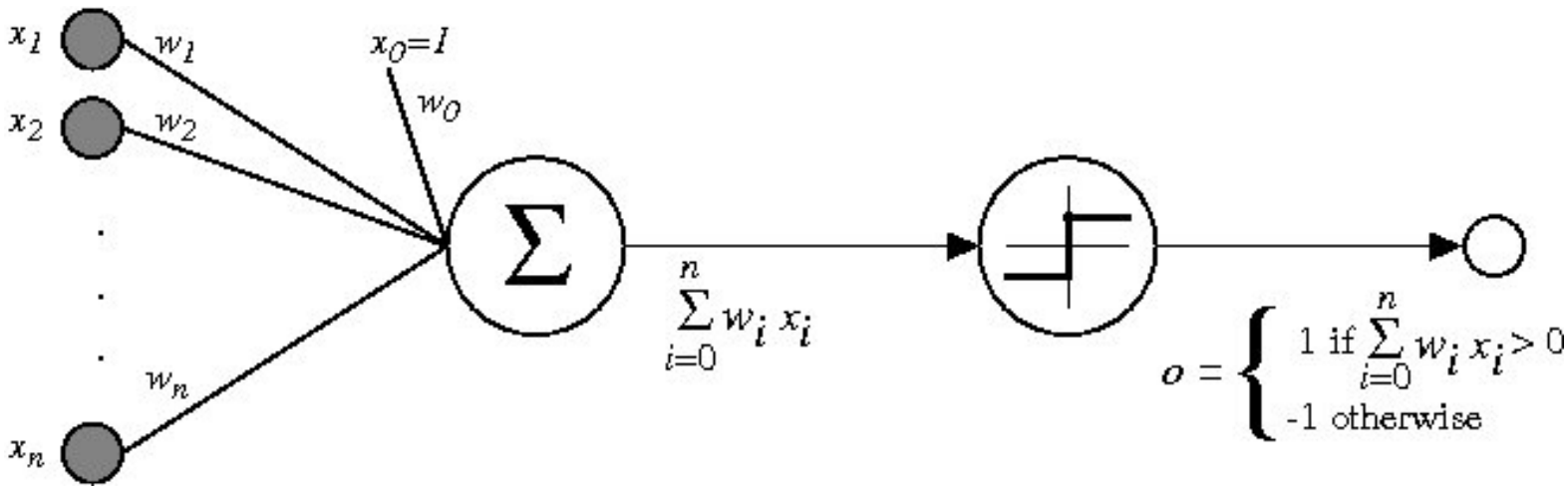
Ejemplos:

- Reconocimiento de fonémas hablados [Waibel]
- Clasificación de imágenes [Kanade, Baluja, Rowley]
- Predicción financiera

Neurona


Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma

Perceptrón

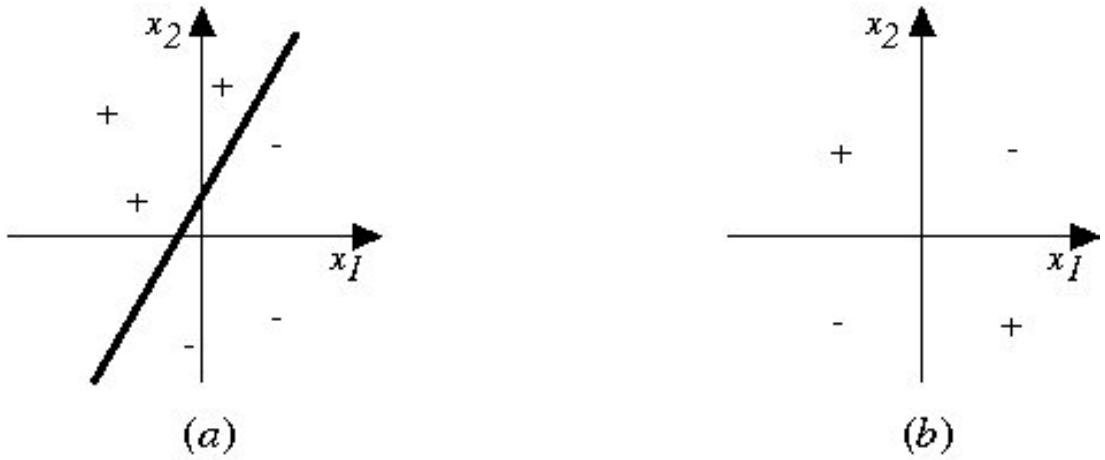


$$o(x_1, ..., x_n) = \begin{cases} 1 & \text{if } w_o + w_1 x_1 + ... + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

Algunas veces usamos la notación vectorial, que es más simple:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w}.\vec{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

Superficies de Decisión de un Perceptrón



Se pueden representar algunas funciones útiles

- Qué pesos representan a

$$g(x_1, x_2) = AND(x_1.x_2)$$

Pero algunas funciones no son representables

- ejem., Funciones que no sean separable linealmente
- Luego, vamos a necesitar redes de estas unidades

Regla de entrenamiento del Perceptrón

$$w_i \leftarrow w_i + \Delta w_i$$

donde:

$$\Delta w_i = \eta(t - o)x_i$$

Donde:

- $t = c(\vec{x})$ es el valor objetivo
- o es la salida del perceptrón
- η es una constante pequeña (ejem., 0,1) denominada tasa de aprendizaje

Regla de entrenamiento del Perceptrón

Se puede demostrar que converge:

- Si los datos de entrenamiento son linealmente separables
- y η es lo suficientemente pequeña

Descenso del Gradiente

Para entender, consideremos la unidad lineal más simple, donde:

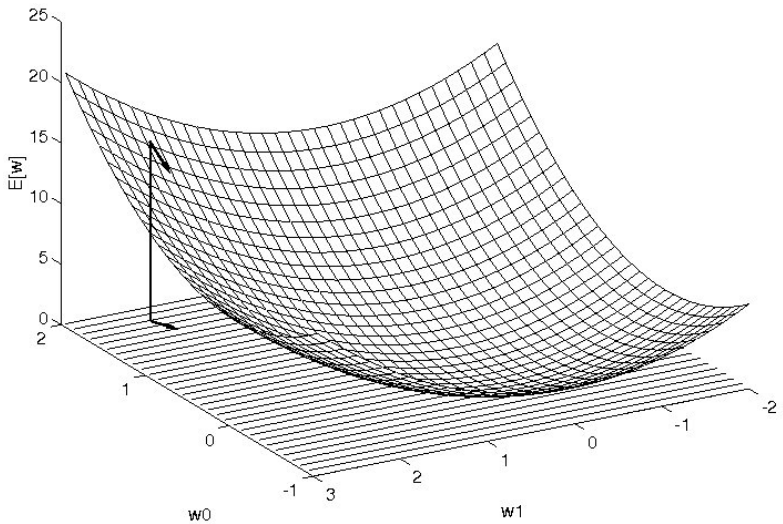
$$o = w_o + w_1 x_1 + ... + w_n x_n$$

Vamos a aprender los w_i que minimicen el error cuadrático

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Donde D es el conjunto de ejemplos de entrenamiento

Descenso del Gradiente



$$\nabla E[\vec{w}] \equiv \left[\frac{\delta E}{\delta w_0}, \frac{\delta E}{\delta w_1}, \cdots \frac{\delta E}{\delta w_n} \right]$$

Regla de entrenamiento:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

es decir:

$$\Delta w_i = -\eta \frac{\delta E}{\delta w_i}$$

Descenso del Gradiente

$$
\begin{aligned}
\frac{\delta E}{\delta w_i} &= \frac{\delta}{\delta w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\delta}{\delta w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\delta}{\delta w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\frac{\delta}{\delta w_i}(t_d - \vec{w}.\vec{x_d}) \\
\frac{\delta E}{\delta w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}
$$

Descenso del Gradiente

GRADIENT-DESCENT($training\_examples, \eta$)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, *where* $\vec{x}$ *is the vector of input values, and* t *is the target output value.* η *is the learning rate (e.g., .05).*

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i$$

Resumen

La regla de entrenamiento del perceptrón tiene un éxito garantizado si:

- Los ejemplos de entrenamiento son linealmente separables
- La tasa de aprendizaje η es suficientemente pequeña

La regla de entrenamiento de unidades lineales usando descenso del gradiente:

- Garantiza converger a la hipótesis con el menor error cuadrático
- Dada una tasa de aprendizaje η es suficientemente pequeña
- Aún cuando los datos de entrenamiento contengan ruido
- Aún cuando los datos de entrenamiento no sean separable por H

Descenso del Gradiente Incremental (Estocástico)

Batch mode Gradient Descent:
Do until satisfied

1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

Incremental mode Gradient Descent:
Do until satisfied

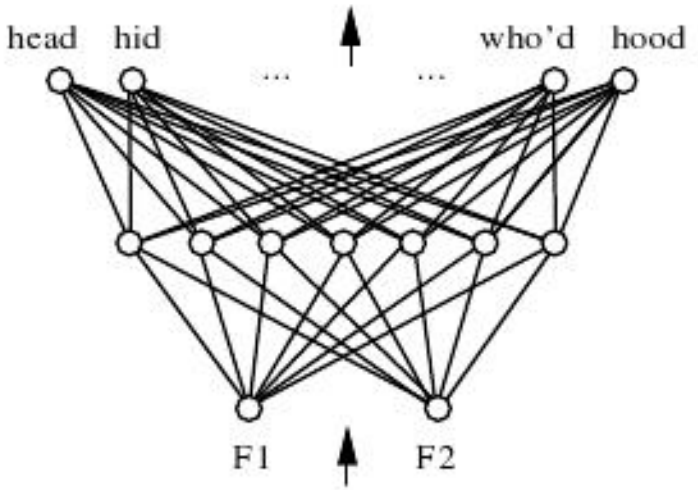
- For each training example d in D
  1. Compute the gradient $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

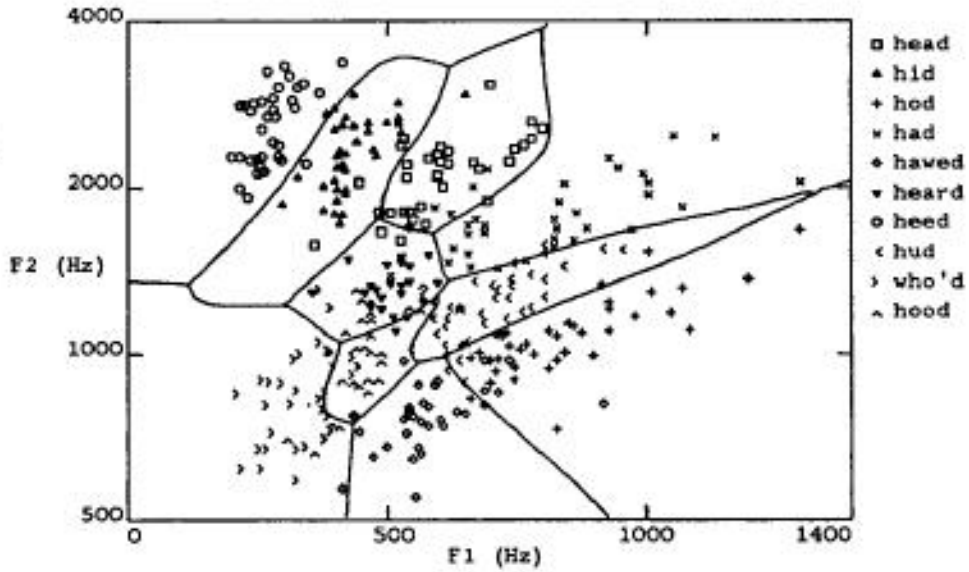
$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$

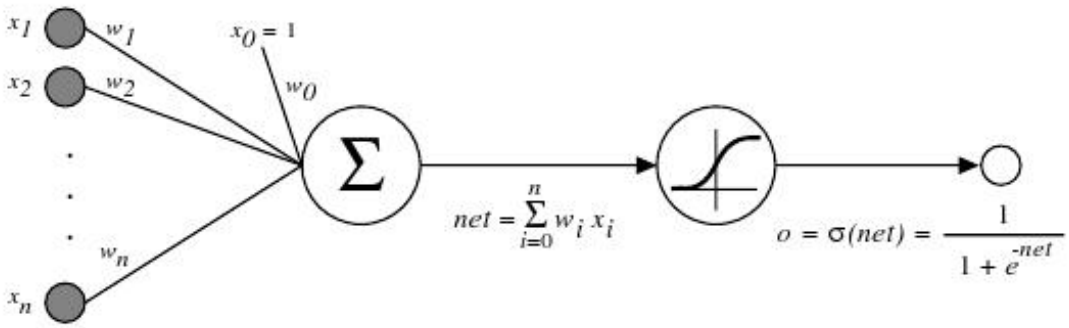
Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough

Redes Multicapas de Unidades Sigmoidales





Unidades Sigmoidales



$\sigma(x)$ es la función sigmoidal

$$\frac{1}{1 + e^{-x}}$$

Buena propiedad: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$
Podemos derivar reglas del descenso de gradiente para entrenar:

- Una unidad sigmoidal
- *Redes multicapa* de unidades sigmoidales $\longrightarrow$ Backpropagation

Gradiente del error para una Unidad Sigmoidal

$$\begin{aligned}
\frac{\delta E}{\delta w_i} &= \frac{\delta}{\delta w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\delta}{\delta w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\delta}{\delta w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\left(-\frac{\delta o_d}{\delta w_i}\right) \\
&= -\sum_d (t_d - o_d)\frac{\delta o_d}{\delta net_d}\frac{\delta net_d}{\delta w_i}
\end{aligned}$$

Pero sabemos que:

$$\frac{\delta o_d}{\delta net_d} = \frac{\delta \sigma(net_d)}{\delta net_d} = o_d(1 - o_d)$$

$$\frac{\delta net_d}{\delta w_i} = \frac{\delta(\vec{(}w).\vec{(}x_d))}{\delta w_i} = x_{i,d}$$

Entonces:

$$\frac{\delta E}{\delta w_i} = -\sum_d (t_d - o_d)o_d(1 - o_d)x_{i,d}$$

Algoritmo de Backpropagation

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

  3. For each hidden unit h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

  4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

  where

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

Más sobre Backpropagation

- Descenso del gradiente sobre el vector de pesos de la red completo
- Fácilmente generalizable para grafos dirigidos arbitrários
- Se encontrará un mínimo local del error, no necesariamente el mínimo error global
  - En la práctica, casi siempre funciona bien (se pueden realizar múltiples corridas)
- Algunas veces se incluye un *momentun* de los pesos

$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

- Minimiza el error sobre los ejemplos de **entrenamiento**
  - Generalizará bien para ejemplos posteriores?
- El entrenamiento puede tomar miles de iteraciones $\rightarrow$ lento!
- El uso de la red despues del entrenamiento es muy rápido