Artificial Neural Network

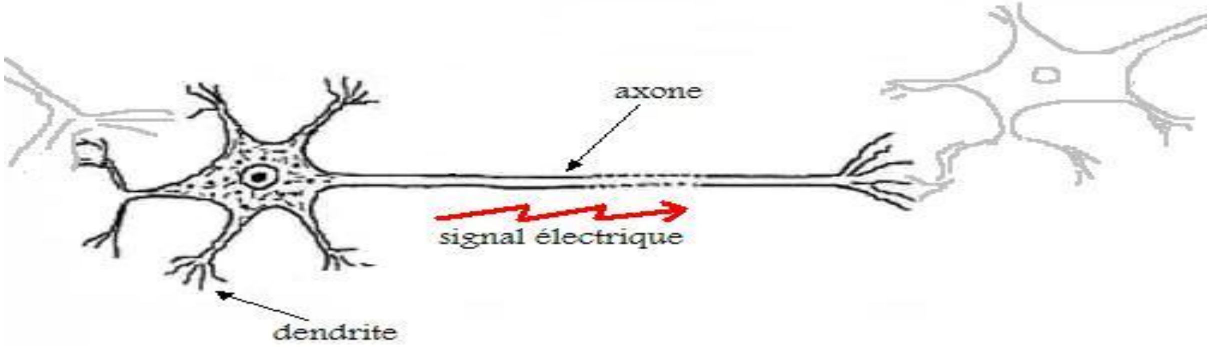
Single-layer and multilayer perceptrons

Neural network for supervised learning

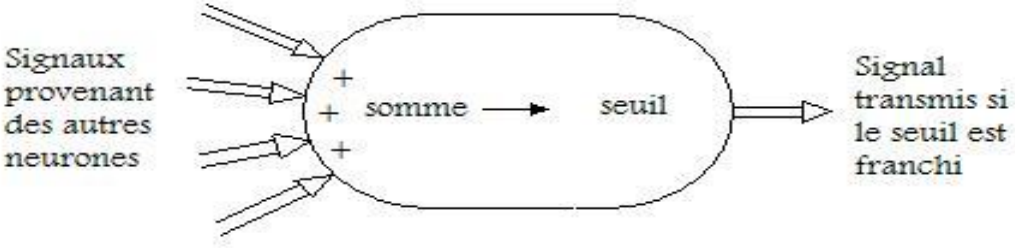
Ricco RAKOTOMALALA

Biological metaphor

Human brain working
Transmission of information and learning process



Important things to retain

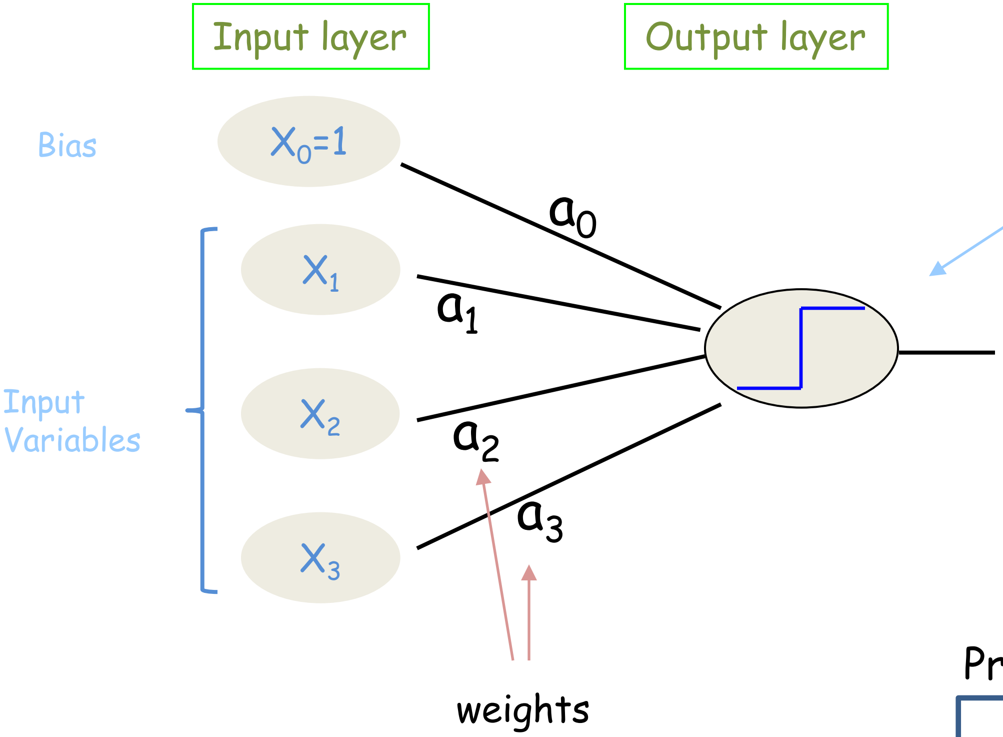

Signaux provenant des autres neurones
+
+
+
somme
seuil
Signal transmis si le seuil est franchi

- Receiving information (signal)
- Activation and processing by a neuron
- Transmission to other neurons (if the signal is enough strong)
- In the long run: strengthening of some connections → LEARNING

Mc Colluch and Pitts' model
Single-layer perceptron

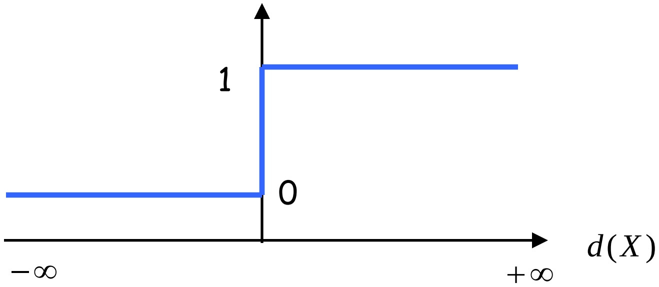
Binary problem (positive vs. negative)

$$Y \in \{1\ (+), 0\ (-)\}$$



Transfer function

Activation function – Heaviside step function



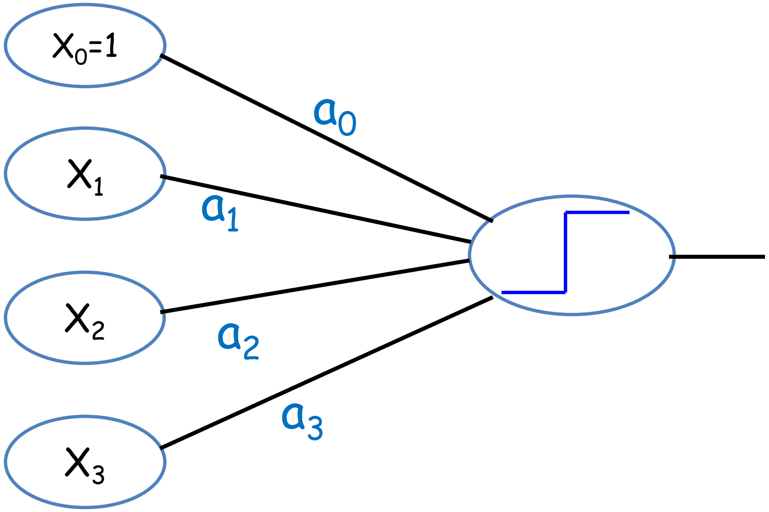
Prediction model and classification rule

$$d(X) = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3$$

$$\text{IF } d(X) > 0 \text{ THEN } Y = 1 \text{ ELSE } Y = 0$$

The single-layer perceptron is a linear classifier !

Learning algorithm – Single-layer perceptron



? How to calculate the weights from a data set (Y ; X1, X2, X3)

Draw a parallel with the least squares regression. A neural network can be used for the regression (linear transfer function)

(1) Criterion to optimize?

(2) Optimization process?

⇒

(1) Minimizing of the prediction error

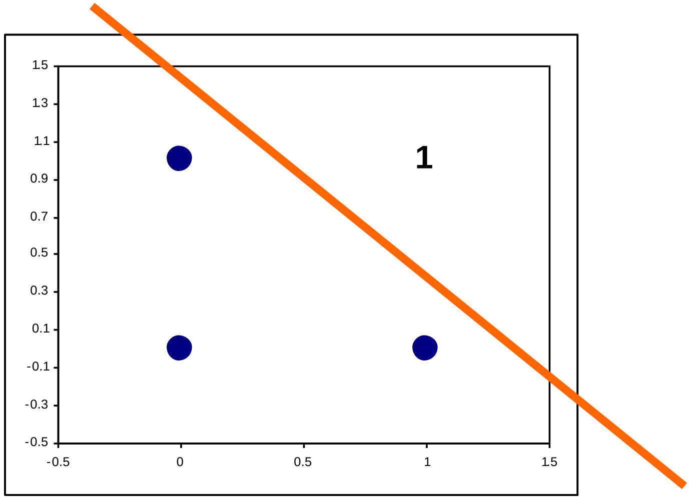
(2) Error correction learning procedure

Example – Learning the logical AND function

Instructive example - The first applications are from the computer science area.

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Dataset



2D representation (scatter plot)

Main steps:

1. Mix up randomly the instances of the learning set
2. Initialize the weights (small random value)
3. For each instance of the training set
   - Calculate the output of the perceptron
   - If the prediction is wrong, update the weights
4. Until convergence (termination condition is satisfied)

Sequential learning procedure

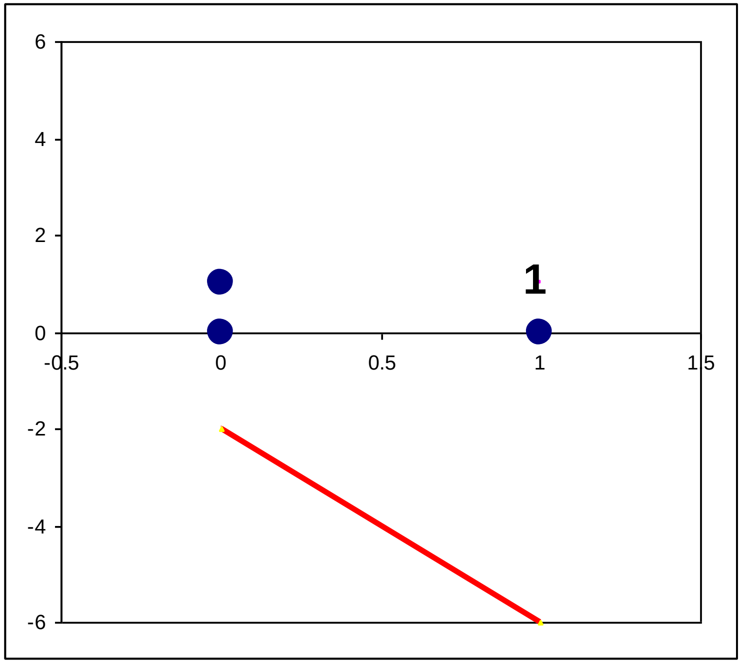
An instance may be processed a few times !!!

Example AND (1)

Initialize (randomly) the weights: $a_0 = 0.1; a_1 = 0.2; a_2 = 0.05$

Decision boundary :

$$0.1 + 0.2x_1 + 0.05x_2 = 0 \Leftrightarrow x2 = -4.0x_1 - 2.0$$



Update rule of the weights

For each instance which is processed

$$a_j \leftarrow a_j + \Delta a_j$$

with

$$\Delta a_j = \eta(y - \hat{y})x_j$$

Strength of the signal

Error

It enables to determine if we correct the weights or not

Learning rate parameter

It determines the intensity of the correction

What is the good value?

Too small → processing is too slow

Too high → oscillation

A rule of thumb, about 0.05 ~ 0.15 (0.1 for our example)

Example AND (2)

One instance of the dataset

$$\begin{cases} x_0 = 1 \\ x_1 = 0 \\ x_2 = 0 \\ y = 0 \end{cases}$$

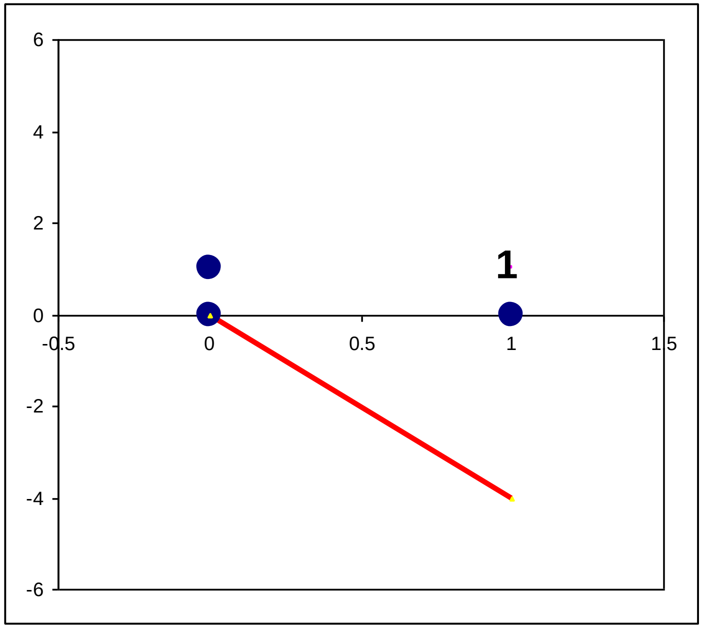
Calculate the output

$$0.1 \times 1 + 0.2 \times 0 + 0.05 \times 0 = 0.1$$
$$\Rightarrow \hat{y} = 1$$

Error => Update the weights

$$\begin{cases} \Delta a_0 = 0.1 \times (-1) \times 1 = -0.1 \\ \Delta a_1 = 0.1 \times (-1) \times 0 = 0 \\ \Delta a_2 = 0.1 \times (-1) \times 0 = 0 \end{cases}$$

New decision boundary: $0.0 + 0.2 x_1 + 0.05 x_2 = 0 \Leftrightarrow x_2 = -4.0 x_1 + 0.0$



Example AND (3)

Other instance

$$\begin{cases} x_0 = 1 \\ x_1 = 1 \\ x_2 = 0 \\ y = 0 \end{cases}$$

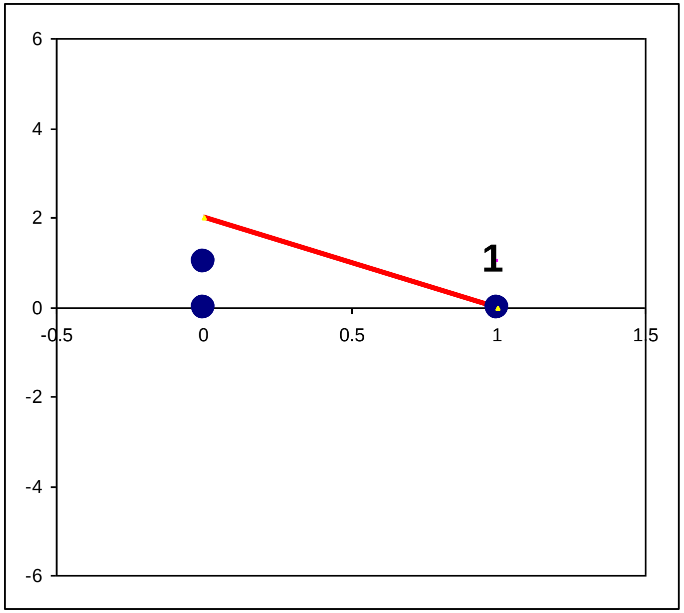
Calculate the output

$$0.0 \times 1 + 0.2 \times 1 + 0.05 \times 0 = 0.2$$
$$\Rightarrow \hat{y} = 1$$

Error => update the weights

$$\begin{cases} \Delta a_0 = 0.1 \times (-1) \times 1 = -0.1 \\ \Delta a_1 = 0.1 \times (-1) \times 1 = -0.1 \\ \Delta a_2 = 0.1 \times (-1) \times 0 = 0 \end{cases}$$

New decision boundary: $-0.1 + 0.1x_1 + 0.05x_2 = 0 \Leftrightarrow x_2 = -2.0x_1 + 2.0$



Example AND (4) – Termination condition

Other instance

$$\begin{cases} x_0 = 1 \\ x_1 = 0 \\ x_2 = 1 \\ y = 0 \end{cases}$$

Calculate the output

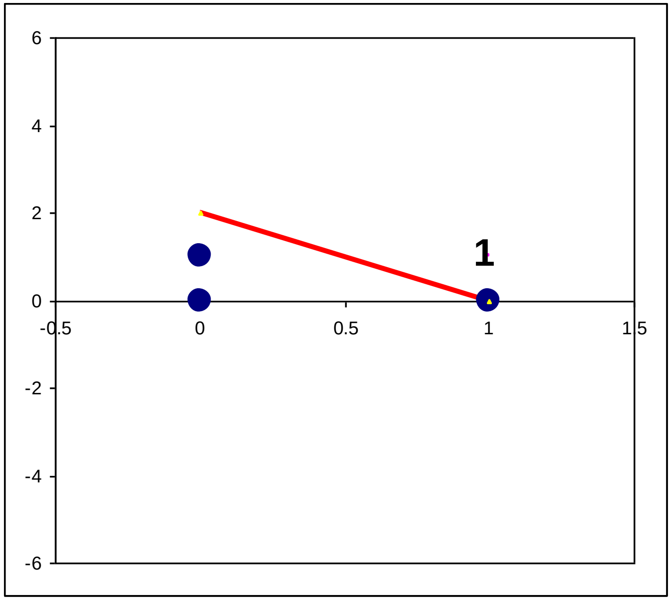
$$-0.1\times1+0.1\times0+0.05\times1=-0.05$$
$$\Rightarrow \hat{y}=0$$

Good prediction => no update

$$\begin{cases} \Delta a_0 = 0.1\times(0)\times1=0 \\ \Delta a_1 = 0.1\times(0)\times0=0 \\ \Delta a_2 = 0.1\times(0)\times1=0 \end{cases}$$

No correction here. Why? See the decision boundary in the scatter plot.

Decision boundary: $-0.1+0.1x_1+0.05x_2=0 \Leftrightarrow x_2=-2.0x_1+2.0$



Note: What happens if we process again (x1=1 ; x2=0)?

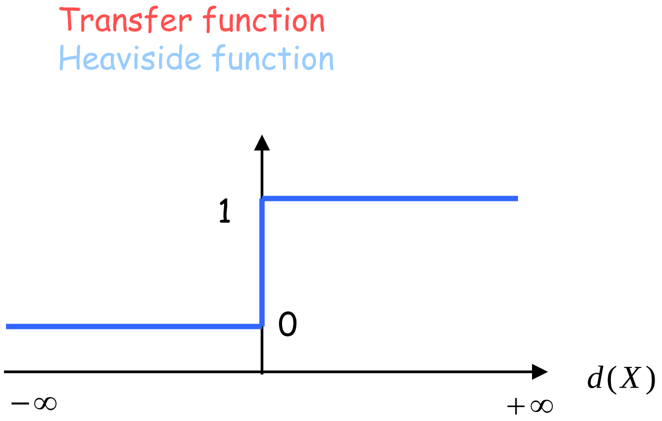
Convergence?

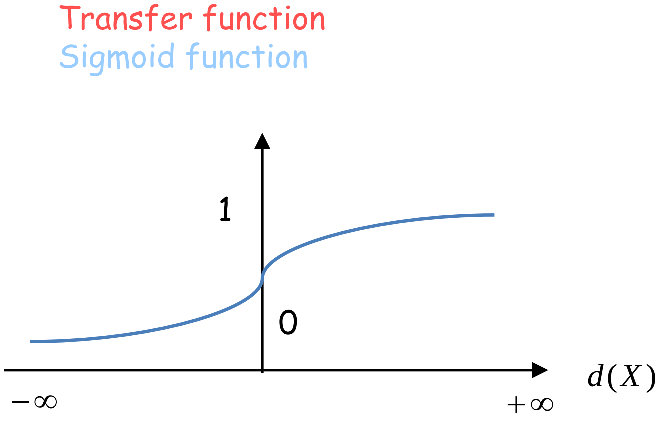
(1) No correction is made whatever the instance handled
(2) The error rate no longer decreases "significantly"
(3) The weights are "stable"
(4) We set a maximum number of iterations
(5) We set a minimum error to achieve

Estimation of the conditional probability **P(Y/X)**

Sigmoid transfer function

The perceptron provides a classification rule
But in some circumstances, we need the estimation of P(Y/X)

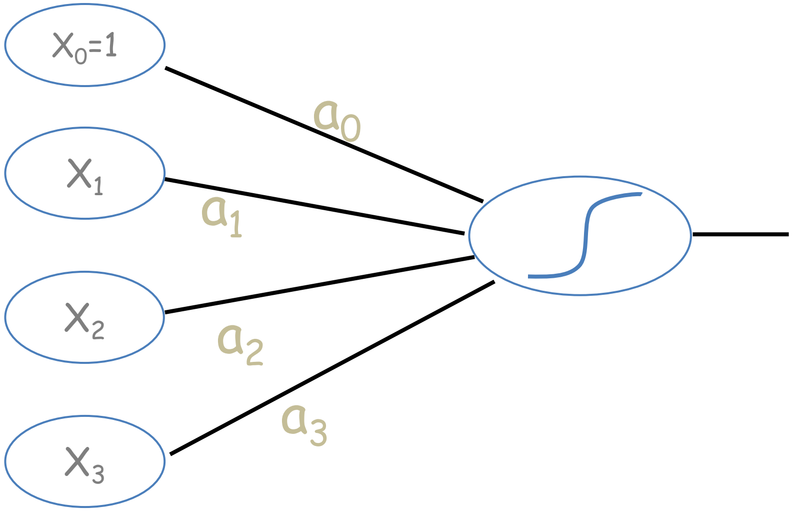




$$g(v) = \frac{1}{1+e^{-v}}$$
$$v = d(X)$$

The decision rule becomes: IF g(v) > 0.5 THEN Y=1 ELSE Y=0

Consequence of the using a derivable real function as activation function
Modification of the optimization criterion



Output of the network

$$\hat{y} = g(v) = f\left[d(x)\right]$$

Least mean squares criterion

$$E = \frac{1}{2}\sum_{\omega\in\Omega}\left(y(\omega) - \hat{y}(\omega)\right)^2$$

But we use always the sequential learning procedure!

Gradient descent optimization algorithm

The derivative of the sigmoid function

$$g'(v) = g(v)(1 - g(v))$$

Optimization: derivative of the objective function (criterion) with respect to the weights

$$\frac{\partial E}{\partial a_j} = -\sum_i [y(\omega) - \hat{y}(\omega)] \times g'[v(\omega)] \times x_j(\omega)$$

Update rule of the weights for each processed instance (Widrow-Hoff learning rule or Delta rule)

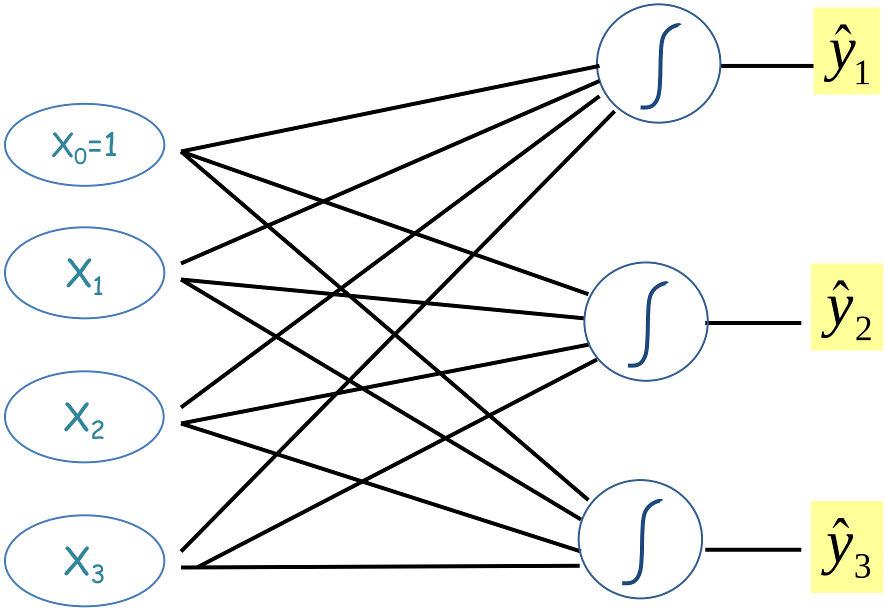
$$a_j \leftarrow a_j + \eta \underbrace{(y - \hat{y}) g'(v) x_j}$$

Gradient: Computing the weights in the direction which minimizes E

+

The convergence toward the minimum is good in practice
Ability to handle correlated input variables
Ability to handle large datasets (rows and columns)
Updating the model is easy if new labeled instances are available

Multiclass perceptron (K the number of classes, K > 2)



(1) Dummy coding of the output

$$y_k = 1 \ \ iif \ \ y = y_k$$

(2) « Output » for each neuron in output layer

$$\hat{y}_k = g[v_k]$$

with $v_k = a_{0,k} + a_{1,k} x_1 + \cdots + a_{J,k} x_J$

(3) P(Y/X) $P(Y = y_k / X) \propto g[v_k]$

(4) Classification rule

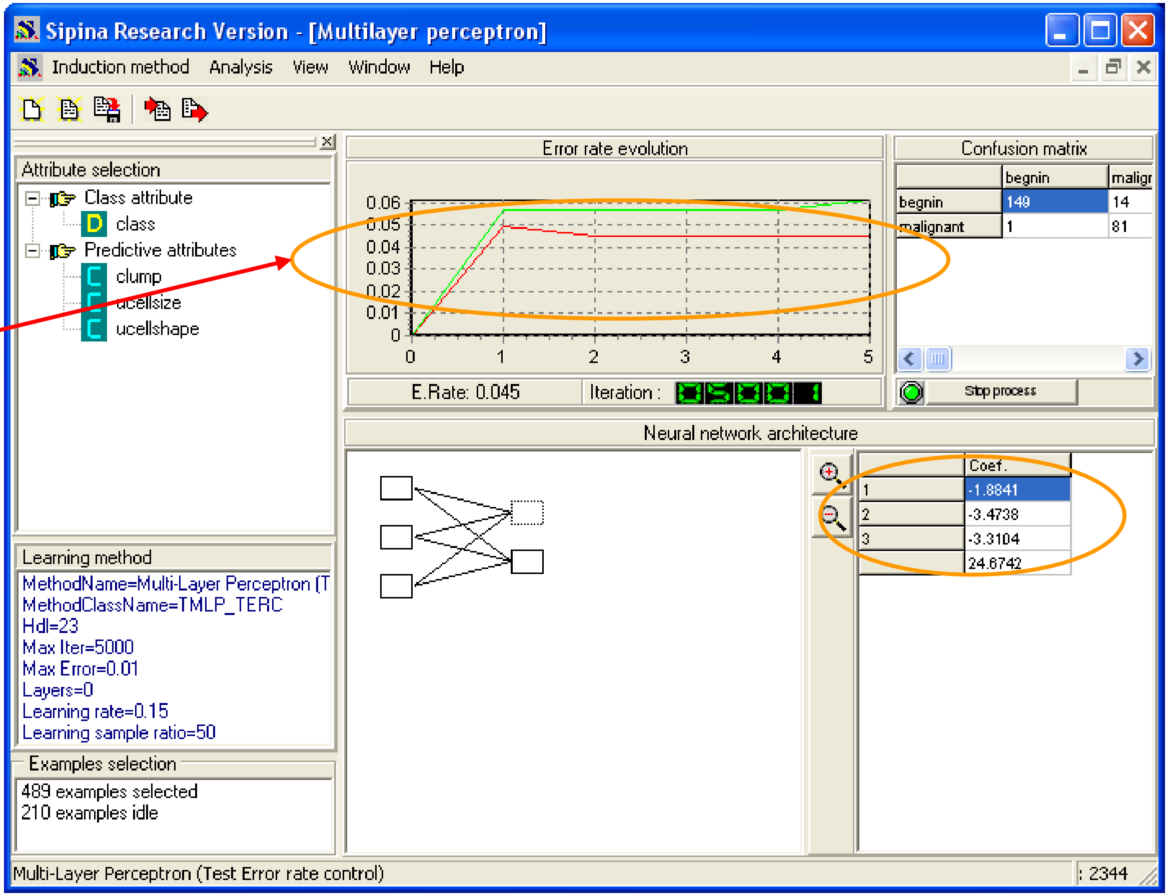
$$\hat{y} = y_{k*} \ \ iif \ \ k* = \arg\max_k \hat{y}_k$$

Minimizing the mean squared error
By processing K perceptrons in parallel !

$$E = \frac{1}{2} \sum_{\omega} \sum_{k=1}^{K} \left(y_k(\omega) - \hat{y}_k(\omega) \right)^2$$

Example on the "breast cancer" dataset (SIPINA tool)

Evolution of the error rate



Attribute selection

- Class attribute
  - D class
- Predictive attributes
  - C clump
  - C ucellsize
  - C ucellshape

E.Rate: 0.045 Iteration : 05001

	begnin	maligr
begnin	149	14
malignant	1	81

	Coef.
1	-1.8841
2	-3.4738
3	-3.3104
	24.6742

Learning method

```
MethodName=Multi-Layer Perceptron (T
MethodClassName=TMLP_TERC
Hdl=23
Max Iter=5000
Max Error=0.01
Layers=0
Learning rate=0.15
Learning sample ratio=50
```

Examples selection

489 examples selected
210 examples idle

Multi-Layer Perceptron (Test Error rate control) : 2344

Weights

Some recommendations

Set the input variables on the same scale (standardization, normalization, etc.)

Sometimes, it is useful to partition the data set in three parts: training set (learning of the weights), validation set (to monitor the error rate), test set (to estimate the generalization performance)

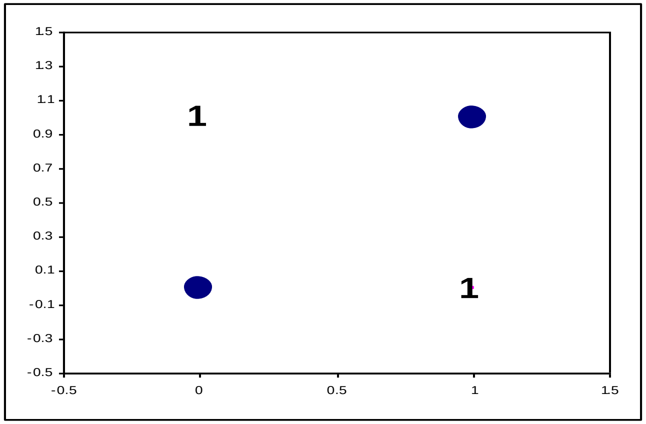
The settings must be handled with care (learning rate, stopping rule, etc.)

Death and resurrection of the perceptron - The XOR problem

(1)

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

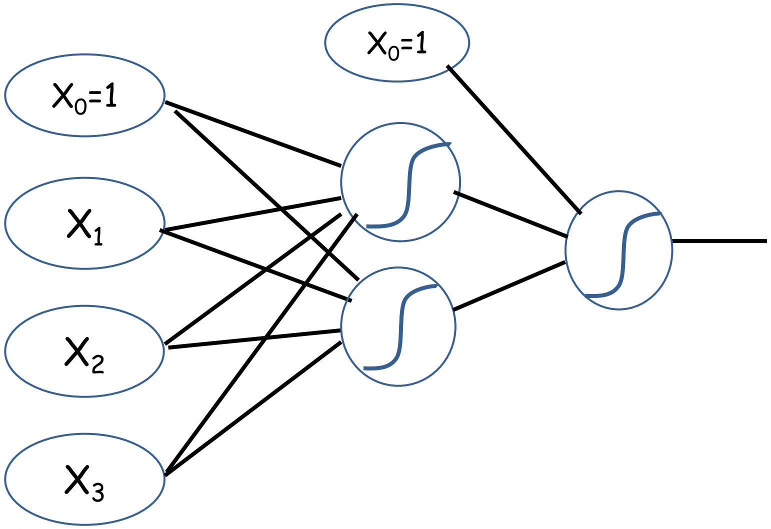
Dataset



Not linearly separable (Minsky & Papert, 1969)

➔ The perceptron cannot handle some kinds of problems. It is the end of the perceptron…

(2)



Input layer

Hidden layer

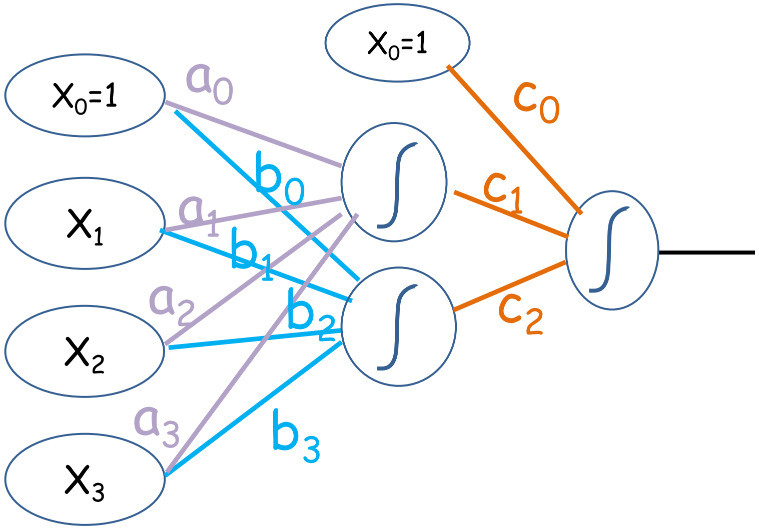
Output layer

A combination of several linear separators provides a global non-linear classifier (Rumelhart, 1986) !

Multilayer perceptron (MLP)

We can have several hidden layers (but this is unusual)

MLP – Formulas



From Input layer → Hidden layer

$$v_1 = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3$$
$$v_2 = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3$$

Output of the hidden layer

$$u_1 = g(v_1) = \frac{1}{1+e^{-v_1}}$$
$$u_2 = g(v_2) = \frac{1}{1+e^{-v_2}}$$

From the hidden layer → Output layer

$$z = c_0 + c_1 u_1 + c_2 u_2$$

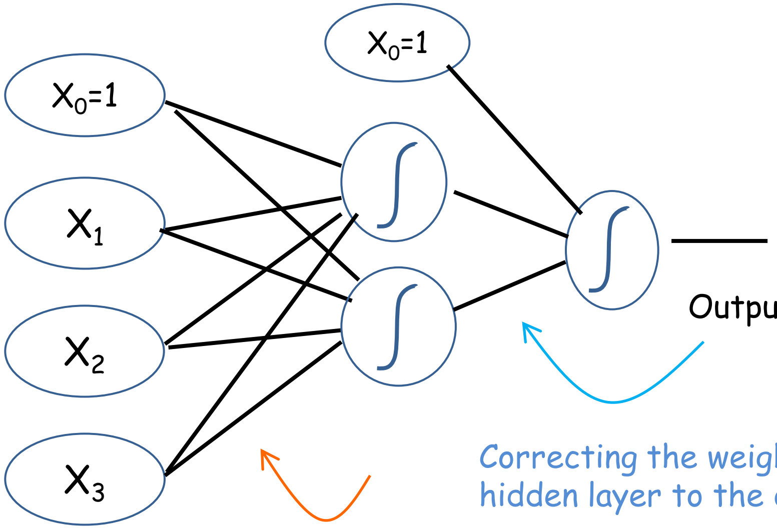
Output of the MLP

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

Representation power: the MLP can represent any boolean function if we set enough neurons in the hidden layer. !

MLP – Learning the weights through back propagation of error

Generalization of the Widrow-Hoff learning rule



Output: Measure the error (for one instance)

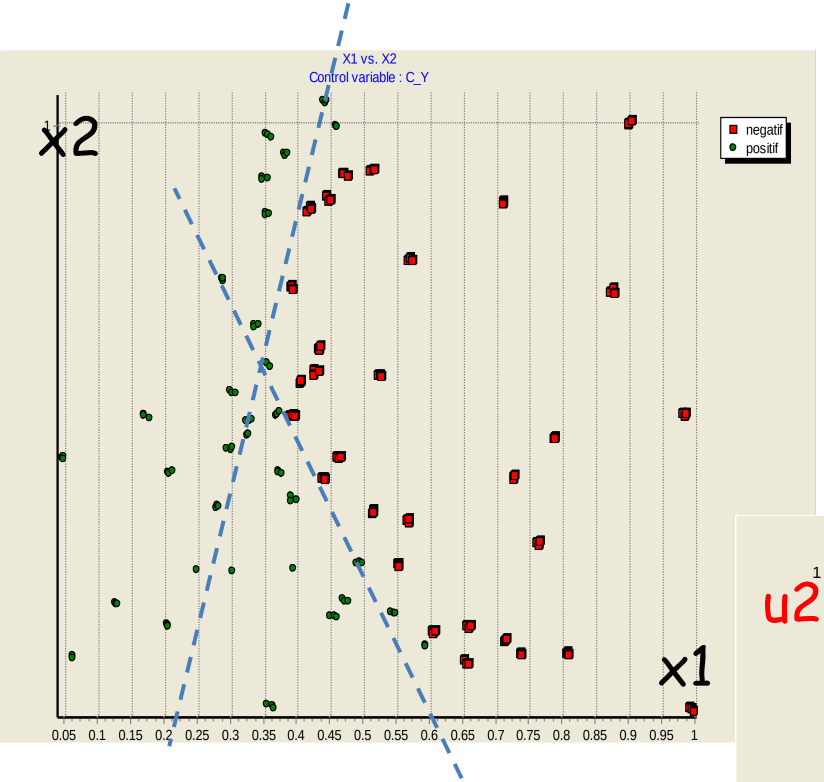
Correcting the weights from the hidden layer to the output layer.

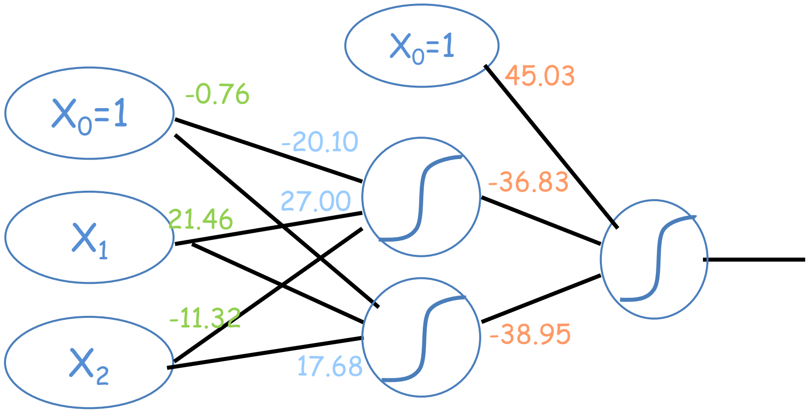
Back propagation of the correction to the weights from the input layer to the hidden layer.

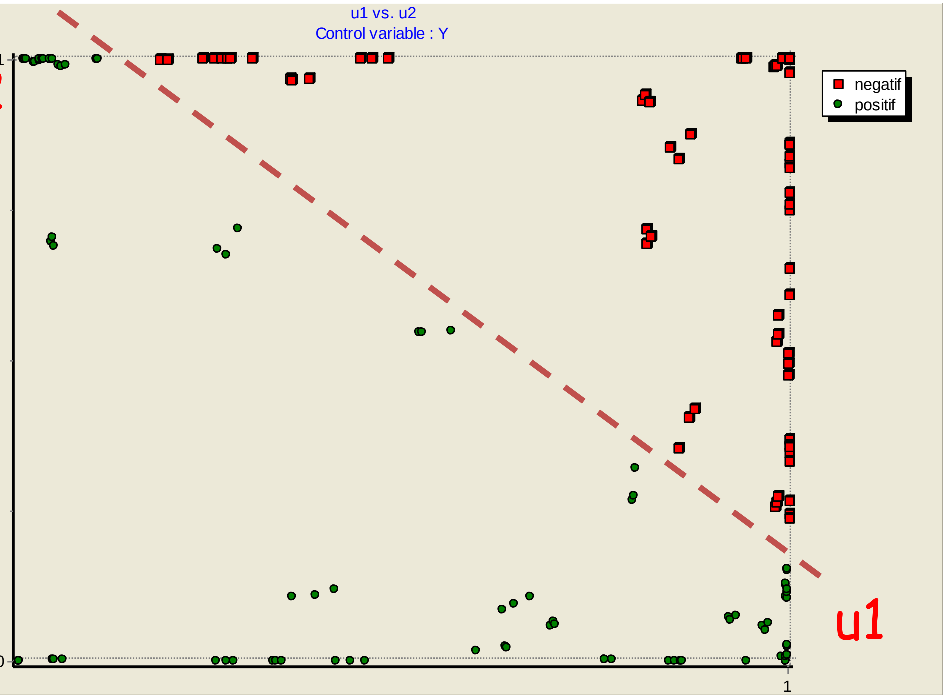
! The back propagation algorithm is efficient in the most of cases, even if the problem of local minima is not negligible. We must normalize (or standardize) the input variables and choose carefully the learning rate. Various sophisticated approaches to avoid local minima exist in the literature.

MLP – An example of non linear problem

Two neurons in the hidden layer = two straight lines in the original 2-dimensional representation space







Another point of view: the output of the hidden layer defines a new representation space where the classes are linearly separable.

MLP – Pros and Cons

+

Very efficient classifier (if well parameterized)

Sequential learning process (among others, easy to update)

Scalability (ability to handle very large dataset)

−

Black box model (no information about the influences of the input variables)

The parameters are hard to determine (e.g. number of neurons in hidden layer)

Problem of convergence in some situations (local minima)

Overfitting (use absolutely a validation set for monitoring the error rate)

References

“Neural network”
Tutorial slides by Andrew Moore
http://www.autonlab.org/tutorials/neural.html

“Perceptron Learning Rule”
Martin Hagan, http://hagan.okstate.edu/4_Perceptron.pdf

“Machine Learning”
Tom Mitchell, Ed. Mc Graw-Hill International, 1997.

“Introduction to machine learning”
Nils Nilsson, 1996, http://robotics.stanford.edu/people/nilsson/mlbook.html