Multi-layer Perceptron

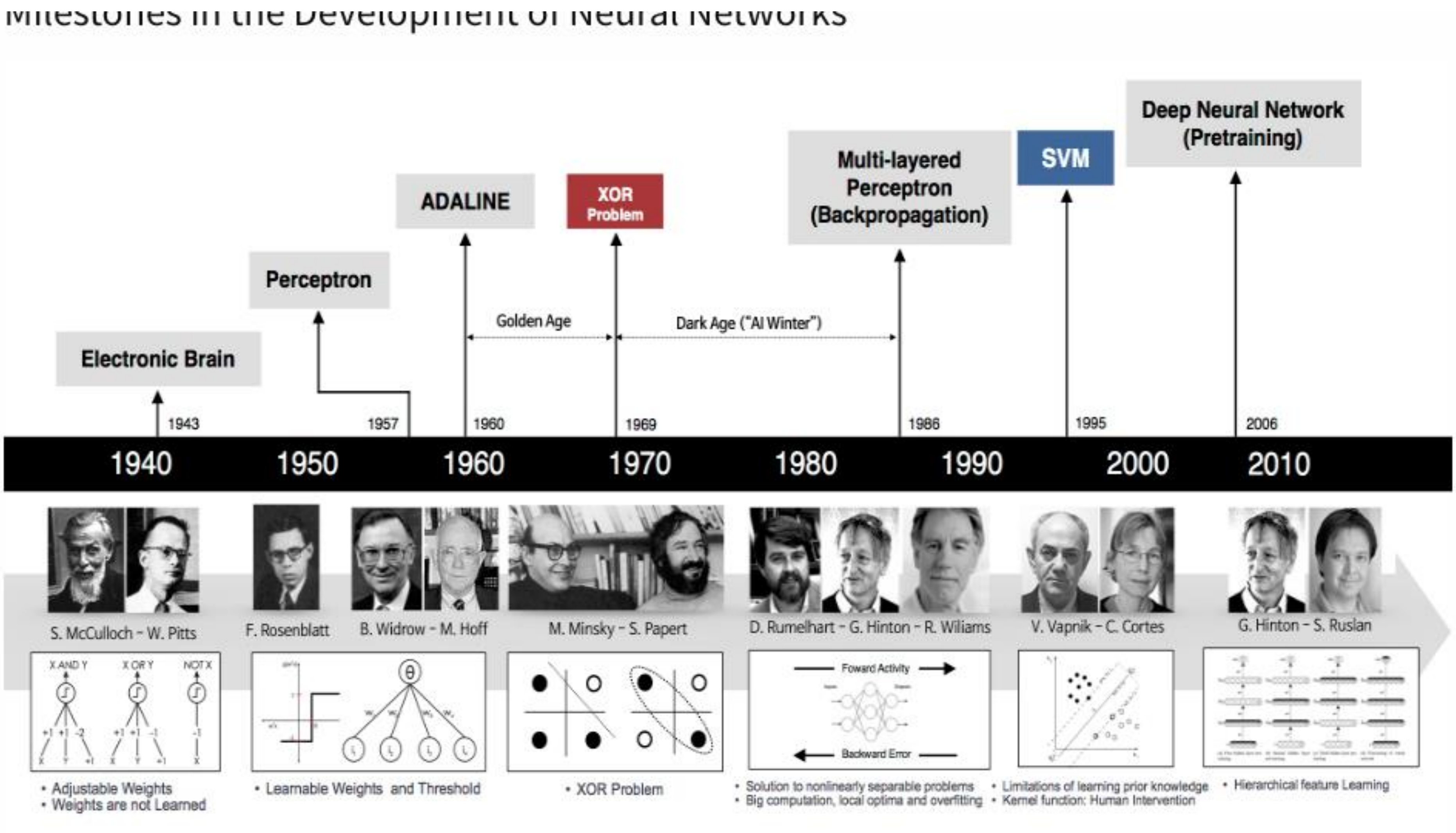
João Gama
jgama@fep.up.pt

November 2018

Summary

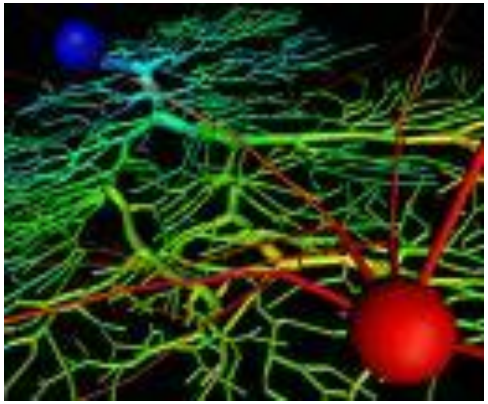
- Linear machines
  - Linear Machine (Perceptron)
  - Gradient Descent
  - Incremental vs. non-incremental (batch)
  - Extension to more than two classes
  - Properties of the algorithm
- Multi-layer networks
  - The sigmoid function
  - An architecture for the XOR
  - The Backpropagation algorithm (Backpropagation)
  - Extensions

Developments of Neural Networks



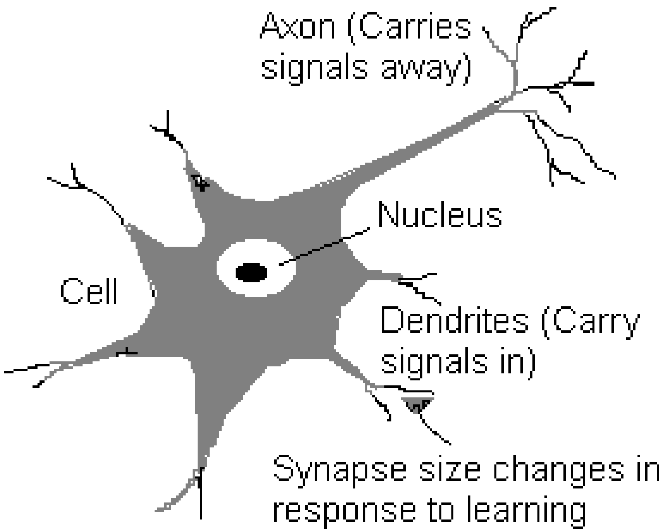
Perceptrons and Neural Nets

- Biological inspiration:
  - Taking the nervous system as reference, McCulloh Pits, 1943 present a model similar to perceptrons.
  - The training algorithm and the proof of convergence for perceptrons is presented by Rosenblatt in 1962.
  - Minsky and Papert showed that perceptrons can not represent XOR.
- In the '80s, Rumelhart and McClelland have the backpropagation, algorithm for training multilayer neural networks.
  - One of the algorithms used in pattern recognition.



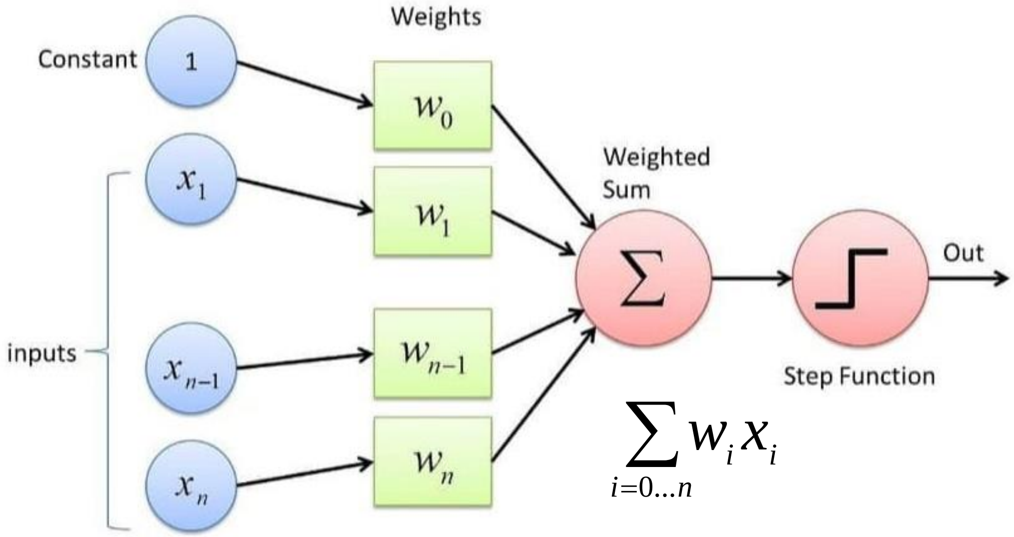
Inspiration from Neurobiology

- A neuron: many-inputs / one-output unit
- output can be *excited* or *not excited*
- incoming signals from other neurons determine if the neuron shall excite ("fire")
- Output subject to attenuation in the *synapses*, which are junction parts of the neuron



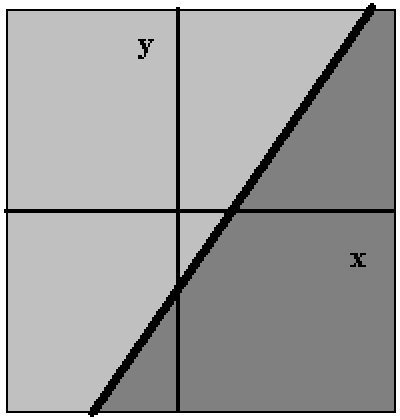
Linear Machines(*Perceptrons*)

- Linear Machine
  - $w_o+w_1x_1+w_2x_2+\ldots+w_nx_n$



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

Decision Surface:



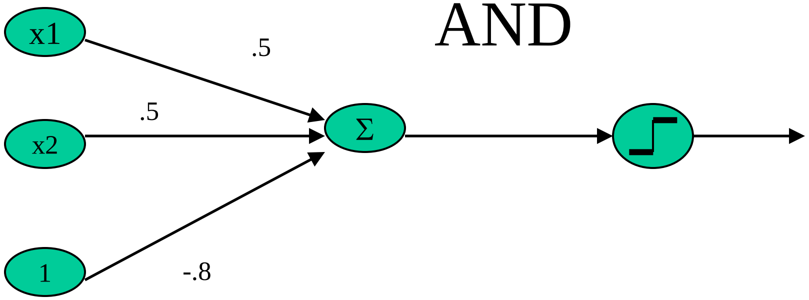
Vectorial Representation:

$$o(\vec{x}) = sig(\vec{w}.\vec{x})$$

Linear Machines

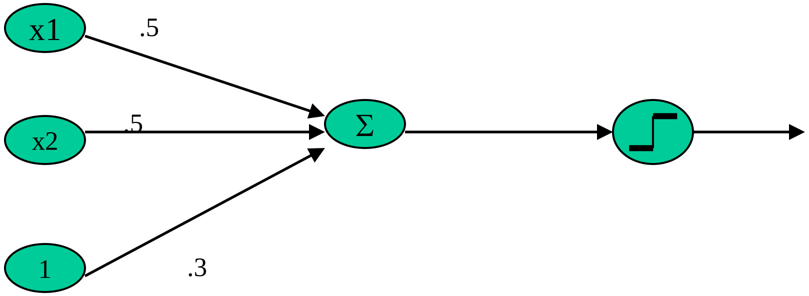
- Representation of Boolean functions

AND



	A	B	C	D	E	F	G	H	I
1	**x1**	**x2**	**y**		**w0**	**w1**	**w2**		
2	-1	-1	-1		-0,8	0,5	0,5		-1,8
3	-1	1	-1						-1,8
4	1	-1	-1						-0,8
5	1	1	1						0,2
6									
7									
8	**x1**	**x2**	**y**		**w0**	**w1**	**w2**		
9	-1	-1	-1		0,3	0,5	0,5		-0,7
10	-1	1	1						0,3
11	1	-1	1						0,3
12	1	1	1						1,3

OR



How to learn w_i?

Linear Machine

The idea behind the Algorithm:

- Initialize the weights with a small random number
    1. For each example **x**
        1. Compute the result of the linear machine: **sign(x . w)**
        2. If the result is correct, go to next example
        3. If predict is 1 and the observed value is 0 (false positive)
            1. Update the weights by subtracting a delta
        4. If predict is 0 and the observed value is 1 (false negative)
            1. Update the weights by adding a delta
    2. If any example has been misclassified go to step 1
    3. Otherwise, return the current value of the weights

– Updating the weights (delta rule):

$$w_i(t+1) = w_i(t) + \eta(Observed - \Pr edicted)x_i$$

$$where\ \eta \text{ is the learning rate}$$

Gradient Descent

- Assume a linear machine:

$$w_0+w_1*x_1+w_2*x_2+...+w_n*x_n$$

  - The goal is to learn the coefficients w_i that minimize the square error:

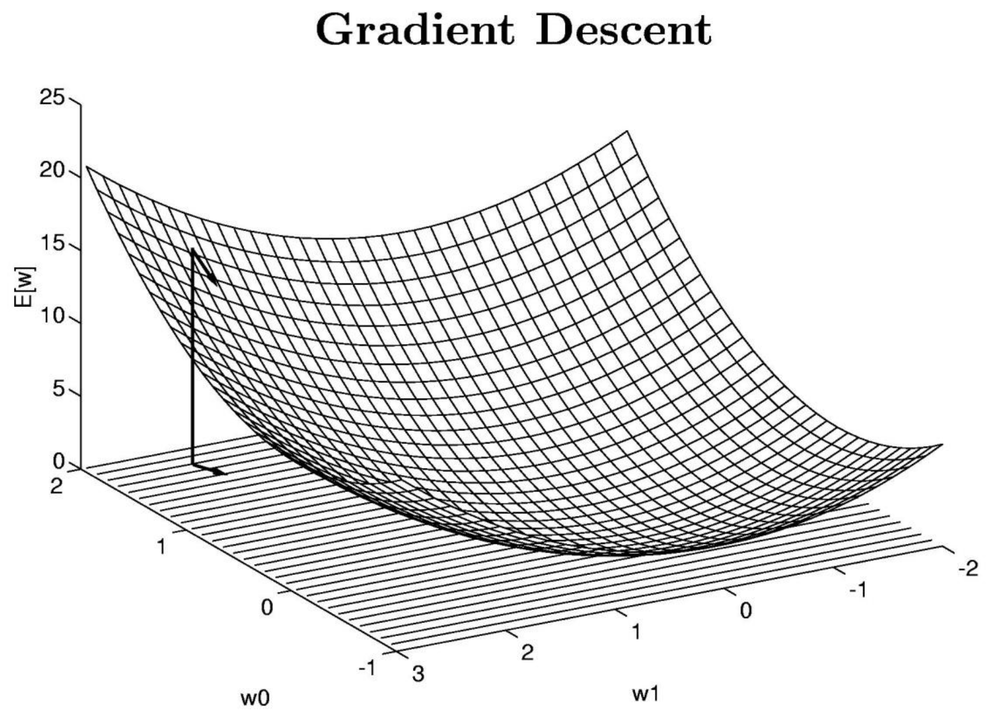
$$E(w_i) = ½ \Sigma(t_d-p_d)^2$$

  - Sum of the square of the differences bwtween the observed value (t_d) and the predict value (p_d)

  - Basic idea:
    - Changing the coefficients to reduce the error following the downward direction of the gradient.

Gradient Descent

- Error surface in the parameters space:
  - The square error defines a parabolic surface.



Deriving the Delta Rule

- The partial derivative of E(w) (with respect to each coefficient) can be computed as:

Gradient:

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \ldots, \frac{\partial E}{\partial w_n} \right]$$

Learning Rule:

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w}$$

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

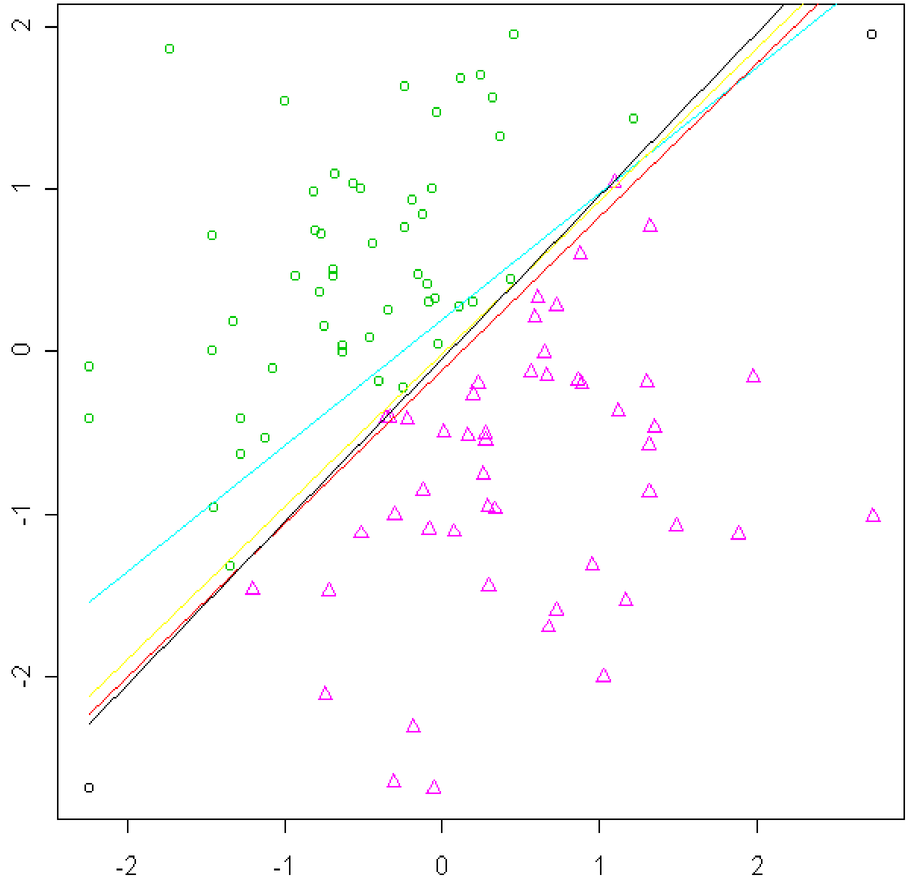
$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - p_d)^2$$

$$\frac{\partial E}{\partial w_i} = \sum (t_d - p_d)(-x_{id})$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - p_d) x_{id}$$

Analysis

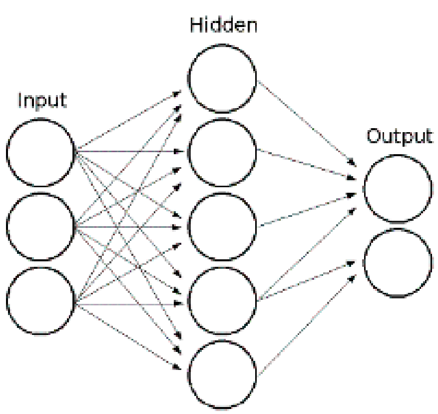
- Does not assume any distribution for the data
- Converge to a solution if the samples are linearly separable.
- The figure illustrates the convergence of a linear machine.
- A linear machine defines decision surfaces that are hyperplanes.
  - It is able to represent AND, OR, and other Boolean functions.
  - It is not capable of representing XOR

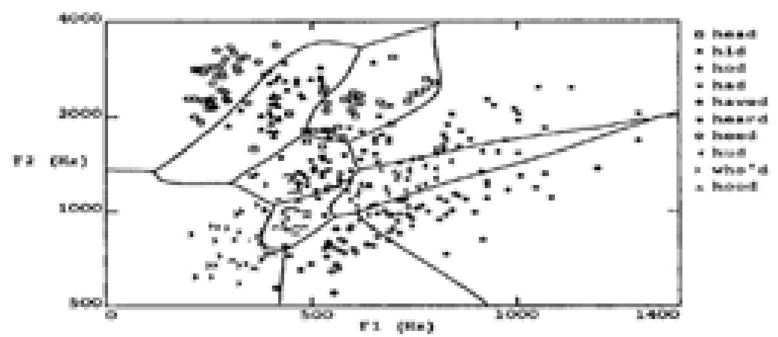


Multilayer Perceptrons

Multi-layer perceptrons

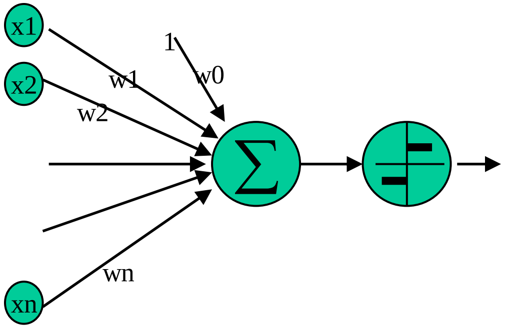
- Linear machines (as well as discriminant functions) only define linear decision surfaces.
- Structuring linear machines in layers is possible to define non-linear decision surfaces
  - The combination of the linear units is also linear.
  - Non-linear unit : the sigmoid function.





The sigmoid function

$$o(x) = \begin{cases} 1 \text{ sse x} > 0 \\ -1 \text{ sse x} < 0 \end{cases}$$

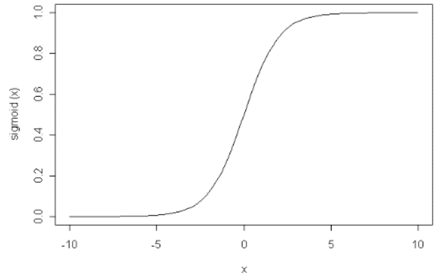


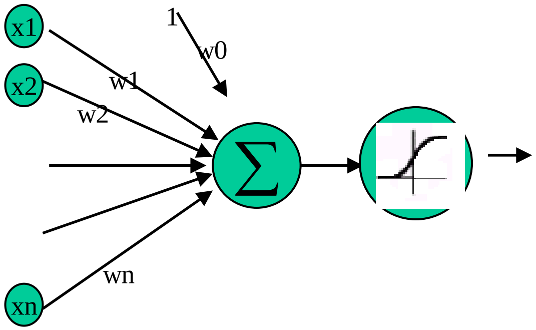
The sigmoid function:

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

$$x \in \Re$$

$$o(x) \in [0;1]$$





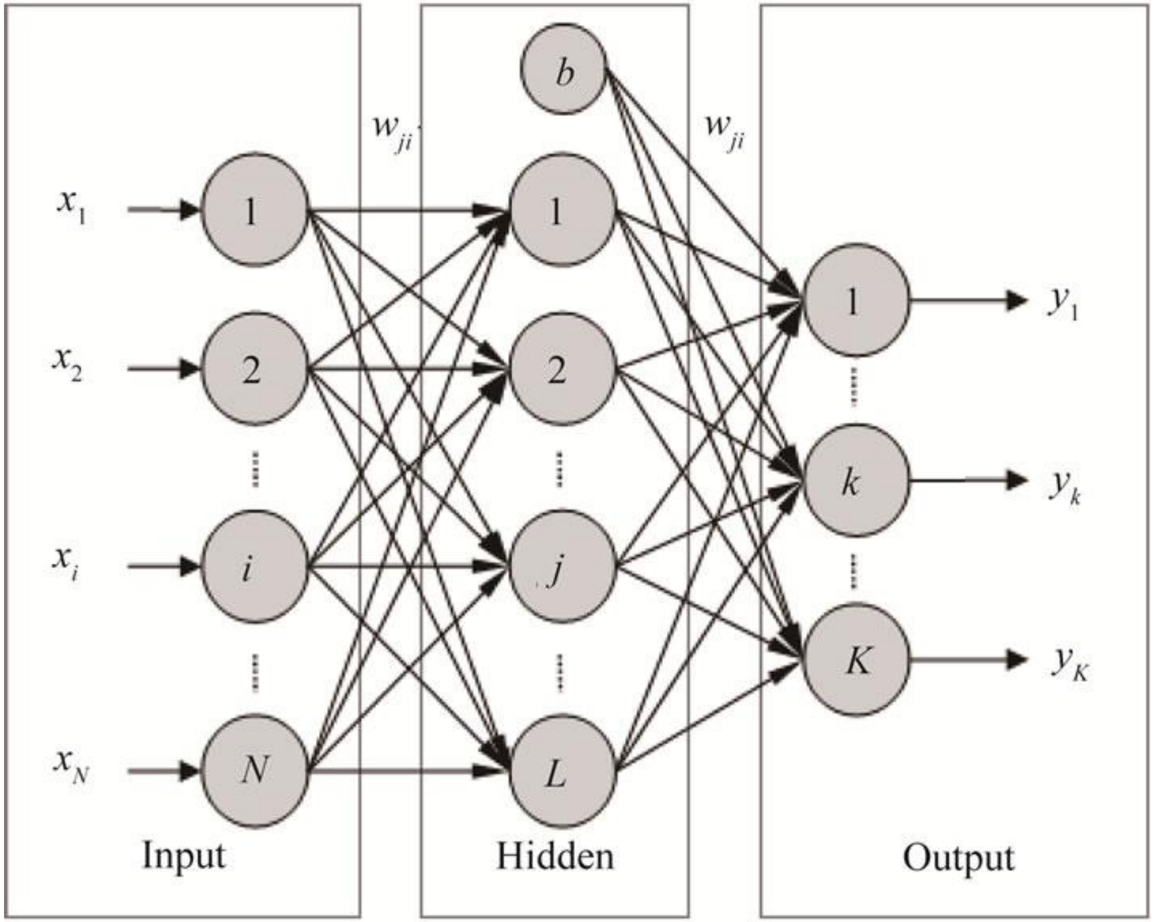
Interesting property:

$$\frac{\partial o(x)}{\partial x} = o(x)(1 - o(x))$$

Artificial Neural Networks

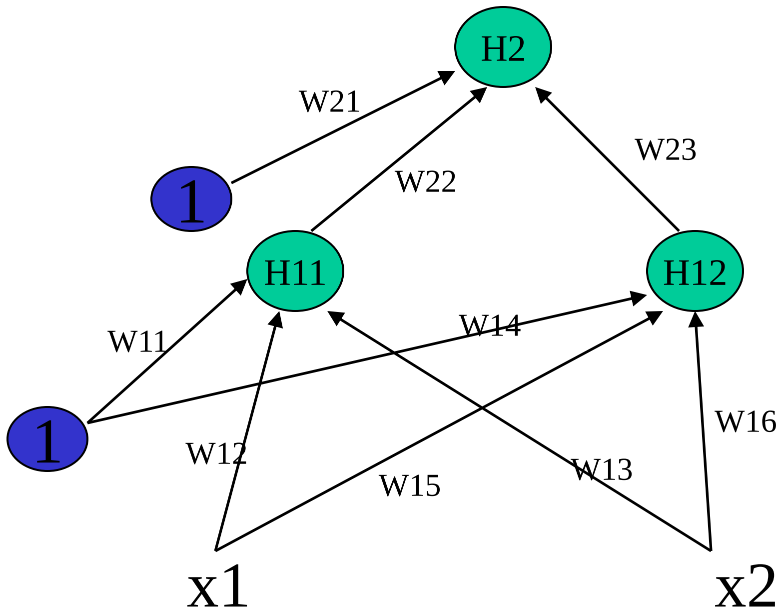
Adaptive interaction between individual neurons

Power: collective behavior of interconnected neurons

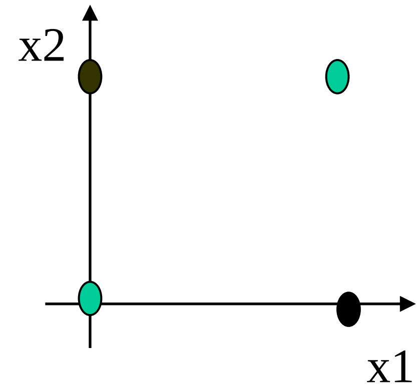


Solving XOR

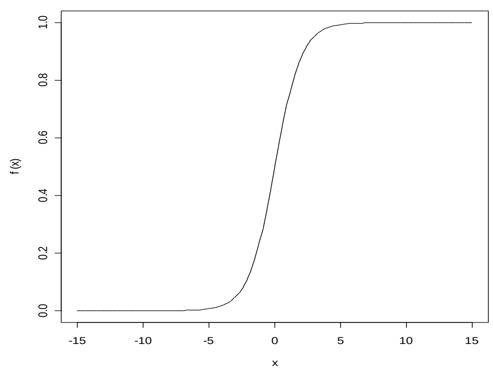
- A solution for XOR

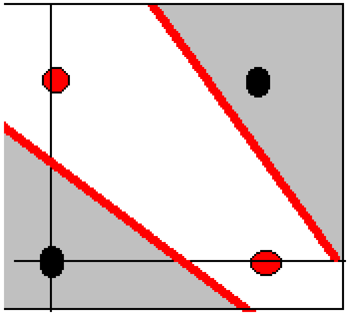


x1	**x2**	**Y**
0	0	0
0	+1	+1
+1	0	+1
+1	+1	0



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



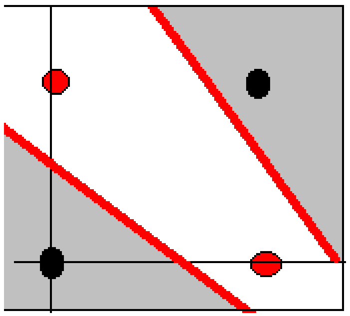


Solving Xor

After 1000 iterations:

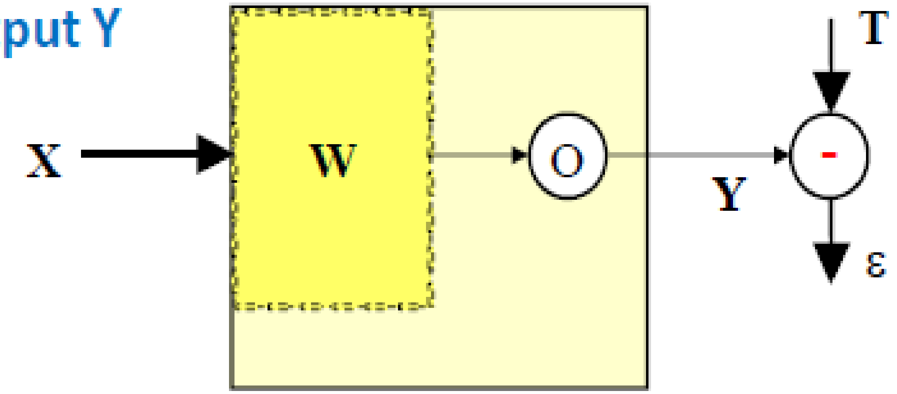
	A	B	C	D	E	F	G	H	I	J	K
1	**w11**	**w12**	**w13**	**w14**	**w15**	**w16**		**w21**	**w22**	**w23**	
2	5,308416	-3,63912	-3,64626	2,079875	-5,7996	-5,74641		-3,15041	7,022172	-7,40422	
3											
4	5,77		-3,92		-3,94	-5,91					
5											
6	**x1**	**x2**	**y**		**H1**		**H2**			**H3**	
7	0	0	0		5,308416	0,995075	2,079875	0,888932		-2,74467	0,060388
8	1	0	1		1,669296	0,841482	-3,71972	0,023667		2,583388	0,929785
9	0	1	1		1,662155	0,840527	-3,66654	0,024928		2,567349	0,92873
10	1	1	0		-1,97697	0,121643	-9,46613	7,74E-05		-2,29678	0,09139

The decision surface:

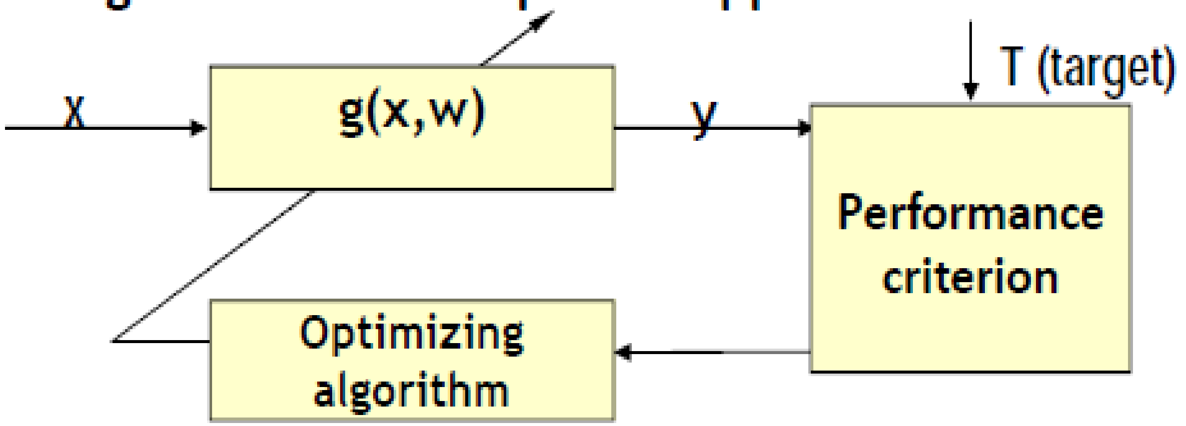


Training a MLP

A mapper relates input X with output Y
Parameters or weights W
Linear output neuron O
Target output T
Error ε

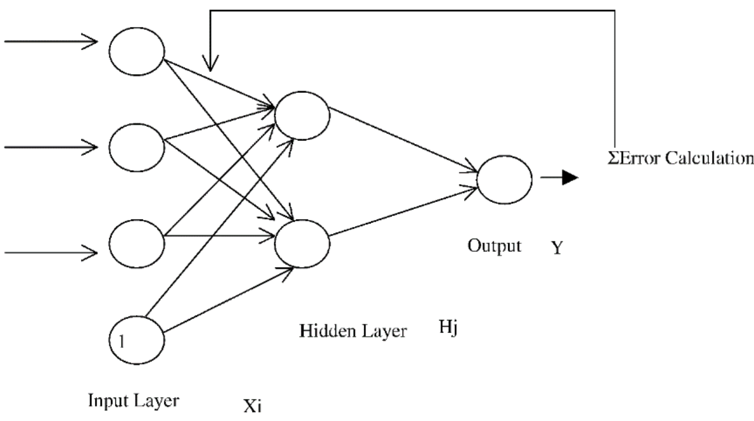


We can train ANN or other mappers with backpropagation and define a *cost* criterion to generate the adequate mapper



The Backpropagation Algorithm

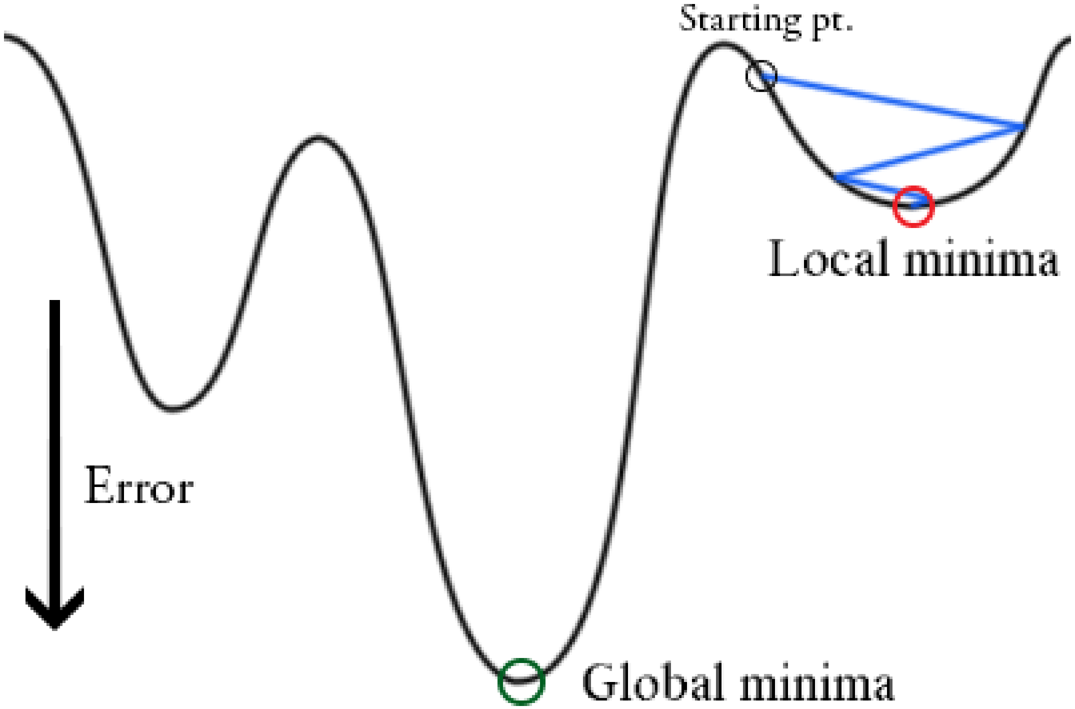
- Algorithm for a MLP with 3 layers



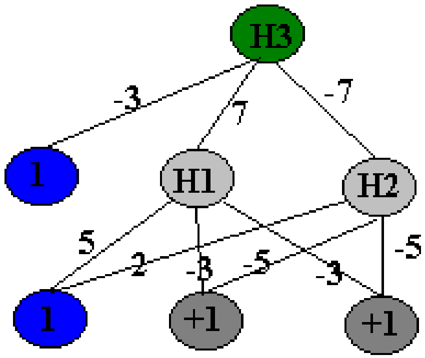
Actual algorithm for a 3-layer network (only one hidden layer):

```
initialize network weights (often small random values)
do
   forEach training example ex
      prediction = neural-net-output(network, ex)  // forward pass
      actual = teacher-output(ex)
      compute error (prediction - actual) at the output units
      compute Δw_h for all weights from hidden layer to output layer  // backward pass
      compute Δw_i for all weights from input layer to hidden layer   // backward pass continued
      update network weights
until all examples classified correctly or stopping criterion satisfied
return the network
```

Stochastic Gradient Descent



Illustrative example



Forward the example on the network:

Oh1 = 5-3*1-3*1 = -1 s(1) = 0.269

Oh2 = 2-5*1-5*1 = -8 s(8) = 0.0003

Oh3 = -3+7*0.269-7*0.0003=-1.119 s(1.119) = 0.25

Propagate the error backwards

dOh3 = 0.25*(1-0.25)*(0-0.25) = -0.046

dOh2 = 0.0003*(1-.0003)*(-7*.0.25) = 0.0006

dOh1 = .269 * (1-0.269)(7*0.25) = -0.34

Update the weights

w11 = 5 + -0.34*1

w12 = -3 + -0.34 * 1

w13= -3+ -0.34*1

....

Back propagation

- Desired output of the training examples
- Error = difference between actual & desired output
- Change weight relative to error size
- Calculate output layer error, then propagate back to previous layer
- Improved performance, very common!

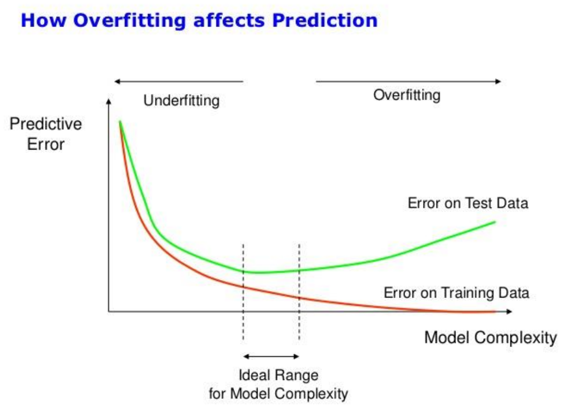
if the desired and actual output are both active or both inactive, increment the connection weight by the learning rate, otherwise decrement the weight by the learning rate.

The backpropagation algorithm

- The backpropagation algorithm implements a search using gradient descent in the space defined by the weights of the network.
  - In multi-layer networks of the error surface may contain several local minima.
  - Not guarantee to find a global minimum.
  - In practice, often works well (can run multiple times)
  - Easily generalized to arbitrary directed graphs
- A black-box. Hard to explain or gain intuition.
- It is possible to use the algorithm either
  - Non-incremental version: correction of the weights after each epoch
  - Incremental (stochastic) version: correction of weights after seeing an example
    - More effective to escape from local minima
- Minimizes error over training examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations !
  - slow!
  - Using network after training is very fast

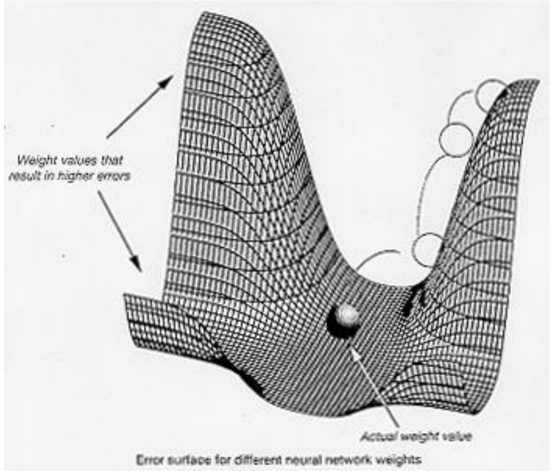
Overfiting

- When should we finish training the network?
  - Stopping Criteria:
    - If stopping too early we run the risk of getting a network not yet trained.
    - If stopping too late: danger of overfiting (adjustment to noise in the data)
  - Usual criteria:
    - Based on the error in the training set
      - When the error in the training set is below a certain limit.
  - Error based on a evaluation set (independent from the training set)
    - When the error on the validation set has reached a minimum.



Issues

- The definition of the network topology can be problematic
  - The number of nodes in the hidden layer
    - Few nodes: underfitting
    - Many nodes: overfitting
  - There are no criteria for defining the number of nodes in the hidden layer
  - Effect of learning rate
    - A learning rate
      - Little has the effect of learning times higher
      - High may lead to non-convergence.



When to Consider Neural Networks

- Use when:
  - Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
  - Output is discrete or real valued
  - Output is a vector of values
  - Possibly noisy data
  - Form of target function is unknown
  - Human readability of result is unimportant
- Examples:
  - Speech phoneme recognition [Waibel]
  - Image classification [Kanade, Baluja, Rowley]
  - Financial prediction

Generalization vs. specialization

- Optimal number of hidden neurons
  - Too many hidden neurons: you get an over fit, training set is memorized, thus making the network useless on new data sets
  - Not enough hidden neurons:
    network is unable to learn problem concept

- Overtraining:
  - Too much examples, the ANN memorizes the examples instead of the general idea
- Generalization vs. specialization trade-off:

 # hidden nodes & training samples

Tips

- Initialize the weights with small random values
  - [-0.05;0.05]
- Shuffling the training set between epochs
  - Change the sequence of the examples
- The learning rate must start with a high value that decreases progressively
- Train the network several times using different initialization of the weights

Expressive Capabilities

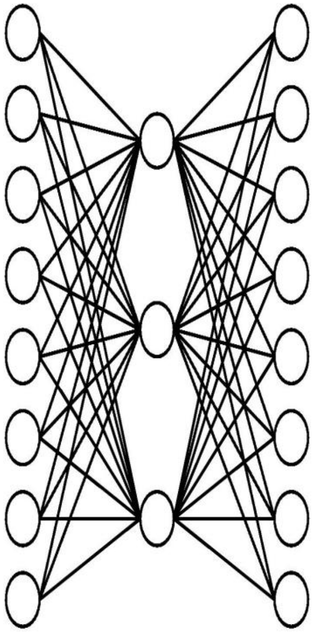
- Representation
  - The multilayer networks (MLP) can approach any function:
    - Boolean Functions
      - Any Boolean function can be represented by a network with one hidden layer
    - Continuous functions
      - Any function can be approximated remains limited (with an arbitrarily small error) for a network with one hidden layer (using sigmoid) and a drive (not run) output.
    - Arbitrary functions
      - Any function can be approximated (with an arbitrarily small error) by a network with two hidden layers.
- Capacity
  - the amount of information that can be stored in the network
  - The capacity of a neural network is *dense*.
    - ability to learn any function given enough data

Properties

- Generalization
  - The generalization ability of examples not used for training raises problems of overfitting and over-search.
  - These problems arise when the network capacity significantly exceeds the number of free parameters needed .
- Convergence
  - There may exist many local minima. This depends on the cost function and the model.
  - The optimization method used might not be guaranteed to converge when far away from a local minimum.
  - For a very large amount of data or parameters, some methods become impractical. In general, it has been found that theoretical guarantees regarding convergence are an unreliable guide to practical application.

Autoencoders

Learning Hidden Layer Representations



Learned hidden layer representation:

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

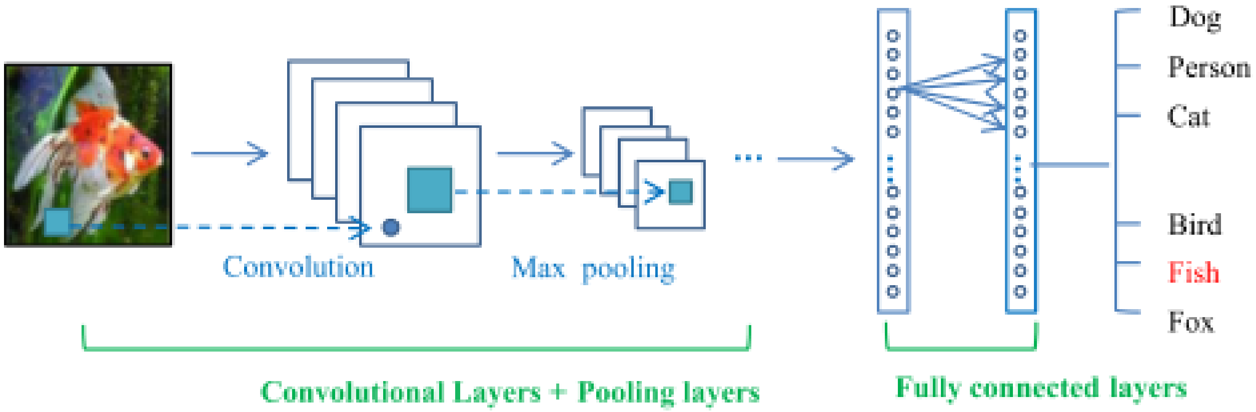
Deep Learning

Aims to discover multiple levels of distributed representations.
It relies on hierarchical architectures to learn high-level abstractions in data

Raw Input

Feature Extraction Construction

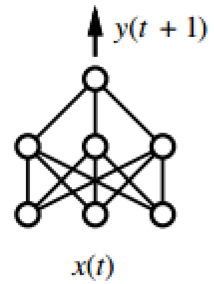
Classifier



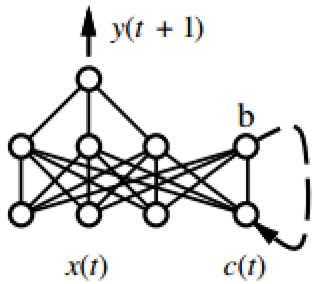
General CNN architecture (Guo et al., 2016)

Developments

- The *Cascade Correlation* architecture
  - Neural network that adds new neurons in the hidden layer has the training proceeds.
- Recurrent Networks



(a) Feedforward network



(b) Recurrent network

- Redes Kohonen
  - SOM (*self-organization maps*)

- Bibliography
  - Tom Mitchell, *Machine Learning*, McGraw-Hill, 1997

R – library(nnet)

Fit Neural Networks

Description

Fit single-hidden-layer neural network, possibly with skip-layer connections.

Usage

```
nnet(x, ...)

## S3 method for class 'formula':
nnet(formula, data, weights, ...,
     subset, na.action, contrasts = NULL)

## Default S3 method:
nnet(x, y, weights, size, Wts, mask,
     linout = FALSE, entropy = FALSE, softmax = FALSE,
     censored = FALSE, skip = FALSE, rang = 0.7, decay = 0,
     maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000,
     abstol = 1.0e-4, reltol = 1.0e-8, ...)
```

Arguments

`formula`	A formula of the form `class ~ x1 + x2 + ...`
`x`	matrix or data frame of `x` values for examples.
`y`	matrix or data frame of target values for examples.
`weights`	(case) weights for each example – if missing defaults to 1.
`size`	number of units in the hidden layer. Can be zero if there are skip-layer units.
`data`	Data frame from which variables specified in `formula` are preferentially to be taken.
`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)

ANN in R

- Run the following code in R
  - library(nnet)
  - data(Boston, package = "MASS")
  - p <- sample(nrow(Boston), 0.7 * nrow(Boston))
  - train <- Boston[p,]
  - test <- Boston[-p,]
  - nn <- nnet(medv ~ ., train, size = 20, decay = 0.001, maxit = 1000,linout = T)
  - prevs <- predict(nn, test)
  - mae.nn <- mean(abs(prevs - test[, "medv"]))
  - mse.nn <- mean((prevs - test[, "medv"])^2)

  - plot(teste[, "medv"], prevs, main = “Neural Net Predictions", ylab = “Observed Values")
  - abline(0, 1, lty = 2, col = "red“)

- Try for different configurations of the ANN

R – library(AMORE)

newff(AMORE) R Documentation

Create a Multilayer Feedforward Neural Network

Description

Creates a feedforward artificial neural network according to the structure established by the AMORE package standard.

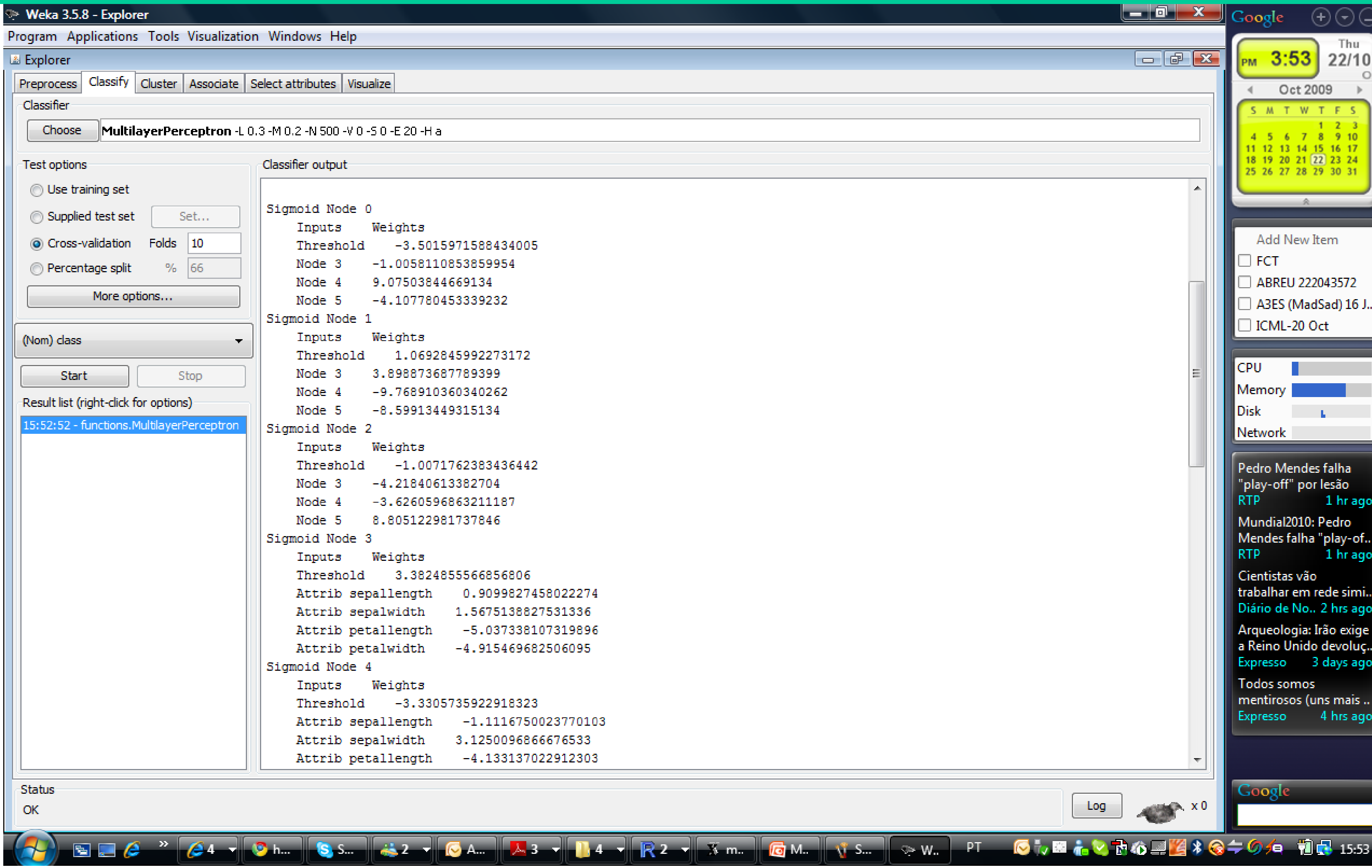
Usage

```
newff(n.neurons, learning.rate.global, momentum.global, error.criterium, Stao, hidden.layer, output.layer, method)
```

Arguments

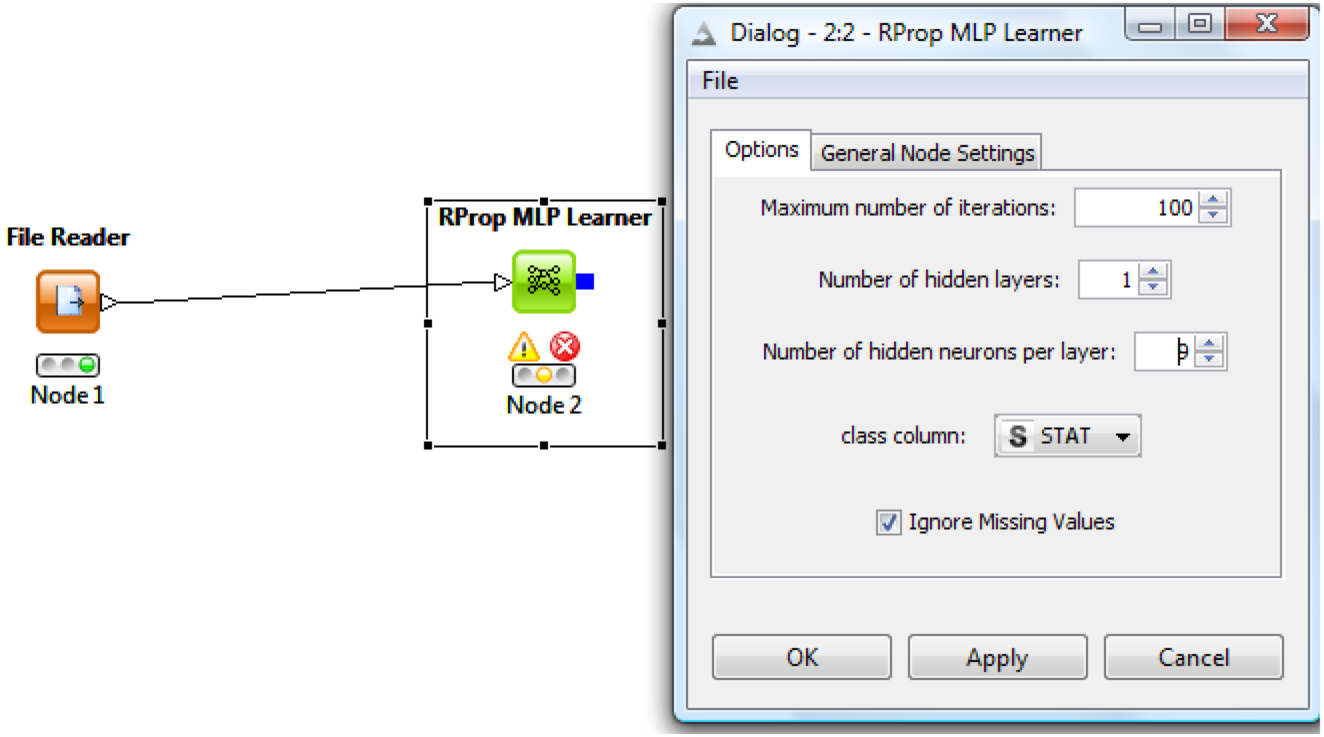
`n.neurons`	Numeric vector containing the number of neurons of each layer. The first element of the vector is the number of input neurons, the last is the number of output neurons and the rest are the number of neuron of the different hidden layers.
`learning.rate.global`	Learning rate at which every neuron is trained.
`momentum.global`	Momentum for every neuron. Needed by several training methods.
`error.criterium`	Criterium used to measure to proximity of the neural network prediction to its target. Currently we can choose amongst: • "LMS": Least Mean Squares. • "LMLS": Least Mean Logarithm Squared (Liano 1996). • "TAO": TAO Error (Pernia, 2004).
`Stao`	Stao parameter for the TAO error criterium. Unused by the rest of criteria.
`hidden.layer`	Activation function of the hidden layer neurons. Available functions are: • "purelin". • "tansig". • "sigmoid".

WEKA



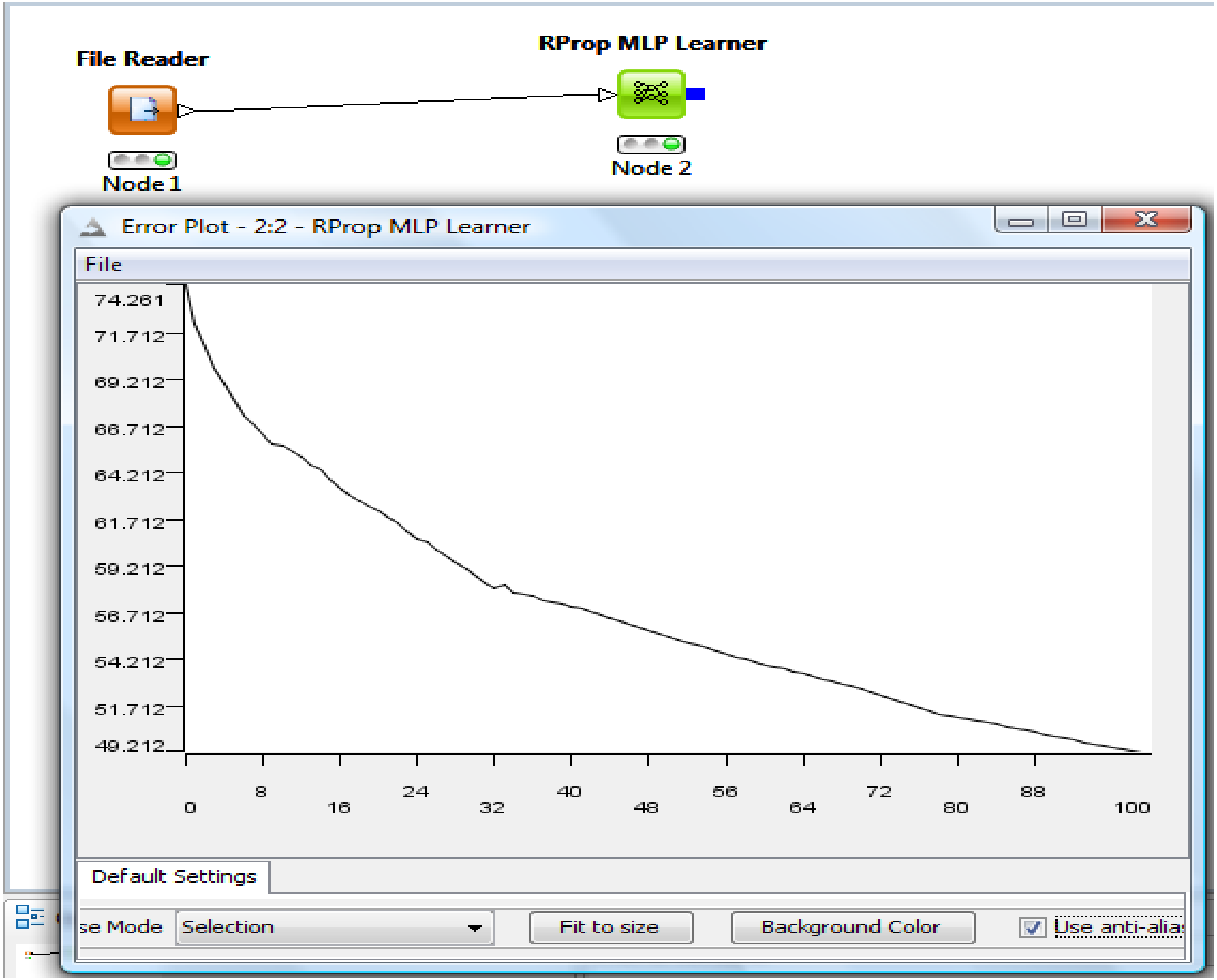
Weka 3.5.8 - Explorer
Program Applications Tools Visualization Windows Help
Explorer
Preprocess Classify Cluster Associate Select attributes Visualize
Classifier
Choose
MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
Test options
Use training set
Supplied test set
Set...
Cross-validation Folds 10
Percentage split % 66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
15:52:52 - functions.MultilayerPerceptron
Classifier output
Sigmoid Node 0
Inputs Weights
Threshold -3.5015971588434005
Node 3 -1.0058110853859954
Node 4 9.07503844669134
Node 5 -4.107780453339232
Sigmoid Node 1
Inputs Weights
Threshold 1.0692845992273172
Node 3 3.898873687789399
Node 4 -9.768910360340262
Node 5 -8.59913449315134
Sigmoid Node 2
Inputs Weights
Threshold -1.0071762383436442
Node 3 -4.21840613382704
Node 4 -3.6260596863211187
Node 5 8.805122981737846
Sigmoid Node 3
Inputs Weights
Threshold 3.3824855566856806
Attrib sepallength 0.9099827458022274
Attrib sepalwidth 1.5675138827531336
Attrib petallength -5.037338107319896
Attrib petalwidth -4.915469682506095
Sigmoid Node 4
Inputs Weights
Threshold -3.3305735922918323
Attrib sepallength -1.1116750023770103
Attrib sepalwidth 3.1250096866676533
Attrib petallength -4.13313702291230
Status
OK
Log
x 0
Google
PM 3:53
Thu 22/10
Oct 2009
Add New Item
FCT
ABREU 222043572
A3ES (MadSad) 16 J..
ICML-20 Oct
CPU
Memory
Disk
Network
Pedro Mendes falha "play-off" por lesão
RTP 1 hr ago
Mundial2010: Pedro Mendes falha "play-of..
RTP 1 hr ago
Cientistas vão trabalhar em rede simi..
Diário de No.. 2 hrs ago
Arqueologia: Irão exige a Reino Unido devoluç..
Expresso 3 days ago
Todos somos mentirosos (uns mais ..
Expresso 4 hrs ago
Google
PT
15:53

KNIME



File Reader
Node 1
RProp MLP Learner
Node 2
Dialog - 2:2 - RProp MLP Learner
File
Options
General Node Settings
Maximum number of iterations:
100
Number of hidden layers:
1
Number of hidden neurons per layer:
class column:
S STAT
Ignore Missing Values
OK
Apply
Cancel

KNIME



File Reader
RProp MLP Learner
Node 1
Node 2
Error Plot - 2:2 - RProp MLP Learner
File
74.261
71.712
69.212
66.712
64.212
61.712
59.212
56.712
54.212
51.712
49.212
0
8
16
24
32
40
48
56
64
72
80
88
100
Default Settings
se Mode
Selection
Fit to size
Background Color
Use anti-alia

Summary

- Linear Machines
  - Perceptrons
  - Gradient Descent
    - Derivtion of the delta rule
    - Incremental versus não-incremental (batch)
  - Extension for more than 2 classes
  - Main properties
- Multi-layer networks
  - The sigmoide function
  - The backpropagation algorithm
- Analisis
  - Representation and Generalization