

# Artificial Neural Networks

Inteligencia Artificial en los Sistemas de Control Autónomo  
Máster Universitario en Ingeniería Industrial

Departamento de Automática

## Objectives

1. Describe biological neurons and networks
2. Basics of artificial neurons and networks
3. Understand the role of training in ANNs
4. Strengths and weaknesses of ANNs

## Bibliography

- A. Tettamanzi, M. Tomassini. Soft Computing. Integrating Evolutionary, Neural, and Fuzzy Systems. Springer-Verlag. 2001
- McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 7:115 - 133.
- Rosenblatt, Frank. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Psychological Review, 65:386-408

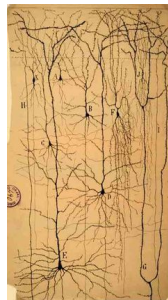
# Table of Contents

1. Introduction
  - History
  - Structure of neurons
2. Artificial neurons
  - Definition
  - Logical gates with a neuron
  - Activation functions
  - Learning limits
3. Artificial Neural Networks
  - Definition
  - Application examples
4. Feedforward networks
  - Definition
  - Separability
  - Bias in a MLP
5. Training algorithms
  - Problem statement
  - Gradient descent algorithm
  - Stochastic Gradient Descent Algorithm
  - Other optimization algorithms
  - Backpropagation algorithm
  - Learning problems

# Introduction

## History

- 1888 Ramón y Cajal. Discovery of biological neurons
- 1943 McCulloch & Pitts. First neural network designers
- 1949 Hebb. First learning rule
- 1958 Rosenblatt. Perceptron
- 1969 Minsky & Papert. Perceptron limitation - Death of ANN
- 1986 Rumelhart et al. Re-emergence of ANN: Backpropagation
- 201X Convolutional Neural Networks (CNNs) - Deep learning
- 2014 Goodfellow et al. Generative Adversarial Networks (GANs)



# Introduction

## Structure of neurons (I)

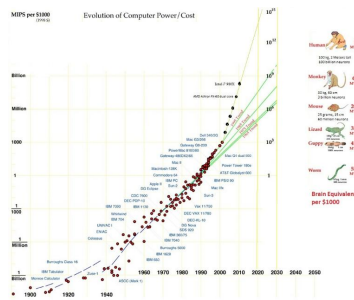
ANIMAL	NEURONS
Sponge	0
Roundworm	302
Jellyfish	800
Ant	250,000
Cockroach	1,000,000
Frog	16,000,000
Mouse	71,000,000
Cat	760,000,000
Macaque	6,376,000,000
Human	86,000,000,000
Elephant	267,000,000,000

## Human brain

Neuron switching time: 0.001 s

Synapsis: 10-100 thousand

Scene recognition time: 0.1 s



(Source)

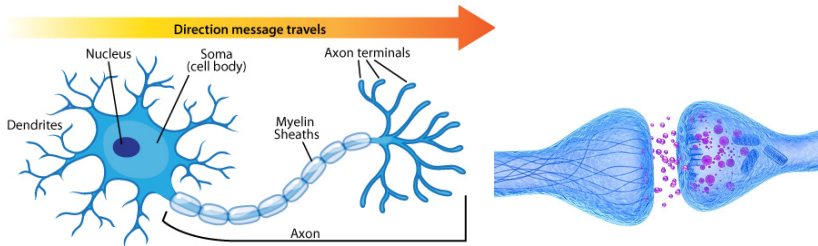
# Introduction

## Structure of neurons (II)

A neuron has a cell body ...

- ... a branching input structure (dendrite) and
- ... a branching output structure (axon)

Axons connect to dendrites via **synapses**



# Introduction

## Structure of neurons (III)

A neuron only fires if its input signal exceeds a threshold

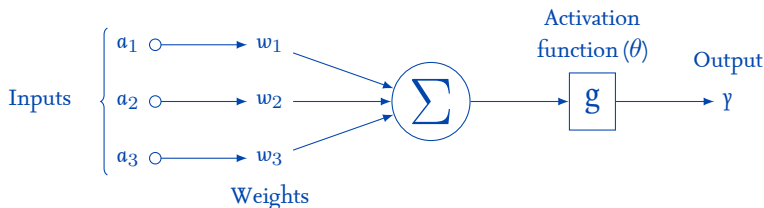
- Good connections allowing a large signal
- Slight connections allowing a weak signal
- Synapses may be either excitatory or inhibitory

Synapses vary in strength

- Biological learning involves setting that strength

# Artificial neurons

## Definition (I)



$a_i$  Normalized input ( $0 \leq a_i \leq 1$ )

$w_i$  Weight of input  $j$

$\theta$  Threshold

$g$  Activation function

Neuron model  
(perceptron)

$$\gamma = g \left( \sum_{i=1}^n w_i a_i \right)$$



# Artificial neurons

## Definition (II)

- Each neuron has a threshold value
- Each neuron has weighted inputs
- The input signals form a weighted sum
- If the activation level exceeds the threshold, the neuron activates

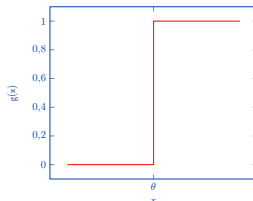
# Artificial neurons

## Definition (III)

The idealized activation function is a step function

$$g(x) = \begin{cases} 1 & \text{if } \sum_{i=1}^N w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

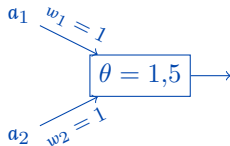
The step function is rarely used in practice



# Artificial neurons

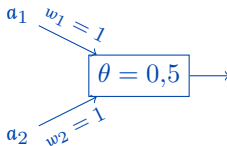
## Logical gates with a neuron

AND



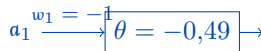
$a_1$	$a_2$	$\gamma$
0	0	0
0	1	0
1	0	0
1	1	1

OR



$a_1$	$a_2$	$\gamma$
0	0	0
0	1	1
1	0	0
1	1	1

NOT

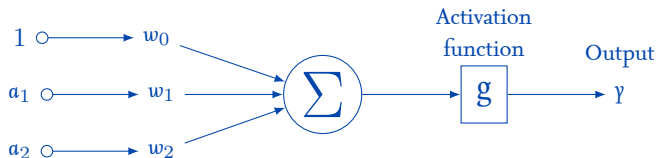


$a_1$	$\gamma$
0	1
1	0

(A neuron in Excel)

# Artificial neurons

## Definition of neuron (alternative version)



$a_i$  Normalized input ( $0 \leq a_i \leq 1$ )

$w_i$  Weight of input  $j$

$w_0$  Bias

$g$  Activation function

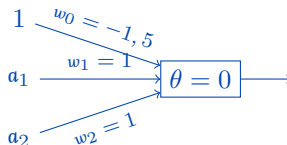
### Neuron model

$$\gamma = g \left( \sum_{i=0}^n w_i a_i \right)$$

# Artificial neurons

## Example of biased neuron

AND logical gate with a biased input

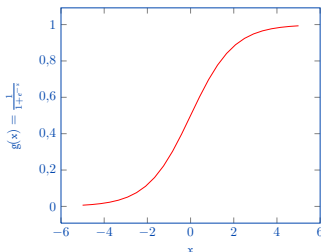


$a_0$	$a_1$	$a_2$	Output
I	O	O	O
I	O	I	O
I	I	O	O
I	I	I	I

# Artificial neurons

## Activation functions: Sigmoid function

- S-shaped, continuous and everywhere differentiable
- Asymptotically approach saturation points
- Derivative fast computation
- Range  $\in [0, 1]$



### Sigmoid function

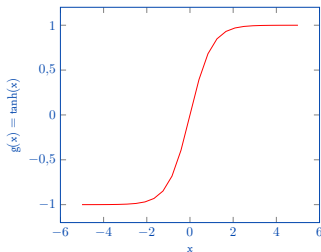
$$g(x) = \frac{1}{1 + e^{-x}}$$

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

# Artificial neurons

## Activation functions: Tanh function

- Asymptotically approach saturation points
- Range  $\in [-1, 1]$
- Bigger derivative than sigmoid (faster training)



### Tanh function

$$g(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

$$g'(x) = 1 - g(x)^2$$

# Artificial neurons

## Activation functions: Softmax function

- Generalization of the logistic function
- Usually used in the output layer in classification problems
- Asymptotically approach saturation points

### Softmax function

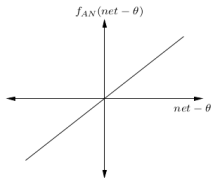
$$g(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \text{ for } j = 1, \dots, K$$

with  $\mathbf{z}$  a K-dimensional vector

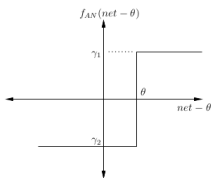


# Artificial neurons

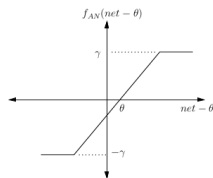
## Other activation functions



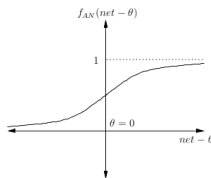
(a) Linear function



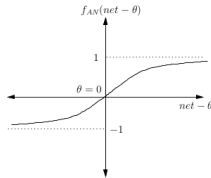
(b) Step function



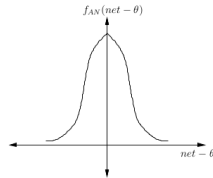
(c) Ramp function



(d) Sigmoid function



(e) Hyperbolic tangent function

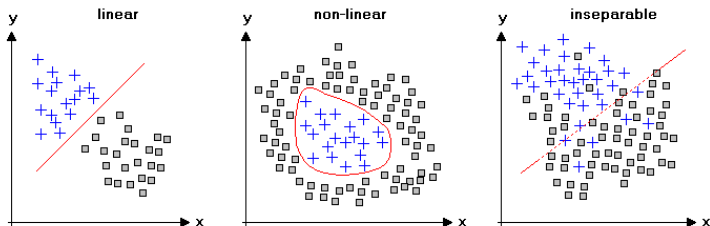


(f) Gaussian function

(Source)

# Artificial neurons

## Learning limits (I)

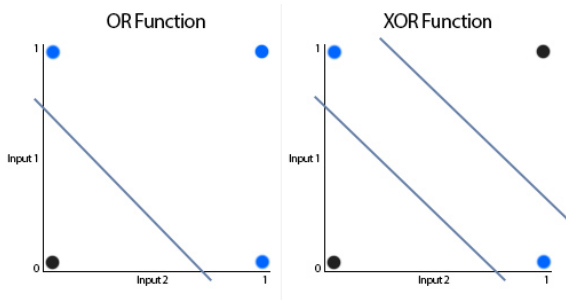


Problem: A single neuron only can solve linearly separable problems

# Artificial neurons

## Learning limits (II)

XOR cannot be implemented with a neuron



Solution: Neuronal networks

# Artificial Neural Networks

## Definition (I)

- A very much simplified version of biological nerve systems
- A set of nodes (neurons)
  - Each node has input and output
  - Each node performs a simple computation
- Weighted connections between nodes
  - Connectivity gives the structure of the net
  - What can be computed by an ANN is primarily determined by the connections and their weights
- It can recognize patterns, learn and generalize

# Artificial Neural Networks

## Definition (II)

### ANN properties

- Noise tolerance
- General function approximator

### Machine Learning tasks

- Supervised learning (classification and regression)
- Unsupervised learning (known as **self-organizing maps** in ANN terminology)

### Many topologies

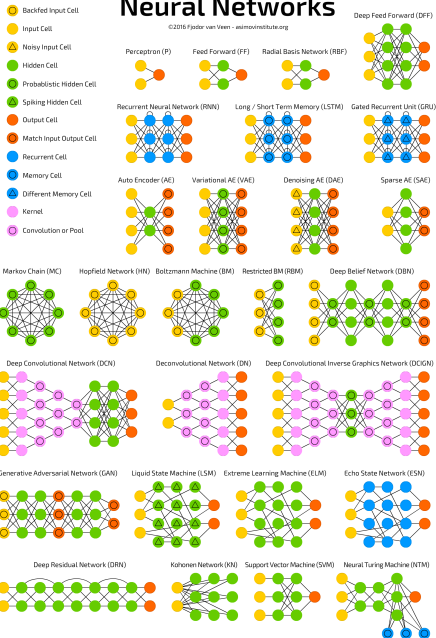
- Acyclic, recurrent (cyclic), modular, etc
- Feed Forward networks (MLPs)

Human readability less important than performance

A mostly complete chart of

# Neural Networks

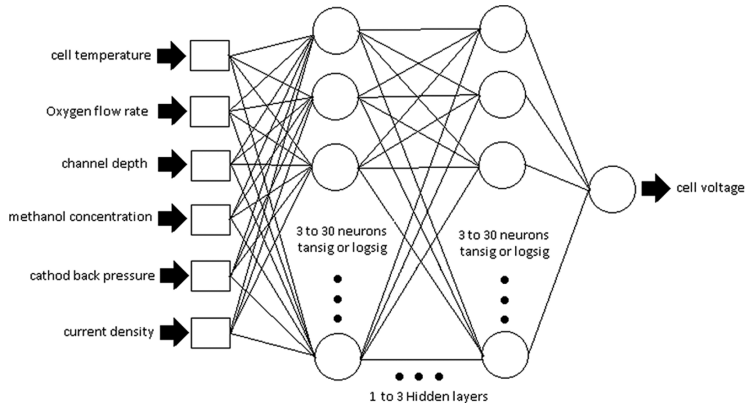
©2016 P.Jodan van Veen - aulmoulinstitute.org



(More info)

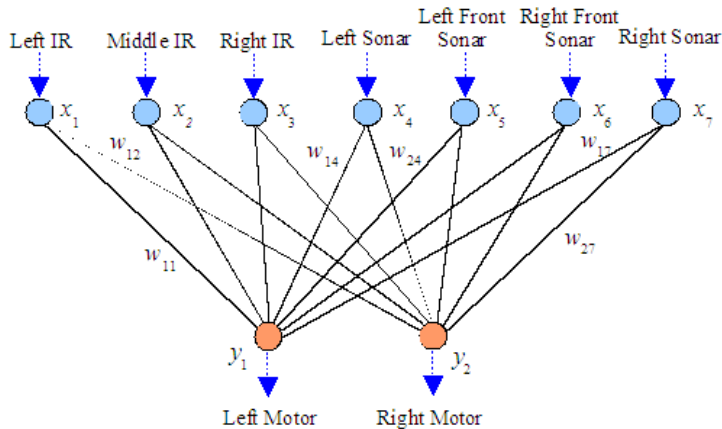
# Artificial Neural Networks

## Application examples (I)



# Artificial Neural Networks

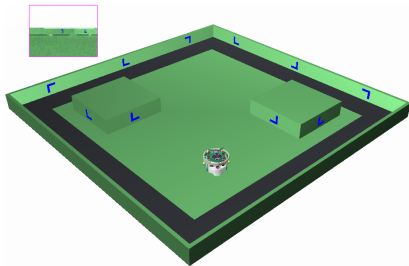
## Application examples (II)



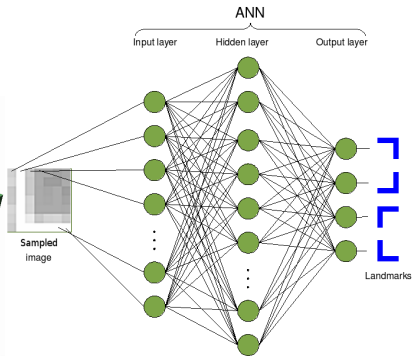


# Artificial Neural Networks

## Application examples (III)

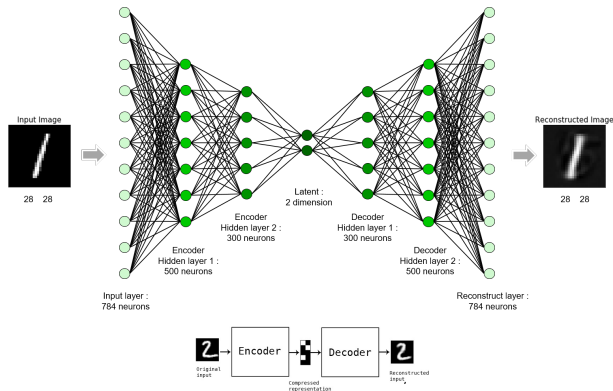


(Source)



# Artificial Neural Networks

## Application examples (III)



(Source)

# Feedforward networks

## Definition (I)

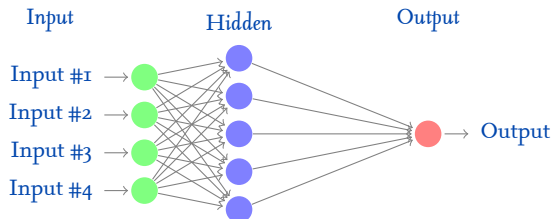
Neurons are arranged in **layers**

**Input** Which consists of any normalized data

**Output** Which are the net outcome

**Hidden** (Optional) No direct interaction

Also known as **multilayer perceptron (MLP)**



# Feedforward networks

## Definition (II)

### The input layer

- Introduces input values into the network
- No activation function or other processing

### The hidden layer(s)


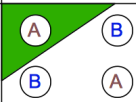
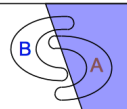


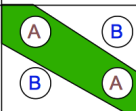
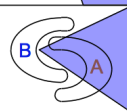
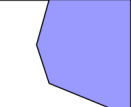
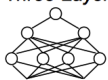
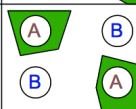
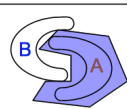
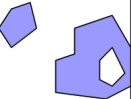
- Perform classification of features
- Two hidden layers are sufficient to solve any problem
- Features imply more layers may be better

### The output layer

- Functionally just like the hidden layers
- Outputs are passed on to the world outside the neural network

# Feedforward Networks

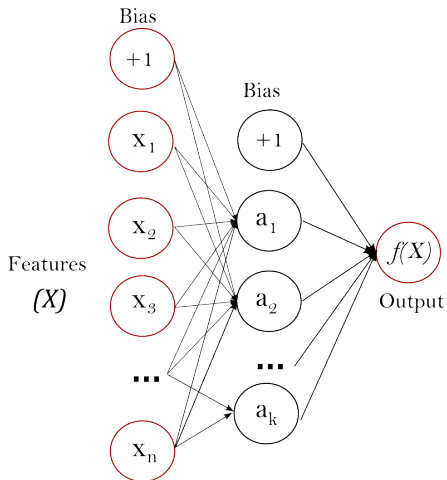
## Separability

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
<b>Single-Layer</b> 	<b>Half Plane Bounded By Hyperplane</b>			
<b>Two-Layer</b> 	<b>Convex Open Or Closed Regions</b>			
<b>Three-Layer</b> 	<b>Arbitrary (Complexity Limited by No. of Nodes)</b>			

(Online demo)

# Feedforward Networks

## Bias in a MLP



(Source)

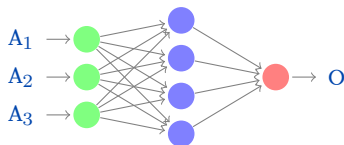
# Training algorithms

## Problem statement (I)

ANN can perform different tasks

- Classification, regression, others

Classification (or supervised learning) uses a training set



A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	O	Y
1,1	2,5	4,5	0,2	-0,1
0,9	2,4	1,2	0,5	0,4
1,0	2,0	9,9	0,4	1,2

Toss function: Measure of the error

- Usually mean squared error (mse):  $E = \frac{1}{2}(\gamma - o)^2 = f(w)$
- Y and O are the desired and observed outputs

# Training algorithms

## Problem statement (II)

$$E = \frac{1}{2} \text{Err}^2 = \frac{1}{2} \left[ \gamma - g \left( \sum_{j=0}^n w_j x_j \right) \right]^2$$

where

$\gamma$  Desired output

$w_j$  Weight connection  $j$

$x_j$  Input  $j$

Problem: Determine  $w$  that minimize  $f(w)$

- This is a classical optimization problem
- Any optimization algorithm can be used
- ... in AI, optimization means search



# Training algorithms

## Gradient Descent Algorithm (I)

Given the error

$$E = \frac{1}{2} \text{Err}^2$$

Take partial derivatives

$$\begin{aligned} \frac{\partial E}{\partial w_j} &= \text{Err} \frac{\partial \text{Err}}{\partial w_j} \\ &= \text{Err} \frac{\partial}{\partial w_j} g \left( y - \sum_{j=0}^n w_j x_j \right) \\ &= -\text{Err} \times g'(w) \times x_j \end{aligned}$$

# Training algorithms

## Gradient Descent Algorithm (II)

### Weight update

$$w_j^{k+1} = w_j^k + \alpha \times \text{Err} \times g'(w) \times x_j$$

with

$\alpha$  Learning rate ( $|\alpha| < 1$ )

err Difference desired and current output

$g'$  Derivate of activation function

$x_j$  Input j

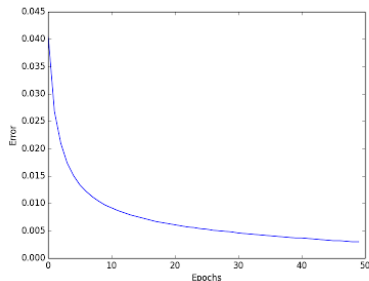
Each iteration is named **epoch**

### Learning algorithm (single neuron)

1. Apply input signal and compute outout
2. If output == desired output, do nothing
3. If output < desired output, increase weights
4. If output > desired output, decrease weights

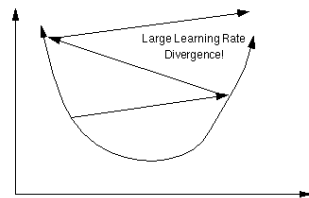
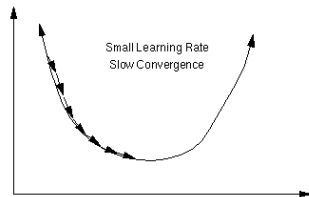
# Training algorithms

## Gradient Descent Algorithm (III)



(Source)

### Learning rate



(Source)

# Training algorithms

## Stochastic Gradient Descent (I)

SDG approximates the gradient sampling the dataset

On-line One sample

Mini-batch Several samples

Batch All the samples (Gradient Descent)

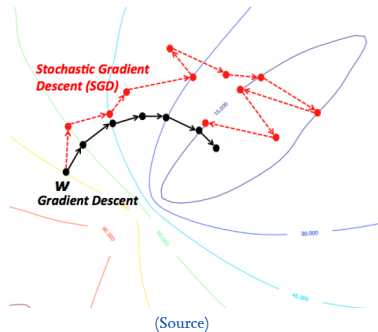
Weights update rule:  $w^{k+1} = w^k - \alpha \nabla g(\text{in})$

- where  $\alpha$  is the learning rate

SGD is slow and prone to local minima

# Training algorithms

## Stochastic Gradient Descent (II)



Usually, a **momentum** is introduced:  $w^{k+1} = w^k - \alpha z^{k+1}$ , where  $z^{k+1} = \beta z^k + \nabla g(\text{in})$

- $\alpha$  is the learning rate
- $\beta$  is the momentum strength
- If  $\beta = 0$  then gradient descent

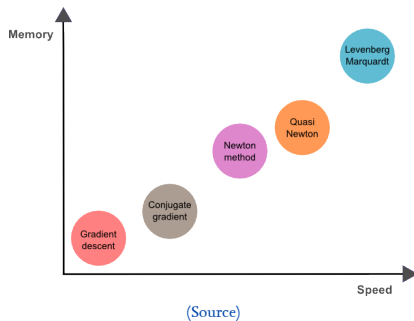
(On-line demo)

# Training algorithms

## Other optimization algorithms

### Other optimization algorithms

- Newton's method
- Quasi-Newton's method
- Levenberg-Marquardt method
- Conjugate Gradient



# Training algorithms

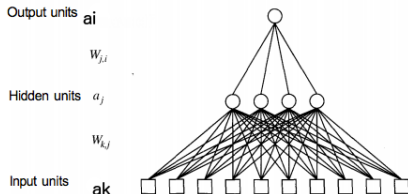
## Backpropagation algorithm (I)

Efficient learning algorithm for multilayer perceptrons. Three steps

1. **Feed-forward step.** Feed input, compute output and error
2. **Feed-backward step.** Compute individual contribution to error
3. **Adjust weights.** Modify weights to minimize error: Input, output and hidden layers

# Training algorithms

## Backpropagation algorithm (II)



Output layer: Same as single neuron

$$W_j \leftarrow W_j + \alpha \times \text{Err} \times g'(in) \times x_j$$

Define modified error as

$$\Delta_i = \text{Err}_i \times g'(in_i), \text{ then}$$

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$



# Training algorithms

## Backpropagation algorithm (III)

Hidden layer: Propagate error

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

where

$$\Delta_j = g'(\text{in}) \times \sum_j W_{j,i} \Delta_i$$

## Backpropagation algorithm

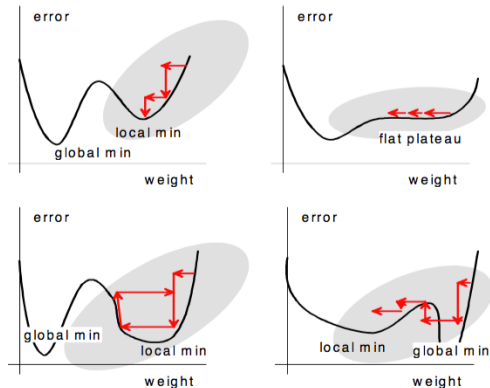
1. Compute output
2. Compute output error  $\Delta$
3. For each layer, repeat the following steps
  - 3.1 Propagate Delta backwards
  - 3.2 Update weights between two layers

# Training algorithms

## Learning problems

### Potential problems

- Local minima
- Flat plateau
- Oscillation
- Missing good minima



# Training algorithms

## Learning problems: Under and overfitting (I)

**Underfitting:** Does not learn

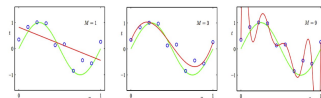
- Topology too simple

**Overfitting:** Memorizes samples

- Topology too complex
- Perhaps, the most serious concern in ML
- The net fails when exposed to new data

### Under- and Over-fitting examples

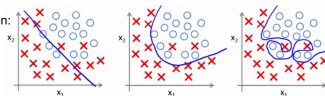
Regression:



predictor too inflexible:  
cannot capture pattern

predictor too flexible:  
fits noise in the data

Classification:



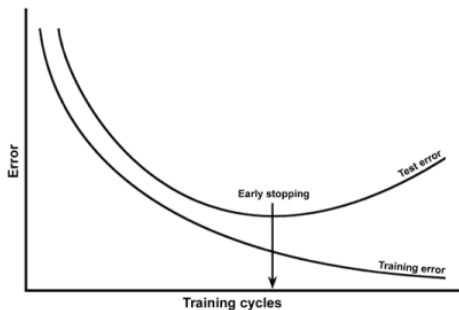
(Source)

# Training algorithms

## Learning problems: Under and overfitting (II)

Solution: Evaluate generalization capabilities

- Split training and validation sets and measure errors



(Source)

# Acknowledgements

- Jesus Aguilar Ruiz, Pablo de Olavide, Seville, Spain
- Daniel Rodríguez, Universidad de Alcalá, Alcalá de Henares, Spain