# Neural Networks Review

Machine Learning 10-601

Fall 2012

Recitation

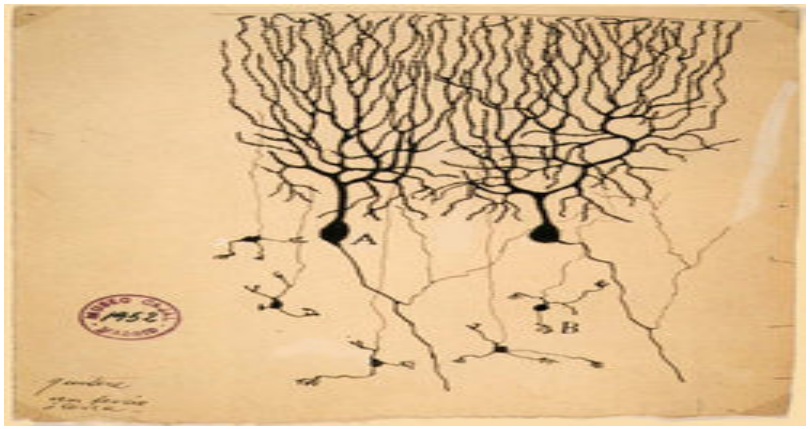
October 2 & 3, 2012

Petia Georgieva

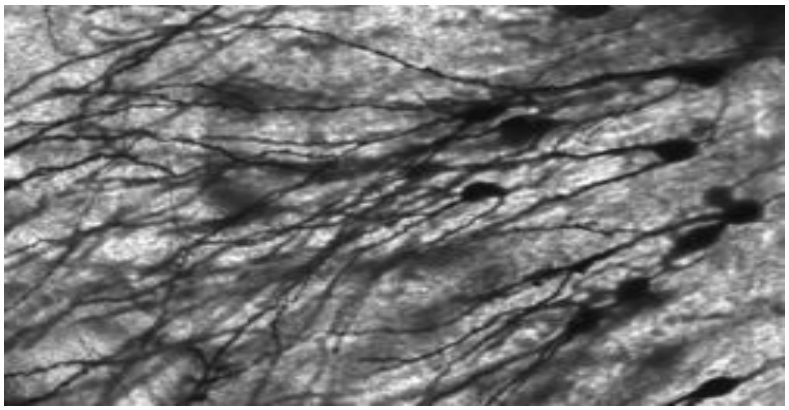
# Outline

- Biological neuron networks
- Artificial neuron model (perceptron)
- Activation functions
- Gradient Descent
- BackPropagation

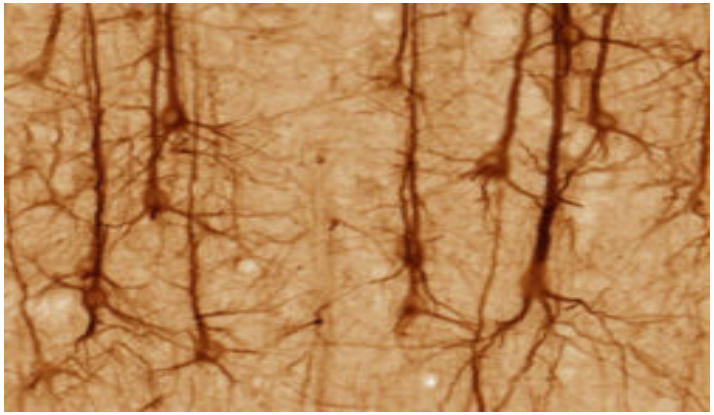
# Biological neuron networks




bird cerebellum



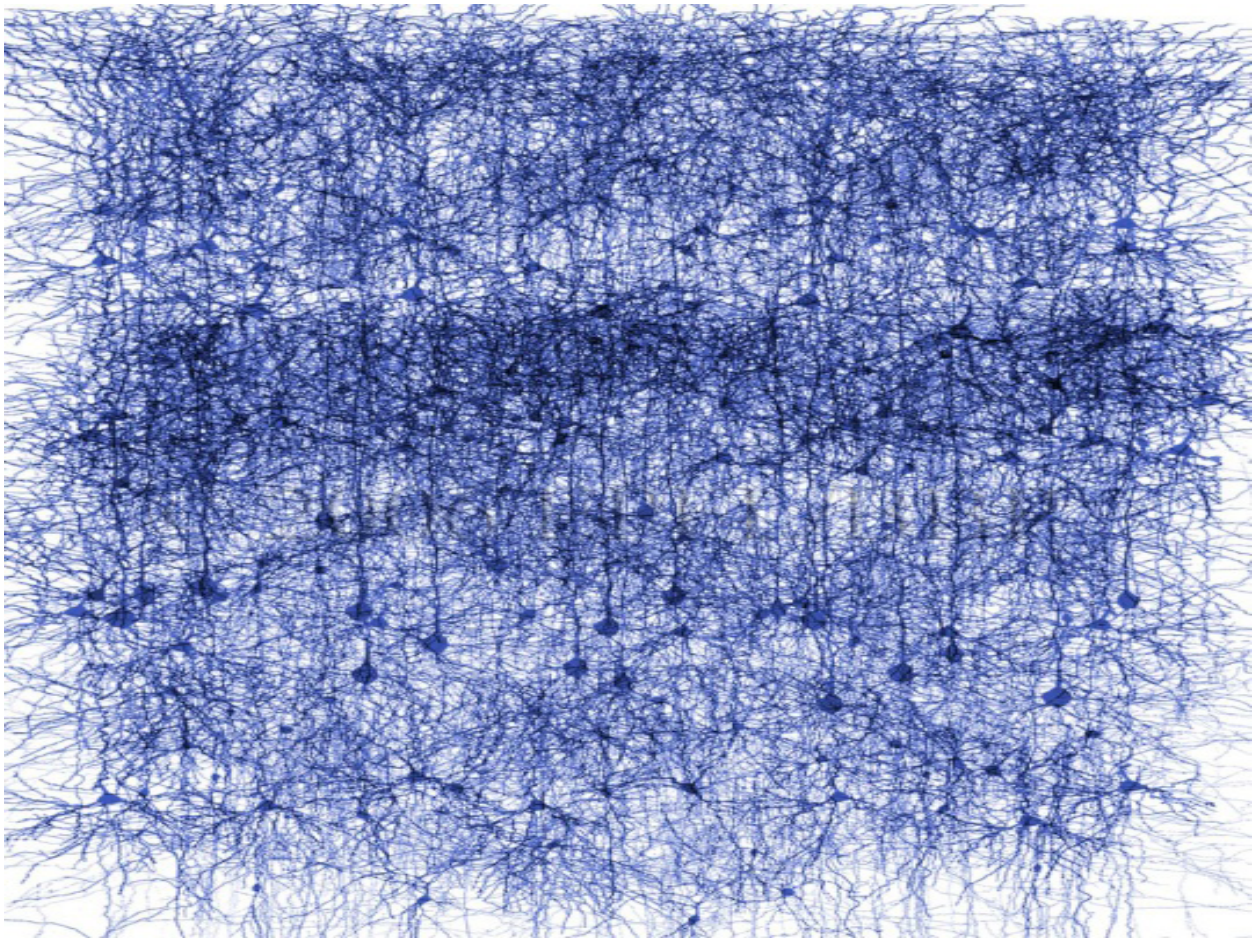
human hippocampus



human cortex

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

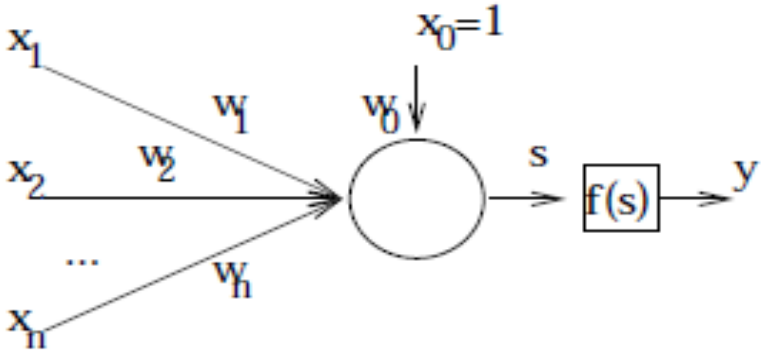
# Rat neo-cortex



BlueBrainProject: http://mediatheque.epfl.ch/sv/Blue_brain

# Artificial neuron model

(McCulloch and Pitts, 1942)




$$g : \Re^n \rightarrow \Re$$

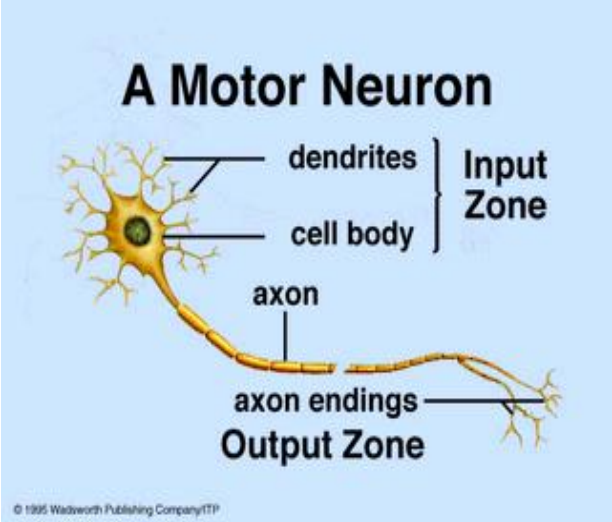
$\mathbf{x} = [x_1, x_2, \ldots . x_n]$ - Neuron Input Vector

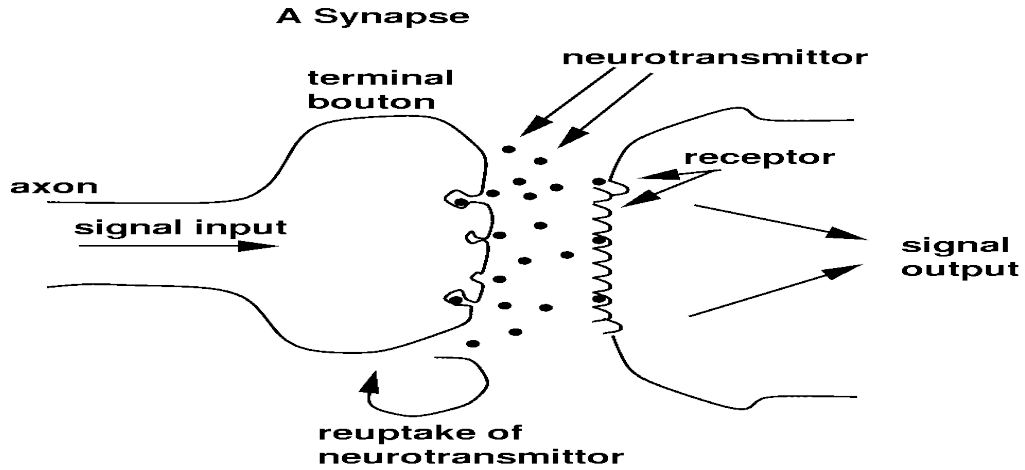
$$s = \sum_{i=0}^{n} w_i x_i, \quad x_0 = 1 \text{ (bias)}$$

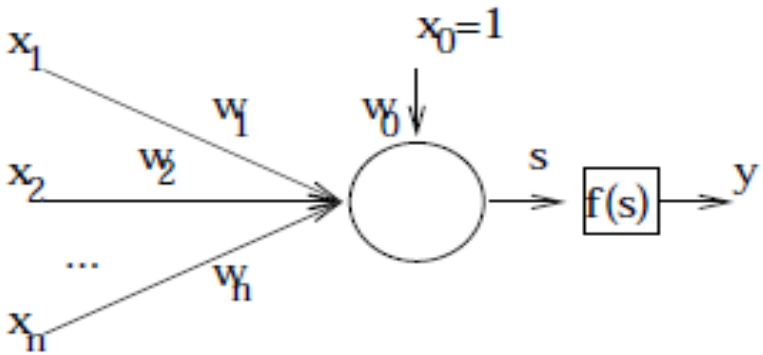
$y = f(s)$ - Neuron Scalar Output

$f(.)$ - Activation Function

# Biological analogy



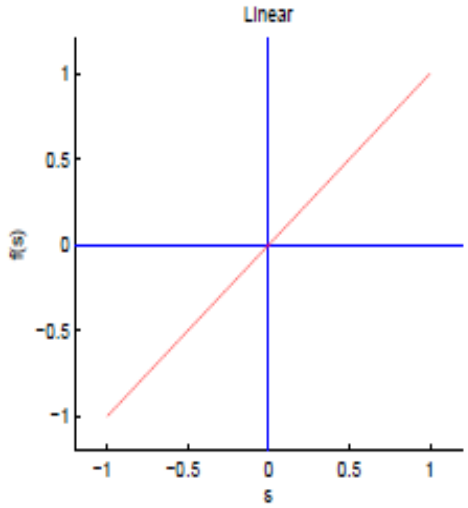



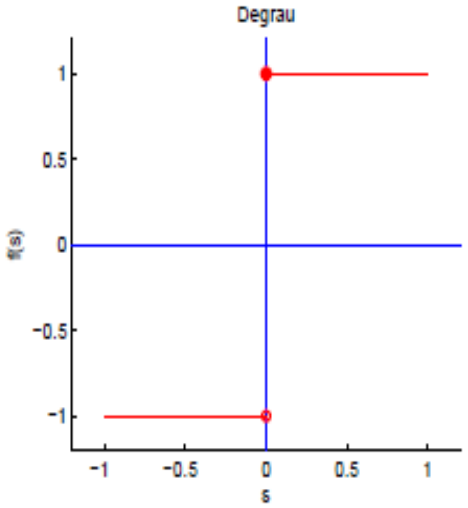



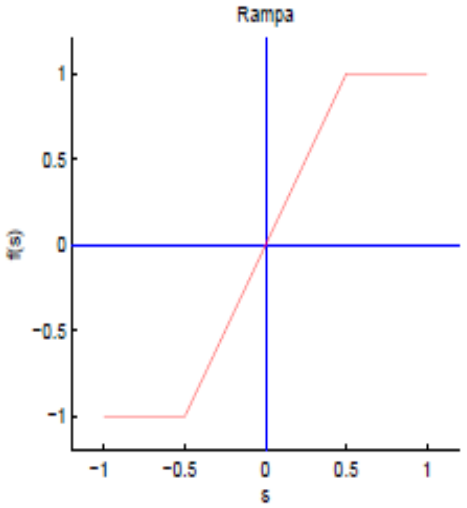
# Activation functions

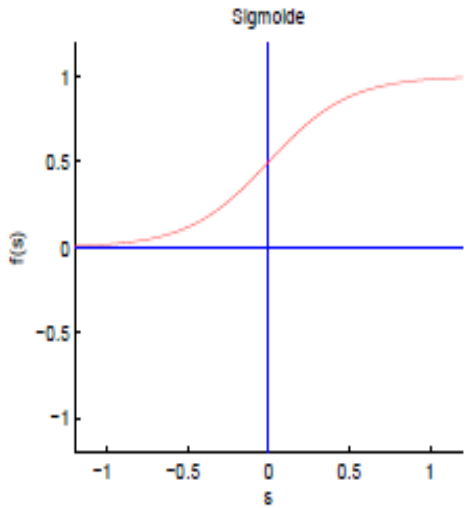
- Linear : $f(s) = \beta s$
- Step (Heaviside): $f(s) = \begin{cases} \beta_2, & s \geq 0 \\ \beta_1, & s < 0 \end{cases}$, normally $(\beta_2 = 1), (\beta_1 = 0 / \beta_1 = -1)$
- Ramp: $f(s) = \begin{cases} \beta, & s > \beta \\ s, & |s| < \beta \\ -\beta, & s > \beta \end{cases}$
- Sigmoid: $f(s) = \dfrac{1}{1 + \exp(-\lambda s)}$
- Tangent hyperbolic: $f(s) = \dfrac{\exp(\lambda s) - \exp(-\lambda s)}{\exp(\lambda s) + \exp(-\lambda s)}$

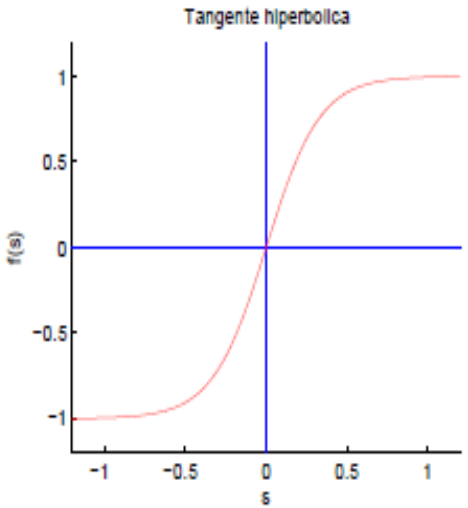
# Activation functions



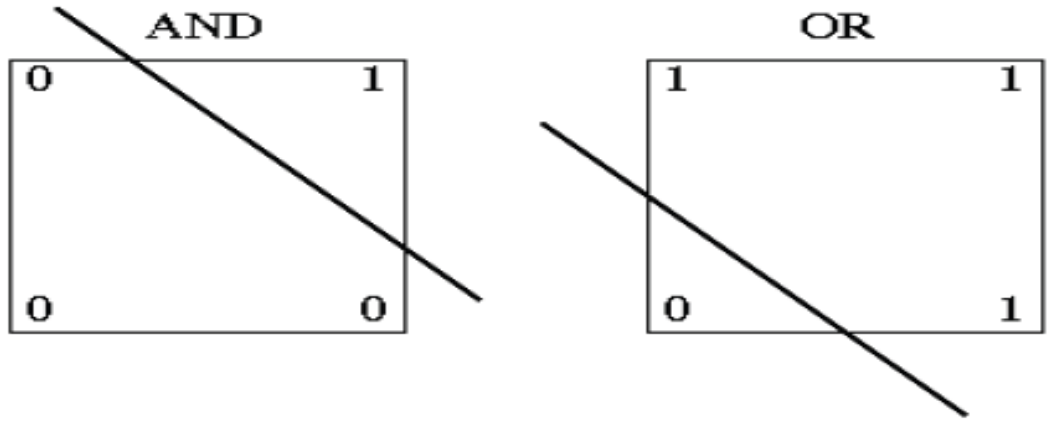
# Discriminative capacity of one neuron

Linearly separable objects – Hyperplane that separates objects: $(y > 0 / y < 0)$

Rosenblatt's perceptron (late 1950's) - simple Learning Machine, step act. funct.

Example : Logical AND or Logical OR are classification problems.

| Input 1 | Input 2 | Output |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |



# Limitations of perception

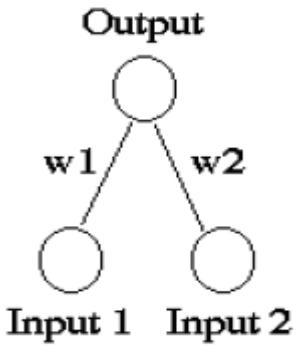
(Minsky and Papert, *Perceptrons*, 1969)

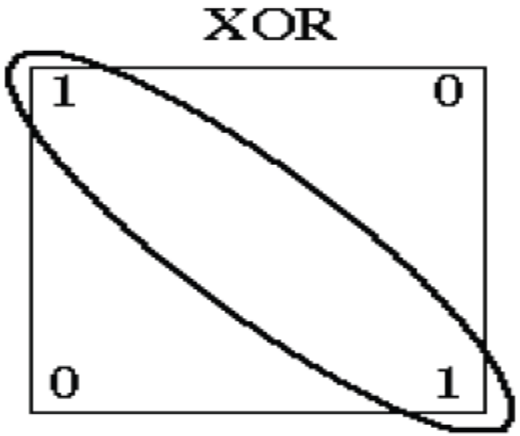
The exclusive-or (XOR) problem cannot be computed by a perceptron (or any two layer network).

XOR problem - the output must be turned on when either of the inputs is turned on, but not when both are turned on.

XOR is not a linearly separable problem.

| Input 1 | Input 2 | Output |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

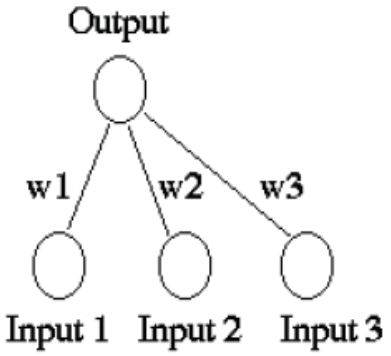


# Why a Hidden Layer?

**Idea 1**: Recode the XOR problem in three dimensions so that it becomes linearly separable.
By adding an appropriate extra feature, it is possible to linearly separate the two classes.

| Input 1 | Input 2 | Input 3 | Output |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 |




**Idea 2:** Another way to make a problem become linearly separable is to add an extra (hidden) layer between the inputs and the outputs.

**!** Given a sufficient number of hidden units, it is possible to recode any unsolvable problem into a linearly separable one.

# Gradient Descent (GD)

Quadratic Error Function: $e_j = \left(d_j - y_j\right)^2$

$d_j$ - target output, $y_j$ - observed (NN) output

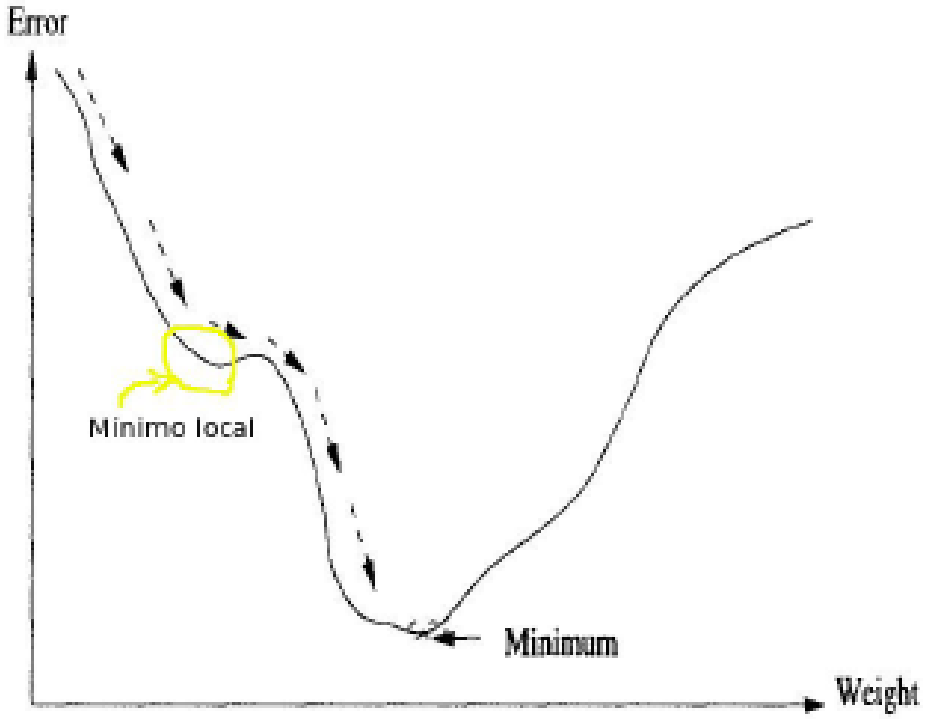



fig.: A. P. Engelbrecht, Computational Intelligence, An Introduction, John Wiley & Sons, 2002

# Gradient Descent (GD)

Given data point $j\left(\mathbf{x}_j \equiv \lfloor x_{j1}, x_{j2}, ..x_{jn} \rfloor \rightarrow d_j\right)$

Each weight $w_i$ at iteration $t$ is updated as

$$w_i(t) = w_i(t-1) + \Delta w_i(t)$$

$$\Delta w_i(t) = \eta\left[-\frac{\partial e_j}{\partial w_i}\right]$$

$$\frac{\partial e_j}{\partial w_i} = -2\left(d_j - y_j\right)\frac{\partial f}{\partial s_j}x_{ji}$$

$x_{ji}$ - input $i$ of data point $j$, $\eta$- learning rate

$f$ – neuron activation function

Any constraints on $f$?

# GD- Sigmoid Activation Function

If $f(s) = \dfrac{1}{1 + \exp(-\lambda s)} \quad \Rightarrow \dfrac{\partial f}{\partial s} = \lambda f(s)\left(1 - f(s)\right)$

$$\Delta w_i(t) = 2\eta\left(d_j - y_j\right)\lambda f(s_j)\left(1 - f(s_j)\right)x_{ji}$$

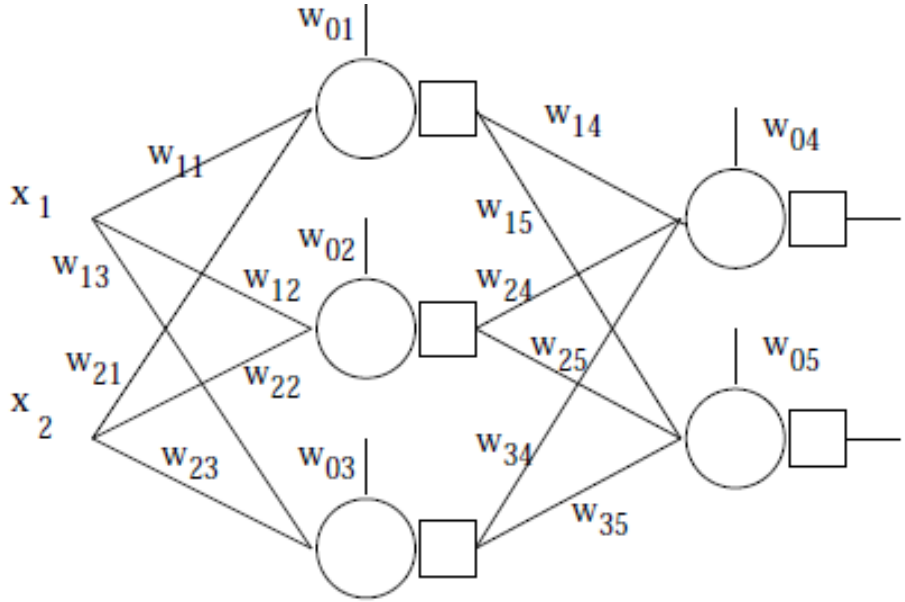
# Least Mean Squares (Delta Rule)

If $f(s) = s$ - linear act. funct. $\Rightarrow \dfrac{\partial f}{\partial s} = 1$

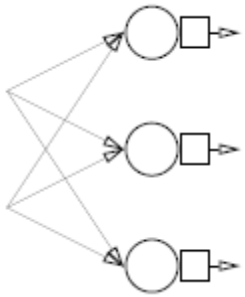
$$\Delta w_i(t) = 2\eta\left(d_j - y_j\right)x_{ji}$$

Adaline (Widrow e Hof, 1959) is the first hardware realization of a NN trained with Delta Rule.

# Multilayer perceptron (MLP)




FeedForward Neural Network (FFNN)



Perceptron (one layer)

# MLP as universal approximator

- FFNN with 1 hidden layer and continuous and differentiable activation functions can approximate any continuous function.


1. G. Cybenko. Approximation by superposition of a sigmoidal function. Mathematics of Control, Signal and Systems, 2:303-314, 1989.
2. K. Hornik. Some new results on neural network approximation. Neural Networks, 6:1060-1072, 1993.


- FFNN with 2 hidden layer and continuous and differentiable activation functions can approximate any function.


3. A.Lapedes and R. Farber. How neural nets work. In Anderson, editor, Neural Information Processing Systems, pages 442-456. New York, American Institute of Physics, 1987.
4. G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Dep. of Computer Science, Tufts University, Medford, MA, 1988.

# BackPropagation (BP)

## 1. Initialize the NN weights

For each object of the training set compute:

## 2. Forward computations

$$s_j = w_{0j} + \sum_i w_{ij} x_i \quad - \text{first layer}$$

$$s_j = w_{0j} + \sum_{i \in \text{previous layer}} w_{ij} y_i \quad - \text{hidden \& output layers}$$

$$y_j = f(s_j) \quad - \quad \text{output of neuron } j$$

# BackPropagation (BP)

**3. Backward computations** (c –cost function)

$$e_j = f'(s_j)\frac{\partial c}{\partial y_j} \quad - \quad \text{output layer}$$

$$e_j = f'(s_j)\sum_{p \in \text{next layer}} w_{jp} e_p \quad - \quad \text{hidden \& input layers}$$

**4. Weights update (batch training)**

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

$$\Delta w_{ij} = -\frac{\eta}{N}\sum_{x \in X} x_i e_j \quad - \quad \text{first layer}$$

$$\Delta w_{ij} = -\frac{\eta}{N}\sum_{x \in X} y_i e_j \quad - \quad \text{hidden \& output layers}$$

**5. Stop if conditions are achieved, if not go back to p. 2**

# BackPropagation (BP)

## 4. Weights update (on-line training)

$$\Delta w_{ij} = -\eta x_i e_j \quad - \quad \text{first layer}$$
$$\Delta w_{ij} = -\eta y_i e_j \quad - \quad \text{hidden \& output layers}$$

For $f$ sigmoide: $f'(s_j) = f(s_j)\left(1 - f(s_j)\right) \qquad \lambda = 1$

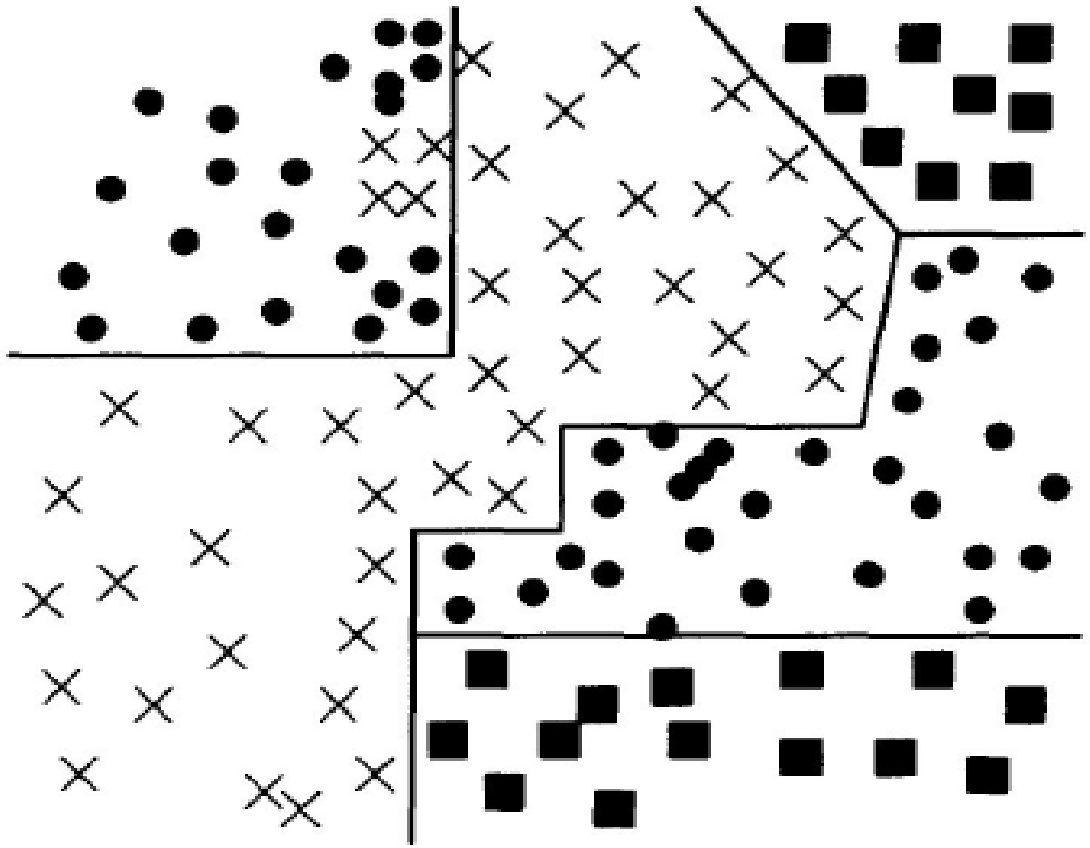
**Mean Squared Error (MSE) cost function**

$$c = \frac{1}{N}\sum_{i=1}^{N}\left(d_j - y_j\right)^2$$

$$\frac{\partial c}{\partial y_j} = -\frac{2}{N}\left(d_j - y_j\right)$$

# Classification with MPL

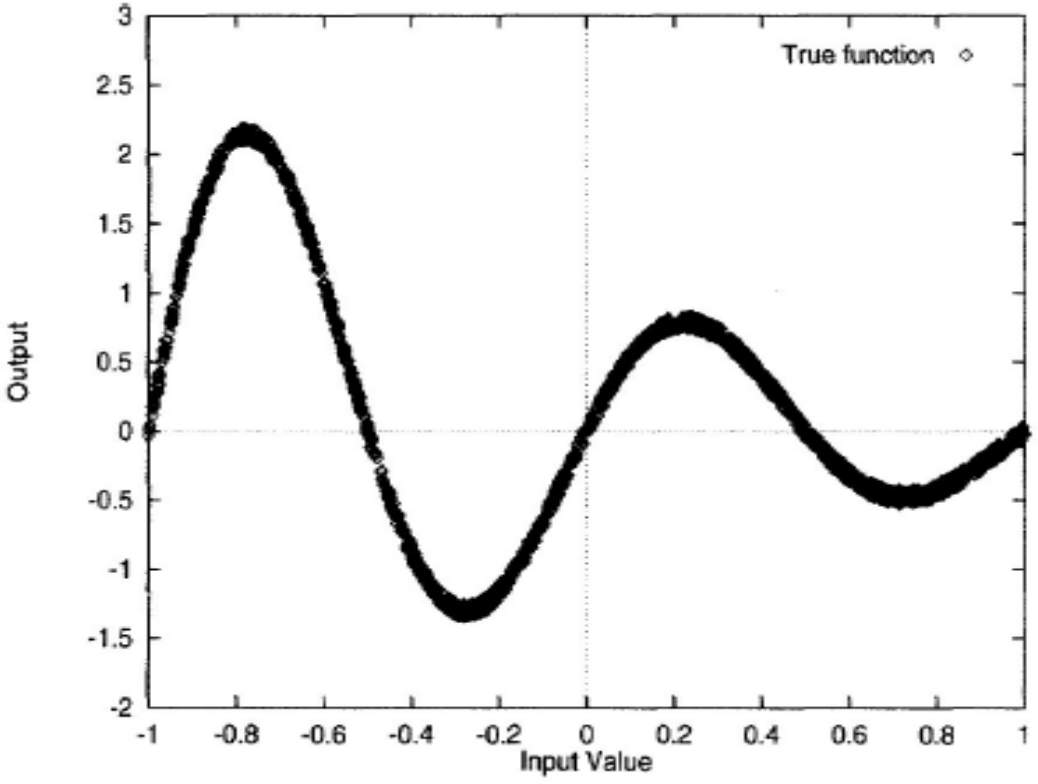
MLP creates a decision boundary between classes.
Rule of thumb: Each line of the boundary corresponds to one hidden layer neuron.



(fig.: Engelbrecht, p.50)

# Regression with MPL
## (function approximation)

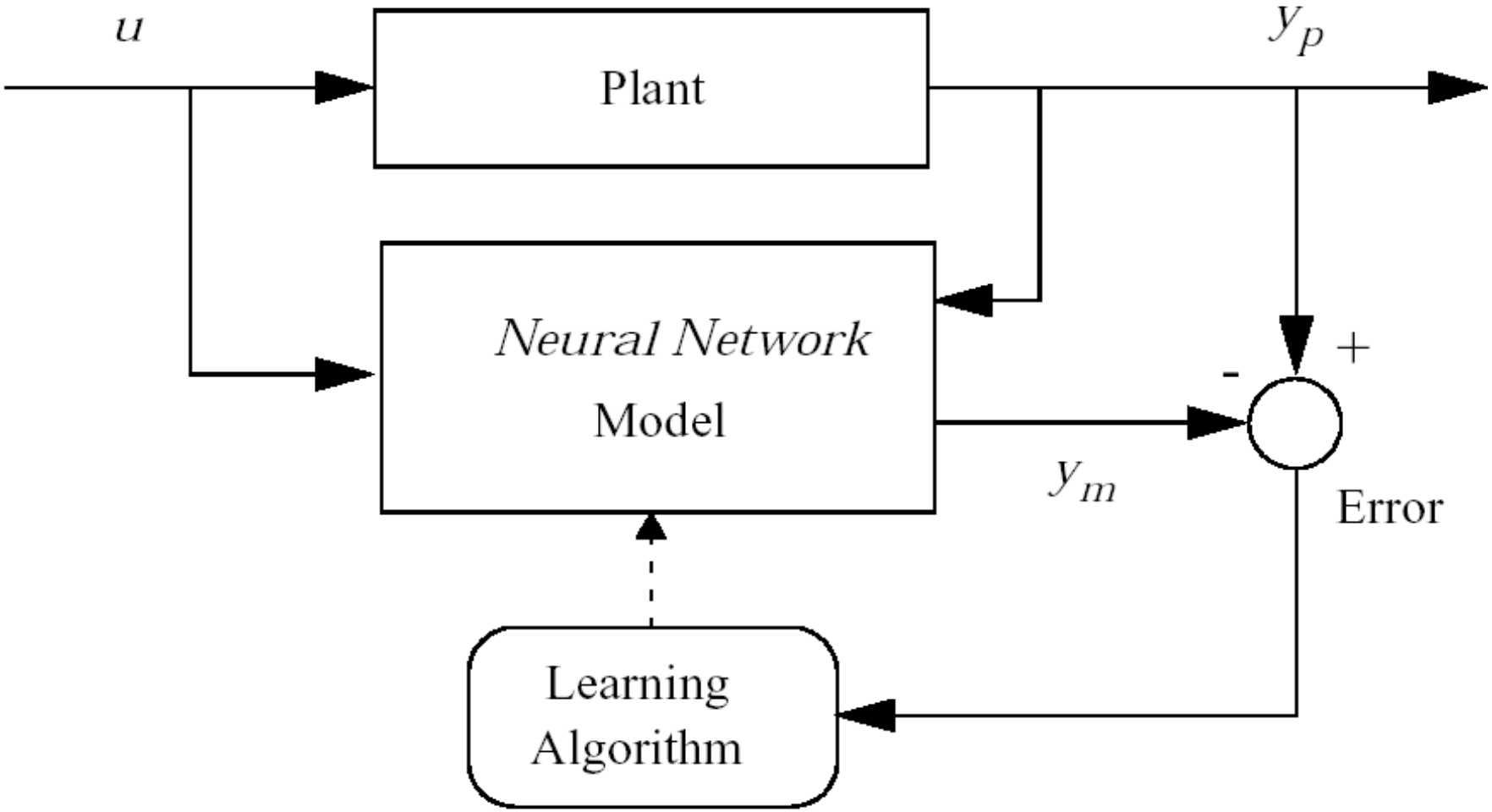
Rule of thumb:
# of hidden layer neurons = inflexion points of the function to approximate



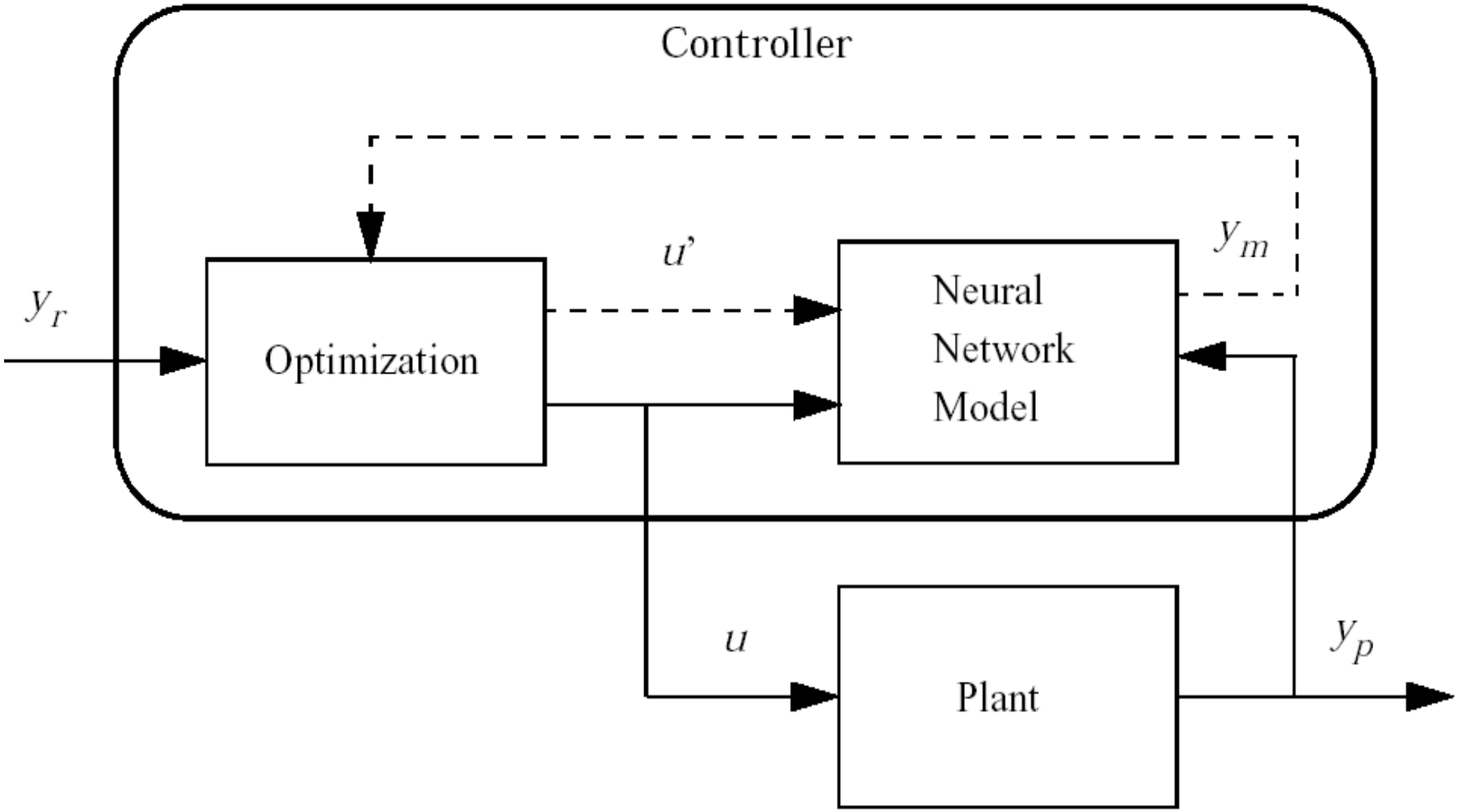

(fig.: Engelbrecht, p.51)

# of hidden layer neurons ?
The NN architecture?

# ANN Applications - System Identification



# ANN Applications - Model Predictive Control (NN MPC)



# ANN Applications -
# BRAIN MACHINE INTERFACE (BMI)