

Tema 5: CLASIFICADORES BASADOS EN REDES NEURONALES

MINERÍA DE DATOS

Alicia Pérez

alicia.perez@ehu.eus

Lengoia eta Sistema Informatikoak Saila

Bilboko Ingeniaritza Eskola



Alicia Pérez (UPV-EHU)

Minería de Datos

Tema 5 1 / 81

Introducción

Artificial Neural Networks (ANN, NN) [Alpaydin, 2010, Chap. 11]

- Inspiración: conexiones neuronales del cerebro
 - ▶ Neurona: unidades de procesamiento
 - ▶ Sinapsis (enlace): conexiones entre neuronas
- Procesamiento en paralelo
- Robustas al ruido y a fallos

Índice

- 1 Introducción
- 2 Perceptrón
- 3 Entrenamiento del perceptrón
- 4 Implementación de funciones booleanas
- 5 Multilayer perceptron
- 6 Entrenamiento del MLP
- 7 Observaciones sobre descenso por gradiente
- 8 Cuestiones prácticas para el entrenamiento
- 9 Reducción de la dimensionalidad
- 10 Conclusiones

Alicia Pérez (UPV-EHU)

Minería de Datos

Tema 5 2 / 81

Introducción

Aplicaciones

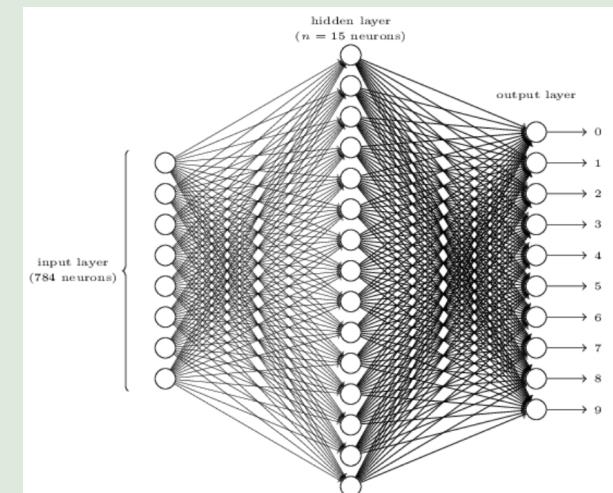
OCR: the MNIST database of handwritten digits

Imágenes de 28x28 pixels (= 784 pixels) en escala de grises ([0-1])



<http://yann.lecun.com/exdb/mnist/>

5 0 4 1 9 2



Alicia Pérez (UPV-EHU)

Minería de Datos

Tema 5 4 / 81

Alicia Pérez (UPV-EHU)

Minería de Datos

Tema 5 5 / 81

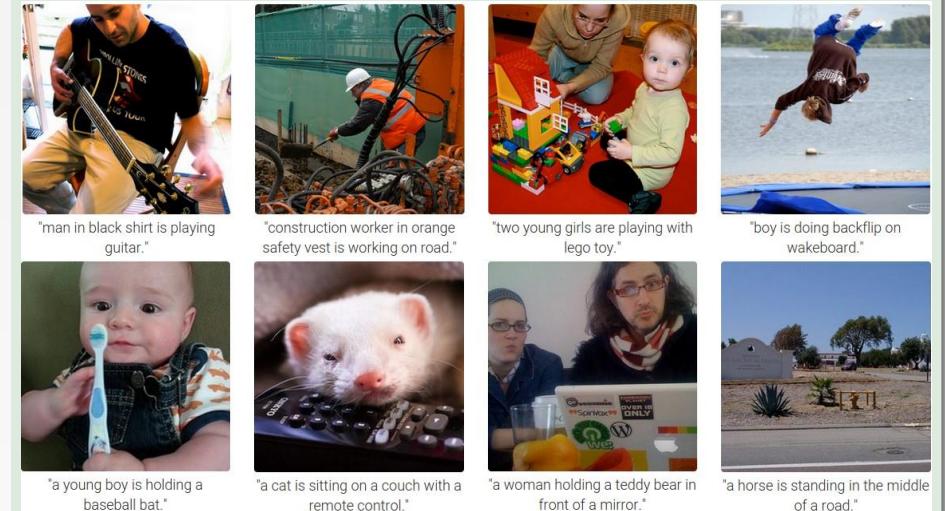
Sistemas de ayuda a la conducción



Reconocimiento e Imitación de estilos artísticos



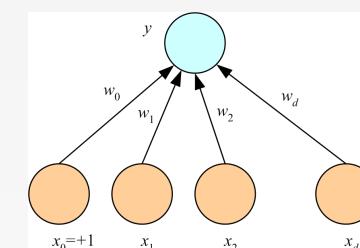
Image captioning



Perceptrón

Neurona perceptrón: estructura básica de procesamiento.

- Entradas ($x_i \in \mathbb{R}$): e.g. instancias descritas mediante d atributos
- Peso sináptico ($w_i \in \mathbb{R}$): w_i es el factor de ponderación asociado a la entrada x_i
- Procesamiento (y): $y = w_0 + \sum_j^d w_j x_j$



Procesamiento como producto escalar:

$$y = \mathbf{w}^T \cdot \mathbf{x}$$

- ▶ Definiendo una entrada artificial: $x_0 = 1$
- ▶ w_0 se conoce como *intercept value, bias o sesgo*

Fuente de la figura: [Alpaydin, 2010, Sec. 11.2]

Figura: Perceptrón

- Output ($s(y) \in \{0, 1\}$): $s(y) = 1$ if $y > 0$, otherwise $s(y) = 0$

Perceptrón

- **Output ($s(y) \in \{0, 1\}$):** y es la función de activación de la salida $s(y)$ del perceptrón (donde $y = \mathbf{w}^T \cdot \mathbf{x}$)

$$s(y) = \begin{cases} 1, & y > 0 \\ 0, & y \leq 0 \end{cases} \quad (\text{función escalón}) \quad \hat{C} = \begin{cases} C_1, & y > 0 \\ C_0, & y \leq 0 \end{cases}$$

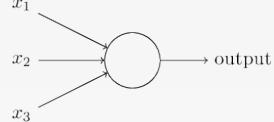


Figura: Perceptrón

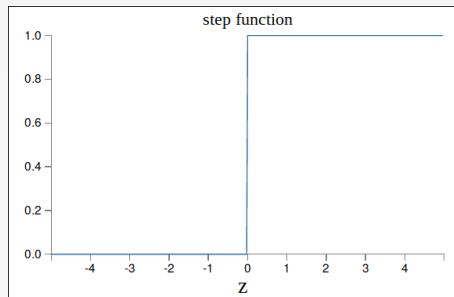
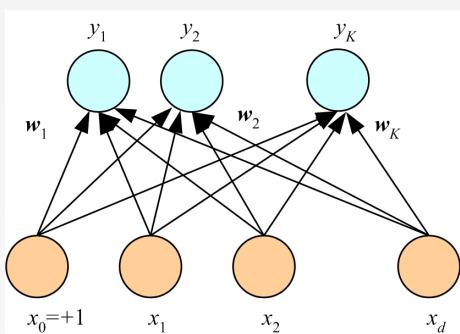


Figura: Salida del perceptrón

Perceptrón

Clasificación: k clases



Elegir $\hat{C} = C_i$ si $y_i = \max_{1 \leq m \leq k} y_m$

Figura: k perceptrones en paralelo

Cálculo matricial: $\mathbf{y} = \mathbf{W}^T \cdot \mathbf{x}$

Perceptrón

Clasificación: 2 clases

Salida bi-clásica:

$$s(y) = \begin{cases} 1, & y > 0 \\ 0, & y \leq 0 \end{cases} \quad \hat{C} = \begin{cases} C_1, & y > 0 \\ C_0, & y \leq 0 \end{cases}$$

Salida probabilística:

$$\begin{aligned} s(y) &= \text{sigmoid}(y) = \sigma(y) = \frac{1}{1 + \exp(-y)} = \\ &= \frac{1}{1 + \exp(-\mathbf{w}^T \cdot \mathbf{x})} = p(C = C_1 | \mathbf{x}) \end{aligned}$$

Perceptrón

Funciones de activación

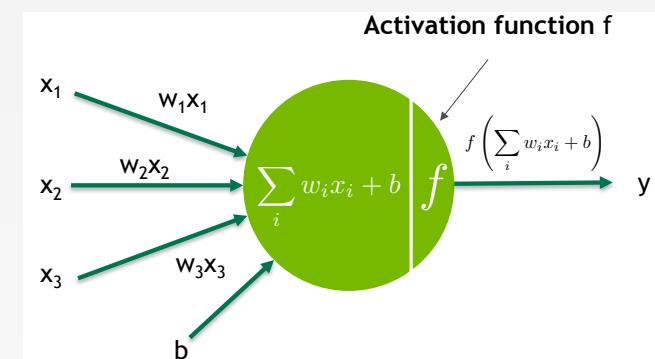
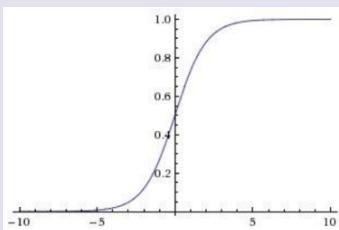


Figura: Neurona Artificial: vista detallada

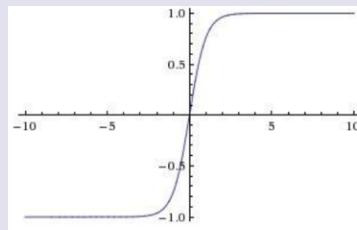
Perceptrón

Funciones de activación

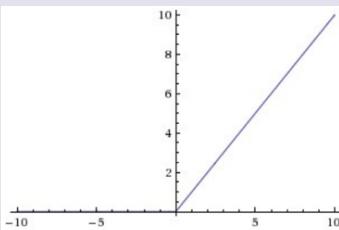
Sigmoid: $\sigma(x) = 1/(1 + e^x)$



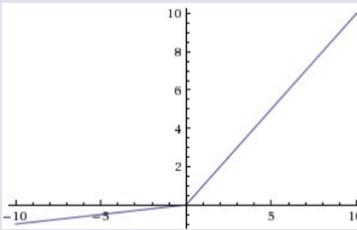
tanh



ReLU: $ReLU(x) = \max(0, x)$



Leaky ReLU



Perceptrón

Funciones de activación

```
```{r}
Let y be a 6-dimensional array of arbitrary real values
y <- c(1.1, 2.0, 3.3, 4.2, 5.5, 6.0)
Compute softmax
softmax <- exp(y)/sum(exp(y))
Note that softmax is a 6-dimensional array of values within the range (0,1)
softmax
Besides, the sum of their elements is 1
sum(softmax)
```
```

Perceptrón

Funciones de activación

Softmax: *normalized exponential function*

- Generalización de la función logística e.g. $(y_1, y_2) = (x/2, -x/2)$
- Mapea un vector k -dimensional $(\mathbf{y} \in \mathbb{R}^k)$ de valores reales arbitrarios a un vector k -dimensional $\sigma(\mathbf{y}) \in (0, 1)^k$ de valores reales en el rango $(0, 1)$ donde la suma de las componentes es 1

$$\sigma : \mathbb{R}^k \longrightarrow [0, 1]^k$$

$$\mathbf{y} \longrightarrow \sigma(\mathbf{y})$$

$$\sigma(\mathbf{y})_i = s_i = \frac{\exp(y_i)}{\sum_j^k \exp(y_j)}$$

$$0 < s_i < 1 \quad \forall i$$

$$\sum_{i=1}^k s_i = 1$$

Perceptrón

Funciones de activación

Perceptrón

Funciones de activación

Ejercicio: perceptrón simple con función de activación tanh

Obtener las ecuaciones para la actualización de pesos del perceptrón simple con función de activación tanh.

Observación:

$$\frac{d \tanh(x)}{dx} = (1 - \tanh^2(x))$$

Ejercicio: implementación

Se desea implementar un perceptrón simple con función de activación ReLU. Solución:
python

```
1 class Neuron:
2
3     def neuron_fire(self, inputs):
4         """ assume inputs and weights are 1-D arrays and bias is a number """
5         cell_body_sum = np.sum(inputs * self.weights) + self.bias
6         activation = np.max(0,cell_body_sum) # ReLU activation function
7
8         return activation
```

Entrenamiento del perceptrón

Online learning [Alpaydin, 2010, sec 11.3]

- El perceptrón define un hiper-plano que se caracteriza por los parámetros \mathbf{W}
- Entrenamiento: buscar los parámetros que mejor se ajusten a la muestra
- Estrategia de entrenamiento:
 - ▶ Inicializar los pesos con valores cercanos a 0
 - ▶ Se analizan las instancias de una en una pero en orden aleatorio
 - ▶ Se calcula la salida de la red para una instancia ¿es correcta la salida? en base al error del modelo actual, se actualizan sus parámetros
 - ▶ El perceptron se va ajustando ligeramente con cada instancia
- Motivación: ¿por qué actualizar W por instancia y no globalmente para la muestra?
 - ▶ Coste en memoria: bajo
 - ▶ Modelos dinámicos: a medida que disponemos de más datos, se pueden seguir entrenando (sin empezar de cero)

Entrenamiento del perceptrón

Actualización de parámetros:

$$\Delta w_{i,j}^t = \eta(r_i^t - y_i^t)x_j^t$$

Las actualizaciones de los parámetros dependen de:

- input x
- Learning rate η
 - ▶ η pequeño: $\Delta w_{i,j}^t$ se actualiza despacio, requiere muchas iteraciones
 - ▶ η grande: dependencia fuerte de la instancia actual (memoria a corto plazo)

Ejercicio:

- 1 Supongamos que el input $x_j > 0$
 - 1 ¿cómo es $\Delta w_{i,j}^t$ cuando la salida de la red es menor que la salida deseada?
 - 2 ¿cómo es $\Delta w_{i,j}^t$ cuando la salida de la red es mayor que la salida deseada?
- 2 Supongamos que el input $x_j < 0$
 - 1 ¿cómo es $\Delta w_{i,j}^t$ cuando la salida de la red es menor que la salida deseada?
 - 2 ¿cómo es $\Delta w_{i,j}^t$ cuando la salida de la red es mayor que la salida deseada?

Solución: [Alpaydin, 2010, Sec. 11.3, pgs 242-243]

Entrenamiento del perceptrón

Algoritmo de inferencia

```
For i = 1,...,K
    For j = 0,...,d
         $w_{ij} \leftarrow \text{rand}(-0.01, 0.01)$ 
Repeat
    For all  $(x^t, r^t) \in \mathcal{X}$  in random order
        For i = 1,...,K
             $o_i \leftarrow 0$ 
            For j = 0,...,d
                 $o_i \leftarrow o_i + w_{ij}x_j^t$ 
            For i = 1,...,K
                 $y_i \leftarrow \exp(o_i) / \sum_k \exp(o_k)$ 
            For i = 1,...,K
                For j = 0,...,d
                     $w_{ij} \leftarrow w_{ij} + \eta(r_i^t - y_i) x_j^t$ 
    Until convergence
```

Figura: Perceptron training algorithm:
Stochastic online Gradient Descent ($K > 2$)

Fuente: [Alpaydin, 2010, Sec. 11.3]

① \mathbf{W} inicialización aleatoria

② Get logits: $\mathbf{o} \leftarrow \mathbf{W} \cdot \mathbf{x}^T$

③ Get softmax: $\mathbf{y} \leftarrow \sigma(\mathbf{o})$

④ Update params:
 $\mathbf{W}^{new} \leftarrow \mathbf{W}^{prev} + \eta(\mathbf{r} - \mathbf{y})\mathbf{x}^T$

⑤ Until convergence: $\Delta \mathbf{W} \approx \mathbf{0}$

Entrenamiento del perceptrón

Actualización de parámetros:

$$\Delta w_{i,j}^t = \eta(r_i^t - y_i^t)x_j^t$$

Las actualizaciones de los parámetros dependen de:

- input x
- Learning rate η
 - ▶ η pequeño: $\Delta w_{i,j}^t$ se actualiza despacio, requiere muchas iteraciones
 - ▶ η grande: dependencia fuerte de la instancia actual (memoria a corto plazo)

Ejercicio:

- 1 Supongamos que el input $x_j > 0$
 - 1 ¿cómo es $\Delta w_{i,j}^t$ cuando la salida de la red es menor que la salida deseada?
 - 2 ¿cómo es $\Delta w_{i,j}^t$ cuando la salida de la red es mayor que la salida deseada?
- 2 Supongamos que el input $x_j < 0$
 - 1 ¿cómo es $\Delta w_{i,j}^t$ cuando la salida de la red es menor que la salida deseada?
 - 2 ¿cómo es $\Delta w_{i,j}^t$ cuando la salida de la red es mayor que la salida deseada?

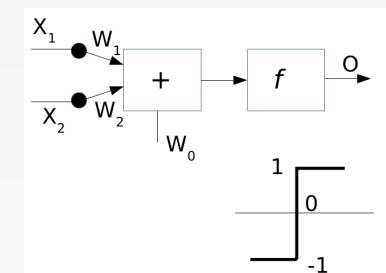
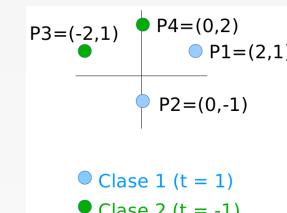
Entrenamiento del perceptrón

Ejemplo inferencia

Ejemplo de inferencia del perceptrón con función de activación signo y $\eta = 1.0$

| | x_1 | x_2 | r |
|----|-------|-------|-----|
| P1 | 2 | 1 | 1 |
| P2 | 0 | -1 | 1 |
| P3 | -2 | 1 | -1 |
| P4 | 0 | 2 | -1 |

Cuadro: Conjunto de datos: X y clase real (r)



Función de activación perceptrón:

$\text{sign}(z) :$
IF $z \leq 0$ THEN return -1
ELSE return 1

Entrenamiento del perceptrón

Ejemplo inferencia

Ejemplo de inferencia del perceptrón

- Inicialización aleatoria $\mathbf{W}^0 = [w_0^0, w_1^0, w_2^0] = [0.5, -0.7, 0.2]$
- Se presenta $\mathbf{P1} = [2, 1]$ (clase $r=1$):
 - ▶ Logits: $o = [0.5, -0.7, 0.2] \cdot [1, 2, 1]^T = -0.7$
 - ▶ Activation: $y = s(-0.7) = -1$
 $error = r - y = 1 - (-1) = 2$ (se clasifica incorrectamente)
 - ▶ Update: $\mathbf{W}^1 \leftarrow \mathbf{W}^0 + \eta(r - y) \cdot \mathbf{x}$
 $[w_0^1, w_1^1, w_2^1] = [0.5, -0.7, 0.2] + 2 \cdot [1, 2, 1] = [2.5, 3.3, 2.2]$
- Se presenta $\mathbf{P2} = [0, 2]$ (clase $r=1$)
 - ▶ Logits: $o = [2.5, 3.3, 2.2] \cdot [1, 0, 2]^T = 0.3$
 - ▶ Activation: $y = s(0.3) = 1$
 $error = r - y = 1 - 1 = 0$ (se clasifica correctamente)
- Se presenta $\mathbf{P3} = [-2, 1]$ (clase $r=-1$)
 - ▶ Logits: $o = [2.5, 3.3, 2.2] \cdot [1, -2, 1]^T = -1.9$
 - ▶ Activation: $s(-1.9) = -1$
 $error = r - y = (-1) - (-1) = 0$ (se clasifica correctamente)

Entrenamiento del perceptrón

Ejemplo inferencia

Ejemplo de inferencia del perceptrón (continuación)

- Se presenta $\mathbf{P4} = [0, 2]$ (clase $r=-1$)
 - ▶ Logits: $o = [2.5, 3.3, 2.2] \cdot [1, 0, 2]^T = 6.9$
 - ▶ Activation: $s(-1.9) = -1$
 $error = r - y = -1 - -1 = -2$ (se clasifica incorrectamente)
 - ▶ Update: get $\mathbf{W}^2 \leftarrow \mathbf{W}^1 + \eta(r - y) \cdot \mathbf{x}$
 $[w_0^2, w_1^2, w_2^2] = [2.5, 3.3, 2.2] + -2 \cdot [1, 0, 2] = [0.5, 3.3, -1.8]$
- Se vuelven a presentar los patrones $\mathbf{P1}, \mathbf{P2}, \mathbf{P3}$ y $\mathbf{P4}$.
 - ▶ $\mathbf{P1}$: R> $o_1 = \text{sum}(c(0.5, 3.3, -1.8) * c(1, 2, 1)) = 5.3 \Rightarrow y_1 = s(5.3) = 1 = r_1$
 - ▶ $\mathbf{P2}$: R> $o_2 = \text{sum}(c(0.5, 3.3, -1.8) * c(1, 0, -1)) = 2.3 \Rightarrow y_2 = s(2.3) = 1 = r_2$
 - ▶ $\mathbf{P3}$: R> $o_3 = \text{sum}(c(0.5, 3.3, -1.8) * c(1, -2, 1)) = -7.9 \Rightarrow y_3 = s(-7.9) = -1 = r_3$
 - ▶ $\mathbf{P4}$: R> $o_4 = \text{sum}(c(0.5, 3.3, -1.8) * c(1, 0, 2)) = -3.1 \Rightarrow y_4 = s(-3.1) = -1 = r_4$
- La clasificación es correcta para todos ellos, los parámetros W se mantienen estables en esta iteración: FIN retornar W : $[w_0, w_1, w_2] = [0.5, 3.3, -1.8]$

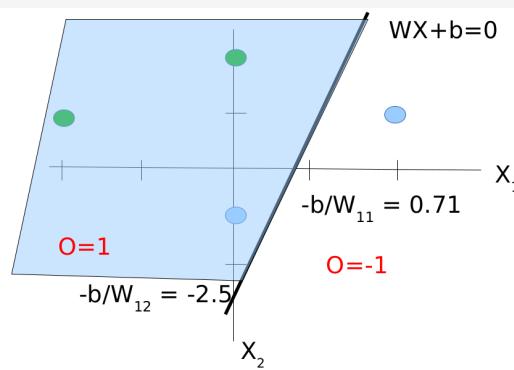
A continuación veremos, gráficamente, cada paso del entrenamiento de esta red.

Entrenamiento del perceptrón

Ejemplo inferencia

Ejemplo gráfico de inferencia del perceptrón simple:

- **Modelo** inicial: $W^0 = [-0.7, 0.2]$ y $b^0 = 0.5$
 $W^0 \mathbf{x} + b^0 = 0 \Rightarrow [-0.7, 0.2] \cdot [x_1, x_2] + 0.5 = 0$
(1) $-0.7x_1 + 0.5 = 0 \Rightarrow x_1 = -0.5/-0.7 = 0.7142857$
(2) $0.2x_2 + 0.5 = 0 \Rightarrow x_2 = -0.5/0.2 = -2.5$
- **Clasificación** inicial de los patrones:

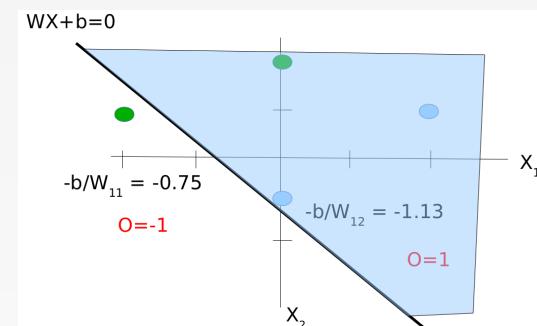


Entrenamiento del perceptrón

Ejemplo inferencia

Ejemplo gráfico de inferencia del perceptrón simple:

- **Modelo** tras la corrección: $W^1 = [3.3, 2.2]$ y $b^1 = 2.5$
 $W^1 \mathbf{x} + b^1 = 0 \Rightarrow [3.3, 2.2] \cdot [x_1, x_2] + 2.5 = 0$
(1) $3.3x_1 + 2.5 = 0 \Rightarrow x_1 = -2.5/3.3 = -0.7575758$
(2) $2.2x_2 + 2.5 = 0 \Rightarrow x_2 = -2.5/2.2 = -1.136364$
- **Clasificación** de los patrones tras la corrección:



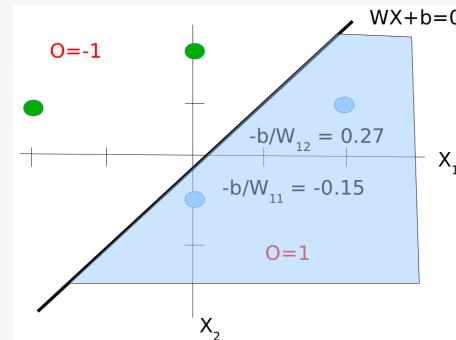
Entrenamiento del perceptrón

Ejemplo inferencia

Ejemplo gráfico de inferencia del perceptrón simple:

- **Modelo** tras la corrección: $W^2 = [3.3, -1.8]$ y $b^2 = 0.5$
 $W^T X + b = 0 \Rightarrow [3.3, -1.8] \cdot [x_1, x_2] + 0.5 = 0$
(1) $3.3 \cdot x_1 + 0.5 = 0 \Rightarrow x_1 = -0.5/3.3 = -0.1515152$
(2) $-1.8 \cdot x_2 + 0.5 = 0 \Rightarrow x_2 = 0.5/-1.8 = 0.2777778$

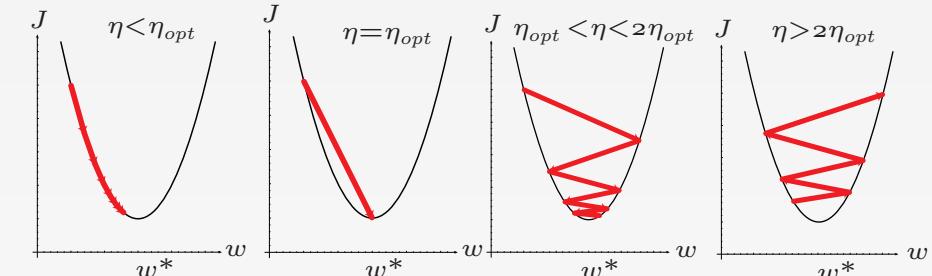
Clasificación de los patrones tras la corrección (FIN)



Entrenamiento del perceptrón

Learning rate

Fuente de la figura: [Duda et al., 2000, Sec 6.8.9]



Ratio de aprendizaje adaptativo [Alpaydin, 2010, Sec. 11.8.1]:

$$\Delta\eta = \begin{cases} +a & \text{if } E^{t+\tau} < E^t \\ -b\eta & \text{otherwise} \end{cases}$$

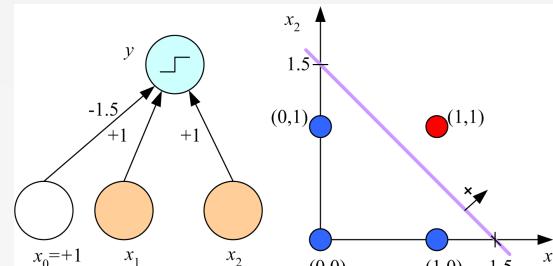
Implementación de funciones booleanas

AND

AND [Alpaydin, 2010, Sec. 11.4]

| x_1 | x_2 | r |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Cuadro: $r(x_1, x_2) = x_1 \text{ AND } x_2$



La función discriminante: $y = \sigma(x_1 + x_2 - 1.5)$

$$\begin{aligned} \mathbf{x} &= (1, x_1, x_2)^T \\ \mathbf{w} &= (-1.5, 1, 1)^T \\ y &= \sigma(\mathbf{w} \cdot \mathbf{x}^T) \end{aligned}$$

Implementación de funciones booleanas

OR

OR [Alpaydin, 2010, Sec. 11.4]

| x_1 | x_2 | r | y |
|-------|-------|-----|-----|
| 0 | 0 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 1 | |
| 1 | 1 | 1 | |

Cuadro: $r(x_1, x_2) = x_1 \text{ OR } x_2$

La función discriminante: $y = \sigma(x_1 + x_2 - 0.5)$

La función booleana OR es linealmente separable mediante la función discriminante

$y = \sigma(x_1 + x_2 - 0.5)$. La tabla muestra el valor real para cada entrada (x_1, x_2) ¿cuál es el valor y de la salida de la red?

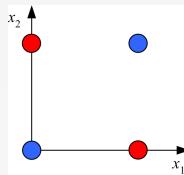
Implementación de funciones booleanas

XOR

Limitación del perceptrón: funciones que no sean linealmente separables e.g. XOR

| x_1 | x_2 | r |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Cuadro: $r(x_1, x_2) = x_1 \text{XOR} x_2$



$$\begin{array}{lll} w_0 \leq 0 & & \\ w_2 + w_1 > 0 & & \\ w_1 + w_2 > 0 & & \\ w_1 + w_2 \leq 0 & & \end{array}$$

¡irresoluble (mediante una línea)!

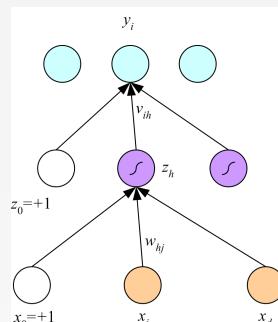
Requiere modelos más complejos: perceptrón multicapa

Multilayer perceptron

- **Limitación del perceptrón simple:** aproxima funciones lineales respecto de la entrada
- **Perceptrón multi-capa:** aproxima cualquier función (incluso XOR)
 - Topología: capas de neuronas (*hidden layers*) entre la capa de entrada y la de salida. Ejemplo: red de 2 capas (oculta y salida)

$$z_h = \sigma(\mathbf{w}_h^T \cdot \mathbf{x}) = \frac{1}{1 + \exp\left(-\sum_{j=1}^d w_{hj} x_j + w_{h0}\right)}$$

$$y_i = \mathbf{v}_i^T \cdot \mathbf{z} = \sum_{h=1}^H (v_{ih} z_h + v_{i0})$$



[Alpaydin, 2010, Sec. 11.5]

Implementación de funciones booleanas

NAND

| x_1 | x_2 | r |
|-------|-------|-----|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Cuadro: $r(x_1, x_2) = x_1 \text{NAND} x_2$

La función discriminante:

$$y = \sigma(-2x_1 - 2x_2 + 3)$$

$$\mathbf{x} = (1, x_1, x_2)^T$$

$$\mathbf{w} = (-2, -2, 3)^T$$

$$y = \sigma(\mathbf{w} \cdot \mathbf{x}^T)$$

- ¡El perceptrón implementa la puerta NAND!
- NAND es una **puerta universal**: combinando solo las puertas NAND se pueden crear todas las puertas lógicas: AND, OR, XOR...

Multilayer perceptron

Estructura MLP:

- unidades de entrada: x_j ($j = 0, \dots, d$)
- unidades ocultas: z_h ($h = 1, \dots, H$) implementan una transformación no-lineal desde el espacio d -dimensional de entrada al H -dimensional de la capa oculta
 - H es la dimensión del espacio oculto
 - z_0 bias de la capa oculta
 - w_{hj} son los pesos de la primera capa
- unidades de salida: y_i ($i = 1, \dots, K$) implementan una transformación lineal del espacio H -dimensional
 - z_0 bias de la capa oculta
 - v_{ih} son los pesos de la segunda capa

Multilayer perceptron

Universalidad del MLP

- ¡El perceptrón implementa la puerta NAND! (el mismo argumento con AND y OR)
- NAND es una **puerta universal**: combinando solo las puertas NAND se puede generar **cualquier computación**
- **Combinando** perceptrones (en redes de neuronas como el MLP) se pueden crear todas las funciones

Multilayer perceptron

Universalidad del MLP

$$x_1 \text{ XOR } x_2 = (x_1 \text{ AND } \neg x_2) \text{ OR } (\neg x_1 \text{ AND } x_2)$$

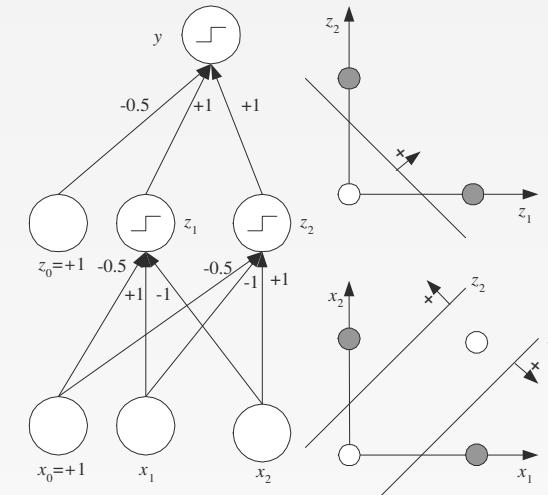
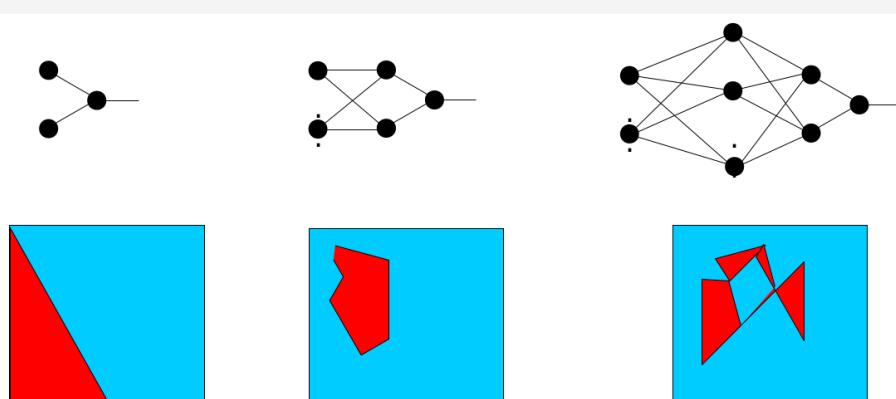


Figura: XOR implementado mediante MLP

Multilayer perceptron

Universalidad del MLP

La complejidad de las funciones que la red puede implementar depende del número de neuronas y de capas ocultas



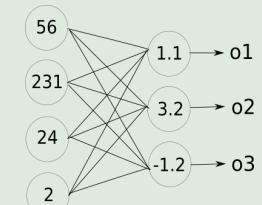
Multilayer perceptron

Universalidad del MLP

Ejercicio 1: calcular la salida del MLP para una entrada dada

- **Modelo MLP dado:** capa input ($\text{capa}^{(0)}$), capa output ($\text{capa}^{(1)}$). Con la siguiente matriz de pesos:

| $w^{(1)}$ | from=1 | from=2 | from=3 | from=4 | $b^{(1)}$ |
|-----------|--------|-----------------|--------|--------|-----------|
| to=1 | 0.2 | $w_{12} = -0.5$ | 0.1 | 2.0 | 1.1 |
| to=2 | 1.5 | 1.3 | 2.1 | 0.0 | 3.2 |
| to=3 | 0 | 0.25 | 0.2 | -0.3 | -1.2 |



- **Input:** $x = (56, 231, 24, 2) \in \mathbb{R}^4$

- **Objetivo:** calcular z

Solución: $o = W \cdot x^T + b$

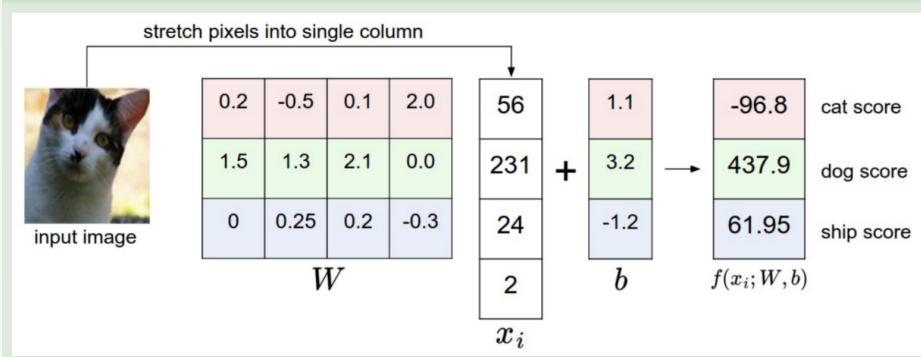
- $o_1 = (56 \cdot 0.2 + 231 \cdot (-0.5) + 24 \cdot 0.1 + 2 \cdot 2.0) + 1.1 = -96.80$
- $o_2 = (56 \cdot 1.5 + 231 \cdot 1.3 + 24 \cdot 2.1 + 2 \cdot 0.0) + 3.2 = 437.90$
- $o_3 = (56 \cdot 0.0 + 231 \cdot 0.25 + 24 \cdot 0.2 + 2 \cdot (-0.3)) + (-1.2) = 60.75$

Resultado: $o = (-96.80, 437.90, 60.75) \in \mathbb{R}^3$

Multilayer perceptron

Universalidad del MLP

Solución: calcular la salida del MLP para una entrada dada



Entrenamiento del MLP

$$z_h = \sigma(w_h^T \cdot x) = \frac{1}{1 + \exp\left(-\sum_{j=1}^d w_{hj}x_j + w_{h0}\right)} \quad 1 \leq h \leq H$$

$$y_i = v_i^T \cdot z = \sum_{h=1}^H (v_{ih}z_h + v_{i0})$$

Objetivo: minimizar la función de error (que se propaga desde la salida y hacia atrás, *backpropagation*)

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}$$

Resultado: en cada iteración del entrenamiento descubrimos la magnitud y la dirección en la que hay que actualizar los pesos (w_{hj}) para minimizar la función de error.

Multilayer perceptron

Universalidad del MLP

Ejercicio 2: calcular la salida del MLP para una entrada dada con función de activación sigmoide

- **Modelo MLP dado:** Ejercicio 1
- **Input:** Ejercicio 1
- **Objetivo:** calcular $\mathbf{y} = \sigma(\mathbf{o})$

Solución: Se aplica la función de activación **función sigmoide** a la salida del ejercicio anterior $\mathbf{o} = (-96.8, 437.9, 60.75) \in \mathbb{R}^3$

- $y_1 = 1/(1+\exp(-96.8)) = 9.126288e-43 \approx 0$
- $y_2 = 1/(1+\exp(-437.9)) = 1 \approx 1$
- $y_3 = 1/(1+\exp(60.75)) = 4.136283e-27 \approx 0$

Resultado: $\mathbf{y} = \sigma(\mathbf{o}) = (0, 1, 0)$, entonces, esta clasifica la instancia como C_2

Entrenamiento del MLP

Nonlinear regression (with single output)

Nonlinear Regression (with single output): [Alpaydin, 2010, Sec. 11.7.1]

$$E(\mathbf{W}, \mathbf{v} | \mathcal{X}) = \frac{1}{2} \sum_t (r^t - y^t)^2$$

$$y^t = \sum_{h=1}^H v_h z_h^t + v_0$$

$$\Delta v_h = \sum_t (r^t - y^t) z_h^t$$

Forward *Backward*

$$z_h = \text{sigmoid}(w_h^T \cdot x)$$

$$\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}}$$

$$= -\eta \sum_t \frac{\partial E}{\partial y^t} \frac{\partial y^t}{\partial z_h^t} \frac{\partial z_h^t}{\partial w_{hj}}$$

$$= -\eta \sum_t -(r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t$$

$$= \eta \sum_t (r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t$$

Entrenamiento del MLP

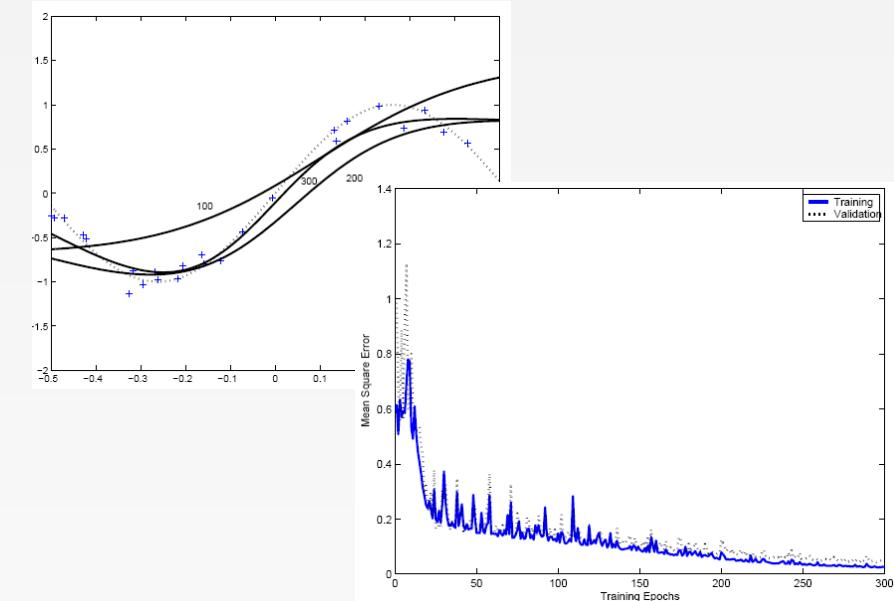
Nonlinear regression (with single output)

Cuestiones prácticas:

- Normalización de la entrada (media=0, varianza=1) antes de empezar.
 - ▶ Motivación: η único para todas las entradas, es conveniente que las entradas guarden la misma escala.
- Inicialización de pesos w_{hj} y v_h : valores pequeños aleatorios en el rango [-0.01, 0.01]
- Estrategias de entrenamiento: actualizamos la magnitud y la dirección en la que hay que actualizar los pesos (w_{hj}) para minimizar la función de error ¿cómo?
 - ▶ Batch learning: los cambios se acumulan según se exploran las instancias y se actualizan después de haber explorado el conjunto de entrenamiento completo.
Observación (en página anterior): \sum_t
 - ▶ Online learning: los pesos se actualizan con cada instancia se conoce como **Stochastic Gradient Descend (SGD)**. Ventaja: convergencia rápida. Requisitos:
 - ★ explorar las instancias en orden aleatorio
 - ★ elegir η pequeño
- Epoch: una iteración completa sobre todas las instancias del conjunto de entrenamiento

Entrenamiento del MLP

Nonlinear regression (with single output)

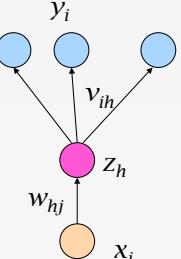


Entrenamiento del MLP

Nonlinear regression (with multiple outputs)

Nonlinear Regression (with multiple outputs): [Alpaydin, 2010, Sec. 11.7.1]

Se infieren múltiples problemas de regresión simultáneamente



- Outputs: $y_i^t = \sum_{h=1}^H (v_{ih} z_h^t + v_{i0})$

- Error: $E(\mathbf{W}, \mathbf{V} | \mathcal{X}) = \frac{1}{2} \sum_t \sum_i (r_i^t - y_i^t)^2$

- Batch update:

$$\Delta v_{ih} = \eta \sum_t (r_i^t - y_i^t) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) v_{ih} \right] z_h^t (1 - z_h^t) x_j^t$$

Entrenamiento del MLP

Nonlinear regression (with multiple outputs)

Initialize all v_{ih} and w_{hj} to $\text{rand}(-0.01, 0.01)$

Repeat

For all $(x^t, r^t) \in \mathcal{X}$ in random order

For $h = 1, \dots, H$

$z_h \leftarrow \text{sigmoid}(w_h^T x^t)$

For $i = 1, \dots, K$

$y_i = \mathbf{v}_i^T \mathbf{z}$

For $i = 1, \dots, K$

$\Delta \mathbf{v}_i = \eta (r_i^t - y_i^t) \mathbf{z}$

For $h = 1, \dots, H$

$\Delta \mathbf{w}_h = \eta (\sum_i (r_i^t - y_i^t) v_{ih}) z_h (1 - z_h) x^t$

For $i = 1, \dots, K$

$\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta \mathbf{v}_i$

For $h = 1, \dots, H$

$\mathbf{w}_h \leftarrow \mathbf{w}_h + \Delta \mathbf{w}_h$

Until convergence

Entrenamiento del MLP

Nonlinear regression (with multiple outputs)

Ejercicio: calcular el error de un MLP dado

Disponemos de tres instancias, con su salida deseada (\mathbf{r}^t) y la salida estimadas por una red MLP (\mathbf{y}^t). Deseamos calcular el error LMS de la red.

| t | \mathbf{r}^t | \mathbf{y}^t | E^t |
|-----|----------------|----------------|-------|
| 1 | (1,0,0) | (0.8,0.1,0.1) | |
| 2 | (0,1,0) | (0.2,0.7,0.1) | |
| 3 | (0,0,1) | (0.2,0.2,0.6) | |
| E | | | |

Solución:

$$\begin{aligned} E^t &= \frac{1}{2} \sum_i (r_i^t - y_i^t)^2 \\ E(\mathbf{W}, \mathbf{V} | \mathcal{X}) &= \sum_t E^t \end{aligned}$$

Entrenamiento del MLP

Nonlinear regression (single output using multiple hidden layers)

Regression (una sola salida) empleando 2 capas ocultas: [Alpaydin, 2010, Sec. 11.7.4]

$$\begin{aligned} z_{1h} &= \text{sigmoid}(\mathbf{w}_{1h}^T \cdot \mathbf{x}) = \text{sigmoid} \left(\sum_{j=1}^d w_{1hj} x_j + w_{1h0} \right) \quad h = 1, \dots, H_1 \\ z_{2h} &= \text{sigmoid}(\mathbf{w}_{2l}^T \cdot \mathbf{z}_1) = \text{sigmoid} \left(\sum_{j=1}^{H_1} w_{2lh} z_{1h} + w_{2l0} \right) \quad l = 1, \dots, H_2 \\ y &= \mathbf{v}^T \cdot \mathbf{z}_2 = \sum_{l=1}^{H_2} v_l z_{2l} + v_0 \end{aligned}$$

Entrenamiento del MLP

Clasificación con k clases

Asignar una de las K clases posibles [Alpaydin, 2010, Sec. 11.7.3]

$$\begin{aligned} o_i^t &= \sum_{h=1}^H (v_{ih} z_h^t + v_{i0}) \quad 1 \leq i \leq K \\ y_i^t &= \text{softmax}(o_i^t) = \frac{\exp(o_i^t)}{\sum_{k=1}^K \exp(o_k^t)} \approx P(C_i | \mathbf{x}^t) \quad (\text{interpretación probabilística}) \end{aligned}$$

Entrenamiento:

- Función de error (objetivo minimizar): $E(\mathbf{W}, \mathbf{V} | \mathcal{X}) = - \sum_t \sum_{i=1}^K r_i^t \log(y_i^t)$
- Actualización de parámetros mediante descenso por gradiente:

$$\begin{aligned} \Delta v_{ih} &= \eta \sum_t (r_i^t - y_i^t) z_h^t \\ \Delta w_{hj} &= \eta \sum_t \left[\sum_i (r_i^t - y_i^t) v_{ih} \right] z_h^t (1 - z_h^t) x_j^t \end{aligned}$$

Entrenamiento del MLP

Nonlinear regression (single output using multiple hidden layers)

Ejercicio: Regression (una sola salida) empleando 3 capas ocultas

$$\begin{aligned} z_{1h} &= \text{sigmoid}(\mathbf{w}_{1h}^T \cdot \mathbf{x}) = \text{sigmoid} \left(\sum_{j=1}^d w_{1hj} x_j + w_{1h0} \right) \quad h = 1, \dots, H_1 \\ z_{2h} &= \text{sigmoid}(\mathbf{w}_{2l}^T \cdot \mathbf{z}_1) = \text{sigmoid} \left(\sum_{j=1}^{H_1} w_{2lh} z_{1h} + w_{2l0} \right) \quad l = 1, \dots, H_2 \\ z_{3h} &= \text{sigmoid}(\mathbf{w}_{3l}^T \cdot \mathbf{z}_2) = \text{sigmoid} \left(\sum_{j=1}^{H_2} w_{3lh} z_{2h} + w_{3l0} \right) \quad l = 1, \dots, H_3 \\ y &= \mathbf{v}^T \cdot \mathbf{z}_3 = \sum_{l=1}^{H_3} v_l z_{3l} + v_0 \end{aligned}$$

Entrenamiento del MLP

Clasificación (2 clases empleando múltiples capas ocultas)

Ejercicio: clasificación (2 clases) empleando 2 capas ocultas

Revisar: Regression (una sola salida) empleando 2 capas ocultas

$$z_{1h} = \text{sigmoid}(\mathbf{w}_{1h}^T \cdot \mathbf{x}) = \text{sigmoid} \left(\sum_{j=1}^d w_{1hj} x_j + w_{1h0} \right) \quad h = 1, \dots, H_1$$

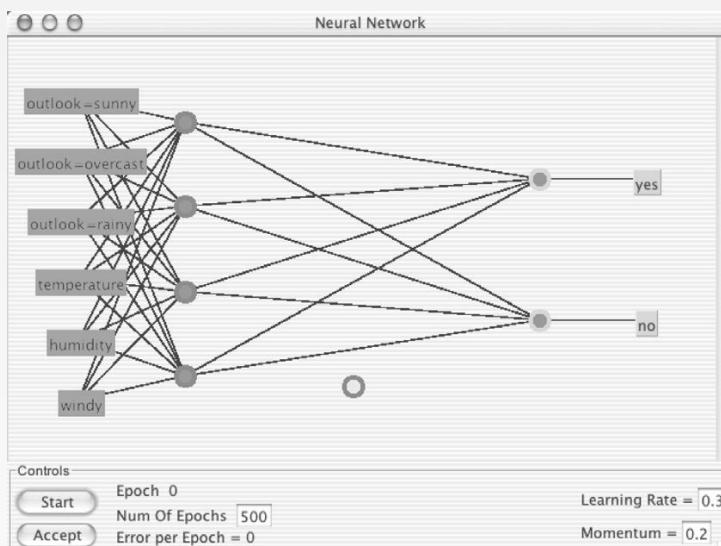
$$z_{2l} = \text{sigmoid}(\mathbf{w}_{2l}^T \cdot z_{1h}) = \text{sigmoid} \left(\sum_{j=1}^{H_1} w_{2lj} z_{1h} + w_{2l0} \right) \quad l = 1, \dots, H_2$$

$$o = \mathbf{v}^T \cdot \mathbf{z}_2$$

y = completar

Entrenamiento del MLP

Weka



[Witten et al., 2011, sec 6.4 y sec 11.4]

Entrenamiento del MLP

Clasificación (2 clases empleando múltiples capas ocultas)

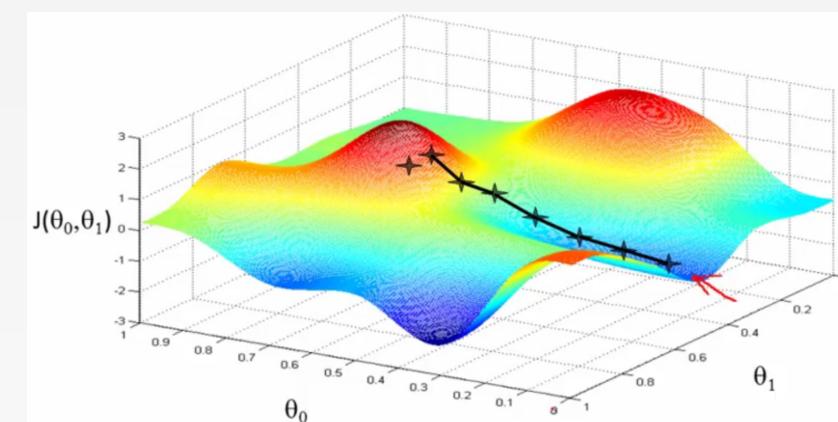
Ejercicio: implementación feedforward

Se desea implementar feedforward en una red con 2 capas ocultas y un único nodo salida.

Solución:

```
1 # forward-pass of a 3-layer neural network
2 f = lambda x: np.max(0,x)      # activation function (ReLU)
3 x = np.random.rand(3,1)         # random input vector of three numbers (3x1)
4 h1 = f(np.dot(W1, x) + b1)     # calculate first hidden layer activations (4x1)
5 h2 = f(np.dot(W2, h1) + b2)     # calculate second hidden layer activations (4x1)
6 out = f(np.dot(W3, h2) + b3)    # output neuron (1x1)
```

Observaciones sobre descenso por gradiente



Observaciones sobre descenso por gradiente

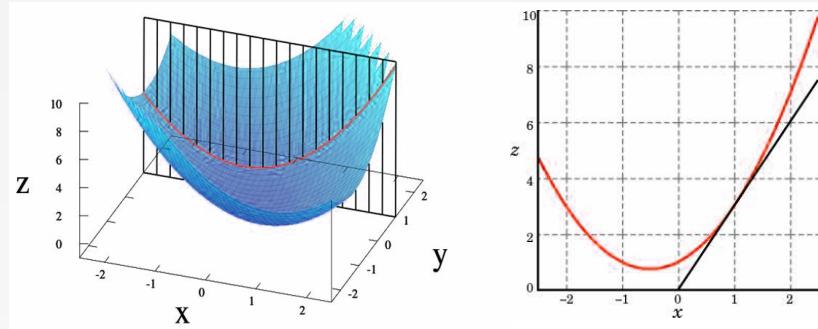


Figura: Descenso por gradiente: intervienen derivadas parciales

Observaciones sobre descenso por gradiente

Ejemplo: dado $\eta = 0.1$ aplica descenso por gradiente a $E = w^2 + 1$

Supongamos que en una neurona se comete el siguiente error: $E = w^2 + 1$. Se desea aplicar [descenso por gradiente](#) para buscar el w que menor E ofrezca.

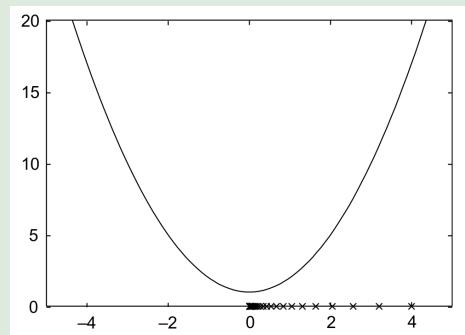


Figura: Representación de E respecto de w

Solución:

El [descenso por gradiente](#) es un proceso iterativo:

$$w_i^{new} \leftarrow w_i^{old} - \eta \frac{\partial E}{\partial w_i}$$

Observación: $\frac{dE}{dw} = 2w$ (pendiente)

- (0) $w^{(0)} = 4$ (inicialización: al azar)
- (1) $w^{(1)} = 4 - 0.1 * (2*4) = 3.2$
- (2) $w^{(2)} = 3.2 - 0.1 * (2*3.2) = 2.56$
- (3) $w^{(3)} = 2.56 - 0.1 * (2*2.56) = 2.048$
- ...
- (K) $w^{(K)} = 0$

Observaciones sobre descenso por gradiente

Ejemplo: recordatorio

$$f(x, y) = x \cdot y$$

- Consideraremos el punto $(x, y) = (4, -3)$
- $f(4, -3) = -12$
- Derivadas parciales en el punto:

$$\frac{\partial f}{\partial x} \Big|_{(x,y)=(4,-3)} = y \Big|_{(x,y)=(4,-3)} = -3 \quad \frac{\partial f}{\partial y} \Big|_{(x,y)=(4,-3)} = x \Big|_{(x,y)=(4,-3)} = 4$$

- Gradiante: $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$

Observaciones sobre descenso por gradiente

Ejemplo: dado $\eta = 0.1$ aplica descenso por gradiente a $E = w^2 + 1$

Supongamos que en una neurona se comete el siguiente error: $E = w^2 + 1$. Se desea aplicar [descenso por gradiente](#) para buscar el w que menor E ofrezca.

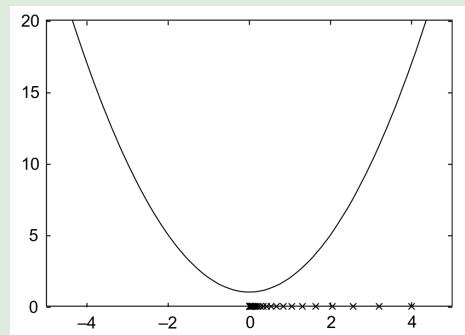


Figura: Representación de E respecto de w

Solución:

El [descenso por gradiente](#) es un proceso iterativo:

$$w_i^{new} \leftarrow w_i^{old} - \eta \frac{\partial E}{\partial w_i}$$

Observación: $\frac{dE}{dw} = 2w$ (pendiente)

- (0) $w^{(0)} = 4$ (inicialización: al azar)
- (1) $w^{(1)} = 4 - 0.1 * (2*4) = 3.2$
- (2) $w^{(2)} = 3.2 - 0.1 * (2*3.2) = 2.56$
- (3) $w^{(3)} = 2.56 - 0.1 * (2*2.56) = 2.048$
- ...
- (K) $w^{(K)} = 0$

Observaciones sobre descenso por gradiente

Ejemplo: descenso por gradiente para neurona con función de activación sigmoide

Descenso por gradiente para el caso de una neurona con función de activación sigmoide (logits: $z = w \cdot x + b$; output: $y = \sigma(z) = 1/(1 - e^{-z})$).

- Error: $E = \sum_t E^t = \frac{1}{2} \sum_t (r^t - y^t)^2$

- Derivada del error:

$$\frac{\partial E^t}{\partial w_i} = \frac{\partial E}{\partial y^t} \frac{\partial y^t}{\partial z^t} \frac{\partial z^t}{\partial w_i} = (r^t - y^t) \left[\frac{-e^{-z^t}}{(1 - e^{-z^t})^2} \right] x^t = (r^t - y^t) \sigma(z^t) (1 - \sigma(z^t)) x^t$$

- Actualización de parámetros (pesos y bias):

$$w_k^{new} \leftarrow w_k^{old} - \eta \frac{\partial E}{\partial w_k}$$

$$b_i^{new} \leftarrow b_i^{old} - \eta \frac{\partial E}{\partial b_i}$$

Cuestiones prácticas para el entrenamiento

Acelerar la convergencia

[Alpaydin, 2010, sec 11.8.1]

- Momentum
- Adaptive learning rate

Cuestiones prácticas para el entrenamiento

Overfitting

Ejercicio 2: complejidad

Consideremos una red neuronal de 4 capas, con 784, 16, 16 y 10 neuronas respectivamente. Se desea calcular el número total de parámetros que habrá que ajustar en el proceso de entrenamiento desglosado como sigue:

- número de pesos (W)
- número de bias (b)

Cuestiones prácticas para el entrenamiento

Overfitting

Ejercicio 1: complejidad del MLP

Calcula el orden de complejidad (\mathcal{O}) del entrenamiento de un MLP con la siguiente estructura:

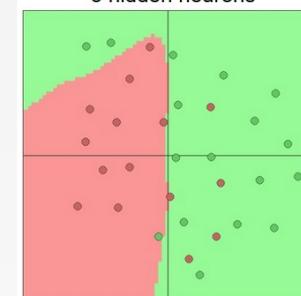
- inputs: d
- neuronas ocultas: H
- salidas: K

Tomando en cuenta que se desean realizar n epochs.

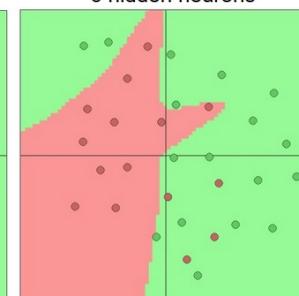
Cuestiones prácticas para el entrenamiento

Overfitting

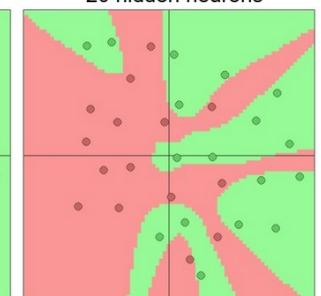
3 hidden neurons



6 hidden neurons



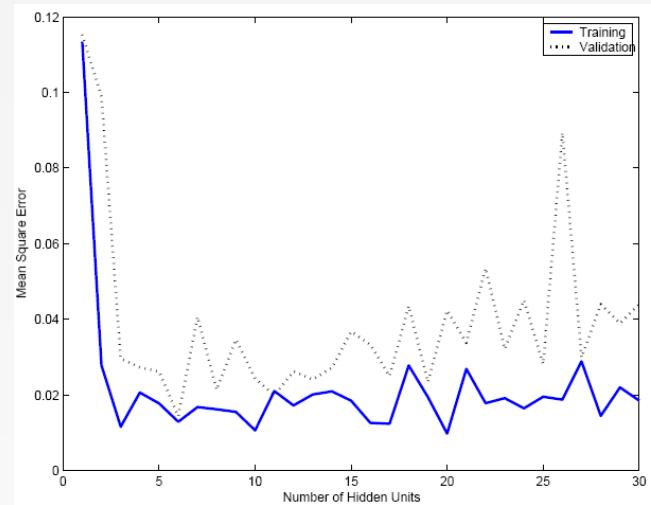
20 hidden neurons



Cuestiones prácticas para el entrenamiento

Overfitting

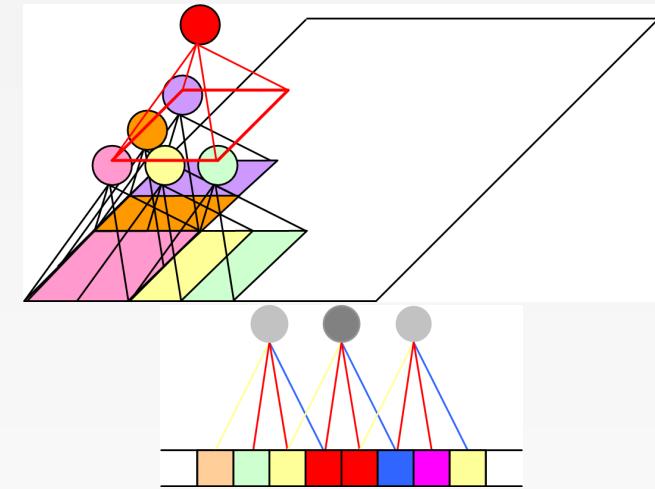
El error del train decremente o se mantiene estable aumentando el número de neuronas, sin embargo, el error del dev aumenta [Alpaydin, 2010, sec 11.8.2].



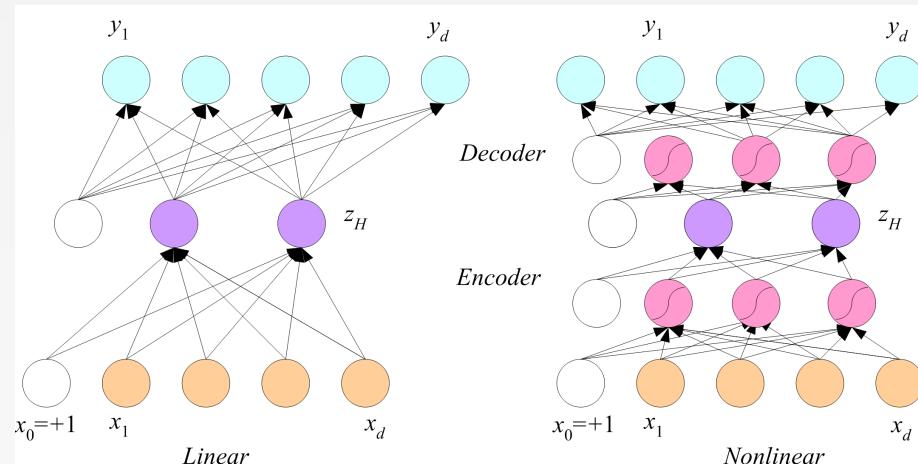
Cuestiones prácticas para el entrenamiento

Modelado explícito de estructura local

Aprovechar la estructura local para facilitar el entrenamiento: quizás las unidades de la capa oculta no estén conectadas a todas las entradas e.g. visión, ASR, etc.

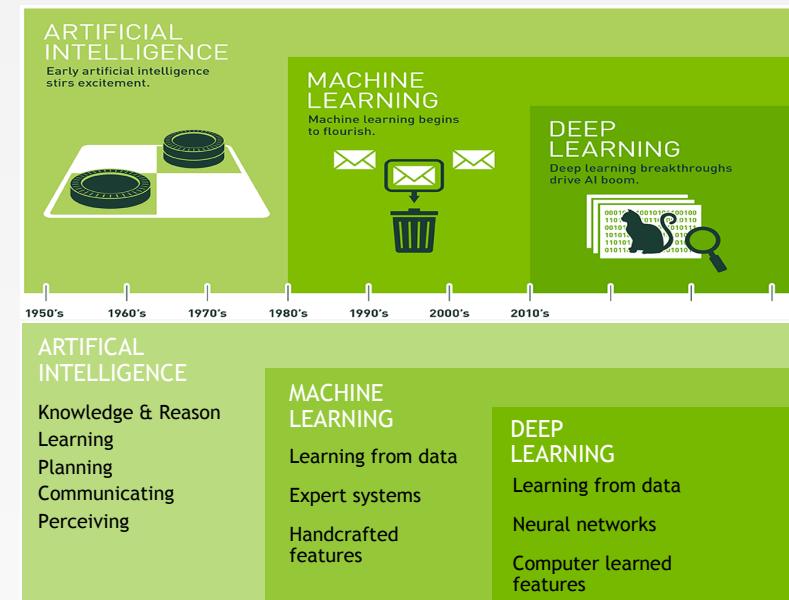


Reducción de la dimensionalidad

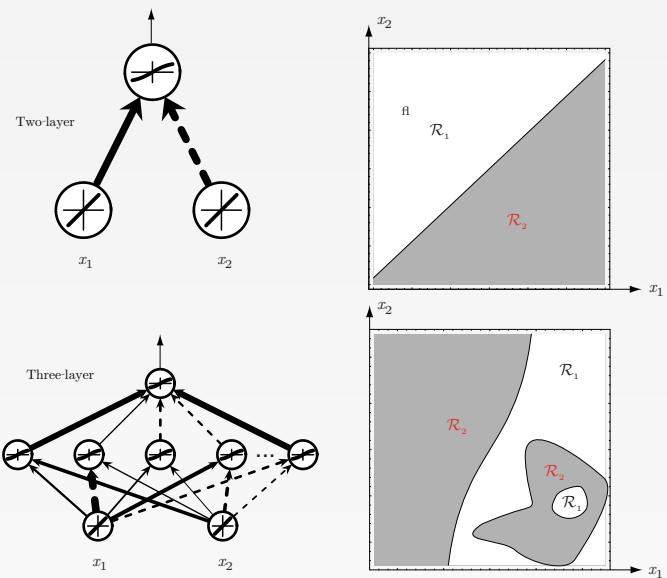


Fuente de la figura: [Alpaydin, 2010, Sec. 11.11]

Conclusiones



Conclusiones



Conclusiones

Error de entrenamiento:

$$J(\mathbf{w}) = \frac{1}{2} \|r - y\| = \frac{1}{2} \sum_i^{\dim(r)} (r_i - y_i)^2$$

Actualización de los pesos:

$$\Delta(\mathbf{w}) = -\eta \frac{\partial J}{\partial \mathbf{w}} \quad (\text{vectorial})$$

$$\Delta(w_{pq}) = -\eta \frac{\partial J}{\partial w_{pq}} \quad (\text{componentes})$$

Conclusiones

- ➊ El perceptrón simple permite aproximar funciones lineales
- ➋ El MLP simple permite aproximar cualquier función
- ➌ Estrategias de entrenamiento: batch vs. online learning
- ➍ Funciones de activación (recomendación: ReLu): regresión vs clasificación
- ➎ Back-propagation
- ➏ Estructuras de redes:
 - ▶ Salida única vs. múltiples salidas
 - ▶ Mono-capa vs. multi-capa
 - ▶ Completamente conectadas vs estructura local
 - ▶ Feedforward vs. Recurrentes
 - ▶ Selección de atributos: *autoassociation encoder-decoder*
- ➐ Cuestiones prácticas:
 - ▶ Normalización de la entrada
 - ▶ Inicialización de pesos
 - ▶ Selección η
 - ▶ Overfitting

Conclusiones

Algorithm 1 (Stochastic backpropagation)

```

1 begin initialize network topology (# hidden units),  $\mathbf{w}$ , criterion  $\theta$ ,  $\eta$ ,  $m \leftarrow 0$ 
2 do  $m \leftarrow m + 1$ 
3    $\mathbf{x}^m \leftarrow$  randomly chosen pattern
4    $w_{ij} \leftarrow w_{ij} + \eta \delta_j x_i; \quad w_{jk} \leftarrow w_{jk} + \eta \delta_k y_j$ 
5   until  $\nabla J(\mathbf{w}) < \theta$ 
6 return  $\mathbf{w}$ 
7 end

```

Fuente: [Duda et al., 2000, Chap. 6]

Conclusiones

Algorithm 2 (Batch backpropagation)

```
1 begin initialize network topology (# hidden units), w, criterion  $\theta$ ,  $\eta$ ,  $r \leftarrow 0$ 
2   do  $r \leftarrow r + 1$  (increment epoch)
3      $m \leftarrow 0$ ;  $\Delta w_{ij} \leftarrow 0$ ;  $\Delta w_{jk} \leftarrow 0$ 
4     do  $m \leftarrow m + 1$ 
5        $\mathbf{x}^m \leftarrow$  select pattern
6        $\Delta w_{ij} \leftarrow \Delta w_{ij} + \eta \delta_j x_i$ ;  $\Delta w_{jk} \leftarrow \Delta w_{jk} + \eta \delta_k y_j$ 
7     until  $m = n$ 
8      $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$ ;  $w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$ 
9   until  $\nabla J(\mathbf{w}) < \theta$ 
10  return w
11 end
```

Fuente: [Duda et al., 2000, Chap. 6]

Conclusiones

Algorithm 3 (Stochastic backpropagation with momentum)

```
1 begin initialize topology (# hidden units), w, criterion,  $\alpha (< 1)$ ,  $\theta$ ,  $\eta$ ,  $m \leftarrow 0$ ,  $b_{ji} \leftarrow 0$ ,  $b_{kj} \leftarrow 0$ 
2   do  $m \leftarrow m + 1$ 
3      $\mathbf{x}^m \leftarrow$  randomly chosen pattern
4      $b_{ji} \leftarrow \eta \delta_j x_i + \alpha b_{ji}$ ;  $b_{kj} \leftarrow \eta \delta_k y_j + \alpha b_{kj}$ 
5      $w_{ji} \leftarrow w_{ji} + b_{ji}$ ;  $w_{kj} \leftarrow w_{kj} + b_{kj}$ 
6   until  $\nabla J(\mathbf{w}) < \theta$ 
```

Fuente: [Duda et al., 2000, Chap. 6]

Bibliografía I

- ▶ Alpaydin, E. (2010).
Introduction to Machine Learning.
MIT Press.
- ▶ Duda, R. O., Hart, P. E., and Stork, D. G. (2000).
Pattern Classification.
Wiley-Interscience.
- ▶ Witten, I. H., Frank, E., and Hall, M. A. (2011).
Data Mining: Practical Machine Learning Tools and Techniques.
The Morgan Kaufmann Series in Data Management Systems, 3rd edition.

Parte II

Apéndice

Índice

11 Regresión vs Clasificación

12 Taxonomía

- Taxonomía según estructura
- RNN
- Taxonomía según modo de inferencia

13 Motivación: neurona con función de activación sigmoide

14 Multilayer perceptron

- Universalidad del MLP
- Backpropagation

15 Overfitting

16 Concluding remarks

Regresión vs Clasificación

$$y = \mathbf{w}^T \cdot \mathbf{x}$$

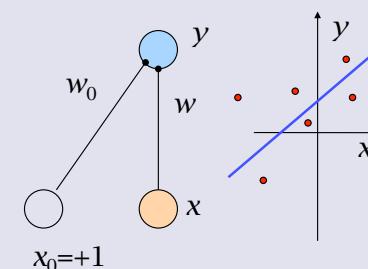


Figura: Regresión lineal

$$s = \sigma(y)$$

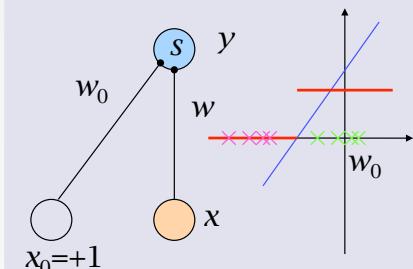


Figura: Clasificación

Taxonomía

Taxonomía según estructura

Taxonomía según **estructura** o topología:

- **Feedforward:** redes de propagación hacia delante.
 - ▶ Todas las conexiones entre las neuronas que componen la red son hacia delante.
 - ▶ Hay buenos algoritmos para su entrenamiento.
 - ▶ Pueden ser mono-capas o multi-capas (*multi-layer*)

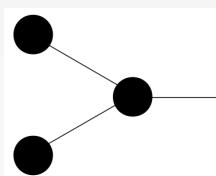


Figura: Monocapa

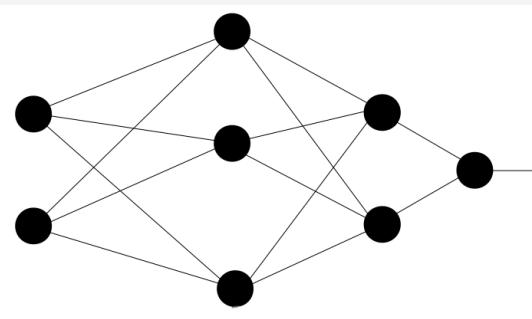


Figura: Multicapa

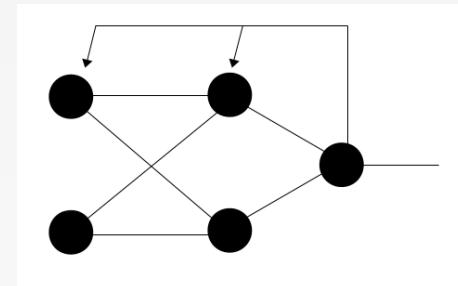
- Redes recurrentes

Taxonomía

Taxonomía según estructura

- **Redes recurrentes**

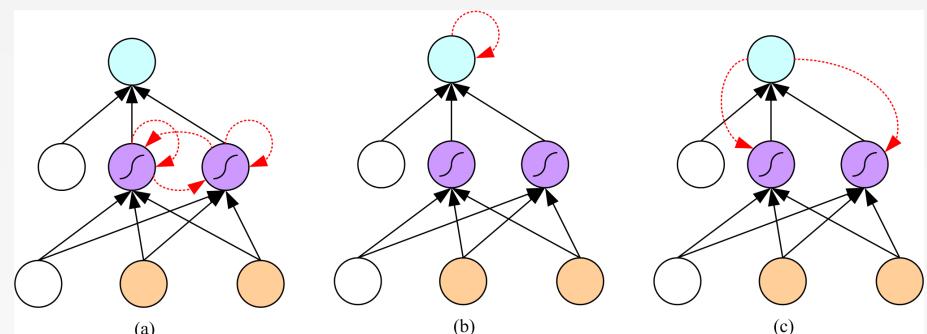
- ▶ Algunas de las conexiones van hacia atrás en la topología, generando ciclos.
- ▶ Con el feedback representa mejor la conexión neuronal del cerebro.
- ▶ Muy prometedor, se piensa que tienen más capacidad de aprendizaje que las redes feedforward.
- ▶ Pero, aún no hay buenos algoritmos para su entrenamiento.



Taxonomía

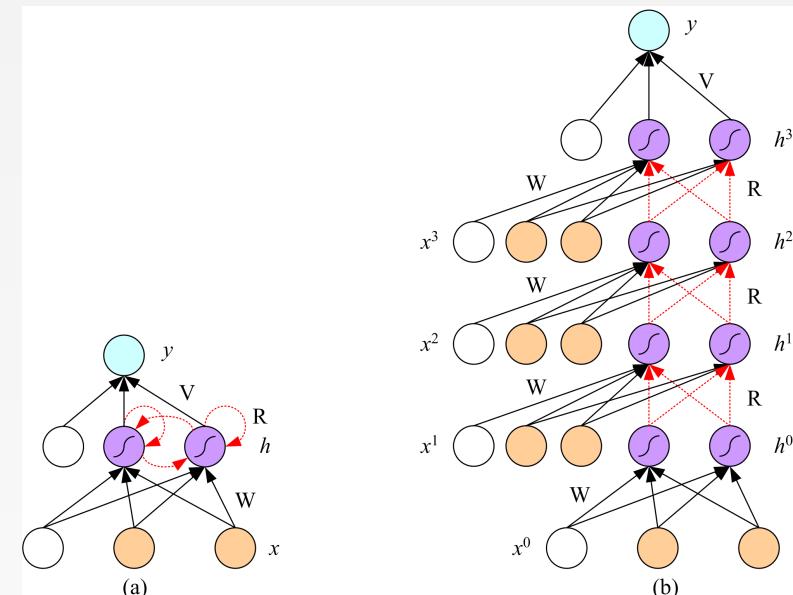
RNN

Recurrent Neural Networks (RNN)



Taxonomía

RNN



Taxonomía

Taxonomía según modo de inferencia

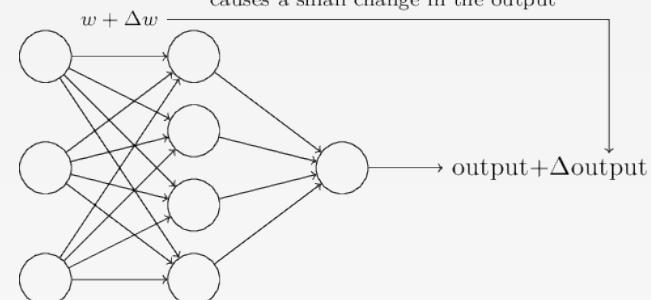
Taxonomía en cuanto al **modo de aprendizaje**.

- **Aprendizaje supervisado**: utilizan un conjunto de entrenamiento previamente clasificado (incluye una variable clase o variable dependiente).
- **Aprendizaje no supervisado (auto-organizativas)**: no necesitan de dicho conjunto preclasificado (no incluye una variable clase o dependiente).

Motivación: neurona con función de activación sigmoidal

¿Por qué usar neurona con función de activación sigmoidal?

small change in any weight (or bias)
causes a small change in the output



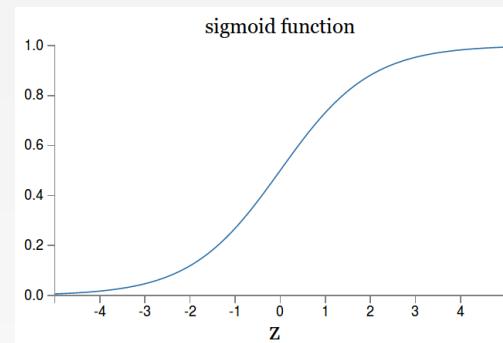
Requisito: pequeños cambios en el peso o bias deben causar pequeños cambios en la salida

- Con las neuronas perceptrones (función de activación escalón, toman valores 0 o 1) no hay garantía
- Solución: neuronas sigmoideas (función de activación sigmoid). Es derivable.

Motivación: neurona con función de activación sigmoidal

- Función de activación sigmoidal: es derivable.

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \quad \text{donde} \\ z = (\sum_j w_j x_j) + b$$

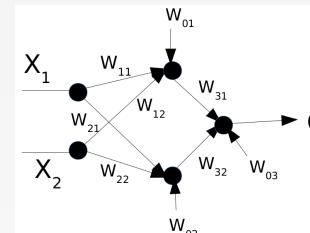


- El cambio en la salida se puede aproximar en función de pequeños cambios de w_j y b

$$\Delta \text{output} \approx (\sum_j \frac{\partial \text{output}}{\partial w_j} \Delta w_j) + \frac{\partial \text{output}}{\partial b} \Delta b$$

Multilayer perceptron

Ejemplo de la estructura de un "perceptrón" multicapa:



Multilayer "Perceptron" - MLP (perceptrón multicapa)

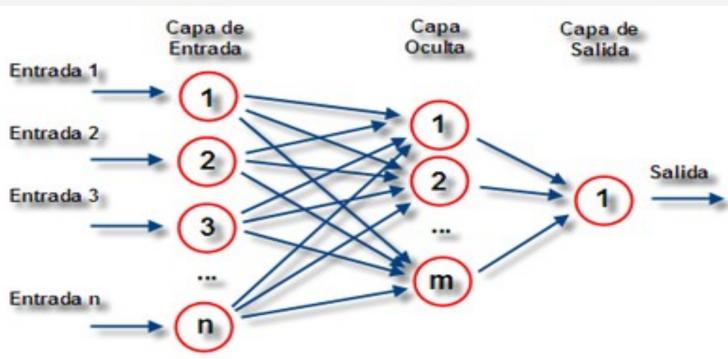
- Mal llamada "perceptrón" porque está compuesta de **neuronas sigmoides**.
- También conocida como: **plain vanilla** neuronal network.

Propiedad de aproximación universal:

- Algunos modelos (MLP: Multilayer Perceptrón, RBF: Radial Basis Functions), con un conjunto infinito de patrones y neuronas, tienen la capacidad de逼近arse a cualquier función discriminante con una determinada precisión.

Multilayer perceptron

- Es una red **feedforward** en la que todas las neuronas de un nivel están conectadas a todas las neuronas del siguiente nivel. Para el caso del MLP la estructura de la red es la siguiente:



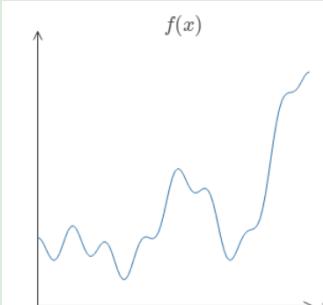
Multilayer perceptron

Universalidad del MLP

Universalidad: el MLP permite逼近ar cualquier función

Ejemplo

$f(x) = 0,2 + 0,4x^2 + 0,3x\sin(15x) + 0,05 \cos(50x)$ está comprendida entre 0 y 1



Iremos逼近ando la función con redes neuronales dispuestas en distintas topologías.

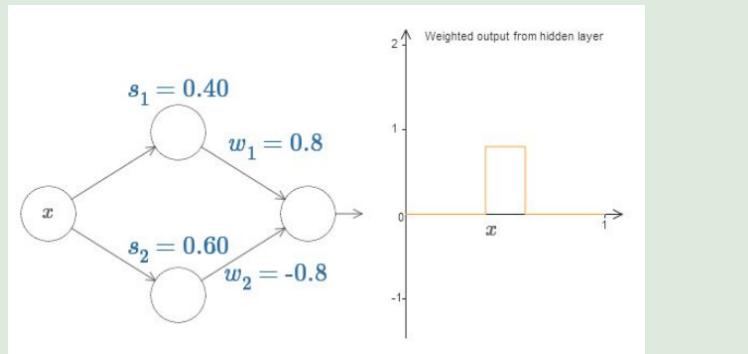
Multilayer perceptron

Universidad del MLP

Ejemplo (aproximación 1): una capa oculta de 2 neuronas

Objetivo: aproximar la función $f(x) = 0,2 + 0,4x^2 + 0,3x\sin(15x) + 0,05\cos(50x)$

Topología (aproximación 1): consideremos una red neuronal con una capa de entrada compuesta por 1 neurona, una capa de salida 1 neurona, una capa oculta de 2 neuronas



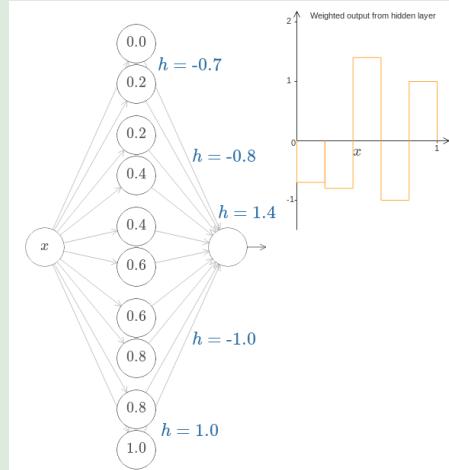
Multilayer perceptron

Universidad del MLP

Ejemplo (aproximación 3): una capa oculta de 10 neuronas

Objetivo: aproximar la función $f(x) = 0,2 + 0,4x^2 + 0,3x\sin(15x) + 0,05\cos(50x)$

Topología (aproximación 1): consideremos una red neuronal con una capa de entrada compuesta por 1 neurona, una capa de salida 1 neurona, una capa oculta de 10 neuronas



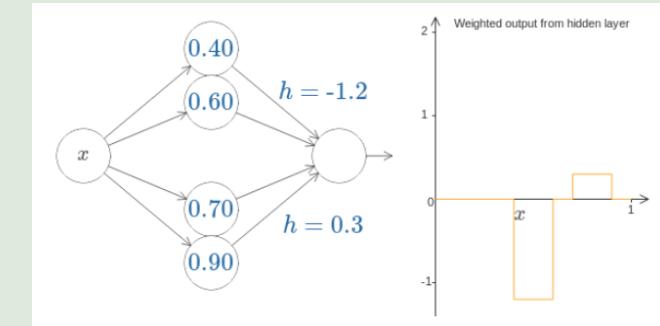
Multilayer perceptron

Universidad del MLP

Ejemplo (aproximación 2): una capa oculta de 4 neuronas

Objetivo: aproximar la función $f(x) = 0,2 + 0,4x^2 + 0,3x\sin(15x) + 0,05\cos(50x)$

Topología (aproximación 1): consideremos una red neuronal con una capa de entrada compuesta por 1 neurona, una capa de salida 1 neurona, una capa oculta de 4 neuronas



Multilayer perceptron

Universidad del MLP

Ejemplo (aproximación 4): varias capas ocultas de cientos de neuronas

Objetivo: aproximar la función $f(x) = 0,2 + 0,4x^2 + 0,3x\sin(15x) + 0,05\cos(50x)$

Topología (aproximación 1): consideremos una red neuronal con una capa de entrada compuesta por 1 neurona, una capa de salida 1 neurona, varias capas ocultas de cientos de neuronas se puede llegar a una buena aproximación.

Multilayer perceptron

Backpropagation

Backpropagation (BP): algoritmo de retropropagación del error para inferir los pesos en un MLP, se desarrolla mediante DG

Actualización de pesos durante el entrenamiento:

$$\Delta w_{ij}^k = \eta \delta_j^{k+1} o_i^k$$

donde δ_j^{k+1} es el error de la neurona j en la capa $k + 1$

- Para una neurona de la capa de [salida](#):

$$\delta_j^L = (t_j - o_j^L) \cdot o_j^L \cdot (1 - o_j^L)$$

- Para una neurona de una capa [intermedia](#):

$$\delta_j^k = o_j^k \cdot (1 - o_j^k) \sum_{l=1}^N (\delta_l^{k+1} \cdot w_{jl}^k)$$

Overfitting

Desviación y Varianza

- Desviación: diferencia esperada entre la salida del modelo (e.g. predicción) y la salida deseada (e.g. realidad)
- Varianza: cuánto difiere el modelo entre conjuntos de entrenamiento distintos

Dilema:

- Bias* ↓: el modelo hace estimaciones acertadas en el conjunto de entrenamiento
- Variance* ↓: el modelo generaliza

Overfitting

Dilema: en la práctica

- ① Primero: llegar al over-fitting ¿cómo?
 - Incrementar la complejidad del modelo (redes más grandes)
 - Conseguir más datos
 - Balancear las clases del entrenamiento
 - Reducir la regularización (restricciones en los valores de los parámetros)
- ② Después simplificar la red para ganar generalidad (e.g. aplicar regularización)

Concluding remarks

Bibliografía

- Ian H. Witten, Eibe Frank, Mark A. Hall. **Data Mining: Practical Machine Learning Tools and Techniques, 3rd Edition.** The Morgan Kaufmann Series in Data Management Systems. 2011.
- Michael Nielsen, **Neural Networks and Deep Learning:** <http://neuralnetworksanddeeplearning.com/>
- A Neuronal Network Playground: <http://playground.tensorflow.org/>