

Sistemas Inteligentes

Redes neuronales

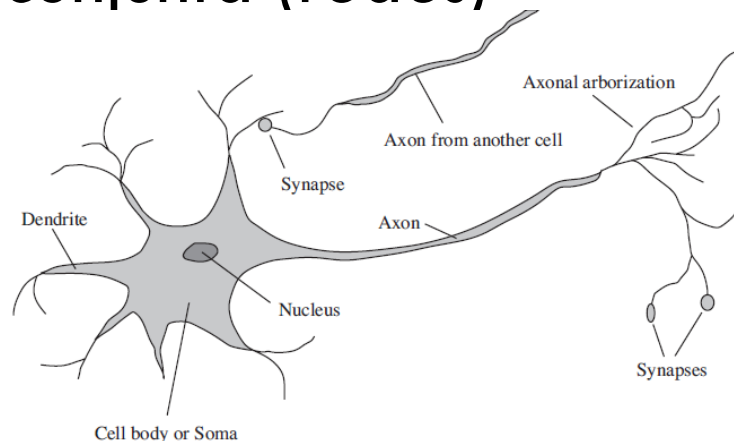
Tema 10



Neuronas naturales

2/27

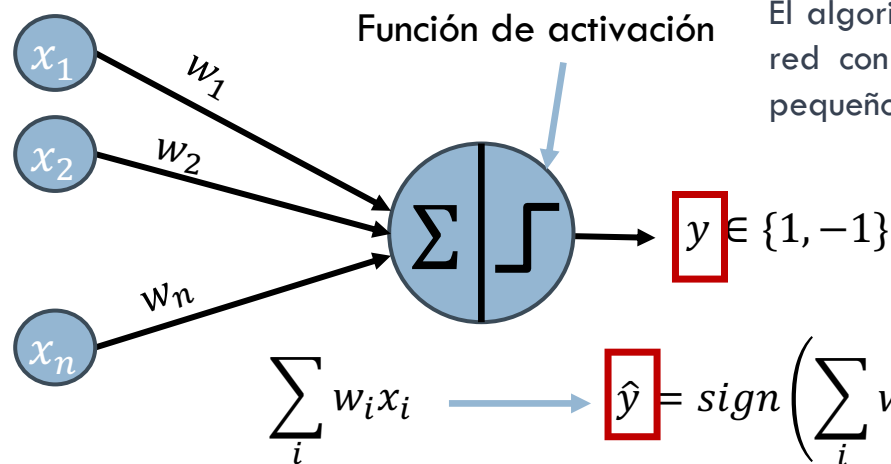
- Célula del cerebro encargada de recibir, enviar y procesar señales eléctricas
- Alrededor de 10^{11} en el cerebro humano
- Cada una puede recibir señales simultaneas de otras 1000
- La capacidad de procesar información se atribuye a la acción conjunta (redes)



Estructura básica: Perceptrón

3/27

- Modelo de un nodo de activación
- Entrada: vector \mathbf{x} (salidas de otras neuronas o variables del problema)
- Pesos: vector \mathbf{w} (dendritas)



El algoritmo de entrenamiento funciona alimentando cada la red con cada ejemplo de entrada de uno por uno (o en pequeños lotes) para crear la predicción

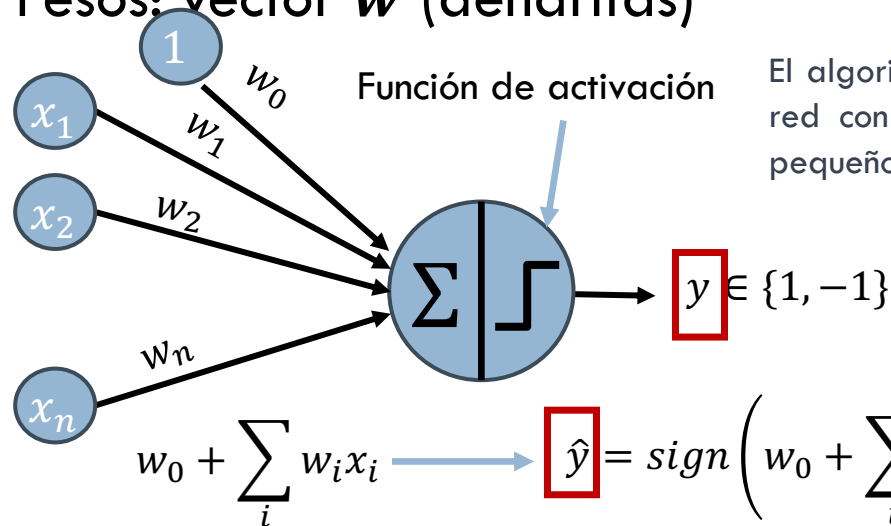
$$\mathbf{w} = \mathbf{w} + \eta(y - \hat{y})\mathbf{x} = \begin{cases} \mathbf{w} + 2\eta\mathbf{x} & \text{si } y = 1, \hat{y} = -1 \\ \mathbf{w} - 2\eta\mathbf{x} & \text{si } y = -1, \hat{y} = 1 \end{cases}$$



Estructura básica: Perceptrón

4/27

- Modelo de un nodo de activación
- Entrada: vector x (salidas de otras neuronas o variables del problema)
- Pesos: vector w (dendritas)



El algoritmo de entrenamiento funciona alimentando cada la red con cada ejemplo de entrada de uno por uno (o en pequeños lotes) para crear la predicción

$$w = w + \eta(y - \hat{y})x = \begin{cases} w + 2\eta x & \text{si } y = 1, \hat{y} = -1 \\ w - 2\eta x & \text{si } y = -1, \hat{y} = 1 \end{cases}$$



Aprendizaje de un perceptrón

5/27

- Por cada instancia $d \in [1, D]$ medimos la discrepancia de la clasificación como

$$E(\mathbf{w}) = y_d - g(\mathbf{w}x_d)$$

- El objetivo es que no haya ningún error
- Para conseguirlo se propone que la variación de los pesos sea

$$\Delta \mathbf{w} = E(\mathbf{w})x_d$$

- Y así los nuevos pesos serán

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(y_d - g(\mathbf{w}x_d))x_d$$



Aprendizaje de un perceptrón

6/27

- Si la instancia d está bien clasificada no hay cambios
- Si $y_d = 1$ pero se predice lo contrario hay que cambiar \mathbf{w} para aumentar el valor de $g(\cdot)$
 - Para $x_i > 0$, w_i aumentará y en caso contrario disminuirá
- Si $y_d = -1$ pero se predice lo contrario hay que cambiar \mathbf{w} para disminuir el valor de $g(\cdot)$
 - Para $x_i > 0$, w_i disminuirá y en caso contrario aumentará



Aprendizaje de un perceptrón

7/27

- Inicialmente los pesos toman un valor aleatorio
- Normalmente hay que procesar los datos de entrenamiento varias veces
- En el caso ideal se para cuando todas las instancias estén bien clasificadas
- Permite un aprendizaje *on-line*
- Se puede introducir una **tasa de aprendizaje** η para graduar como se cambian los pesos

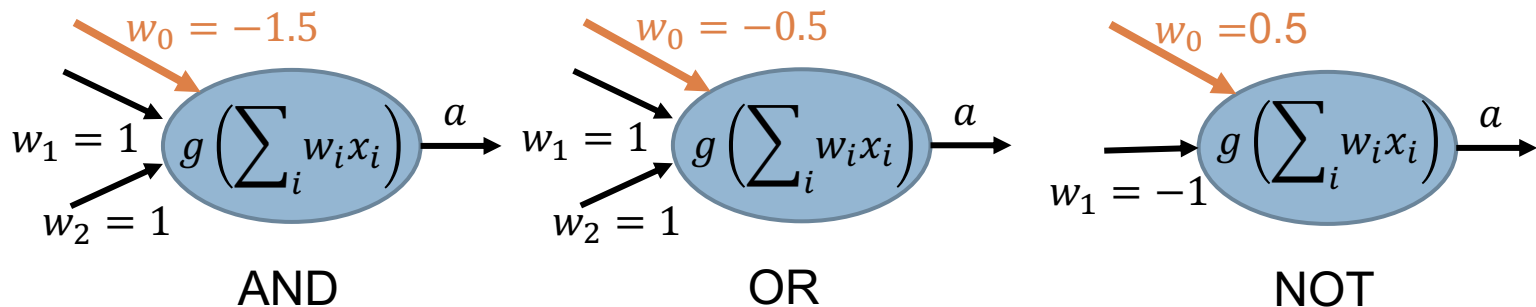
$$\Delta \mathbf{w} = \eta E(\mathbf{w}) \mathbf{x}_d$$



Limitaciones del Perceptrón

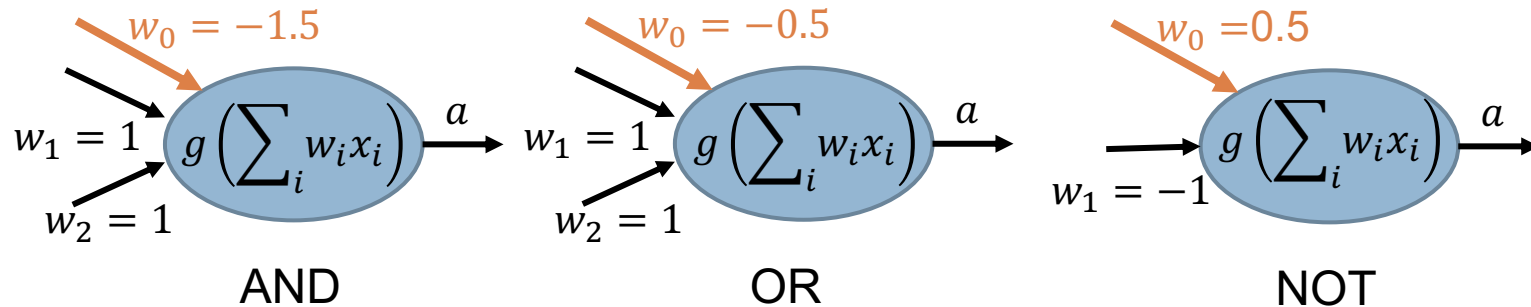
8/27

- Con esta única unidad podemos solucionar problemas linealmente separables
- En concreto podemos utilizar perceptrones para representar las funciones primitivas booleanas



Limitaciones del Perceptrón

9/27



x	y	X AND Y
1	1	$1*1+1*1-1.5=0.5>0 \rightarrow 1$
1	0	$1*1+1*0-1.5=-0.5<0 \rightarrow 0$
0	1	$0*1+1*1-1.5=-0.5<0 \rightarrow 0$
0	0	$0*1+0*1-1.5=-1.5<0 \rightarrow 0$

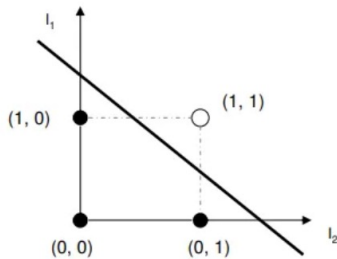


Limitaciones del Perceptrón

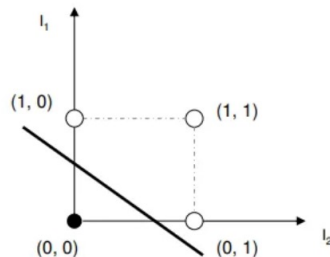
10/27

$$XOR(x_1, x_2) = \text{AND}(\text{NOT}(\text{AND}(x_1, x_2)), \text{OR}(x_1, x_2))$$

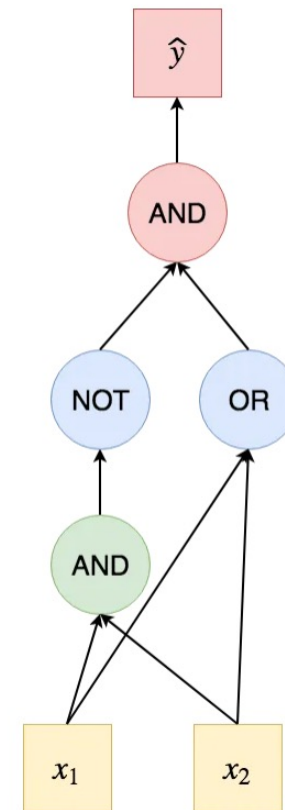
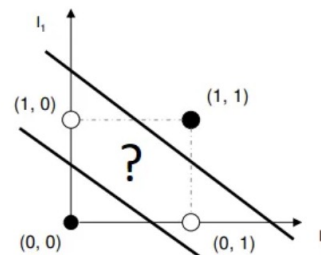
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



Perceptrón

11/27



Problemas con la convergencia

12/27

- Cuando el problema es linealmente separable la **convergencia** está garantizada
- Si hay ruido o no hay una separación lineal no hay garantía de obtener una salida
- Para solventar esto se propone usar el **descenso de gradiente**
- De esta forma el algoritmo convergerá a una solución que se ajusta a los datos (posiblemente **subóptima**)



Descenso de gradiente

13/27

- Se usa para obtener los parámetros que minimizan una función
- Se garantiza un mínimo local
- ¿Qué función queremos minimizar?
 - El error que se comete al clasificar
- Sin embargo no se puede usar con la definición anterior
 - La función de activación no es derivable
- Definamos esta función como la identidad

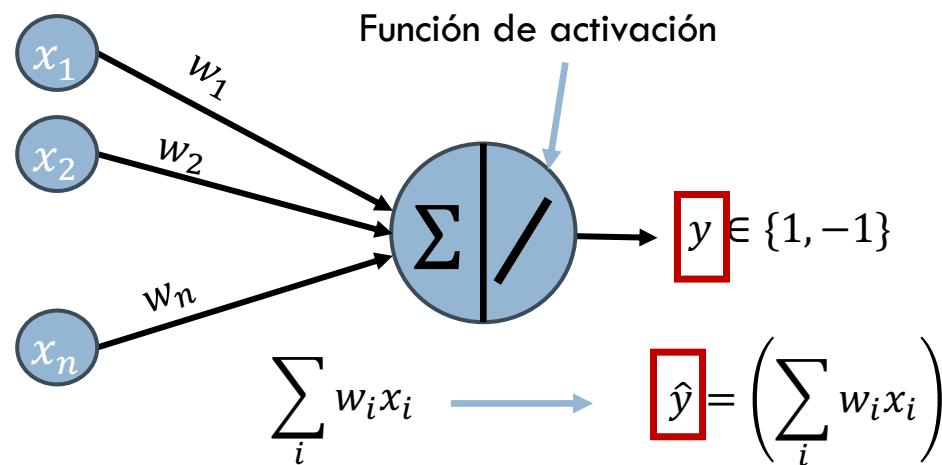
$$g(wx) = wx$$



RN con función de activación lineal

14/27

- Modelo de un nodo de activación
- Entrada: vector x (salidas de otras neuronas o variables del problema)
- Pesos: vector w (dendritas)



Obteniendo delta

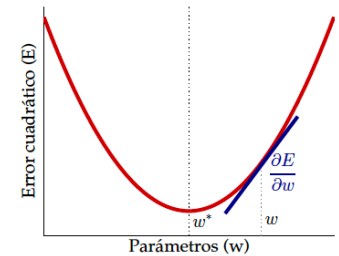
15/27

- Por conveniencia retocamos también el error

$$E(\mathbf{w}) = \frac{1}{2}(y_d - \mathbf{w}\mathbf{x}_d)^2$$

- Derivando

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= (y_d - \mathbf{w}\mathbf{x}_d) \frac{\partial E}{\partial w_i} (y_d - \mathbf{w}\mathbf{x}_d) \\ &= (y_d - \mathbf{w}\mathbf{x}_d)(-x_{di})\end{aligned}$$



- Por tanto

$$\Delta w_i = \eta(y_d - \mathbf{w}\mathbf{x}_d)x_{di}$$

Regla de actualización: Regla delta



Algoritmo de descenso de gradiente (incremental)

16/27

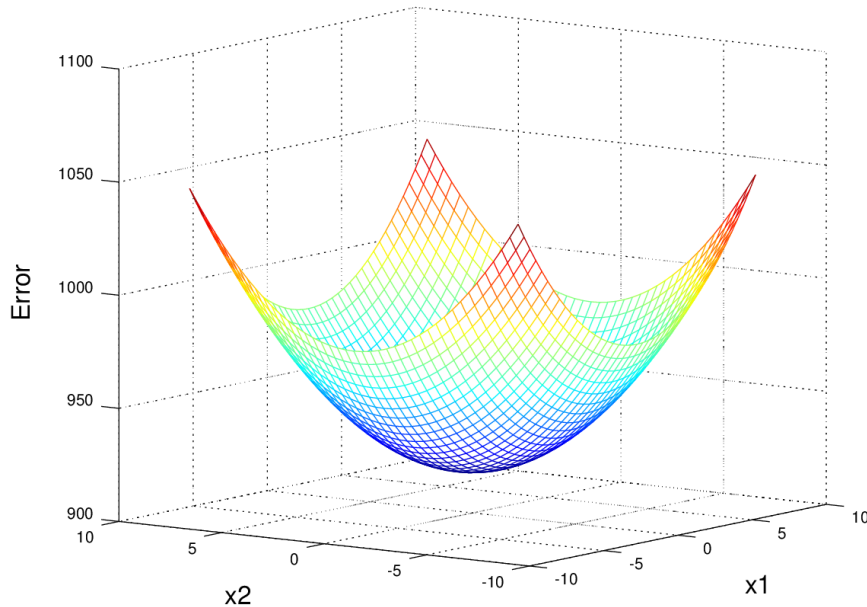
1. Inicializar \mathbf{W} con valores aleatorios
 2. Mientras no se cumpla la condición de parada
 3. Para cada par de entrenamiento $\langle \mathbf{x}_d, y_d \rangle$
 4. Calcular la predicción $o_d = \mathbf{W}\mathbf{x}_d$
 5. Para cada peso
 6. $w_i \leftarrow w_i + \eta(y_d - o_d)x_{di}$
- El espacio de búsqueda es **convexo** si el problema es linealmente separable



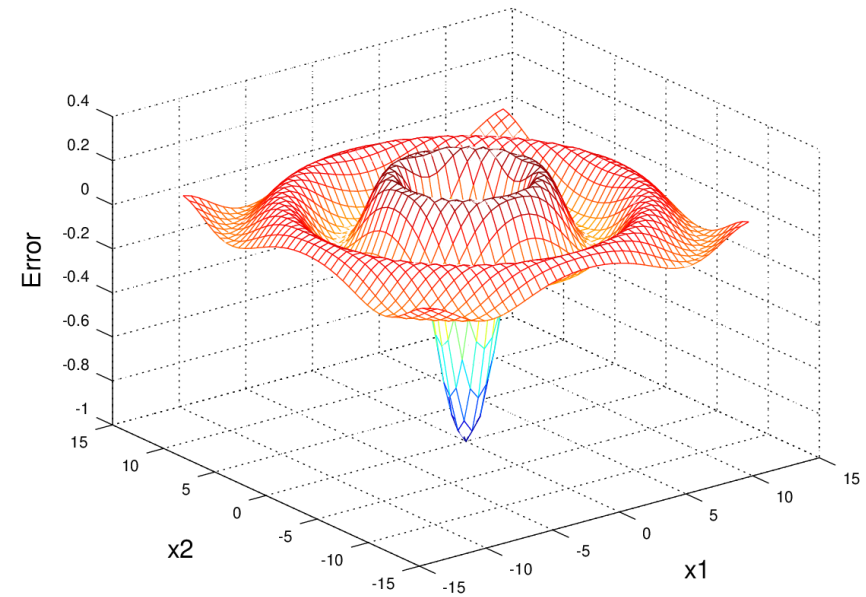
Visualización del espacio de búsqueda

17/27

■ Espacio convexo



■ Espacio no convexo



Resumen hasta ahora

18/27

- Un perceptrón usa una función de activación no lineal, pero solo se puede ajustar a problemas lineales
- Una neurona artificial con activación lineal se puede aplicar a problemas no lineales, pero todavía la frontera de decisión es lineal
- ¿Cómo podemos hacer el paradigma más versátil?



Redes de neuronas

19/27

- Parece lógico pensar que combinando neuronas se puede conseguir un modelo más complejo
- Problema: ¿Qué neuronas ponemos?
- ¿Perceptrones?
 - No se puede usar el descenso de gradiente
- ¿Neuronas con función de activación lineal?
 - Una combinación de estas neuronas todavía no es capaz de representar funciones no lineales
- Se necesita una unidad que produzca una salida no lineal y sea diferenciable

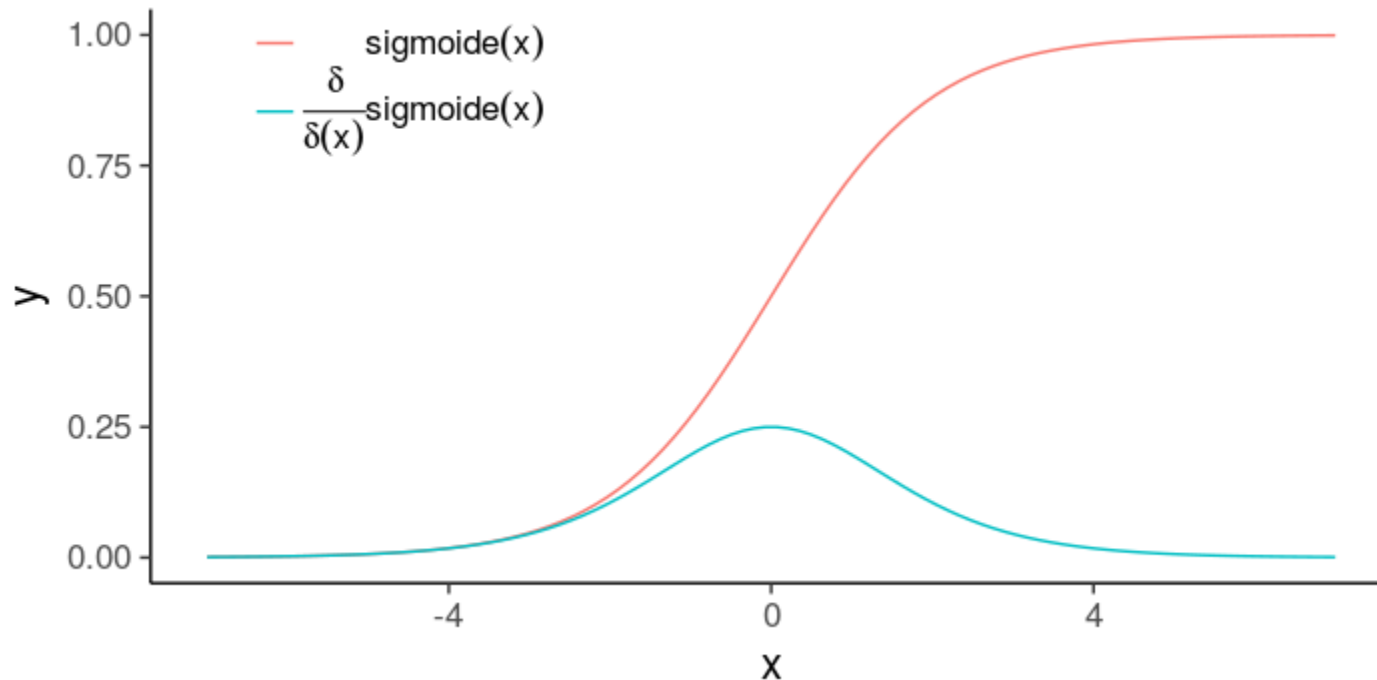


Función sigmoide

20/27

- Esta es una solución al problema anterior

$$g(x) = \frac{1}{1+e^{-x}} ; \quad \frac{\delta g(x)}{\delta x} = g(x)(1 - g(x))$$

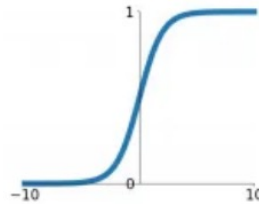


Otras funciones de activación

21/27

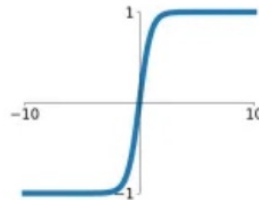
Sigmoid

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



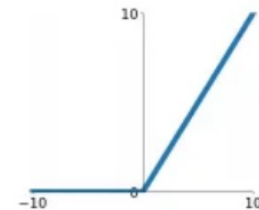
tanh

$$\tanh(x)$$



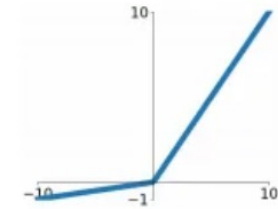
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

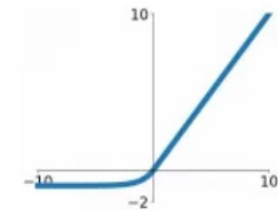


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

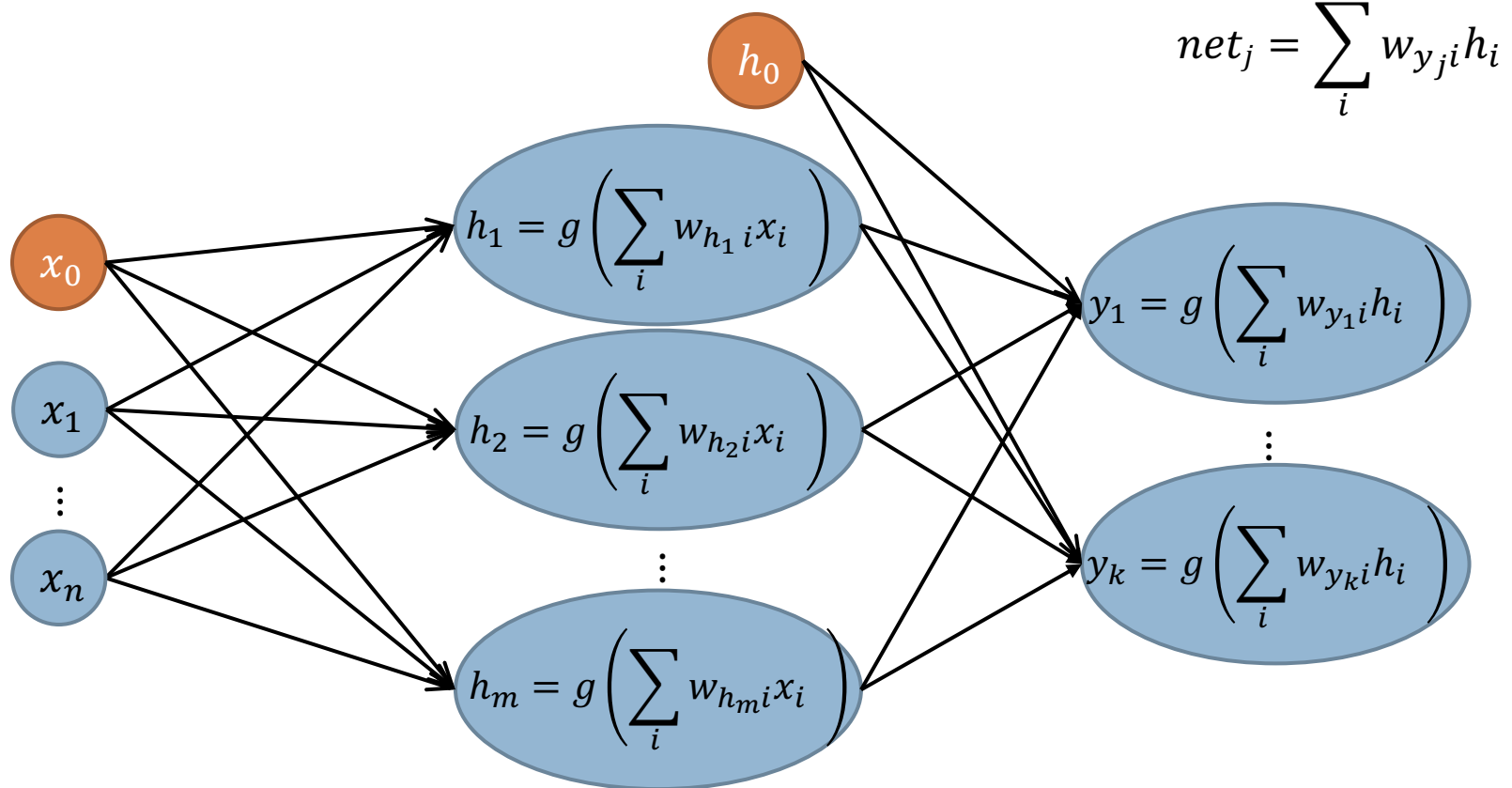
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Red neuronal de 1 capa oculta

22/27



Red de neuronas

23/27

- La conexión entre neuronas se puede extender
- En **profundidad**, varias capas ocultas
- En **anchura**, más neuronas en la capa oculta
- En muchos casos basta con una capa oculta, siempre que hayan **suficientes neuronas**



Clasificación k-aria

24/27

- Cada neurona de salida permite codificar dos valores (verdadero/falso)
- Para problemas con variables clases de más de dos valores tendremos una neurona de salida por cada valor
- Para entrenar se usará la codificación *one-hot*
- Para predecir se escoge la clase de la neurona con mayor valor de salida



Aprendizaje

25/27

- La tarea sigue siendo minimizar el error

$$E(\mathbf{w}) = \frac{1}{2} \sum_d \sum_k (y_{kd} - o_{kd})^2$$

- Pero ahora el número de pesos se ha multiplicado
 - El espacio de búsqueda puede tener mínimos locales
- ¿Cómo se evalúa el error para las neuronas ocultas?
- ¡Pues que las neuronas de salida indiquen a las ocultas el valor del error!
- Por esto algoritmo se llama “**propagación hacia atrás**”



Propagación hacia atrás I

26/27

- **Backpropagation(training_set, η , D , n_h , K)**
 - training_set contiene D pares $\langle x, y \rangle$, donde x es un vector de valores de entrada, e y es la variable a predecir con K valores. η es la tasa de aprendizaje y n_h es el número de neuronas ocultas.
- 1. Crea una red neuronal con $|x|$ entradas, n_h neuronas ocultas y K salidas.
- 2. Inicializa todos los pesos con números aleatorios pequeños (entre -0.05 y 0.05)
- 3. Repite hasta que se dé la condición de parada



Propagación hacia atrás II (con función sigmoide)

27/27

4. Para cada instancia de entrenamiento $\langle x, y \rangle$
// Propagar la entrada a través de la red
5. Para cada unidad de salida k calcula o_k
// Propagar el error hacia atrás
6. Para cada unidad de salida k calcula δ_k
$$\delta_k = o_k(1 - o_k)(y_k - o_k) \quad \frac{\delta g(x)}{\delta x} = g(x)(1 - g(x))$$
7. Para cada unidad oculta h calcula δ_h
$$\delta_h = o_h(1 - o_h) \sum_k w_{kh} \delta_k$$

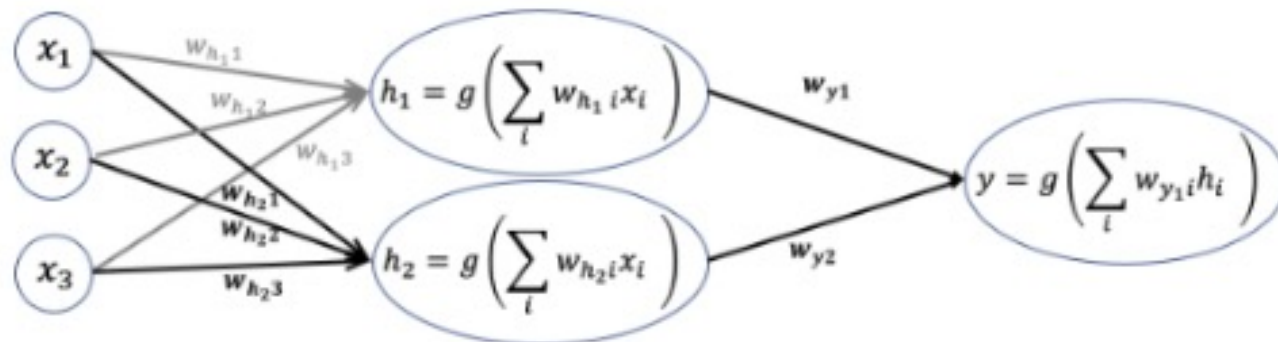
// w_{ji} peso de la unidad i a la j , x'_{ji} entrada de i a j
8. Actualiza cada peso de la red w_{ji}
$$w_{ji} \leftarrow w_{ji} + \eta \delta_j x'_i$$



Ejemplo: Propagación hacia delante

28/27

$$g(x) = \frac{1}{1-e^{-x}}; \quad \frac{\delta g(x)}{\delta x} = g(x)(1 - g(x))$$



$$w_{h_1,1} = w_{h_1,2} = w_{h_1,3} = 0.1$$

$$w_{h_2,1} = w_{h_2,2} = w_{h_2,3} = 0.2$$

$$w_{y,1} = w_{y,2} = 0.3$$

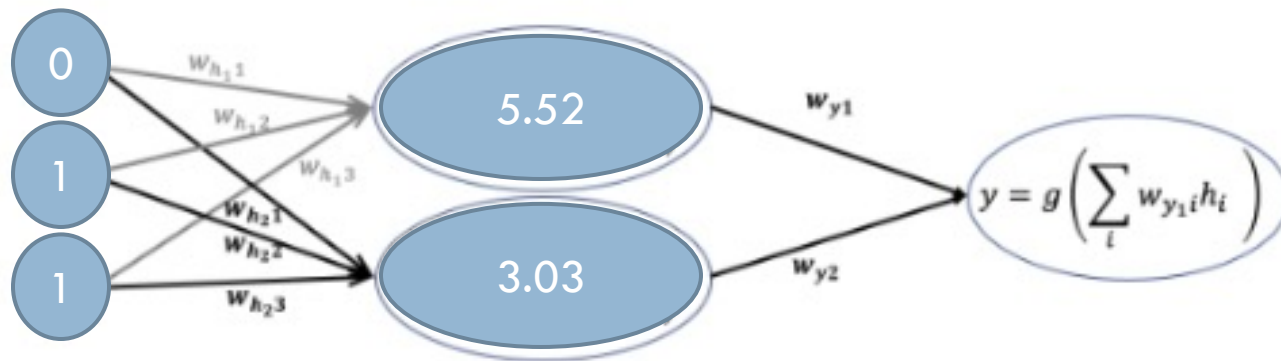


Ejemplo: Propagación hacia delante

29/27

Ejemplo (0,1,1,2)

$$g(x) = \frac{1}{1-e^{-x}}; \quad \frac{\delta g(x)}{\delta x} = g(x)(1 - g(x))$$



$$w_{h1,1} = w_{h1,2} = w_{h1,3} = 0.1 \quad w_{h2,1} = w_{h2,2} = w_{h2,3} = 0.2 \quad w_{y,1} = w_{y,2} = 0.3$$

$$o_{h_1} = g(w_{h1,1}x_1 + w_{h1,2}x_2 + w_{h1,3}x_3) = g(0,1 * 0 + 0,1 * 1 + 0,1 * 1) = g(0,2) = \frac{1}{1 - e^{-0,2}} = 5,52$$

$$o_{h_2} = g(w_{h2,1}x_1 + w_{h2,2}x_2 + w_{h2,3}x_3) = g(0,2 * 0 + 0,2 * 1 + 0,2 * 1) = g(0,4) = \frac{1}{1 - e^{-0,4}} = 3,03$$

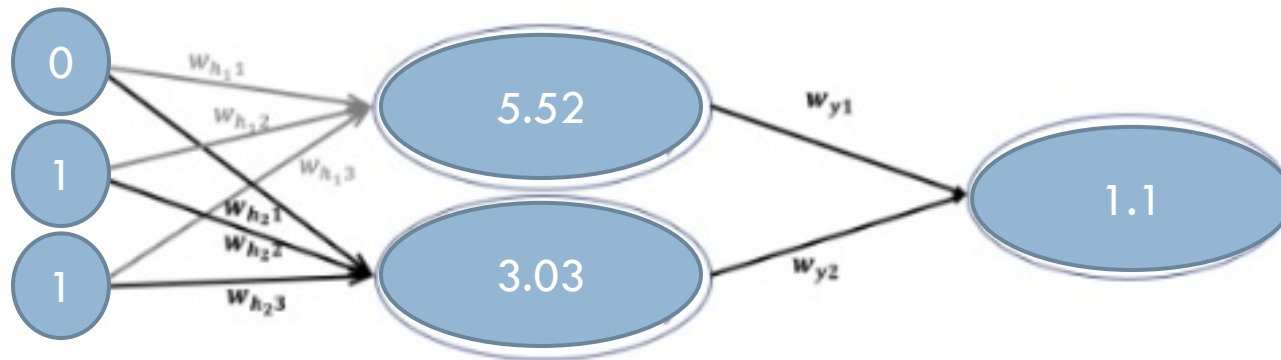


Ejemplo: Propagación hacia delante

30/27

Ejemplo (0,1,1,2)

$$g(x) = \frac{1}{1+e^{-x}}; \quad \frac{\delta g(x)}{\delta x} = g(x)(1 - g(x))$$



$$w_{h1,1} = w_{h1,2} = w_{h1,3} = 0.1 \quad w_{h2,1} = w_{h2,2} = w_{h2,3} = 0.2 \quad w_{y,1} = w_{y,2} = 0.3$$

$$o_y = g(w_{y1}o_{h1} + w_{y2}o_{h2}) = g(0.3 \cdot 5.52 + 0.3 \cdot 3.03) = g(2.565) = 1.1$$

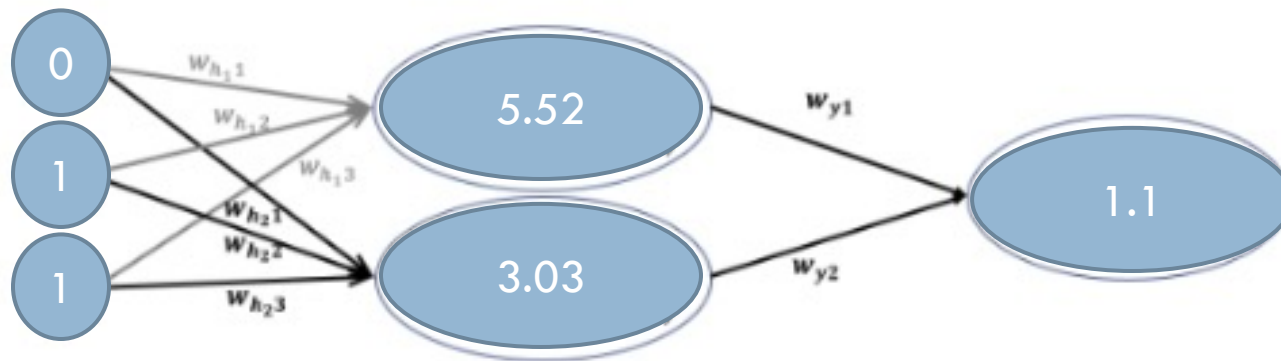


Ejemplo: Propagación hacia atrás

31/27

Ejemplo (0,1,1,**2**)

$$g(x) = \frac{1}{1+e^{-x}}; \quad \frac{\delta g(x)}{\delta x} = g(x)(1 - g(x))$$



$$w_{h1,1} = w_{h1,2} = w_{h1,3} = 0.1 \quad w_{h2,1} = w_{h2,2} = w_{h2,3} = 0.2 \quad w_{y,1} = w_{y,2} = 0.3$$



Ejemplo: Propagación hacia atrás

32/27

Para cada instancia de entrenamiento $\langle x, y \rangle$

// Propagar la entrada a través de la red

Para cada unidad de salida k calcula o_k

// Propagar el error hacia atrás

Para cada unidad de salida k calcula δ_k

$$\delta_k = o_k(1 - o_k)(y_k - o_k)$$

Para cada unidad oculta h calcula δ_h

$$\delta_h = o_h(1 - o_h) \sum_k w_{kh} \delta_k$$

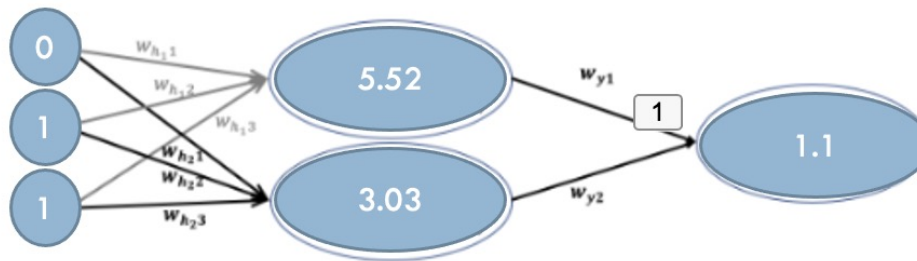
// w_{ji} peso de la unidad i a la j , x'_{ji} entrada de i a j

Actualiza cada peso de la red w_{ji}

$$w_{ji} \leftarrow w_{ji} + \eta \delta_j x'_i$$

Ejemplo (0,1,1,**2**)

$$g(x) = \frac{1}{1+e^{-x}}; \quad \frac{\partial g(x)}{\partial x} = g(x)(1 - g(x))$$



$$w_{h1j} = 0.1, \quad w_{h2j} = 0.2, \quad w_{yj} = 0.3, \quad \eta = 0.2$$



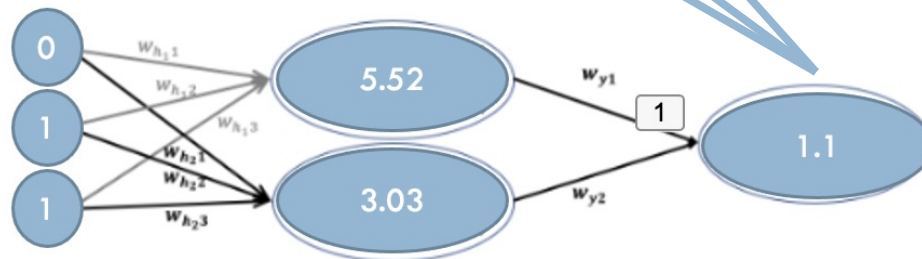
Ejemplo: Propagación hacia atrás

33/27

Para cada unidad de salida k calcula δ_k
$$\delta_k = o_k(1 - o_k)(y_k - o_k)$$

$$\delta_y = o_y(1 - o_y)(y - o_y) = 1.1(1 - 1.1)(2 - 1.1) = -0.099$$

Ejemplo (0,1,1, **2**)



$$g(x) = \frac{1}{1 + e^{-x}}, \quad \frac{\delta g(x)}{\delta x} = g(x)(1 - g(x))$$

$$w_{h1j} = 0.1, \quad w_{h2j} = 0.2, \quad w_{yj} = 0.3, \quad \eta = 0.2$$



Ejemplo: Propagación hacia atrás

34/27

Para cada unidad oculta h calcula δ_h

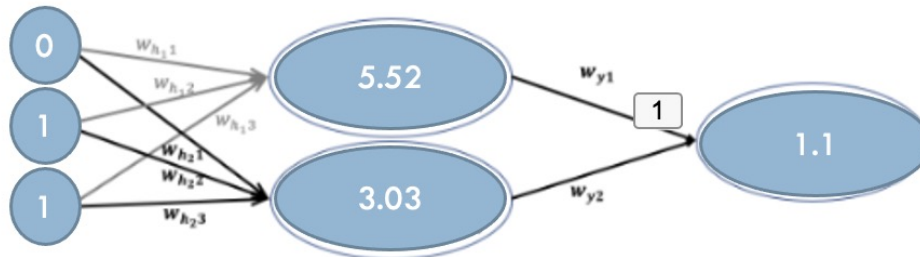
$$\delta_h = o_h(1 - o_h) \sum_k w_{kh} \delta_k$$

$$\delta_{h1} = o_{h1}(1 - o_{h1})w_{y1}\delta_y = 5.52*(1-5.52)*0.3*(-0.099) = 0.74$$

$$\delta_{h2} = o_{h2}(1 - o_{h2})w_{y2}\delta_y = 3.03*(1-3.03)*0.3*(-0.099) = 0.18$$

Ejemplo (0,1,1,2)

$$g(x) = \frac{1}{1+e^{-x}}; \quad \frac{\delta g(x)}{\delta x} = g(x)(1 - g(x))$$



$$w_{h1,j} = 0.1, \quad w_{h2,j} = 0.2, \quad w_{y,j} = 0.3, \quad \eta = 0.2$$



Ejemplo: Propagación hacia atrás

35/27

Actualiza cada peso de la red w_{ji} $\delta_y = -0.099$ $\delta_{h1} = 0.74$ $\delta_{h2} = 0.18$
 $w_{ji} \leftarrow w_{ji} + \eta \delta_j x'_i$

$$w_{h1,1} \leftarrow w_{h1,1} + \eta \delta_{h1} x'_1 = 0.1 + 0.2 * 0.74 * 0 = 0.1$$

$$w_{h1,2} \leftarrow w_{h1,2} + \eta \delta_{h1} x'_2 = 0.1 + 0.2 * 0.74 * 1 = 0.248$$

$$w_{h1,3} \leftarrow w_{h1,3} + \eta \delta_{h1} x'_3 = 0.1 + 0.2 * 0.74 * 1 = 0.248$$

$$w_{h2,1} \leftarrow w_{h2,1} + \eta \delta_{h2} x'_1 = 0.2 + 0.2 * 0.18 * 0 = 0.2$$

$$w_{h2,2} \leftarrow w_{h2,2} + \eta \delta_{h2} x'_2 = 0.2 + 0.2 * 0.18 * 1 = 0.236$$

$$w_{h2,3} \leftarrow w_{h2,3} + \eta \delta_{h2} x'_3 = 0.2 + 0.2 * 0.18 * 1 = 0.236$$

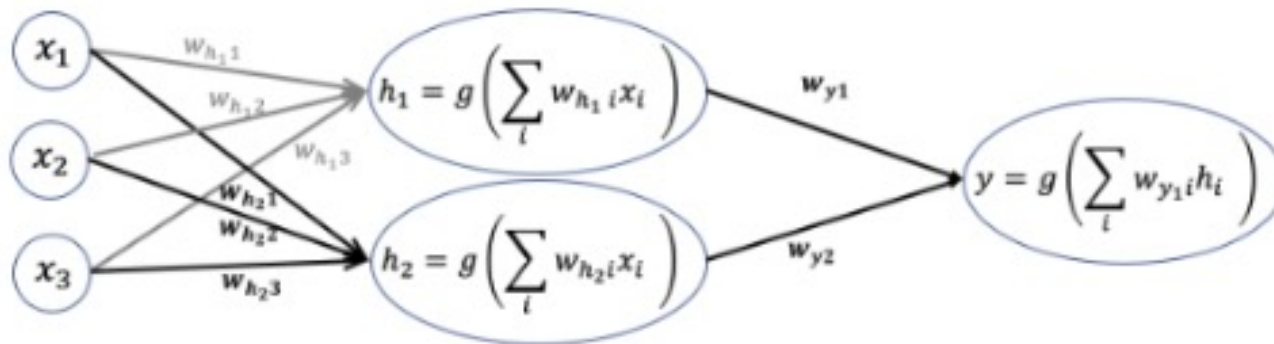
$$w_{y,1} \leftarrow w_{y,1} + \eta \delta_y o_{h1} = 0.3 + 0.3 * -0.099 * 5.52 = 0.13$$

$$w_{y,2} \leftarrow w_{y,2} + \eta \delta_y o_{h2} = 0.3 + 0.3 * -0.099 * 3.03 = 0.21$$



Ejemplo: Propagación hacia atrás

36/27



$$w_{h_1,1} = 0.1$$

$$w_{h_1,2} = 0.248$$

$$w_{h_1,3} = 0.248$$

$$w_{h_2,1} = 0.2$$

$$w_{h_2,2} = 0.236$$

$$w_{h_2,3} = 0.236$$

$$w_{y,1} = 0.13$$

$$w_{y,2} = 0.21$$



¿Cuándo actualizar los pesos?

37/27

- Hasta ahora nos hemos centrado en DG incremental
 - Esto implica que se modifican los peso por cada instancia
- El otro extremo es calcular la variación de los pesos con todas las instancias
- O algo intermedio, solo se actualizan los pesos cuando se ven un número dado de instancias
- **Batch** es el conjunto de instancias que hay que procesar para actualizar los pesos
- **Época (Epoch)** es el lapso en que se han procesado todas las instancias de entrenamiento



Variaciones

38/27

■ Añadir Momentum

- La variación de los pesos de una iteración tiene en cuenta la variación de la anterior

■ Weight decay

- Para evitar sobreajuste se añade un término al error a minimizar con un parámetro que controla este término




■ Descenso de gradiente estocástico

- Se elige de forma aleatoria las instancias que forman parte del *Batch*



¿Cuándo paramos?

39/27

- Descenso de gradiente es un proceso iterativo
- ¿Hasta que error sea menor que un valor dado?
 - Esta estrategia puede conducir al sobreajuste
- ¿Con un número fijo de iteraciones?
 - Mejor, pero es difícil saber cuantas
- ¿Y si monitorizamos la capacidad de predicción de red con datos de validación?
 - Pararíamos cuando el error de validación empieza a subir

Potencial de las redes neuronales

40/27

- Una red de una capa oculta puede representar cualquier función booleana
- También puede representar cualquier función continua
- Con dos capas ocultas se pueden representar funciones discontinuas
- ¿Hace falta más? ¿Porqué Deep Neural Networks?
- Parece ser que con similar número de parámetros mayor profundidad muestra mejores resultados



Ventajas y desventajas

41/27

Fortalezas	Debilidades
Se puede aplicar a cualquier tipo de problema (si hay suficientes datos)	Proclive al sobreajuste
Aprendizaje online	Muchos parámetros que determinar
Tolerante al ruido	Entrenamiento costoso



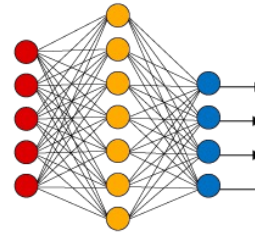
Deep Learning

TOC

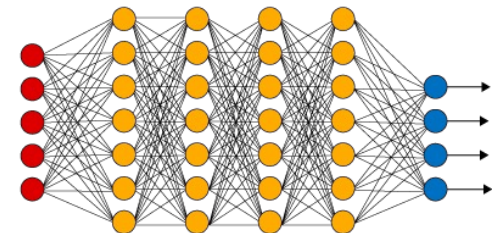
■ Qué es Deep Learning?

En un vistazo...

Simple Neural Network



Deep Learning Neural Network



● Input Layer ● Hidden Layer ● Output Layer

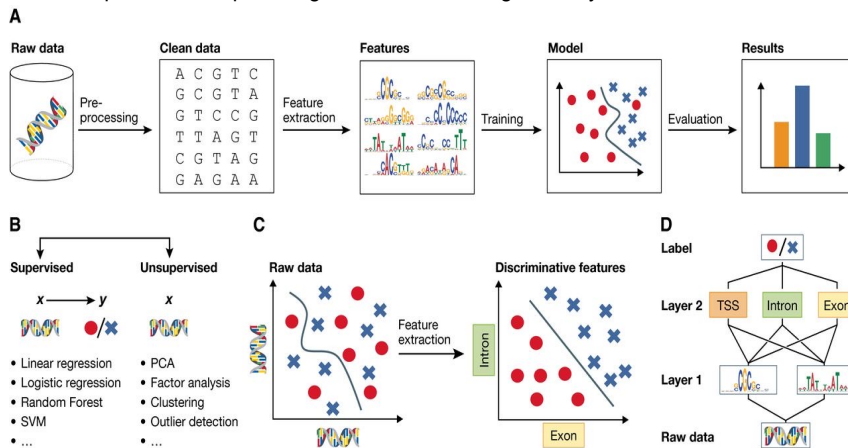
https://cdn-images-1.medium.com/max/1600/1*dnvGC-PORSoCo7VXT3PV_A.png

Qué es Deep learning

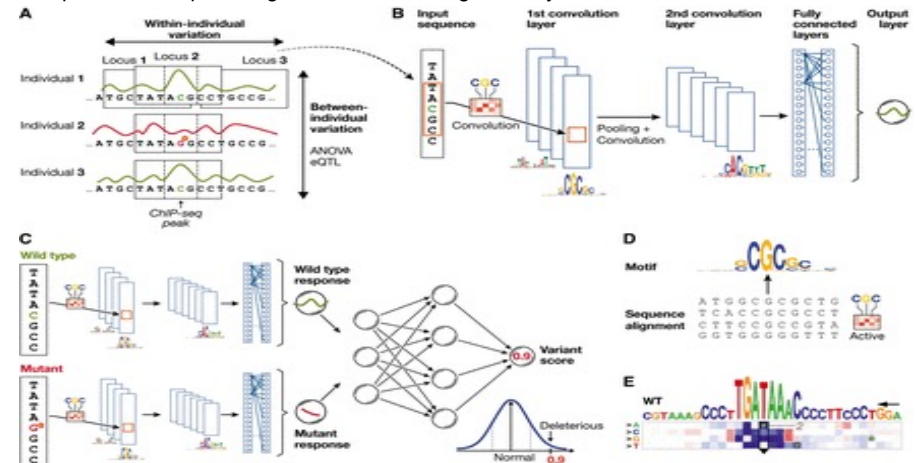
ML clásico

ML con DL

<http://msb.embopress.org/content/12/7/878.figures-only>



<http://msb.embopress.org/content/12/7/878.figures-only>

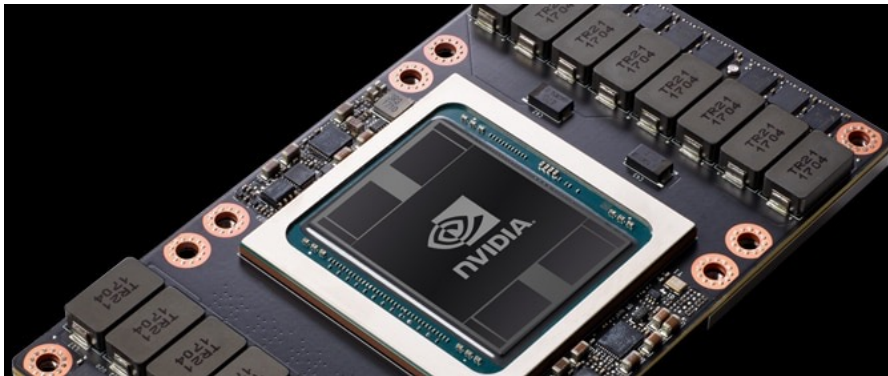


Qué es DEEP LEARNING?

Diferencia?

- Todo se entrena a la vez!
- Por qué ahora?

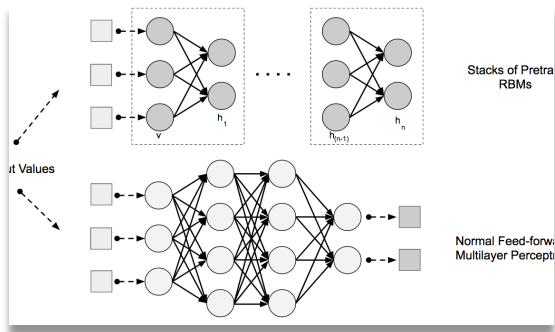
<https://cloud.oracle.com/opc/images/solutions-nvidia-gpu-banner4.jpg>



Por lo tanto...

- Podemos decir que Deep Learning es el apilamiento de modelos en secuencia cuyos parámetros se entrenan conjuntamente.
- El modelo más usual es el de la red neuronal.
- El uso de Deep Learning evita la costosa tarea de pre-procesado e ingeniería de atributos.

Tipos de Deep learning



Unsupervised Pretrained Networks (UPNs)

Autoencoders, Variational Autoencoders (VAEs)

Deep Belief Networks (DBNs)

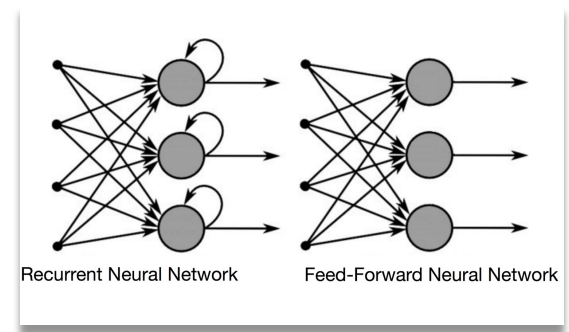
Generative Adversarial Networks (GANs)



Convolutional Neural Networks (CNNs)

Capas de Convolución + ReLU + Pool

ReLU rectified Linear Unit



Recurrent and Recursive Neural Networks (RNNs)

Recursive → shared weights

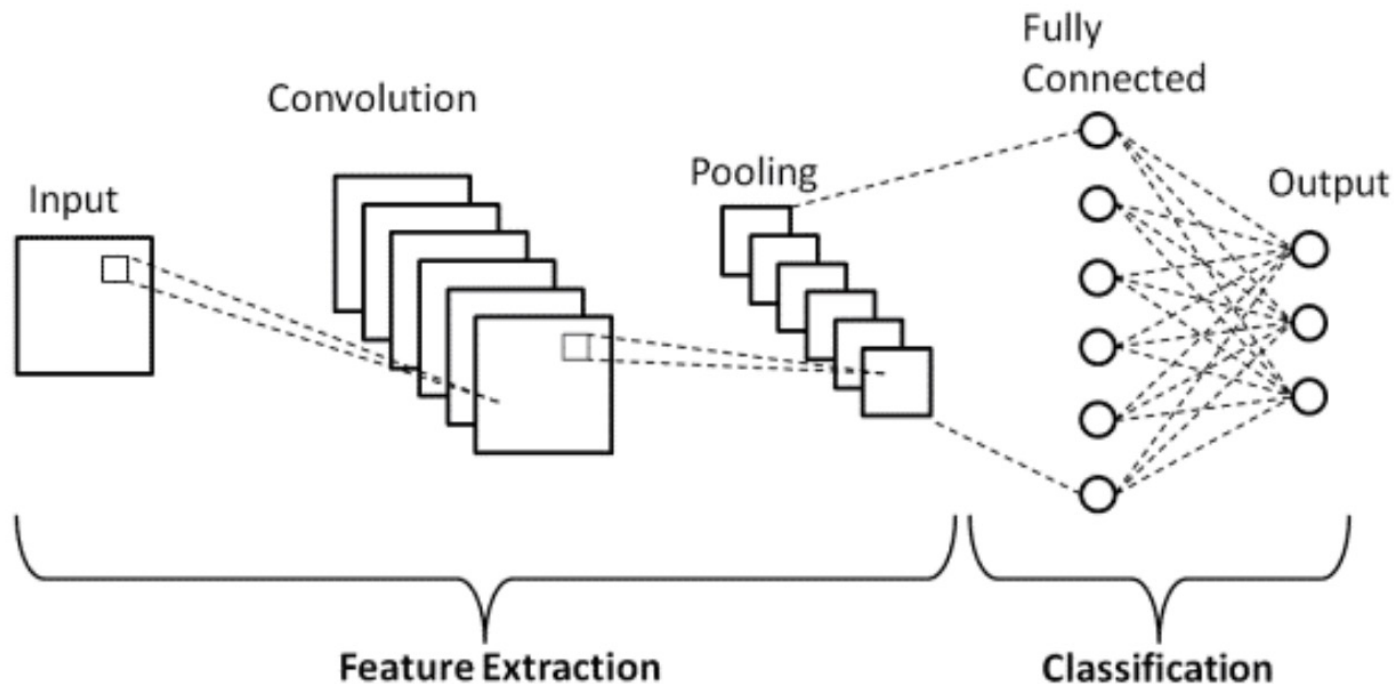
Usos

Usos de DL

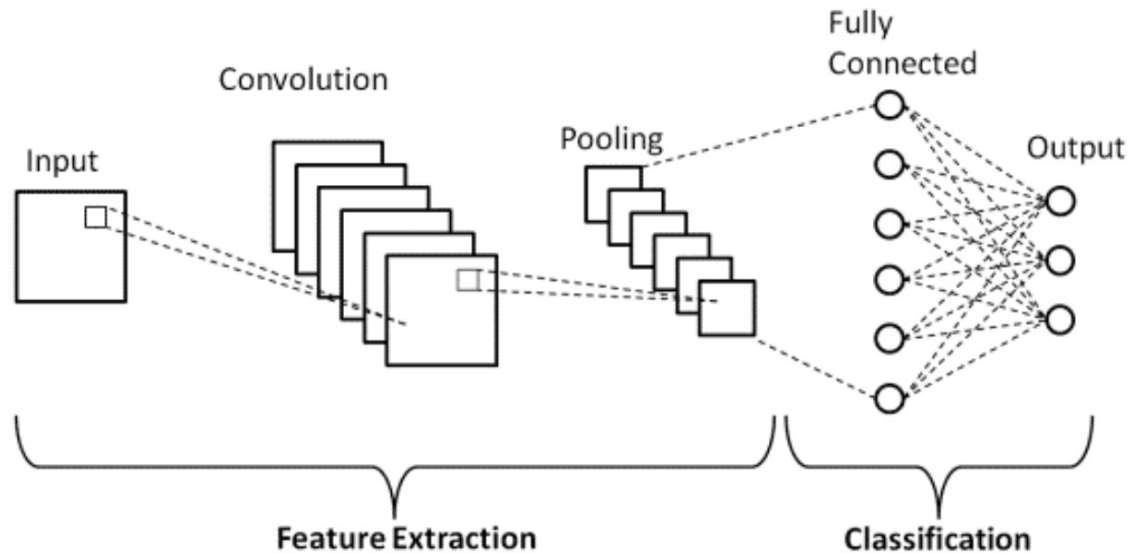
- Generación de datos a partir de entrada (imágenes, audio, texto): GANs, VAEs, Recurrent Neural Networks
- Para modelar imágenes: CNNs y DBNs
- Para modelar secuencias de datos: Recurrent Neural Networks, LSTM

Redes Neuronales Convolucionales

- Las CNN (Redes Neuronales Convolucionales) son una clase de Redes Neuronales Profundas que pueden reconocer y clasificar características particulares en imágenes, y se utilizan ampliamente para analizar imágenes



Redes Neuronales Convolucionales



- Capa de convolución
- Capas de agrupación
- Capas completamente conectadas

CNN: Capa de Convolución

Convolución es el operador matemático que convierte dos funciones **f** y **g** en una tercera función que representa la magnitud en la que se superponen f y una versión trasladada e invertida de g.

Definición para funciones continuas

$$(f * g)(\mathbf{x}) = \int f(\mathbf{z})g(\mathbf{x} - \mathbf{z})d\mathbf{z}.$$

Definición para funciones discretas

$$(f * g)(i) = \sum_a f(a)g(i - a)$$

En terminología de redes esto se traduce por el producto de dos arrays multidimensionales, llamados **tensores**.

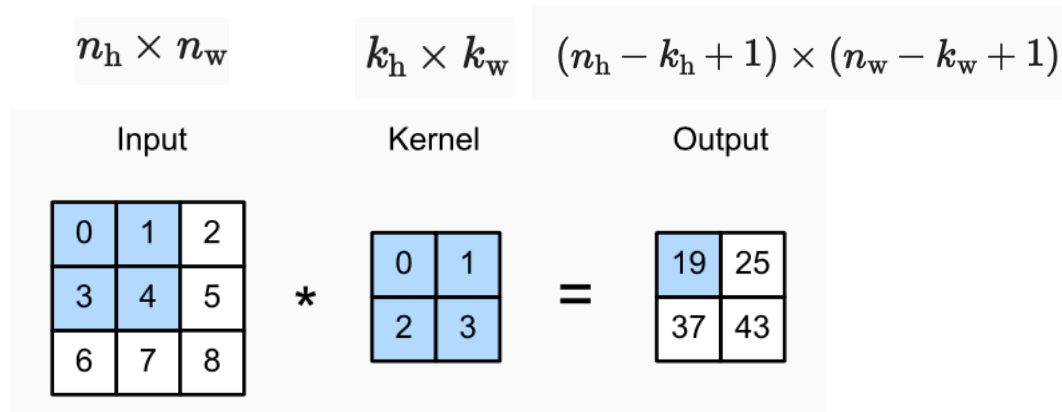
El primer tensor es el *input* y el segundo se llama *kernel*

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n).$$

CNN: Convolución

Filtro: Conjunto de kernels



$$\begin{aligned} 0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 &= 19, \\ 1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 &= 25, \\ 3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 &= 37, \\ 4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 &= 43. \end{aligned}$$

CNN: Convolución

Kernel

0	1	2
2	2	0
0	1	2

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3 ₀	2 ₁	1 ₂	0
0	0 ₂	1 ₂	3 ₀	1
3	1 ₀	2 ₁	2 ₂	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2 ₀	1 ₁	0 ₂
0	0	1 ₂	3 ₂	1 ₀
3	1	2 ₀	2 ₁	3 ₂
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0 ₀	0 ₁	1 ₂	3	1
3 ₂	1 ₂	2 ₀	2	3
2 ₀	0 ₁	0 ₂	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0 ₀	1 ₁	3 ₂	1
3	1 ₂	2 ₂	2 ₀	3
2	0 ₀	0 ₁	2 ₂	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0	1 ₀	3 ₁	1 ₂
3	1	2 ₂	2 ₂	3 ₀
2	0	0 ₀	2 ₁	2 ₂
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0	1	3	1
3 ₀	1 ₁	2 ₂	2	3
2 ₂	0 ₂	0 ₀	2	2
2 ₀	0 ₁	0 ₂	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

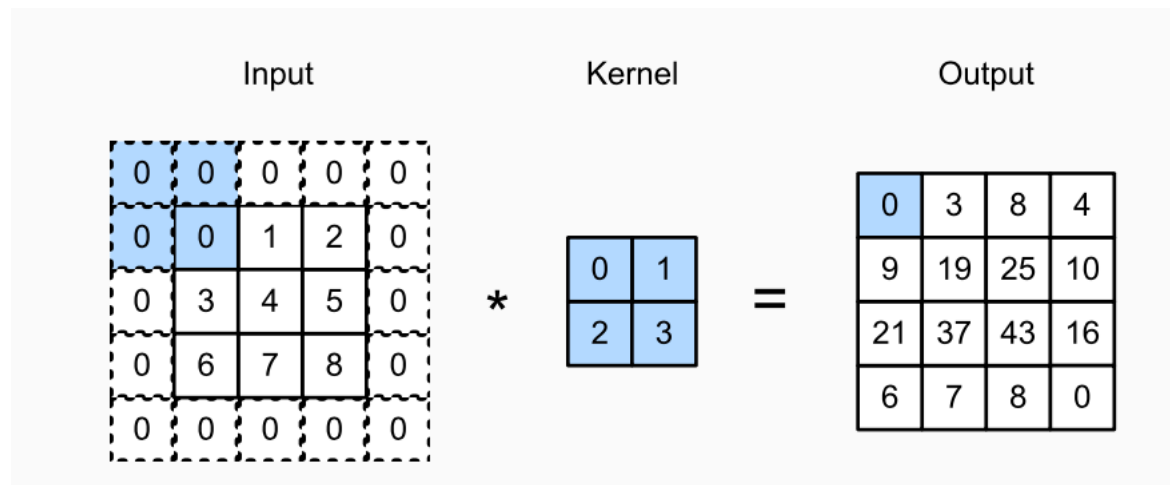
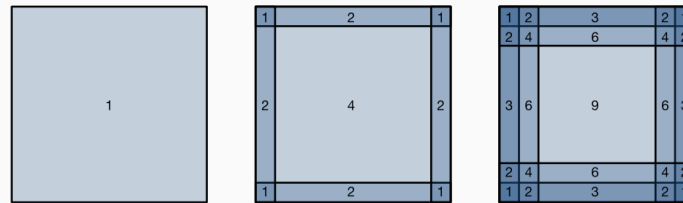
3	3	2	1	0
0	0	1	3	1
3	1 ₀	2 ₁	2 ₂	3
2	0 ₂	0 ₂	2 ₀	2
2	0 ₀	0 ₁	0 ₂	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

3	3	2	1	0
0	0	1	3	1
3	1	2 ₀	2 ₁	3 ₂
2	0	0 ₂	2 ₂	2 ₀
2	0	0 ₀	0 ₁	1 ₂

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

CNN: Relleno - *Padding*



Si añadimos p_h filas y p_w columnas

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1).$$

CNN: Relleno - *Padding*

Input

0	1	2
3	4	5
6	7	8

*

Kernel

0	1
2	3

=

Output

19	25
37	43

Input

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

*

Kernel

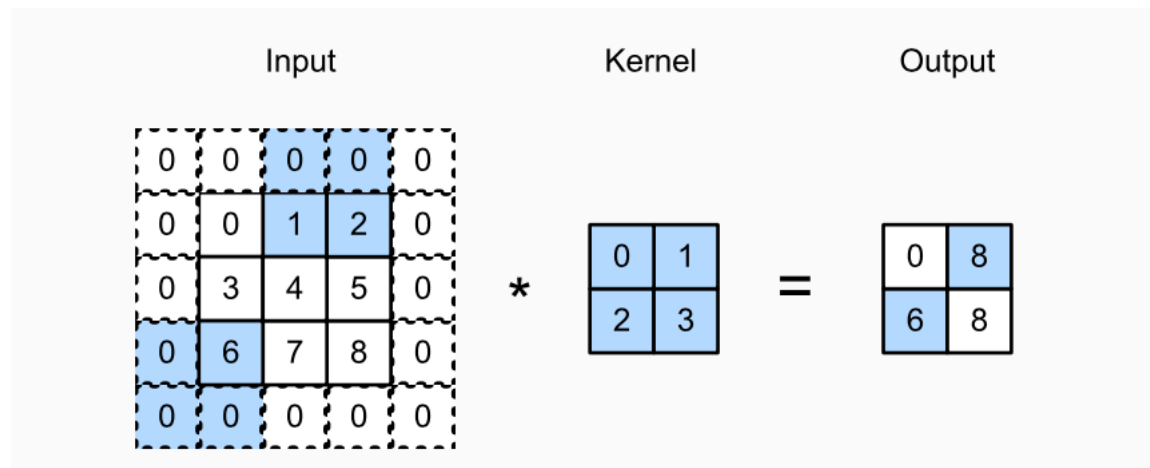
0	1
2	3

=

Output

0	3	8	4
9	19	25	10
21	37	43	16
6	7	8	0

CNN: Paso - *Stride*



Si añadimos un paso de s_h filas y s_w columnas

$$\left\lfloor \frac{(n_h - k_h + p_h + s_h)}{s_h} \right\rfloor \times \left\lfloor \frac{(n_w - k_w + p_w + s_w)}{s_w} \right\rfloor$$

CNN: Convolution + padding + stride

0 ₀	0 ₁	0 ₂	0	0	0	0
0 ₂	3 ₂	3 ₀	2	1	0	0
0 ₀	0 ₁	0 ₂	1	3	1	0
0	3	1	2	2	3	0
0	2	0	0	2	2	0
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

0	0	0 ₀	0 ₁	0 ₂	0	0
0	3	3 ₂	2 ₂	1 ₀	0	0
0	0	0 ₀	1 ₁	3 ₂	1	0
0	3	1	2	2	3	0
0	2	0	0	2	2	0
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

0	0	0	0	0 ₀	0 ₁	0 ₂
0	3	3	2	1 ₂	0 ₂	0 ₀
0	0	0	1	3 ₀	1 ₁	0 ₂
0	3	1	2	2	3	0
0	2	0	0	2	2	0
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

0	0	0	0	0	0	0
0	3	3	2	1	0	0
0 ₀	0 ₁	0 ₂	1	3	1	0
0 ₂	3 ₂	1 ₀	2	2	3	0
0 ₀	2 ₁	0 ₂	0	2	2	0
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

0	0	0	0	0	0	0
0	3	3	2	1	0	0
0	0	0 ₀	1 ₁	3 ₂	1	0
0	3	1 ₂	2 ₂	2 ₀	3	0
0	2	0 ₀	0 ₁	2 ₂	2	0
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

0	0	0	0	0	0	0
0	3	3	2	1	0	0
0	0	0	1	3 ₀	1 ₁	0 ₂
0	3	1	2	2 ₂	3 ₂	0 ₀
0	2	0	0	2 ₀	2 ₁	0 ₂
0	2	0	0	0	1	0
0	0	0	0	0	0	0

6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

0	0	0	0	0	0	0
0	3	3	2	1	0	0
0	0	0	1	3	1	0
0	3	1	2	2	3	0
0 ₀	2 ₁	0 ₂	0	2	2	0
0 ₂	2 ₂	0 ₀	0	0	1	0
0	0	0 ₁	0 ₂	0	0	0

6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

0	0	0	0	0	0	0
0	3	3	2	1	0	0
0	0	0	1	3	1	0
0	3	1	2	2	3	0
0	2	0	0	0	2	0
0	2	0 ₂	0 ₂	0 ₀	1	0
0	0	0 ₀	0 ₁	0 ₂	0	0

6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

0	0	0	0	0	0	0
0	3	3	2	1	0	0
0	0	0	1	3	1	0
0	3	1	2	2	3	0
0	2	0	0	2 ₀	2 ₁	0 ₂
0	2	0	0	0 ₂	1 ₂	0 ₀
0	0	0	0	0	0 ₁	0 ₂

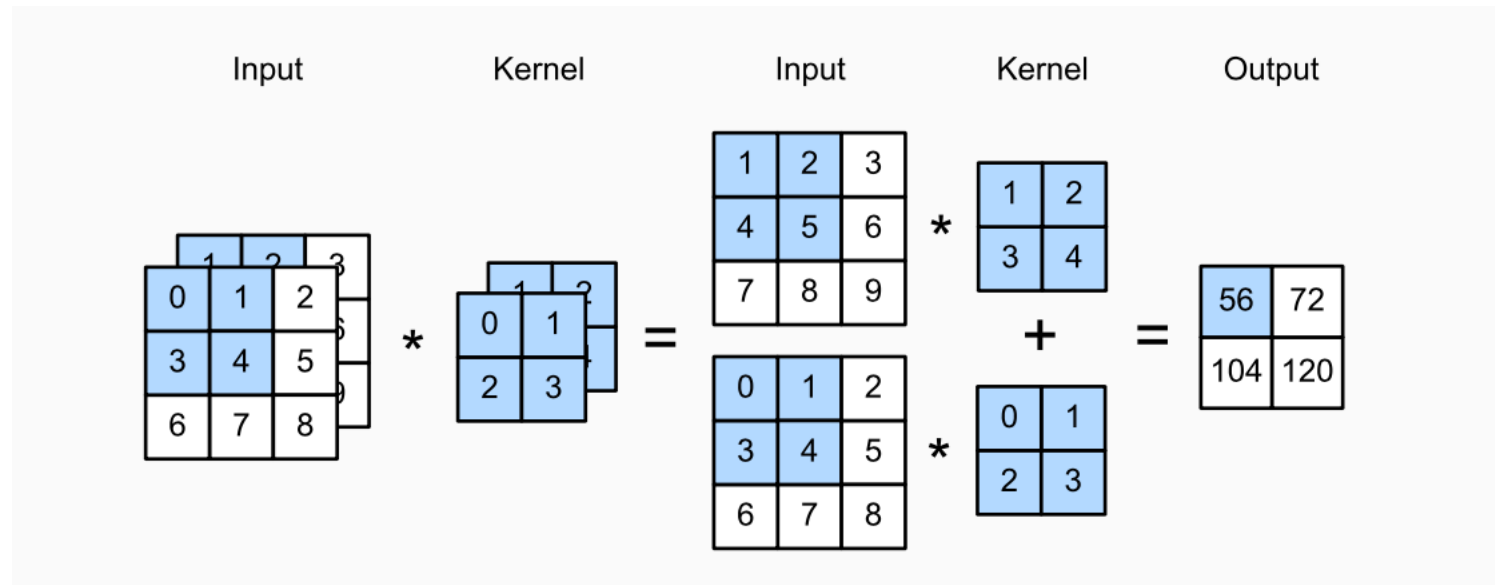
6.0	17.0	3.0
8.0	17.0	13.0
6.0	4.0	4.0

CNN: Función de activación

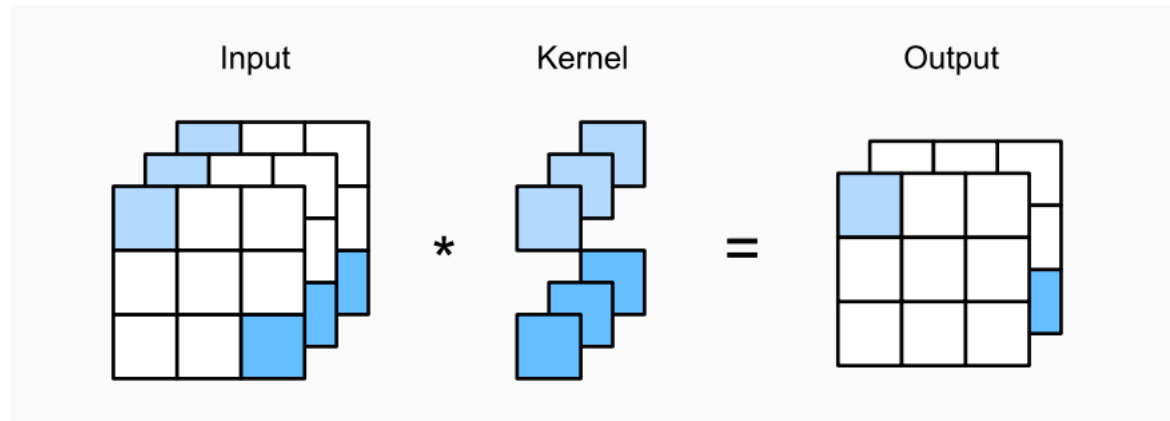
Añade no linealidad

- Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$
- Tanh $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- ReLU $f(x) = \max(0, x)$

CNN: Múltiples canales de entrada



CNN: Convolución



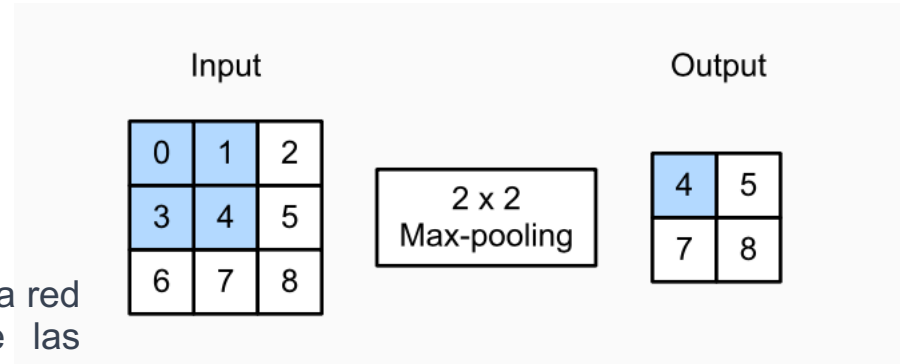
CNN: Capa de Pooling

Pooling:

- ❑ La capa de agrupación reemplaza la salida de la red en ciertas ubicaciones por un resumen de las salidas cercanas.
- ❑ Reducción del tamaño espacial de la representación
- ❑ La operación de agrupación se procesa en cada segmento de la representación de forma individual.

Funciones de pooling:

- Media
- Norma L2
- Media ponderada
- Máximo



CNN: Capa de Pooling

1	1	2	4
5	6	7	8
3	2	1	9
1	2	3	4

Max-pool con filtro 2x2

6	7	8
6	7	9
3	3	9

CNN: Capa de Pooling

1	1	2	4
5	6	7	8
3	2	1	9
1	2	3	4

Max-pool con filtro 2x2

6	7	8
6	7	9
3	3	9

CNN: Capa de Pooling

1	1	2	4
5	6	7	8
3	2	1	9
1	2	3	4

Max-pool con filtro 2x2

6	7	8
6	7	9
3	3	9

CNN: Capa de Pooling

1	1	2	4
5	6	7	8
3	2	1	9
1	2	3	4

Max-pool con filtro 2x2

6	7	8
6	7	9
3	3	9

CNN: Capa de Pooling

1	1	2	4
5	6	7	8
3	2	1	9
1	2	3	4

Max-pool con filtro 2x2

6	7	8
6	7	9
3	3	9

CNN: Capa de Pooling

1	1	2	4
5	6	7	8
3	2	1	9
1	2	3	4

Max-pool con filtro 2x2

6	7	8
6	7	9
3	3	9

CNN: Capa de Pooling

1	1	2	4
5	6	7	8
3	2	1	9
1	2	3	4

Max-pool con filtro 2x2

6	7	8
6	7	9
3	3	9

CNN: Capa de Pooling

1	1	2	4
5	6	7	8
3	2	1	9
1	2	3	4

Max-pool con filtro 2x2

6	7	8
6	7	9
3	3	9

CNN: Capa de Pooling

1	1	2	4
5	6	7	8
3	2	1	9
1	2	3	4

Max-pool con filtro 2x2

6	7	8
6	7	9
3	3	9

1	1	2	4
5	6	7	8
3	2	1	9
1	2	3	4

Max-pool con filtro 2x2

Paso 2

6	8
3	9

CNN

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

Y ahora...diseñar arquitectura

