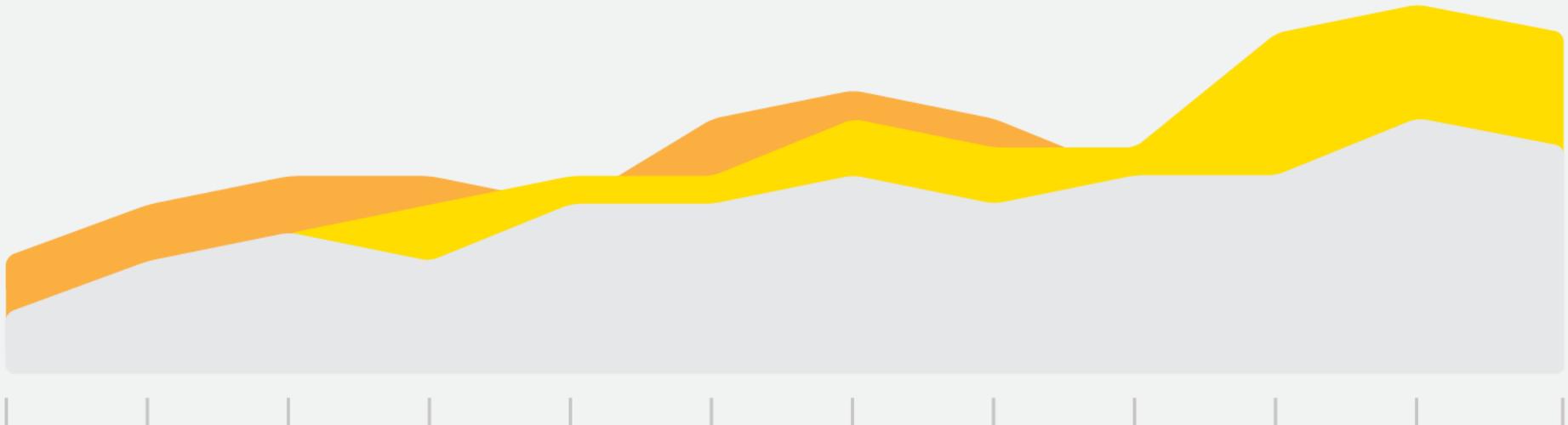




Aprendizaje Supervisado

Redes Neuronales

(Deep Learning)



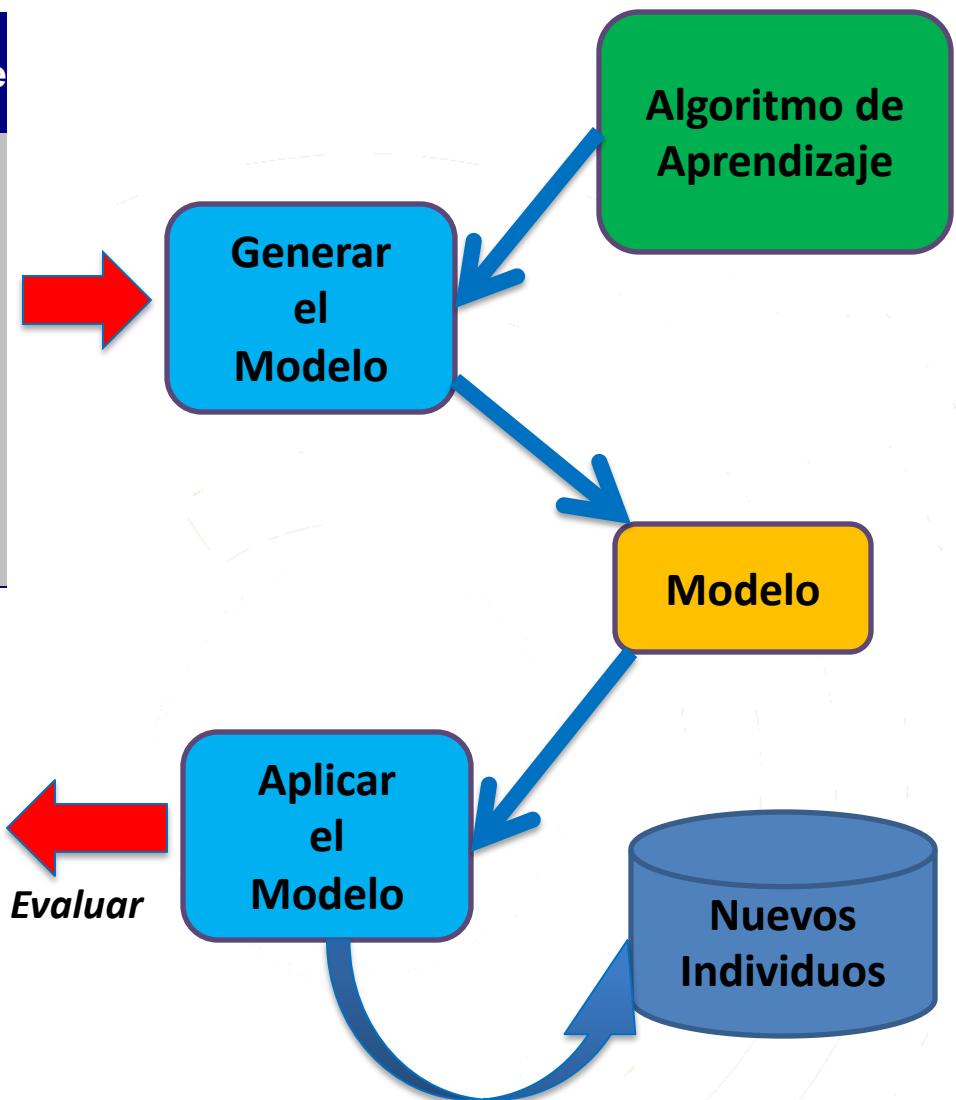
Modelo general de los métodos de Clasificación

/d	Reembolso	Estado Civil	Ingresos Anuales	Fraude
1	Sí	Soltero	125K	No
2	No	Casado	100K	No
3	No	Soltero	70K	No
4	Sí	Casado	120K	No
5	No	Divorciado	95K	Sí
6	No	Casado	60K	No

Tabla de Aprendizaje

/d	Reembolso	Estado Civil	Ingresos Anuales	Fraude
7	No	Soltero	80K	No
8	Si	Casado	100K	No
9	No	Soltero	70K	No

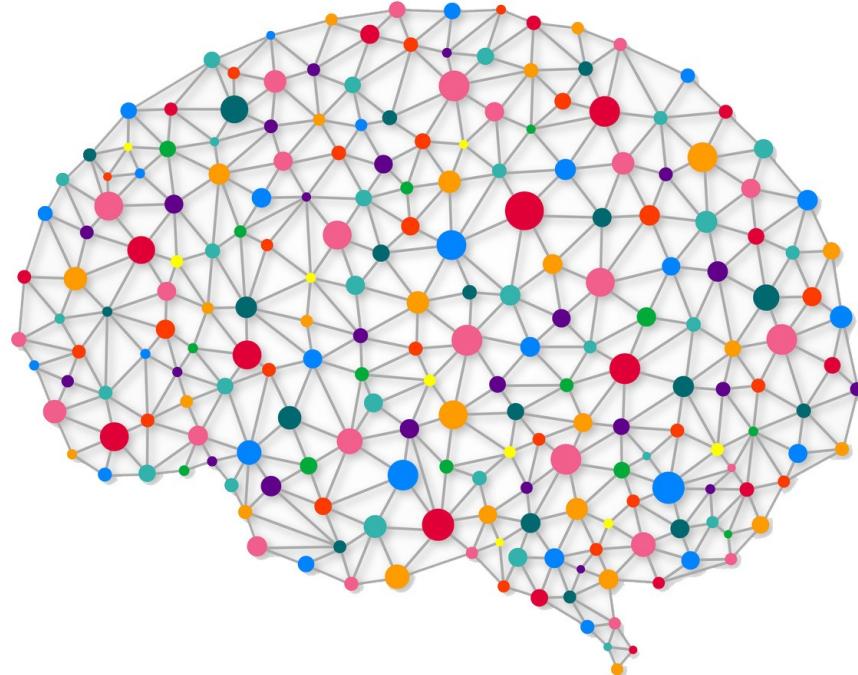
Tabla de Testing



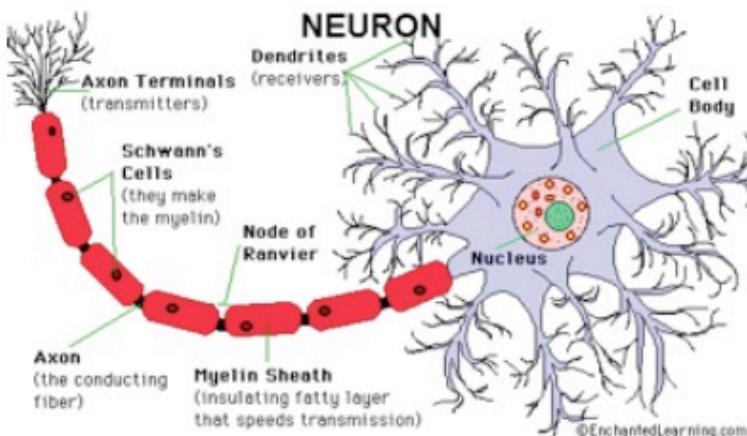
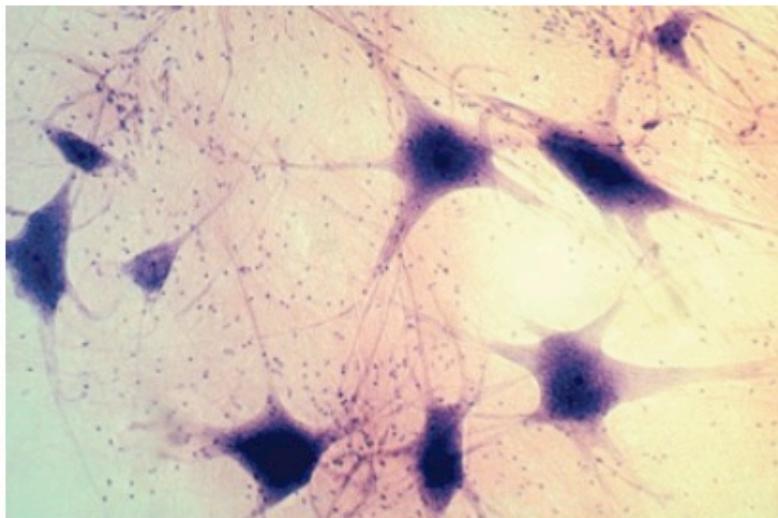
Definición de Clasificación

- Dada una base de datos $D = \{t_1, t_2, \dots, t_n\}$ de tuplas o registros (individuos) y un conjunto de clases $C = \{C_1, C_2, \dots, C_m\}$, el **problema de la clasificación** es encontrar una función $f: D \rightarrow C$ tal que cada t_i es asignada una clase C_j .
- $f: D \rightarrow C$ podría ser una Red Neuronal, un Árbol de Decisión, un modelo basado en Análisis Discriminante, o una Red Bayesiana.

Analogía con el cerebro humano



Redes Neuronales - Perceptrón

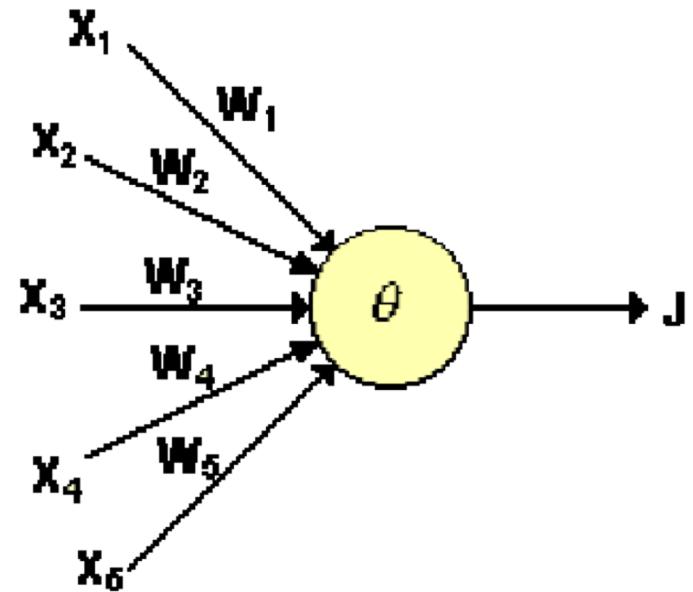
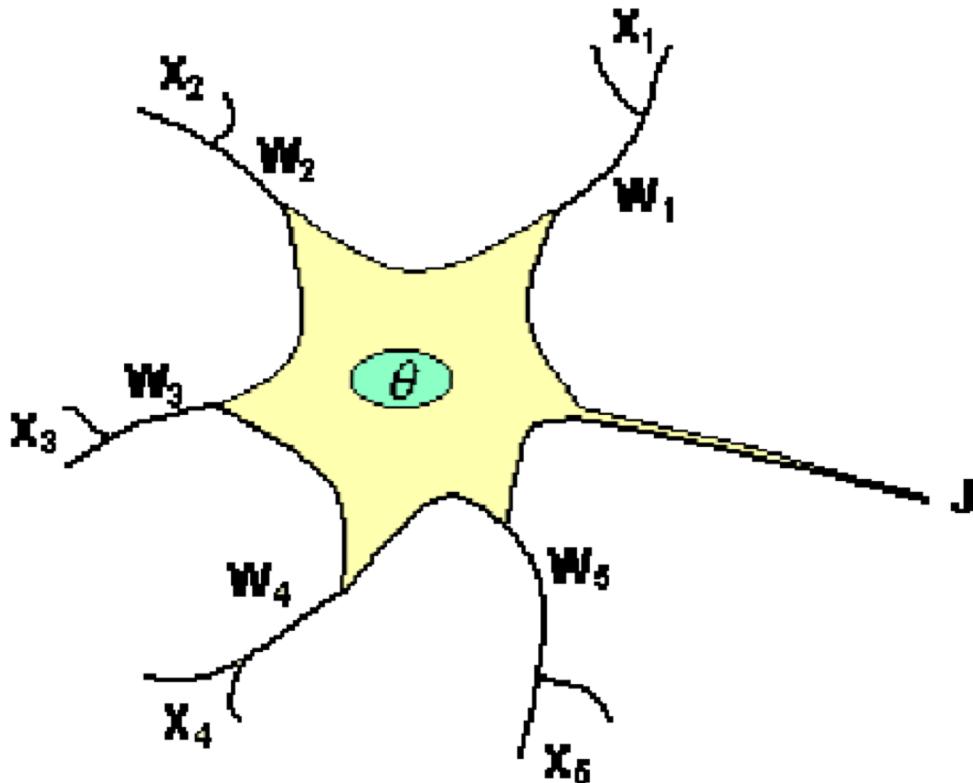


- El cerebro humano está compuesto principalmente de células nerviosas llamada **Neuronas**.
- Estas neuronas están ligadas mediante unas fibras llamadas “**Axons**”.
- Una Neurona está conectada al Axón de otra Neorona mediante las **Dentritas**.
- En punto de contacto entre una Dentrita y el Axón se llama **Synapse**.
- Las **Redes Neuronales Artificiales** tratan de emular este esquema mediante **Nodos y Links**.

Usos de las Redes Neuronales

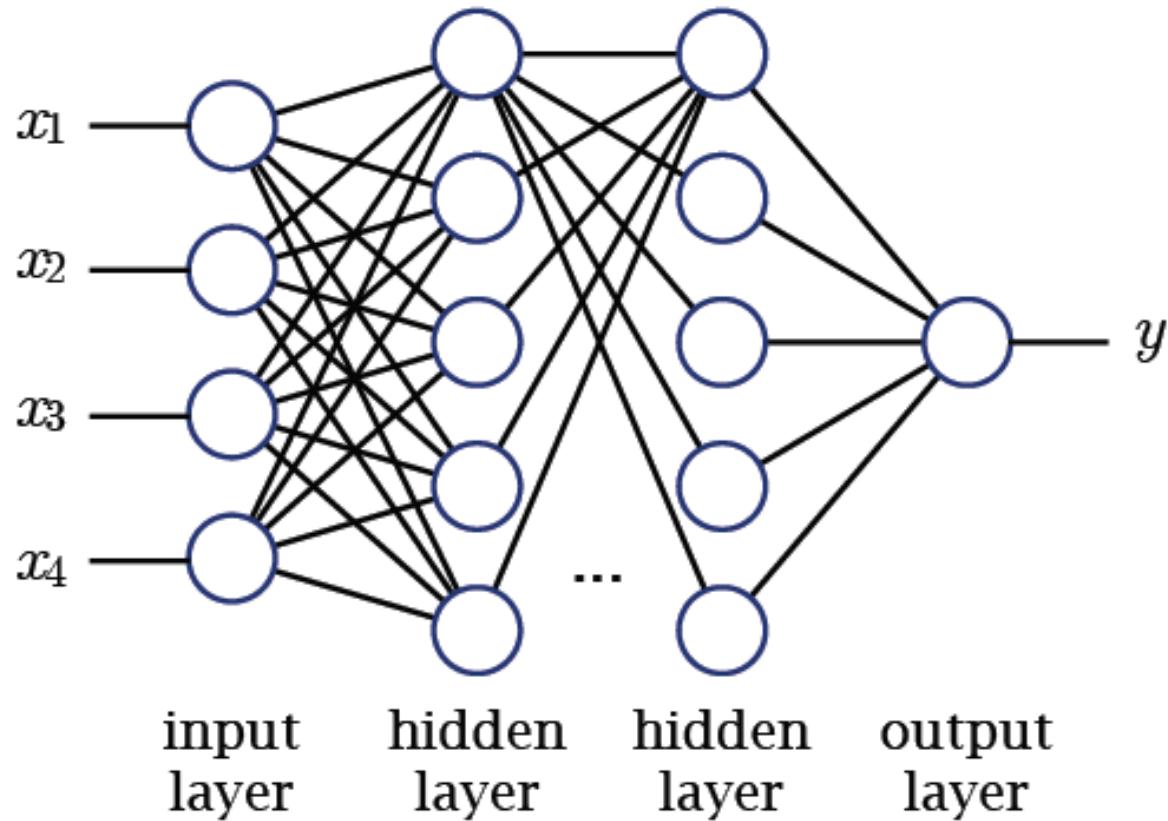
- Clasificación, regresión, series de tiempo.
- Reconocimiento de imágenes (*computer vision*).
- Reconocimiento de voz.
- Traducción automática.
- Autos de conducción automática.
- Publicidad dirigida.
- Arte, música.

Redes Neuronales - Perceptrón

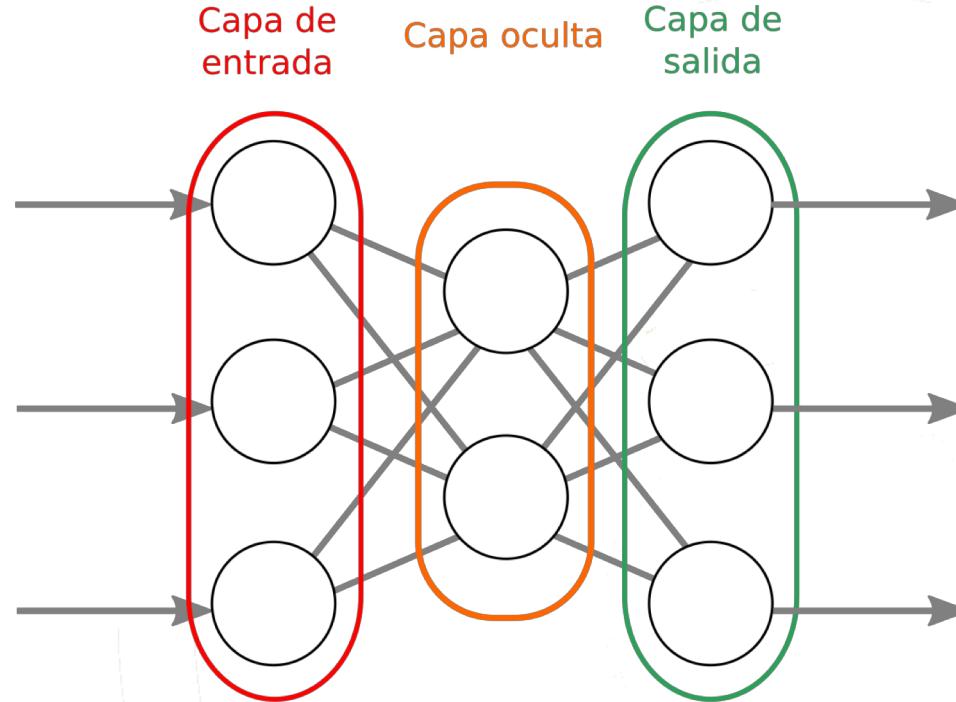


Multi-layer perceptrons

feed-forward network

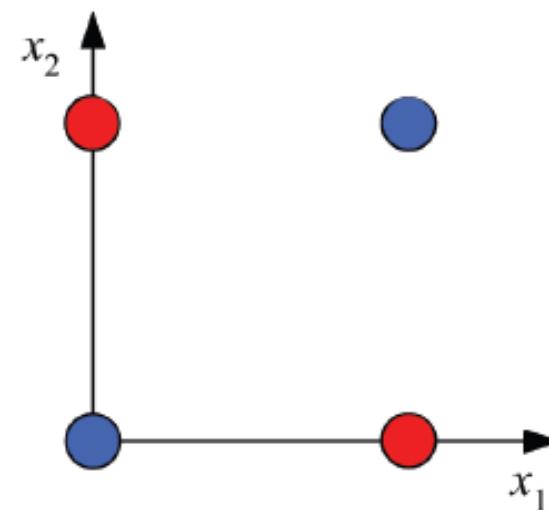


Paquete nnet – solo una capa oculta



Perceptrón - xor

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0

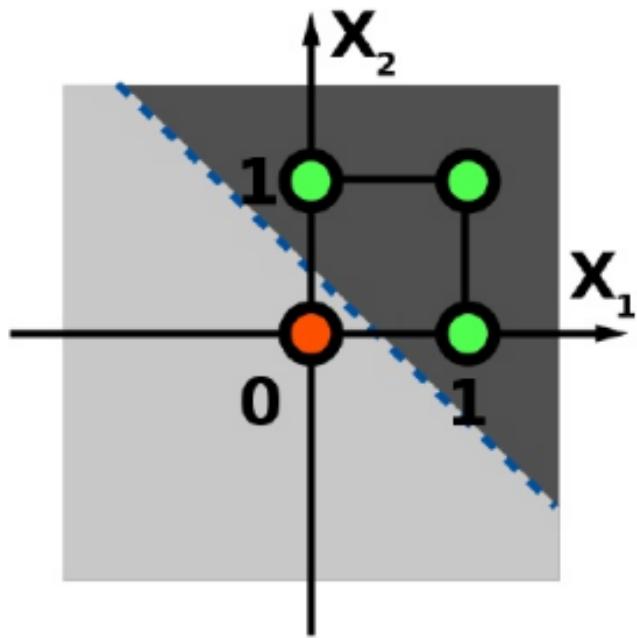


[Minsky and Papert, *Perceptrons*, 1969]



Marvin Minsky
Turing Award 1969

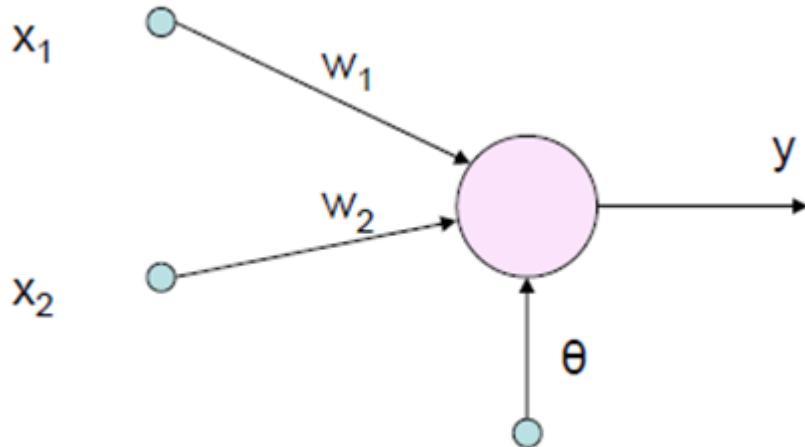
Perceptrón - or



OR ($x_1 \cup x_2$)

X1	X2	OR
0	0	0
0	1	1
1	0	1
1	1	1

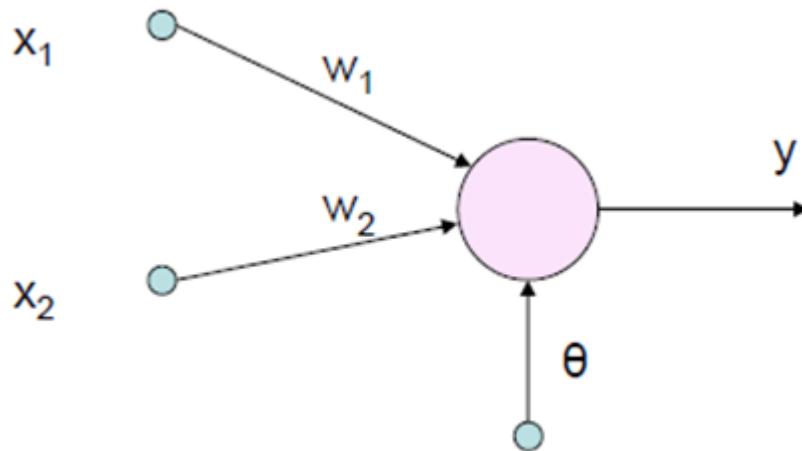
Perceptrón - or



$$Y = I(W_1 X_1 + W_2 X_2 > \theta)$$

donde $I(z) = \begin{cases} 1 & \text{Si } z \text{ es verdadero} \\ 0 & \text{En otro caso} \end{cases}$

Perceptrón - or



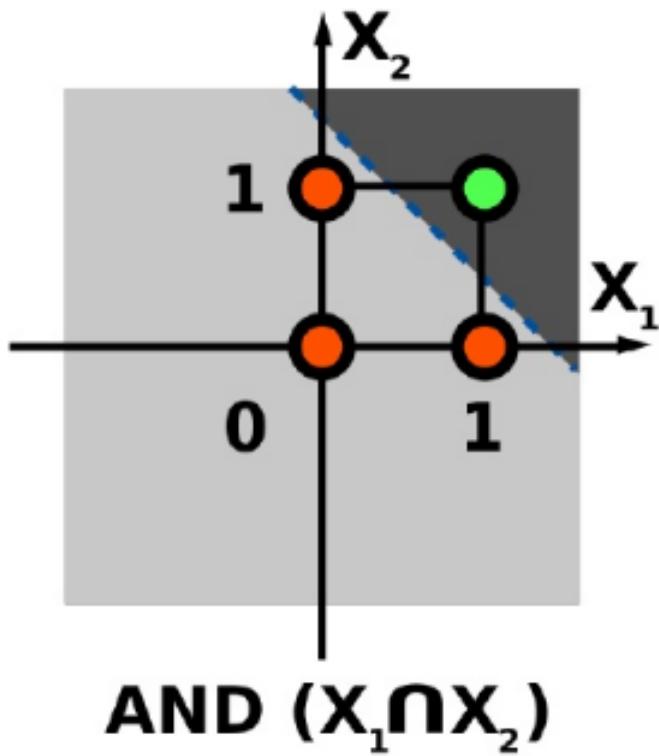
Si $\theta = 0.5$, $W_1 = 1$ y $W_2 = 1$ entonces

$$Y = I(X_1 + X_2 > 0.5)$$

donde $I(z) = \begin{cases} 1 & \text{Si } z \text{ es verdadero} \\ 0 & \text{En otro caso} \end{cases}$

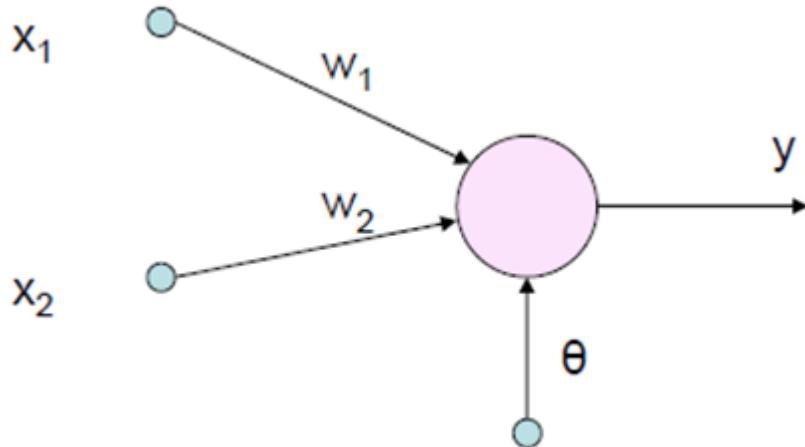
x_1	x_2	OR
0	0	0
0	1	1
1	0	1
1	1	1

Perceptrón - and



X1	X2	AND
0	0	0
0	1	0
1	0	0
1	1	1

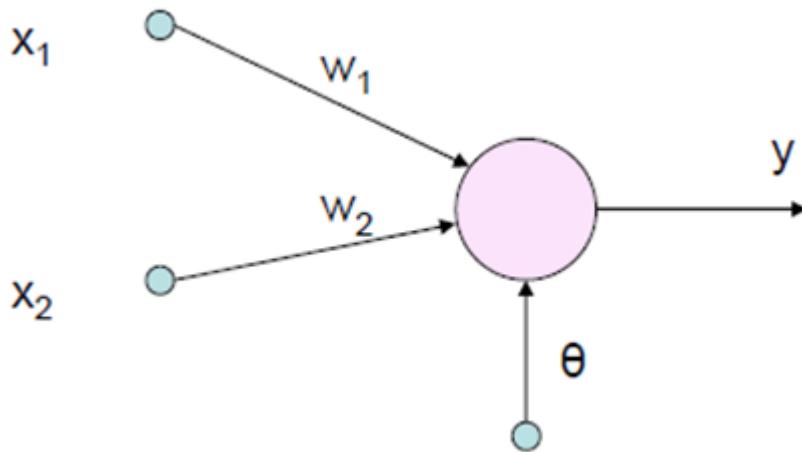
Perceptrón - and



$$Y = I(W_1X_1 + W_2X_2 > \theta)$$

donde $I(z) = \begin{cases} 1 & \text{Si } z \text{ es verdadero} \\ 0 & \text{En otro caso} \end{cases}$

Perceptrón - and



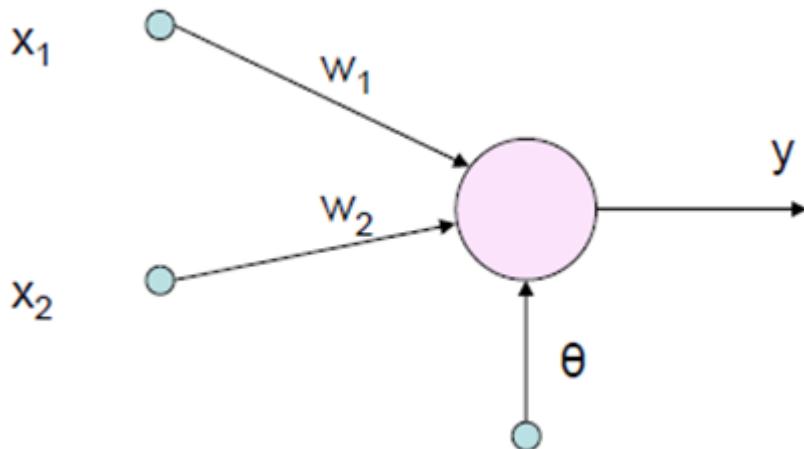
Si $\theta = 1.5$, $W_1 = 1$ y $W_2 = 1$ entonces

$$Y = I(X_1 + X_2 > 1.5)$$

donde $I(z) = \begin{cases} 1 & \text{Si } z \text{ es verdadero} \\ 0 & \text{En otro caso} \end{cases}$

x1	x2	AND
0	0	0
0	1	0
1	0	0
1	1	1

Perceptrón - and



θ se llama Umbral

W_1 y W_2 se llaman los Pesos

$I(Z)$ es la regla de decisión

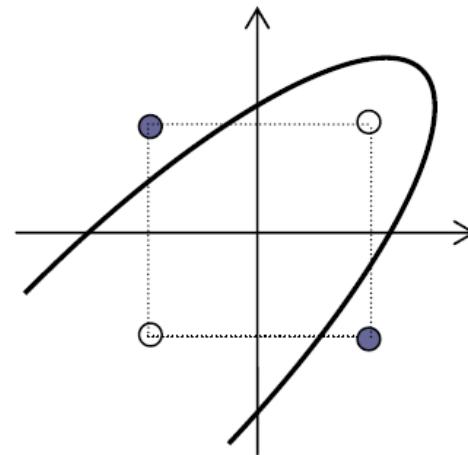
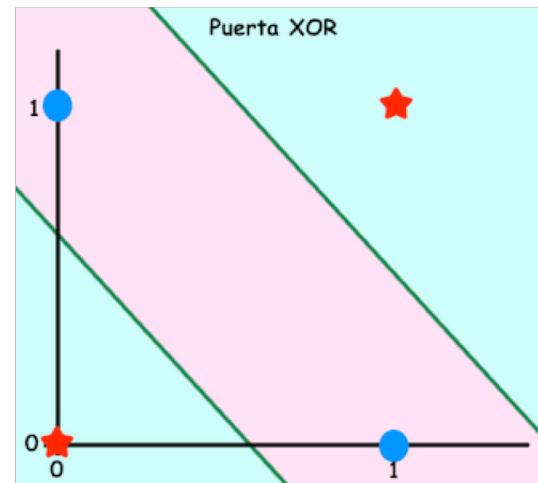
La Función de Activación usada

en todos los casos anteriores fue la identidad

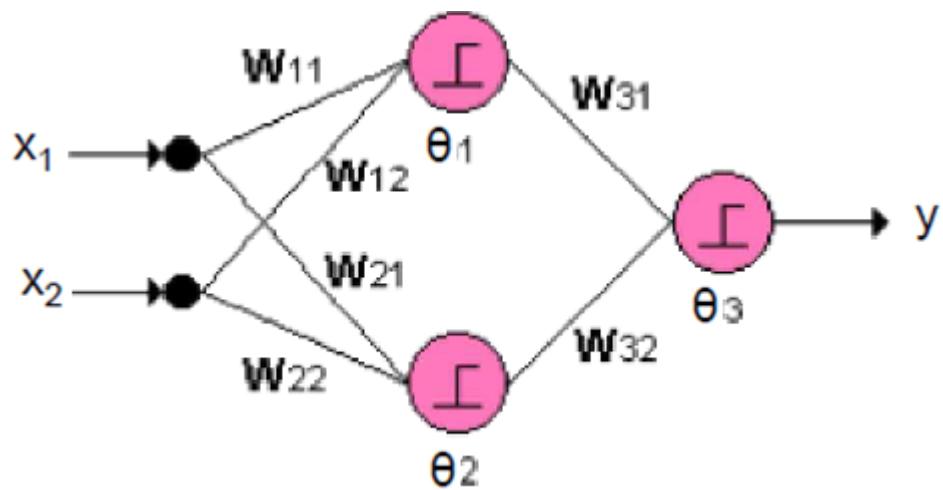
Perceptrón - xor

XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0



Perceptrón multicapa - xor

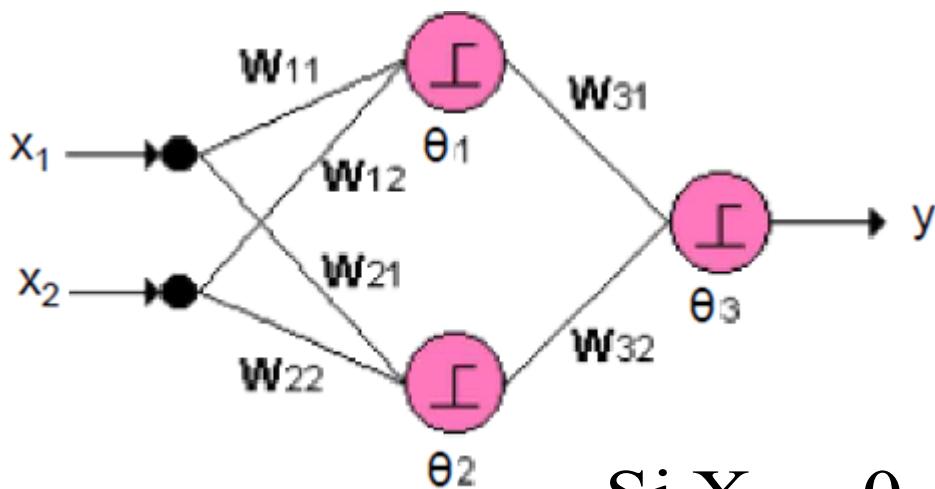


XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$$\begin{aligned} w_{11} &= 1 \quad w_{12} = 1 \\ w_{21} &= 1 \quad w_{22} = 1 \\ w_{31} &= 1 \quad w_{32} = -1.5 \\ \theta_1 &= 0.5 \quad \theta_2 = 1.5 \quad \theta_3 = 0.5 \end{aligned}$$

Perceptrón multicapa - xor



$$\begin{aligned} w_{11} &= 1 & w_{12} &= 1 \\ w_{21} &= 1 & w_{22} &= 1 \\ w_{31} &= 1 & w_{32} &= -1.5 \\ \theta_1 &= 0.5 & \theta_2 &= 1.5 & \theta_3 &= 0.5 \end{aligned}$$

Si $X_1 = 0$ y $X_2 = 0$ entonces

$$I(W_{11}X_1 + W_{12}X_2 = 0 > 0.5) = 0$$

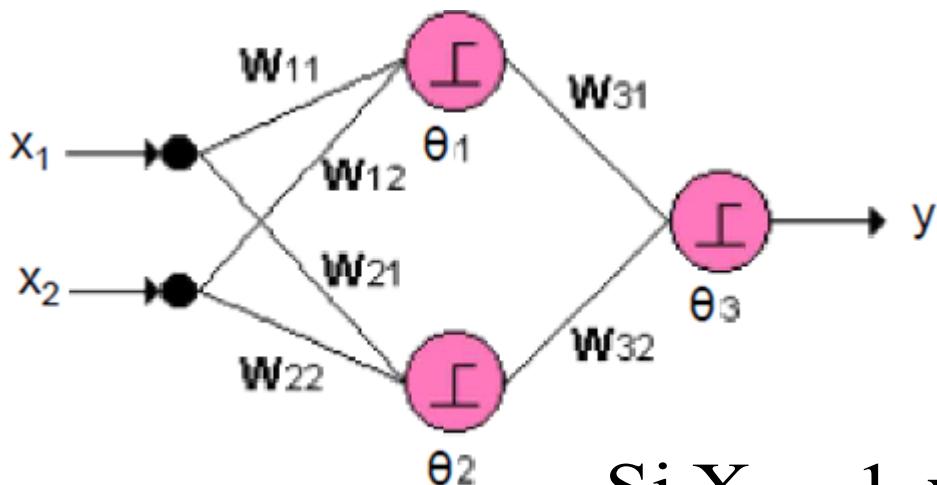
$$I(W_{21}X_1 + W_{22}X_2 = 0 > 1.5) = 0$$

$$Y = I(W_{31} * 0 + W_{32} * 0 > 0.5) = 0$$

donde $I(z) = \begin{cases} 1 & \text{Si } z \text{ es verdadero} \\ 0 & \text{En otro caso} \end{cases}$

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Perceptrón multicapa - xor



$$\begin{aligned} w_{11} &= 1 & w_{12} &= 1 \\ w_{21} &= 1 & w_{22} &= 1 \\ w_{31} &= 1 & w_{32} &= -1.5 \\ \theta_1 &= 0.5 & \theta_2 &= 1.5 & \theta_3 &= 0.5 \end{aligned}$$

Si $X_1 = 1$ y $X_2 = 0$ entonces

$$I(W_{11}X_1 + 0 = 1 > 0.5) = 1$$

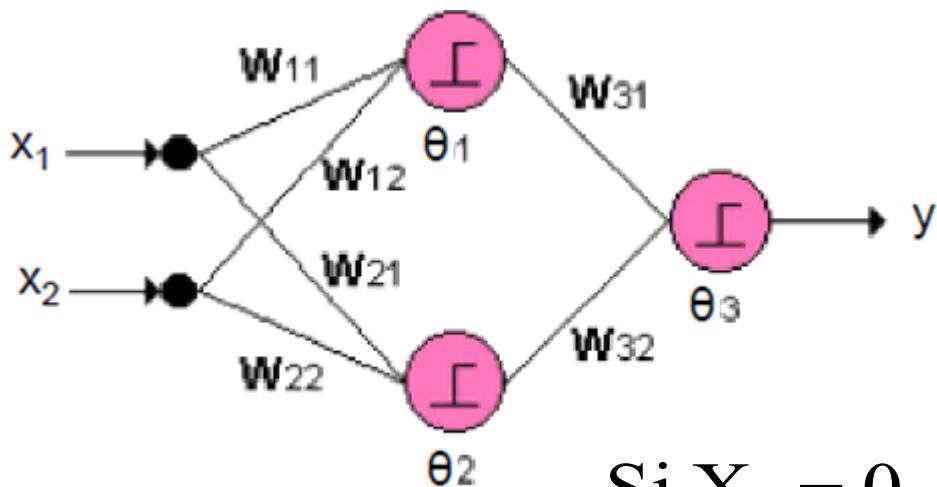
$$I(W_{21}X_1 + 0 = 1 > 1.5) = 0$$

$$Y = I(W_{31} * 1 + 0 = 1 > 0.5) = 1$$

donde $I(z) = \begin{cases} 1 & \text{Si } z \text{ es verdadero} \\ 0 & \text{En otro caso} \end{cases}$

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Perceptrón multicapa - xor



$$\begin{aligned} w_{11} &= 1 \quad w_{12} = 1 \\ w_{21} &= 1 \quad w_{22} = 1 \\ w_{31} &= 1 \quad w_{32} = -1.5 \\ \theta_1 &= 0.5 \quad \theta_2 = 1.5 \quad \theta_3 = 0.5 \end{aligned}$$

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Si $X_1 = 0$ y $X_2 = 1$ entonces

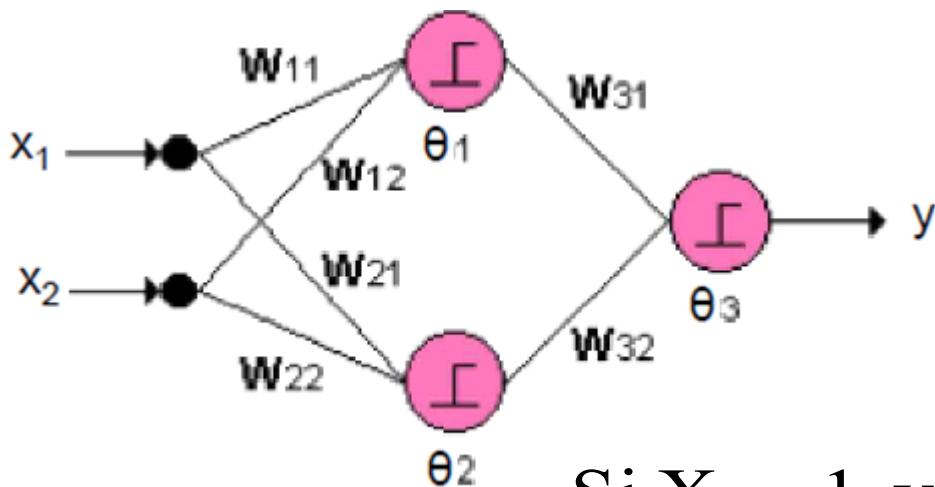
$$I(0 + W_{12}X_2 = 1 > 0.5) = 1$$

$$I(W_{22}X_2 + 0 = 1 > 1.5) = 0$$

$$Y = I(W_{31}X_1 + W_{32} * 0 = 1 > 0.5) = 1$$

donde $I(z) = \begin{cases} 1 & \text{Si } z \text{ es verdadero} \\ 0 & \text{En otro caso} \end{cases}$

Perceptrón multicapa - xor



$$\begin{aligned} w_{11} &= 1 & w_{12} &= 1 \\ w_{21} &= 1 & w_{22} &= 1 \\ w_{31} &= 1 & w_{32} &= -1.5 \\ \theta_1 &= 0.5 & \theta_2 &= 1.5 & \theta_3 &= 0.5 \end{aligned}$$

XOR

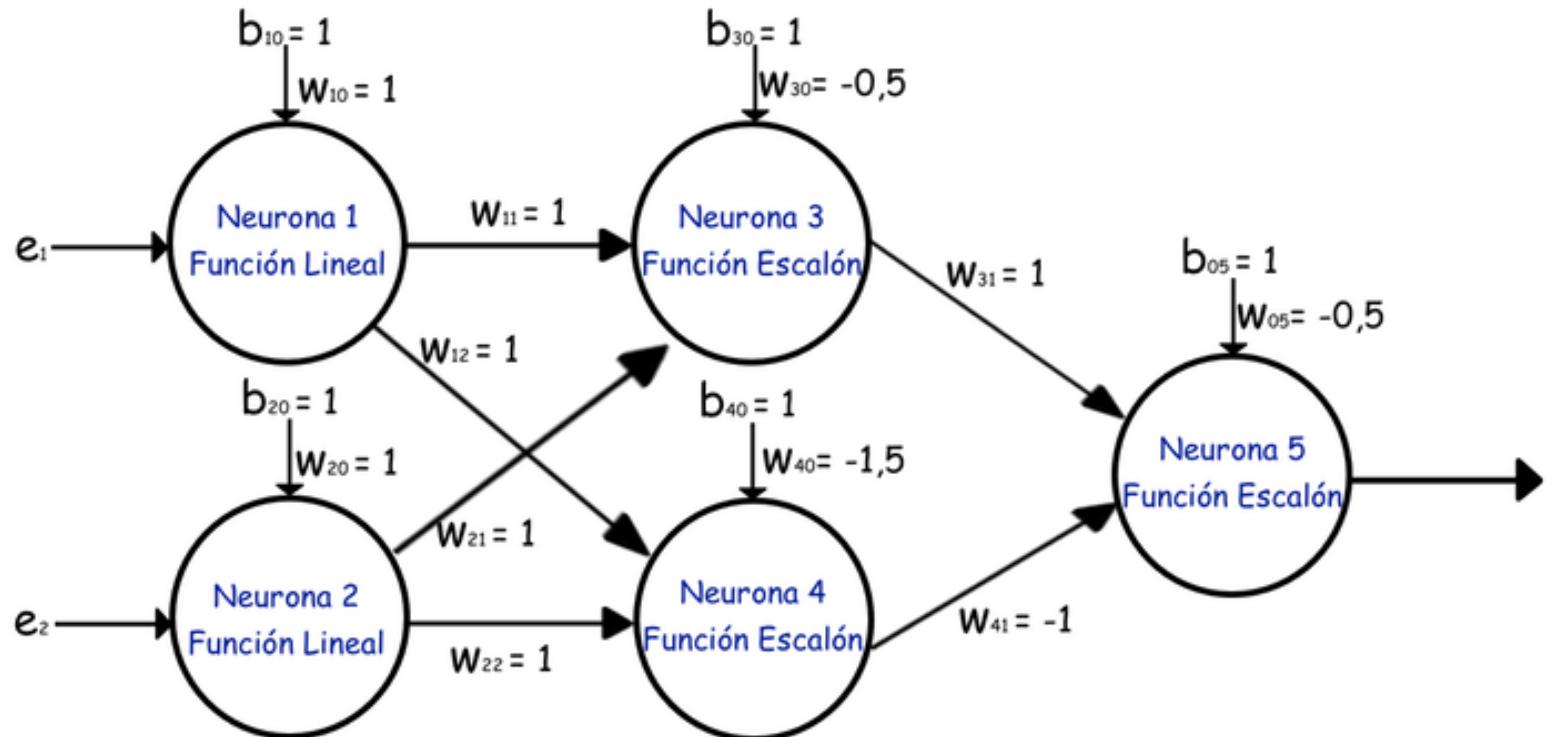
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Si $X_1 = 1$ y $X_2 = 1$ entonces

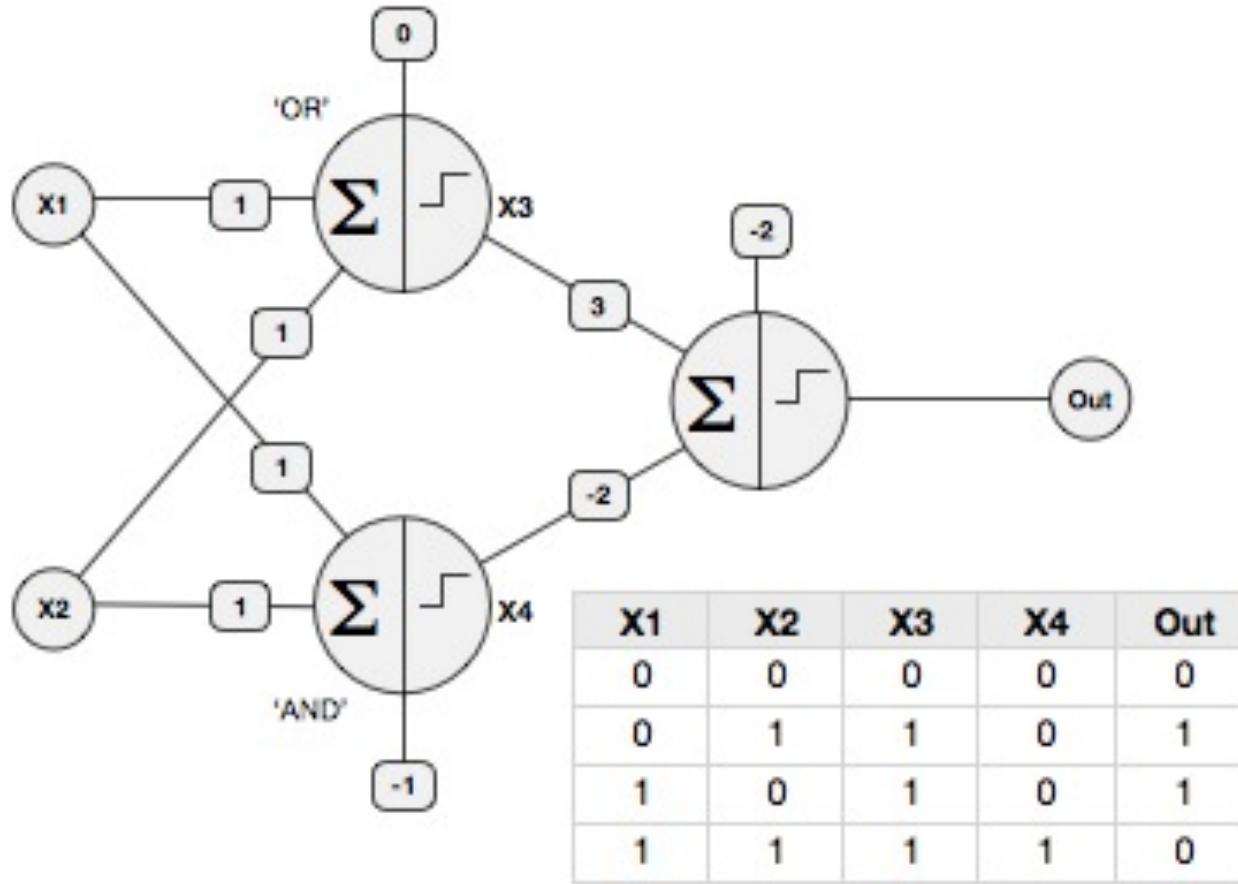
$$I(W_{11}X_1 + W_{12}X_2 = 2 > 0.5) = 1$$
$$I(W_{21}X_1 + W_{22}X_2 = 2 > 1.5) = 1$$
$$Y = I(W_{31} * 1 + W_{32} * 1 = -0.5 > 0.5) = 0$$

donde $I(z) = \begin{cases} 1 & \text{Si } z \text{ es verdadero} \\ 0 & \text{En otro caso} \end{cases}$

Perceptrón multicapa - xor



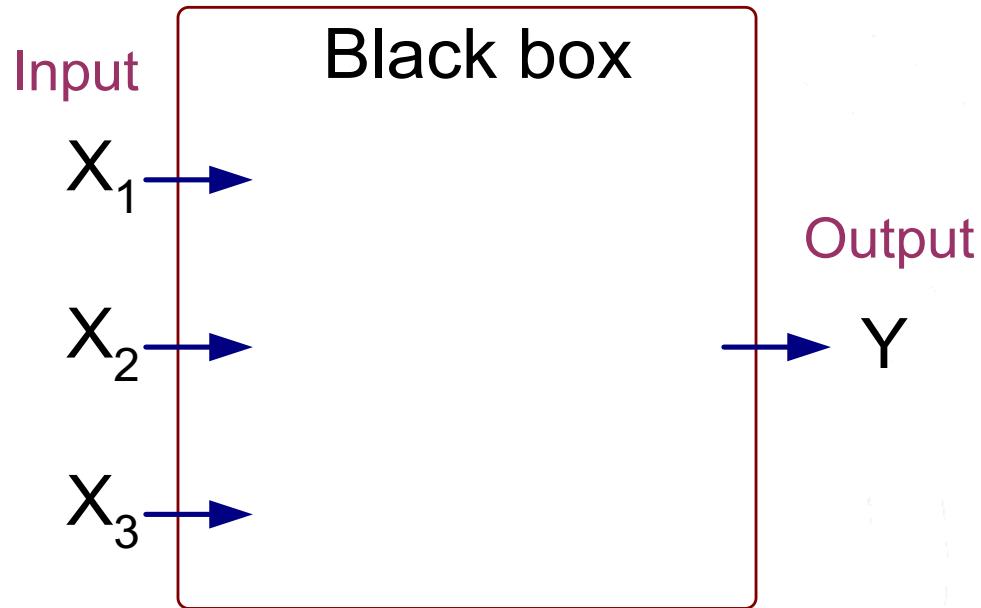
Perceptrón multicapa “xor” basado en el perceptrón “and” y el Perceptrón “or”



Redes Neuronales

Perceptrón

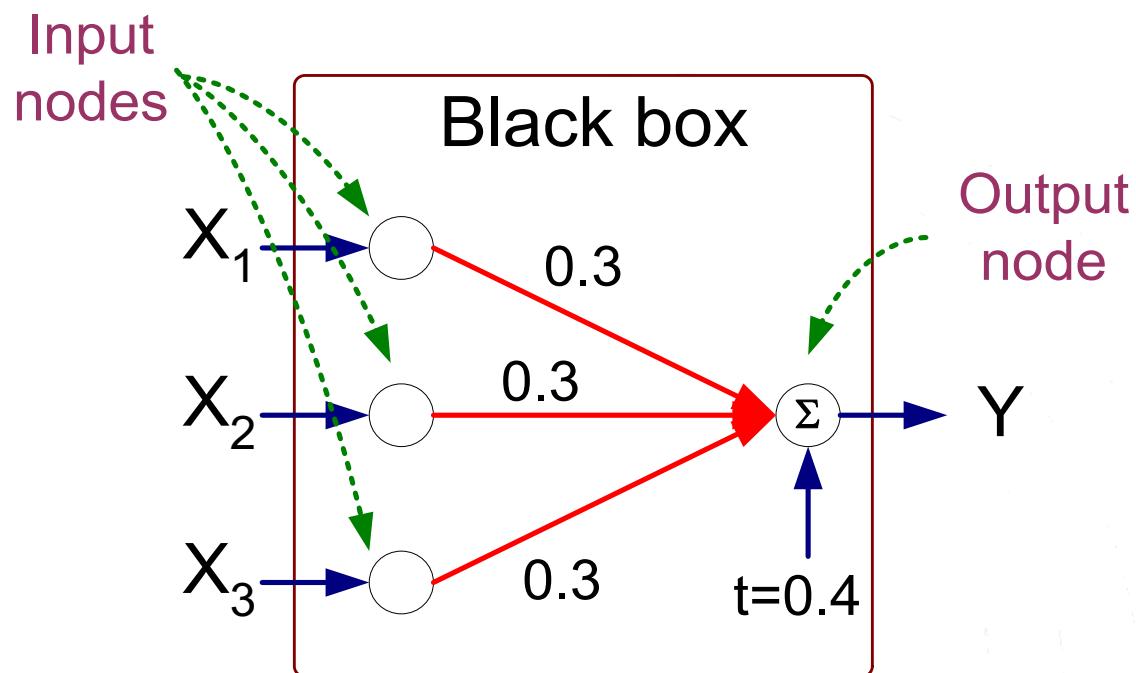
X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



La salida Y es 1 si al menos 2 de las 3 entradas son iguales a 1.

Redes Neuronales

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0

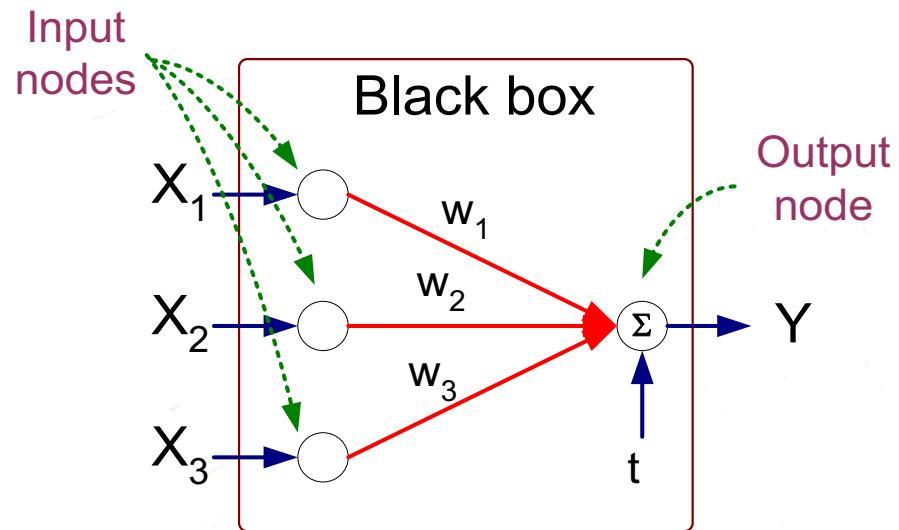


$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 > 0.4)$$

donde $I(z) = \begin{cases} 1 & \text{Si } z \text{ es verdadero} \\ 0 & \text{En otro caso} \end{cases}$

Redes Neuronales

- Modelo es un conjunto de nodos interconectados y enlaces ponderados
- Nodo de salida suma cada uno de su valor de entrada de acuerdo a los pesos de sus vínculos
- Comparar nodo de salida contra un **umbral t**

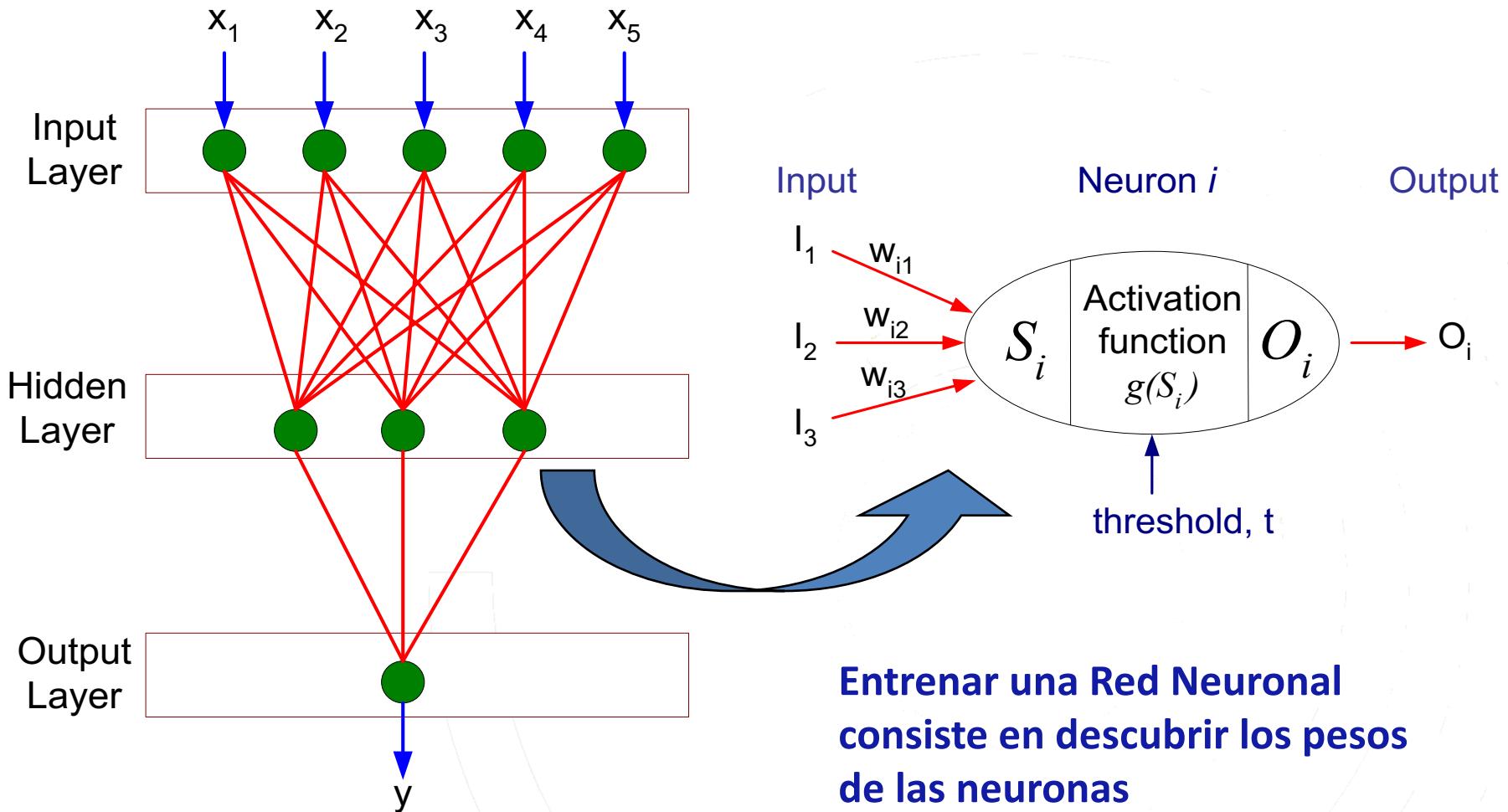


Modelo Perceptron

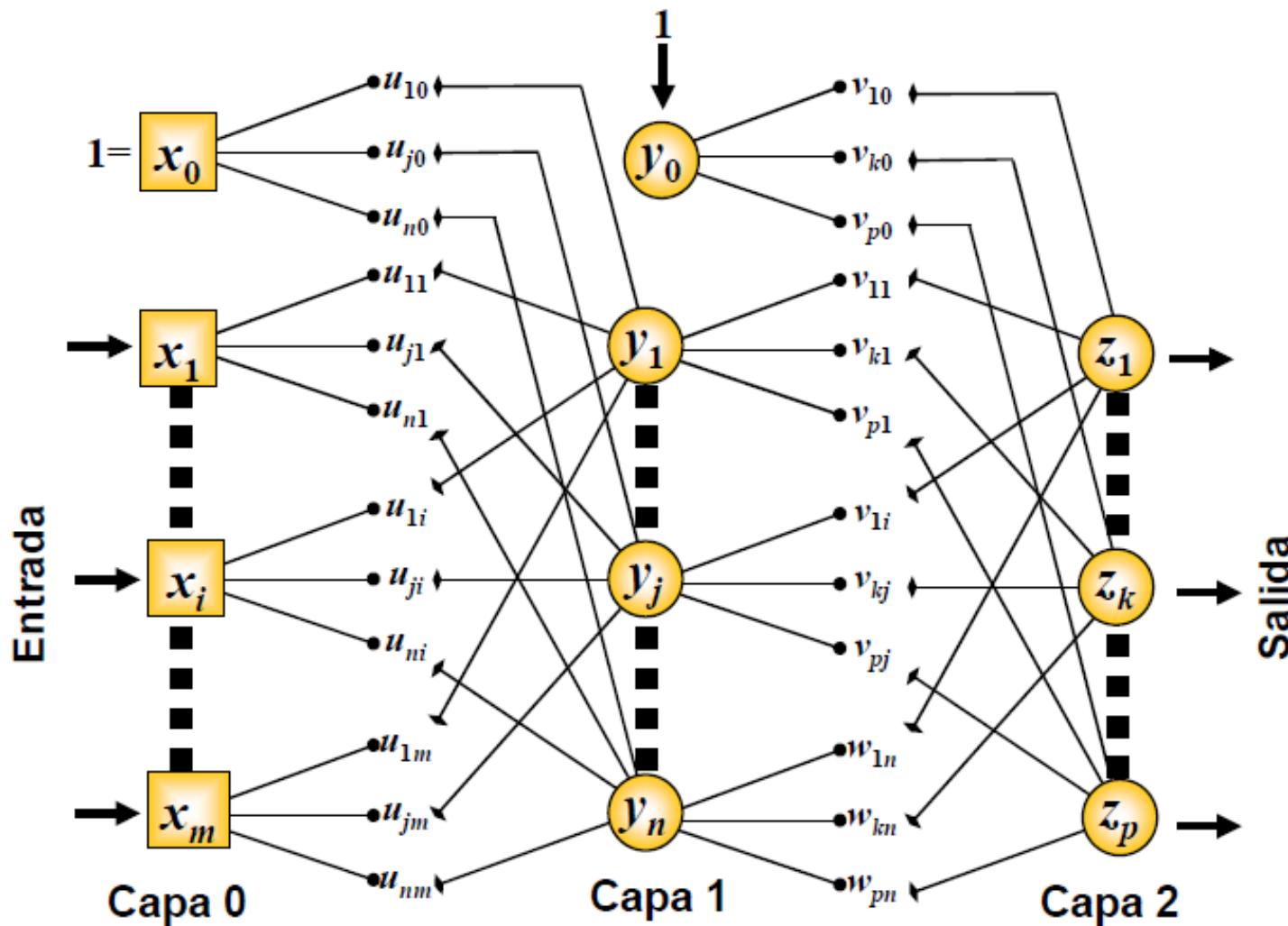
$$Y = I\left(\sum_i w_i X_i - t\right)$$

$$Y = \text{sign}\left(\sum_i w_i X_i - t\right)$$

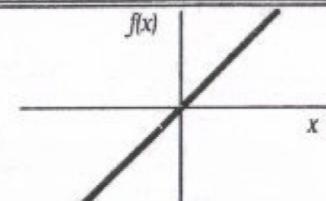
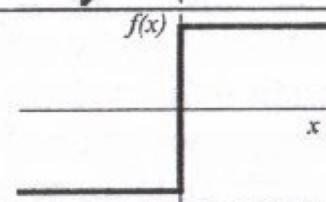
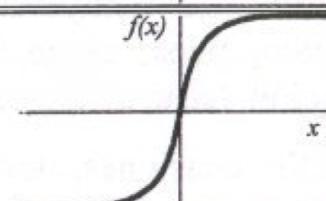
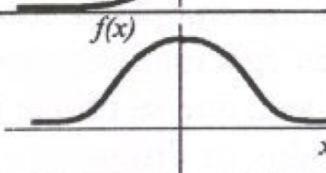
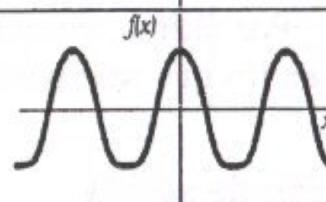
Estructura General de una Red Neuronal



Estructura General de una Red Neuronal



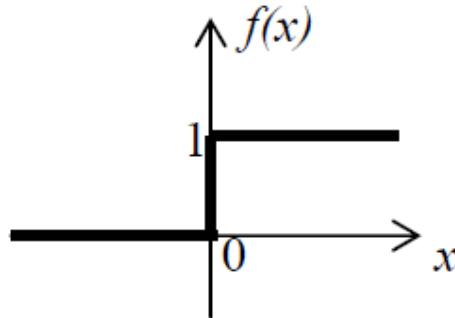
Algunas funciones de Activación

	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0, +1]$ $[-1, +1]$	
Gaussiana	$y = Ae^{-Bx^2}$	$[0, +1]$	
Sinusoidal	$y = A \sen(\omega x + \phi)$	$[-1, +1]$	

Algunas funciones de Activación:

En todos los ejemplos anteriores hemos usado la función Escalón de Heaviside como función de activación:

Función paso o
De Heaviside

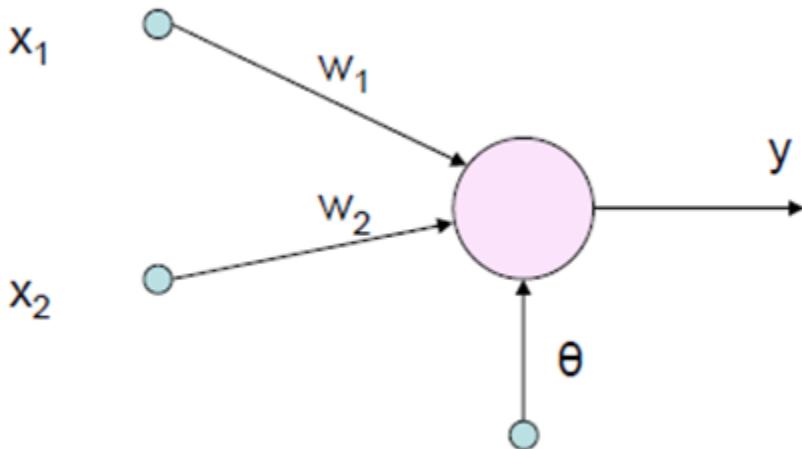


$$H(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$$

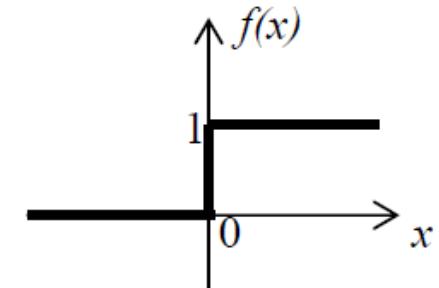
Otra forma de ver Perceptron del or que es equivalente a la vista anteriormente se muestra en la siguiente filmina:

Función de activación $H(x)$ =escalón

Perceptrón del or



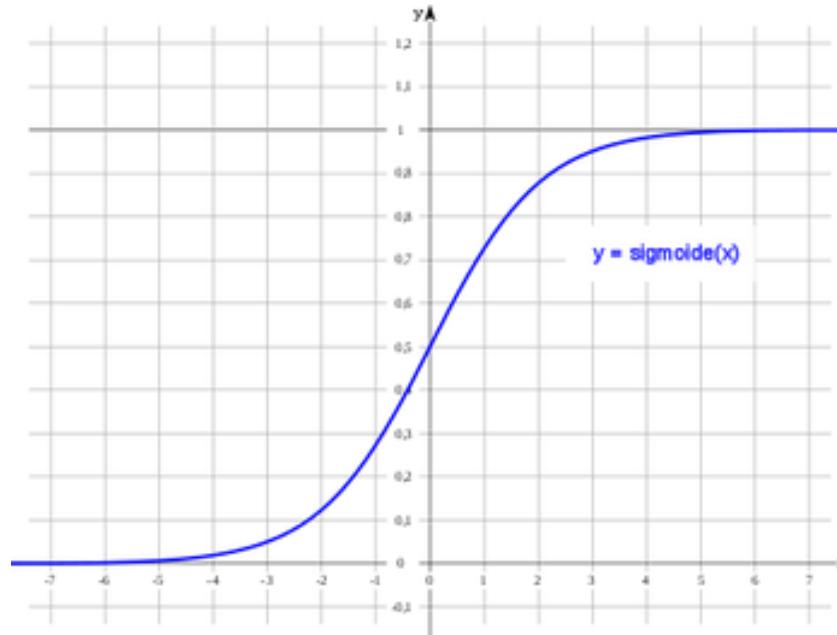
Función paso o
De Heaviside



$$Y = I(z) = H(z) = H(W_1 X_1 + W_2 X_2 - \theta)$$

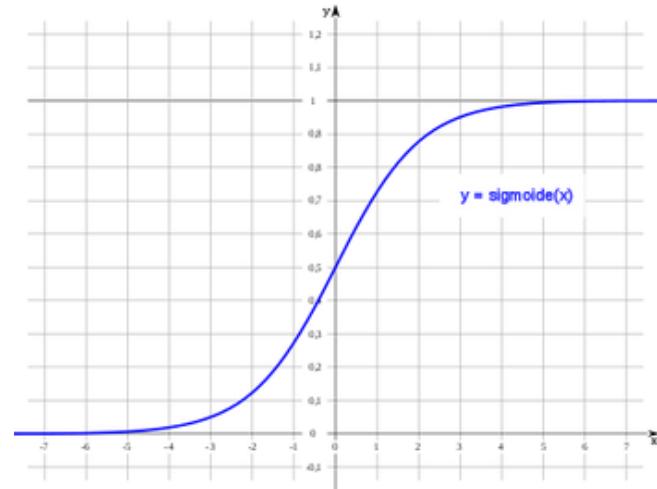
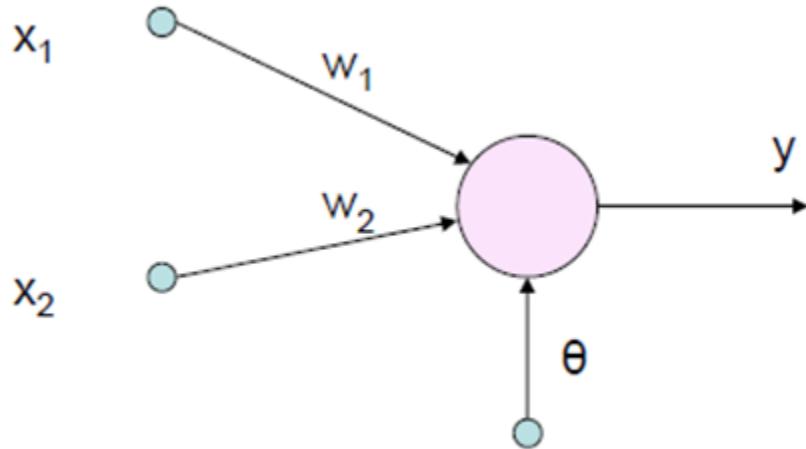
donde $H(z) = \begin{cases} 1 & \text{Si } z \geq 0 \\ 0 & \text{En otro caso} \end{cases}$

Función de activación $S(x)$ =Sigmoidea



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

Función de activación $S(x)$ =Sigmoidea en el Perceptrón del or



$$S(z) = S(W_1 X_1 + W_2 X_2 - \theta) = \frac{1}{1 + e^{-(W_1 X_1 + W_2 X_2 - \theta)}}$$

Como este valor $S(z)$ queda entre 0 y 1, $I(S(z))$ asigna 1 si $S(z) \geq 0.5$ y 0 en caso contrario

Algoritmo de Aprendizaje

- Inicializar los pesos $w=(w_0, w_1, \dots, w_k)$
- Ajustar los pesos de tal manera que la salida de la red Neuronal sea consistente con etiquetas de clase en los casos de entrenamiento:
$$E(w) = \sum_i [Y_i - f(w_i, X_i)]^2$$
- Función Objetivo:
 - Encuentra el peso w_i 's de que minimizan la función objetivo anterior (Error Cuadrático)
 - Ej. Algoritmo “backpropagation”

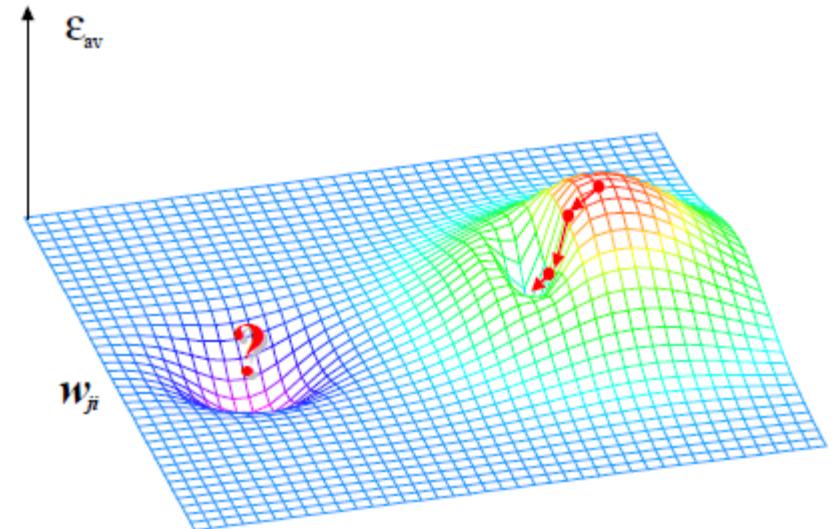
Descenso del Gradiente y Mínimos locales

Se debe minimizar la función :

$$E(w) = \sum_i [Y_i - f(w_i, X_i)]^2$$

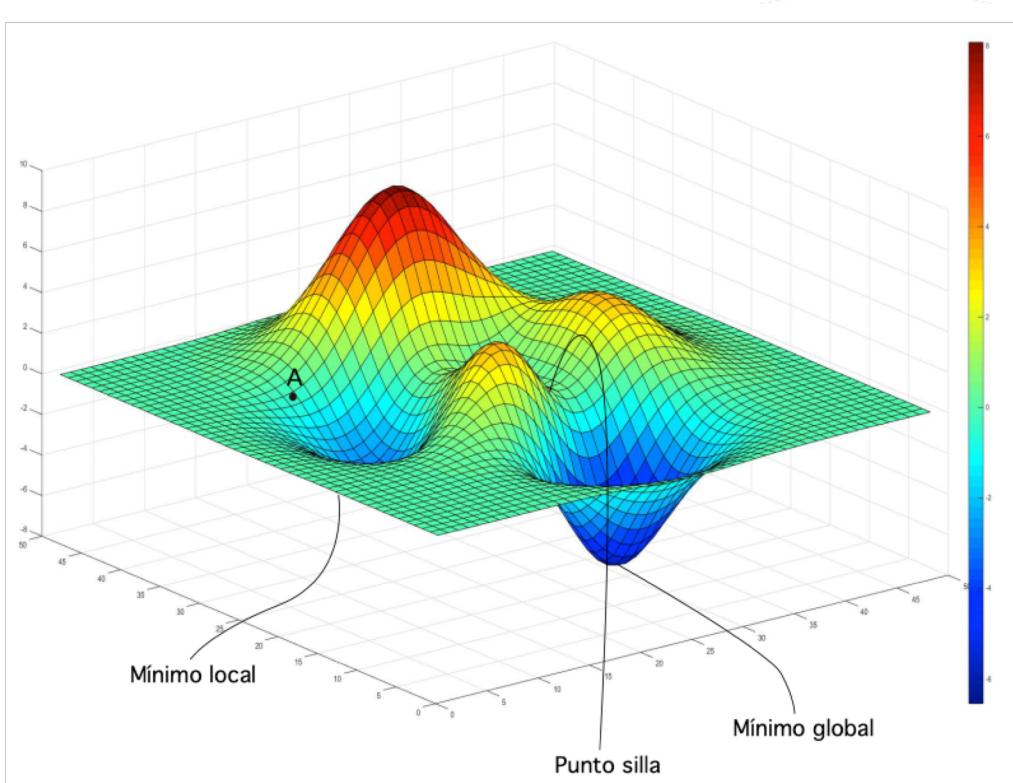
Usualmente se hace utilizando el descenso del gradiente :

$$w_j \leftarrow w_j - \lambda \frac{\partial E(w)}{\partial (w_j)}$$

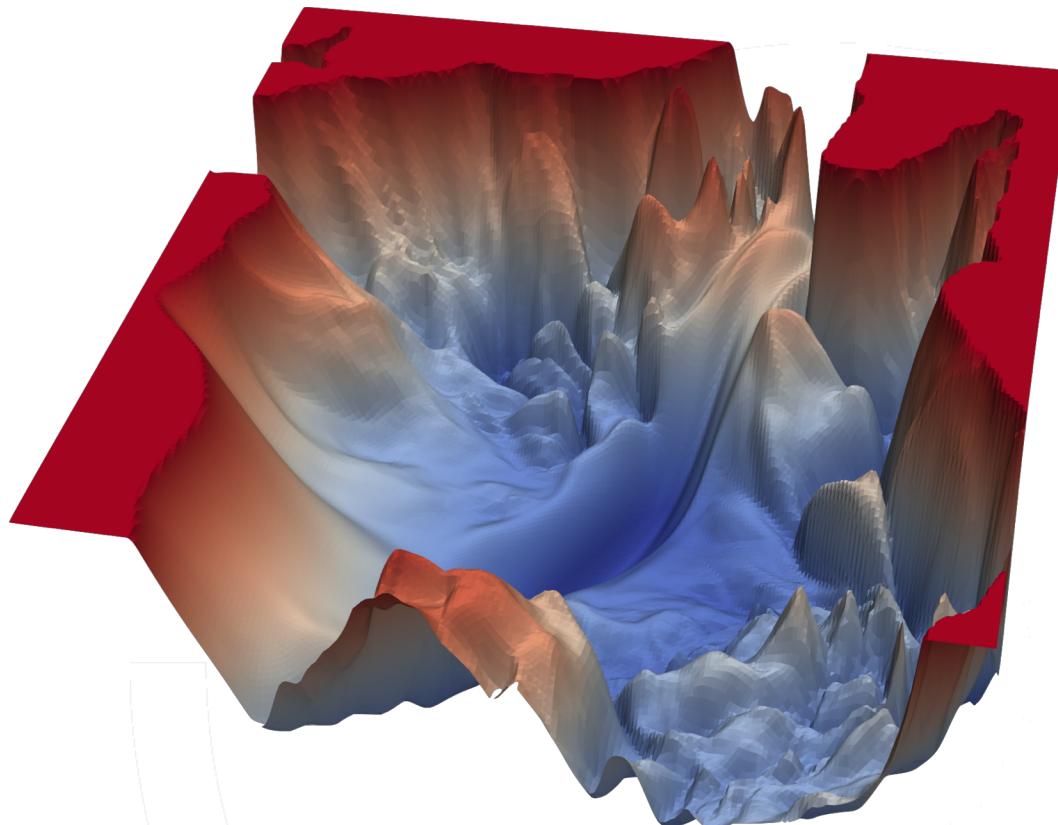


Peligro de caer en mínimo local.

Mínimos Locales



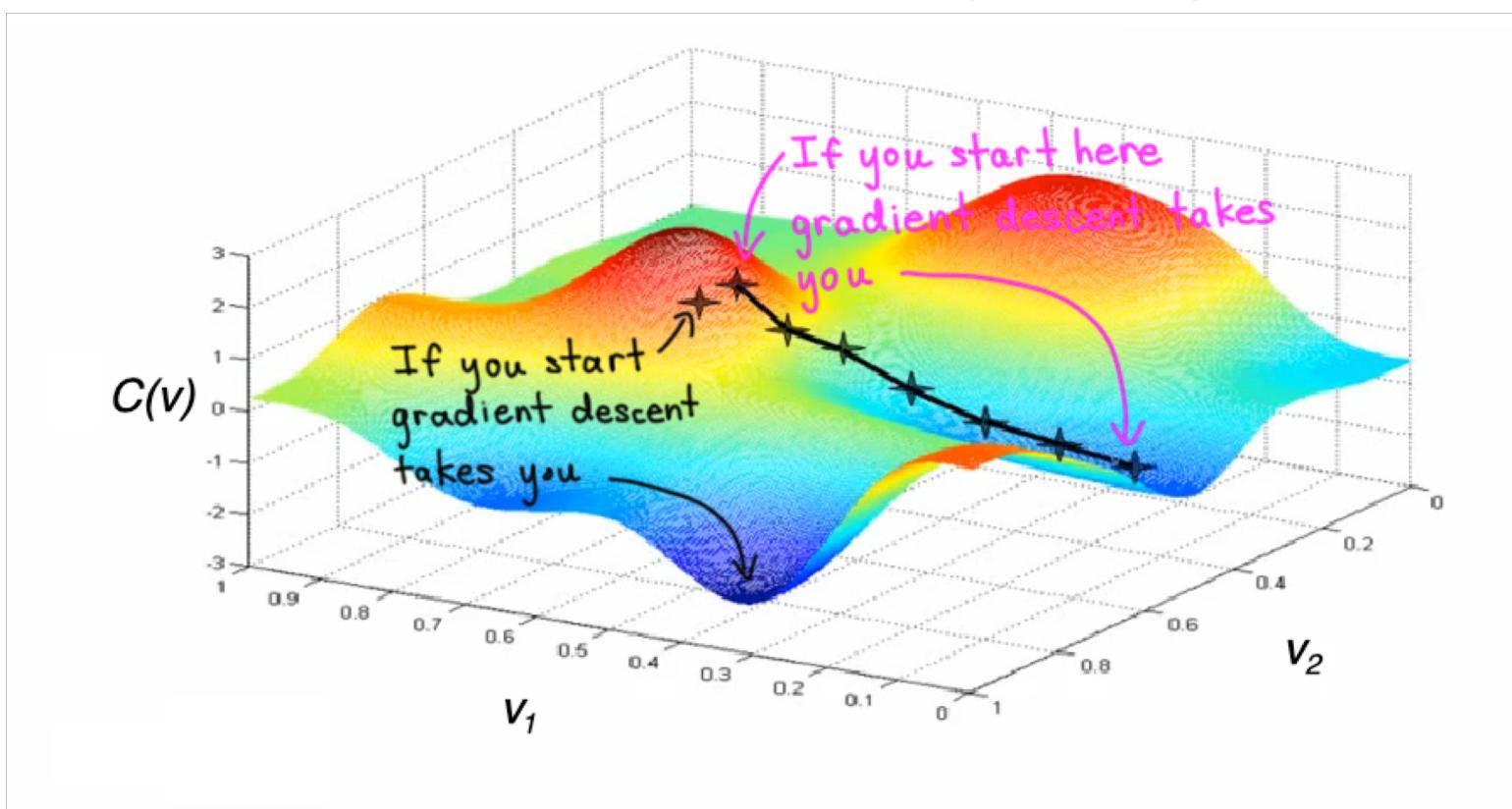
Una superficie más compleja



Tomada de:

<https://www.cs.umd.edu/~toma/projects/landscapes/>

Valor inicial v^*



Ventajas y Desventajas de las Redes Neuronales

- **Ventaja:** Frontera de decisión no lineal
- **Desventajas:**
 - Convergencia muy lenta
 - Muchos óptimos locales, lo cual hace el método muy inestable
 - La mejor estructura de la red es desconocida
 - Difícil de manejar variables nominales



El paquete “nnet”

nnet {nnet}

R Documentation

Fit Neural Networks

Description

Fit single-hidden-layer neural network, possibly with skip-layer connections.

Usage

```
nnet(x, ...)

## S3 method for class 'formula'
nnet(formula, data, weights, ...,
     subset, na.action, contrasts = NULL)

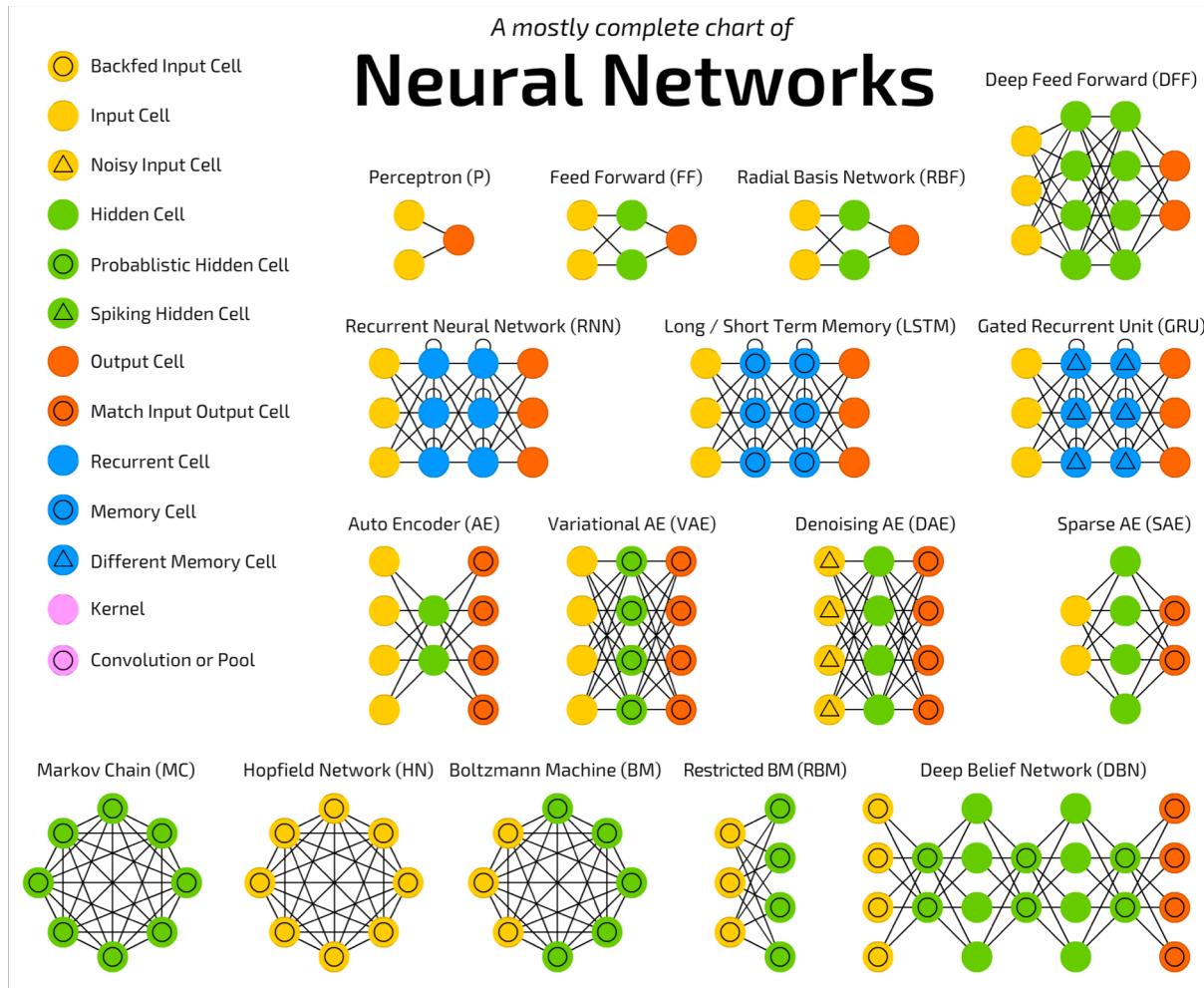
## Default S3 method:
nnet(x, y, weights, size, Wts, mask,
     linout = FALSE, entropy = FALSE, softmax = FALSE,
     censored = FALSE, skip = FALSE, rang = 0.7, decay = 0,
     maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000,
     abstol = 1.0e-4, reltol = 1.0e-8, ...)
```

Función de activación en “nnet”

Este es el código C++ de la Función de Activación del paquete “nnet” tipo Sigmoidea
(extracted from the C-sources; filennet.c):

```
static double
sigmoid(double sum)
{
    if (sum < -15.0)
        return (0.0);
    else if (sum > 15.0)
        return (1.0);
    else
        return (1.0 / (1.0 + exp(-sum)));
}
```

Tipos de Redes



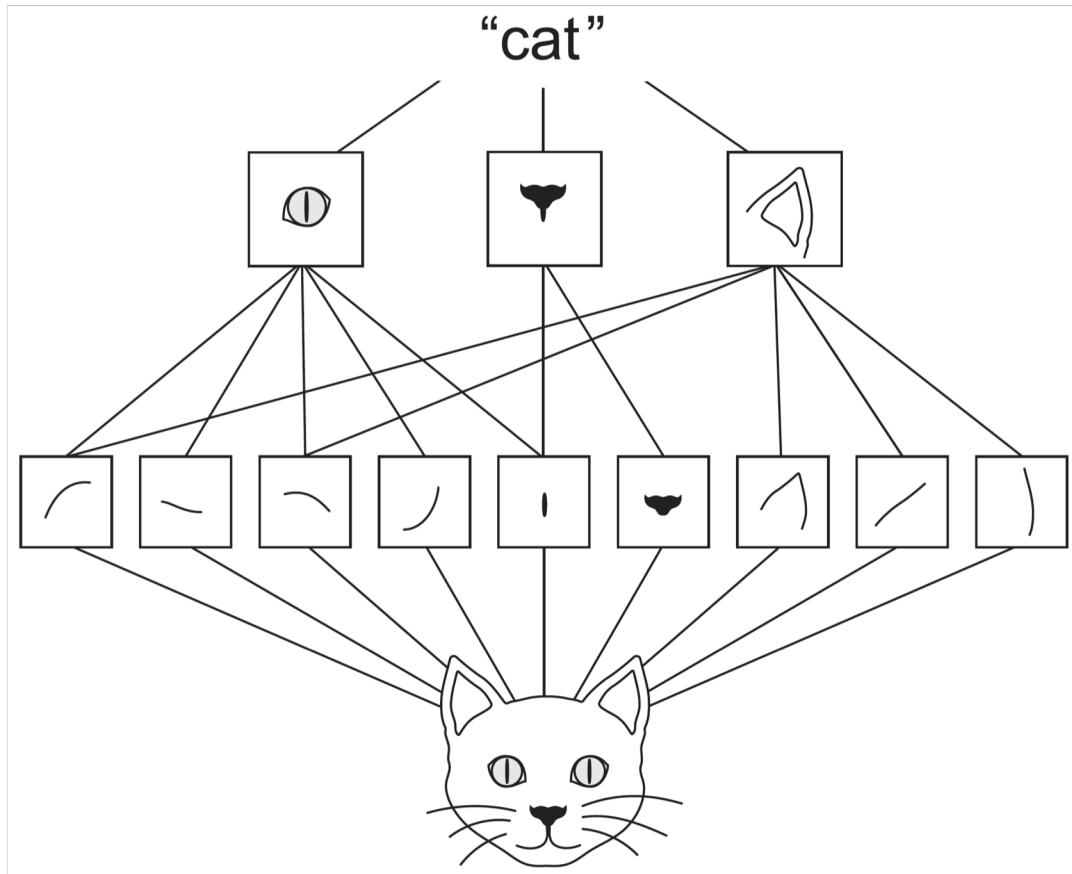
[Ver tabla completa](#)

Paquetes para Deep Learning

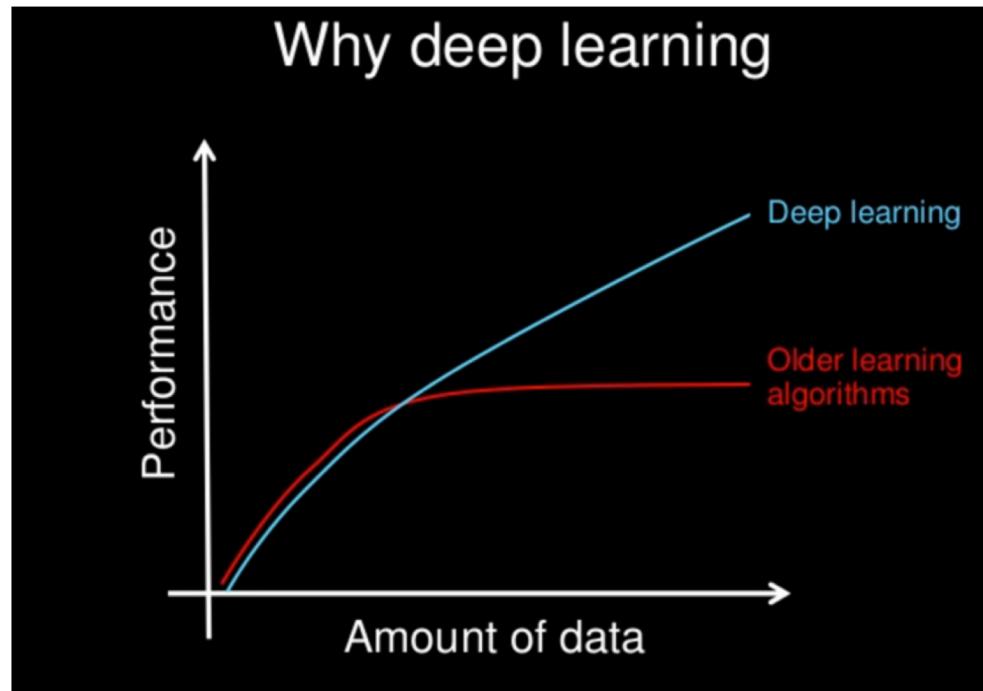
- **TensorFlow**: proporciona APIs para Python, C++, Haskell, Java, Go, Rust y también hay un paquete de terceros para R llamado tensorflow.
- **Keras**: es una API de alto nivel para construir y entrenar modelos de Deep Learning.
- Una de las bibliotecas de Python más poderosas y fáciles de usar para desarrollar y evaluar modelos de aprendizaje profundo es ***Keras***; Utiliza las bibliotecas de computación numérica Theano y TensorFlow.

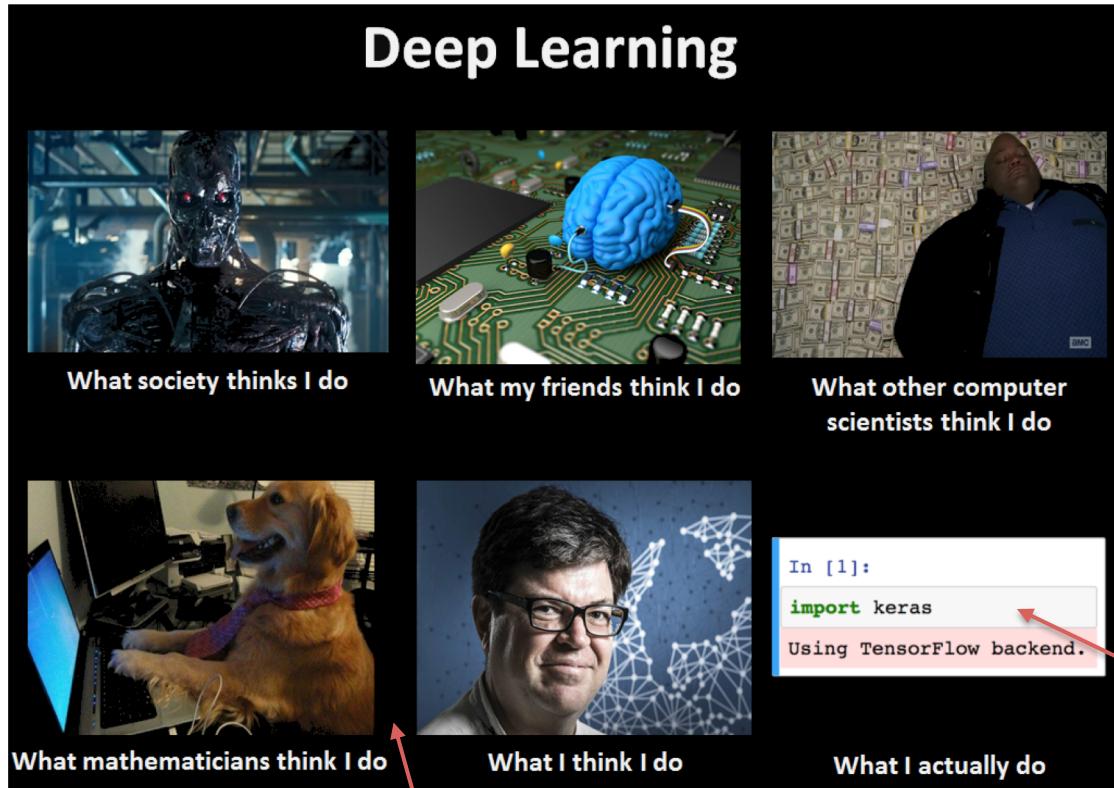
“Deep Learning”

- Conjunto de técnicas de aprendizaje computacional que involucran el uso de estructuras de redes con 2 o más capas ocultas.
- Cada capa tiene una tarea particular. Dividen el problema en pequeños problemas.



¿Por qué usar Deep Learning?





Yann LeCun. Jefe del equipo de Inteligencia Artificial de Facebook. Introduce el uso de Deep Learning para visión computacional.

Esto es en Python 😅.
En R usamos library(keras).

Variables Cualitativas

- Por defecto estos modelos no aceptan variables cualitativas.
- Lo correcto es pasarlas a *variables dummy* antes de aplicar los métodos.

neuralnet: Training of Neural Networks

Training of neural networks using backpropagation, resilient backpropagation with (Riedmiller, 1994) or without weight backtracking (Riedmiller and Braun, 1993) or the modified globally convergent version by Anastasiadis et al. (2005). The package allows flexible settings through custom-choice of error and activation function. Furthermore, the calculation of generalized weights (Intrator O & Intrator N, 1993) is implemented.

Version: 1.44.2
Depends: R (\geq 2.9.0)
Imports: grid, [MASS](#), grDevices, stats, utils, [Deriv](#)
Suggests: [testthat](#)
Published: 2019-02-07
Author: Stefan Fritsch [aut], Frauke Guenther [aut], Marvin N. Wright [aut, cre], Marc Suling [ctb], Sebastian M. Mueller [ctb]
Maintainer: Marvin N. Wright <wright at leibniz-bips.de>
BugReports: <https://github.com/bips-hb/neuralnet/issues>
License: [GPL-2](#) | [GPL-3](#) [expanded from: GPL (\geq 2)]
URL: <https://github.com/bips-hb/neuralnet>
NeedsCompilation: no
Materials: [NEWS](#)
CRAN checks: [neuralnet results](#)

Downloads:

Reference manual: [neuralnet.pdf](#)
Package source: [neuralnet_1.44.2.tar.gz](#)
Windows binaries: r-devel: [neuralnet_1.44.2.zip](#), r-release: [neuralnet_1.44.2.zip](#), r-oldrel: [neuralnet_1.44.2.zip](#)
OS X binaries: r-release: [neuralnet_1.44.2.tgz](#), r-oldrel: [neuralnet_1.44.2.tgz](#)
Old sources: [neuralnet archive](#)

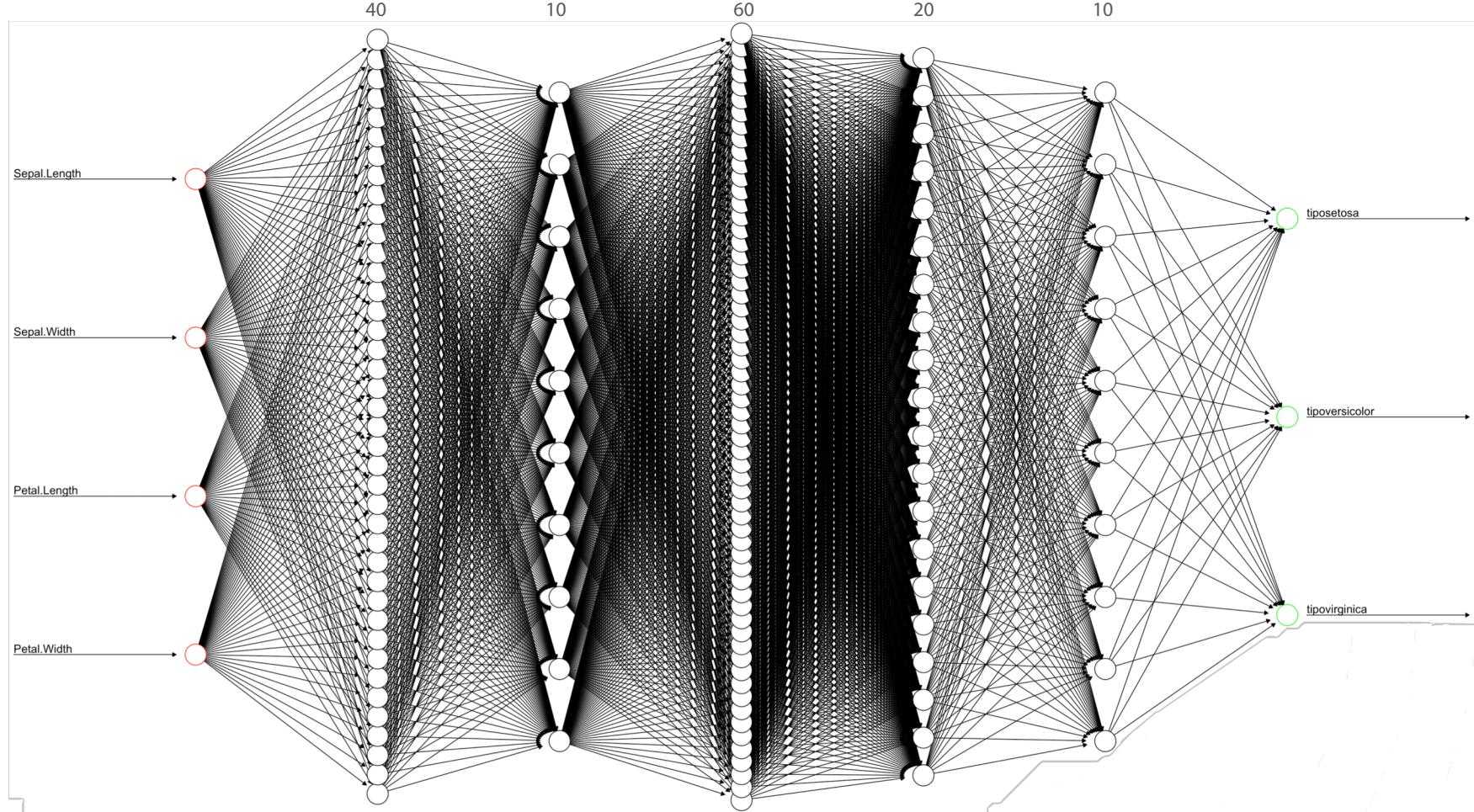
Reverse dependencies:

Reverse depends: [quarrint](#)
Reverse imports: [Modeler](#), [nnfor](#), [predictoR](#), [RTransProb](#), [spartan](#), [trackdem](#), [TrafficBDE](#)
Reverse suggests: [fscaret](#), [mlr](#), [NeuralNetTools](#), [nnetpredint](#), [plotmo](#), [shipunov](#)

Linking:

Please use the canonical form <https://CRAN.R-project.org/package=neuralnet> to link to this page.

Con el paquete neuralnet



Gracias....



oldemar rodíguez
CONSULTOR en M1NER14 D8 D4T0S