From Shallow to Deep representation for multimedia data classification

Lecture #5

Artificial Neural Networks, Convolutional Neural Networks and Adversarial Examples

Frédéric Precioso

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

ARTIFICIAL NEURAL NETWORKS

UNIVERSITÉ CÔTE D'AZUR

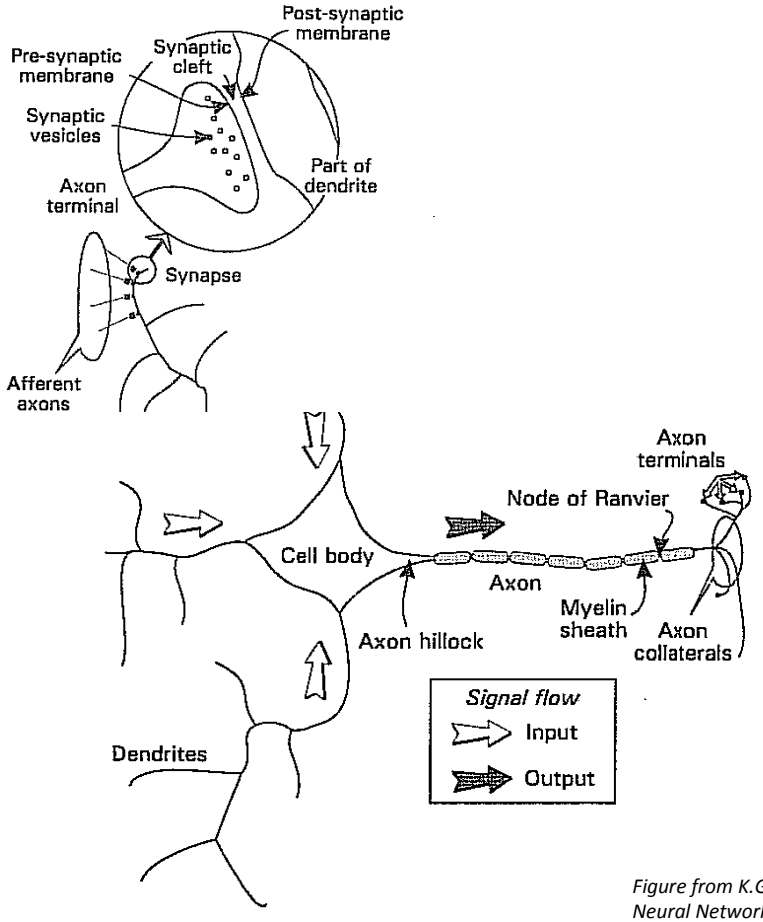
Initial Model: Perceptron

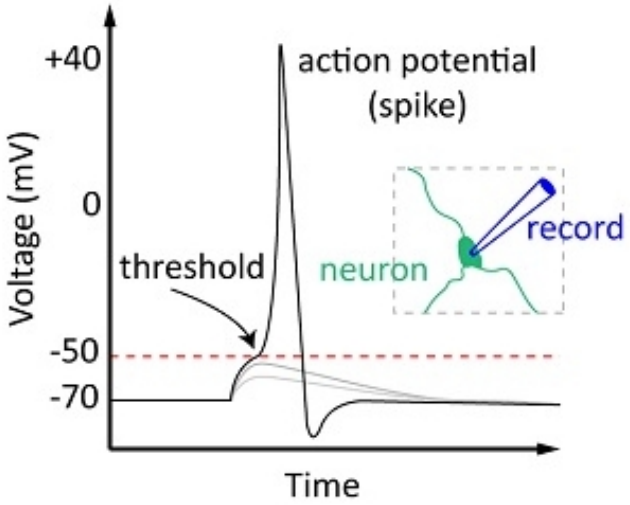
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Biological neuron

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





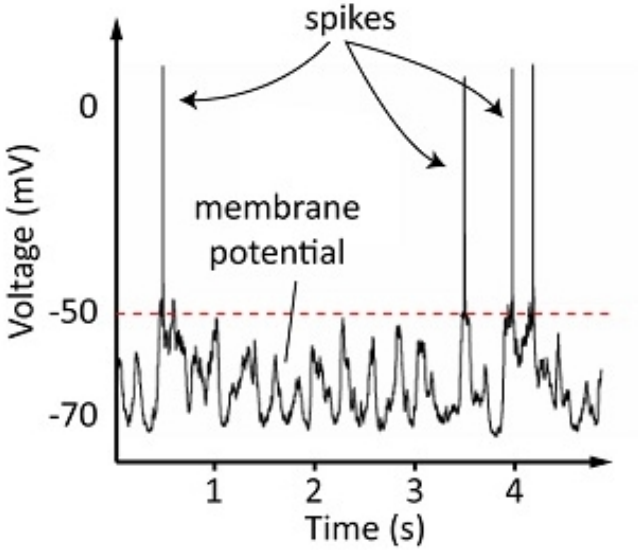


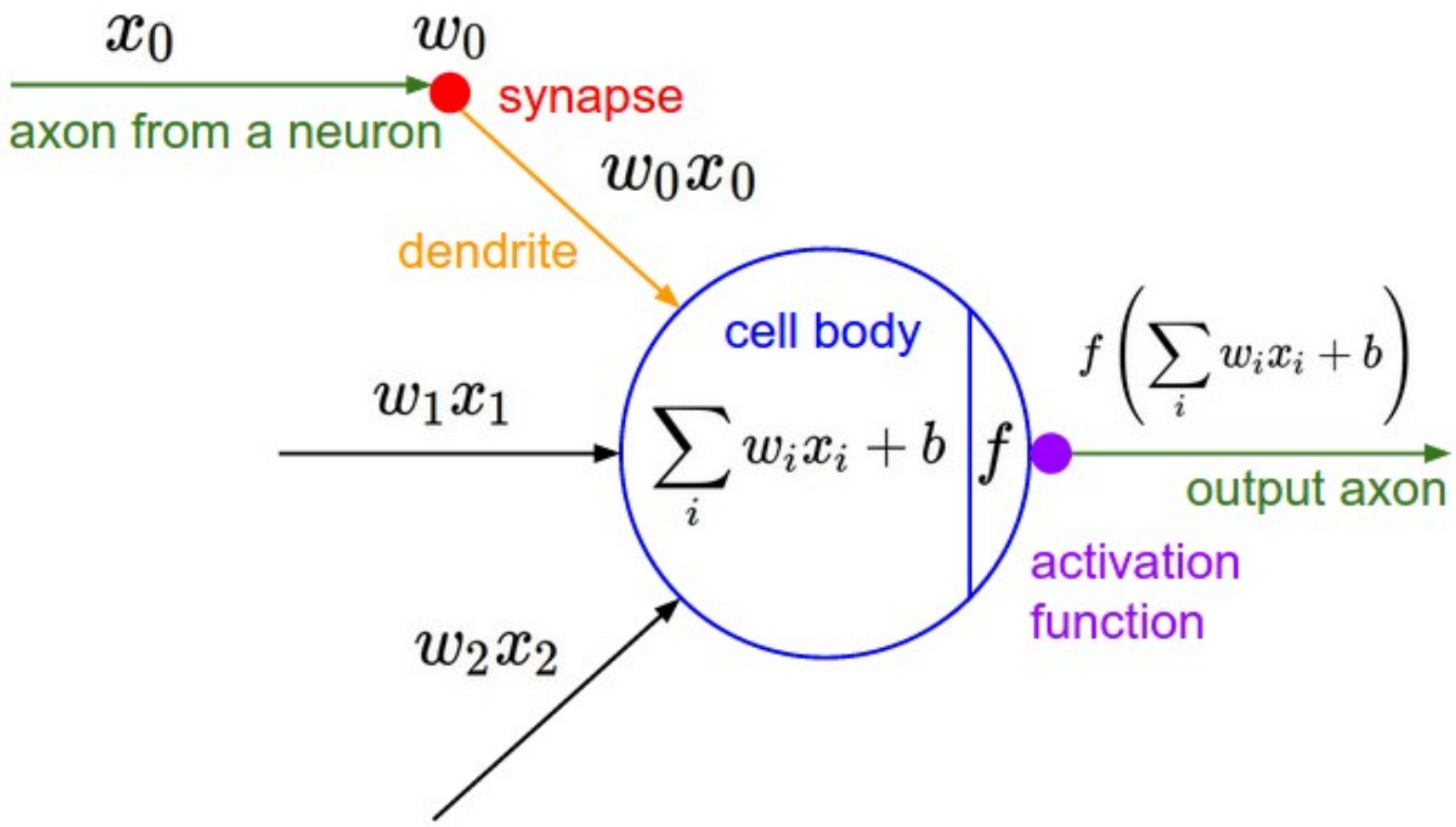
Figure from K.Gurney, An Introduction to Neural Networks

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

A single artificial neuron

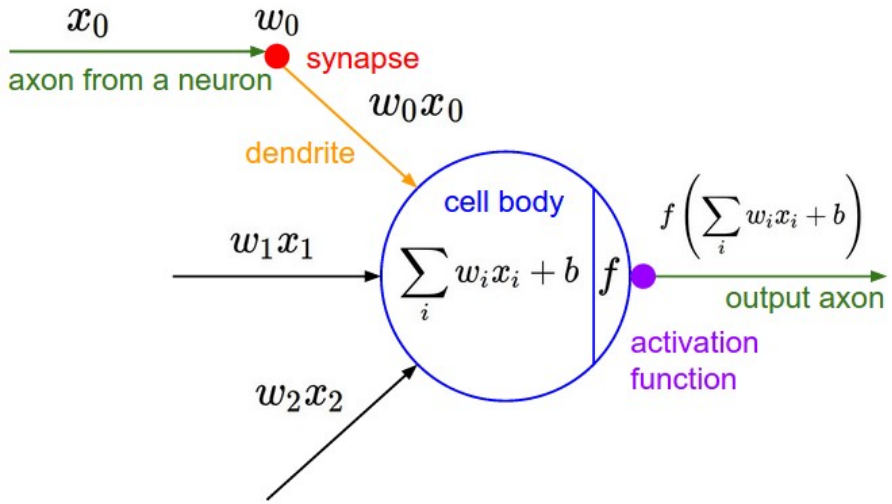
- It is a **very simple abstract** of a biological neuron (McCulloch & Pitts, 1943)



UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

A single artificial neuron



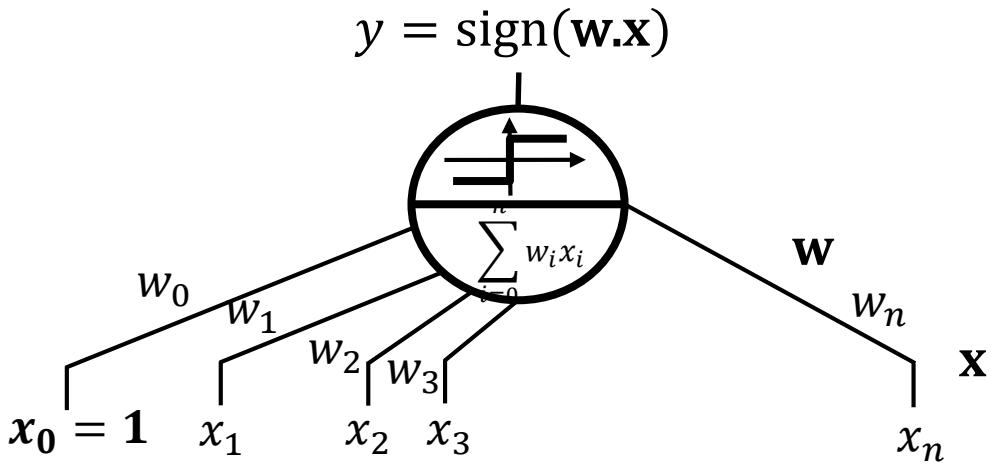
- Each input *x* has an associated **weight *w*** which can be modified
- Inputs *x* corresponds to signals from other neuron axons
  - x_0 - Bias are 'special' inputs, with weight w_0
- Weights ***W*** corresponds to synaptic modulation (i.e. something like strength/amount of neurotransmitters)
- The summation corresponds to 'cell body'
- The activation function corresponds to axon hillock - computes some function *f* of the weighted sum of its inputs
- So, output *y=f(z)*, corresponds to axon signal

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

- We start by looking at a simpler kind of "neural-like" unit called a **Perceptron**.

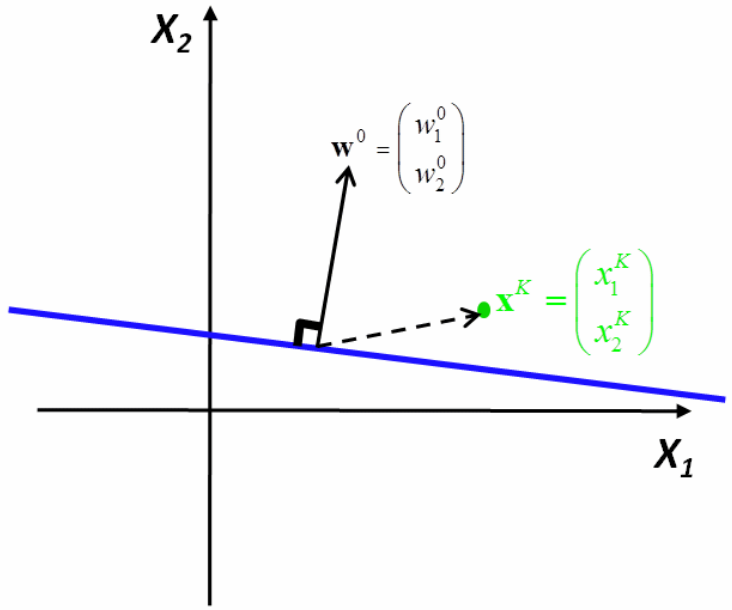


let $x_0 = 1$ and $w_0 = b$

then the hyperplane equation is

$$\mathbf{w} \cdot \mathbf{x} = 0$$

or $$\sum_{j=0}^{n} w_j x_j = 0$$

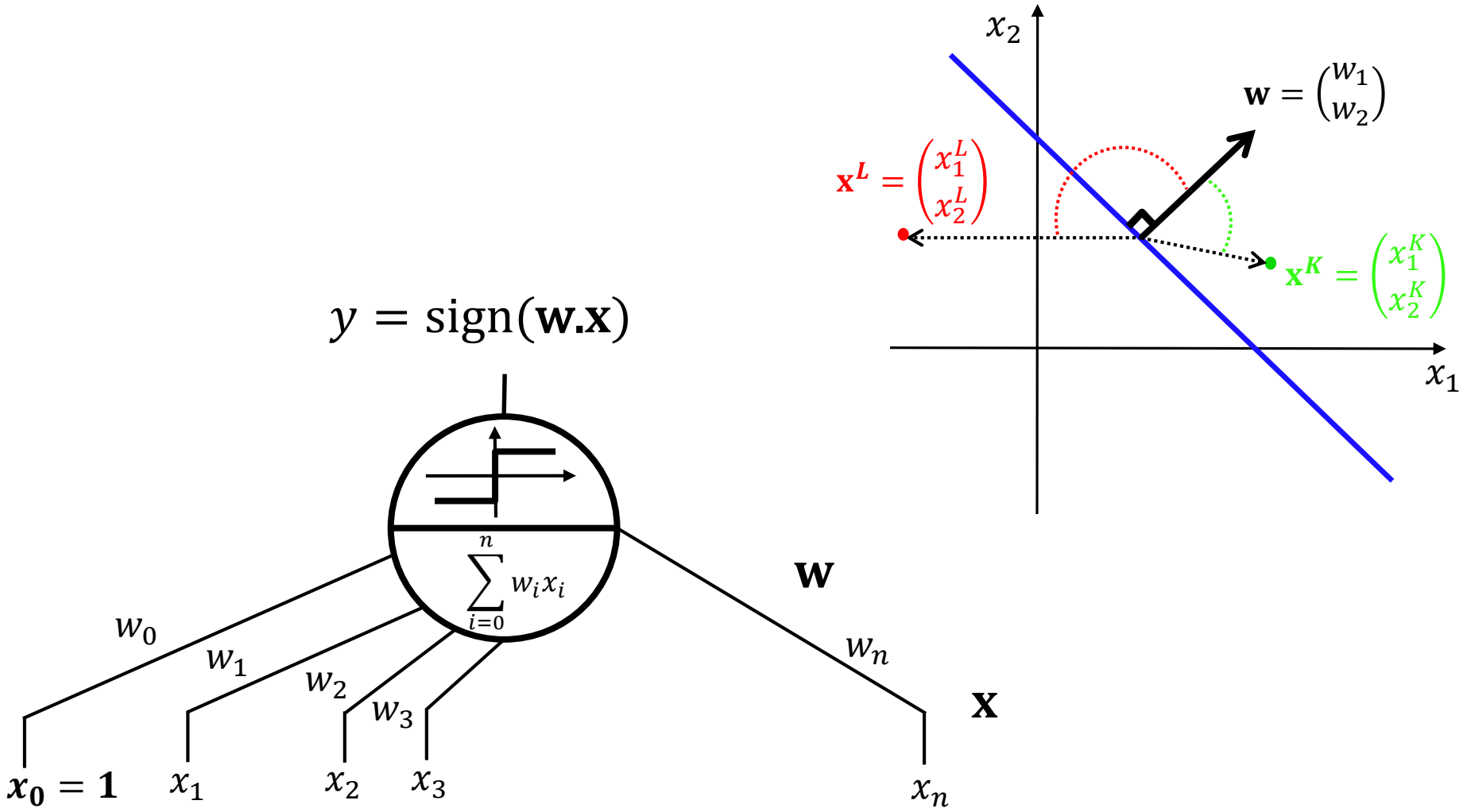


2D illustration

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

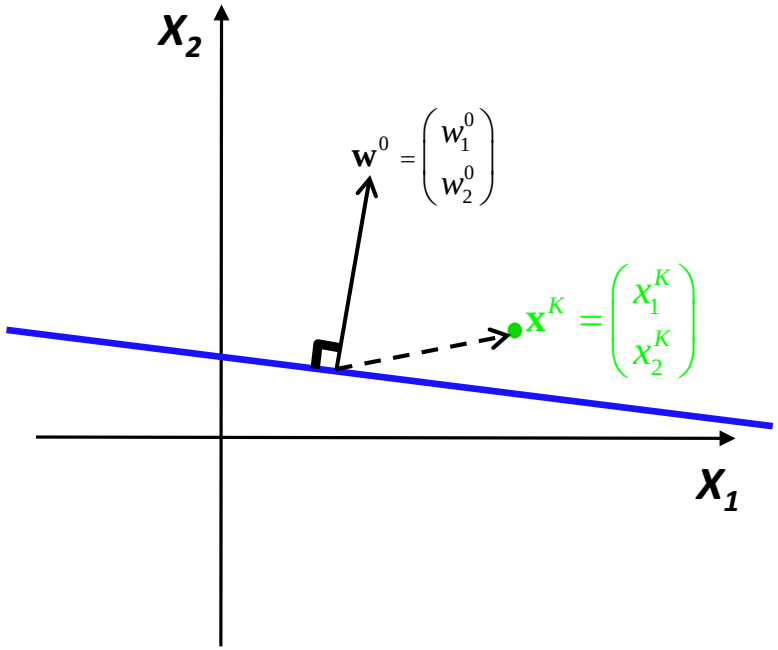


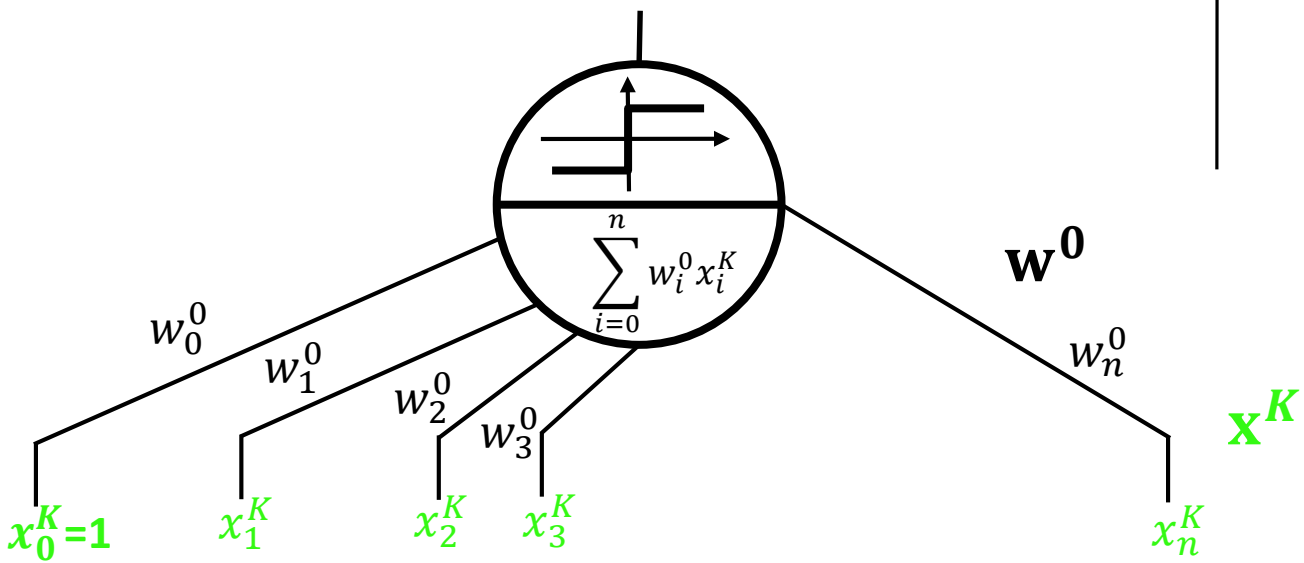
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

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



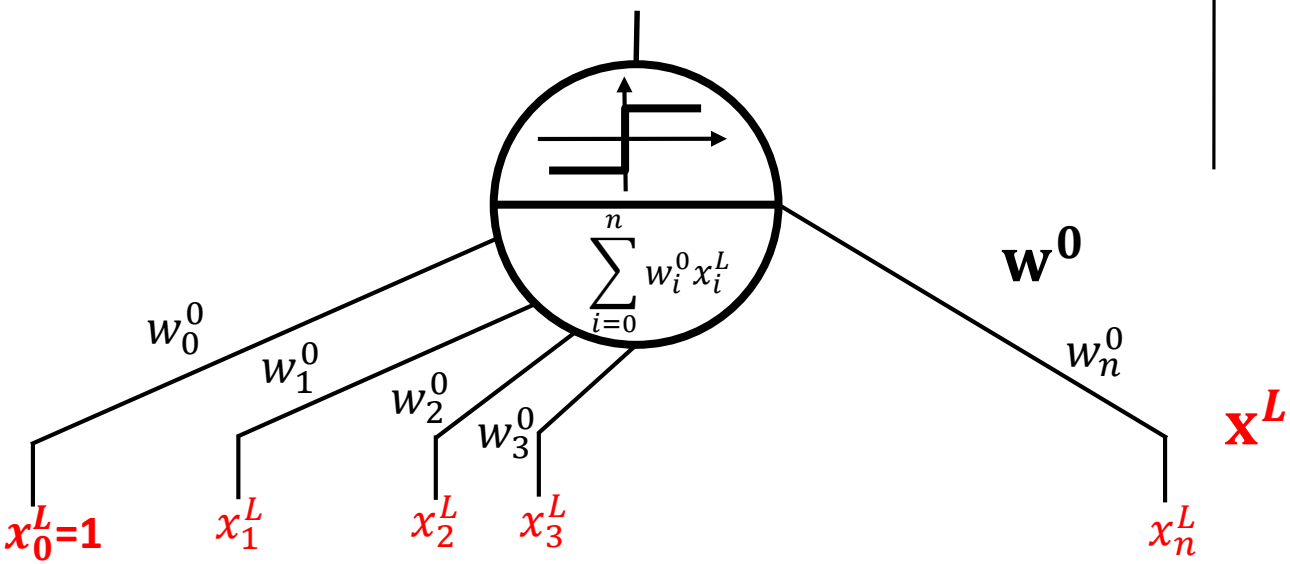


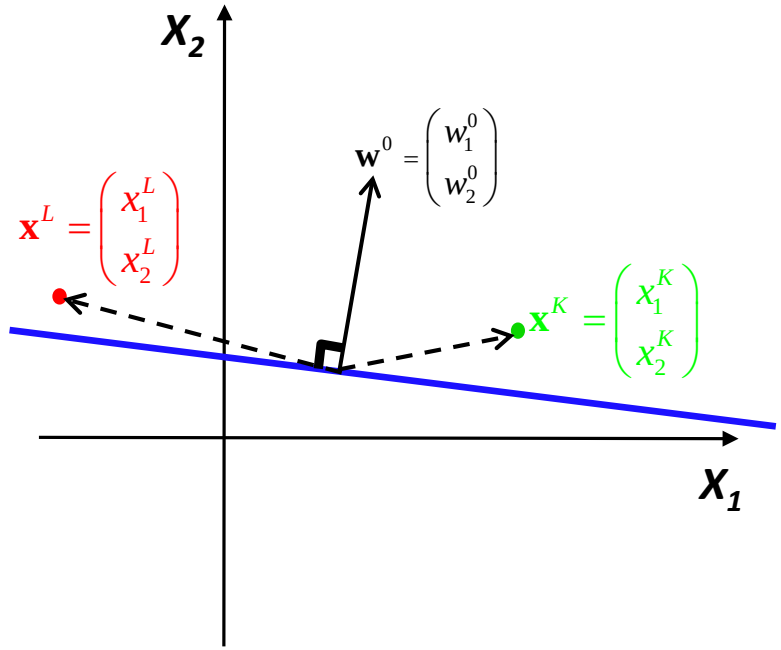
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

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



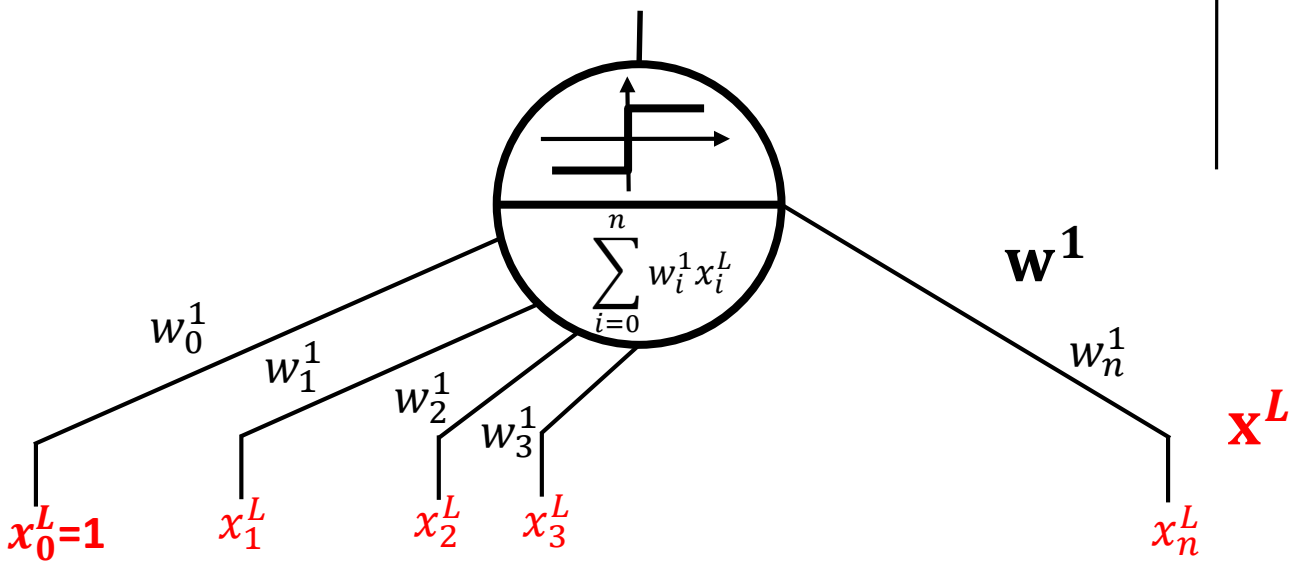


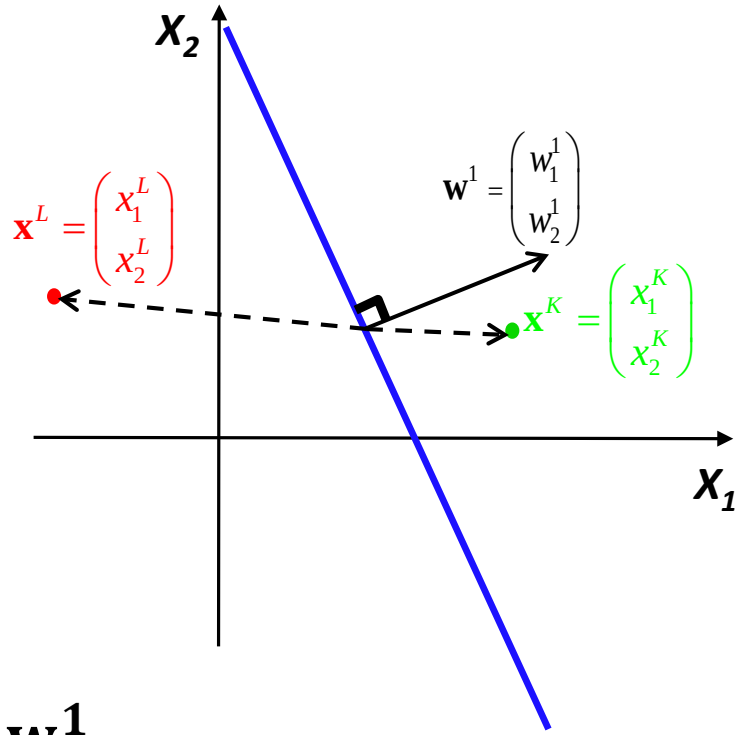
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

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

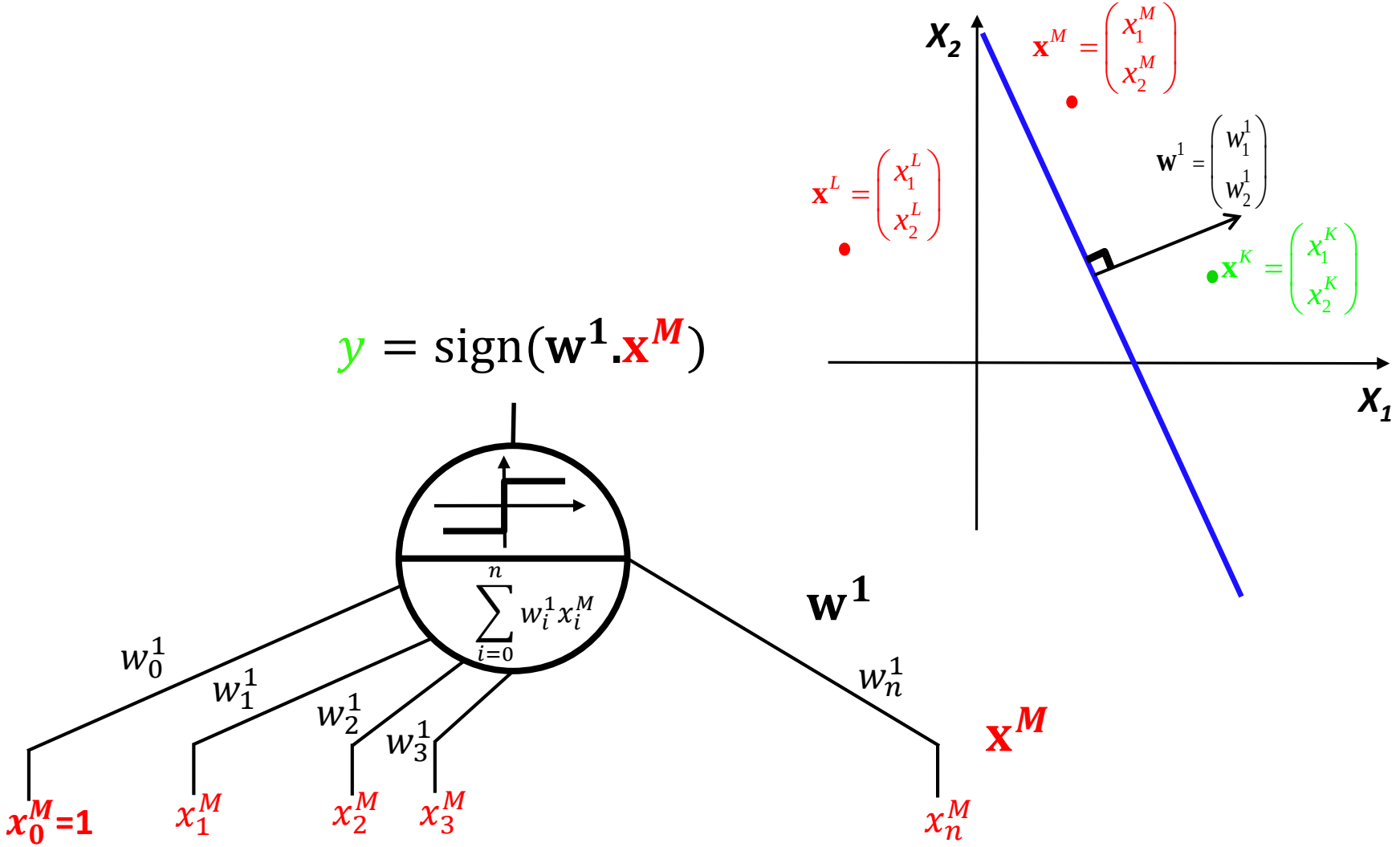




UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

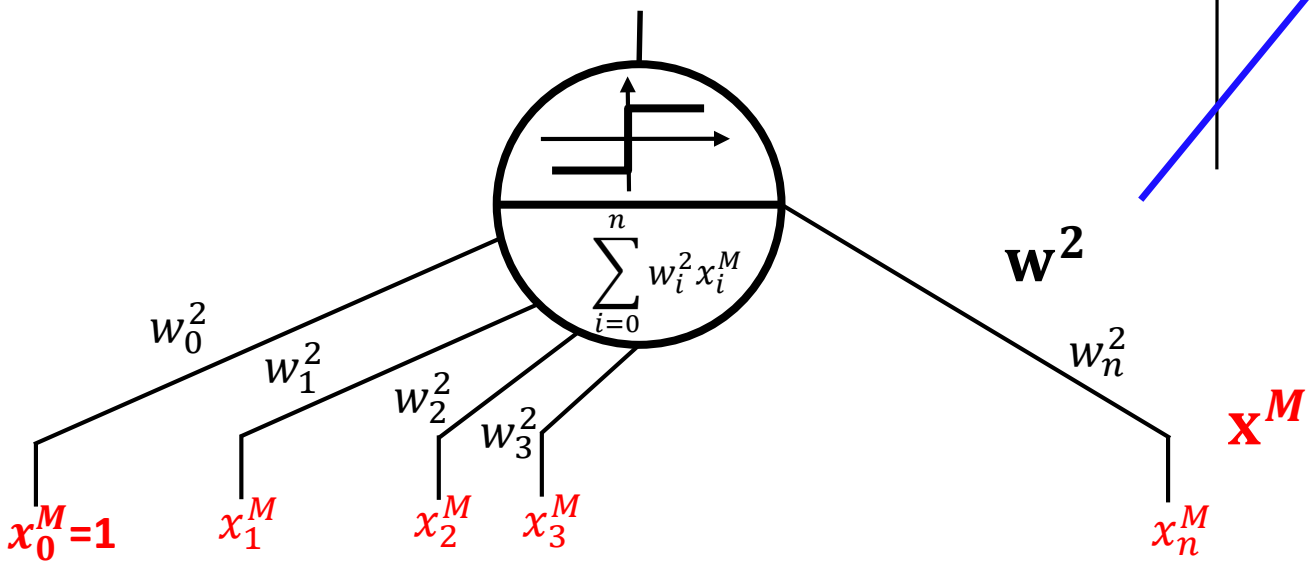


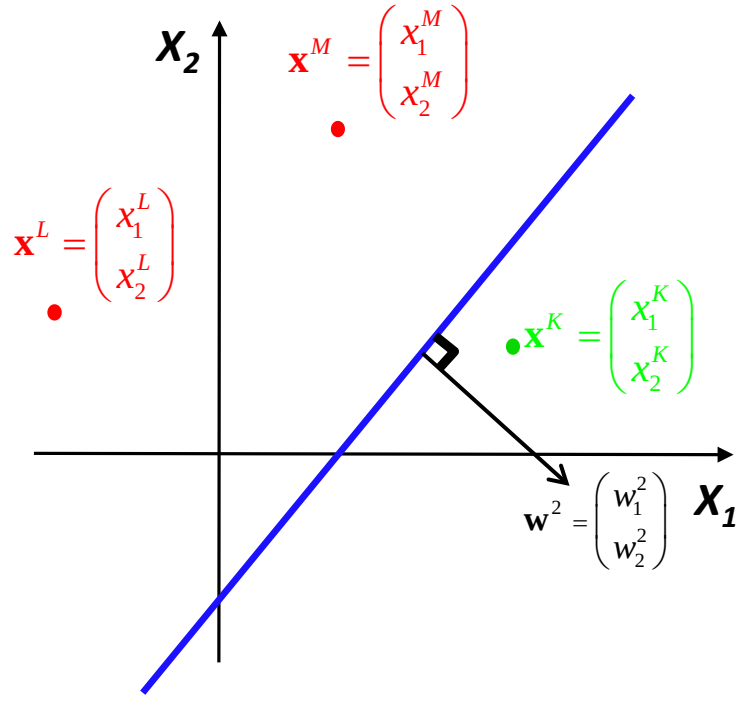
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

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



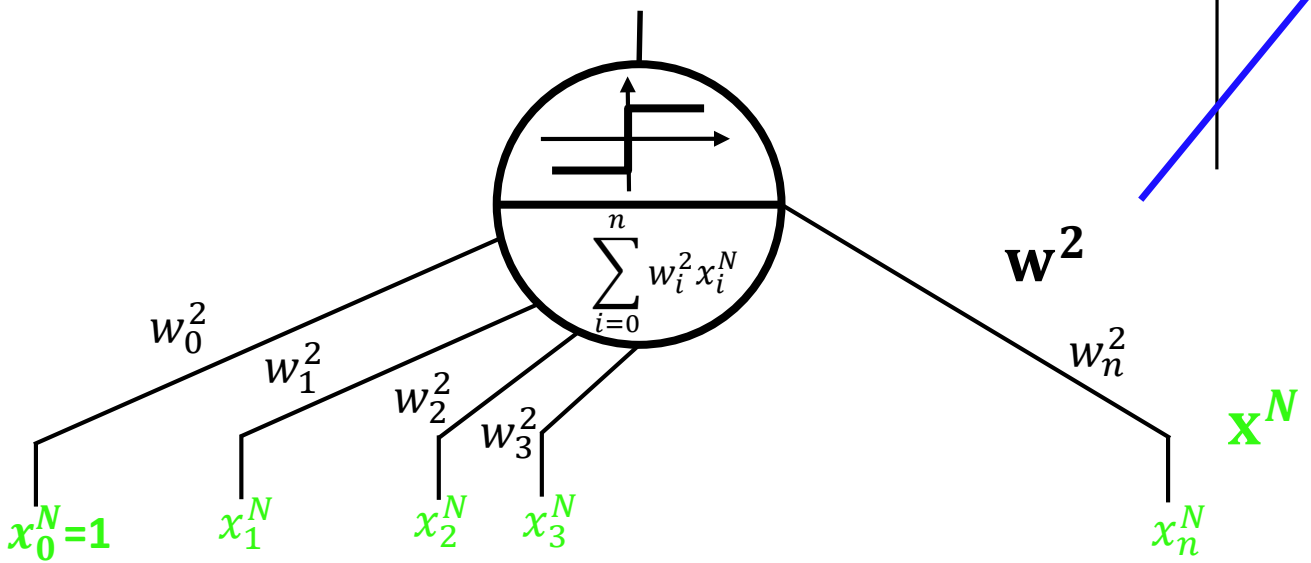


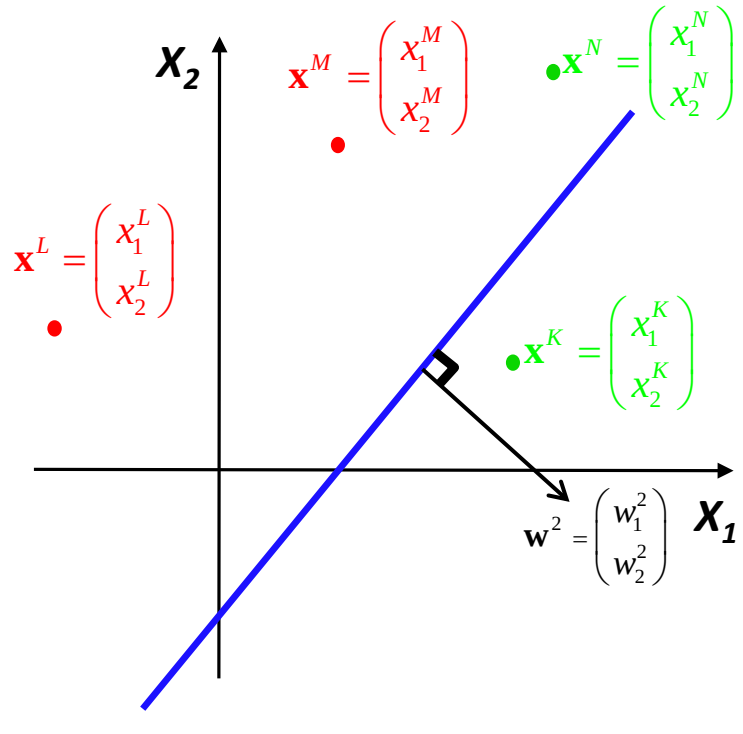
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

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



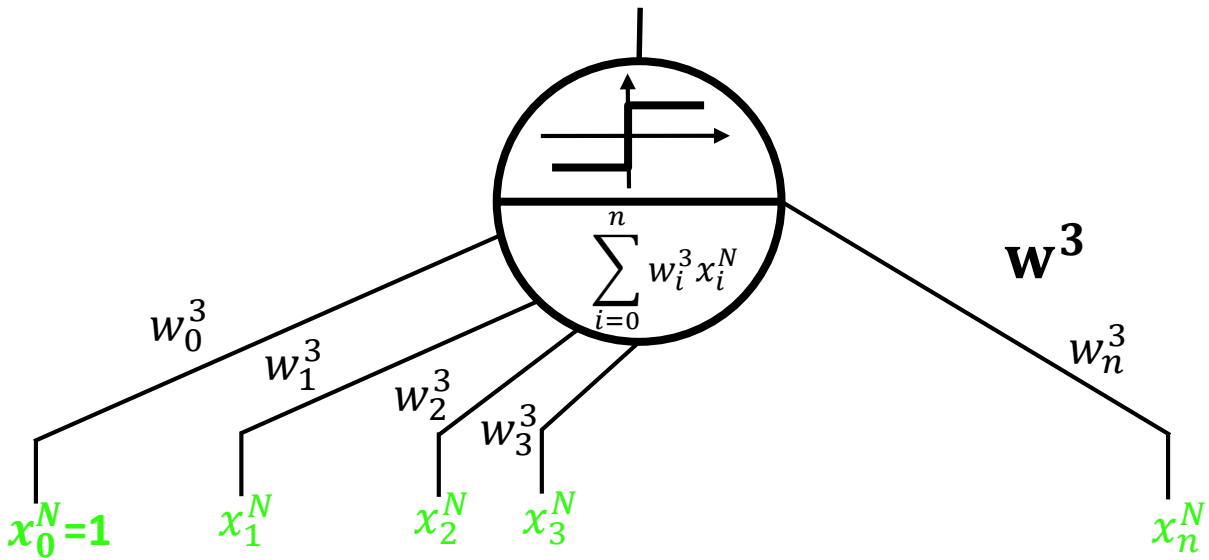


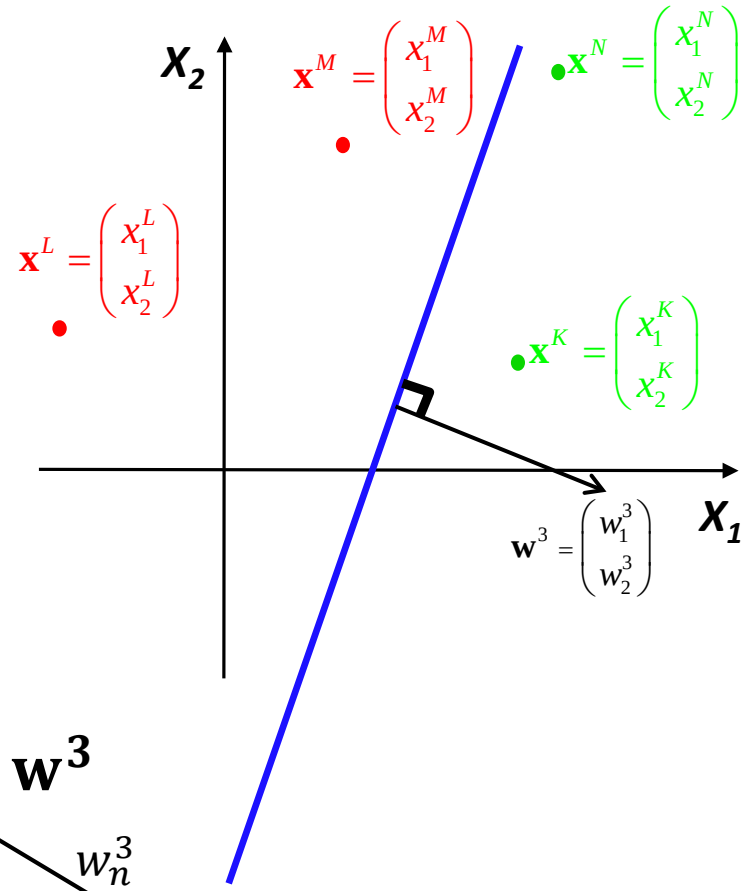
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

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



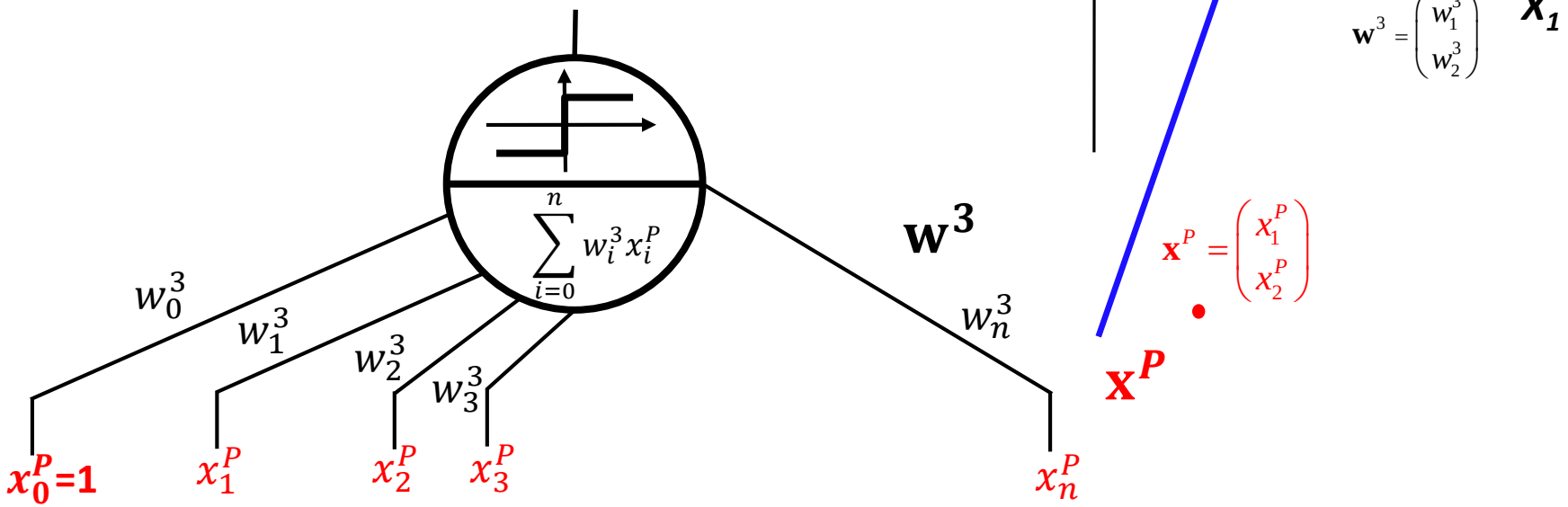


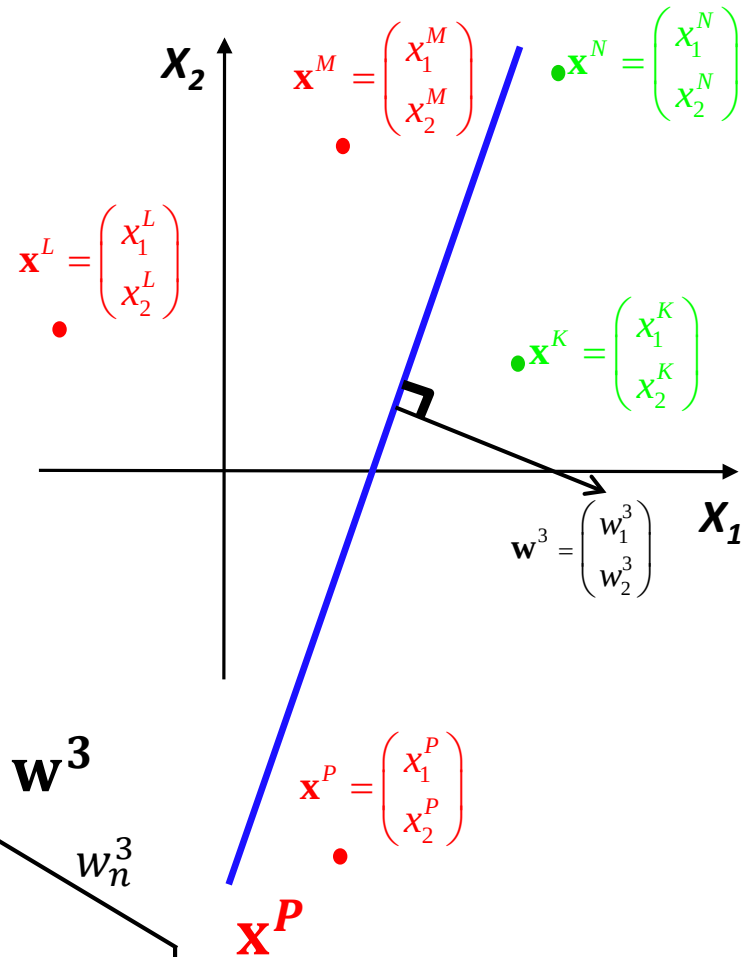
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

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



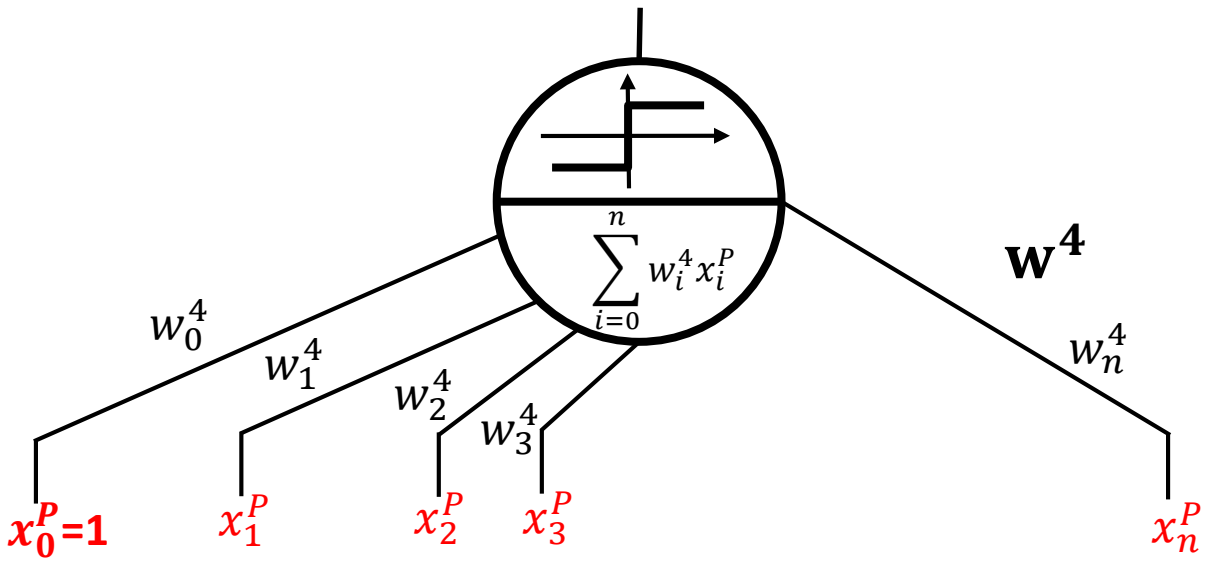


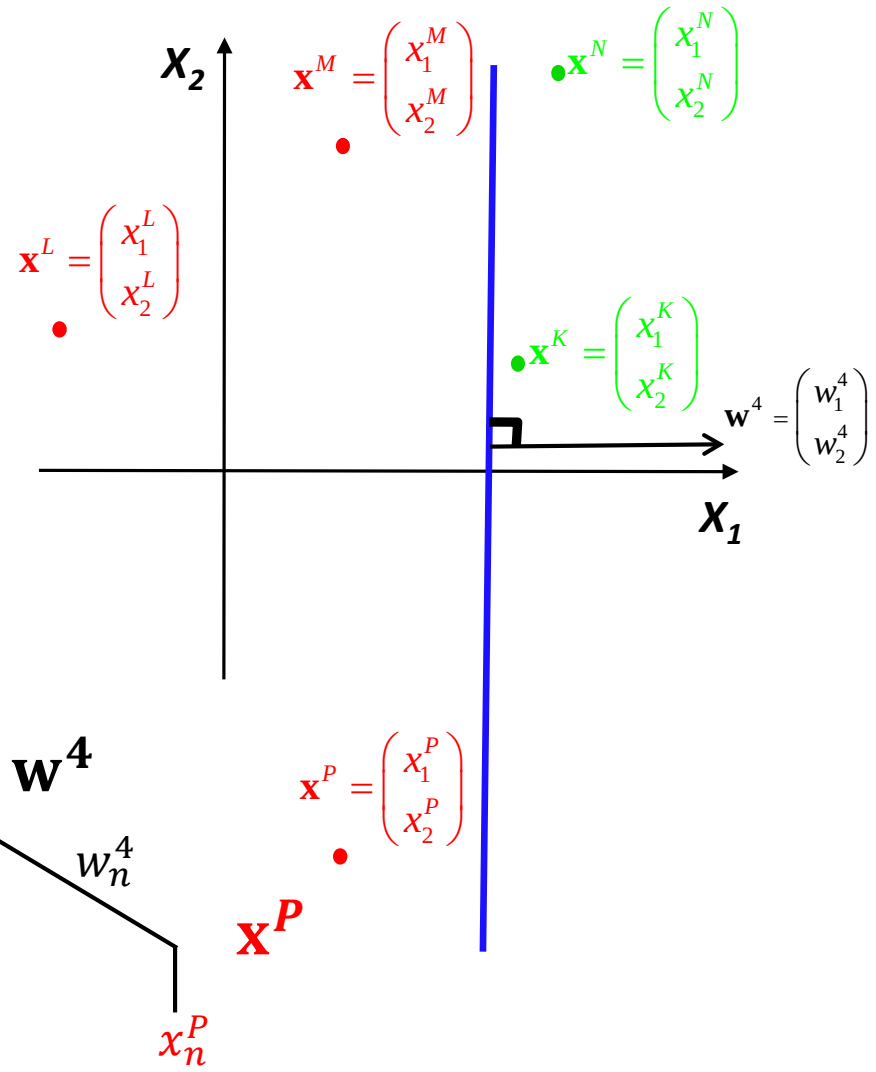
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Single Perceptron Unit

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





UNIVERSITÉ CÔTE D'AZUR

Perceptron: Rosenblatt's Algorithm (1956-1958)

UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

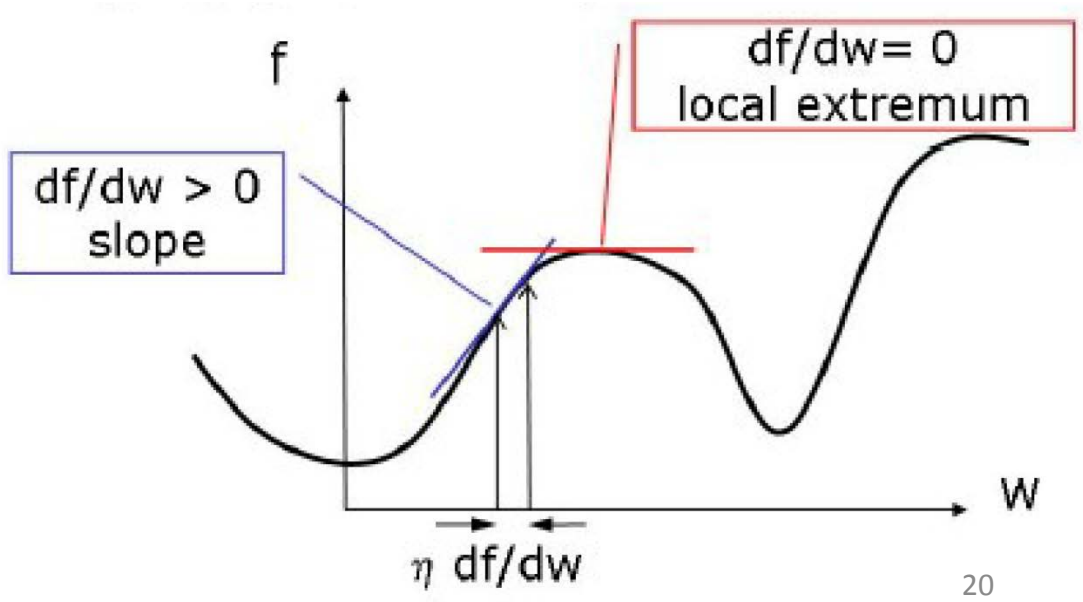
Perceptron Algorithm

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

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

Gradient 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 $\mathbf{w}$ to $\mathbf{w} + \eta\, df/d\mathbf{w}$ ($\eta > 0$, small)
  - Until f stops changing ($df/d\mathbf{w} \approx 0$)



Gradient Ascent

- To maximize a multivariate function $f(\mathbf{w})$
  - Pick initial $\mathbf{w}$
  - Change $\mathbf{w}$ to $\mathbf{w} + \eta \nabla f_{\mathbf{w}}$ ($\eta > 0$, small)
  - Until f stops changing ($\nabla f_{\mathbf{w}} \approx 0$)
- Find local maximum, unless function is globally convex

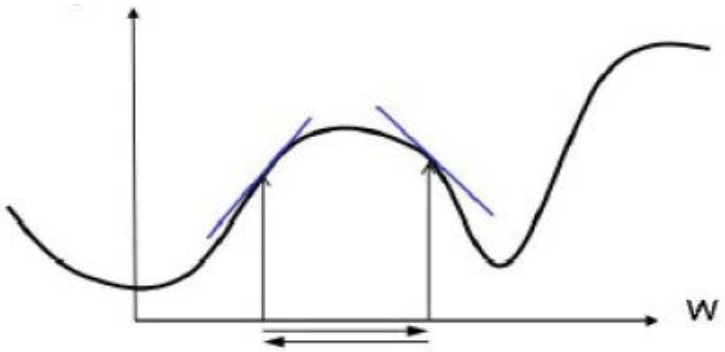
$$\nabla f_{\mathbf{w}} = \left[\frac{\partial f}{\partial \mathbf{w}_1}, \quad \dots \quad , \frac{\partial f}{\partial \mathbf{w}_n} \right]$$

Gradient Ascent

- To maximize a multivariate function $f(\mathbf{w})$
  - Pick initial $\mathbf{w}$
  - Change $\mathbf{w}$ to $\mathbf{w} + \eta \nabla f_{\mathbf{w}}$ ($\eta > 0$, small)
  - Until f stops changing ($\nabla f_{\mathbf{w}} \approx 0$)

$$\nabla f_{\mathbf{w}} = \left[\frac{\partial f}{\partial \mathbf{w}_1}, \quad \ldots \quad , \frac{\partial f}{\partial \mathbf{w}_n} \right]$$

- Find local maximum, unless function is globally convex
- If f is non-linear, the learning rate η has to be chosen very carefully
  - Too small $\Rightarrow$ slow convergence
  - Too big $\Rightarrow$ osccilations



Gradient Ascent

- Maximize margin of misclassified points

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

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

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

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

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

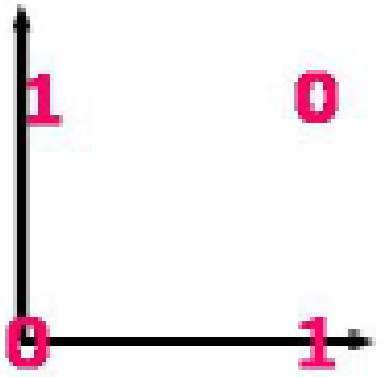
- If data are not linearly separable, then this algorithm loops indefinitely

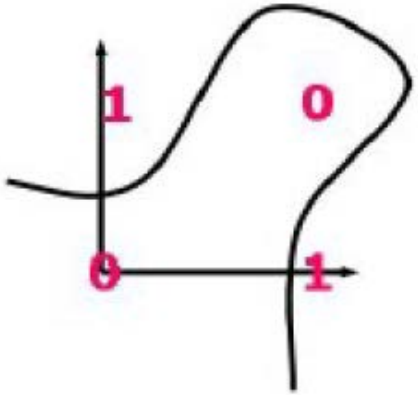
Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6):386.

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

UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

Thomas Cover's Theorem (1965)

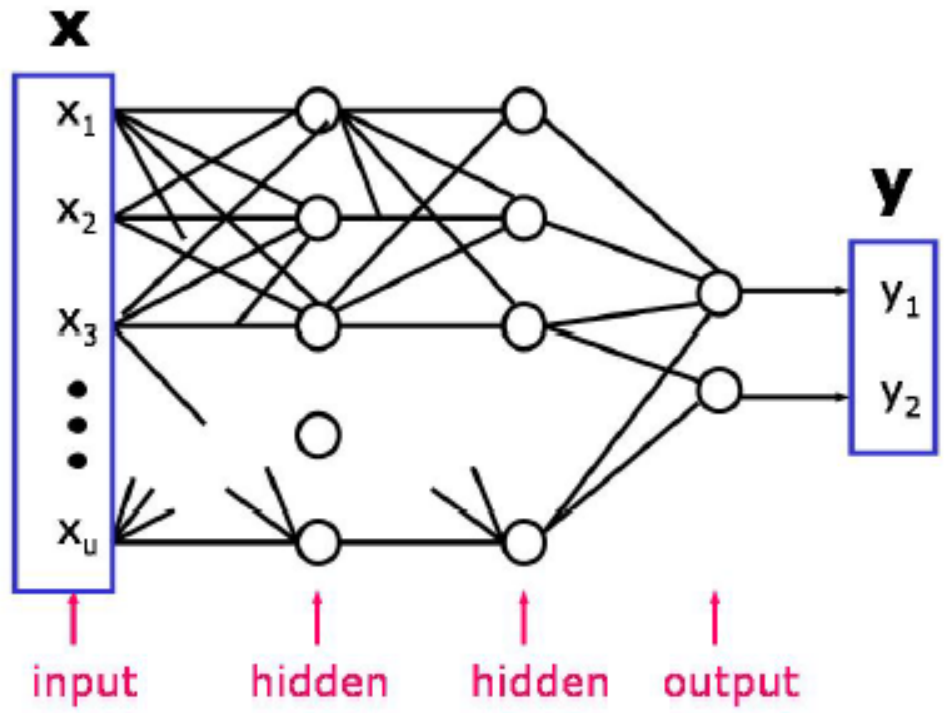
Cover's theorem states: A complex pattern-classification problem cast in a high-dimensional space nonlinearly is more likely to be linearly separable than in a low-dimensional space.

(repeated sequence of Bernoulli trials)

UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

Solution: Multi-Layer Perceptron

- **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.

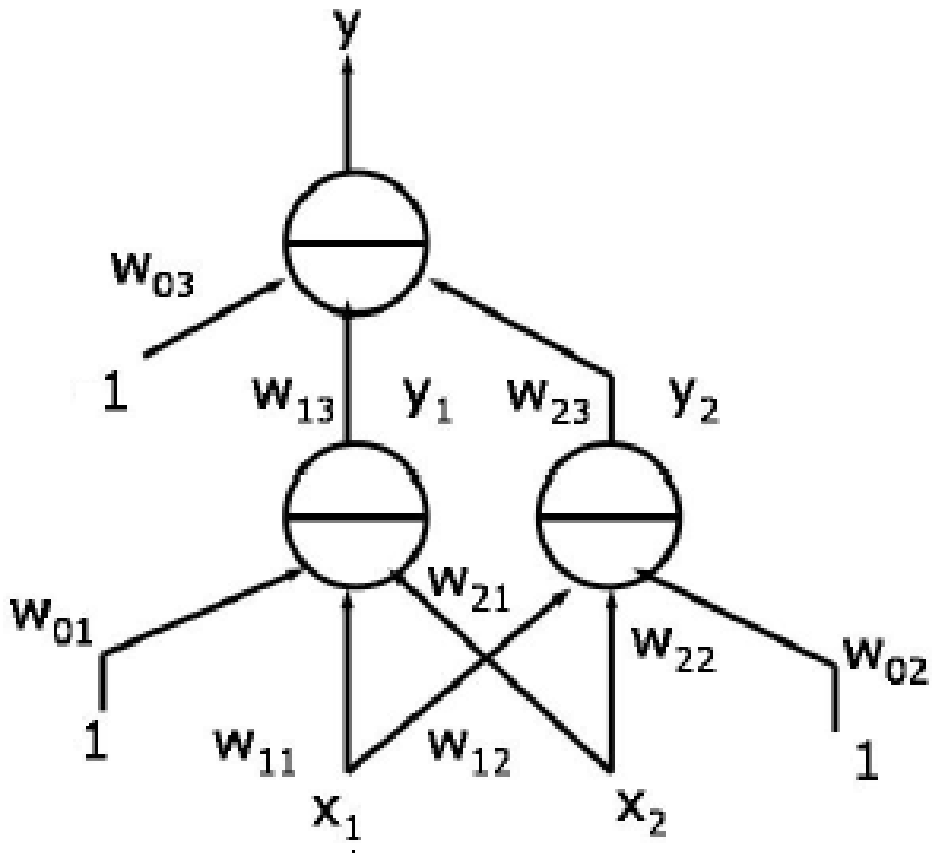


UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

Feedforward networks

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

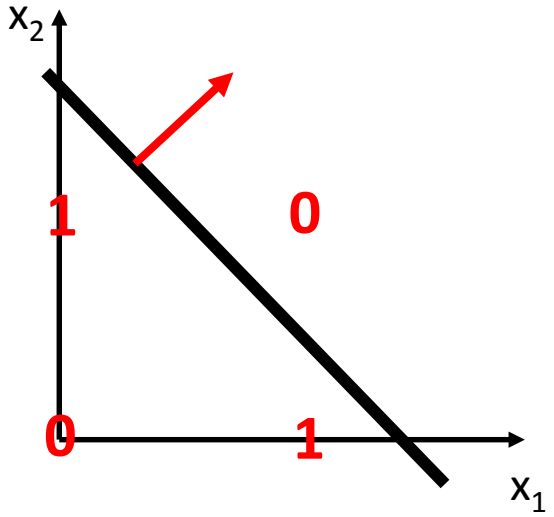
- Learning takes place by adjusting the weights in the network, so that the desired output is produced whenever a training instance is presented.

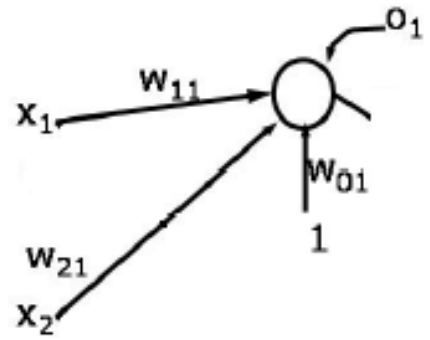


UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

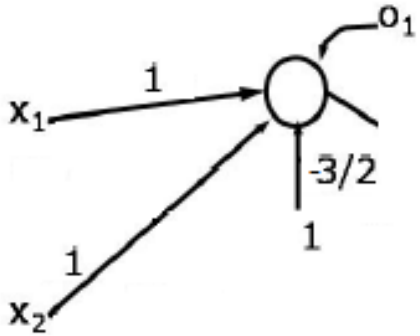
Example: XOR problem





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

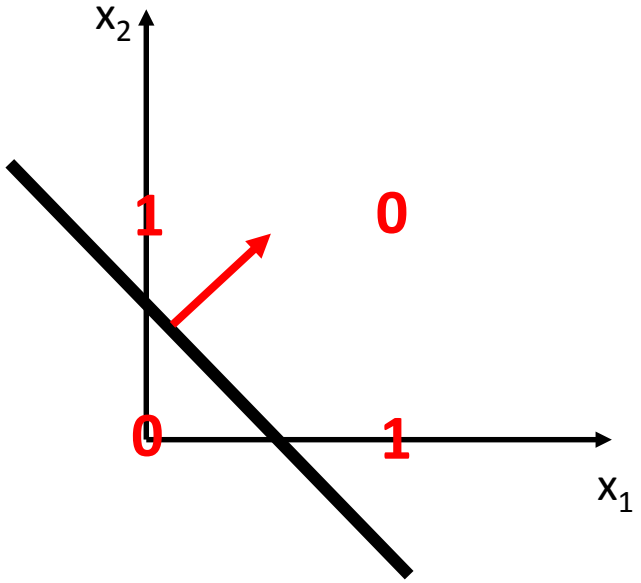
x_1	x_2	o_1
0	0	0
0	1	0
1	0	0
1	1	1



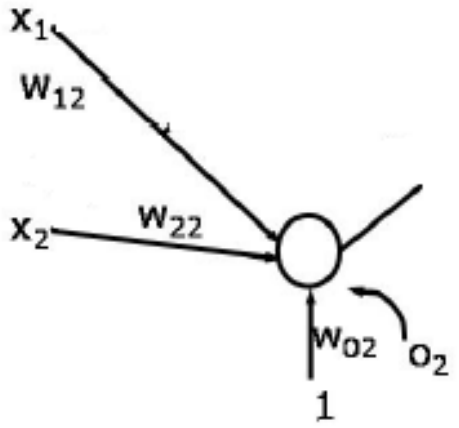
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

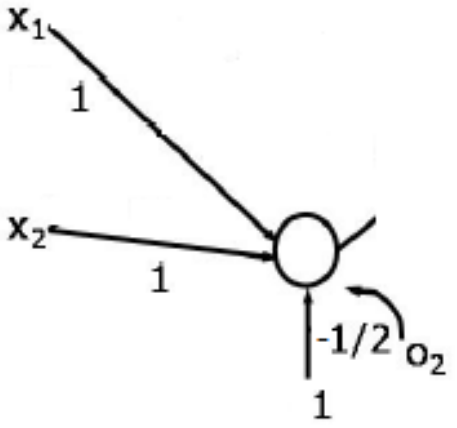
Example: XOR problem



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



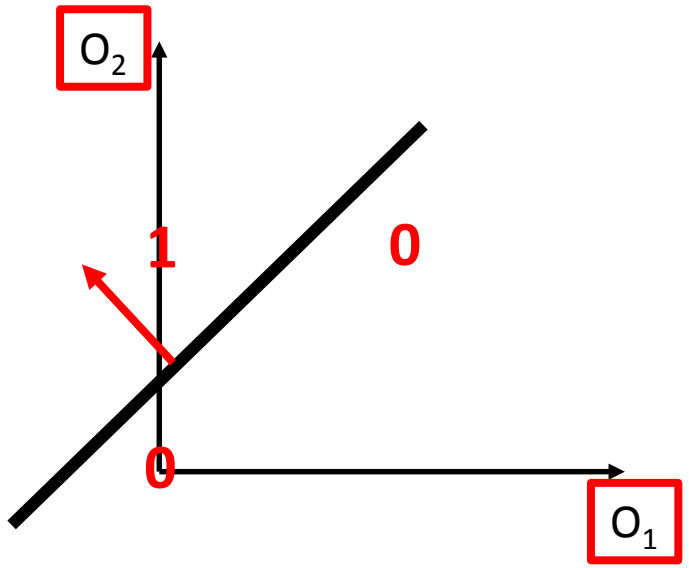
$w_{02} = -1/2 \quad w_{12} = w_{22} = 1$



UNIVERSITÉ CÔTE D'AZUR

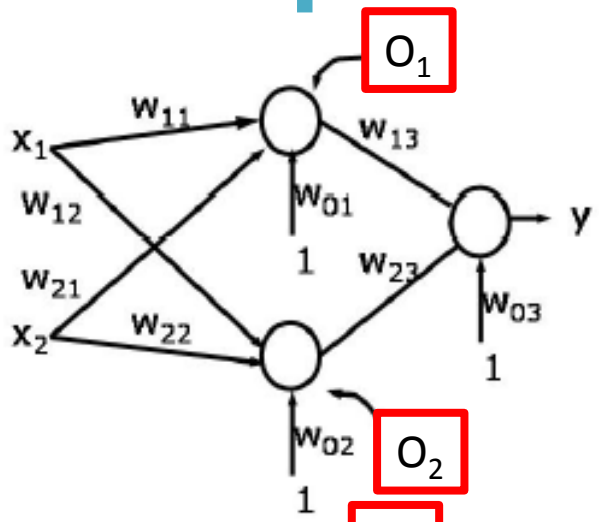
univ-cotedazur.fr

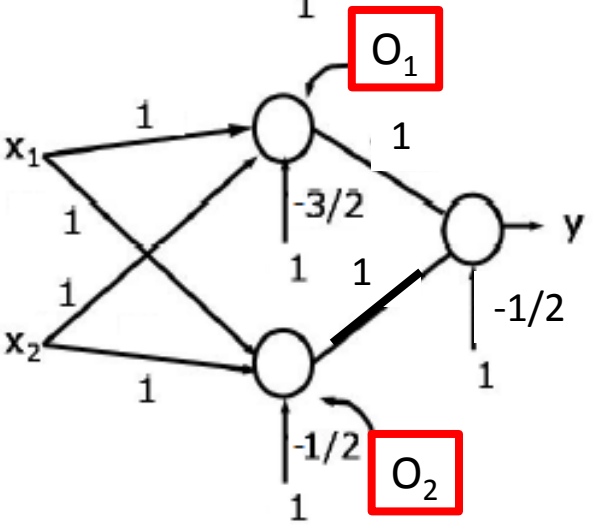
Example: XOR problem



$O_2-O_1-1/2=0$

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





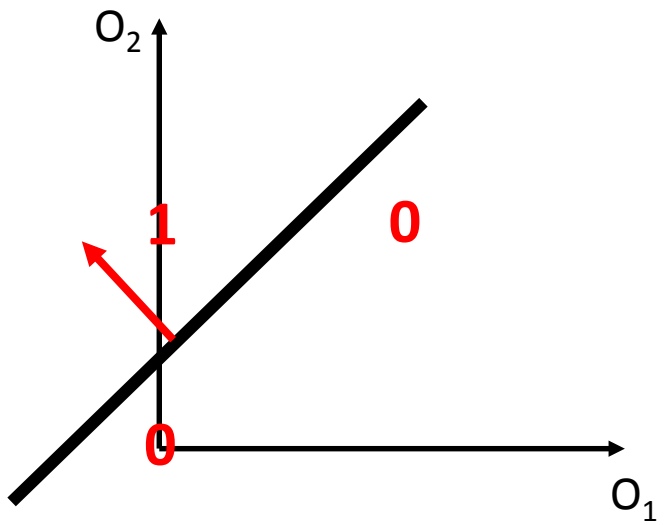
$w_{23}o_2+w_{13}o_1+w_{03}=0$

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

UNIVERSITÉ CÔTE D'AZUR

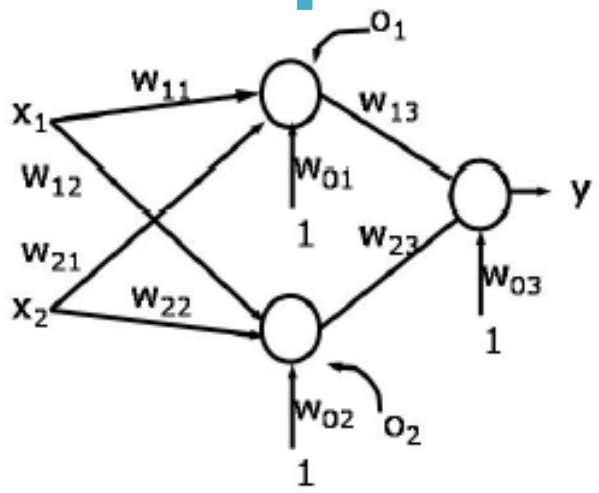
univ-cotedazur.fr

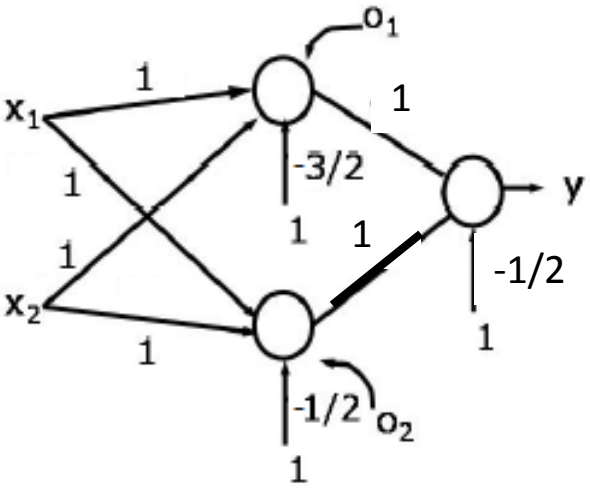
Example: XOR problem



$O_2-O_1-1/2=0$

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

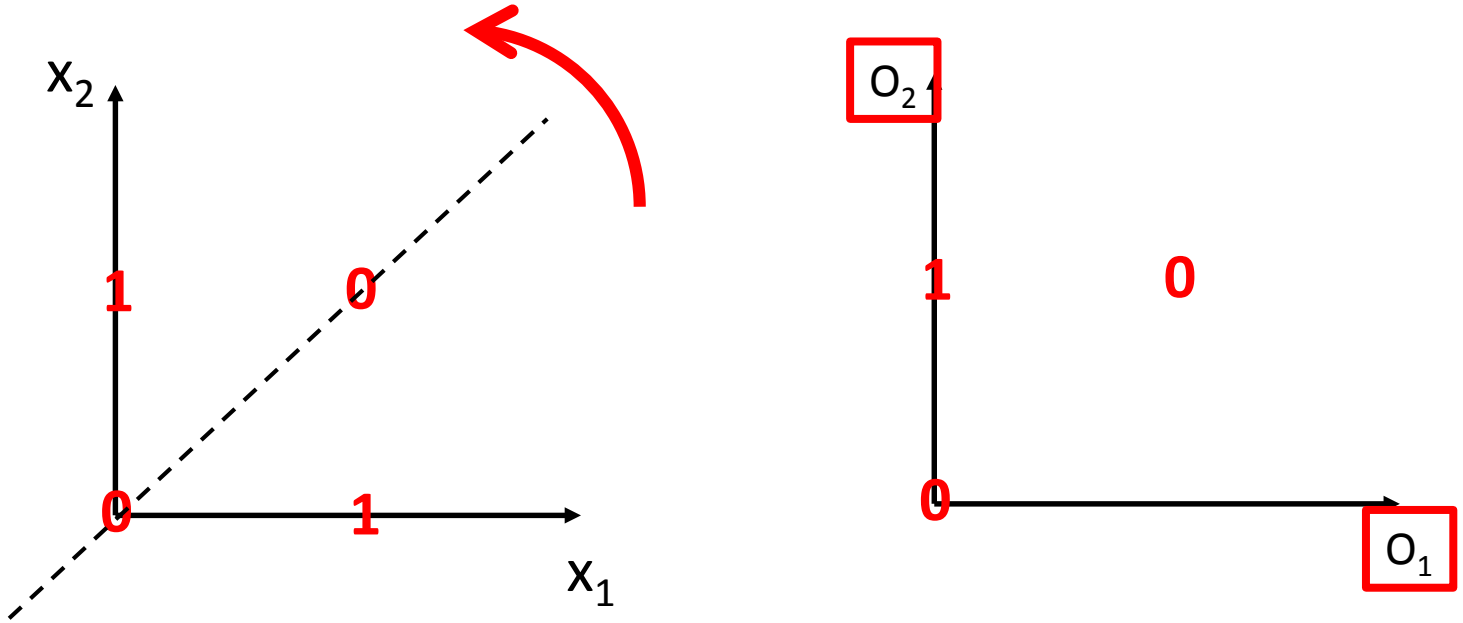




$w_{23}o_2+w_{13}o_1+w_{03}=0$

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

Example: XOR problem

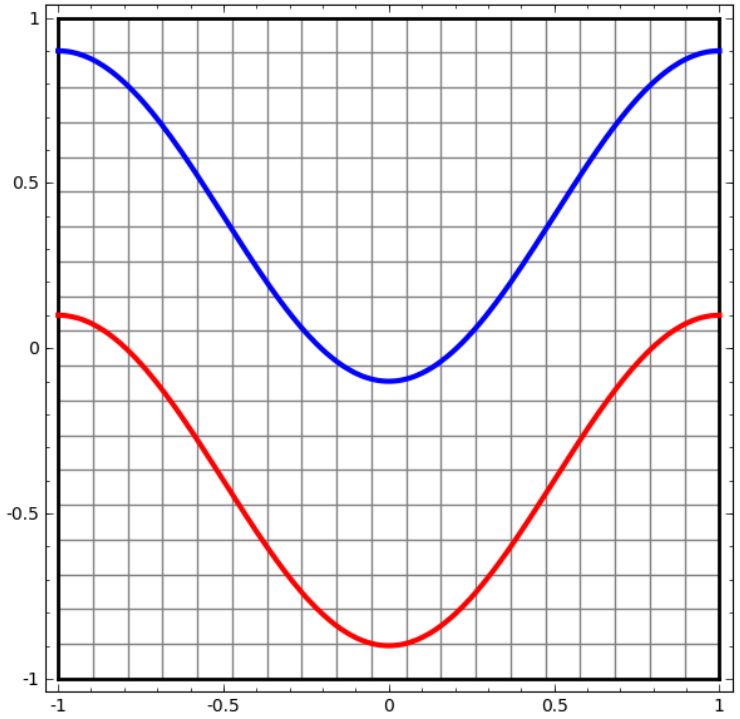


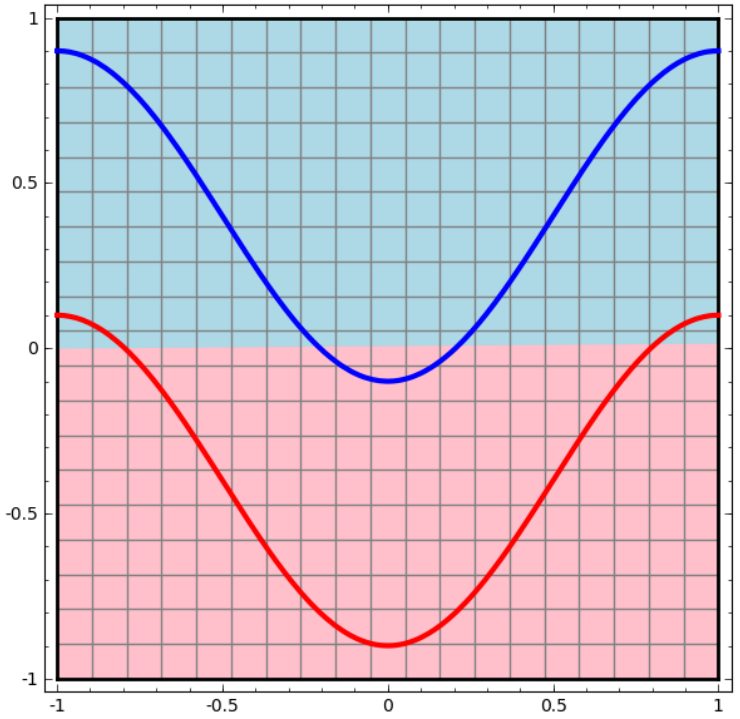
Adding a hidden layer has folded the input space

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

One perceptron



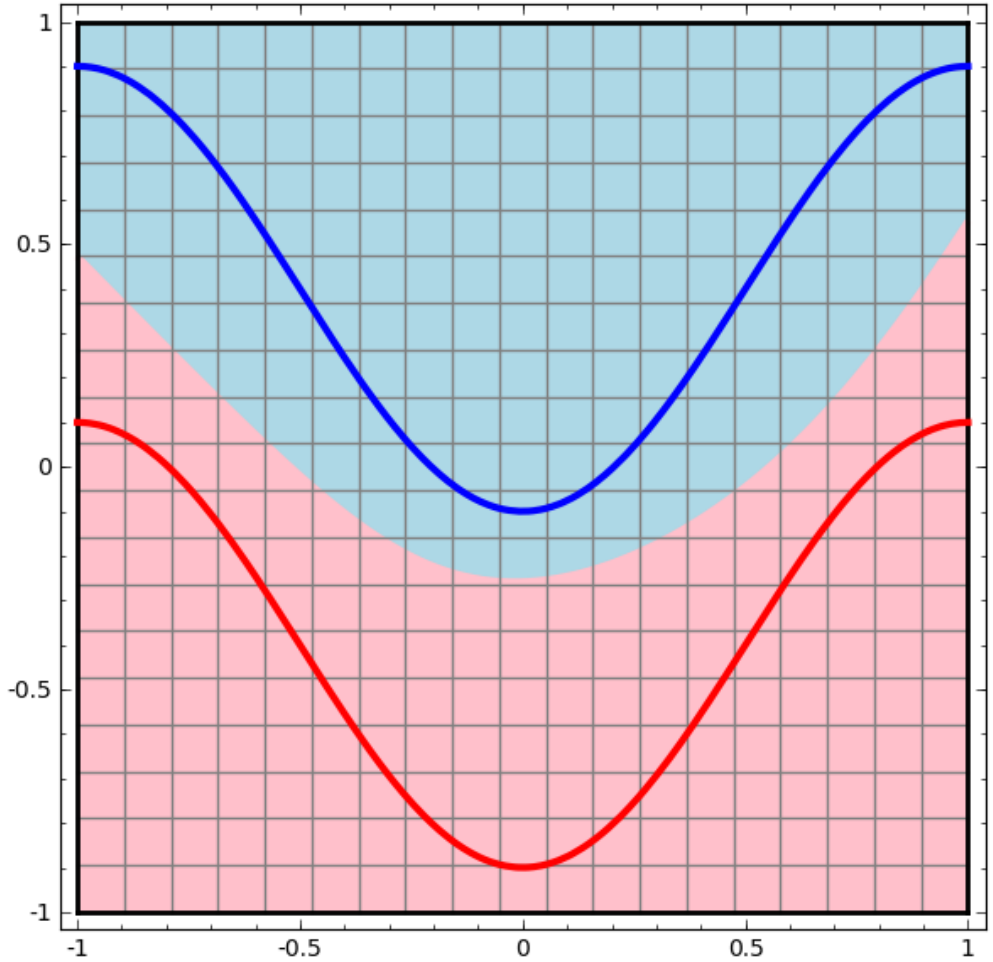


Illustrations from: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Multi-Layer Perceptron, manifold disentanglement

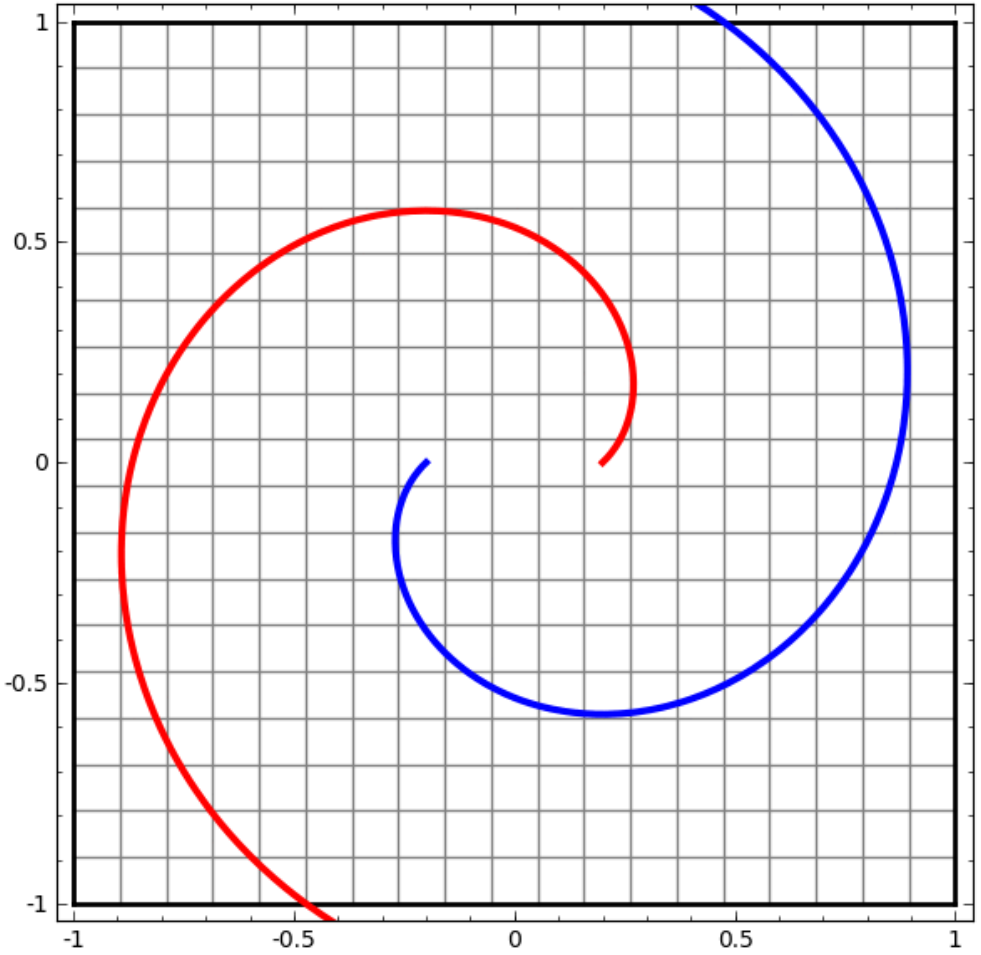


Illustrations from: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Multi-Layer Perceptron, manifold disentanglement

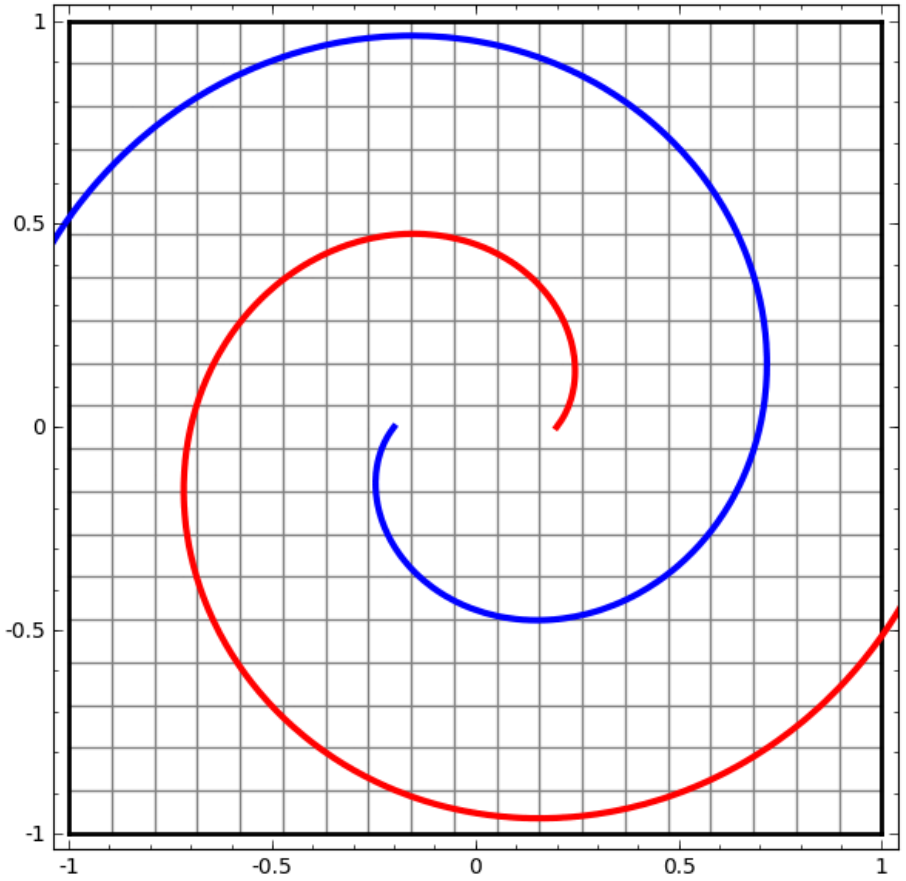


Illustrations from: http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

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

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:**

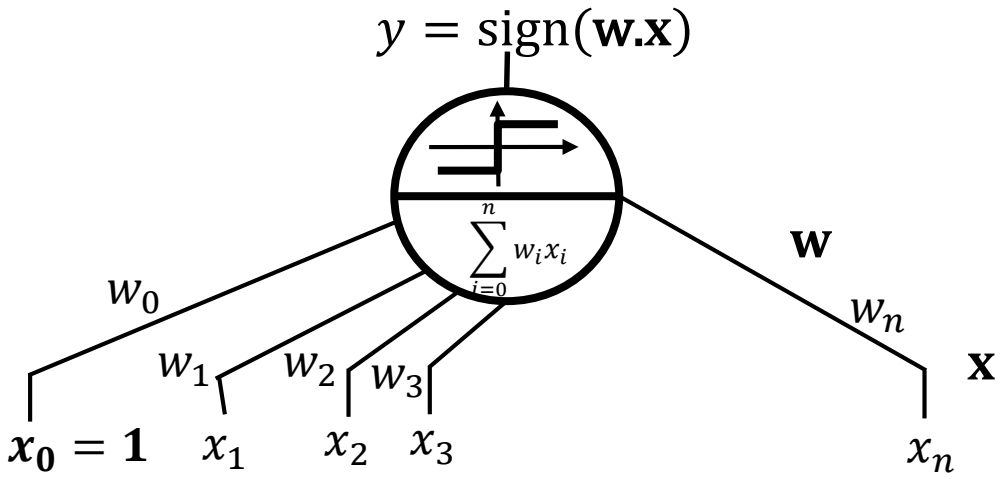
  "…networks with one internal layer and an arbitrary continuous sigmoidal function can approximate continuous functions with arbitrary precision providing that no constraints are placed on the number of nodes or the size of the weights"
- **But no efficient learning rule is known** and **the size of the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

UNIVERSITÉ CÔTE D'AZUR

Multi-Layer Perceptron with smooth activation

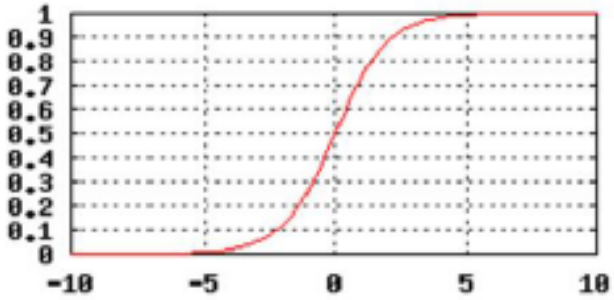
Soft Threshold

- 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.



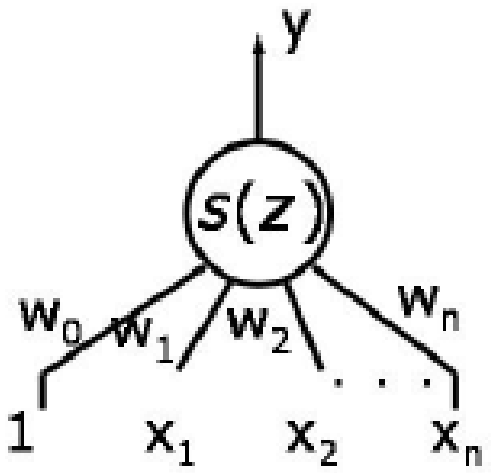
Sigmoid Unit

- 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.



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

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



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.

Training

- $y(\mathbf{x},\mathbf{w})$ corresponds to the predicted output of the network of weights **w** (a vector), with input data **x** (a vector).
- y^m is the **known** output for the training data $\mathbf{x}^m$.
- Error over the training set for a given weight vector:

$$E = \frac{1}{2} \sum_{on\ the\ training\ samples} (y(\mathbf{x},\mathbf{w}) - y^m)^2$$

- Our goal is to find the weight vector that minimizes the 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

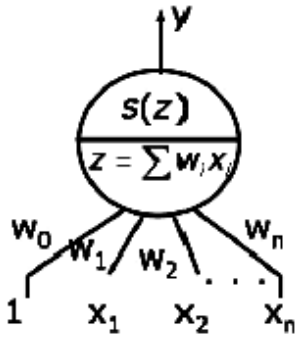
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} E \quad \Leftrightarrow \quad \mathbf{w} \leftarrow \mathbf{w} - \eta (y(\mathbf{x}^m, \mathbf{w}) - y^m)\left[\frac{\partial y}{\partial w_1}, \ldots, \frac{\partial y}{\partial w_n}\right]$$

Gradient Descent – Single Unit

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

$$z = \sum_{i=0}^{n} w_i x_i^m \qquad y = s(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\partial y}{\partial w_i} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i} = \frac{\partial s(z)}{\partial z}\frac{\partial z}{\partial w_i} = \frac{\partial s(z)}{\partial z} x_i^m$$

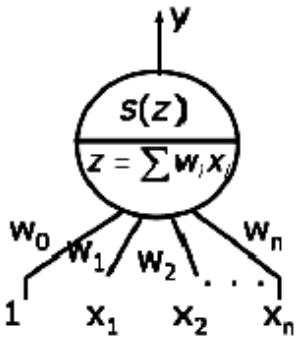


Substituting in the equation of previous slide we get (for the arbitrary *i* th element of **w**):

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

$$w_i \leftarrow w_i - \eta(y - y^m) \frac{\partial s(z)}{\partial z} x_i^m$$

Derivative of the sigmoid



$$s(z) = \frac{1}{1+e^{-z}}$$

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

$$= [-(1+e^{-z})^{-2}][-e^{-z}]$$

$$= \left[\frac{1}{1+e^{-z}}\right]\left[\frac{e^{-z}}{1+e^{-z}}\right]$$

$$= s(z)(1-s(z))$$

$$= y(1-y)$$

Substituting in the equation of previous slide we get (for the arbitrary *i* th element of **w**):

$$\Delta w_i = -\eta(y-y^m)\frac{\partial s(z)}{\partial z}x_i^m$$

$$= -\eta(y-y^m)y(1-y)\,x_i^m$$

$$= -\eta\, y(1-y)(y-y^m)\,x_i^m$$

$$\Delta w_i = -\eta\delta\, x_i^m$$

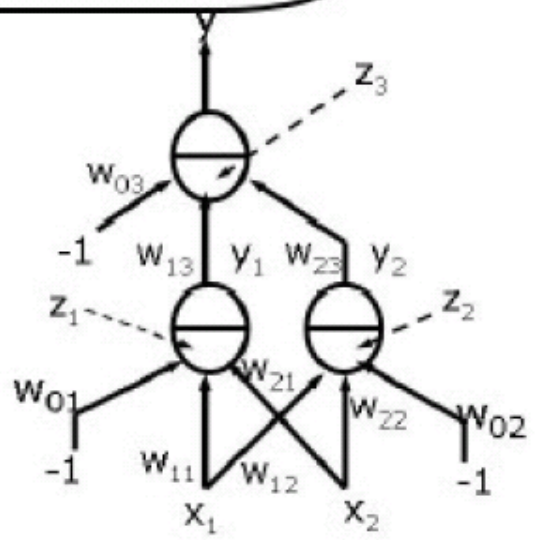
Delta rule

$$w_i \leftarrow w_i + \Delta w_i$$

Gradient of Error

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

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

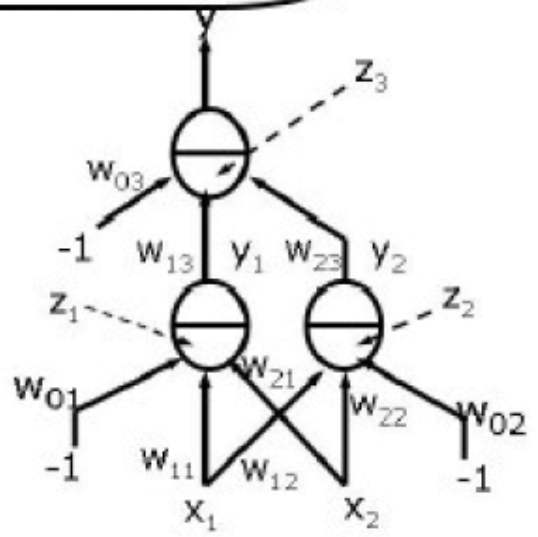


Gradient of Error

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

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

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



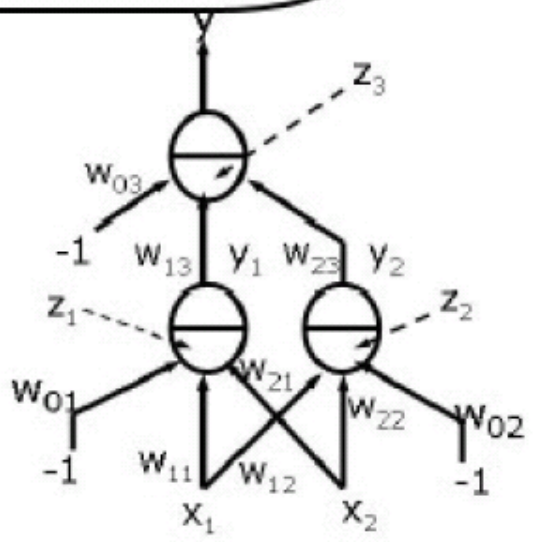
Gradient of Error

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

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

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

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

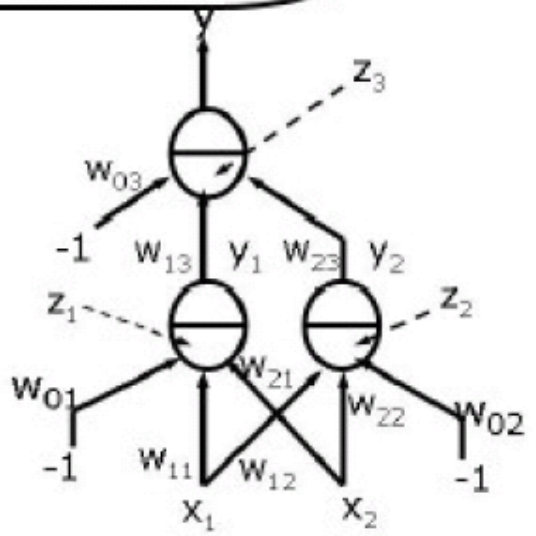


Gradient of Error

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

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

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



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

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

Generalized Delta Rule

In general, for a hidden unit j we have

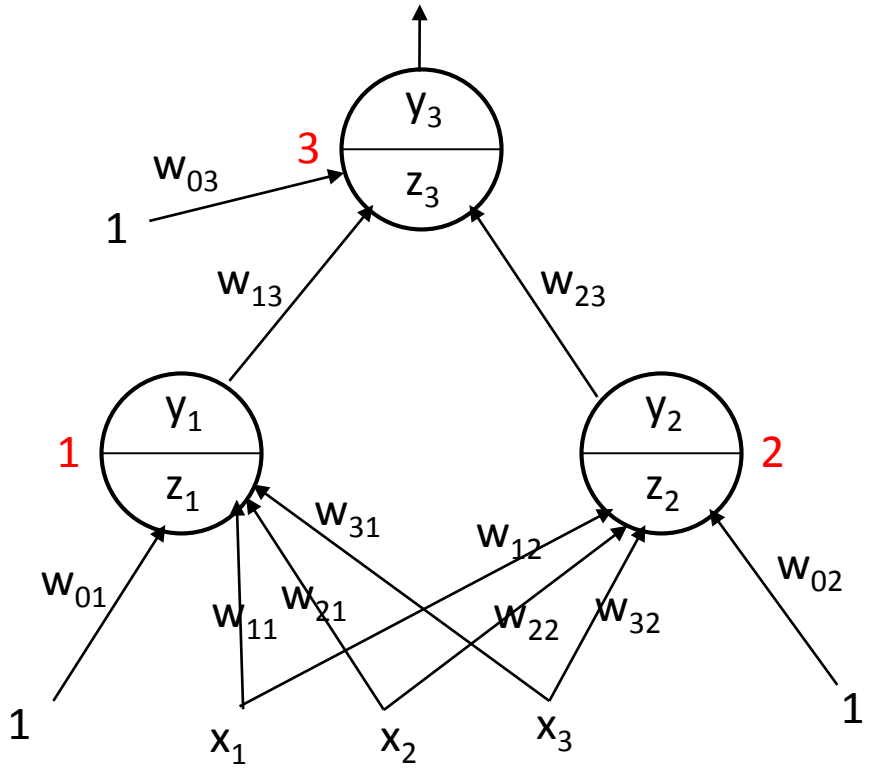
$$\delta_j = y_j(1-y_j) \sum_{k \in downstream(j)} \delta_k w_{jk}$$ **Backprop rule**

$$w_{ij} \leftarrow w_{ij} - \eta \delta_j y_i$$ **Descent rule**

Learning Weights

For an output unit *p* we have:

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

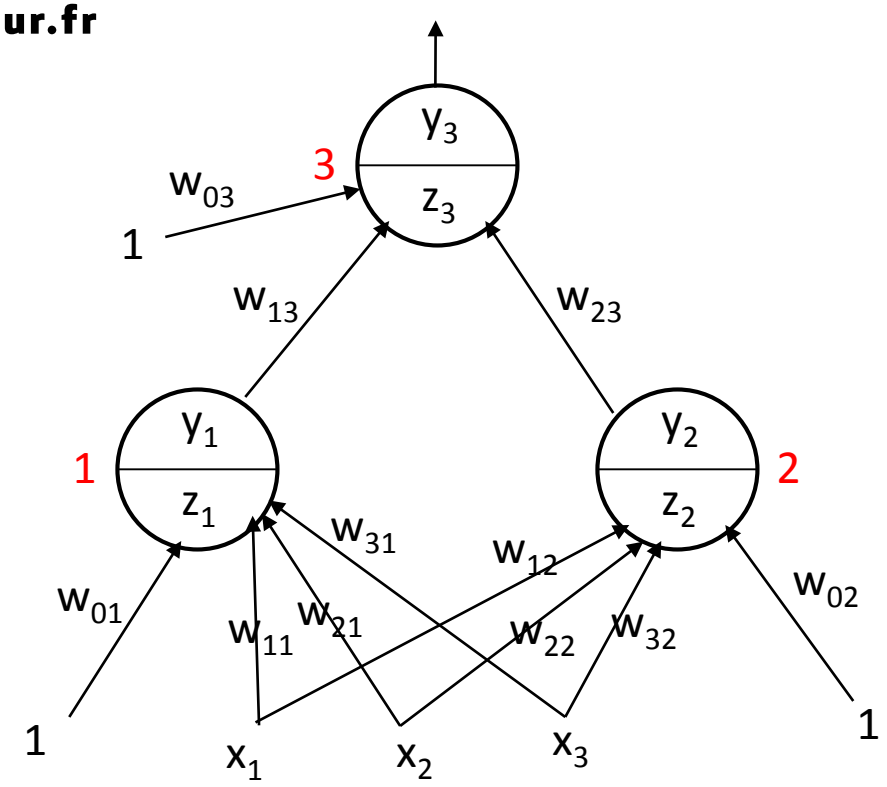


p=3 in this example

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Backpropagation



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

$$\delta_3 = y_3(1-y_3)(y_3 - y^m)$$
$$\delta_2 = y_2(1-y_2)\delta_3 w_{23}$$
$$\delta_1 = y_1(1-y_1)\delta_3 w_{13}$$

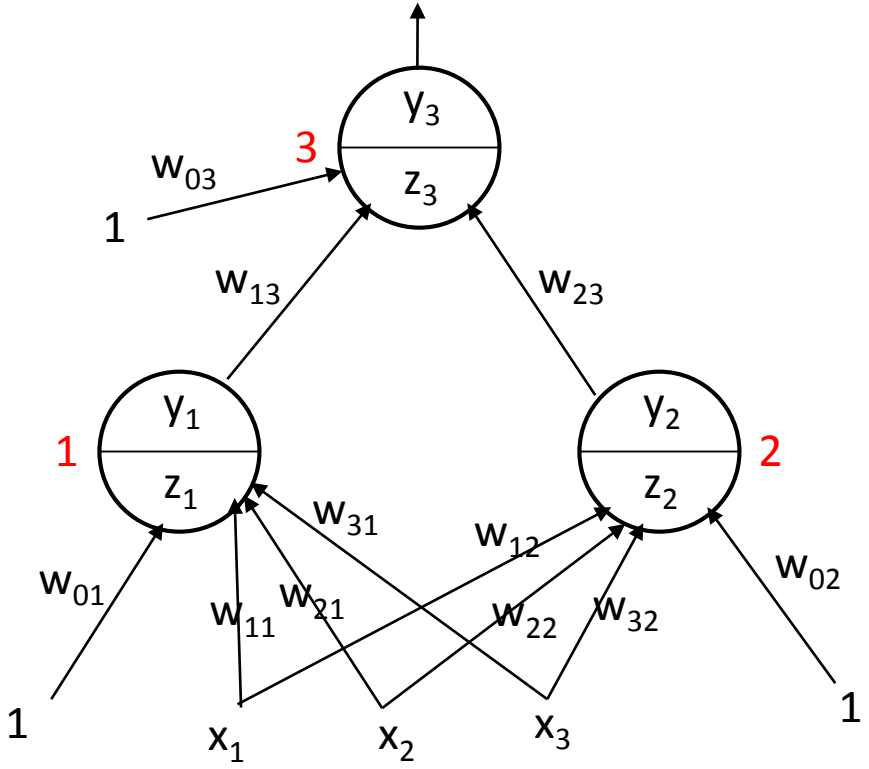
$$w_{03} = w_{03} - \eta\delta_3(1)$$
$$w_{13} = w_{13} - \eta\delta_3 y_1$$
$$w_{23} = w_{23} - \eta\delta_3 y_2$$

$$w_{02} = w_{02} - \eta\delta_2(1) \qquad w_{01} = w_{01} - \eta\delta_1(1)$$
$$w_{12} = w_{12} - \eta\delta_2 x_1 \qquad w_{11} = w_{11} - \eta\delta_1 x_1$$
$$w_{22} = w_{22} - \eta\delta_2 x_2 \qquad w_{21} = w_{21} - \eta\delta_1 x_2$$
$$w_{32} = w_{32} - \eta\delta_2 x_3 \qquad w_{31} = w_{31} - \eta\delta_1 x_3$$

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

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.

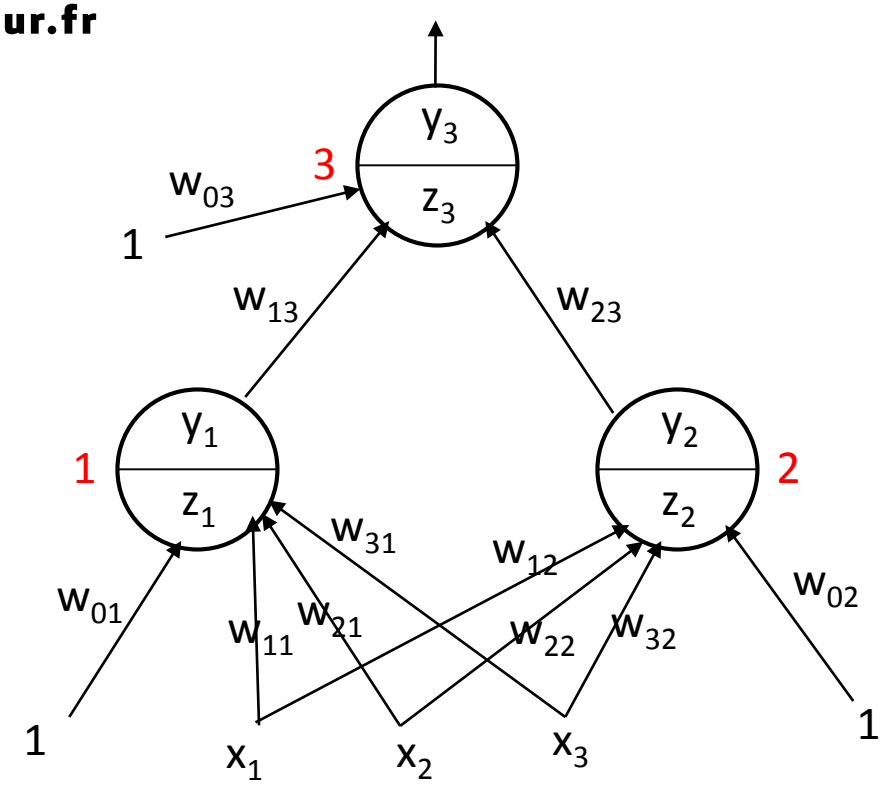
Also, in the table is given one training instance (first column).

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

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Feed-Forward Example



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

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

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

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

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:

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

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

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

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

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:

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

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

UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

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)

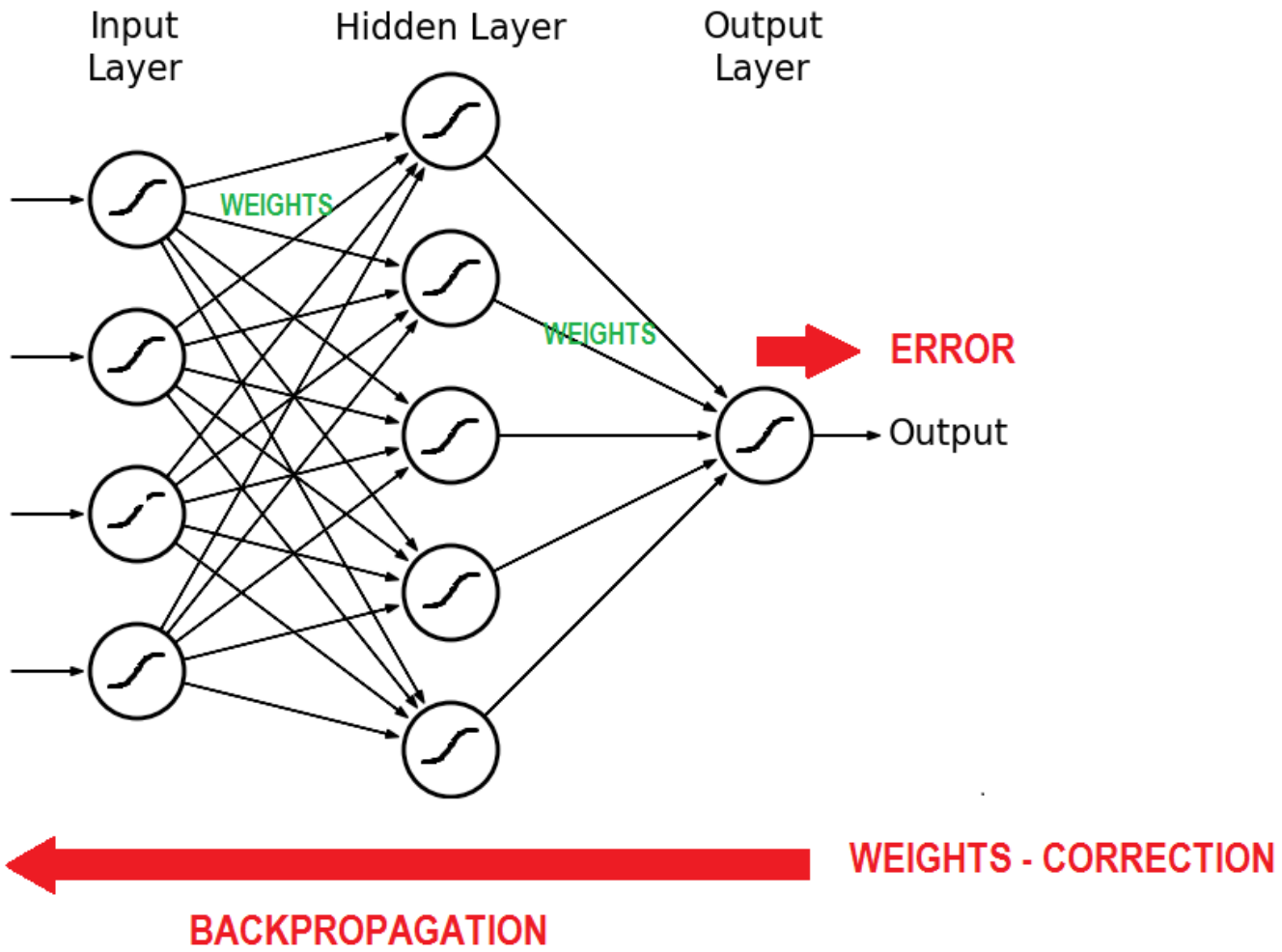
- Note that each expression involves data local to a particular unit, we do not have to look around summing things over the whole network.

- 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



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.

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

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. 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.

UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

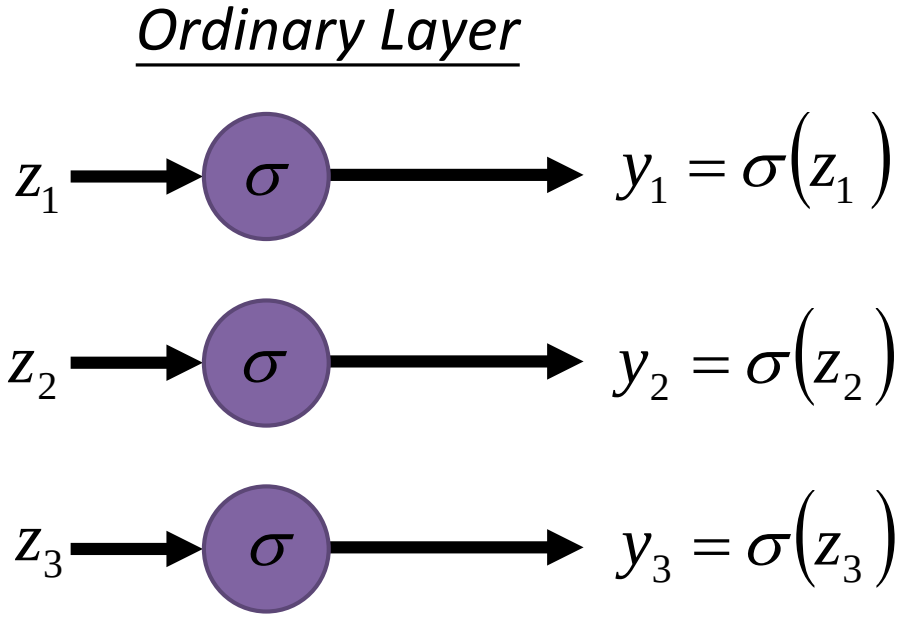
Multiple Output Nodes

- 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.

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

Softmax…

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



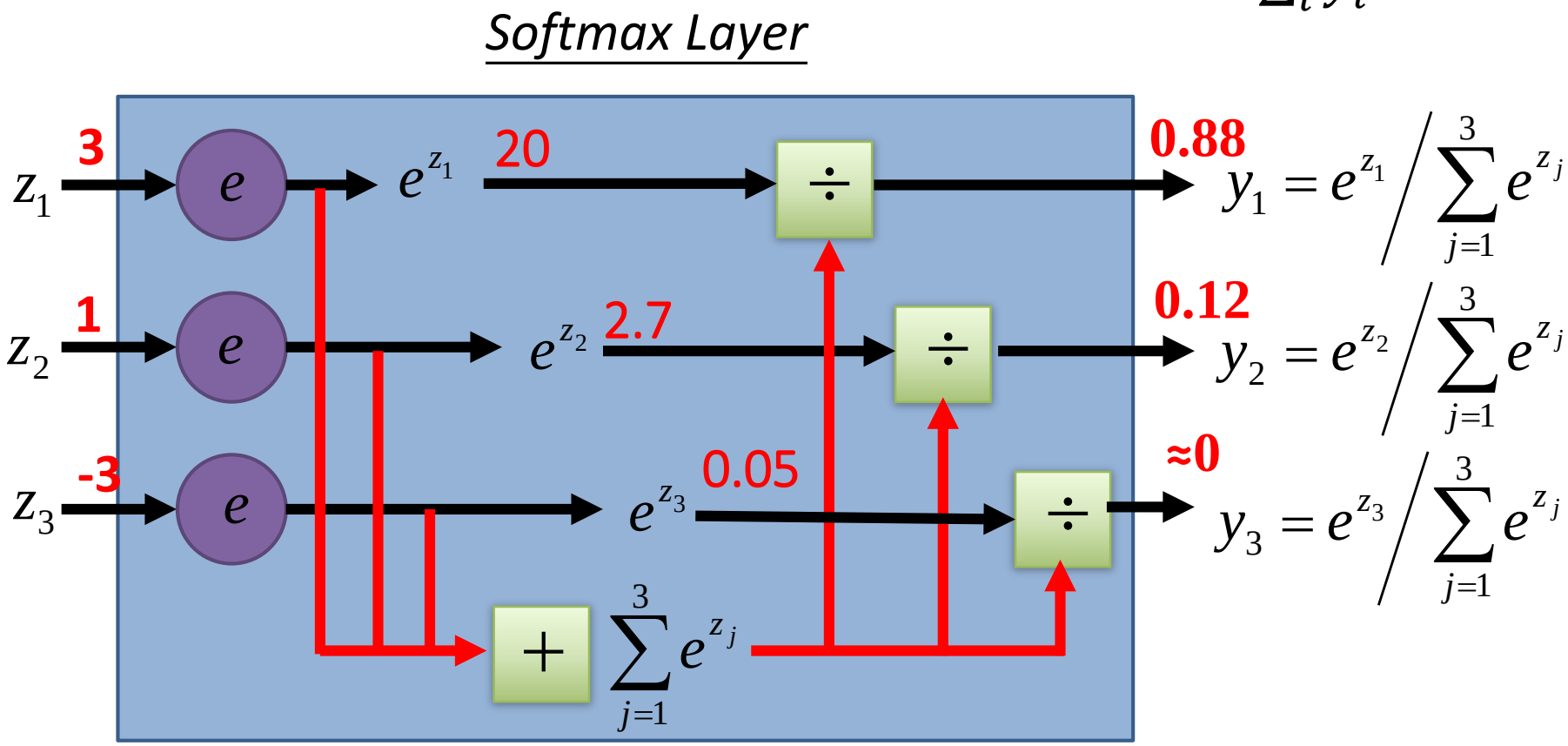
In general, the output of network can be any value.

May not be easy to interpret

Softmax!

- **Softmax layer as the output layer**

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



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.

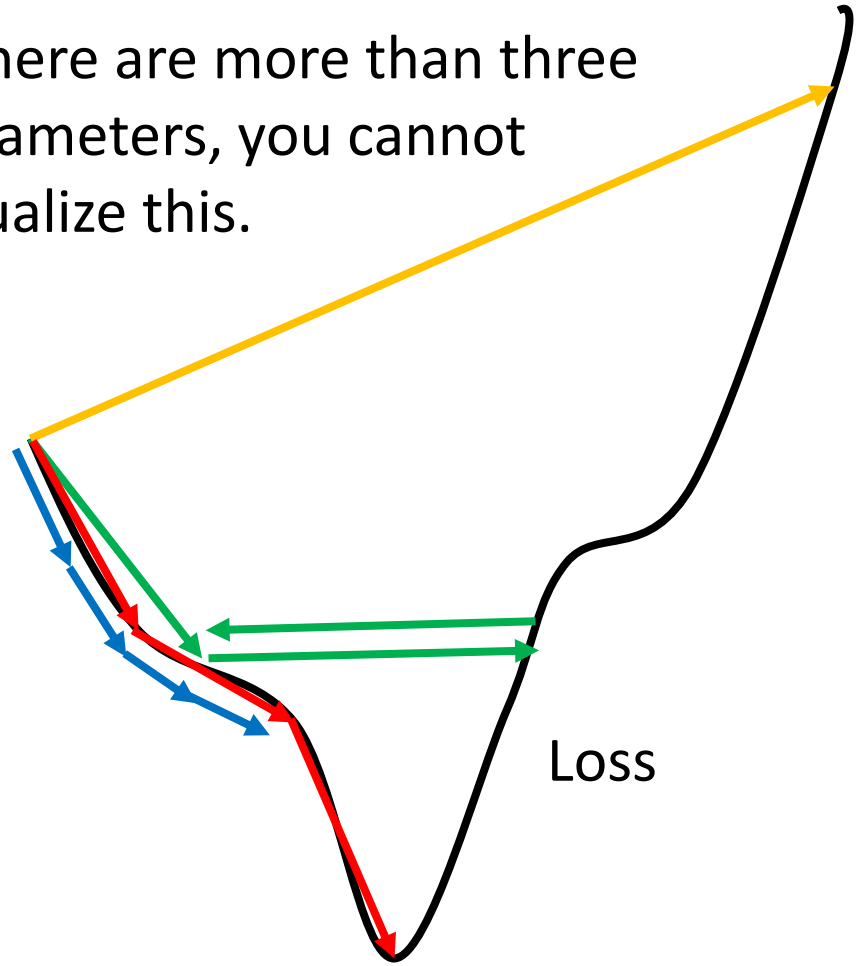
1. Split data set (randomly) into three subsets:
   - Training set – used for picking weights
   - Validation set – used to stop training
   - Test set – used to evaluate performance
2. Pick random, small weights as initial values
3. Perform iterative minimization of error over training set.
4. Stop 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.

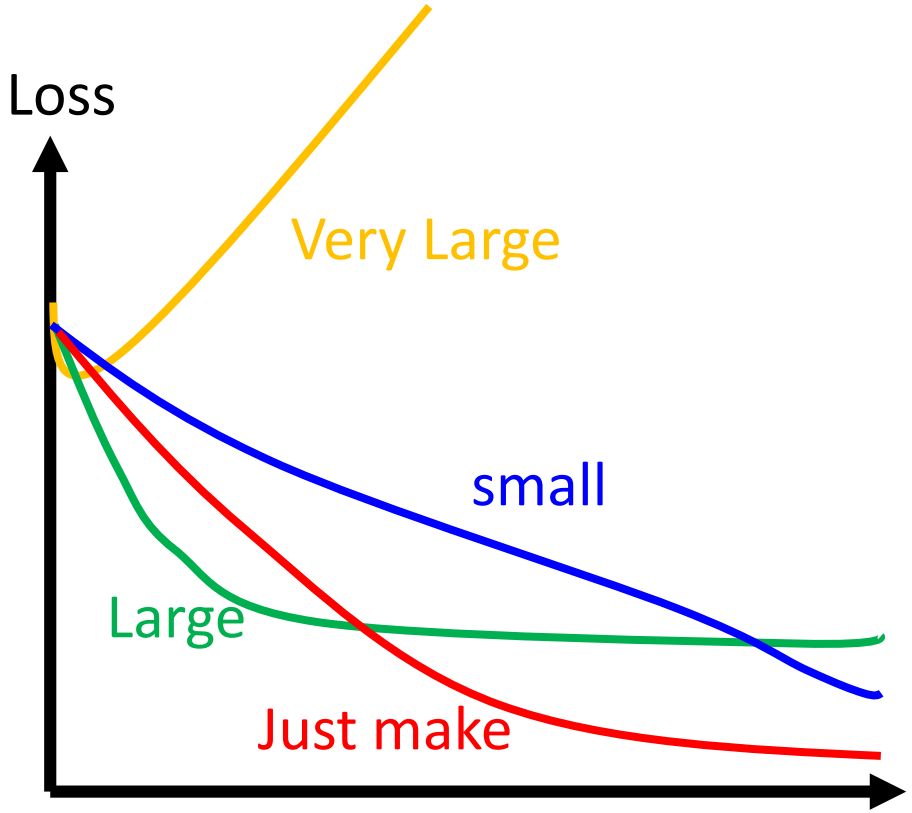
Can use cross-validation if data set is too small to divide into three subsets.

Learning Rate

$$\mathbf{W}^{i} = \mathbf{W}^{i-1} - \eta \nabla L\left(\mathbf{W}^{i-1}\right)$$ Set the learning rate η carefully

If there are more than three parameters, you cannot visualize this.





But you can always visualize this.

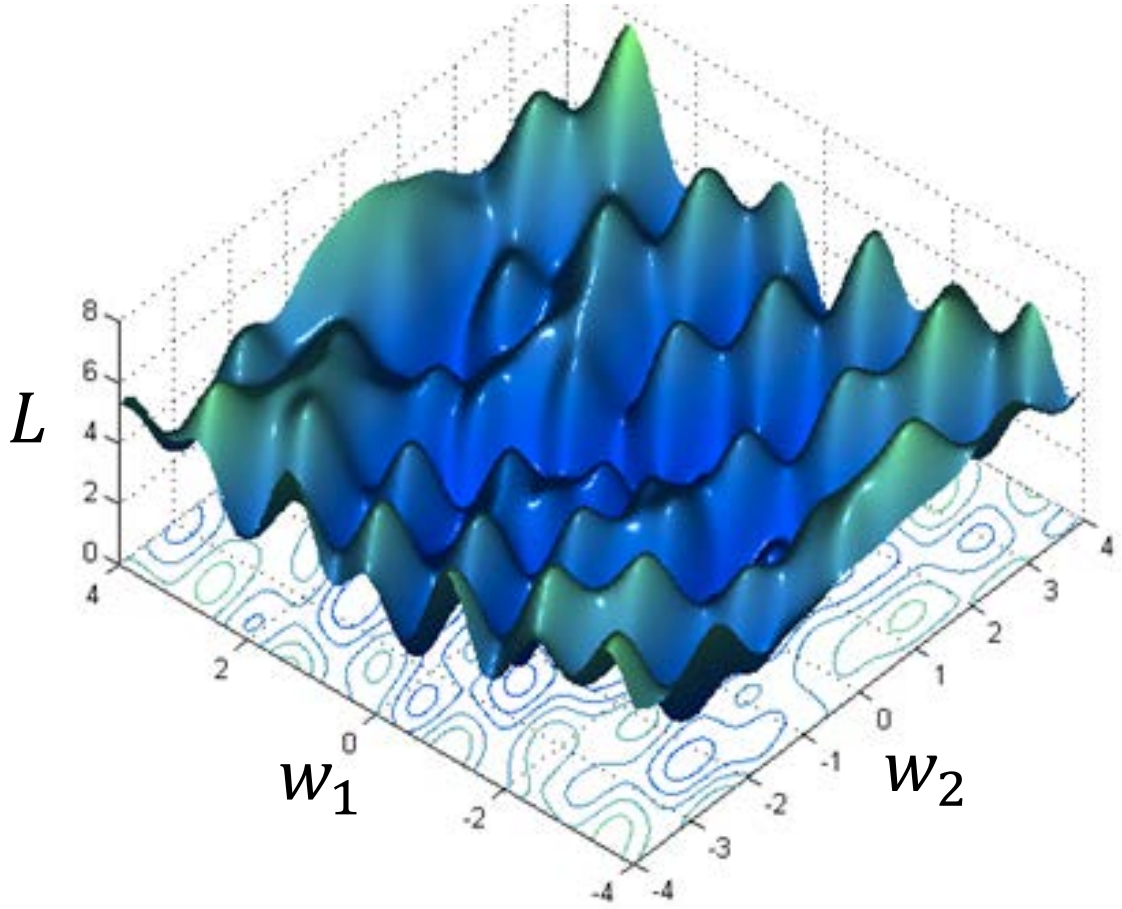
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Momentum

Problem of local minima

- Gradient descent never guarantee global minima



Different initial point W^0

Reach different minima, so different results

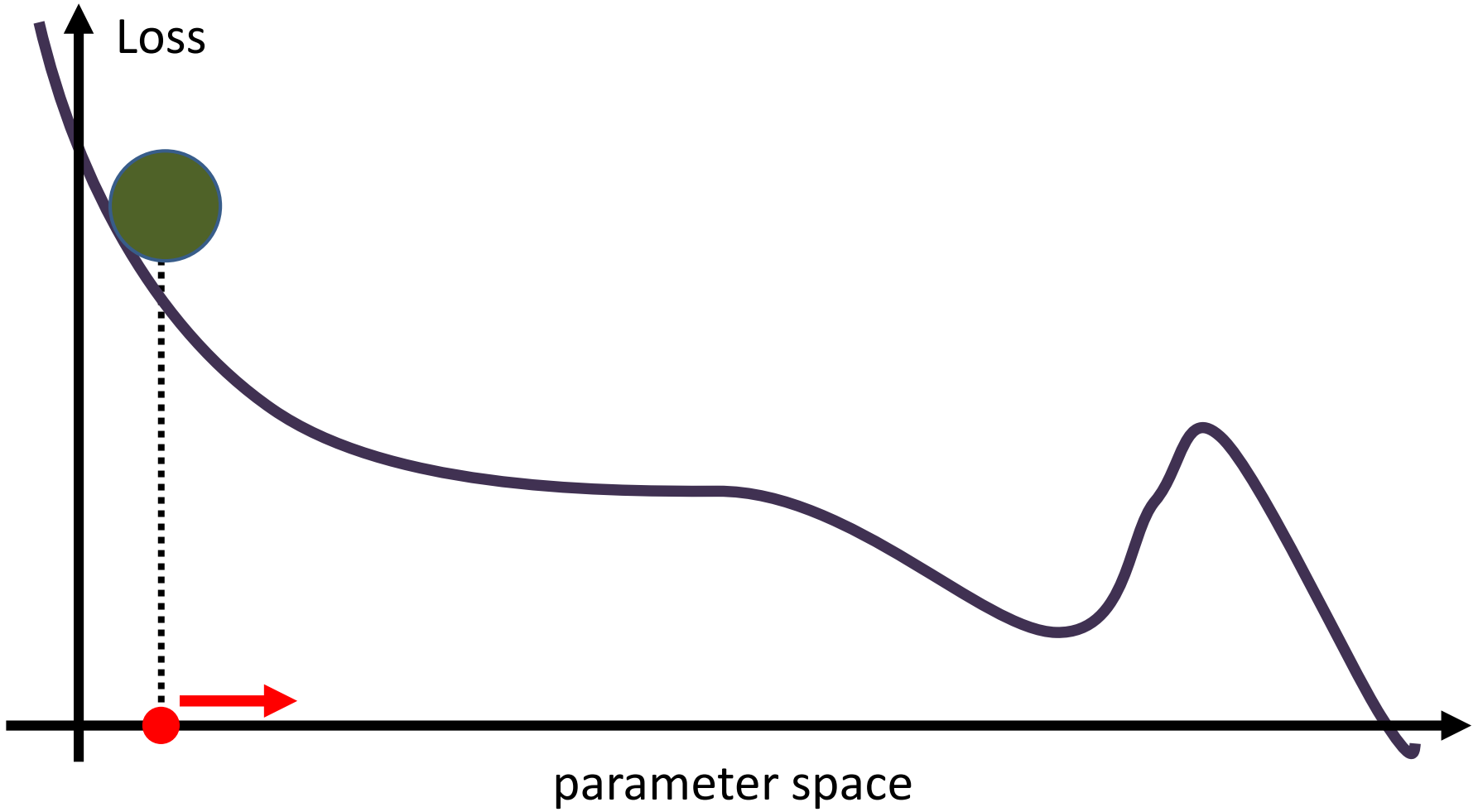
Who is Afraid of Non-Convex Loss Functions?
http://videolectures.net/eml07_lecun_wia/

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

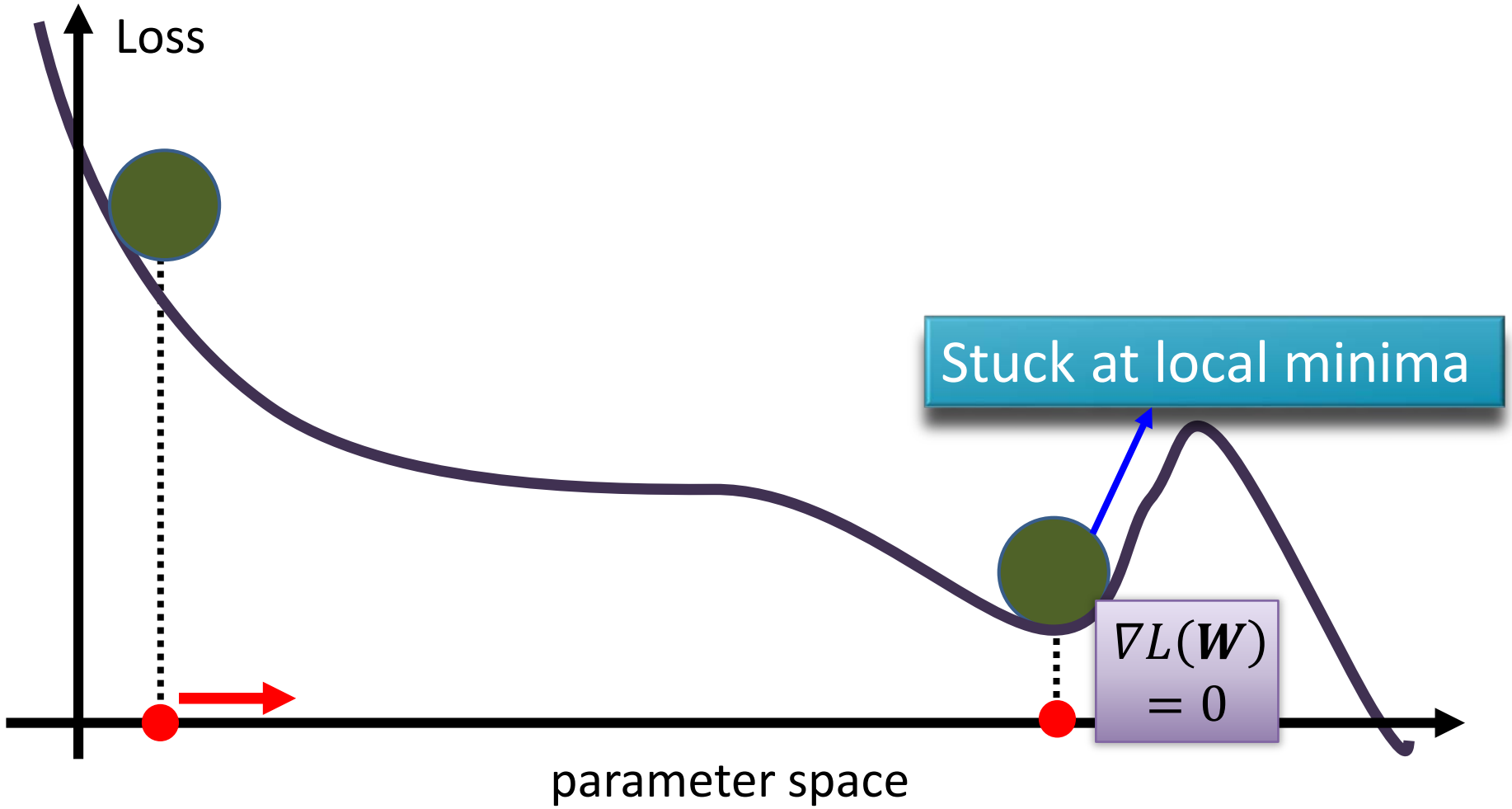
Momentum

Besides local minima…



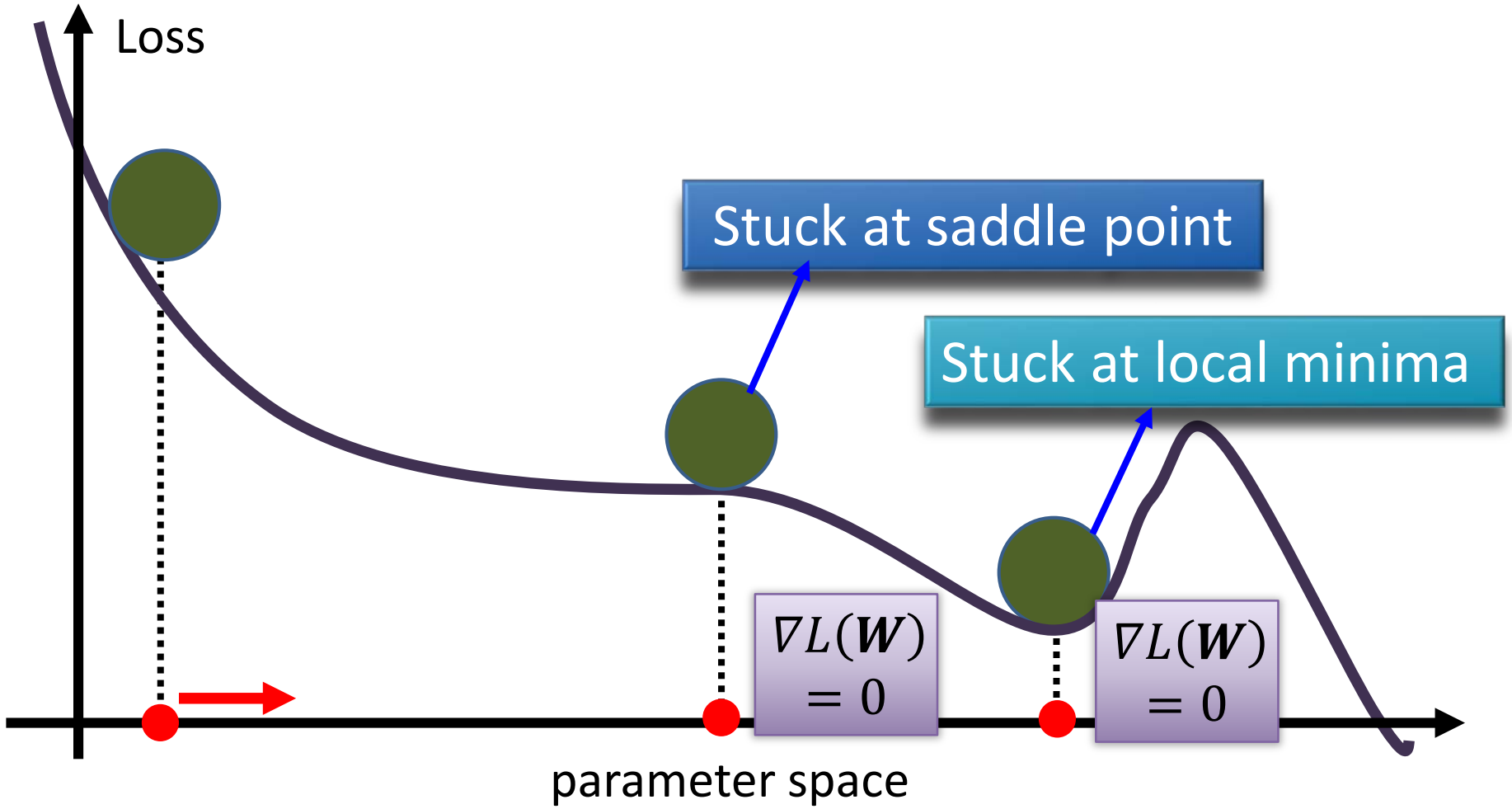
Momentum

Besides local minima…



Momentum

Besides local minima…

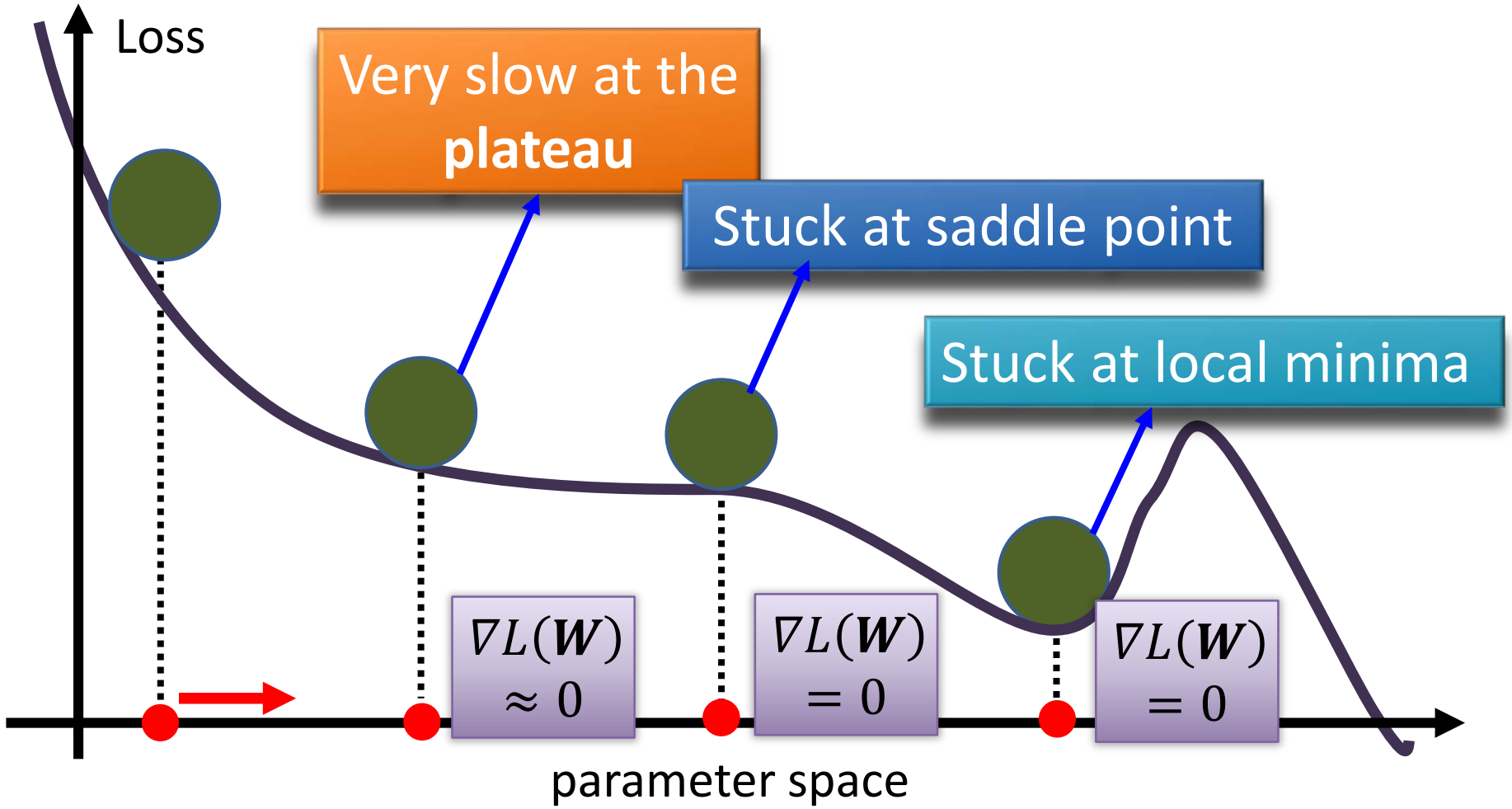


UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Momentum

Besides local minima…

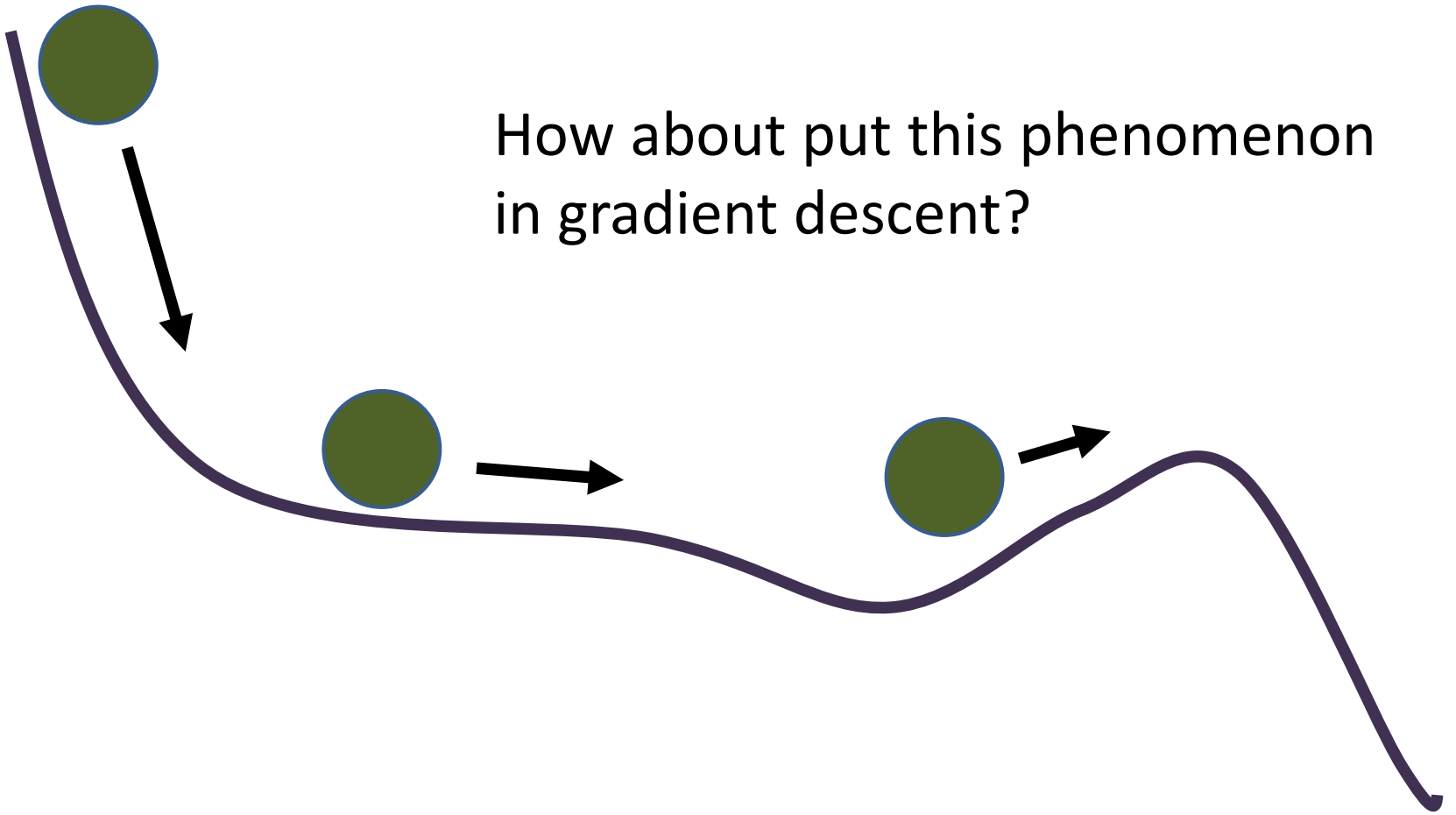


UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Momentum

- Momentum



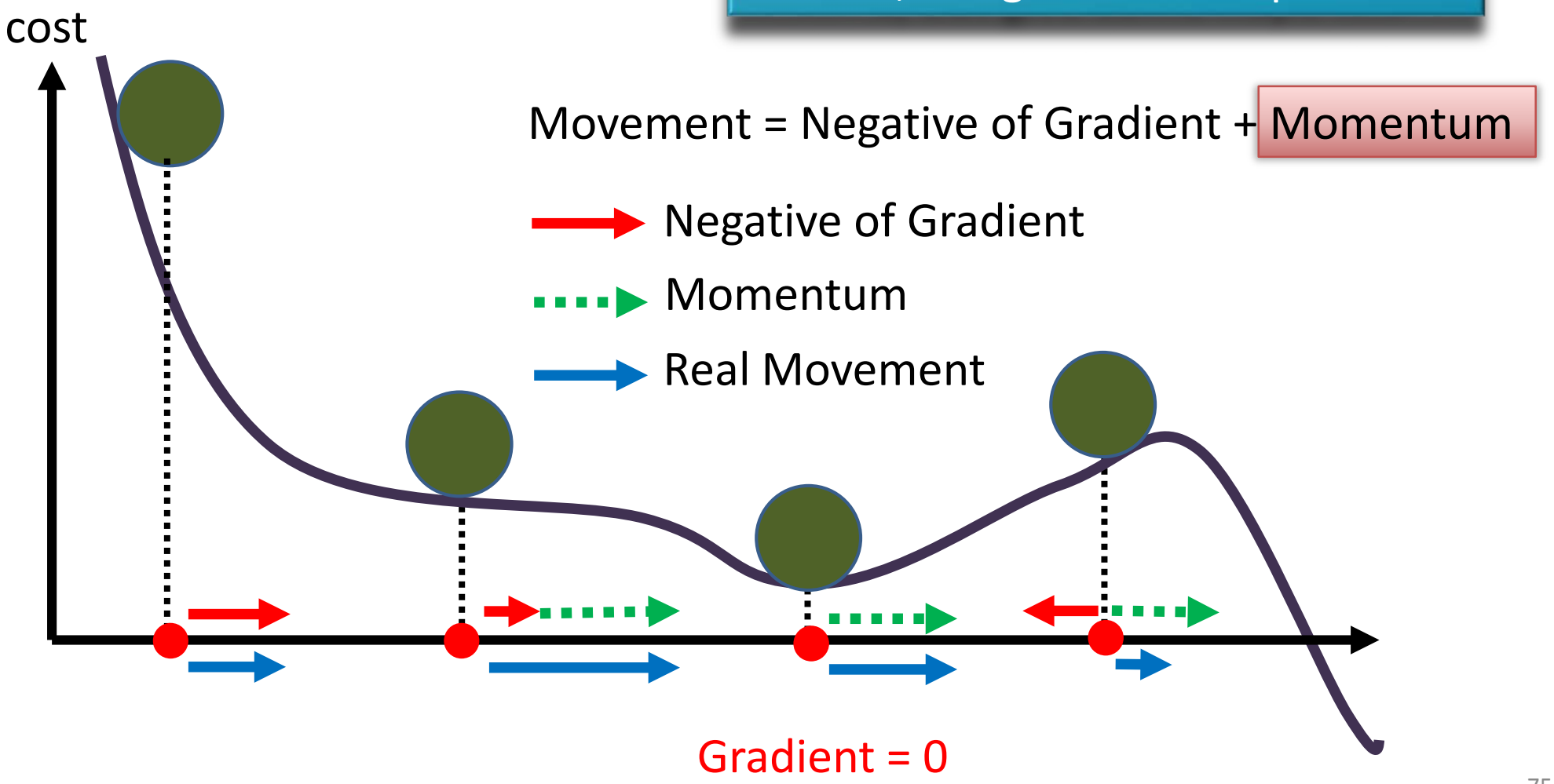
How about put this phenomenon in gradient descent?

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Momentum

Still not guarantee reaching global minima, but give some hope ……



UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

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).

Autonomous Land Vehicle In a Neural Network (ALVINN)

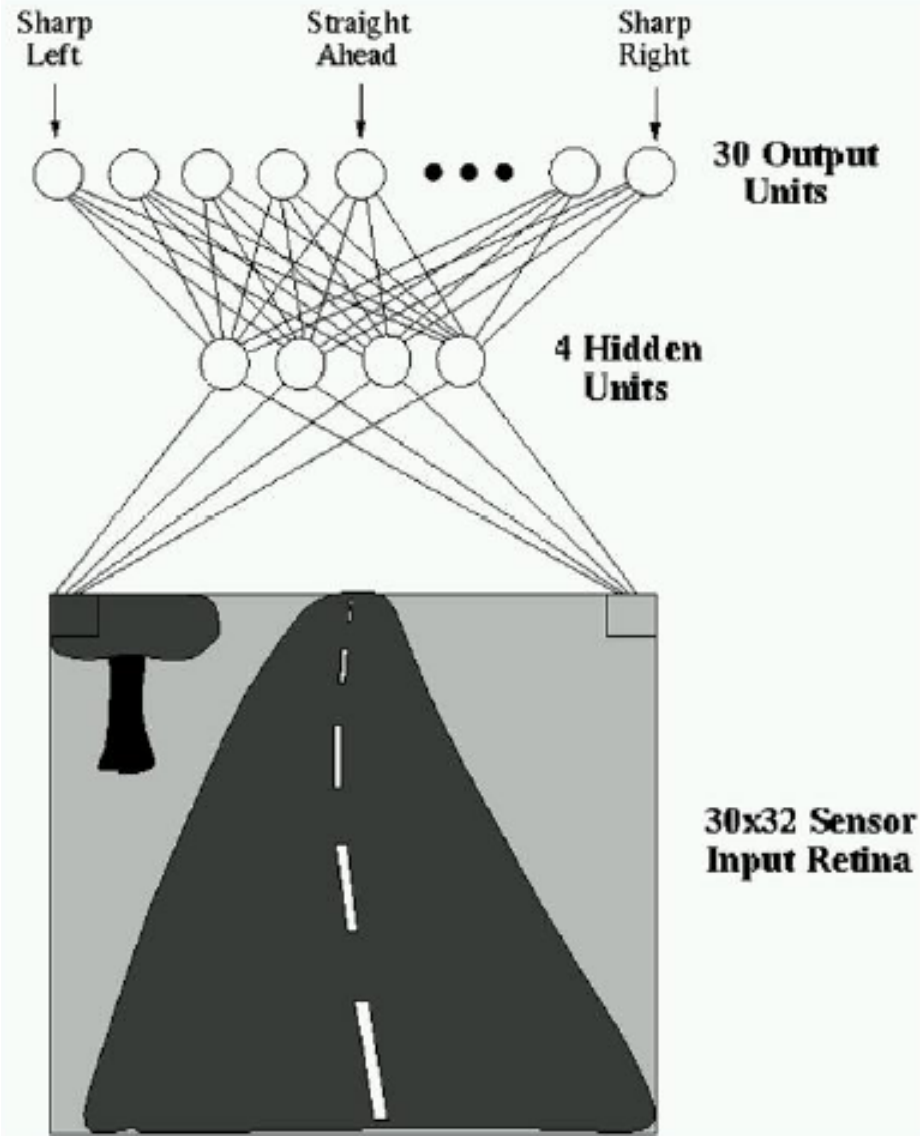
ALVINN is an automatic steering system for a car based on input from a camera mounted on the vehicle.

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).



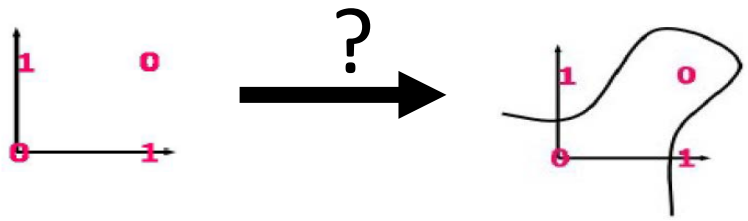
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

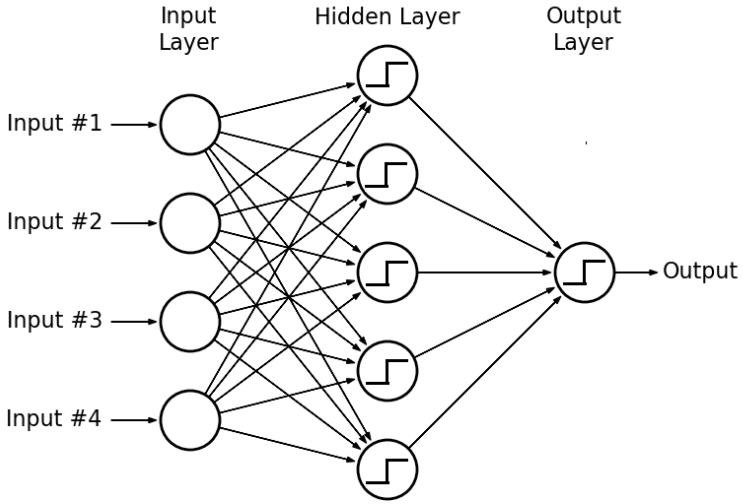
Artificial Neural Network in a Nutshell

Single Perceptron Unit

- **Perceptron** only learns linear function [Minsky and Papert, 1969]



- Non-linear function needs layer(s) of neurons → Neural Network
- Neural Network = input layer + hidden layer(s) + output layer

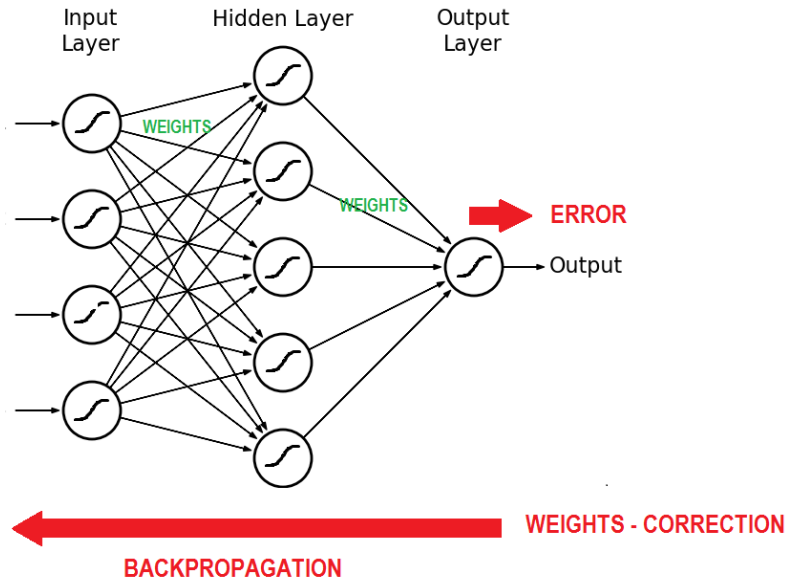


UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Multi-Layer Perceptron

- Training a neural network [Rumelhart et al. / Yann Le Cun et al. 1985]
- **Unknown parameters: weights on the synapses**
- Minimizing a cost function: some metric between the predicted output and the given output



- Step function: non-continuous functions are replaced by continuous non-linear ones

Neural Networks

- All the neurons of the Multi-Layer Perceptron are interconnected.
- A layer with all its neurons connected to neurons from other layers is called ***Fully Connected*** or ***Dense***.
- A **Multi-Layer Perceptron is** thus a network of only ***fully connected layers***, or ***dense layers***.
- There is **no (underlying) structure** in **Multi-Layer Perceptron**.

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Bibliography

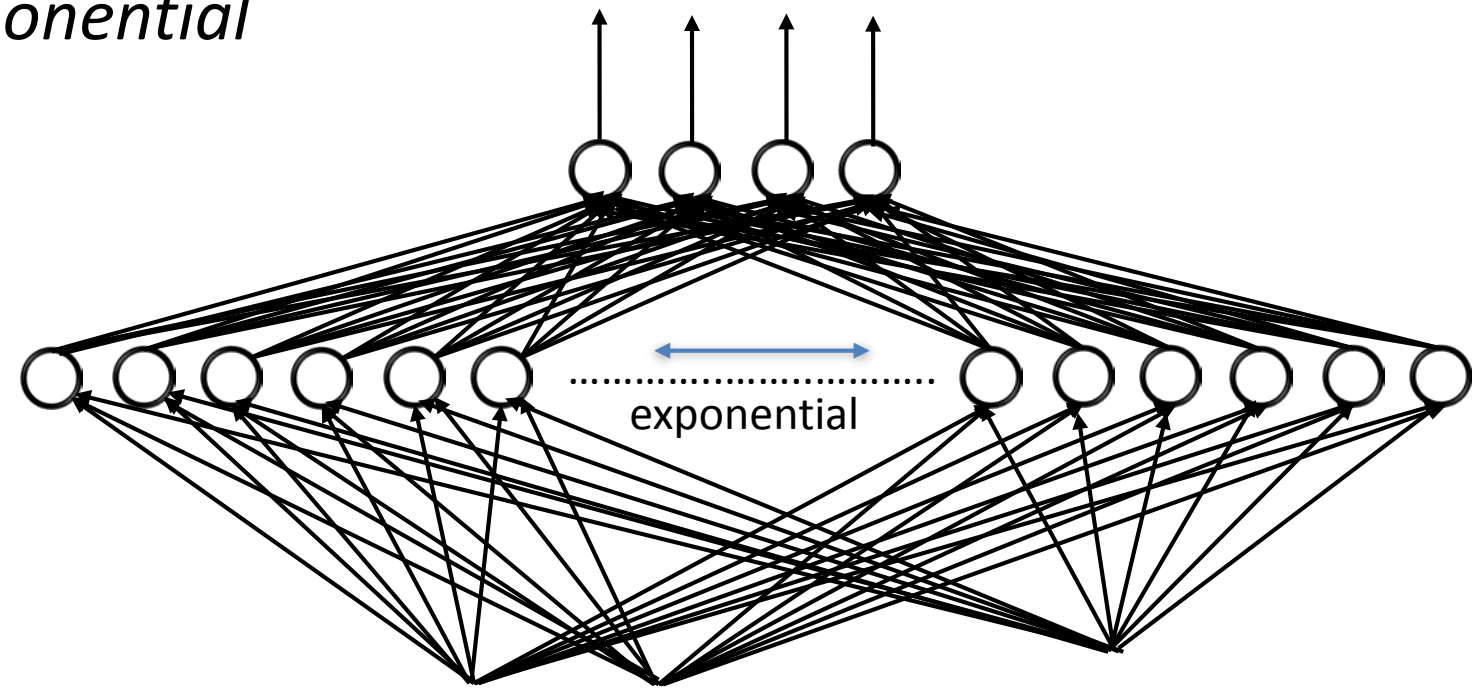
- *Neural networks: tricks of the trade*. Orr, Genevieve B., and Klaus-Robert Müller, eds. Springer, 2003.
- http://neuralnetworksanddeeplearning.com/index.html
- *Neural networks and learning machines* (Vol. 3). Haykin, S. S. Upper Saddle River, NJ, USA:: Pearson, 2009. https://cours.etsmtl.ca/sys843/REFS/Books/ebook_Haykin09.pdf
- *Make your own neural network*. Rashid, Tariq. CreateSpace Independent Publishing Platform, 2016.

UNIVERSITÉ CÔTE D'AZUR

DEEP LEARNING

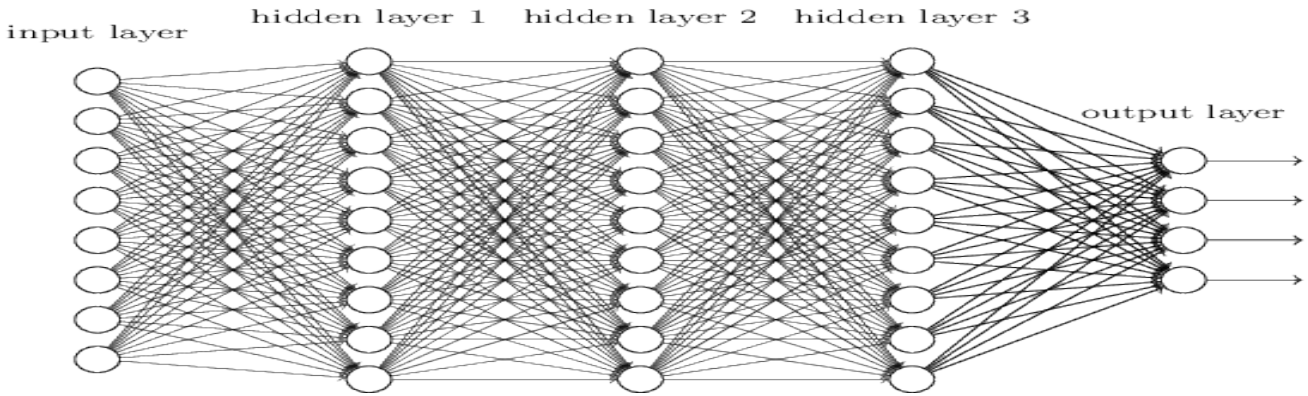
Deep representation origins

- **Theorem Cybenko** (1989) *A neural network with one single hidden layer is a universal "approximator", it can represent any continuous function on compact subsets of* $\boldsymbol{R}^n \Rightarrow$ *2 layers are enough…but hidden layer size may be exponential*



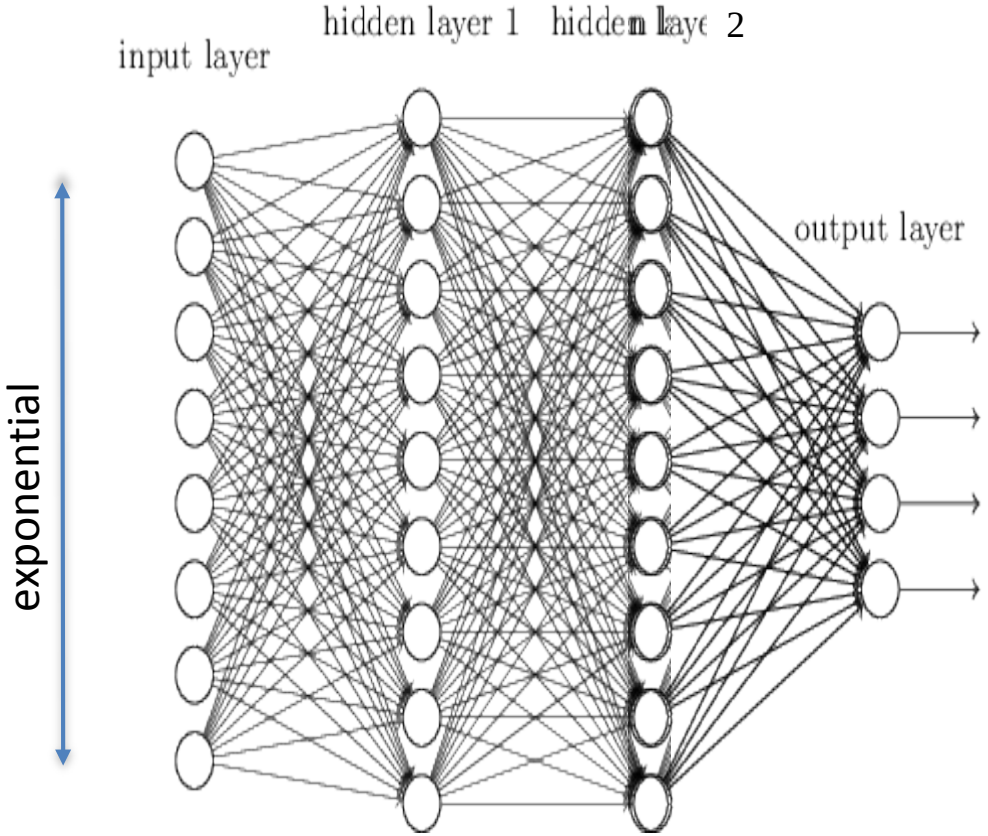
Deep representation origins

- **Theorem Hastad** (1986), **Bengio** et al. (2007) Functions representable compactly with *k* layers may require exponentially size with *k-1* layers



Deep representation origins

- **Theorem Hastad** (1986), **Bengio** et al. (2007) Functions representable compactly with *k* layers may require exponentially size with *k-1* layers



Enabling factors

- Why do it now ? Before 2006, training deep networks was unsuccessful because of practical aspects
  - faster CPU's
  - parallel CPU architectures
  - advent of GPU computing

- Hinton, Osindero & Teh « A Fast Learning Algorithm for Deep Belief Nets », *Neural Computation*, 2006
- Bengio, Lamblin, Popovici, Larochelle « Greedy Layer-Wise Training of Deep Networks », *NIPS'2006*
- Ranzato, Poultney, Chopra, LeCun « Efficient Learning of Sparse Representations with an Energy-Based Model », *NIPS'2006*

- Results…
  - 2009, sound, interspeech + ~24%
  - 2010, images, ImageNet + ~20%
  - 2011, text, + ~15% without linguistic at all

UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

The Blessing of dimensionality: Thomas Cover's Theorem (1965)

Cover's theorem states: A complex pattern-classification problem cast in a high-dimensional space nonlinearly is more likely to be linearly separable than in a low-dimensional space
(repeated sequence of Bernoulli trials).

The number of groupings that can be formed by (*l-1*)-dimensional hyperplanes to separate N points in two classes is

$$O(N,l) = 2\sum_{i=0}^{l} \frac{(N-1)!}{(N-1-i)!i!}$$

Notice: The total number of possible groupings is 2^N

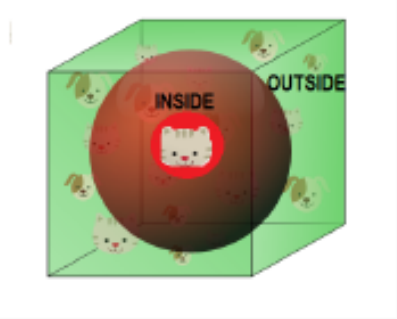
UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

The curse of dimensionality [Bellman, 1956]

- Euclidian distance is not relevant in high dimension: d $\geq$10
  1. look at the examples at distance at most r
  2. the hypersphere volume is too small: practically empty of examples

$$\frac{volume\ of\ the\ sphere\ of\ radial\ r}{hypersphere\ of\ 2r\ width} \rightarrow_{d\rightarrow\infty} 0$$



  3. need a number of examples exponential in d

Remark

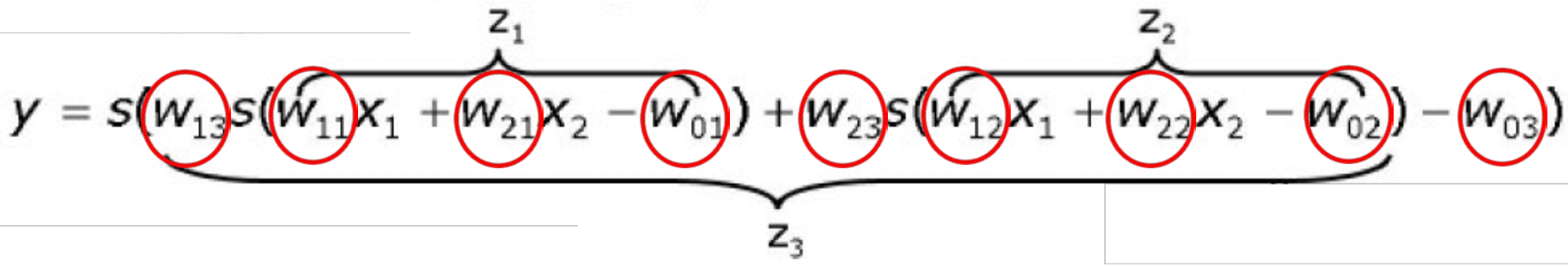
Specific care for data representation

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

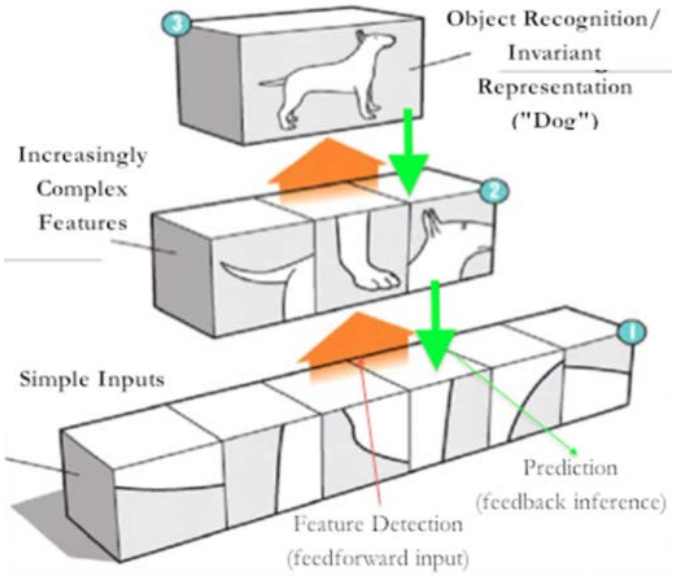
Structure the network?

- Can we put any structure reducing the space of exploration and providing useful properties (invariance, robustness…)?

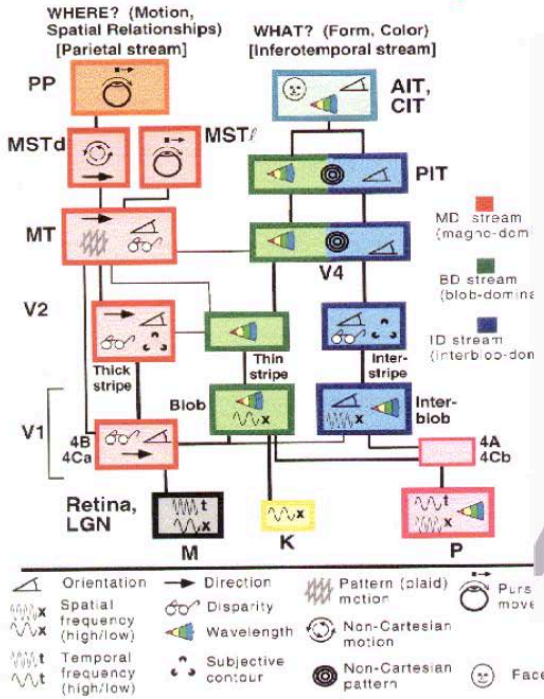


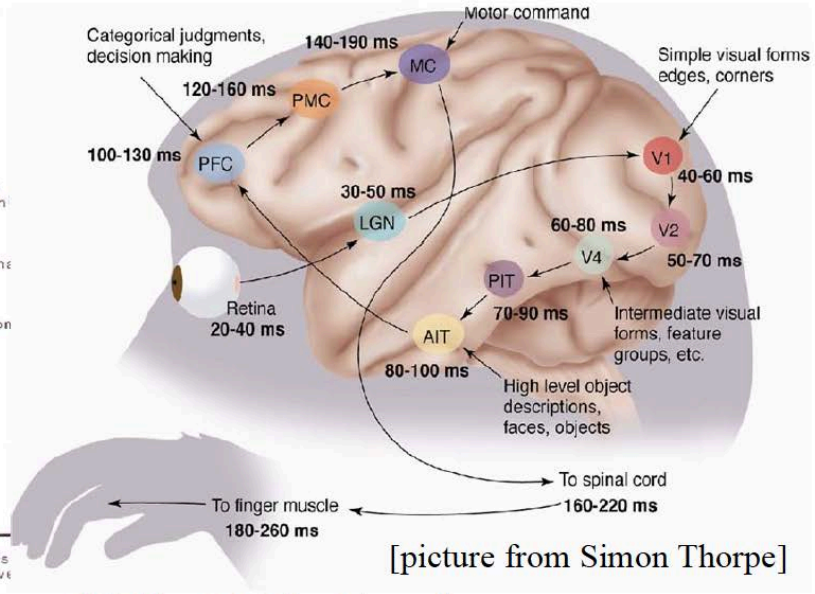
Convolutional Neural Networks (aka CNN, ConvNet)

The Mammalian Visual Cortex Inspires CNN



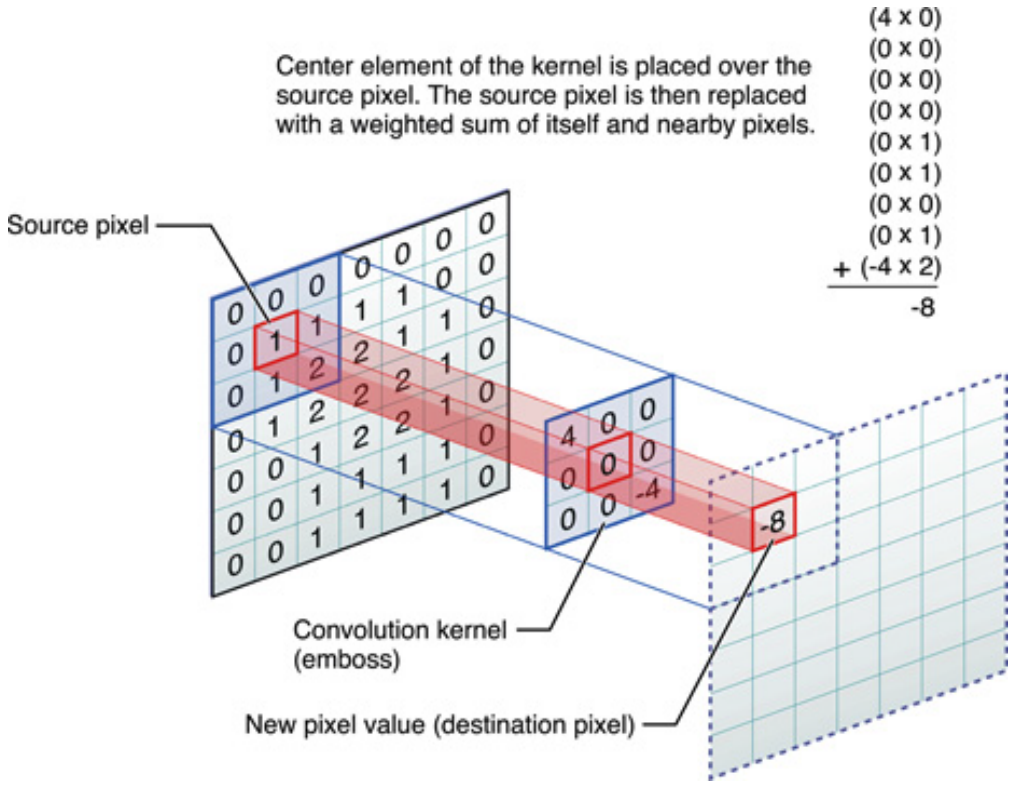
- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations

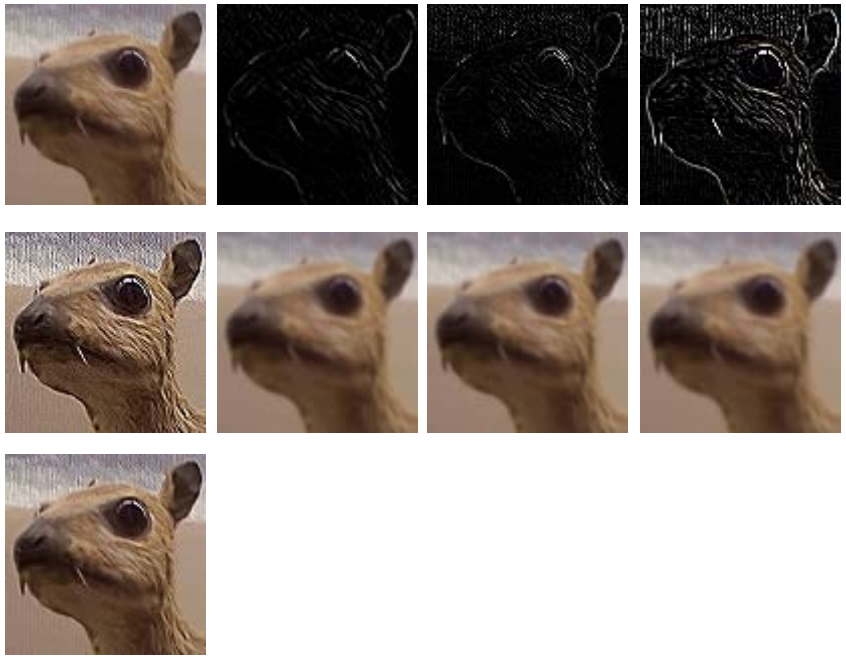




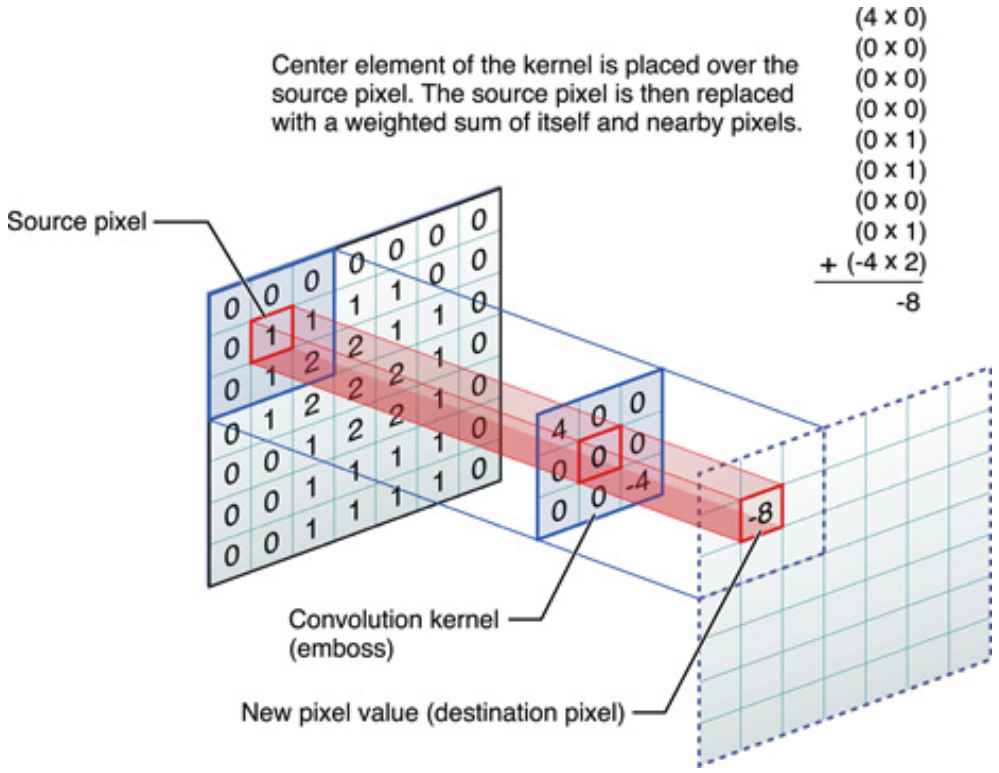
[Gallant & Van Essen]

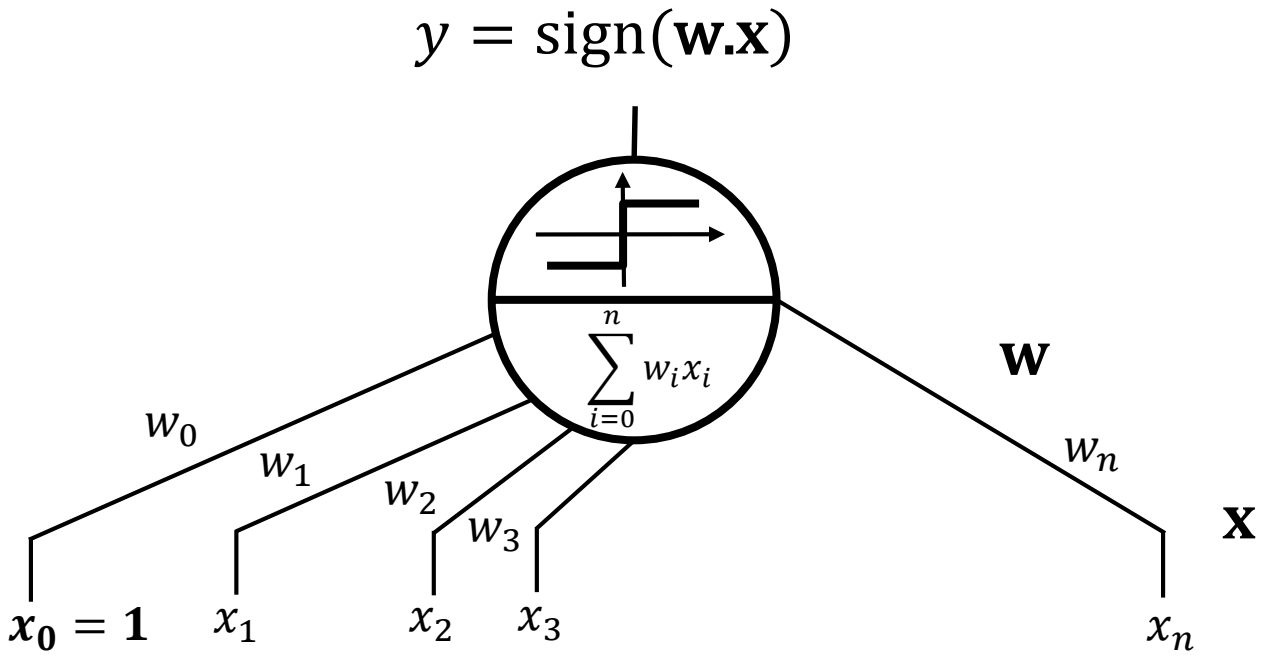
Deep representation by CNN





Deep representation by CNN





UNIVERSITÉ CÔTE D'AZUR

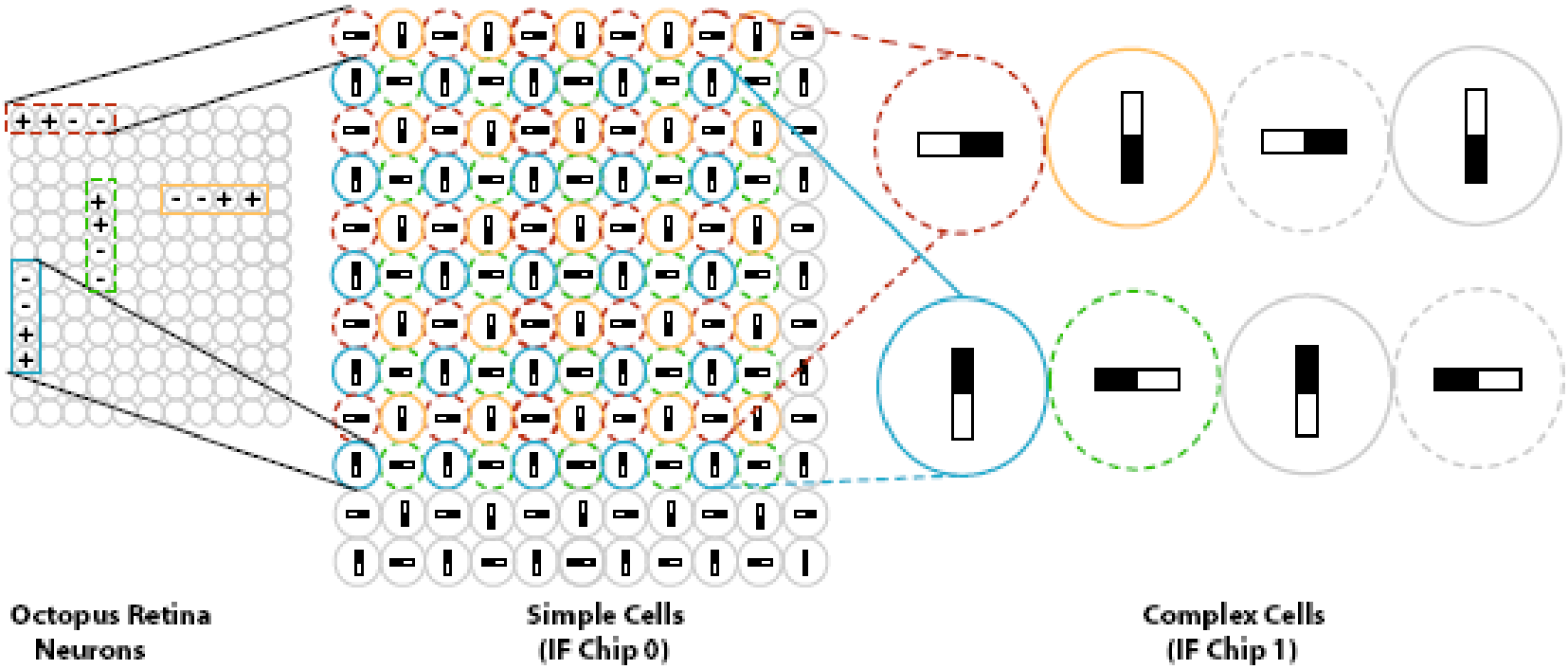
univ-cotedazur.fr

Deep representation by CNN

A cell is related to a subpart of the field of vision

Two main kind of cells:

1) S cells: extract the characteristics
2) C cells: assemble the characteristics

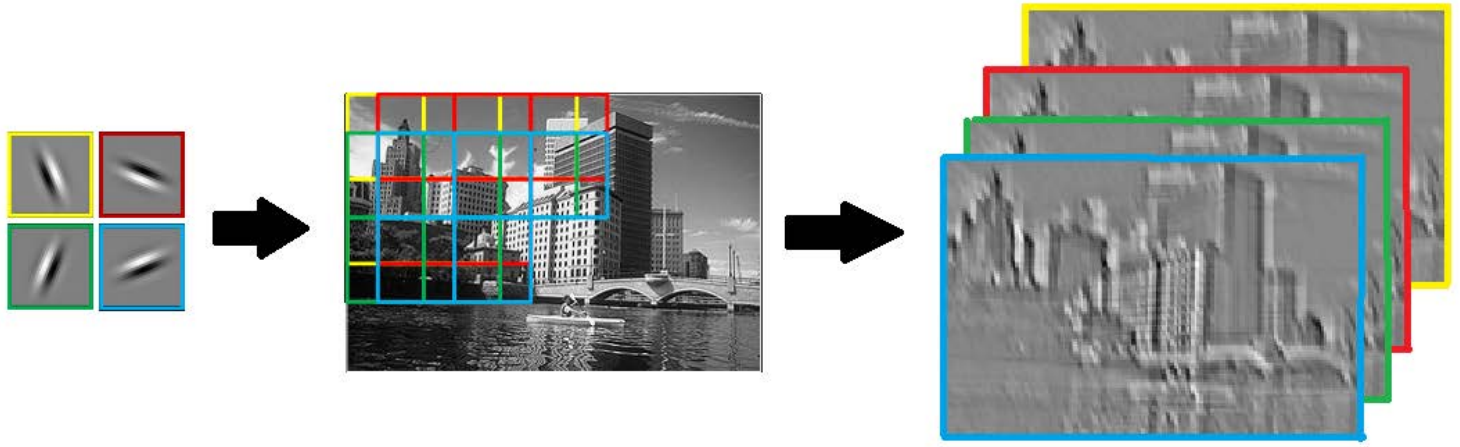


UNIVERSITÉ CÔTE D'AZUR

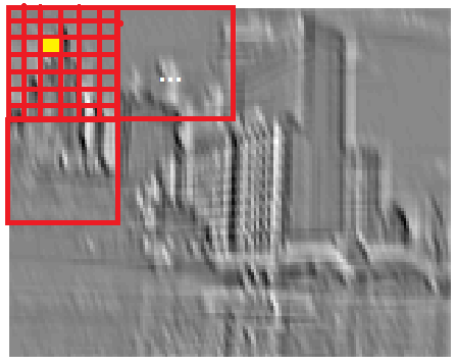
univ-cotedazur.fr

Deep representation by CNN

1. Hubel et Wiesel's work on cat's visual cells (1962)
2. Convolution

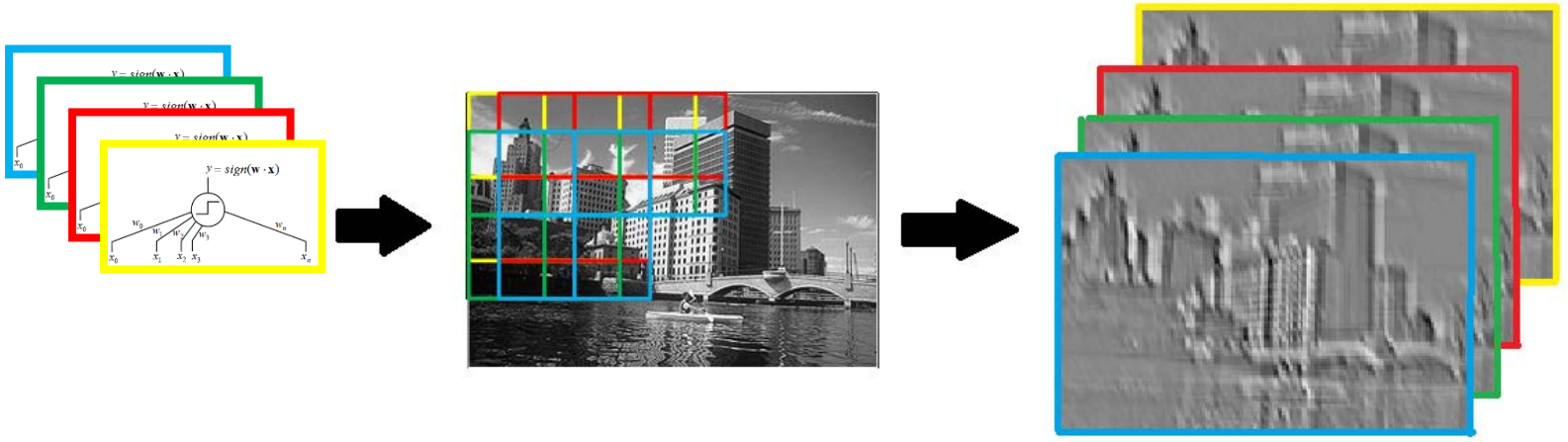


3. Max Pooling

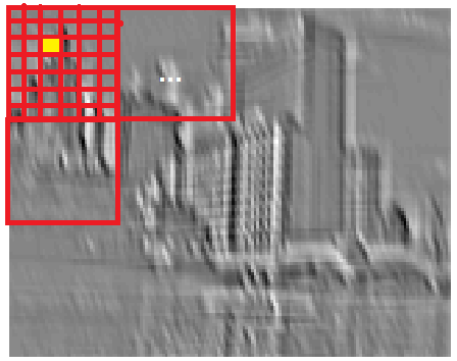


Deep representation by CNN

1. Hubel et Wiesel's work on cat's visual cells (1962)
2. Convolution



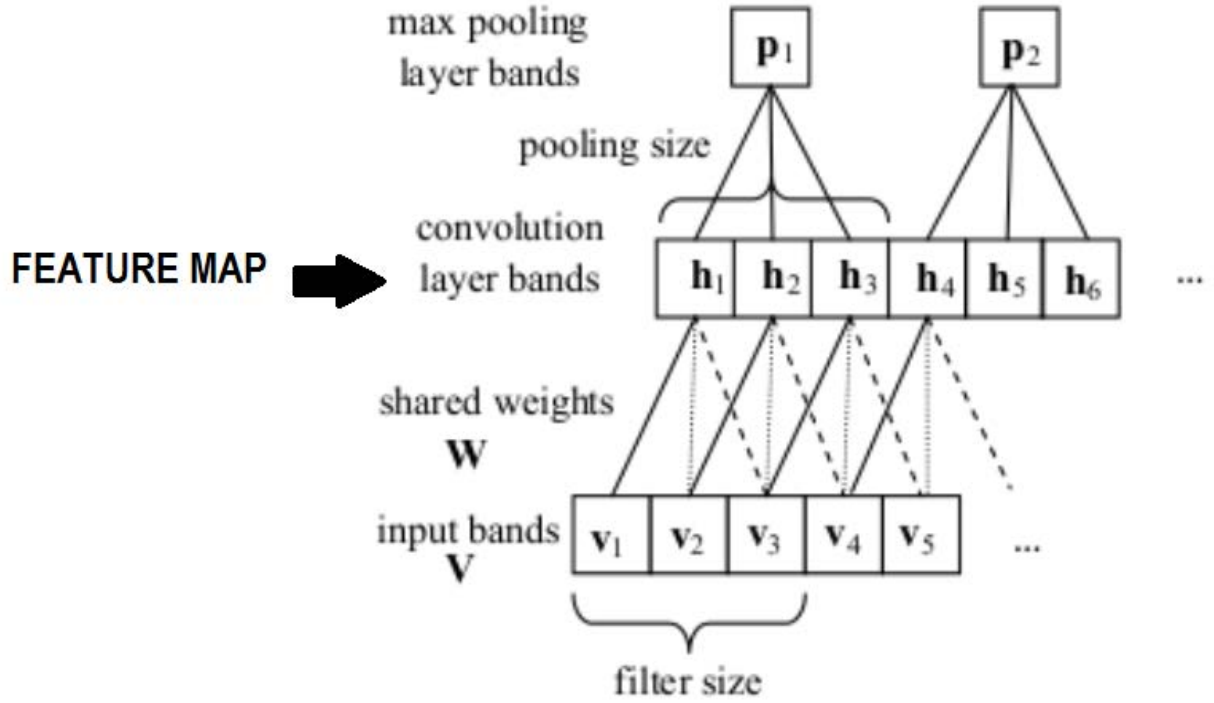
3. Max Pooling



Deep representation by CNN

Yann Lecun, [LeCun et al., 1998]

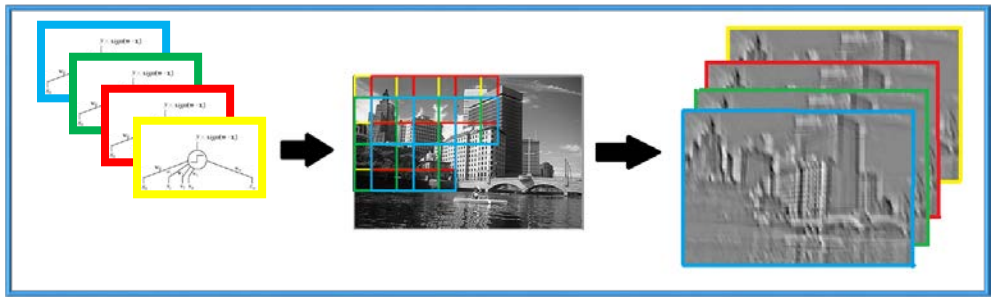
1. Subpart of the field of vision and translation invariant
2. S cells: convolution with filters
3. C cells: max pooling

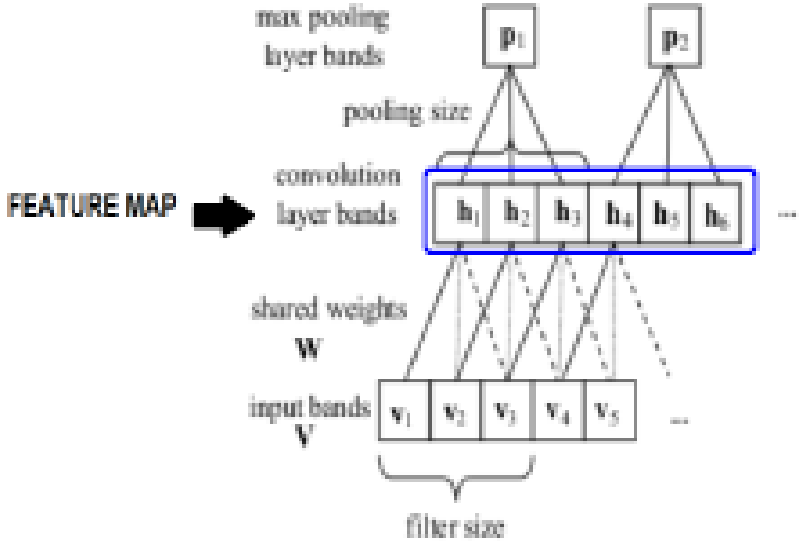


Deep representation by CNN

Yann Lecun, [LeCun et al., 1998]

1. Subpart of the field of vision and translation invariant
2. S cells: convolution with filters
3. C cells: max pooling

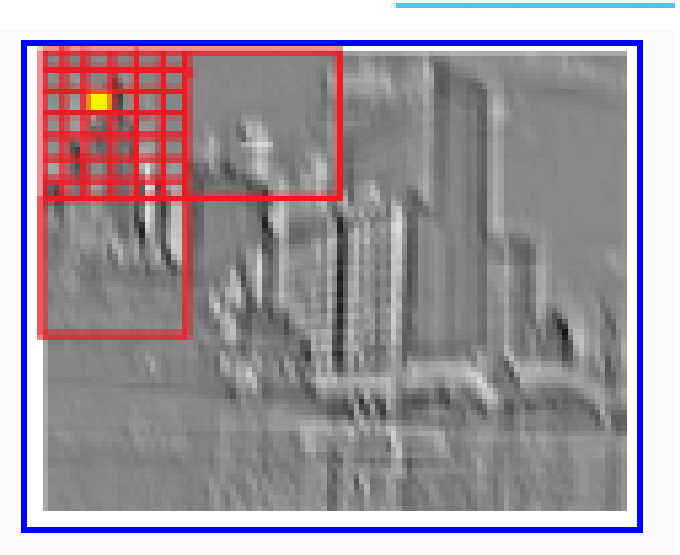


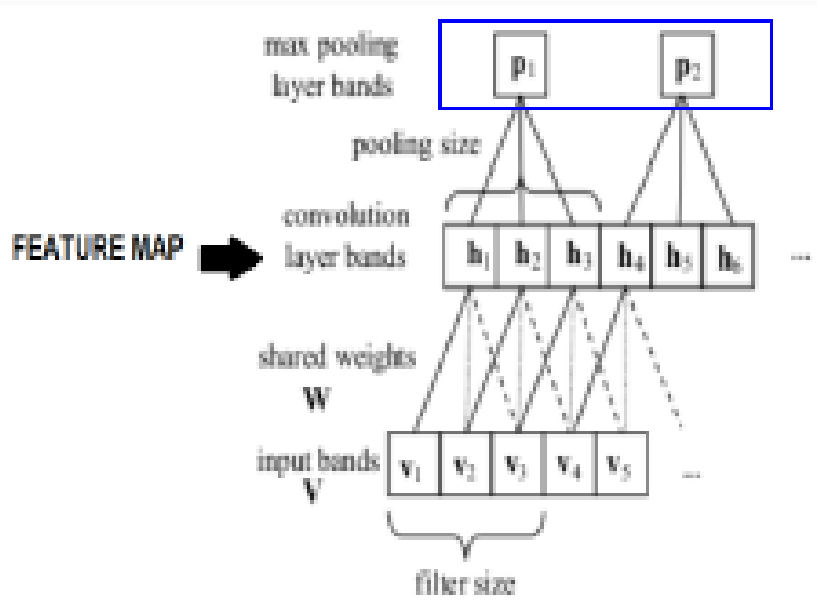


Deep representation by CNN

Yann Lecun, [LeCun et al., 1998]

1. Subpart of the field of vision and translation invariant
2. S cells: convolution with filters
3. C cells: max pooling



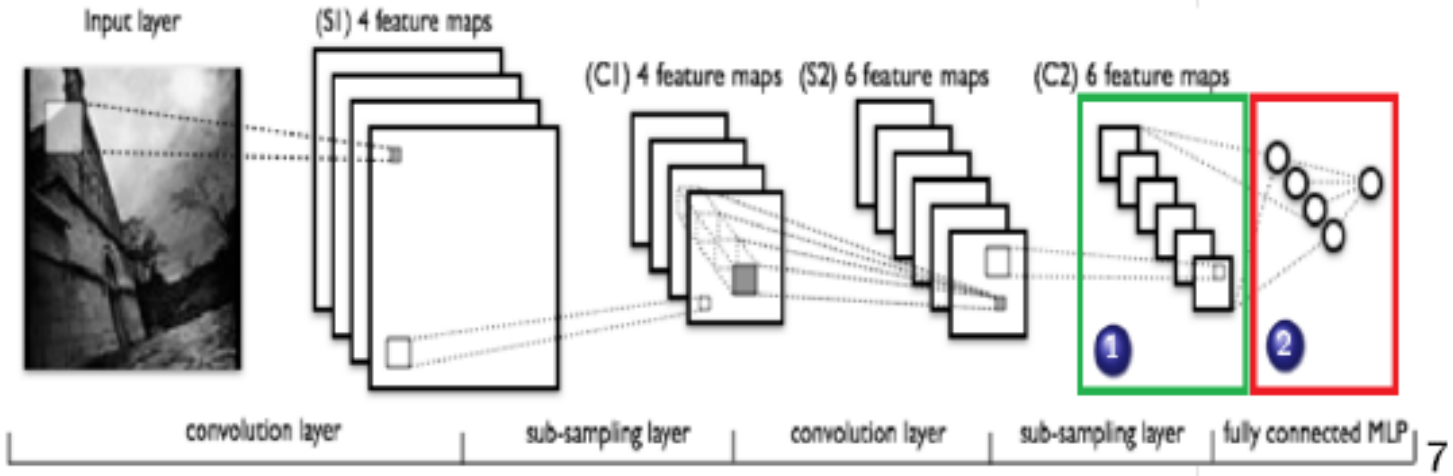


UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Deep representation by CNN

- feature map = result of the convolution
- convolution with a filter extract characteristics (*edge detectors*)
- extract parallelised characteristics at each layer



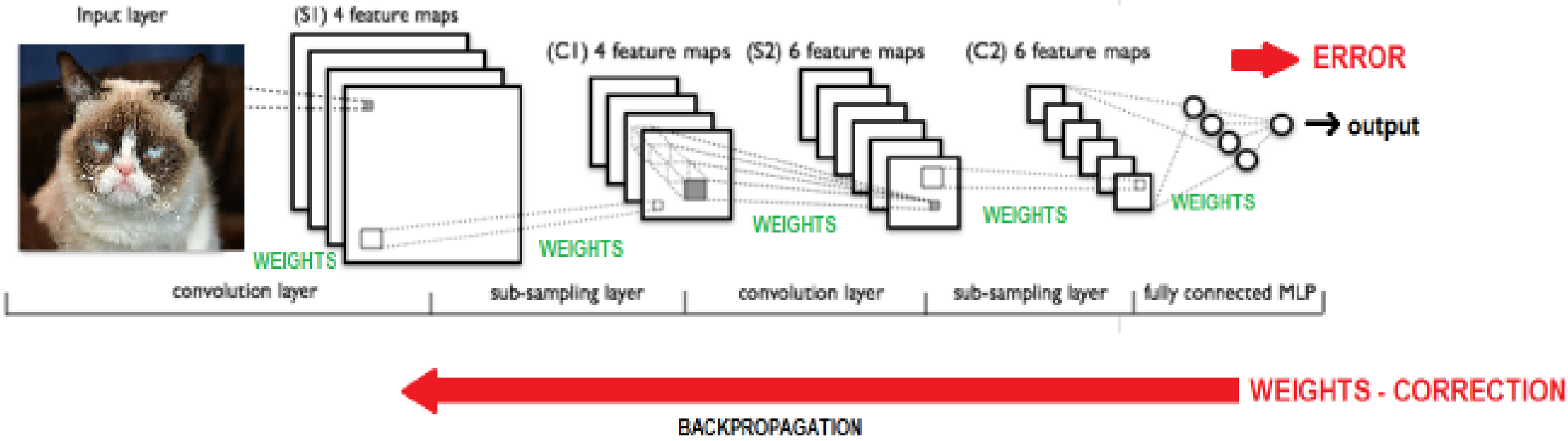
7

1. final representation of our data
2. classifier (MLP)

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

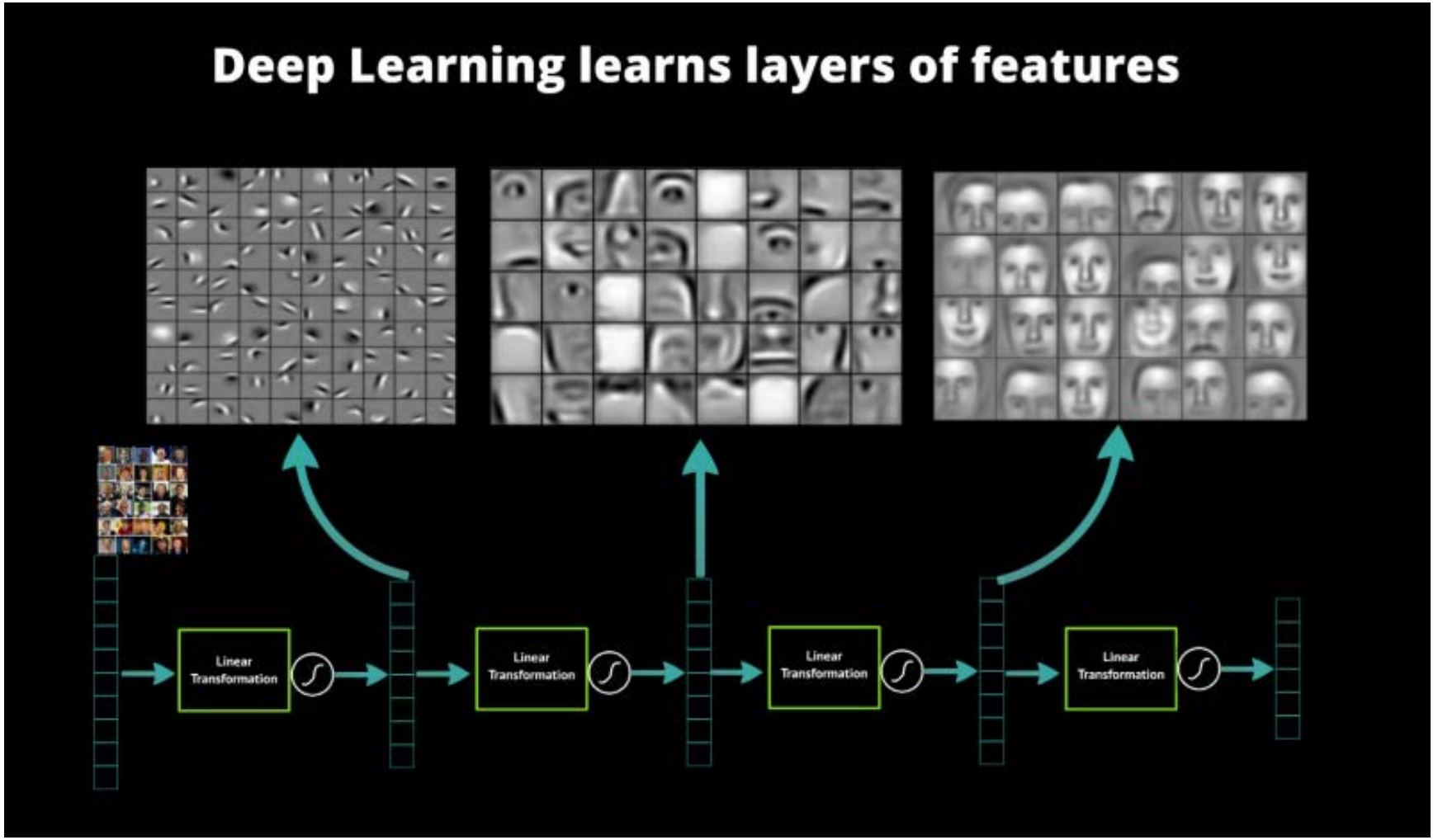
Deep representation by CNN



UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

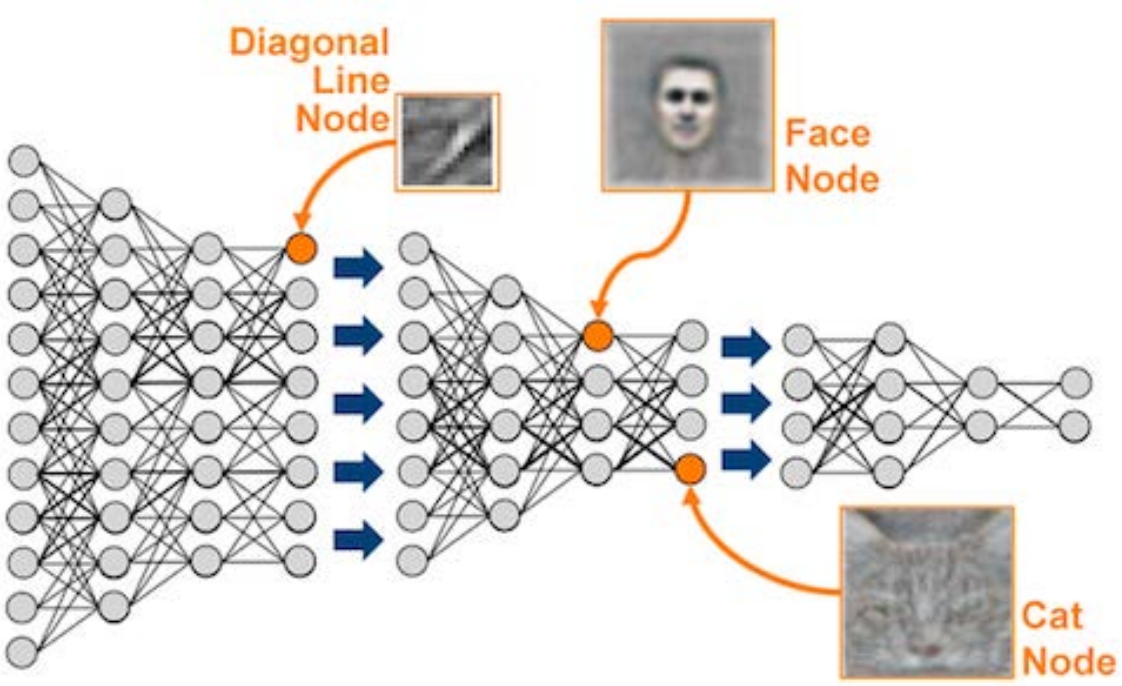
Deep representation by CNN

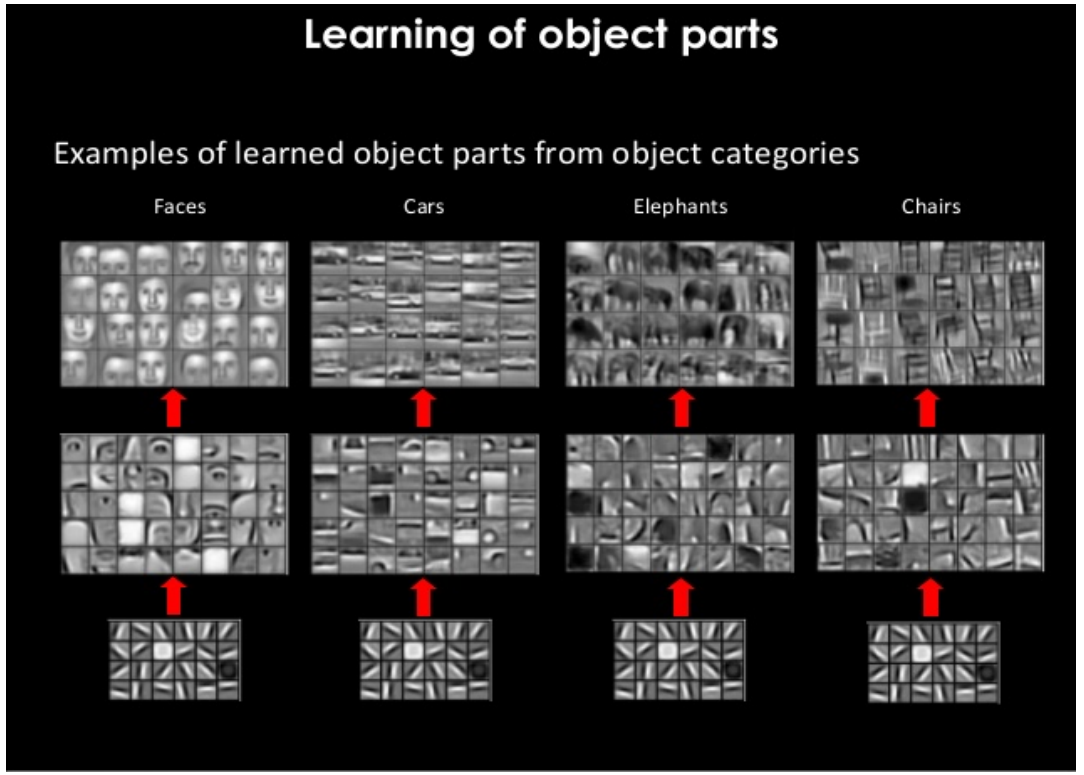


UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Deep representation by CNN



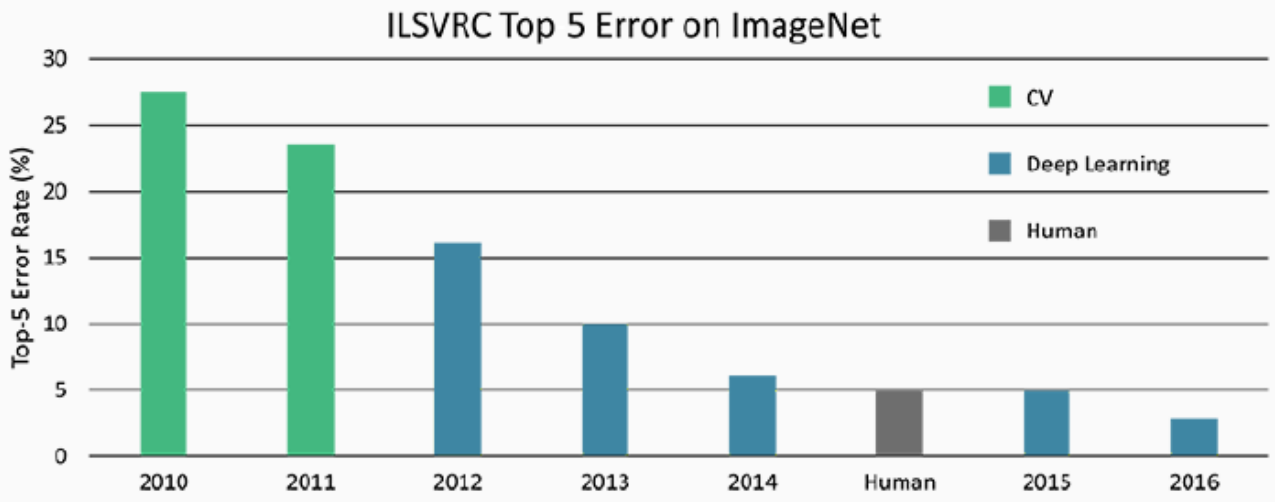


UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Deep representation by CNN

- Deep Networks are as good as humans at recognition, identification...

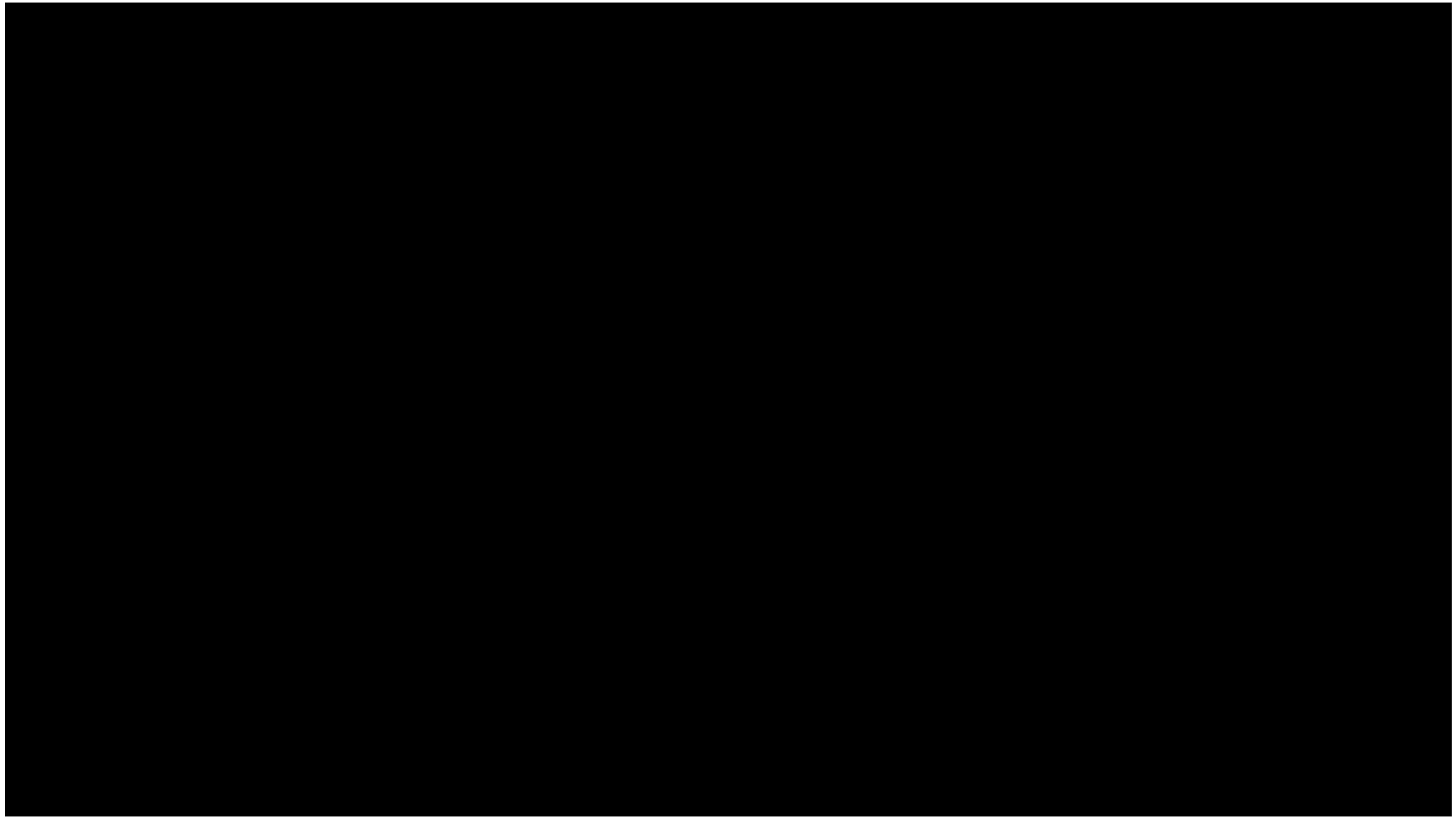


How much does a deep network understands those tasks?

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

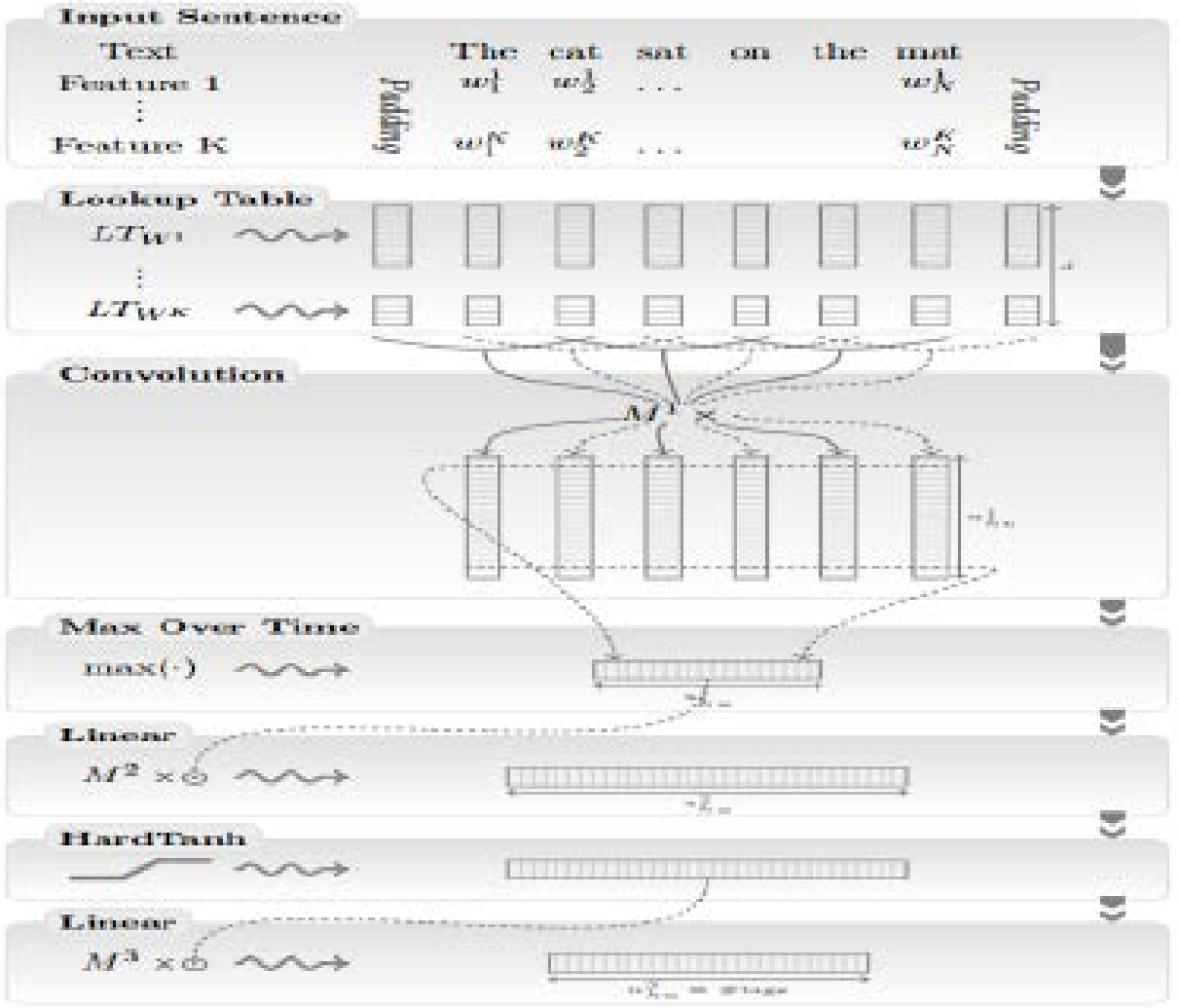
Deep representation by CNN



UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Extension to text



Extension to text

Task	Benchmark	Collobert
Part of Speech	97.24%	97.29%
Chunking	94.29%	94.32%
Named Entity Recognition	77.92%	75.49%
Semantic Role Labeling	89.31%	89.59%

Collobert is working quite well but:

1. 852 million words
2. 4 weeks

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Extension to text

http://mesure-du-discours.unice.fr/?selected_base=Campagne2017

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Transfer Learning!!

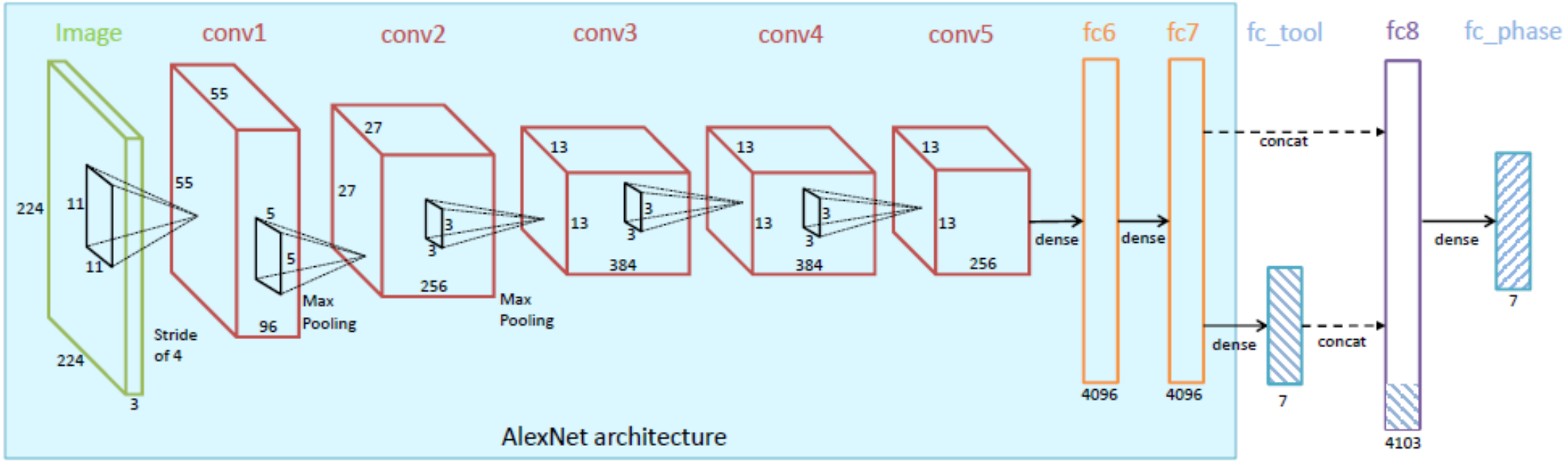


Fig. 2: EndoNet architecture (best seen in color). The layers shown in the turquoise rectangle are the same as in the AlexNet architecture.

Adversarial examples

UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

Amazing but…Adversarial examples

Intriguing properties of neural networks

C. Szegedy, w. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus

arXiv preprint arXiv:1312.6199

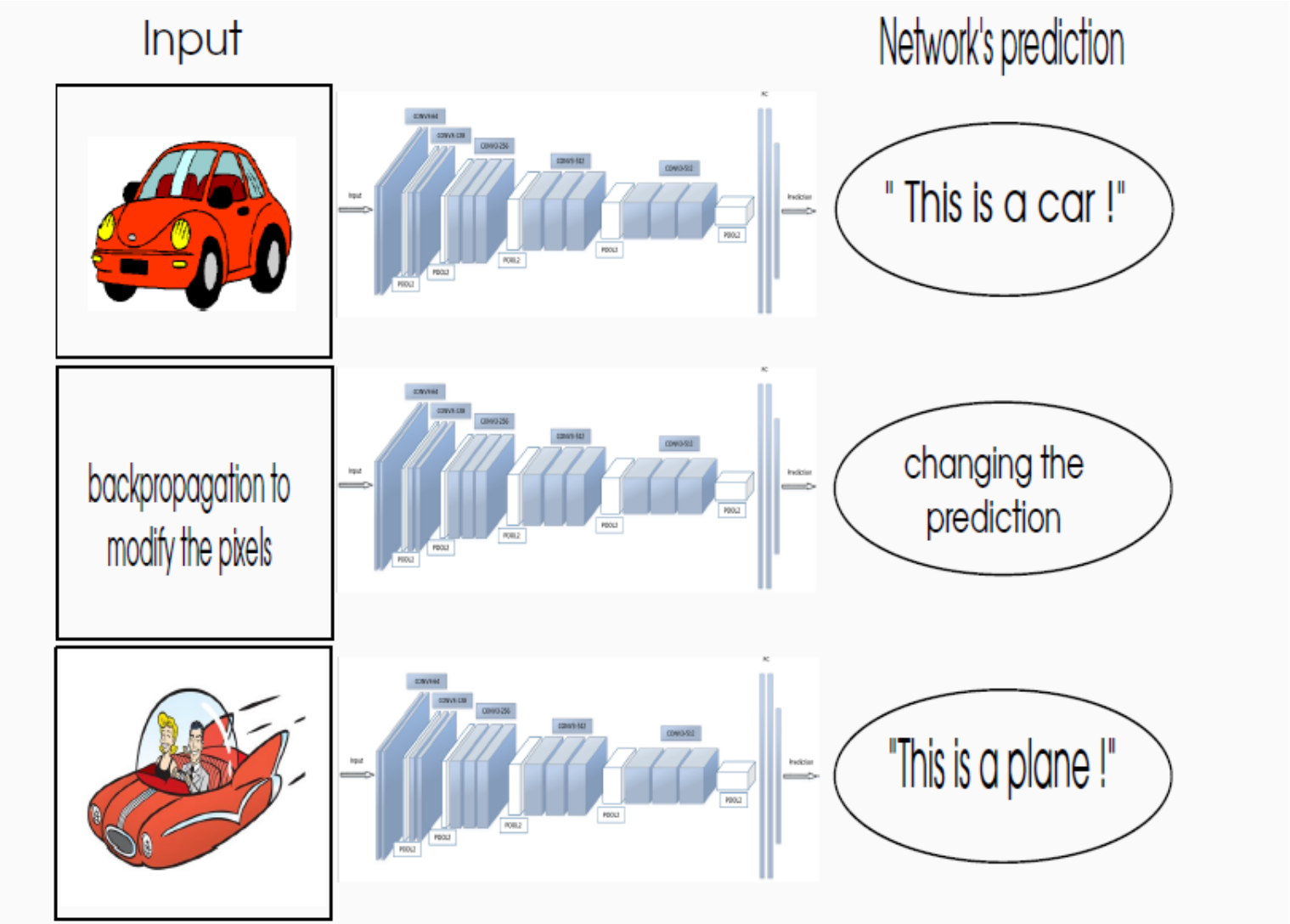
2013

[1312.6199] Intriguing properties of neural networks - arXiv.org
https://arxiv.org › cs - Traduire cette page
de C Szegedy - 2013 - Cité 449 fois - Autres articles
21 déc. 2013 - In this paper we report two such **properties**. First, we ... Second, we find that deep **neural networks** learn input-output mappings that are fairly ...

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

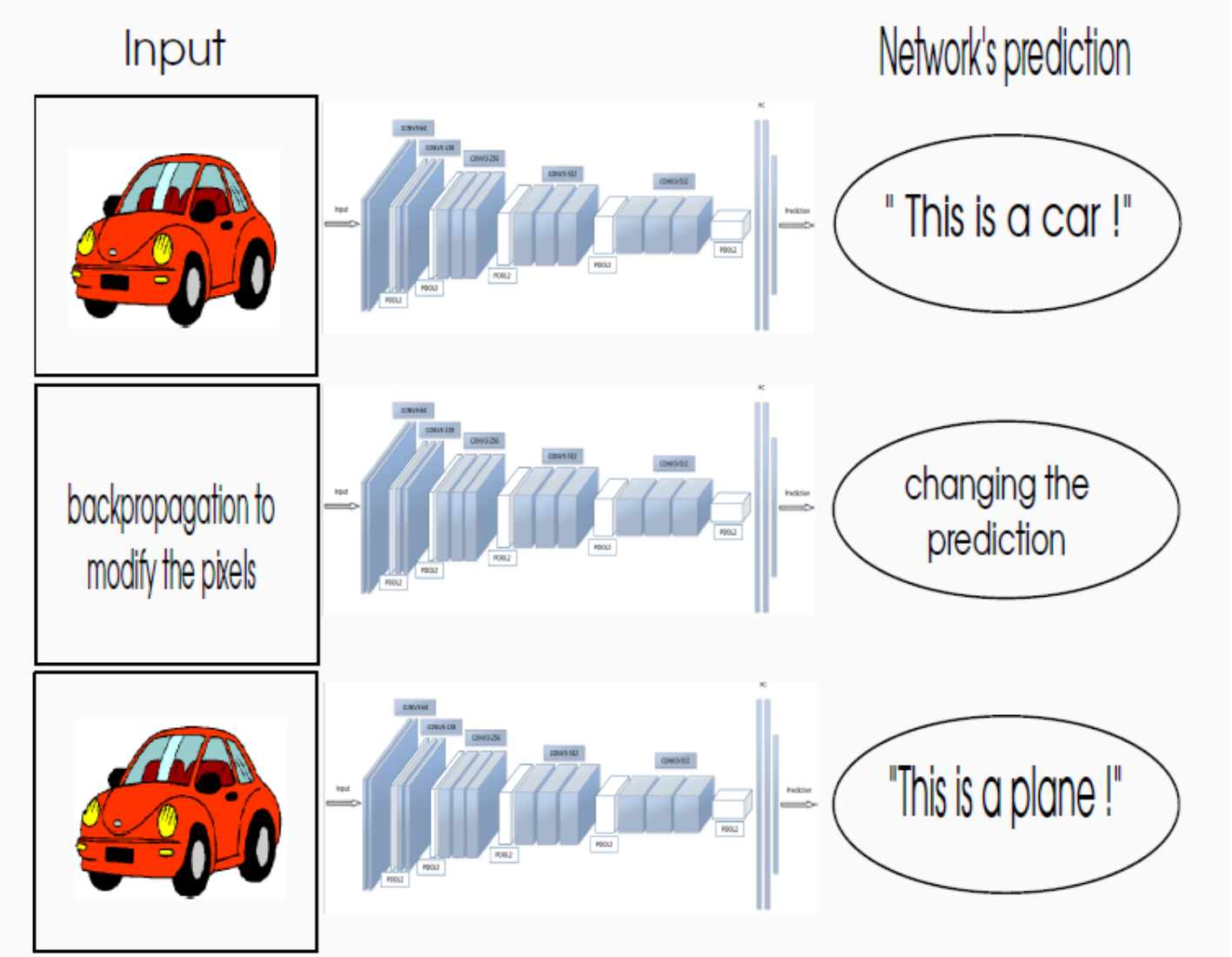
Amazing but…Adversarial examples



UNIVERSITÉ CÔTE D'AZUR

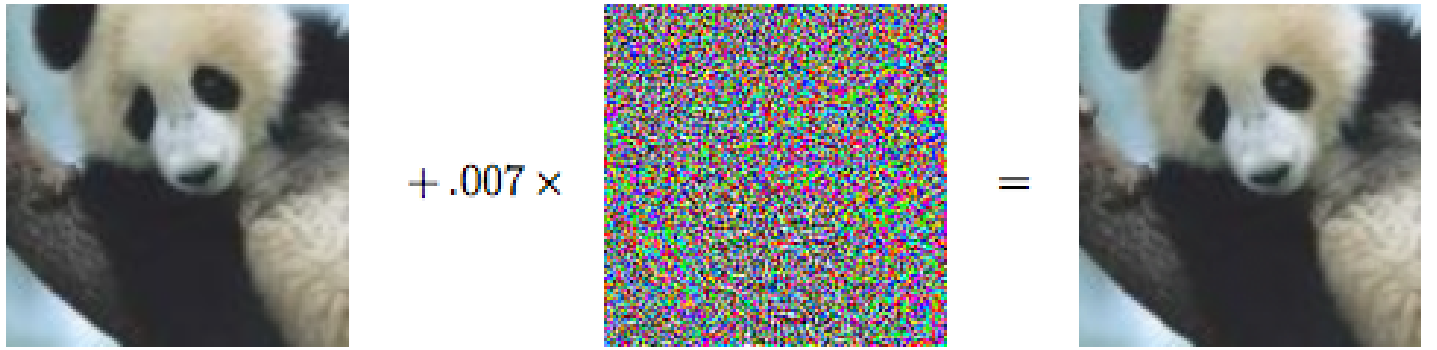
univ-cotedazur.fr

Amazing but…Adversarial examples



UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

Amazing but…Adversarial examples

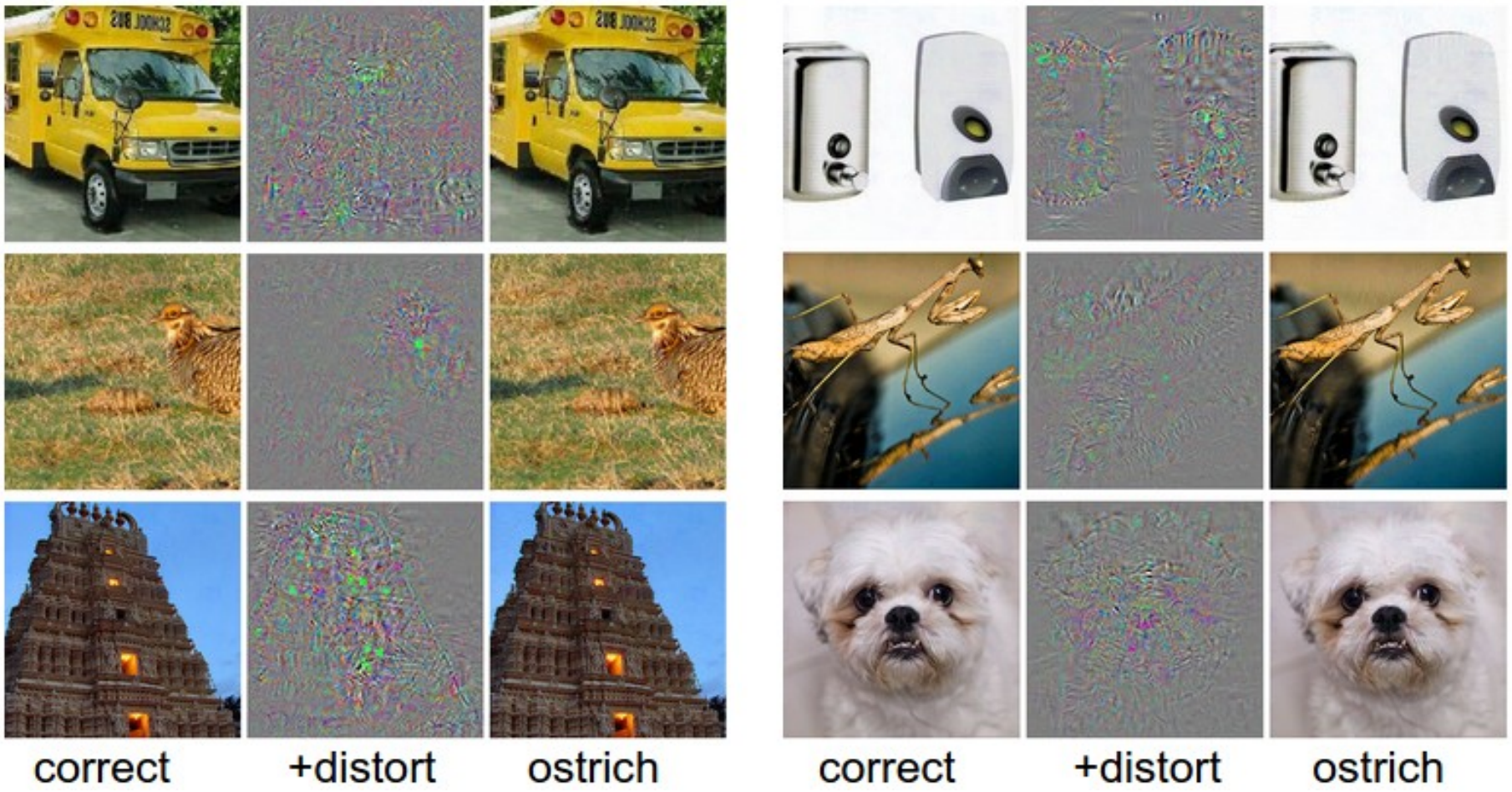


Andrej Karpathy blog, http://karpathy.github.io/2015/03/30/breaking-convnets/

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

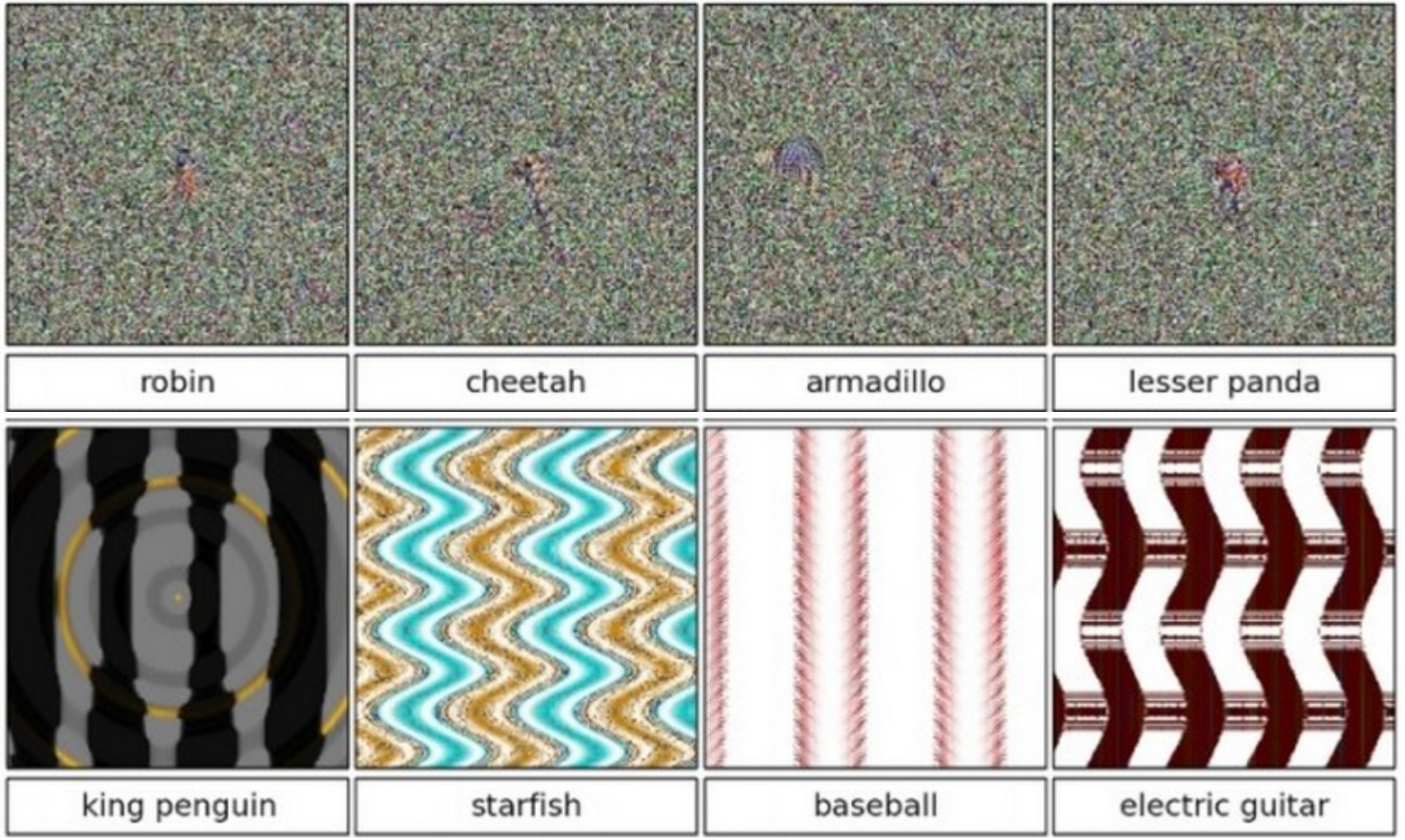
Amazing but…Adversarial examples



Andrej Karpathy blog, http://karpathy.github.io/2015/03/30/breaking-convnets/

UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

Amazing but…Adversarial examples



Andrej Karpathy blog, http://karpathy.github.io/2015/03/30/breaking-convnets/

UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

Amazing but…Adversarial examples

Definition: $\hat{x}$ is called adversarial iff:

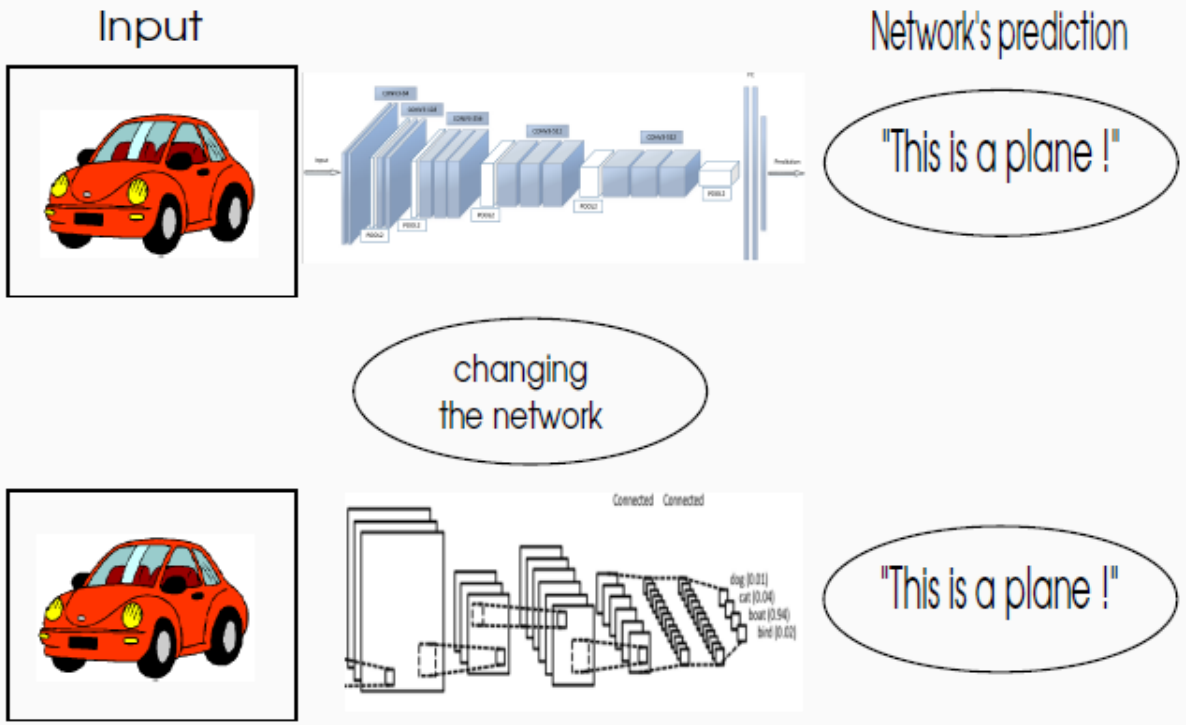
- given image x
- low distortion $|| x - \hat{x} ||< \epsilon, \quad (\epsilon > 0$, few pixels)
- given network's probabilities $f_\theta(x)$
- Different predictions! $argmax f_\theta(x) \neq argmax f_\theta(\hat{x})$

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

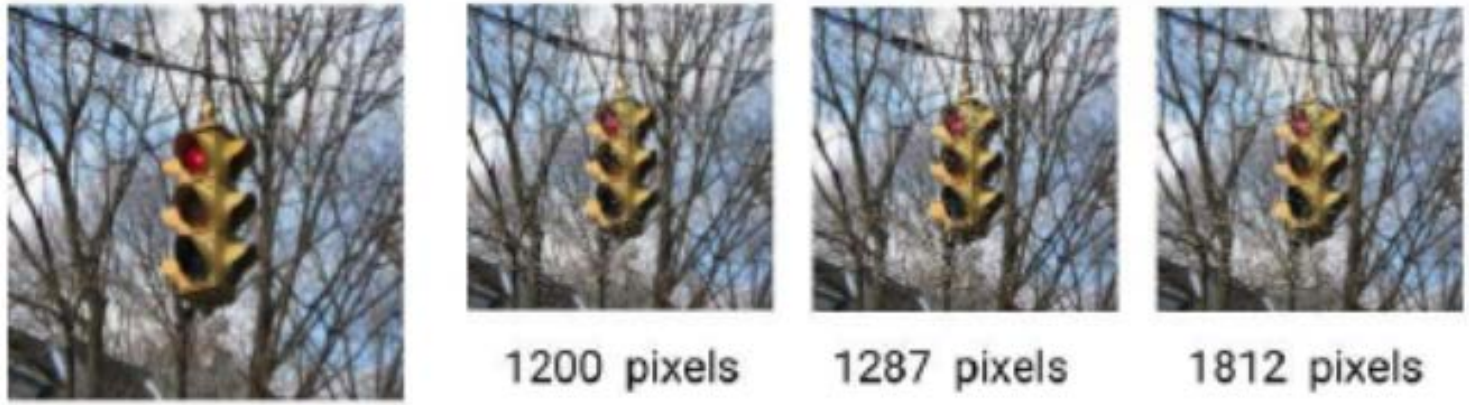
Amazing but…Adversarial examples

- $\neq$ outliers
- regularization: correct one… find another
- high confidence predictions
- **Transferability**



UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

Amazing but…Adversarial examples



Original Traffic light (ImageNet class 920)

VGG16

VGG19 Misclassified

RESNET

State-of-the art deep neural networks on ImageNet

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Amazing but…Adversarial examples

Red Light Modified to Green after 18 white pixels. Probability: 59%

Red Light Modified to Green after 9 green pixels. Probability: 50.9%

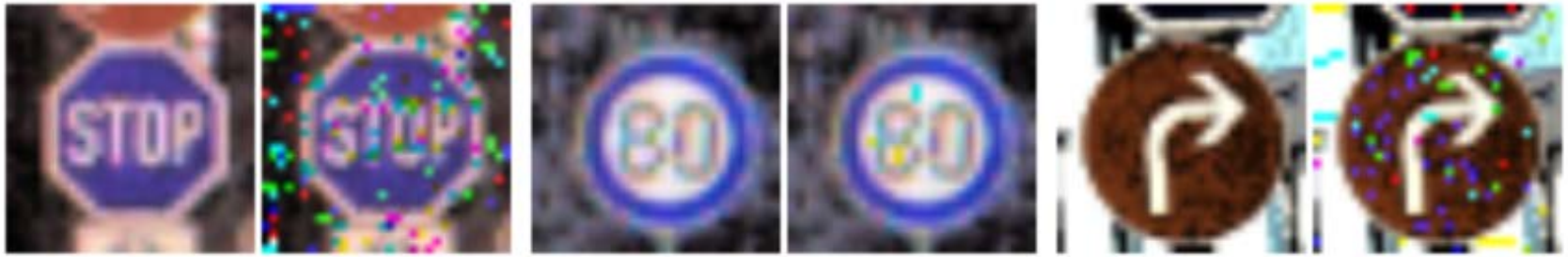
Red Light Modified to Green after 9 green pixels. Probability: 53%

No Light Modified to Green after 4 green pixels. Probability: 51.9%

UNIVERSITÉ CÔTE D'AZUR

univ-cotedazur.fr

Amazing but…Adversarial examples



stop	30m speed limit	80m speed limit	30m speed limit	go right	go straight
Confidence	0.999964		0.99		

UNIVERSITÉ CÔTE D'AZUR
univ-cotedazur.fr

Tutorial on adversarial examples

- A tutorial **in french**: Guillaume Debard
  - Slides here: http://www.telecom-valley.fr/wp-content/uploads/2017/05/DEBARD.pdf
  - Video here: https://www.youtube.com/watch?v=1wyXPY0VxTc

UNIVERSITÉ
CÔTE D'AZUR