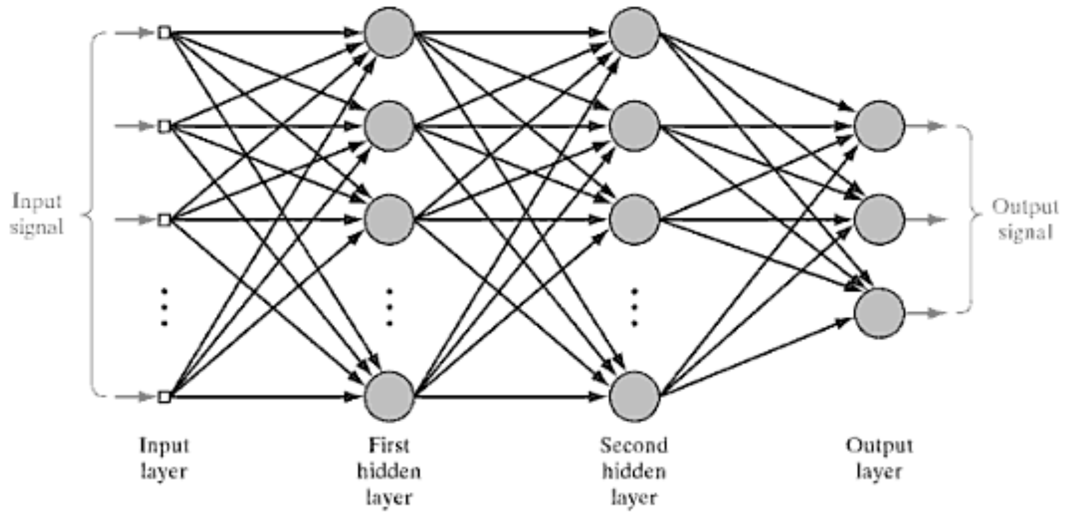
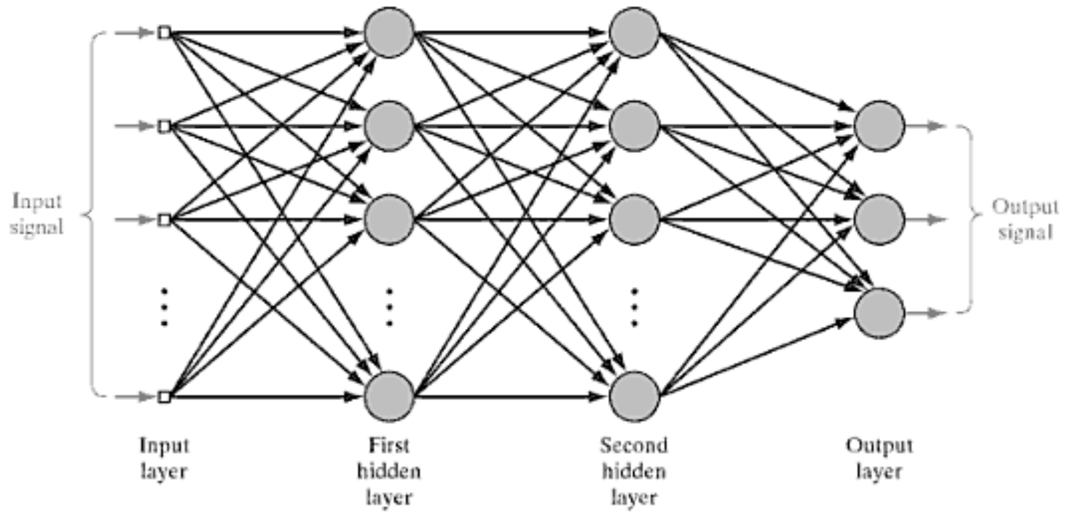
Redes Multicapa

Redes Multicapa



Redes Multicapa

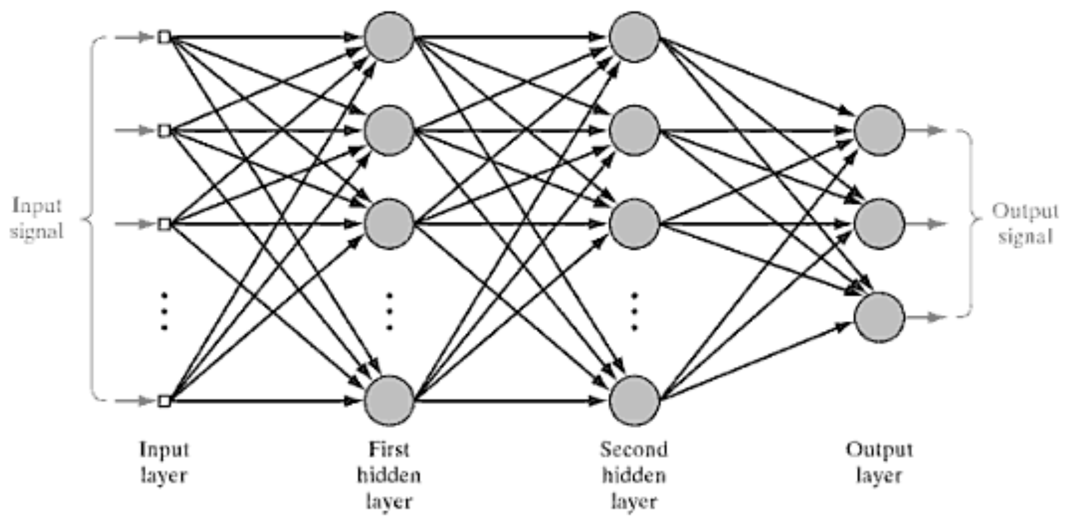
- Como hemos visto, los **perceptrones** tienen una capacidad **expresiva limitada**. Es por esto que vamos a estudiar las redes multicapa



Redes Multicapa

- Como hemos visto, los **perceptrones** tienen una capacidad **expresiva limitada**. Es por esto que vamos a estudiar las redes multicapa

- En una red **multicapa**, las neuronas se estructuran en **capas** en las que cada una **recibe su entrada de la salida** de las neuronas de la capa **anterior**



Redes Multicapa

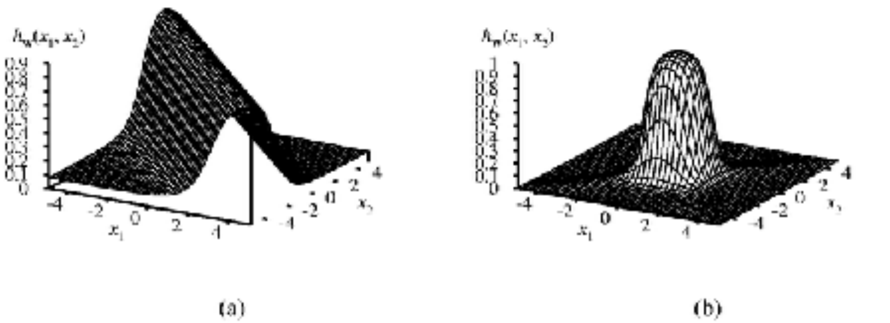
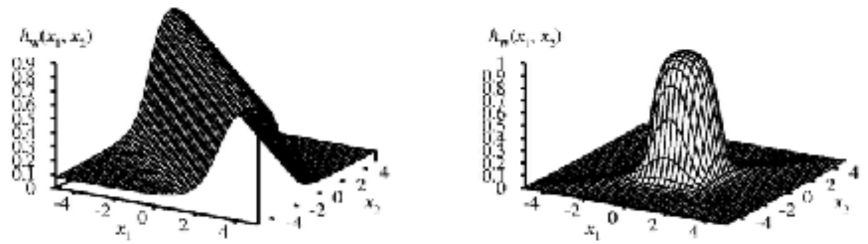


Figure 20.23 (a) The result of combining two opposite-facing soft threshold functions to produce a ridge. (b) The result of combining two ridges to produce a bump.

Redes Multicapa

- Combinando neuronas en distintas capas (y siempre que la función de activación sea no lineal) aumentamos la capacidad expresiva de la red



(a) (b)

Figure 20.23 (a) The result of combining two opposite-facing soft threshold functions to produce a ridge. (b) The result of combining two ridges to produce a bump.

Redes Multicapa

- Combinando neuronas en distintas capas (y siempre que la función de activación sea no lineal) aumentamos la capacidad expresiva de la red

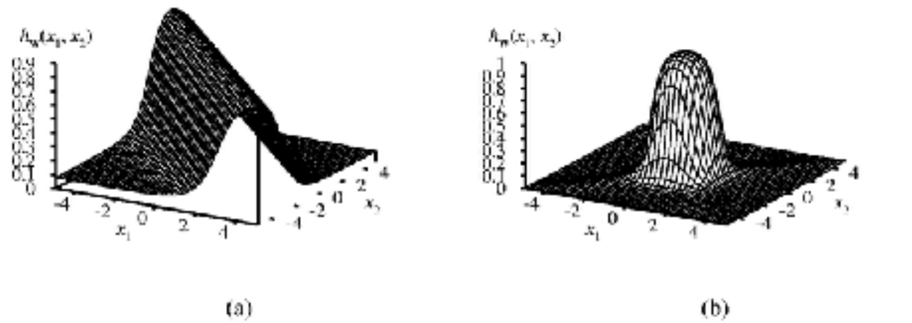


Figure 20.23 (a) The result of combining two opposite-facing soft threshold functions to produce a ridge. (b) The result of combining two ridges to produce a bump.

Redes Multicapa

- Combinando neuronas en distintas capas (y siempre que la función de activación sea no lineal) aumentamos la capacidad expresiva de la red

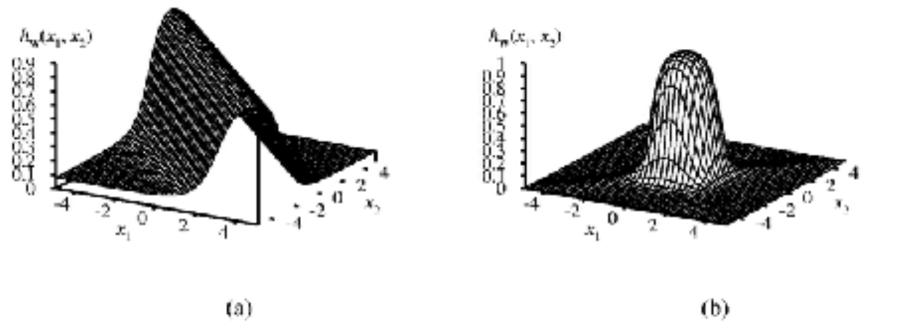


Figure 20.23 (a) The result of combining two opposite-facing soft threshold functions to produce a ridge. (b) The result of combining two ridges to produce a bump.

Redes Multicapa

- Combinando neuronas en distintas capas (y siempre que la función de activación sea no lineal) aumentamos la capacidad expresiva de la red

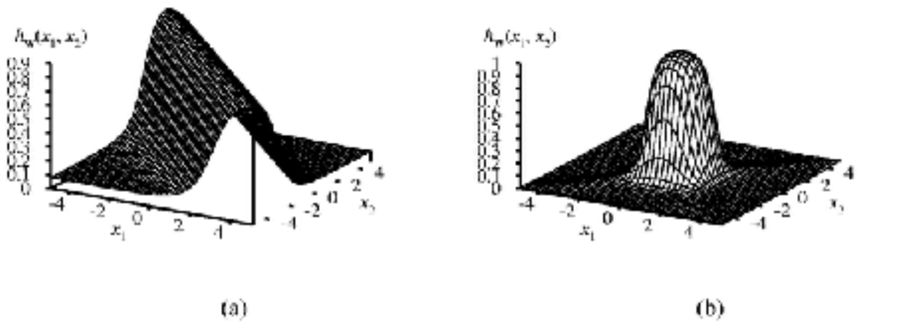


Figure 20.23 (a) The result of combining two opposite-facing soft threshold functions to produce a ridge. (b) The result of combining two ridges to produce a bump.

Redes Multicapa

- Combinando neuronas en distintas capas (y siempre que la función de activación sea no lineal) aumentamos la capacidad expresiva de la red

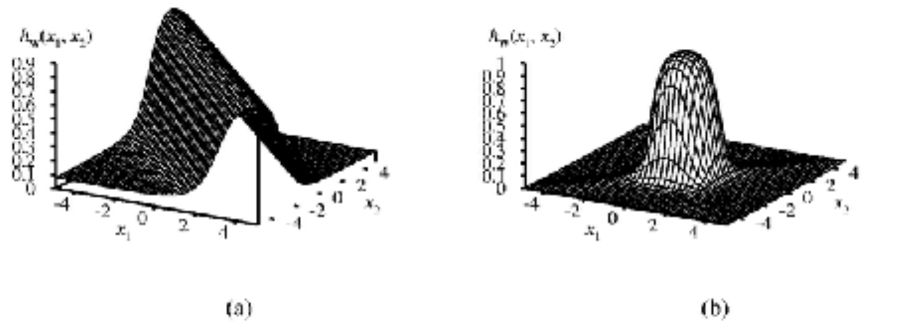


Figure 20.23 (a) The result of combining two opposite-facing soft threshold functions to produce a ridge. (b) The result of combining two ridges to produce a bump.

- Habitualmente, con una capa oculta basta para la mayoría de las aplicaciones reales

Redes Multicapa

Redes Multicapa

- Definición del problema de aprendizaje:
  (Análogamente al caso del perceptrón)

  - **Dado un conjunto de entrenamiento** D tal que cada $(\vec{x}, \vec{y}) \in D$ contiene una salida esperada $\vec{y} \in R^m$ para la entrada $\vec{x} \in R^n$

  - Partiendo de una red multicapa con una estructura dada, queremos **encontrar los pesos de la red** de manera que la función que calcula la red se ajuste lo mejor posible a los ejemplos

Redes Multicapa

Redes Multicapa

- El aprendizaje de estos pesos se realiza mediante un proceso de actualizaciones sucesivas

Redes Multicapa

- El aprendizaje de estos pesos se realiza mediante un proceso de actualizaciones sucesivas
- Lo vamos a basar en la misma idea del descenso por gradiente, aunque con modificaciones necesarias,

Redes Multicapa

- El aprendizaje de estos pesos se realiza mediante un proceso de actualizaciones sucesivas
- Lo vamos a basar en la misma idea del descenso por gradiente, aunque con modificaciones necesarias,
- y se llama "**Retropropagación**"
  - *Backpropagation*, para Google

Redes Multicapa

Redes Multicapa

- Supondremos una red neuronal con **n unidades en la capa de entrada, m en la de salida y L capas en tota**l
    - La capa 1 es la de entrada y la capa L es la de salida
    - **Cada unidad de una capa ℓ está conectada con todas las unidades de la capa $\ell + 1$**

Redes Multicapa

- Supondremos una red neuronal con **n unidades en la capa de entrada, m en la de salida y L capas en tota**l
    - La capa 1 es la de entrada y la capa L es la de salida
    - **Cada unidad de una capa ℓ está conectada con todas las unidades de la capa ℓ + 1**
- Supondremos una función de **activación** g **diferenciable** (usualmente, el sigmoide)

Redes Multicapa

Redes Multicapa

- Dado un ejemplo $(x_i, y_i) \in D$:
  - Si i es una neurona de la capa de **entrada**, notaremos por $\mathbf{x_i}$ la componente de $\vec{x}$ correspondiente a dicha neurona
  - Si i es una neurona de la capa de **salida**, notaremos por $\mathbf{y_i}$ la componente de $\vec{y}$ correspondiente a dicha neurona
  - notaremos $\mathbf{in_i}$ a la **entrada** que recibe una neurona i cualquiera y $\mathbf{a_i}$ a la **salida** por la misma neurona i

Redes Multicapa

$$a_i = x_i$$

$$in_i = \sum_{\forall j}^{capaanterior} w_{ji} a_j$$

$$a_i = g(in_i)$$

Redes Multicapa

- Si i es una neurona de entrada (es decir, de la capa 1), entonces

$$a_i = x_i$$

- Si i es una unidad de una capa $\ell \neq 1$, entonces:

$$in_i = \sum_{\forall j}^{capaanterior} w_{ji} a_j$$

$$a_i = g(in_i)$$

Retropropagación

Retropropagación

- El método de Retropropagación es un método de aproximaciones sucesivas.

Retropropagación

- El método de Retropropagación es un método de aproximaciones sucesivas.
- Formalmente, se puede considerar como descenso por el gradiente del error.

Retropropagación

Retropropagación

- Algoritmo

Retropropagación

- Algoritmo
- 1) Inicialización aleatoria de ~w

Retropropagación

- Algoritmo
- 1) Inicialización aleatoria de $\vec{w}$
- 2) Repetir hasta criterio de parada
    - Para cada ejemplo $(x, y) \in D$ hacer:
        - 1) Propagación de valores hacia adelante
        - 2) Propagación de errores hacía atrás
          Actualizando los pesos

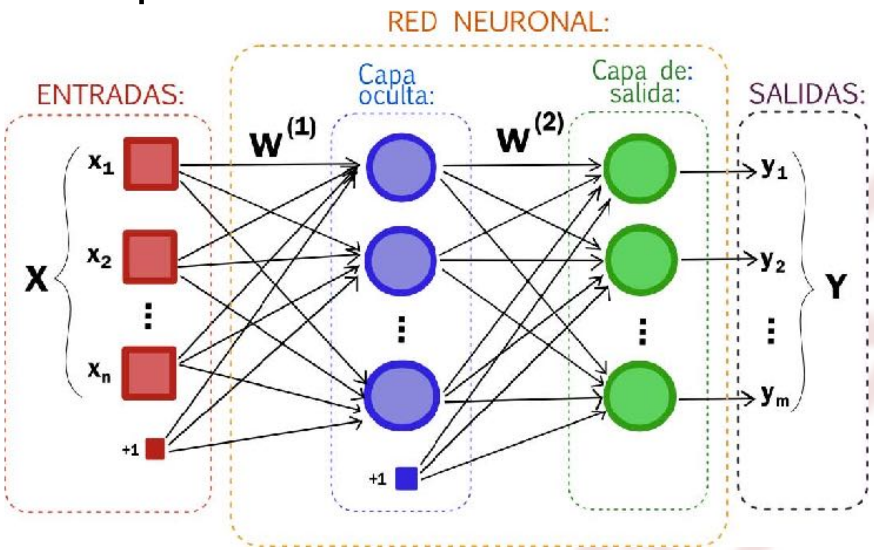
Retropropagación

- Algoritmo
- 1) Inicialización aleatoria de ~w
- 2) Repetir hasta criterio de parada
    - Para cada ejemplo (x, y) ∈ D hacer:
        - 1) Propagación de valores hacia adelante
        - 2) Propagación de errores hacía atrás
          Actualizando los pesos
- 3) Devolver ~w

Propagación hacia adelante

- A partir de los datos de entrada ~x, vamos calculando todas las neuronas, desde la capa 1 hasta la capa L

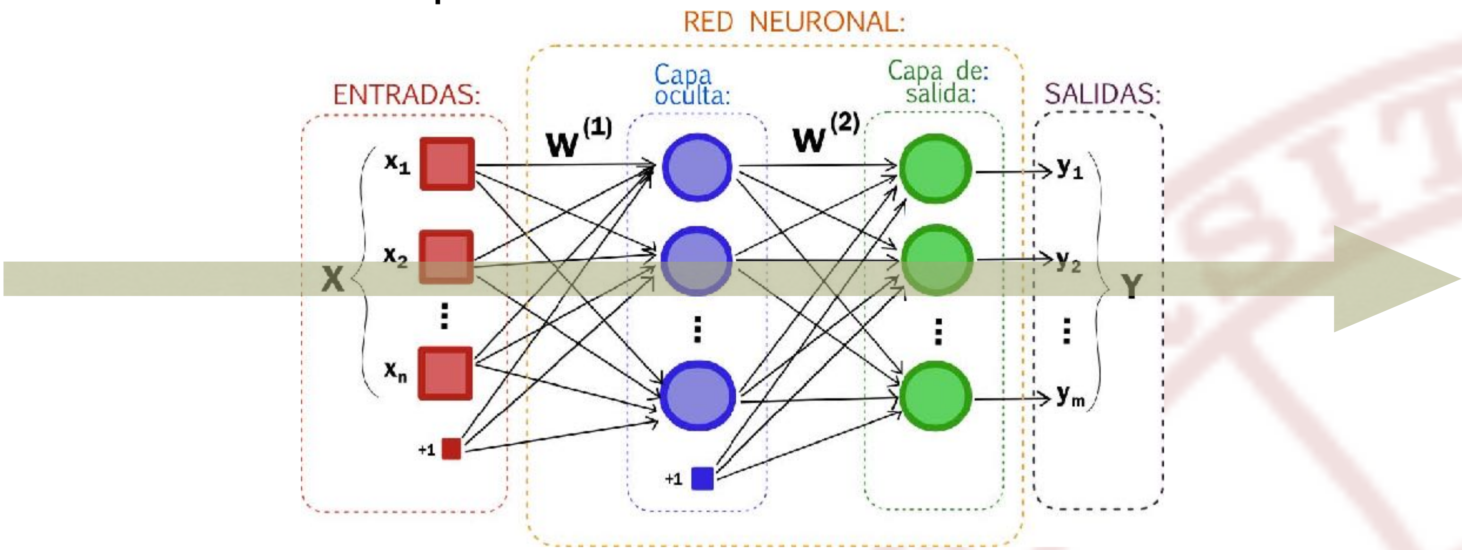
- La salida de las neuronas de la capa ℓ, ponderada por los pesos correspondientes, sirve de entrada para las neuronas de la capa ℓ+1



Propagación hacia adelante

- A partir de los datos de entrada ~x, vamos calculando todas las neuronas, desde la capa 1 hasta la capa L

- La salida de las neuronas de la capa ℓ, ponderada por los pesos correspondientes, sirve de entrada para las neuronas de la capa ℓ+1



Propagación hacia adelante

$$a_i \leftarrow x_i$$

$$in_i = \sum_{\forall j}^{capaanterior} w_{ji} a_j$$

$$a_i \leftarrow g(in_i)$$

Propagación hacia adelante

- 1) Para cada nodo i de la capa de entrada hacer $a_i \leftarrow x_i$

$$in_i = \sum_{\forall j}^{capaanterior} w_{ji} a_j$$

$$a_i \leftarrow g(in_i)$$

Propagación hacia adelante

- 1) Para cada nodo i de la capa de entrada hacer $a_i \leftarrow x_i$
- 2) Para ℓ, desde 2 hasta L, hacer
  - 1) Para cada neurona i de la capa ℓ
    - Calcular su salida con:

$$in_i = \sum_{\forall j}^{capaanterior} w_{ji} a_j$$

$$a_i \leftarrow g(in_i)$$

Propagación hacia atrás

- Ahora, en vez de propagar valores, vamos a propagar errores
- La dificultad está en que sólo sabemos el error en la capa final, que es donde tenemos el valor teórico de la salida (~y)
    - En las capas anteriores, no sabemos cuanto es el valor teórico que deberían de dar.
- Creamos un símbolo Δ para significar el factor de error y es eso lo que vamos a propagar hacia detrás

Propagación hacia atrás

$$\Delta_i \leftarrow g'(in_i)(y_i - a_i)$$

$$\Delta_j \leftarrow g'(in_j) \sum_{i}^{capa\ l+1} w_{ji}\Delta_i$$

$$w_{ji} \leftarrow w_{ji} + \eta a_j \Delta_i$$

Propagación hacia atrás

- 1) Para cada neurona i en la capa de **salida**, calcular:

$$\Delta_i \leftarrow g'(in_i)(y_i - a_i)$$

$$\Delta_j \leftarrow g'(in_j) \sum_{i}^{capa\ l+1} w_{ji}\Delta_i$$

$$w_{ji} \leftarrow w_{ji} + \eta a_j \Delta_i$$

Propagación hacia atrás

- 1) Para cada neurona i en la capa de **salida**, calcular:

$$\Delta_i \leftarrow g'(in_i)(y_i - a_i)$$

$$\Delta_j \leftarrow g'(in_j) \sum_{i}^{capa\ l+1} w_{ji}\Delta_i$$

$$w_{ji} \ \leftarrow \ w_{ji} + \eta a_j \Delta_i$$

Propagación hacia atrás

- 1) Para cada neurona i en la capa de **salida**, calcular:

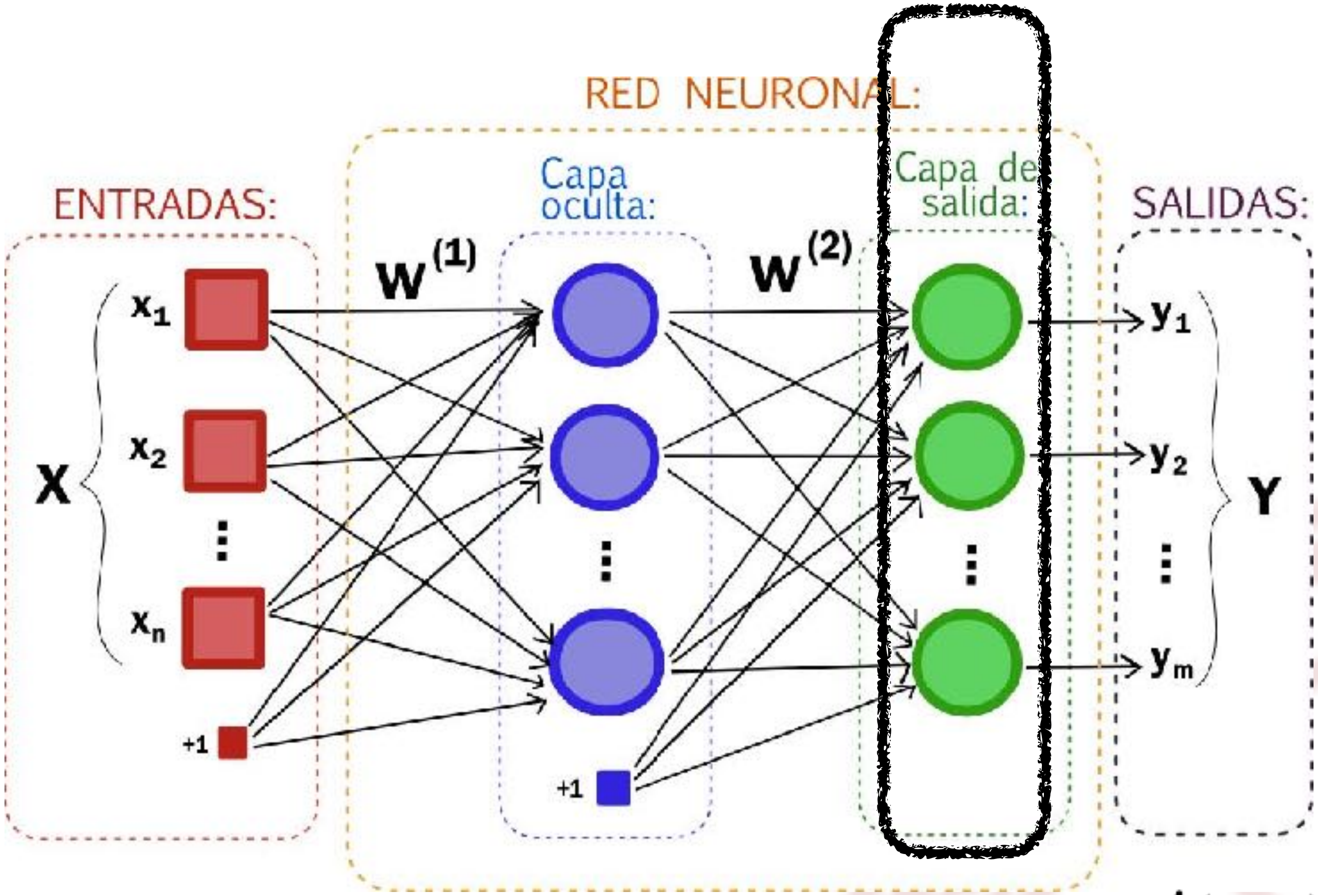
$$\Delta_i \leftarrow g'(in_i)(y_i - a_i)$$

- 2) Para ℓ, **desde L – 1 hasta 1**, hacer
  - 1) Para cada neurona j en la capa ℓ hacer
    - 1) $$\Delta_j \leftarrow g'(in_j) \sum_{i}^{capa\ \ l+1} w_{ji}\Delta_i$$
    - 2) Para cada neurona i en la capa ℓ + 1 hacer

$$w_{ji} \ \leftarrow \ w_{ji} + \eta a_j \Delta_i$$

Propagación hacia atrás

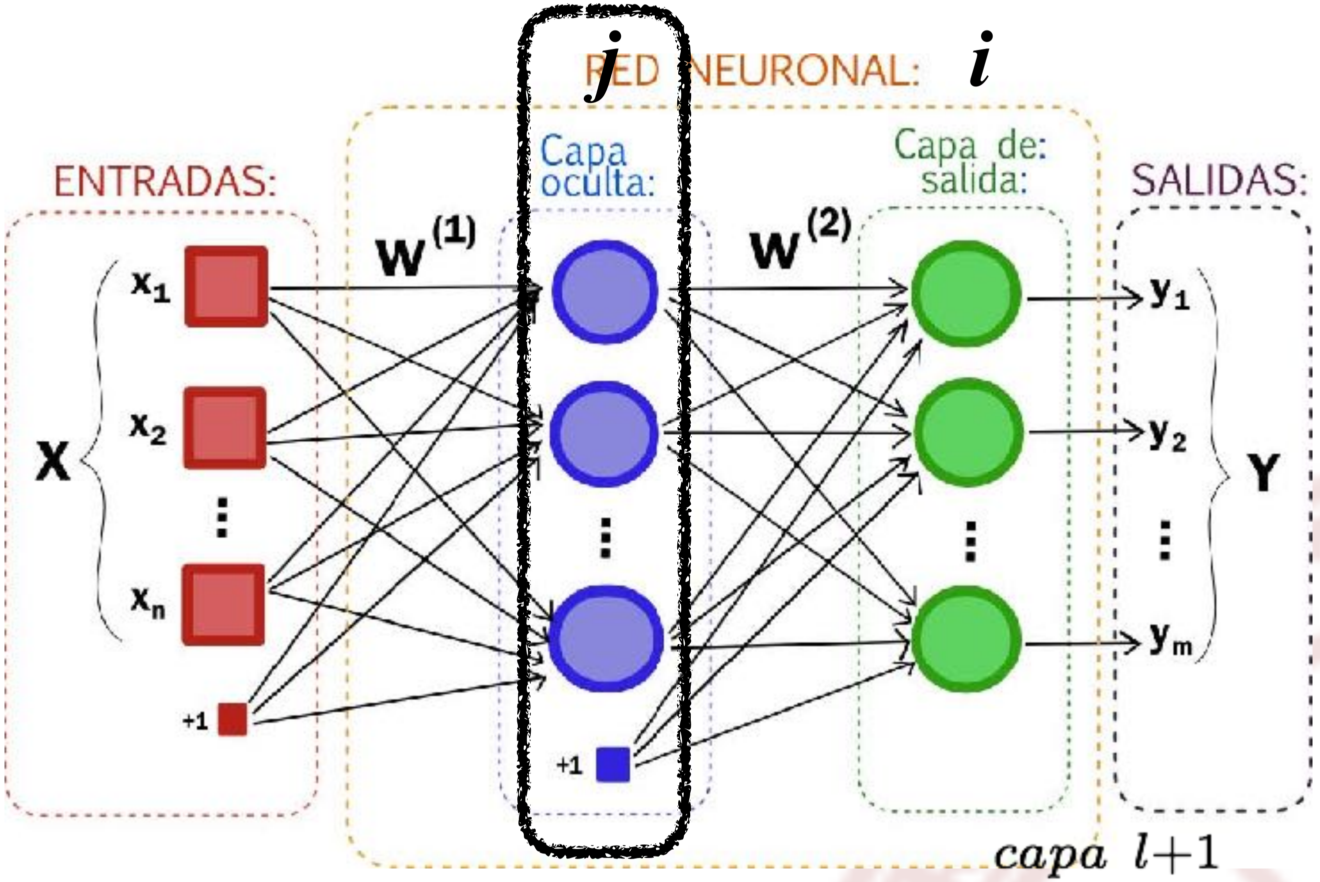
1. Cálculo del Δ



$$\Delta_i \leftarrow g'(in_i)(y_i - a_i)$$

Propagación hacia atrás

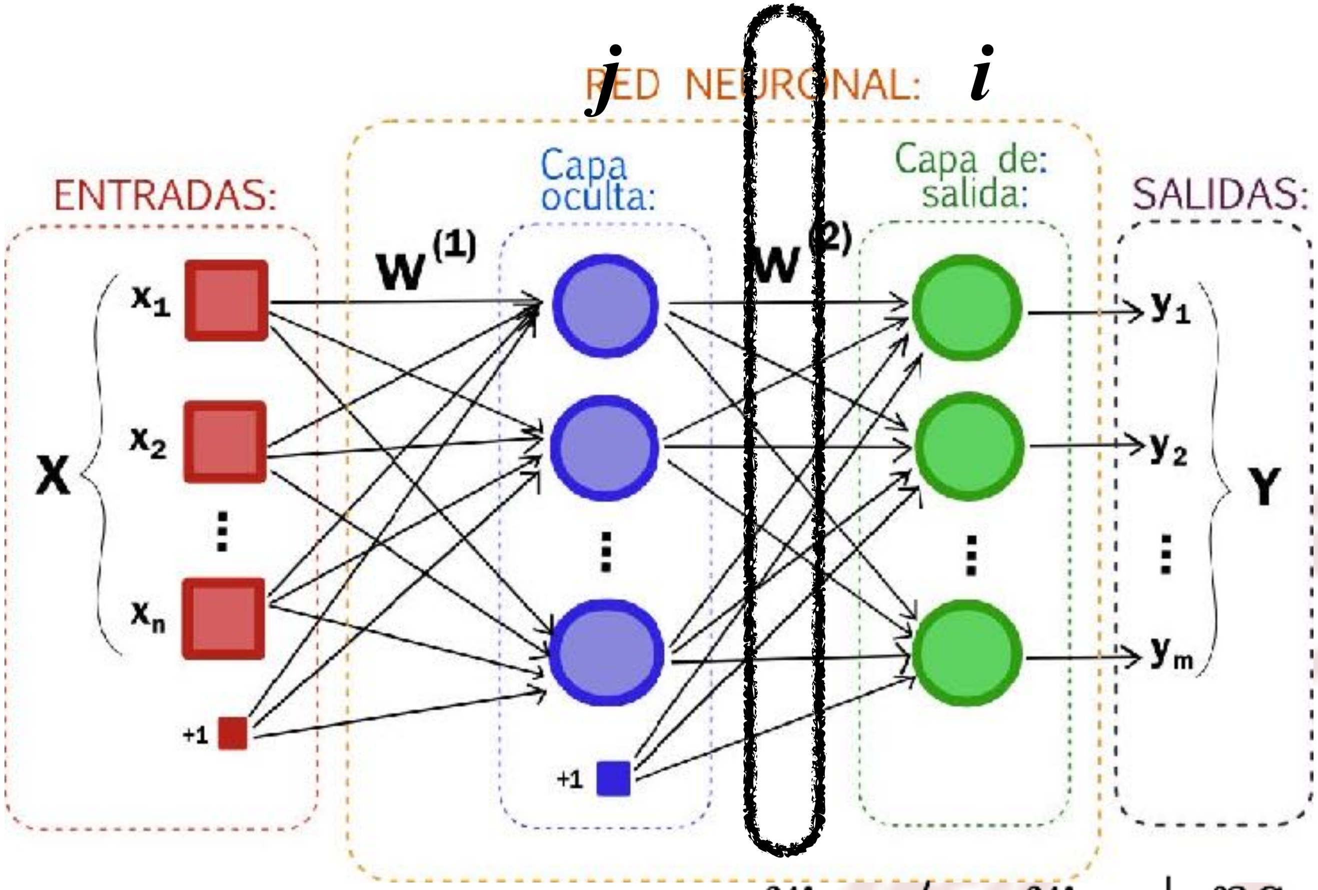
2. Cálculo del Δ en la capa interior para cada neurona



$$\Delta_j \leftarrow g'(in_j) \sum_{i}^{capa\ l+1} w_{ji} \Delta_i$$

Propagación hacia atrás

3.Actualización de los pesos



$$w_{ji} \leftarrow w_{ji} + \eta a_j \Delta_i$$

Retropropagación con unidades sigmoide

$$g(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

$$g'(x) = g(x)(1 - g(x)) = a_i(1 - a_i)$$

$$\Delta_i \leftarrow a_i(1 - a_i)(y_i - a_i)$$

$$\Delta_j \leftarrow a_j(1 - a_j) \sum_i w_{ji} \Delta_i$$

Retropropagación con unidades sigmoide

- Función de activación: $g(x) = \sigma(x) = \dfrac{1}{1 + e^{-x}}$

$$g'(x) = g(x)(1 - g(x)) = a_i(1 - a_i)$$

$$\Delta_i \leftarrow a_i(1 - a_i)(y_i - a_i)$$

$$\Delta_j \leftarrow a_j(1 - a_j) \sum_i w_{ji} \Delta_i$$

Retropropagación con unidades sigmoide

- Función de activación: $g(x) = \sigma(x) = \dfrac{1}{1+e^{-x}}$

$$g'(x) = g(x)(1-g(x)) = a_i(1-a_i)$$

$$\Delta_i \leftarrow a_i(1-a_i)(y_i - a_i)$$

$$\Delta_j \leftarrow a_j(1-a_j)\sum_i w_{ji}\Delta_i$$

Retropropagación con unidades sigmoide

- Función de activación: $g(x) = \sigma(x) = \dfrac{1}{1+e^{-x}}$

$$g'(x) = g(x)(1 - g(x)) = a_i(1 - a_i)$$

$$\Delta_i \leftarrow a_i(1 - a_i)(y_i - a_i)$$

$$\Delta_j \leftarrow a_j(1 - a_j)\sum_i w_{ji}\Delta_i$$

Retropropagación con unidades sigmoide

- Función de activación: $g(x) = \sigma(x) = \dfrac{1}{1 + e^{-x}}$

$$g'(x) = g(x)(1 - g(x)) = a_i(1 - a_i)$$

- Así, el cálculo de errores en el Paso 2 queda:
  - Para la capa de salida, $\Delta_i \leftarrow a_i(1 - a_i)(y_i - a_i)$
  - Para las capas ocultas, $\Delta_j \leftarrow a_j(1 - a_j) \sum_i w_{ji} \Delta_i$

Retropropagación con unidades sigmoide

- Función de activación: $g(x) = \sigma(x) = \dfrac{1}{1 + e^{-x}}$

$$g'(x) = g(x)(1 - g(x)) = a_i(1 - a_i)$$

- Así, el cálculo de errores en el Paso 2 queda:
  - Para la capa de salida, $\Delta_i \leftarrow a_i(1 - a_i)(y_i - a_i)$
  - Para las capas ocultas, $\Delta_j \leftarrow a_j(1 - a_j) \sum_i w_{ji} \Delta_i$
- Esto significa que no necesitamos almacenar los ini del Paso 1 para usarlos en el Paso 2

Momentum

- Retropropagación es un método de descenso por gradiente y, por tanto, existe el problema de mínimos locales
- Una variante muy común en el algoritmo de retropropagación es introducir un sumando adicional en la actualización de los pesos.
- Este sumando hace que en cada actualización de pesos se tenga también en cuenta la actualización realizada en la iteración anterior.

Momentum

- Concretamente:
  - En la iteración n-ésima, se actualizan los pesos de forma:

$$w_{ji} \leftarrow wji + \Delta w_{ji}^{(n)}$$

  - con:

$$\Delta w_{ji}^{(n)} = \eta a_j \Delta_i + \alpha \Delta w_{ji}^{(n-1)}$$

- α es una constante denominada momentum ($0 < \alpha \leq 1$)
- La técnica del momentum puede ser eficaz, a veces, para escapar de mínimos locales

Criterio parada

- Se pueden usar diferentes criterios de parada
  - Número de iteraciones prefijadas
  - Error por debajo de una cota
- En esta última, es fácil que caigamos en el sobreaprendizaje.
  - Habría que validar el resultado con un dataset independiente o alguna técnica de validación como las que hemos visto en prácticas.

Estructura de la red

Estructura de la red

Estructura de la red

- El algoritmo de retropropagación parte de una estructura de red fija

Estructura de la red

- El algoritmo de retropropagación parte de una estructura de red fija
- Hasta ahora no hemos dicho nada sobre qué estructuras son las mejores para cada problema

Estructura de la red

- El algoritmo de retropropagación parte de una estructura de red fija
- Hasta ahora no hemos dicho nada sobre qué estructuras son las mejores para cada problema
- En nuestro caso, se trata de decidir cuántas capas ocultas se toman, y cuántas unidades en cada capa

Estructura de la red

Estructura de la red

- En general es un problema que no está completamente resuelto aún

Estructura de la red

- En general es un problema que no está completamente resuelto aún
- Lo más usual es hacer búsqueda experimental de la mejor estructura, medida sobre un conjunto de prueba independiente

Estructura de la red

- En general es un problema que no está completamente resuelto aún
- Lo más usual es hacer búsqueda experimental de la mejor estructura, medida sobre un conjunto de prueba independiente
- La mayoría de las veces, una sola capa oculta con pocas unidades basta para obtener buenos resultados

Estructura de la red

- En general es un problema que no está completamente resuelto aún
- Lo más usual es hacer búsqueda experimental de la mejor estructura, medida sobre un conjunto de prueba independiente
- La mayoría de las veces, una sola capa oculta con pocas unidades basta para obtener buenos resultados
- Las redes grandes corren un mayor peligro de sobreajuste

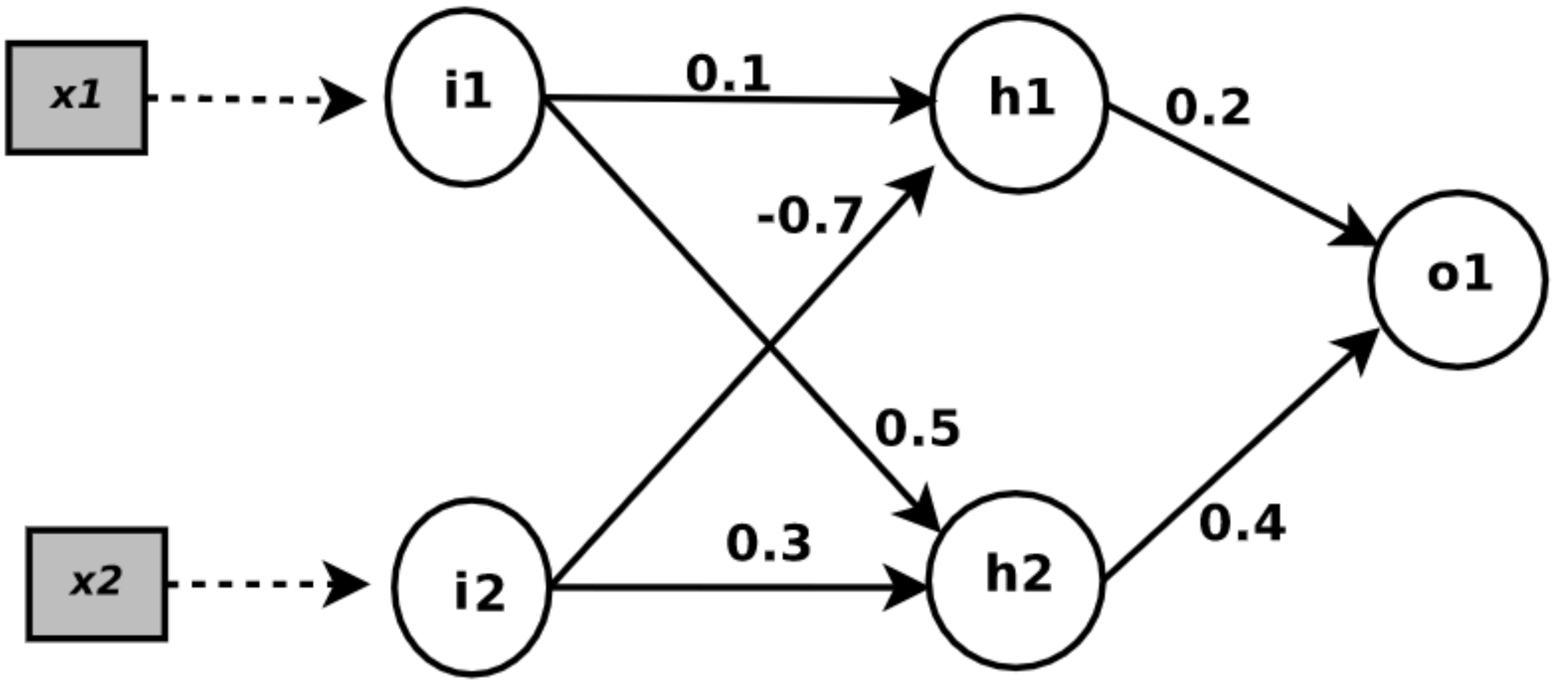
Ejemplo

Ejemplo

- Entrenamiento de un perceptrón multicapa para realizar la operación XOR
- Descripción de la red.
  - 1 capa oculta
  - 2 neuronas en capa de entrada (i1, i2)
  - 2 neuronas en capa oculta (h1, h2)
  - 1 neurona en capa de salida (o1)
  - Tasa de aprendizaje: $\alpha = 0.25$

Ejemplo

- Gráficamente



Ejemplo

Conjunto de entrenamiento:

	x1	x2	y
e1	0	1	1
e2	1	0	1
e3	1	1	0
e4	0	0	0

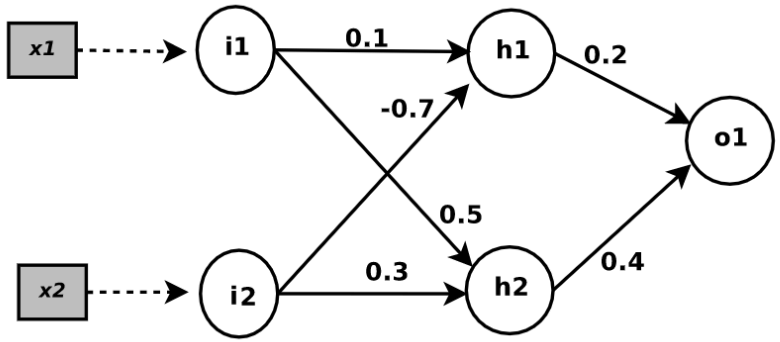
Ejemplo

- Seleccionamos el **primer ejemplo**

	x1	x2	y
e1	0	1	1
e2	1	0	1
e3	1	1	0
e4	0	0	0

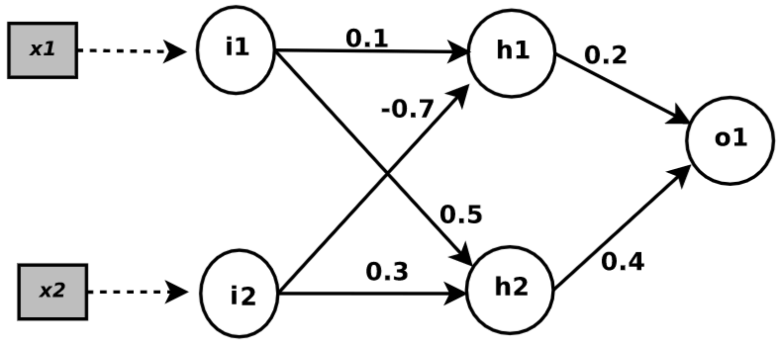
Hacia adelante

- **Cálculo de las "a"s de cada neurona**



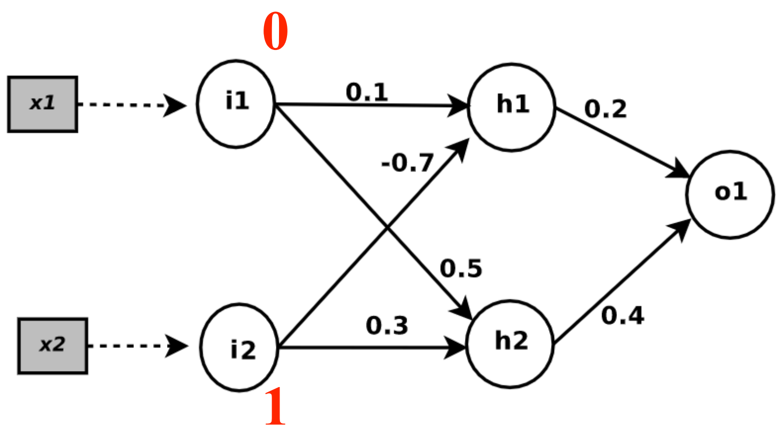
Hacia adelante

- **Cálculo de las "a"s de cada neurona**
- Capa de entrada



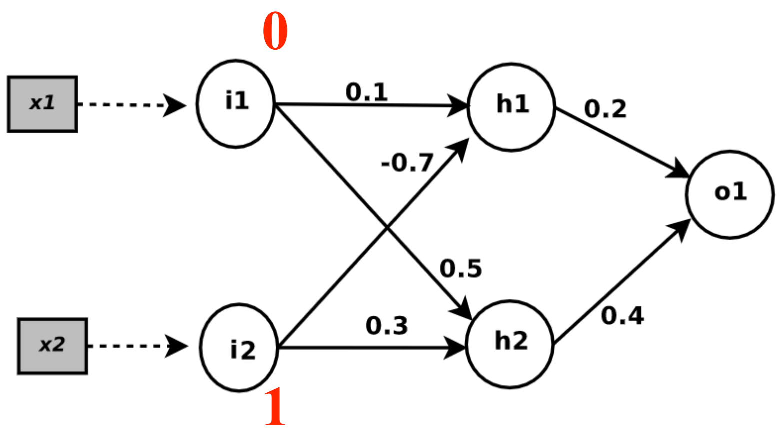
Hacia adelante

- **Cálculo de las "a"s de cada neurona**
- Capa de entrada
  - **a_x1** = x1 = **0**
    a_x2 = x2 = **1**



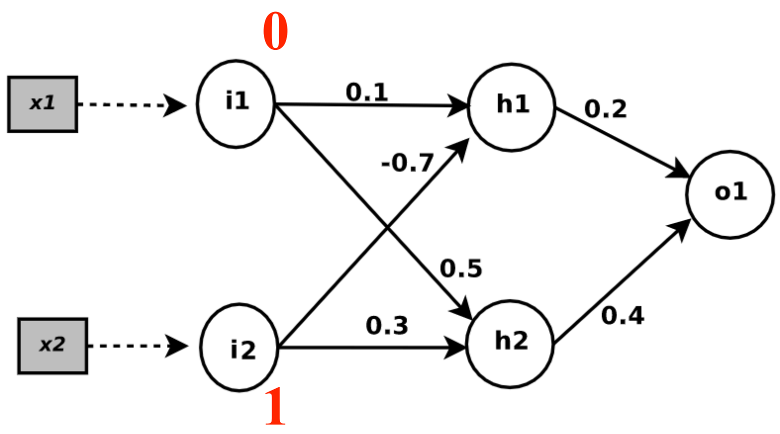
Hacia adelante

- **Cálculo de las "a"s de cada neurona**
- Capa de entrada
  - **a_x1** = x1 = **0**
    a_x2 = x2 = **1**
- Capa Oculta



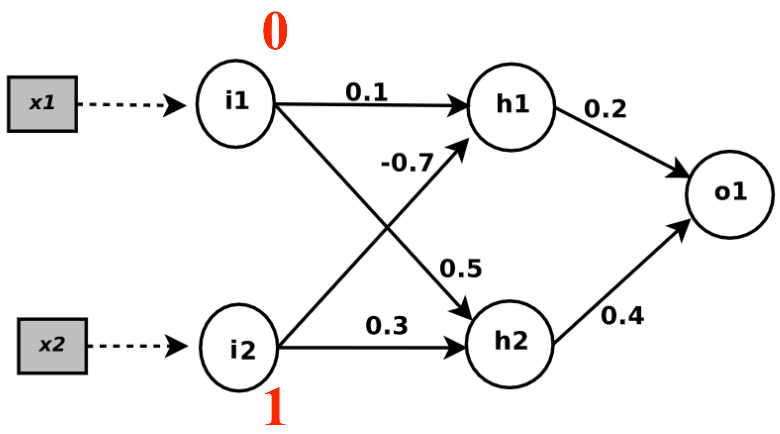
Hacia adelante

- **Cálculo de las “a”s de cada neurona**
- Capa de entrada
    - **a_x1** = x1 = **0**
      a_x2 = x2 = **1**
- Capa Oculta
    - h1



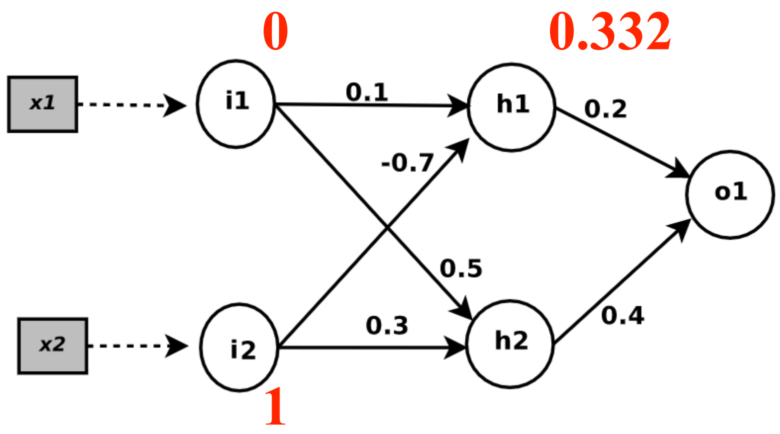
Hacia adelante

- **Cálculo de las “a”s de cada neurona**
- Capa de entrada
  - **a_x1** = x1 = **0**
    a_x2 = x2 = **1**
- Capa Oculta
  - h1
    - in_h1 = 0.1 * 0 + (-0.7)*1 = -0.7



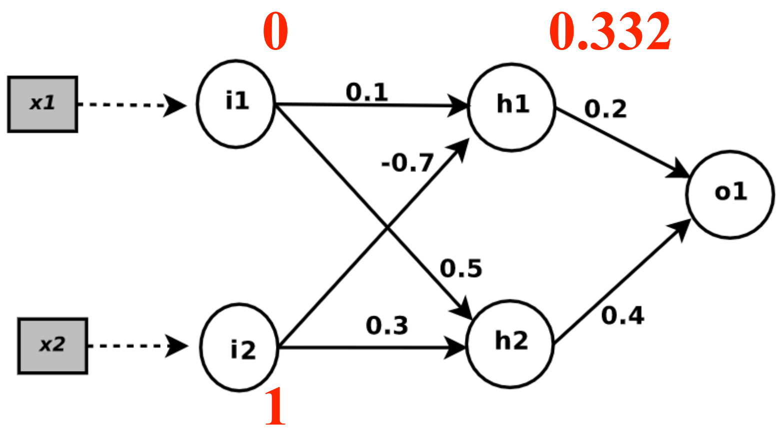
Hacia adelante

- **Cálculo de las “a”s de cada neurona**
- Capa de entrada
  - **a_x1** = x1 = **0**
    a_x2 = x2 = **1**
- Capa Oculta
  - h1
    - in_h1 = 0.1 * 0 + (-0.7)*1 = -0.7
    - **a_h1** = g(-0.7) = 1/(1+$e^{0.7}$) = **0.332**



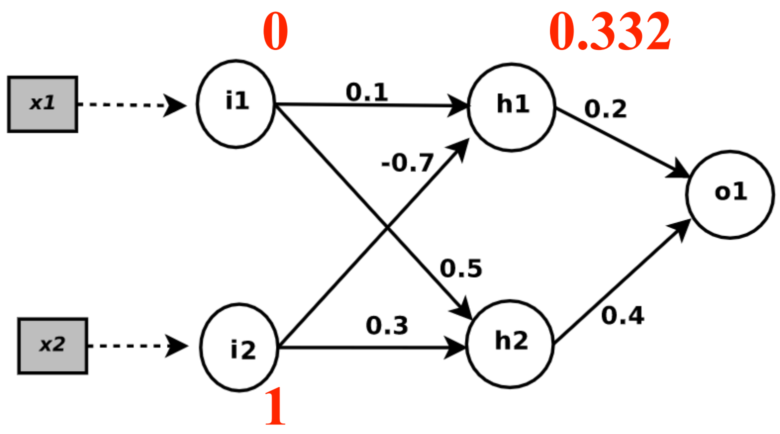
Hacia adelante

- **Cálculo de las "a"s de cada neurona**
- Capa de entrada
  - **a_x1** = x1 = **0**
    a_x2 = x2 = **1**
- Capa Oculta
  - h1
    - in_h1 = 0.1 * 0 + (-0.7)*1 = -0.7
    - **a_h1** = g(-0.7) = 1/(1+$e^{0.7}$) = **0.332**
  - h2



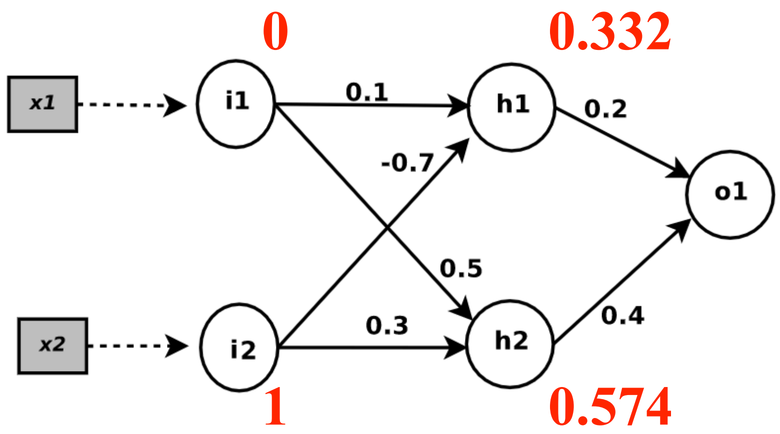
Hacia adelante

- **Cálculo de las “a”s de cada neurona**
- Capa de entrada
  - **a_x1** = x1 = **0**
    a_x2 = x2 = **1**
- Capa Oculta
  - h1
    - in_h1 = 0.1 * 0 + (-0.7)*1 = -0.7
    - **a_h1** = g(-0.7) = 1/(1+$e^{0.7}$) = **0.332**
  - h2
    - in_h2 = 0.5 * 0 + 0.3 * 1 = 0.3



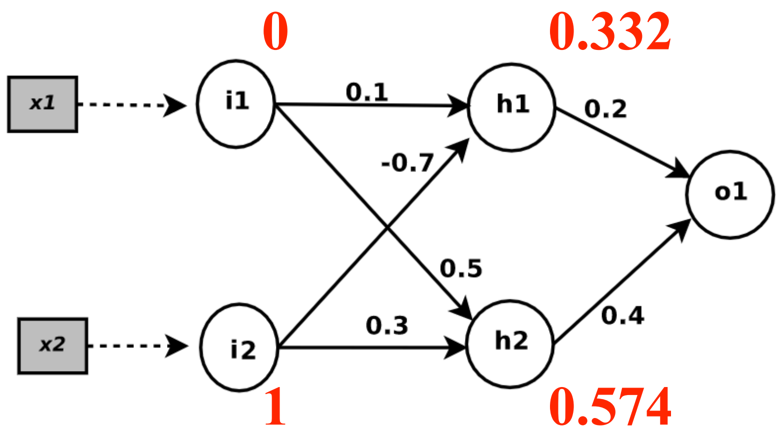
Hacia adelante

- **Cálculo de las “a”s de cada neurona**
- Capa de entrada
  - **a_x1** = x1 = **0**
    a_x2 = x2 = **1**
- Capa Oculta
  - h1
    - in_h1 = 0.1 * 0 + (-0.7)*1 = -0.7
    - **a_h1** = g(-0.7) = 1/(1+$e^{0.7}$) = **0.332**
  - h2
    - in_h2 = 0.5 * 0 + 0.3 * 1 = 0.3
    - **a_h2** = g(0.3) = 1/(1+$e^{-0.3}$) = **0.574**



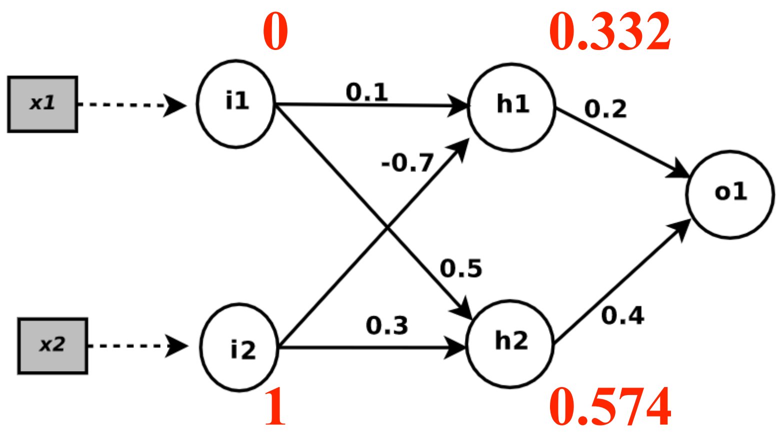
Hacia adelante

- **Cálculo de las "a"s de cada neurona**
- Capa de entrada
  - **a_x1** = x1 = **0**
    a_x2 = x2 = **1**
- Capa Oculta
  - h1
    - in_h1 = 0.1 * 0 + (-0.7)*1 = -0.7
    - **a_h1** = g(-0.7) = 1/(1+$e^{0.7}$) = **0.332**
  - h2
    - in_h2 = 0.5 * 0 + 0.3 * 1 = 0.3
    - **a_h2** = g(0.3) = 1/(1+$e^{-0.3}$) = **0.574**
- Capa de salida



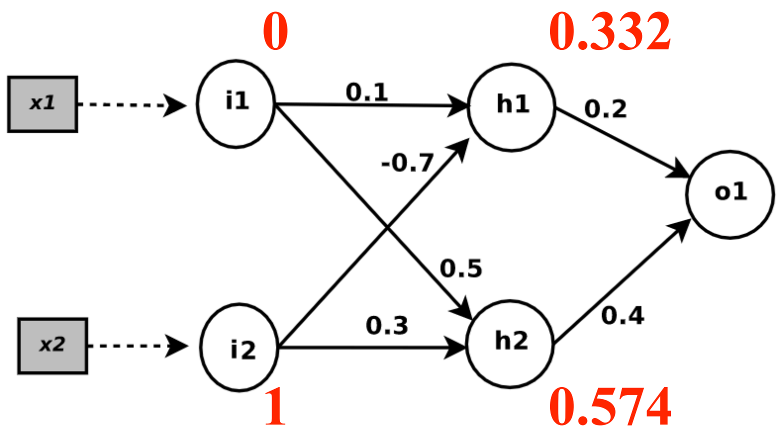
Hacia adelante

- **Cálculo de las “a”s de cada neurona**
- Capa de entrada
    - **a_x1** = x1 = **0**
      a_x2 = x2 = **1**
- Capa Oculta
    - h1
        - in_h1 = 0.1 * 0 + (-0.7)*1 = -0.7
        - **a_h1** = g(-0.7) = 1/(1+$e^{0.7}$) = **0.332**
    - h2
        - in_h2 = 0.5 * 0 + 0.3 * 1 = 0.3
        - **a_h2** = g(0.3) = 1/(1+$e^{-0.3}$) = **0.574**
- Capa de salida
    - o1



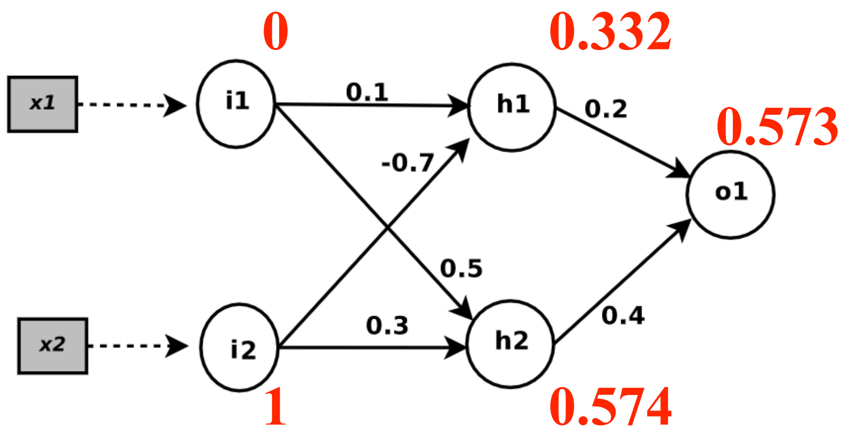
Hacia adelante

- **Cálculo de las "a"s de cada neurona**
- Capa de entrada
  - **a_x1** = x1 = **0**
    a_x2 = x2 = **1**
- Capa Oculta
  - h1
    - in_h1 = 0.1 * 0 + (-0.7)*1 = -0.7
    - **a_h1** = g(-0.7) = 1/(1+$e^{0.7}$) = **0.332**
  - h2
    - in_h2 = 0.5 * 0 + 0.3 * 1 = 0.3
    - **a_h2** = g(0.3) = 1/(1+$e^{-0.3}$) = **0.574**
- Capa de salida
  - o1
    - in_o1 = 0.2 * 0.332 + 0.4 * 0.574 = 0.296



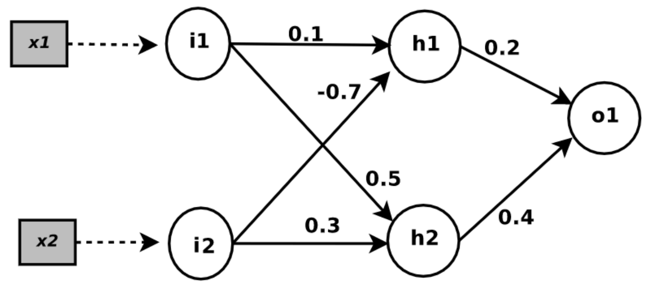
Hacia adelante

- **Cálculo de las “a”s de cada neurona**
- Capa de entrada
  - **a_x1** = x1 = **0**
    a_x2 = x2 = **1**
- Capa Oculta
  - h1
    - in_h1 = 0.1 * 0 + (-0.7)*1 = -0.7
    - **a_h1** = g(-0.7) = $1/(1+e^{0.7})$ = **0.332**
  - h2
    - in_h2 = 0.5 * 0 + 0.3 * 1 = 0.3
    - **a_h2** = g(0.3) = $1/(1+e^{-0.3})$ = **0.574**
- Capa de salida
  - o1
    - in_o1 = 0.2 * 0.332 + 0.4 * 0.574 = 0.296
    - **a_o1** = g(0.296) = $1/(1+e^{-0.296})$ = **0.573**



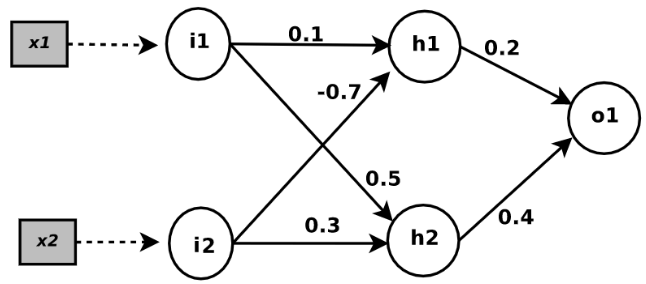
Hacia atrás

- Capa de salida:

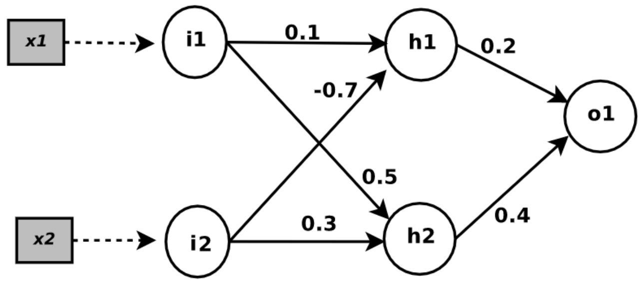


Hacia atrás

- Capa de salida:
  - Neurona o1:

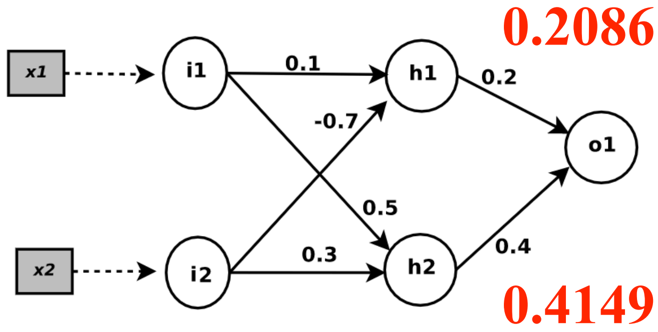


Hacia atrás



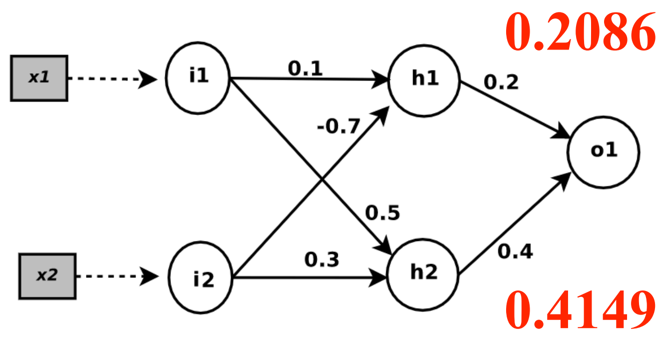
- Capa de salida:
  - Neurona o1:
    - Δo1 = a_oi (1-a_oi)(y_oi - a_oi) = 0.573 (1-0.573)(1-0.573) = 0.1044

Hacia atrás



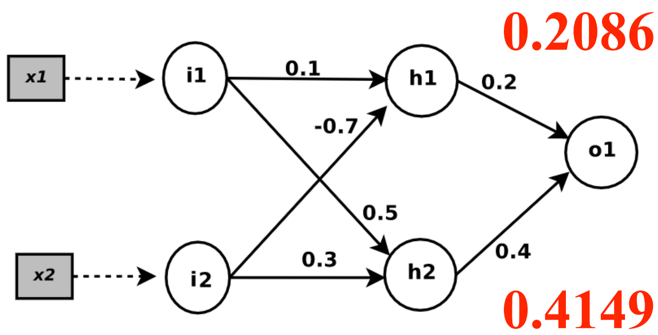
- Capa de salida:
  - Neurona o1:
    - Δo1 = a_oi (1-a_oi)(y_oi - a_oi) = 0.573 (1-0.573)(1-0.573) = 0.1044
    - **w_h1_o1** = w_h1_o1+alfa * a_h1 * Δo1 = 0.2 + 0.25 * 0.332 * 0.1044 = **0.2086**
      w_h2_o1 = w_h2_o1+alfa * a_h2 * Δo1 = 0.4 + 0.25 * 0.574 * 0.1044 = **0.4149**

Hacia atrás



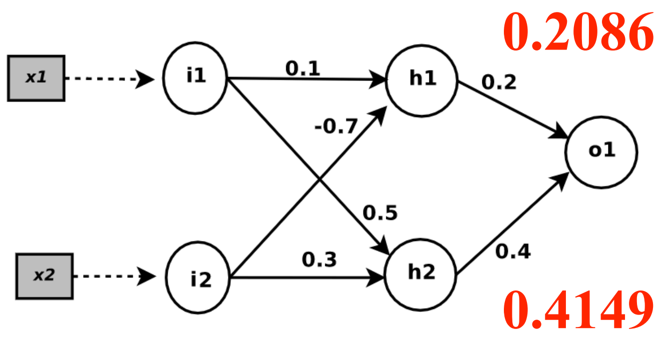
- Capa de salida:
    - Neurona o1:
        - Δo1 = a_oi (1-a_oi)(y_oi - a_oi) = 0.573 (1-0.573)(1-0.573) = 0.1044
        - **w_h1_o1** = w_h1_o1+alfa * a_h1 * Δo1 = 0.2 + 0.25 * 0.332 * 0.1044 = **0.2086**
          w_h2_o1 = w_h2_o1+alfa * a_h2 * Δo1 = 0.4 + 0.25 * 0.574 * 0.1044 = **0.4149**

Hacia atrás



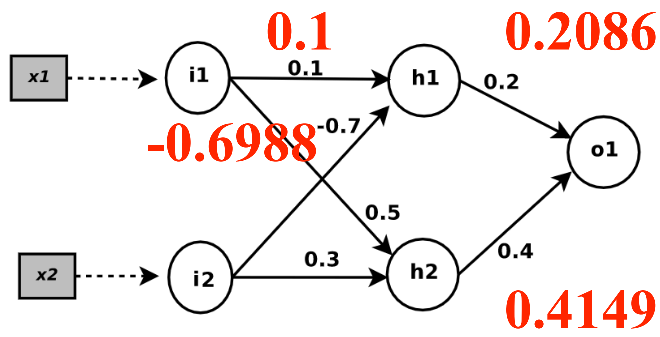
- Capa de salida:
    - Neurona o1:
        - Δo1 = a_oi (1-a_oi)(y_oi - a_oi) = 0.573 (1-0.573)(1-0.573) = 0.1044
        - **w_h1_o1** = w_h1_o1+alfa * a_h1 * Δo1 = 0.2 + 0.25 * 0.332 * 0.1044 = **0.2086**
          w_h2_o1 = w_h2_o1+alfa * a_h2 * Δo1 = 0.4 + 0.25 * 0.574 * 0.1044 = **0.4149**
    - Capa oculta

Hacia atrás



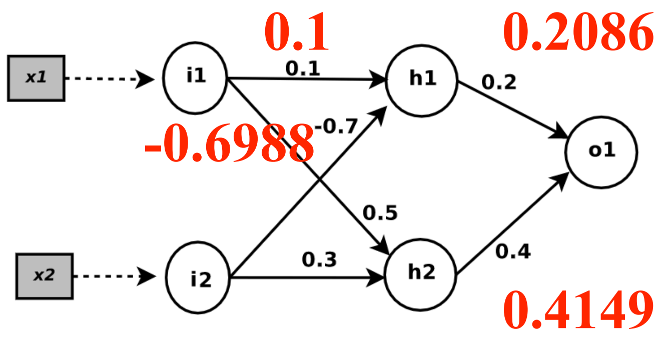
- Capa de salida:
  - Neurona o1:
    - Δo1 = a_oi (1-a_oi)(y_oi - a_oi) = 0.573 (1-0.573)(1-0.573) = 0.1044
    - **w_h1_o1** = w_h1_o1+alfa * a_h1 * Δo1 = 0.2 + 0.25 * 0.332 * 0.1044 = **0.2086**
      w_h2_o1 = w_h2_o1+alfa * a_h2 * Δo1 = 0.4 + 0.25 * 0.574 * 0.1044 = **0.4149**
  - Capa oculta
    - Δh1 = a_h1(1-a_h1) * [w_h1_o1 * Δo1] = 0.332(1-0.332) * [**0.2** * 0.1044] = 0.0046

Hacia atrás



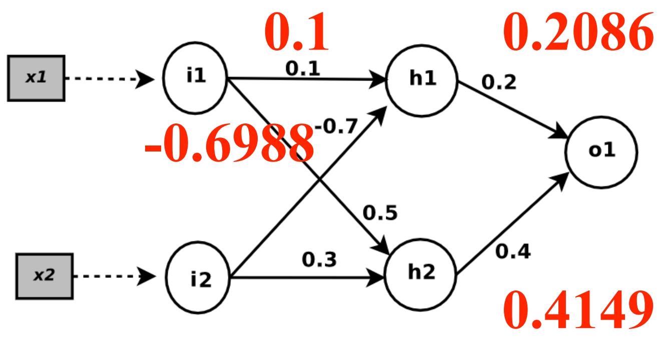
- Capa de salida:
  - Neurona o1:
    - Δo1 = a_oi (1-a_oi)(y_oi - a_oi) = 0.573 (1-0.573)(1-0.573) = 0.1044
    - **w_h1_o1** = w_h1_o1+alfa * a_h1 * Δo1 = 0.2 + 0.25 * 0.332 * 0.1044 = **0.2086**
      w_h2_o1 = w_h2_o1+alfa * a_h2 * Δo1 = 0.4 + 0.25 * 0.574 * 0.1044 = **0.4149**
  - Capa oculta
    - Δh1 = a_h1(1-a_h1) * [w_h1_o1 * Δo1] = 0.332(1-0.332) * [**0.2** * 0.1044] = 0.0046
    - **w_i1_h1** = w_i1_h1 + alfa * a_i1 * Δh1 = 0.1 + 0.25 * 0 * 0.0046 = **0.1**
      w_i2_h1 = w_i2_h1 + alfa * a_i2 * Δh1 = -0.7 + 0.25 * 1 * 0.0046 = **-0.6988**

Hacia atrás



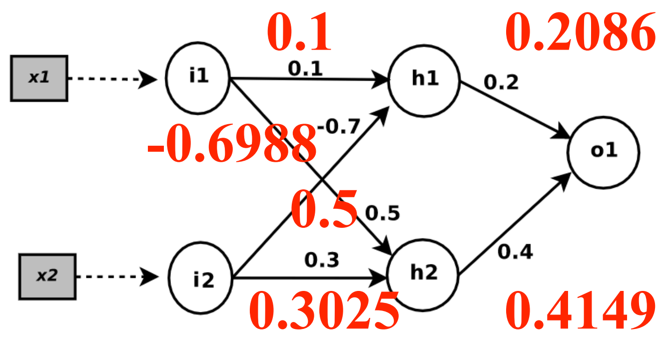
- Capa de salida:
  - Neurona o1:
    - Δo1 = a_oi (1-a_oi)(y_oi - a_oi) = 0.573 (1-0.573)(1-0.573) = 0.1044
    - **w_h1_o1** = w_h1_o1+alfa * a_h1 * Δo1 = 0.2 + 0.25 * 0.332 * 0.1044 = **0.2086**
      w_h2_o1 = w_h2_o1+alfa * a_h2 * Δo1 = 0.4 + 0.25 * 0.574 * 0.1044 = **0.4149**
  - Capa oculta
    - Δh1 = a_h1(1-a_h1) * [w_h1_o1 * Δo1] = 0.332(1-0.332) * [**0.2** * 0.1044] = 0.0046
    - **w_i1_h1** = w_i1_h1 + alfa * a_i1 * Δh1 = 0.1 + 0.25 * 0 * 0.0046 = **0.1**
      w_i2_h1 = w_i2_h1 + alfa * a_i2 * Δh1 = -0.7 + 0.25 * 1 * 0.0046 = **-0.6988**

Hacia atrás



- Capa de salida:
  - Neurona o1:
    - Δo1 = a_oi (1-a_oi)(y_oi - a_oi) = 0.573 (1-0.573)(1-0.573) = 0.1044
    - **w_h1_o1** = w_h1_o1+alfa * a_h1 * Δo1 = 0.2 + 0.25 * 0.332 * 0.1044 = **0.2086**
      w_h2_o1 = w_h2_o1+alfa * a_h2 * Δo1 = 0.4 + 0.25 * 0.574 * 0.1044 = **0.4149**
  - Capa oculta
    - Δh1 = a_h1(1-a_h1) * [w_h1_o1 * Δo1] = 0.332(1-0.332) * [**0.2** * 0.1044] = 0.0046
    - **w_i1_h1** = w_i1_h1 + alfa * a_i1 * Δh1 = 0.1 + 0.25 * 0 * 0.0046 = **0.1**
      w_i2_h1 = w_i2_h1 + alfa * a_i2 * Δh1 = -0.7 + 0.25 * 1 * 0.0046 = **-0.6988**
    - Δh2 = a_h2(1-a_h2) * [w_h2_o1 * Δo1] = 0.574(1-0.574) * [**0.4** * 0.1044] = 0.0102

Hacia atrás

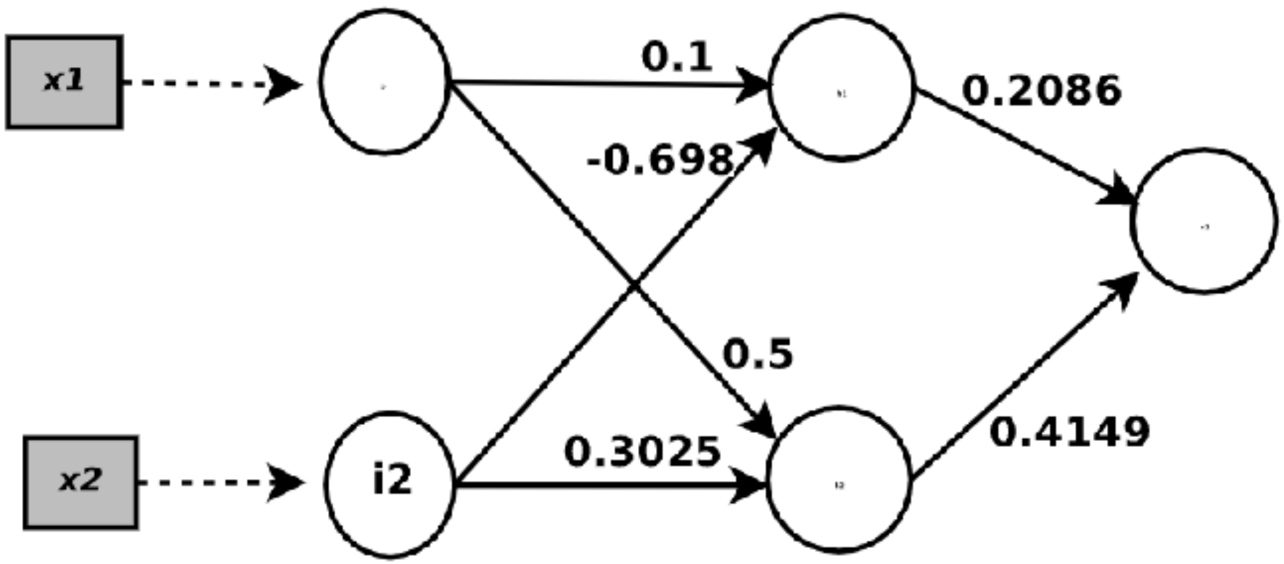


- Capa de salida:
  - Neurona o1:
    - Δo1 = a_oi (1-a_oi)(y_oi - a_oi) = 0.573 (1-0.573)(1-0.573) = 0.1044
    - **w_h1_o1** = w_h1_o1+alfa * a_h1 * Δo1 = 0.2 + 0.25 * 0.332 * 0.1044 = **0.2086**
      w_h2_o1 = w_h2_o1+alfa * a_h2 * Δo1 = 0.4 + 0.25 * 0.574 * 0.1044 = **0.4149**
  - Capa oculta
    - Δh1 = a_h1(1-a_h1) * [w_h1_o1 * Δo1] = 0.332(1-0.332) * [0.2 * 0.1044] = 0.0046
    - **w_i1_h1** = w_i1_h1 + alfa * a_i1 * Δh1 = 0.1 + 0.25 * 0 * 0.0046 = **0.1**
      w_i2_h1 = w_i2_h1 + alfa * a_i2 * Δh1 = -0.7 + 0.25 * 1 * 0.0046 = **-0.6988**
    - Δh2 = a_h2(1-a_h2) * [w_h2_o1 * Δo1] = 0.574(1-0.574) * [0.4 * 0.1044] = 0.0102
    - **w_i1_h2** = w_i1_h2 + alfa * a_i1 * Δh2 = 0.5 + 0.25 * 0 * 0.0102 = **0.5**
      w_i2_h2 = w_i2_h2 + alfa * a_i2 * Δh2 = 0.3 + 0.25 * 1 * 0.0102 = **0.3025**

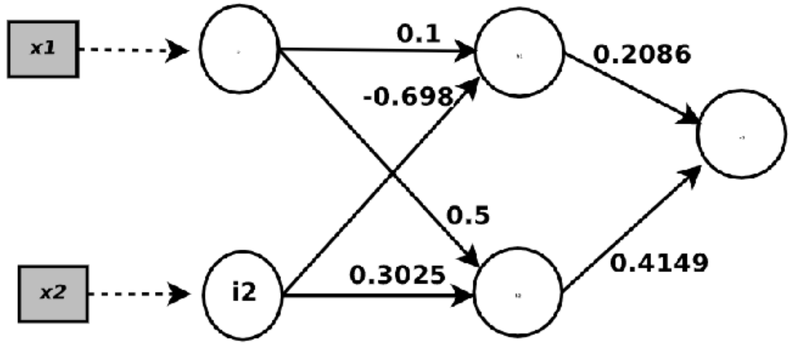
Nueva red

- Obtenemos una nueva configuración de pesos que nos van a dar un comportamiento ligeramente diferente de la red

- Para el mismo ejemplo e1, la salida sería un poco mejor: 0.576 (antes 0.573)



(nuevo) Hacia adelante



- Cálculo de las “a”s de cada neurona
- Capa de entrada
  - a_x1 = x1 = 0
    a_x2 = x2 = 1
- Capa Oculta
  - h1
    - in_h1 = 0.1 * 0 + (-0.698)*1 = -0.6988
    - a_h1 = g(-0.7) = $1/(1+e^{0.6988})$ = 0.3320
  - h2
    - in_h2 = 0.5 * 0 + 0.3025 * 1 = 0.3025
    - a_h2 = g(-0.7) = $1/(1+e^{-0.3025})$ = 0.5750
- Capa de salida
  - o1
    - in_o1 = 0.2086 * 0.3320 + 0.4149 * 0.5750 = 0.3078
    - a_h1 = g(-0.7) = $1/(1+e^{-0.3078})$ = **0.576**

Bibliografía

Bibliografía

- Russell, S. y Norvig, P. Artificial Intelligence (A modern approach) (Second edition) (Prentice Hall, 2003) (o su versión en español)
    - Cap. 20: “Statistical Learning” (disponible on-line en la web del libro)

Bibliografía

- Russell, S. y Norvig, P. Artificial Intelligence (A modern approach) (Second edition) (Prentice Hall, 2003) (o su versión en español)
    - Cap. 20: "Statistical Learning" (disponible on-line en la web del libro)
- Mitchell, T.M. Machine Learning (McGraw-Hill, 1997)
    - Cap. 4: "Artificial Neural Networks"