Backpropagation

Andrés Amaya   Naren Mantilla   Alexander Sepúlveda

Universidad Industrial de Santander

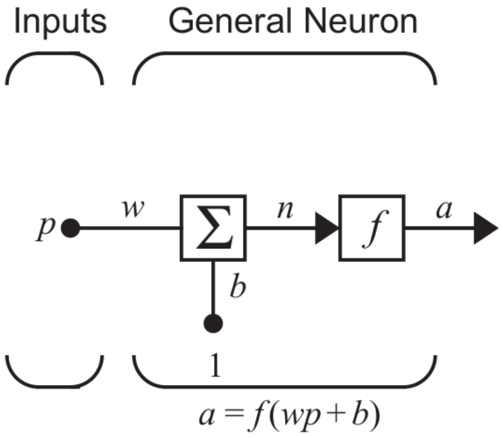
December 5, 2019

Overview

Outline

Outline

Single input neuron



Single input neuron

The neuron output is calculated as

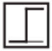
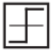
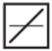
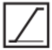
$$a = f(wp + b). \tag{1}$$

If, for instance, $w = 3$, $p = 2$ and $b = -1.5$, then

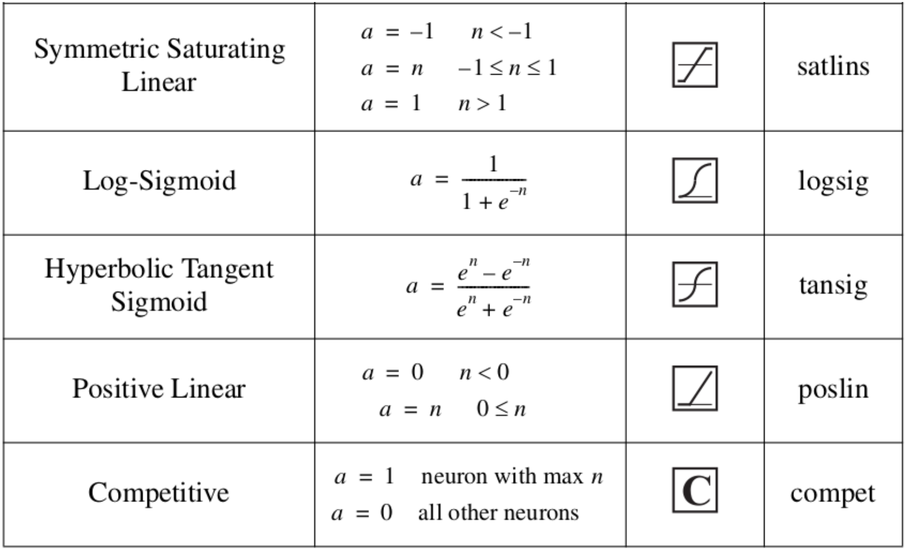
$$a = f(3(2) - 1.5) = f(4.5). \tag{2}$$

The actual output depends on the particular transfer function that is chosen. These functions are also known as **Activation functions**, and some of them are described.

Activation functions

Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin

Activation functions



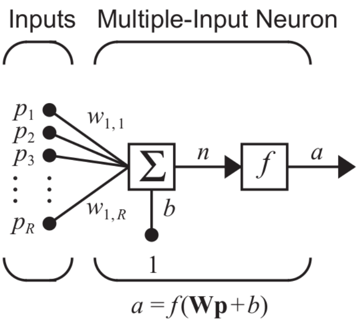
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \le n \le 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1+e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \le n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons	$\boxed{C}$	compet

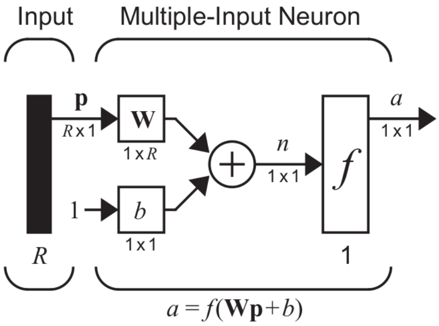
Try the example

Experiment

Try with code **nnd2n1.m**.

Multiple input neuron





Multiple input neuron

Be a neuron with R inputs, the adder output is:

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \cdots + w_{1,R}p_R + b. \tag{3}$$

This expression can be written in matrix form:

$$n = \mathbf{W}\mathbf{p} + b, \tag{4}$$

where the matrix $\mathbf{W}$ for the single neuron case has only one row. Now, the neuron output can be written as

$$a = f(\mathbf{W}\mathbf{p} + b). \tag{5}$$

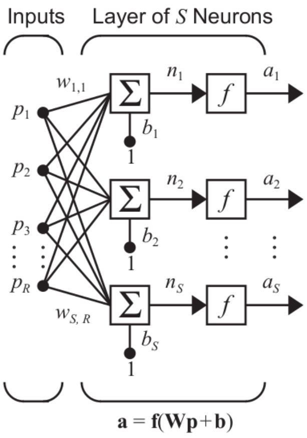
Try the example

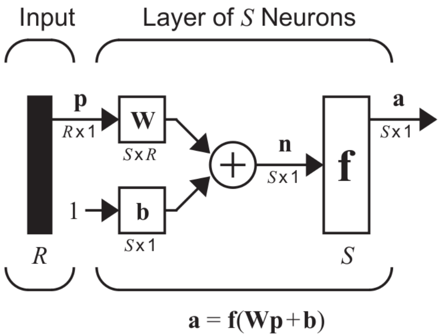
Experiment

Try with code **nnd2n2.m**.

Outline

Layer of S neurons





Layer of S neurons

The input vector elements enter the network through the weight matrix $\mathbf{W}$:

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix} \tag{6}$$

The row indices of the elements of matrix $\mathbf{W}$ indicate the destination neuron associated with that weight, while the column indices indicate the source of the input for that weight.

Layer of S neurons

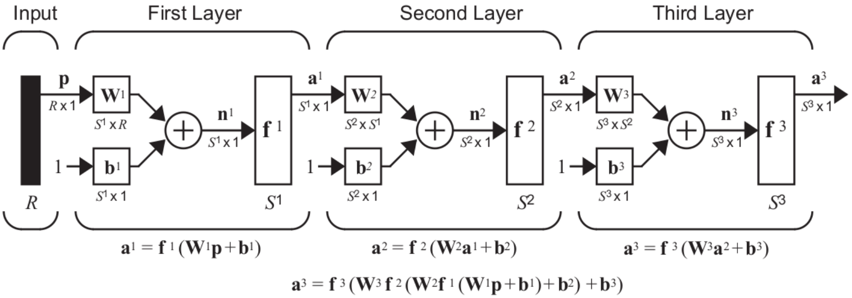
Each element of the input vector $\mathbf{p}$ is connected to each neuron through the weight matrix $\mathbf{W}$. Each neuron has a bias b_i, an adder, an activation function f and an output a_i. Taken together, the outputs form the output vector $\mathbf{a}$:

$$\mathbf{a} = \mathbf{f}(\mathbf{W}\mathbf{p} + \mathbf{b}), \tag{7}$$

where $\mathbf{p}$ is a vector of length R, $\mathbf{W}$ is an $S \times R$ matrix, and $\mathbf{a}$ and $\mathbf{b}$ are vectors of length S.

Multiple layers network

Consider a network with several layers. Each layer has its own weight matrix $\mathbf{W}$, its own bias vector $\mathbf{b}$, a net input vector $\mathbf{n}$ and an output vector $\mathbf{a}$.



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

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

Multiple layers network

Superscripts are used to identify the layers. Thus, the weight matrix for the first layer is written as W^1, and the weight matrix for the second layer is written as W^2.

There are R inputs, S^1 neurons in the first layer, S^2 neurons in the second layer, etc. The output of layer one is the input for layer two, and so on.

A layer whose output is the network output is called and *output layer*. The other layers are called *hidden layers*.

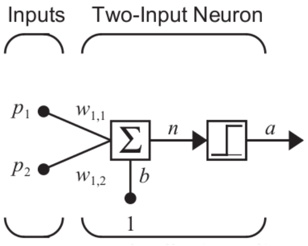
In a three-layer network, the output is:

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3). \tag{8}$$

Outline

Outline

Single-neuron perceptron



The output of this network is determined by

$$a = hardlim(n) = hardlim(\mathbf{W}\mathbf{p} + b), \tag{9}$$

where

$$hardlim(n) = \begin{cases} 1 & \text{if } n \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

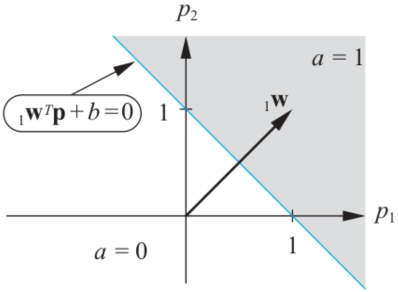
Single-neuron perceptrons can classify input vectors into two categories.

Single-neuron perceptron

The *decision boundary* is determined by the input vectors for which the net input n is zero:

$$n = \mathbf{W}\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b = 0. \tag{11}$$

Case $w_{1,1} = w_{1,2} = 1$, $b = -1$:



Perceptron Learning Rule

The learning rule is provided with a set of examples of proper network behavior:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}, \tag{12}$$

where $\mathbf{p}_q$ is an input to the network and $\mathbf{t}_q$ is the corresponding target output. As each input is applied, the network output is compared to the target. The learning rule adjusts the weights and biases to move the output closer to the target.

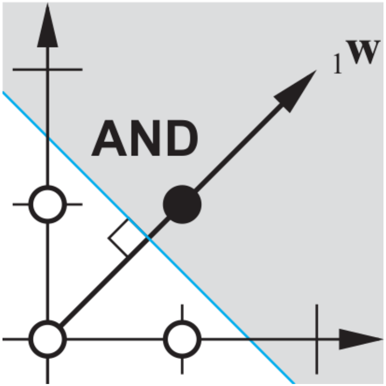
The perceptron learning rule is:

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T, \tag{13}$$

$$\mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e}, \tag{14}$$

$$\text{where } \mathbf{e} = \mathbf{t} - \mathbf{a}. \tag{15}$$

Example: AND gate

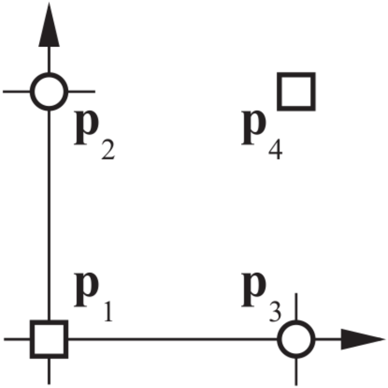


The dark circles indicate that the target is 1, and the light circles indicate the target is 0.

Outline

XOR gate

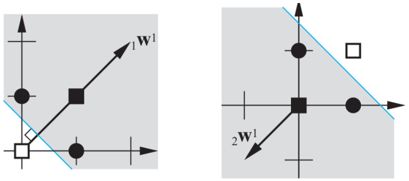
Single layer perceptrons has limmitations. For example, consider the XOR gate problem:



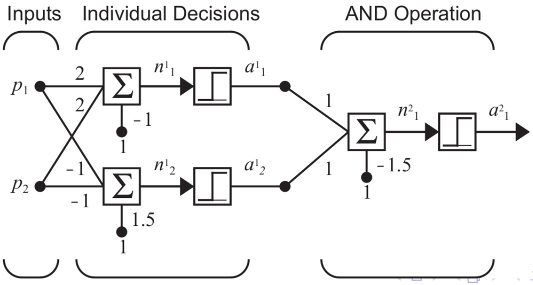
Because the two categories are not linearly separable, a single layer perceptron cannot perform the classification.

Multiple layer perceptron

A two-layer network can solve the XOR problem. In fact, there are many different multilayer solutions.



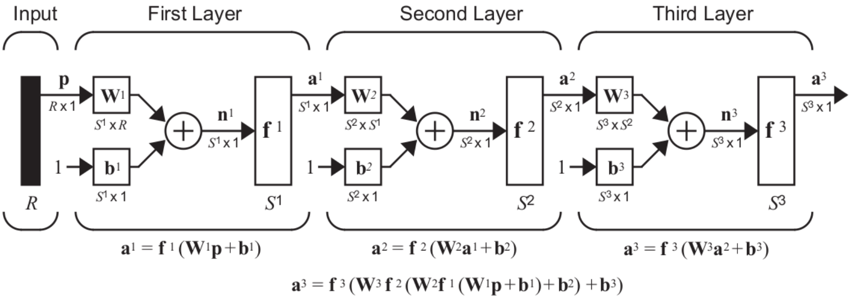
Layer 1/Neuron 1 Layer 1/Neuron 2



Outline

Back propagation algorithm

Back propagation is an approximate steepest descent algorithm, in which the performance index is mean square error (MSE).



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

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

Forward Propagation

Forward propagation

The first step is to propagate the input forward through the network.

$$\mathbf{a}^0 = \mathbf{p}, \tag{16}$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}), \qquad \text{for } m = 0, 1, \cdots, M-1, \tag{17}$$

$$\mathbf{a} = \mathbf{a}^M, \tag{18}$$

where m is the current layer and M is the total number of layers.

Performance Index

The algorithm is provided with a set of examples of proper network behavior:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}, \tag{19}$$

where $\mathbf{p}_q$ is an input to the network and $\mathbf{t}_q$ is the corresponding target output. The algorithm should adjust the network parameters in order to minimize the MSE:

$$F(\mathbf{x}) = E\left[\mathbf{e}^T\mathbf{e}\right] = E\left[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})\right], \tag{20}$$

where $\mathbf{x}$ is the vector of network weights and biases. The MSE is approximated by the squared error at iteration k:

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k)-\mathbf{a}(k))^T(\mathbf{t}(k)-\mathbf{a}(k)) = \mathbf{e}^T(k)\mathbf{e}(k). \tag{21}$$

Performance Index

The steepest descent algorithm for the approximate mean square error (stochastic gradient descent) is

$$w_{i,j}^{m}(k+1) = w_{i,j}^{m}(k) - \alpha \frac{\partial \hat{F}}{\partial w_{i,j}^{m}}, \tag{22}$$

$$b_{i}^{m}(k+1) = b_{i}^{k} - \alpha \frac{\partial \hat{F}}{\partial b_{i}^{m}}, \tag{23}$$

where α is known as the **learning rate**. To find the derivatives, it is used the **chain rule**.

Chain rule

Applying the chain rule to the derivatives of the function $\hat{F}$:

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^m}, \tag{24}$$

$$\frac{\partial \hat{F}}{\partial b_i^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial b_i^m}, \tag{25}$$

As the net input to layer m is an explicit function of the weights and bias in that layer:

$$n_i^m = \sum_{j=1}^{s^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m. \tag{26}$$

Therefore

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1}, \qquad \frac{\partial n_i^m}{\partial b_i^m} = 1. \tag{27}$$

Sensitivity

Defining the **sensitivity** of $\hat{F}$ to changes in the ith element of the net input at layer m as

$$s_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}, \tag{28}$$

the derivatives can be simplified to

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = s_i^m a_j^{m-1}, \tag{29}$$

$$\frac{\partial \hat{F}}{\partial b_i^m} = s_i^m. \tag{30}$$

We can now express the approximate steepest descent algorithm as

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha s_i^m a_j^{m-1}, \tag{31}$$

$$b_i^m(k+1) = b_i^m(k) - \alpha s_i^m. \tag{32}$$

Falta

Falta terminar

Backpropagating the sensitivities

Outline

Summary of back propagation algorithm

Step 1: Forward propagation

The first step is to propagate the input forward through the network.

$$\mathbf{a}^0 = \mathbf{p}, \tag{33}$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}), \quad \text{for } m = 0, 1, \cdots, M-1, \tag{34}$$

$$\mathbf{a} = \mathbf{a}^M, \tag{35}$$

where m is the current layer and M is the total number of layers.

Summary of back propagation algorithm

Step 2: Back propagation

The next step is to propagate the sensitivities backward through the network.

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a}), \tag{36}$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}, \qquad \text{for } m = M-1, \cdots, 2, 1. \tag{37}$$

Summary of back propagation algorithm

Step 3: Weights and biases update

Finally, the weights and biases are updated using the approximate steepest descent rule.

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T, \tag{38}$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m, \tag{39}$$

where α is the learning rate.

Try the example

Experiment

Try with code **nnd11bc.m**.

References

Howard B Demuth, Mark H Beale, Orlando De Jess, and Martin T Hagan.
Neural network design.
Martin Hagan, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
Deep learning.
MIT press, 2016.