Machine Learning & Computer Vision
MAM4 2017 - 2018

Artificial Neural Networks

Frederic Precioso

UNIVERSITÉ CÔTE D'AZUR

Necessary mathematical concepts

UNIVERSITÉ CÔTE D'AZUR

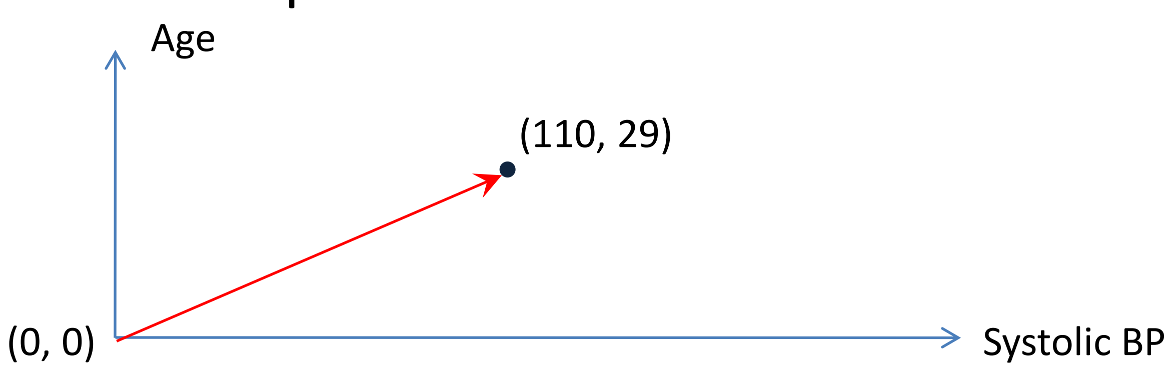
univ-cotedazur.fr

How to represent samples geometrically? Vectors & points in n-dimensional space ($\mathbb{R}^n$)

- Assume that a sample/patient is described by *n* characteristics ("features" or "variables")
- **Vector representation:** Every sample/patient is a vector in $\mathbb{R}^n$ with tail at point with 0 coordinates and arrow-head at point with the feature values.
- **Example:** Consider a patient described by 2 features:

  Systolic BP = 110 and *Age* = 29.

  This patient can be represented as a vector in $\mathbb{R}^2$:



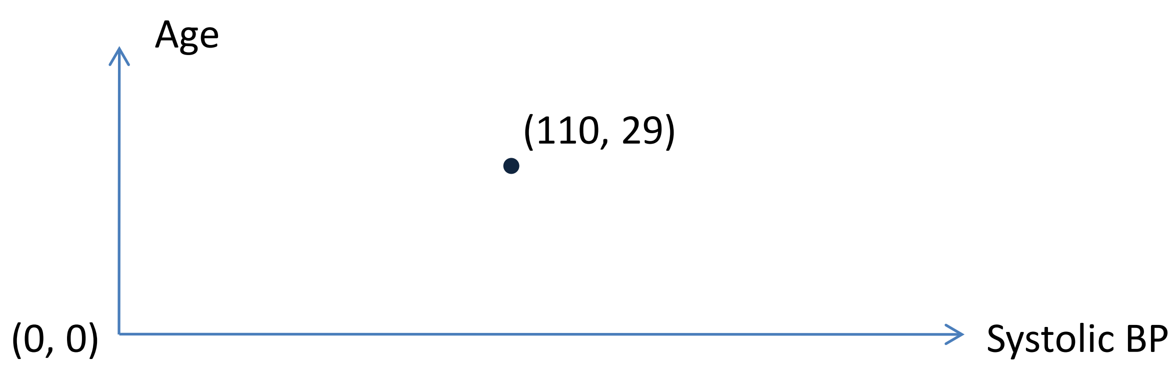
From "A Gentle Introduction to Support Vector Machines in Biomedicine", A. Statnikov, et al, AMIA 2010.

How to represent samples geometrically? Vectors & points in n-dimensional space ($\mathbb{R}^n$)

- **<u>Point representation:</u>** Every sample/patient is represented as a point in the n-dimensional space ($\mathbb{R}^n$) with coordinates given by the values of its features.
- **<u>Example:</u>** Consider a patient described by 2 features:

  Systolic BP = 110 and *Age* = 29.

  This patient can be represented as a point in $\mathbb{R}^2$:

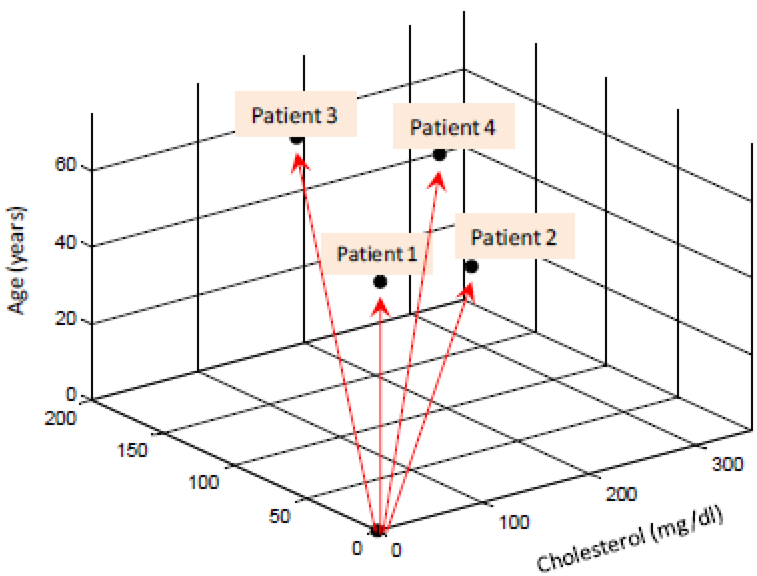


From "A Gentle Introduction to Support Vector Machines in Biomedicine", A. Statnikov, et al, AMIA 2010.

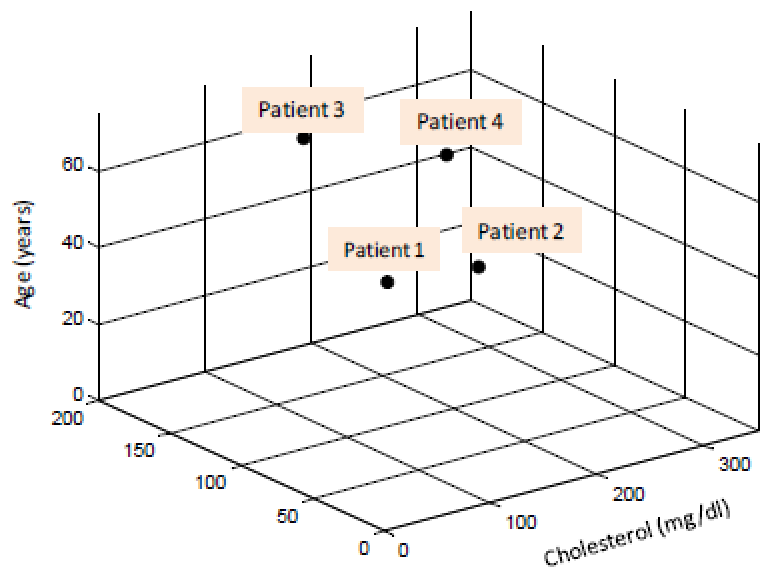
How to represent samples geometrically? Vectors & points in n-dimensional space ($\mathbb{R}^n$)

Patient id	*Cholesterol* (mg/dl)	*Systolic BP* (mmHg)	*Age* (years)	**Tail of the vector**	**Arrow-head of the vector**
1	150	110	35	(0,0,0)	(150, 110, 35)
2	250	120	30	(0,0,0)	(250, 120, 30)
3	140	160	65	(0,0,0)	(140, 160, 65)
4	300	180	45	(0,0,0)	(300, 180, 45)

Vector representation



Point representation



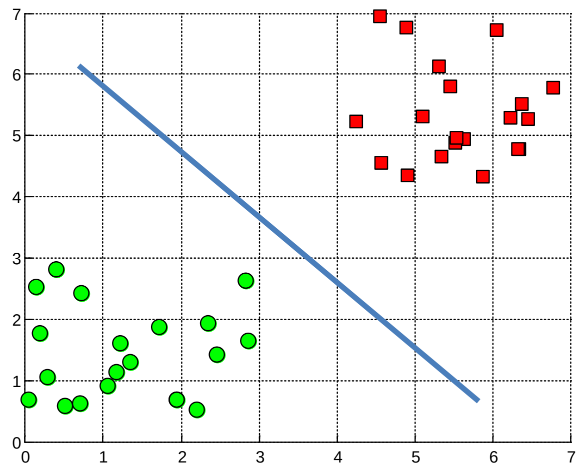
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

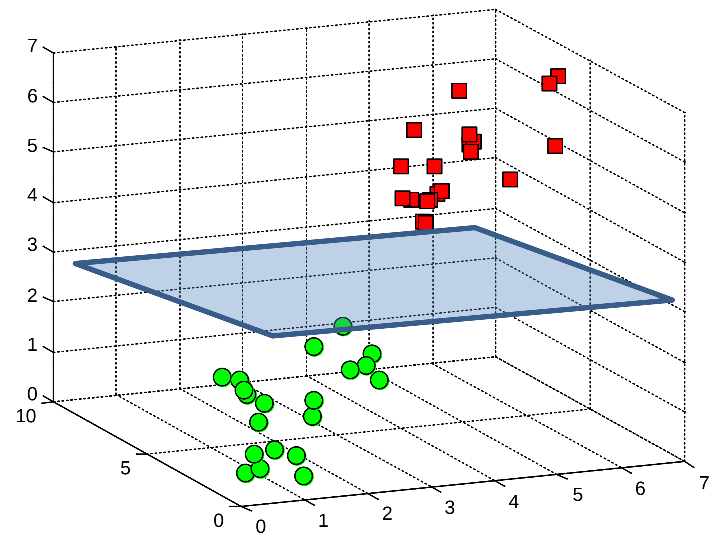
Purpose of vector representation

- Having represented each sample/patient allows now to geometrically represent the decision surface that separates two groups of samples/patients.

A decision surface in $\mathbb{R}^2$



A decision surface in $\mathbb{R}^3$



- In order to define the decision surface, we need to introduce some basic math elements…

From "A Gentle Introduction to Support Vector Machines in Biomedicine", A. Statnikov, et al, AMIA 2010.

Basic operation on vectors in $\mathbb{R}^n$

1. Multiplication by a scalar

Consider a vector $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and a scalar c

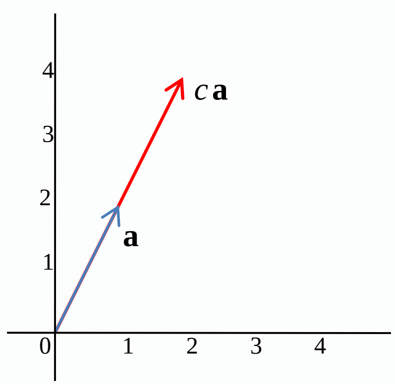
Define: $c\,\mathbf{a} = (ca_1, ca_2, \ldots, ca_n)$

When you multiply a vector by a scalar, you "stretch" it in the same or opposite direction depending on whether the scalar is positive or negative.

$\mathbf{a} = (1, 2)$

$c = 2$

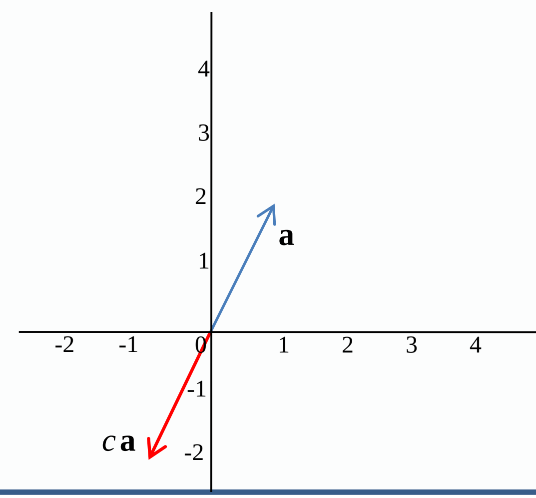
$c\,\mathbf{a} = (2, 4)$



$\mathbf{a} = (1, 2)$

$c = -1$

$c\,\mathbf{a} = (-1, -2)$



Basic operation on vectors in $\mathbb{R}^n$

2. Addition

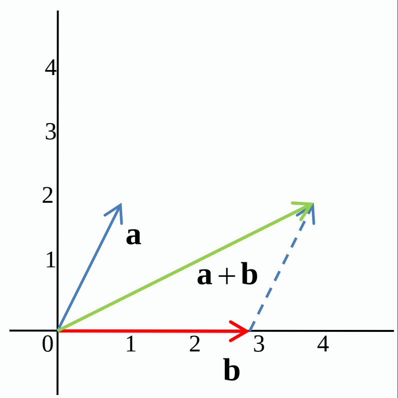
Consider vectors $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$

Define: $\mathbf{a} + \mathbf{b} = (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n)$

$\mathbf{a} = (1, 2)$

$\mathbf{b} = (3, 0)$

$\mathbf{a} + \mathbf{b} = (4, 2)$



Recall addition of forces in classical mechanics.

From "A Gentle Introduction to Support Vector Machines in Biomedicine", A. Statnikov, et al, AMIA 2010.

Basic operation on vectors in $\mathbb{R}^n$

3. Subtraction

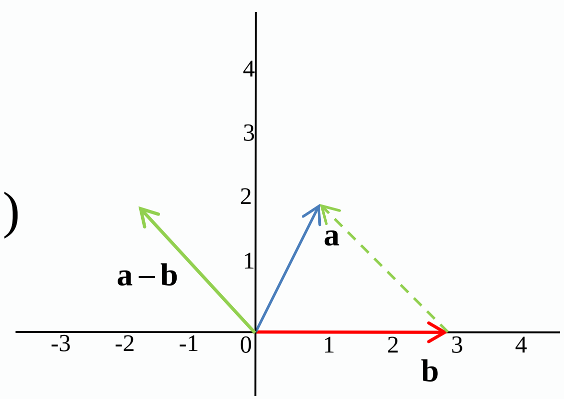
Consider vectors $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$

Define: $\mathbf{a} - \mathbf{b} = (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n)$

$\mathbf{a} = (1, 2)$

$\mathbf{b} = (3, 0)$

$\mathbf{a} - \mathbf{b} = (-2, 2)$



What vector do we need to add to $\vec{b}$ to get $\vec{a}$? I.e., similar to subtraction of real numbers.

From "A Gentle Introduction to Support Vector Machines in Biomedicine", A. Statnikov, et al, AMIA 2010.

Basic operation on vectors in $\mathbb{R}^n$

4. Euclidian length or L2-norm

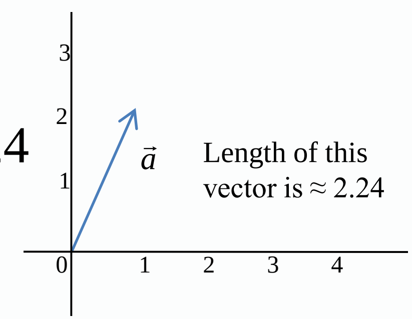
Consider a vector $\mathbf{a} = (a_1, a_2, \ldots, a_n)$

Define the L2-norm: $\|\mathbf{a}\|_2 = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$

We often denote the L2-norm without subscript, i.e. $\|\mathbf{a}\|$

$\mathbf{a} = (1, 2)$

$\|\mathbf{a}\|_2 = \sqrt{5} \approx 2.24$



L2-norm is a typical way to measure length of a vector; other methods to measure length also exist.

From "A Gentle Introduction to Support Vector Machines in Biomedicine", A. Statnikov, et al, AMIA 2010.

Basic operation on vectors in $\mathbb{R}^n$

5. Dot product

Consider vectors $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$

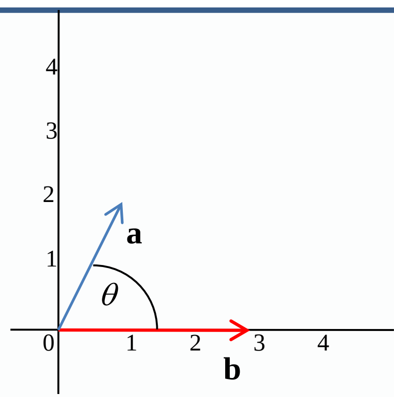
Define dot product: $\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n = \sum_{i=1}^{n} a_i b_i$

The law of cosines says that $\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 \cos\theta$ where θ is the angle between $\mathbf{a}$ and $\mathbf{b}$. Therefore, when the vectors are perpendicular $\mathbf{a} \cdot \mathbf{b} = 0$.

$\mathbf{a} = (1, 2)$

$\mathbf{b} = (3, 0)$

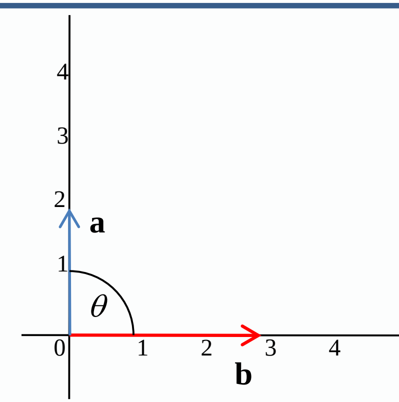
$\mathbf{a} \cdot \mathbf{b} = 3$



$\mathbf{a} = (0, 2)$

$\mathbf{b} = (3, 0)$

$\mathbf{a} \cdot \mathbf{b} = 0$



Basic operation on vectors in $\mathbb{R}^n$

5. Dot product (continued)

$$\mathbf{a}\cdot\mathbf{b} = a_1b_1 + a_2b_2 + \ldots + a_nb_n = \sum_{i=1}^{n} a_ib_i$$

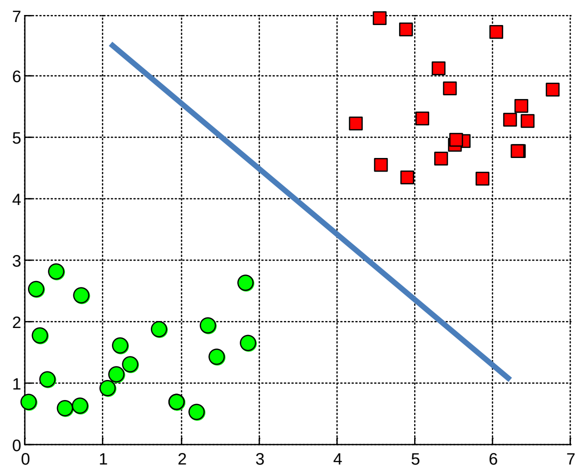
- Property: $\mathbf{a}\cdot\mathbf{a} = a_1a_1 + a_2a_2 + \ldots + a_na_n = \|\mathbf{a}\|_2^2$
- In the classical regression equation $y = \mathbf{w}\cdot\mathbf{x} + b$ the response variable *y* is just a dot product of the vector representing patient characteristics ($\mathbf{x}$) and the regression weights vector ($\mathbf{w}$) which is common across all patients plus an offset b.

From "A Gentle Introduction to Support Vector Machines in Biomedicine", A. Statnikov, et al, AMIA 2010.

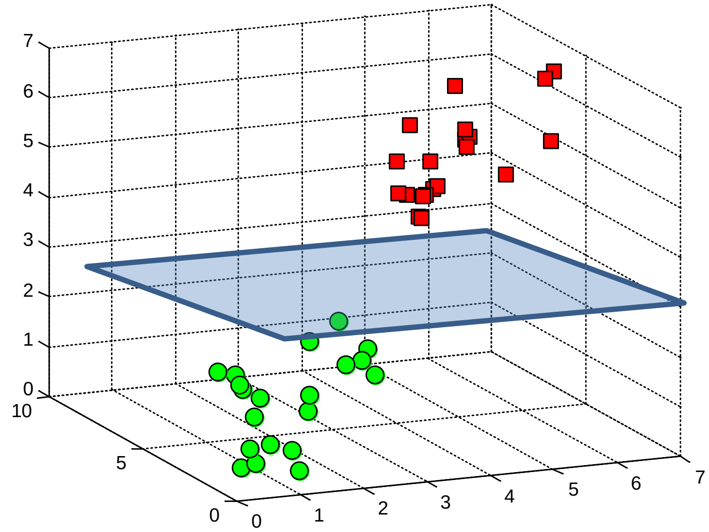
Hyperplanes as decision surfaces

- A hyperplane is a linear decision surface that splits the space into two parts;
- It is obvious that a hyperplane is a binary classifier.

A hyperplane in $\mathbb{R}^2$ is a line

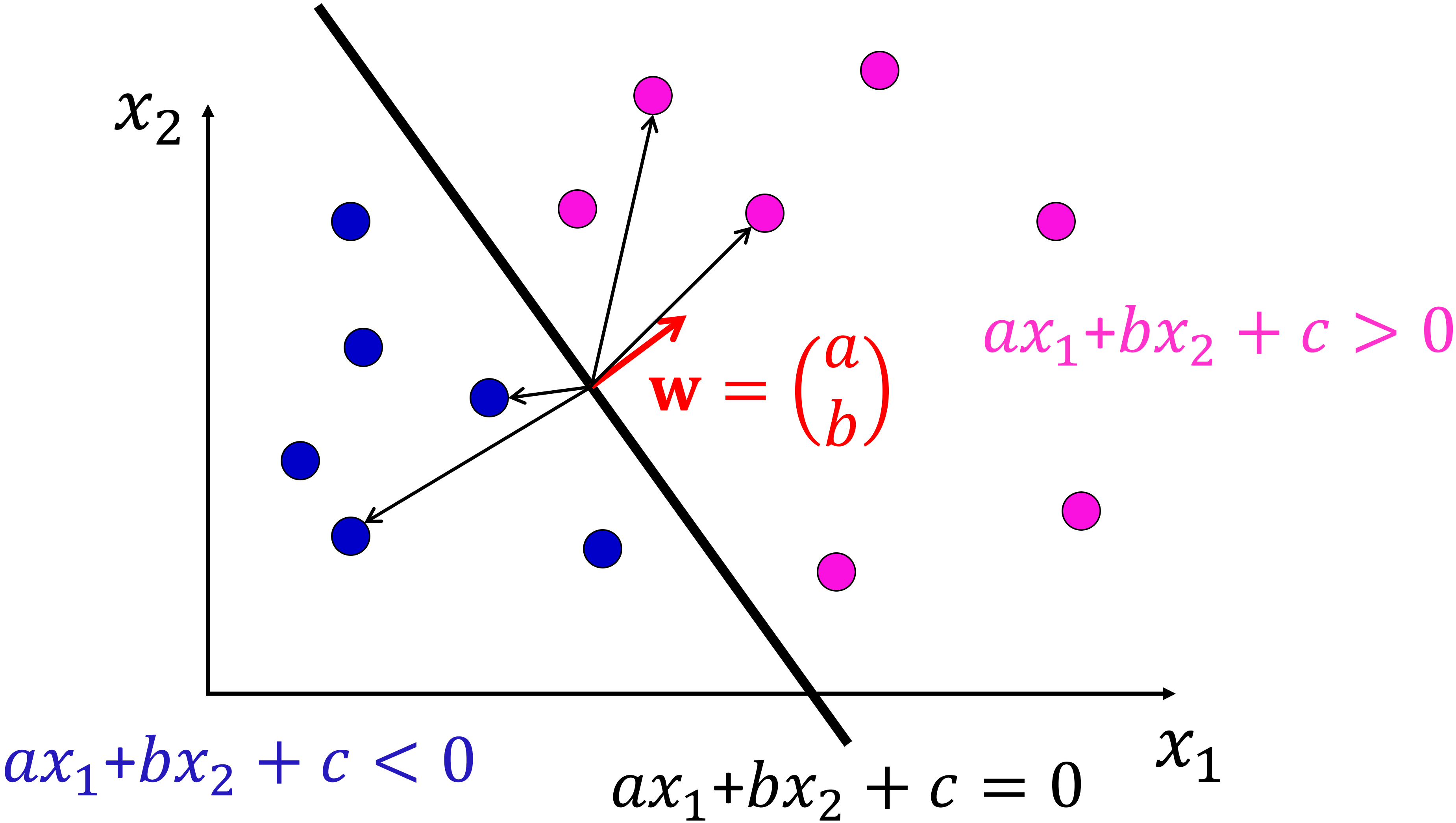


A hyperplane in $\mathbb{R}^3$ is a plane

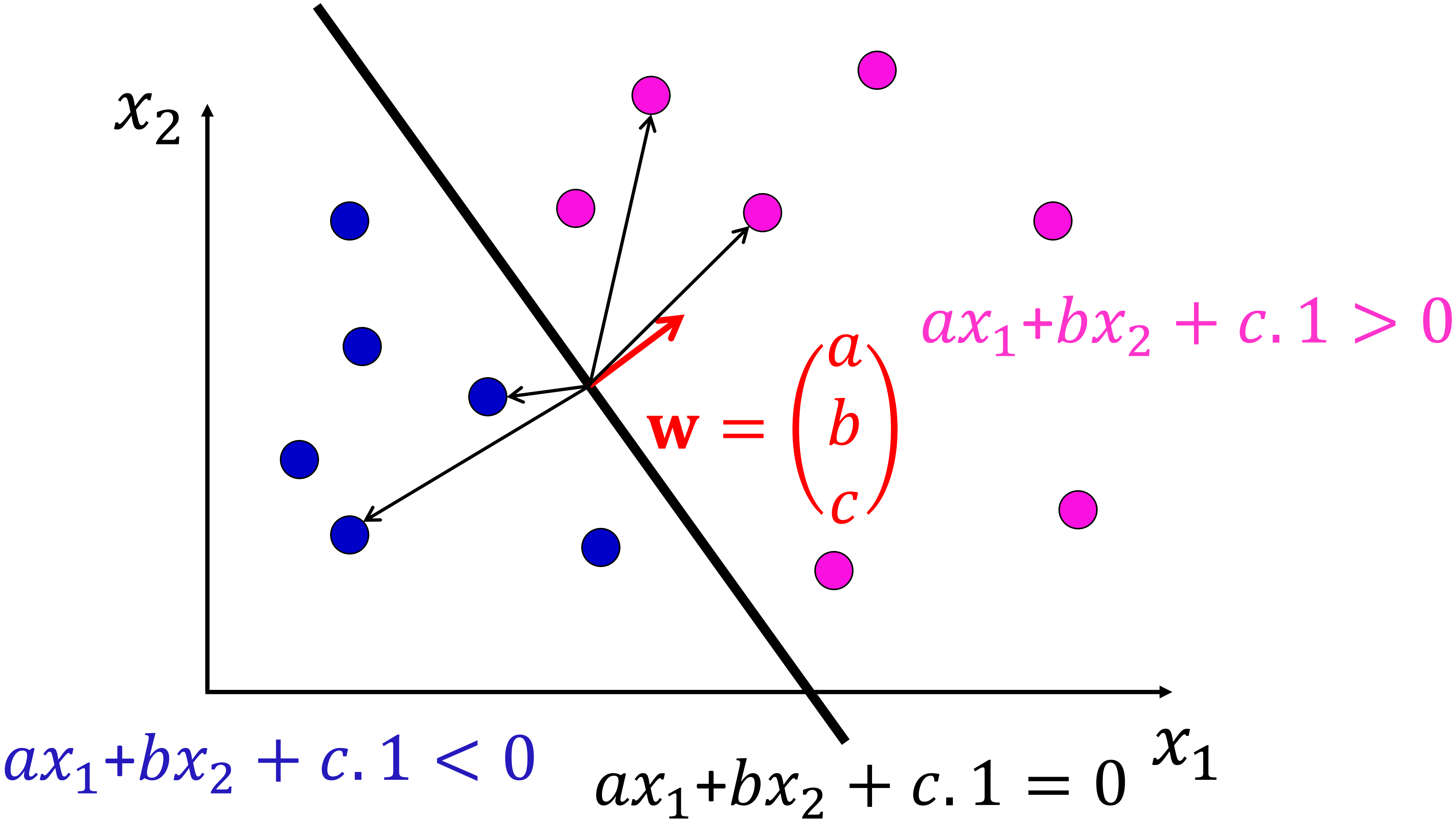


A hyperplane in $\mathbb{R}^n$ is an n-1 dimensional subspace

Geometry and Algebra



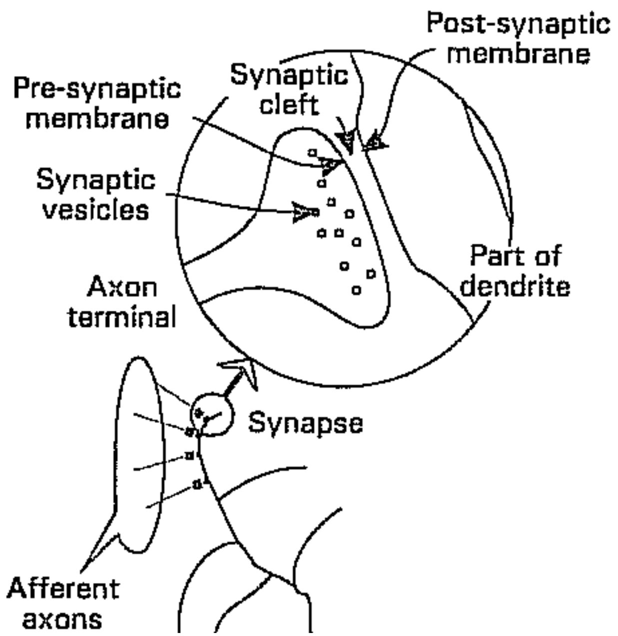
Convention


x_2
x_1
$\mathbf{w} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$
$ax_1 + bx_2 + c.1 > 0$
$ax_1 + bx_2 + c.1 < 0$
$ax_1 + bx_2 + c.1 = 0$

Initial Model: Perceptron

Biological neuron

- Before we study artificial neurons, let’s look at a biological neuron



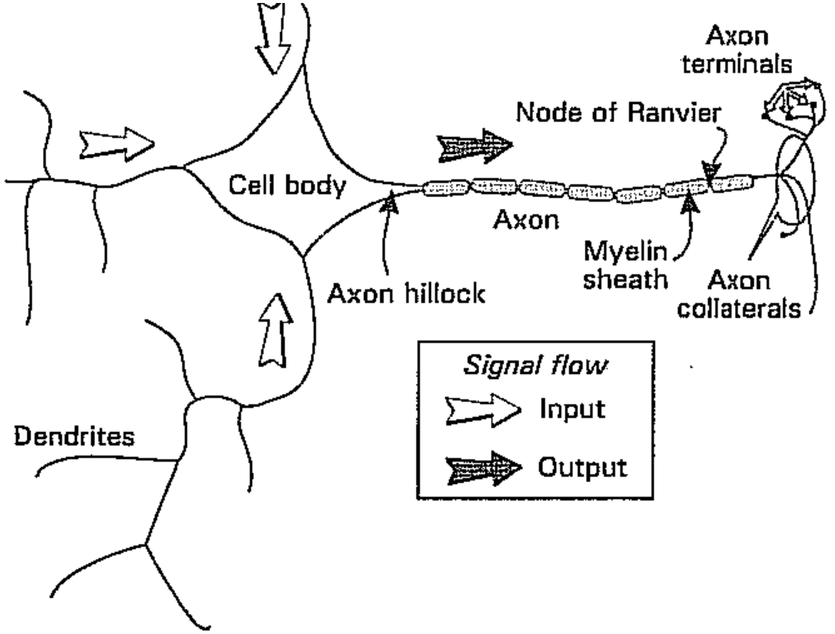
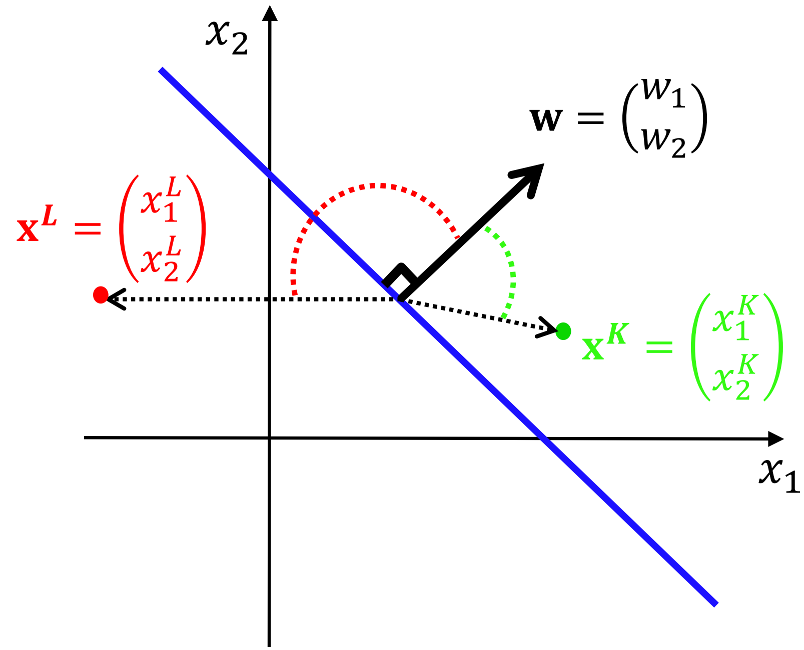


Figure from K.Gurney, An Introduction to Neural Networks

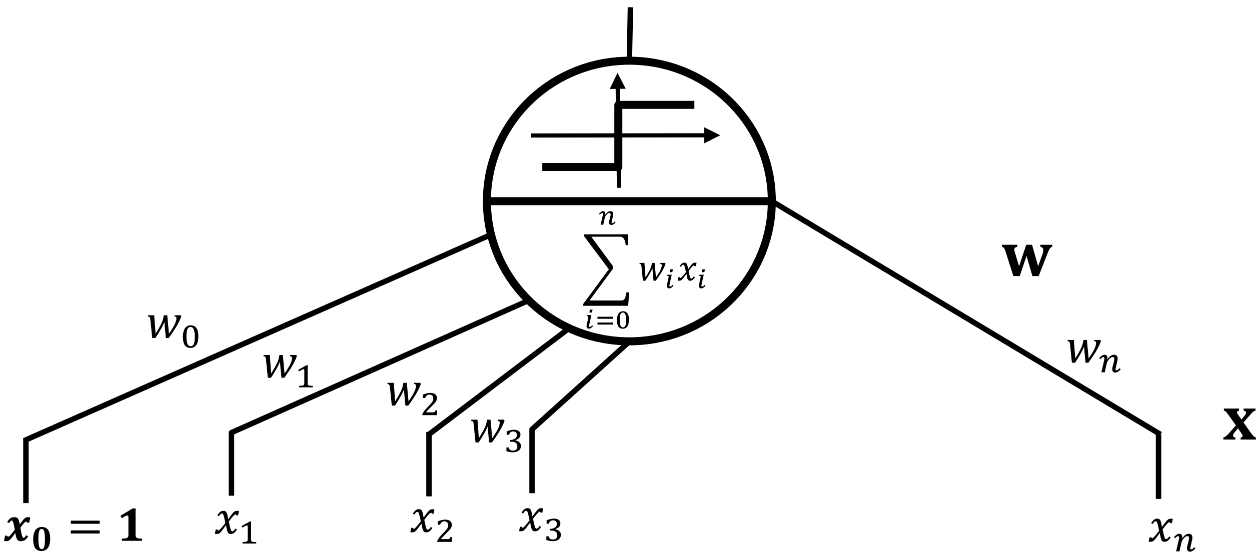
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

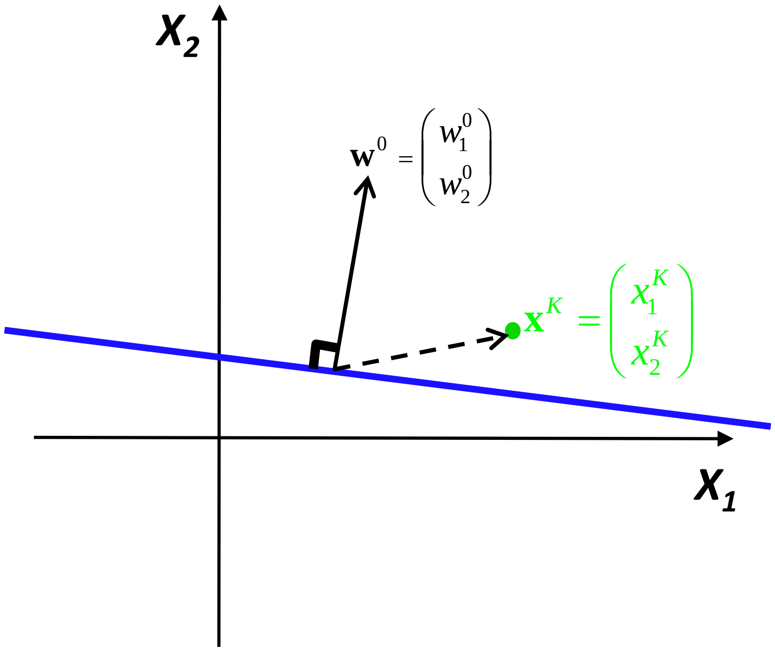
Single Perceptron Unit



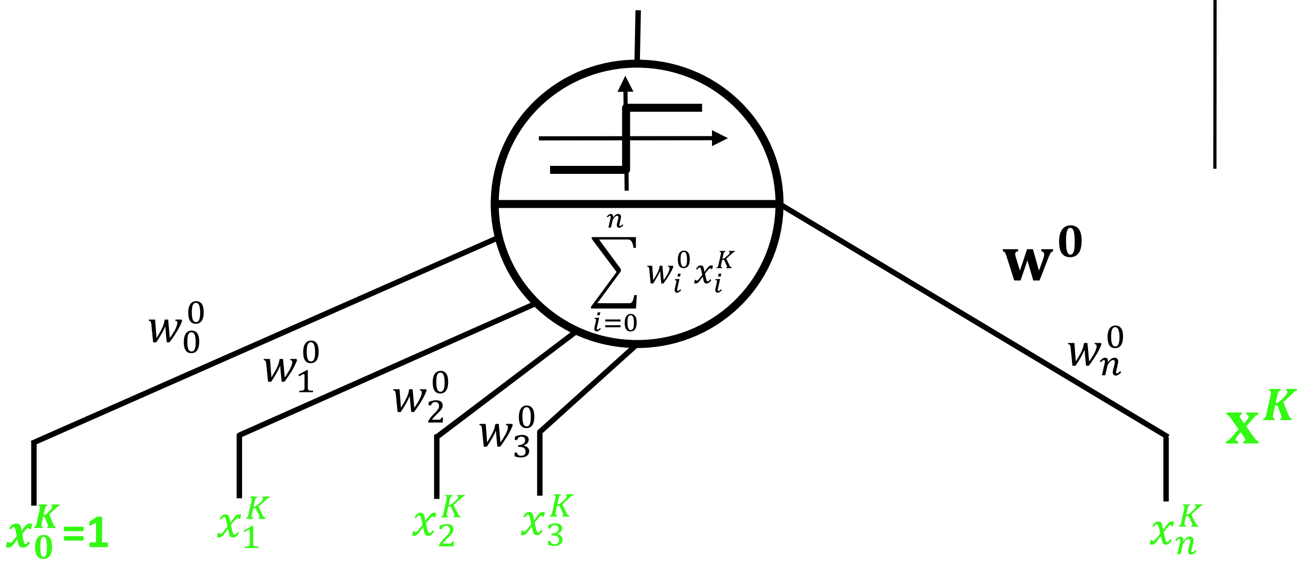
$$y = \text{sign}(\mathbf{w.x})$$



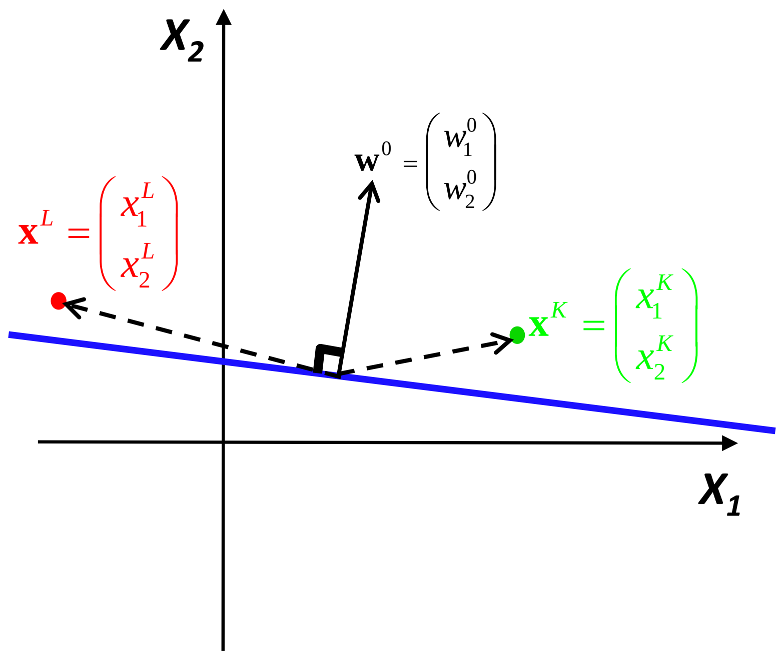
Single Perceptron Unit



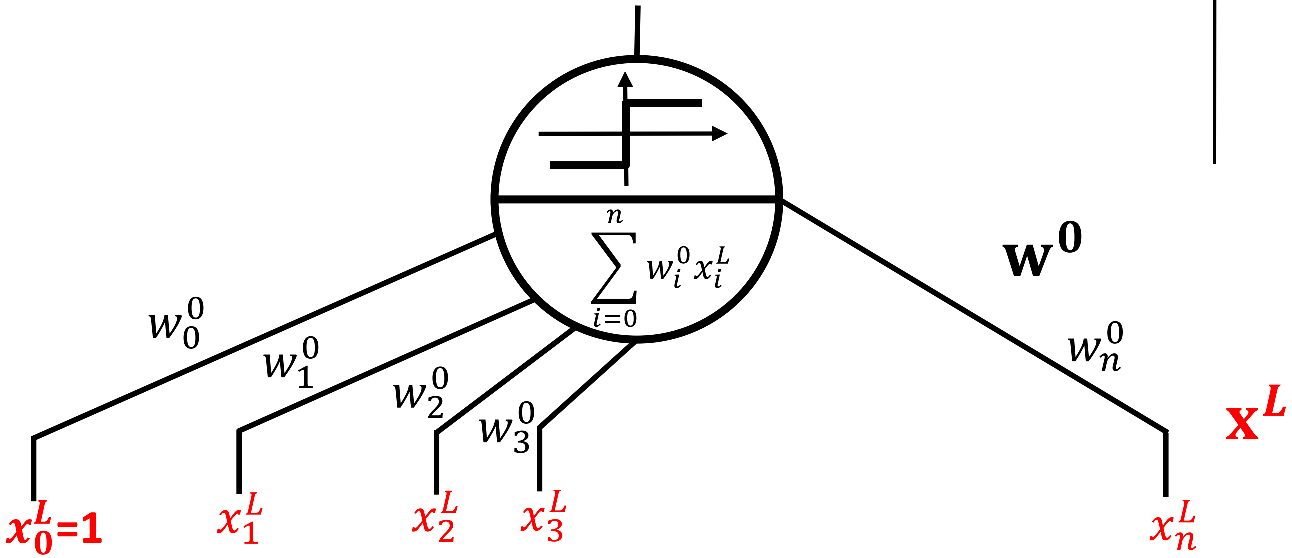
$$y = \mathrm{sign}(\mathbf{w^0}.\mathbf{x}^K)$$



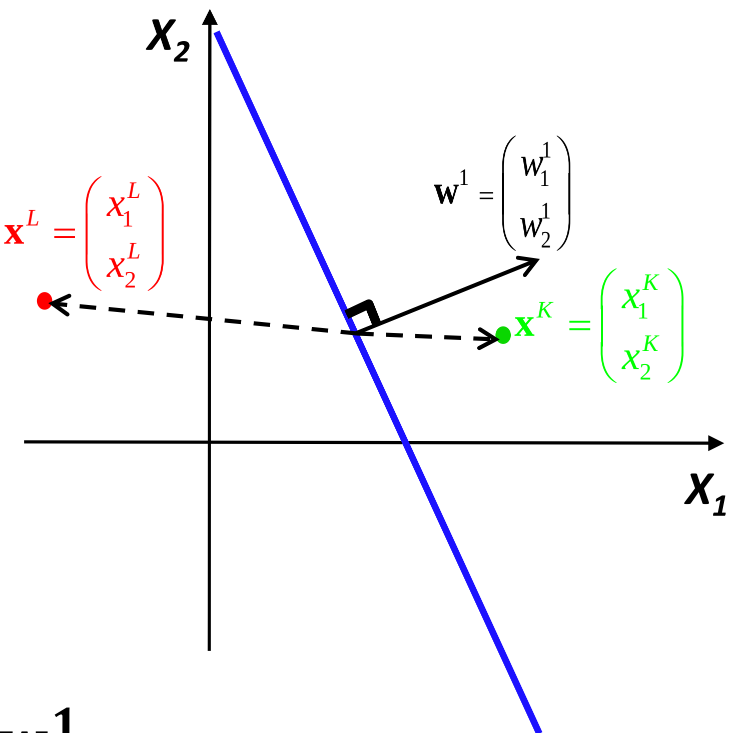
Single Perceptron Unit



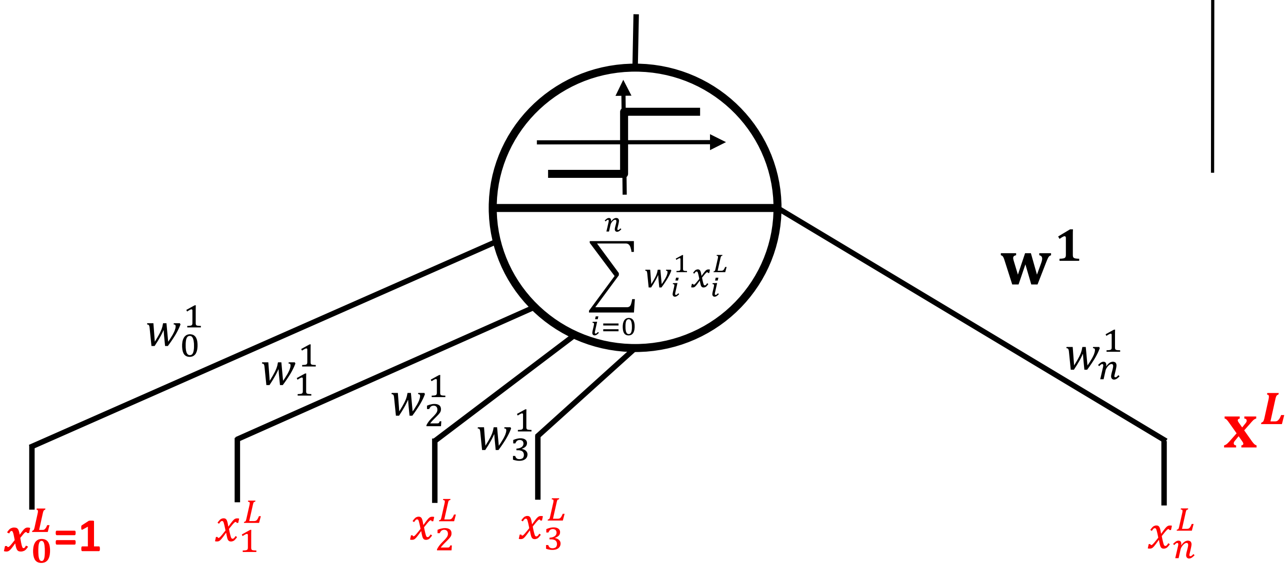
$$y = \mathrm{sign}(\mathbf{w}^0.\mathbf{x}^L)$$



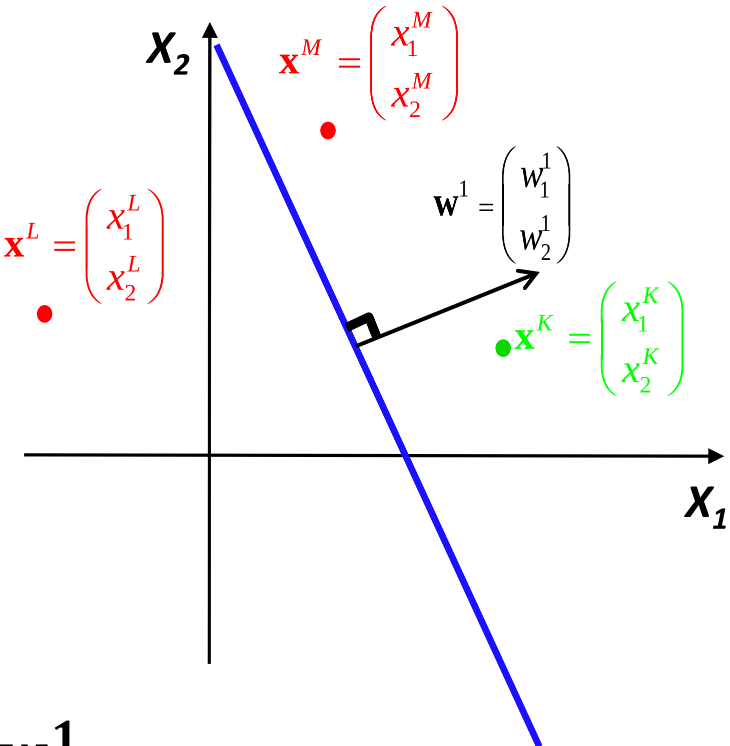
Single Perceptron Unit



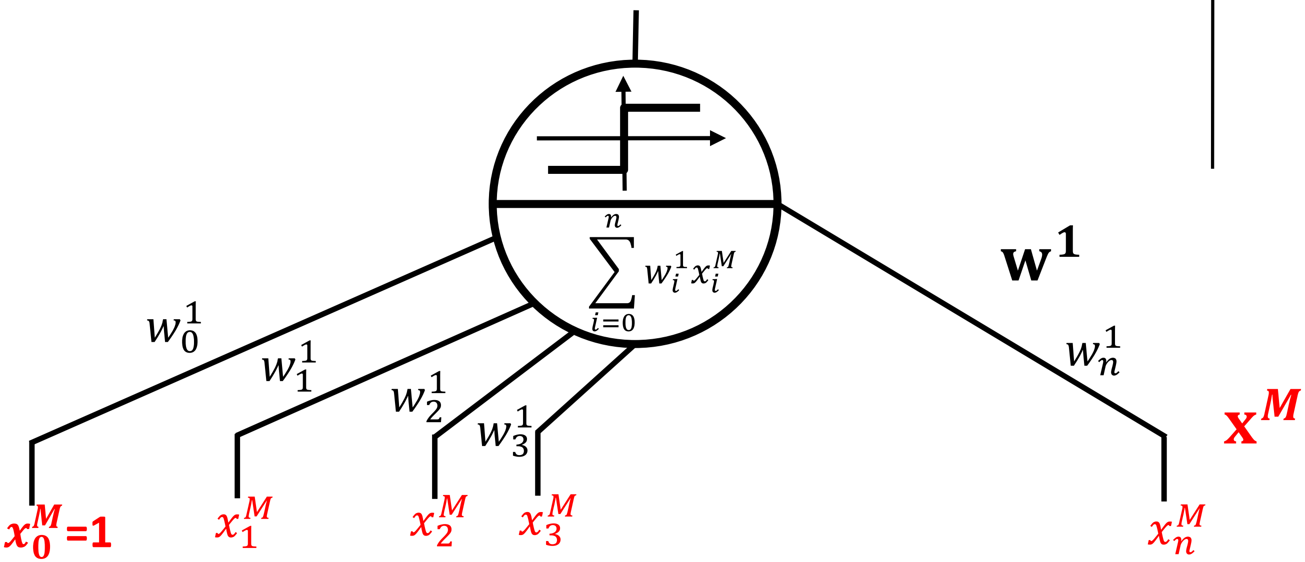
$$y = \mathrm{sign}(\mathbf{w}^1.\mathbf{x}^L)$$



Single Perceptron Unit

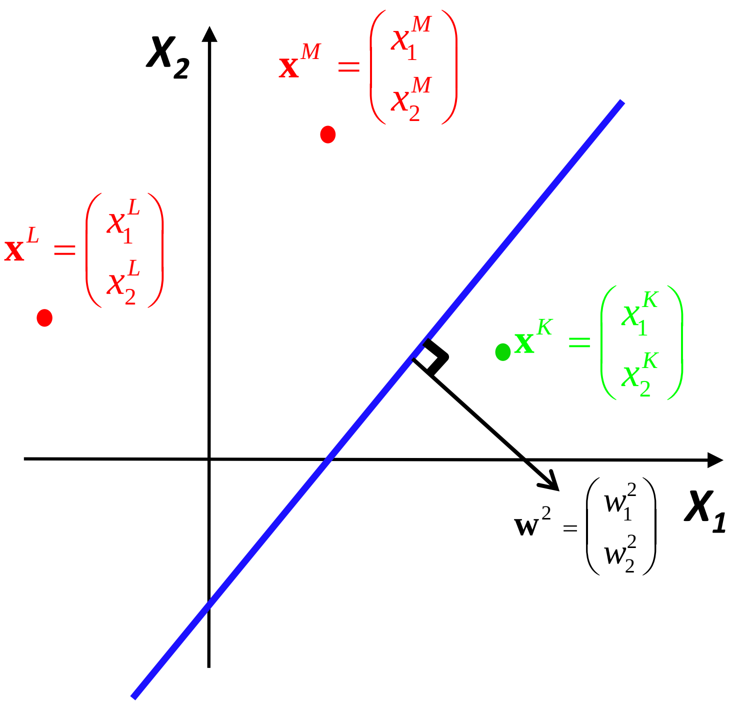


$$y = \mathrm{sign}(\mathbf{w}^1 . \mathbf{x}^M)$$

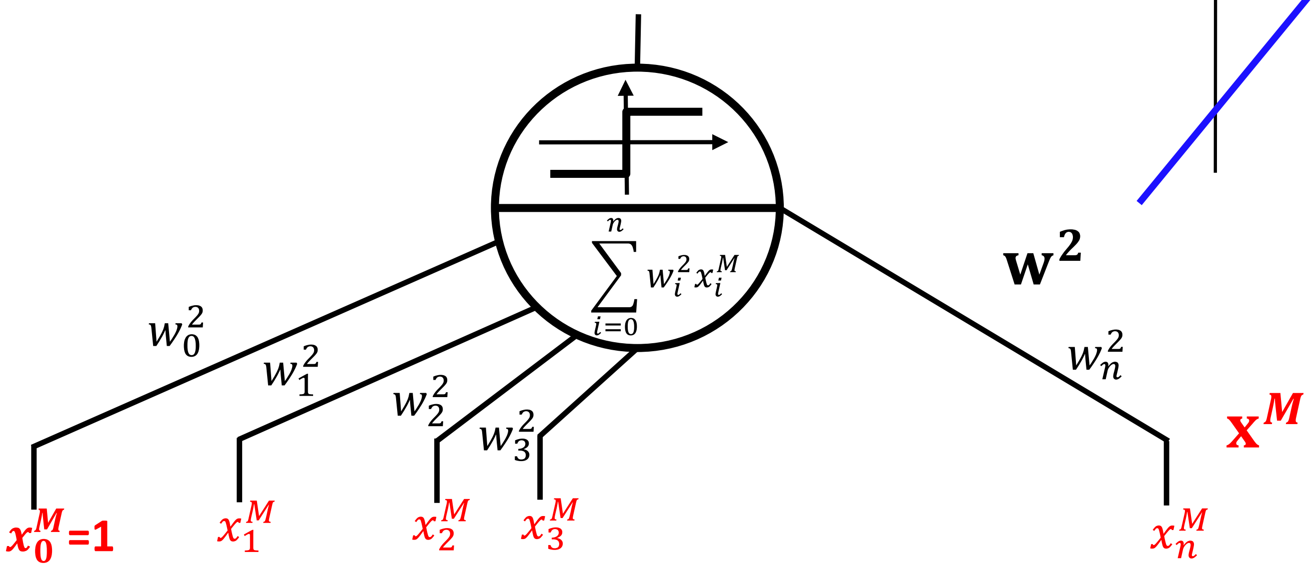


UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

Single Perceptron Unit



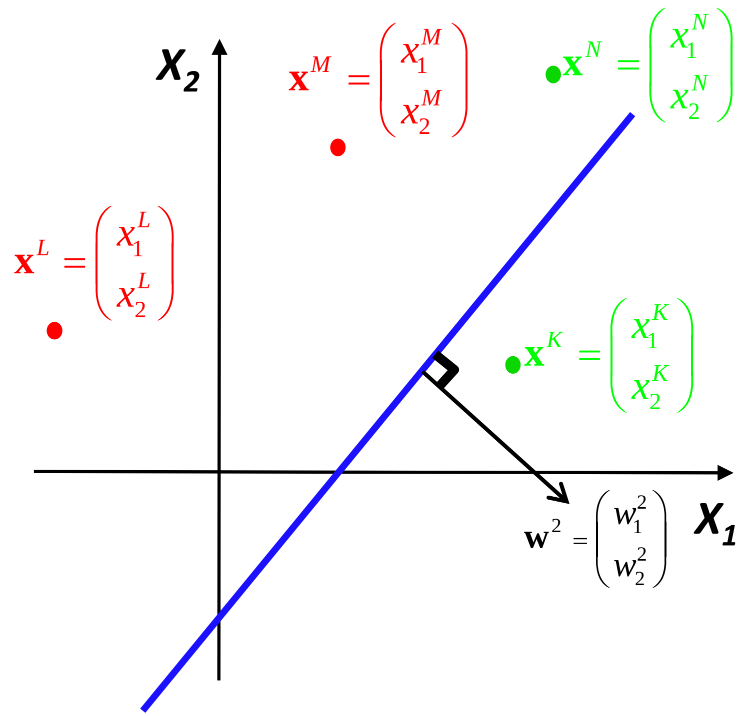
$$y = \mathrm{sign}(\mathbf{w}^2.\mathbf{x}^M)$$



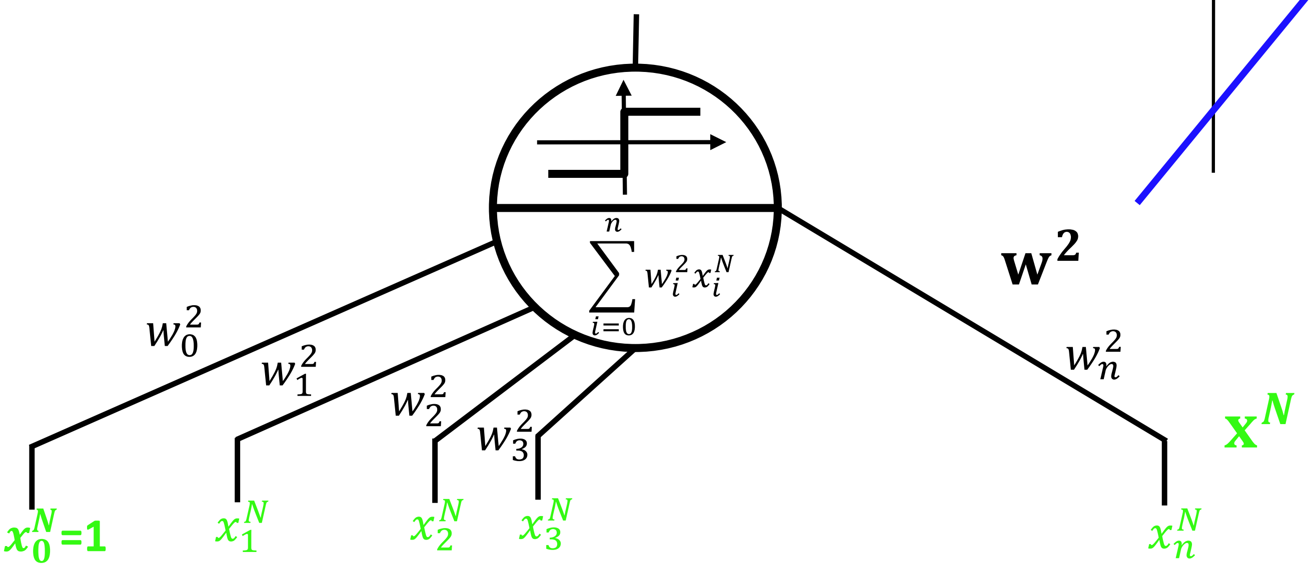
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

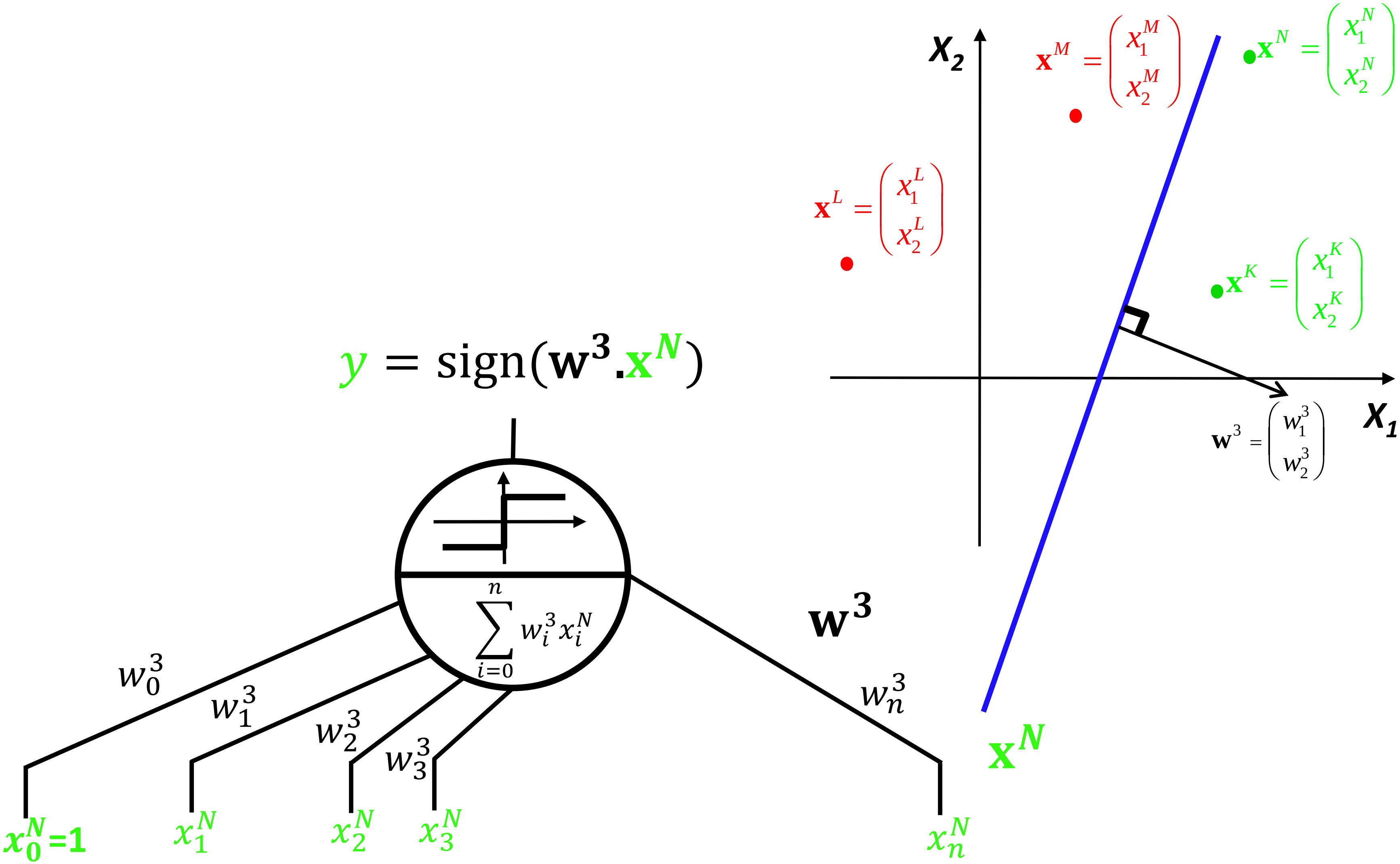


$$y = \mathrm{sign}(\mathbf{w}^2.\mathbf{x}^N)$$

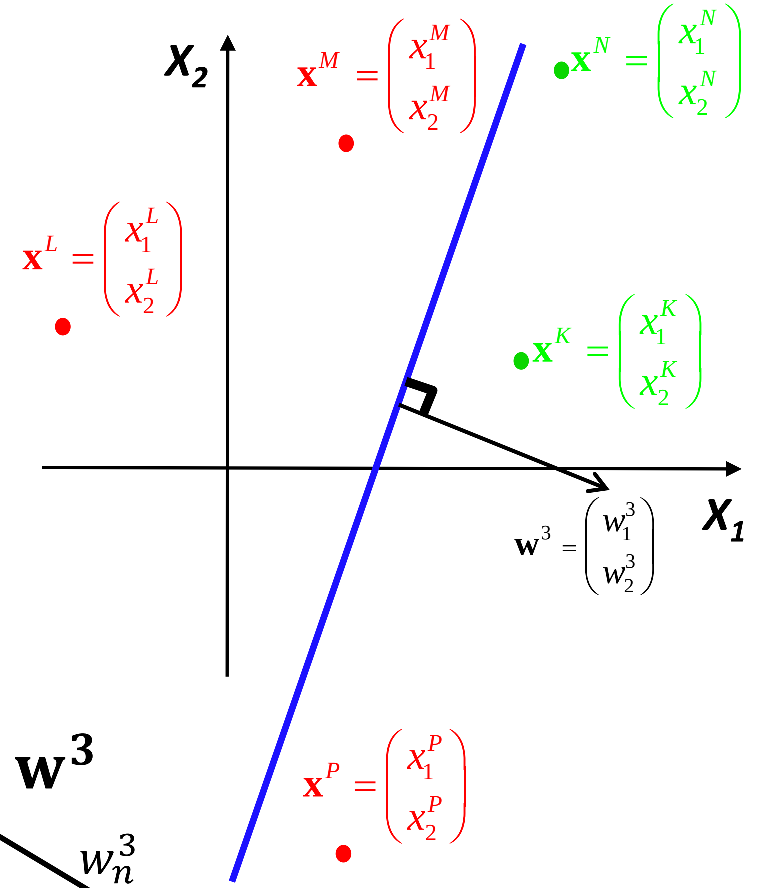


Single Perceptron Unit

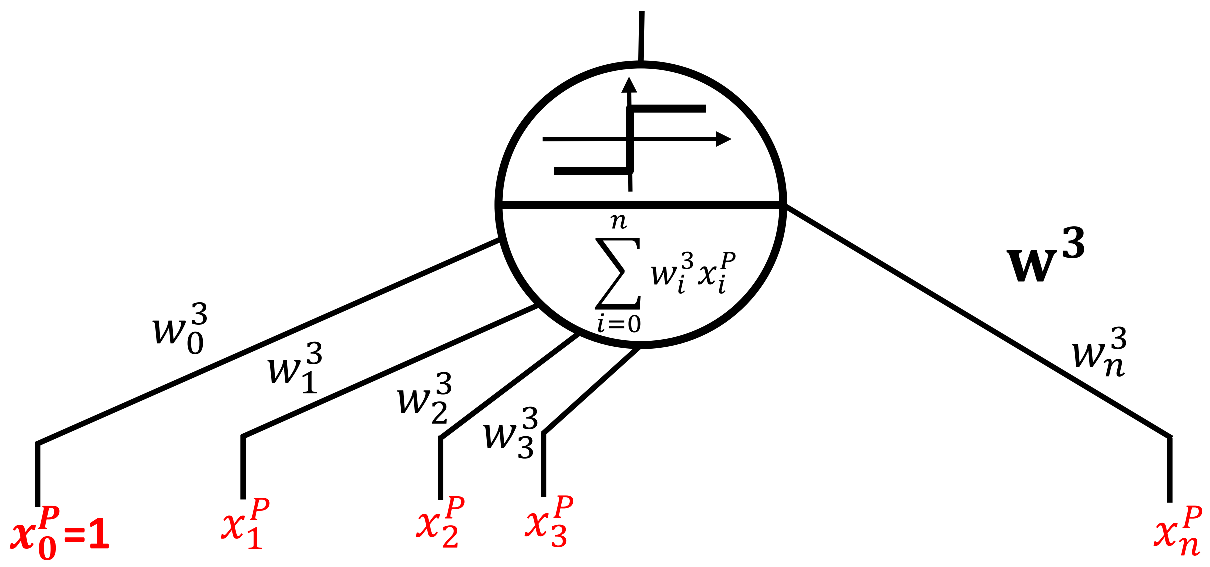
$$y = \mathrm{sign}(\mathbf{w}^3.\mathbf{x}^N)$$



Single Perceptron Unit



$$y = \text{sign}(\mathbf{w}^3.\mathbf{x}^P)$$

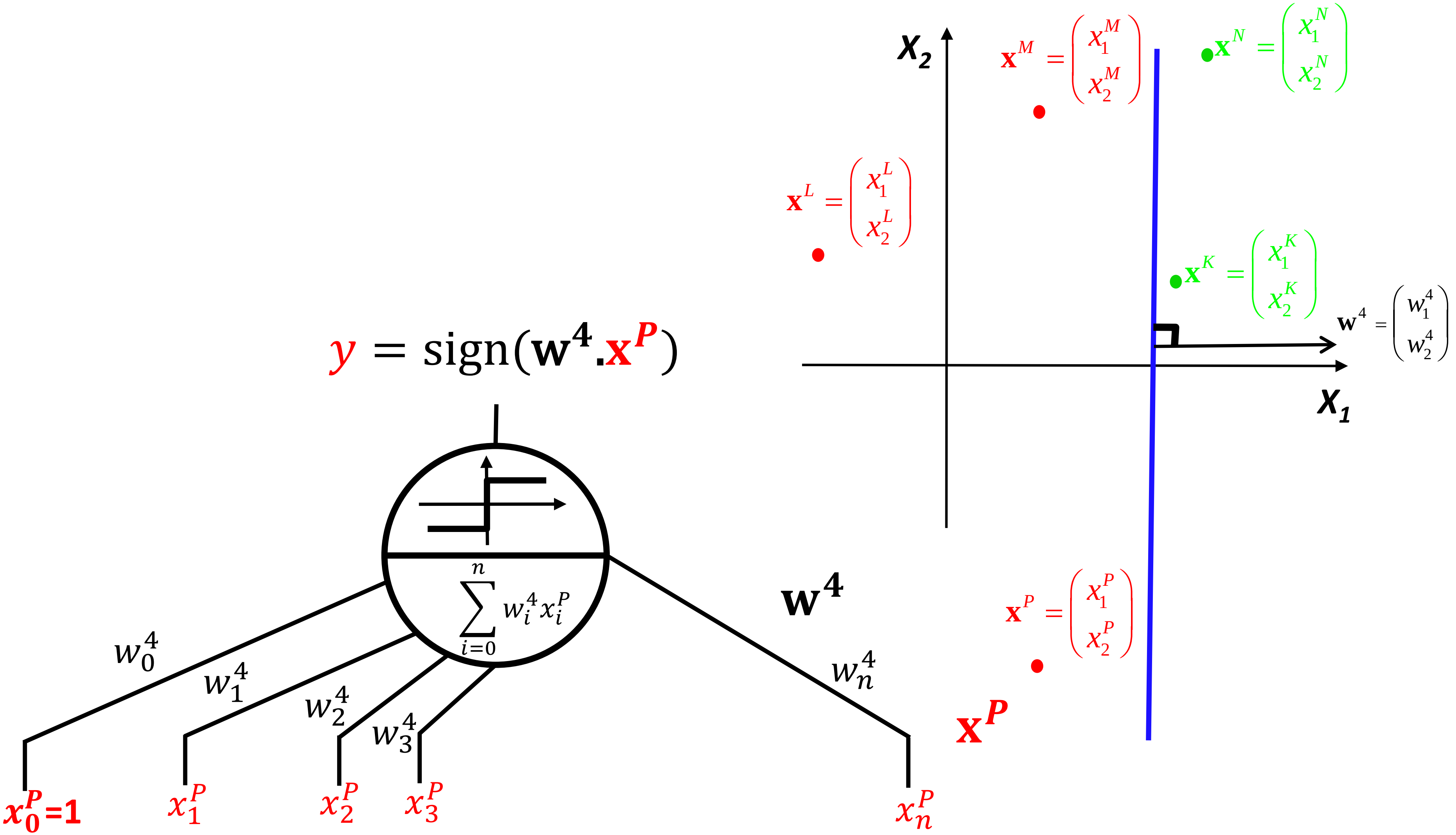


UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

$$y = \mathrm{sign}(\mathbf{w}^4 . \mathbf{x}^P)$$



Perceptron: Rosenblatt's Algorithm (1956)

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

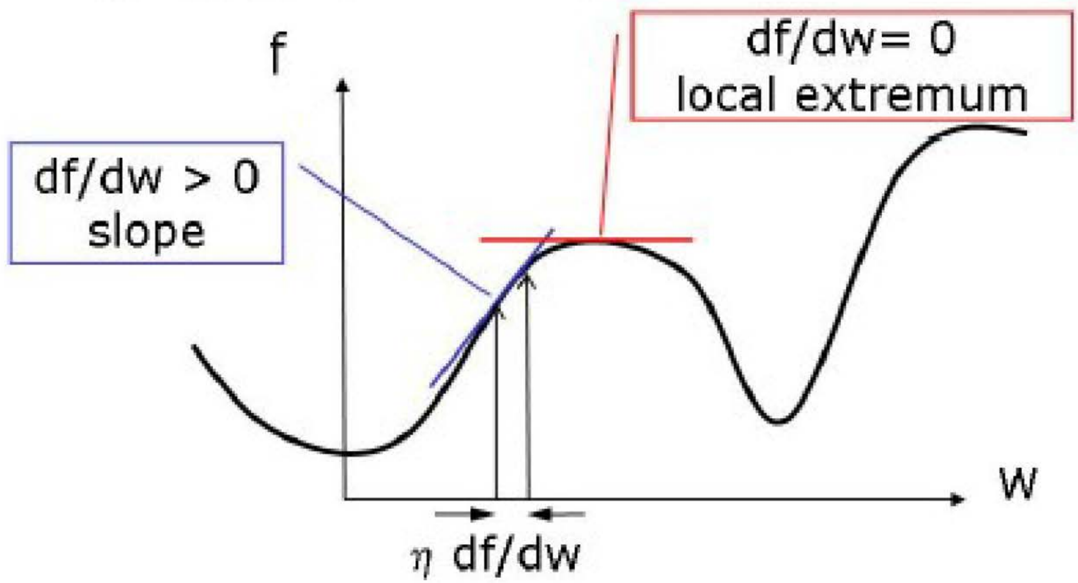
Perceptron Algorithm

- Pick initial weight vector (including w_0), e.g. (0, 0,…,0)
- Repeat until all points are correctly classified
  - Repeat for each point
    - Calculate $y^i\mathbf{w}\mathbf{x}^{\mathrm{i}}$ for point i
    - If $y^i\mathbf{w}\mathbf{x}^{\mathrm{i}} > 0$, the point is correctly classified
    - Else change the weights to increase the value of $y^i\mathbf{w}\mathbf{x}^{\mathrm{i}}$; change in weight proportional to $y^i\mathbf{x}^{\mathrm{i}}$

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6):386.

Gradien Ascent

- Why pick $y^i\mathbf{x}^i$ as increment to weights?
- To maximize scalar function of one variable f($\mathbf{w}$)
  - Pick initial $\mathbf{w}$
  - Change w to $\mathbf{w} + \eta$ df/d$\mathbf{w}$ ($\eta > 0$, small)
  - until f stops changing (df/d$\mathbf{w} \approx 0$)



Gradien Ascent

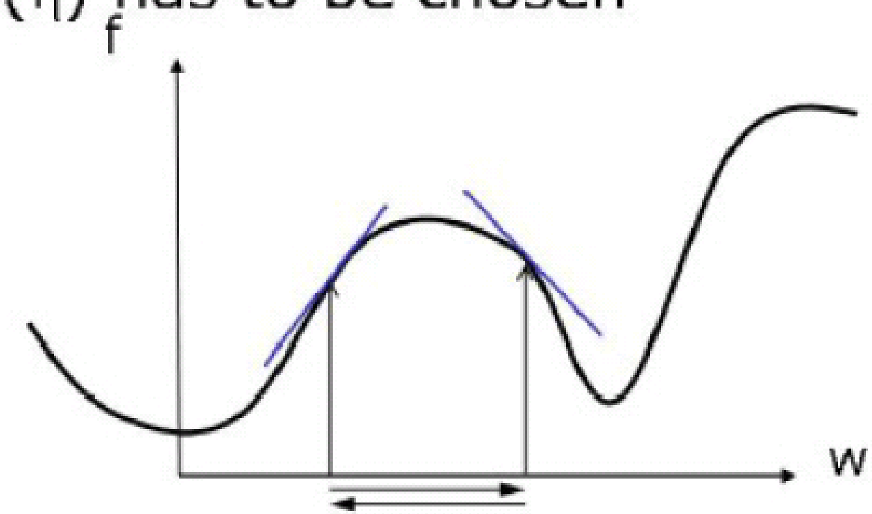
- To maximize f(**w**)
  - Pick initial **w**
  - Change **w** to **w** + $\eta \nabla_{\mathbf{w}} f$ ($\eta > 0$, small)
  - until f stops changing ($\nabla_{\mathbf{w}} f \approx 0$)
- Finds local maximum, unless function is globally convex.

$$\nabla_{\mathbf{w}} f = \left[\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_n} \right]$$

Gradien Ascent

$$\nabla_{\mathbf{w}} f = \left[\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_n} \right]$$

- To maximize f($\mathbf{w}$)
  - Pick initial $\mathbf{w}$
  - Change $\mathbf{w}$ to $\mathbf{w} + \eta \nabla_{\mathbf{w}} f$ ($\eta > 0$, small)
  - until f stops changing ($\nabla_{\mathbf{w}} f \approx 0$)
- Finds local maximum, unless function is globally convex.
- If f is non-linear, rate (η) has to be chosen carefully.
  - Too small – slow convergence
  - Too big – oscillation



Gradien Ascent

- Maximize margin of misclassified points

$$f(\mathbf{w}) = \sum_{i \text{ misclassified}} y^i \mathbf{w} \mathbf{x}^{\mathrm{i}}$$

$$\nabla_{\mathbf{w}} f = \sum_{i \text{ misclassified}} y^i \mathbf{x}^{\mathrm{i}}$$

- Off-line training: Compute gradient as sum over all training points.
- On-line training: Approximate gradient by one of the terms in the sum: $y^i \mathbf{x}^{\mathrm{i}}$

Perceptron Algorithm

- Each change of **w** decreases the error on a specific point. However, changes for several points are correlated, that is different points could change the weights in opposite directions. Thus, this iterative algorithm requires several loops to converge.

- Guarantee to find a separating hyperplane if one exists – if data is linearly separable

- If data are not linearly separable, then this algorithm loops indefinitely

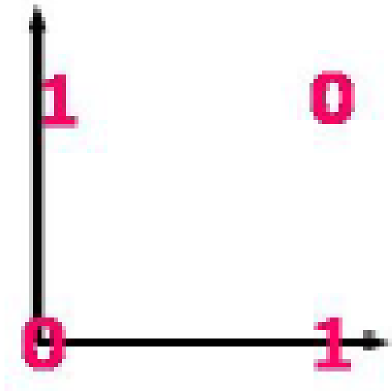
Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6):386.

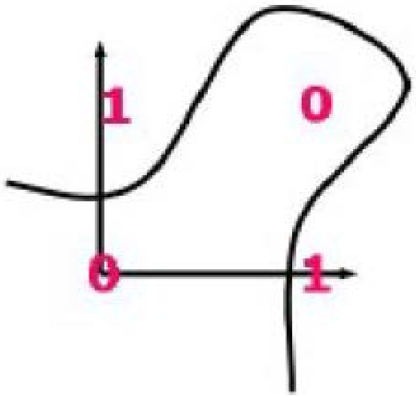
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Beyond Linear Separability

- Values of the XOR boolean function cannot be separated by a single perceptron unit [Minsky and Papert, 1969].





Minsky, M. and Papert, S. (1969).
Perceptrons: An Introduction to Computational Geometry. MIT Press.

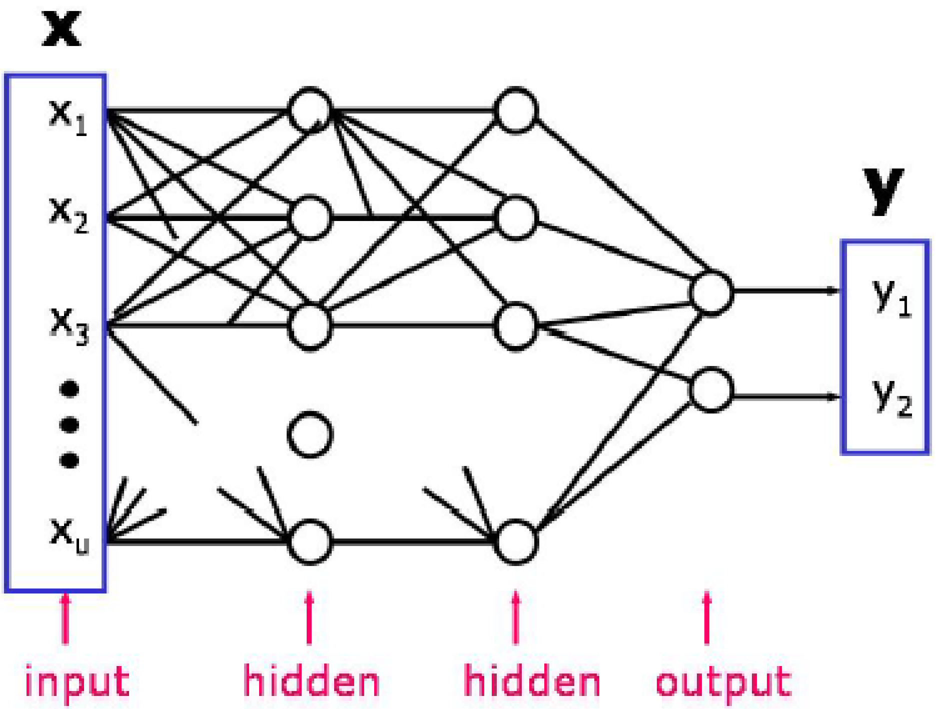
UNIVERSITÉ CÔTE D'AZUR

Multi-Layer Perceptron

Solution: Multi-Layer Perceptron

univ-cotedazur.fr

- **Solution:** Combine multiple linear separators.
- Introduction of "hidden" units into NN make them much more powerful: they are no longer limited to linearly separable problems.
- Earlier layers transform the problem into more tractable problems for the latter layers.



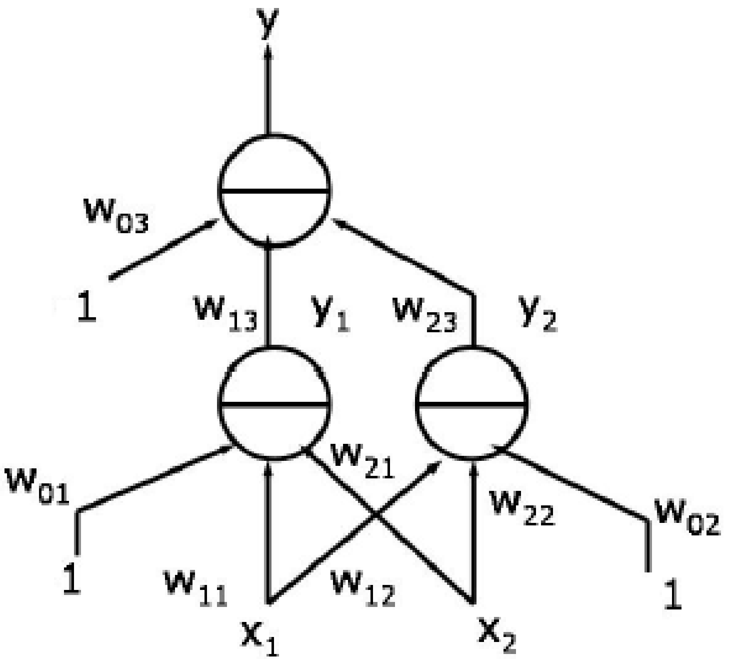
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

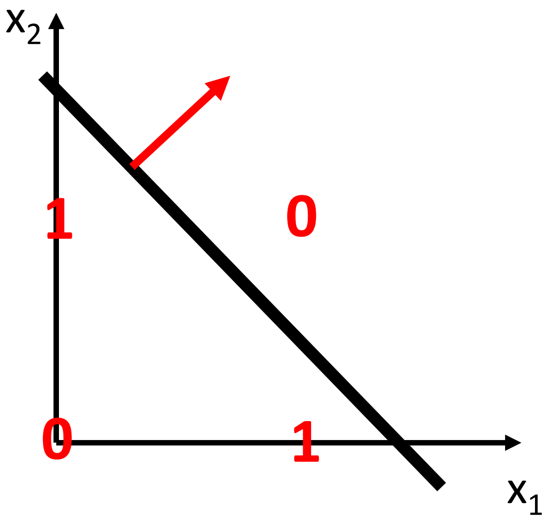
Feedforward networks

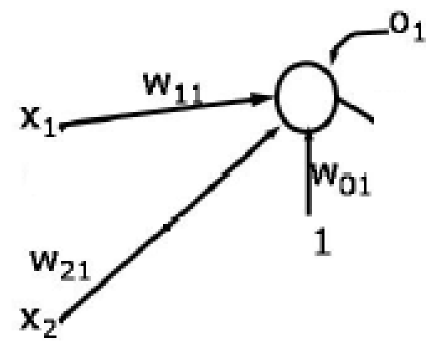
- Interconnected networks of simple perceptron units ("artificial neurons"): Weight w_{ij} is the weight of the i th input into unit j.

- Learning takes place by adjusting the weights in the network, so that the desired output is produced whenever a training instance is presented.

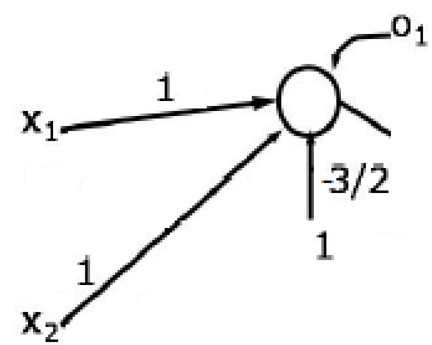


Example: XOR problem





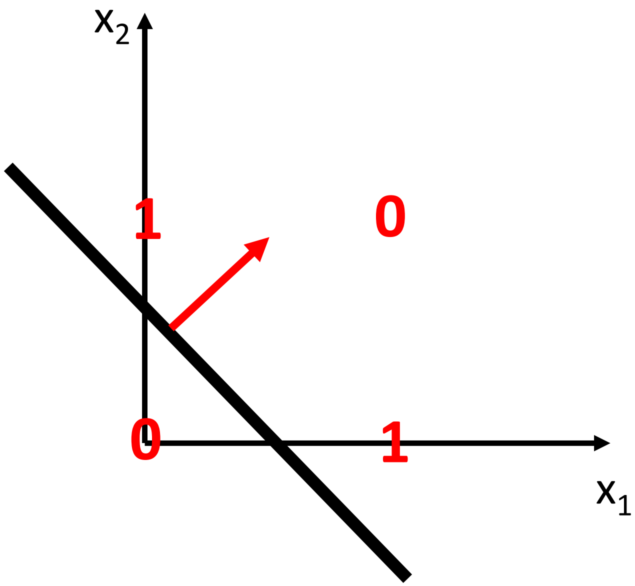
$w_{01} = -3/2 \quad w_{11} = w_{21} = 1$



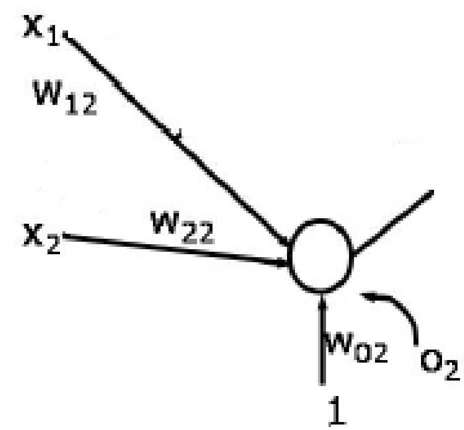
x_1	x_2	o_1
0	0	0
0	1	0
1	0	0
1	1	1

Example: XOR problem

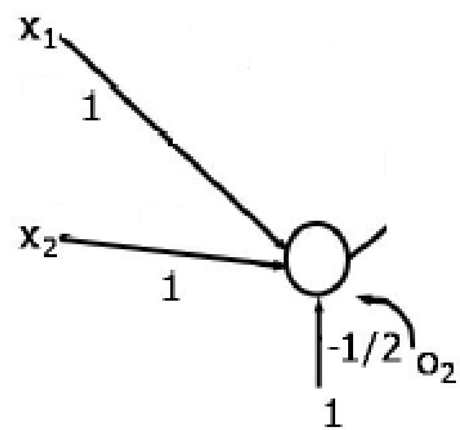
univ-cotedazur.fr



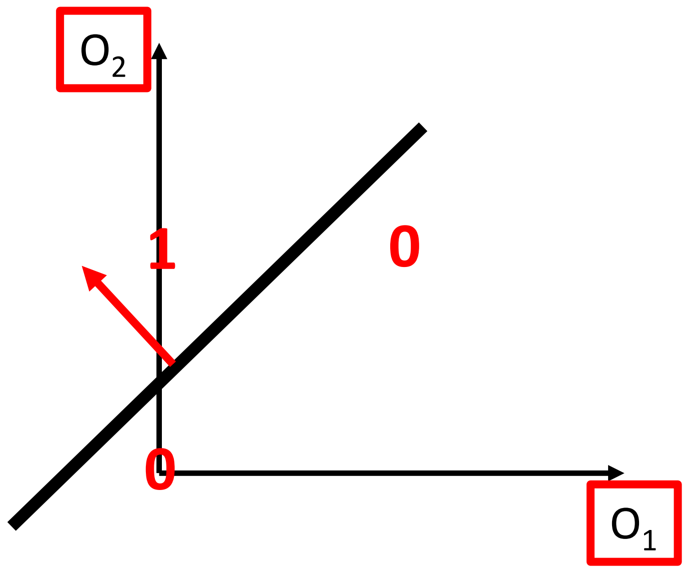
x_1	x_2	o_1	o_2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1



$w_{02} = -1/2 \qquad w_{12} = w_{22} = 1$

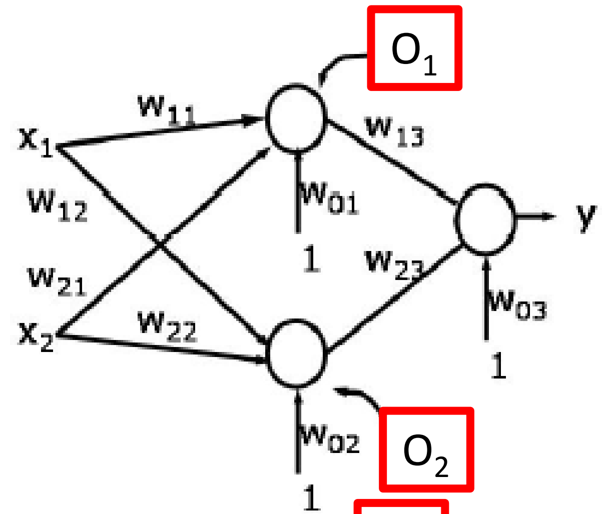


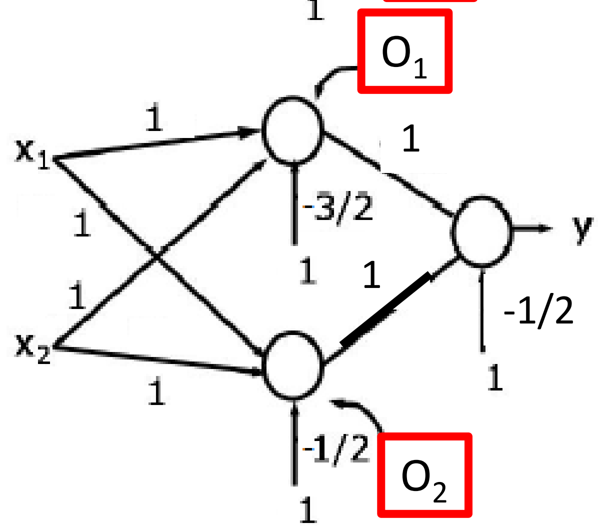
Example: XOR problem



$O_2 - O_1 - 1/2 = 0$

o_1	o_2	y
0	0	0
0	1	1
0	1	1
1	1	0

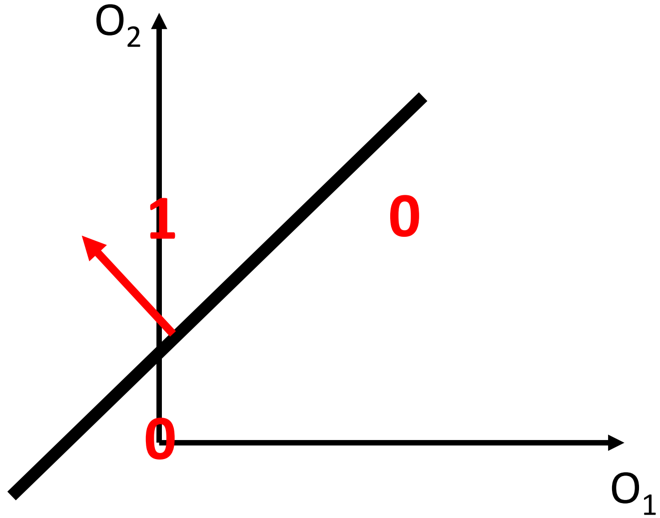




$w_{23}o_2 + w_{13}o_1 + w_{03} = 0$

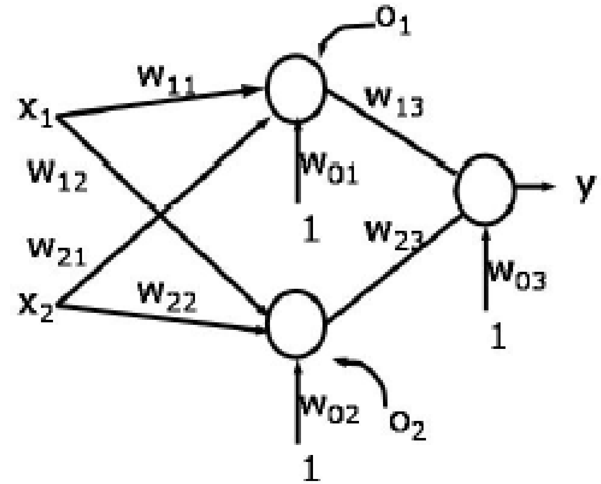
$w_{03} = -1/2$, $w_{13} = -1$, $w_{23} = 1$

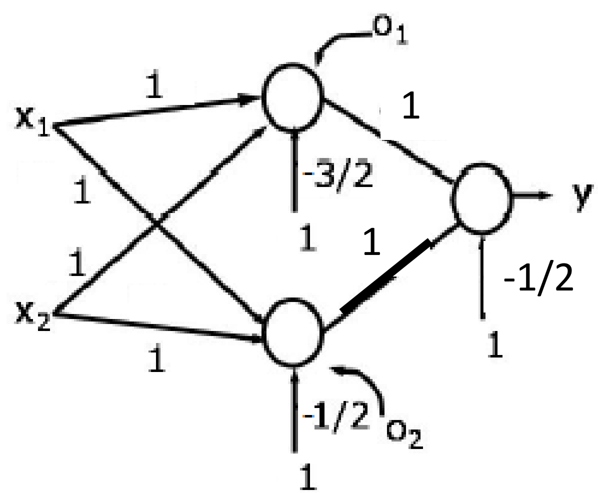
Example: XOR problem



$O_2 - O_1 - 1/2 = 0$

x_1	x_2	o_1	o_2	y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

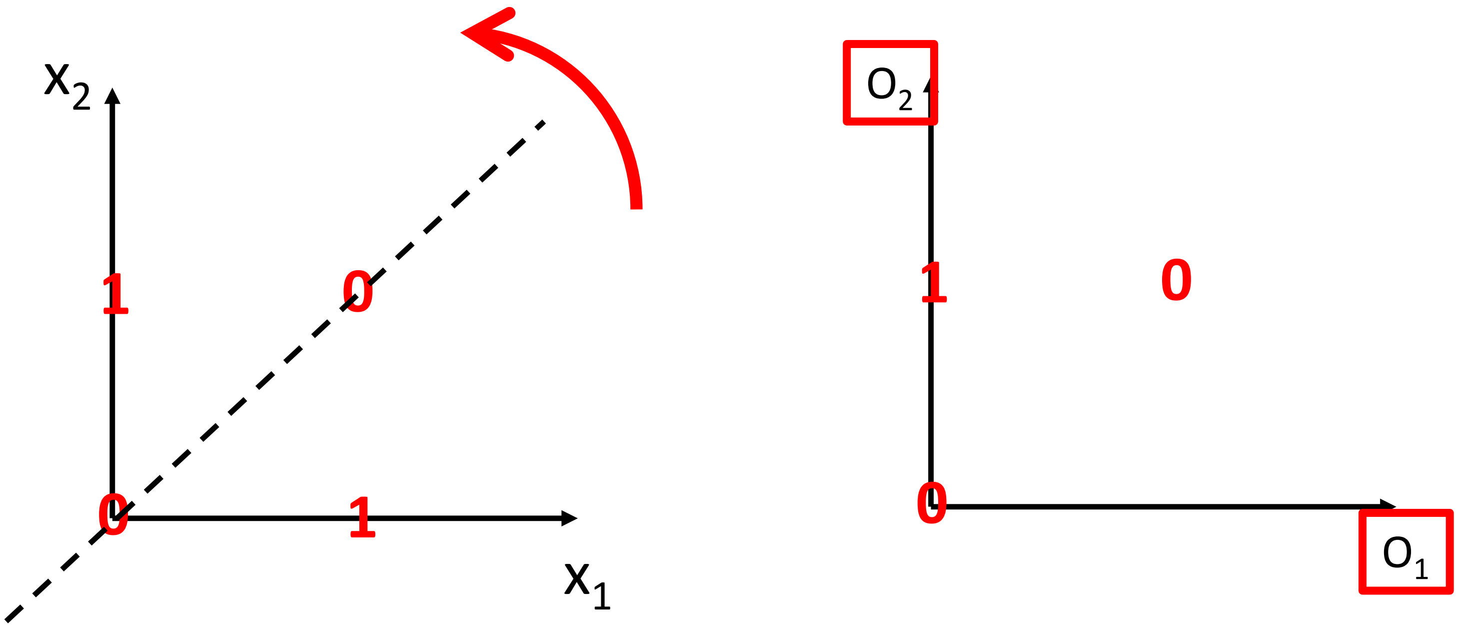




$w_{23}o_2 + w_{13}o_1 + w_{03} = 0$

$w_{03} = -1/2$, $w_{13} = -1$, $w_{23} = 1$

Example: XOR problem

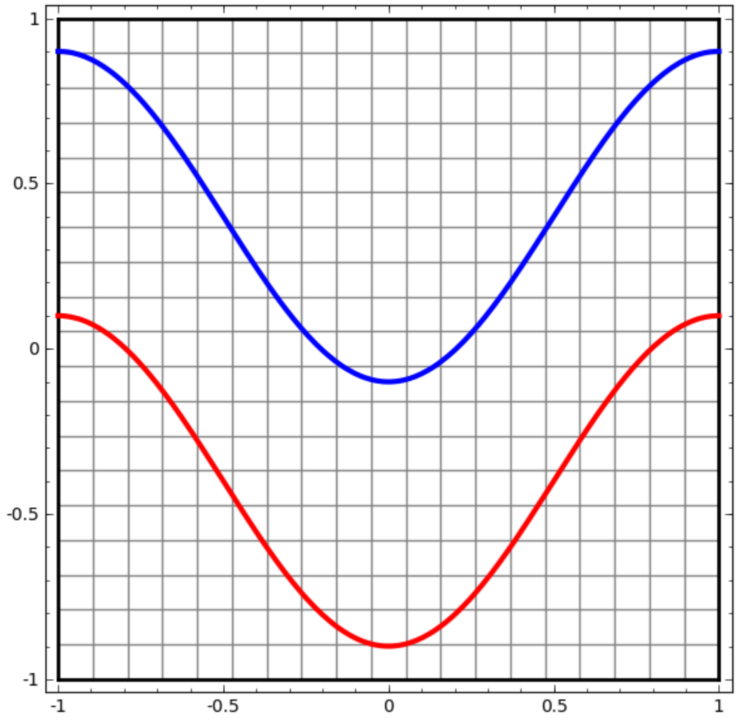


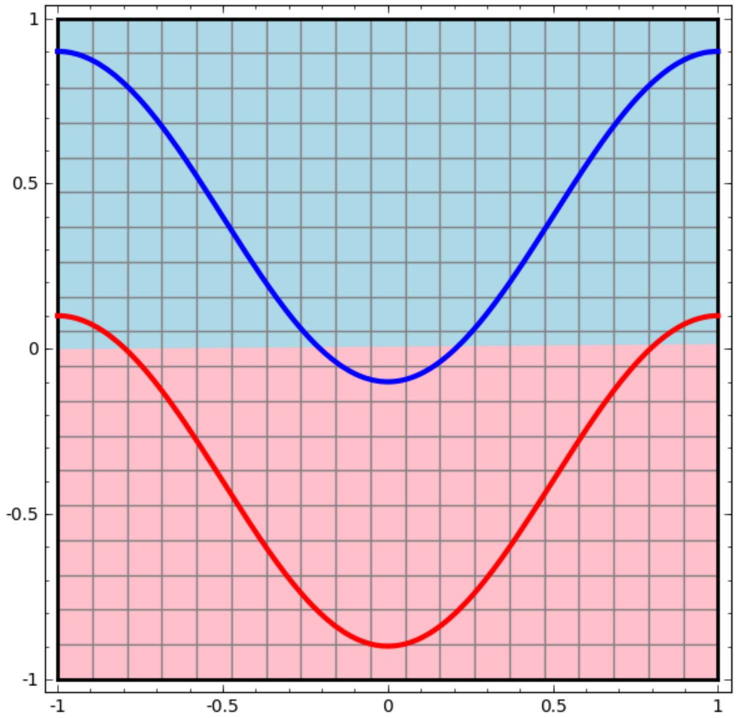
Adding a hidden layer has folded the input space

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

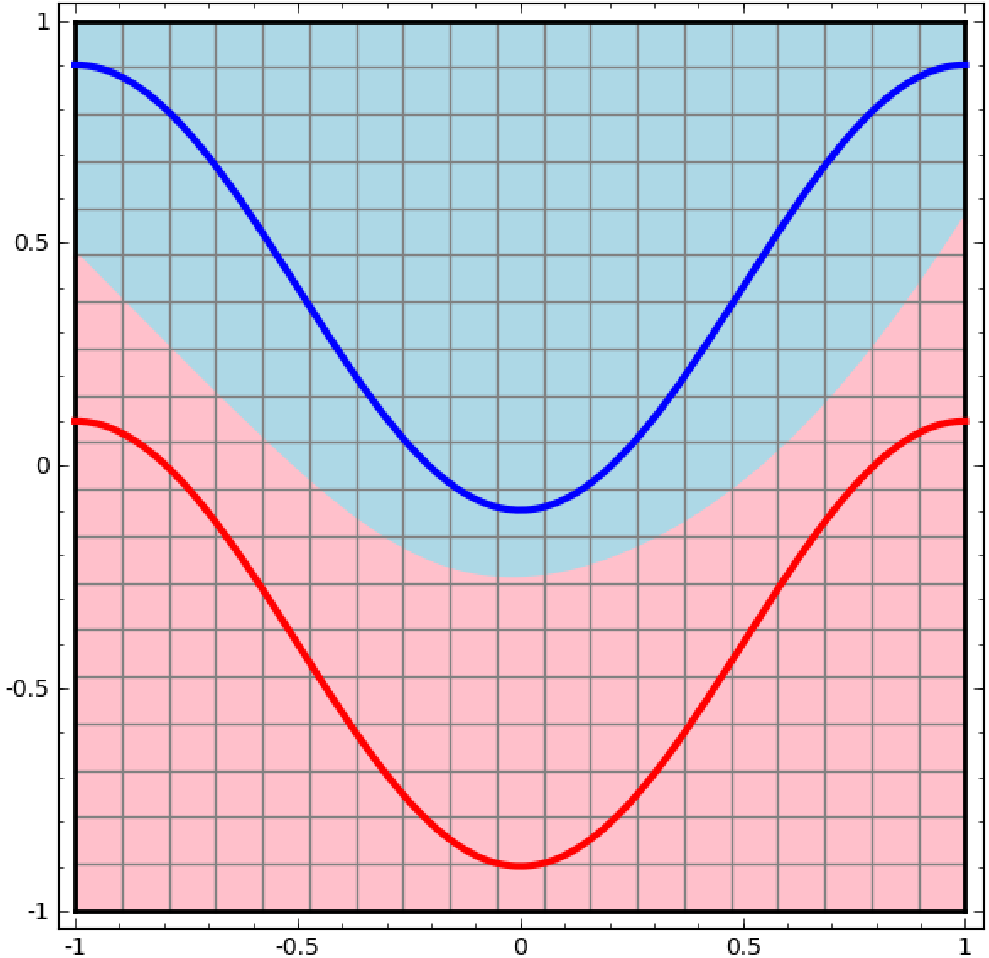
One perceptron





Illustrations from: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

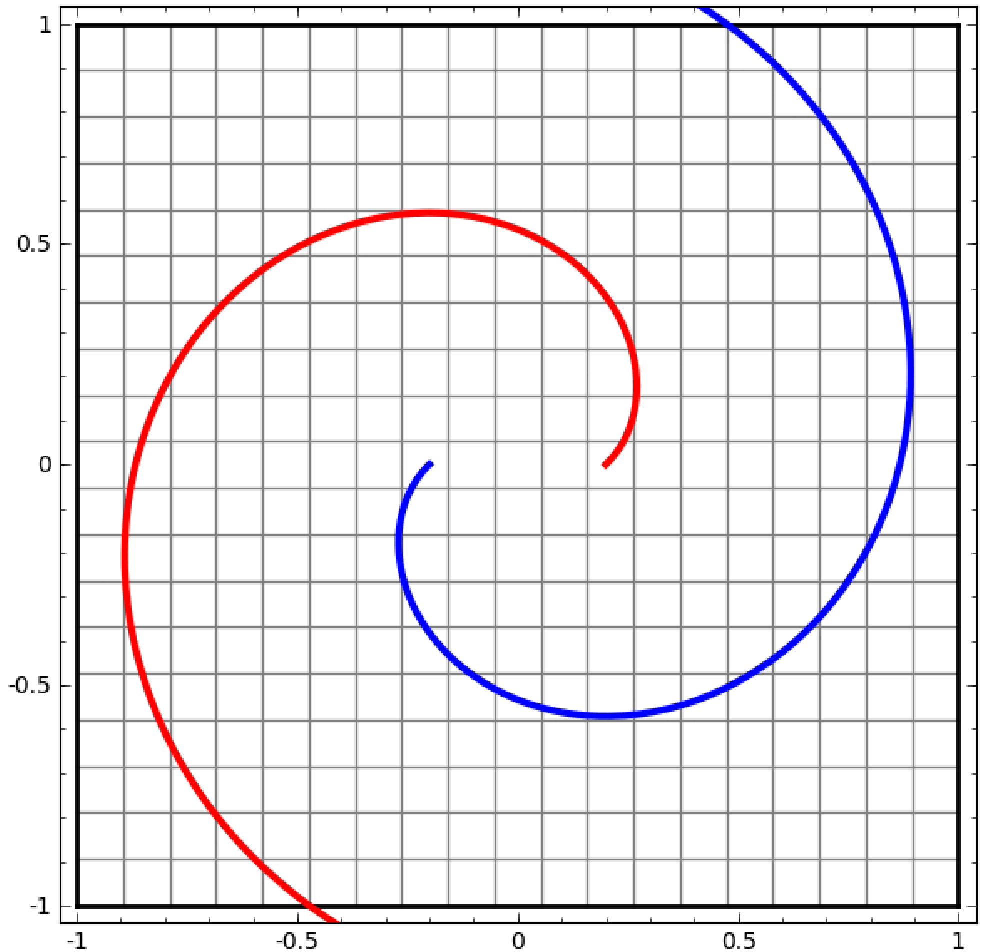
Multi-Layer Perceptron, manifold disentanglement



Illustrations from: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

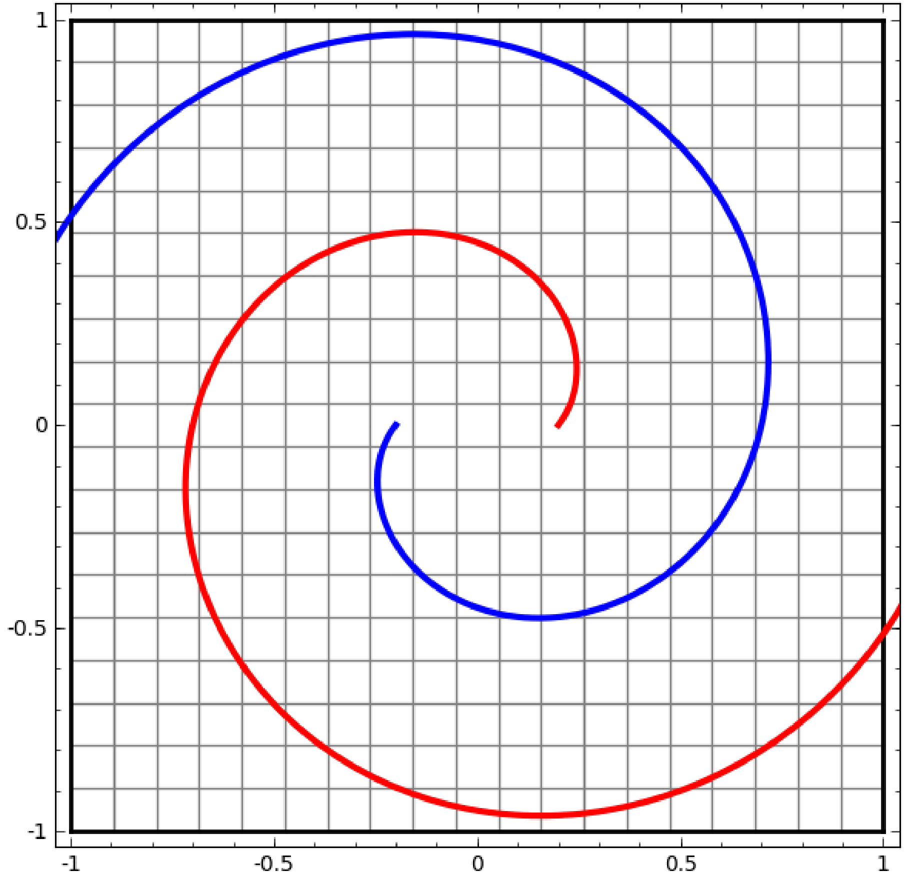
UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

Multi-Layer Perceptron, manifold disentanglement



Illustrations from: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Multi-Layer Perceptron, manifold disentanglement



Illustrations from: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Multi-Layer Perceptron

Theorem

- **A neural network with one single hidden layer is a universal approximator**: it can represent any continuous function on compact subsets of $\mathbf{R}^n$ [Cybenko, 1989]
- **2 layers is enough ... theoretically:**

  "...networks with one internal layer and an arbitrary continuous sigmoidal function can approximate continuous functions with arbitrary precision providing that no constraints are placed on the number of nodes or the size of the weights"
- **But no efficient learning rule is known** and **the size of the hidden layer may be exponential with** the complexity of **the problem** which is unknown beforehand.

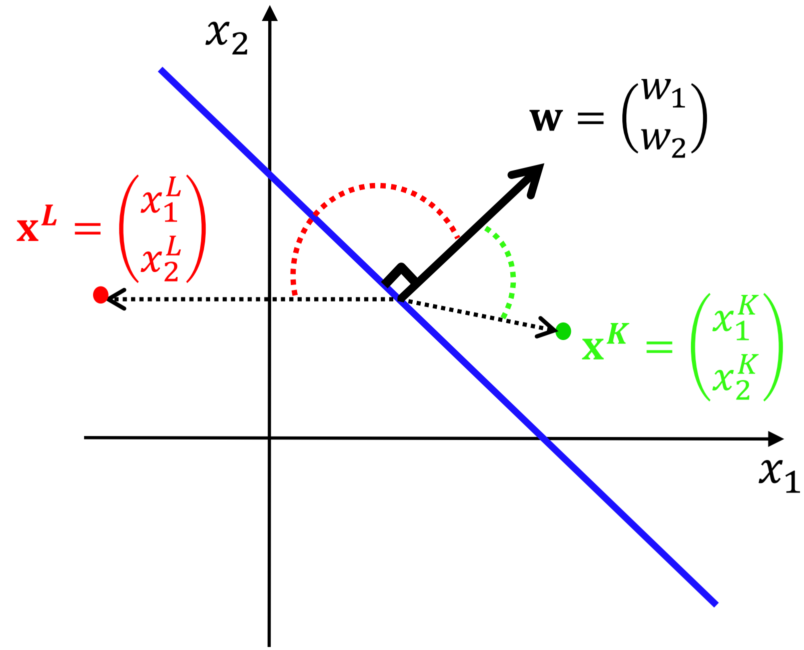
Soft Threshold

- A natural question to ask is whether we could use gradient ascent/descent to train a multi-layer perceptron?

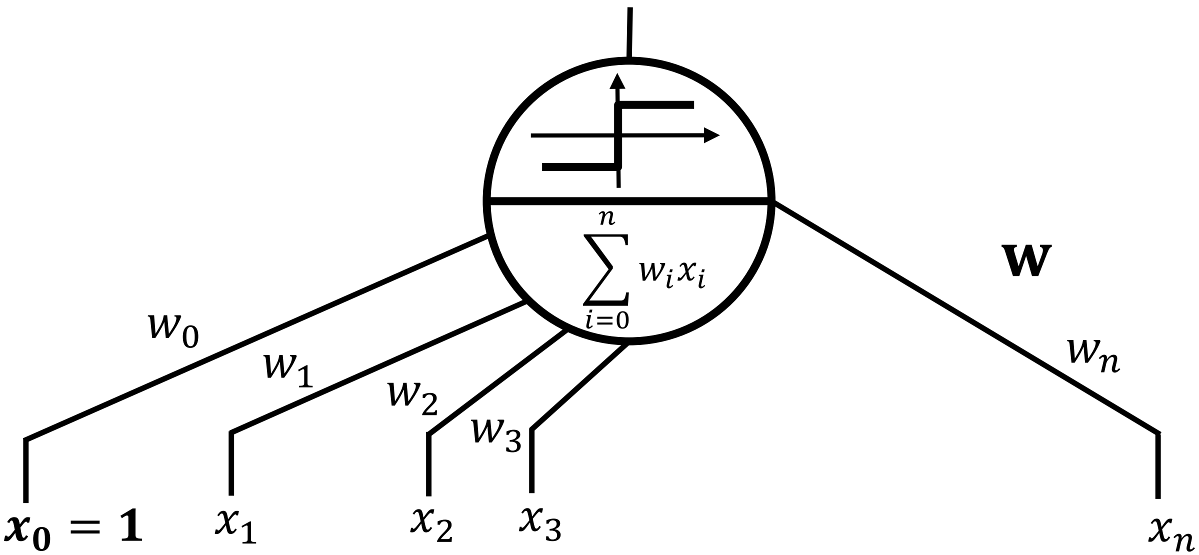
- The answer is that we cannot as long as the output is discontinuous with respect to changes in the inputs and the weights.
  - In a perceptron unit it does not matter how far a point is from the decision boundary, we will still get a 0 or a 1.

- We need a smooth output (as a function of changes in the network weights) if we are to do gradient descent.

Single Perceptron Unit

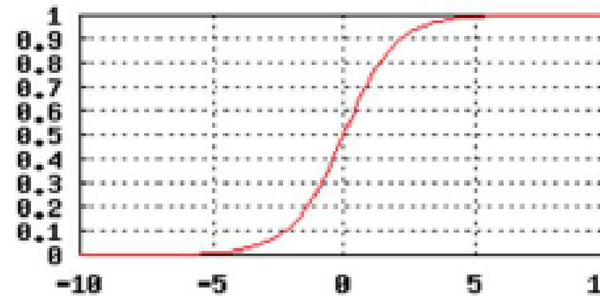


$$y = \mathrm{sign}(\mathbf{w.x})$$

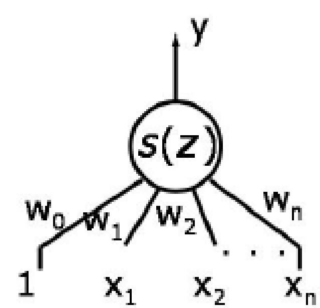


Sigmoid Unit

- Commonly used in neural nets is a "sigmoid" (S-like) function (see on the right).
  - The one used here is called the **logistic** function.
- Value *z* is also called the "activation" of a neuron.



$$z = \sum_{i}^{n} w_i x_i \qquad s(z) = \frac{1}{1+e^{-z}}$$



$$\frac{ds(z)}{dz} = \frac{d}{dz}\left[(1+e^{-z})^{-1}\right]$$
$$= s(z)(1-s(z))$$
$$= y(1-y)$$

Training

- Key property of the sigmoid is that it is differentiable.
  - This means that we can use gradient based methods of minimization for training.

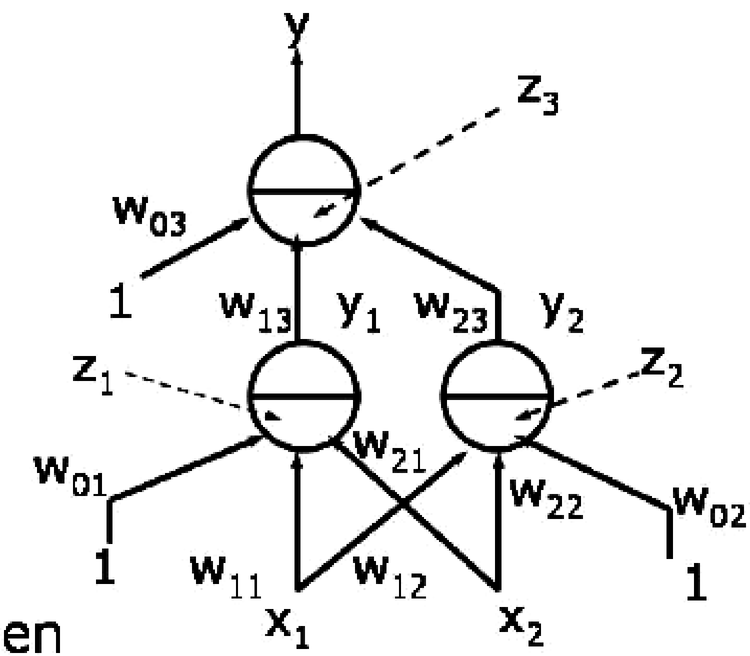
- The output of a multi-layer net of sigmoid units is a function of two vectors, the inputs ($\mathbf{x}$) and the weights ($\mathbf{w}$).
  - As we train the ANN the training instances are considered fixed.

- The output of this function (y) varies smoothly with changes in the weights.

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Training



$y(\mathbf{x}, \mathbf{w})$

w is a vector of weights

x is a vector of inputs

y^m is desired output:

Error over the training set for a given weight vector:

$$E = \frac{1}{2} \sum_i (y(\mathbf{x}^m, \mathbf{w}) - y^m)^2$$

Our goal is to find weight vector that minimizes error.

Gradient Descent

We follow **gradient descent**
Gradient of the training error is computed as a function of the weights.

$$E = \frac{1}{2}(y(\mathbf{x}^m, \mathbf{w}) - y^m)^2$$

$$\nabla_{\mathbf{w}} E = (y(\mathbf{x}^m, \mathbf{w}) - y^m)\nabla_{\mathbf{w}} y(\mathbf{x}^m, \mathbf{w})$$

where $$\nabla_{\mathbf{w}} y(\mathbf{x}^m, \mathbf{w}) = \left[\frac{\partial y}{\partial w_1}, \ldots, \frac{\partial y}{\partial w_n}\right]$$

As a shorthand, we will denote $y(\mathbf{x}^m,\mathbf{w})$ just by y.

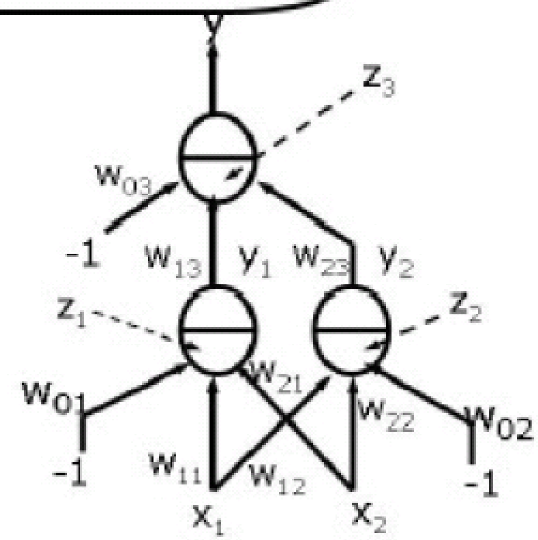
We change $\mathbf{w}$ as follows

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} E \quad \Leftrightarrow \quad \mathbf{w} \leftarrow \mathbf{w} - \eta(y(\mathbf{x}^m, \mathbf{w}) - y^m)\left[\frac{\partial y}{\partial w_1}, \ldots, \frac{\partial y}{\partial w_n}\right]$$

Gradient of Error

$$E = \frac{1}{2}\sum_i \left(y(\mathbf{x}^i, \mathbf{w}) - y^{i*}\right)^2$$

$$y = s\Big(\underbrace{w_{13}s(\overbrace{w_{11}x_1 + w_{21}x_2 - w_{01}}^{z_1}) + w_{23}s(\overbrace{w_{12}x_1 + w_{22}x_2 - w_{02}}^{z_2})}_{z_3} - w_{03}\Big)$$



UNIVERSITÉ CÔTE D'AZUR

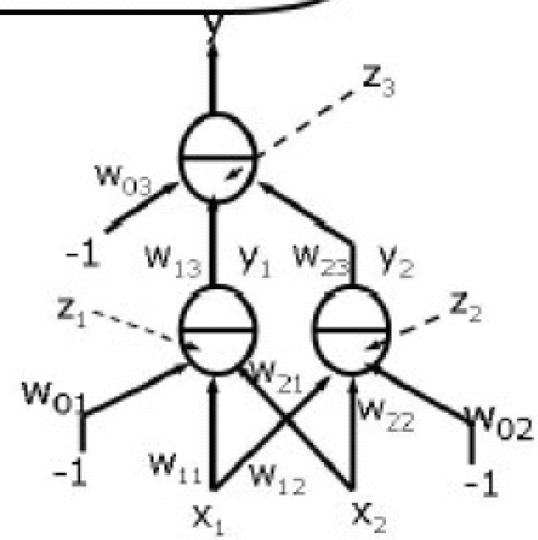
univ-cotedazur.fr

Gradient of Error

$$E = \frac{1}{2}\sum_i \left(y(\mathbf{x}^i, \mathbf{w}) - y^{i*}\right)^2$$

$$y = s(\underbrace{w_{13}s(\overbrace{w_{11}x_1 + w_{21}x_2 - w_{01}}^{z_1}) + w_{23}s(\overbrace{w_{12}x_1 + w_{22}x_2 - w_{02}}^{z_2})}_{z_3} - w_{03})$$

$$\frac{\partial E}{\partial w_j} = (y - y^{i*})\frac{\partial y}{\partial w_j}$$



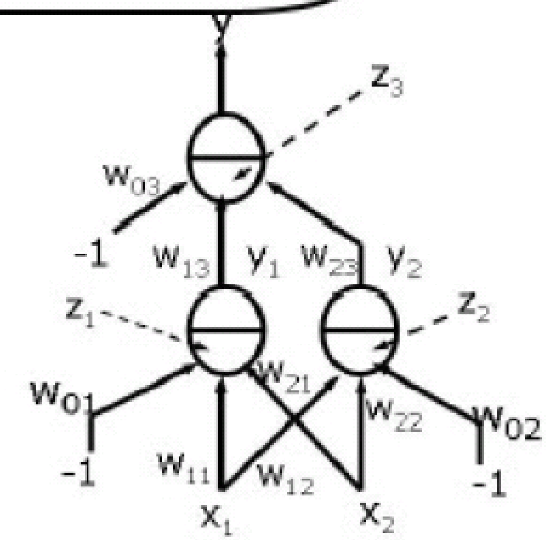
Gradient of Error

$$E = \frac{1}{2}\sum_i \left(y(\mathbf{x}^i, \mathbf{w}) - y^{i*}\right)^2$$

$$y = \underbrace{s\Big(w_{13}s(\overbrace{w_{11}x_1 + w_{21}x_2 - w_{01}}^{z_1}) + w_{23}s(\overbrace{w_{12}x_1 + w_{22}x_2 - w_{02}}^{z_2}) - w_{03}\Big)}_{z_3}$$

$$\frac{\partial E}{\partial w_j} = (y - y^{i*})\frac{\partial y}{\partial w_j}$$

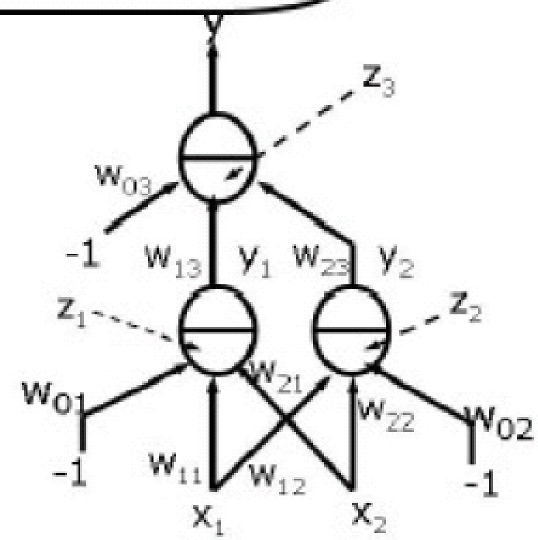
$$\frac{\partial y}{\partial w_{13}} = \frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial w_{13}} = \frac{\partial y}{\partial z_3}y_1$$



Gradient of Error

$$E = \frac{1}{2}\sum_i \left(y(\mathbf{x}^i, \mathbf{w}) - y^{i*}\right)^2$$

$$y = s(\underbrace{w_{13}s(\overbrace{w_{11}x_1 + w_{21}x_2 - w_{01}}^{z_1}) + w_{23}s(\overbrace{w_{12}x_1 + w_{22}x_2 - w_{02}}^{z_2}) - w_{03}}_{z_3})$$



$$\frac{\partial E}{\partial w_j} = (y - y^{i*})\frac{\partial y}{\partial w_j}$$

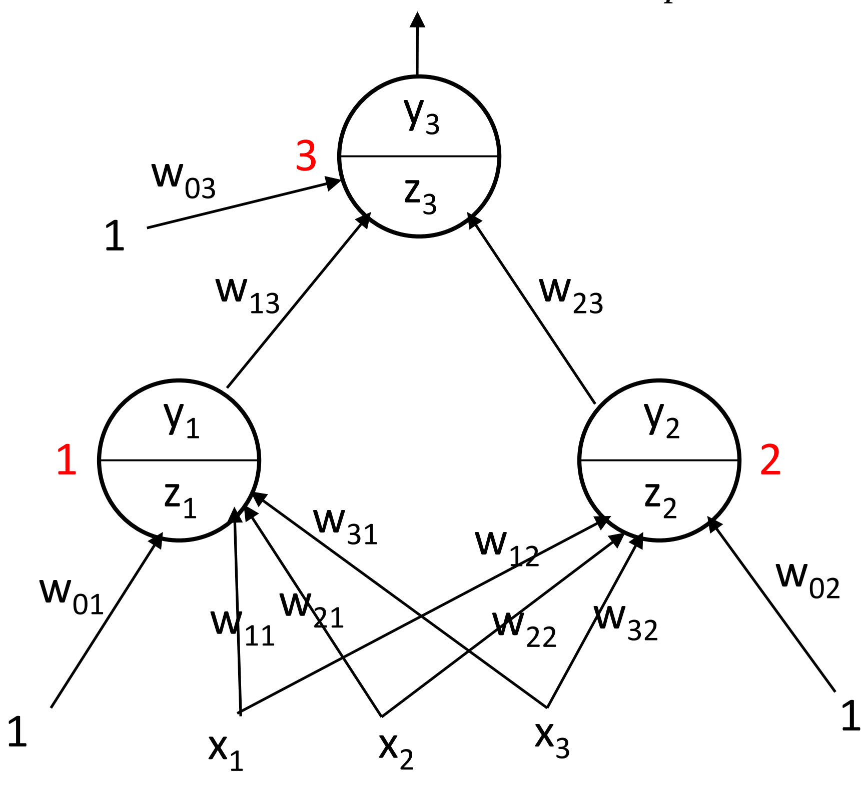
$$\frac{\partial y}{\partial w_{13}} = \frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial w_{13}} = \frac{\partial y}{\partial z_3} y_1$$

$$\frac{\partial y}{\partial w_{11}} = \frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial w_{11}} = \frac{\partial y}{\partial z_3}\left(w_{13}\frac{\partial y_1}{\partial z_1}\frac{\partial z_1}{\partial w_{11}}\right) = \frac{\partial y}{\partial z_3}\left(w_{13}\frac{\partial y_1}{\partial z_1}x_1\right)$$

Learning Weights

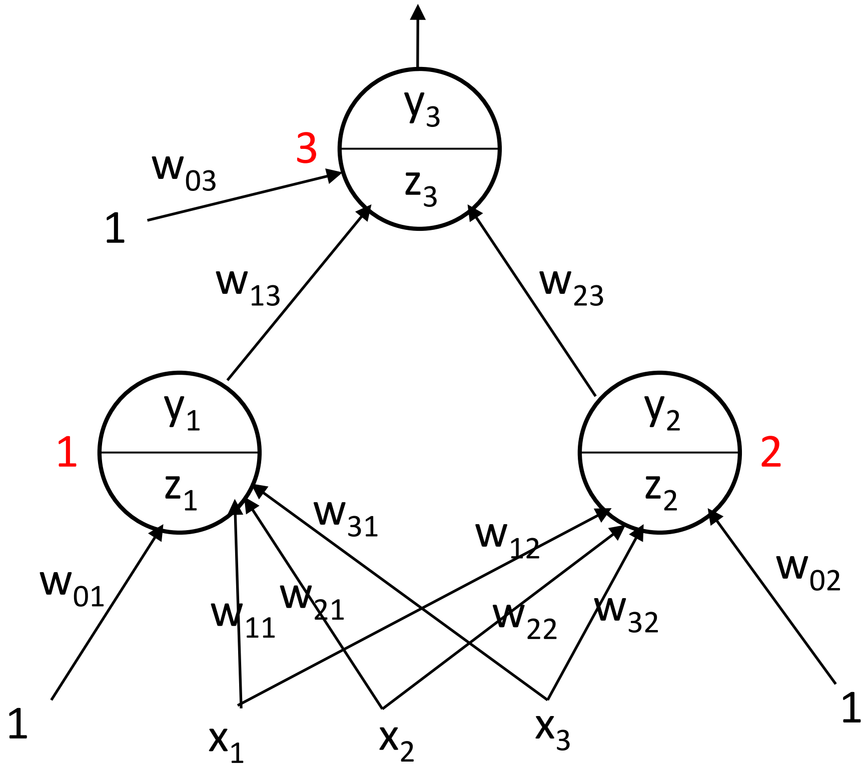
For an output unit p we similarly have:

$$\Delta w_{ip} = -\eta \delta_p y_i^m = -\eta \, y_p (1 - y_p)(y_p - y^m) \, y_i^m$$



p=3 in this example

Backpropagation



First do forward propagation: Compute z_i's and y_i's.

$$\delta_3 = y_3(1-y_3)(y_3 - y^m)$$

$$\delta_2 = y_2(1-y_2)\delta_3 w_{23}$$

$$\delta_1 = y_1(1-y_1)\delta_3 w_{13}$$

$$w_{03} = w_{03} - \eta\delta_3(1)$$

$$w_{13} = w_{13} - \eta\delta_3 y_1$$

$$w_{23} = w_{23} - \eta\delta_3 y_2$$

$$w_{02} = w_{02} - \eta\delta_2(1)$$

$$w_{12} = w_{12} - \eta\delta_2 x_1$$

$$w_{22} = w_{22} - \eta\delta_2 x_2$$

$$w_{32} = w_{32} - \eta\delta_2 x_3$$

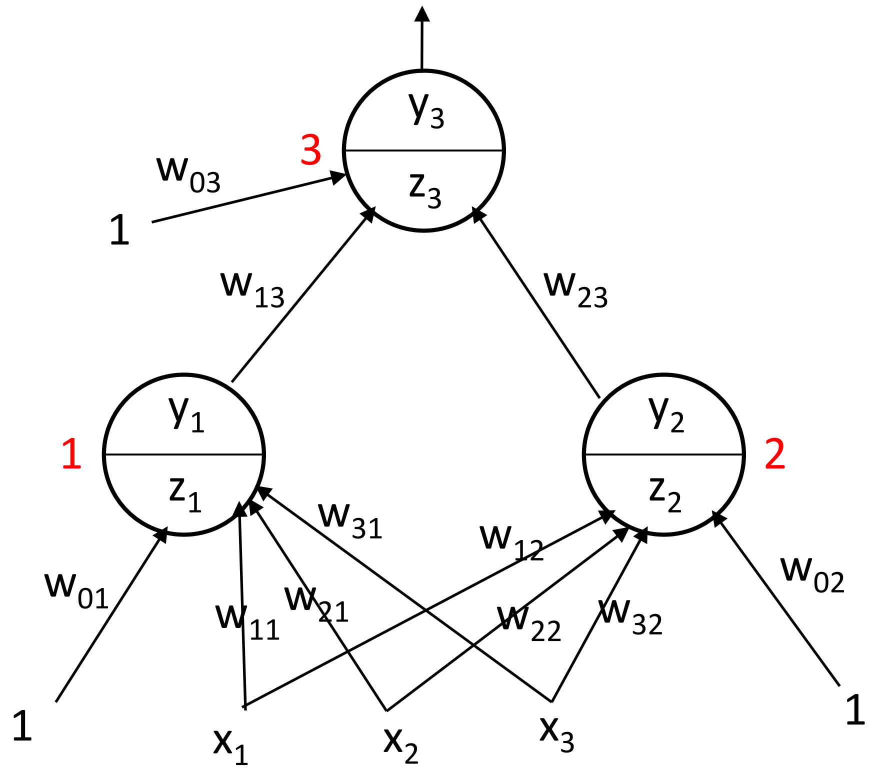
$$w_{01} = w_{01} - \eta\delta_1(1)$$

$$w_{11} = w_{11} - \eta\delta_1 x_1$$

$$w_{21} = w_{21} - \eta\delta_1 x_2$$

$$w_{31} = w_{31} - \eta\delta_1 x_3$$

Backpropagation - Example



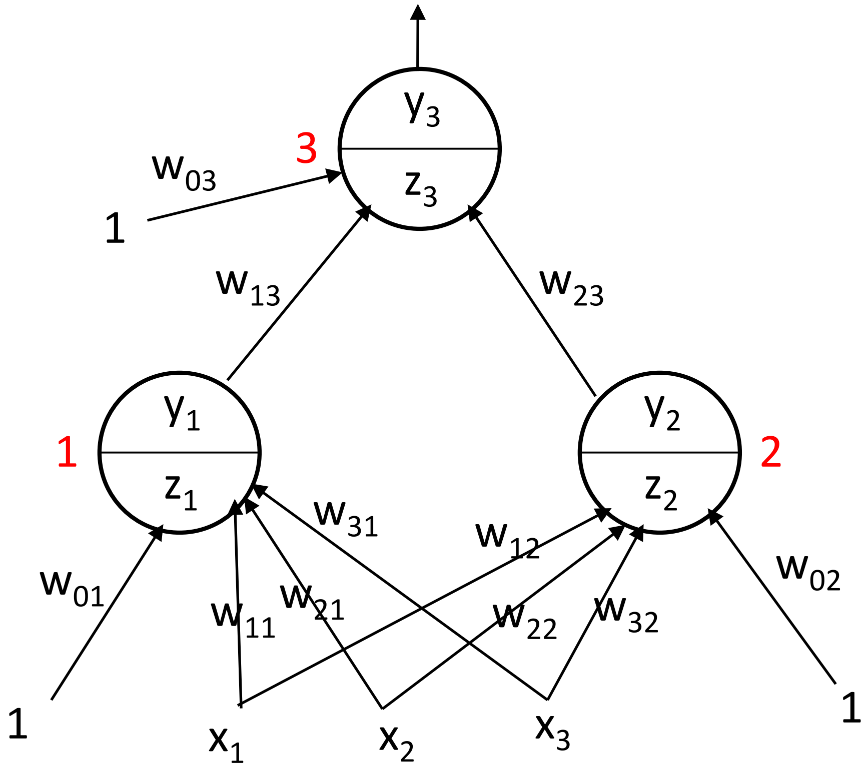
First do forward propagation: Compute z_i's and y_i's.

Suppose we have initially chosen (randomly) the weights given in the table.

Also, in the table is given one training instance (first column).

$x_0 = 1.0$	$w_{01} = 0.5$	$w_{02} = 0.7$	$w_{03} = 0.5$
$x_1 = 0.4$	$w_{11} = 0.6$	$w_{12} = 0.9$	$w_{13} = 0.9$
$x_2 = 0.2$	$w_{21} = 0.8$	$w_{22} = 0.8$	$w_{23} = 0.9$
$x_3 = 0.7$	$w_{31} = 0.6$	$w_{32} = 0.4$	

Feed-Forward Example



$z_1 = 1.0*0.5+0.4*0.6+0.2*0.8+0.7*0.6 = 1.32$
$y_1 = 1/(1+e\verb|^|(-z_1)) = 1/(1+e(-1.32)) = 0.7892$

$z_2 = 1.0*0.7+0.4*0.9+0.2*0.8+0.7*0.4 = 1.5$
$y_2 = 1/(1+e\verb|^|(-z_2)) = 1/(1+e(-1.5)) = 0.8175$

$z_3 = 1.0*0.5+ 0.79*0.9+ 0.82*0.9= 1.95$
$y_3 = 1/(1+e\verb|^|(-z_3)) = 1/(1+e(-1.95)) = 0.87$

$x_0 = 1.0$	$w_{01} = 0.5$	$w_{02} = 0.7$	$w_{03} = 0.5$
$x_1 = 0.4$	$w_{11} = 0.6$	$w_{12} = 0.9$	$w_{13} = 0.9$
$x_2 = 0.2$	$w_{21} = 0.8$	$w_{22} = 0.8$	$w_{23} = 0.9$
$x_3 = 0.7$	$w_{31} = 0.6$	$w_{32} = 0.4$	

Backpropagation

- So, the network output, for the given training example, is y_3=0.87.
- Assume the actual value of the target attribute is y=0.8
- Then the *prediction error* equals 0.8 – 0.8750 = -0.075.

Now

- $\delta_3 = y_3(1-y_3)(y_3-y) = 0.87*(1-0.87)*(0.87-0.8) = 0.008$
- Let's have a learning rate of η=0.01. Then, we update weights:

  $w_{03} = w_{03} - \eta\ \delta_3\ (1) = 0.5 - 0.01*0.008*1 = 0.49918$

  $w_{13} = w_{13} - \eta\ \delta_3\ y_1 = 0.9 - 0.01*0.008*\ 0.7892 = 0.8999$

  $w_{23} = w_{23} - \eta\ \delta_3\ y_2 = 0.9 - 0.01*0.008*\ 0.8175 = 0.8999$

Backpropagation

- $\delta_2 = y_2(1-y_2)\delta_3 w_{23}$ = 0.8175*(1-0.8175)*0.008*0.9 = 0.001
- $\delta_1 = y_1(1-y_1)\delta_3 w_{13}$ = 0.7892*(1- 0.7892)*0.008*0.9 = 0.0012
- Then, we update weights:

$w_{02} = w_{02} - \eta\, \delta_2\, (1)$ = 0.7 - 0.01*0.001*1 = 0.6999
$w_{12} = w_{12} - \eta\, \delta_2\, x_1$ = 0.9 - 0.01*0.001* 0.4 = 0.8999
$w_{22} = w_{22} - \eta\, \delta_2\, x_2$ = 0.8 - 0.01*0.001* 0.2 = 0.7999
$w_{32} = w_{32} - \eta\, \delta_2\, x_3$ = 0.4 - 0.01*0.001* 0.7 = 0.3999

$w_{01} = w_{01} - \eta\, \delta_1\, (1)$ = 0.5 - 0.01*0.001*1 = 0.4999
$w_{11} = w_{11} - \eta\, \delta_1\, x_1$ = 0.6 - 0.01*0.001* 0.4 = 0.5999
$w_{21} = w_{21} - \eta\, \delta_1\, x_2$ = 0.8 - 0.01*0.001* 0.2 = 0.7999
$w_{31} = w_{31} - \eta\, \delta_1\, x_3$ = 0.6 - 0.01*0.001* 0.7 = 0.5999

Backpropagation Algorithm

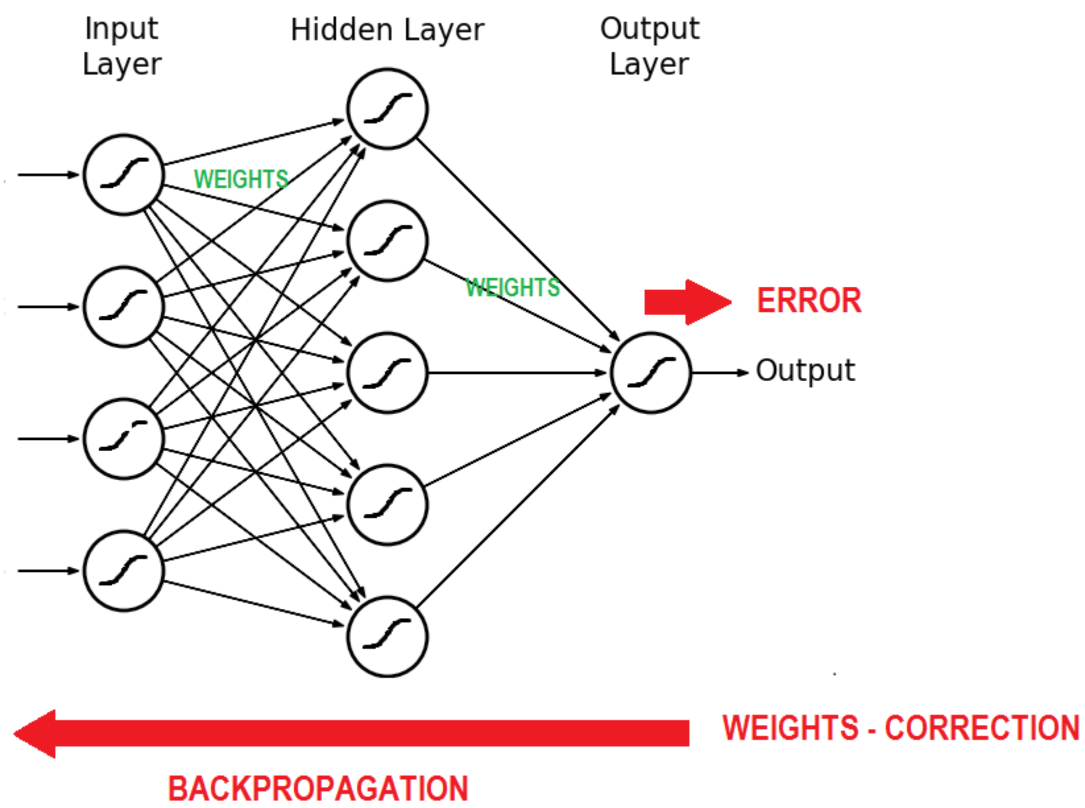
1. Initialize weights to small random values
2. Choose a random sample training item, say $(\mathbf{x}^m, y^m)$
3. Compute total input z_j and output y_j for each unit (***forward prop***)
4. Compute δ_p for output layer $\delta_p = y_p(1-y_p)(y_p-y^m)$
5. Compute δ_j for all preceding layers by ***backprop rule***
6. Compute weight change by ***descent rule*** (repeat for all weights)

- Note that each expression involves data local to a particular unit, we do not have to look around summing things over the whole network.
- It is for this reason, simplicity, locality and, therefore, efficiency that backpropagation has become the dominant paradigm for training neural nets.

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Backpropagation



UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Input Encoding

- For neural networks, all attribute values must be encoded in a standardized manner, taking values between 0 and 1, even for categorical variables.
- For continuous variables, we simply apply the *min-max quantization/normalization*:
  $X^* = [X - \min(X)]/[\max(X)-\min(X)]$
- For categorical variables use *indicator (flag) variables* (ie unary encoding).
  - E.g. *marital status attribute,* containing values *single, married, divorced.*
  - Records for *single* would have
    1 for single, and 0 for the rest, i.e. (1,0,0)
  - Records for *married* would have
    1 for *married*, and 0 for the rest, i.e. (0,1,0)
  - Records for *divorced* would have
    1 for *divorced*, and 0 for the rest, i.e. (0,0,1)
  - Records for *unknown* would have
    0 for all, i.e. (0,0,0)
- In general, categorical attributes with *k* values can be translated into *k*-1 indicator attributes.

Single Output Node

- Neural network output nodes always return a continuous value between 0 and 1 as output.
- Many classification problems can be seen as binary classification problems, with only two possible outcomes.
  - E.g., "Meningitis, yes or not"
- For such problems, one option is to use a single output node, with a threshold value set *a priori* which would separate the classes.
  - For example, with the threshold of "Yes if *output* $\geq$ 0.3," an output of 0.4 from the output node would classify that record as likely to be "Yes".
- Single output nodes may also be used when the classes are clearly ordered. E.g., suppose that we would like to classify patients' disease levels. We can say:
  - If $0 \leq$ *output* $<$ 0.33, classify *"mild"*
  - If $0.33 \leq$ *output* $<$ 0.66, classify "*severe*"
  - If $0.66 \leq$ *output* $<$ 1, *classify "grave"*

Multiple Output Nodes

- If we have unordered categories for the target attribute, we create one output node for each possible category and encode the target classes with unary codes.
  - E.g. for marital status as target attribute, the network would have four output nodes in the output layer, one for each of:
    - Single (1,0,0,0), married (0,1,0,0), divorced (0,0,1,0), and unknown (0,0,0,0).
- Output node with the highest *real* value is then chosen as the classification for that particular sample.

Multiple Output Nodes

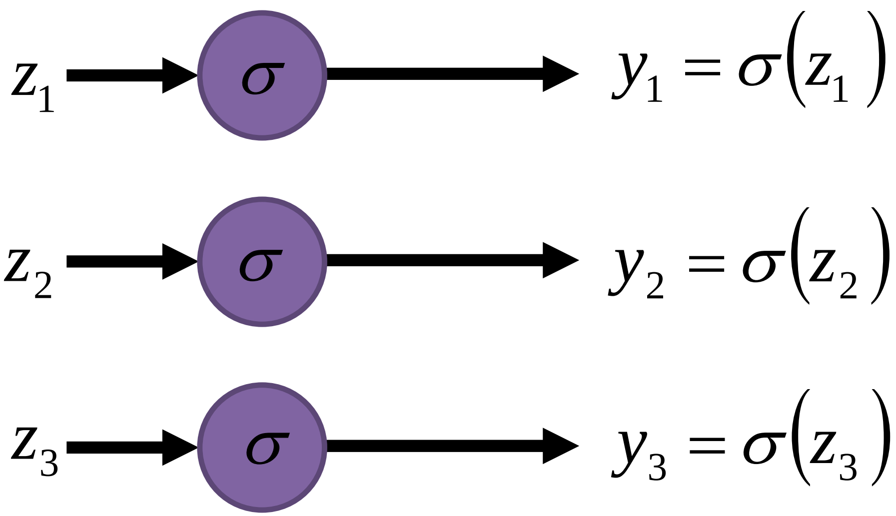
- However, there may be some confusing outputs: two Outputs > 0.5 and with very close values in unary encoding.
- A more sophisticated method is to use special ***softmax*** units as the output nodes, which forces outputs to sum to 1.

$$P(y=j|\mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

Softmax…

- ***Which output layer?*** **Softmax layer!**

Ordinary Layer



In general, the output of network can be any value.

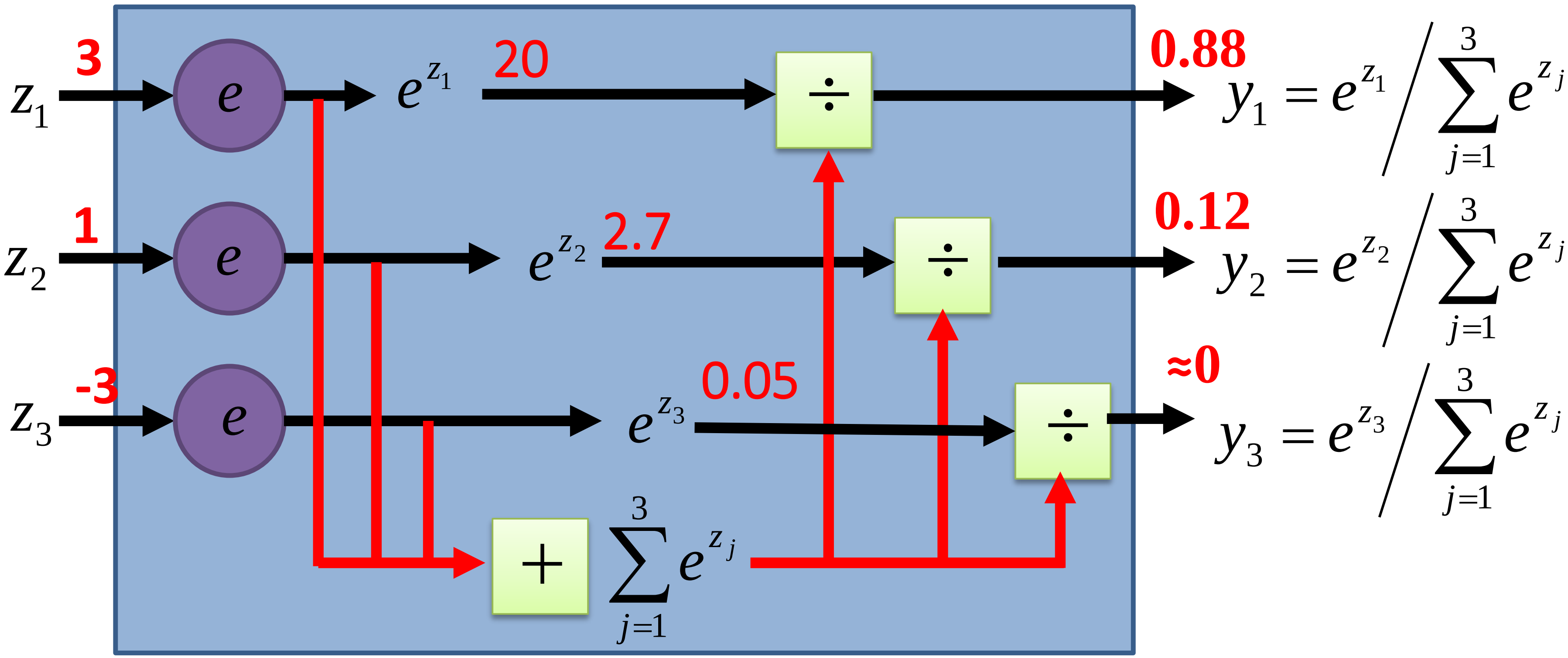
May not be easy to interpret

Softmax!

- **Softmax layer as the output layer**

Probability:
- $1 > y_i > 0$
- $\sum_i y_i = 1$

Softmax Layer



Training neural nets

without overfitting, hopefully…

Given: Data set, desired outputs and a neural net with m weights. Find a setting for the weights that will give good predictive performance on new data. Estimate expected performance on new data.

1. Split data set (randomly) into three subsets:
   - Training set – used for picking weights
   - Validation set – used to stop training
   - Test set – used to evaluate performance
2. Pick random, small weights as initial values
3. Perform iterative minimization of error over training set.
4. Stop when error on validation set reaches a minimum (to avoid overfitting).
5. Repeat training (from step 2) several times (avoid local minima)
6. Use best weights to compute error on test set, which is estimate of performance on new data. Do not repeat training to improve this.

Can use cross-validation if data set is too small to divide into three subsets.

Autonomous Land Vehicle In a Neural Network (ALVINN)

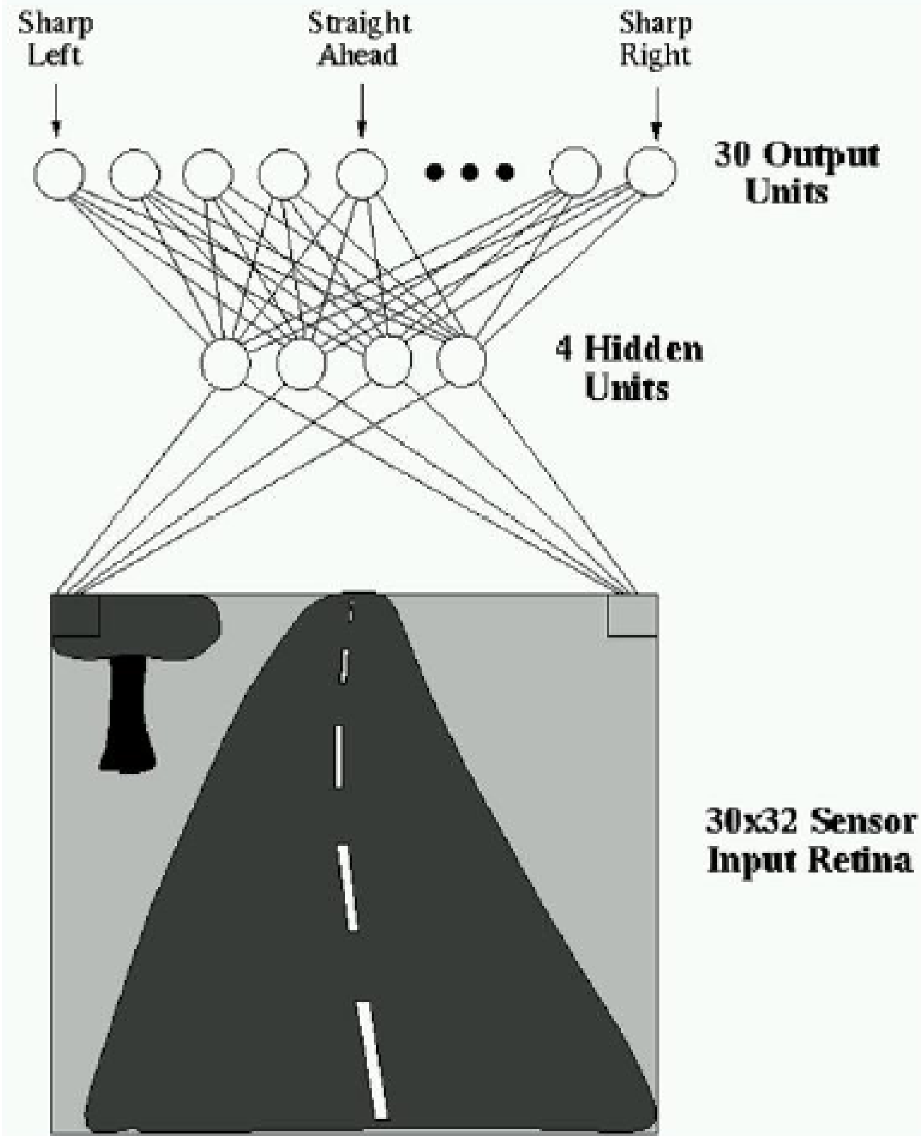
ALVINN is an automatic steering system for a car based on input from a camera mounted on the vehicle.

Successfully demonstrated in a cross-country trip

Dean Pomerleau
CMU

ALVINN (1989)

- The ALVINN neural network is shown here. It has
  - 960 inputs (a 30x32 array derived from the pixels of an image),
  - 4 hidden units and
  - 30 output units (each representing a steering command).



Some observations…

- Although Neural Nets kicked off the current phase of interest in machine learning, they are extremely problematic…
  - Too many parameters (weights, learning rate, momentum, etc)
  - Hard to choose the architecture
  - Very slow to train
  - Easy to get stuck in local minima
- Interest has shifted to other methods, such as support vector machines, which can be viewed as variants of perceptrons (with a twist or two).

SVMs vs. ANNs

- Comparable in practice.

- Some comment:

  "*SVMs have been developed in the reverse order to the development of neural networks (NNs). SVMs evolved from the sound theory to the implementation and experiments, while the NNs followed more heuristic path, from applications and extensive experimentation to the theory.*" (Wang 2005)

- Recently, new deep architectures brought back neural networks to “the front of the scene”.