Intelligent Systems

Susana M. Vieira

Universidade de Lisboa, Instituto Superior Técnico

IS4, Center of Intelligent Systems, IDMEC, LAETA, Portugal
{susana.vieira}@tecnico.ulisboa.pt

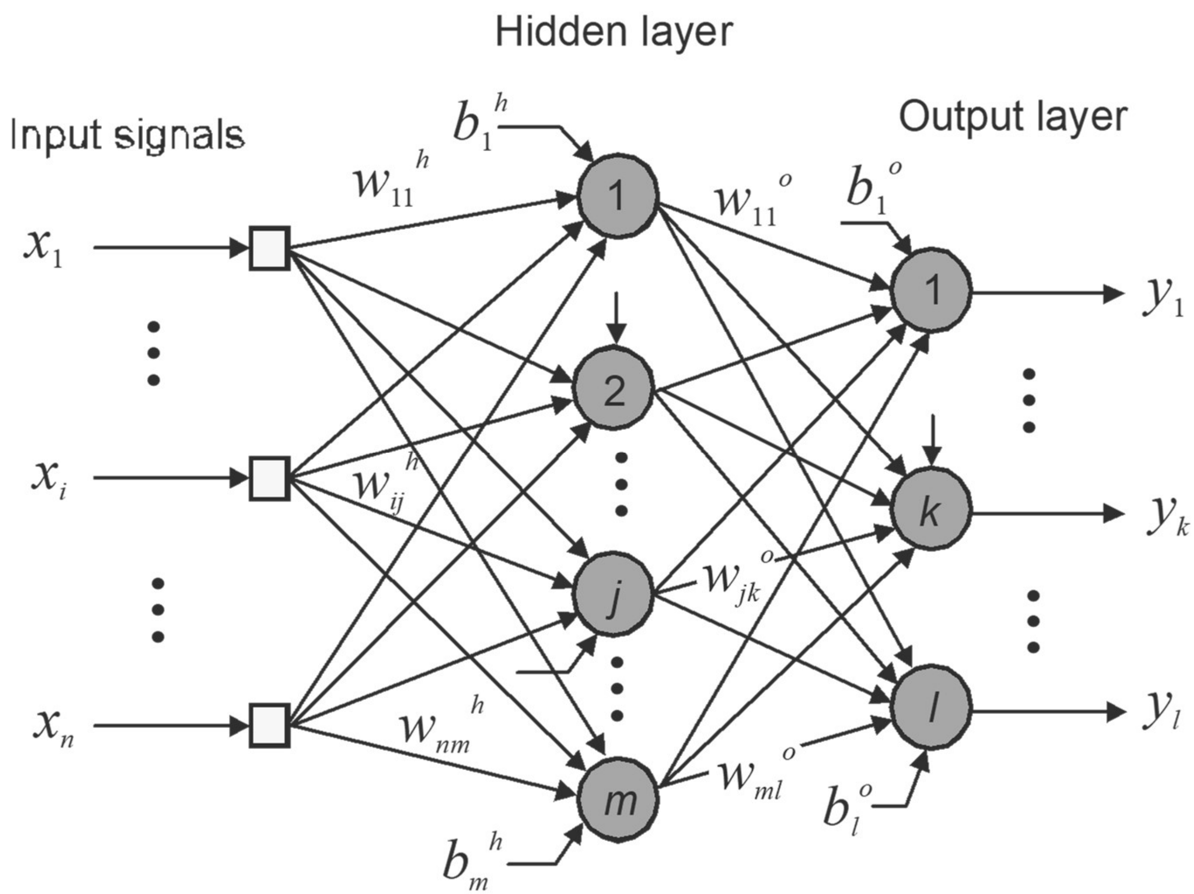
REVIEW ON NEURAL NETWORKS

SI5 – Review on Neural Networks

Reading:

- J.-S. Jang, C.-T. Sun and E. Mizutani. ***Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence***. Prentice Hall, New Jersey, 1997.
- S. Haykin. ***Neural Networks and Learning Machines***. Pearson Education, 2016.

Most common MLP



Most common MLP

- Output of neurons in the **hidden-layer** h_j:

$$h_j = \sigma\left(\sum_{i=1}^{n} w_{ij}^{h} x_i + b_j^{h}\right) = \sigma\left(\sum_{i=0}^{n} w_{ij}^{h} x_i\right) \qquad \sigma \Rightarrow \text{sigmoid}$$

$$= \tanh\left(\sum_{i=0}^{n} w_{ij}^{h} x_i\right)$$

- Output of neurons in the **output-layer** y_k:

$$y_k = \sigma\left(\sum_{j=1}^{m} w_{jk}^{o} h_j + b_j^{o}\right) = \sigma\left(\sum_{j=0}^{m} w_{jk}^{o} h_k\right) \qquad \sigma \Rightarrow \text{linear}$$

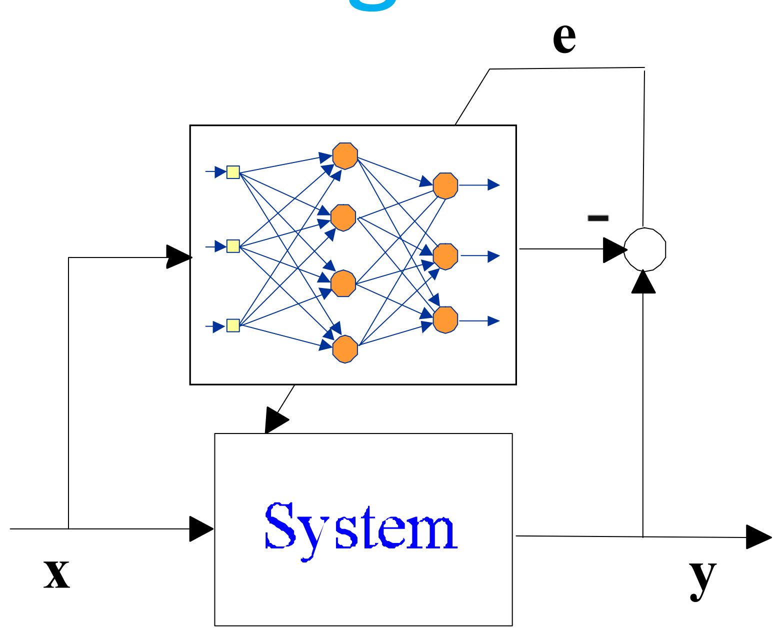
$$= \sum_{j=0}^{m} w_{jk}^{o} h_j$$

TÉCNICO LISBOA

Learning in NN

- **Biological neural networks:**
  - Synaptic connections amongst neurons which simultaneously exhibit high activity are strengthned.
- **Artificial neural networks:**
  - Mathematical approximation of biological learning.
  - Error minimization (***nonlinear*** optimization problem).
    - Error backpropagation (first-order gradient)
    - Newton methods (second-order gradient)
    - Levenberg-Marquardt (second-order gradient)
    - Conjugate gradients
    - ...

Supervised learning



- Training data:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T & \mathbf{x}_2^T & \cdots & \mathbf{x}_N^T \end{bmatrix}^T$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T & \mathbf{y}_2^T & \cdots & \mathbf{y}_N^T \end{bmatrix}^T$$

Error backpropagation

- Initialize all weights and thresholds to small random numbers

Repeat

1. Input training examples and compute network and hidden layer outputs
2. Adjust output weights using output error
3. Propagating output error backwards, adjust hidden-layer weights

Until satisfied with approximation

Backpropagation in MLP

- Compute the output of the output-layer, and compute error:

$$e_k = y_{d,k} - y_k, \quad k = 1, \ldots, l$$

- The cost function to be minimized is the following:

$$J(w) = \frac{1}{2} \sum_{k=1}^{l} \sum_{q=1}^{N} e_{kq}^2$$

- N – number of data points

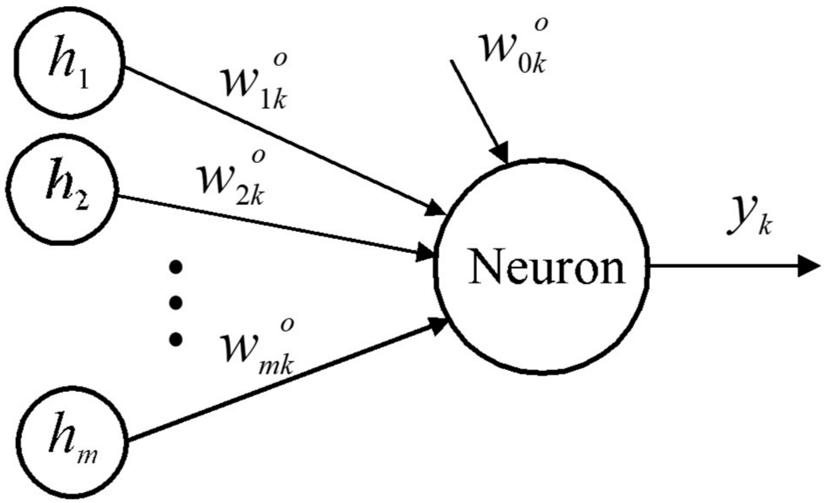
Learning using gradient

- Ouput weight learning for output y_k:

$$w_{jk}^{o}(p+1) = w_{jk}^{o}(p) - \alpha \nabla J(w_{jk}^{o})$$

$$\nabla J(w_{jk}^{o}) = \left(\frac{\partial J}{\partial w_{1k}^{o}}, \frac{\partial J}{\partial w_{2k}^{o}}, \ldots, \frac{\partial J}{\partial w_{mk}^{o}} \right)^{T}$$

Output-layer weights



$$y_k = \sum_{j=0}^{m} w_{jk}^o h_j, \quad e_k = y_{d,k} - y_k, \quad J(w_{jk}^o, w_{ij}^h) = \frac{1}{2}\sum_{k=1}^{l} e_k^2$$

Output-layer weights

- Applying the chain rule $$\frac{\partial J}{\partial w_{jk}^{o}} = \frac{\partial J}{\partial e_k}\frac{\partial e_k}{\partial y_k}\frac{\partial y_k}{\partial w_{jk}^{o}}$$

  with $$\frac{\partial J}{\partial e_k} = e_k, \quad \frac{\partial e_k}{\partial y_k} = -1, \quad \frac{\partial y_k}{\partial w_{jk}^{o}} = h_j$$

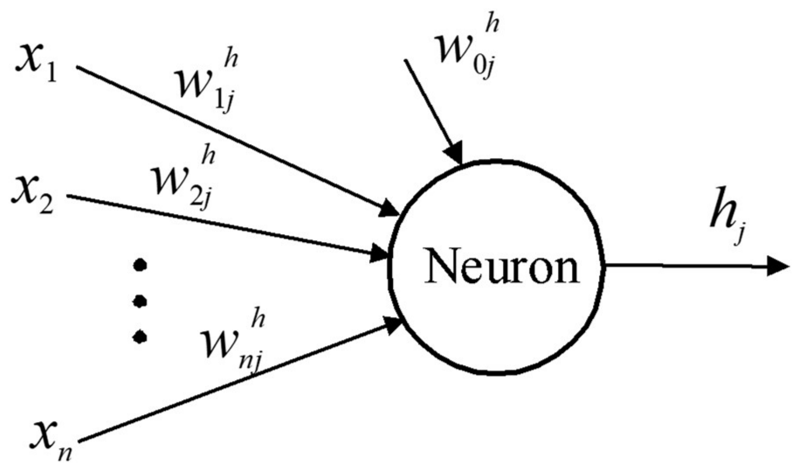
  then $$\frac{\partial J}{\partial w_{jk}^{o}} = -h_j e_k$$

- Thus: $$w_{jk}^{o}(p+1) = w_{jk}^{o}(p) - \alpha \nabla J(w_{jk}^{o}) = w_{jk}^{o}(p) + \alpha h_j e_k$$

- Recall that for SLP: $$\Delta w_i = \alpha\, x_i\, e$$

$$y_k = \sum_{j=0}^{m} w_{jk}^{o} h_j, \quad e_k = y_{d,k} - y_k, \quad J(w_{jk}^{o}, w_{ij}^{h}) = \frac{1}{2}\sum_{k=1}^{l} e_k^2$$

Hidden-layer weights



$$net_j = \sum_{i=0}^{n} w_{ij}^h x_i, \quad h_j = \tanh(net_j)$$

$$w_{ij}^h(p+1) = w_{ij}^h(p) - \alpha \nabla J(w_{ij}^h) \qquad \frac{\partial J}{\partial w_{ij}^h} = \frac{\partial J}{\partial h_j} \frac{\partial h_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}^h}$$

TÉCNICO LISBOA

Hidden-layer weights

- Partial derivatives: $\dfrac{\partial J}{\partial h_j} = \sum_{k=1}^{k} -e_k w_{jk}^o, \quad \dfrac{\partial h_j}{\partial net_j} = \sigma'_j(h_j), \quad \dfrac{\partial net_j}{\partial w_{ij}^h} = x_i$

- then $\dfrac{\partial J}{\partial w_{ij}^h} = -x_i \sigma'_j(h_j) \sum_{k=1}^{l} (-e_k w_{jk}^o)$

- and $\Delta w_{ij}^h(p) = \alpha x_i \sigma'_j(h_j) \sum_{k=1}^{l} (-e_k w_{jk}^o)$

$$\boxed{net_j = \sum_{i=0}^{n} w_{ij}^h x_i, \quad h_j = \tanh(net_j)}$$

Error backpropagation algorithm

- Initialize all weights to small random numbers

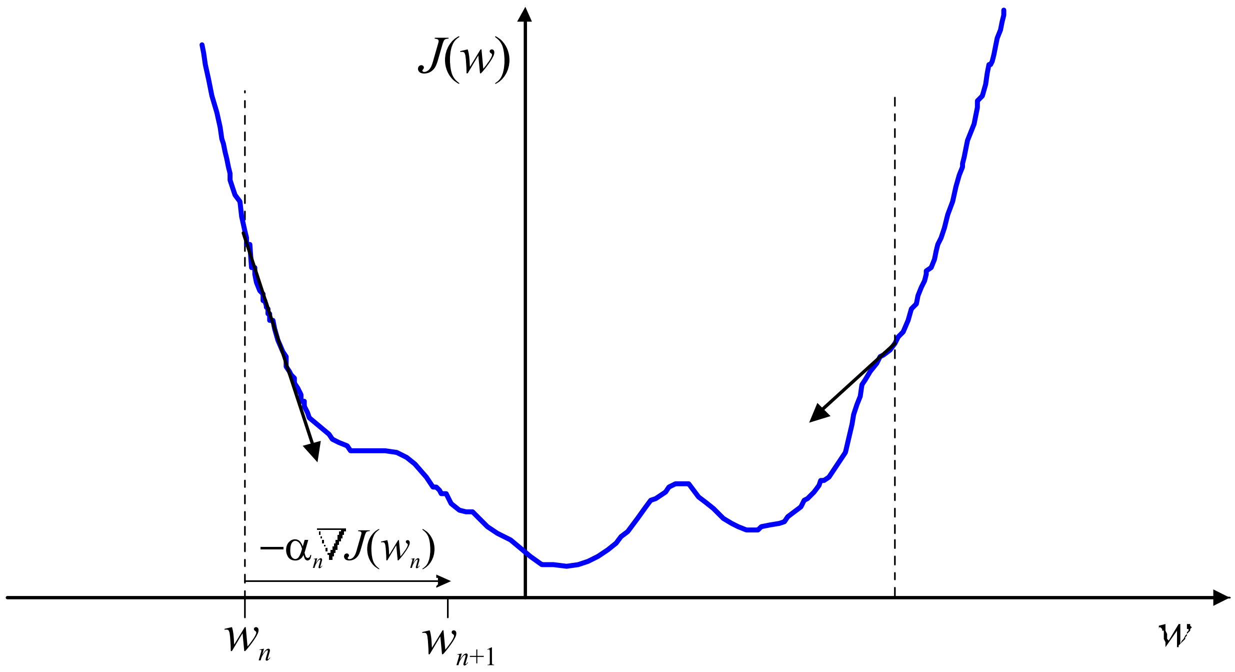
Repeat:

1. Input training example and compute network outputs.
2. Adjust output weights using gradients:
$$w^o_{jk}(p+1) = w^o_{jk}(p) + \alpha\, h_j e_k$$
3. Adjust hidden-layer weights:
$$w^h_{ij}(p+1) = w^h_{ij}(p) + \alpha x_i\, \sigma'_j(h_j) \sum_{k=1}^{l} (-e_k w^o_{jk})$$

Until satisfied or fixed number of epochs p

TÉCNICO LISBOA

First-order gradient methods



Second-order gradient methods

- **Update rule** for the weights:

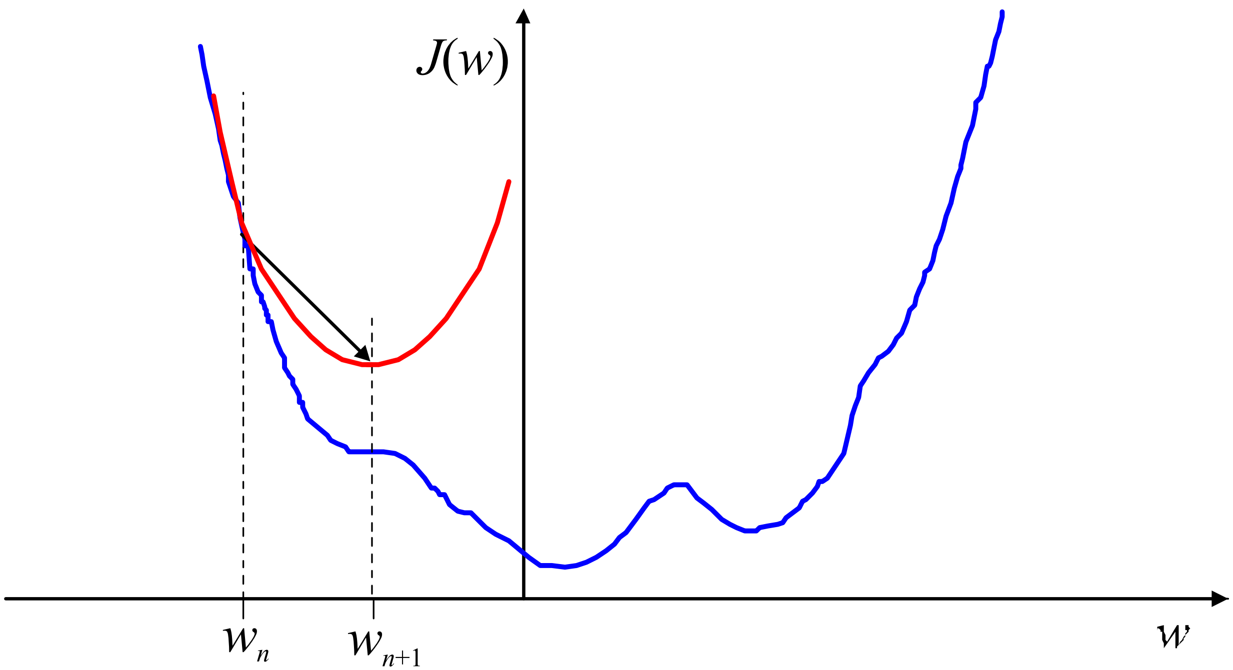
$$\mathbf{w}(p+1) = \mathbf{w}(p) - \mathbf{H}(\mathbf{w}(p))\nabla J(\mathbf{w}(p))$$

$$\mathbf{w}(p) = w_{ij}^{h}, w_{jk}^{o}, \ldots$$

  - $\mathbf{H}(\mathbf{w})$ is the Hessian matrix of $\mathbf{w}$

- Learning does not depend on a learning coefficient α
- Much more efficient in general

Second-order gradient methods

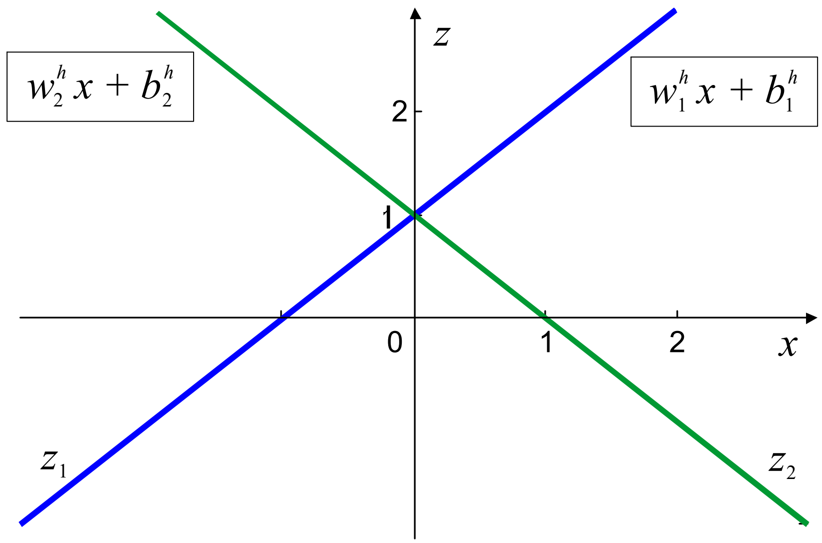


Approximation power

- General function approximators
- *“Feedforward neural network with one hidden layer and sigmoidal activation functions can approximate any continuous function arbitrarily well on a compact set”* (Cybenko)
- Intuitive relation

Function approximation

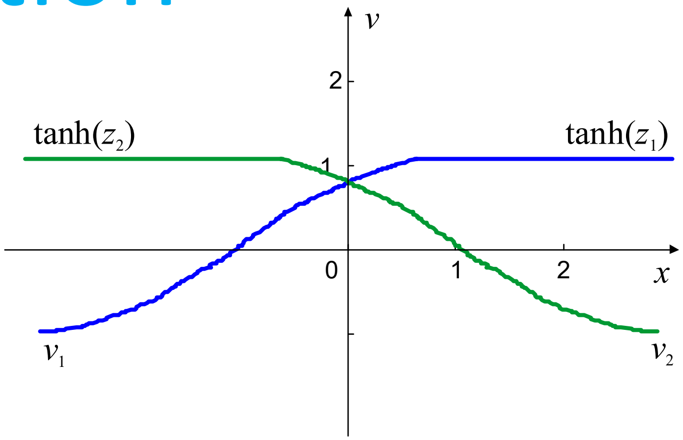
$$y = w_1^o \tanh(w_1^h x + b_1^h) + w_2^o \tanh(w_2^h x + b_2^h)$$

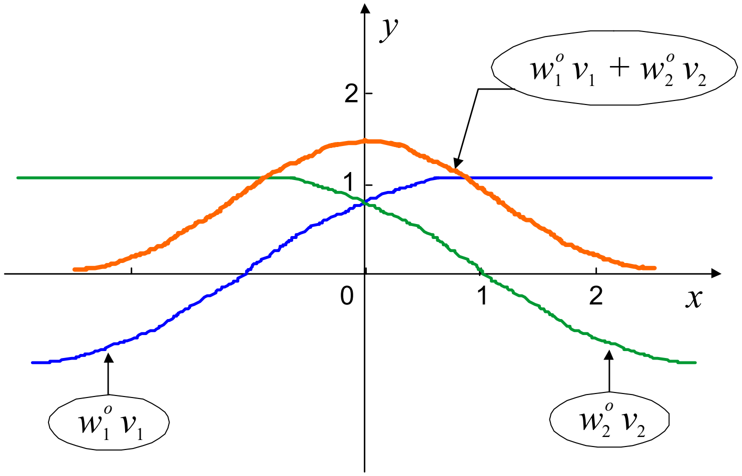


Activation (weighted summation)

Function approximation

Transformation through tanh





Summation of neuron outputs

RADIAL BASIS FUNCTION NETWORKS

Radial Basis Function Networks

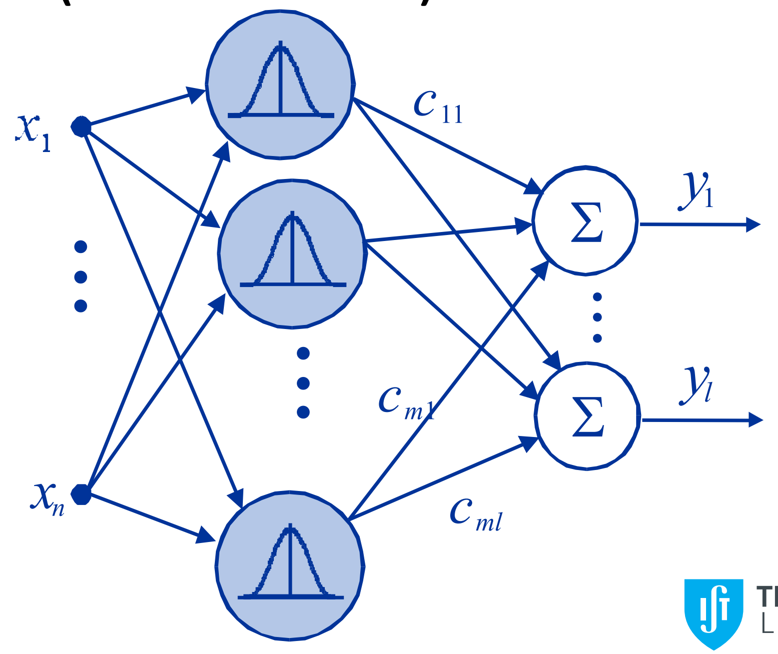
- Feedforward neural networks where hidden units do not implement an activation function; they represent a ***radial basis function***.
- Developed as an approach to improve accuracy and decrease training time complexity.

Radial Basis Function Networks

- Activation functions are radial basis functions
- Activation level of i^{th} receptive field (hidden unit):

$$R_i(\mathbf{x}) = R_i\left(\frac{\|\mathbf{x}-\mathbf{u}_i\|}{\sigma_i}\right)$$

  - $\mathbf{u}_i$ – center of basis function
  - σ_i – spread of basis function
  - $j = 1, 2, \ldots, n$
  - No connection weights between input and hidden layers



Radial Basis Function Networks

- Localized activation functions. Gaussian and logistic:

$$R_i(\mathbf{x}) = \exp\left(-\frac{\left\|\mathbf{x}-\mathbf{u}_i\right\|^2}{2\sigma_i^2}\right) \qquad R_i(\mathbf{x}) = \frac{1}{1+\exp\left(\left\|\mathbf{x}-\mathbf{u}_i\right\|^2 / \sigma_i^2\right)}$$

- **Weighted sum** or **average output**:

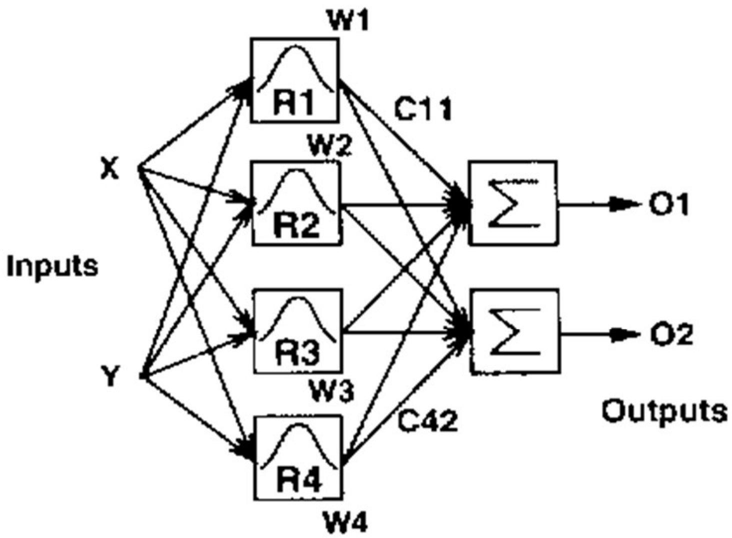
$$y(\mathbf{x}) = \sum_{i=1}^{H} c_i w_i = \sum_{i=1}^{H} c_i R_i(\mathbf{x}) \qquad y(\mathbf{x}) = \frac{\sum_{i=1}^{H} c_i R_i(\mathbf{x})}{\sum_{i=1}^{H} R_i(\mathbf{x})}$$

- c_i can be constants or functions of inputs: $c_i = \mathbf{a}_i^{\mathrm{T}}\mathbf{x} + b_i$

TÉCNICO LISBOA

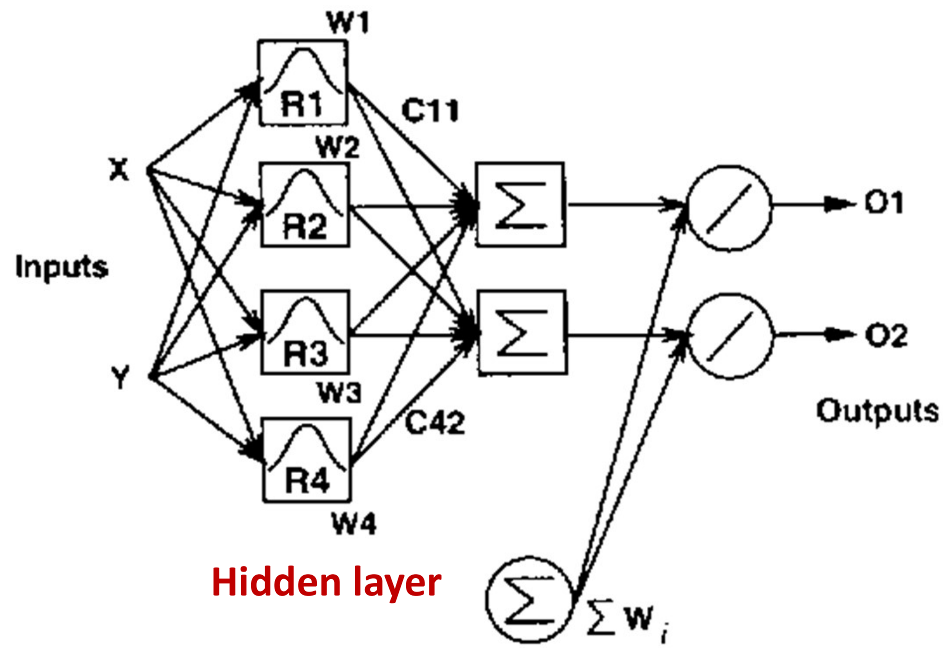
RBFN architecture

Weighted sum



Hidden layer

Weighted average



Hidden layer

Localized activation functions in the hidden layer

RBFN learning

- Supervised learning to update all parameters (e.g. with Genetic Algorithms)
- Sequential training: fix basis functions and then adjust output weights by:
  - orthogonal least squares
  - data clustering
  - soft competition based on “maximum likelihood estimate”
- σ_i sometimes estimated based on standard deviations
- Many other schemes also exist

Least-squares estimate of weights

- Given basis functions R and a set of input-output data: $[\mathbf{x}_k, y_k]$, $k = 1,\ldots,N$, estimate optimal weights c_{ij}

1. Compute the output of the neurons:

$$R_i(x_k) = e^{-\frac{\|\mathbf{x}_k - \mathbf{u}_i\|^2}{2\sigma_i^2}}$$

   The output is linear in the weights: $\mathbf{y} = \mathbf{R}\,\mathbf{c}$.

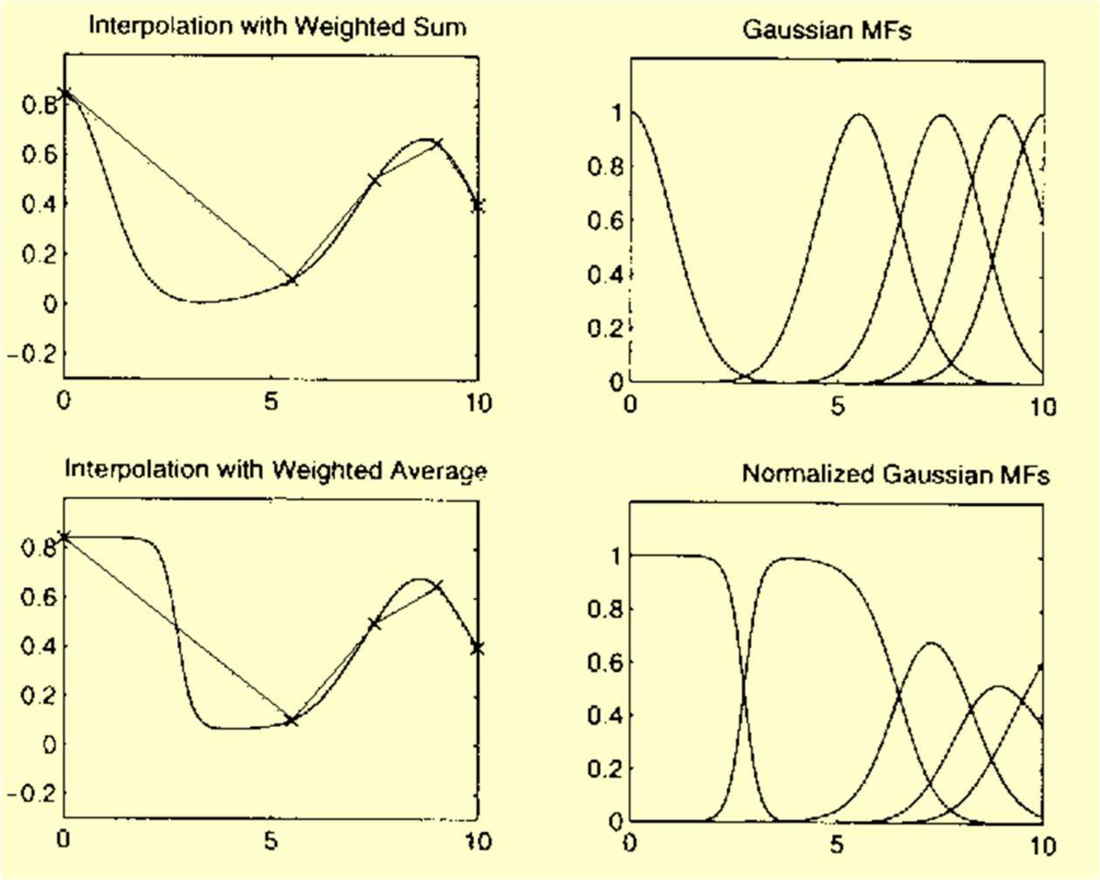
2. Least squares estimate: $\mathbf{c} = [\mathbf{R}^T\mathbf{R}]^{-1}\mathbf{R}^T\mathbf{y}$

RBFN and Sugeno systems

Equivalent if the following hold:

- Both RBFN and TS use same aggregation method for output (weighted sum or weighted average).
- Number of basis functions in RBFN equals number of rules in TS.
- TS uses Gaussian membership functions with same σ (variance) as basis functions and rule firing is determined by product.
- RBFN response function (c_i) and TS rule consequents are equal.

General function approximator



Approximation properties of NN

- [**Cybenko, 1989**]: A feedforward NN with *at least one hidden layer* can approximate **any** continuous function $\mathcal{R}^p \rightarrow \mathcal{R}^n$ on a compact interval, if sufficient hidden neurons are available.
- [**Barron, 1993**]: A feedforward NN with *one hidden layer* and *sigmoidal activation functions* can achieve an *integrated squared error of the order* $J = \mathcal{O}(1 / h)$.
  - independently of the dimension of the input space p
  - h: number of hidden neurons (for smooth functions)

Approximation properties

- For a *basis function expansion* (polynomial, trigonometric, singleton fuzzy model, etc.) with h terms, $J = \mathcal{O}(1 / h^{2/p})$, where p is the dimension of the input.
- **Examples:**

1. $p = 2$: ***polynomial*** $J = \mathcal{O}(1 / h^{2/2}) = \mathcal{O}(1 / h)$

neural net $J = \mathcal{O}(1 / h)$

2. $p = 10, h = 21$: ***polynomial*** $J = \mathcal{O}(1/21^{2/10}) = 0.54$

neural net $J = \mathcal{O}(1/21) = 0.048$

Example of aproximation

- **To achieve the same accuracy:**

- $J = \mathcal{O}(1 / h_n) = \mathcal{O}(1 / h_b),$
- $h_n = h_b^{2/p}, h_b = \sqrt{h_n^p} = \sqrt{21^{10}} \approx 4 \times 10^6$
-

ADAPTIVE NEURO-FUZZY INFERENCE SYSTEMS

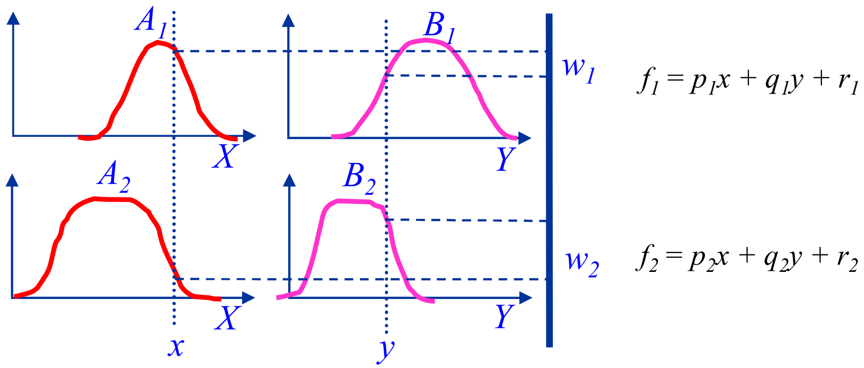
ANFIS

Adaptive Neuro-Fuzzy Inference Systems (ANFIS)

- Takagi-Sugeno fuzzy system mapped onto a neural network structure.
- Different representations are possible, but one with 5 layers is the most common.
- Network nodes in different layers have different structures.

ANFIS

- Consider a first-order Sugeno fuzzy model, with two inputs, *x* and *y*, and one output, *z*.
- **Rule set**
  - **Rule 1: If *x* is A_1 and *y* is B_1, then $f_1 = p_1x + q_1y + r_1$**
  - **Rule 2: If *x* is A_2 and *y* is B_2, then $f_2 = p_2x + q_2y + r_2$**

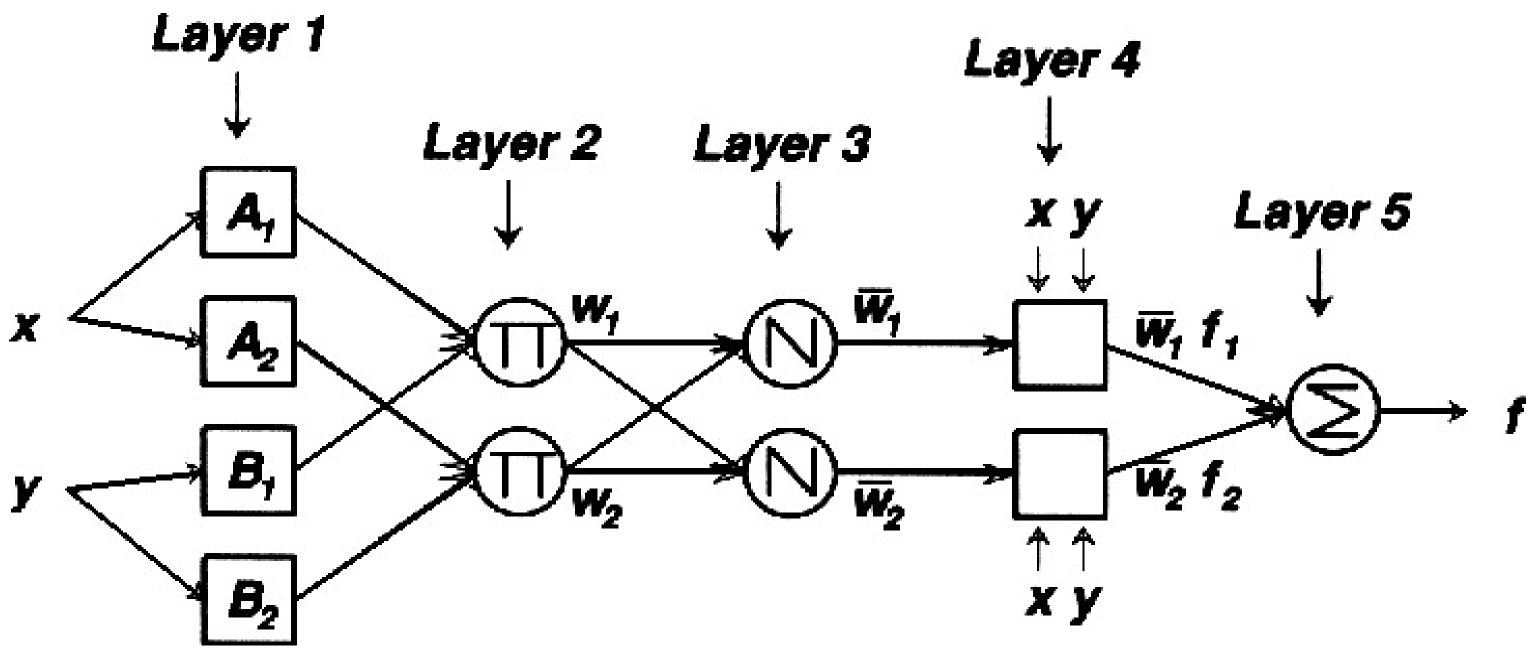


Weighted fuzzy-mean:

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2}$$
$$= \overline{w}_1 f_1 + \overline{w}_2 f_2$$

ANFIS architecture

- Corresponding equivalent ANFIS architecture:



ANFIS layers

- **Layer 1**: every node is an adaptive node with node function:

$$O_{1,i} = \mu_i(x_i)$$

  - Parameters in this layer are called ***premise parameters***.
- **Layer 2**: every node is fixed whose output (representing firing strength) is the product of the inputs:

$$O_{2,i} = w_i = \prod_j \mu_j$$

- **Layer 3**: every node is fixed (normalization): $O_{3,i} = \overline{w}_i = \dfrac{w_i}{\sum_j w_j}$

ANFIS layers

- **Layer 4**: every node is adaptive (***consequent parameters***) *:*

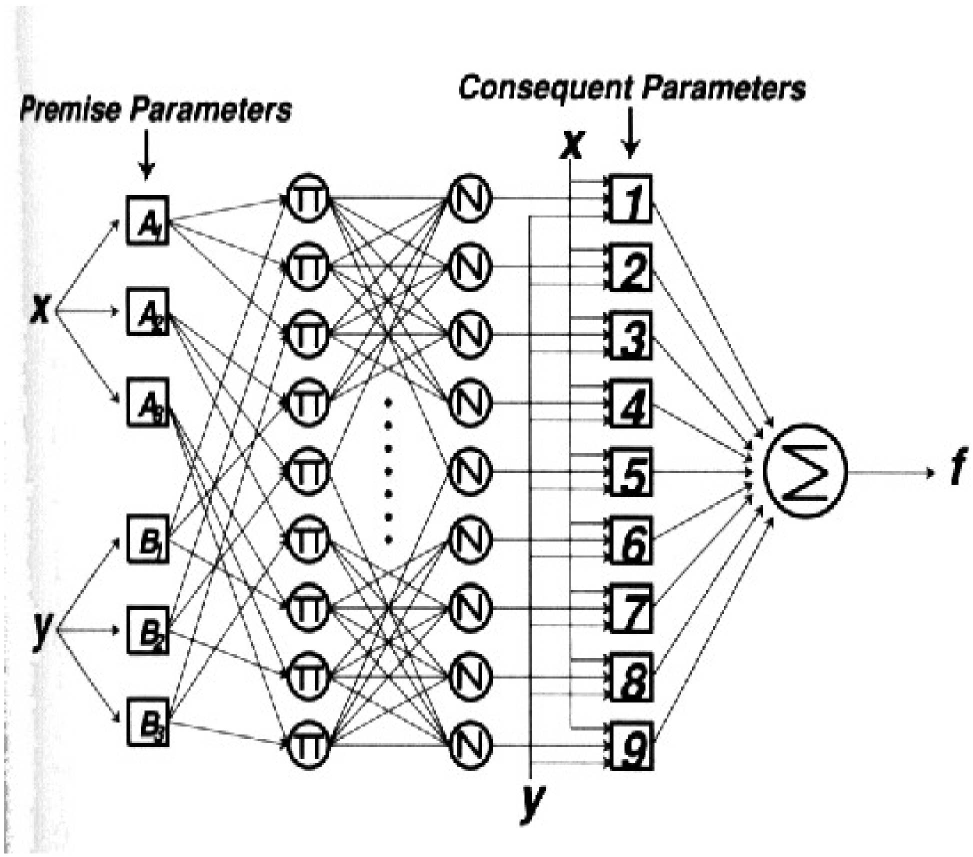
$$O_{4,i} = O_{3,i} f_i = \overline{w}_i (p_0 + p_1 x_1 + \ldots + p_n x_n)$$

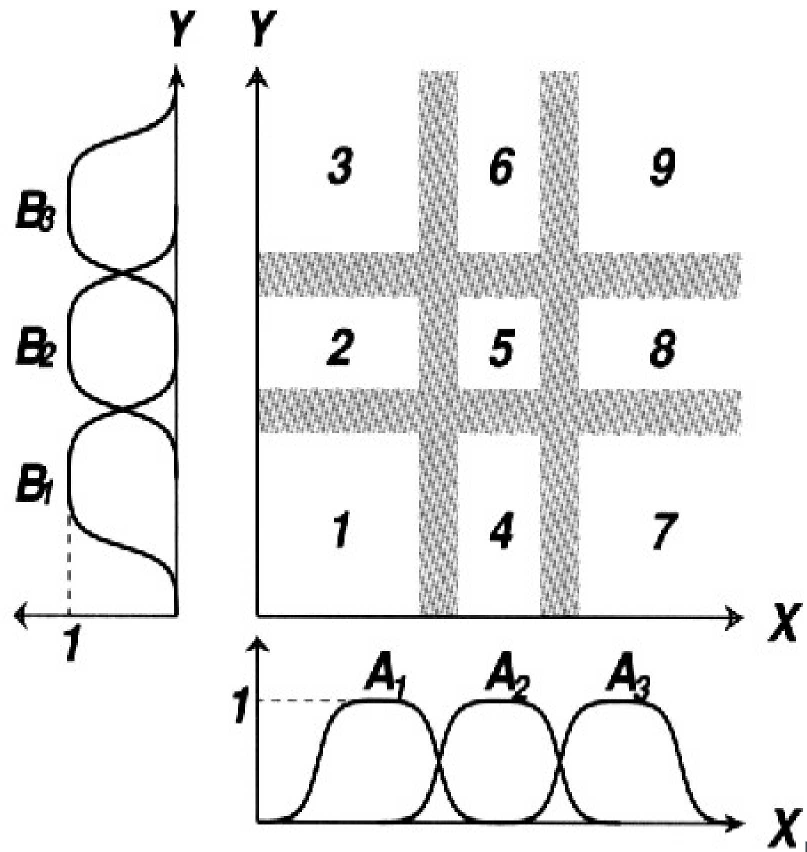
- **Layer 5**: single node, sums up inputs:

$$O_{5,i} = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}$$

➢ ***Adaptive network is functionally equivalent to a Sugeno fuzzy model!***

ANFIS with multiple rules





Hybrid learning for ANFIS

- Consider the two rules ANFIS with two inputs x and y and one output z;
- Let the premise parameters be fixed;
- ANFIS output is given by linear combination of consequent parameters p, q and r:

$$
\begin{aligned}
z &= \frac{w_1}{w_1 + w_2} f_1 + \frac{w_2}{w_1 + w_2} f_2 \\
&= \overline{w}_1(p_1 x + q_1 y + r_1) + \overline{w}_2(p_2 x + q_2 y + r_2) \\
&= (\overline{w}_1 x)p_1 + (\overline{w}_1 y)q_1 + (\overline{w}_1)r_1 + (\overline{w}_2 x)p_2 + (\overline{w}_2 y)q_2 + (\overline{w}_2)r_2 \\
&= \mathbf{A\theta}
\end{aligned}
$$

TÉCNICO LISBOA

Hybrid learning for ANFIS

- Partition total parameters set S as:
  - S_1: set of **premise** (nonlinear) parameters
  - S_2: set of **consequent** (linear) parameters
- $\boldsymbol{\theta}$: unknown vector which elements are parameters in S_2
- $z = \mathbf{A}\boldsymbol{\theta}$: standard linear least-squares problem
- **Best solution** for $\boldsymbol{\theta}$ that minimizes $\|\mathbf{A}\boldsymbol{\theta} - z\|^2$ is the *least-squares estimator* $\boldsymbol{\theta}^*$:

$$\boldsymbol{\theta}^* = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}z$$

TÉCNICO LISBOA

Hybrid learning for ANFIS

- What if premise parameters are **not optimal**?
- Combine *steepest descent* and *least-squares estimator* to update parameters in adaptive network.
- Each *epoch* is composed of:

1. **Forward pass**: node outputs go forward until Layer 4 and consequent parameters are identified by *least-squares estimator*;
2. **Backward pass**: error signals propagate backward and the premise parameters are updated by *gradient descent*.

TÉCNICO LISBOA

Hybrid learning for ANFIS

- Error signals: derivative of error measure with respect to each node output.

	Forward pass	Backward pass
Premise parameters	Fixed	Gradient descent
Consequent parameters	Least-squares estimator	Fixed
Signals	Node outputs	Error signals

- Hybrid approach converges much faster by reducing the search space of pure backpropagation method.

TÉCNICO LISBOA

Stone-Weierstrass theorem

Let D be a compact space of N dimensions and let $\mathcal{F}$ be a set of continuous real-valued functions on D satisfying:

1. **Identity function**: the constant $f(x) = 1$ is in $\mathcal{F}$.
2. **Separability**: for any two points $x_1 \neq x_2$ in D, there is an f in $\mathcal{F}$ such that $f(x_1) \neq f(x_2)$.
3. **Algebraic closure**: if f and g are two functions in $\mathcal{F}$, then fg and $af + bg$ are also in $\mathcal{F}$ for any reals a and b.

Then, $\mathcal{F}$ is dense in the closure $C(D)$ of D, i.e.:

$$\forall\ \epsilon > 0,\ \forall\ g \in C(D),\ \exists\ f \in \mathcal{F} : |g(x) - f(x)| < \epsilon,\ \forall\ x \in D.$$

Universal approximator ANFIS

- According to Stone-Weierstrass theorem, an ANFIS has ***unlimited approximation power*** for matching any continuous nonlinear function arbitrarily well
- ***Identity***: obtained by having a constant consequent
- ***Separability***: obtained by selecting appropriate parameters in the network

Algebraic closure

- Consider two systems with two rules and final outputs:

$$z = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} \text{ and } \hat{z} = \frac{\widehat{w}_1 \hat{f}_1 + \widehat{w}_2 \hat{f}_2}{\widehat{w}_1 + \widehat{w}_2}$$

- **Additive:**

$$az + b\hat{z} = a\frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} + b\frac{\widehat{w}_1 \hat{f}_1 + \widehat{w}_2 \hat{f}_2}{\widehat{w}_1 + \widehat{w}_2}$$

$$= \frac{w_1\widehat{w}_1(af_1 + b\hat{f}_1) + w_1\widehat{w}_2(af_1 + b\hat{f}_2) + w_2\widehat{w}_1(af_2 + b\hat{f}_1) + w_2\widehat{w}_2(af_2 + b\hat{f}_2)}{w_1\widehat{w}_1 + w_1\widehat{w}_2 + w_2\widehat{w}_1 + w_2\widehat{w}_2}$$

- Construct 4 rule inference system that computes:

$$az + b\hat{z}$$

Algebraic closure

- **Multiplicative:**

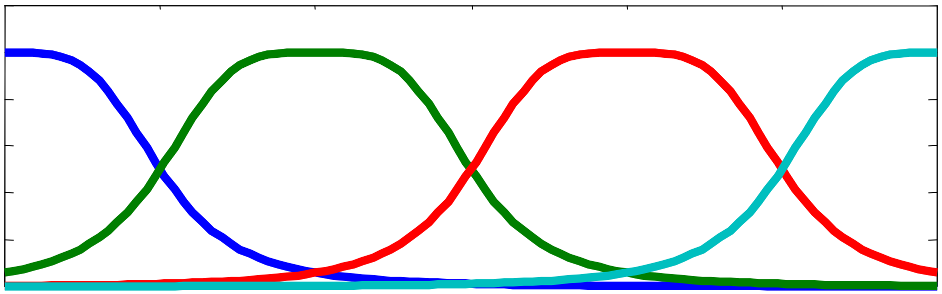
$$z\hat{z} = \left(\frac{w_1 f_1 + w_2 f_2}{w_1 + w_2}\right)\left(\frac{\hat{w}_1 \hat{f}_1 + \hat{w}_2 \hat{f}_2}{\hat{w}_1 + \hat{w}_2}\right)$$
$$= \frac{w_1 \hat{w}_1 f_1 \hat{f}_1 + w_1 \hat{w}_2 f_1 \hat{f}_2 + w_2 \hat{w}_1 f_2 \hat{f}_1 + w_2 \hat{w}_2 f_2 \hat{f}_2}{w_1 \hat{w}_1 + w_1 \hat{w}_2 + w_2 \hat{w}_1 + w_2 \hat{w}_2}$$

$$z\hat{z}$$

- Construct 4 rule inference system that computes:

Model building guidelines

- Select number of fuzzy sets per variable:
  - empirically by examining data or trial and error
  - using clustering techniques
  - using regression trees (CART)
- Initially, distribute bell-shaped membership functions evenly:



- Using an adaptive step size can speed up training.

How to design ANFIS?

- Initialization
  - Define number and type of inputs
  - Define number and type of outputs
  - Define number of rules and type of consequents
  - Define objective function and stop conditions
- Collect data
- Normalize inputs
- Determine initial rules
- Initialize network

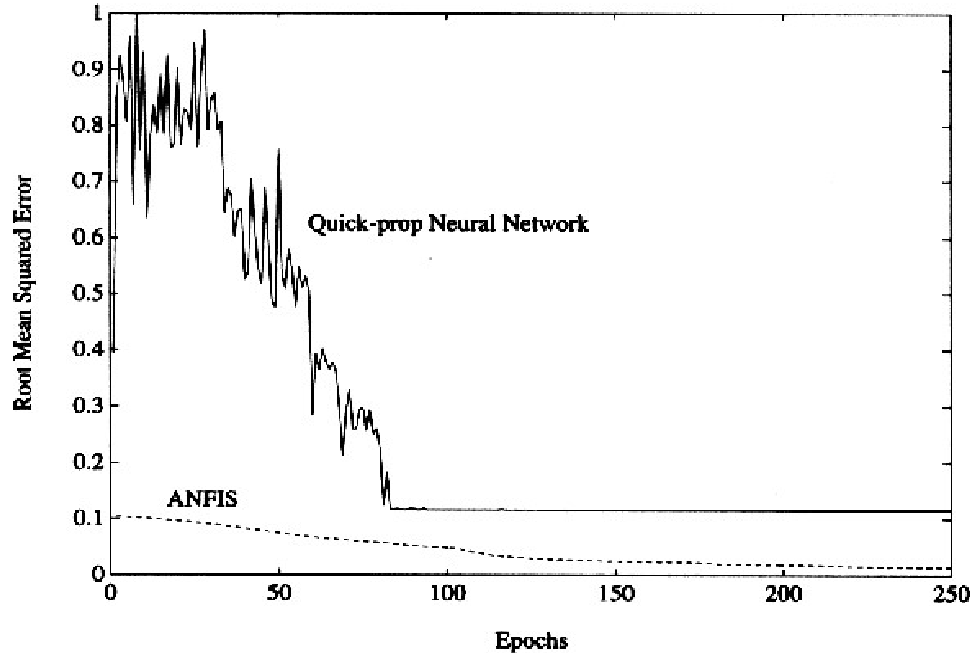
TRAIN

Ex. 1: Two-input sinc function

$$z = \sin c(x, y) = \frac{\sin(x)\sin(y)}{xy}$$

- Input range: $[-10,10]\times[-10,10]$, 121 training data pairs.
- Multi-Layer Perceptron vs. ANFIS:
  - **MLP:** 18 neurons in hidden layer, 73 parameters, quick propagation (best learning algorithm for backpropagation MLP).
  - **ANFIS:** 16 rules, 4 membership functions per variable, 72 fitting parameters (48 linear, 24 nonlinear), hybrid learning rule.

MLP vs. ANFIS results

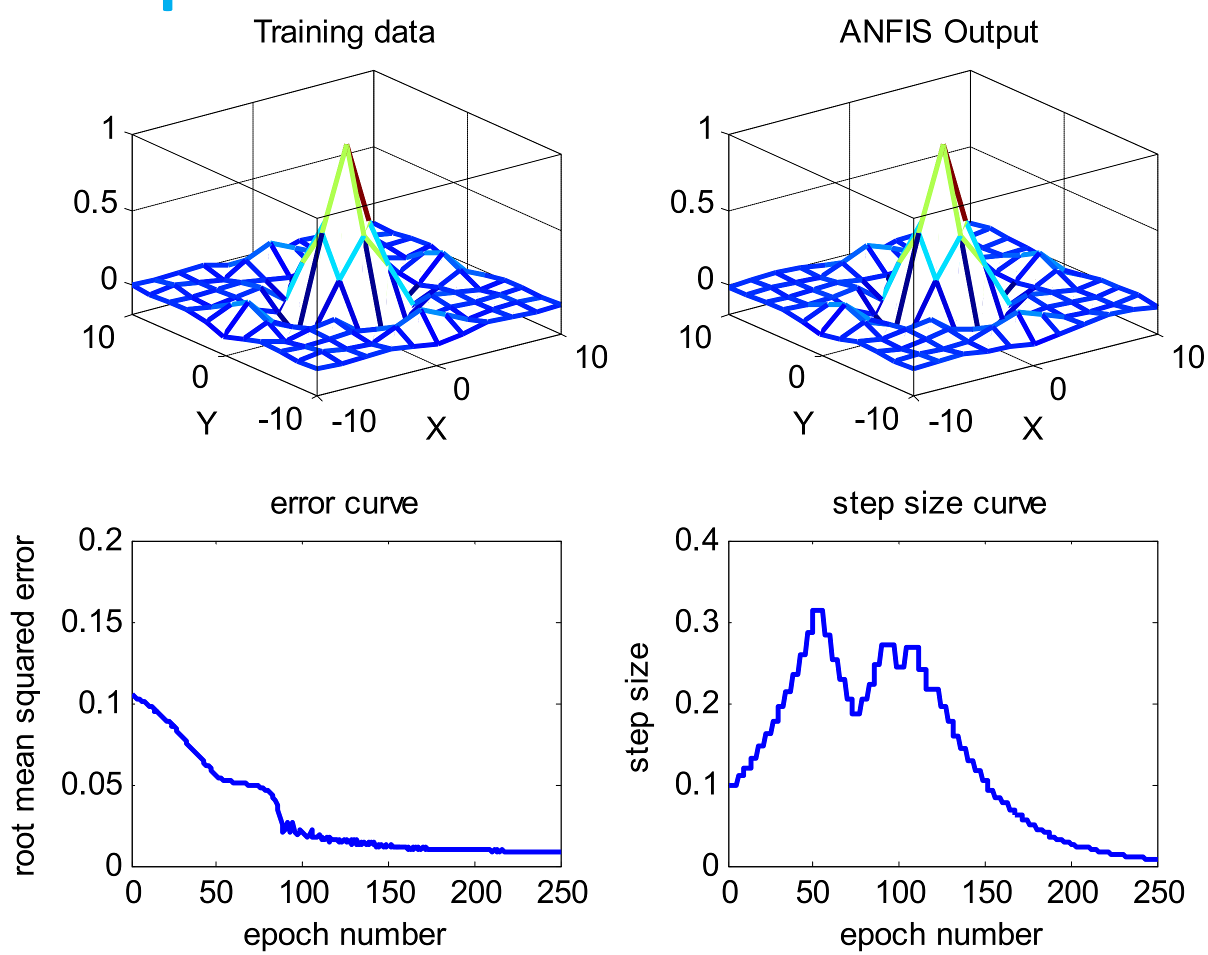


Average of 10 runs:

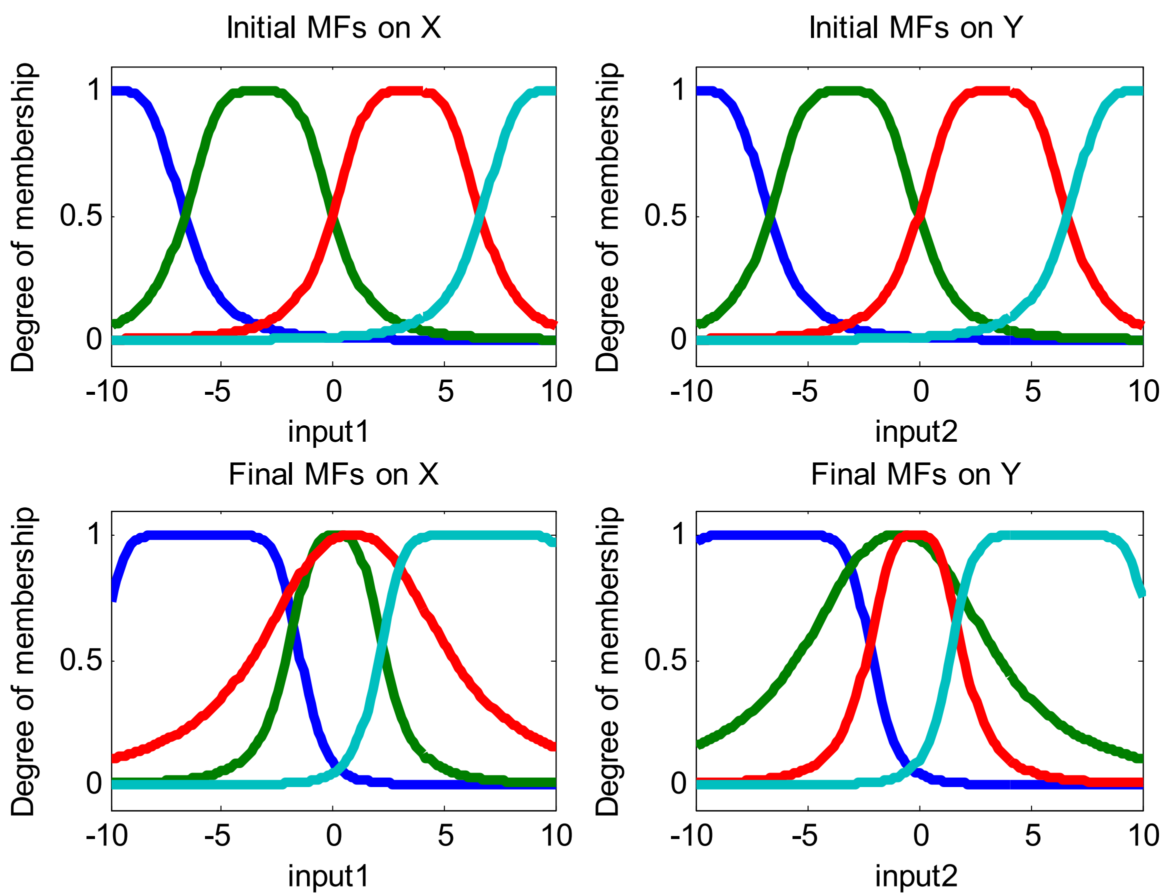
- **MLP:** different sets of initial random weights;
- **ANFIS:** 10 step sizes between 0.01 and 0.10.

  - ***MLP's approximation power decrease due to*:** learning processes trapped in local minima or some neurons can be pushed into saturation during training.

ANFIS output



ANFIS model

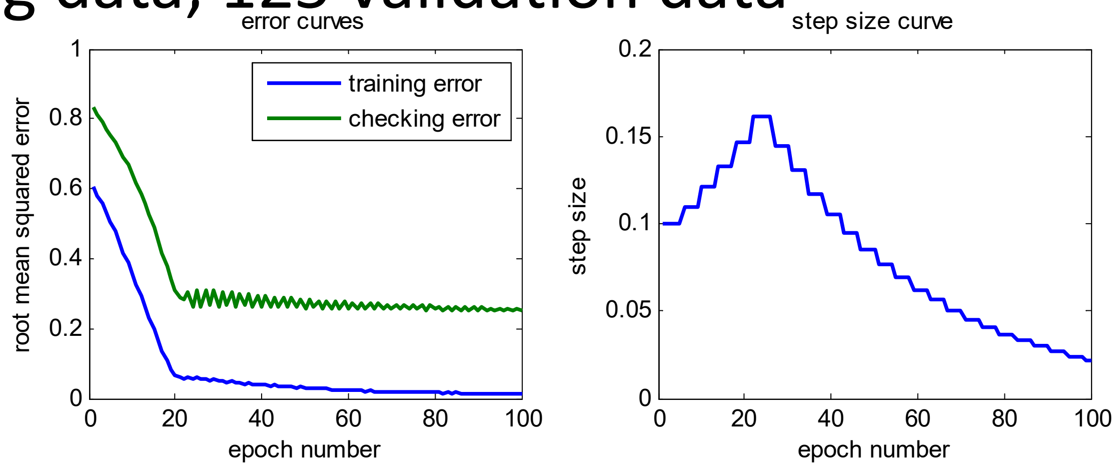


Initial MFs on X
Initial MFs on Y
Final MFs on X
Final MFs on Y
Degree of membership
1
0.5
0
-10
-5
0
5
10
input1
input2

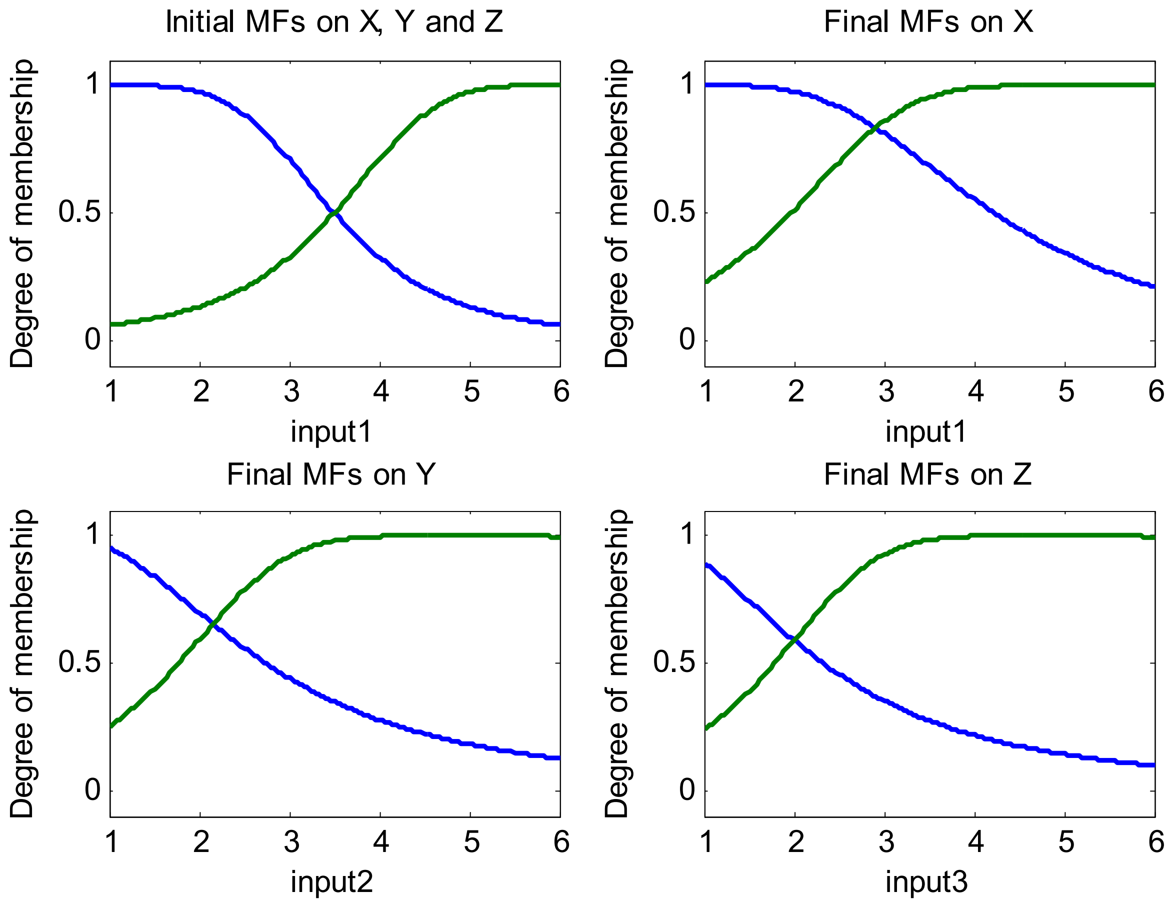
Ex. 2: 3-input nonlinear function

$$\text{output} = \left(1 + x^{0.5} + y^{-1} + z^{-1.5}\right)^2$$

- Two membership functions per variable, 8 rules
- Input ranges: [1,6]×[1,6]×[1,6]
- 216 training data, 125 validation data



ANFIS model



TÉCNICO LISBOA

Results comparison

[1] T. Kondo. Revised GMDH algorithm estimating degree of the complete polynomial. *Trans. of the Society of Instrument and Control Engineers*, 22(9):928:934, 1986.

[2] M. Sugeno and G. T. Kang, Structure Identification of fuzzy model. *Fuzzy Sets and Systems*, 28:15-33, 1988.

Model	Training error	Checking error	# Param.	Training data size	Checking data size
ANFIS	0.043%	1.066%	50	216	125
GMDH model [1]	4.7%	5.7%	-	20	20
Fuzzy model 1 [2]	1.5%	2.1%	22	20	20
Fuzzy model 2 [2]	0.59%	3.4%	32	20	20

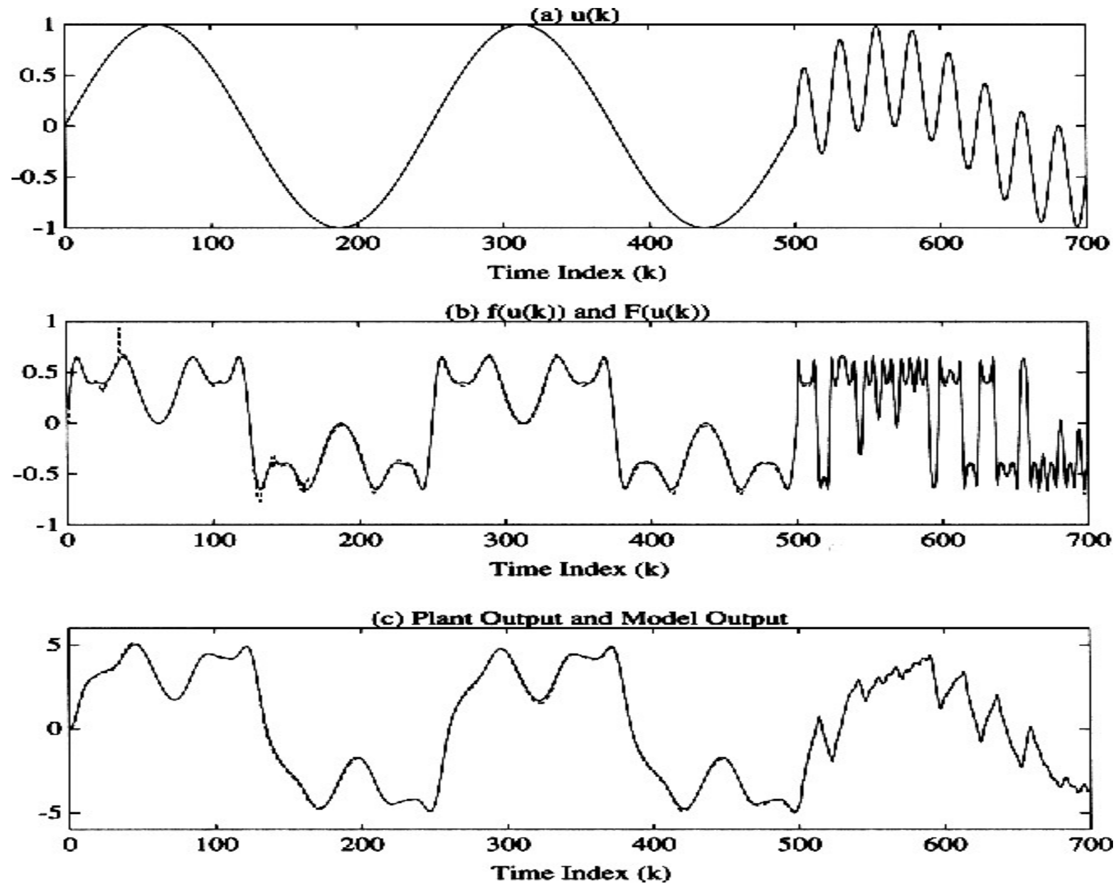
Group method of data handling – first deep learning methods back in 1971

$$\text{APE} = \text{Average Percentage Error} = \frac{1}{P}\sum_{i=1}^{P}\frac{|T(i)-O(i)|}{|T(i)|}.100\%$$

Ex. 3: Modeling dynamic system

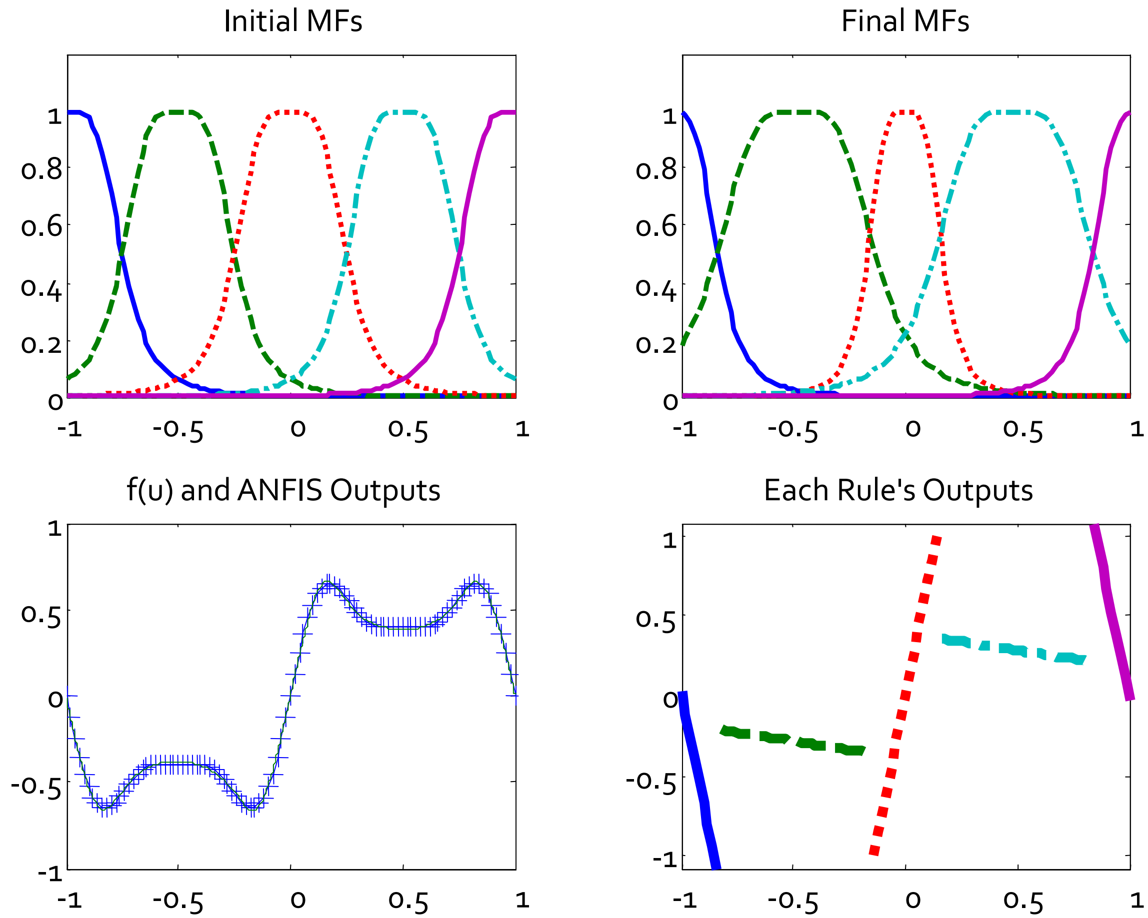
- Plant equation $$y(k+1) = 0.3y(k) + 0.6y(k-1) + f(u(k))$$
- $f(.)$ has the following form $$f(u) = 0.6\sin(\pi u) + 0.3\sin(3\pi u) + 0.1\sin(5\pi u)$$
- Estimate nonlinear function *F* with ANFIS $$\hat{y}(k+1) = 0.3\hat{y}(k) + 0.6\hat{y}(k-1) + F(u(k))$$
- Plant input: $$u(k) = \sin(2\pi k / 250)$$
- ANFIS parameters updated at each step (on-line)
- Learning rate: $\eta = 0.1$; forgetting factor: $\lambda = 0.99$
- ANFIS can adapt even after the input changes
- Question: was the input signal rich enough?

Plant and model outputs



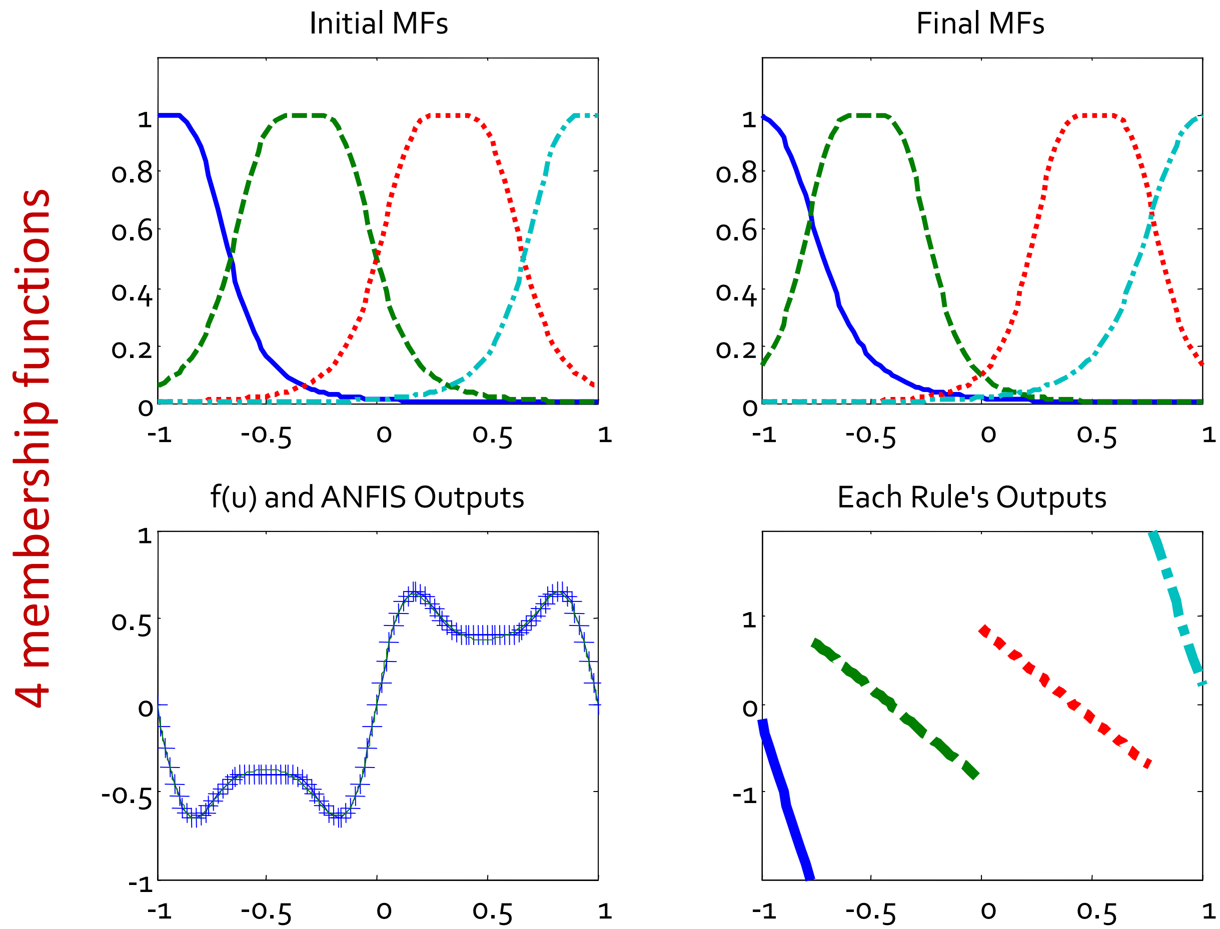
Effect of number of MFs

5 membership functions



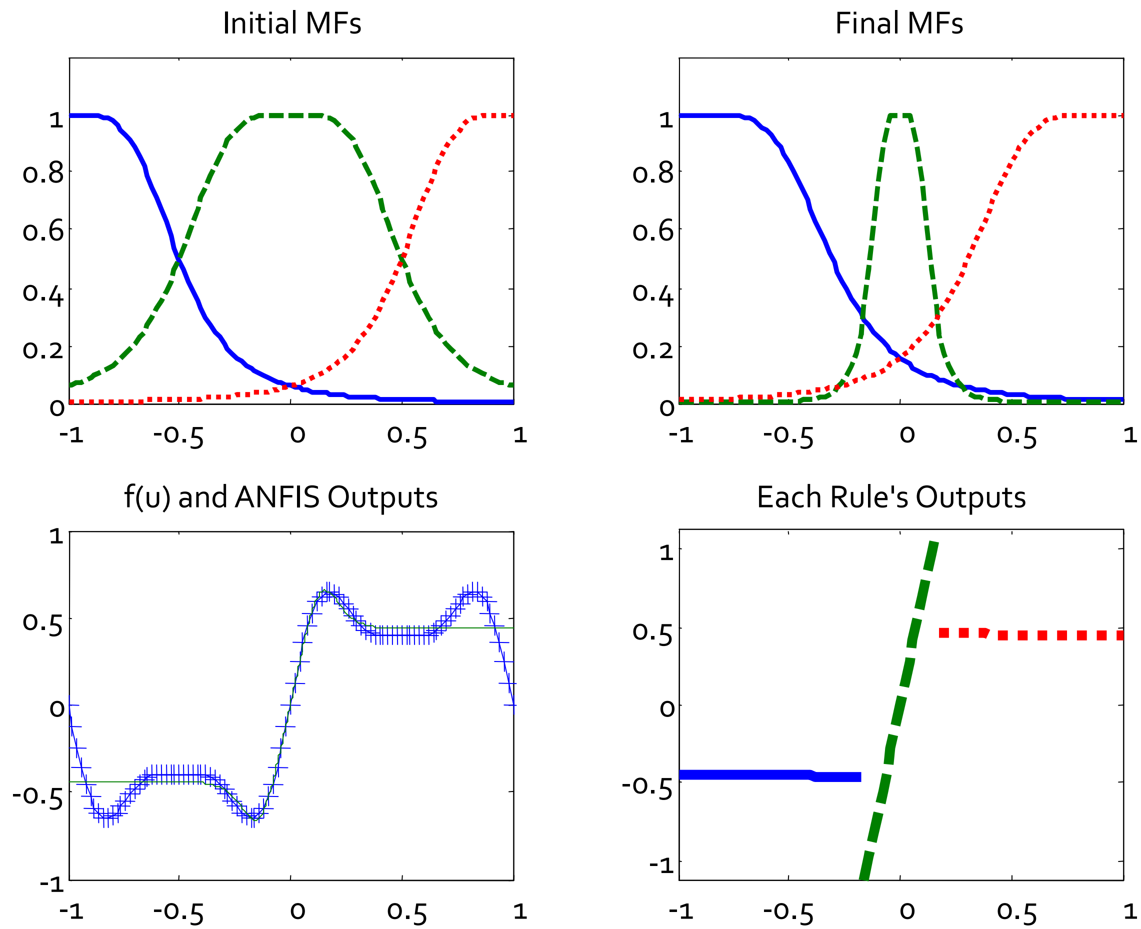
TÉCNICO LISBOA

Effect of number of MFs



Effect of number of MFs

3 membership functions



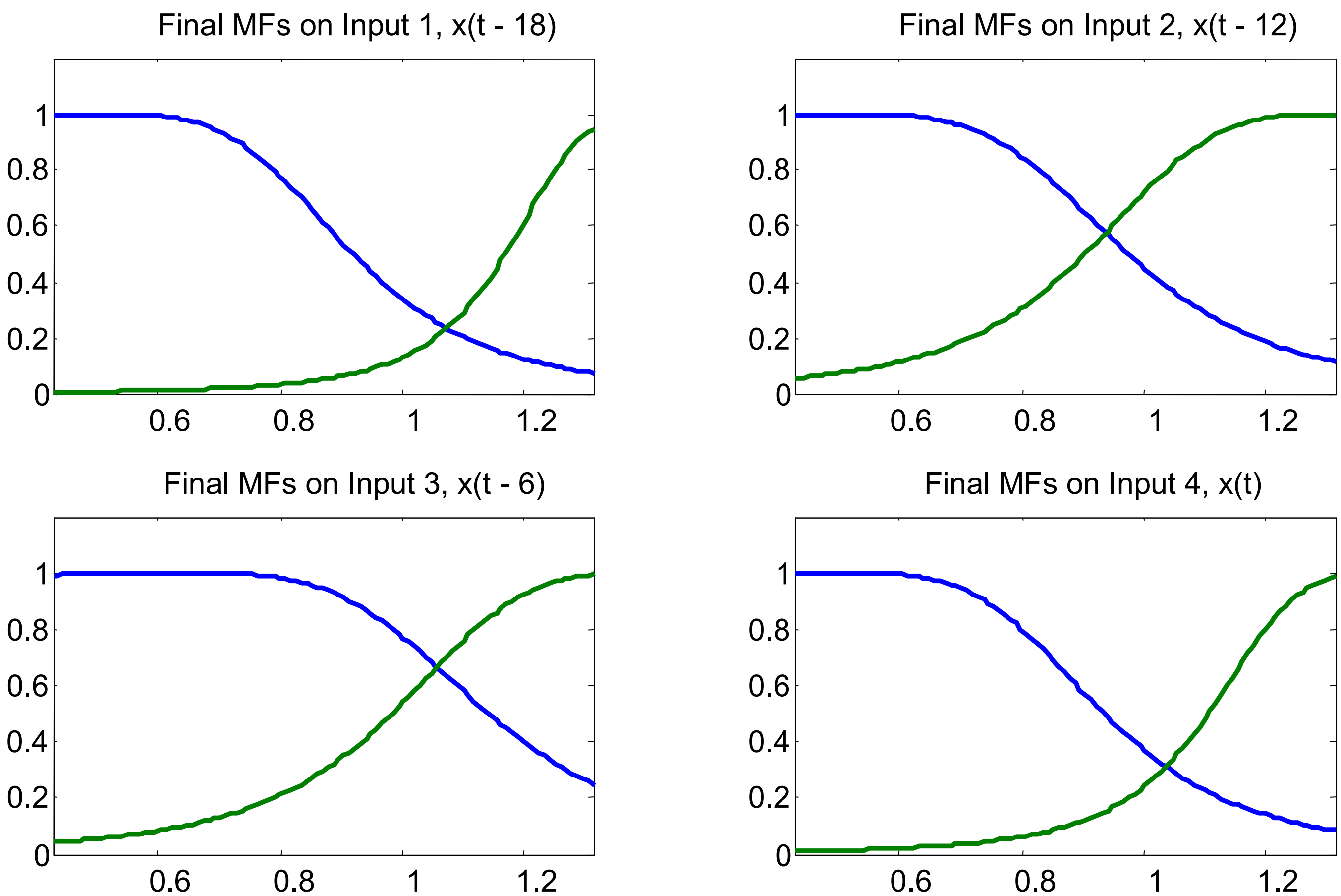
Ex. 4: Chaotic time series

- Consider a chaotic time series generated by

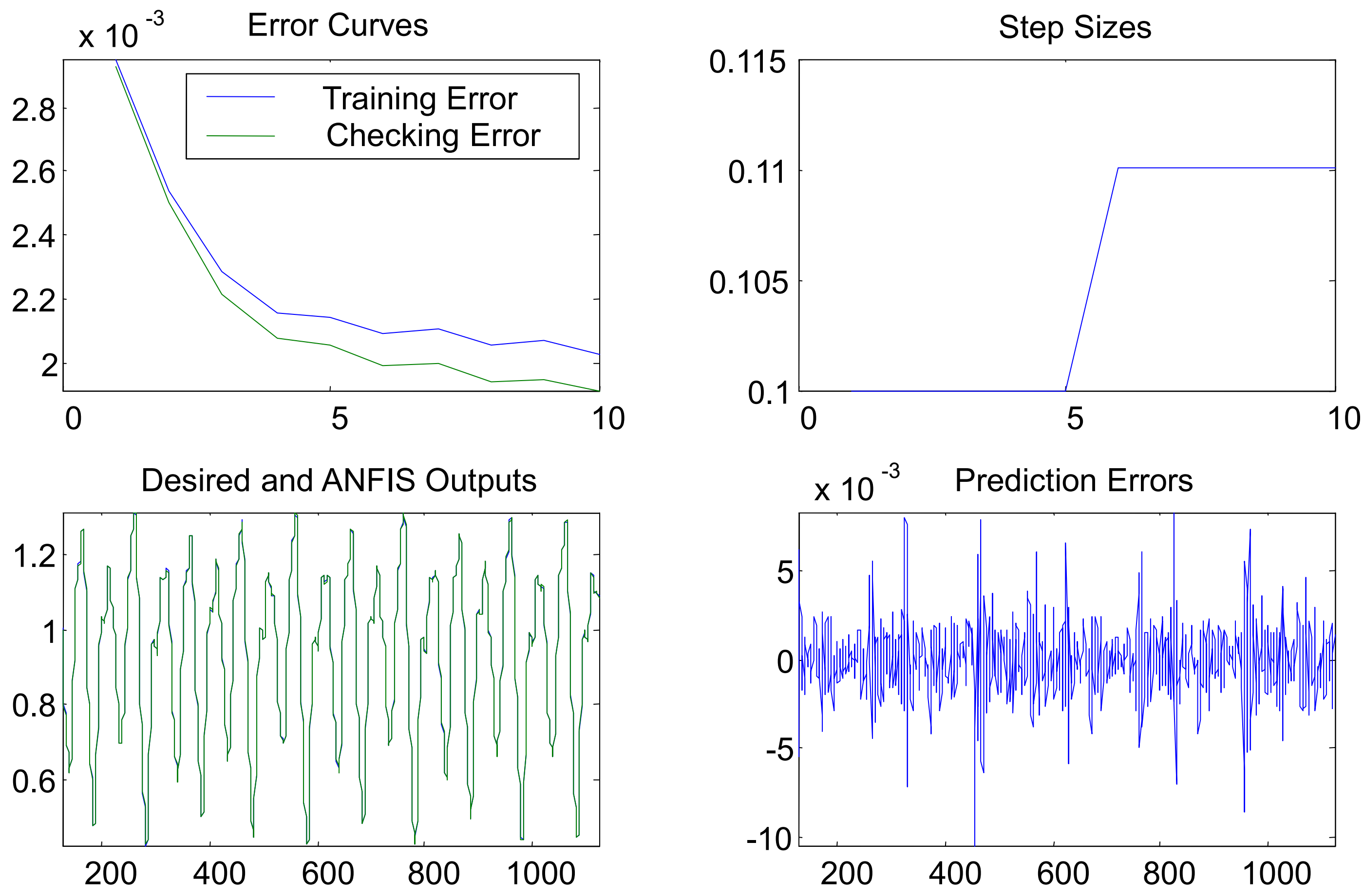
$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t)$$

- Task: predict system output at some future instance $t+P$ by using past outputs
- 500 training data, 500 validation data
- ANFIS input: $[x(t-18), x(t-12), x(t-6), x(t)]$
- ANFIS output: $x(t+6)$
- Two MFs per variable, 16 rules
- 104 parameters (24 premise, 80 consequent)
- Data generated from $t=118$ to $t=1117$

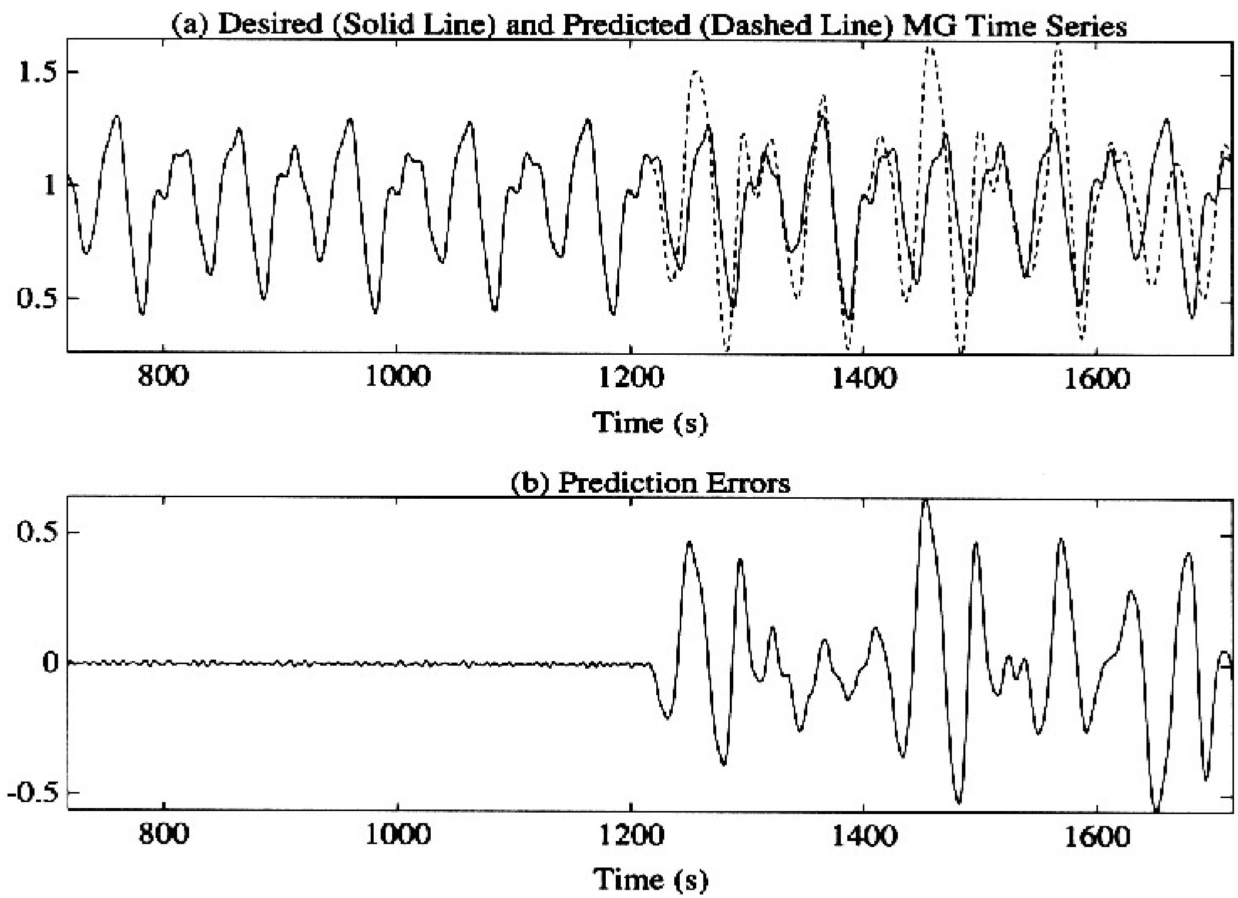
ANFIS model


Final MFs on Input 1, x(t - 18)
Final MFs on Input 2, x(t - 12)
Final MFs on Input 3, x(t - 6)
Final MFs on Input 4, x(t)
0
0.2
0.4
0.6
0.8
1
0.6
0.8
1
1.2

Model output



103^{rd} order AR model

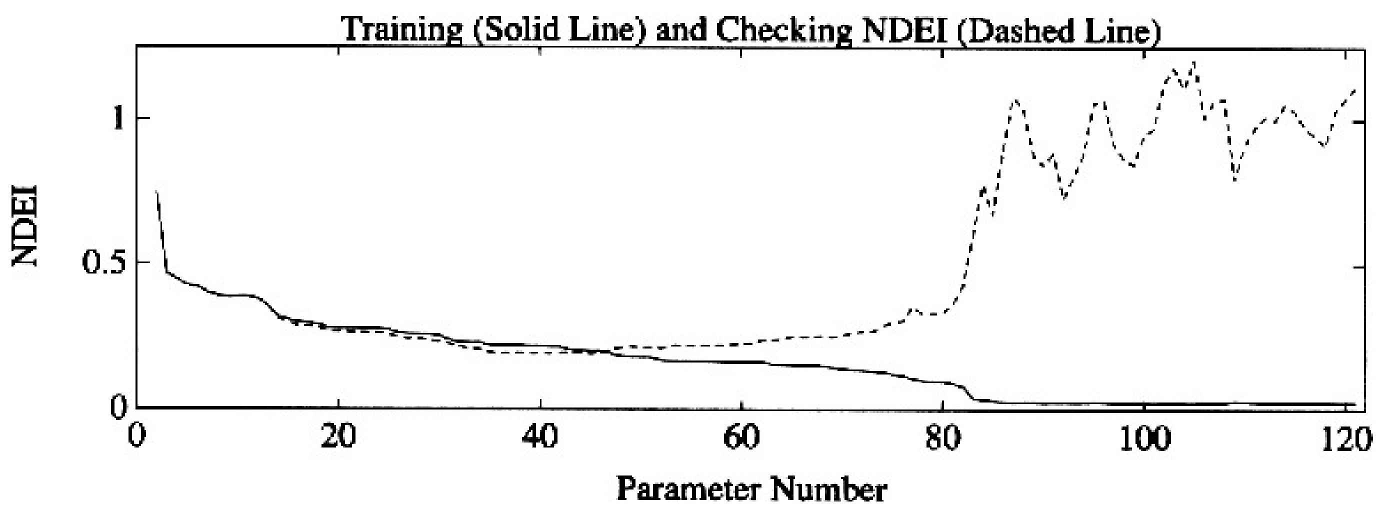


TÉCNICO LISBOA

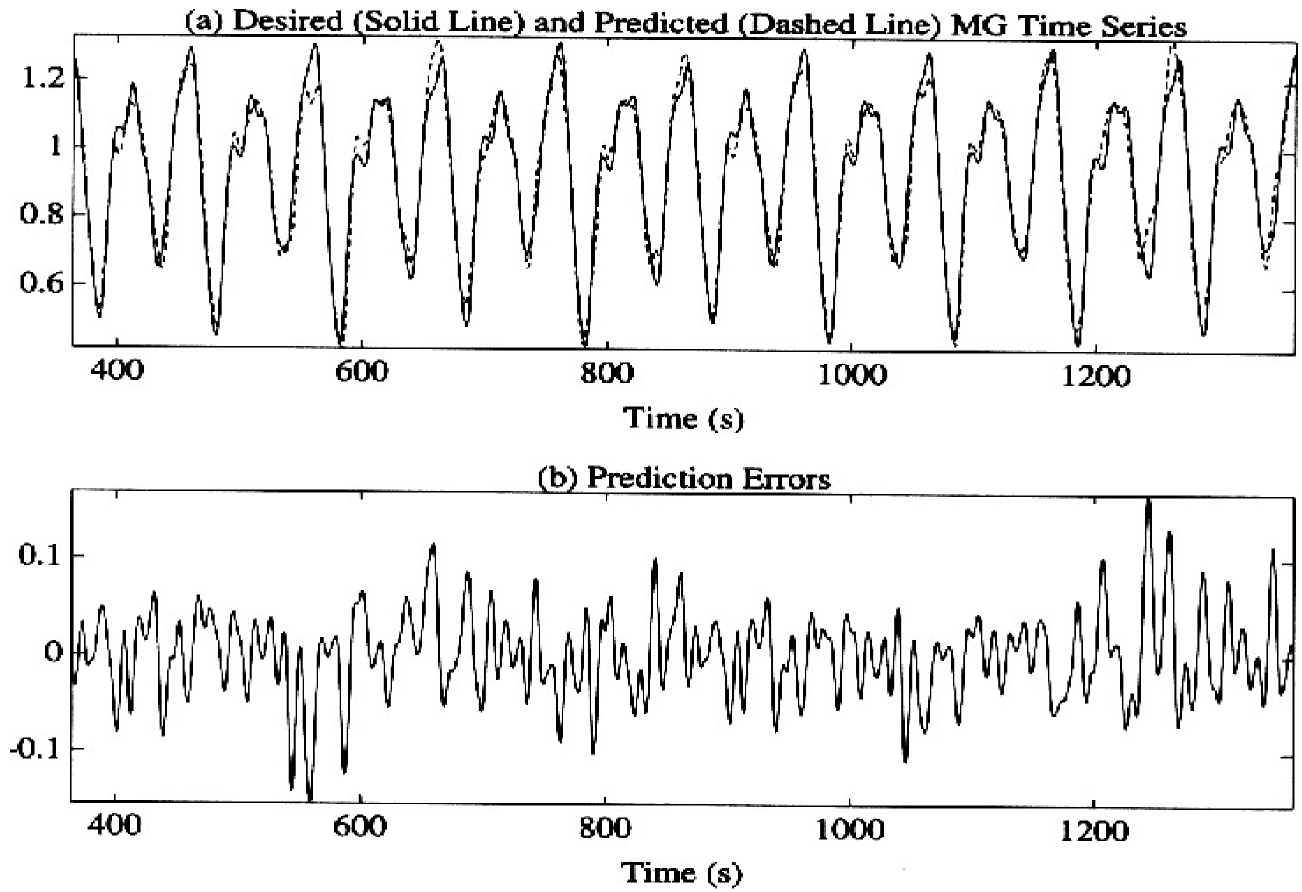
Order selection

$$y^{(n)}(t) + y^{(n-1)}(t) + \cdots + y^{(1)}(t) + y(t) = u(t)$$

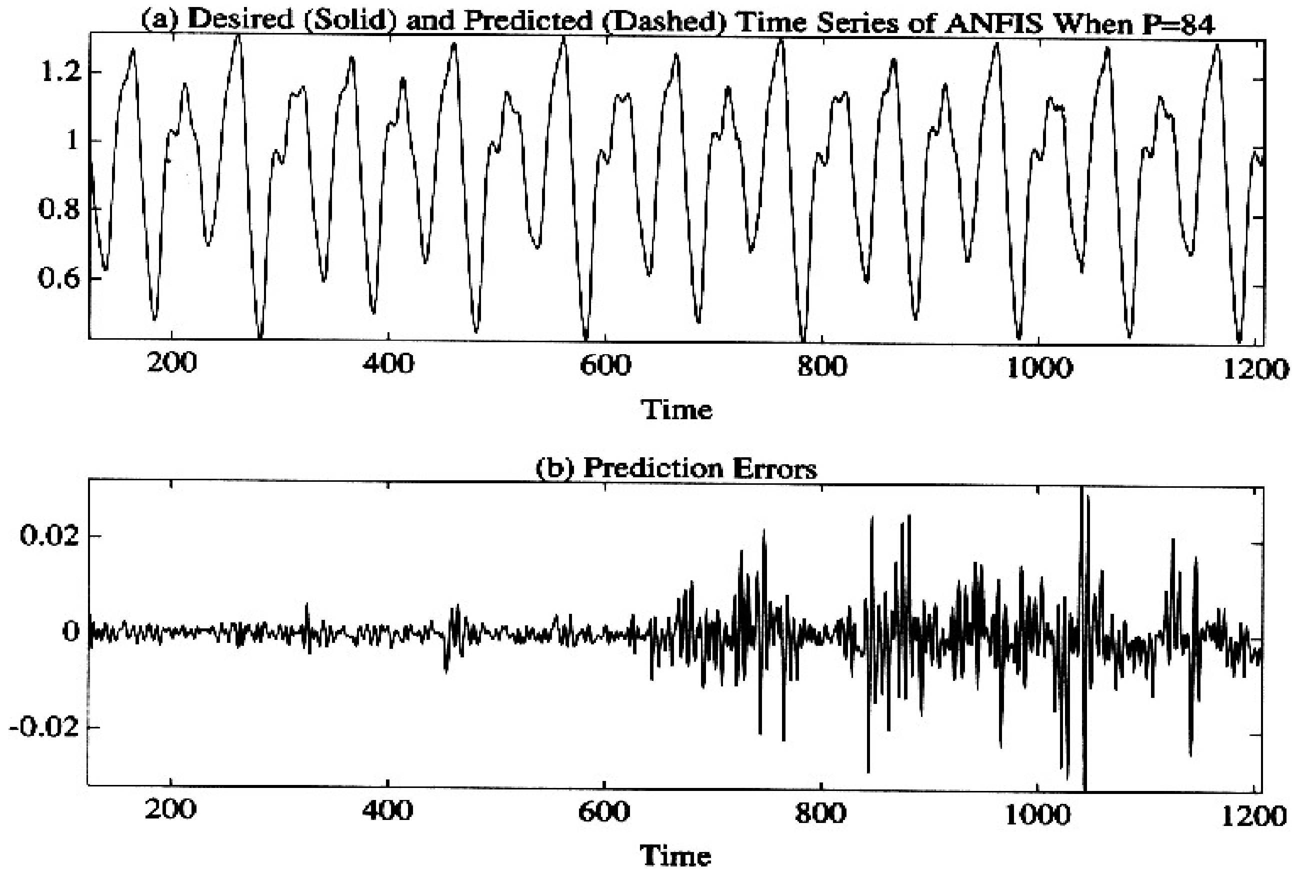
- Select optimal order of AR model in order to prevent overfitting
- Select the order that minimizes the error on a test set



44th order AR model



ANFIS output for P = 84



ANFIS extensions

- Different types of membership functions in layer 1
- Parameterized *t*-norms in layer 2
- Interpretability
  - constrained gradient descent optimization
  - bounds on fuzziness

$$E' = E + \beta \sum_{i=1}^{N_P} \overline{w}_i \ln(\overline{w}_i)$$

  - parameterize to reflect constraints
- Structure identification

TÉCNICO LISBOA