

Redes Neuronales Artificiales

BIG DATA with PYTHON



INTRODUCTION TO
PYTHON FOR BIG DATA

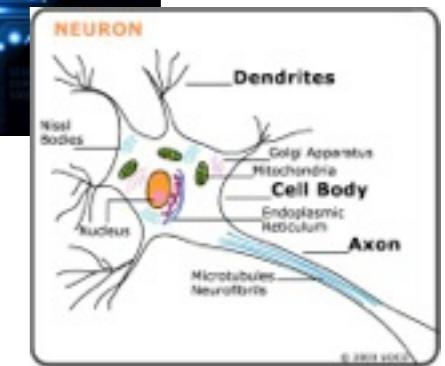
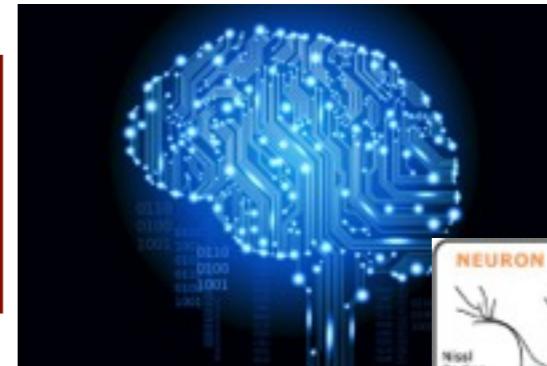


<https://github.com/biuse/NeuralNetworks2019>

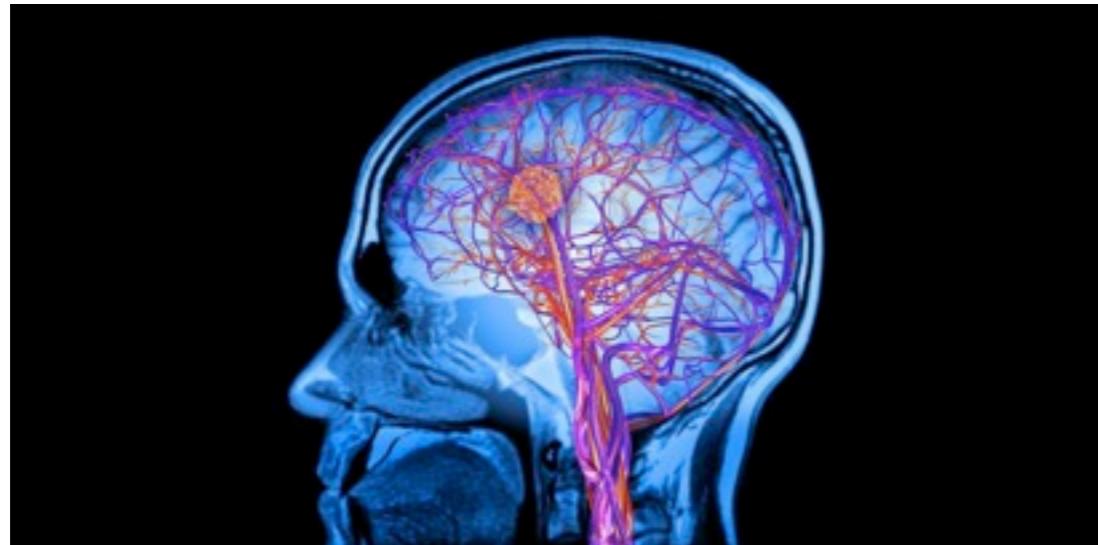
REDES NEURONALES

Herramienta de *machine learning* que nació con dos objetivos:

- modelizar las neuronas biológicas
- la inteligencia artificial (McCulloch and Pitts 1943)

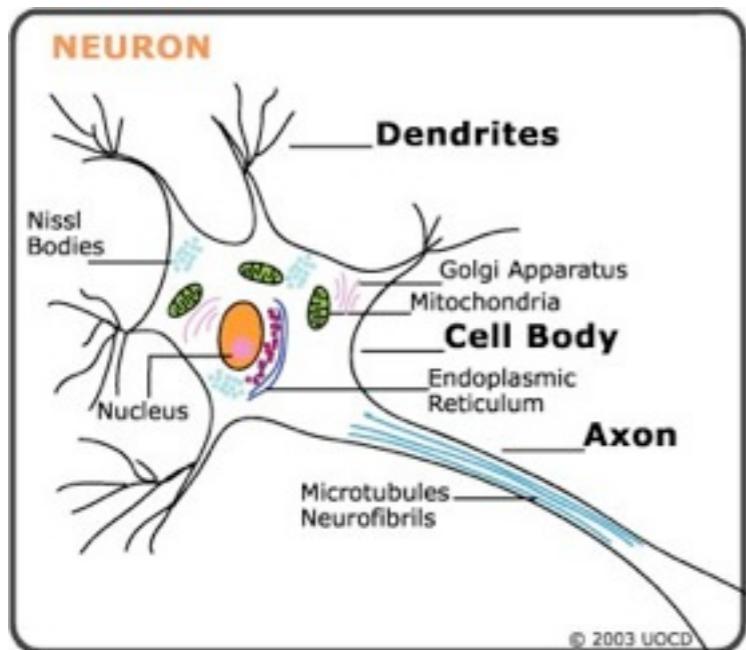


La idea era desarrollar un algoritmo que fuera capaz de aprender, emulando el comportamiento del cerebro humano

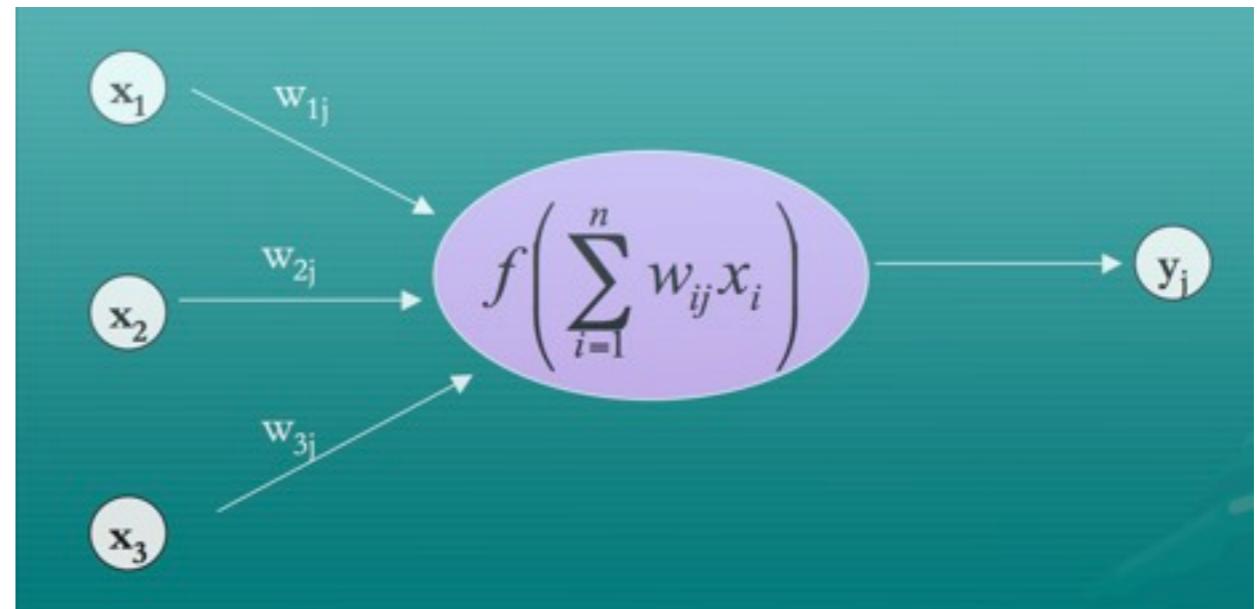


Millones de neuronas
altamente
interconnectadas

REDES NEURONALES

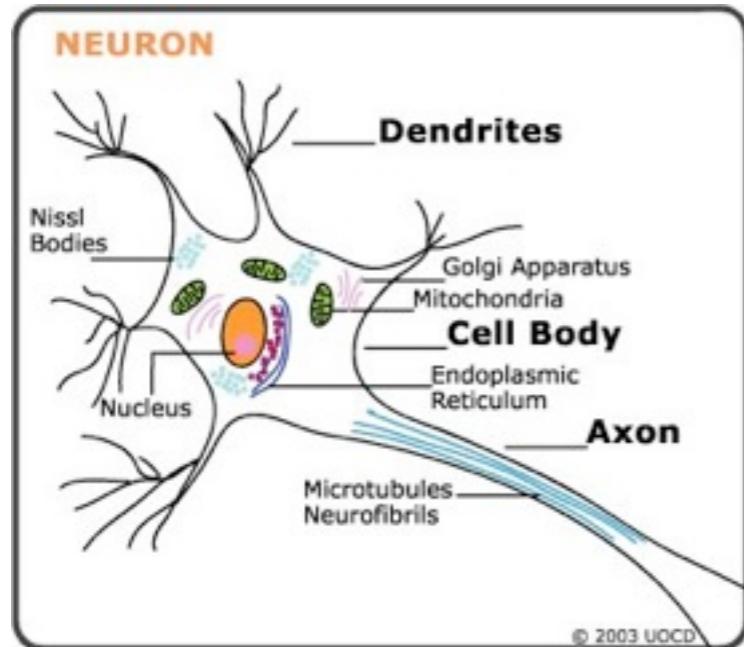


Neurona biológica (simplificación)



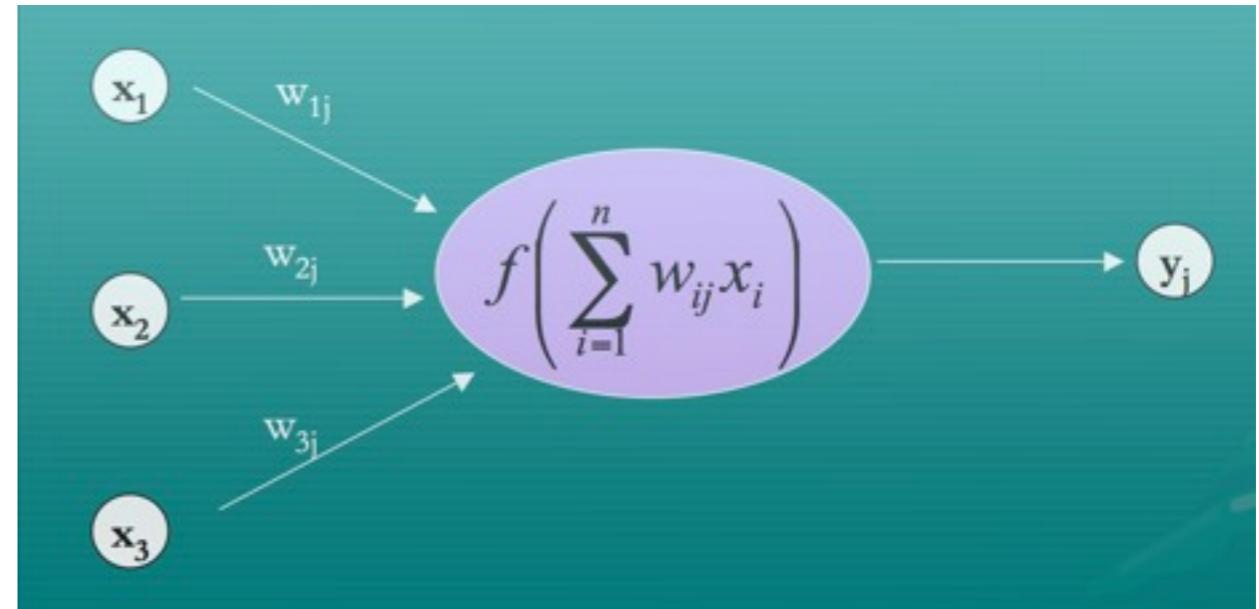
Neurona artificial

REDES NEURONALES



Neurona biológica (simplificación)

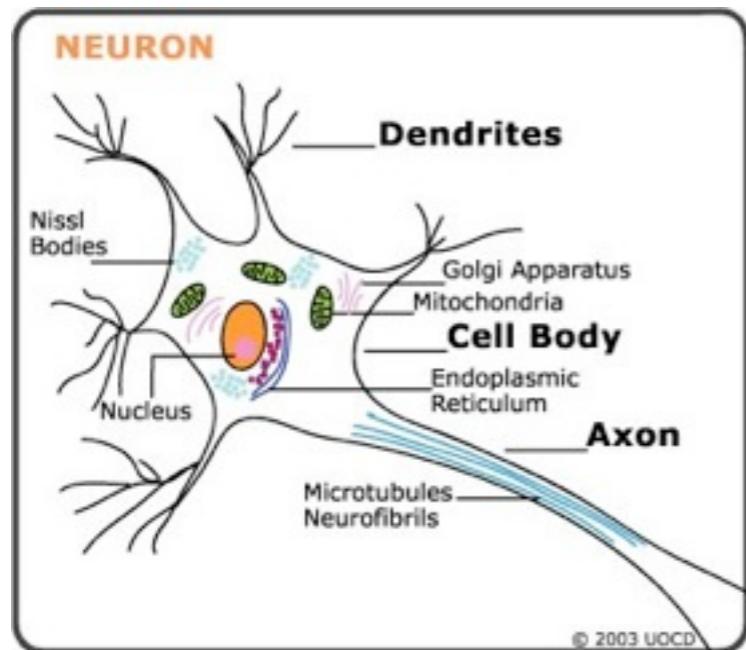
La información llega a las neuronas por las **dendritas**



Neurona artificial

Enlaces de llegada a la neurona, caracterizados por unos **pesos** (w) que dan la fortaleza de esa conexión.

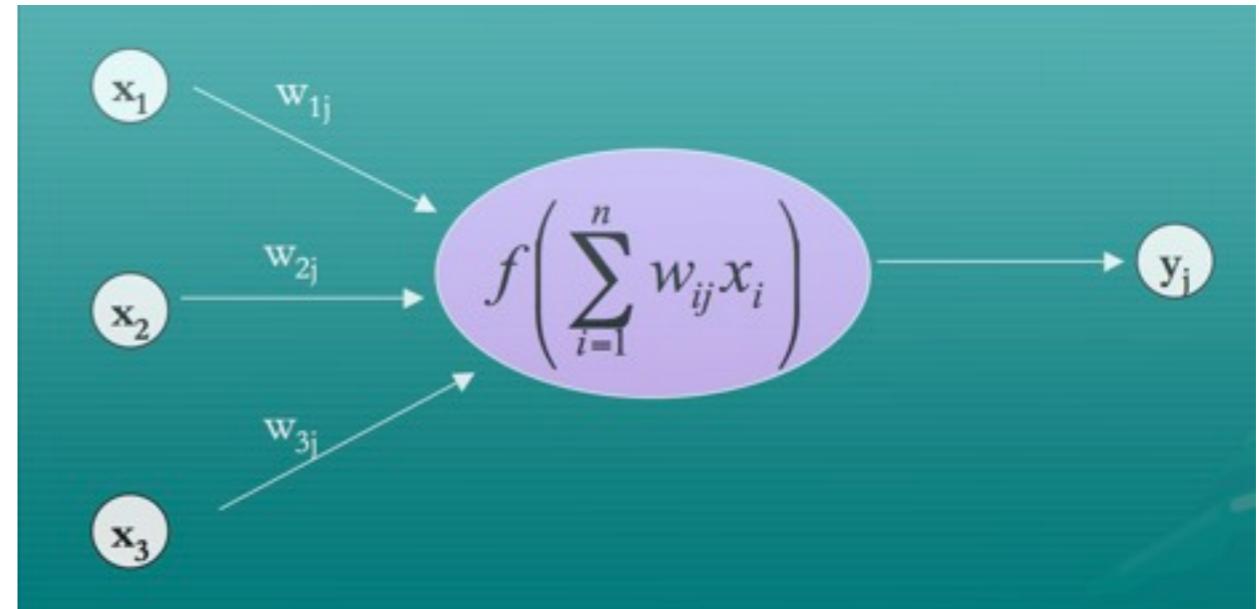
REDES NEURONALES



Neurona biológica (simplificación)

La información llega a las neuronas por las **dendritas**

En el **núcleo** de la célula hay el **proceso biológico** que procesa la señal.

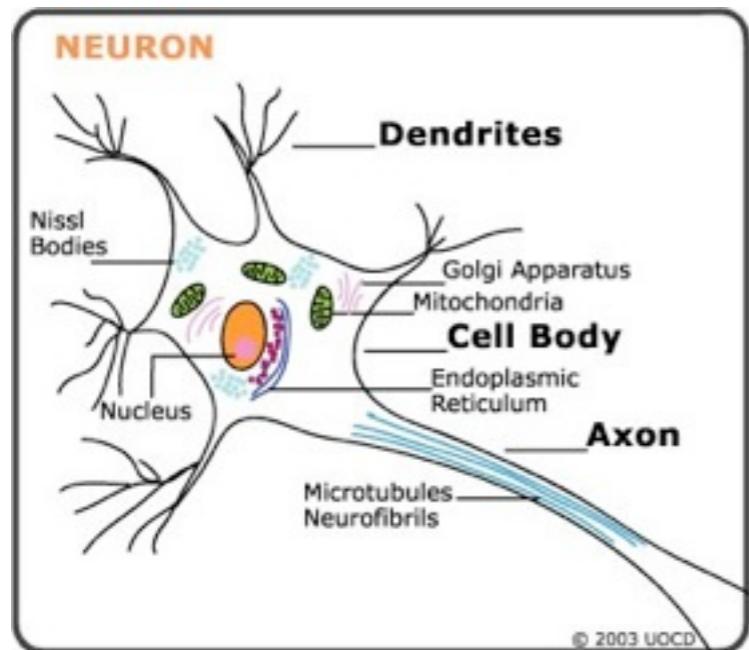


Neurona artificial

Enlaces de llegada a la neurona, caracterizados por unos **pesos** (w) que dan la fortaleza de esa conexión.

En el núcleo de la neurona artificial, se aplica una función a una combinación lineal de los inputs y los pesos. (**Función de activación**).

REDES NEURONALES

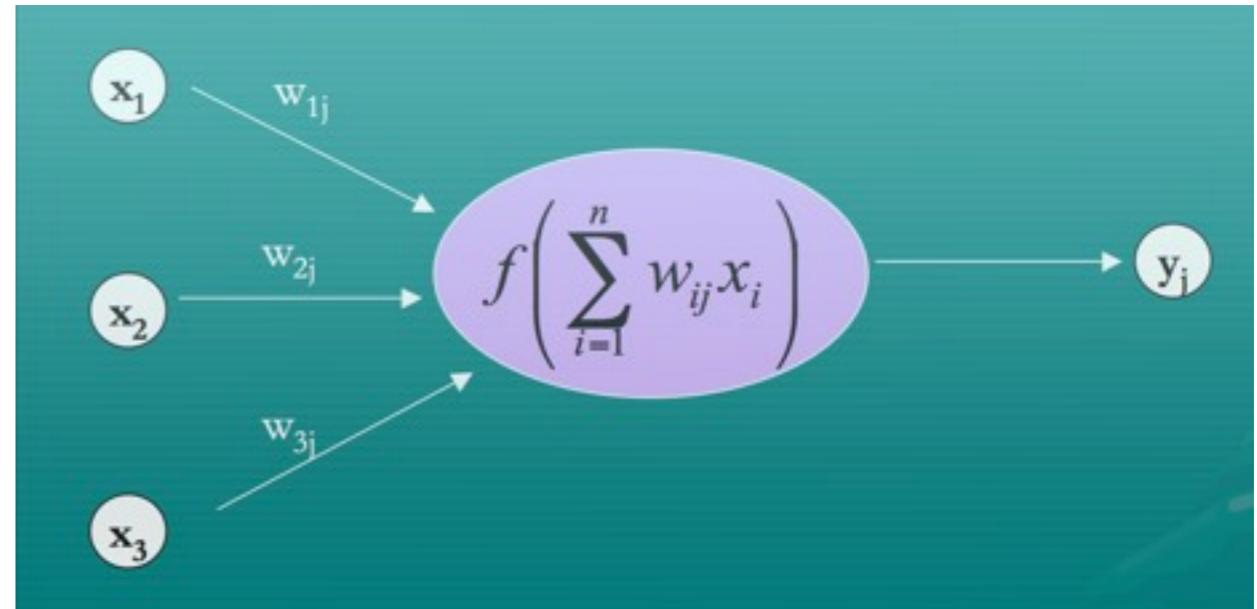


Neurona biológica (simplificación)

La información llega a las neuronas por las **dendritas**

En el **núcleo** de la célula hay el **proceso biológico** que procesa la señal.

Axon: Prolongación de la célula que lleva la información procesada por el núcleo hasta la dendrita de otra neurona



Neurona artificial

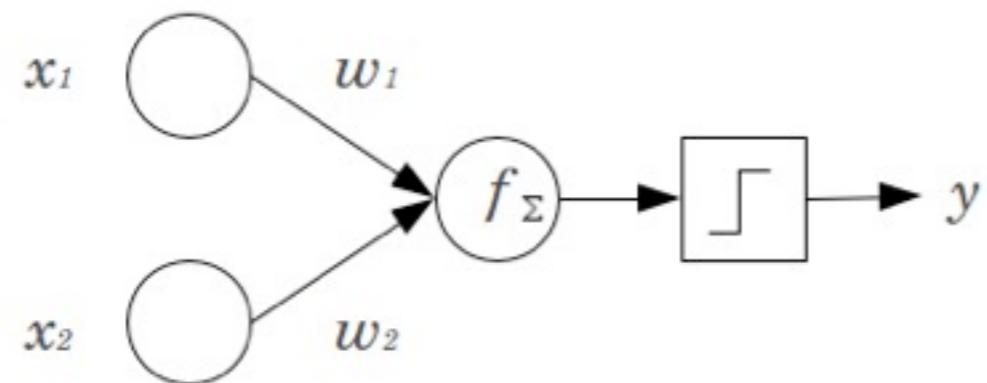
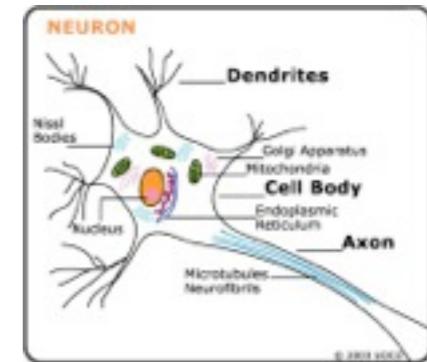
Enlaces de llegada a la neurona, caracterizados por unos **pesos** (w) que dan la fortaleza de esa conexión.

En el núcleo de la neurona artificial, se aplica una función a una combinación lineal de los inputs y los pesos. (**Función de activación**).

Enlace de salida a la siguiente neurona que lleva información a la siguiente neurona.

REDES NEURONALES

Perceptron. Primer algoritmo sencillo emulando una neurona. Sentó las bases de las redes neuronales actuales.
(Rosenblatt 1958)



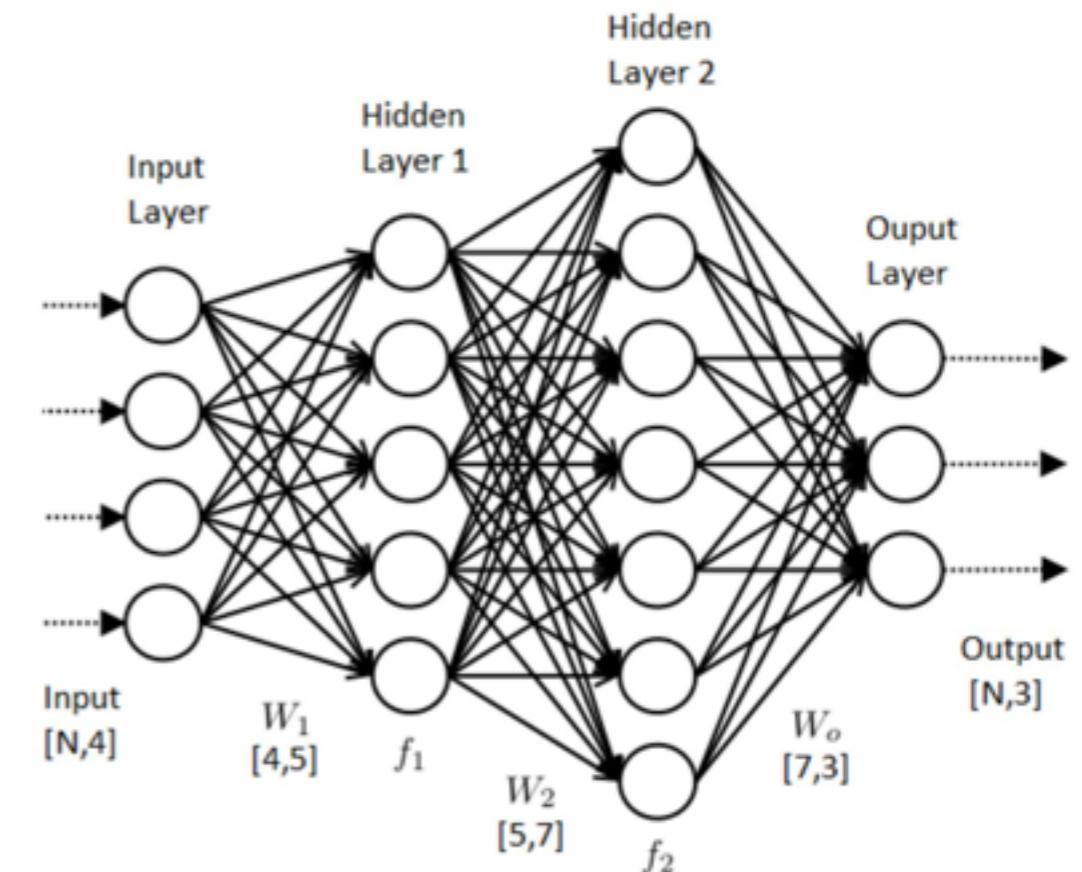
$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

Clasificador binario

REDES NEURONALES

Red neuronal biológica

Un cerebro humano tiene alrededor de 10^{11} neuronas, cada una de ellas con unos pocos de miles de conexiones.



Red neuronal artificial (ANN)

Se organiza, en general, por capas unas pocas neuronas conectadas entre ellas.

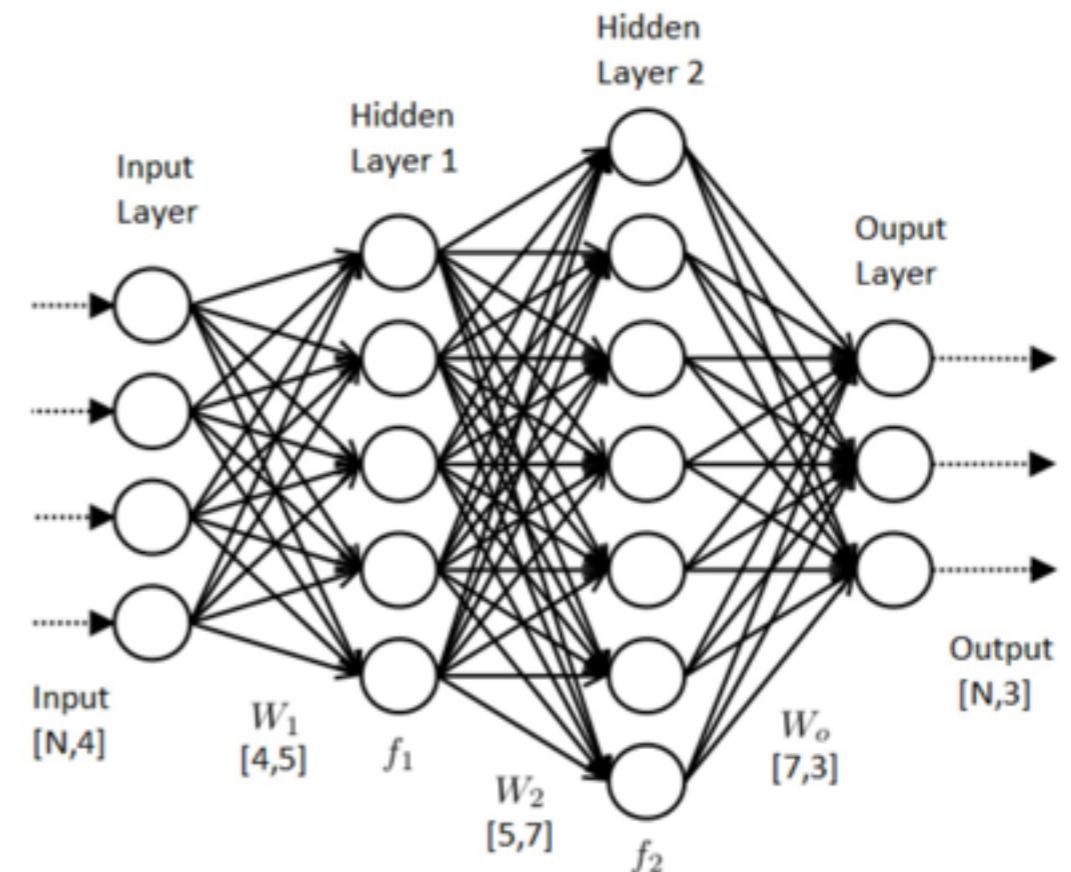
REDES NEURONALES

Red neuronal biológica

Un cerebro humano tiene alrededor de 10^{11} neuronas, cada una de ellas con unos pocos de miles de conexiones.



Una red neuronal biológica es muy compleja, la ANN que presentamos aquí tiene sus limitaciones, pero es muy eficiente en algunos problemas.



Red neuronal artificial (ANN)

Se organiza, en general, por capas unas pocas neuronas conectadas entre ellas.

REDES NEURONALES

TIPOS

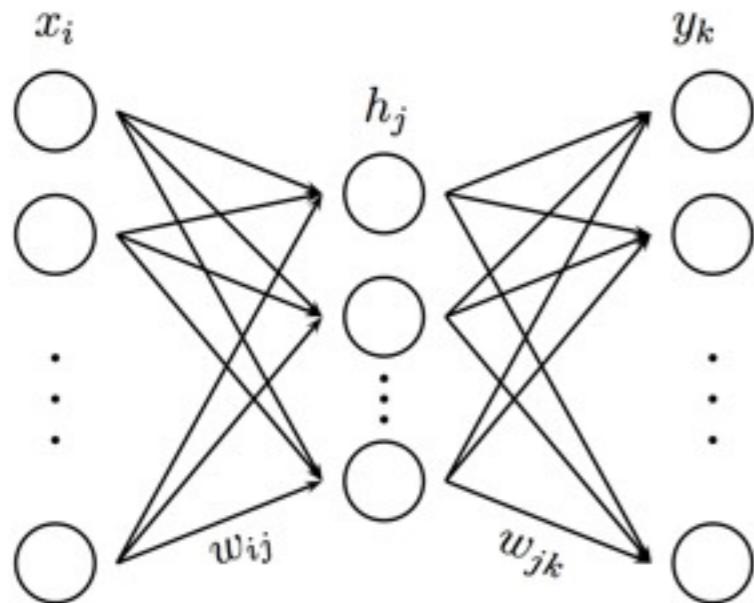
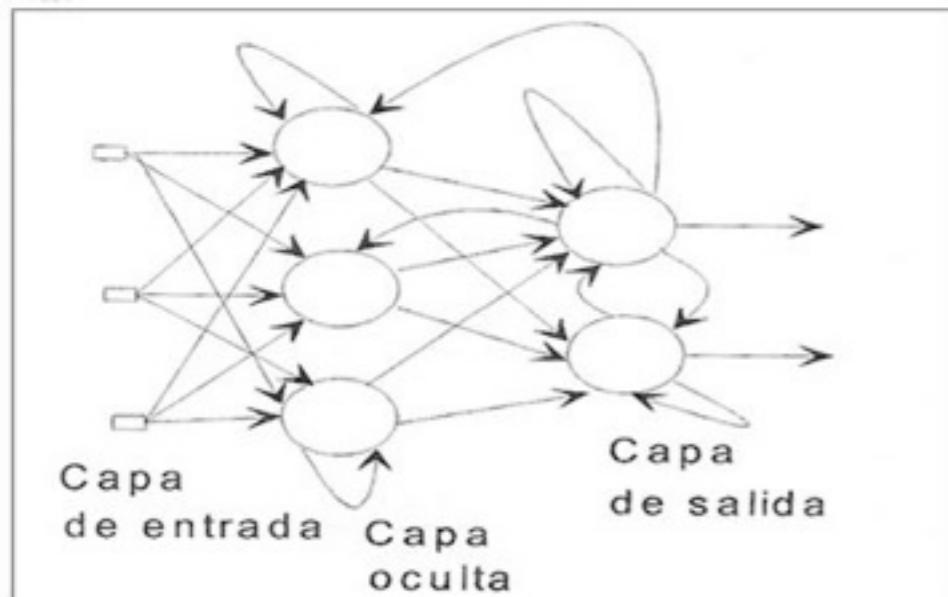
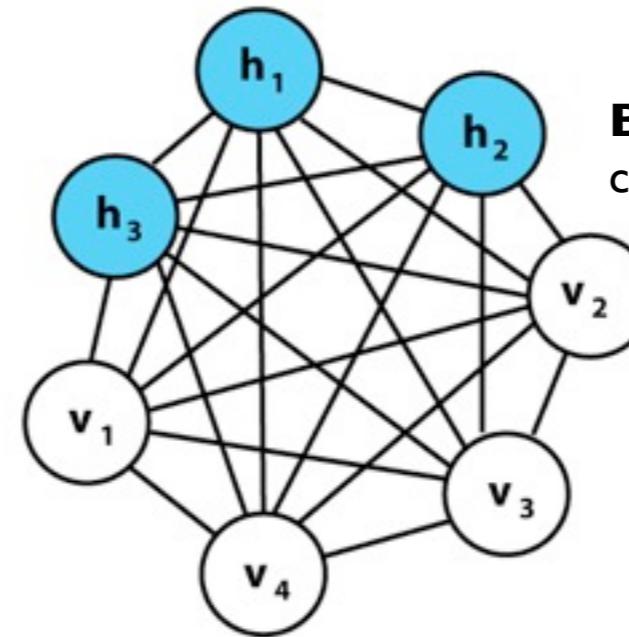


Figure 1. Schematic of a 3-layer feed-forward neural network.

Redes neuronales recurrentes. Pueden haber conexiones en varias direcciones



Feed-forward. Solo conexiones hacia delante



Boltzmann machine. Todas conectadas con todas

Redes neuronales convolucionales. deep learning

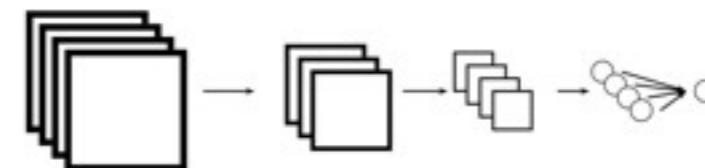


Figure 2. Schematic convolutional neural network example, with 2 hidden layers in the CNN part and 1 hidden layer in the FC part. In this example, the input depth is $k = 4$, the first layer depth $m_1 = 3$ and the second layer depth is $m_2 = 4$, the number of hidden nodes in the FC hidden layer is 4, and only one output is present.

REDES NEURONALES

TIPOS

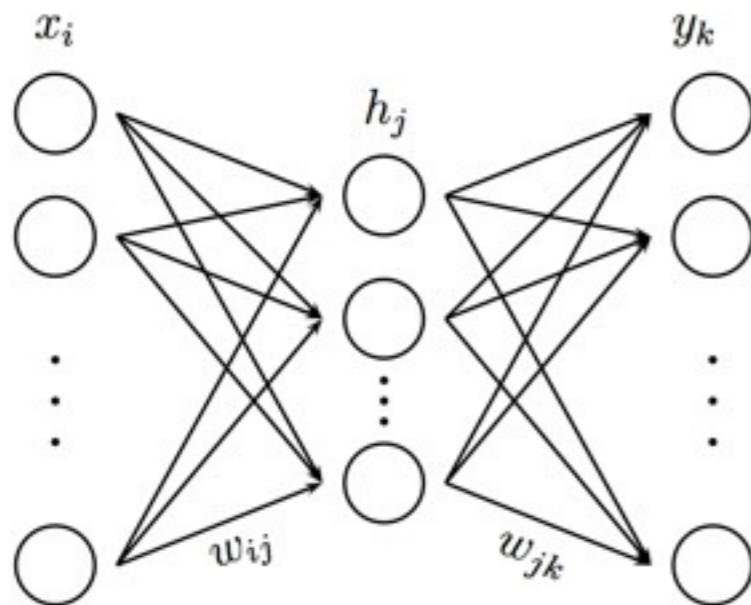
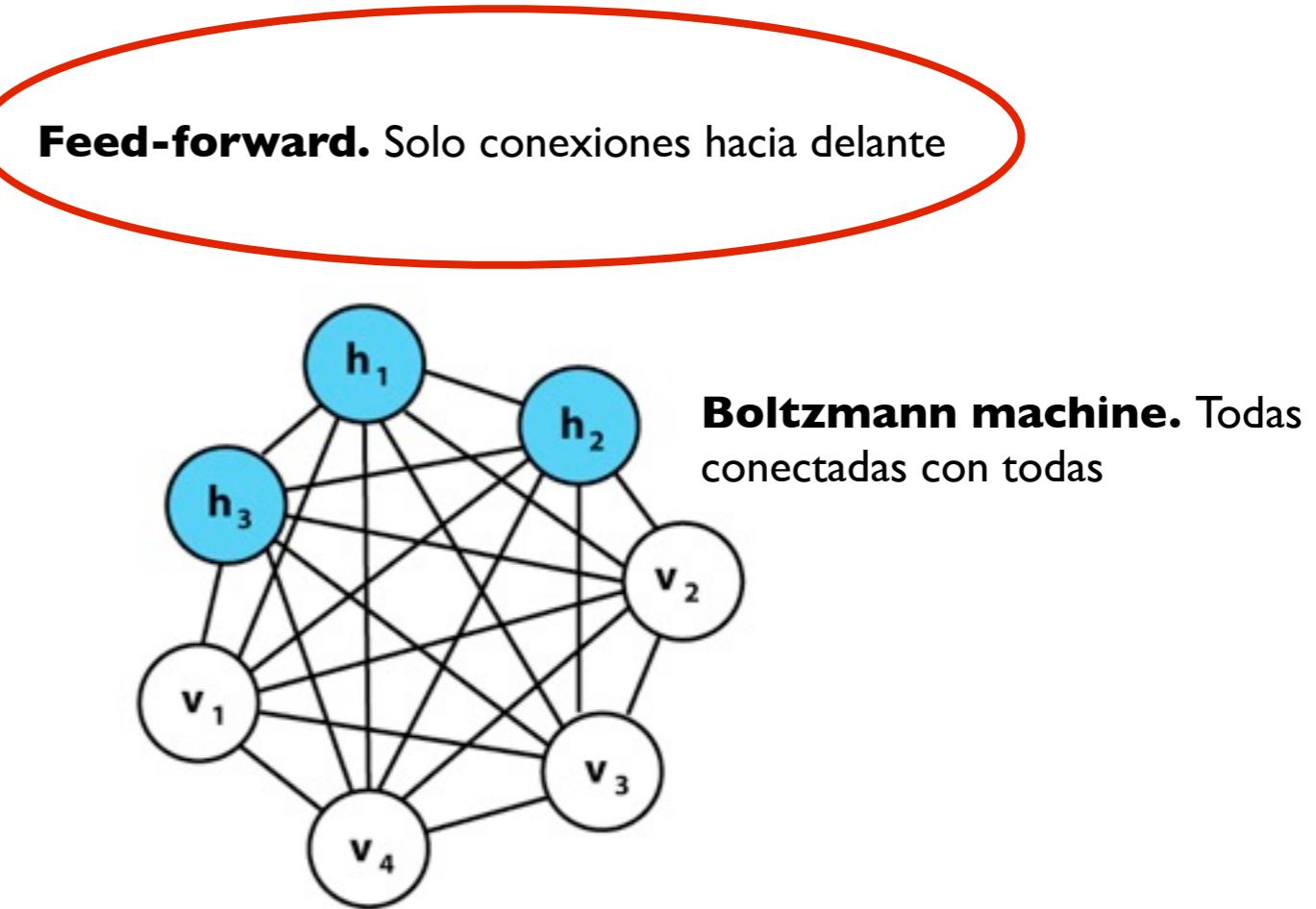
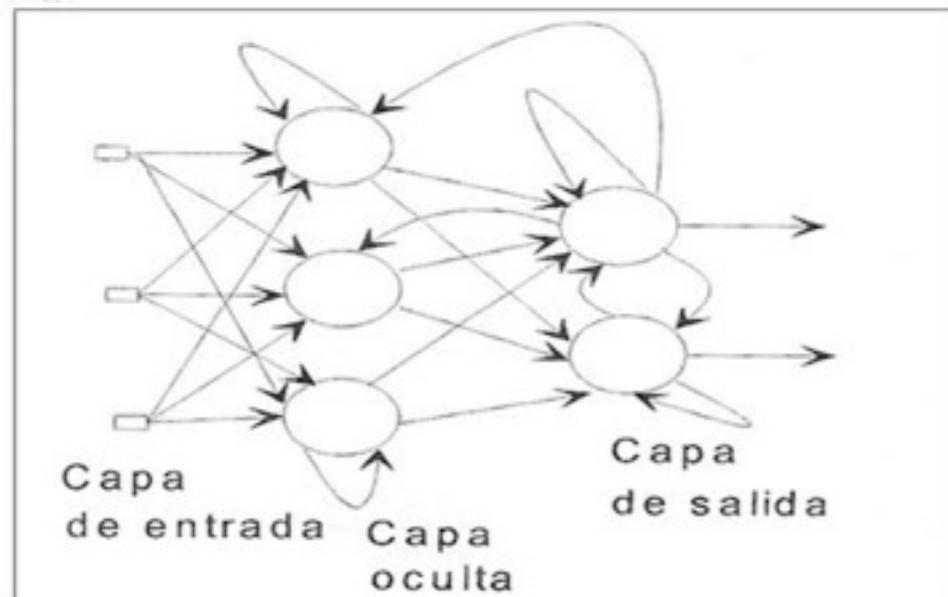


Figure 1. Schematic of a 3-layer feed-forward neural network.

Redes neuronales recurrentes. Pueden haber conexiones en varias direcciones



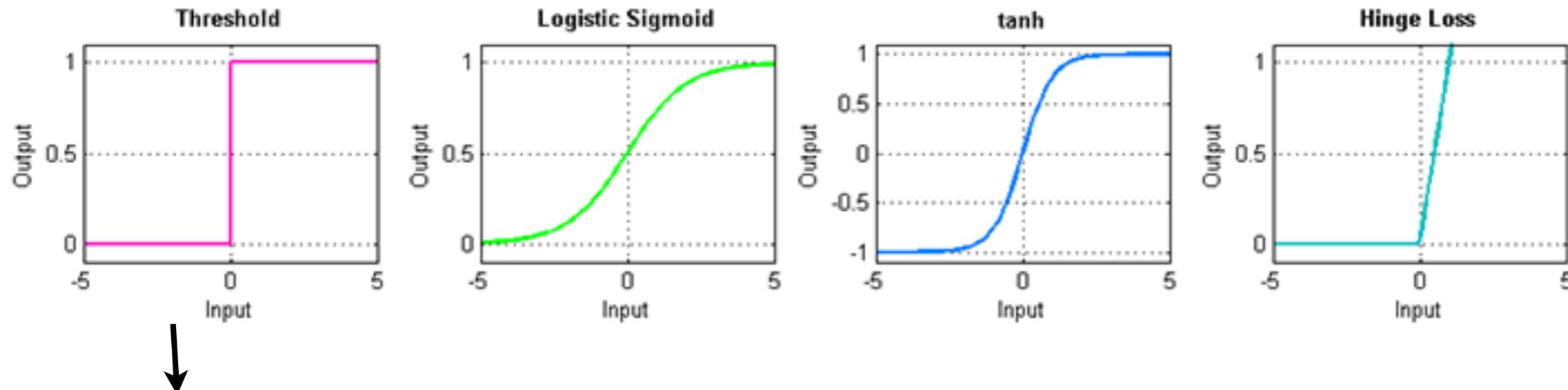
Redes neuronales convolucionales. deep learning



Figure 2. Schematic convolutional neural network example, with 2 hidden layers in the CNN part and 1 hidden layer in the FC part. In this example, the input depth is $k = 4$, the first layer depth $m_1 = 3$ and the second layer depth is $m_2 = 4$, the number of hidden nodes in the FC hidden layer is 4, and only one output is present.

REDES NEURONALES

FUNCIONES DE ACTIVACIÓN



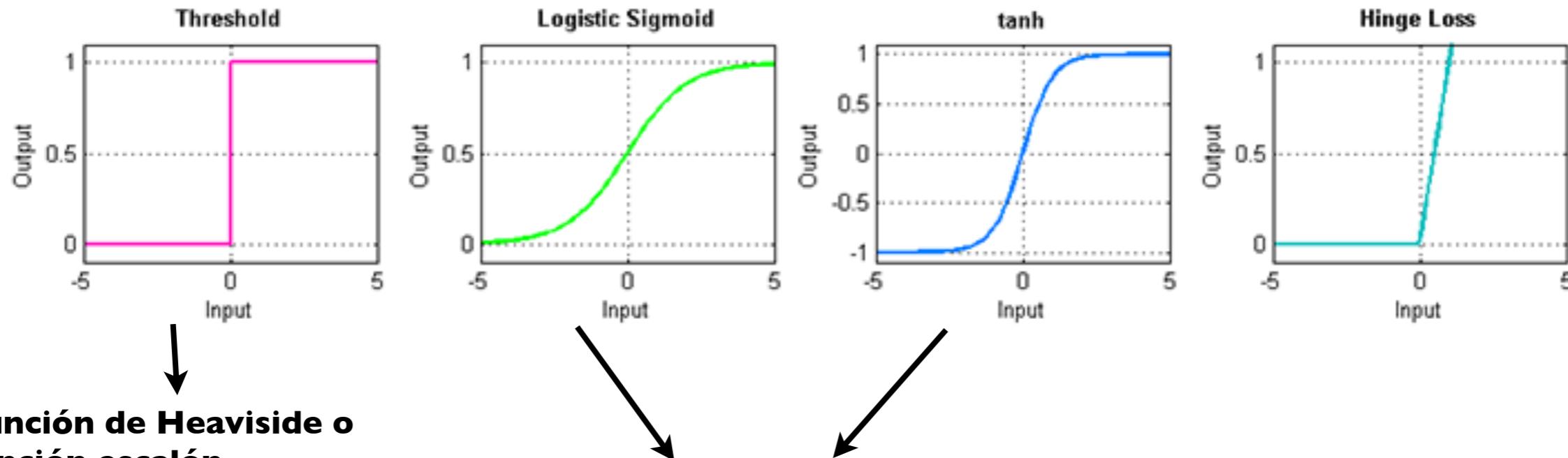
↓
Función de Heaviside o función escalón.

$$H[n] = \begin{cases} 0, & n < 0, \\ 1, & n \geq 0, \end{cases}$$

Originalmente se creía que las neuronas biológicas se **activaban o no**. Por eso esta fue la función escogida en los primeros modelos.

REDES NEURONALES

FUNCIONES DE ACTIVACIÓN



Función de Heaviside o función escalón.

$$H[n] = \begin{cases} 0, & n < 0, \\ 1, & n \geq 0, \end{cases}$$

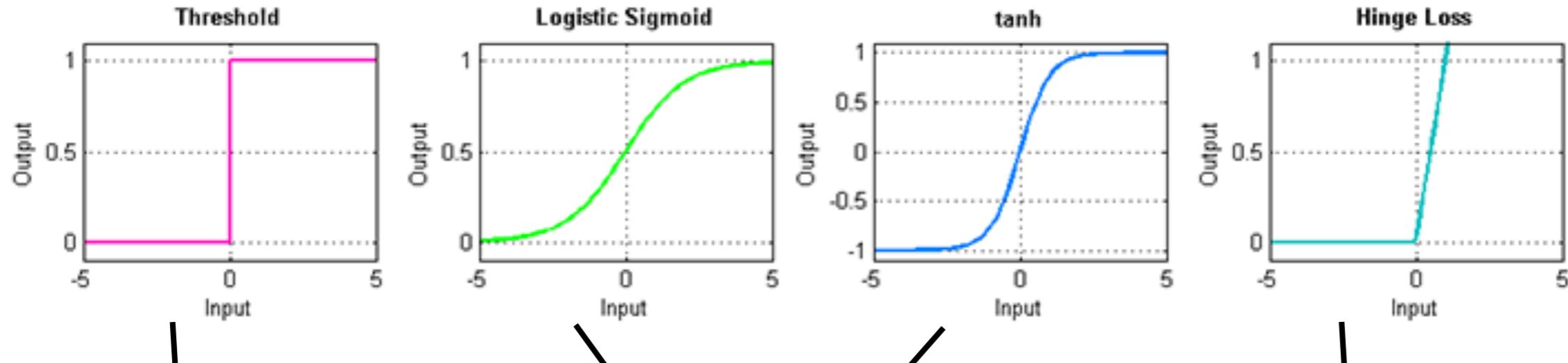
Originalmente se creía que las neuronas biológicas se **activaban o no**. Por eso esta fue la función escogida en los primeros modelos.

Tangente hiperbólica o logistic sigmoid. Funciones derivables hacen que el problema sea más fácil de tratar matemáticamente. Siguen siendo funciones de activación, pero continuas, la más usada es tanh

$$f(x) = \frac{1}{1+e^{-x}} \quad f(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

REDES NEURONALES

FUNCIONES DE ACTIVACIÓN



Función de Heaviside o función escalón.

$$H[n] = \begin{cases} 0, & n < 0, \\ 1, & n \geq 0, \end{cases}$$

Originalmente se creía que las neuronas biológicas se **activaban o no**. Por eso esta fue la función escogida en los primeros modelos.

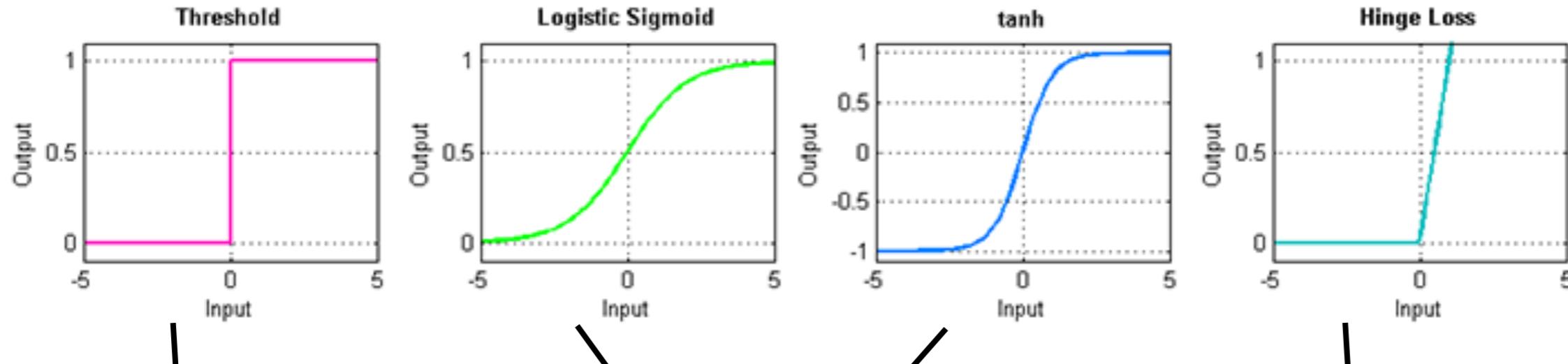
Tangente hiperbólica o logistic sigmoid. Funciones derivables hacen que el problema sea más fácil de tratar matemáticamente. Siguen siendo funciones de activación, pero continuas, la más usada es tanh

$$f(x) = \frac{1}{1+e^{-x}} \quad f(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

Recifier (ReLU). Se usa en problemas de clasificación, a veces se prefieren versiones más suaves.

REDES NEURONALES

FUNCIONES DE ACTIVACIÓN



Función de Heaviside o función escalón.

$$H[n] = \begin{cases} 0, & n < 0, \\ 1, & n \geq 0, \end{cases}$$

Originalmente se creía que las neuronas biológicas se **activaban o no**. Por eso esta fue la función escogida en los primeros modelos.

Tangente hiperbólica o logistic sigmoid. Funciones derivables hacen que el problema sea más fácil de tratar matemáticamente. Siguen siendo funciones de activación, pero continuas, la más usada es tanh

$$f(x) = \frac{1}{1+e^{-x}} \quad f(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

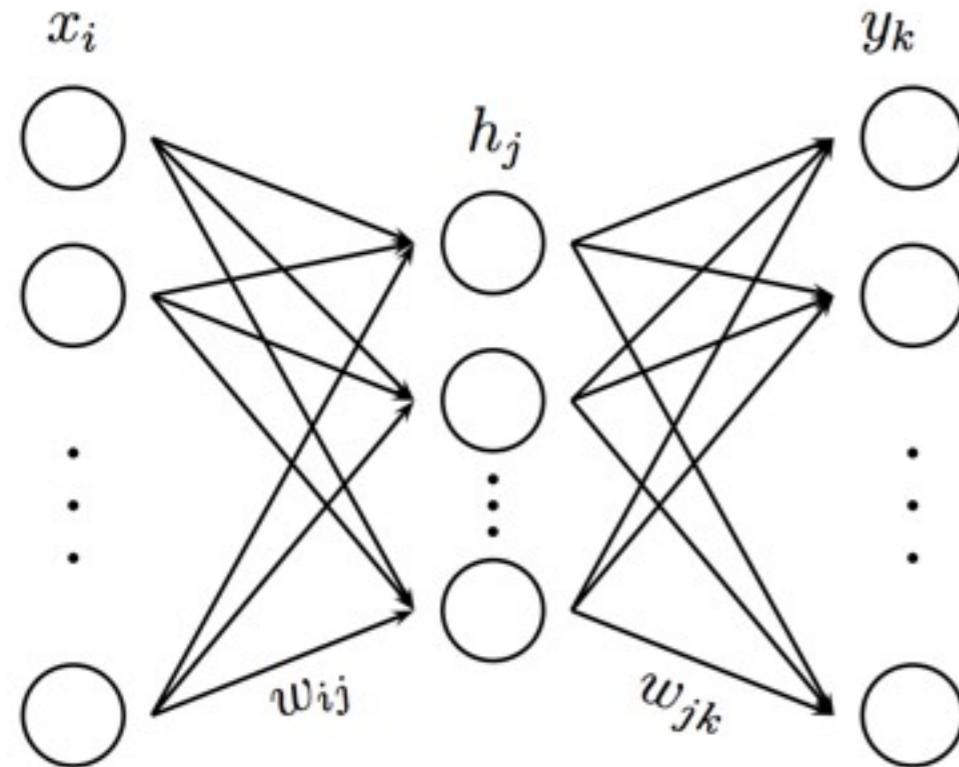
Recifier (ReLU). Se usa en problemas de clasificación, a veces se prefieren versiones más suaves.

Función lineal! Se suele usar en la capa final, así los outputs pueden tener cualquier valor.

$$f(x) = x$$

REDES NEURONALES

ARQUITECTURA



Una red neuronal feed forward se organiza por capas conectadas las unas con las otras.

Figure 1. Schematic of a 3-layer feed-forward neural network.

REDES NEURONALES

ARQUITECTURA

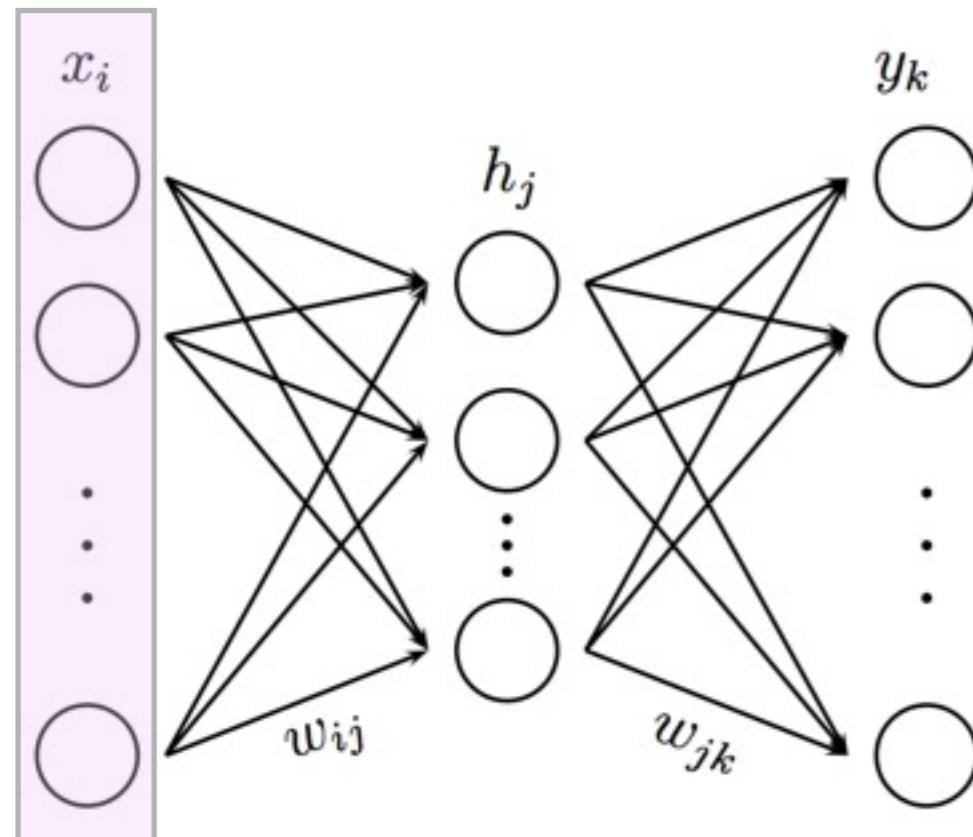


Figure 1. Schematic of a 3-layer feed-forward neural network.

Una red neuronal feed forward se organiza por capas conectadas las unas con las otras.

Están la capa inicial de **inputs** x

REDES NEURONALES

ARQUITECTURA

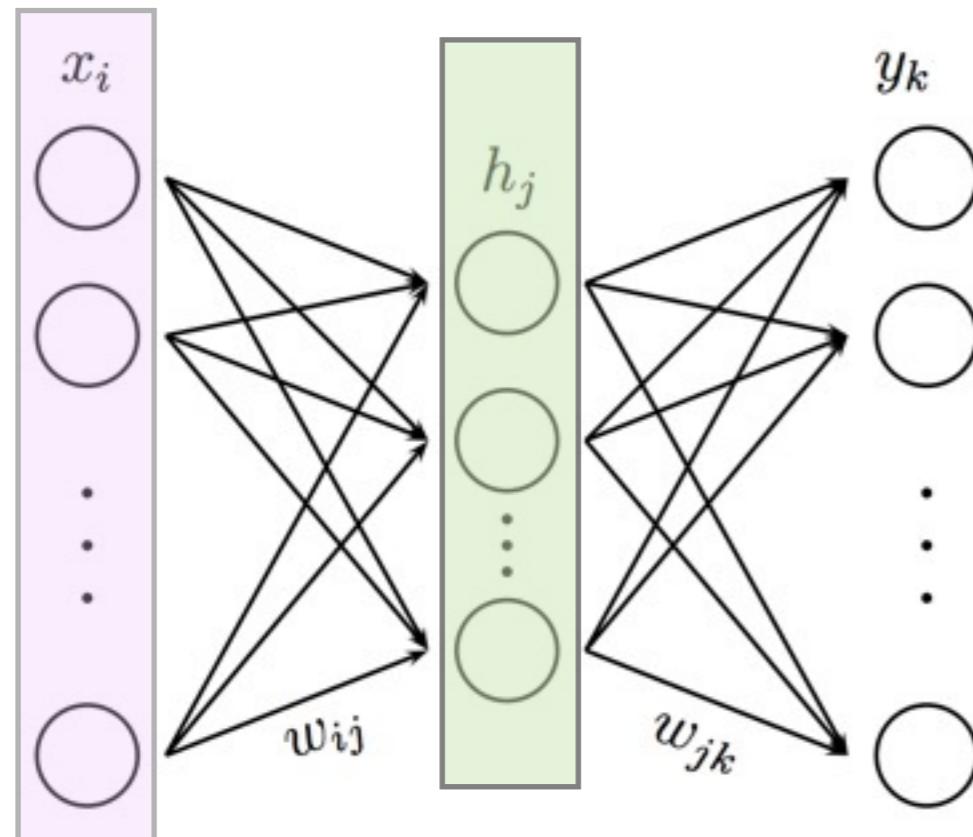


Figure 1. Schematic of a 3-layer feed-forward neural network.

Una red neuronal feed forward se organiza por capas conectadas las unas con las otras.

Están la capa inicial de **inputs** x

Las capas de neuronas (h) que se suelen llamar **capas ocultas**,

REDES NEURONALES

ARQUITECTURA

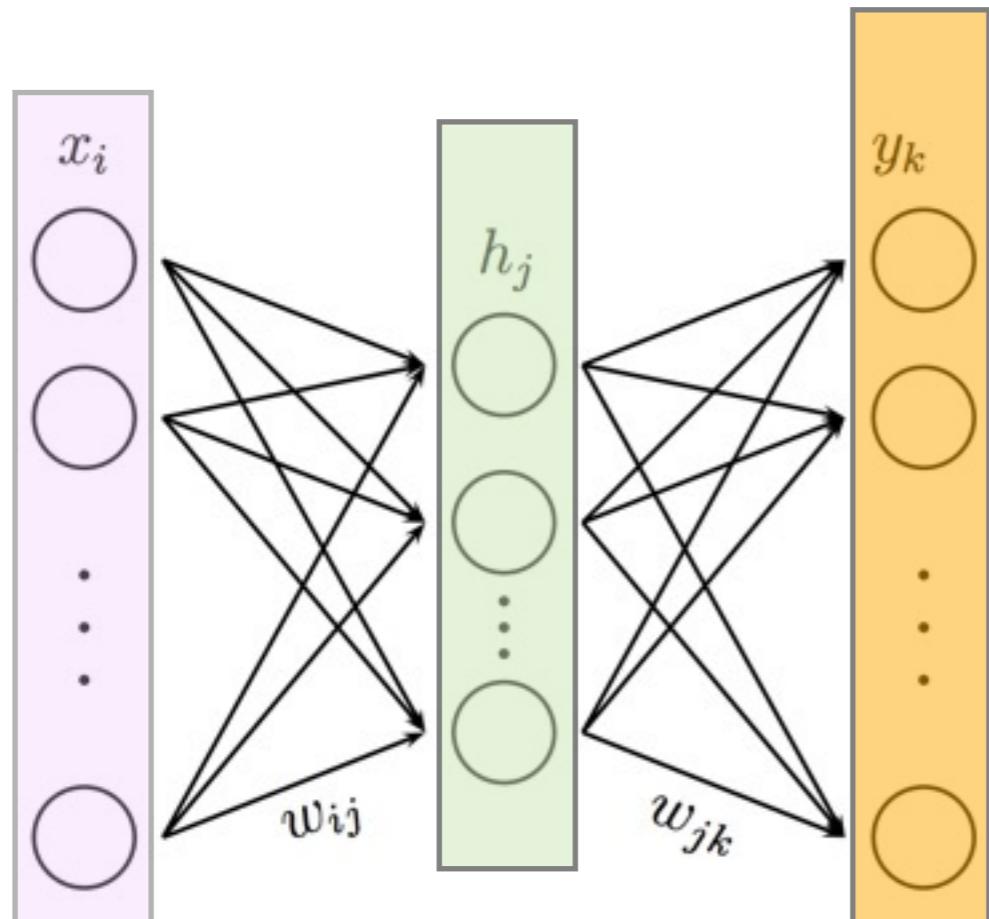


Figure 1. Schematic of a 3-layer feed-forward neural network.

Una red neuronal feed forward se organiza por capas conectadas las unas con las otras.

Están la capa inicial de **inputs** x

Las capas de neuronas (h) que se suelen llamar **capas ocultas**,

La capa de **outputs** (y)

REDES NEURONALES

ARQUITECTURA

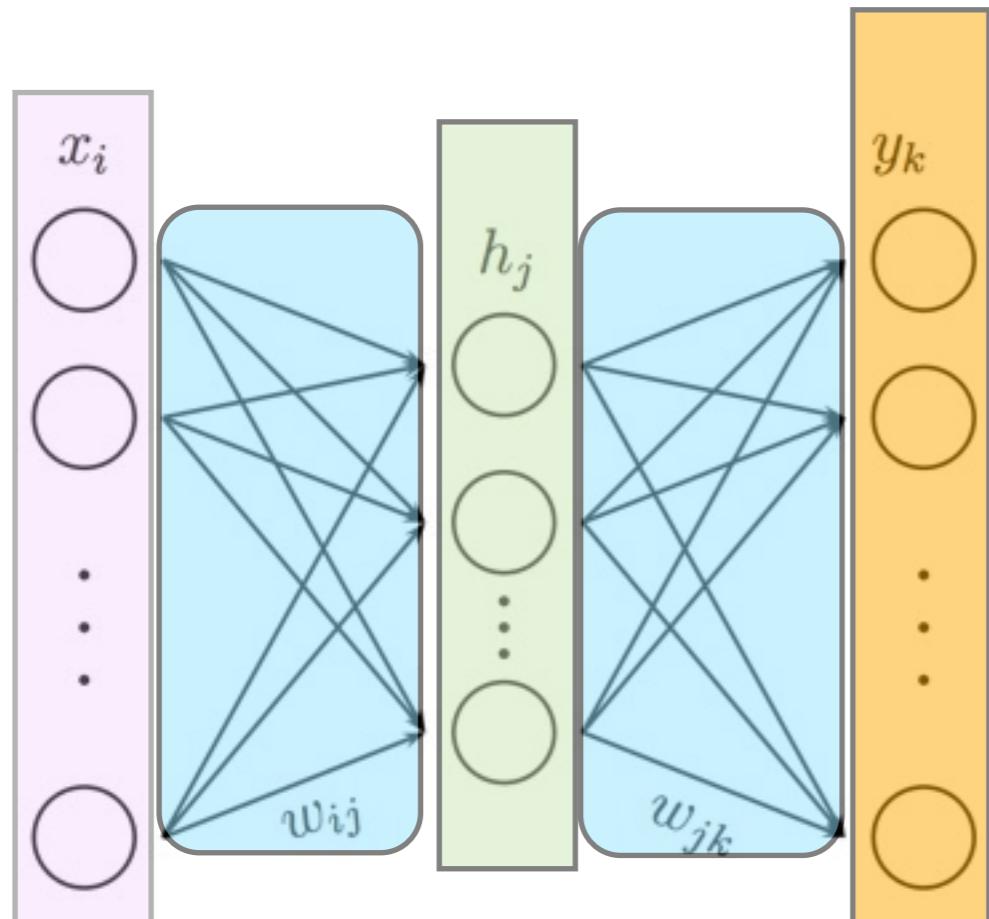


Figure 1. Schematic of a 3-layer feed-forward neural network.

Una red neuronal feed forward se organiza por capas conectadas las unas con las otras.

Están la capa inicial de **inputs** x

Las capas de neuronas (h) que se suelen llamar **capas ocultas**,

La capa de **outputs** (y)

Las conexiones que están representadas por pesos W

REDES NEURONALES

ARQUITECTURA

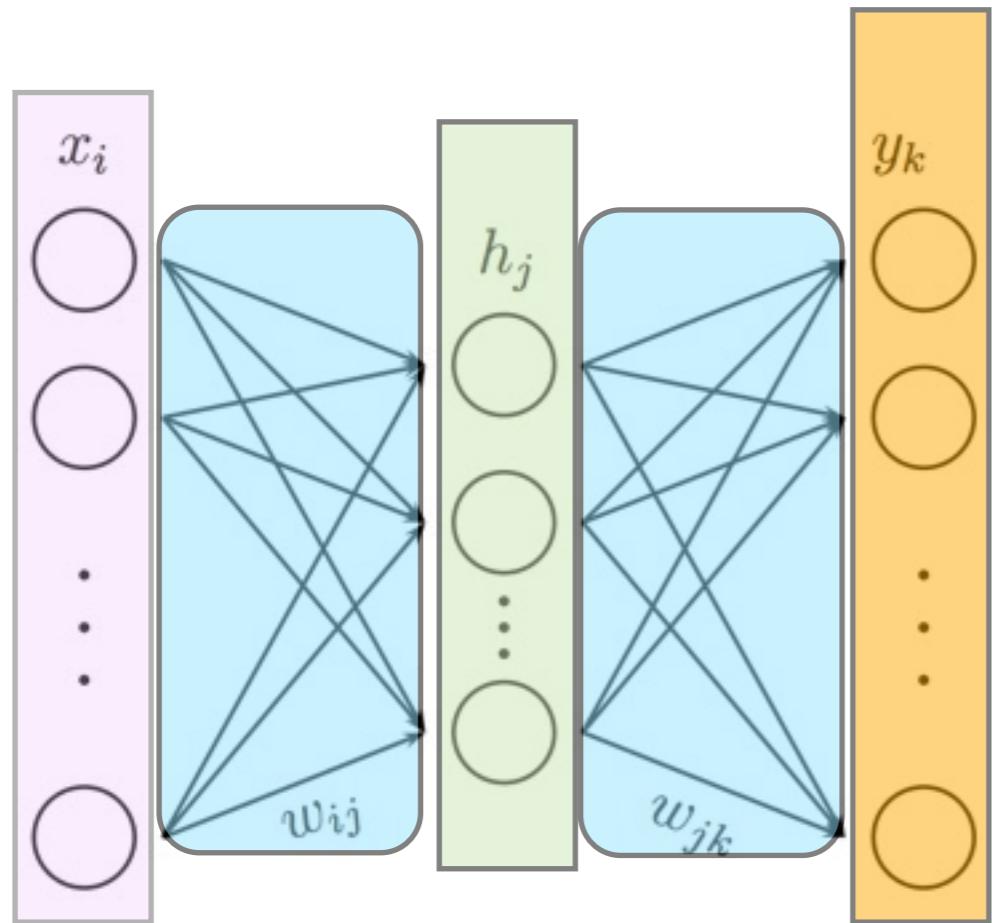


Figure 1. Schematic of a 3-layer feed-forward neural network.

Una red neuronal feed forward se organiza por capas conectadas las unas con las otras.

Están la capa inicial de **inputs** x

Las capas de neuronas (h) que se suelen llamar **capas ocultas**,

La capa de **outputs** (y)

Cada neurona tiene asignada una **función de activación** (excepto en la capa input) y se calcula un nuevo número con las entradas que le llegan.

Las conexiones que están representadas por pesos W

REDES NEURONALES

¿Que son los inputs y los outputs?

Un ejemplo muy sencillo sería intentar construir una red para que sume 2 números . Los inputs serían los sumandos y el output la solución.

Capa inputs: vector de dimension 2
Capa outputs : 1 output
Capa oculta ??

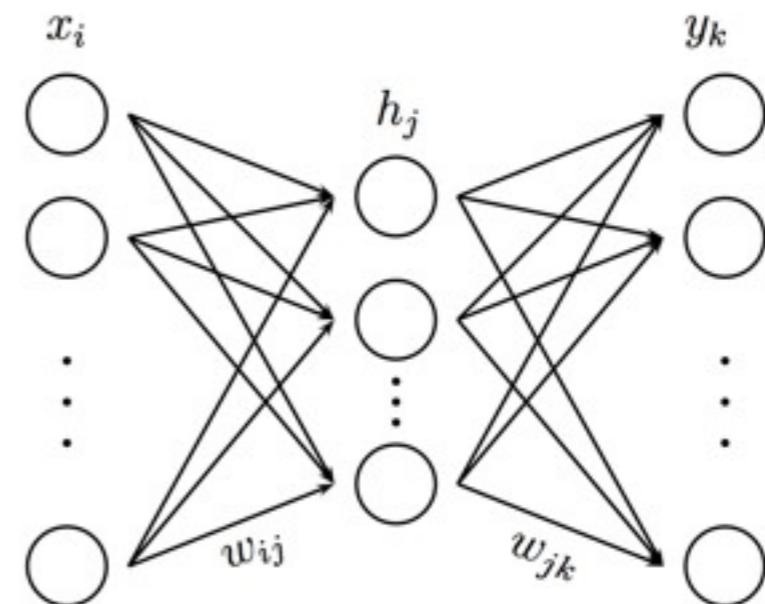


Figure 1. Schematic of a 3-layer feed-forward neural network.

REDES NEURONALES

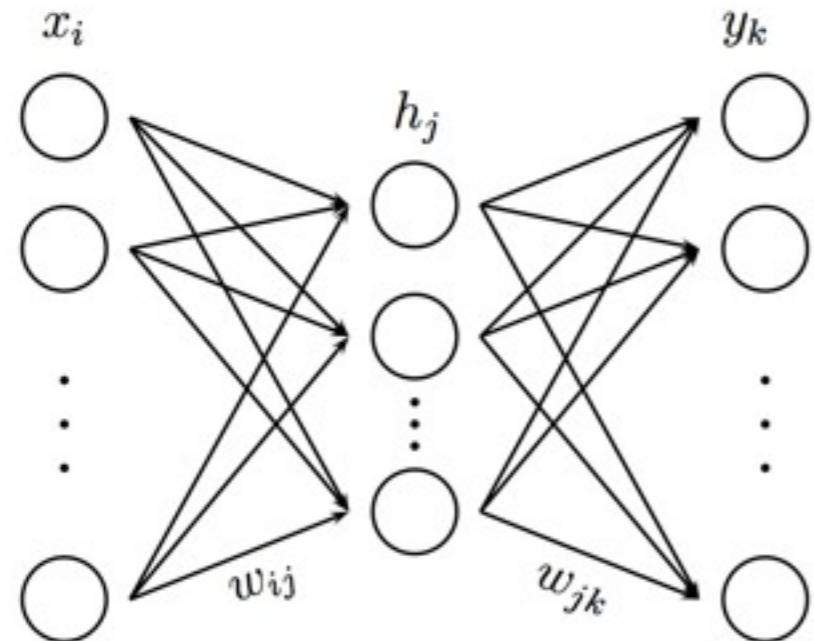


Figure 1. Schematic of a 3-layer feed-forward neural network.

En la capa de los inputs no pasa nada. Están ahí los valores de nuestras variables.

REDES NEURONALES

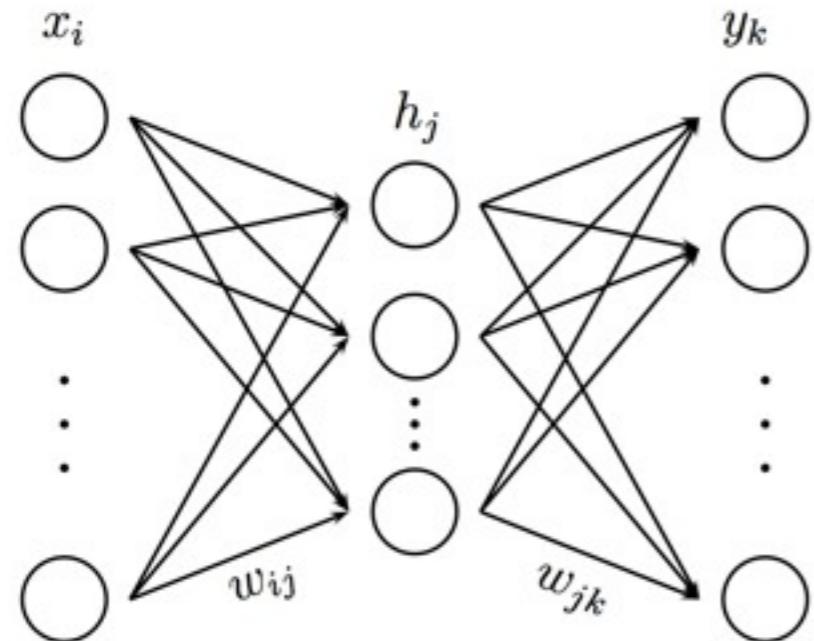


Figure 1. Schematic of a 3-layer feed-forward neural network.

En la capa de los inputs no pasa nada. Están ahí los valores de nuestras variables.

En la siguiente capa, en cada una de las neuronas se hace una operación matemática, combinando los inputs y los pesos de esa conexión:

REDES NEURONALES

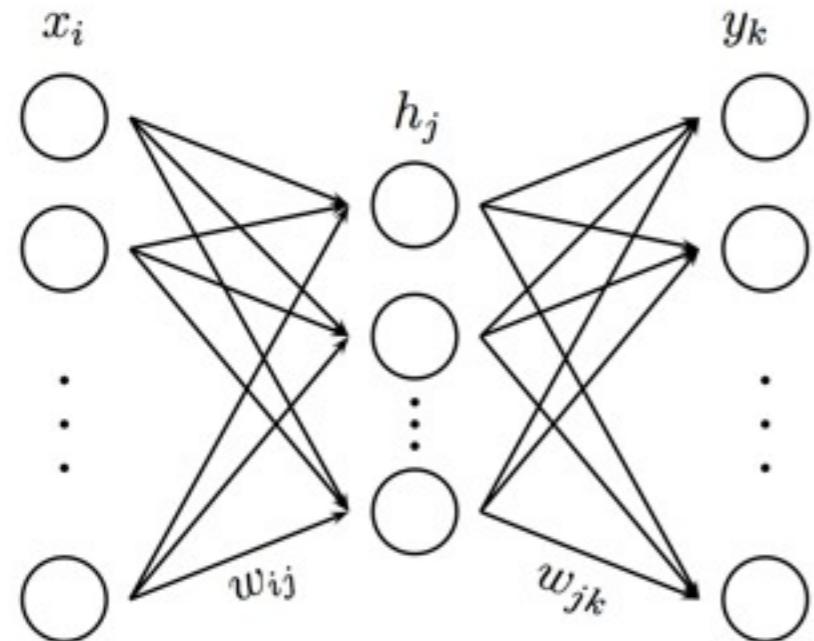


Figure 1. Schematic of a 3-layer feed-forward neural network.

En la capa de los inputs no pasa nada. Están ahí los valores de nuestras variables.

En la siguiente capa, en cada una de las neuronas se hace una operación matemática, combinando los inputs y los pesos de esa conexión:

$$h_1 = f(w_{11}x_1 + w_{21}x_2 + w_{31}x_3)$$

REDES NEURONALES

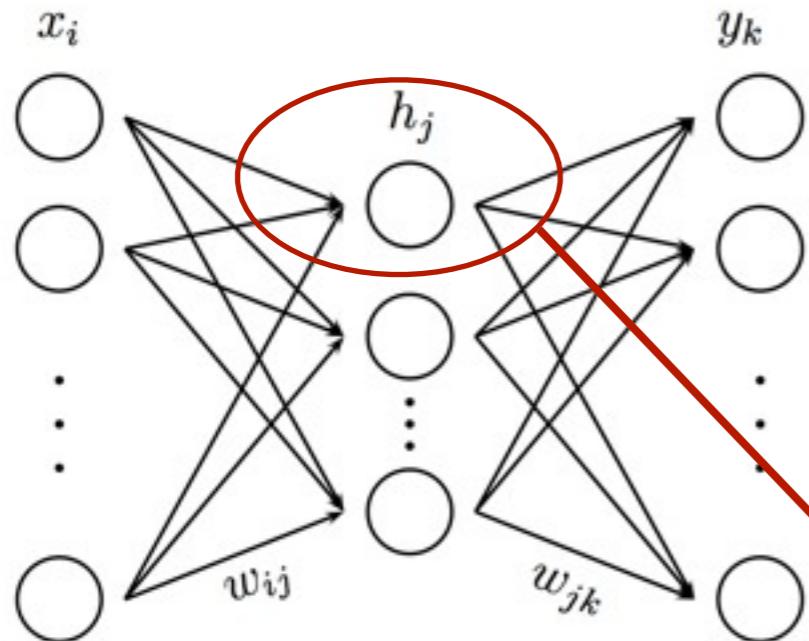
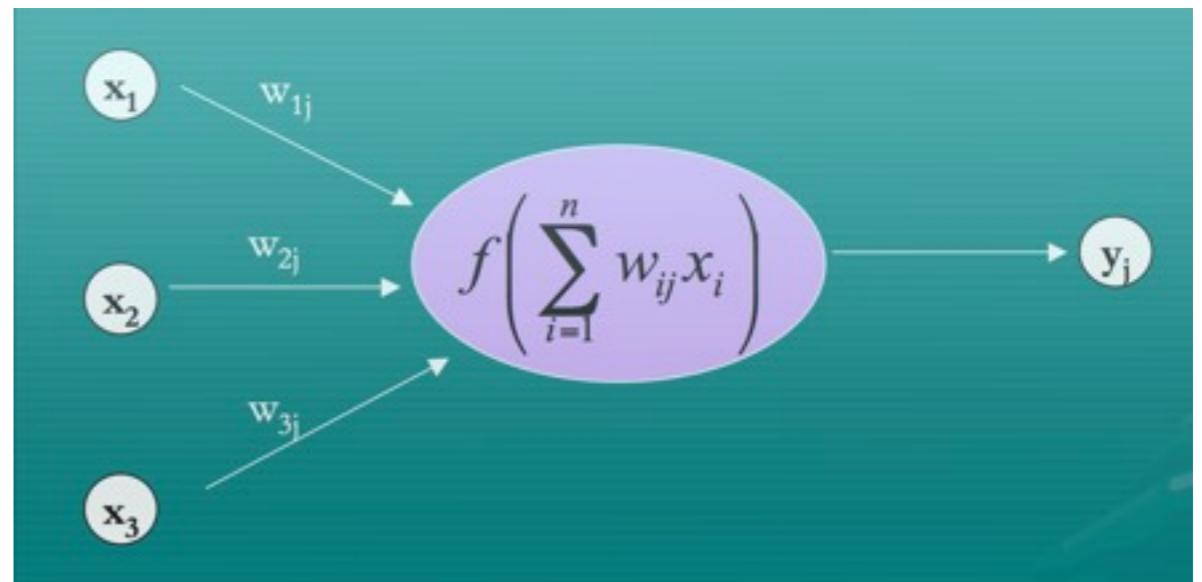


Figure 1. Schematic of a 3-layer feed-forward neural network.

En la capa de los inputs no pasa nada. Están ahí los valores de nuestras variables.

En la siguiente capa, en cada una de las neuronas se hace una operación matemática, combinando los inputs y los pesos de esa conexión:

$$h_1 = f(w_{11}x_1 + w_{21}x_2 + w_{31}x_3)$$



REDES NEURONALES

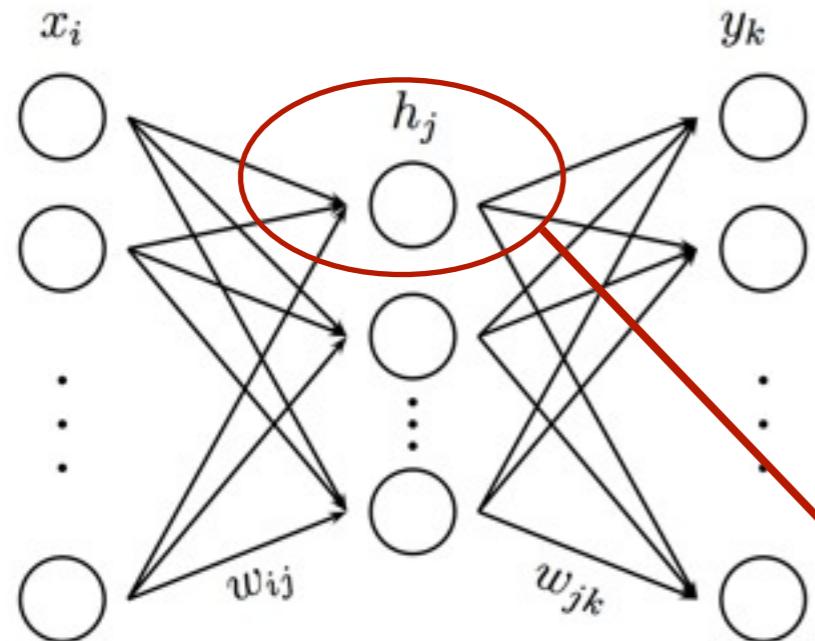


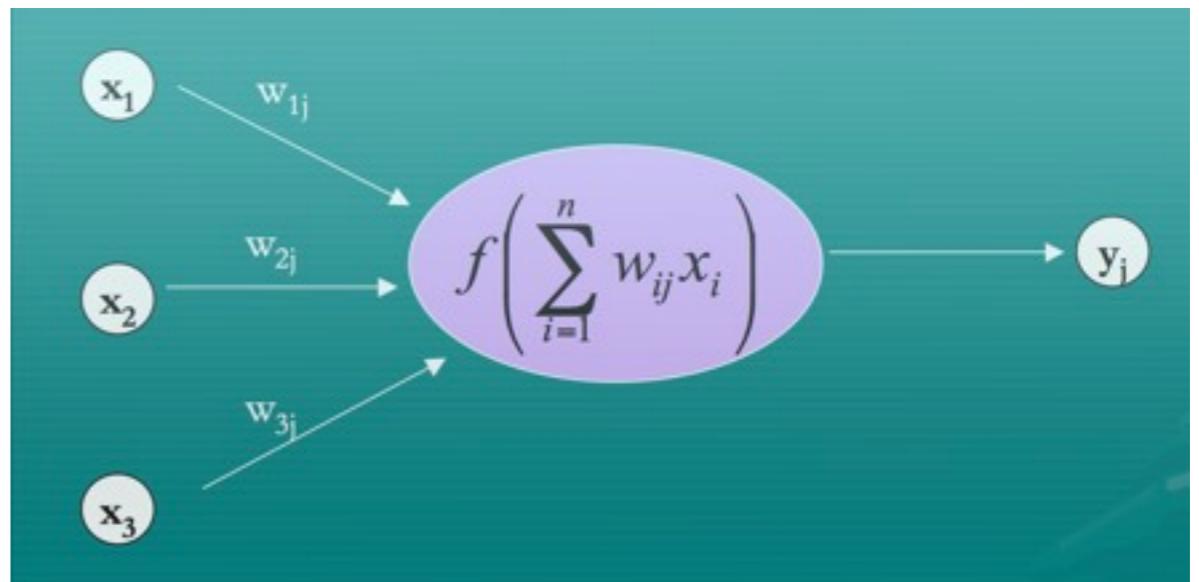
Figure 1. Schematic of a 3-layer feed-forward neural network.

Cara cada neurona de la capa oculta hacemos este cálculo y los h_i serían los valores de entrada para la siguiente capa.

En la capa de los inputs no pasa nada. Están ahí los valores de nuestras variables.

En la siguiente capa, en cada una de las neuronas se hace una operación matemática, combinando los inputs y los pesos de esa conexión:

$$h_1 = f(w_{11}x_1 + w_{21}x_2 + w_{31}x_3)$$



REDES NEURONALES

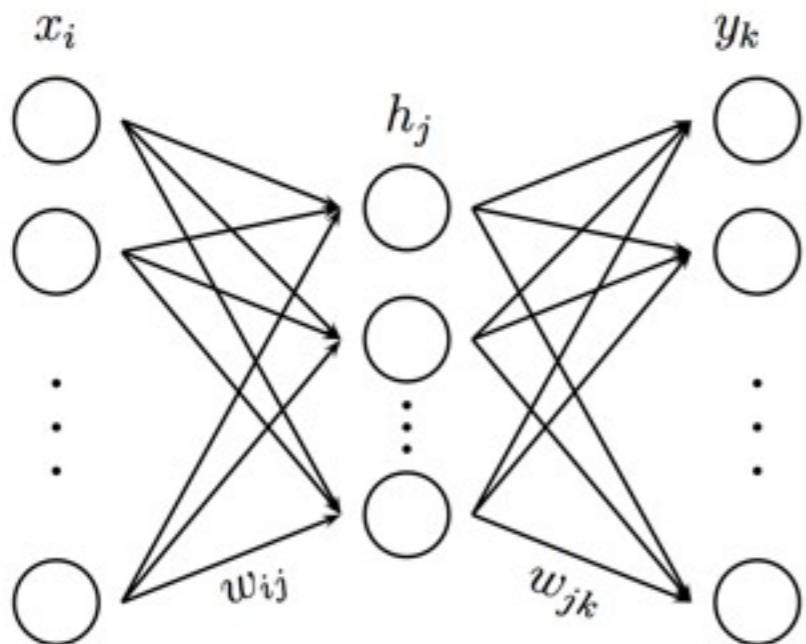


Figure 1. Schematic of a 3-layer feed-forward neural network.

La capa oculta tenemos una serie de valores en cada h

$$h_j = f\left(\sum_i w_{ij}x_i + \theta_j\right)$$

$$y_k = g\left(\sum_j w_{jk}h_j + \theta_k\right)$$

REDES NEURONALES

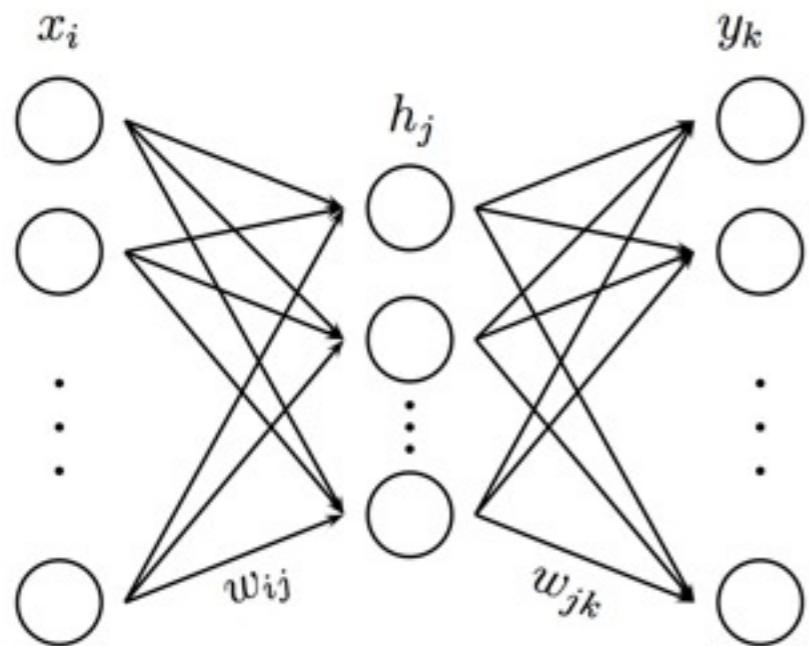


Figure 1. Schematic of a 3-layer feed-forward neural network.

La capa oculta tenemos una serie de valores en cada h

$$h_j = f\left(\sum_i w_{ij}x_i + \theta_j\right)$$

Ahora en la siguiente capa vuelve a pasar lo mismo. La función que aplicamos no tiene porque ser la misma

$$y_k = g\left(\sum_j w_{jk}h_j + \theta_k\right)$$

REDES NEURONALES

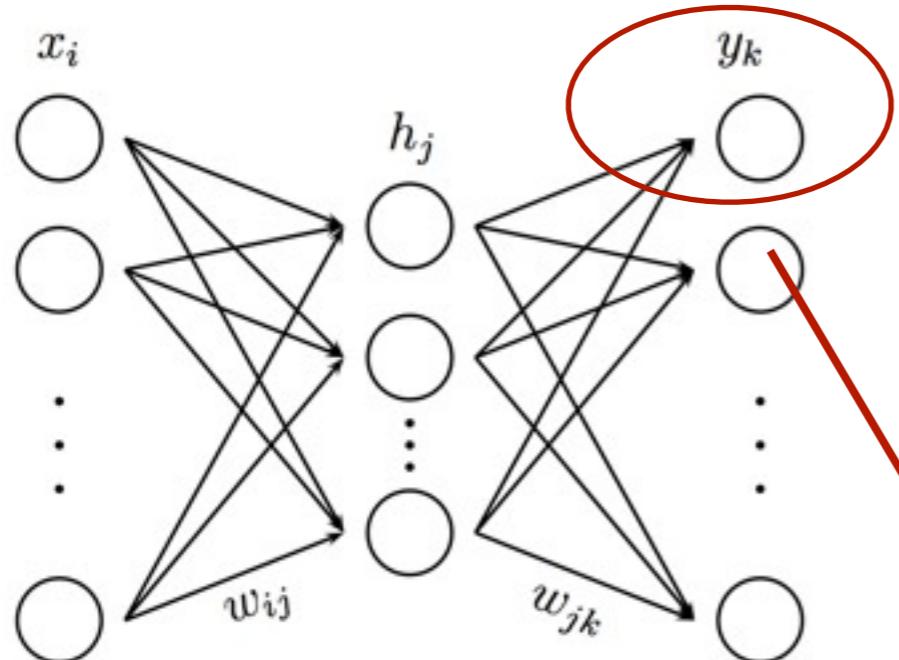


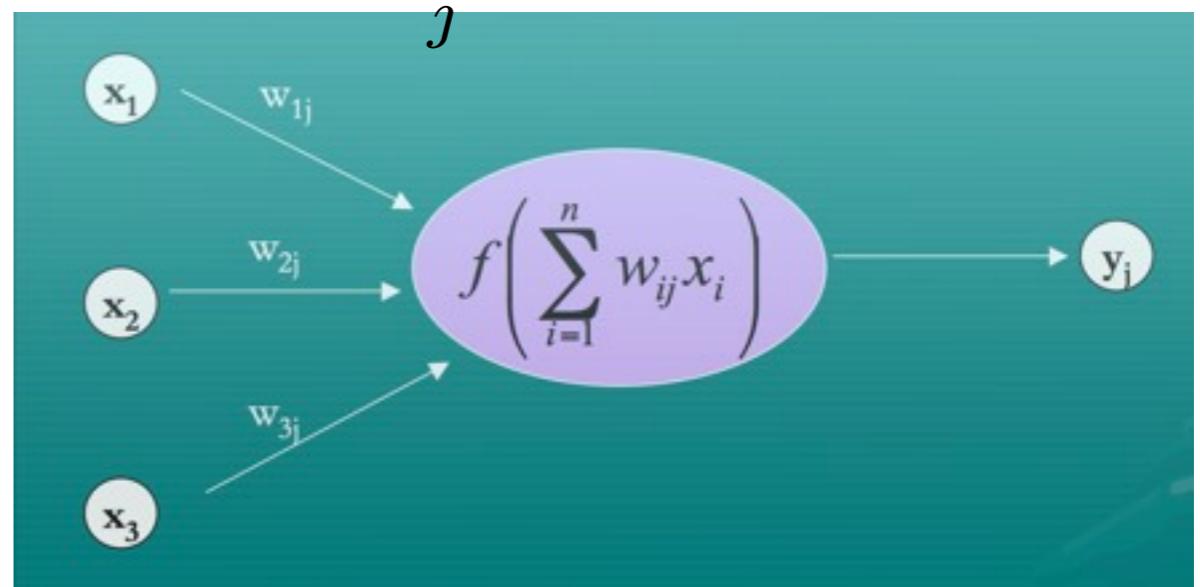
Figure 1. Schematic of a 3-layer feed-forward neural network.

La capa oculta tenemos una serie de valores en cada h

$$h_j = f\left(\sum_i w_{ij}x_i + \theta_j\right)$$

Ahora en la siguiente capa vuelve a pasar lo mismo. La función que aplicamos no tiene porque ser la misma

$$y_k = g\left(\sum_j w_{jk}h_j + \theta_k\right)$$



REDES NEURONALES

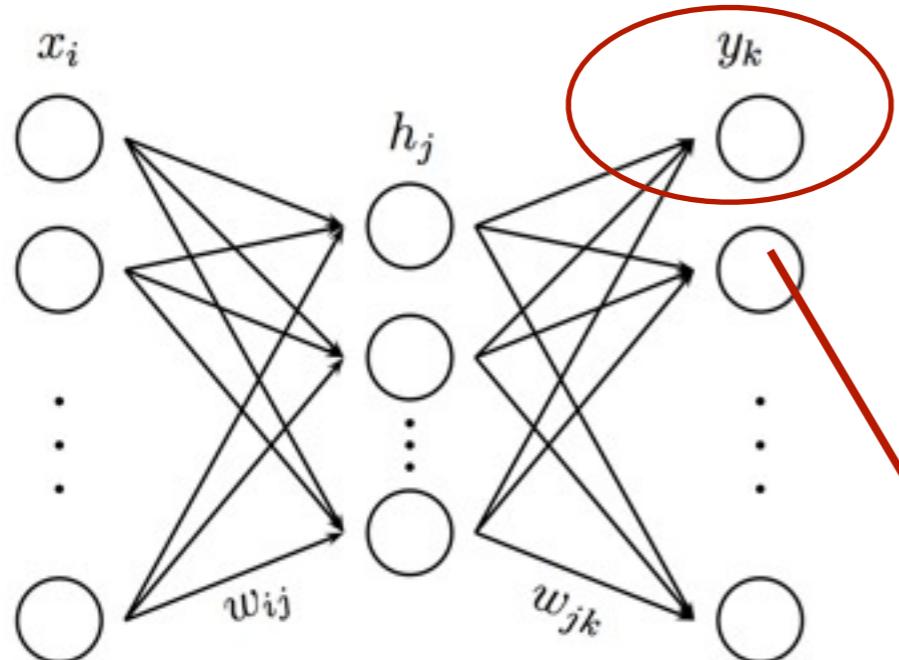


Figure 1. Schematic of a 3-layer feed-forward neural network.

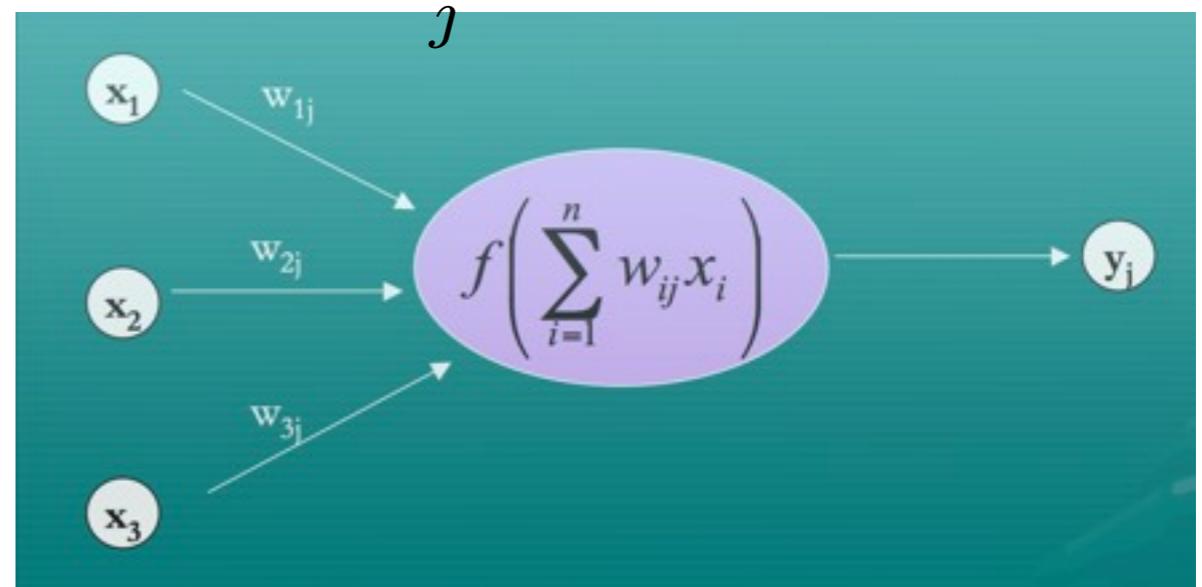
Y así hasta la última capa

La capa oculta tenemos una serie de valores en cada h

$$h_j = f\left(\sum_i w_{ij} x_i + \theta_j\right)$$

Ahora en la siguiente capa vuelve a pasar lo mismo. La función que aplicamos no tiene porque ser la misma

$$y_k = g\left(\sum_j w_{jk} h_j + \theta_k\right)$$



REDES NEURONALES

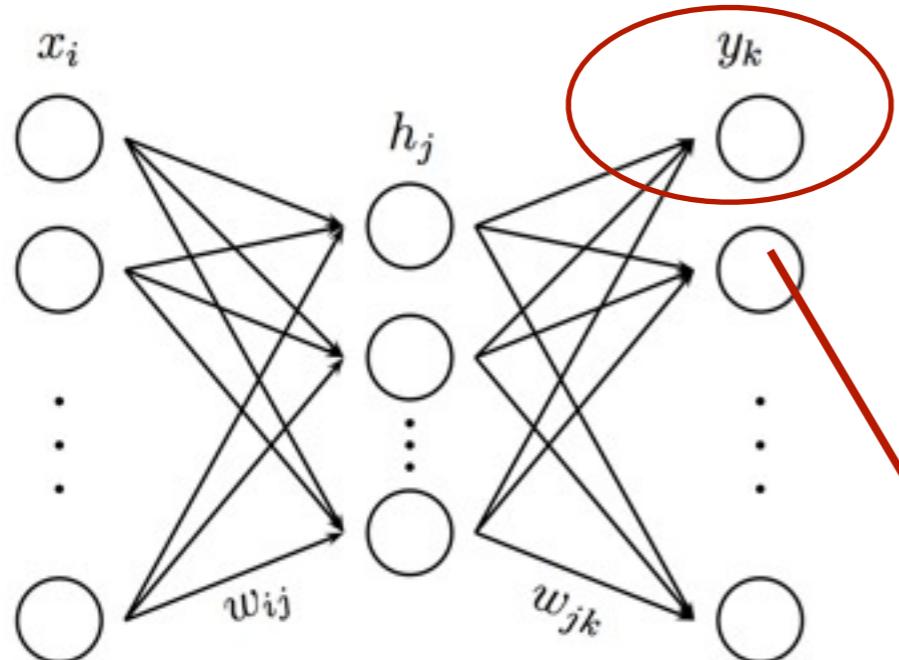


Figure 1. Schematic of a 3-layer feed-forward neural network.

Y así hasta la última capa

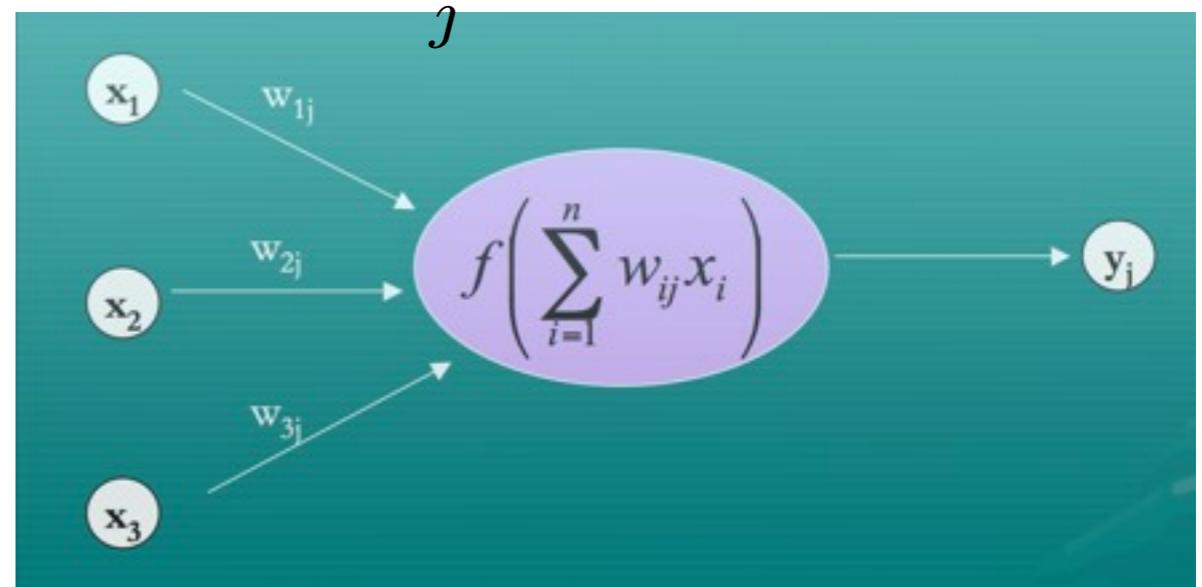
$$y_k = g \left(\left(\sum_j w_{jk} \dots f \left(\sum_i w_{ij} x_i \right) \right) \right)$$

La capa oculta tenemos una serie de valores en cada h

$$h_j = f \left(\sum_i w_{ij} x_i + \theta_j \right)$$

Ahora en la siguiente capa vuelve a pasar lo mismo. La función que aplicamos no tiene porque ser la misma

$$y_k = g \left(\sum_j w_{jk} h_j + \theta_k \right)$$



REDES NEURONALES

**Podemos poner los outputs en función de
los inputs y de los pesos**

Forma general:

$$y_k = g \left(\sum_j w_{jk} \dots f \left(\sum_i w_{ij} x_i \right) \right)$$

REDES NEURONALES

Podemos poner los outputs en función de los inputs y de los pesos

Forma general:

$$y_k = g \left(\sum_j w_{jk} \dots f \left(\sum_i w_{ij} x_i \right) \right)$$

caso particular:

función activación capa oculta : tanh
función activación capa outputs:
lineal

$$y_k = \sum_j w_{jk} \tanh \left(\sum_i w_{ij} x_i \right)$$

REDES NEURONALES

Podemos poner los outputs en función de los inputs y de los pesos

Forma general:

$$y_k = g \left(\left(\sum_j w_{jk} \dots f \left(\sum_i w_{ij} x_i \right) \right) \right)$$

En el fondo estamos construyendo una función (o familia de funciones) para poder sacar un modelo

función activación capa oculta : tanh
función activación capa outputs:
lineal

$$y_k = \sum_j w_{jk} \tanh \left(\sum_i w_{ij} x_i \right)$$

REDES NEURONALES

Set de entrenamiento

Pero ¿como sacamos los pesos (o parametro del modelo)?

Necesitamos unos datos aquí le llamaremos muestra de entrenamiento (o **training set**).

x_1	x_2	t
0	1	1
10	-2	8
4	12	16
5	1	6
...

Ejemplo: Queremos entrenar una red para que aprenda a sumar 2 números! Eso sería una **red de regresión**

Le damos una serie de datos :

2 inputs, 1 output N veces

A los outputs del training set se le suele llamar **label** o **target**

REDES NEURONALES

Set de entrenamiento

x_1	x_2	t
0	1	1
10	-2	8
4	12	16
5	1	6
...

Básicamente hacemos un **ajuste** de esta función:

Usando ajuste por mínimos cuadrados:

REDES NEURONALES

Set de entrenamiento

x_1	x_2	t
0	1	1
10	-2	8
4	12	16
5	1	6
...

Básicamente hacemos un **ajuste** de esta función:

$$y_k = g\left(\left(\sum_j w_{jk} \dots f\left(\sum_i w_{ij} x_i\right)\right)\right)$$

Usando ajuste por mínimos cuadrados:

REDES NEURONALES

Set de entrenamiento

x_1	x_2	t
0	1	1
10	-2	8
4	12	16
5	1	6
...

Básicamente hacemos un **ajuste** de esta función:

$$y_k = g\left(\left(\sum_j w_{jk} \dots f\left(\sum_i w_{ij} x_i\right)\right)\right)$$

a estos datos

Usando ajuste por mínimos cuadrados:

REDES NEURONALES

Set de entrenamiento

x_1	x_2	t
0	1	1
10	-2	8
4	12	16
5	1	6
...

Básicamente hacemos un **ajuste** de esta función:

$$y_k = g\left(\left(\sum_j w_{jk} \dots f\left(\sum_i w_{ij} x_i\right)\right)\right)$$

a estos datos

Usando ajuste por mínimos cuadrados:

$$E = \sum_n \frac{(t^n - y^n)^2}{N}$$

REDES NEURONALES

Set de entrenamiento

x_1	x_2	t
0	1	1
10	-2	8
4	12	16
5	1	6
...

Básicamente hacemos un **ajuste** de esta función:

$$y_k = g\left(\left(\sum_j w_{jk} \dots f\left(\sum_i w_{ij} x_i\right)\right)\right)$$

a estos datos

Usando ajuste por mínimos cuadrados:

$$E = \sum_n \frac{(t^n - y^n)^2}{N}$$

en este caso $k=1$, pero si hay más de un output, la función objetivo general quedaría (K número de outputs, N número set training) :

$$E = \sum_n \sum_k \frac{(t_k^n - y_k^n)^2}{K \times N}$$

REDES NEURONALES

Set de entrenamiento

x_1	x_2	t
0	1	1
10	-2	8
4	12	16
5	1	6
...

Básicamente hacemos un **ajuste** de esta función:

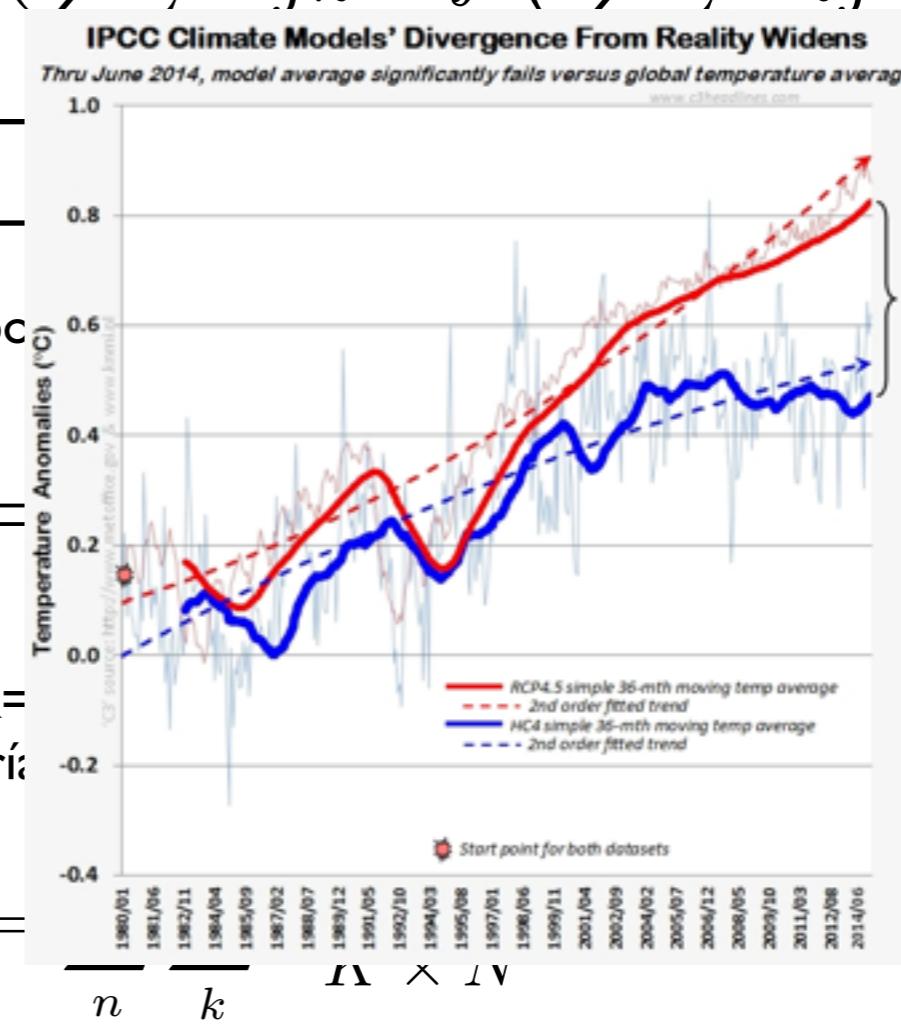
$$y_k = g\left(\left(\sum w_{jk} \dots f\left(\sum w_{ij} x_i\right)\right)\right)$$

a estos datos

Usando ajuste pc

$$E =$$

en este caso $k=$
general quedaría



el objetivo
(training) :

REDES NEURONALES

¿¿Mucha información??

Parece complicado pero es más sencillo de lo que parece y tiene muchísimas aplicaciones. Si el set de **datos es bueno** suelen ser muy eficientes.

BASICAMENTE:

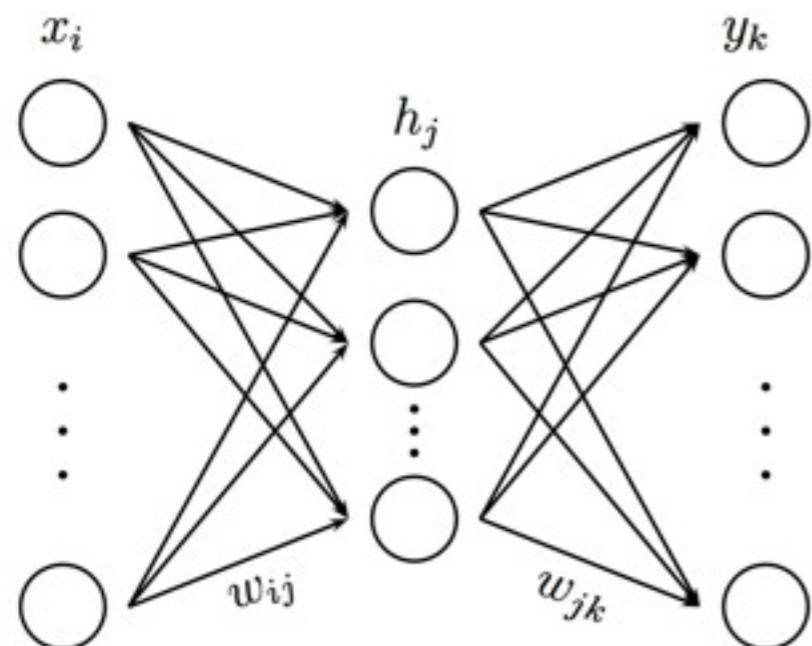


Figure 1. Schematic of a 3-layer feed-forward neural network.

REDES NEURONALES

¿¿Mucha información??

Parece complicado pero es más sencillo de lo que parece y tiene muchísimas aplicaciones. Si el set de **datos es bueno** suelen ser muy eficientes.

BASICAMENTE:

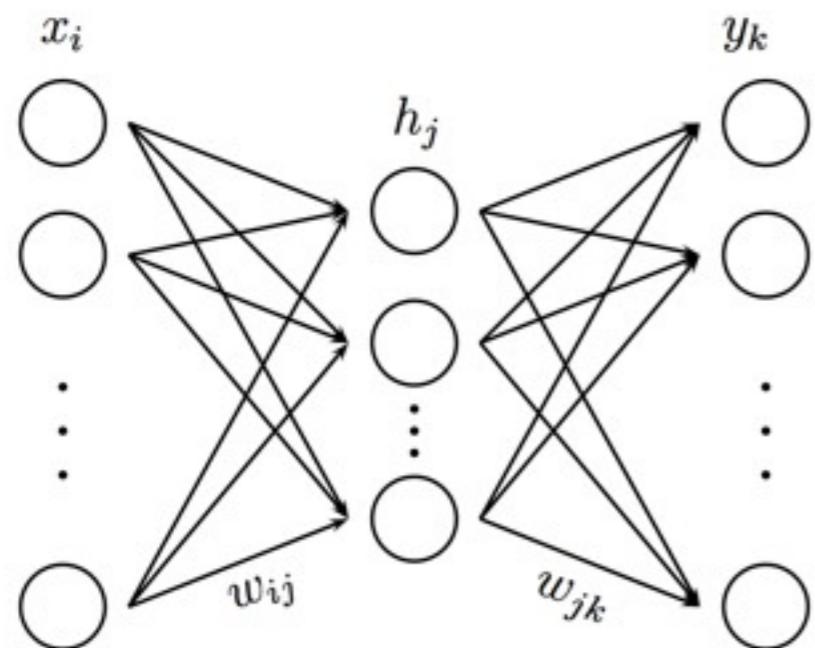


Figure 1. Schematic of a 3-layer feed-forward neural network.

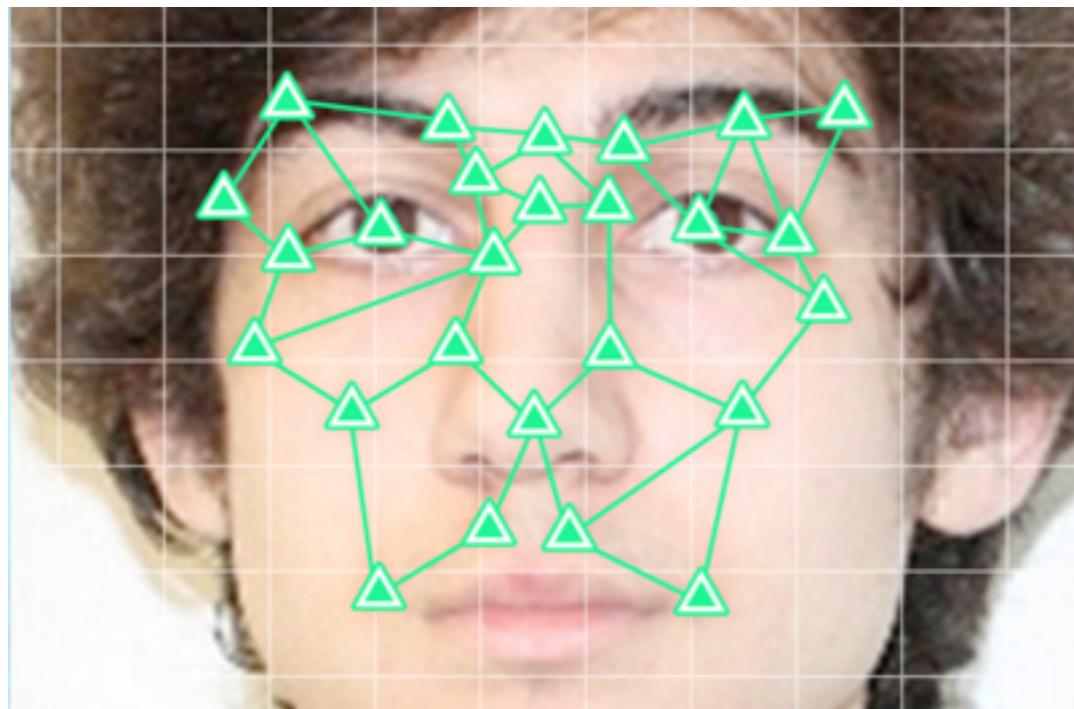
Una red neuronal encuentra una relación entre inputs y outputs.

Una vez la red este entrenada tendremos un **modelo** que nos permite hacer predicciones (si está bien hecho, claro)

* Probablemente solo válido en el rango estudiado

APLICACIONES

Reconocimiento de patrones



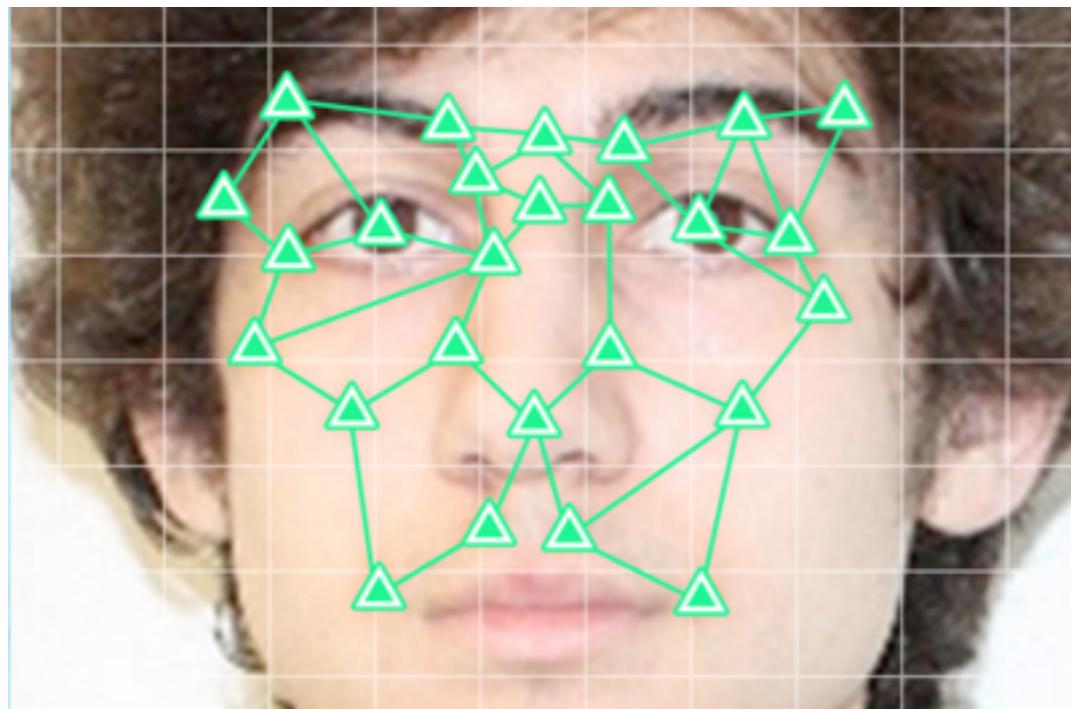
Lo trasladamos al lenguaje
maquina:

But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

APLICACIONES

Reconocimiento de patrones



Si le damos un set de entrenamiento lo suficientemente grande, será capaz de detectar rostros

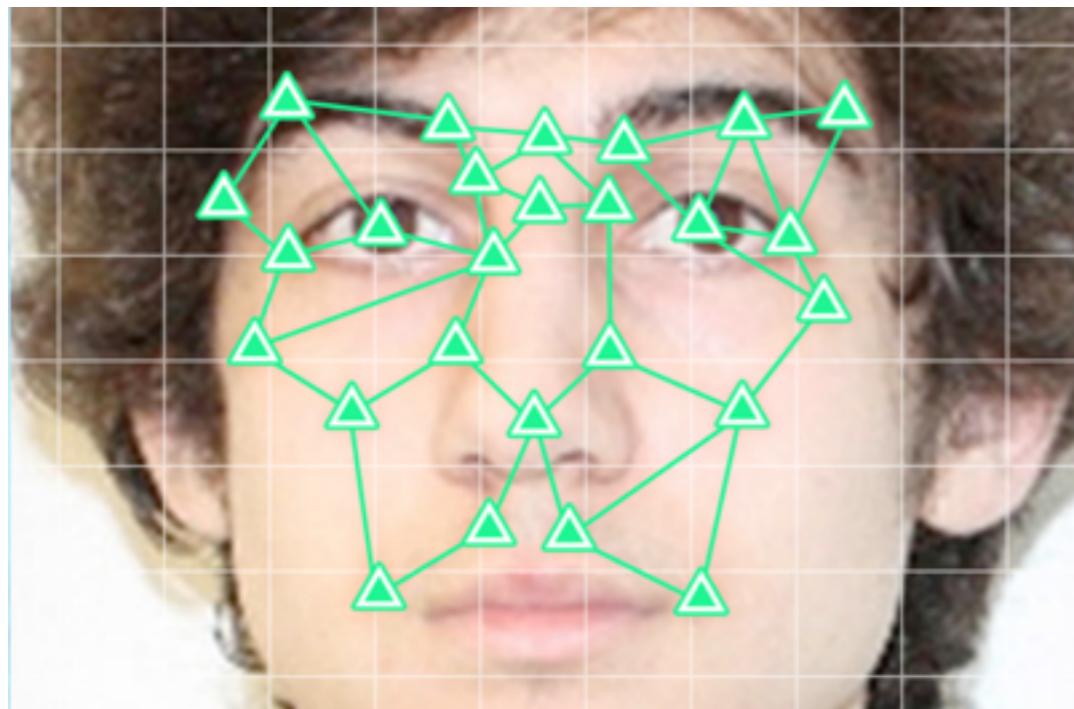
Lo trasladamos al lenguaje maquina:

But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

APLICACIONES

Reconocimiento de patrones



Si le damos un set de entrenamiento lo suficientemente grande, será capaz de detectar rostros

Lo trasladamos al lenguaje maquina:

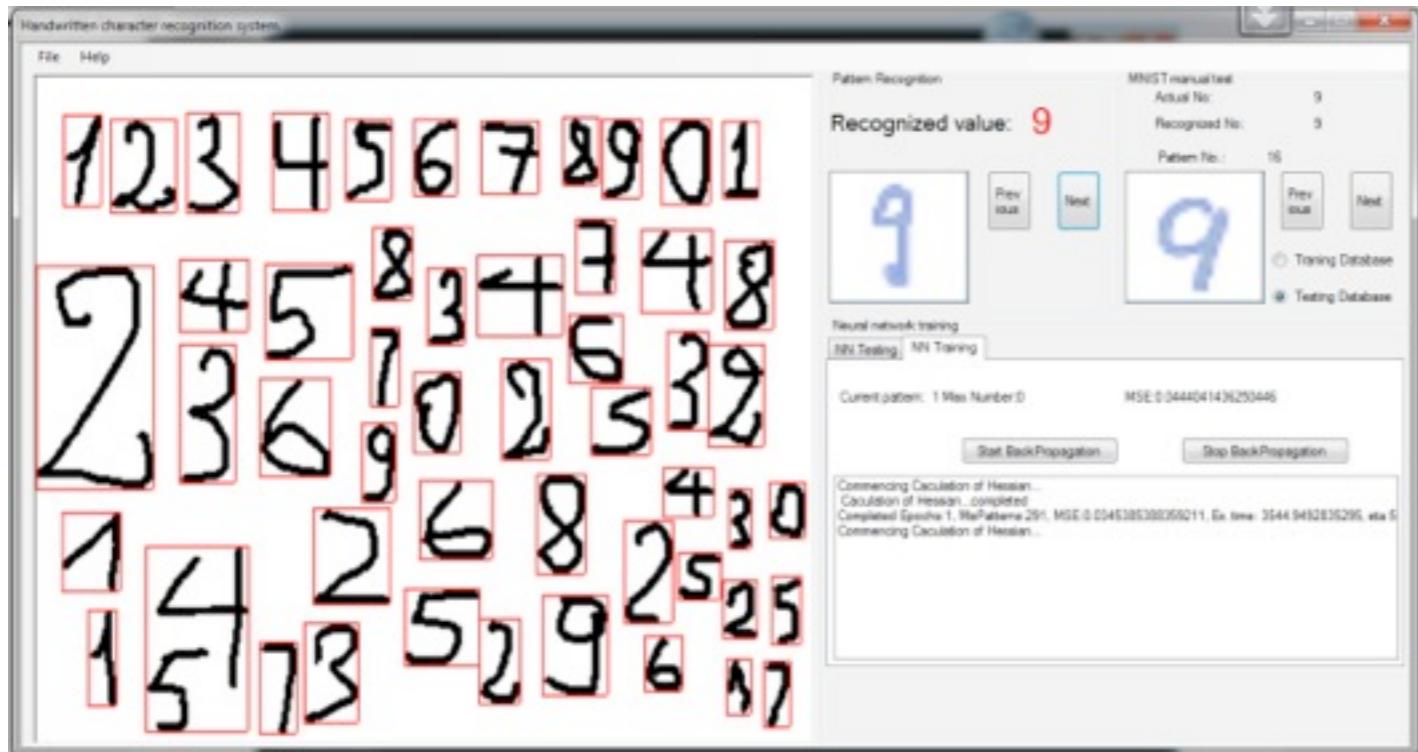
But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

CNN deep learning,
resultados impresionantes!

APLICACIONES

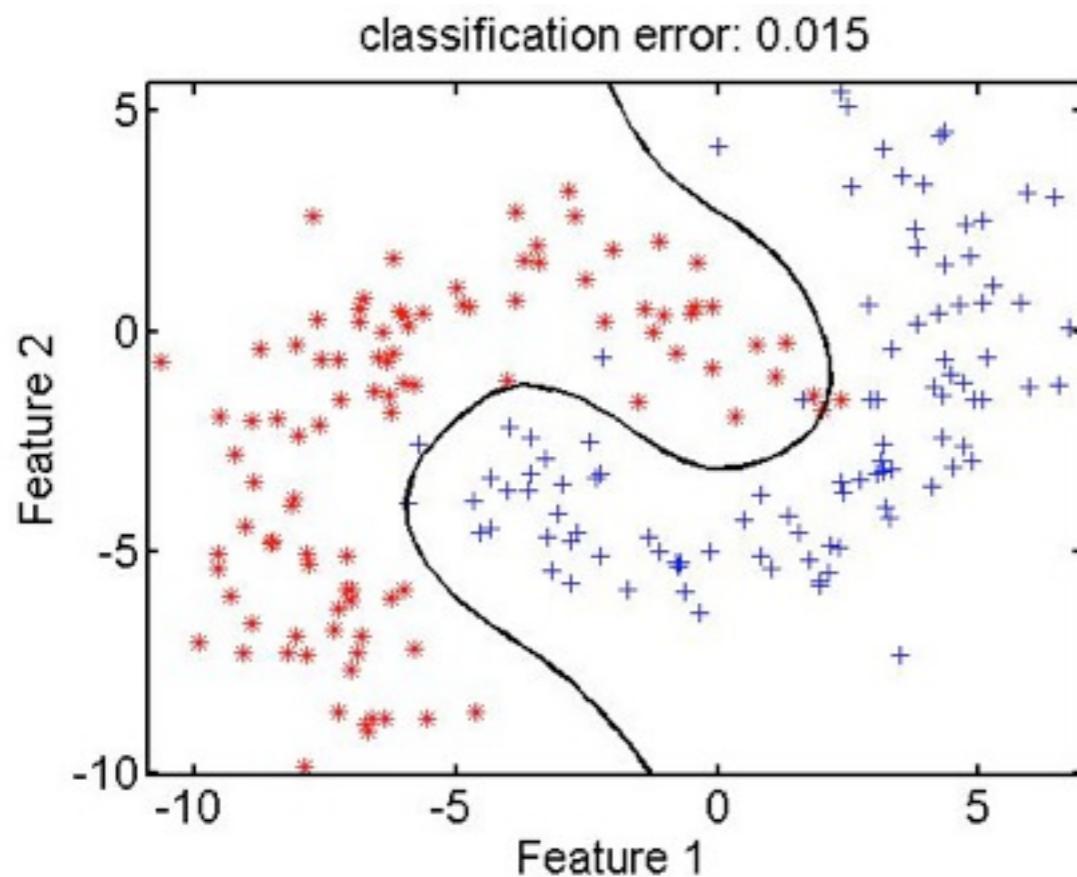
Reconocimiento de patrones



1. Se pasa lenguaje escrito a imágenes de 0 y 1
2. Se entrena la red con muchos ejemplos
3. Una vez entrenada la red es capaz de leer algo escrito a mano
(o una matrícula de coche, ...)

APLICACIONES

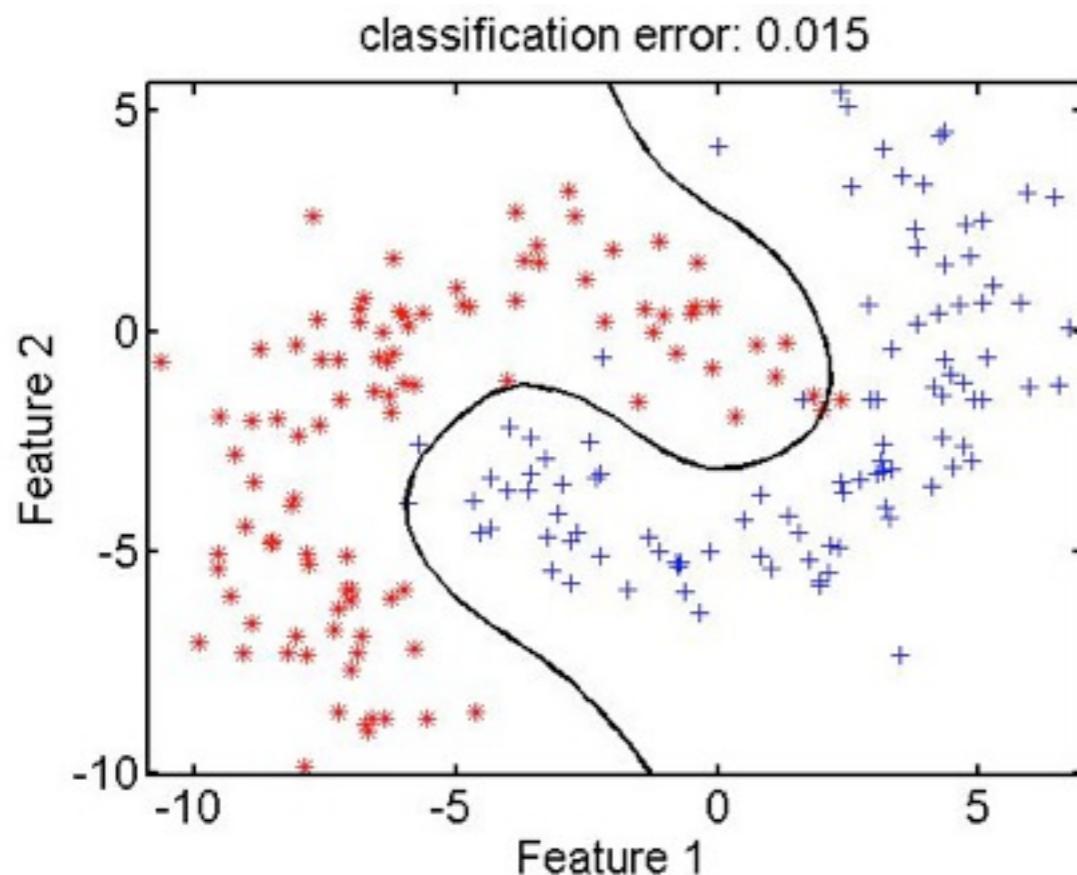
Clasificación



- Es capaz de separar objetos en distintas clases
- Muy útil para problemas de muchas dimensiones y altamente no lineal

APLICACIONES

Clasificación



- Es capaz de separar objetos en distintas clases
- Muy útil para problemas de muchas dimensiones y altamente no lineal

[World Congress on Medical Physics and Biomedical Engineering 2018](#) pp 809-813 | Cite as

Baby Cry Recognition Using Deep Neural Networks

APLICACIONES

Clasificación

1. Graban sonido.
2. Lo pasan a serie temporal $f(t)$
3. Lo clasifican (saben en ese caso lo que le pasa el niño)
(clase 1, Hambre, clase 2 Dolor, clase 3 sueño,
clase 4 mamá...)
4. Entrenan la red
5. Grabas a tu hijo y la red te dirá que le pasa (!!) (en realidad te dice probabilidad de
que le pase 1, 2, 3 o 4,...)



APLICACIONES

Clasificación

[World Congress on Medical Physics and Biomedical Engineering 2018](#) pp 809-813 | [Cite as](#)

Baby Cry Recognition Using Deep Neural Networks

1. Graban sonido.
2. Lo pasan a serie temporal $f(t)$
3. Lo clasifican (saben en ese caso lo que le pasa el niño)
(clase 1, Hambre, clase 2 Dolor, clase 3 sueño,
clase 4 mamá...)
4. Entrenan la red
5. Grabas a tu hijo y la red te dirá que le pasa (!!) (en realidad te dice probabilidad de
que le pase 1, 2, 3 o 4,...)



APLICACIONES

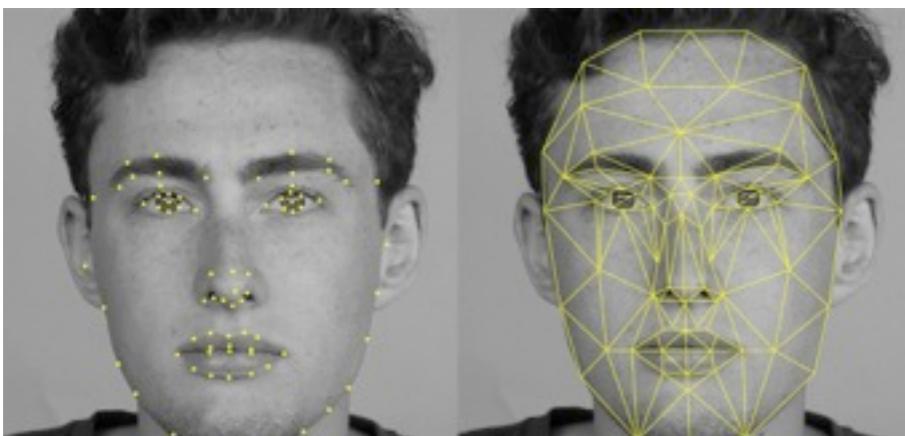
Otras

Reconocimiento de etiquetas, texto,...

Reconocimiento imágenes. Una máquina lee una imagen como la intensidad en cada pixel.
Eso se puede dar como input a una red neuronal



Identificación de plantas, animales



Clasificación de Galaxias



Reconocimiento facial

APLICACIONES

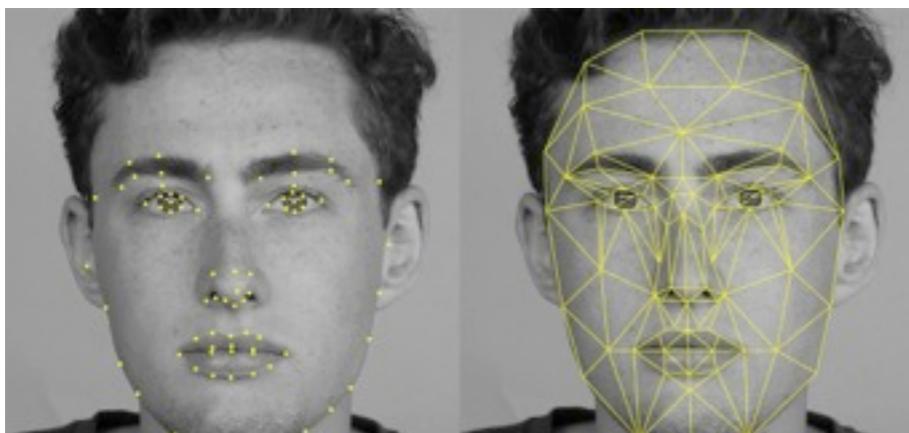
Otras

Reconocimiento de etiquetas, texto,...

Reconocimiento imágenes. Una máquina lee una imagen como la intensidad en cada pixel.
Eso se puede dar como input a una red neuronal



Identificación de plantas, animales



Clasificación de Galaxias



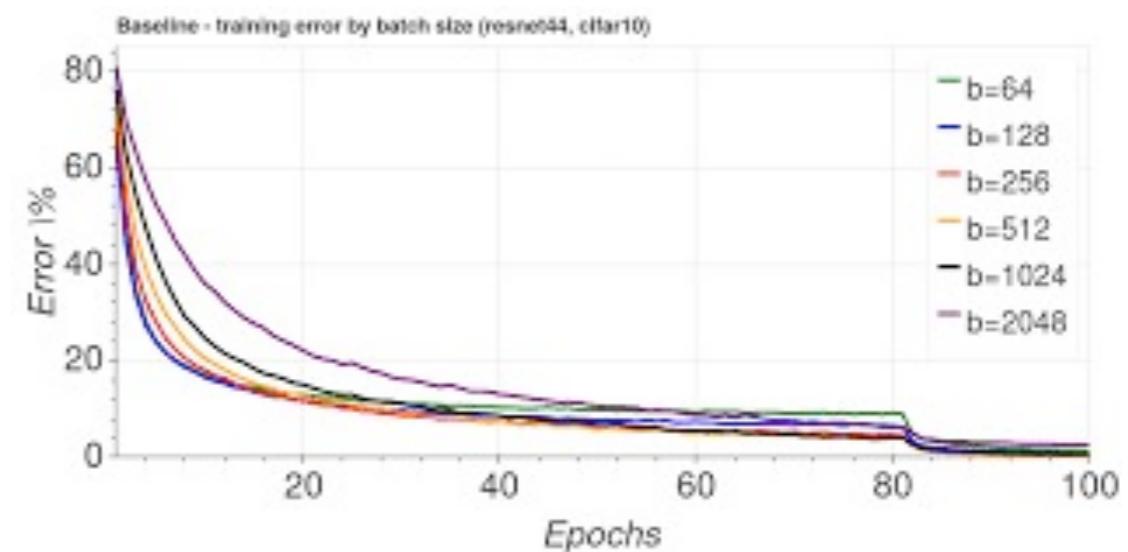
DEEP LEARNING

Reconocimiento facial

REDES NEURONALES

ENTRENAMIENTO

Proceso donde minimizamos la función para encontrar los pesos. Pero se suele hacer a trozos:

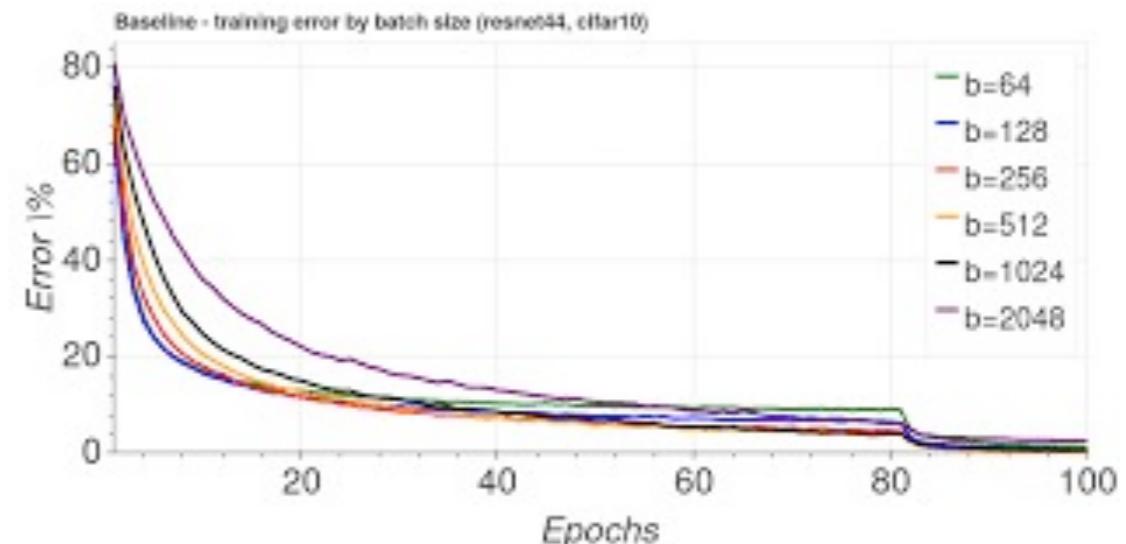


REDES NEURONALES

ENTRENAMIENTO

Proceso donde minimizamos la función para encontrar los pesos. Pero se suele hacer a trozos:

- I. Empezamos el entrenamiento con en un punto inicial de pesos (aleatorio) con un pequeño subgrupo del set (ej. **batch size=50**).

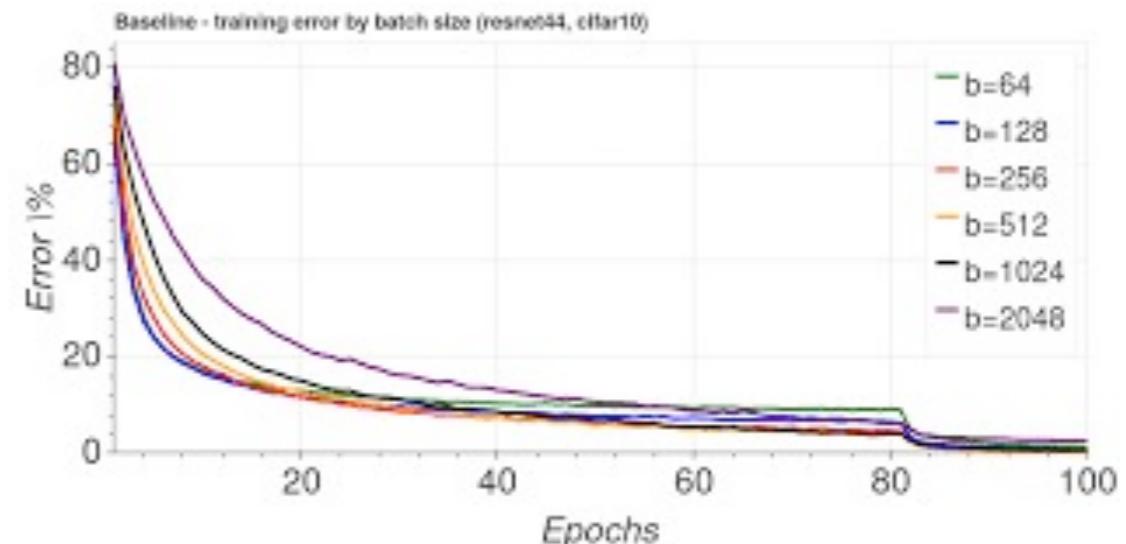


REDES NEURONALES

ENTRENAMIENTO

Proceso donde minimizamos la función para encontrar los pesos. Pero se suele hacer a trozos:

1. Empezamos el entrenamiento con en un punto inicial de pesos (aleatorio) con un pequeño subgrupo del set (ej. **batch size=50**).
2. Hacemos el ajuste a los primeros 50 datos, con esos pesos como punto inicial hacemos el ajuste a los 50 siguientes. A esto le llamamos **iteración**.

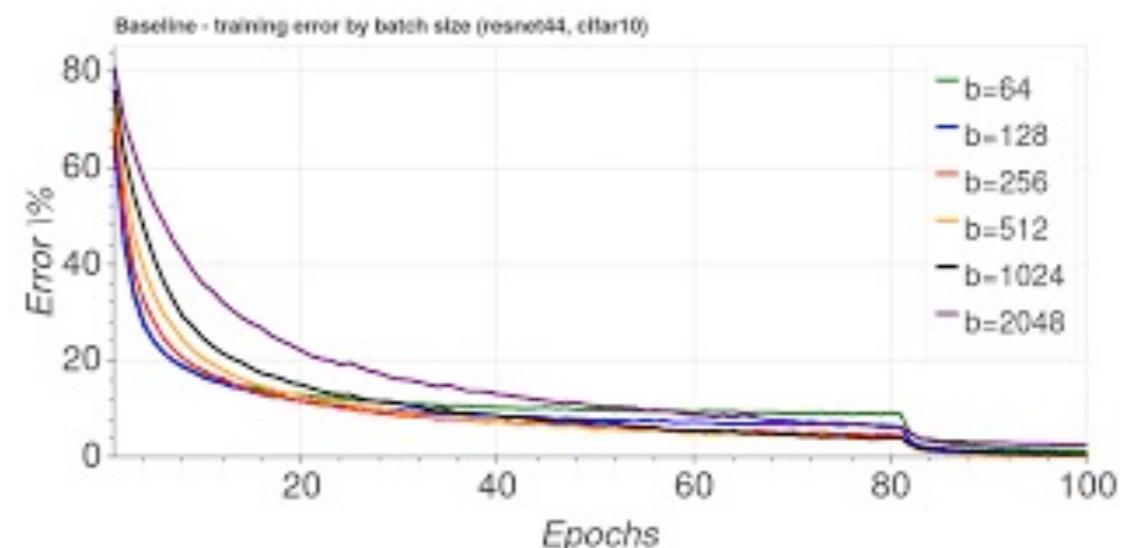


REDES NEURONALES

ENTRENAMIENTO

Proceso donde minimizamos la función para encontrar los pesos. Pero se suele hacer a trozos:

1. Empezamos el entrenamiento con en un punto inicial de pesos (aleatorio) con un pequeño subgrupo del set (ej. **batch size=50**).
2. Hacemos el ajuste a los primeros 50 datos, con esos pesos como punto inicial hacemos el ajuste a los 50 siguientes. A esto le llamamos **iteración**.
3. Esto lo hacemos hasta que hayamos pasado por todo el set de entrenamiento. A la vuelta entera se le llama **época**.

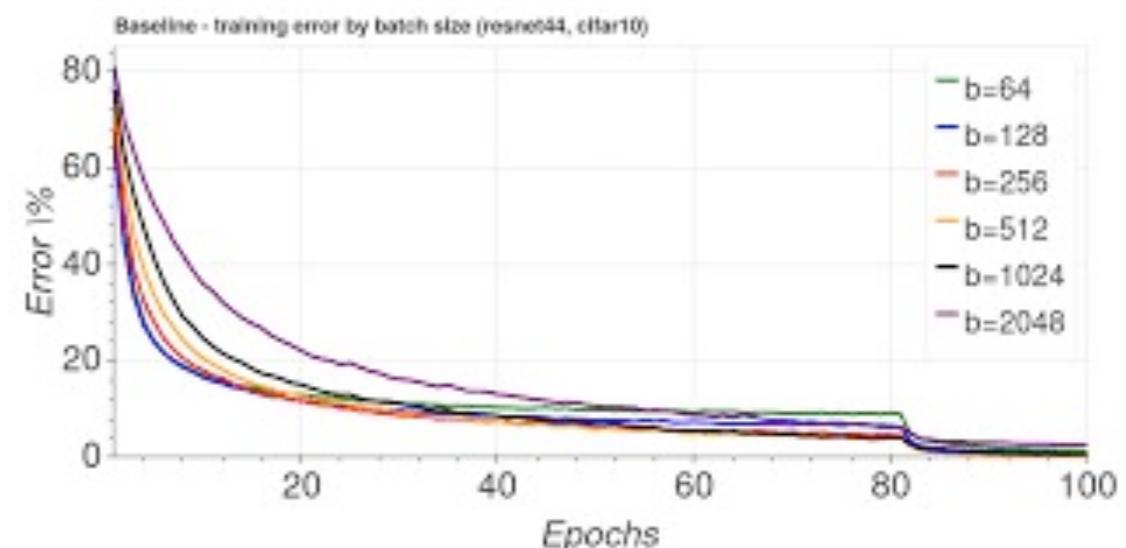


REDES NEURONALES

ENTRENAMIENTO

Proceso donde minimizamos la función para encontrar los pesos. Pero se suele hacer a trozos:

1. Empezamos el entrenamiento con en un punto inicial de pesos (aleatorio) con un pequeño subgrupo del set (ej. **batch size=50**).
2. Hacemos el ajuste a los primeros 50 datos, con esos pesos como punto inicial hacemos el ajuste a los 50 siguientes. A esto le llamamos **iteración**.
3. Esto lo hacemos hasta que hayamos pasado por todo el set de entrenamiento. A la vuelta entera se le llama **época**.
4. Volvemos a empezar y repetimos tantas veces como épocas pongamos o hasta que converja.



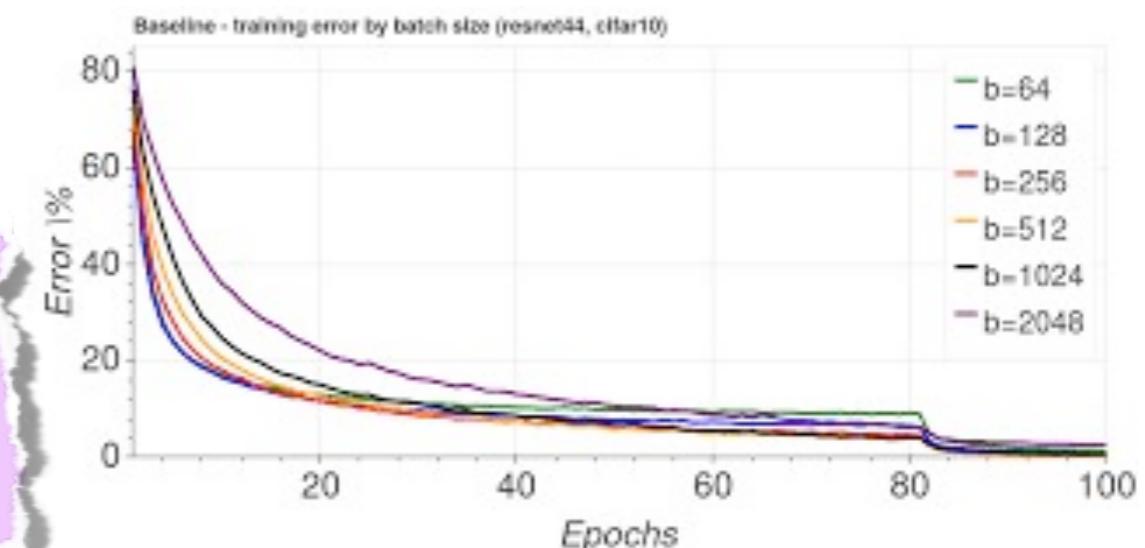
REDES NEURONALES

ENTRENAMIENTO

Proceso donde minimizamos la función para encontrar los pesos. Pero se suele hacer a trozos:

1. Empezamos el entrenamiento con en un punto inicial de pesos (aleatorio) con un pequeño subgrupo del set (ej. **batch size=50**).
2. Hacemos el ajuste a los primeros 50 datos, con esos pesos como punto inicial hacemos el ajuste a los 50 siguientes. A esto le llamamos **iteración**.
3. Esto lo hacemos hasta que hayamos pasado por todo el set de entrenamiento. A la vuelta entera se le llama **época**.
4. Volvemos a empezar y repetimos tantas veces como épocas pongamos o hasta que converja.

Este proceso nos permite ir **evaluando** el ajuste durante el entrenamiento y optimizar **memoria** usada.



REDES NEURONALES

ARQUITECTURA

¿Cuántas capas ocultas hay que poner? Y cuantas neuronas en cada capa?

No hay muchas pistas, hay que probar bastantes configuraciones.

Pista 1. Universal approximation theorem (Cybenko 1989): Una red neuronal feed-forward de una **sola capa** con un número finito de neuronas puede aproximar cualquier función continua real.

Pista 2. Pocas neuronas pueden hacer que no lleguemos a trazar bien el problema y demasiadas puede ser que sobreajustemos (overfitting). Un número para empezar $(N_{in} + N_{out})/2$, **pero no es una regla fiable!!**

¡Prueba y error!

REDES NEURONALES

ENTRENAMIENTO

Una vez decidido todo: función de activación, loss function, training set
ENTRENAMOS la red

¿Cómo sabemos que lo está haciendo bien?

- Vamos viendo el resultado con algún estadístico:

REDES NEURONALES

ENTRENAMIENTO

Una vez decidido todo: función de activación, loss function, training set
ENTRENAMOS la red

¿Cómo sabemos que lo está haciendo bien?

- Vamos viendo el resultado con algún estadístico:

Regresión:

$$E = \sum_n \frac{(t^n - y^n)^2}{N}$$

$$\rho = \frac{\sigma_{ty}}{\sigma_t \sigma_y}$$

REDES NEURONALES

ENTRENAMIENTO

Una vez decidido todo: función de activación, loss function, training set
ENTRENAMOS la red

¿Cómo sabemos que lo está haciendo bien?

- Vamos viendo el resultado con algún estadístico:

Regresión:

$$E = \sum_n \frac{(t^n - y^n)^2}{N}$$

$$\rho = \frac{\sigma_{ty}}{\sigma_t \sigma_y}$$

$$F1 = 2(precision * recall) / (precision + recall)$$

REDES NEURONALES

ENTRENAMIENTO

Una vez decidido todo: función de activación, loss function, training set
ENTRENAMOS la red

¿Cómo sabemos que lo está haciendo bien?

- Vamos viendo el resultado con algún estadístico:

Clasificación

Precisión (o TPR)

$$TPR = \frac{TP}{TP+FP}$$

Compleitud (o recall)

$$Rec = \frac{TP}{TP+FN}$$

Regresión:

$$E = \sum_n \frac{(t^n - y^n)^2}{N}$$

$$\rho = \frac{\sigma_{ty}}{\sigma_t \sigma_y}$$

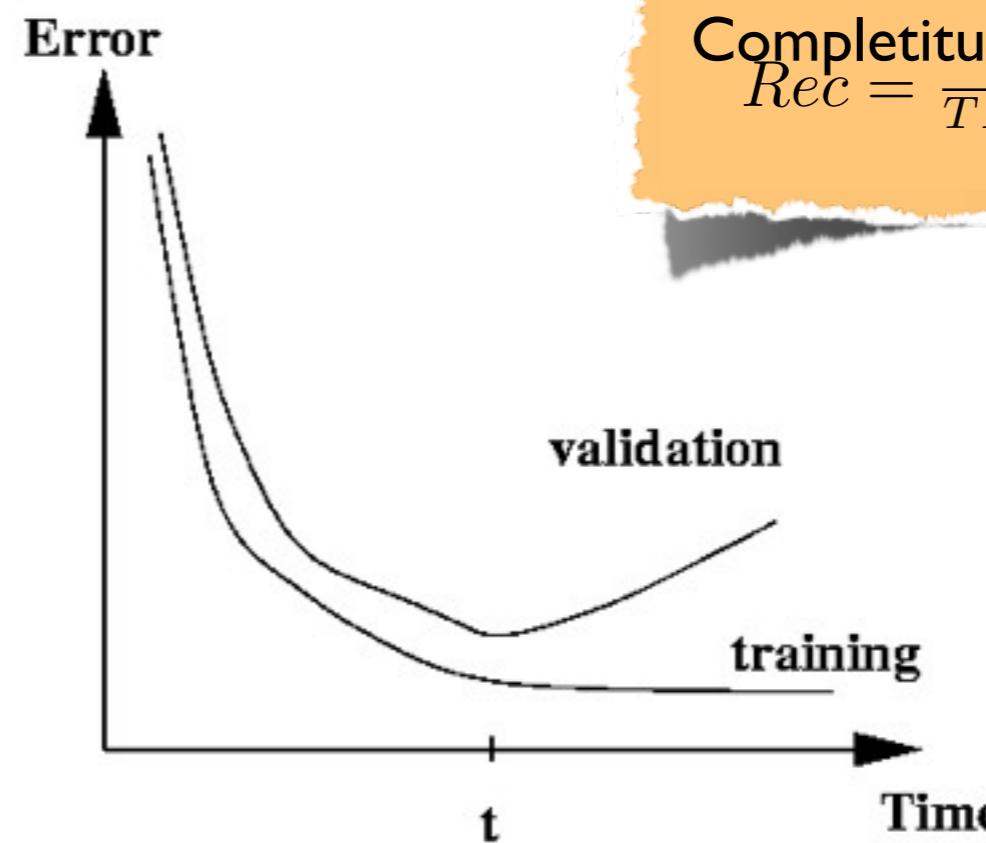
$$F1 = 2(precision * recall) / (precision + recall)$$

REDES NEURONALES

ENTRENAMIENTO

y ¿Como sabemos que lo está haciendo bien?

- Con un set independiente de datos al que llamamos **testing set**, que va calculando los mismos estadísticos a la vez. Importante para detectar **overfitting**



Clasificación

Precisión (o TPR)

$$TPR = \frac{TP}{TP+FP}$$

Compleitud (o recall)
 $Rec = \frac{TP}{TP+FN}$

REDES NEURONALES

ENTRENAMIENTO

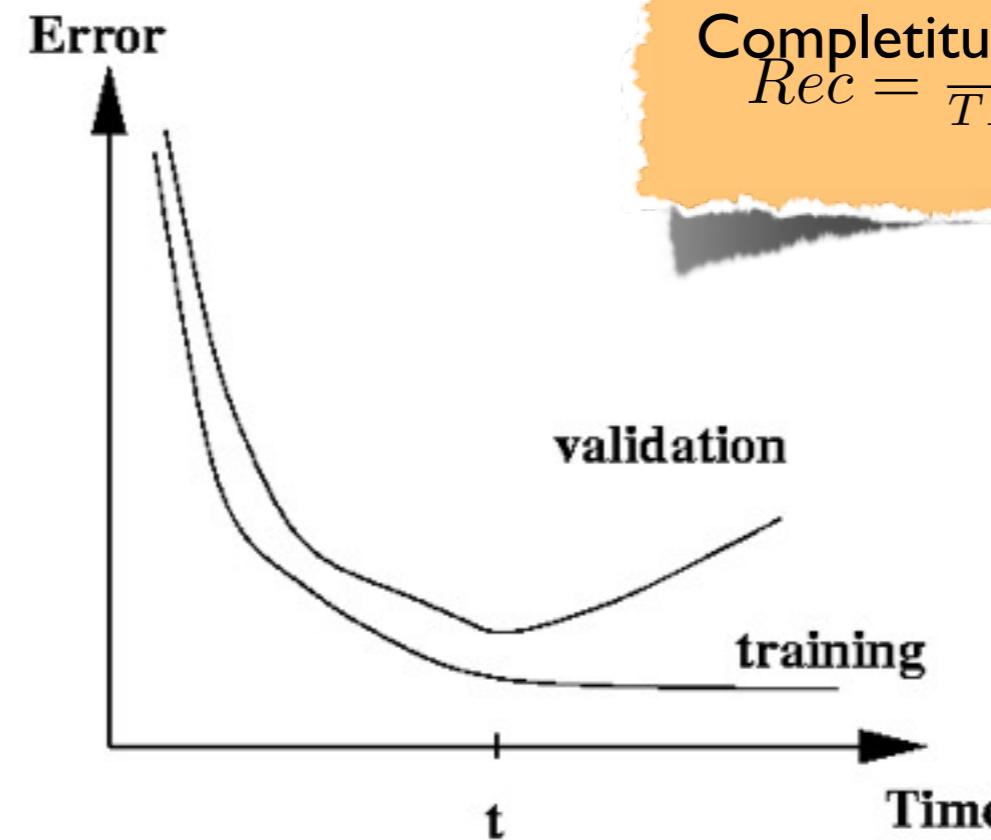
y ¿Como sabemos que lo está haciendo bien?

- Con un set independiente de datos al que llamamos **testing set**, que va calculando los mismos estadísticos a la vez. Importante para detectar **overfitting**

Regresión:

$$E = \sum_n \frac{(t^n - y^n)^2}{N}$$

$$\rho = \frac{\sigma_{ty}}{\sigma_t \sigma_y}$$



Clasificación

Precisión (o TPR)

$$TPR = \frac{TP}{TP+FP}$$

Complejidad (o recall)
 $Rec = \frac{TP}{TP+FN}$

REDES NEURONALES

GENERALIZACIÓN

Una vez entrenada la red ponemos cualquier input x y nos dará el resultado que predice la red.

REDES NEURONALES

GENERALIZACIÓN

Una vez entrenada la red ponemos cualquier input x y nos dará el resultado que predice la red.

$$y_k = g\left(\left(\sum_j w_{jk} \dots f\left(\sum_i w_{ij} x_i\right)\right)\right)$$

REDES NEURONALES

Observaciones

Algunas cosas a tener en cuenta:

- * Algunos problemas es mejor estandarizar los inputs y los outputs del **training** para hacer más compacto el espacio de parámetros y dar pesos parecidos

$$x_i = \frac{x_i - \langle x_i \rangle}{\sigma_i}$$

- * Podremos hacer predicciones en el rango del entrenamiento. Siendo menos efectivo en los bordes (edge problem)

- * En ocasiones puede ser útil entrenar varias redes con training sets distintos y hacer promedio de los resultados.

REDES NEURONALES

BIG DATA

- Con la digitalización y la tecnología Big Data se dan las condiciones perfectas para sacar el máximo provecho de estas herramientas. (Sets de entrenamiento grandes y extensos.)
- con DEEP Learning (redes neuronales convolucionales) se obtienen resultados **impresionantes** en reconocimiento de imágenes, detección de objetos, ...



NN con Python

(vemos ejemplo con notebook)

sklearn
(otros Keras, pylearn,...)



`mlp=MLPClassifier(hidden_layer_sizes=,activation=,batch_size=,verbose=...)` Objeto donde definimos arquitectura de la red para clasificación

`mlp=MLPRegressor(hidden_layer_sizes=,max_iter=,verbose=True,...)` Objeto donde definimos arquitectura de la red para regresión

`mlp.fit(x,y)` Entrena la red

`mlp.predict(xnew)` Hace predicciones, nos da `ynew` para `xnew`

`confusion_matrix(t,predictions)` True positives and false positives de cada clase

`classification_report(t,predictions)` precision and recall for classification problems

REDES NEURONALES

LAB

Repaso

- El ejemplo que hemos visto está en NNlab.ipynb, y los datos en el repositorio que os pasé
- Probad varias configuraciones (número neuronas, número de capas ocultas, funciones de activación y ver que funciona mejor)

Ejercicio I

- Entrenad una red para que aprenda a multiplicar 4 números (siguiendo ejemplo para que sume 3 números en NNExampleRegression.ipynb)

REDES NEURONALES

LAB

Ejercicio 2. Usar red neuronal para clasificar vino según su cultivador. Datos en wine_shuffled.data
(datos reales sacados de <https://archive.ics.uci.edu/>)

```
: df = pd.read_csv('wine_shuffled.data')
df = df.loc[:, ~df.columns.str.contains('^\u00c1nnamed') ]
df
```

	Cultivator	"Alchol"	"Malic_Acid"	"Ash"	"Alcalinity_of_Ash"	"Magnesium"	"Total_phenols"	"Falvanoids"	"Nonflavanoid_phenols"	"Proanthocyanins"	"C"
0	2	13.49	1.66	2.24	24.0	87	1.88	1.84	0.27	1.03	
1	3	12.86	1.35	2.32	18.0	122	1.51	1.25	0.21	0.94	
2	3	12.70	3.55	2.36	21.5	106	1.70	1.20	0.17	0.84	
3	3	13.32	3.24	2.38	21.5	92	1.93	0.76	0.45	1.25	
4	3	12.25	3.88	2.20	18.5	112	1.38	0.78	0.29	1.14	
5	1	14.75	1.73	2.39	11.4	91	3.10	3.69	0.43	2.81	
6	2	11.66	1.88	1.92	16.0	97	1.61	1.57	0.34	1.15	
7	2	12.08	1.33	2.30	23.6	70	2.20	1.59	0.42	1.38	
8	2	11.84	0.89	2.58	18.0	94	2.20	2.21	0.22	2.35	
9	3	12.53	5.51	2.64	25.0	96	1.79	0.60	0.63	1.10	
10	2	11.45	2.40	2.42	20.0	96	2.90	2.79	0.32	1.83	
11	2	12.37	1.13	2.16	19.0	87	3.50	3.10	0.19	1.87	
12	3	13.84	4.12	2.38	19.5	89	1.80	0.83	0.48	1.56	
13	1	14.06	2.15	2.61	17.6	121	2.60	2.51	0.31	1.25	
14	2	12.33	1.10	2.28	16.0	101	2.05	1.09	0.63	0.41	