HUB IQ

Inteligencia Artificial & Machine Learning

Applicaciones en movilidad

Dr. Iván S. Razo Zapata

EngineeringX
Founded by the Royal Academy of Engineering and Lloyd's Register Foundation

Aprendizaje Supervisado

Redes Neuronales Artificiales

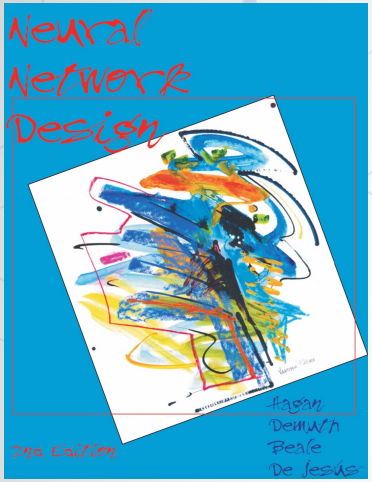
Recap

- Regresión Lineal
- Regresión Logística
- KNN
  - Regresión
  - Clasificación

Redes Neuronales Artificiales (RNAs)

Artificial Neural Networks (ANNs)

RNAs



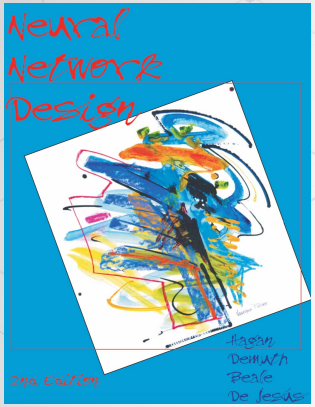
Demuth, H. B., Beale, M. H., De Jesús, O., & Hagan, M. T. (2014). *Neural network design*. Martin Hagan.

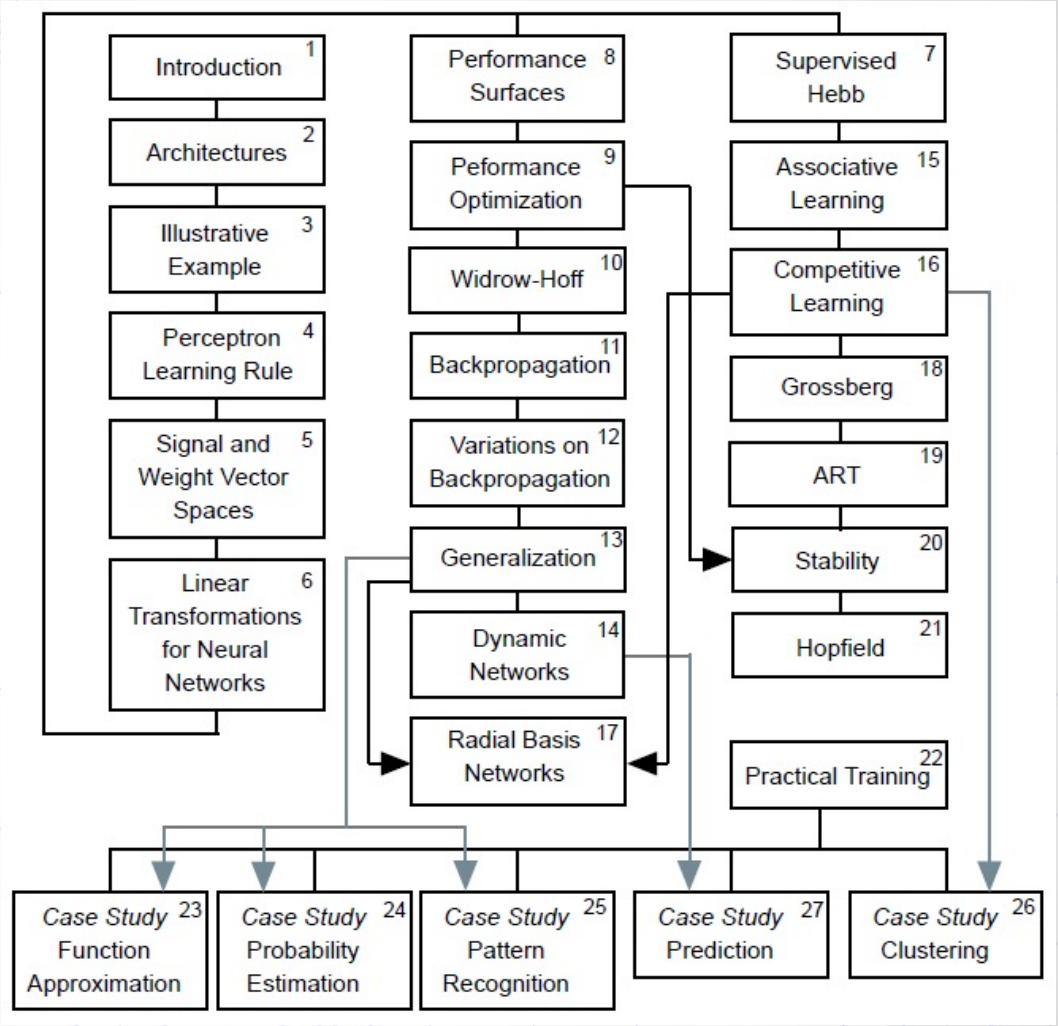
Obtaining the Book

A free 1012 page eBook version of the book (11.4 MB PDF) can be downloaded from here. A somewhat condensed 800 page paperback edition of the book can be ordered from Amazon.

https://hagan.okstate.edu/nnd.html

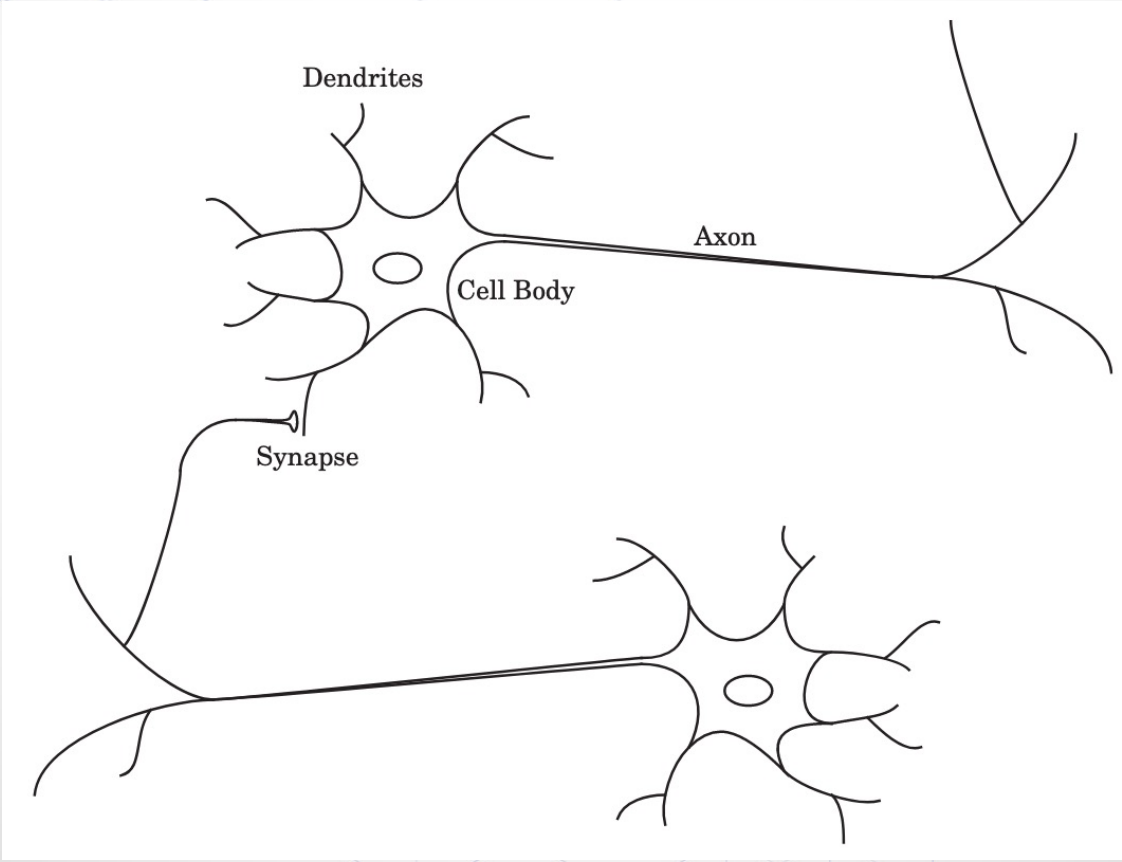
RNAs


Neural
Network
Design


Introduction 1
Architectures 2
Illustrative Example 3
Perceptron Learning Rule 4
Signal and Weight Vector Spaces 5
Linear Transformations for Neural Networks 6
Performance Surfaces 8
Peformance Optimization 9
Widrow-Hoff 10
Backpropagation 11
Variations on Backpropagation 12
Generalization 13
Dynamic Networks 14
Radial Basis Networks 17
Supervised Hebb 7
Associative Learning 15
Competitive Learning 16
Grossberg 18
ART 19
Stability 20
Hopfield 21
Practical Training 22
Case Study Function Approximation 23
Case Study Probability Estimation 24
Case Study Pattern Recognition 25
Case Study Prediction 27
Case Study Clustering 26

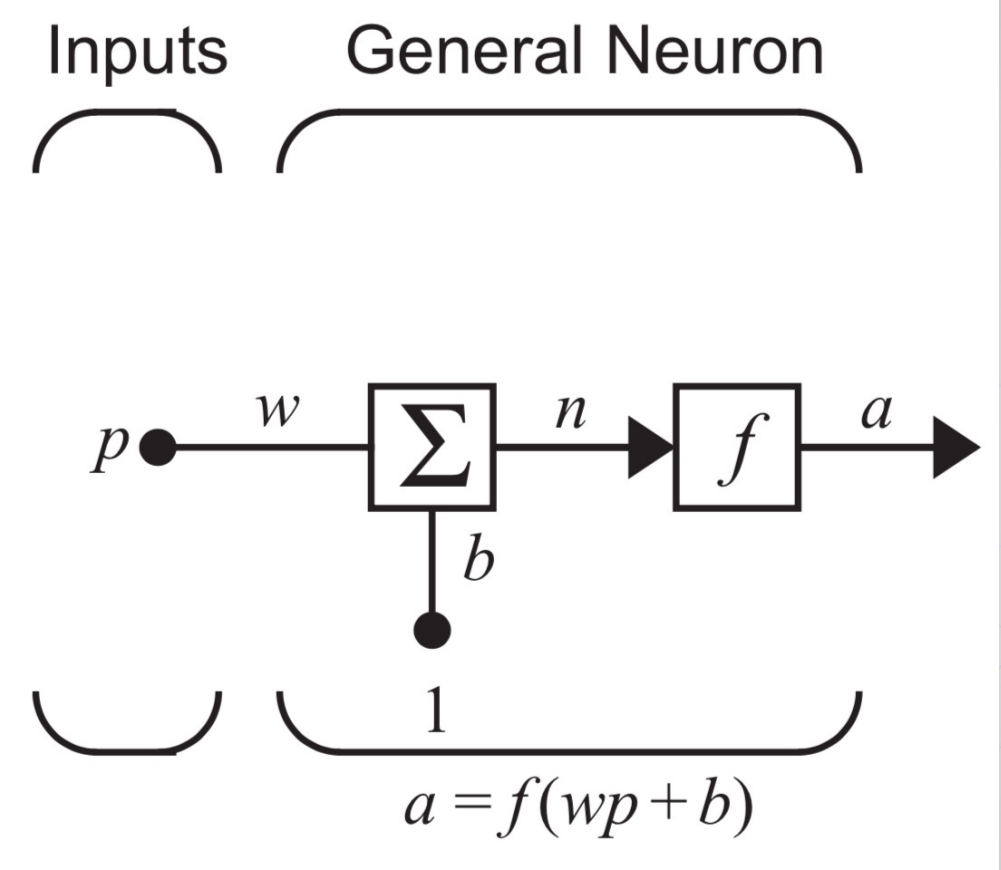
Representación de una neurona biológica

- Inspiradas en los componentes básicos de las redes neuronales biológicas



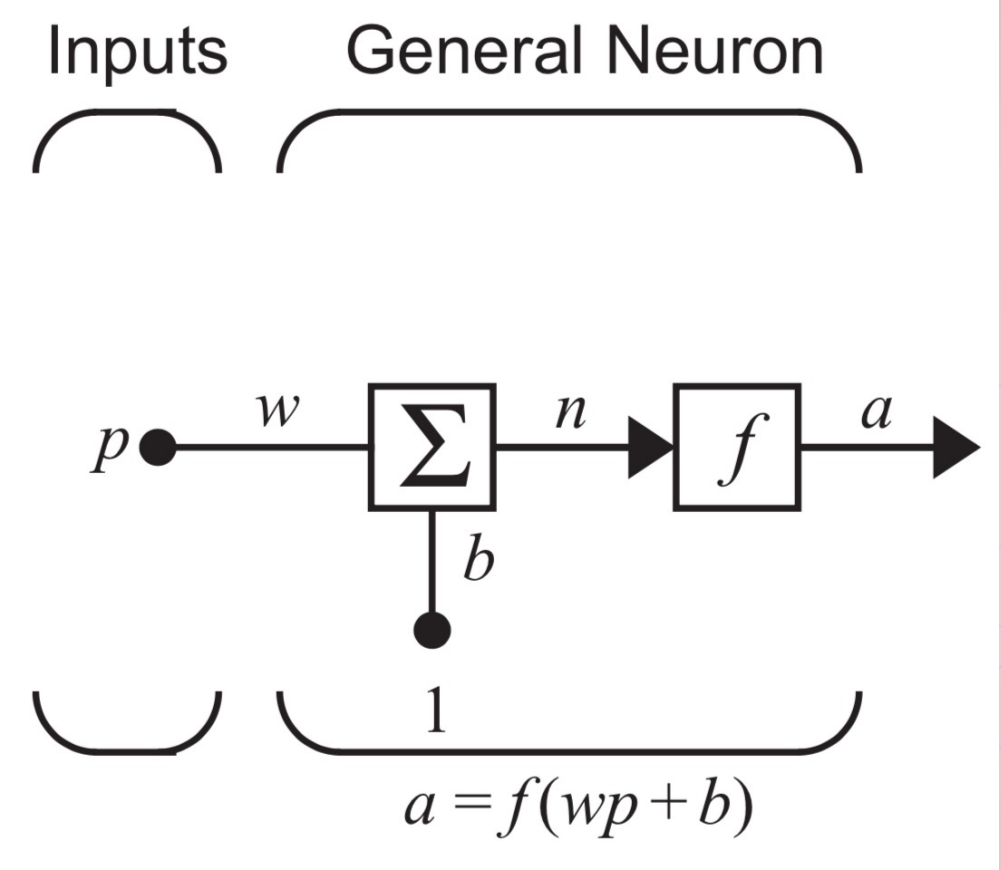
Representación de una neurona artificial

- p - valor de entrada (p.ej., temperatura)
- w - peso
- b – sesgo (bias)
- n - salida del sumador
- f - función de activación / transferencia
- a - salida de la neurona
- Nota: p, w, n, b y a son todos escalares

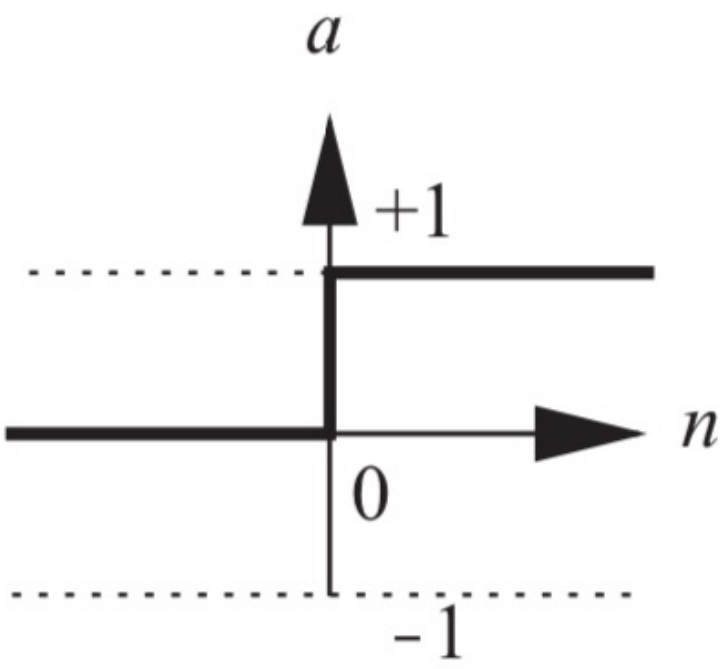


Calculando salida de una neurona

- p = 5
- w = 4
- b = -2.5
- n = ?
- f = ?
- a = ?



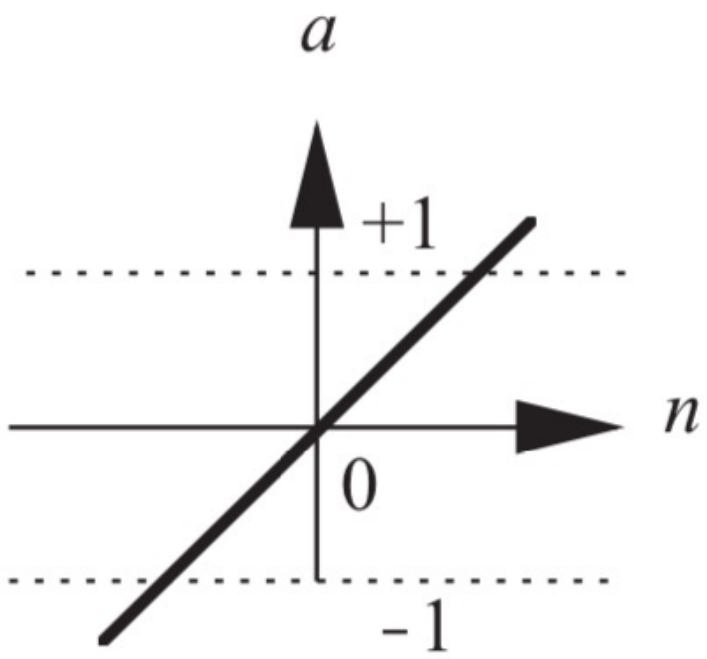
Algunas funciones de transferencia



$$a = hardlim(n)$$

Hard Limit Transfer Function

Algunas funciones de transferencia

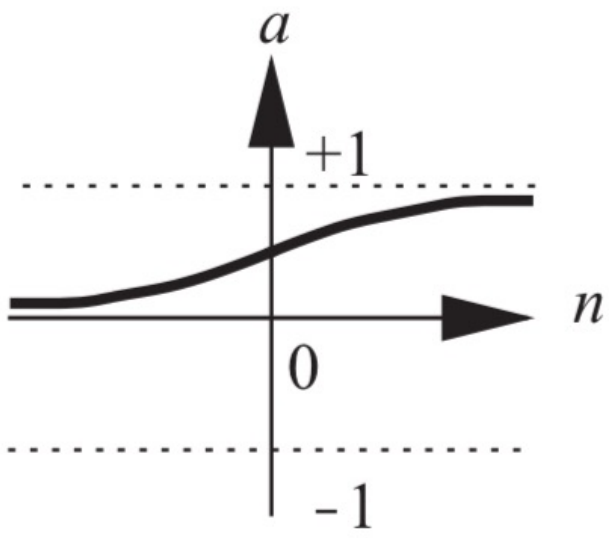


$$a = purelin(n)$$

Linear Transfer Function

Algunas funciones de transferencia

$$a = \frac{1}{1 + e^{-n}}$$

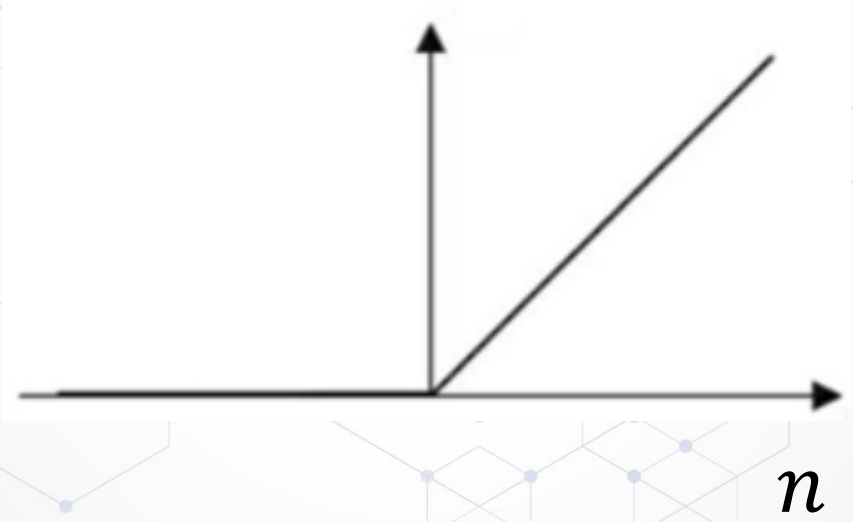


$a = logsig(n)$

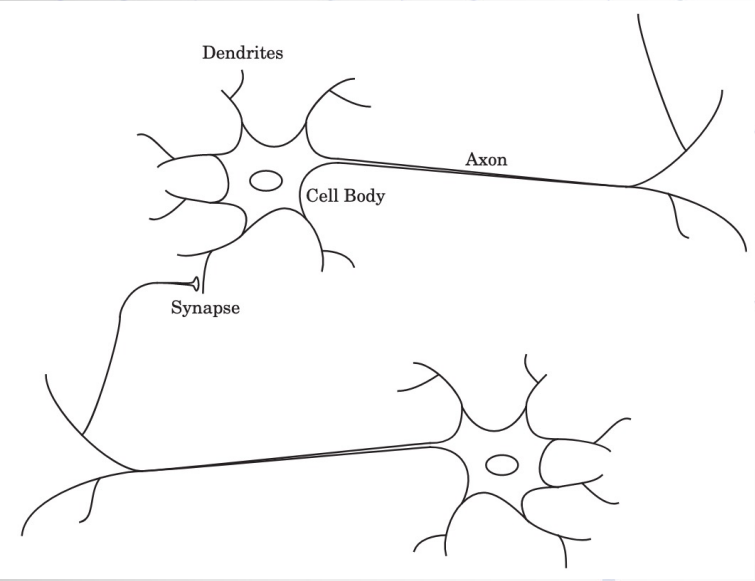
Log-Sigmoid Transfer Function

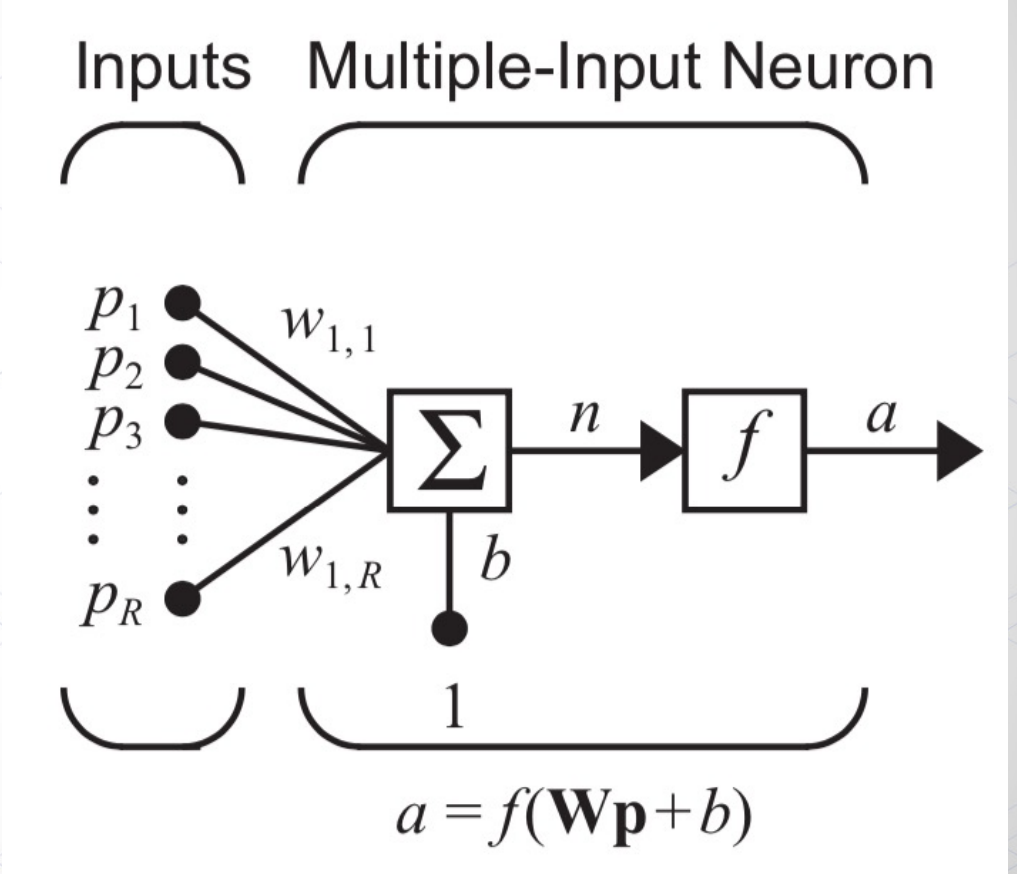
Rectified Linear Unit (ReLU)

$$a = max(0, n)$$



Neurona con multiples entradas





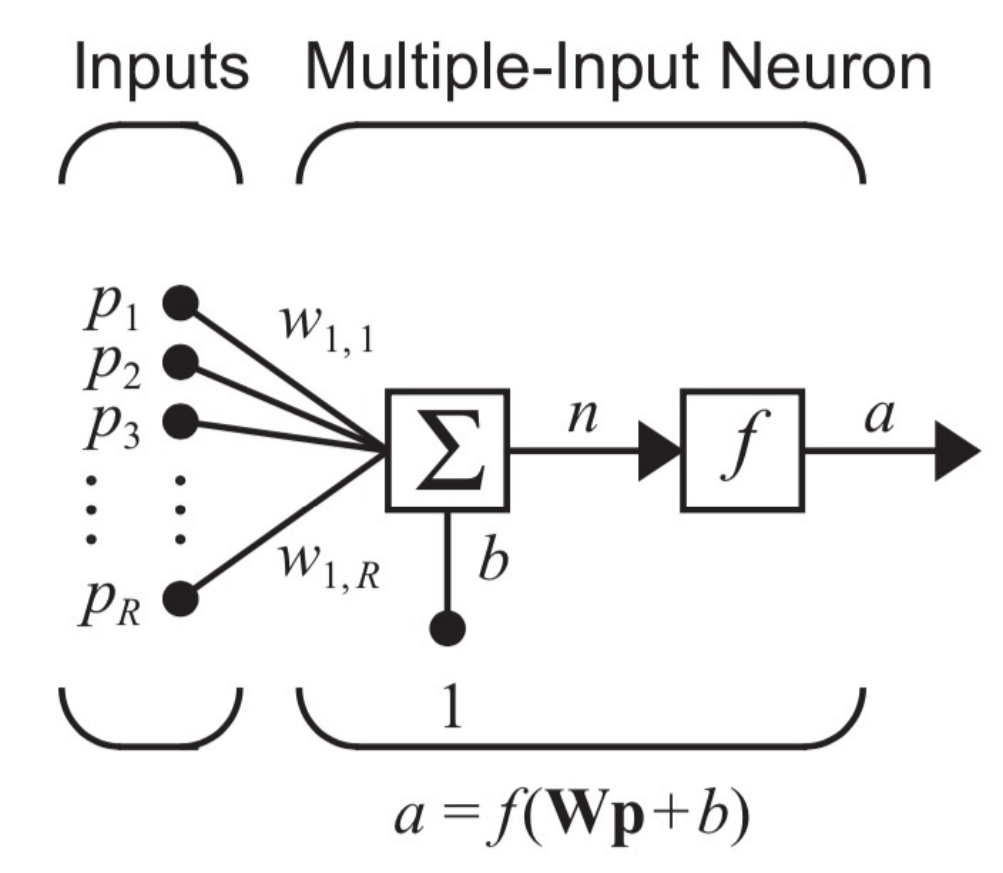
Neurona con multiples entradas

R valores de entrada

$$p_1, p_2, \ldots, p_R$$

R pesos para cada valor de entrada

$$w_{1,1}, w_{1,2}, \ldots, w_{1,R}$$



Neurona con multiples entradas

$w_{1,R}$ Peso hacia la neurona uno (1) proveniente de la entrada R

1 Destino

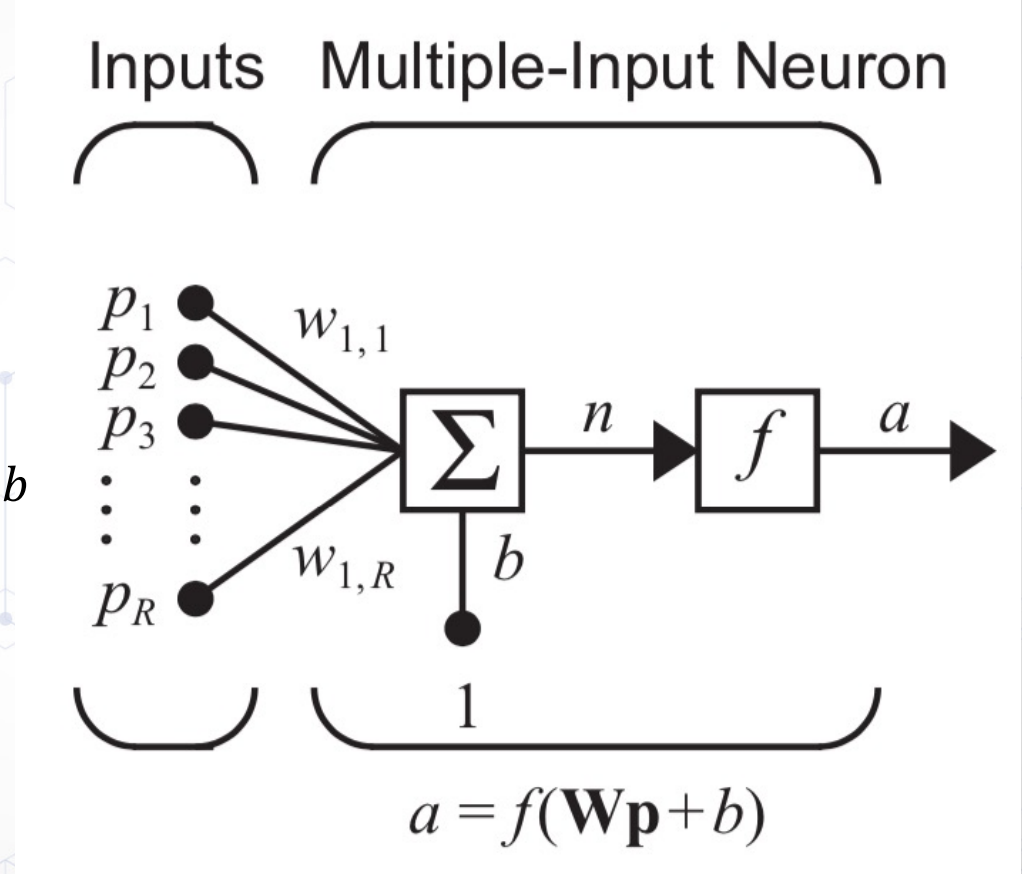
R Fuente

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \cdots + w_{1,R}p_R + b$$

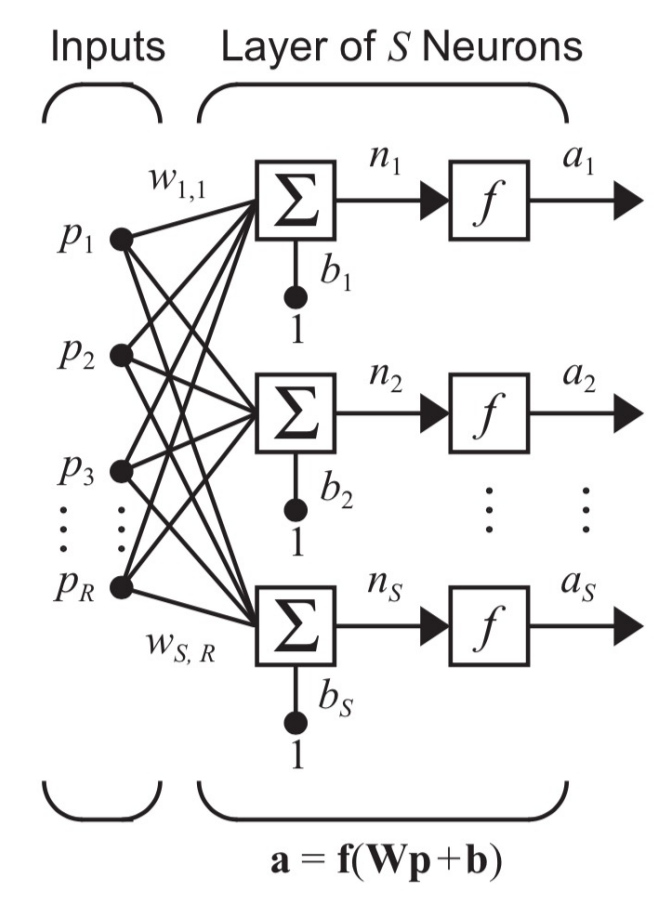
$$n = Wp + b$$

$W =$ *Matriz de pesos*

$p =$ *Vector de entradas*



Una capa de neuronas

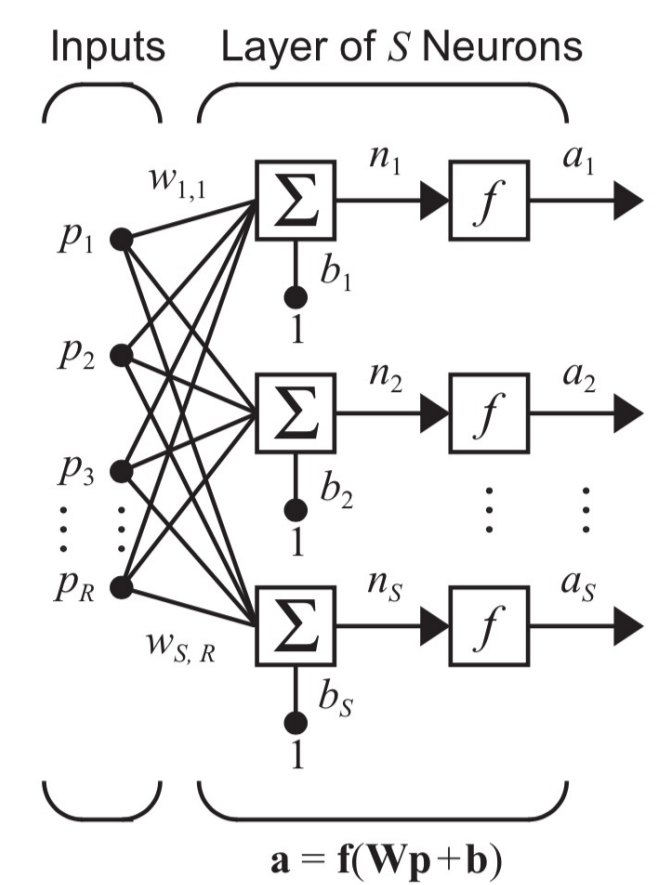


Usualmente, todas las entradas están "conectadas" a todas la neuronas

$$w_{S,R} = ?$$

Destino, Fuente

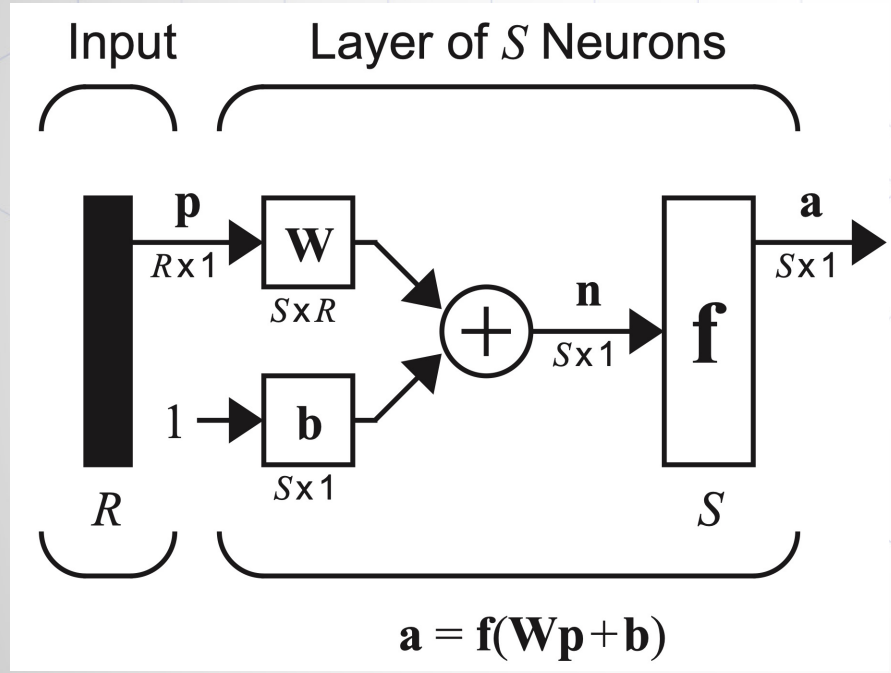
Una capa de neuronas



$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_S \end{bmatrix}$$

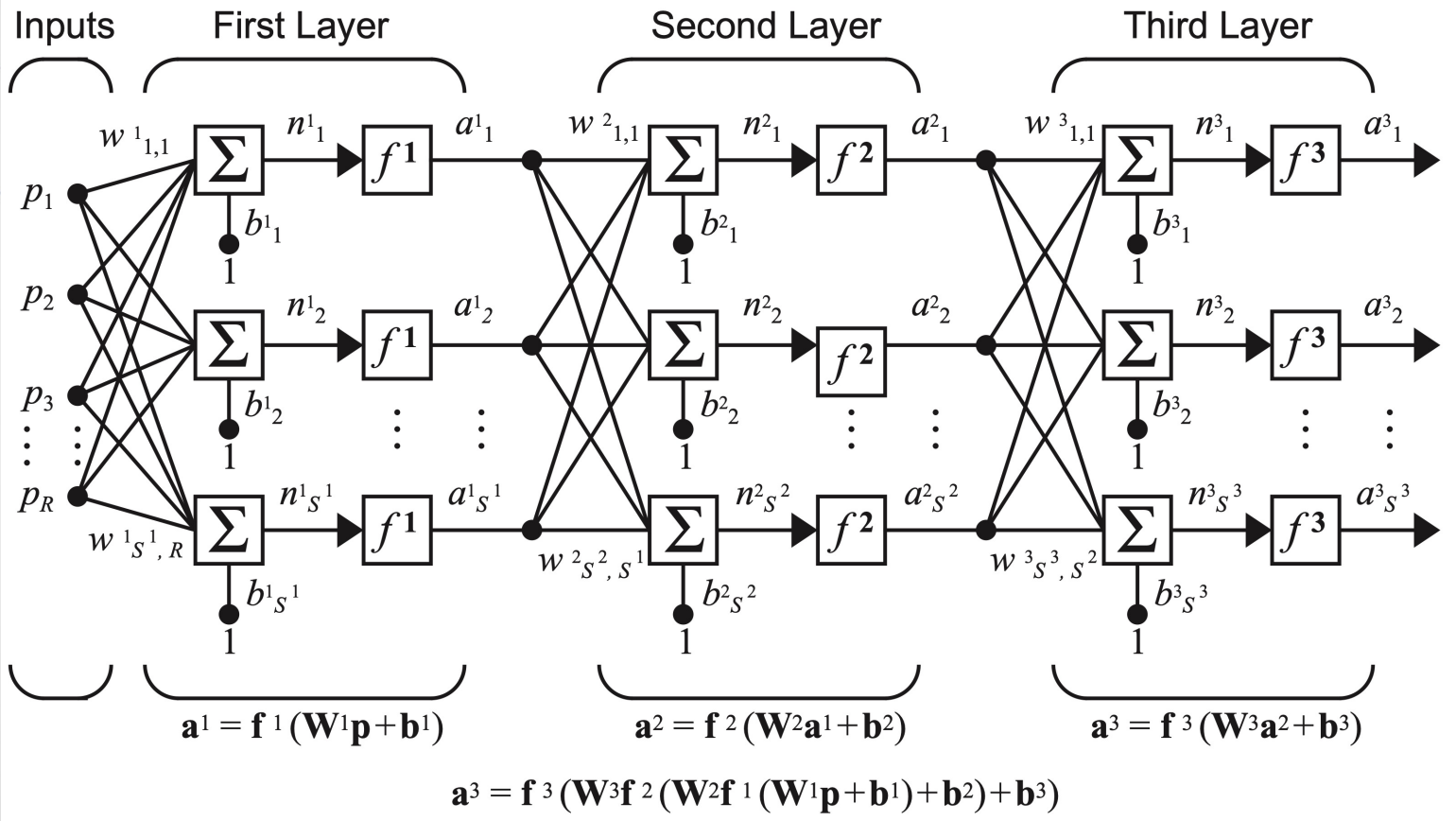
Una capa de neuronas



$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

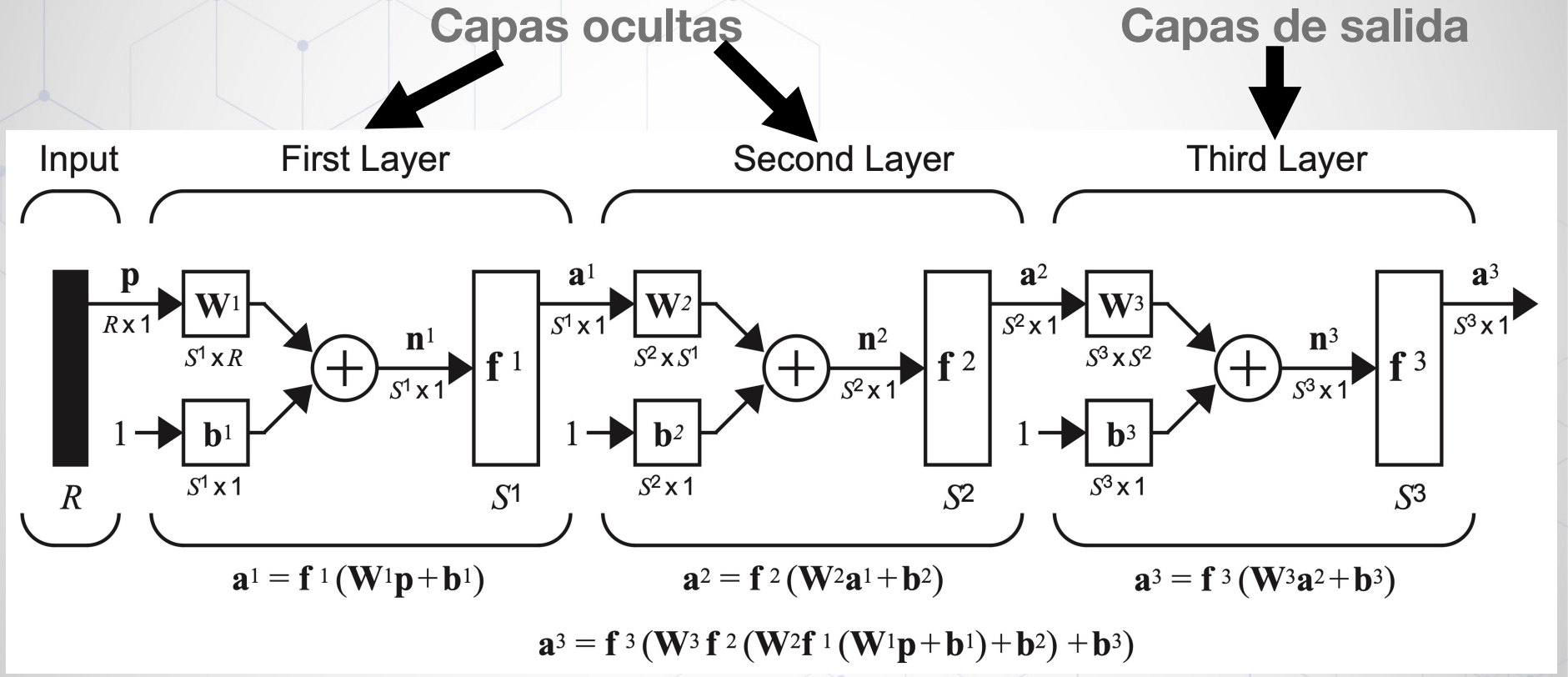
$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_S \end{bmatrix}$$

Varias capas de neuronas - Arquitectura Feedforward



$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

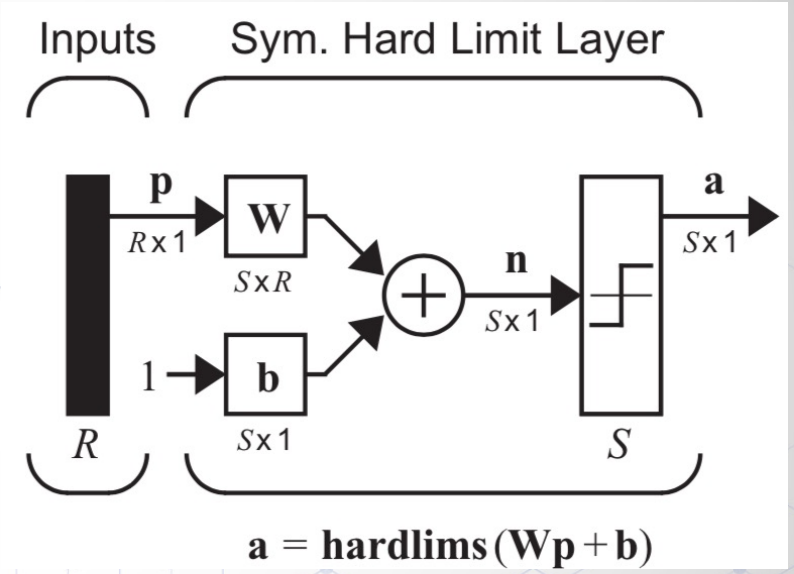
Varias capas de neuronas



Perceptrón y Modelo de Neurona Simple

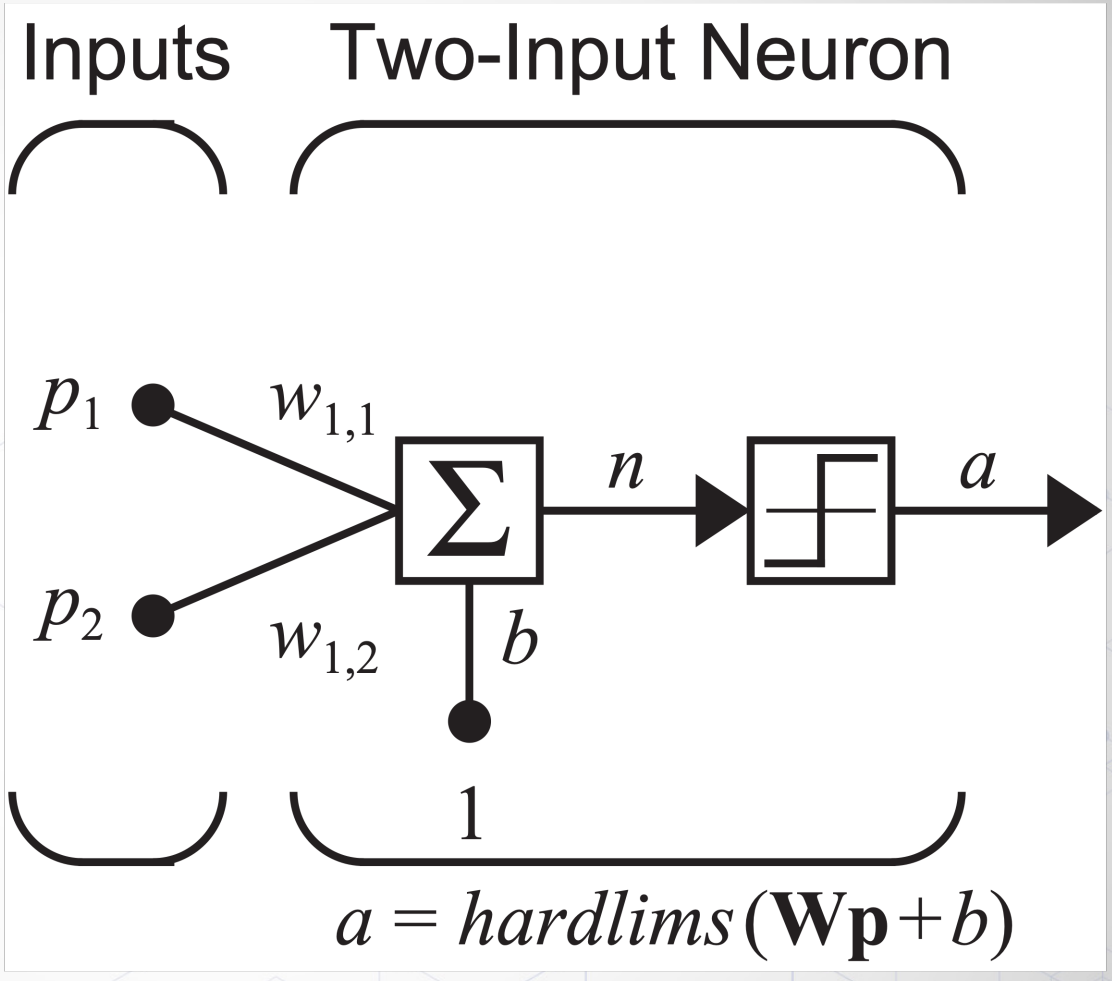
Perceptrón

- En 1943, Warren McCulloch y Walter Pitts presentan uno de los primeros modelos de neuronas artificiales
- Al final de la década de 1950s, Frank Rosenblatt junto con otros investigadores desarrollaron una clase de red neuronal a la que llamaron **perceptrón**



Perceptrón

- Dos valores de entrada

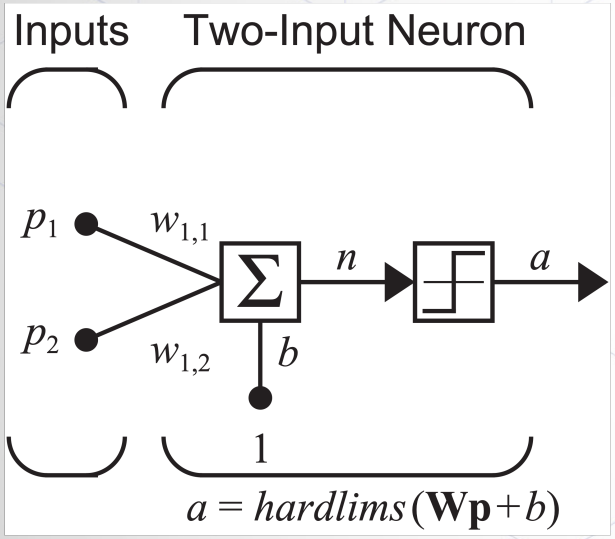


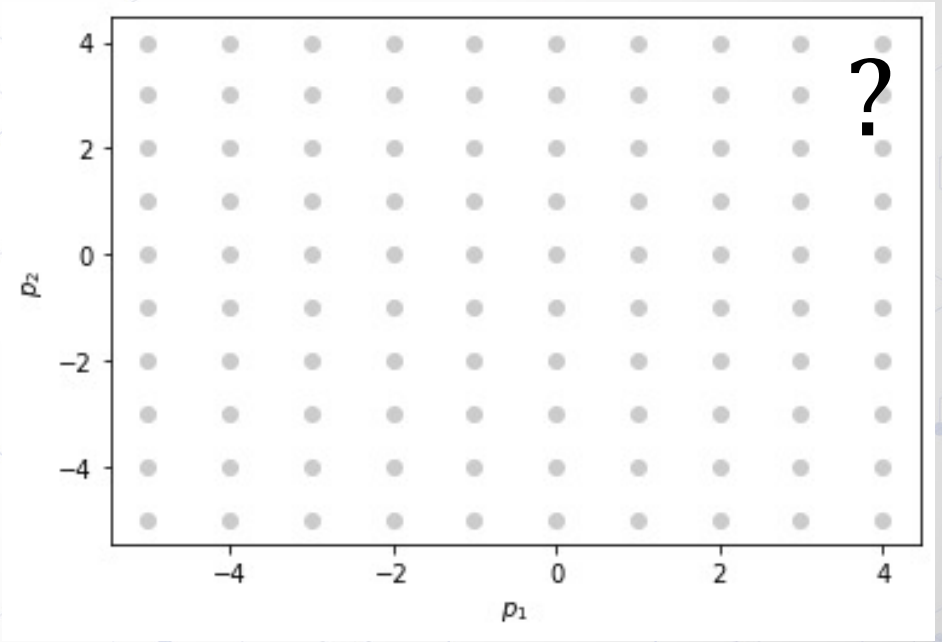
Perceptrón

- Dos valores de entrada

$$W = [-1, 1]$$

$$a = hardlims(\mathbf{W}\mathbf{p} + b)$$



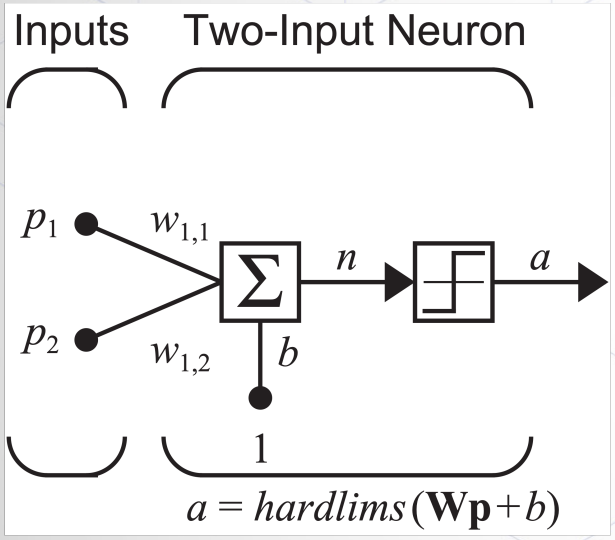


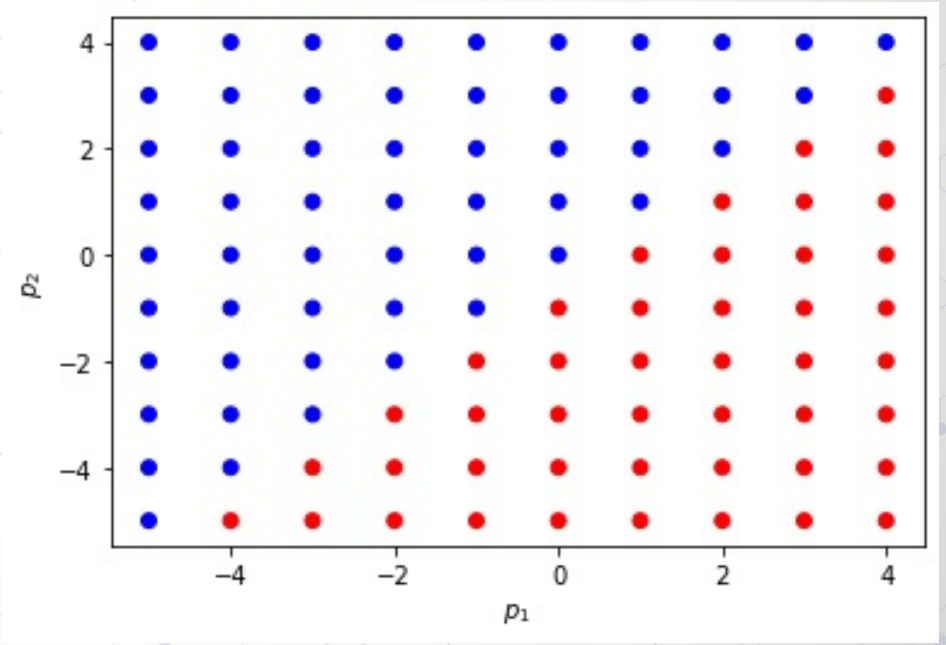
Perceptrón

- Dos valores de entrada

$$W = [-1, 1]$$

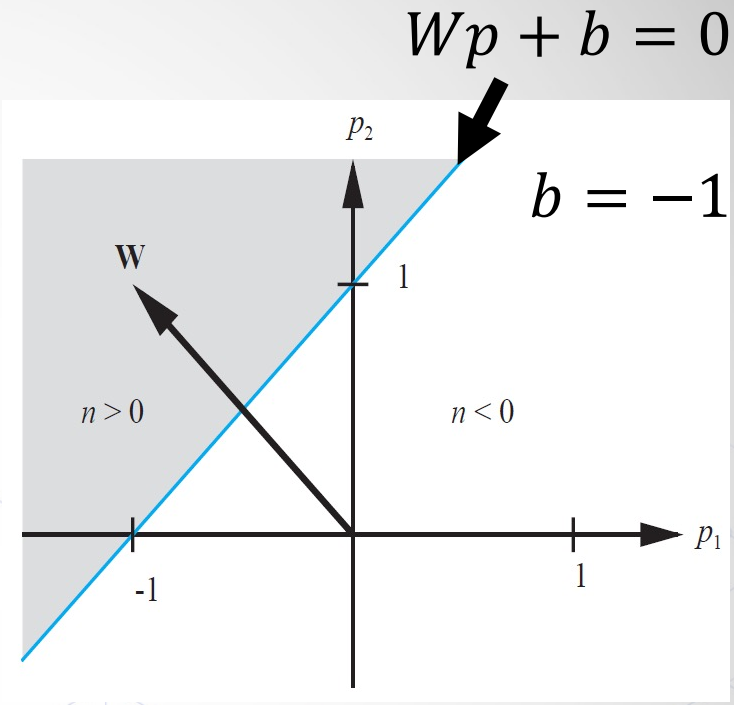
$$a = hardlims(\mathbf{W}\mathbf{p} + b)$$

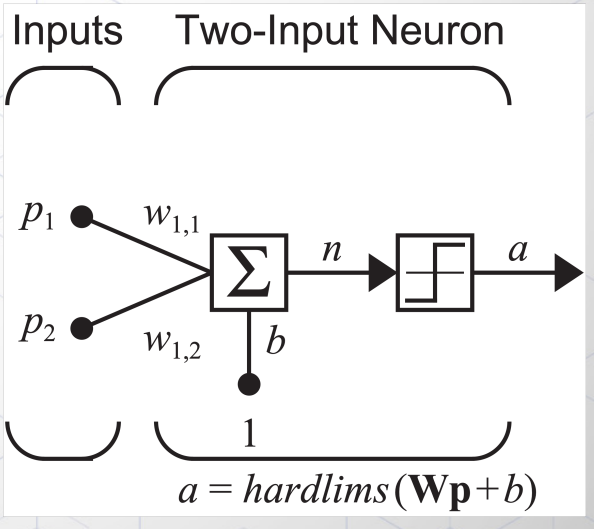




Perceptrón

- El vector de pesos (W) es ortogonal a la frontera de decisión
- El perceptron (con una sola neurona) puede clasificar "vectores" en dos categorías
- Si el producto punto (interno) entre el vector de pesos W y el vector de entrada p es mayor o igual –b, la salida es 1. $Wp \geq -b \rightarrow 1$
- En caso contrario la salida es cero. $Wp < -b \rightarrow 0$
- Esto divide el espacio en dos partes

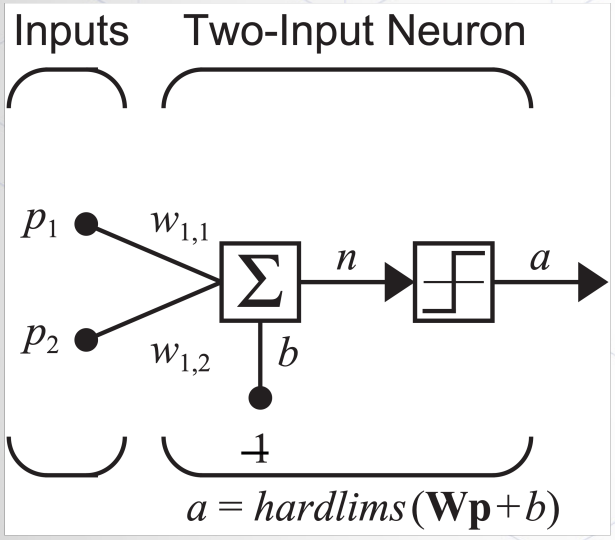


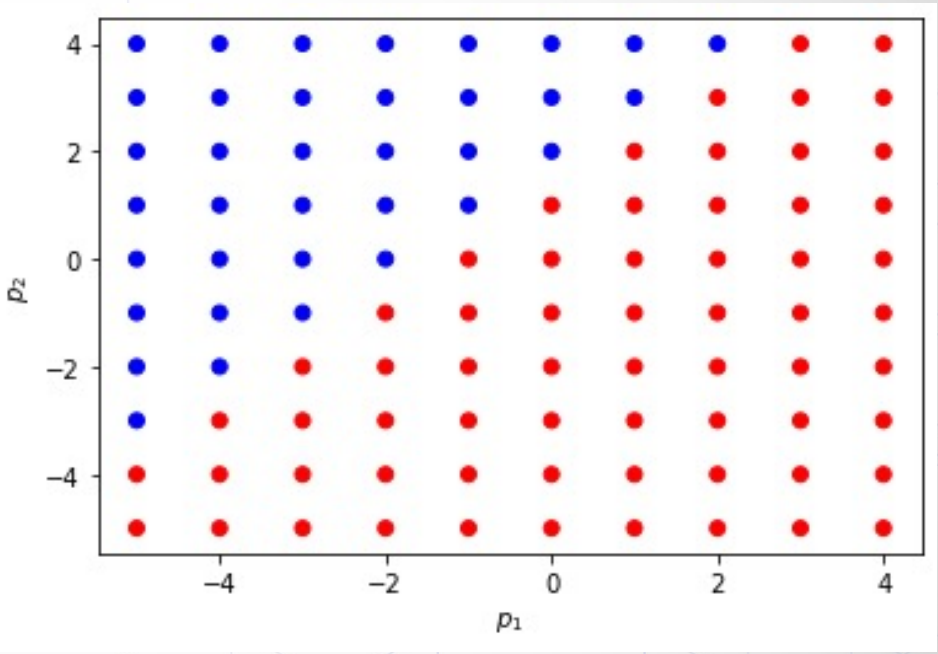


Perceptrón

- Dos valores de entrada

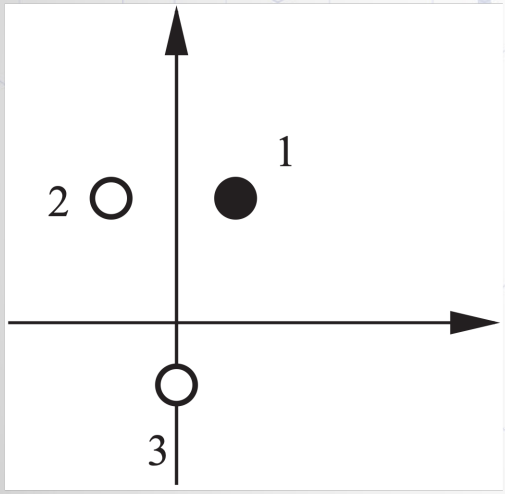
$$b = -1$$

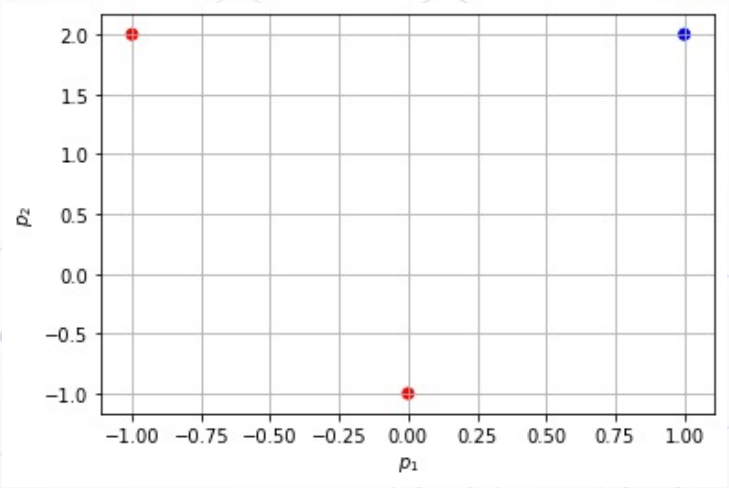


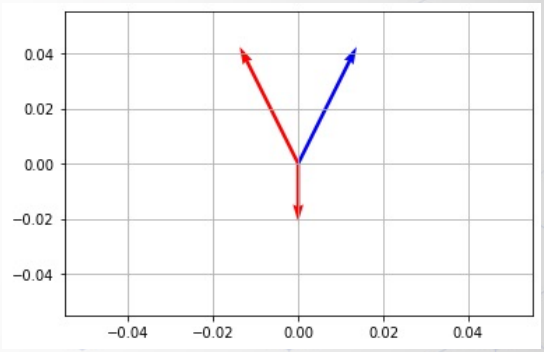


Ejemplo

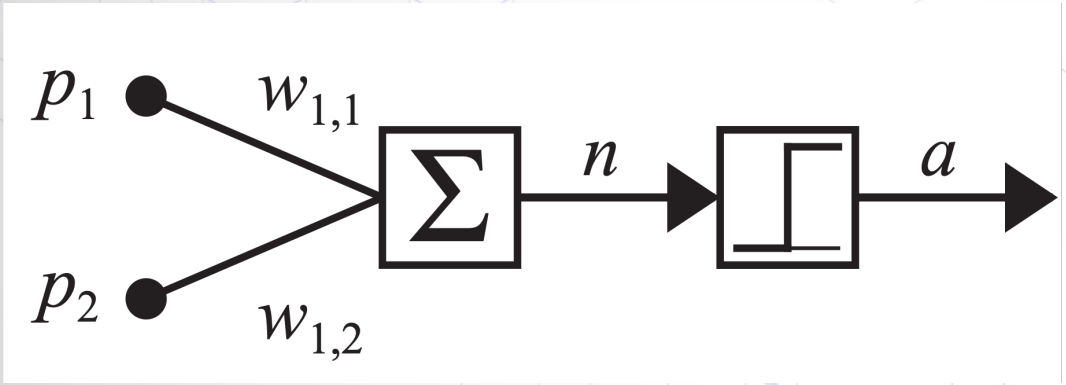
$$\left\{\mathbf{p}_1 = \begin{bmatrix}1\\2\end{bmatrix}, t_1 = 1\right\}\left\{\mathbf{p}_2 = \begin{bmatrix}-1\\2\end{bmatrix}, t_2 = 0\right\}\left\{\mathbf{p}_3 = \begin{bmatrix}0\\-1\end{bmatrix}, t_3 = 0\right\}$$



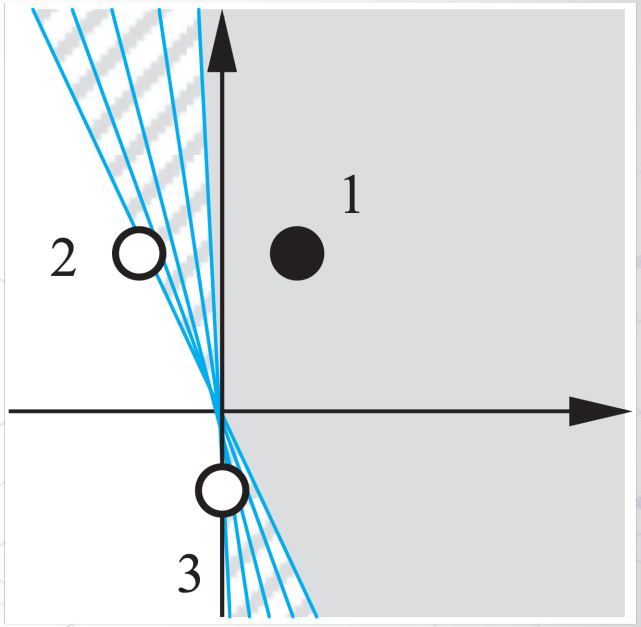




Ejemplo

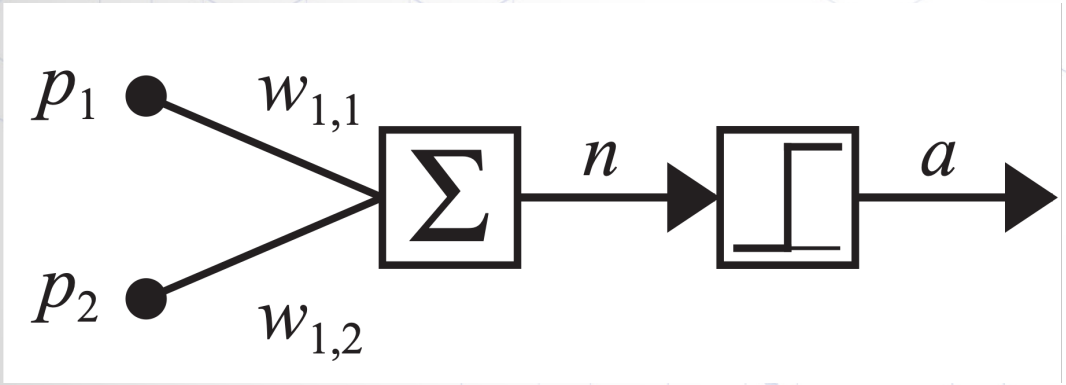


$$a = hardlim(\mathbf{Wp})$$

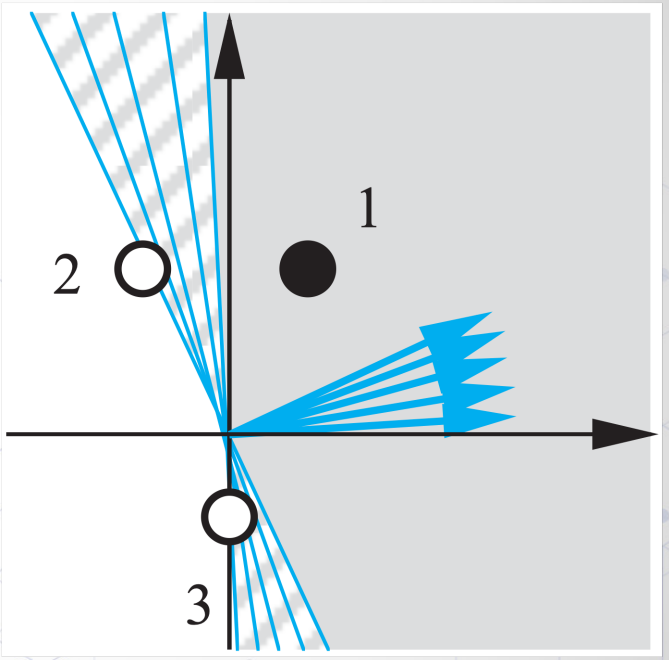


Presentar cada punto de manera iterativa

Ejemplo



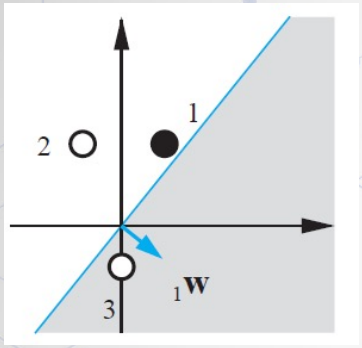
$$a = hardlim(\mathbf{Wp})$$



Presentar cada punto de manera iterativa

Ejemplo – Primera época

$$p_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$



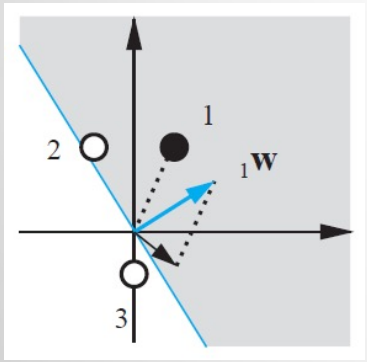
1. Inicializar

$$W^T = [1.0, -0.8]$$

2. Observar

$$a = hardlim(W^T p_1) = \text{hardlim}\left([1.0, -0.8]\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right)$$

$$a = hardlim(-0.6) = 0$$



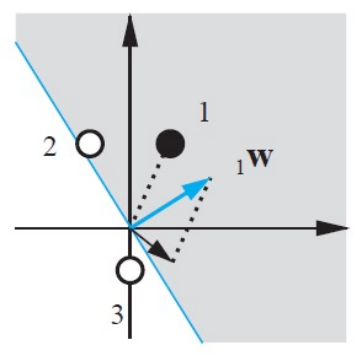
3. Actualizar

$$Si\ t == 1\ y\ a == 0, entonces\ W^{nuevo} = W^{anterior} + p$$

$$W^{nuevo} = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix}$$

Ejemplo – Primera época

$$p_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$



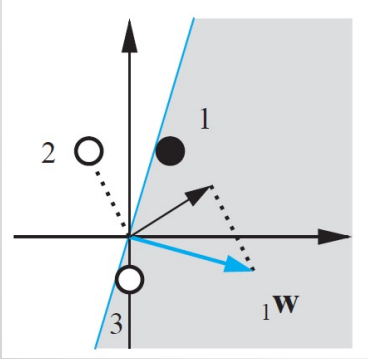
- W

$$W^T = [2.0, 1.2]$$

- Observar

$$a = hardlim(W^T p_2) = \text{hardlim}\left([2.0, 1.2]\begin{bmatrix} -1 \\ 2 \end{bmatrix}\right)$$

$$a = hardlim(0.4) = 1$$



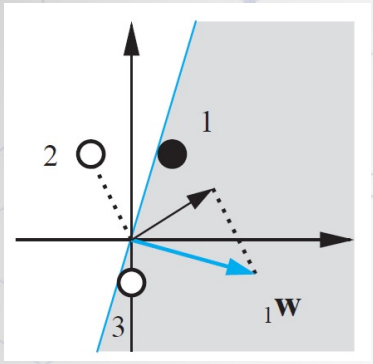
- Actualizar

$$Si\ t == 0\ y\ a == 1, entonces\ W^{nuevo} = W^{anterior} - p$$

$$W^{nuevo} = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix} - \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3.0 \\ -0.8 \end{bmatrix}$$

Ejemplo – Primera época

$$p_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$



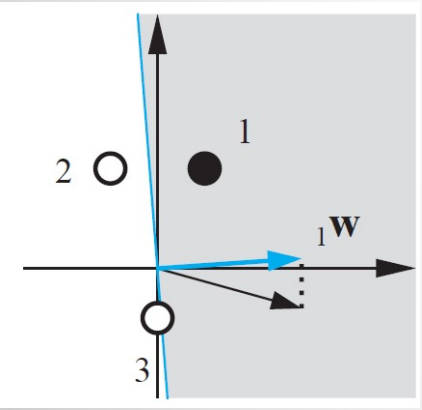
- W

$$W^T = [3.0, -0.8]$$

- Observar

$$a = hardlim(W^T p_3) = \text{hardlim}\left([3.0, -0.8]\begin{bmatrix} 0 \\ -1 \end{bmatrix}\right)$$

$$a = hardlim(0.8) = 1$$



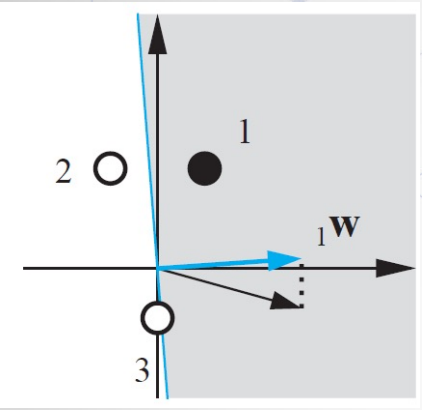
- Actualizar

$$Si\ t == 0\ y\ a == 1, entonces\ W^{nuevo} = W^{anterior} - p$$

$$W^{nuevo} = \begin{bmatrix} 3.0 \\ -0.8 \end{bmatrix} - \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 3.0 \\ 0.2 \end{bmatrix}$$

Ejemplo – Segunda época

$$p_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$



- W

$$W^T = [3.0, 0.2]$$

- Observar

$$a = hardlim(W^T p_1) = \text{hardlim}\left([3.0,0.2]\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right)$$

$$a = hardlim(3.4) = 1$$

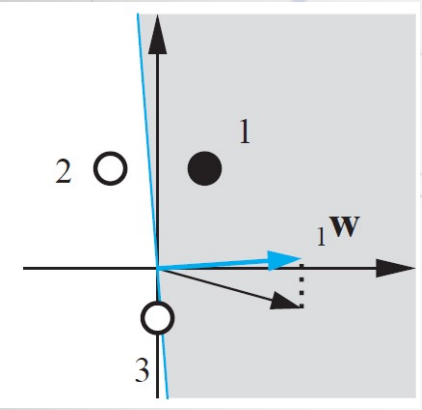
- Actualizar

$$Si\ t == a, entonces\ W^{nuevo} = W^{anterior}$$

$$W^{nuevo} = \begin{bmatrix} 3.0 \\ 0.2 \end{bmatrix}$$

Ejemplo – Segunda época

$$p_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$



- W

$$W^T = [3.0, 0.2]$$

- Observar

$$a = hardlim(W^T p_2) = \text{hardlim}\left([3.0,0.2]\begin{bmatrix} -1 \\ 2 \end{bmatrix}\right)$$

$$a = hardlim(-2.6) = 0$$

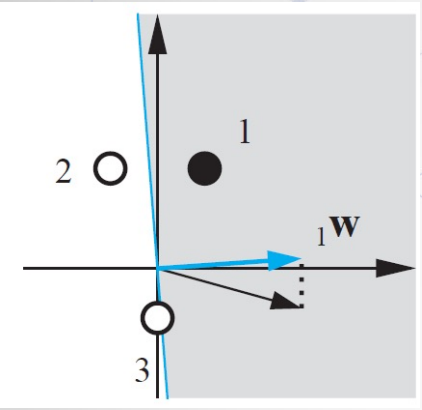
- Actualizar

$$Si\ t == a, entonces\ W^{nuevo} = W^{anterior}$$

$$W^{nuevo} = \begin{bmatrix} 3.0 \\ 0.2 \end{bmatrix}$$

Ejemplo – Segunda época

$$p_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$



- W

$$W^T = [3.0, 0.2]$$

- Observar

$$a = hardlim(W^T p_3) = \text{hardlim}\left([3.0,0.2]\begin{bmatrix} 0 \\ -1 \end{bmatrix}\right)$$

$$a = hardlim(-0.2) = 0$$

- Actualizar

$$Si\ t == a, entonces\ W^{nuevo} = W^{anterior}$$

$$W^{nuevo} = \begin{bmatrix} 3.0 \\ 0.2 \end{bmatrix}$$

Ejemplo

$$Si\ t == 1\ y\ a == 0, entonces\ W^{nuevo} = W^{anterior} + p$$

$$Si\ t == 0\ y\ a == 1, entonces\ W^{nuevo} = W^{anterior} - p$$

$$Si\ t == a, entonces\ W^{nuevo} = W^{anterior}$$

Ejemplo

$$e = t - a$$

$$Si\ e == 1, entonces\ W^{nuevo} = W^{anterior} + p$$

$$Si\ e == -1, entonces\ W^{nuevo} = W^{anterior} - p$$

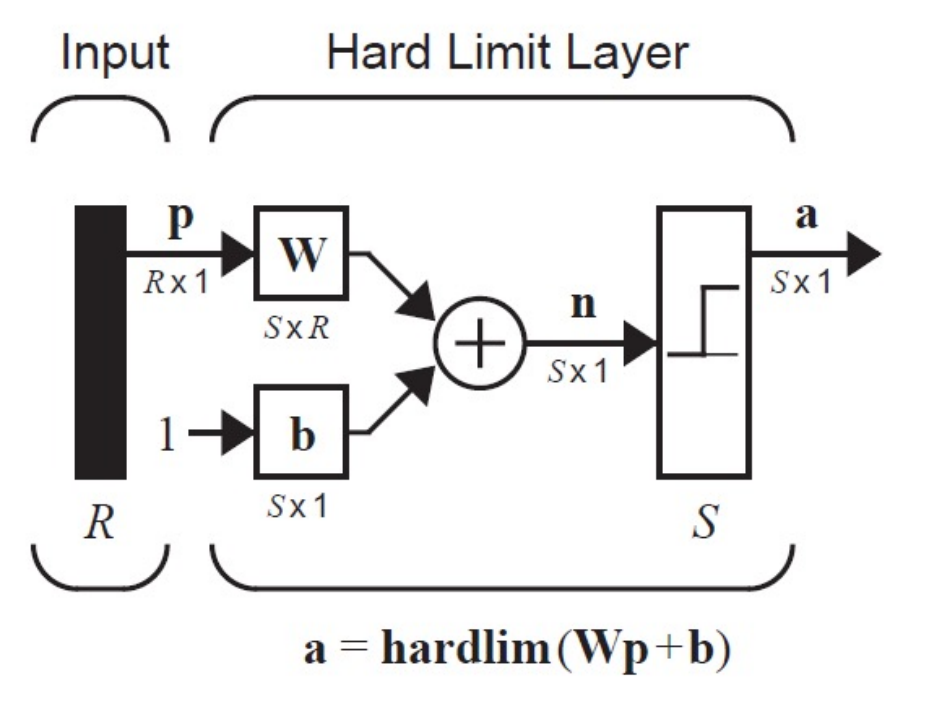
$$Si\ e == 0, entonces\ W^{nuevo} = W^{anterior}$$

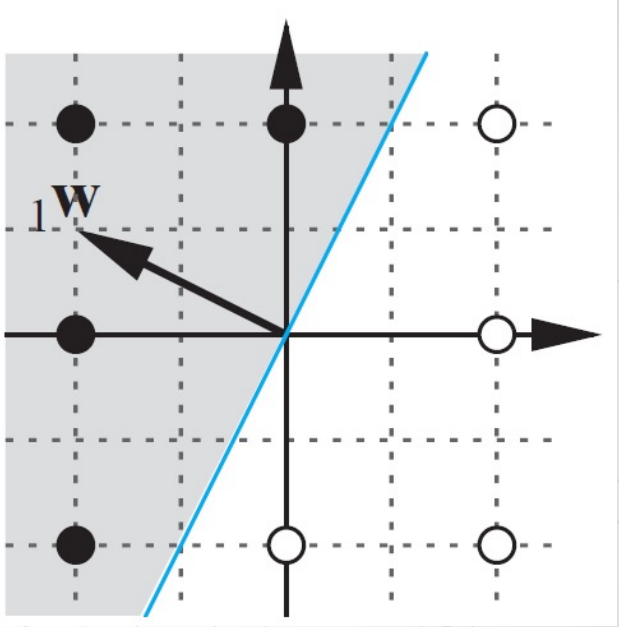
$$W^{nuevo} = W^{anterior} + ep = W^{anterior} + (t - a)p$$

$$b^{nuevo} = b^{anterior} + e$$

Código

Take home message



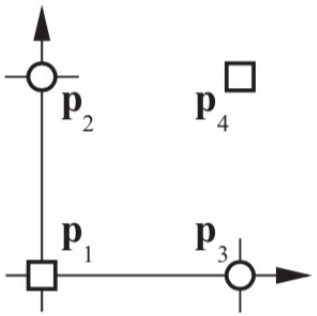


$$W^{nuevo} = W^{anterior} + ep = W^{anterior} + (t - a)p$$

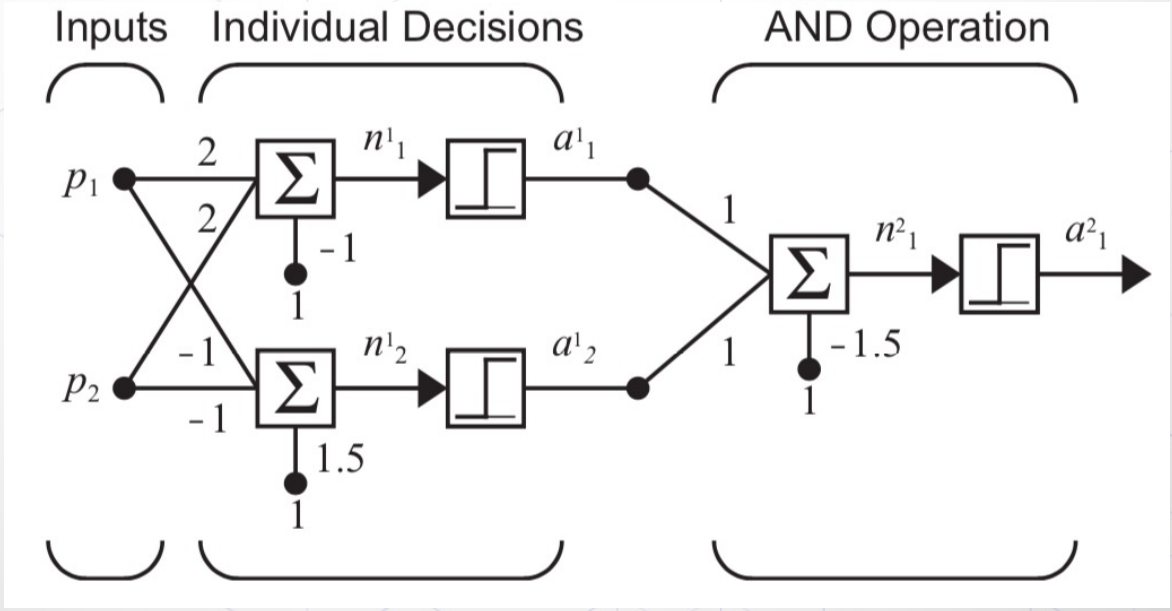
$$b^{nuevo} = b^{anterior} + e$$

Clases no linealmente separables

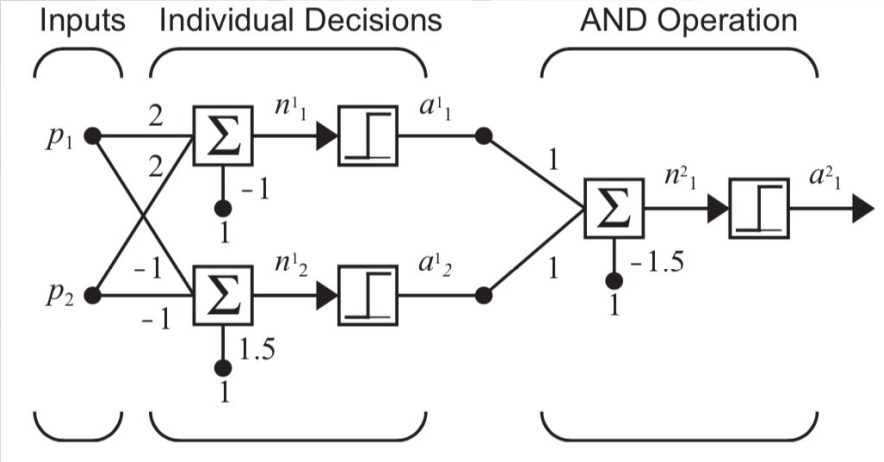
Clases no linealmente separables

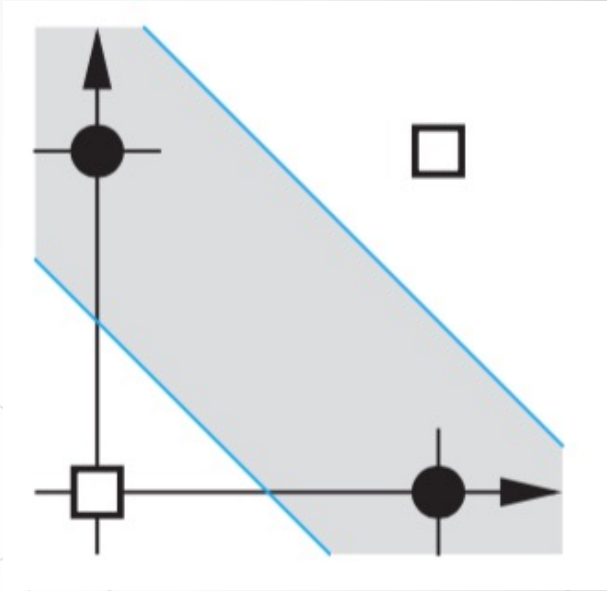


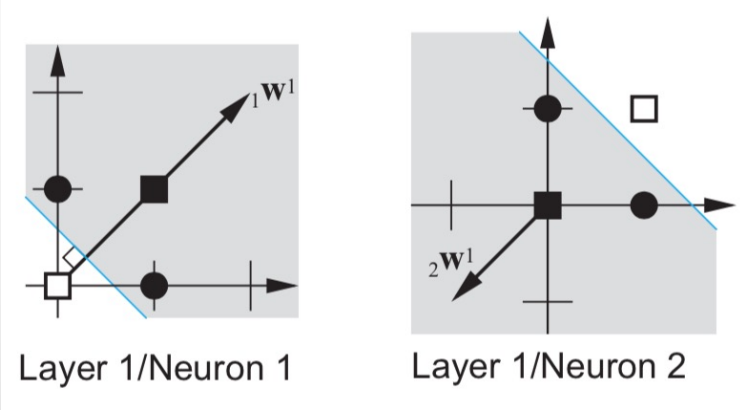
$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\0\end{bmatrix}, t_1 = 0\right\}\left\{\mathbf{p}_2 = \begin{bmatrix}0\\1\end{bmatrix}, t_2 = 1\right\}\left\{\mathbf{p}_3 = \begin{bmatrix}1\\0\end{bmatrix}, t_3 = 1\right\}\left\{\mathbf{p}_4 = \begin{bmatrix}1\\1\end{bmatrix}, t_4 = 0\right\}.$$



Clases no linealmente separables







Ajustando pesos (w) y bias (b)

$$W^{nuevo} = W^{anterior} + \alpha(t-a)p$$

$$W^{nuevo} = W^{anterior} + \alpha e p$$

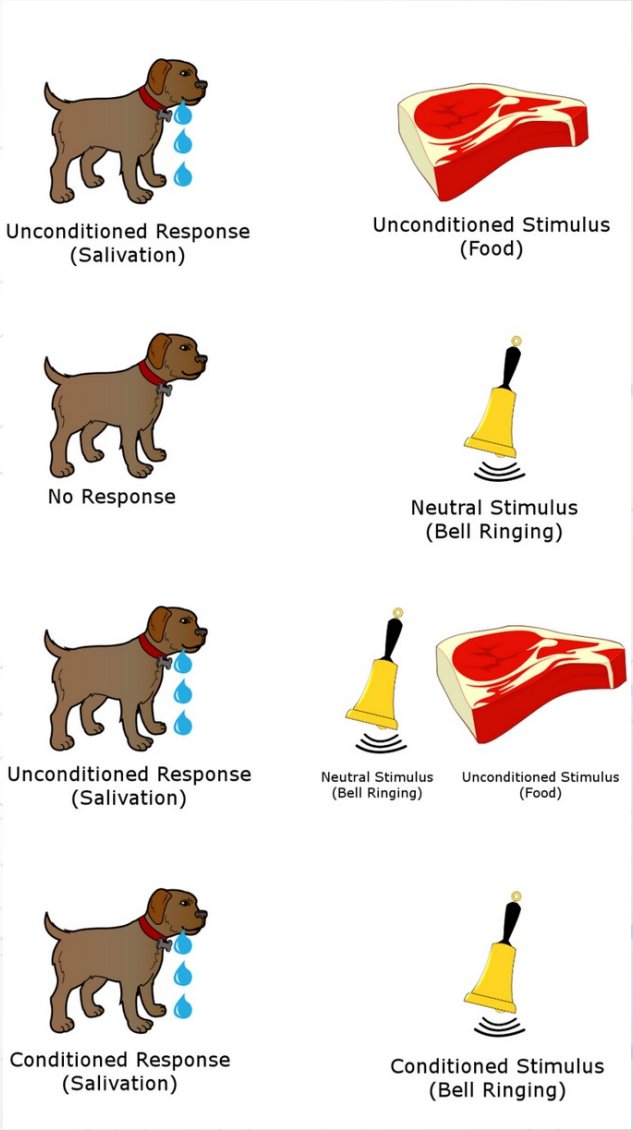
$$b^{nuevo} = b^{anterior} + \alpha e$$

Regla Delta

Aprendizaje Widrow-Hoff (basado en el trabajo de Donald O. Hebb)

Intermezzo

- D.O. Hebb
- Pavlovian condition



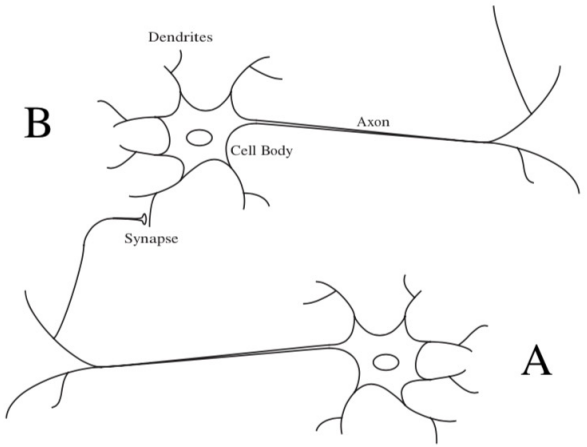
Intermezzo

- "… behavior could be explained by the action of neurons"

Postulado de Hebb

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

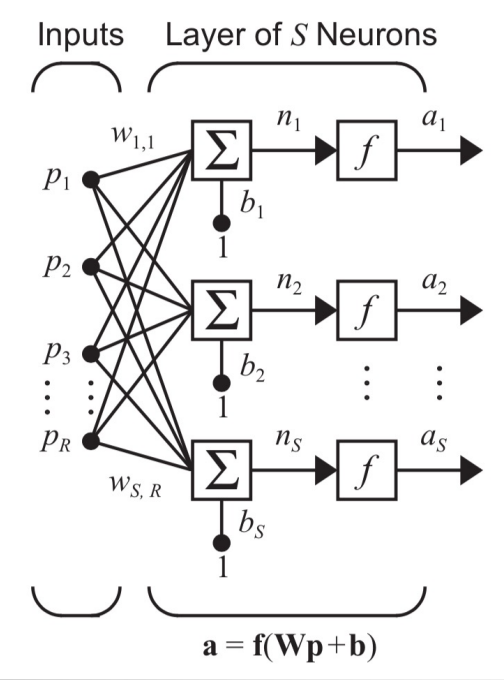
D. O. Hebb, 1949



Forma matricial

$$\mathbf{W}(k+1) = \mathbf{W}(k) + \alpha \mathbf{e}(k)\mathbf{p}^T(k)$$

$$\mathbf{b}(k+1) = \mathbf{b}(k) + \alpha \mathbf{e}(k)$$



$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_S \end{bmatrix}$$

Tarea moral

Scikit-learn - Neural network models

1.17. Neural network models (supervised)

Warning: This implementation is not intended for large-scale applications. In particular, scikit-learn offers no GPU support. For much faster, GPU-based implementations, as well as frameworks offering much more flexibility to build deep learning architectures, see Related Projects.

1.17.1. Multi-layer Perceptron

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function $f(\cdot) : R^m \rightarrow R^o$ by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features $X = x_1, x_2, \ldots, x_m$ and a target y, it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. Figure 1 shows a one hidden layer MLP with scalar output.

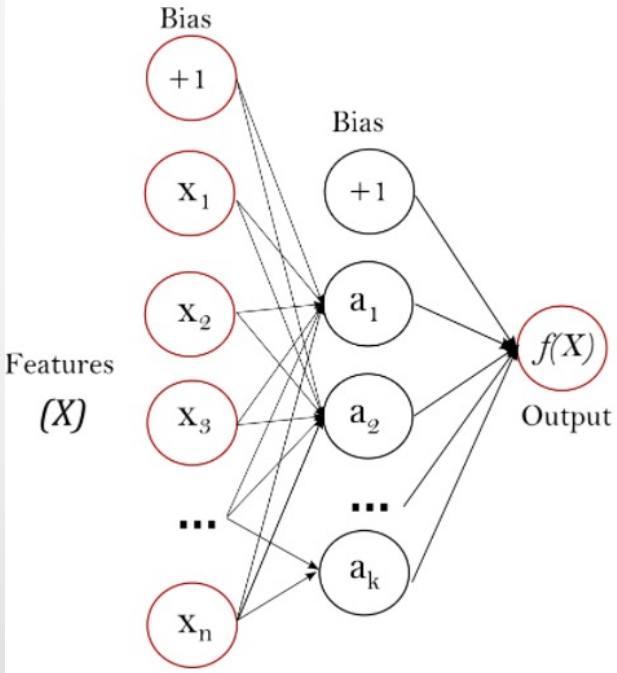


Figure 1 : One hidden layer MLP.

Neural network models

1.17.2. Classification

Class `MLPClassifier` implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation.

MLP trains on two arrays: array X of size (n_samples, n_features), which holds the training samples represented as floating point feature vectors; and array y of size (n_samples,), which holds the target values (class labels) for the training samples:

1.17.3. Regression

Class `MLPRegressor` implements a multi-layer perceptron (MLP) that trains using backpropagation with no activation function in the output layer, which can also be seen as using the identity function as activation function. Therefore, it uses the square error as the loss function, and the output is a set of continuous values.

`MLPRegressor` also supports multi-output regression, in which a sample can have more than one target.

Neural network models

1.17.2. Classification

Class `MLPClassifier` implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation.

MLP trains on two arrays: array X of size (n_samples, n_features), which holds the training samples represented as floating point feature vectors; and array y of size (n_samples,), which holds the target values (class labels) for the training samples:

sklearn.neural_network.MLPClassifier

class `sklearn.neural_network.` **`MLPClassifier`**(*hidden_layer_sizes=100, activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*)

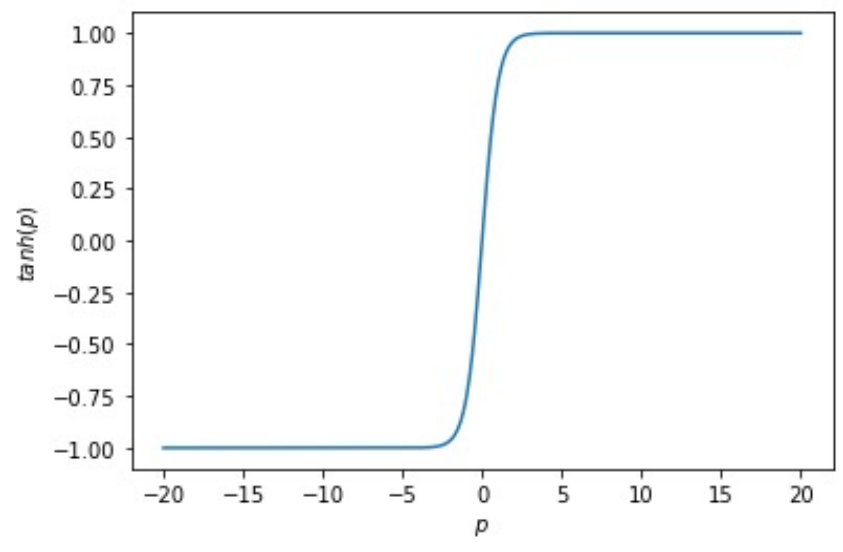
[source]

Neural network models

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*

Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max(0, x)



Código

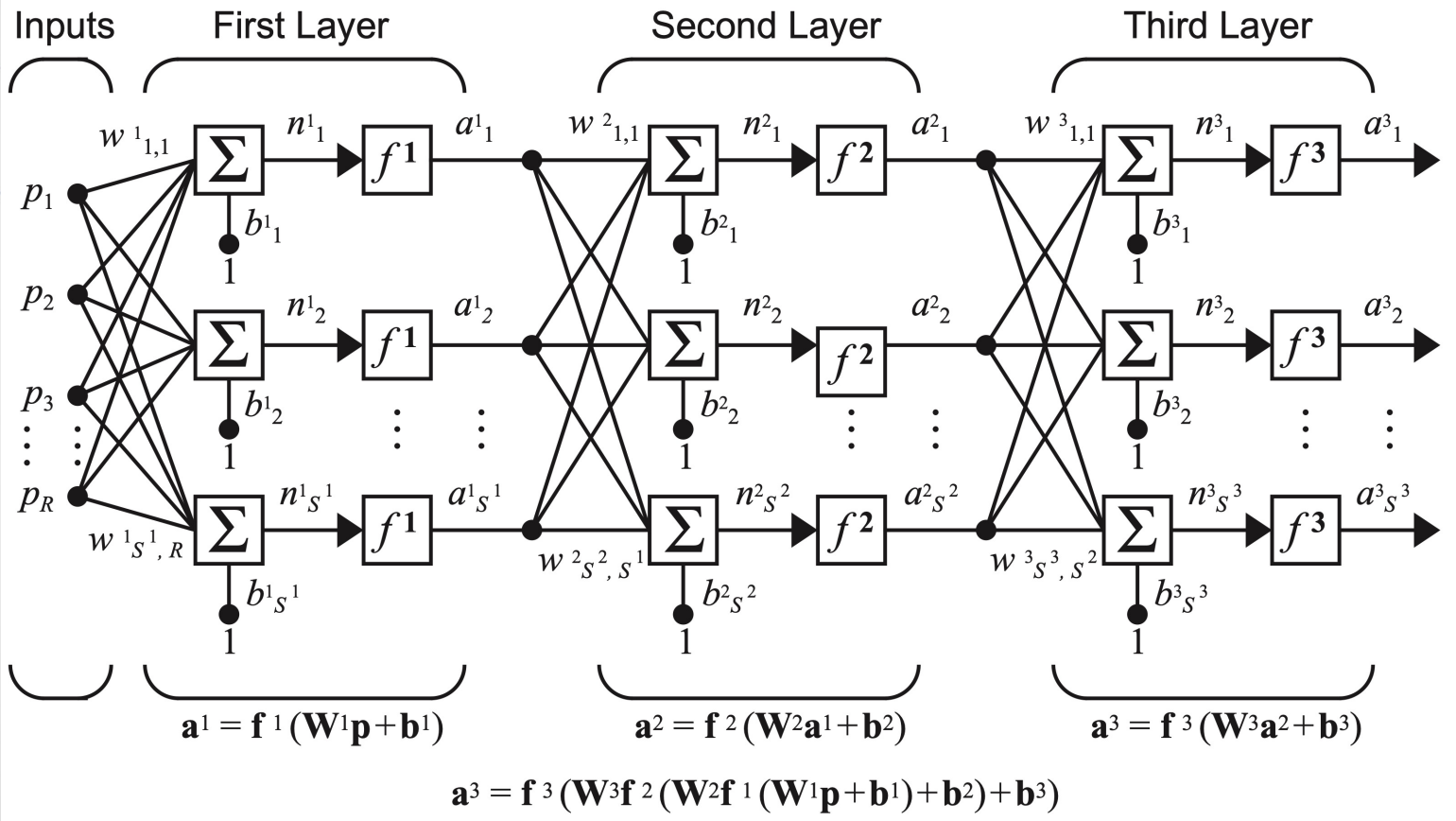
Clases no linealmente separables

```
import numpy as np
import sklearn.neural_network

p = np.array([[0,0],[0,1],[1,0],[1,1]])
t = np.array([0,1,1,0])

modelo = sklearn.neural_network.MLPClassifier(
                    activation='tanh',
                    max_iter=1000,
                    hidden_layer_sizes=(2),
                    learning_rate_init=0.1,
                    solver='sgd')
modelo.fit(p, t)
print('Predicciones del modelo:', modelo.predict(p))
```

Perceptrón mullticapa

Multilayer perceptron (MLP)

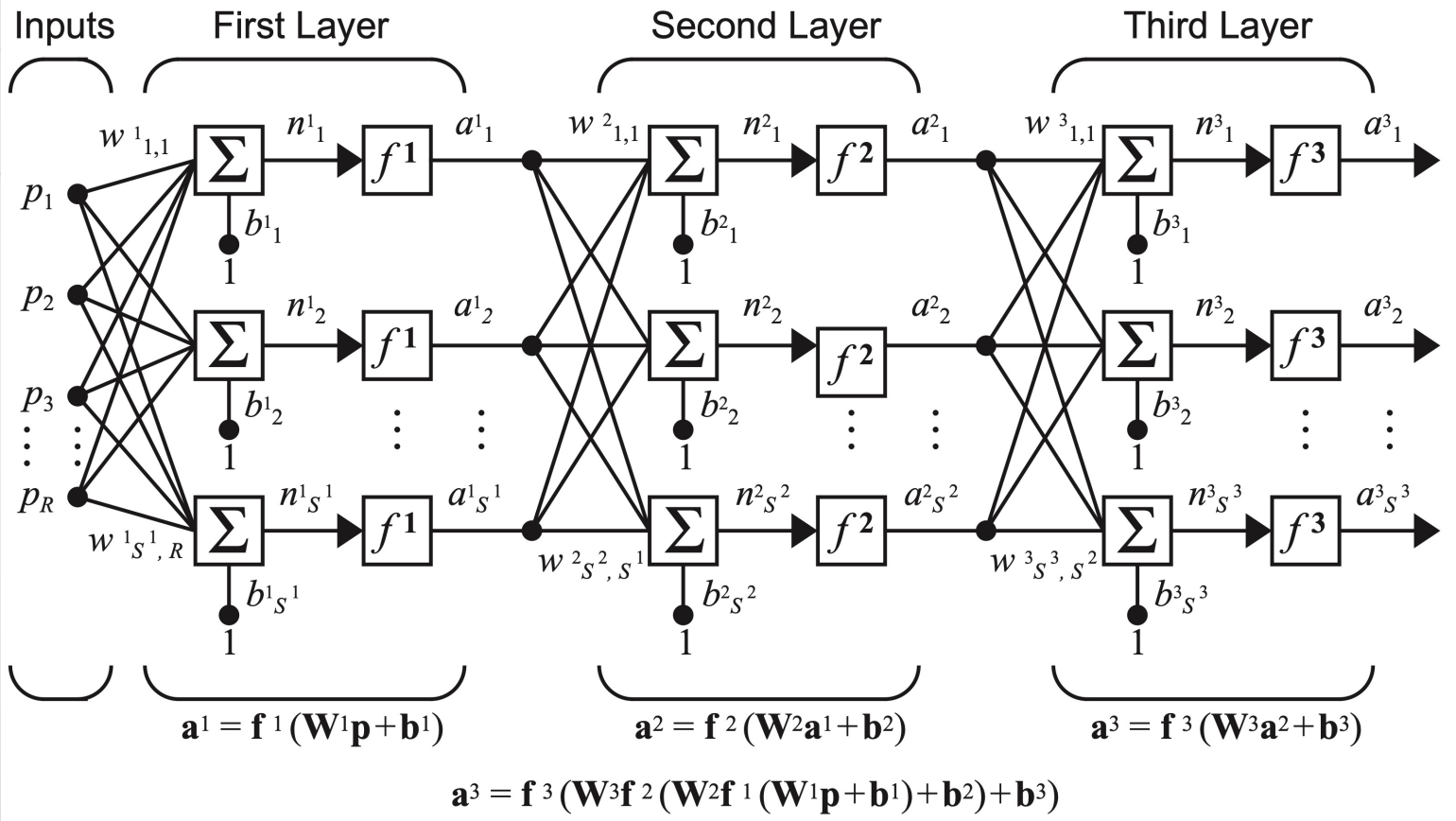
Varias capas de neuronas - Arquitectura Feedforward



$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

¿Cómo entrenar una red multicapa?

Idea general: Parámetros a ajustar y propagación del error



Propagación hacia atrás - Backpropagation

Datos de entrenamiento

$$\{p_1, t_1\}, \{p_2, t_2\}, \ldots, \{p_R, t_R\}$$

Error cuadrático

$$F(x) = E[e^2] = E[(t - a)]$$

Forma vectorial

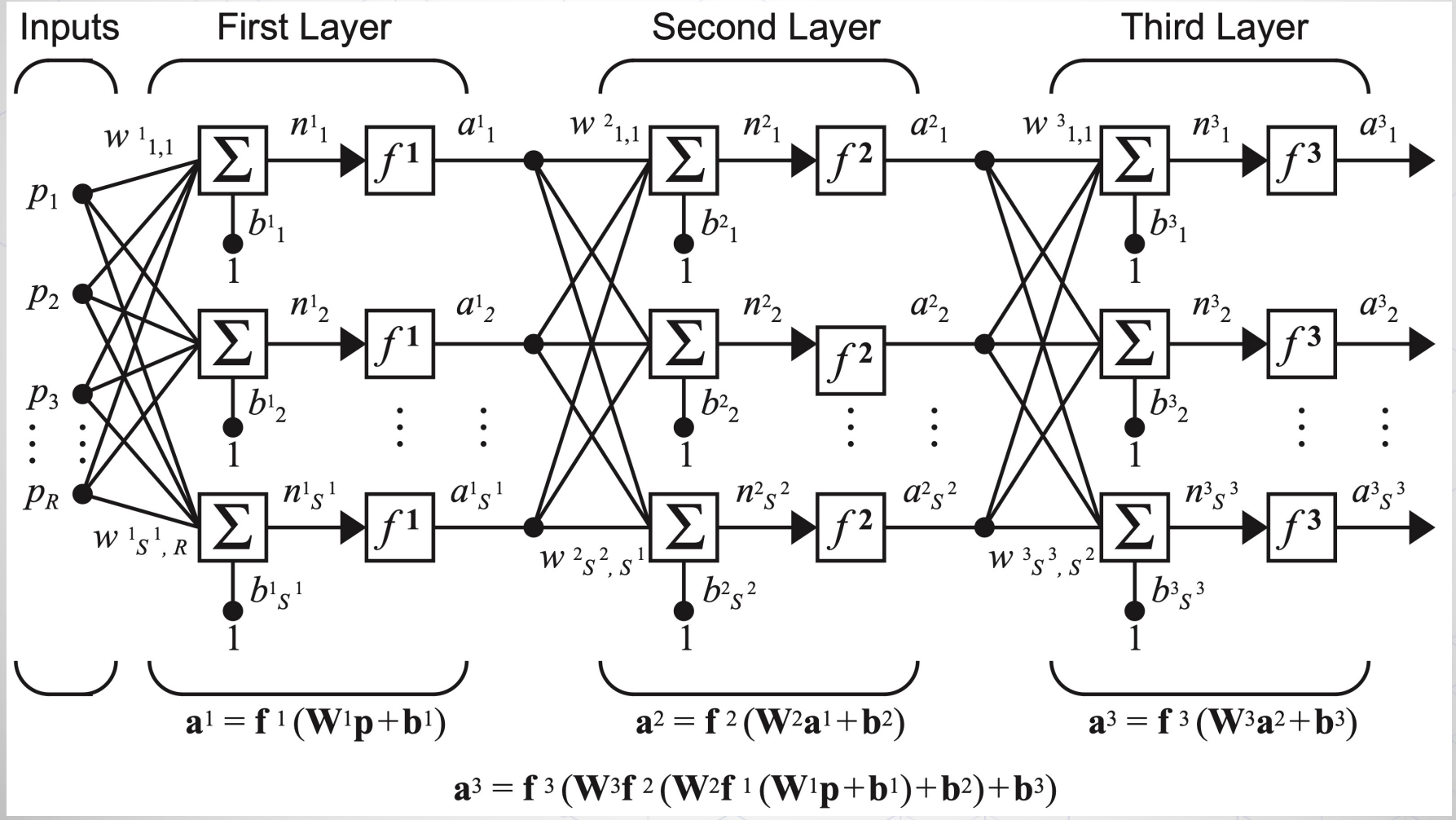
$$F(x) = E[\boldsymbol{e}^T \boldsymbol{e}] = E[(\boldsymbol{t} - \boldsymbol{a})^T (\boldsymbol{t} - \boldsymbol{a})]$$

Error aproximado en una iteración k

$$\hat{F}(x) = (\boldsymbol{t}(k) - \boldsymbol{a}(k))^T (\boldsymbol{t}(k) - \boldsymbol{a}(k)) = \boldsymbol{e}^T(k)\boldsymbol{e}(k)$$

Ajuste de pesos y bias … pero

¿Parámetros a ajustar?



$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

Ajuste de pesos y bias

$$w_{i,j}^{m}(k+1) = w_{i,j}^{m}(k) - \alpha \frac{\partial \hat{F}}{\partial w_{i,j}^{m}}$$

$$b_{i}^{m}(k+1) = b_{i}^{m}(k) - \alpha \frac{\partial \hat{F}}{\partial b_{i}^{m}}$$

Resumen

Propagación hacia adelante

$$\mathbf{a}^0 = \mathbf{p}$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a} = \mathbf{a}^M$$

Ejemplo con tres capas

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) \qquad \mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2) \qquad \mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2 + \mathbf{b}^3)$$

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3)$$

Resumen

Propagación hacia atrás

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a})$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1} \qquad m = M-1, \ldots, 2, 1$$

Resumen

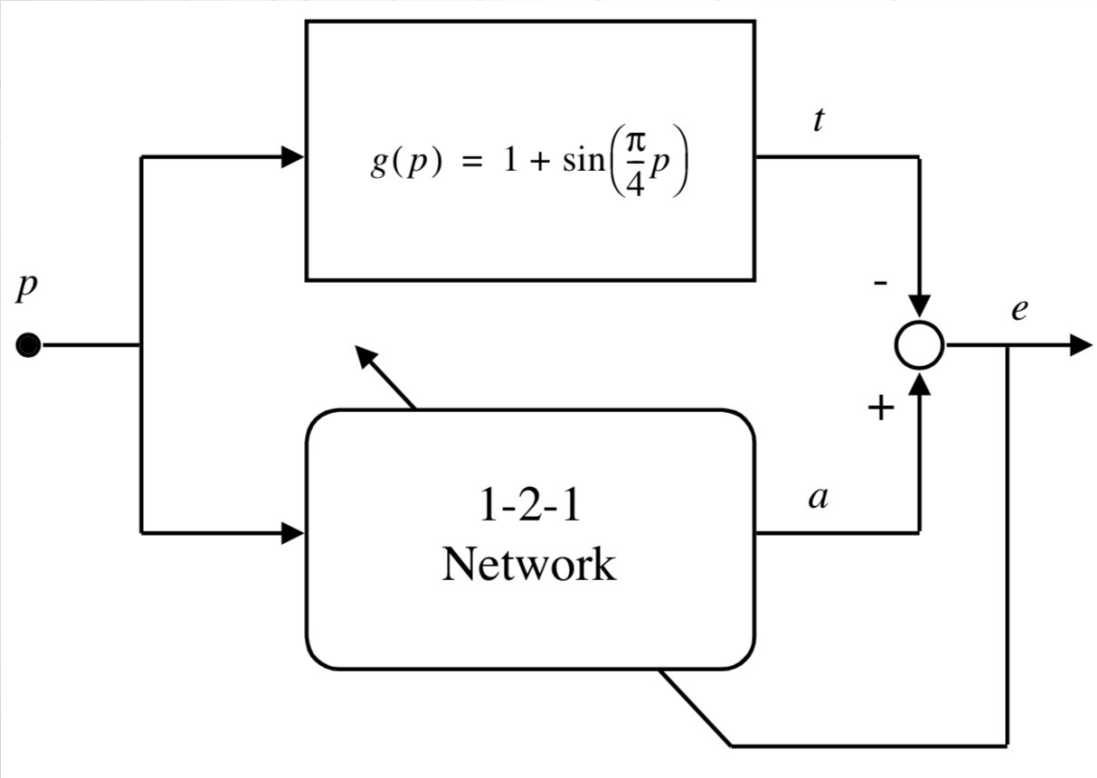
Actualización de pesos y bias

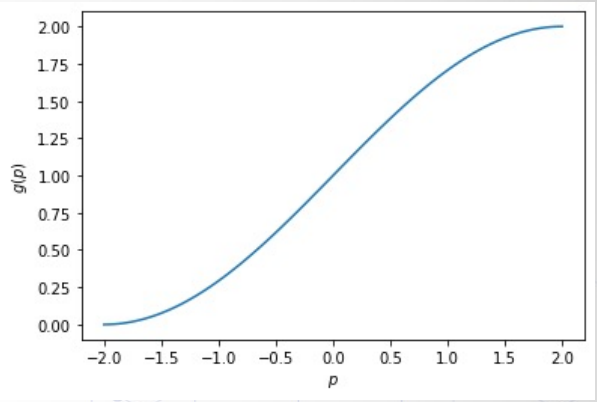
$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m$$

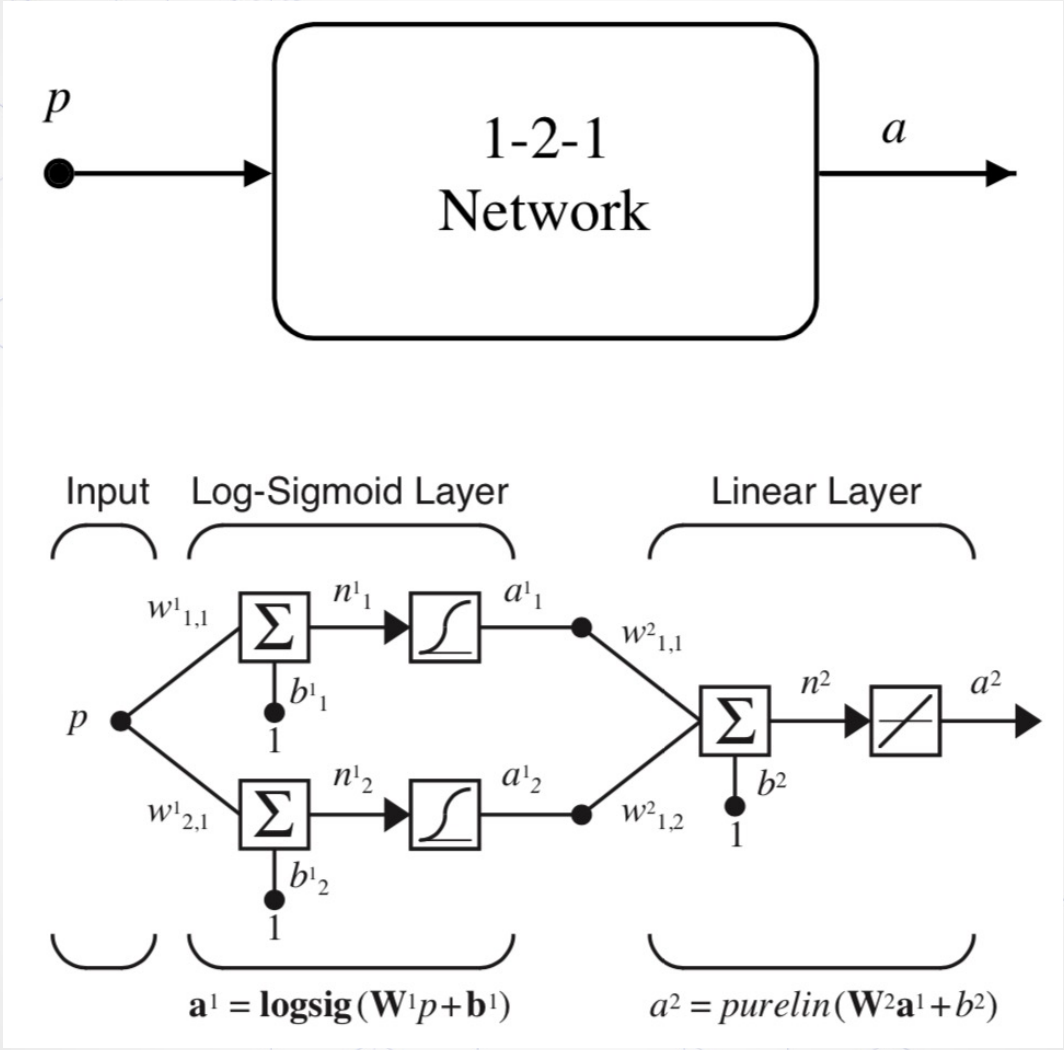
Ejemplo

Aproximación de una función



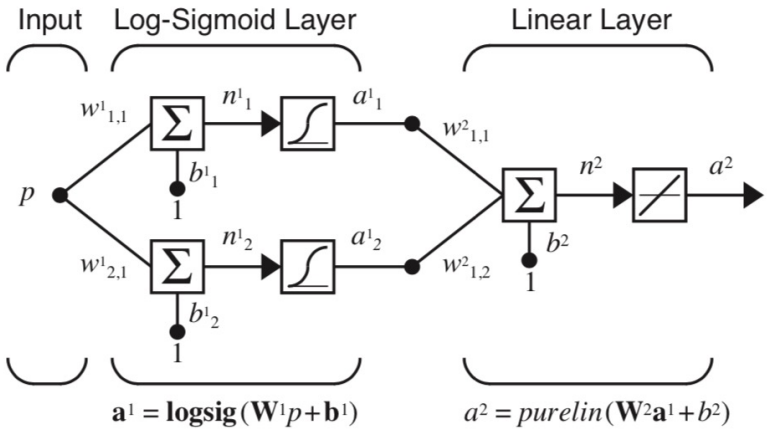


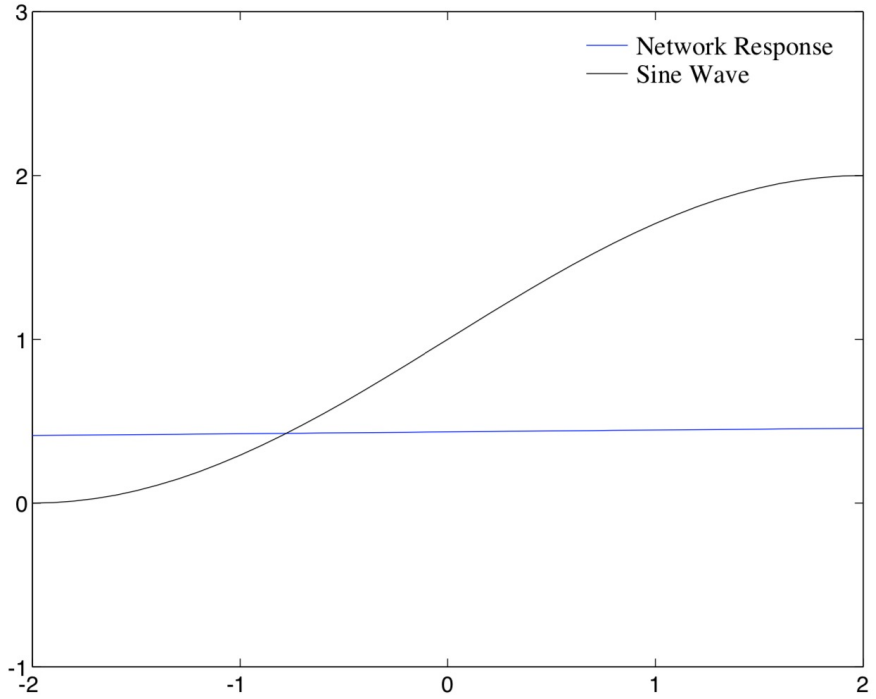
ANN


p
1-2-1
Network
a
Input
Log-Sigmoid Layer
Linear Layer
$w^1_{1,1}$
n^1_1
a^1_1
$w^2_{1,1}$
b^1_1
n^2
a^2
p
1
b^2
$w^1_{2,1}$
n^1_2
a^1_2
$w^2_{1,2}$
b^1_2
1
$\mathbf{a}^1 = \mathbf{logsig}(\mathbf{W}^1 p + \mathbf{b}^1)$
$a^2 = purelin(\mathbf{W}^2\mathbf{a}^1 + b^2)$

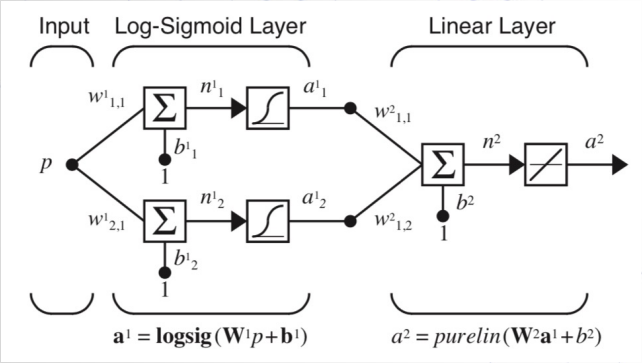
Condiciones iniciales

$$\mathbf{W}^1(0) = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} \quad \mathbf{b}^1(0) = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} \quad \mathbf{W}^2(0) = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} \quad \mathbf{b}^2(0) = \begin{bmatrix} 0.48 \end{bmatrix}$$





Propagación hacia adelante



$$a^0 = p = 1$$

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{a}^0 + \mathbf{b}^1) = \mathbf{logsig}\left(\begin{bmatrix}-0.27\\-0.41\end{bmatrix}[1] + \begin{bmatrix}-0.48\\-0.13\end{bmatrix}\right) = \mathbf{logsig}\left(\begin{bmatrix}-0.75\\-0.54\end{bmatrix}\right)$$

$$\mathbf{a}^1 = \begin{bmatrix}\dfrac{1}{1+e^{0.75}}\\[2ex]\dfrac{1}{1+e^{0.54}}\end{bmatrix} = \begin{bmatrix}0.321\\0.368\end{bmatrix}$$

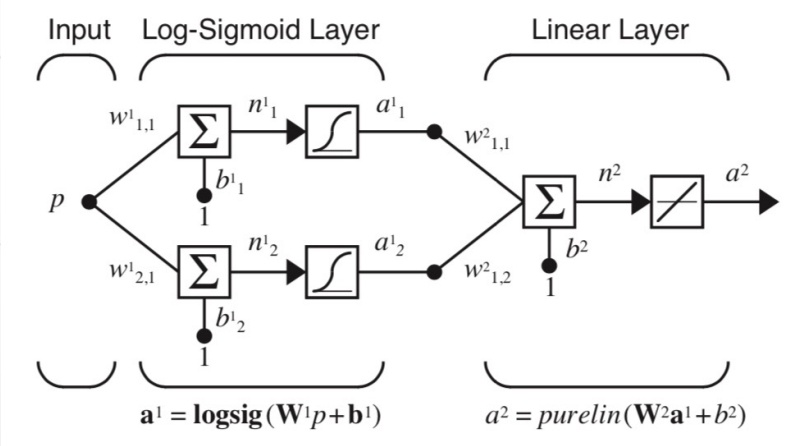
$$a^2 = \mathrm{f}^2(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2) = purelin\left(\begin{bmatrix}0.09 & -0.17\end{bmatrix}\begin{bmatrix}0.321\\0.368\end{bmatrix} + [0.48]\right) = [0.446]$$

$$e = t - a = \left\{1 + \sin\left(\frac{\pi}{4}p\right)\right\} - a^2 = \left\{1 + \sin\left(\frac{\pi}{4}1\right)\right\} - 0.446 = 1.261$$

Derivadas para cada capa

$$\dot{f}^1(n) = \frac{d}{dn}\left(\frac{1}{1+e^{-n}}\right) = \frac{e^{-n}}{(1+e^{-n})^2} = \left(1-\frac{1}{1+e^{-n}}\right)\left(\frac{1}{1+e^{-n}}\right) = (1-a^1)(a^1)$$

$$\dot{f}^2(n) = \frac{d}{dn}(n) = 1$$



Propagación hacia atrás

$$\mathbf{s}^2 = -2\dot{\mathbf{F}}^2(\mathbf{n}^2)(\mathbf{t}-\mathbf{a}) = -2\left[\dot{f}^2(n^2)\right](1.261) = -2\left[1\right](1.261) = -2.522$$

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T\mathbf{s}^2 = \begin{bmatrix}(1-a_1^1)(a_1^1) & 0\\ 0 & (1-a_2^1)(a_2^1)\end{bmatrix}\begin{bmatrix}0.09\\ -0.17\end{bmatrix}\begin{bmatrix}-2.522\end{bmatrix}$$

$$\mathbf{s}^1 = \begin{bmatrix}(1-0.321)(0.321) & 0\\ 0 & (1-0.368)(0.368)\end{bmatrix}\begin{bmatrix}0.09\\ -0.17\end{bmatrix}\begin{bmatrix}-2.522\end{bmatrix}$$

$$\mathbf{s}^1 = \begin{bmatrix}0.218 & 0\\ 0 & 0.233\end{bmatrix}\begin{bmatrix}-0.227\\ 0.429\end{bmatrix} = \begin{bmatrix}-0.0495\\ 0.0997\end{bmatrix}$$

Actualización de pesos y bias

$$\alpha = 0.1$$

$$\mathbf{W}^2(1) = \mathbf{W}^2(0) - \alpha \mathbf{s}^2 (\mathbf{a}^1)^T = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} \begin{bmatrix} 0.321 & 0.368 \end{bmatrix}$$

$$\mathbf{W}^2(1) = \begin{bmatrix} 0.171 & -0.0772 \end{bmatrix}$$

$$\mathbf{b}^2(1) = \mathbf{b}^2(0) - \alpha \mathbf{s}^2 = \begin{bmatrix} 0.48 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} = \begin{bmatrix} 0.732 \end{bmatrix}$$

$$\mathbf{W}^1(1) = \mathbf{W}^1(0) - \alpha \mathbf{s}^1 (\mathbf{a}^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix}$$

$$\mathbf{b}^1(1) = \mathbf{b}^1(0) - \alpha \mathbf{s}^1 = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} = \begin{bmatrix} -0.475 \\ -0.140 \end{bmatrix}$$

Código

Regresión

1.17.3. Regression

Class `MLPRegressor` implements a multi-layer perceptron (MLP) that trains using backpropagation with no activation function in the output layer, which can also be seen as using the identity function as activation function. Therefore, it uses the square error as the loss function, and the output is a set of continuous values.

`MLPRegressor` also supports multi-output regression, in which a sample can have more than one target.

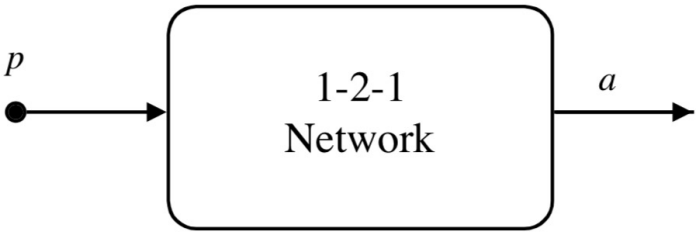
sklearn.neural_network.MLPRegressor

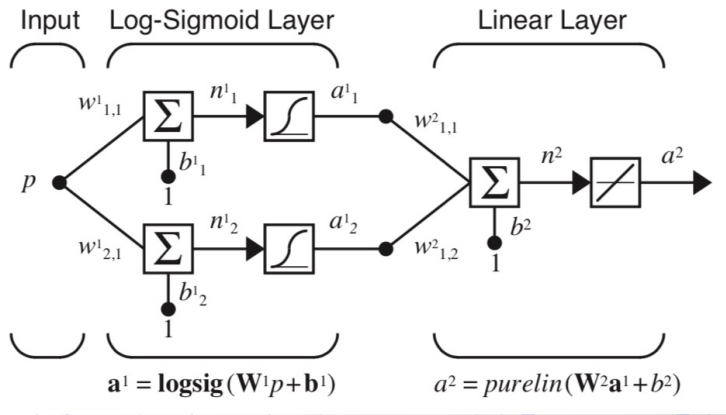
class `sklearn.neural_network.` **`MLPRegressor`**(*hidden_layer_sizes=100, activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*)

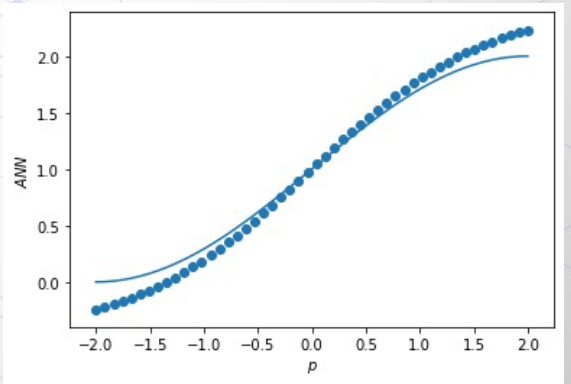
[source]

Regresión - $g(p) = 1 + sin\left(\frac{\pi}{4}p\right)$

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.neural_network

def myF(p):
    res = 1 + np.sin( (np.pi / 4) * p)
    return res

p  = np.linspace(-2,2,50)
gp = myF(p)

p = p.reshape(-1,1)
modelo = sklearn.neural_network.MLPRegressor(
        hidden_layer_sizes=(2),
        activation='logistic',
        learning_rate_init=0.1,
        max_iter=1000,
        solver='sgd'
        )

modelo.fit(p,gp)
plt.plot(p,gp)
plt.scatter(p,modelo.predict(p))
plt.xlabel(r'$p$')
plt.ylabel(r'$ANN$')
plt.show()
```

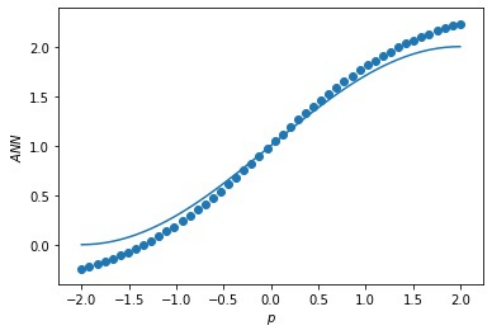


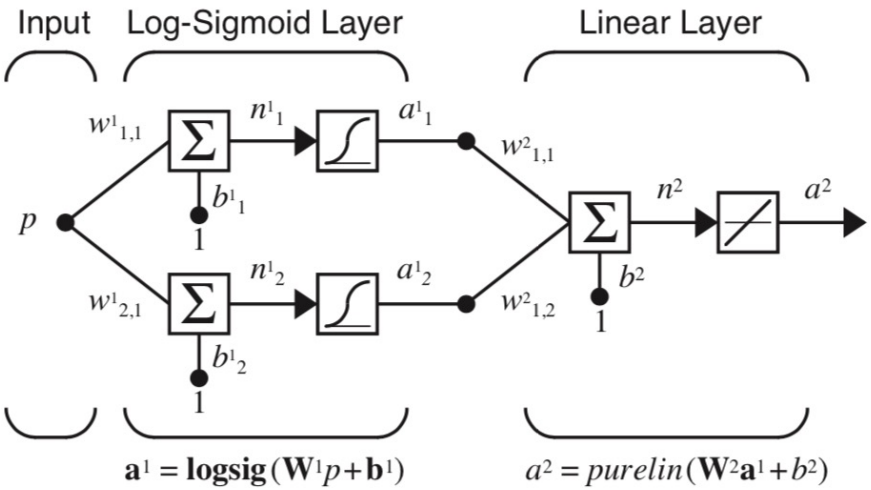




Regresión - $g(p) = 1 + sin\left(\frac{\pi}{4}p\right)$

```
In [49]: modelo.n_layers_
Out[49]: 3

In [50]: modelo.coefs_
Out[50]:
[array([[ 1.13913612, -1.3550187 ]]), array([[ 1.44739596],
        [-1.49592444]])]

In [51]: modelo.intercepts_
Out[51]: [array([-0.29711573, -0.28436463]), array([1.03153269])]

In [52]: modelo.n_iter_
Out[52]: 35
```





EngineeringX

Founded by the Royal Academy of Engineering
and Lloyd's Register Foundation

GRACIAS

https://hubiq.mx/

HUBIQRO HUBIQ HUBIQRO