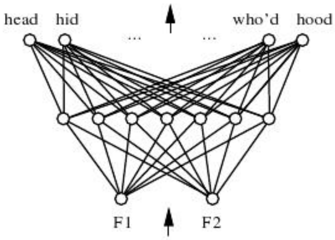
CI5438. Inteligencia Artificial II
Clase 7: Redes Multicapas - Backpropagation
Cap 20.5 Russel & Norvig
Cap 4 Mitchell

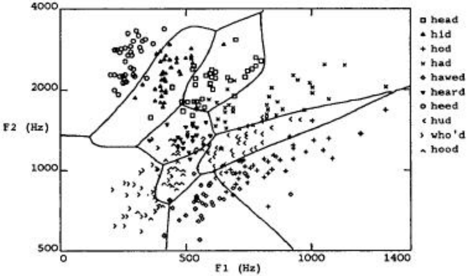
Ivette C. Martínez

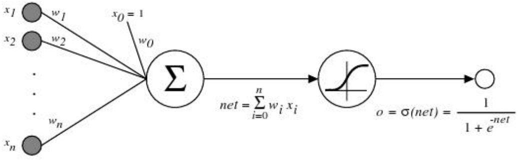
Universidad Simón Bolívar

31 de enero de 2013

Redes Multicapas de 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 $\rightarrow$ 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

Inicializar todos los pesos de forma aleatoria
while NOT Condicion de parada **do**
    for cada ejemplo de entrenamiento e_i **do**
        Calcular la salida de la red (o) para e_i
        for Cada unidad de salida k **do**
            $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
        end for
        for Cada unidad oculta h **do**
            $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{k,h}\delta_k$
        end for
        Actualizar cada peso de la red $w_{j,i}$:
        $w_{j,i} \leftarrow w_{j,i} + \Delta w_{j,i}$,
        donde $\Delta w_{j,i} = \eta \delta_j x_{j,i}$
    end for
end while

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

Ejemplo: Representaciones en las capas ocultas

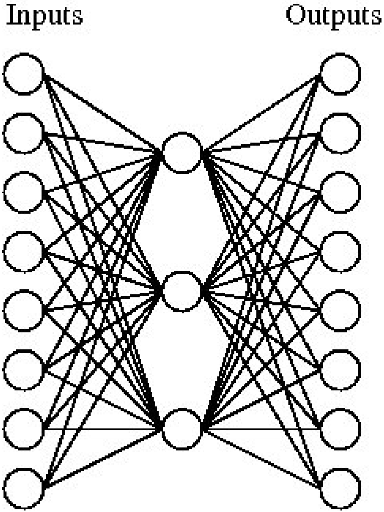
Sea la siguiente función objetivo?

Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

Puede ser aprendida?

Ejemplo: Representaciones en las capas ocultas

Una Red



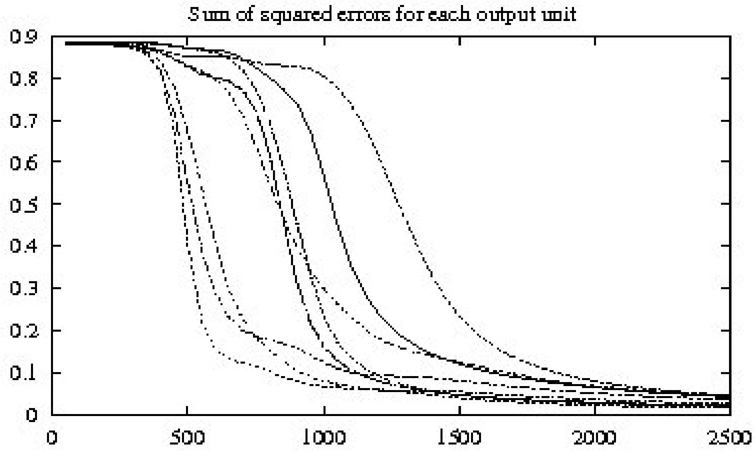
Ejemplo: Representaciones en las capas ocultas

Representación aprendida en la capa intermedia:

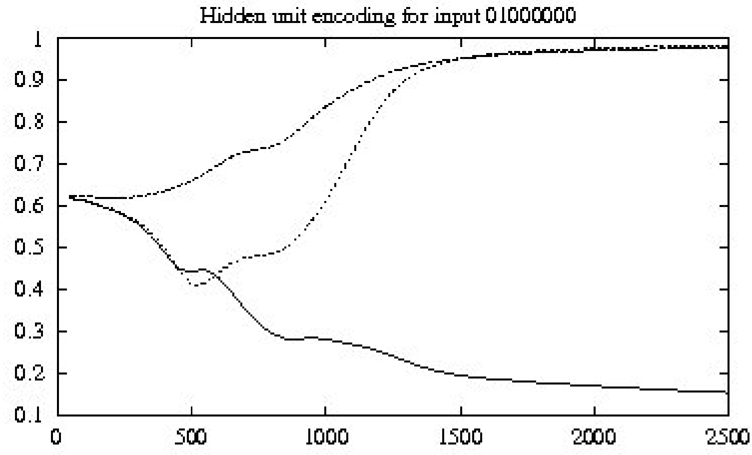
Input		Hidden Values				Output
10000000	$\rightarrow$	.89	.04	.08	$\rightarrow$	10000000
01000000	$\rightarrow$	.01	.11	.88	$\rightarrow$	01000000
00100000	$\rightarrow$	.01	.97	.27	$\rightarrow$	00100000
00010000	$\rightarrow$	.99	.97	.71	$\rightarrow$	00010000
00001000	$\rightarrow$	.03	.05	.02	$\rightarrow$	00001000
00000100	$\rightarrow$	.22	.99	.99	$\rightarrow$	00000100
00000010	$\rightarrow$	.80	.01	.98	$\rightarrow$	00000010
00000001	$\rightarrow$	.60	.94	.01	$\rightarrow$	00000001

Ejemplo: Representaciones en las capas ocultas

Entrenamiento:

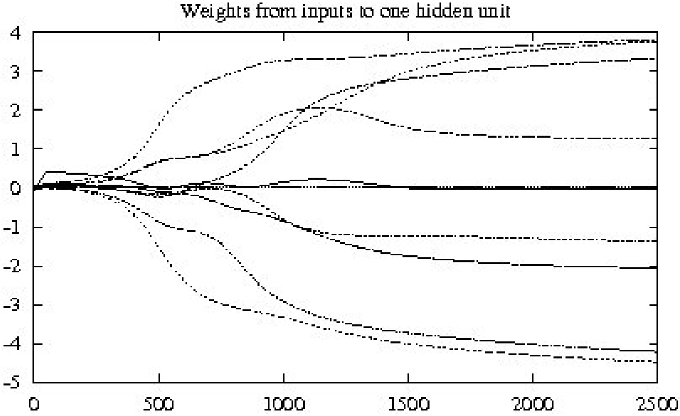


Codificación en la capa intermedia



Ejemplo: Representaciones en las capas ocultas

Pesos de las entradas a una unidad intermedia



Convergencia de Backpropagation

Descenso del gradiente a algún mínimo local

- Quizás no sea un mínimo global...
- Agregar **momentum**
- Descenso del gradiente estocástico
- Entrenar varias redes con pesos iniciales diferentes

Naturaleza de la convergencia

- Inicializar pesos cerca de cero
- Luego, las redes iniciales son casi-lineales
- A medida que el entrenamiento progresa, incrementalmente se hacen posibles funciones no-lineales

Expresividad de las Redes Multicapas

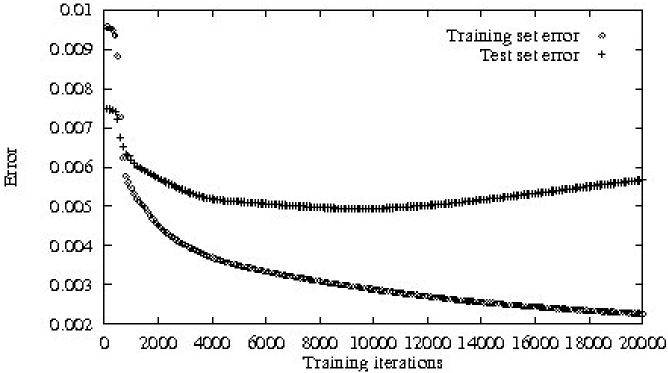
Funciones Booleanas:

- Cualquier función booleana puede ser representada por una red con una sola capa oculta
- pero puede requerie un número exponencial (en el número de entradas) de unidades ocultas

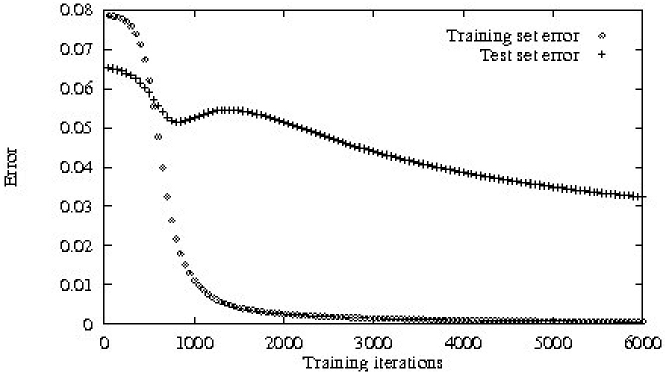
Funciones Contínuas:

- Toda función contínua acotada puede ser aproximada con un un error arbitráriamente pequeño, por una red con una capa oculta [Cybenko 1989; Hornik et al. 1989]
- Cualquier función puede ser aproximada con una precición arbitraria por una red con dos capa ocultas [Cybenko 1988]

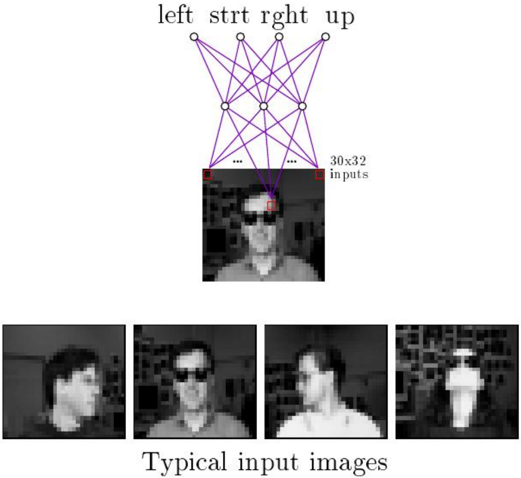
Ejemplo: Overfitting 1



Ejemplo: Overfitting 2

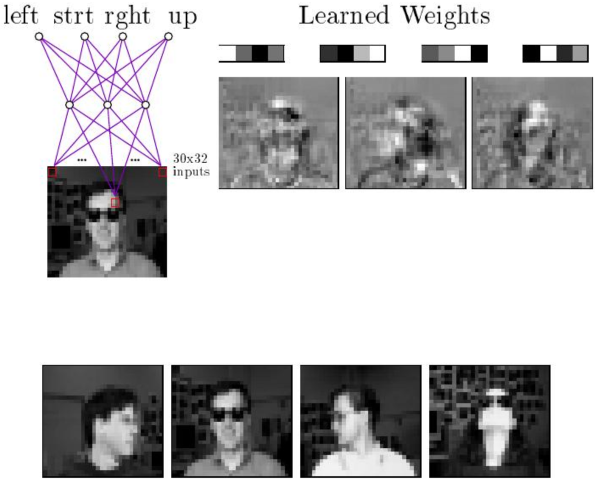


Ejemplo: Reconocimiento de caras



Aprendizaje de la pose facial con 90 % de precisión, y reconocimiento de 1-de-20 caras

Ejemplo: Reconocimiento de caras



Typical input images

http://www.cs.cmu.edu/ tom/faces.html

Backpropagation

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
  inputs: *examples*, a set of examples, each with input vector **x** and output vector **y**
    network, a multilayer network with L layers, weights $W_{j,i}$, activation function g

  repeat
    for each e **in** *examples* **do**
      for each node j in the input layer **do** $a_j \leftarrow x_j[e]$
      for $\ell = 2$ **to** M **do**
        $in_i \leftarrow \sum_j W_{j,i}\, a_j$
        $a_i \leftarrow g(in_i)$
      for each node i in the output layer **do**
        $\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$
      for $\ell = M - 1$ **to** 1 **do**
        for each node j in layer ℓ **do**
          $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i}\, \Delta_i$
          for each node i in layer $\ell + 1$ **do**
            $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$
  until some stopping criterion is satisfied
  return NEURAL-NET-HYPOTHESIS(*network*)

Figure 20.25 The back-propagation algorithm for learning in multilayer networks.