Νευρο-Ασαφής Υπολογιστική
Neuro-Fuzzy Computing

Διδάσκων –
Δημήτριος Κατσαρός

@ Τμ. ΗΜΜΥ
Πανεπιστήμιο Θεσσαλίας

Διάλεξη 4η

Perceptron’s convergence

Proof of convergence

- Suppose that we have n training patterns that belong to two classes ω_1 and ω_2
  - the patterns of class ω_2 have been multiplied by -1
- If the classes are linearly separable, the learning algorithm yields a **solution weight vector w*** with the property

$$w^{*'} x_i > 0, \qquad i = 1, \ldots, n$$

- We can generalize using a non-negative threshold T

$$w^{*'} x_i > T, \qquad i = 1, \ldots, n$$

Proof of convergence

- Thus, the learning algorithm becomes

$$w(k+1) = \begin{cases} w(k), & \text{if } w^{'}(k)x_i(k) > T \\ w(k) + x_i(k), & \text{if } w^{'}(k)x_i(k) \leq T \end{cases}$$

$w(1)$ is arbitrary, and $c = 1$

- We will consider only the indices k for which a correction takes place during training; thus

$$w(k+1) = w(k) + x_i(k)$$

and

$$w^{'}(k)x_i(k) \leq T$$

Proof of convergence

- Convergence of the algorithm means that, after some finite index k_m

$$w(k_m) = w(k_m + 1) = w(k_m + 2) = \dots$$

<u>The proof is as follows</u>:

$$w(k+1) = w(1) + x_i(1) + x_i(2) + \cdots + x_i(k)$$

Taking the inner product of w* with both sides

$$w'(k+1)w^* = w'(1)w^* + x_i^{'}(1)w^* + x_i^{'}(2)w^* + \cdots + x_i^{'}(k)w^*$$

Since each term $x_i^{'}(j)w^*, j = 1, 2, \dots, k$ is greater than T, then $w'(k+1)w^* \geq w'(1)w^* + kT$

Proof of convergence

Taking the Cauchy-Schwartz inequality, results in

$$[w'(k+1)w^*]^2 \leq ||w(k+1)||^2 \; ||w^*||^2$$

Which may be written in the following form

$$||w(k+1)||^2 \geq \frac{[w'(k+1)w^*]^2}{||w^*||^2}$$

Substituting the nominator from previous inequality, we get

$$||w(k+1)||^2 \geq \frac{[w'(1)w^* + kT]^2}{||w^*||^2}$$

Proof of convergence

Now, an alternative line of reasoning leads to a contradiction regarding $||w(k+1)||^2$

From the perceptron updating rule, we have

$$||w(j+1)||^2 = ||w(j)||^2 + 2w'(j)x_i(j) + ||x_i(j)||^2$$

or

$$||w(j+1)||^2 - ||w(j)||^2 = 2w'(j)x_i(j) + ||x_i(j)||^2$$

Having in mind that $w'(k)x_i(k) \leq T$

and letting $Q = max||x_i(j)||^2$ we have that

$$||w(j+1)||^2 - ||w(j)||^2 \leq 2T + Q$$

Proof of convergence

Adding these inequalities for j=1,2,…,k yields

$$||w(k+1)||^2 \leq ||w(1)||^2 + (2T+Q)k$$

Comparing this relation with the following which we derived earlier, i.e.,

$$||w(k+1)||^2 \geq \frac{[w'(1)w^* + kT]^2}{||w^*||^2}$$

we see that they establish conflicting bounds on $||w(k+1)||^2$ for sufficiently large k. In fact, k can be no larger than k_m which is the solution to the equation:

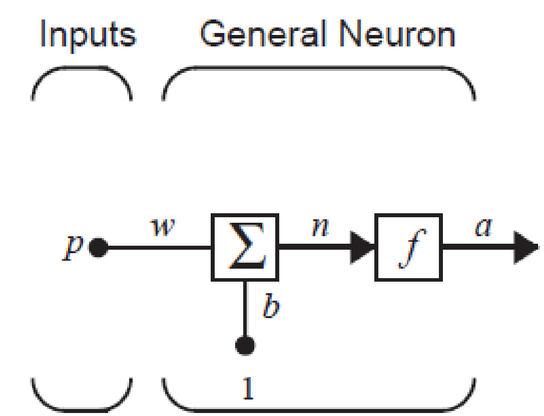
$$\frac{[w'(1)w^* + k_m T]^2}{||w^*||^2} = ||w(1)||^2 + (2T+Q)k_m$$

Therefore, k is finite. **Q.E.D.**

Activation functions

A single-input neuron

- The scalar *input p* is multiplied by the scalar *weight w* to form *wp*, one of the terms that is sent to the summer
- The other input, 1, is multiplied by a *bias b* and then passed to the summer
  (The bias is much like a weight, except that it has a constant input of 1. If you do not want to have a bias in a particular neuron, it can be omitted.)

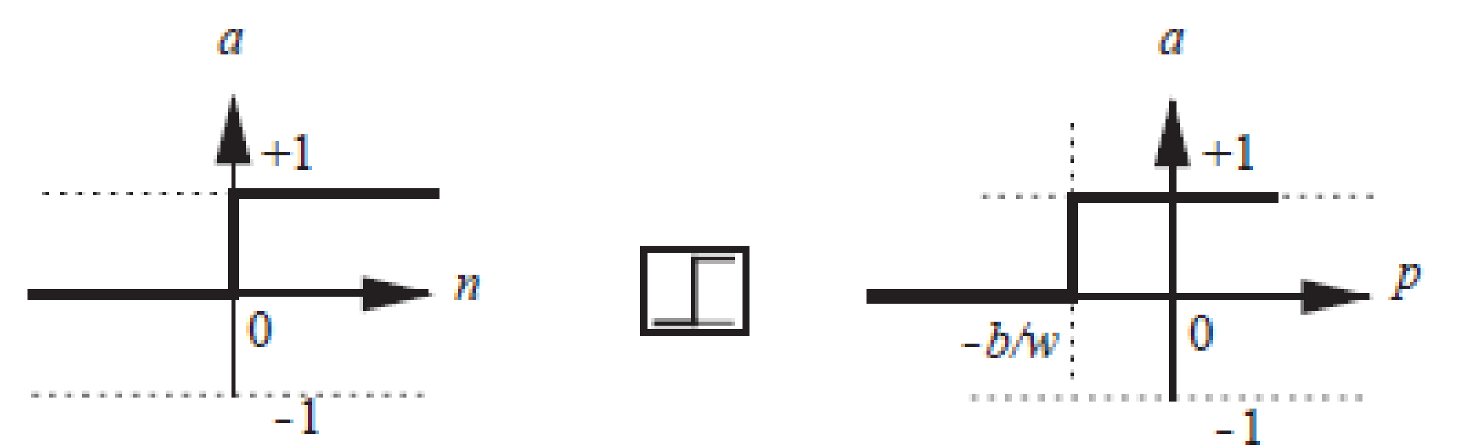


$a = f(wp + b)$

- The summer output *n*, often referred to as the *net input*, goes into a *transfer function f*, which produces the scalar neuron output *a*. (We may also use the term "*activation function*" rather than transfer function and "*offset*" rather than bias.)
- The transfer function is chosen by the designer, and then the parameters *w* and *b* will be adjusted by some learning rule
- If we relate this simple model back to the biological neuron, the weight corresponds to the *strength of a synapse*, the *cell body* is represented by the summation and the transfer function, and the neuron output represents the *signal on the axon*

The hard limit transfer function

- The *hard limit transfer function* sets the output of the neuron to 0 if the function argument is less than 0, or 1 if its argument is greater than or equal to 0
- We will use this function to create neurons that classify inputs into two distinct categories
- Observe the effect of the weight and the bias



$a = hardlim(n)$

Hard Limit Transfer Function

$a = hardlim(wp + b)$

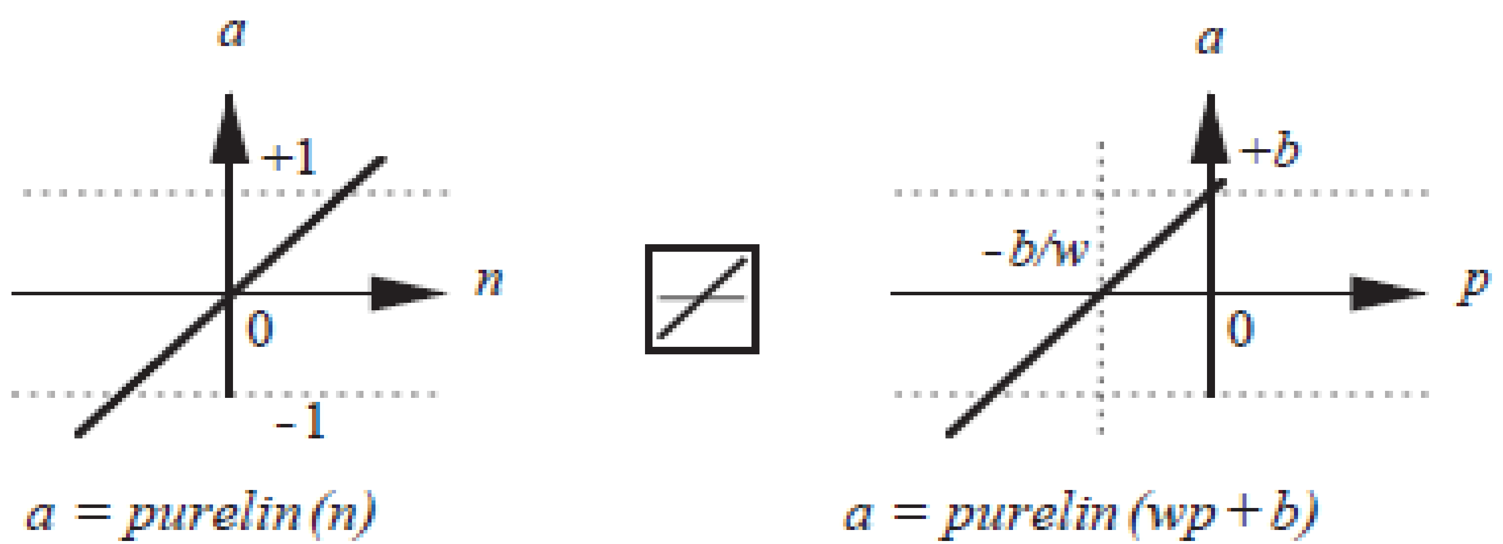
Single-Input *hardlim* Neuron

The linear transfer function

- The output of a *linear transfer function* is equal to its input:

$$\alpha = n$$

- Neurons with this transfer function are used in the ADALINE networks

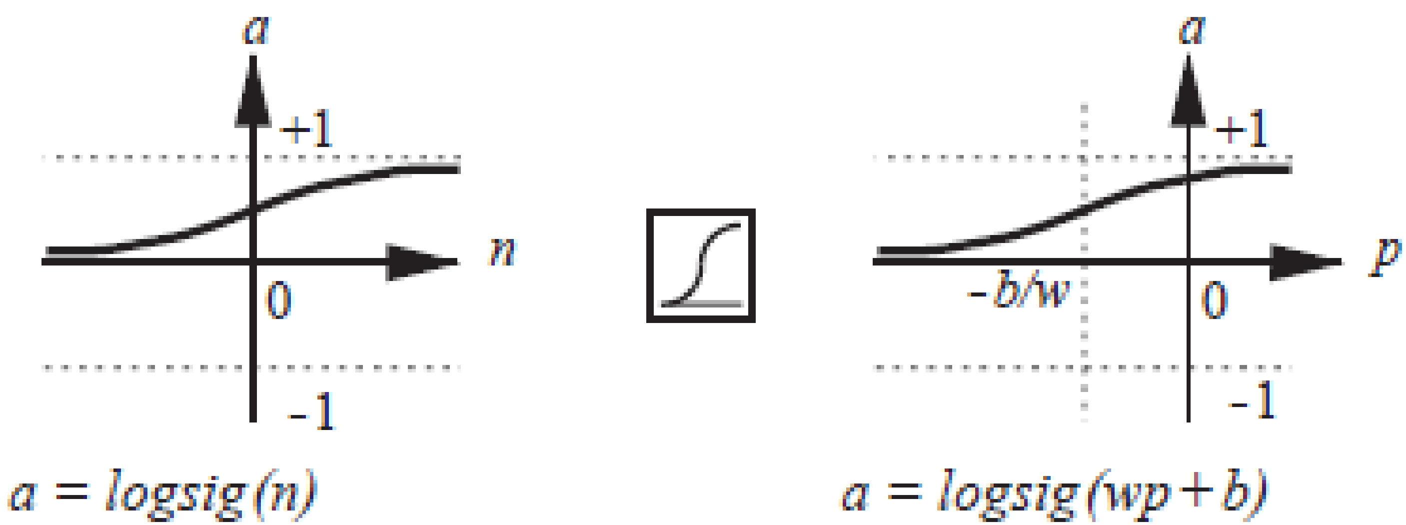


Linear Transfer Function

Single-Input *purelin* Neuron

The log-sigmoid transfer function

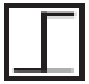
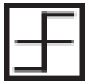
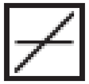
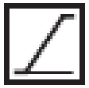
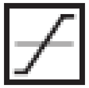
- The *log-sigmoid transfer function* takes the input (may have any value between +∞ and - ∞) and outputs into the range 0 to 1, according to: $$a = \frac{1}{1+e^{-n}}$$
- It is commonly used in multilayer networks that are trained using *backpropagation* (in part because it is differentiable)



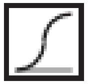
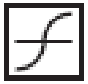
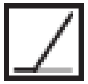
Log-Sigmoid Transfer Function

Single-Input *logsig* Neuron

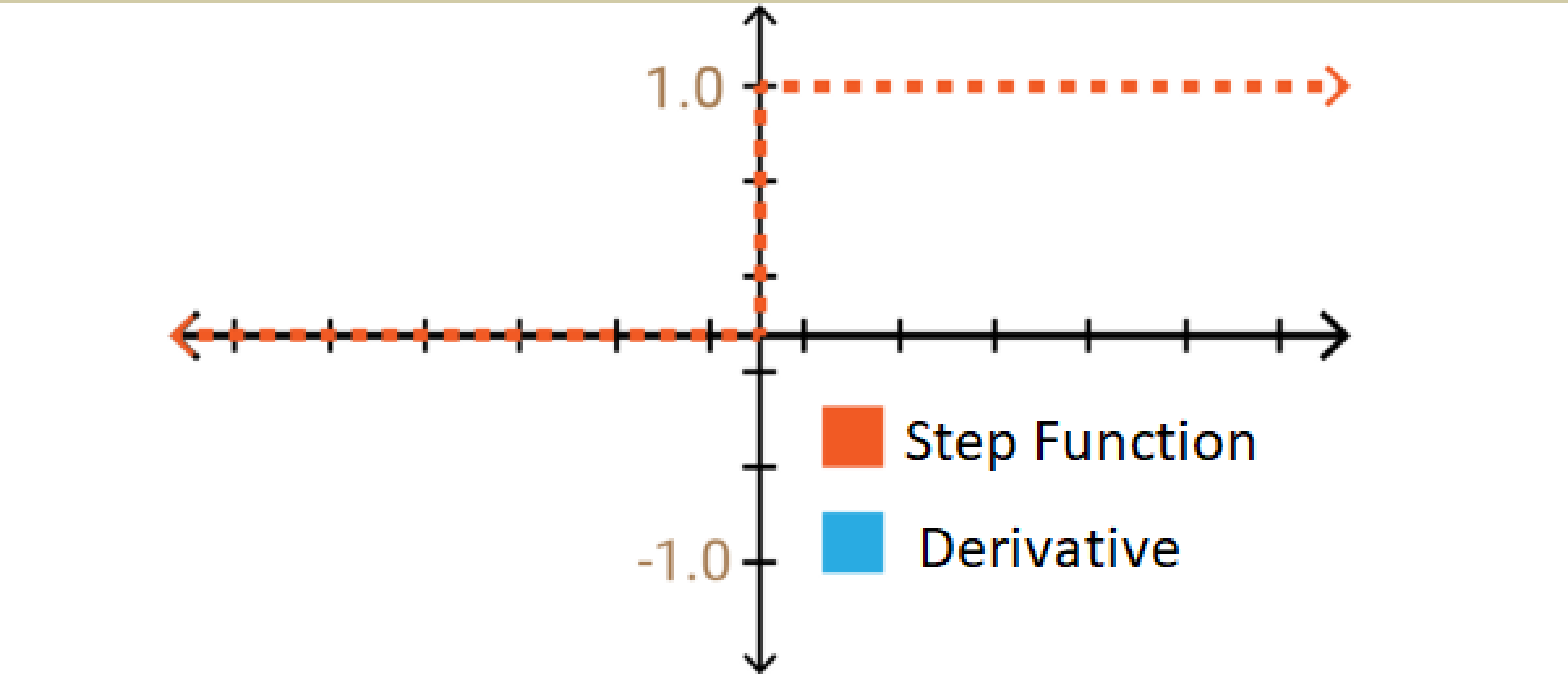
More transfer functions

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
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins

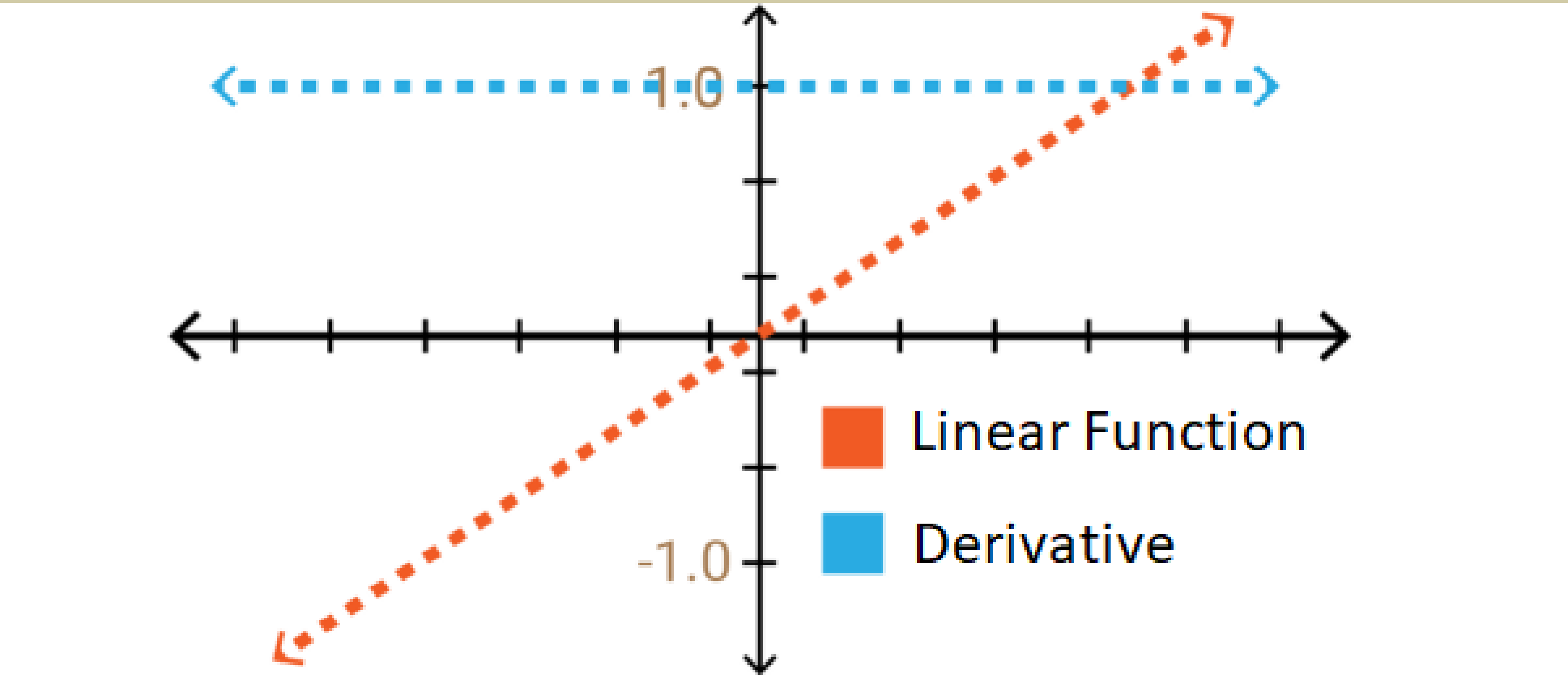
More transfer functions

Name	Input/Output Relation	Icon	MATLAB Function
Log-Sigmoid	$a = \dfrac{1}{1+e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \dfrac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons	$\boxed{C}$	compet

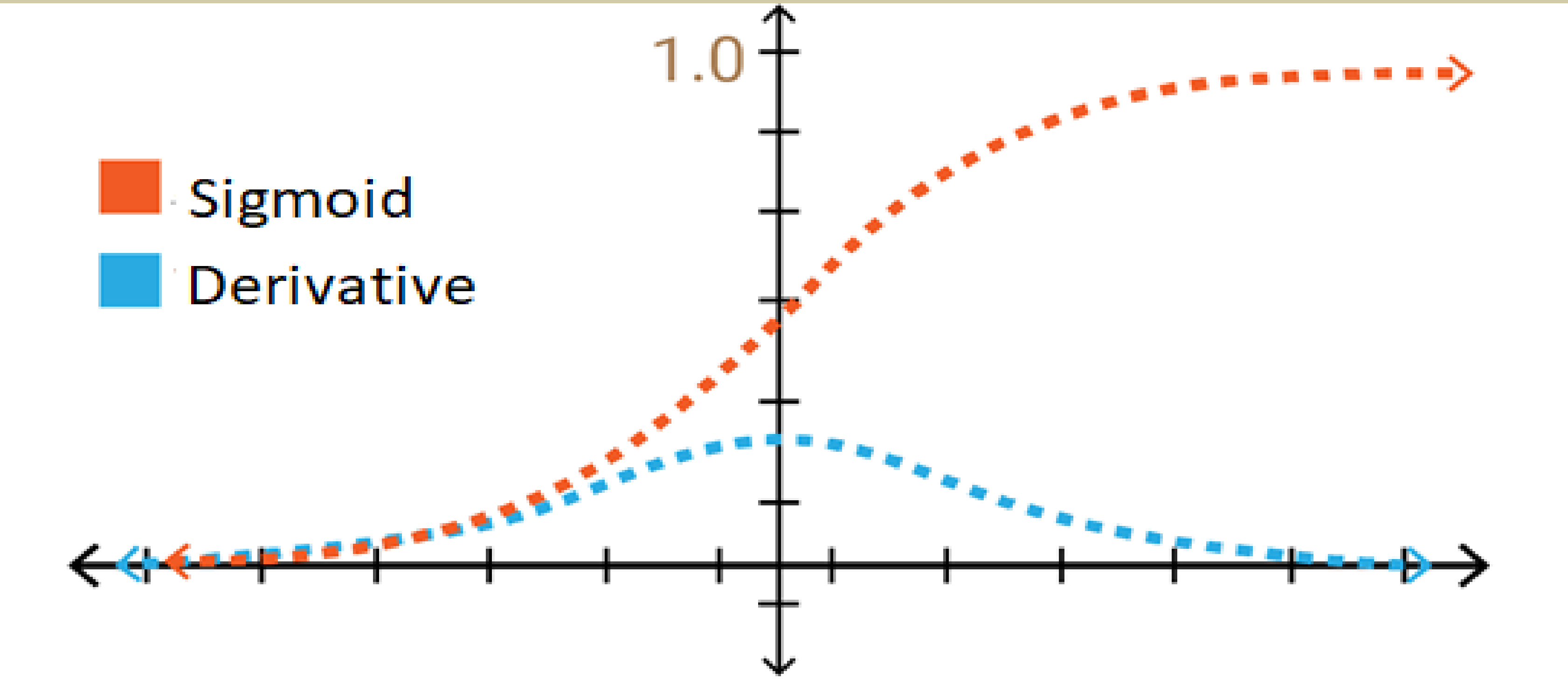
Derivatives of functions: *hardlim*



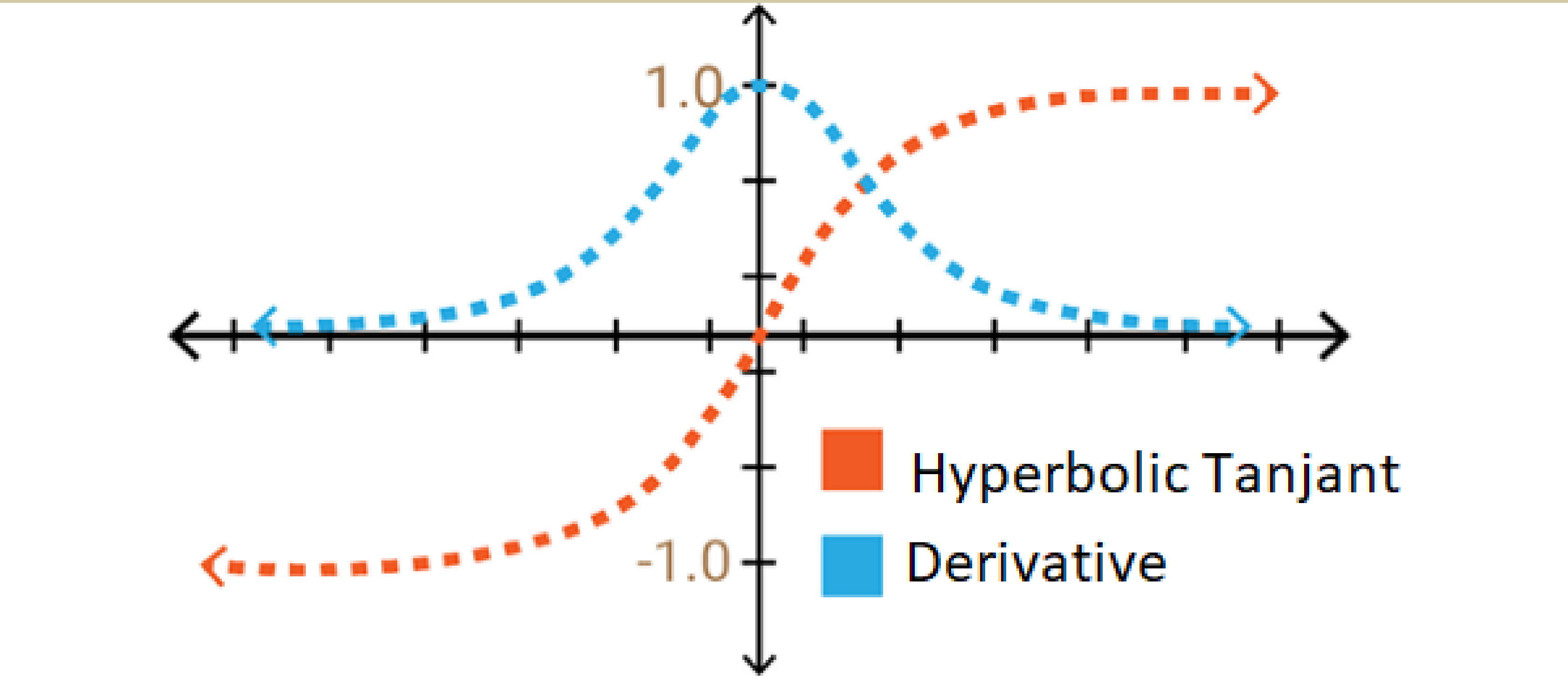
Derivatives of functions: *purelin*



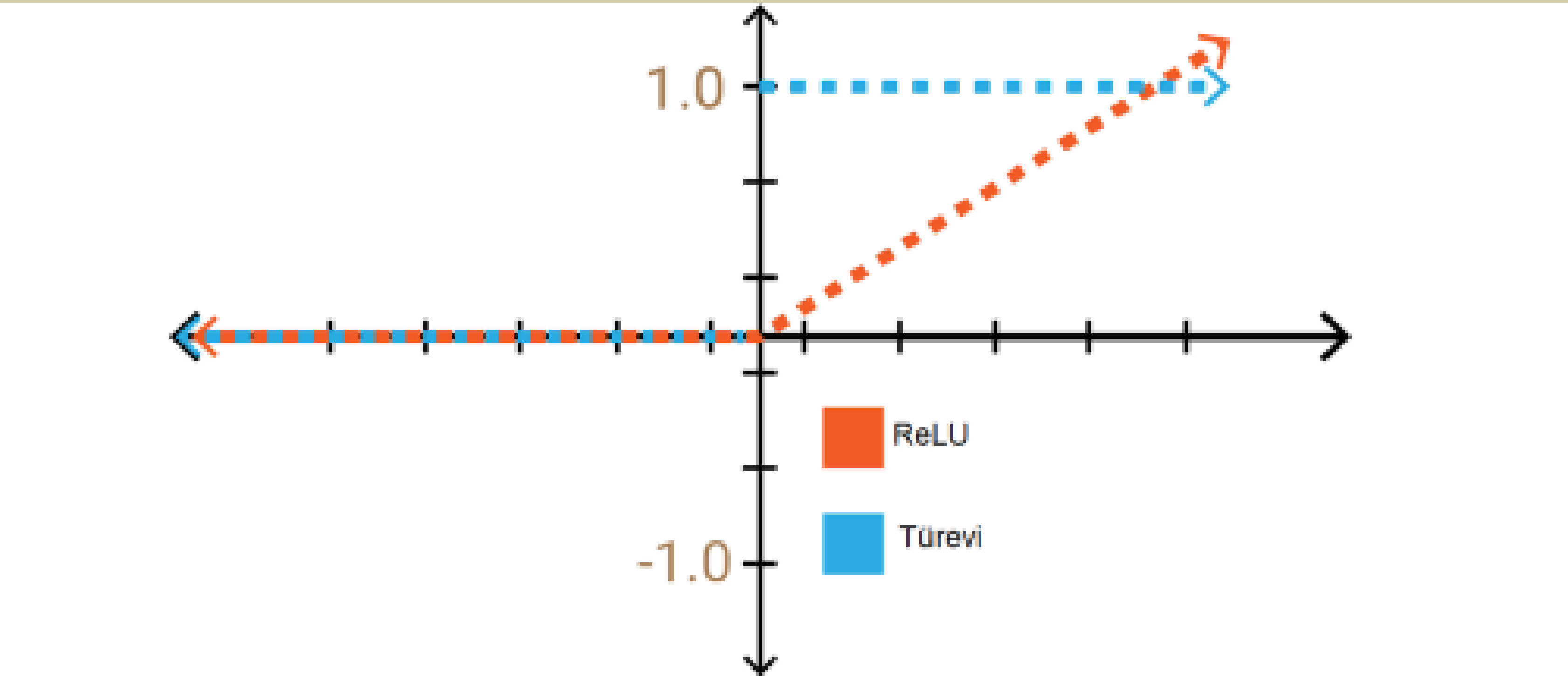
Derivatives of functions: *logsig*



Derivatives of functions: *tansig*

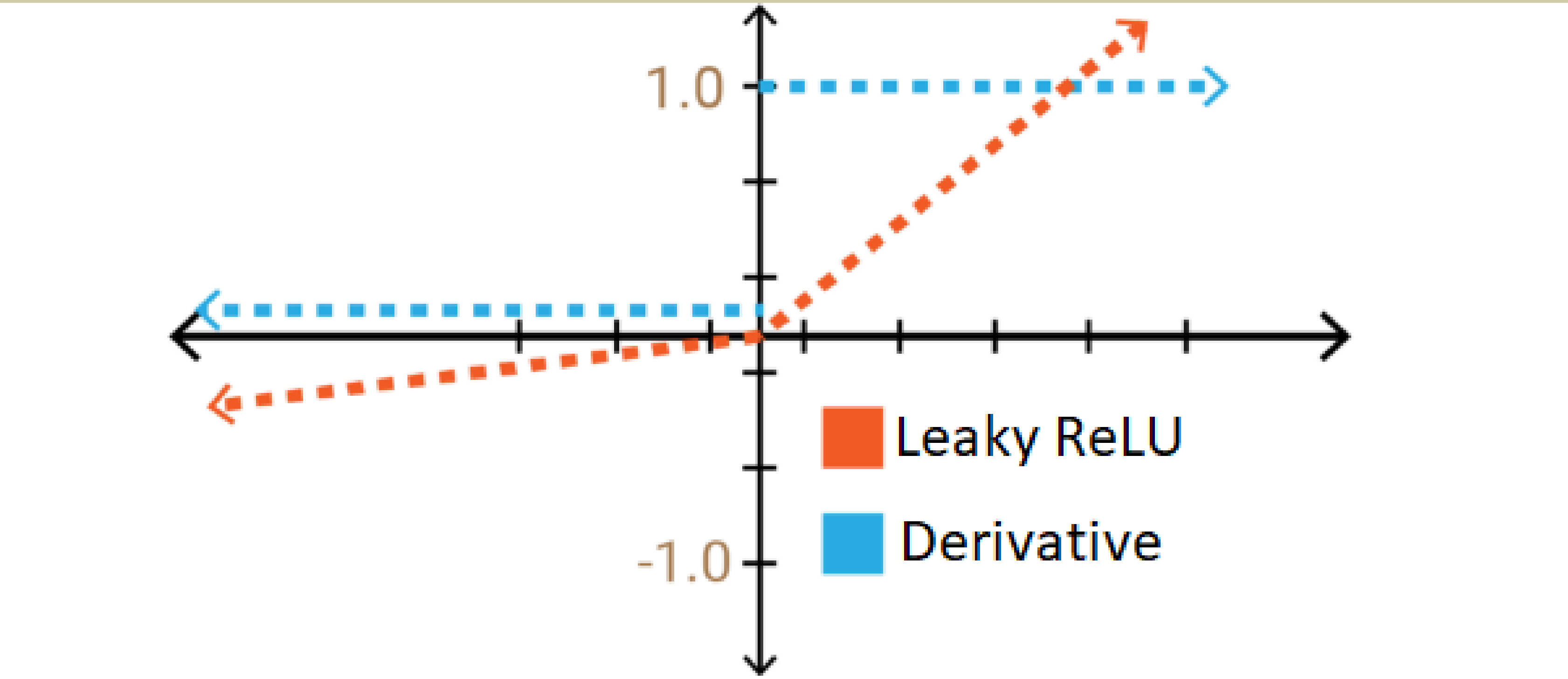


Derivatives of functions: *poslin* (*ReLU*)



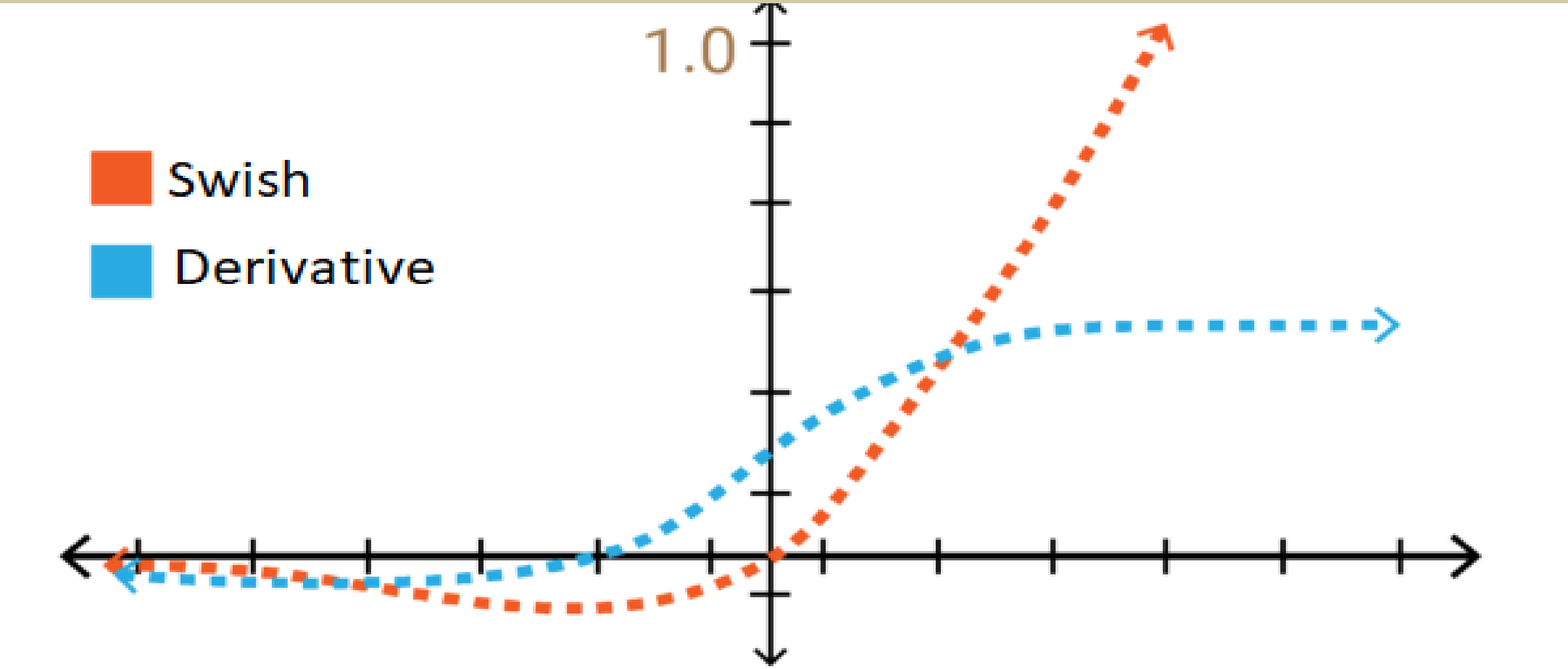
Derivatives of functions: *Leaky-ReLU*

ACTIVATION FUNCTION	EQUATION	RANGE
Leaky ReLU	$f(x) = \begin{cases} 0.01\, for\ x < 0 \\ x\ for\ x \geq 0 \end{cases}$	$(-\infty, \infty)$

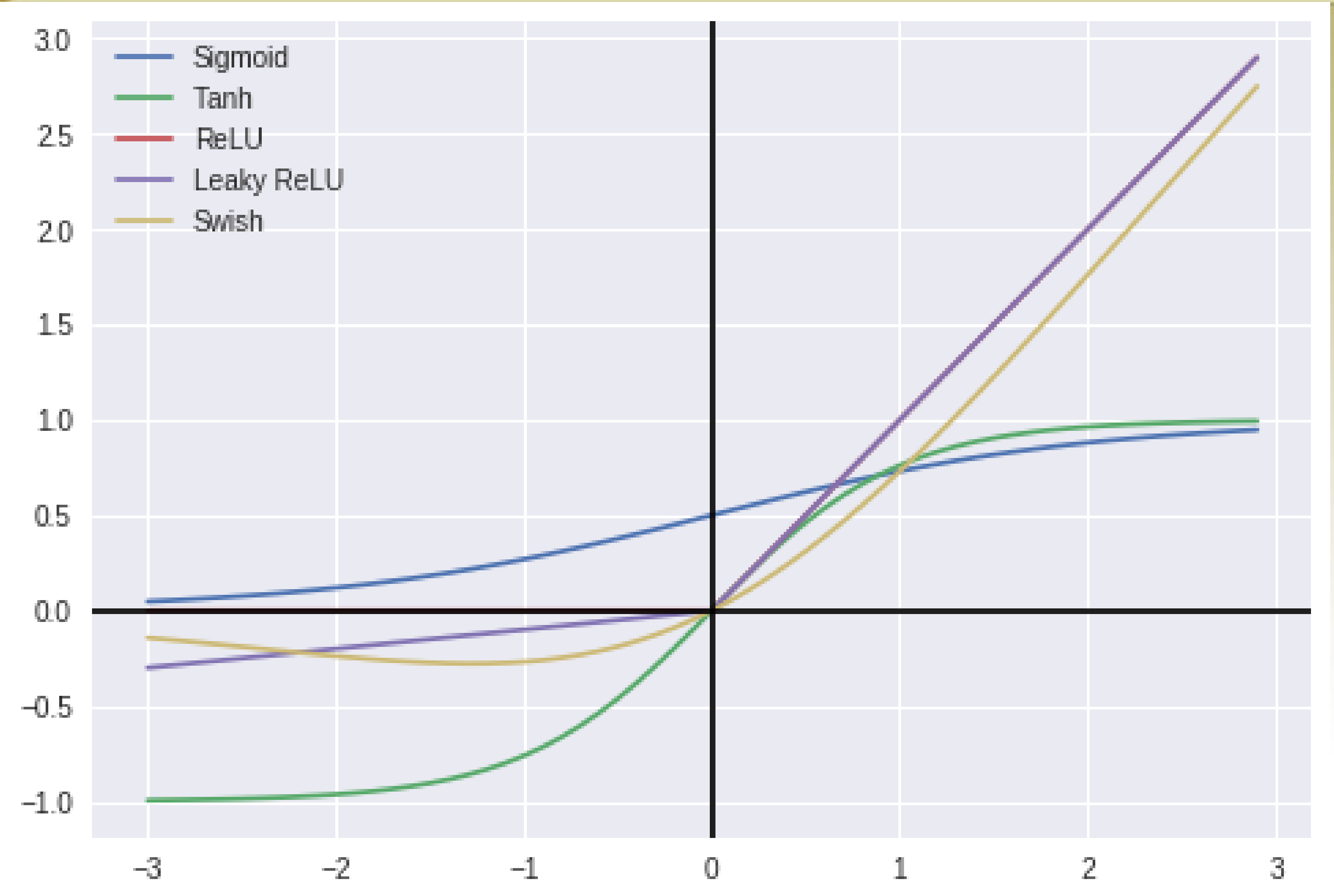


Derivatives of functions: *Swish*

ACTIVATION FUNCTION	EQUATION	RANGE
Swish Function	$f(x) = 2x\sigma(\beta x) = \begin{cases} \beta = 0 \; for \; f(x) = x \\ \beta \to \infty \; for \; f(x) = 2\max(0, x) \end{cases}$	$(-\infty, \infty)$

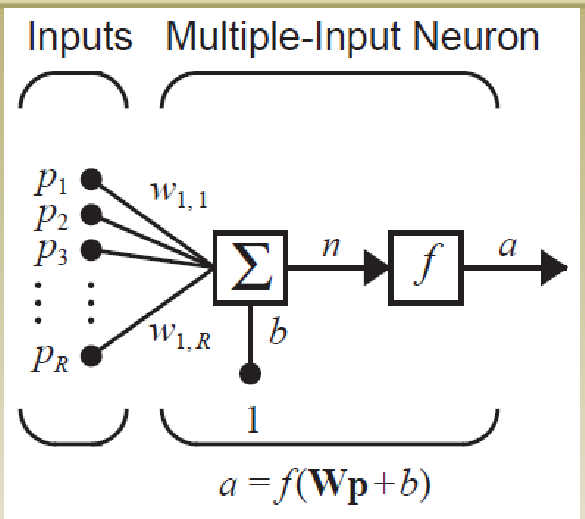


Popular activation functions

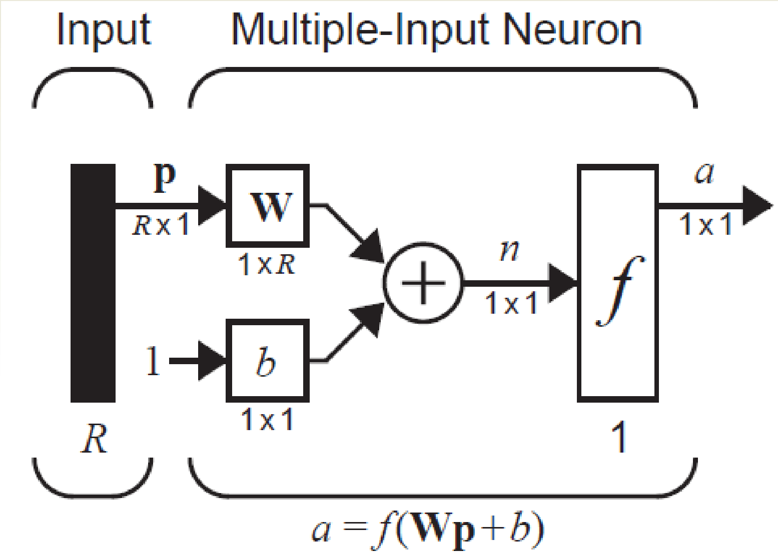


Multiple-input neuron

- The net input is $n = w_{1,1}, p_1 + w_{1,2} p_2 + \ldots + w_{1,R} p_R + b$
- In matrix form: $n = \mathbf{Wp} + b$, where matrix **W** for the single neuron case has only one row
- The neuron output can be written as: $\alpha = f(\mathbf{Wp} + b)$
- Notation: The first index indicates the particular neuron destination for that weight. The second index indicates the source of the signal fed to the neuron
  - Thus, the indices in $w_{1,2}$ say that this weight represents the connection to the first (and only) neuron from the second source

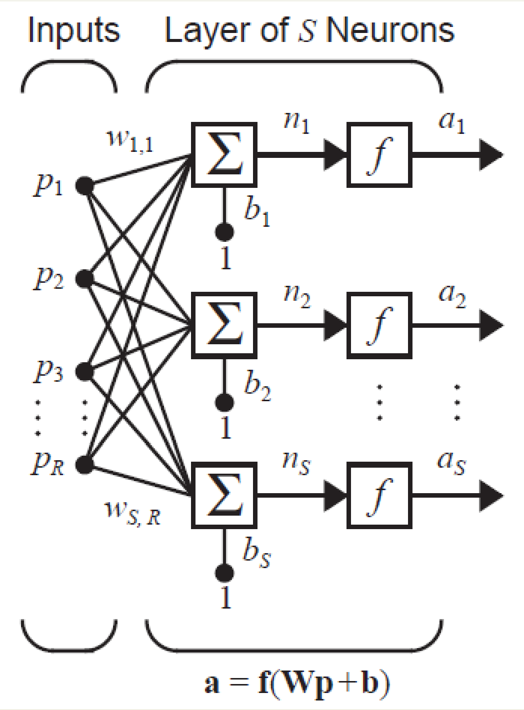


Abbreviated notation

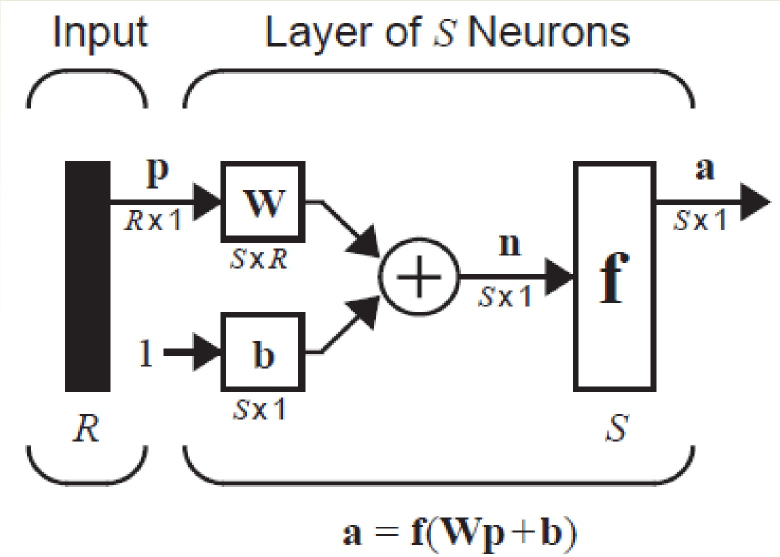


A layer of neurons

- A single-layer network of S neurons
  - each of the R inputs is connected to each of the neurons, and
  - the weight matrix now has S rows
- You might ask if all the neurons in a layer must have the same transfer function. The answer is **NO**
- **W=** $\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$

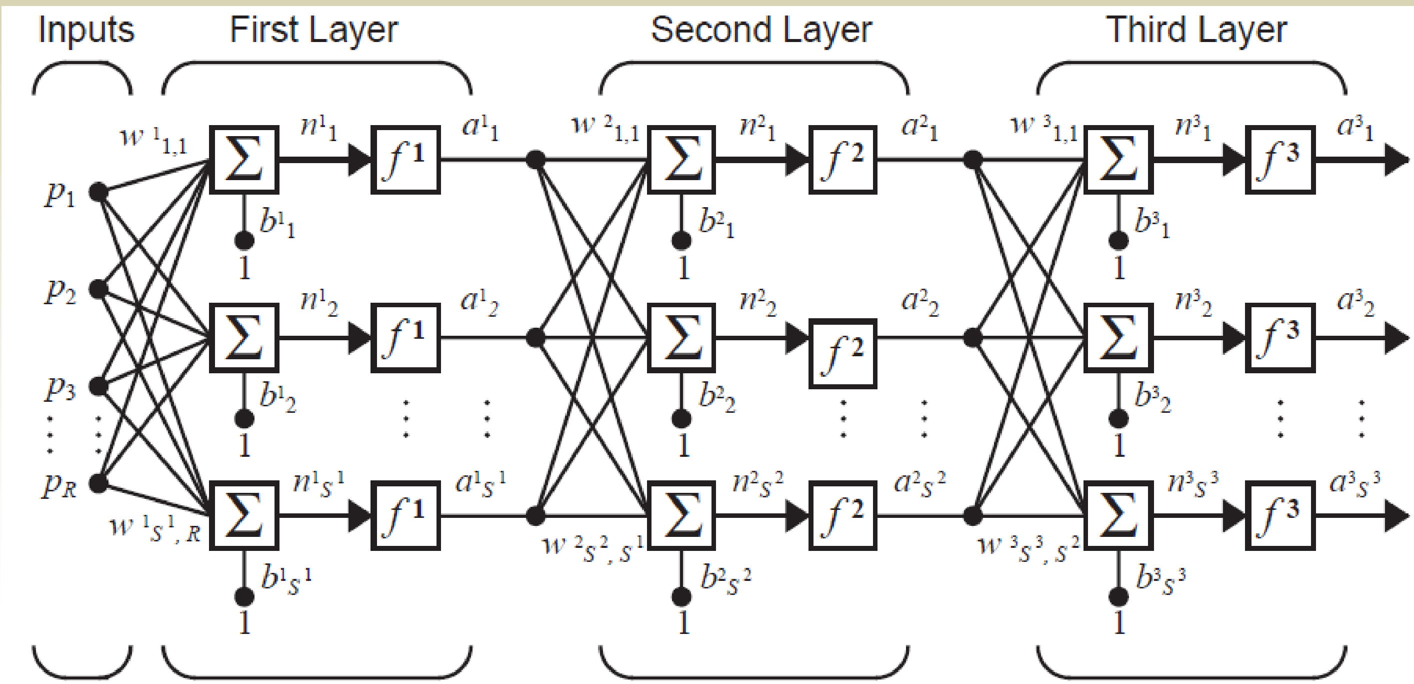


Abbreviated notation



Multiple layers of neurons

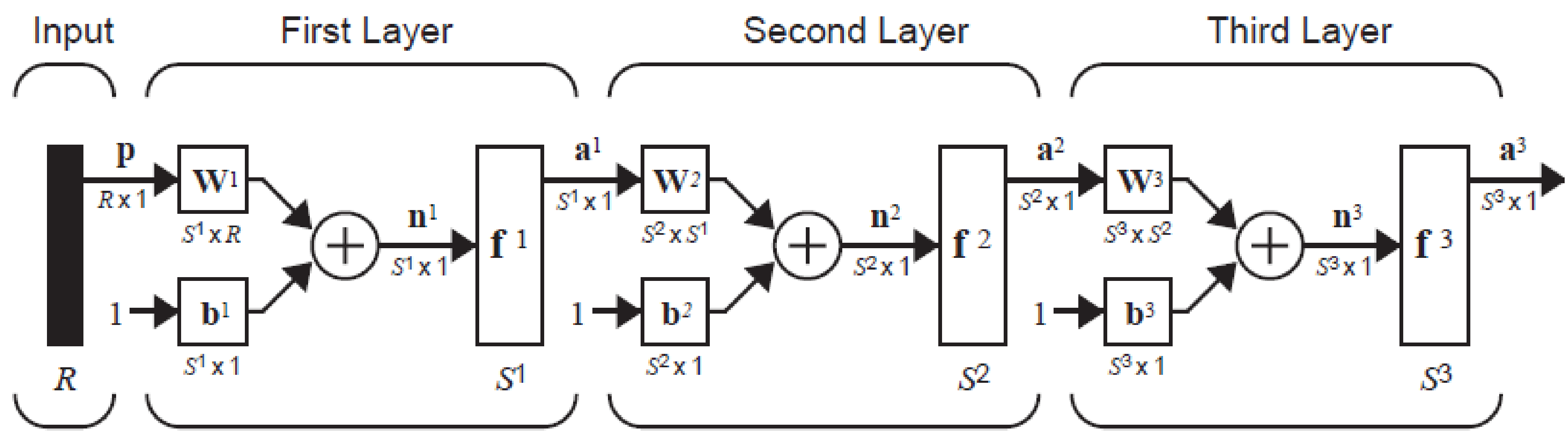
- Now consider a network with several layers. Each layer has its own weight matrix **W**, its own bias vector **b**, a net input vector **n** and an output vector **a**
- We may use superscripts to identify the layers. Specifically, we append the number of the layer as a superscript to the names for each of these variables



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

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

Multiple layers of neurons: Abbreviated notation



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

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

Perceptron for multiple classes

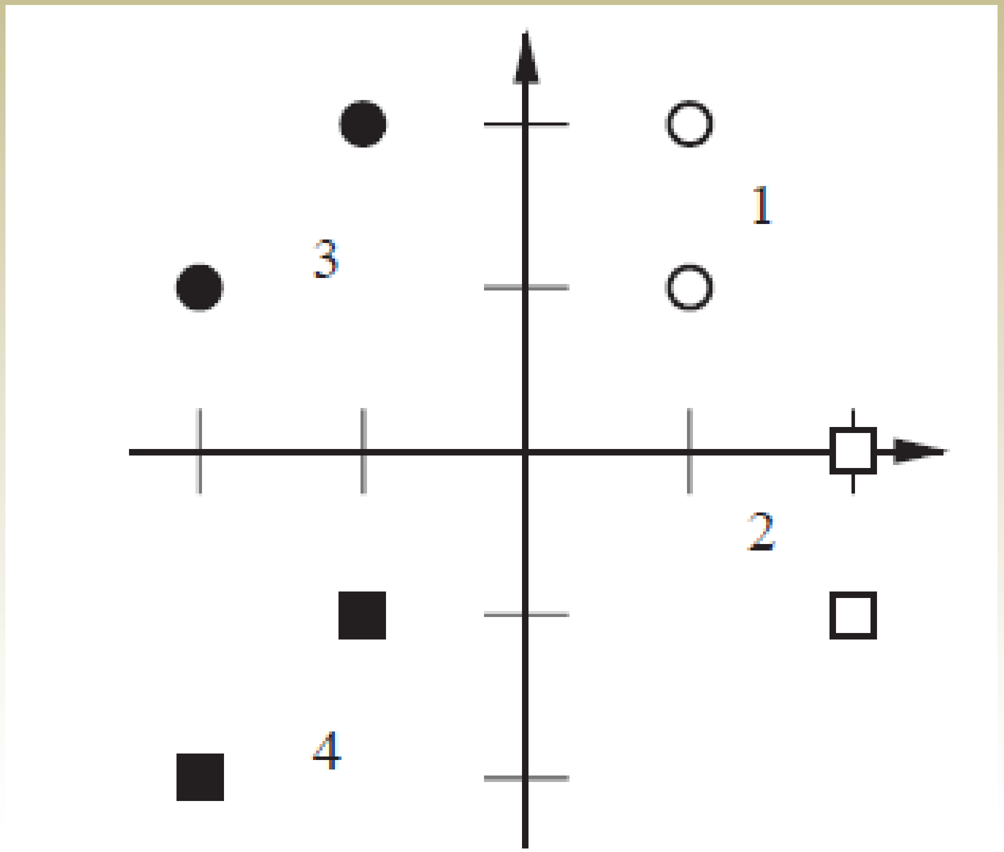
- Using the perceptron (training) rule to solve (during class lecture) the following classification problem:
  - Use *hardlim* as the activation function
  - and start with: $\mathbf{W}(0) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{b}(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$$\text{class 1:}\left\{\mathbf{p}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}\right\}, \text{class 2: }\left\{\mathbf{p}_3 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \mathbf{p}_4 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}\right\},$$

$$\text{class 3:}\left\{\mathbf{p}_5 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \mathbf{p}_6 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}\right\}, \text{class 4: }\left\{\mathbf{p}_7 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \mathbf{p}_8 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}\right\}$$

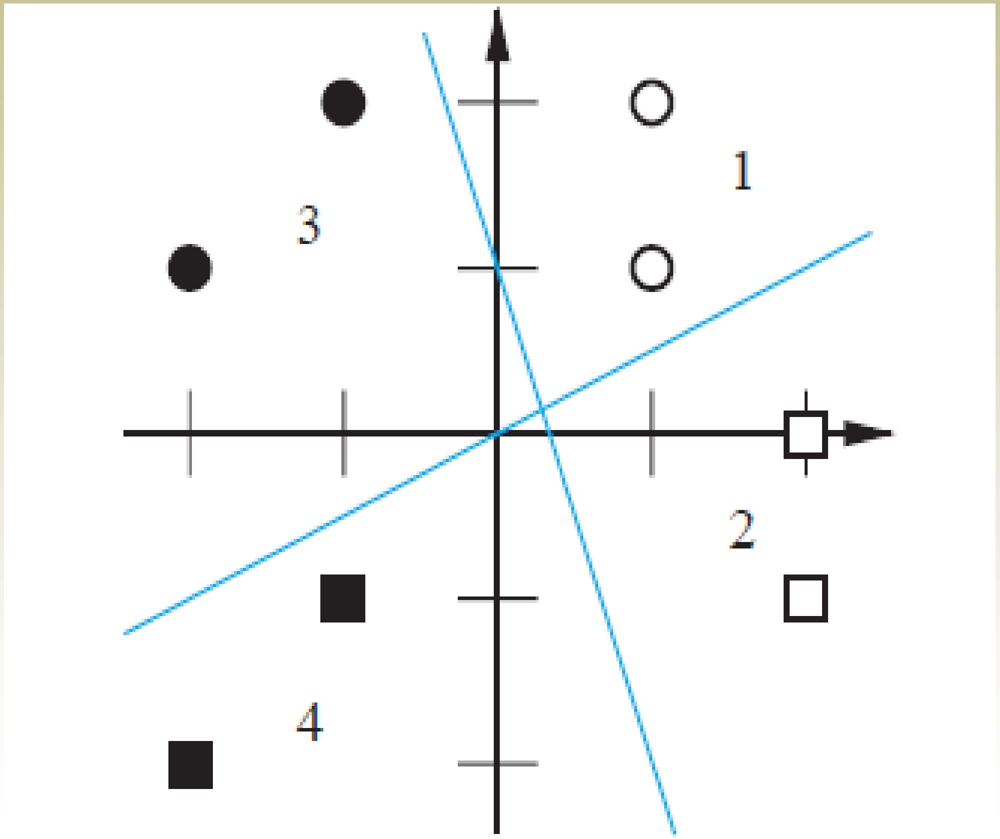
Perceptron for multiple classes

- Graphical illustration of the input



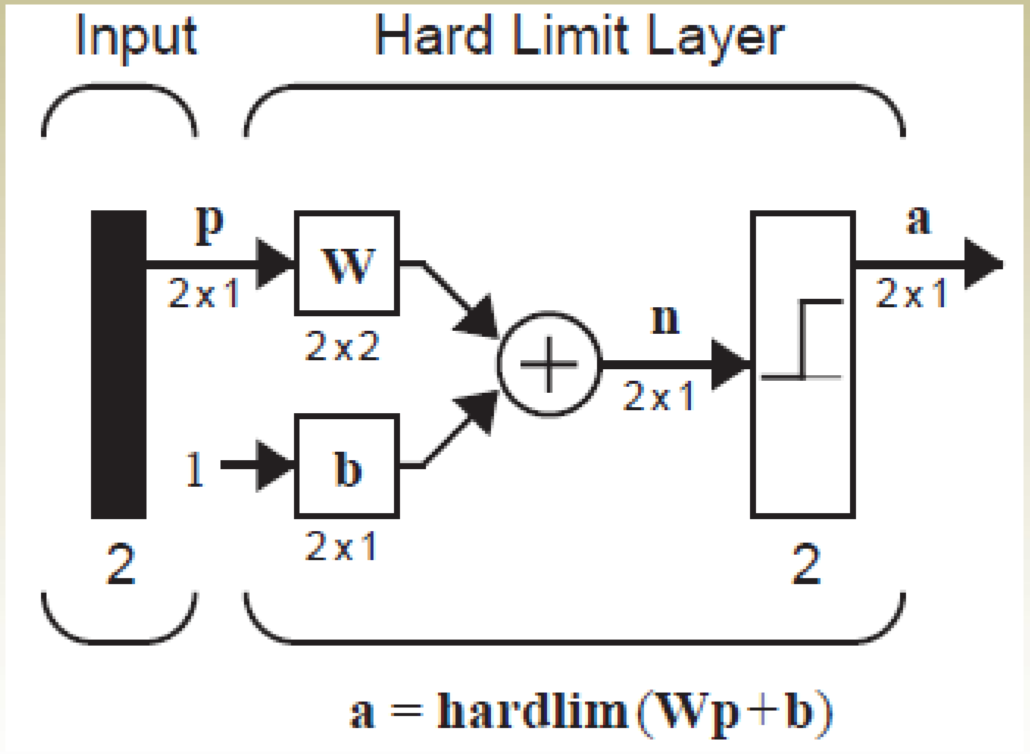
Perceptron for multiple classes

- Graphical illustration of the input



Perceptron for multiple classes

- The solution neural network is as follows:



Perceptron for multiple classes

- The solution decision boundaries