Notes on Neural Networks: part 1

Alessio Micheli

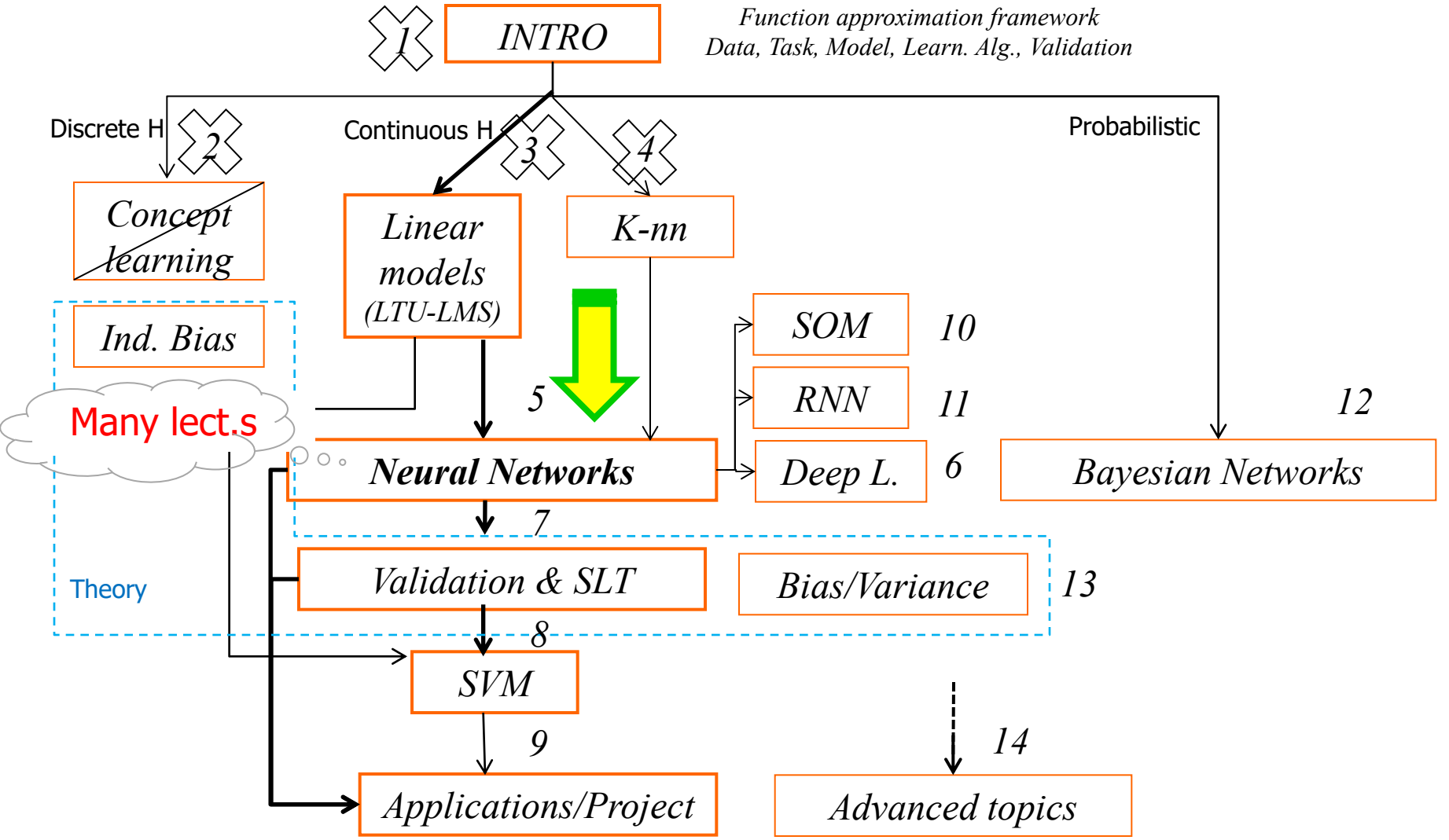
micheli@di.unipi.it

2025

Dipartimento di Informatica
Università di Pisa - Italy

ML Course structure
Where we are

Dip. Informatica
University of Pisa



NN: which target?

Dip. Informatica
University of Pisa

- NN to study and model biological systems and learning processes
  - Philosophy, Psychology and neurobiology, cognitive science, Computational neuroscience.
  - Biological realism is essential

- NN to introduce effective ML systems/algorithms (often losing a strict biological realism)
  - Statistics, Artificial Intelligence, Physics, Math., Engineering, …
  - ML, computational and algorithmic properties are essential

- For us: **ANN** (*Artificial* Neural Networks):

  indeed a (flexible) Machine Learning tool

  (in the sense of approximation of functions: it realizes a mathematical functions $h(\boldsymbol{x})$ with special properties)

A first look to NN (as ML tool)

Dip. Informatica
University of Pisa

A historical learning machine,
while still a powerful approach to ML

- NN can learn from examples
- NN are universal approximators (Theorem of Cybenko):
  flexible approaches for arbitrary functions (including non–linear) !
- NN can deal with noise and incomplete data
  - Performance degrades gracefully under adverse operating conditions
- NN can handle continuous real and discrete data
  for both regression and classification tasks
- Successful model in ML due to the flexibility in applications

- NN encompasses a wide set of models: it is a paradigm !
  (in the class of subsymbolic approaches)

"Philosophy": Connectionism

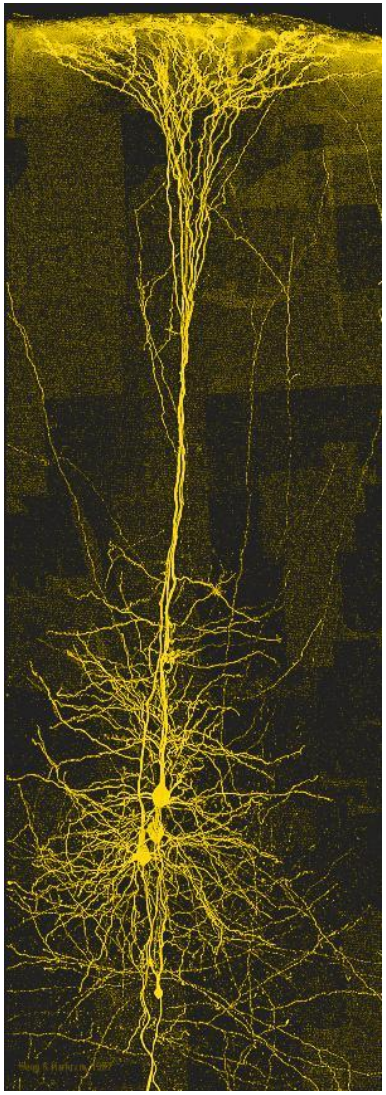
Dip. Informatica
University of Pisa

- Complex behavior emerging from the interaction of simple computational units

- Connectionism: mental or behavioural phenomena as the emergent processes of *interconnected networks of simple units*.

Nervous system

Dip. Informatica
University of Pisa

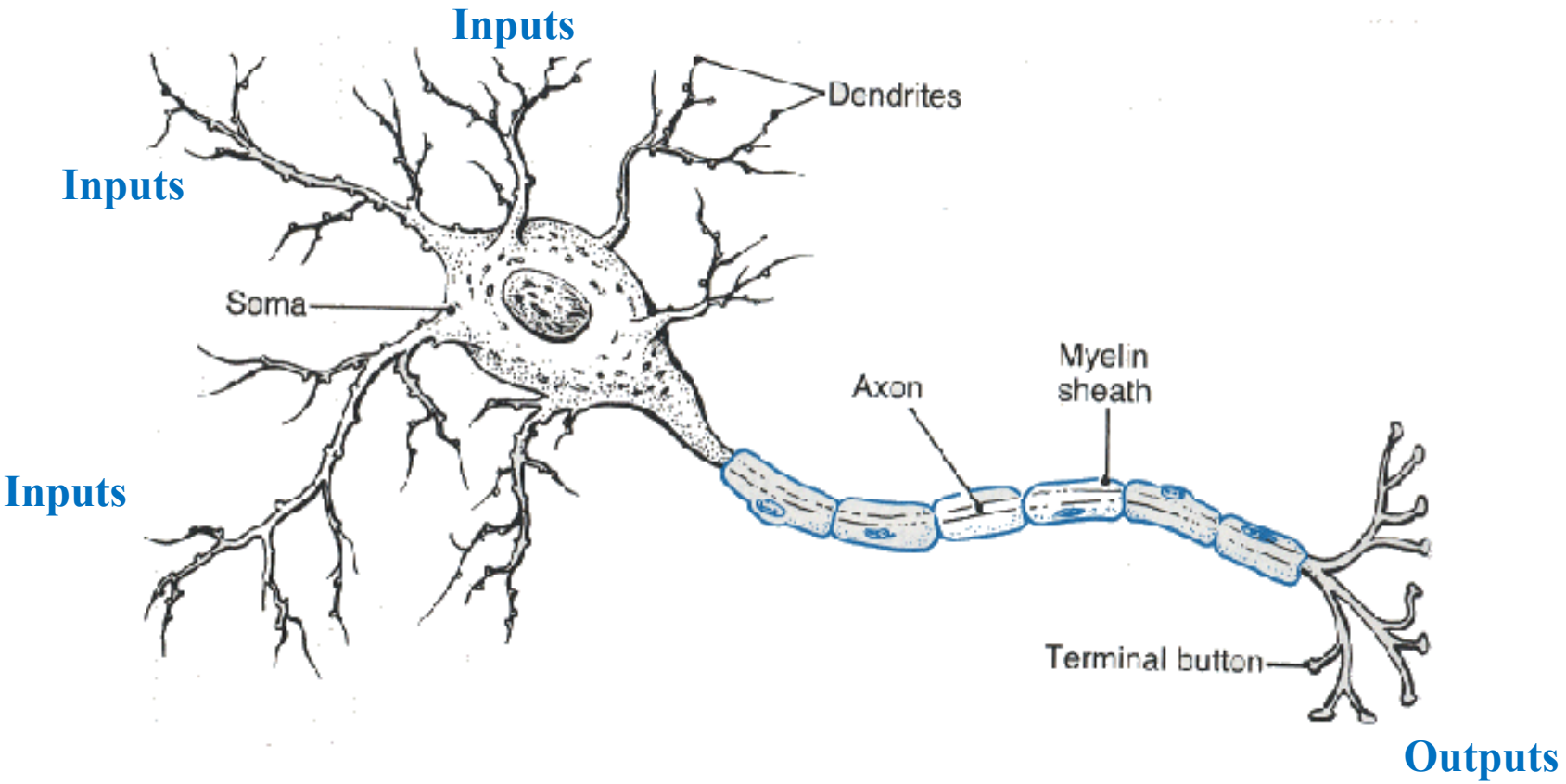


- Human Brain: more than 10^{10} neurons
- 10^4-10^5 connections for each neurons
- Response: ~0.001 sec. (msec order)
- Image recognition: 0.1 sec (only 100 serial computations)→ highly parallel computations
- (Note that face recognition can be complex for conventional computers)

Three stained pyramidal neurons. These neurons stand nearly 2mm high and receive over 10,000 inputs from other neurons which they process in their complex dendritic arbors using active regenerative mechanisms.

Biological Neuron

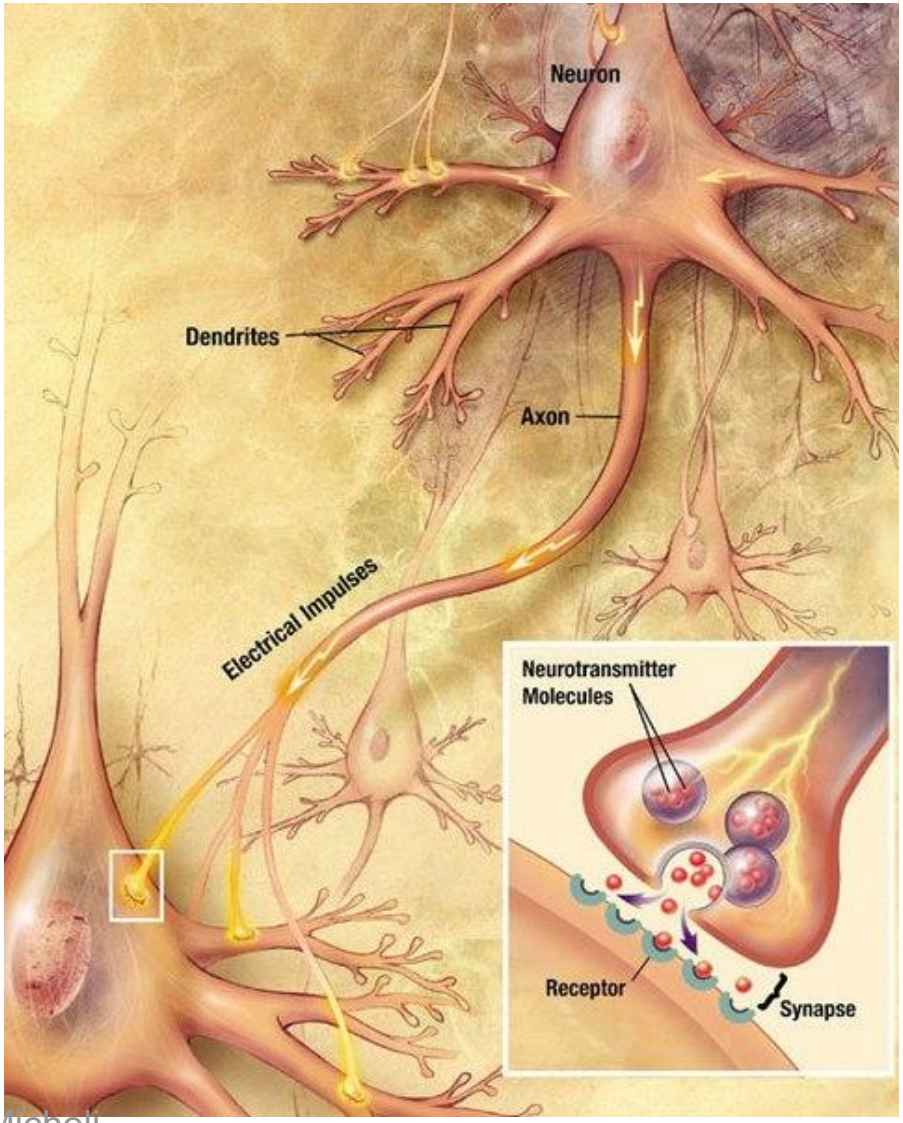
Dip. Informatica
University of Pisa



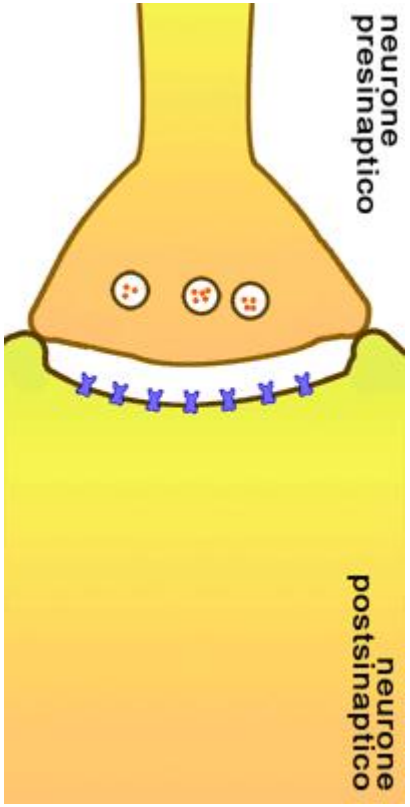
- charge / discharge cycle (sodium / potassium ions)
- changing the potential of the cell (from dendrites), threshold
- spike generation (voltage pulse), time interval among spikes

Signal propagation and Synapse

Dip. Informatica
University of Pisa



A signal propagating down on axon to the cell body and dendrites of the next cell



Electr.

Chem.

Electr.

GIF: https://it.m.wikipedia.org/wiki/Neurone#/media/File%3ASinapsi.gif

Synapse

Dip. Informatica
University of Pisa

- Pre and post synaptic membrane
- Neurotransmitter
- Inhibitory/excitatory

Change due to *learning*:

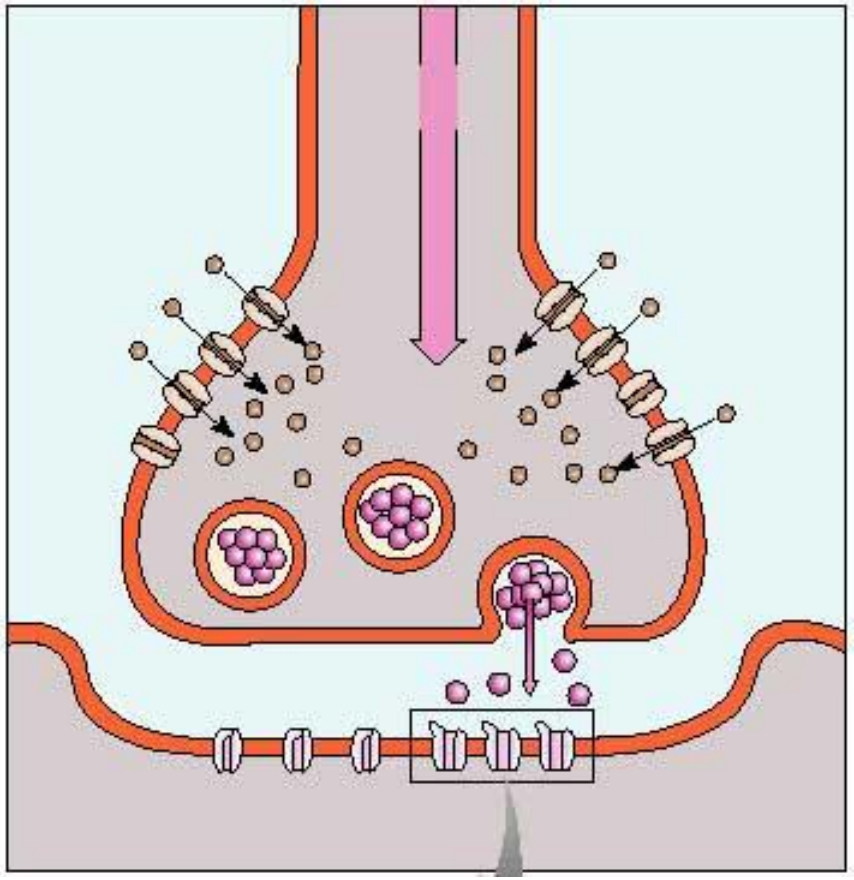
- Strength (weight) reinforcement due to stimuli
- Plasticity/adaptivity of the nervous system

- Hebbian learning (Hebb 1949)

A synapse is strengthened when the neuron inputs (and correlated outputs) are repeated.

Input stimulates synapsis strength,

so *weights become closer to inputs*



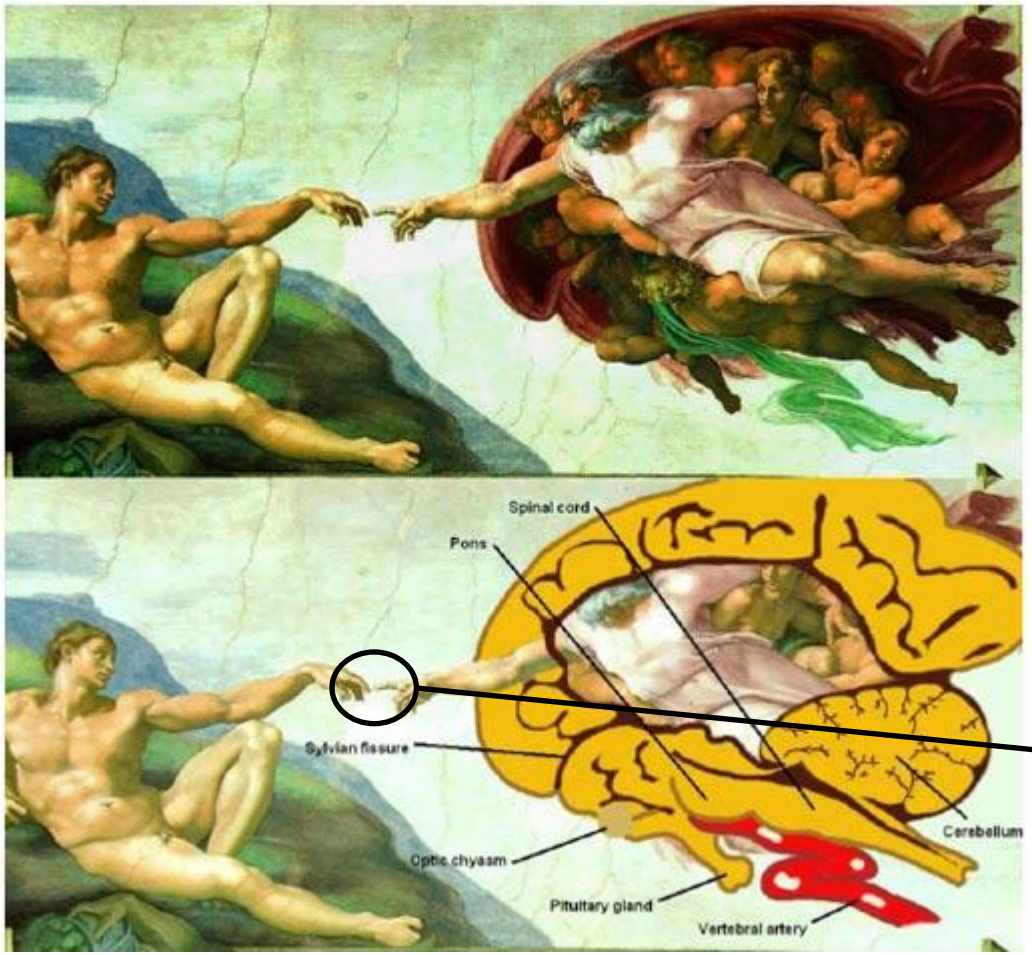
Neurons in the art
Michelangelo “The Creation of Adam”

Dip. Informatica
University of Pisa

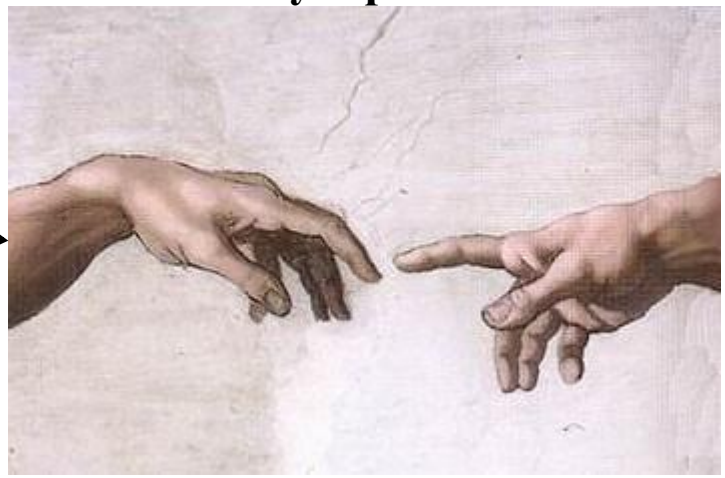
Neurons in the art

Michelangelo "The Creation of Adam"

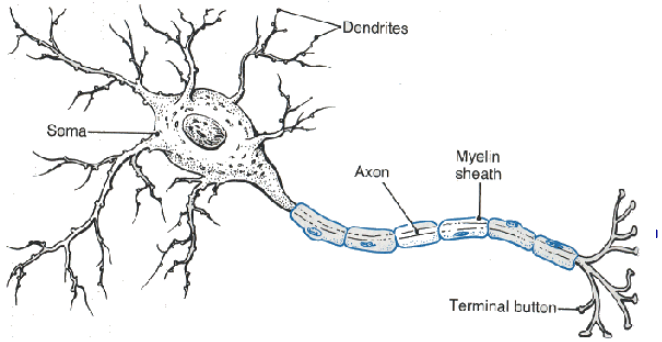
Dip. Informatica
University of Pisa



Synapse?

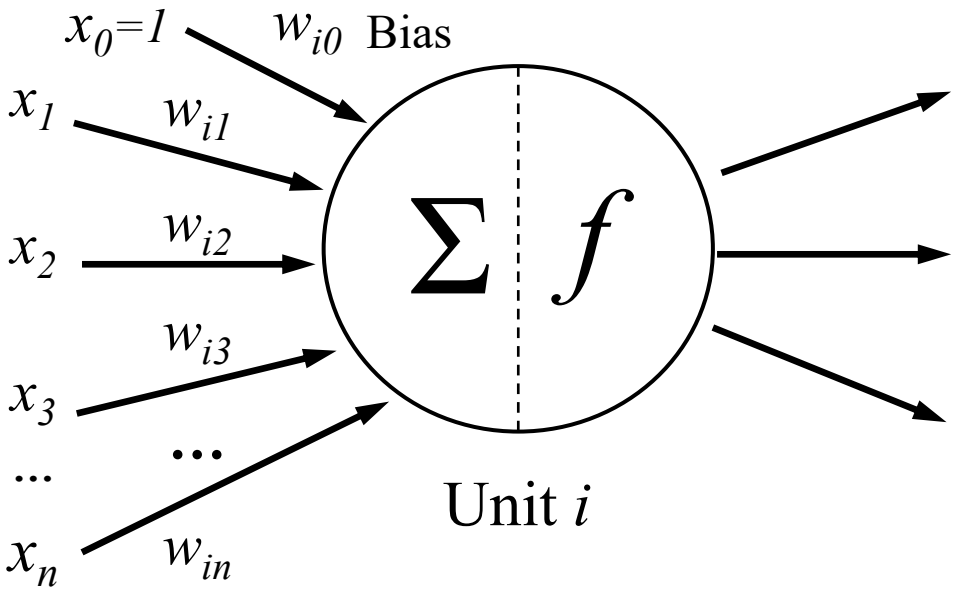


Artificial Neuron: processing unit



- Node, neuron or **unit**
- Input: from extern source or other units (R)
- Input Connections: **weights** w: free parameters: these can be modified by learning (synaptic strength)

$$\begin{cases} net_i(\boldsymbol{x}) = \sum_j w_{ij} x_j \\ o_i(\boldsymbol{x}) = f\,(net_i(\boldsymbol{x})) \end{cases}$$

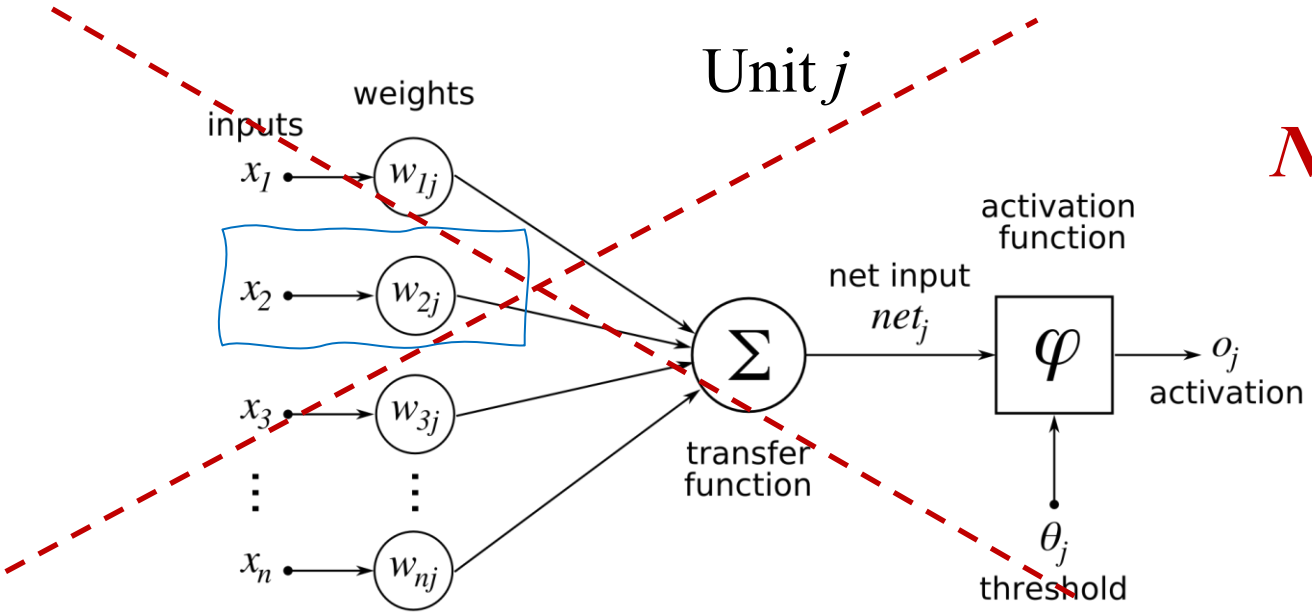


- The weighted sum net_i is called the **net input** to unit i
- Note that w_{ij} refers to the weight of the unit i, i.e.from unit/input j to unit i (not the other way around, e.g. see the next slide).
- The function f is the unit's **activation function (e.g linear, LTU, ...)**.

Notation for w_{ij}

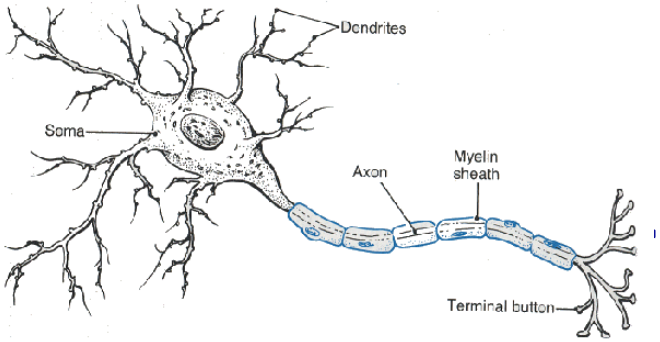
Dip. Informatica
University of Pisa

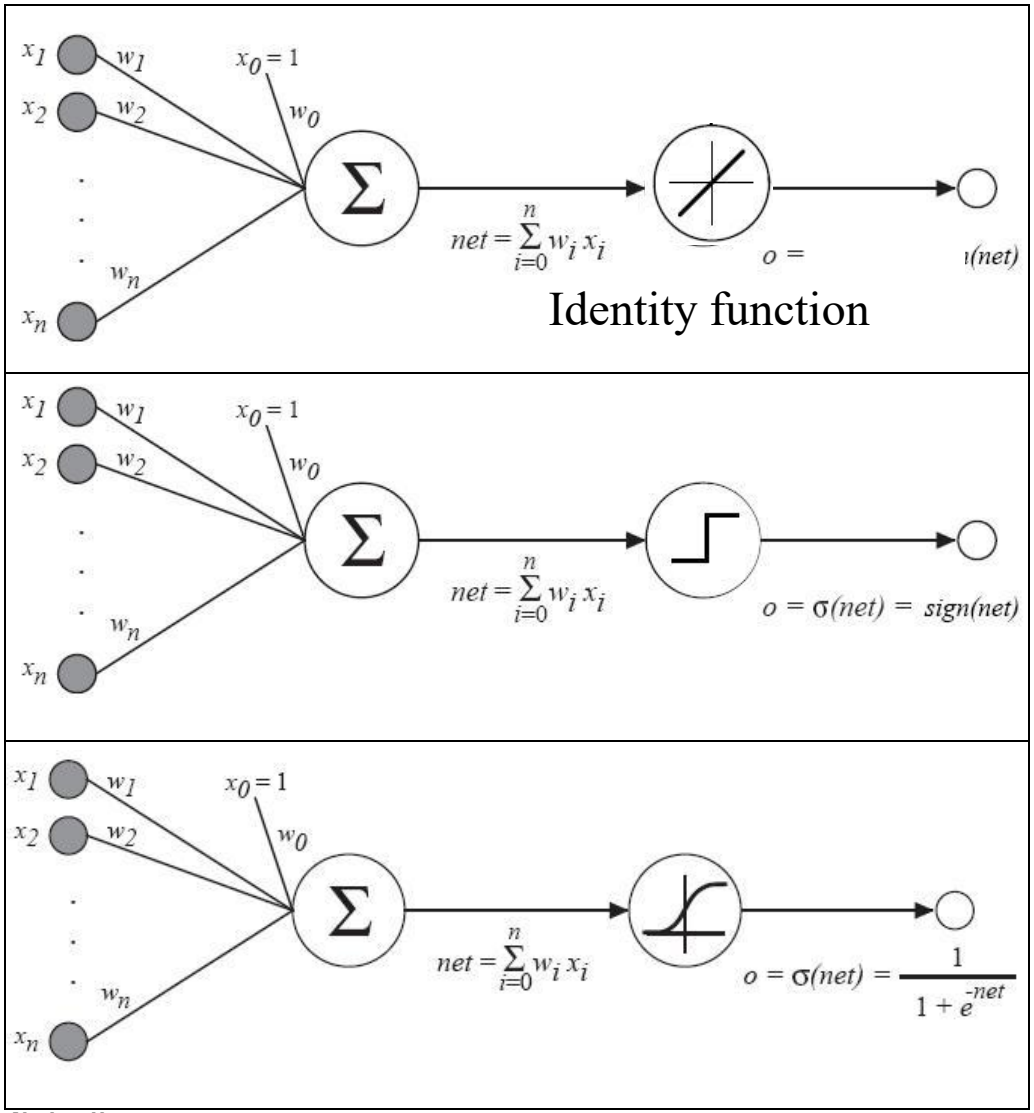
- Some media e.g. Wikpedia (backprogation section) and some NN simulators/libraries use a different notation, whereas, for example w_{2j} (in the blue box) is the weight from input 2 to unit j, instead of using as us the traditional w_{j2} (because w_j belong to unit j)
- Take care: We fix the traditional one provided in the previous slide during our course.



Not used by us

Neuron: Three activation functions





$$h(\boldsymbol{x}) = \sum_i w_i x_i$$

Linear activation function

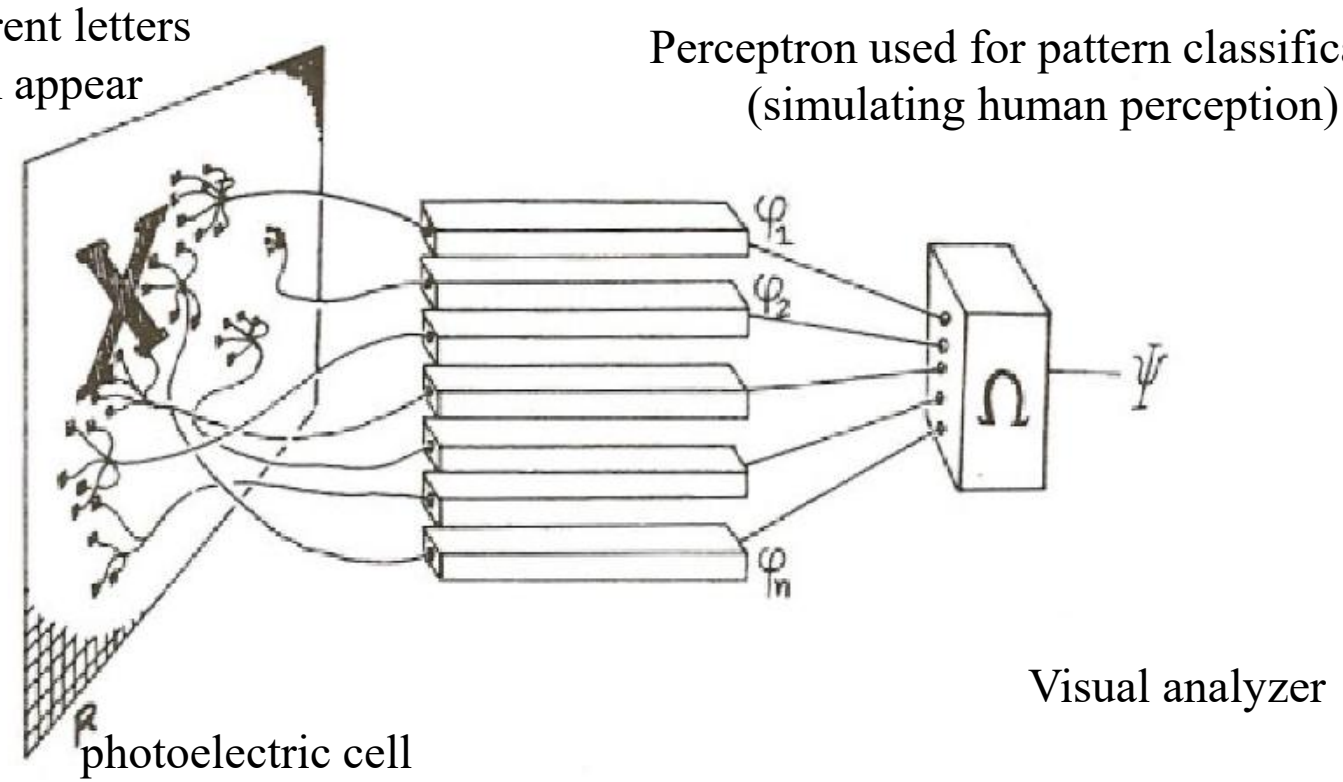
Perceptron (see LTU)
Threshold activation function

Logistic function : sigmoid
(see later)

Perceptron

Dip. Informatica
University of Pisa

- Frank Rosenblatt (1957-1958,1960, ...)



Perceptron II

Dip. Informatica
University of Pisa

- Nice: single neuron is a very simple computational unit
- Minsky: "for the kind of recognition it can do, it is such a simple machine that it would astonishing if nature did not make use of it somewhere"

Hence, we are going to discuss :

- A biologically inspired model with a simple computational unit (and its learning algorithm)
- that is a model of historical importance for the ML, with different approaches since the early 60s

Models with Perceptron

Dip. Informatica
University of Pisa

- Perceptrons can be composed and connected to build a networks:
  *from LTU to NN !!! (**MLP** NN = Multi Layer Perceptron)*
  - Paradigmatic for the transition from linear to non-linear models towards the flexible models at the state-of-the-art in ML
- *First, by an historical model…McCulloch&Pitts*
- Note: actually NN are realized by software (*simulators*) or hardware on chips

McCulloch & Pitts Networks

Dip. Informatica
University of Pisa

- ``A logical calculus of the ideas immanent in nervous activity" 1943
  - What does a given net compute?
  - Can a given net compute a given logical sentence?
- Neurons are in two possible states: firing (1) and not firing (0)
- All synapses (connections) are equivalent and characterized by a real number (their strength w), which is positive for excitatory connections and negative for inhibitory connections;
- a neuron i becomes active when the sum of those connections w_{ij} coming from neurons j connected to it which are active, plus a bias, is larger than zero.
- Binary inputs
- Again binary output (aka binary classification task)

- "Basically" the LTU/Perceptron

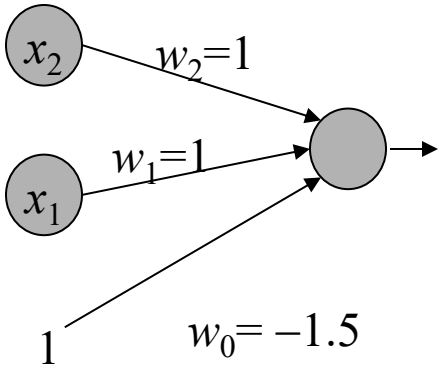
AND, OR Boolean functions

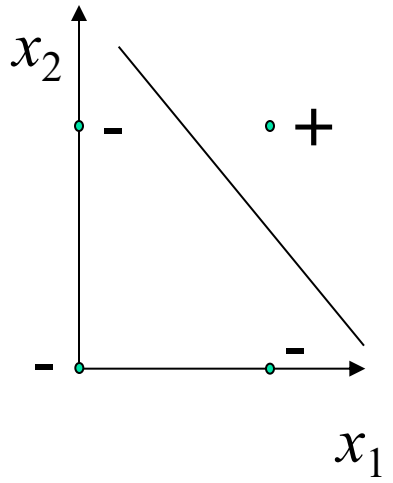
Dip. Informatica
University of Pisa

Assume $net = \mathbf{w}^T\mathbf{x} = \sum_{i=0}^{n} w_i x_i$ $\quad \text{sign}(net) = \begin{cases} 1 \text{ if } net > 0 \\ 0 \text{ (or } -1\text{) otherwise} \end{cases}$

AND

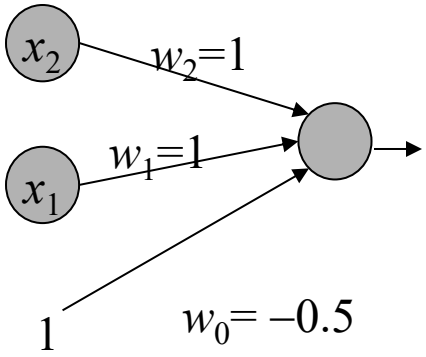
x_1	x_2	$f(x)$
0	0	0
0	1	0
1	0	0
1	1	1

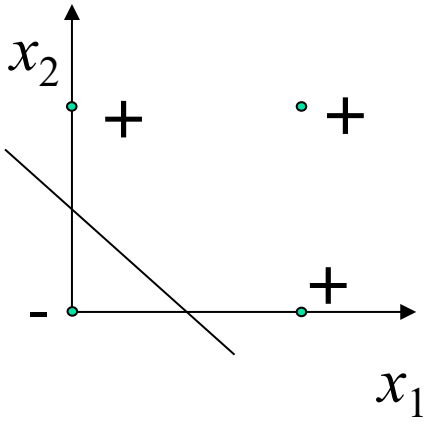




OR

x_1	x_2	$f(x)$
0	0	0
0	1	1
1	0	1
1	1	1



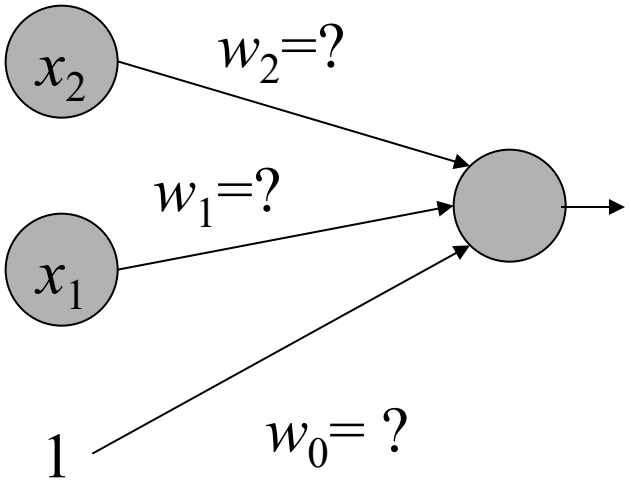


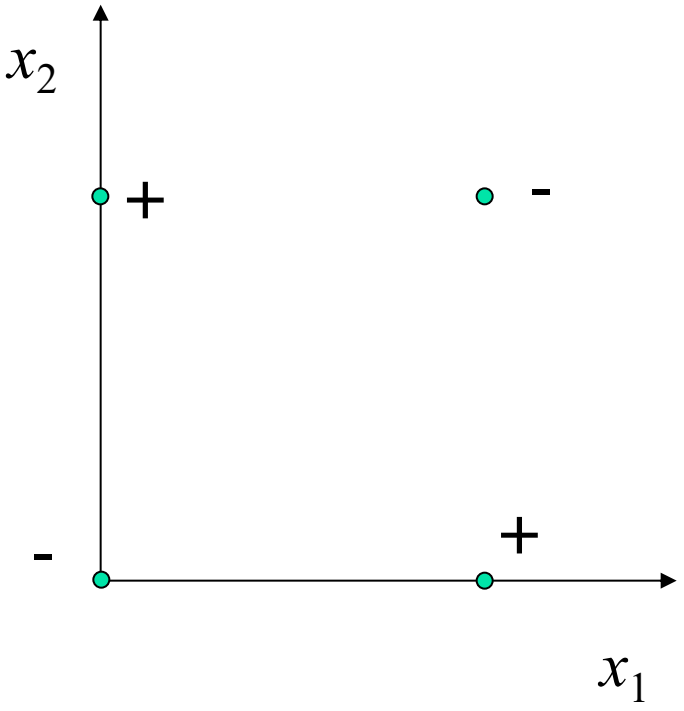
Exercise: find a network for the NOT

Exclusive OR (XOR)

Dip. Informatica
University of Pisa

x_1	x_2	$f(x)$
0	0	0
0	1	1
1	0	1
1	1	0





No linear separation surface exists!!

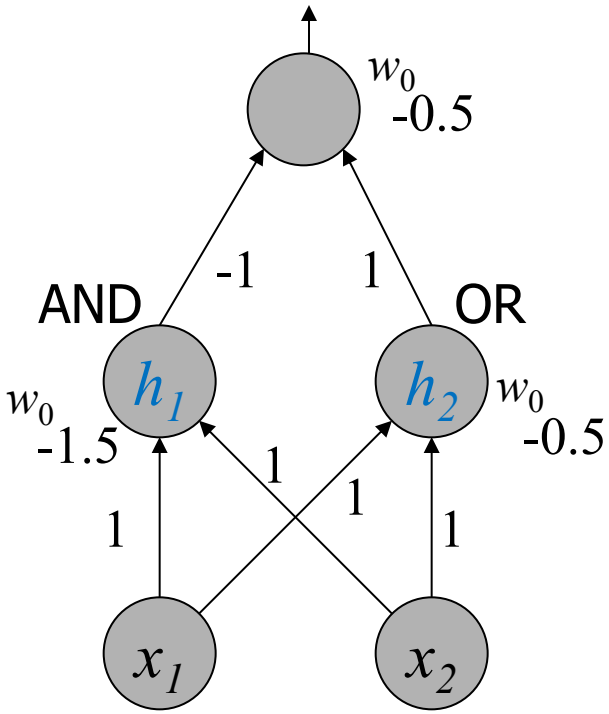
Minsky and Papert (1969)

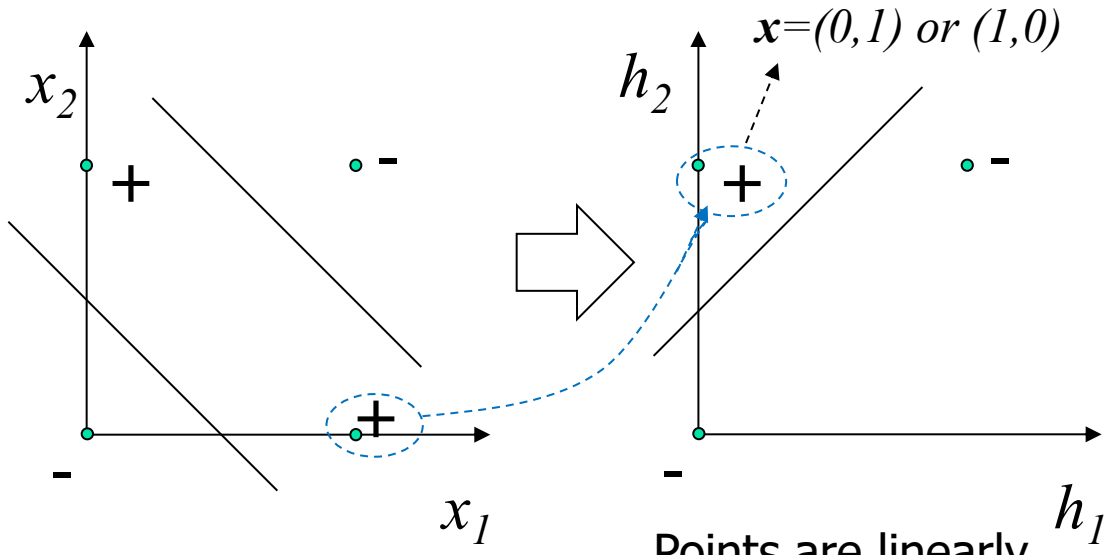
XOR by a Two Layers Network

Dip. Informatica
University of Pisa

$$x_1 \oplus x_2 = x_1 \cdot \overline{x_2} + \overline{x_1} \cdot x_2 \qquad \text{let :} \quad h_1 = x_1 \cdot x_2 \quad \text{(dot = **and**)} \qquad h_2 = x_1 + x_2 \quad \text{(+ = **or**)}$$

$$\text{then we have : } x_1 \oplus x_2 = \overline{h_1} \cdot h_2 \quad \text{(next slide)}$$





E.g. (1,0) becames h_1=0 h_2=1

Points are linearly separable in the *new space!*

Exercise: separate by other planes→ find different networks solving XOR

Detalis to solve the XOR (proof)

Dip. Informatica
University of Pisa

- DE MORGAN rule : ***not***$(a$ ***and*** $b) =$ ***not*** (a) ***or not***(b)

- Hence, if $h_1 = (a$ ***and*** $b)$ → ***not***$(h_1)=($***not***(a) ***or not***$(b))$
- And if $h_2 = (a$ ***or*** $b)$ →

not(h_1) ***and*** $h_2 =$ ***(not***(a) ***or not***$(b))$ ***and*** $(a$ ***or*** $b)=$
$= ($~~***not***(a) ***and*** $a)$ ***or*** $($***not***(a) ***and*** $b)$ ***or*** $(a$ ***and not***$(b))$ ***or***
$($~~b ***and not***$(b))$~~ $=$
$=$***(not***(a) ***and*** $b)$ ***or*** $(a$ ***and not***$(b)) =$ *(by* ***xor*** *def.)*
$= a$ ***xor*** b *(C.V.D.)*

Exercise: redo by yourself with x, +, etc

Hidden layer & Representation

Dip. Informatica
University of Pisa

- Useful concept of internal **(re)representation** of input variables via internal (hidden) units
- Developing *high level (hidden) features* is a key factor in NN
- The representation in the hidden layer makes easier the task to the output layer (last unit).
  - E.g. See the figure on the right: point *(1,0)* (in the sud-est corner) becames $h_1=0\ h_2=1$ (north-west corner) → now it is a linear separable problem

A look ahead:

- Such ***composition*** of sub-/intermediate operations to perform a complex task can be extended through *many layers* of abstraction
- In NN such internal representation/intermediate features can be ***learned***
- Learning internal distributed representation → *representation learning* and *deep learning* concepts (link to next parts of the ML course)

McCulloch&Pitts Summary

Dip. Informatica
University of Pisa

- Properties of Networks of Perceptrons:

- Perceptrons can represent AND, OR, NOT (NAND, NOR)
- → every Boolean function can be represented by some network of interconnected perceptrons.

- Only **two** levels deep is sufficient (or more can be more efficient*).
- … but **single** layer cannot model all the possible functions (Minsky&Papert 1969): limitations of single perceptrons due to the linear separable problems!

- Note: **no** provided (up to now) learning algorithm (even for one unit)!!!!
- Let us start (again) with a single unit model

Learning for one unit model - Overview

Dip. Informatica
University of Pisa

Two kinds of method for learning (also historical view):

1. **Adaline** = Adaptive Linear Neuron (Widrow, Hoff): linear unit during training: LMS direct solution and gradient descent solution
   - Regression tasks: See the LMS algorithm
   - For classification: See the LTU and LMS algorithm of a previous lectures (on linear model)
   - An approach that we will generalize to MLP

2. **Perceptron** (Rosenblatt): non-linear unit during training: with hard limiter or Threshold activation function
   - Only classification: Capabilities studies, convergence theorem
   - Next slides

Introduction to 2.: the Perceptron Learning Alg.

Dip. Informatica
University of Pisa

- Rosenblatt's Perceptron Learning Algorithm (1958-1962)

- Minimize the number of misclassified patterns:
  - find $\boldsymbol{w}$ s.t. $sign(\boldsymbol{w}^T\boldsymbol{x}) = d$
  - On-line algorithm: a step can be made for each input pattern
  - Note that to build linear classifier we will have from now **3 learning alg:**
    1. Adaline LMS direct solution
    2. Adaline LMS gradient based alg.
    3. Perceptron learning algorithm (now)
    - And others will come (later in the course, e.g. SVM)

The "Perceptron Learning Algorithm"

Dip. Informatica
University of Pisa

1. Initialize the weights (either to zero or to a small random value)
2. pick a learning rate η (this is a number between 0 and 1)
3. until stopping condition is satisfied (e.g. weights don't change):

For each training pattern $(\boldsymbol{x}, d)$ $(d = +1 \text{ or } -1)$ [let out be the output]:

- Compute output activation $out = sign(\boldsymbol{w}^T \boldsymbol{x})$ $[+1,-1]$

- If $out = d$, don't change weights I.e. "minimize (only) misclassifications"
- If $out \neq d$, update the weights:
  - $\boldsymbol{w}_{new} = \boldsymbol{w} + \eta\ d\ \boldsymbol{x}$

add $+\ \eta\, \boldsymbol{x}$ if $\boldsymbol{w}^T\boldsymbol{x} \leq 0$ and $d=+1$
$-\ \eta\, \mathbf{x}$ if $\boldsymbol{w}^T\boldsymbol{x} > 0$ and $d=-1$

Or (in a different form):

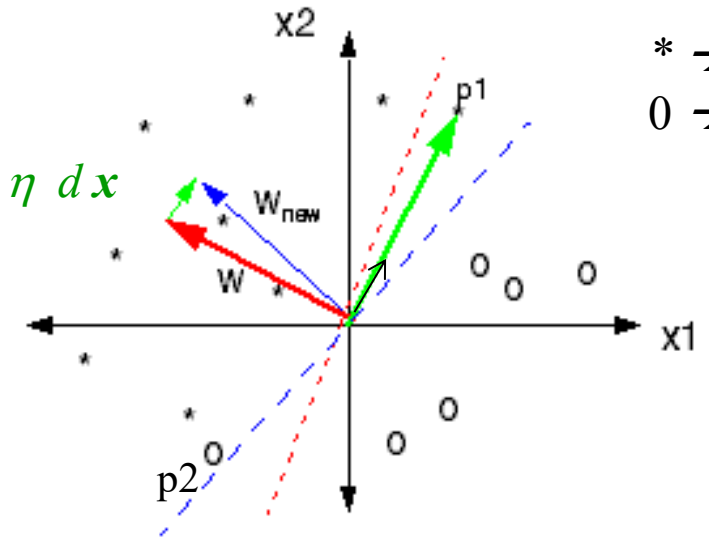
- $\boldsymbol{w}_{new} = \boldsymbol{w} + 1/2\,\eta\ (d\text{-}out)\ \boldsymbol{x}$

Note: Δw *in LMS was different because we have here the sign() in the out computation*

Geometrical view

Dip. Informatica
University of Pisa

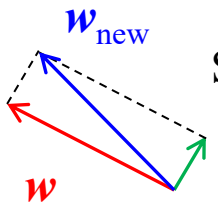
An example: Before updating the weight $\boldsymbol{w}$ (pointing the positive region), we note that both p1 and p2 are incorrectly classified (the red dashed line is decision boundary).
Suppose we choose p1 to update the weights as in picture below.
p1 has target value d=1, so that $\boldsymbol{w}$ is moved a small amount in the direction of p1.
The new boundary (blue dashed line) is better than before (and $\boldsymbol{w}_{new}$ closer to p1).
Exercise: what happens if the training pattern p2 is chosen ?



* → d=+1
0 → d= -1

$$\boldsymbol{w}_{new} = \boldsymbol{w} + \eta\ d\ \boldsymbol{x}$$

Note: only for this example (w_0) bias =0



Sum of vectors

Note the *Hebbian rule*: $\boldsymbol{w}$ moves towards $\boldsymbol{x}$

Delta Rule

Dip. Informatica
University of Pisa

- The form $\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + \eta\ d\,\boldsymbol{x}$ is seen as *Hebbian learning*
- The other form $\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + \eta\ (d\text{-}out)\,\boldsymbol{x} = \boldsymbol{w} + \eta\,\delta\,\boldsymbol{x}$

  is in the form of *error-correction* learning

- Recall from LMS: This is an "*error correction*" rule (delta**/** Widrow-Hoff**/Adaline/LMS rule**) that changes the $\boldsymbol{w}$ proportionally to the error (target d -output):
  - E.g. (target d – output) = err=0 → no correction
  - (input>0) if err + (output is too low), increase $\boldsymbol{w}$ → incr. $\boldsymbol{w}^T\boldsymbol{x}$ → reduce err
  - …
- In terms of "neurons": *the adjustment made to a synaptic weight is proportional to the product of error signal and the input signal that excite the synapse*
- Easy to compute when errors signal δ is directly measurable (we know the desired response for each unit)

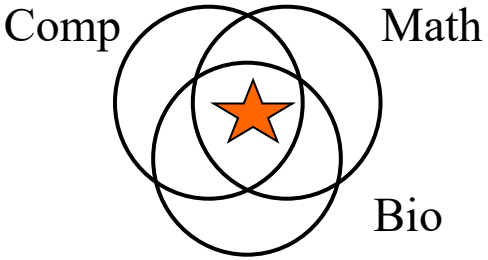
Represent / Learn

Dip. Informatica
University of Pisa

- What can Perceptron represent? linear decision boundaries (see LTU)
  - Perceptron can solve linearly separable problems
- Can it always learn the solution?
- Yes! The Perceptron with the "Perceptron Learning Algorithm" is always able to learn what it is able to represent:

Perceptron Convergence Theorem

Proof in the next lesson!



Note: A milestone results!!!! A biologically (Bio) inspired model with well-defined and proved computational capabilities (Comp)!!! And proved by a theorem (Math)

Perceptron Convergence Theorem

Dip. Informatica
University of Pisa

- *The perceptron is guaranteed to converge (classifying correctly all the input patterns) in a finite number of steps if the problem is linearly separable.*
- (independently of the starting point, although the final solution is not unique and it depends on the starting point)

- May be "unstable" if the problem is not separable
  - In particular it develop cycles (with a set of weights that are not necessarily optimal)

Note: I'll simply the notations in the proof (e.g. assume the *not* reported scalar product with ^{T} as usual among vectors)

Perceptron Convergence Theorem: preliminaries (1)

Dip. Informatica
University of Pisa

We can focus on a "all positive patterns" task (as shown in the following):

Assume $(\boldsymbol{x}_i, d_i)$ in the TR set, with $d_i = +1 \text{ or } -1 \text{ and } i=1\ldots l$

Linearly separable → $\exists\ \boldsymbol{w}^*$ solution s.t. $(\text{omitting } ^T)$

$$d_i\ (\boldsymbol{w}^* \boldsymbol{x}_i) \geq \alpha, \text{ with } \alpha = \min_i d_i\ (\boldsymbol{w}^* \boldsymbol{x}_i) > 0$$

Hence, $\boldsymbol{w}^* (\ d_i\ \boldsymbol{x}_i) \geq \alpha$

Defining $\boldsymbol{x}_i' = (d_i\ \boldsymbol{x}_i)$, then $\boldsymbol{w}^*$ is a solution iff

$\boldsymbol{w}^*$ is a solution of $(\boldsymbol{x}_i',\ +1)$

(if) $\boldsymbol{w}^*$ solve → $d_i\ (\boldsymbol{w}^* \boldsymbol{x}_i) \geq \alpha$ → $(\boldsymbol{w}^*\ d_i \boldsymbol{x}_i) \geq \alpha$ → $(\boldsymbol{w}^* \boldsymbol{x}_i') \geq \alpha$ → $\boldsymbol{w}^*$ is a solution of $(\boldsymbol{x}_i',\ +1)$

(only if) $\boldsymbol{w}^*$ is a solution of $(\boldsymbol{x}_i',\ +1)$ → $(\boldsymbol{w}^*\ d_i \boldsymbol{x}_i) \geq \alpha$ → $d_i\ (\boldsymbol{w}^* \boldsymbol{x}_i) \geq \alpha$ → $\boldsymbol{w}^*$ solve for $\boldsymbol{x}_i$

Perceptron Convergence Theorem: preliminaries (2)

Dip. Informatica
University of Pisa

Assuming $\boldsymbol{w(0)=0}$ *(at step 0),*

η =1,

$\beta = max_i ||\boldsymbol{x}_i||^2$ (*i=1…l, and* || || denotes the Euclidian norm, see first lesson)

After q errors (all false negative)

$$\boldsymbol{w}(q) = \sum_{j=1 \rightarrow q} \boldsymbol{x}_{(i_j)}$$

i_j is used to denote the patterns belonging to the subset of misclassified patterns e.g., the indices 2, 8, 9,14 in the TR set.

Because $\boldsymbol{w(j)} = \boldsymbol{w(j-1)} + \boldsymbol{x}_{i_j}$

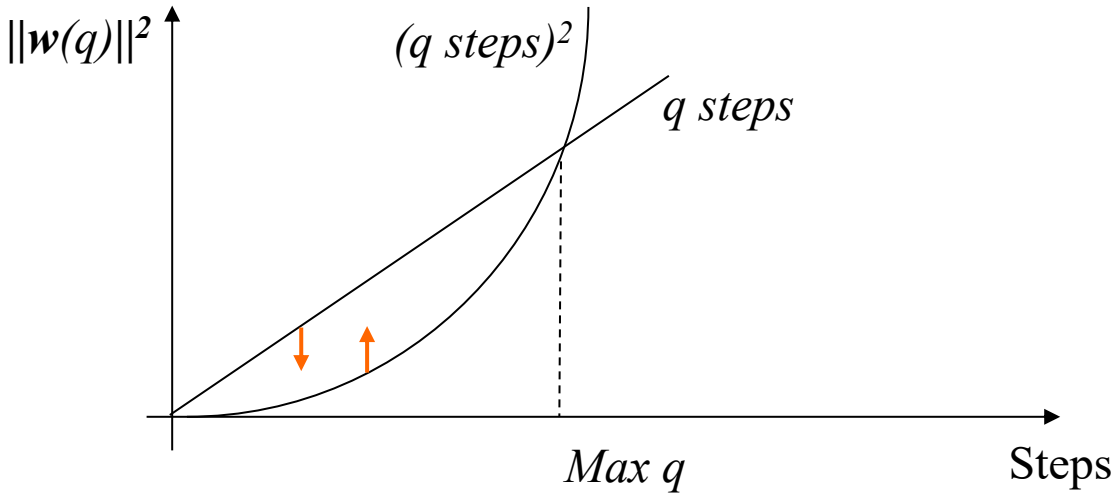
Rule: $\boldsymbol{w}_{new} = \boldsymbol{w} + \eta\ d\ \boldsymbol{x}$ (but all d are +1 here, only positive patterns, and $\eta=1$)

Perceptron Convergence Theorem: Proof

Dip. Informatica
University of Pisa

Theorem "For linearly separable tasks, the perceptron algorithm converges after a finite number of steps"

Basic idea: we can find lower and upper bound to $\|w(q)\|^2$ as a function of q^2 steps (lower bound ↑) and q steps (upper bound ↓) → we can find q (number of steps) s.t. the algorithm converges:
[Proof also in Haykin 2nd ed. pages 139-141]



Perceptron Convergence Theorem: Proof (part 1)

Dip. Informatica
University of Pisa

- Lower bound on $\|w(q)\|^2$:

$$(\boldsymbol{w}^*)^T \boldsymbol{w}(q) = (\boldsymbol{w}^*)^T \sum_{j=1 \to q} \mathbf{x}_{i_j} \geq q\alpha$$

Recall that :

- $\boldsymbol{w}^*$ is the solution
- $\alpha = min_i \ ((\boldsymbol{w}^*)^T \boldsymbol{x}_i)$

Cauchy-Swartz: $(\boldsymbol{a}^T \boldsymbol{b})^2 \leq \|\boldsymbol{a}\|^2 \|\boldsymbol{b}\|^2 \implies \|\boldsymbol{a}\|^2 \|\boldsymbol{b}\|^2 \geq (\boldsymbol{a}^T \boldsymbol{b})^2$

Use $\boldsymbol{a} = \boldsymbol{w}^*$ and $\boldsymbol{b} = \boldsymbol{w}(q)$

$$\|\boldsymbol{w}^*\|^2 \|\boldsymbol{w}(q)\|^2 \geq ((\boldsymbol{w}^*)^T \boldsymbol{w}(q))^2 \geq (q\alpha)^2$$

$$|\boldsymbol{w}(q)|^2 \geq (q\alpha)^2 / \|\boldsymbol{w}^*\|^2$$

The lower bound

Proof (part 2)

Dip. Informatica
University of Pisa

$$||\mathbf{a}+\mathbf{b}||^2 = \sum_i (a_i + b_i)^2 = \sum_i a_i^2 + \sum_i 2a_i b_i + \sum_i b_i^2 = ||\mathbf{a}||^2 + 2\mathbf{ab} + ||\mathbf{b}||^2$$

- Upper bound on $||\boldsymbol{w}(q)||^2$:

$$||\boldsymbol{w}(q)||^2 = ||\boldsymbol{w}(q-1) + \mathbf{x}_{i_q}||^2 = ||\boldsymbol{w}(q-1)||^2 + 2\boldsymbol{w}(q-1)\mathbf{x}_{i_q} + ||\mathbf{x}_{i_q}||^2$$

For the *q-th error: this is* <0, hence, deleting a negative term:

$$||\boldsymbol{w}(q)||^2 \leq ||\boldsymbol{w}(q-1)||^2 + ||\mathbf{x}_{i_q}||^2$$

And by iteration $(\boldsymbol{w}(0)=0)$

$$||\boldsymbol{w}(q)||^2 \leq \sum_{j=1->q} ||\mathbf{x}_{i_j}||^2 \leq q\beta$$

$$\beta = max_i ||\boldsymbol{x}_i||^2$$

The upper bound

Proof (part 3): bring them together

Dip. Informatica
University of Pisa

Upper bound Lower bound

$$q\beta \geq ||\boldsymbol{w}(q)||^2 \geq (q\alpha)^2/||\boldsymbol{w}^*||^2$$

$$q\beta \geq q^2\alpha'$$

$$\beta \geq q\alpha'$$

$$\boxed{q \leq \beta/\alpha'}$$

i.e. a finite number of steps

Differences between "Perc. Learning Alg." and LMS Alg. (I)

Dip. Informatica
University of Pisa

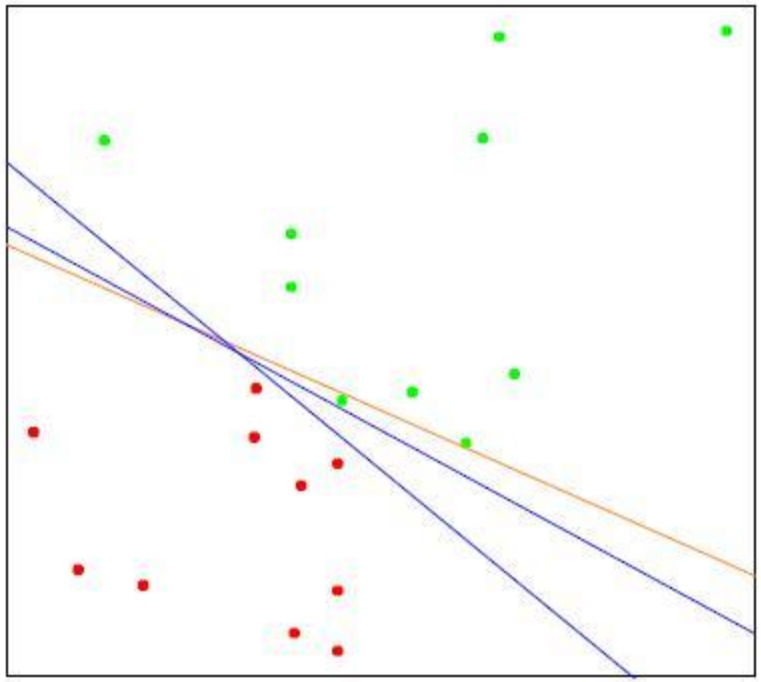
$$\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + \eta\ (d\text{-}out)\ \boldsymbol{x}$$

- Apparently similar but note that:
  - LMS rule was derived (by the gradient) without threshold activation functions: minimization of the error of the linear unit (using directly $\boldsymbol{w}^T\boldsymbol{x}$)
  - The perceptron use $out = sign(\boldsymbol{w}^T\boldsymbol{x})$
- Hence, for training:
  - $\delta = (d - \boldsymbol{w}^T\boldsymbol{x})$ for the LMS approach
    versus
  - $\delta = (d - sign(\boldsymbol{w}^T\boldsymbol{x}))$ for the perceptron learning alg.
- Of course the model trained with LMS can still be used for classification applying the threshold function $h(\boldsymbol{x}) = sign(\boldsymbol{w}^T\boldsymbol{x})$ (LTU)

LMS can misclassify in linear separable problems

Dip. Informatica
University of Pisa

- LMS not necessarily minimize the number of TR examples misclassified by the LTU



$$E(\mathbf{w}) = \sum_{p} (y_p - \mathbf{x}_p{}^T \mathbf{w})^2$$

It changes the weights **not only** for the misclassified patterns!

FIGURE 4.14. *A toy example with two classes separable by a hyperplane. The orange line is the least squares solution, which misclassifies one of the training points. Also shown are two blue separating hyperplanes found by the perceptron learning algorithm with different random starts.*

Differences Summary

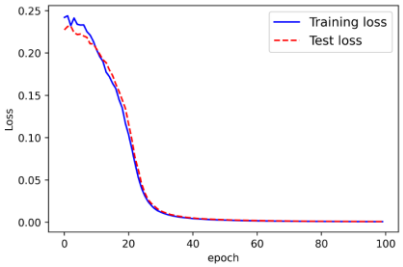
Dip. Informatica
University of Pisa

Perceptron Learn. Alg.

- Min. misclassifications ($out = sign(\mathbf{w}^T \mathbf{x})$)
- it always converges for linear separable problems (in a finite number of steps) to a perfect classifier
- (else no convergence)
- difficult to be extended to networks of units (NN)

LMS Alg.

- Min. $E(\mathbf{w})$ with $out = \mathbf{w}^T \mathbf{x}$
- asymptotic convergence (*), also for not linear separable problems
- not always zero classification errors, linear separable problems
- it can be extended to networks of units (NN), using the gradient based approach

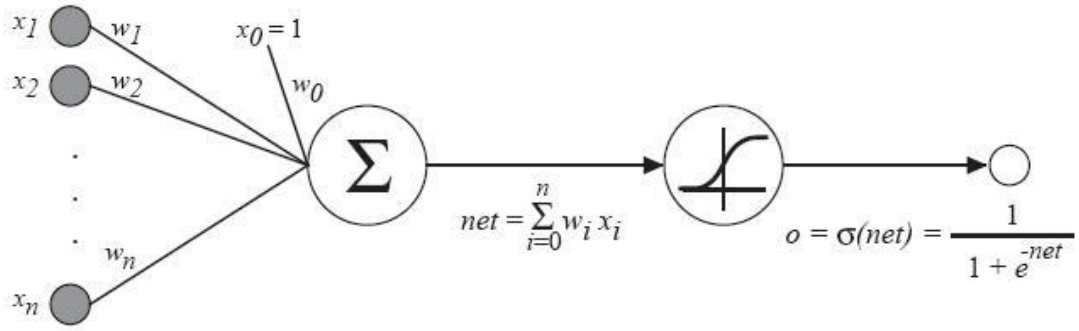


(*) example of *asymptotic* convergence (training with MSE loss)

Activation functions

Dip. Informatica
University of Pisa

1. Linear function
2. Threshold (or step) function (Perceptron/LTU)
3. A non-linear *squashing* function like the **sigmoidal** logistic function: assumes a continuous range of values in the bounded interval [0,1]



$$f_\sigma(x) = \frac{1}{1 + e^{(-ax)}}$$

- The sigmoidal-logistic function has the property to be a *smoothed differentiable threshold function*.
- a is the *slope* parameter of the sigmoid function

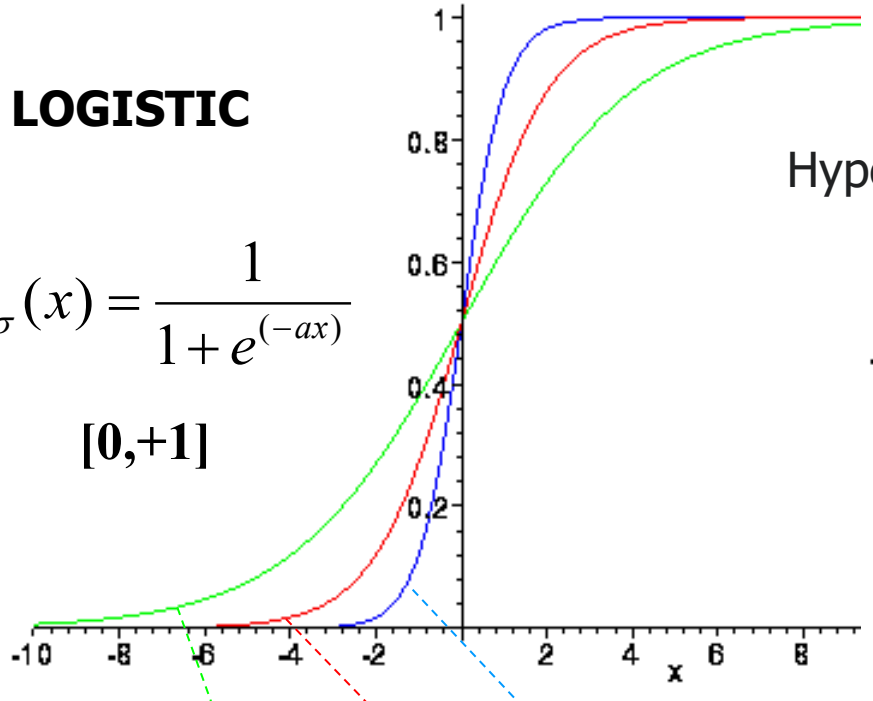
Activation functions (type 3.)

Dip. Informatica
University of Pisa

LOGISTIC

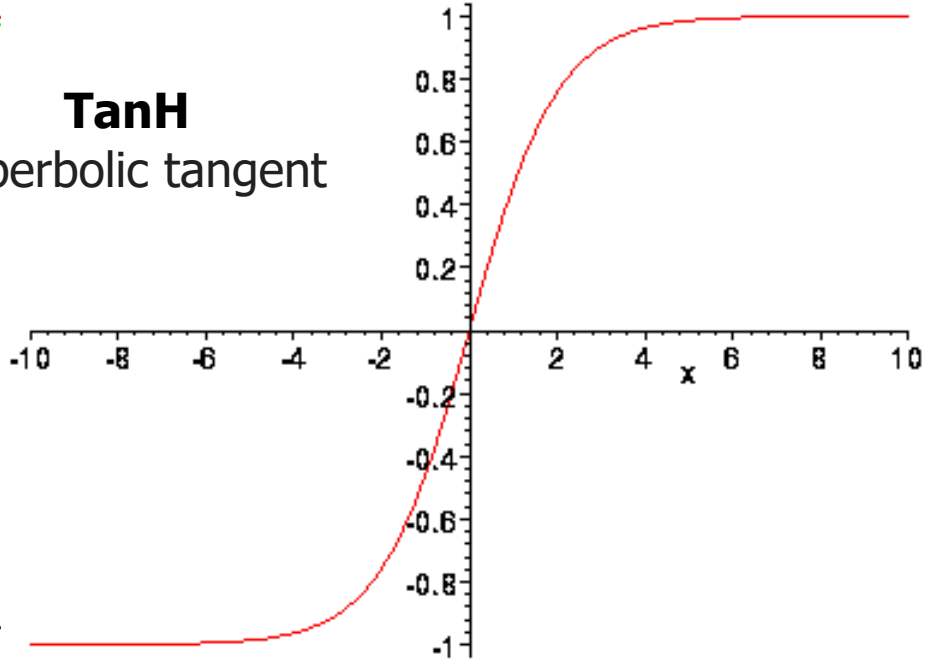
$$f_\sigma(x) = \frac{1}{1+e^{(-ax)}}$$

[0,+1]



a=0.5 green, *a*=1 red, *a*=2 blue

TanH
Hyperbolic tangent



$f_{symm} = 2\, f_\sigma(x) - 1 = tanh(ax/2)$

[-1,+1]

Semilinearity close to 0

What if $a \rightarrow 0$? (to linear)

$a \rightarrow \infty$? (to LTU)

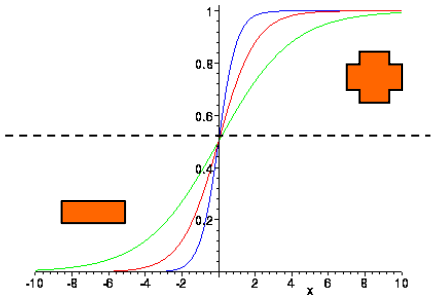
Sigmoidal functions and classifier outcomes

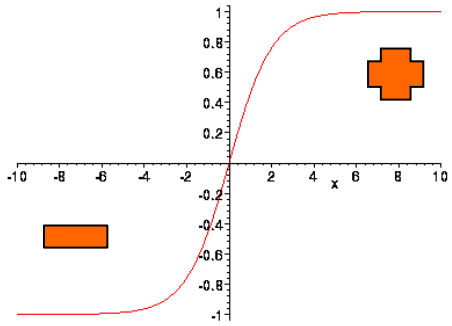
Dip. Informatica
University of Pisa

These functions provide continues outputs but…

- For the Logistic function an output value
  - ≥0.5 (**threshold**) correspond to the **positive class**
  - <0.5 correspond to the **zero or negative class**
- It is possible to change this threshold (e.g. studying the effect on FP/FN etc, or by a ROC),
- and even to consider a *rejection zone* in an interval around the threshold value (to avoid fragile decisions)

- For the TanH the **threshold is in 0** (with the analogues possibilities)





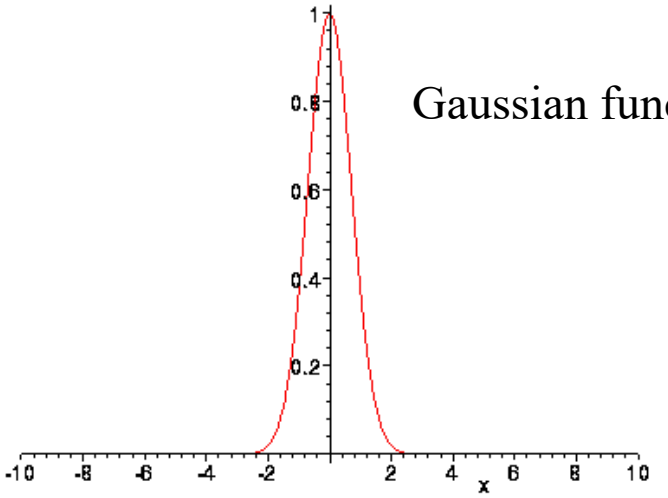
Other Activation Functions

Dip. Informatica
University of Pisa

- Radial Basis Functions → *RBF networks*

$$f(x) = e^{(-ax^2)}$$

$$\left\| \boldsymbol{w} - \boldsymbol{x}_{input} \right\|$$



Gaussian function

- Softmax: see next lectures (multi-output case)
- Stochastic neurons: output is +1 with probability $P(net)$ or –1 with $1-P(net)$→ *Boltzmann machines and other models rooted in statistical mechanics*

Other Activation Functions (2)

Dip. Informatica
University of Pisa

- Tanh-like (piecewise linear approximation) for efficient computation

$$f(x) = \text{sign}(x)\left[1 + \frac{1}{2^{\lfloor 2^n |x| \rfloor}}\left(\frac{2^n |x| - \lfloor 2^n |x| \rfloor}{2} - 1\right)\right]$$

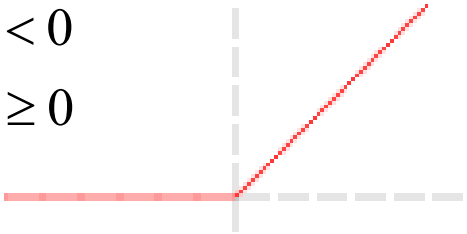
n steepness (slope) of the function
Rounding operator (floor)
Shift operation in binary rep.

- Rectifier →**ReLU** (Rectified Linear Unit): see *Deep Learning* section

It has become a default choice for Deep models, so it (and its variants) deserves more attention discussing Deep Learning

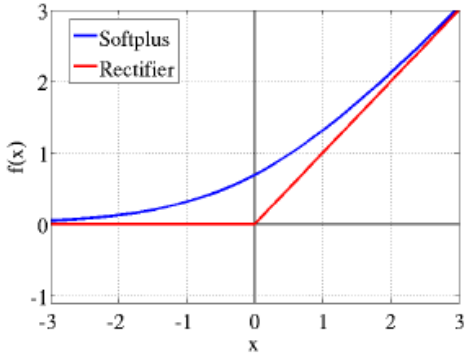
$$f(x) = \begin{cases} 0 & for \quad x < 0 \\ x & for \quad x \geq 0 \end{cases}$$

$$f(x) = \max(0, x)$$



- **Softplus** (smooth approximation)

$$f(x) = \ln(1 + e^x)$$



- An updated list (Note: many perform comparably!):

https://en.wikipedia.org/wiki/Activation_function

Activation functions: derivatives

Dip. Informatica
University of Pisa

- The derivative of the **identity function** is 1.
- The derivative of the **step** (threshold) **function** is not defined, which is exactly why it isn't used with LMS .
- **Sigmoids**: for asymmetric and symmetric case we have ($a=1$):

$$\frac{df_\sigma(x)}{dx} = f_\sigma(x)(1-f_\sigma(x)) \qquad \frac{df_{\tanh}(x)}{dx} = 1-f_{\tanh}(x)^2$$

Show by exercise: also with any a

LMS with f_σ

Dip. Informatica
University of Pisa

- The sigmoidal-logistic function has the property to be a *smoothed differentiable threshold function*

- Hence, we can derive a Least (Mean) Square algorithm by computing the gradient of the mean square loss function as for the linear units (also for a classifier)

- From $o(\mathbf{x}) = \mathbf{x}^T\mathbf{w}$ to $o(\mathbf{x}) = f_\sigma(\mathbf{x}^T\mathbf{w})$ *where* f_σ *is a logistic function*
- Find $\mathbf{w}$ to *minimize* the residual sum of squares:

$$E(\mathbf{w}) = \sum_p (d_p - o(\mathbf{x}_p))^2 = \sum_p (d_p - f_\sigma(\mathbf{x}_p{}^T\mathbf{w}))^2$$

LMS with 1 non-linear unit
Compute the gradient

Dip. Informatica
University of Pisa

Important *Exercise*: compute the gradient of the error function

(do it before the next lecture)

$$\frac{\partial E(\boldsymbol{w})}{\partial w_j} \qquad \text{with } o(\boldsymbol{x}_p)=out(\boldsymbol{x}_p)=f_\sigma(\boldsymbol{x}_p{}^T\boldsymbol{w})=f_\sigma(net(\boldsymbol{x}_p))$$

- directly (by composition of functions)

- or by the chain rule $\quad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$

Whereas E is f and w is x, using *first:* auxiliary $g = net$, *and then* auxiliary $g = out$

- Suggestion: compute it starting with the singular E_p and sum the derivatives to obtain the total for l data

- Interpretation: This derivatives answers to: *How much variations of w affects variation of E* ?

LMS with 1 non-linear unit Compute the gradient

Dip. Informatica
University of Pisa

Important *Exercise*: compute the gradient of the error function

(do it before the next lecture)

$$\frac{\partial E(\boldsymbol{w})}{\partial w_j} \quad \text{with } o(\boldsymbol{x}_p)=out(\boldsymbol{x}_p)=f_\sigma(\boldsymbol{x}_p{}^T\boldsymbol{w})=f_\sigma(net(\boldsymbol{x}_p))$$

Result (1 unit, all the patterns $p = 1..l$):

$$\frac{\partial E(\boldsymbol{w})}{\partial w_j} = -2\sum_{p->l}(\boldsymbol{x}_p)_j(d_p - f_\sigma(\boldsymbol{x}_p{}^T\boldsymbol{w}))f'_\sigma(\boldsymbol{x}_p{}^T\boldsymbol{w}) = -2\sum_{p->l}(\boldsymbol{x}_p)_j \boxed{\text{new } \delta_p \quad \delta_p f'_\sigma(net(\boldsymbol{x}_p))}$$

We can simplify the notation as:

$$\frac{\partial E(\boldsymbol{w})}{\partial w_j} = -2\sum_{p=1}^{l} \boxed{\text{new } \delta_p \quad (d_p - o(\boldsymbol{x}_p))f'_\sigma}\, x_{p,j}$$

Note: here we have just 1 unit, more indices will be used with more units, see backprop.

Gradient descent algorithm

Dip. Informatica
University of Pisa

- The same as for linear unit using the *new delta rule* (batch/on-line):

$$\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + \eta \ \delta_p \boldsymbol{x}_p$$

Pattern p
The unique Unit has not index here

$$(d_p - f_\sigma(net(\boldsymbol{x}_p)))f'_\sigma(net(\boldsymbol{x}_p))$$

…or simplifying the notation

$$(d_p - o(\boldsymbol{x}_p))f'_\sigma$$

Gradient descent algorithm as an error correction rule

Dip. Informatica
University of Pisa

- The same as for linear unit using the *new delta rule* (batch/on-line):

$$\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + \eta\ \delta_p\ \boldsymbol{x}_p \qquad \boxed{\delta_p = (d_p - o(\boldsymbol{x}_p))f'_\sigma}$$

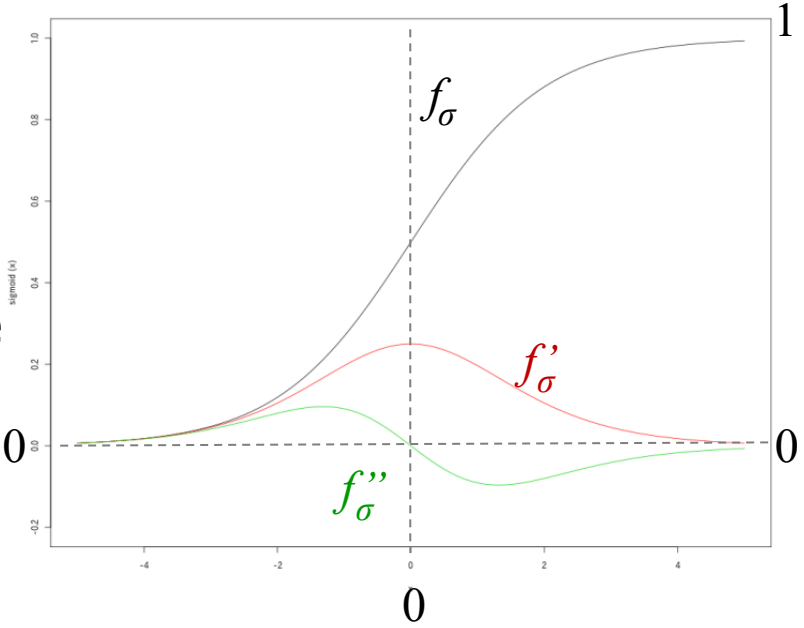
Again, an **error correction rule**. Moreover:

• The parameters a (slope of f) can affect the step of gradient descent

• Max of f_σ' for inputs (*net*) close to 0 (~ linear unit). Here high value of the δ are possible

• Minimum of f_σ' are for **saturated cases** (where f go to 0 or 1 asymptotically) → better to avoid a premature saturation, small δ, and hence very slow changes of $\boldsymbol{w}$, at the beginning (starting with small weights) or later

• Insight: we see also a bridge between the two alg.s: toward no corrections for correct outputs also for LMS alg. (as it was for the perceptron learn. alg.).
I.e. Using f_σ we are approximating not only the LTU but also the perceptron learn. alg.



Plot of f_σ (black) f'_σ (red) f''_σ (green)

Neural Networks (NN)

Dip. Informatica
University of Pisa

All the ingredients are ready:
NN by two views of Multi-Layer Perceptron (MLP)-NN:

A. A network of interconnected units

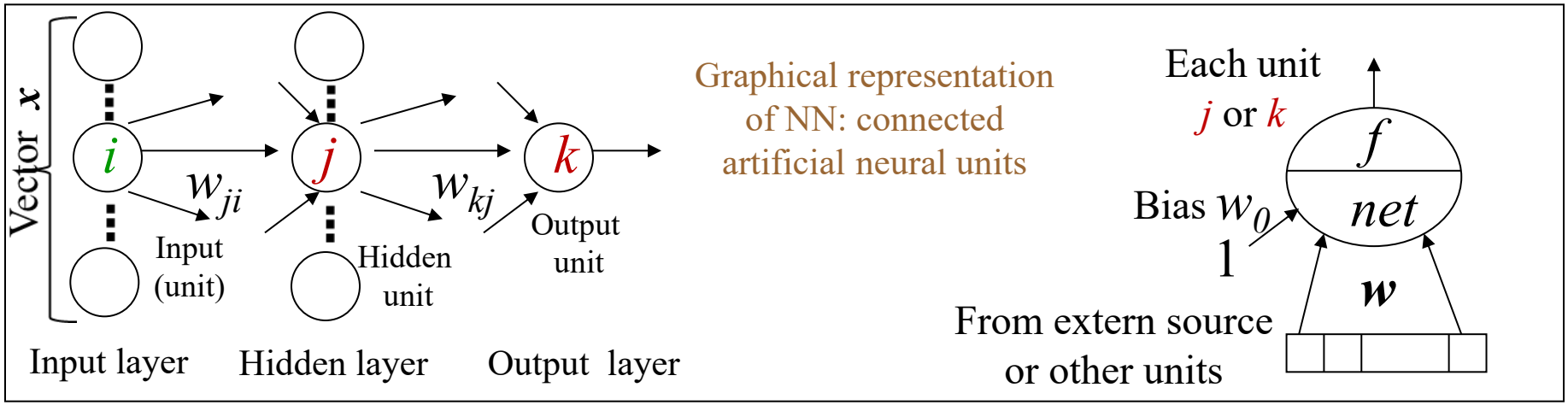
B. A flexible function $h(x)$ (as nested non-linear functions)

The Neural Network (NN)

Standard feedforward NN (with one hidden layer)

Dip. Informatica
University of Pisa

A. A network of units



In a **MLP** (Multi Layer Perceptron) architecture

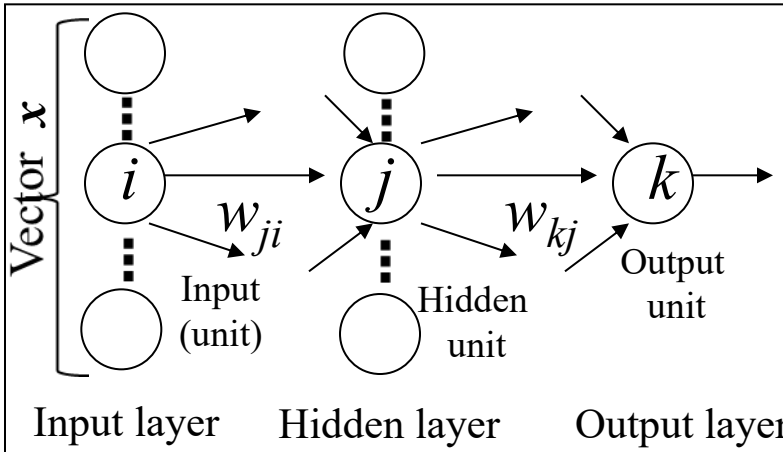
- the units are connected by **weighted links** and they are organized in the form of **layers**
- the **input layer** is simply the source of the input ***x*** that projects onto the hidden layer of units: it loads (copy) the (extern) input patterns ***x*** (the input units *i* do not compute the *net* and *f*)
- the **hidden layer** projects onto the **output layer** (feedforward computation of this two-layers network in the fig.) or to another hidden layer

Neural Networks: two views

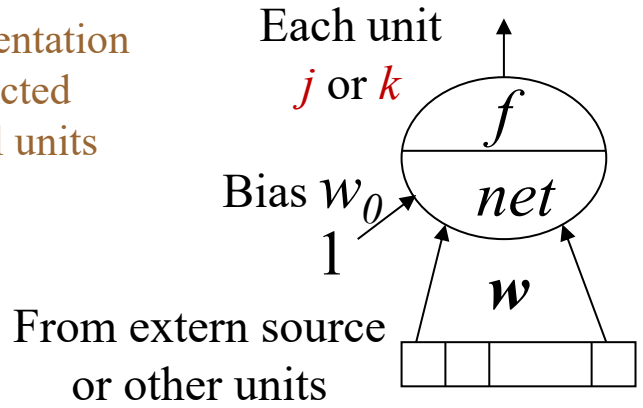
Standard feedforward NN (with one hidden layer)

Dip. Informatica
University of Pisa

A. A network of units



Graphical representation of NN: connected artificial neural units



B. A flexible function

$$h(\boldsymbol{x}) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$$

Free parameters (weight matrix)

Non-linear function (sigmoid)

Input variables (vector $\boldsymbol{x}$)

Recall of a linear function

$$h(\boldsymbol{x}) = \sum_i w_i x_i$$

Neural Networks: components

Dip. Informatica
University of Pisa

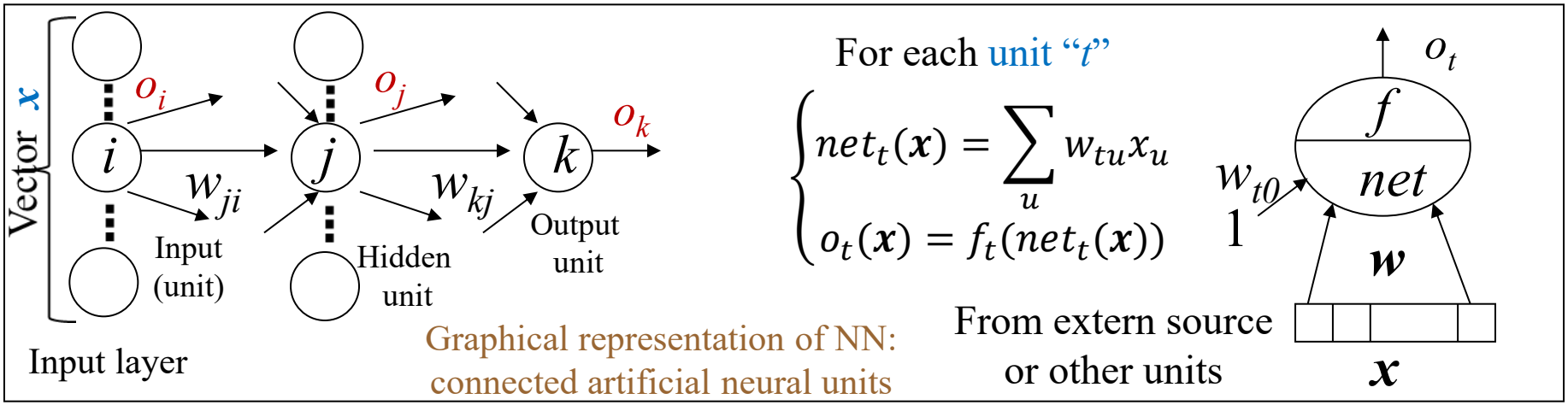
- NN model traditionally presented by the type of
  - UNIT: net, activation functions
  - ARCHITECTURE: number of units, topology (e.g. also number of layers)
  - LEARNING ALGORITHM

The NN and their Units

Standard feedforward NN (with one hidden layer)

Dip. Informatica
University of Pisa

A. A network of units



Notation:

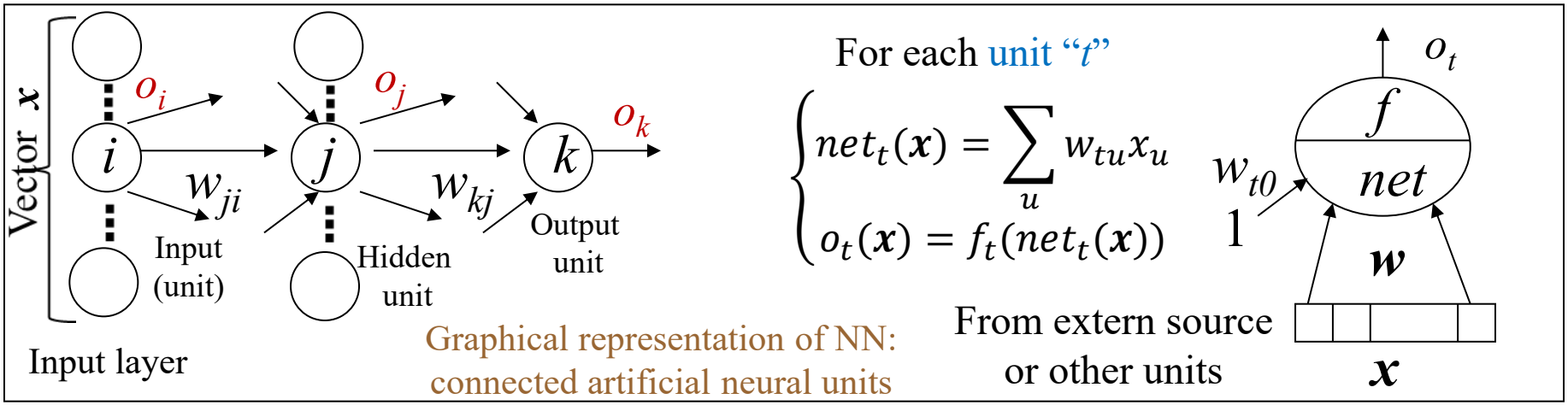
- The index t denotes a generic unit, it can be either j or k (*)
- The index u denotes a generic input component, it can be either i or j (*)
- In the unit, $\boldsymbol{x}$ is a generic input from an external source (input vector) or from other units (*) *according to the position of the unit in the network*
- If we load the pattern $\boldsymbol{x}$ in the input layer, we can use the notation with $\boldsymbol{o}$ for both the inputs and the hidden units outputs. Hence, *inside the network*, the input to each unit t from *any source* u (through the connection w_{tu}) is typically denoted as o_u (we will use this style in the back-prop derivation).

The NN and their Units

Exercise to use a uniform notation for the inputs

Dip. Informatica
University of Pisa

A. A network of units



Important Exercise *(do it before the next lecture):*

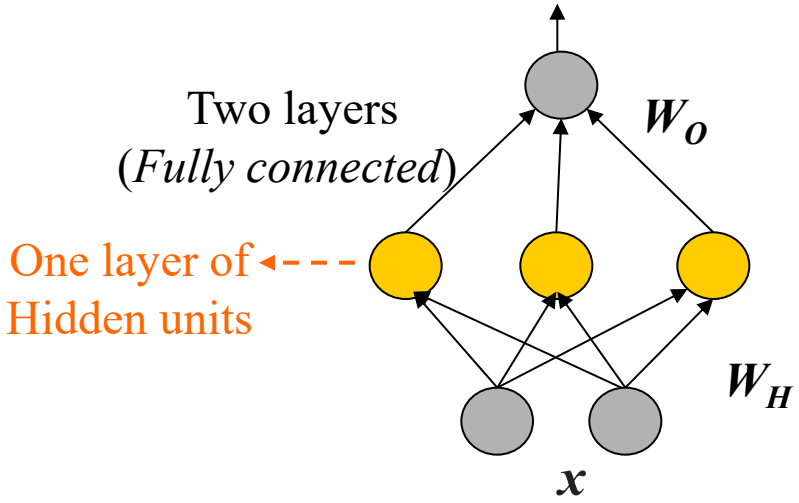
For the eq. of units "*t*", using the symbol *o* for any input to them:

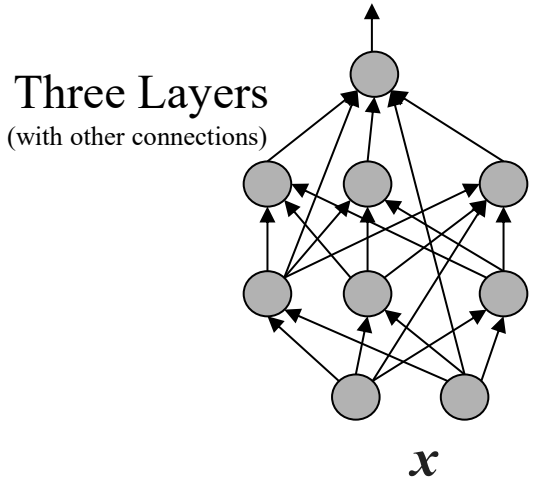
1. What is o_u for w_{ji} ?, i.e the input to the unit *j*?
2. What is o_u for w_{kj} ?
3. What is w_{t0} for each unit *t* ?
4. Write the equations for a hidden unit "*j*" (*t*=*j*) and an output unit "*k*" (*t*=*k*), using the symbol *o* for the any *input* to them (either extern source or other units).

MLP Architecture
Standard feedforward NN

Dip. Informatica
University of Pisa

- The *architecture* of a NN defines the topology of the connections among the units.
- The two-layer **feedforward neural network** described in Equation $h(x)$ corresponds to the well-know **MLP** (Multi Layer Perceptron) architecture:

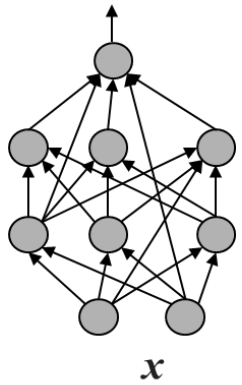




Feedforward processing

Dip. Informatica
University of Pisa

The processing of a pattern for feedforward NN precedes as in the following (from the input layer to the output layer):



- For each input pattern $\boldsymbol{x}$

1. the input pattern is load in the input layer
2. we compute the output of all the units of the 1st hidden layer
3. we compute the output of all the units of the 2st hidden layer and so on for all the hidden layers
4. we compute the output of all the units of the output layer (NN output $h(\boldsymbol{x})$)
5. we can now compute the error (delta) at the output level

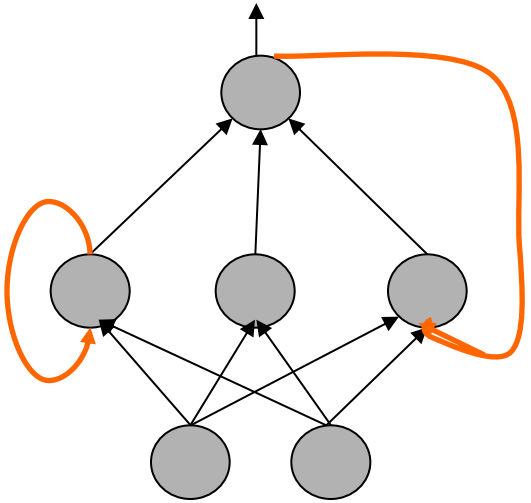
Architectures: Feedforward versus Recurrent

Dip. Informatica
University of Pisa

- **Feedforward**: direction: input→ output
- **Recurrent** neural networks: A different category of architecture, based on the addition of *feedback loops* connections in the network topology,
  - The presence of **self-loop** connections provides the network with dynamical properties, letting a memory of the past computations in the model.
  - This allows us to extend the representation capability of the model to the processing of sequences (and structured data).

Recurrent neural networks:

- See separate slides in this course
- They will be the subject (further developed) of the CNS (Computation Neuroscience) course.



Analysis and questions

Dip. Informatica
University of Pisa

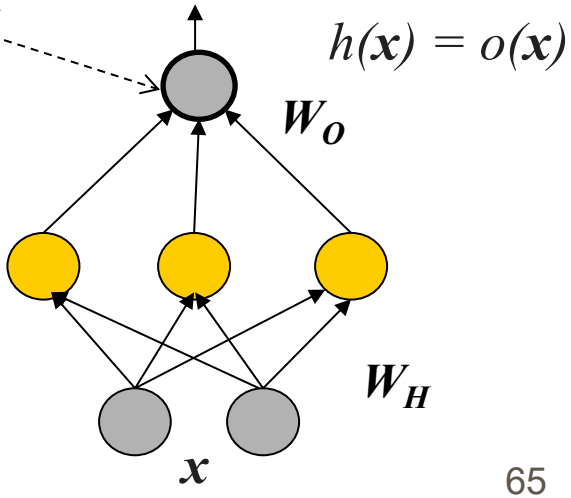
1. **Why NN is a flexible model? (why does it work well)?**
   - **Which tasks?**
   - **Can be interpreted as LBE?** (an important connection for us)
2. **Is the flexibility theoretically grounded?**
3. **How to learn the weights of a NN?**

1. Notes on NN flexibility: Which Tasks?

Dip. Informatica
University of Pisa

- **Hypothesis space**: *continuous* space of all the functions that can be represented by assigning the weight values of the given architecture.
- Note that, depending on the class of values produced by the network output units, discrete or continuous,
  the model can deal, respectively, with **classification tasks** (sigmoidal output *f.*) or **regression tasks** (linear output *f.*) tasks.

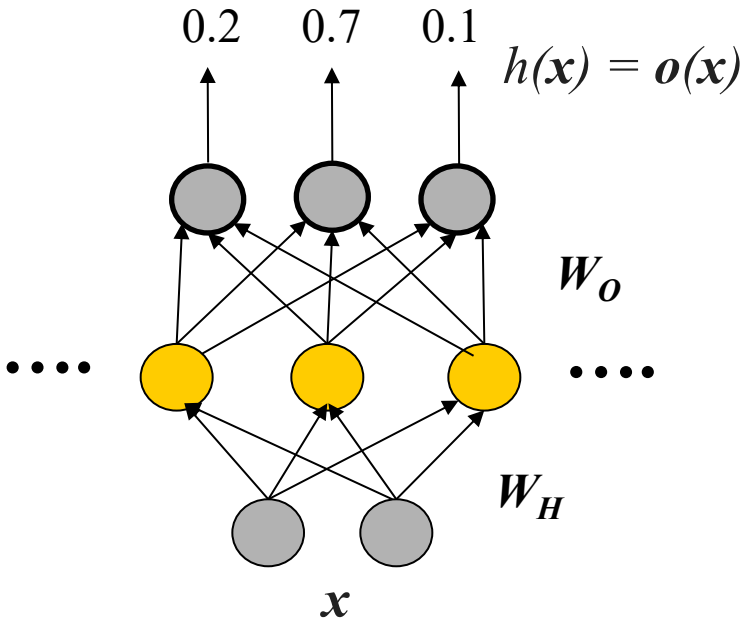
$$h(\boldsymbol{x}) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$$



1. Notes on NN flexibility
Which Tasks?

Dip. Informatica
University of Pisa

- Also **multi-regression** or **multi-classes classifier** can be obtained by using **multiple output units**



1. NN as a *function*

Dip. Informatica
University of Pisa

$$h(\boldsymbol{x}) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$$

- This is the *function* computed by a two-layer feedforward neural network
- Units and architecture just a graphical representation (of the data flow process)
- Each $f_j\left(\sum_i w_{ji} x_i\right)$
- can be seen as computed by an independent processing element (unit, a hidden unit), or a **special kind of phi** (ϕ) of LBE
- Also, NN is a function **non linear** in the parameters $\boldsymbol{w}$

1. NN as a dictionary approach (LBE) (1, equations)

Dip. Informatica
University of Pisa

- **LBE**: (Recall) **fixed** linear basis functions

$$h(\boldsymbol{x}) = \sum_j w_j \phi_j(\boldsymbol{x}) \qquad h(\boldsymbol{x}) = f\left(\sum_j w_j \phi_j(\boldsymbol{x})\right)$$

Regression

Classification
(either regression with f = identity function)

- **Neural Network**

$$h(\boldsymbol{x}) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right) = f\left(\sum_j w_j \phi_j(\boldsymbol{x}, \boldsymbol{w})\right)$$

using

$$\phi_j(\boldsymbol{x}, \boldsymbol{w}) = f_j\left(\sum_i w_{ji} x_i\right)$$

hidden units' output

because now ϕ depends also on $\boldsymbol{w}$

1. NN as a dictionary approach (LBE) (2, discussion)

$$h(\boldsymbol{x}) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$$

Dip. Informatica
University of Pisa

- **LBE**: (Recall) **fixed** linear basis functions (phi preselected: does not change with training data),
  - And linear w.r.t. parameters w:

$$\sum_j w_j \phi_j(\boldsymbol{x})$$

- **NN: Adaptive** (flexible) basis functions approach:

the basis functions themselves are **adapted** to data (by fitting the $\boldsymbol{w}$ in phi)

$$\phi_j(\boldsymbol{x}, \underline{\boldsymbol{w}}) = f_j\left(\sum_i w_{ji} x_i\right)$$

- Note that we fix the same type of basis functions for all the terms in the basis expansion (given by the activation function)

- $h(\boldsymbol{x})$ as (nonlinear function of weighted) sums of nonlinearly transformed linear models + the important enhancement of adaptivity

Please came back to these slides after SVM (another form of LBE) to compare

1. Hidden Layer Relevance and Interpretation

Dip. Informatica
University of Pisa

- Each basis function (**hidden unit**) computes a **new** nonlinear derived features, **adaptively** (by learning, according to the training data)
- I.e. the parameters of the basis function $\boldsymbol{w}$ are learned from data by learning

$$\phi_j(\boldsymbol{x}, \underline{\boldsymbol{w}}) = f_j\left(\sum_i w_{ji} x_i\right)$$

In other words:

- The representational capacity of the model is related to the presence of a hidden layer of units, with the use of non-linear activation function, that transforms the input pattern into the **internal representation** of the network.
- The learning process can define a suitable internal representation, also visible as new hidden features of data, allowing the model to extract from data the higher-order statistic that are relevant to approximate the target function.

1. Note: Non-linear hidden layer is need

Dip. Informatica
University of Pisa

- Non-linear units are essential:
  MLP with linear units = 1 layer NN !

$$h(\boldsymbol{x}) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$$

- Exercise: using f as identity function you can rewrite $h(\boldsymbol{x})$ using a singular layer of units

- Anyway for the learning algorithm this introduces an issue: The model is non-linear in the parameters, i.e. w.r.t. to $\boldsymbol{w}$ → we have a non linear optimization problem

Analysis and questions

Dip. Informatica
University of Pisa

1. **Why NN is a flexible model? (why does it work well)?**
   - **Can be interpreted as LBE?**
2. **Is the flexibility theoretically grounded?**
3. **How to learn the weights of a NN?**

2. Universal approximation (important!)

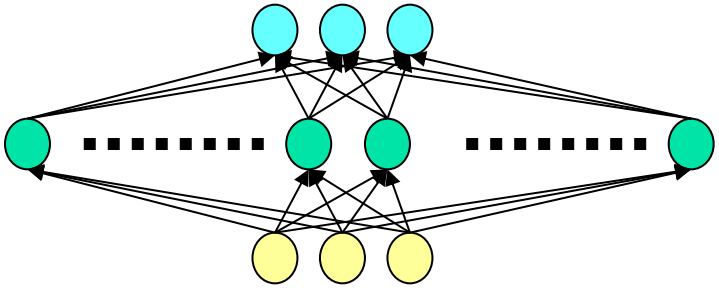
Dip. Informatica
University of Pisa

- Many early results (Cybenko 1989, Hornik et al. 1993, etc…)
- Shortly:
  - A single hidden-layer network (with logistic activation functions) can approximate (arbitrarily well) every continuous (on hyper cubes) function (provided enough units in the hidden layer)

    [e.g. image it generalizes approximation by finite Fourier series]
  - A MLP network can approximate (arbitrarily well) every input-output mapping (provided enough units in the hidden layers)

$$\text{Given } \varepsilon,\ \exists\ h(\boldsymbol{x}) \text{ s.t. } |(f(\boldsymbol{x})-h(\boldsymbol{x})|<\varepsilon \quad \forall\ \boldsymbol{x} \text{ in the hypercube}$$

$$h(\boldsymbol{x}) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$$



Existence theorem! This not provide the algorithm nor the number of units

Two fundamental issues

Dip. Informatica
University of Pisa

After this fundamental result (MLP is able to *represent* any function), two issues will deserve our attention:

- How to learn by NN → next lecture for the first alg.
- How to decide a NN architecture → next lectures

NN Expressive power

Dip. Informatica
University of Pisa

- The ***expressive power*** of NN is strongly influenced by two aspects: the number of units and their configuration (architecture)

- The number of units can be related to the discussion of the **VC-dimension** of the model.

  Specifically, the network capabilities are influenced by the number of parameters w, that is proportional to the number of units *,

  and further studies report also the dependencies on their value sizes, e.g.:
  - Weights = 0 → minimal VC-dim
  - Small weights → linear part of the activation function (small VC-dim)
  - Higher weights values → more complex model

* Number of parameters (fully-conn. MLP)= #input-units X #hidden-units X #output-units

- Next lectures: NN as a *variable-size hypothesis space*
- Indeed, we start with weight values near 0, then w values increase during training → higher VC-dim → it is a variable size hp space

How many layers?

Dip. Informatica
University of Pisa

A look ahead (toward **deep learning**).

- The univ. approx. theorem is a fundamental contribution
- It show that 1 hidden layer is sufficient in general, but it does **not** assure that a "small number" of units could be sufficient (it does not provide a limit on such number)
- It is possible to find boundaries on such number (for many f. families)
- But also to find "*no flattening*" results (on efficiency, not expressivity):
  - cases for which the implementation by a single hidden layer would require an exponential number of units (w.r.t *input dim.*), or non-zero weights,
  - while more layers can help (it can help for the number of units/weights and/or for learning such approximation)
  - But is it easy to optimize (training) a MLP with many layers?
- See later for deep learning.

On the inductive bias of NN

Dip. Informatica
University of Pisa

- NN with backpropagation learning algorithm:
- Generally related to the ***smoothness*** properties of functions:
  - Small input variations → small output variations
  - E.g. a locally limited value of the first derivative
  - A very common assumption in ML

- Why make sense? A target function that is **Non-smooth**: random number generator → generalization cannot be achieved !

Analysis and questions

Dip. Informatica
University of Pisa

1. **Why NN is a flexible model? (why doe it work well)?**
   - **Can be interpreted as LBE?**
2. **Is the flexibility theoretically grounded?**
3. **How to learn the weights of a NN?**

3. NN Learning Algorithm

Dip. Informatica
University of Pisa

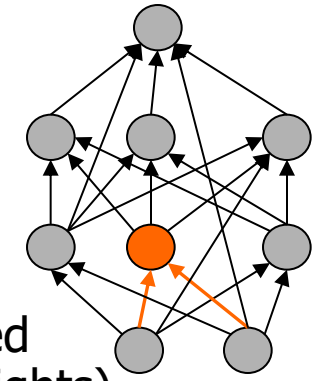
- Repetita: **The learning algorithm** allows adapting the free-parameters $\boldsymbol{w}$ of the model, i.e. the values of the connection-weights, in order to obtain the best approximation of the target function.

- As usual, we will realize this in terms of minimization of an error (or loss) function on the training data set.

From Perceptron to NN

Dip. Informatica
University of Pisa

- **Perceptron**:
  - We have a Perceptron learning algorithm (delta rule)
  - cannot represent all the Boolean functions (e.g. XOR)
- A **network of Perceptrons** can represent every Boolean function
- Problem: define a **Learning Algorithm for the network (MLP)**



- *What is different? Credit assignment problem*:
  - Which credit to the **hidden units**?
  - not easy (as for single Percetron) when errors signal is not directly measurable: we *don't* know the error(***delta***)/desired response for the **hidden units** (useful to change their weights).
- (Repetita) Non-linear wrt to w → *non linear optimization problem*
- Supposed too difficult by Minsky-Papert (1969), while faced by many researchers (see historical notes) with the *backpropagation* algorithm popularized by Rumelhart, Hinton, Williams in the PDP book (1986) → renaissance of NN

The loading problem

Dip. Informatica
University of Pisa

Note that in general, it is indeed a difficult problem:

- *Loading problem*: ("loading" a given set of TR data into the free par. of the NN)
  - given a network and a set of examples
  - answer yes/no: is there a set of weights so that the network will be consistent with the examples?

- The loading problem is *NP-complete* (Judd, 1990),
  it is *not* known a polynomial alg. to solve it.

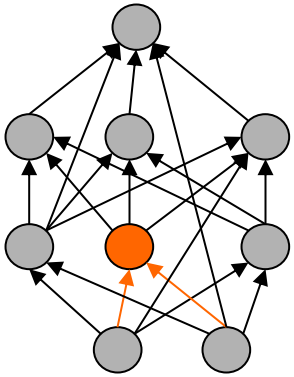
- In practice networks can be trained in a reasonable amount of time (see the back prop alg.) although optimal solution is not guaranteed.

How to solve?

Dip. Informatica
University of Pisa

Key steps:

- Credit assignment problem: how to change the hidden layer weights?
- The ***gradient descend approach***, minimizing a loss function, can be extended to MLP, provided that the *loss function* and *the activation functions* are *differentiable functions*
  - So, to find the delta for every units in the network



Intro to Backpropgation (BP)

Dip. Informatica
University of Pisa

- **The problem is the same we have seen** for the other models, starting again with the **LMS** approach (error/data term):
  - **Given** a set of l training example $(\boldsymbol{x}_{p,}\, d_p)$ and a (inner) loss measure L *(e.g.* $L(h(\boldsymbol{x}_p), dp) = (\, d_p - h_{\boldsymbol{w}}(\boldsymbol{x}_p))^2$ *for the MSE)*
  - **Find**: The weight vector $\boldsymbol{w}$ that minimizes the expected error on the training data (we first focus on the data term), **by computing the gradient of the error function** :

$$E(\boldsymbol{w}) = R_{emp} = \frac{1}{l}\sum_{p=1}^{l}(d_p - h(\boldsymbol{x}_p))^2$$

- **What we need**: differentiable loss, differentiable activation functions, a network to follow the information flow (!)

Backprop. Properties

Dip. Informatica
University of Pisa

- **Nice properties of the Backprog. alg.** (also for programming):
  - Easy because of the *compositional* form of the model
  - It keeps track only of quantities local to each unit (by local variables) → *modularity* of the units is preserved
  - *Efficiency*: O(#W) instead of O(#W^2) (factorization of deltas, see later)
  - On the *biological plausibility*: controversial, but it has been supposed that in the brain, to learn, we have a local suboptimal approx. of BP

Caminante no hay camino, se hace camino al andar. **Antonio Machado**
(There is no path, traveler; the path is made by walking)

Backprogation documents (next lecture)

Dip. Informatica
University of Pisa

- **GO TO PDF file: *Backprop*.pdf**

We will develop it in the *next class*
See (as basic bibliography):

- Course Latex notes (*Backprop*notes…latex*.pdf")
- D. E. Rumelhart, J. L. McClelland. Parallel Distributed Processing, Vol.1, by, MIT Press: Chapter 8 by Rumelhart, Hinton, Williams (1986): "*Learning internal representations by error propagation*", pages: 322-328
  - You can find a pdf on the web, searching for the pdf of "Learning internal representations by error propagation""
  - E.g. a document with exactly the same content (but the links change continuously): https://web.stanford.edu/class/psych209a/ReadingsByDate/02_06/PDPVolIChapter8.pdf

- Prepare yourself with the ***exercises*** suggested in this class!
  - 1 volunteer for in-class solution via Teams (with an electr. board): a plus!

Issues in Training NN

Dip. Informatica
University of Pisa

- Heuristic guidelines in setting the back-prop training

General problems:

- The model is often overparametrized
- Optimization problem is not convex and potentially unstable
- We discuss few of them (see e.g. Haykin for details)
  - Hyperparameters: Starting values, learning rate, on-line/batch
  - Overfitting and regularization
  - Input scaling/ Output representation
  - Hyperparameters: Number of hidden units
  - Multiple Minima
  - Stopping criteria
  - ……

See next lectures (after the backprogatation lecture)

Q&A lecture set-up

Dip. Informatica
University of Pisa

- Please send to me your questions by **email** so to set up Q&A lectures

  Please, **USE in the subject the TAG [ML-24-Q]**

  - We can discuss
    - Self-evaluation quiz results
    - New questions to better understand some parts of the course (no questions, no Q&A lecture ;-))

- Memento
  - Special interactive classes will be used to assess activities and to make a ML discussion forum (**Q&A classes**).
    This makes the course a class, not just a set of records
  - Within a **cooperative** and collaborative context!

Bibliography (NN: first part)

Dip. Informatica
University of Pisa

- Haykin (2nd ed.):
  - chapters 1, 3 (with convergence theorem)
  - chapter 4 (with MLP/backprop alg.)
- Haykin (3nd ed.):
  - chapters Introduction, 1 (with convergence theorem)
  - chapter 4 (with MLP/backprop alg.)
- Mitchell: chapter 4

Other: Every rigorous NN/ML book ! E.g.

- Hastie, Tibshirani, Friedman: cap 11
- Bishop
- ……

For the next lecture:

- Parallel Distributed Processing, Vol.1, by D. E. Rumelhart, J. L. McClelland, MIT Press (an historical book): **chap. 8** (see link to pdf in previous slides)