Machine Learning: Introduction to Neural Networks

Francesco Collovà
francesco.collova@gmail.com

Raffaele Capaldo
raffaele.capaldo@gmail.com

http://uroutes.blogspot.it/

Machine Learning Definition

In 1959, Arthur Samuel defined machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed".[1]

Tom M. Mitchell provided a widely quoted, more formal definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [2]

This definition is notable for its defining machine learning in fundamentally operational rather than cognitive terms, thus following Alan Turing's proposal in Turing's paper "Computing Machinery and Intelligence" that the question "Can machines think?" be replaced with the question "Can machines do what we (as thinking entities) can do?“ [3]

[1] Phil Simon (March 18, 2013). Too Big to Ignore: The Business Case for Big Data. Wiley. p. 89. ISBN 978-1118638170.

[2] Mitchell, T. (1997). Machine Learning, McGraw Hill. ISBN 0-07-042807-7, p.2.

[3] Harnad, Stevan (2008), "The Annotation Game: On Turing (1950) on Computing, Machinery, and Intelligence", in Epstein, Robert; Peters, Grace, *The Turing Test Sourcebook: Philosophical and Methodological Issues in the Quest for the Thinking Computer*, Kluwer

Machine Learning Algo

- Supervised Learning
- Unsupervised Learning
- Others: reinforcement learning, recommender systems.

Examples Supervised/Unsupervised

- Set of email labeled as Spam/noSpam, learn a spam filter;
- Given a set of news articles found on the web, group them into set of articles about the same story
- Database of customer data: automatically discover market segments and group customers into different market segments;
- Given a dataset of patients diagnosed has having diabetes or not, learn to classify new patients having diabetes or not;

Classification: Supervised Learning

- Supervised learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples .
- In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances.

- This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way (see inductive bias).

http://en.wikipedia.org/wiki/Supervised_learning

Clustering: Unsupervised Learning

- Unsupervised learning is the machine learning task of inferring a function from unlabeled training data. The training data consist of a set of training examples .
- In unsupervised learning, each example is consisting of only input object (typically a vector) whitout output value (targets).
- A unsupervised learning algorithm analyzes the training data, separating and grouping (clustering) with the similarity metric, without using comparisons with output data.
  It is an autonomous learning and there is no external control on the error.
- Models that use this type of learning are:
  -Self-Organizing Maps (SOM) of Kohonen
  -Hopfield Networks

http://en.wikipedia.org/wiki/Supervised_learning

Supervised learning: classification problem

- The problem is then reduced to the determination of the set of best weights (w_0, w_1, …, w_n) to minimize the classification errors.

- So the hypothesis space H is infinite and is given by all the possible assignments of values to the n +1 weights (w_0, w_1, …, w_n):

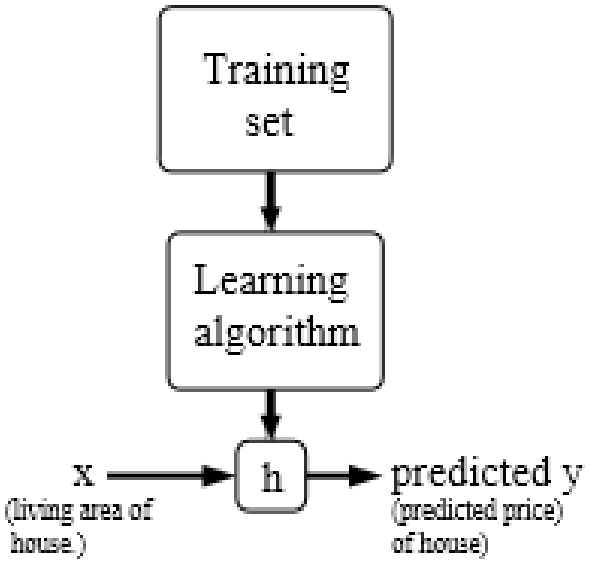
$$H = \{\vec{w} : \vec{w} \in \Re^{n+1}\}$$

Supervised Learning

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function

$$h : X \rightarrow Y$$

so that h(x) is a “good” predictor for the corresponding value of y.



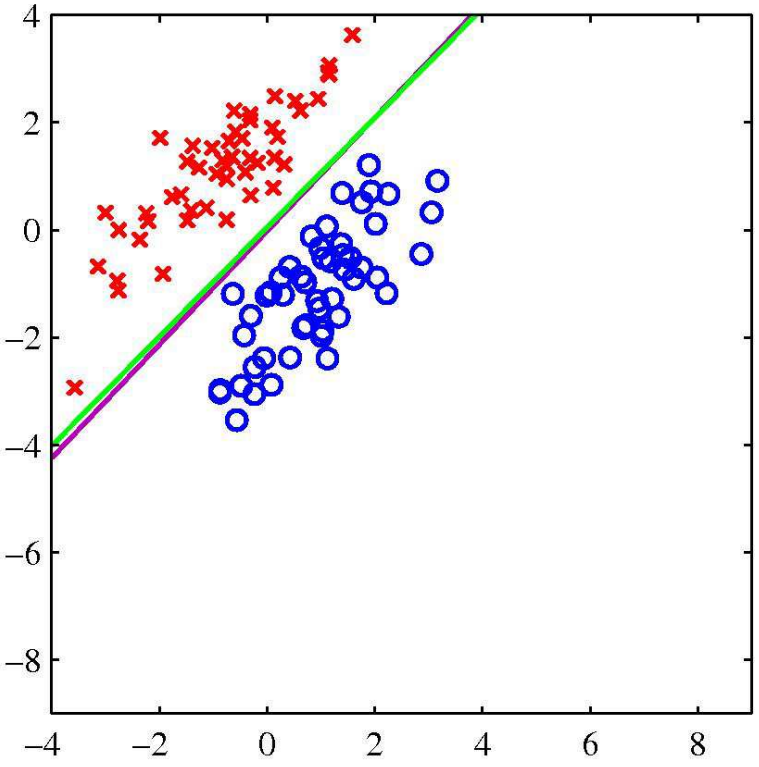
For historical reasons, this function h is called a hypothesis.

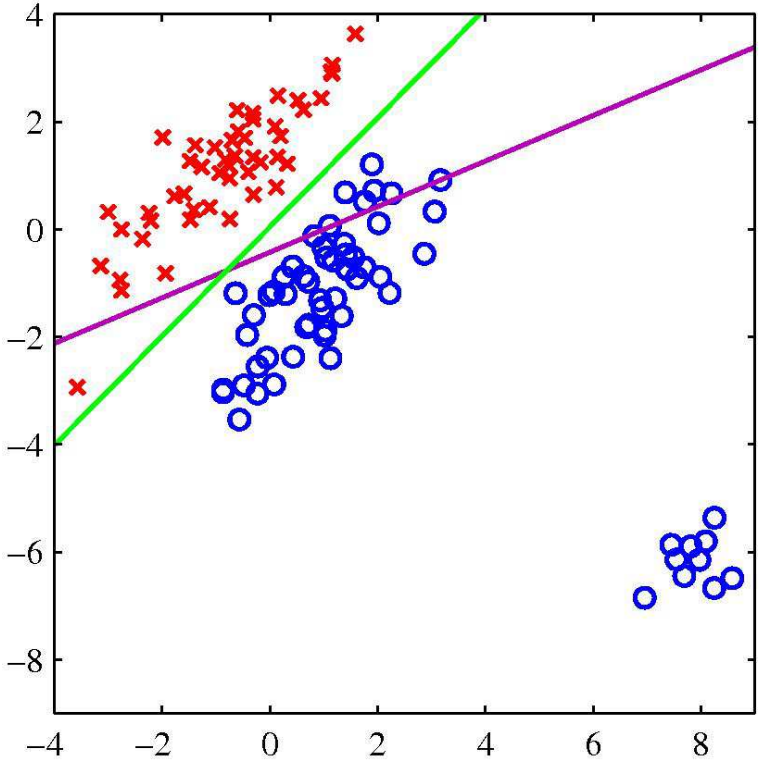
When the target variable that we’re trying to predict is continuous, we call the learning problem a regression problem.

When y can take on only a small number of discrete values, we call it a classification problem.

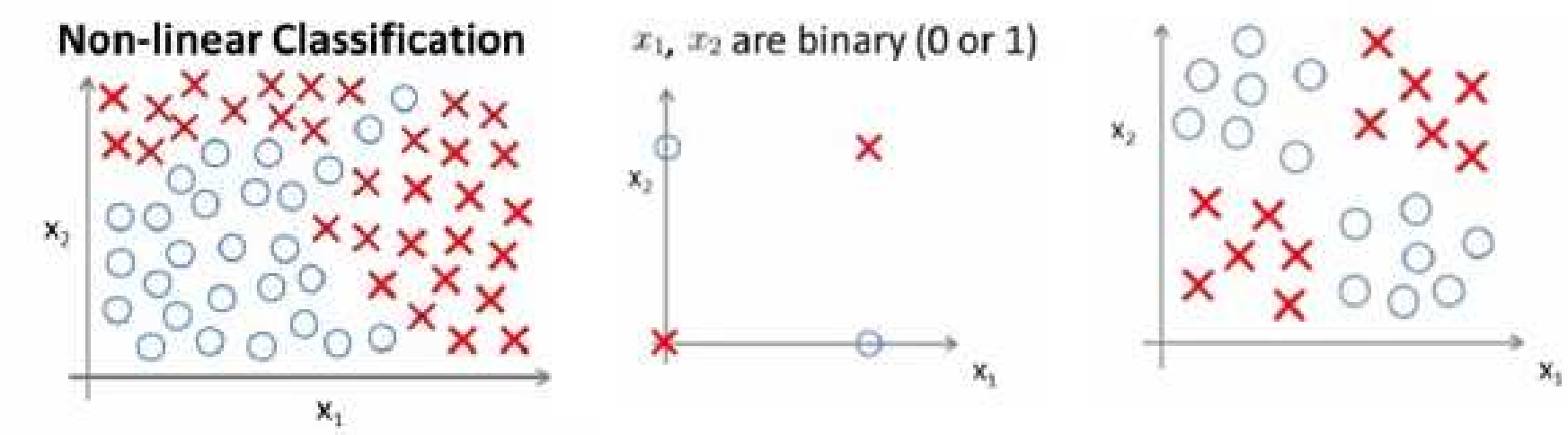
Linear classification

Find the parameters that minimize the squared distance between the data set and the decision boundary

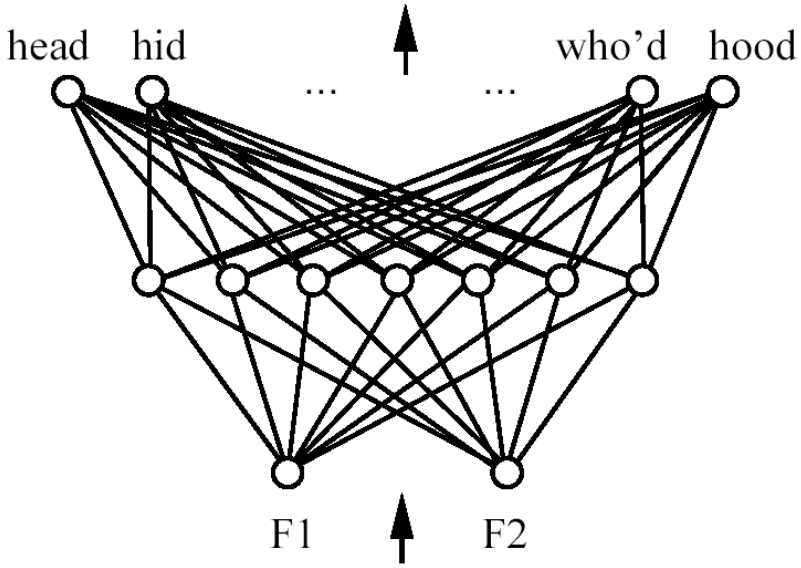


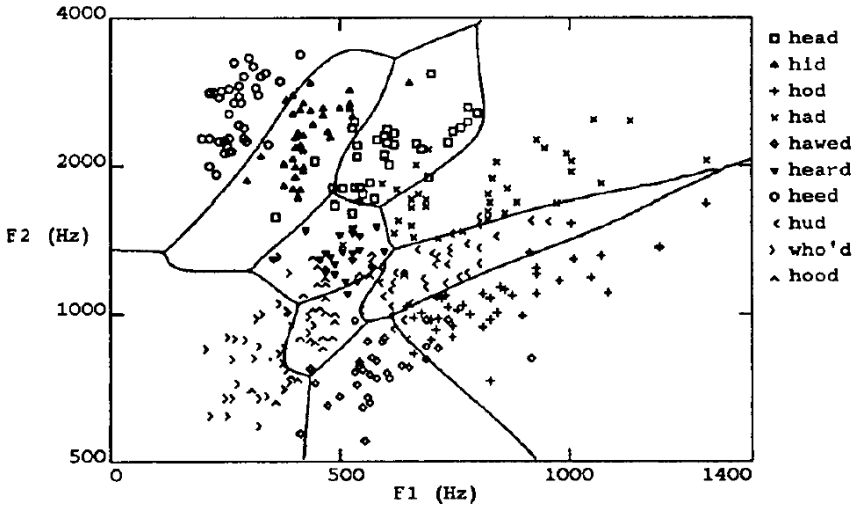


Non Linear Classification



Learning non-linear decision boundary!

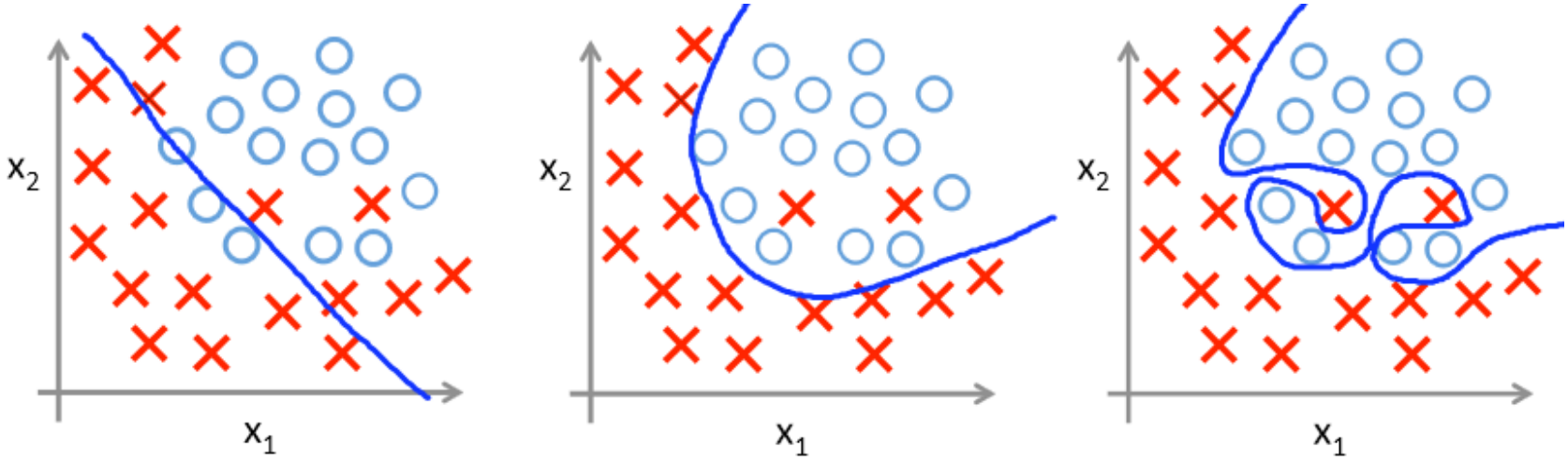




Overfitting

Learn the “data”and not the underlying function

Overfitting may occur when learned function performs well on the data used during the training and poorly with new data.

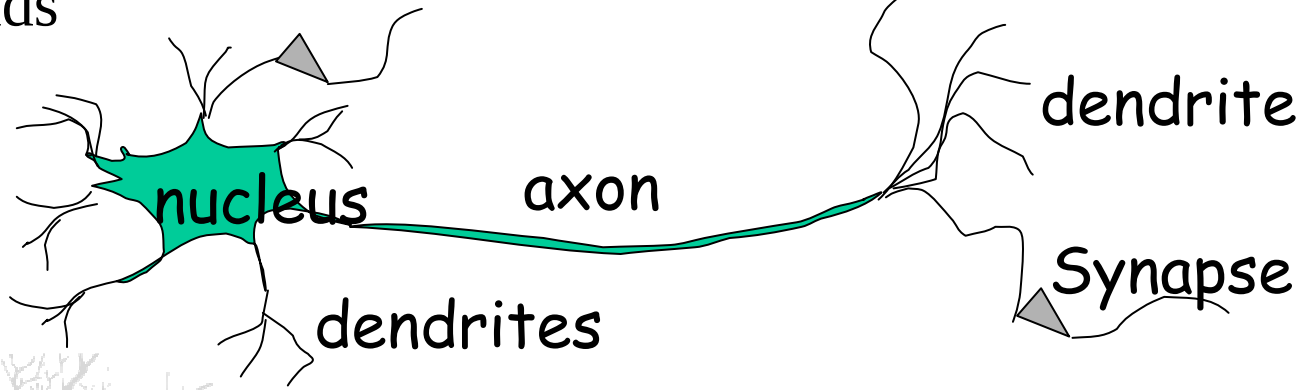


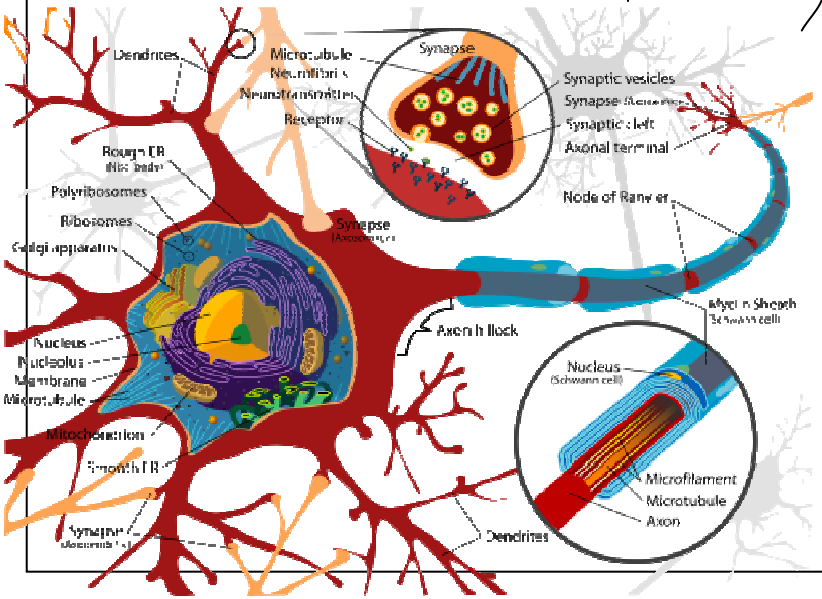
Neural Networks: history

- Artificial Neural Networks (ANN) are a simulation abstract of our nervous system, which contains a collection of neurons which communicate each other through connections called axons
- The ANN model has a certain resemblance to the axons and dendrites in a nervous system
- The first model of neural networks was proposed in 1943 by McCulloch and Pitts in terms of a computational model of neural activity.
- This model was followed by other proposed by John Von Neumann, Marvin Minsky, Frank Rosenblatt, and many others

Brain Neurons

- Many neurons possess arboreal structures called dendrites which receive signals from other neurons via junctions called synapses
- Some neurons communicate by means of a few synapses, others possess thousands





Functioning of a Neuron

- It is estimated that the human brain contains over 100 billion neurons and that a neuron can have over 1000 synapses in the input and output
- Switching time of a few milliseconds (much slower than a logic gate), but connectivity hundreds of times higher;
- A neuron transmits information to other neurons through its axon;
  - The axon transmits electrical impulses, which depend on its potential;
  - The transmitted data can be excitatory or inhibitory;
- A neuron receives input signals of various nature, which are summed;
- If the excitatory influence is predominant, the neuron is activated and generates informational messages to the output synapses;

Neural Network and the Brain

There is this fascinating hypothesis that the way the brain does all of these different things is not worth like a thousand different programs, but instead, the way the brain does it is worth just a single learning algorithm.

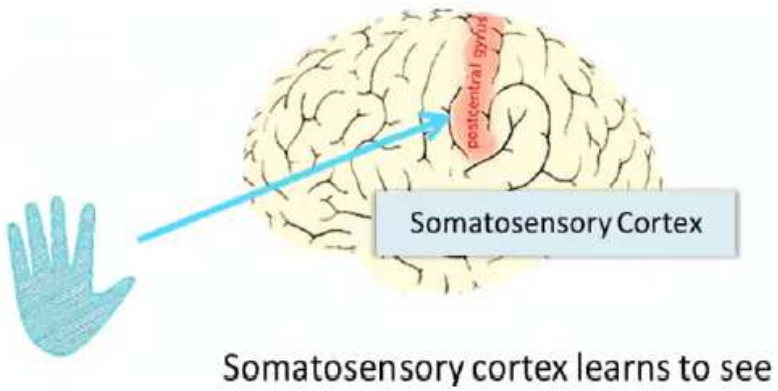
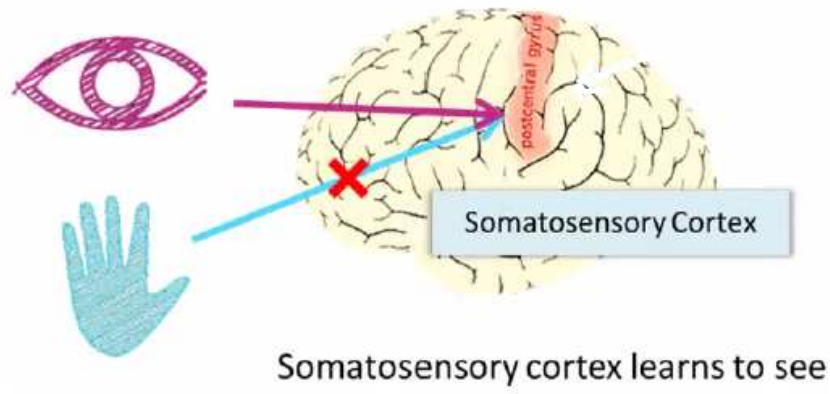


Figure 8: Somatosensory cortex learns to see.



More Examples

Seeing with your tongue

Human echolocation (sonar)

Haptic belt: Direction sense

Implanting a 3rd eye

Brainport: http://www.youtube.com/watch?v=xNkw28fz9u0
http://www.youtube.com/watch?v=CNR2gLKnd0g

Ecolocation:
http://www.youtube.com/watch?v=qLziFMF4DHA&list=TL9k0aIpmZTxg

Haptic belt: http://www.youtube.com/watch?v=mQWzaOaSqk8

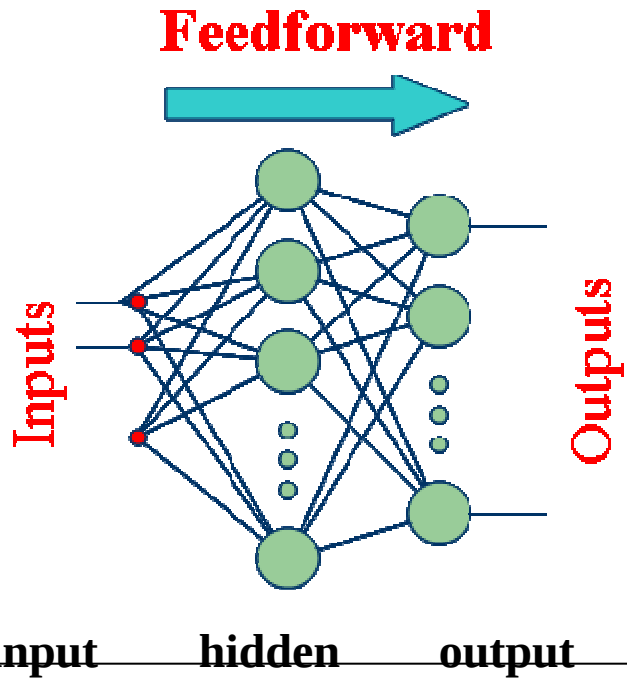
Structure of a Neural Network

- A neural network consists of:
  - A set of nodes (neurons) or units connected by links
  - A set of weights associated with links
  - A set of thresholds or levels of activation

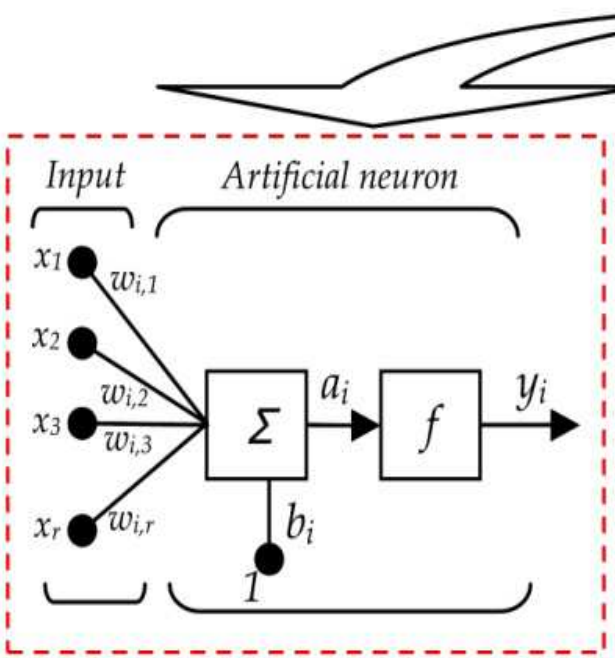
- The design of a neural network requires:
  - The choice of the number and type of units
  - The determination of the morphological structure (layers)
  - Coding of training examples, in terms of inputs and outputs from the network
  - The initialization and training of the weights on the interconnections through the training set

Multi Layer Network Feedforward

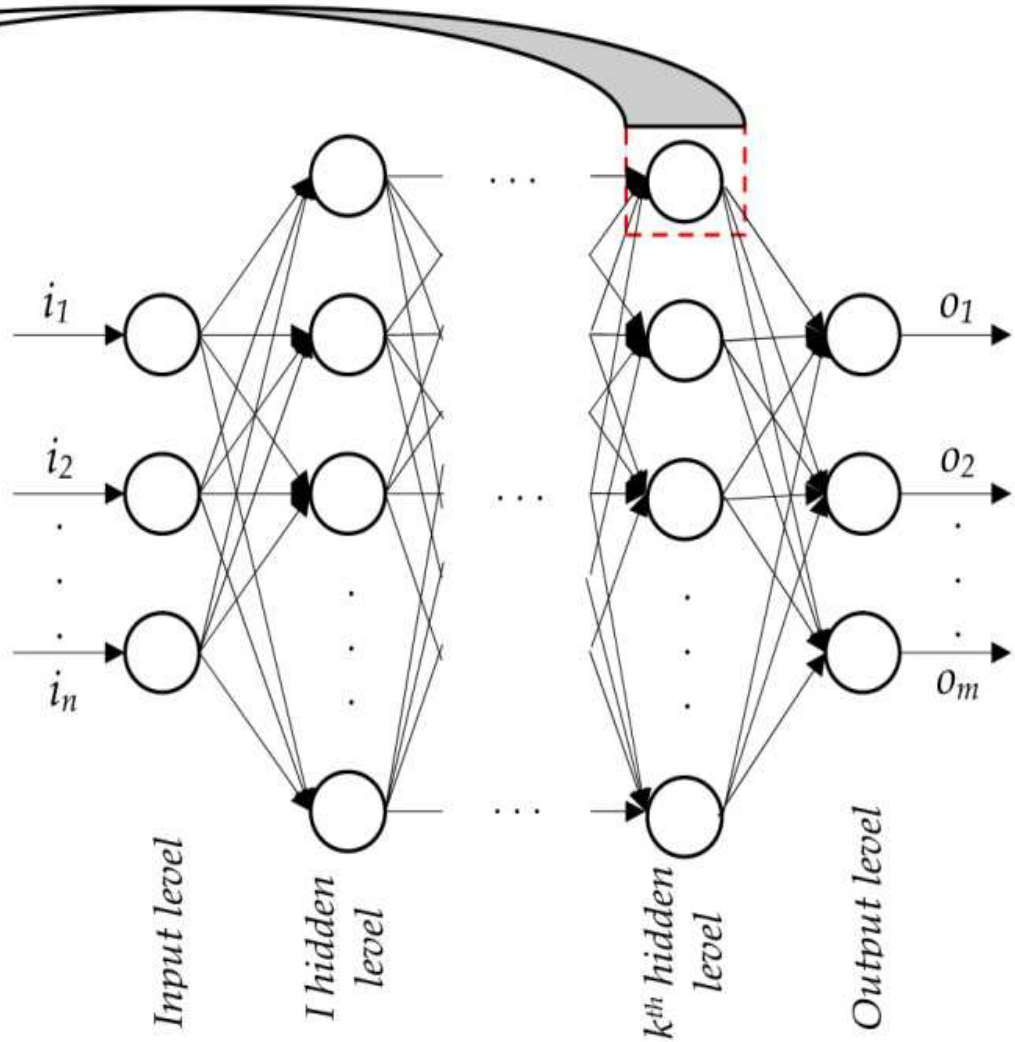
- Feedforward Neural Networks
  - Each unit is connected only to that of the next layer
  - The processing proceeds smoothly from the input unit to output
  - There is no feedback (directed acyclic graph or DAG)
  - They have no internal state



Multi Layer Network Feedforward



Perceptron

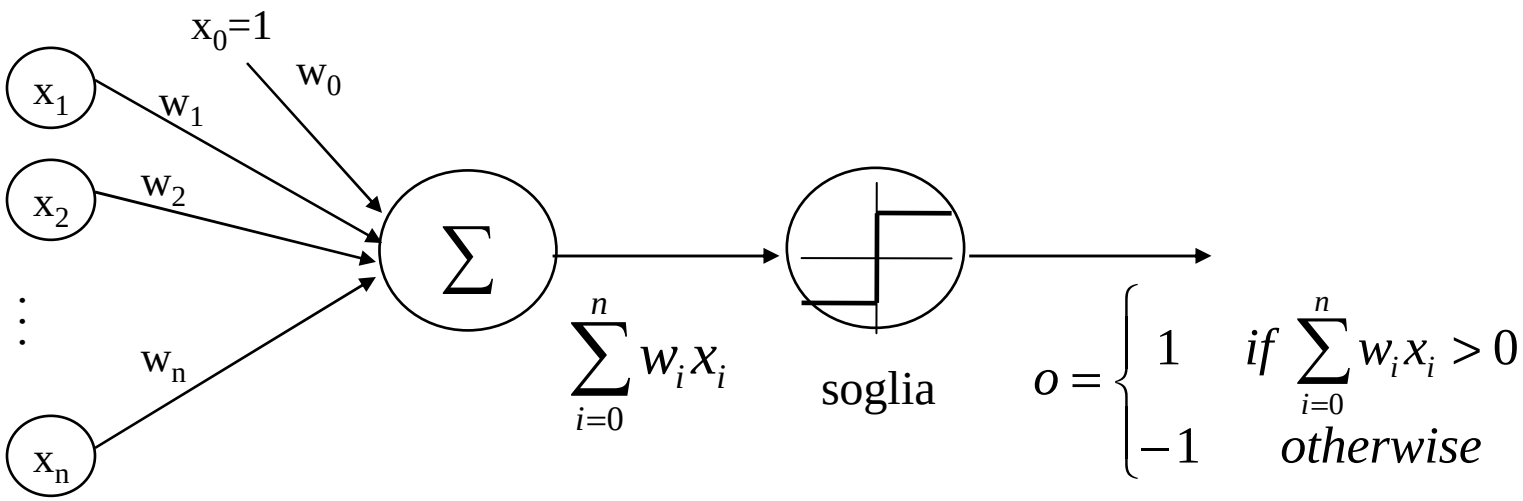


Problems solvable with Neural Networks

- Network characteristics:
  - Instances are represented by many features in many of the values, also real
  - The target objective function can be real-valued
  - Examples can be noisy
  - The training time can be long
  - The evaluation of the network must be able to be made quickly learned
  - It isn't crucial to understand the semantics of the function wait
- Applications: robotics, image understanding, biological systems, financial predictions, etc..

The Perceptron

- The perceptron is milestone of neural networks
- Idea belong to Rosenblatt (1962)
- Tries to simulate the operation of the single neuron

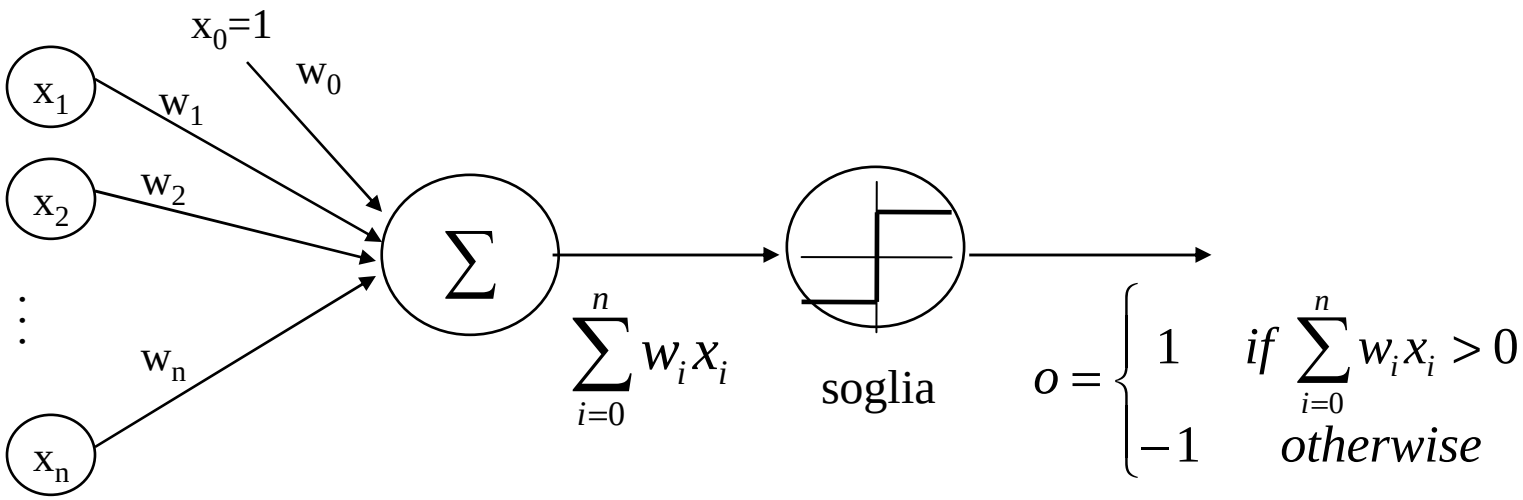


$$\sum_{i=0}^{n} w_i x_i$$

$$o = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

The Perceptron

- The output values are boolean: 0 or 1
- The inputs xi and weights wi are positive or negative real values
- Three elements: inputs, sum, threshold
- The learning is to select weights and threshold



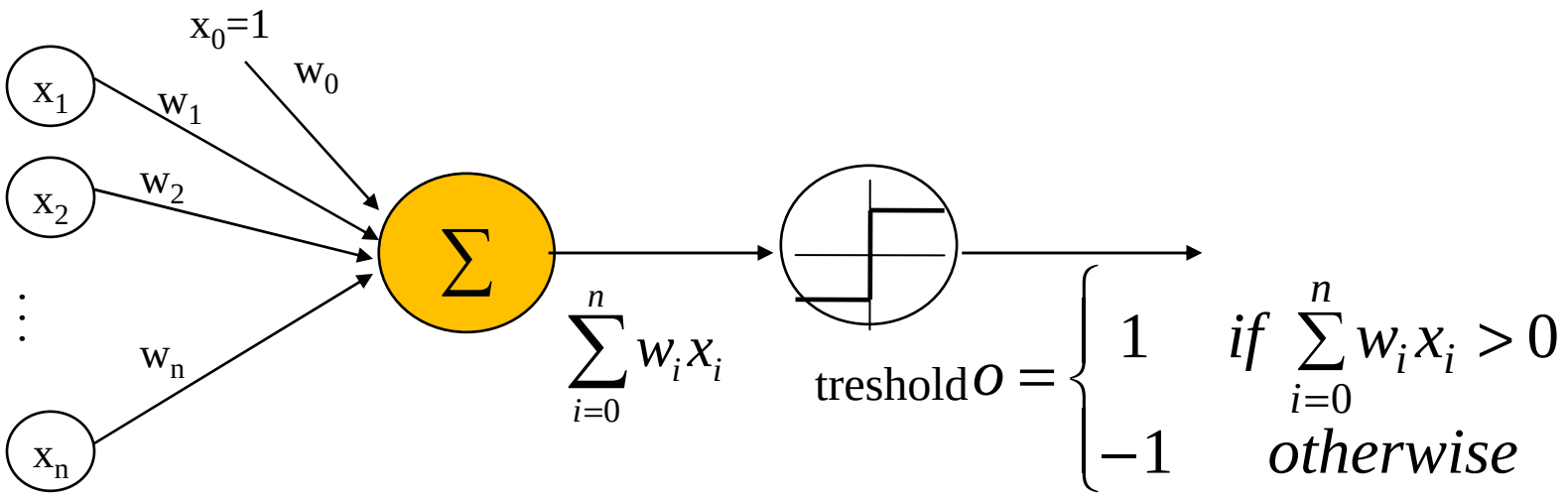
$$\sum_{i=0}^{n} w_i x_i$$

$$o = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

Function sum and treshold (1)

- The **input function** (linear sum of the input components of $x = (x_1, \ldots, x_n)$)

$$w_0 + w_1 x_1 + \ldots + w_n x_n = \sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$$

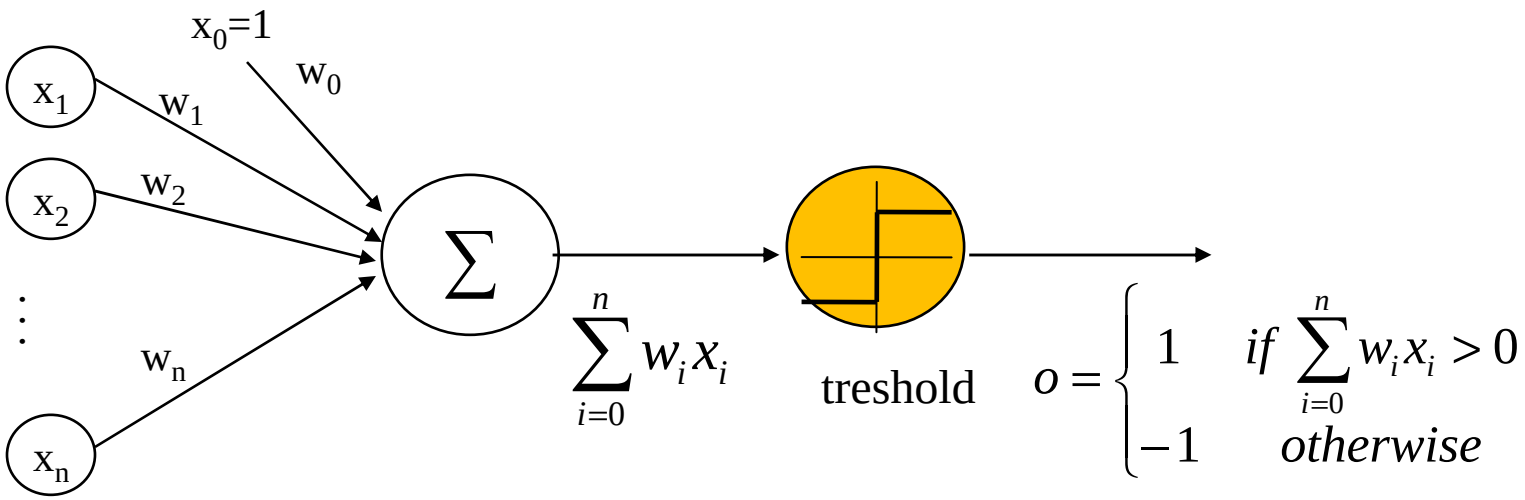


Function sum and treshold (2)

- The **activation function** (non linear, treshold)

$$o(x_1,\ldots,x_n) = g\left(\sum_{i=0}^{n} w_i x_i\right)$$

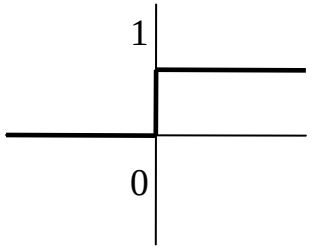
  - We want the perceptron **active** (close to +1) when the correct inputs are provided and **inactive** otherwise
  - It's better that g is **not linear**, otherwise the neural network collapses to a linear function of the input



Activation functions

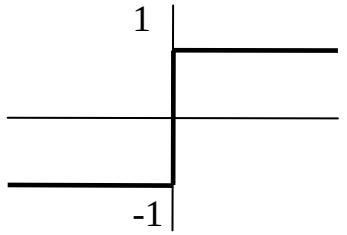
- **Step** function

$$step_t(x) = \begin{cases} 1 & x > t \\ 0 & otherwise \end{cases}$$



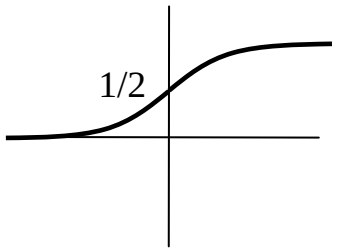
- **Sign** function

$$sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & altrimenti \end{cases}$$



- **Sigmoid** function

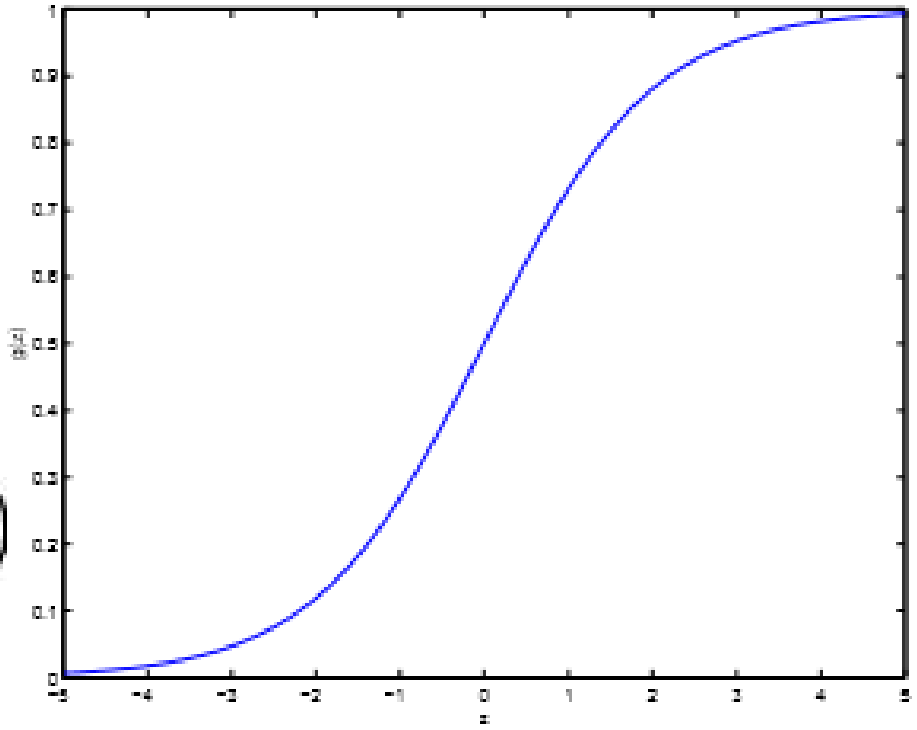
$$sigmoide(x) = \frac{1}{1+e^{-x}}$$



Logistic function (Sigmoid)

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

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}(e^{-z}) \\ &= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)). \end{aligned}$$

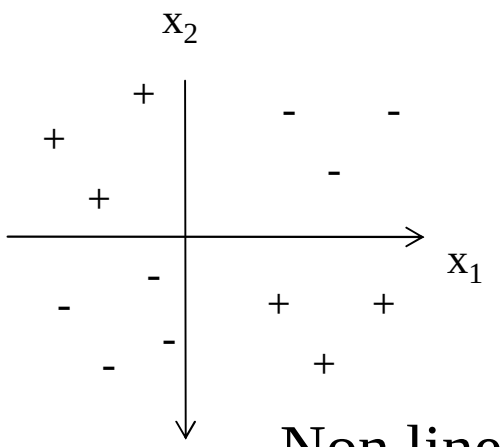


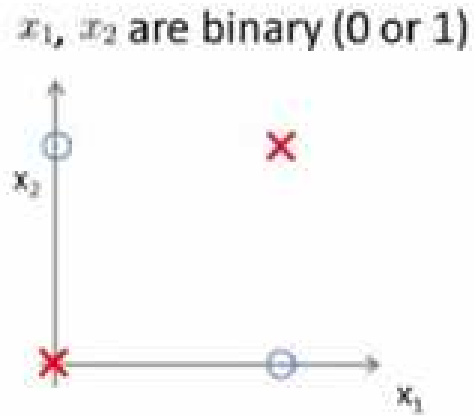
The derivative of logistic function has a nice feature:

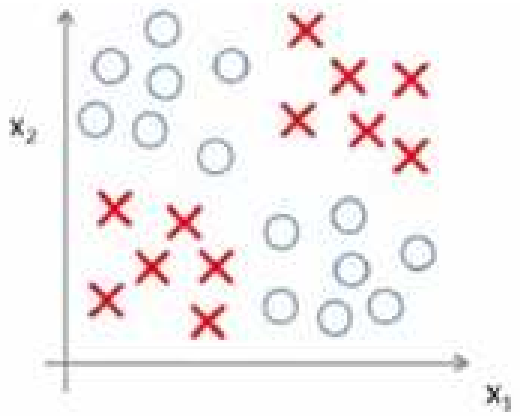
g'(z) = g(z)(1 - g(z))

Functions represented by the perceptron

- The perceptron can represent all boolean primitive functions AND, OR, NAND e NOR
- Some boolean functions can not be represented
  - E.g. the XOR function (that is 1 if and only if $x_1 \neq x_2$) requires more perceptrons







Non linear classification require a network of perceptrons

Learning: Neuron Error

The rule commonly used to adjust the weights of a neuron is the delta rule or Widrow-Hoff rule. Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ provided the input to the neuron.
If t and y are, rispectivly, the desired output and the output neural, the error δ is given by

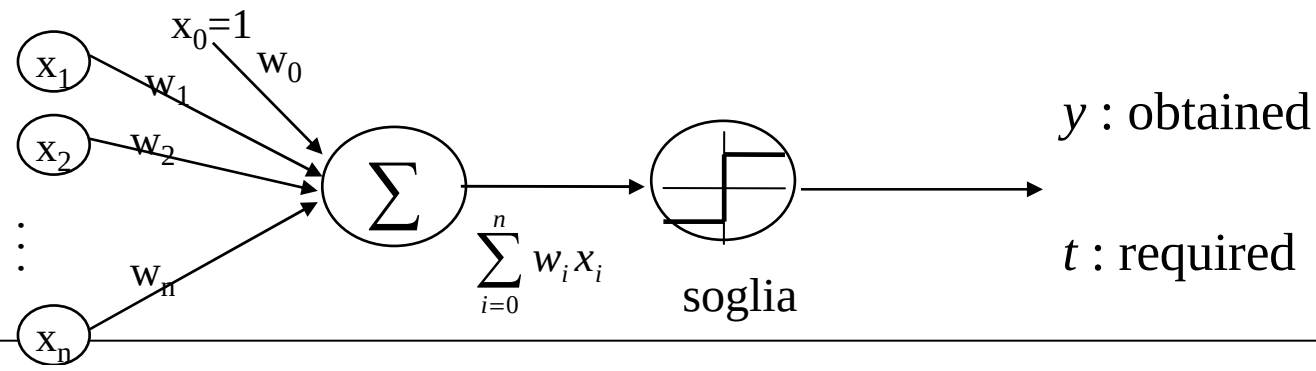
$$\delta = t - y.$$

The delta rule states that the change in the general weight Δw_i is:

$$\Delta wi = \eta \delta x_i \quad \text{where} \quad \eta \in [0,1] \text{ is learning rate.}$$

The learning rate determines the learning speed of the neuron.
The delta rule change in a way proportional to the error only the weights of the connections that have contributed to the error (ie that $x_i \neq 0$). The new value of the weights is:

$$w_i = w_i + \Delta w_i$$



Local and Global Error

Local Error

The local error of the *k*-th output neuron is given by

$$\varepsilon_k=(1/2)(t_k-y_k)^2$$

Its purpose is to be minimized by the change (delta rule) the connection weights w_k so that the output y_k is as close as possible to desired response t_k.

Global error (cost function)

The global error relative to the *N* output nodes and the *M* input pattern is given by

$$E=\frac{1}{2M}\sum_{r,k=1}^{M,N}(t(r)_k-y(r)_k)^2$$

where '*n*' is the index of the pattern .
For a given training set, the value *E* is a "cost" function that indicates the performance of network learning. The learning takes place minimizing this value through the back-propagation algorithm.

Back propagation principle

The back propagation algorithm is a generalization of the *delta rule* for training multilayer networks (MLN). This algorithm updates the weights w_i of the network by means of successive iterations, that minimize the cost function of the error ***E***.

The minimization of the error is obtained using the gradient of the cost function, which consists of the first derivative of the function with respect to all the weights w_i namely:

$$\frac{\partial}{\partial w_i}(E)$$

On the basis of this gradient the weights will be updated with the following mechanism:

$$w_i = w^0{}_i - \eta \frac{\partial}{\partial w_i}(E)$$

where w_i are the weights updated , $w^0{}_i$ are the random weights that initiate the process of adjustment and η is the learning rate.

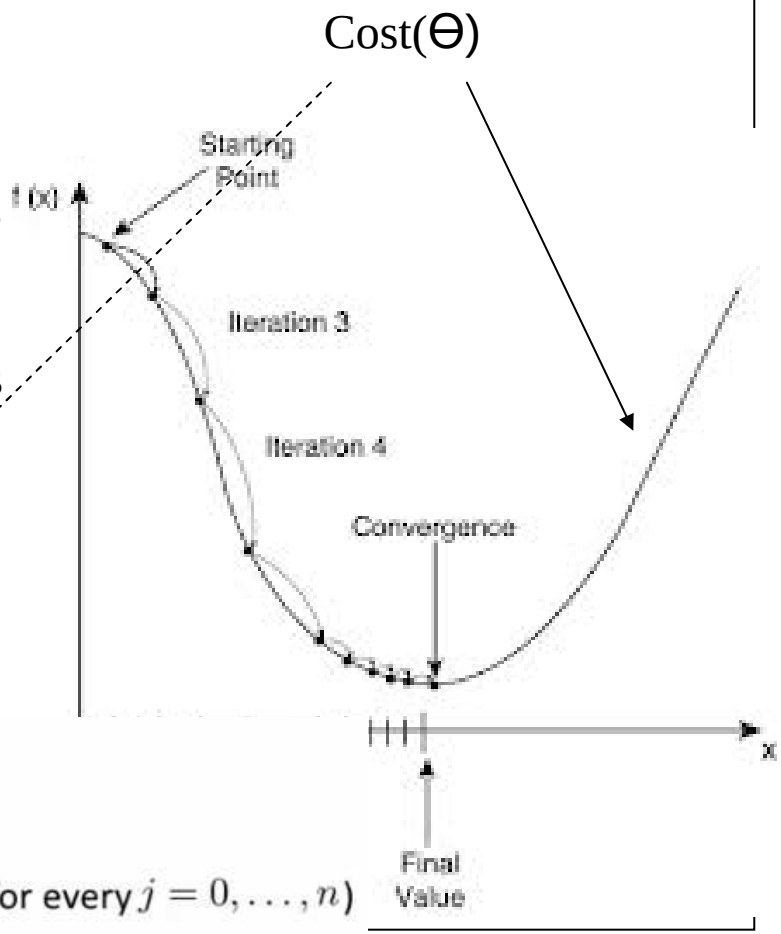
Gradient descend elements

Updating process

$\Theta_{n+1} = \Theta_n - \alpha Cost'(\Theta n)$

The gradient descent is an optimization technique of local type. Given a multi-dimensional mathematical function, the gradient descent allows you to find a local minimum of this function.

The technique consists in evaluating, initially in a random point in the multidimensional space (first step), and the function itself and its gradient. The gradient, being a descent direction indicates the direction in which the function tends to a minimum. It then chooses another point (second point) in the direction indicated by the gradient.



Repeat $\{$

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$

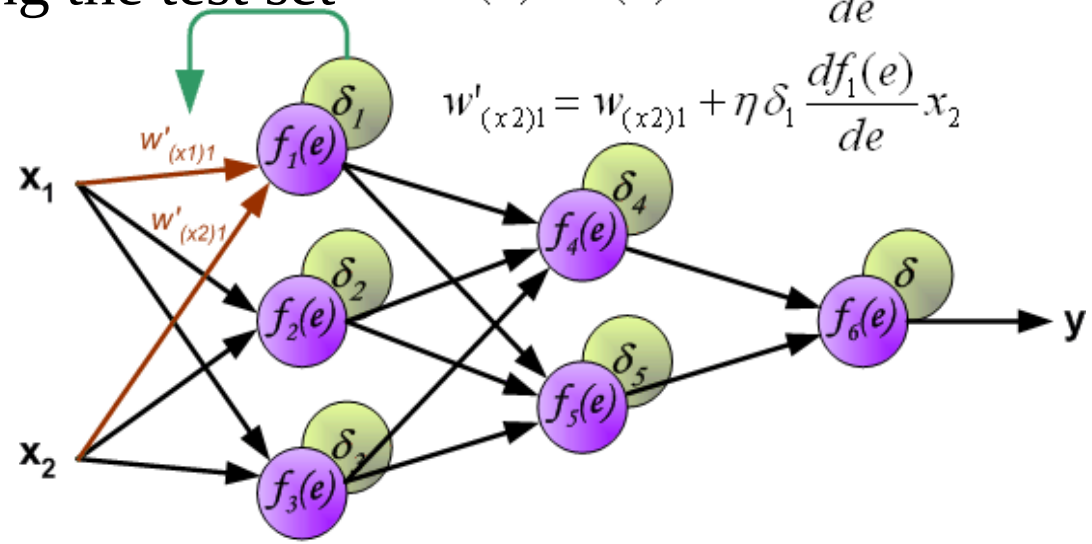
$\}$ (simultaneously update for every $j = 0, \ldots, n$)

Feedforward Network Training by Backpropagation: Process Summary

- Select a network architecture
- Randomly initialize weights
- While error is too large
  - Select training pattern and feedforward to find actual network output
  - Calculate errors and backpropagate error signals
  - Adjust weights
- Evaluate performance using the test set

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



Backpropagation algorithm (more detail)

function BackProp (D, η, n_{in}, n_{hidden}, n_{out})

- D is the training set consists of m pairs: $\{(x_i, y_i)^m\}$
- η is the learning rate as an example (0.1)
- n_{in}, n_{hidden} e n_{out} are the numbero of imput hidden and output unit of neural network

Make a feed-forward network with n_{in}, n_{hidden} e n_{out} units

Initialize all the weight to short randomly number (es. [-0.05 0.05])

Repeat until termination condition are verifyed:

For any sample in D:

Forward propagate the network computing the output o_u of every unit u of the network

Back propagate the errors onto the network:

- For every output unit k, compute the error δ_k: $\delta_k = o_k(1-o_k)(t_k-o_k)$
- For every hidden unit h compute the error δ_h: $\delta_h = o_h(1-o_h)\sum_{k \in outputs} w_{kh}\delta_k$
- Update the network weight w_{ji}: $w_{ji} = w_{ji} + \Delta w_{ji}, \quad where \;\; \Delta w_{ji} = \eta \delta_j x_{ji}$

(x_{ji} is the input of unit j from coming from unit i)

But does algorithm converges ?

- Gradient algorithm problems:
  - May stop at local minima
  - A local minimum can give solutions very worst of the global minimum
  - There may be many local minima

- Possible solutions:

  training the network with different initial weights, train different network architectures

Termination conditions

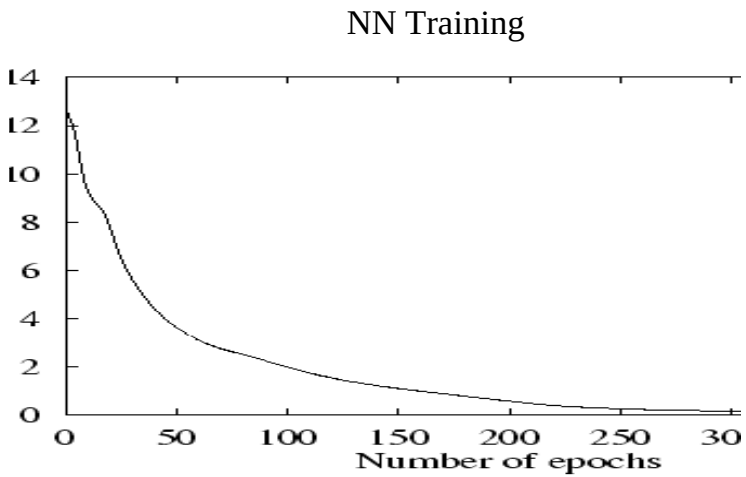
- The process continues until you have exhausted all of the examples (time), then again
- When do you stop the process? Minimize errors on the set D (trainning set) is not a good criterion (*overfitting*)
- It is preferred to minimize errors on a test set T, that is, subdivide D in D’$\cup$T, train on D' and use T to determine the termination condition.

Correct chain: validating

Three subsets of the data set: training set D, validation test V and test set T

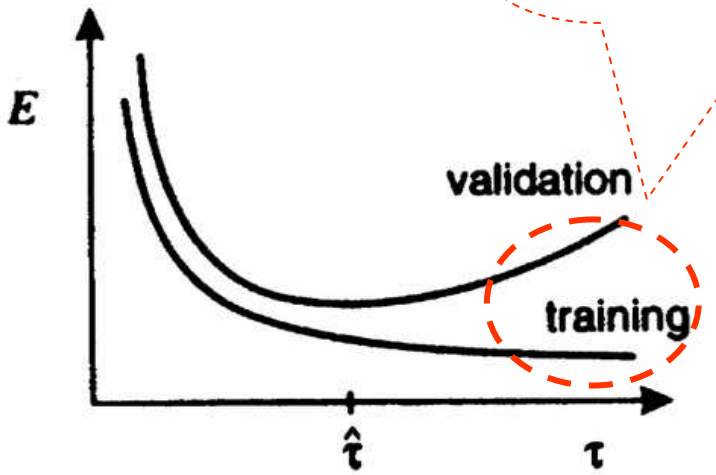
- # nodes in input = # features.
- # nodes in output = # classes.
- # hidden layer = # nodes per level: k-fold cross validation on the training set.
- Train the network selection with the whole training set, limiting overfitting with validation set.
- Rate the accuracy on the final test set.

Plot of the error on a trainning set D and validation test V



Overfitting area

Here the network is learning the data not the model. Stop the learning when the error in the validation set start to increase.



Error backpropagation

- Limits
  - Absence of general theorems of convergence
  - May result in local minima of E
  - Difficulty for the choice of parameters
  - Poor generalization ability, even in the case of good minimization of E

- Possible changes for improvement
  - Adaptive learning rate
  - Period of time
  - Deviations from the steepest descent
  - Variations in the architecture (number of hidden layers)
  - Inserting backward connections

Error backpropagation

- The learning rate
  - Large learning rate, risk of oscillatory behavior
  - Small learning rate, slow learning

- Strategies to identify the optimal architecture
  - Large network easily, but generalizes poorly
  - From a big network remove hidden neurons, if estimate that can continue to learn even with less neurons
  - Small network learns with difficulties, but generalizes well. Starting from a small network add hidden neurons, if the descent of the function E is too slow or blocked

Some pratical considerations

- The choice of initial weights has a large impact on the convergence problem! If the size of the input vectors is N and N is large, a good heuristic is to choose the initial weights between -1/N and 1/N
- The BP algorithm is very sensitive to the learning factor η. If it is too big, the network diverges.
- Sometimes, it is preferable to use different values of η for the different layers of the network
- The choice of the encoding mode of the inputs and the architecture of the network can dramatically affect the performance!

References

[1] Stephen Boyd Convex Optimization Cambridge University Press (2004)

[2] Christopher M. Bishop Pattern Recognition and Machine Learning Springer (2007)

[3] Nils J. Nilsson Introduction to Machine Learning Robotics Laboratory Department of Computer Science Stanford University (1996)

[4] Andrew Ng Stanford University https://www.coursera.org/course/ml

[5] Ethem Alpaydin Introduction to Machine Learning Second Edition The MIT Press Cambridge, Massachusett London, England (2010)

[6] Velardi Paola Università di Roma "La Sapienza" twiki.di.uniroma1.it/pub/ApprAuto/AnnoAcc0708/4Neural.ppt

[7] Francesco Sambo Università degli studi di Padova Apprendimento automatico e Reti Neurali http://www.dei.unipd.it/~sambofra/Apprendimento_Automatico_e_Reti_Neurali-0910.pdf