Supervised learning neural networks

- Multilayer perceptron
- Adaptive-Network-based Fuzzy Inference System (ANFIS)

First part based on slides by Walter Kosters

Neural networks

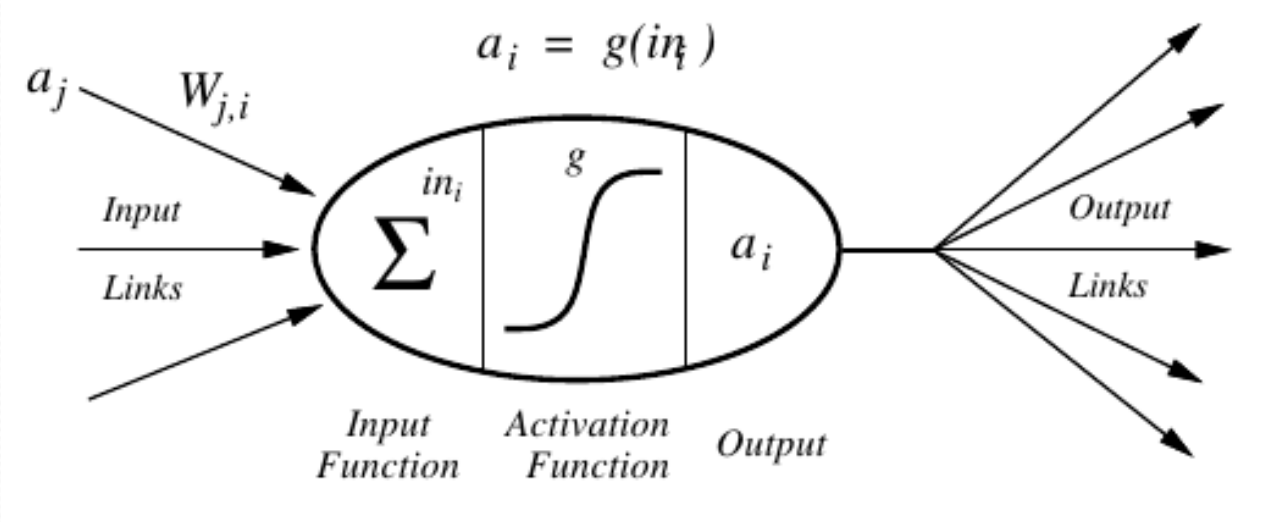
- Massively connected computational units inspired by the working of the human brain
- Provide a mathematical model for biological neural networks (brains)
- Characteristics:
  - learning from examples
  - adaptive and fault tolerant
  - robust for fulfilling complex tasks

Network variations

- Learning methods (supervised, unsupervised)
- Architectures (feedforward, recurrent)
- Output types (binary, continuous)
- Node types (uniform, hybrid)
- Implementations (software, hardware)
- Connection weights (adjustable, hard-wired)
- Inspirations (biological, psychological)

Artificial Neuron

- Input: $in_i = \sum_j W_{j,i} \; a_j$



(Neuron/Unit)

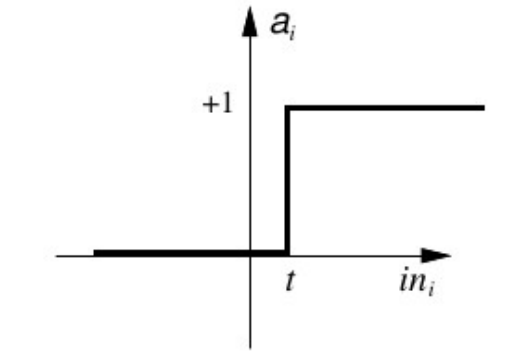
- Network with one unit: perceptron

Activation Functions

- ***Step***

  $step_t(x)=1 \quad if\ x >= t$

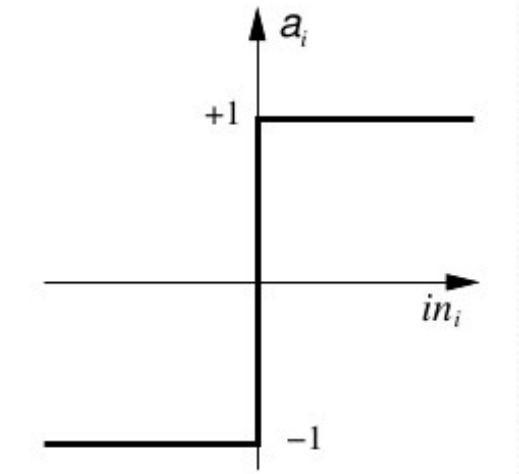
  $step_t(x)=0 \quad otherwise$



- ***Sign***

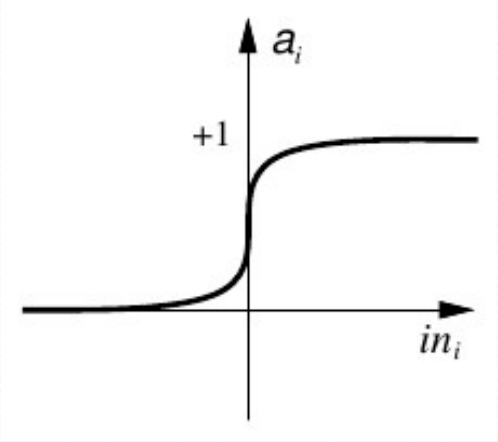
  $sign(x)=1 \quad if\ x>=0$

  $sign(x)=-1 \quad if\ x<0$



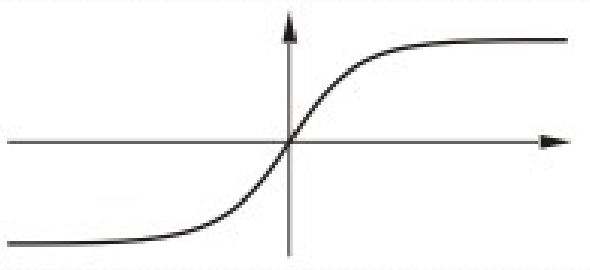
Activation Functions

- ***Sigmoid (Logistic)***
  $sigmoid(x)=1/(1+e^{-bx})$



- ***Hyperbolic tangent***

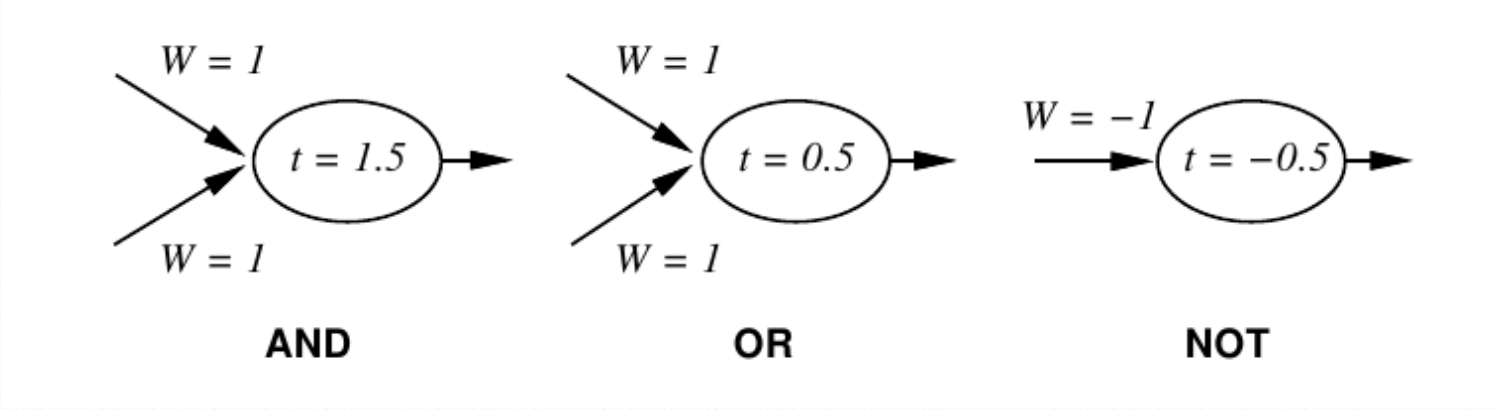
$$htan(x)= \tanh\left(\frac{x}{2}\right)=\frac{1-e^{-x}}{1+e^{-x}}$$



- *Fuzzy membership functions!*

Perceptron: Encoding Simple Boolean Functions

- Step function
- Input in [0,1]
- Output in [0,1]

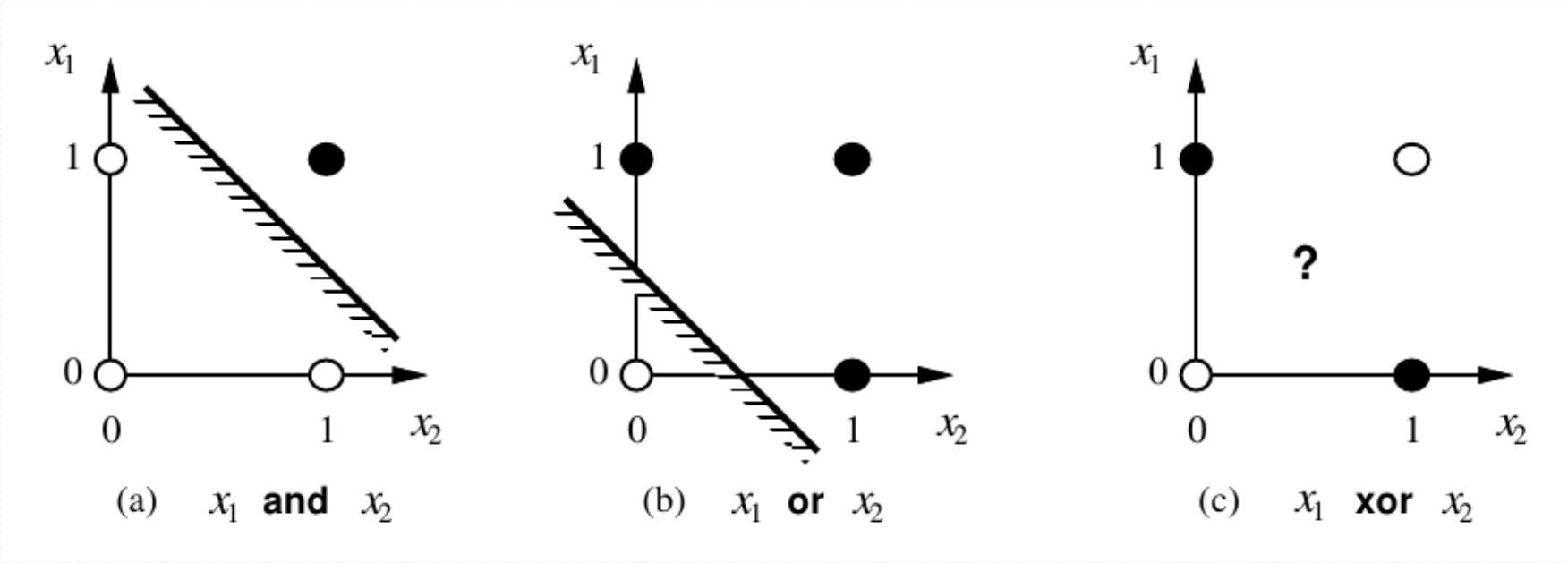


Perceptron: Linear Separable

- step function:

  1 if $-W_0 + W_1 x_1 + \ldots + W_n x_n \geq 0$

  0 otherwise

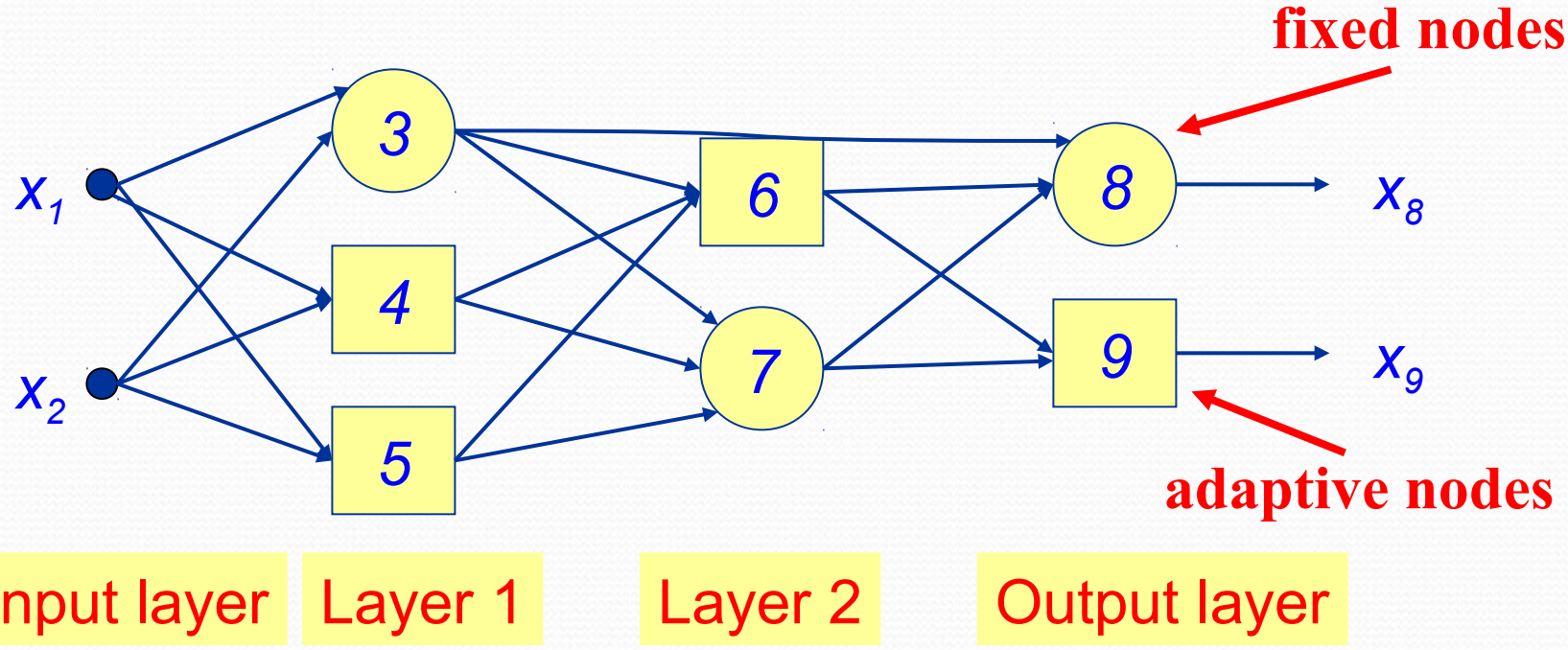


Perceptron: Learning

- Update rule: $W_j \longleftarrow W_j + \alpha \cdot x_j \cdot \text{Error}$ where
  - Error = target – predicted
  - α = learning rate

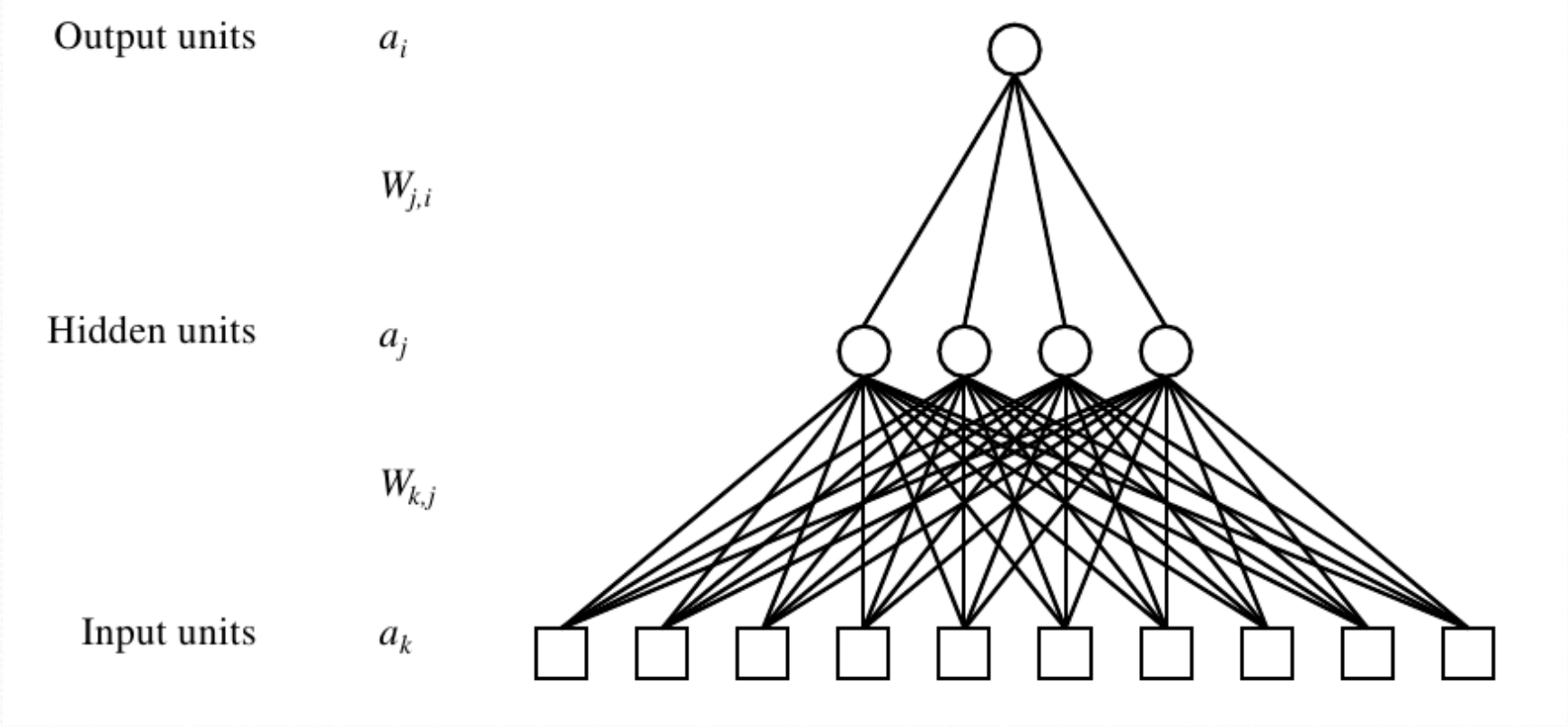
	$x_0 x_1 x_2$	W_0	W_1		predicted	target	Error
1	-1 1 1	-0.160	-0.606	-0.217	0.000	1.000	1.000
2	-1 1 0	-0.260	-0.506	-0.117	0.000	0.000	0.000
3	-1 0 1	-0.260	-0.506	-0.117	1.000	0.000	-1.000
4	-1 1 0	-0.160	-0.506	-0.217	0.000	0.000	0.000
5	-1 1 0	-0.160	-0.506	-0.217	0.000	0.000	0.000
6	-1 1 1	-0.160	-0.506	-0.217	0.000	1.000	1.000
7	-1 0 0	-0.260	-0.406	-0.117	1.000	0.000	-1.000
8	-1 0 0	-0.160	-0.406	-0.117	1.000	0.000	-1.000
...							
70	-1 1 0	0.140	0.194	0.183	1.000	0.000	-1.000
71	-1 1 1	0.240	0.094	0.183	1.000	1.000	0.000
...							

Feed-forward network



- If one layer in network: perceptron (no hidden units)
- Otherwise: multi-layer

Backpropagation



Backpropagation

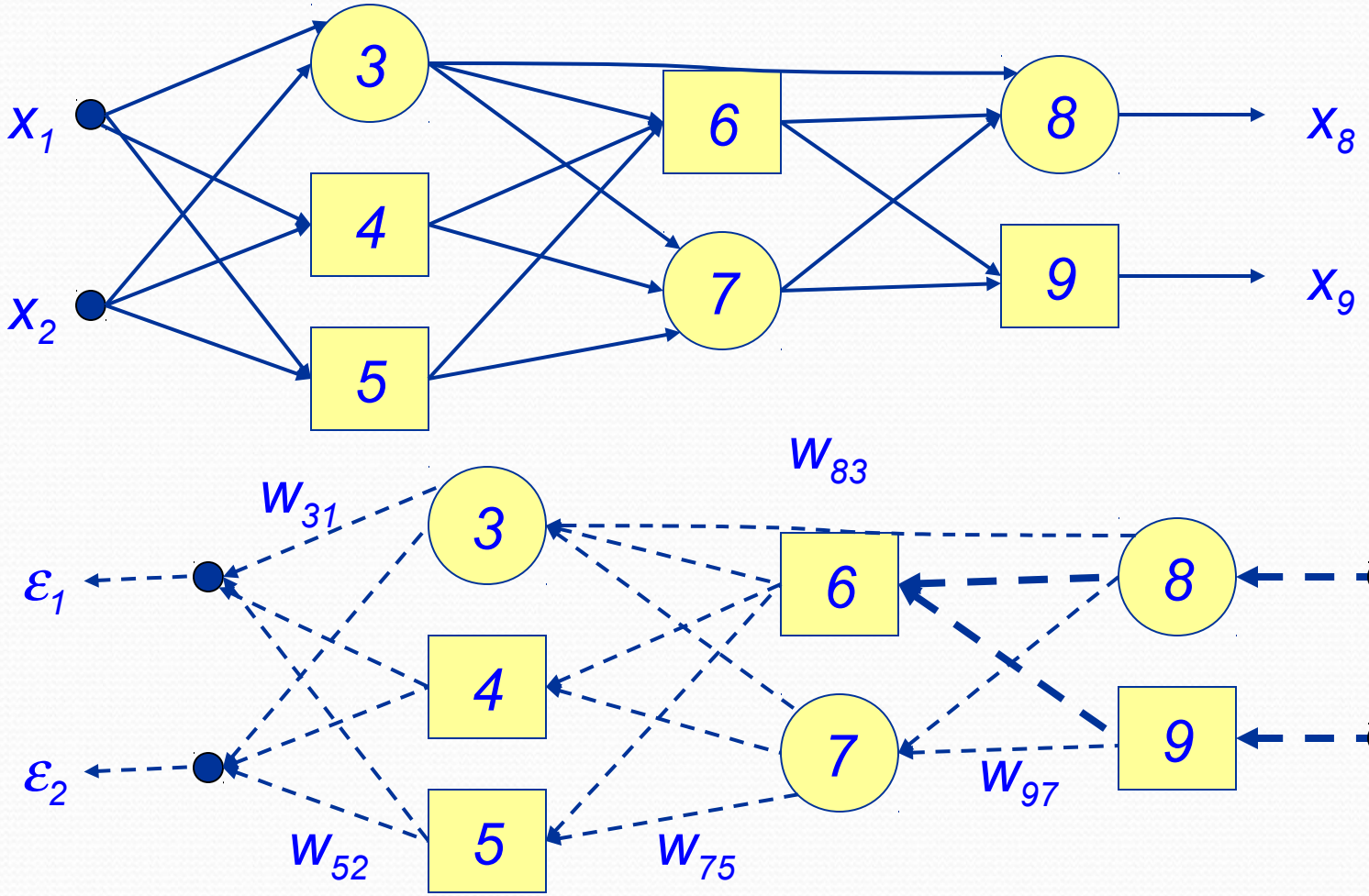
- Update rule output layer:
  $W_{j,i} \leftarrow W_{j,i} + \underbrace{\alpha \cdot a_j \cdot \Delta_i}_{\Delta W_{j,i}}$ with $\Delta_i = \text{Error}_i \cdot g'(\text{in}_i)$
- Update rule hidden layer:
  $W_{k,j} \leftarrow W_{k,j} + \underbrace{\alpha \cdot a_k \cdot \Delta_j}_{\Delta W_{k,j}}$ with $\Delta_j = g'(\text{in}_j) \sum_i W_{j,i} \Delta_i$
- α learning rate

  a_k activation input node

  in_i weighted input *i*th output node

  in_j weighted input *j*th hidden node

Error propagation network



Backpropagation Derivation

- Gradient descent
- Error is $E = \frac{1}{2}\sum_i (y_i - a_i)^2$ for all output nodes
- $$\begin{aligned} \frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i) \cdot \frac{\partial a_i}{\partial W_{j,i}} \\ &= -(y_i - a_i) \cdot \frac{\partial g(\sum_j W_{j,i} a_j)}{\partial W_{j,i}} \\ &= -(y_i - a_i) \cdot g'(in_i) \cdot a_j \\ &= -a_j \cdot \Delta_i \end{aligned}$$

$$\Delta_i = \text{Error}_i \cdot g'(\text{in}_i)$$

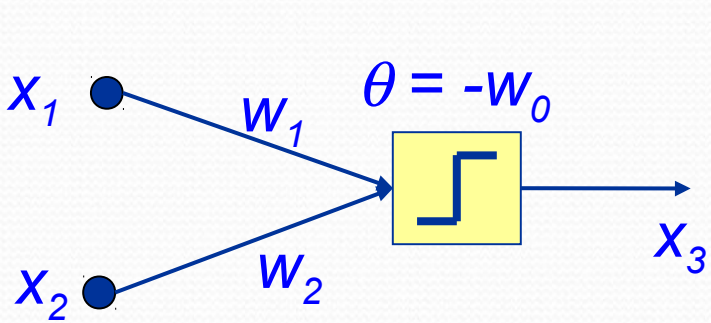
$$\Delta_i = \text{Error}_i \cdot g'(\text{in}_i)$$

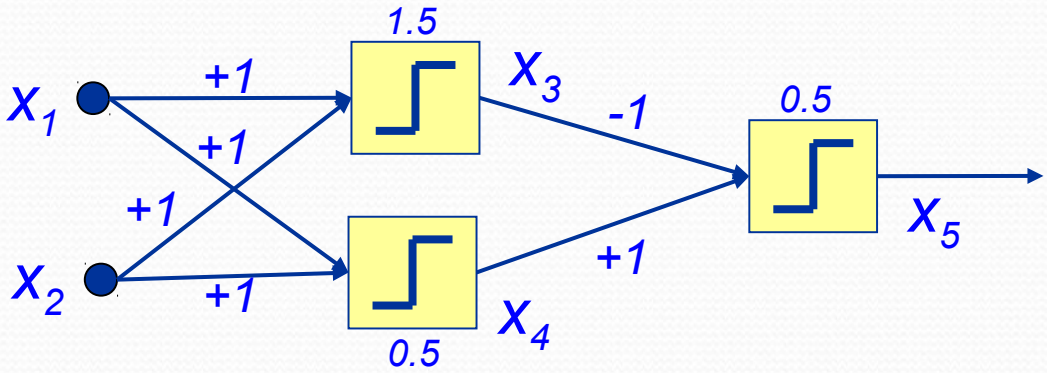
Backpropagation Derivation

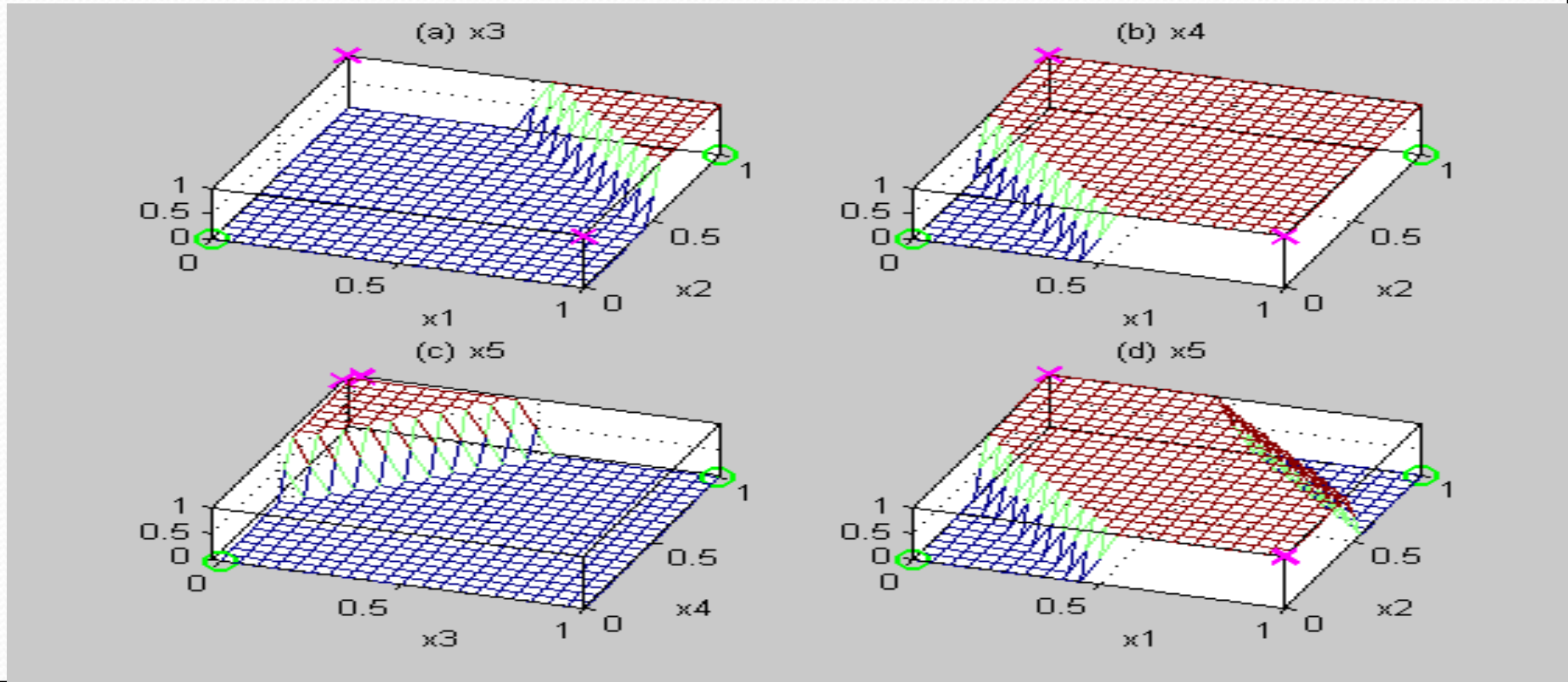
- $$\begin{aligned}
\frac{\partial E}{\partial W_{k,j}} &= -\sum_i (y_i - a_i) \cdot \frac{\partial a_i}{\partial W_{k,j}} \\
&= -\sum_i (y_i - a_i) \cdot \frac{\partial g\left(\sum_j W_{j,i} a_j\right)}{\partial W_{k,j}} \\
&= -\sum_i (y_i - a_i) \cdot g'(in_i) \cdot \frac{\partial \left(\sum_j W_{j,i} a_j\right)}{\partial W_{k,j}} \\
&= -\sum_i \Delta_i \cdot W_{j,i} \cdot \frac{\partial a_j}{\partial W_{k,j}} \\
&= -\sum_i \Delta_i \cdot W_{j,i} \cdot \frac{\partial g(in_j)}{\partial W_{k,j}} \\
&= -\sum_i \Delta_i \cdot W_{j,i} \cdot g'(in_j) \cdot \frac{\partial \left(\sum_k W_{k,j} a_k\right)}{\partial W_{k,j}} \\
&= -\sum_i \Delta_i \cdot W_{j,i} \cdot g'(in_j) \cdot a_k \\
&= -a_k \Delta_j
\end{aligned}$$

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

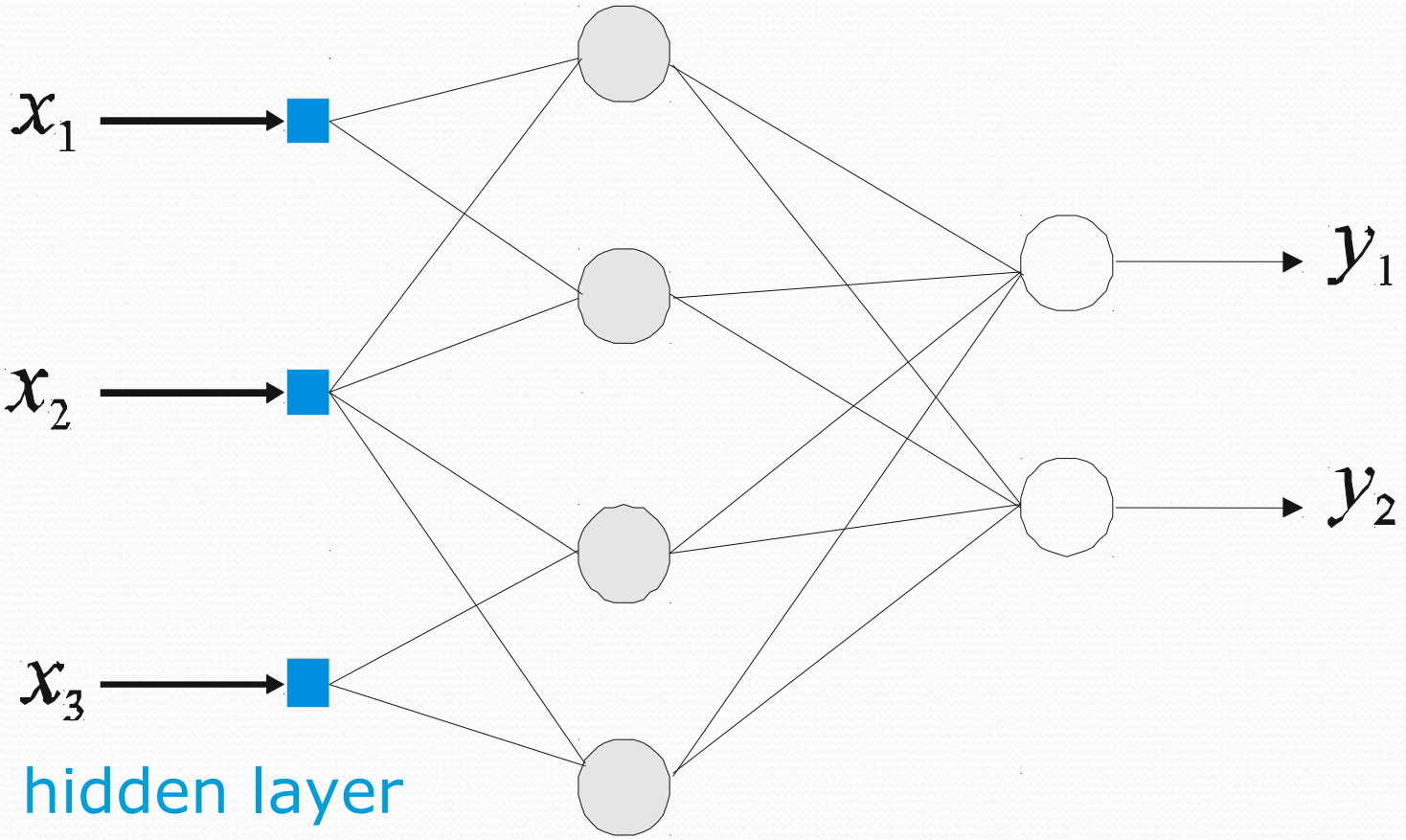
Multilayer perceptron







Common MLP architecture



- One hidden layer
- Hidden neurons: tanh or logistic activation
- Output neurons: linear activation

Online vs Offline learning

- online: update after each example p

$$W_{j,i} \leftarrow W_{j,i} - \alpha \frac{\partial E^p}{\partial W_{j,i}}$$

- offline: update after seeing all examples

$$W_{j,i} \leftarrow W_{j,i} - \alpha \sum_p \frac{\partial E^p}{\partial W_{j,i}}$$

Speeding up backpropagation

- Momentum term

$$\Delta W_{j,i}^{t} = \alpha \cdot a_j \cdot \Delta_i + \eta \Delta W_{j,i}^{t-1} \text{ (Output layer)}$$

$$\Delta W_{j,i}^{t} = -\alpha \cdot \frac{\partial E}{\partial W_{j,i}} + \eta \Delta W_{j,i}^{t-1} \text{ (General case)}$$

- Regularized initialization

- Learning rate rescaling
  learning rate of layers decreases towards the output layer

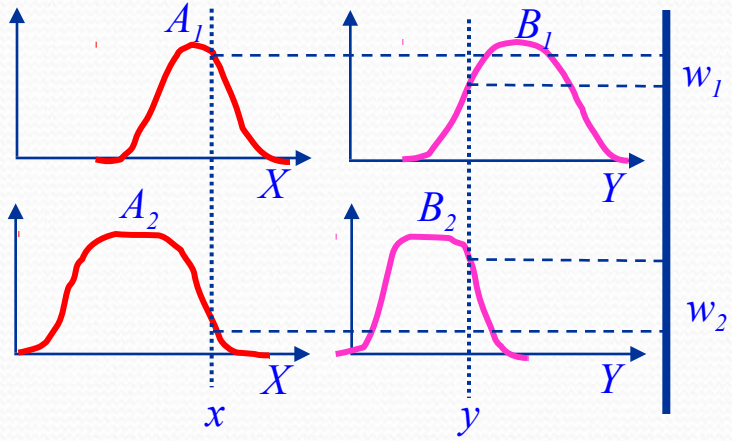
Approximation power

- General function approximators

- Feed-forward neural network with one hidden layer and sigmoidal activation functions can approximate any continuous function arbitrarily well (Cybenko)

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

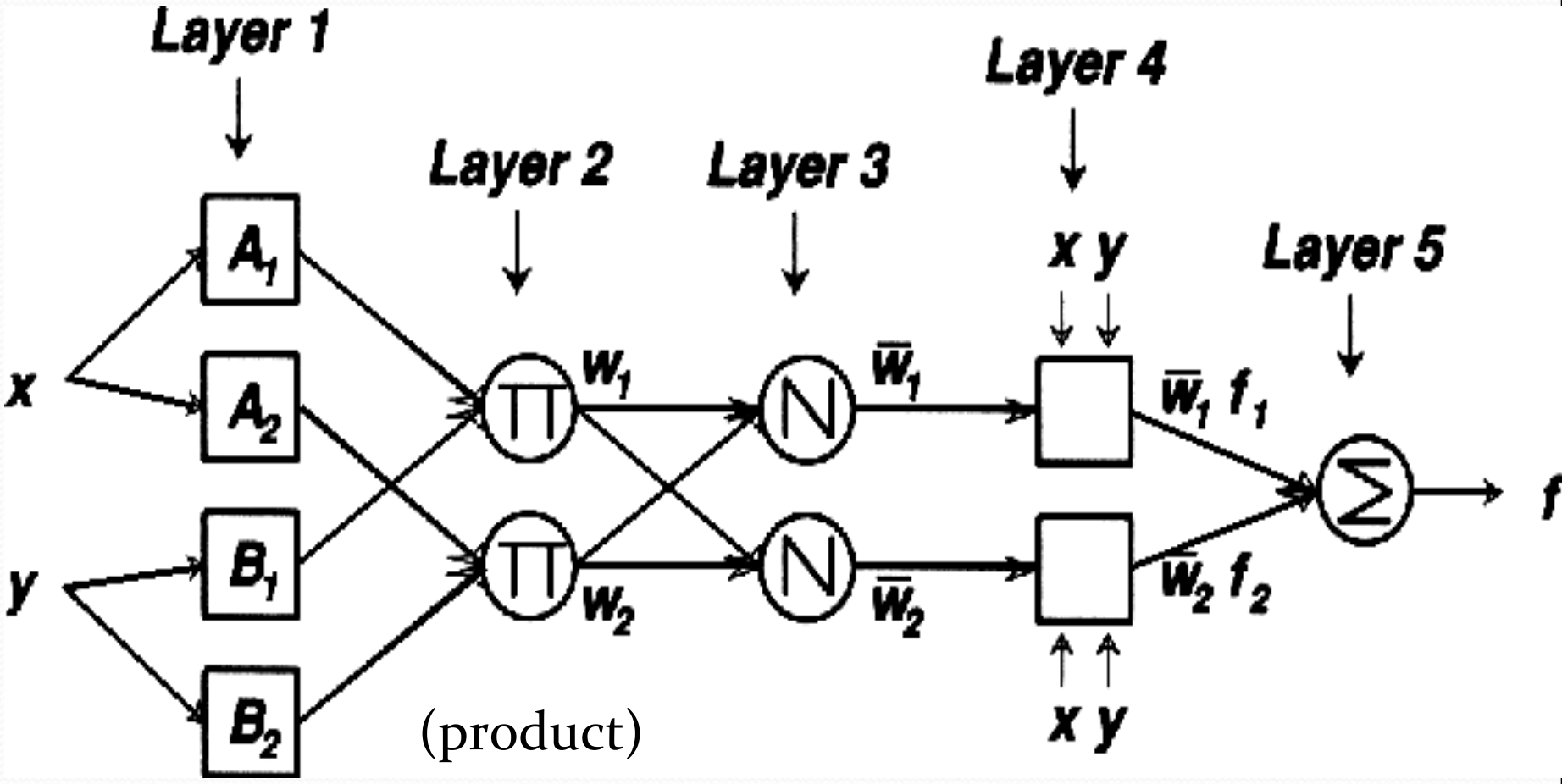


$$f_1 = p_1 x + q_1 y + r_1$$

$$f_2 = p_2 x + q_2 y + r_2$$

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

ANFIS architecture



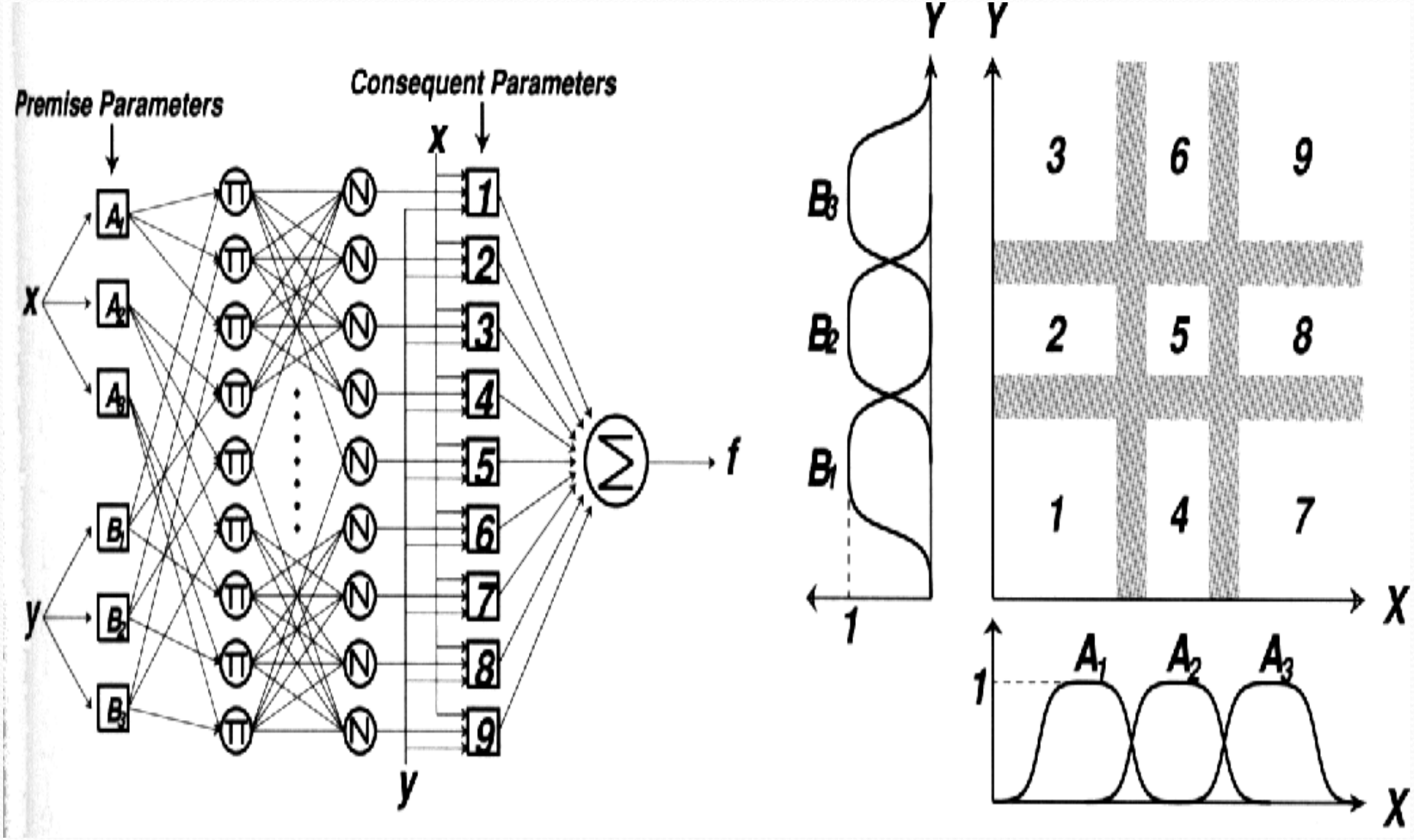
(product)

(normalization)

ANFIS layers

- Layer 1: every node is adaptive with function
$$O_{1,i} = \mu_i(x_i)$$
- Layer 2: every node is fixed and computes a T-norm of the inputs $O_{2,i} = \bigwedge_j O_{1,j}$
- Layer 3: every node is fixed with function
$$O_{3,i} = O_{2,i} / \sum_j O_{2,j}$$
- Layer 4: every node is adaptive with function
$$O_{4,i} = O_{3i}(p_0 + p_1 x_1 + \cdots + p_n x_n)$$
- Layer 5: single node sums up inputs $O_{5,i} = \sum_j O_{4,j}$

ANFIS with multiple rules



Hybrid learning for ANFIS

- Consider an ANFIS with two inputs x & y
- Let there be 2 nodes in layers 2-3-4
- Denote the consequent parameters by p, q, r
- ANFIS output is now given by

$$f = \frac{w_1}{w_1+w_2} f_1 + \frac{w_2}{w_1+w_2} f_2$$

$$= \overline{w}_1 (p_1 x + q_1 y + r_1) + \overline{w}_2 (p_2 x + q_2 y + r_2)$$

$$= (\overline{w}_1 x) p_1 + (\overline{w}_1 y) q_1 + (\overline{w}_1) r_1 + (\overline{w}_2 x) p_2 + (\overline{w}_2 y) q_2 + (\overline{w}_2) r_2$$

- Partition S as follows:
  S_1: premise parameters (nonlinear)
  S_2: consequent parameters (linear)

Hybrid learning for ANFIS

- How to learn the parameters of a linear model

$$f(\vec{x}, \vec{\beta}) = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

such that $E = \frac{1}{2} \sum_i (f(\vec{x}_i, \vec{\beta}) - y_i)^2$ is minimal?

(where $\vec{x}_i$ and y_i represent input and output for training examples)

→ least squares linear regression

Linear Regression

- Error function:

$$E = \frac{1}{2}\sum_i (\vec{\beta}^T \vec{x}_i - y_i)^2$$

- Compute global minimum by means of derivative:

$$\nabla E = \sum_i (\vec{\beta}^T \cdot \vec{x}_i - y_i)\vec{x}_i^T$$

$$0 = \vec{\beta}^T \cdot \left(\sum_i \vec{x}_i \cdot \vec{x}_i^T\right) - \sum_i y_i \cdot \vec{x}_i^T$$

$$\vec{\beta} = \left(\sum_i \vec{x}_i \cdot \vec{x}_i^T\right)^{-1} \sum_i y_i \cdot \vec{x}_i^T = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y}$$

Stone-Weierstrass theorem

Let D be a compact space of N dimensions and let F be a set of continuous real-valued functions on D satisfying the following

- Identity function: $f(x)=1$ is in F
- Separability: for any two points $x_1 \neq x_2$ in D, there is an f in F s.t. $f(x_1) \neq f(x_2)$
- Algebraic closure: if f and g are two functions in F, then fg and $af+bg$ are also in F for real a and b

➔ F is *dense* in the set of continuous functions $C(D)$ on D:

$$(\forall \varepsilon > 0)(\forall g \in C(D))(\exists f \in F)(\forall x \in D : |g(x) - f(x)| < \varepsilon)$$

Universal approximation in ANFIS

- According to Stone-Weierstrass theorem, an ANFIS can approximate any continuous nonlinear function arbitrarily well

- Restriction to rules with **a constant** in the consequent and **Gaussian** membership functions

- Identity: $f(x)=1$ is in F
  Obtained by having a constant consequent

- Separability: for any two points $x_1 \neq x_2$ in D, there is f in F s.t. $f(x_1) \neq f(x_2)$
  Obtained by selecting different parameters in the network

Algebraic closure

Consider two systems with two rules

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

- Additive

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

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

- Multiplicative

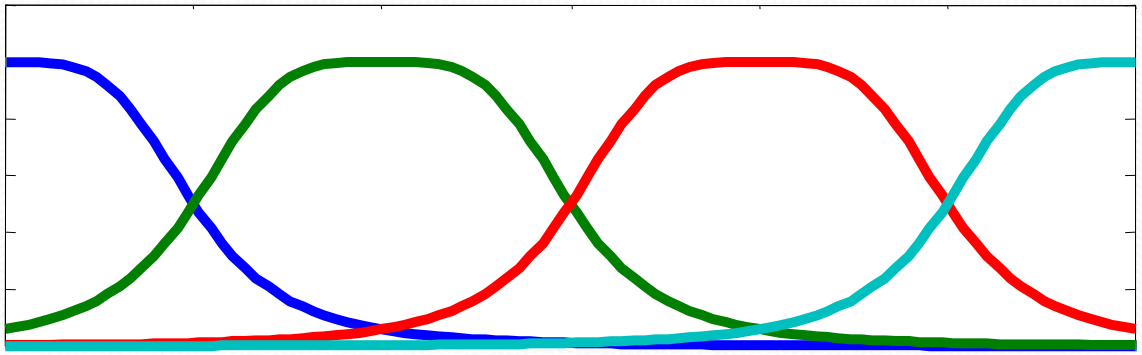
$$z\hat{z} = \left(\frac{w_1 f_1 + w_2 f_2}{w_1 + w_2}\right)\left(\frac{\hat{w}_1 \hat{f}_1 + \hat{w}_2 \hat{f}_2}{\hat{w}_1 + \hat{w}_2}\right)$$

Use Gaussian membership functions!

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

Model building guidelines

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



- Using an adaptive step size can speed up training

What to do?

- Number of inputs
- What inputs
- Number of outputs
- What outputs
- Number of rules
- Type of consequent functions
- Normalize inputs
- Collect data
- Define objective function
- Determine initial rules
- Initialize network
- Define stop conditions

TRAIN

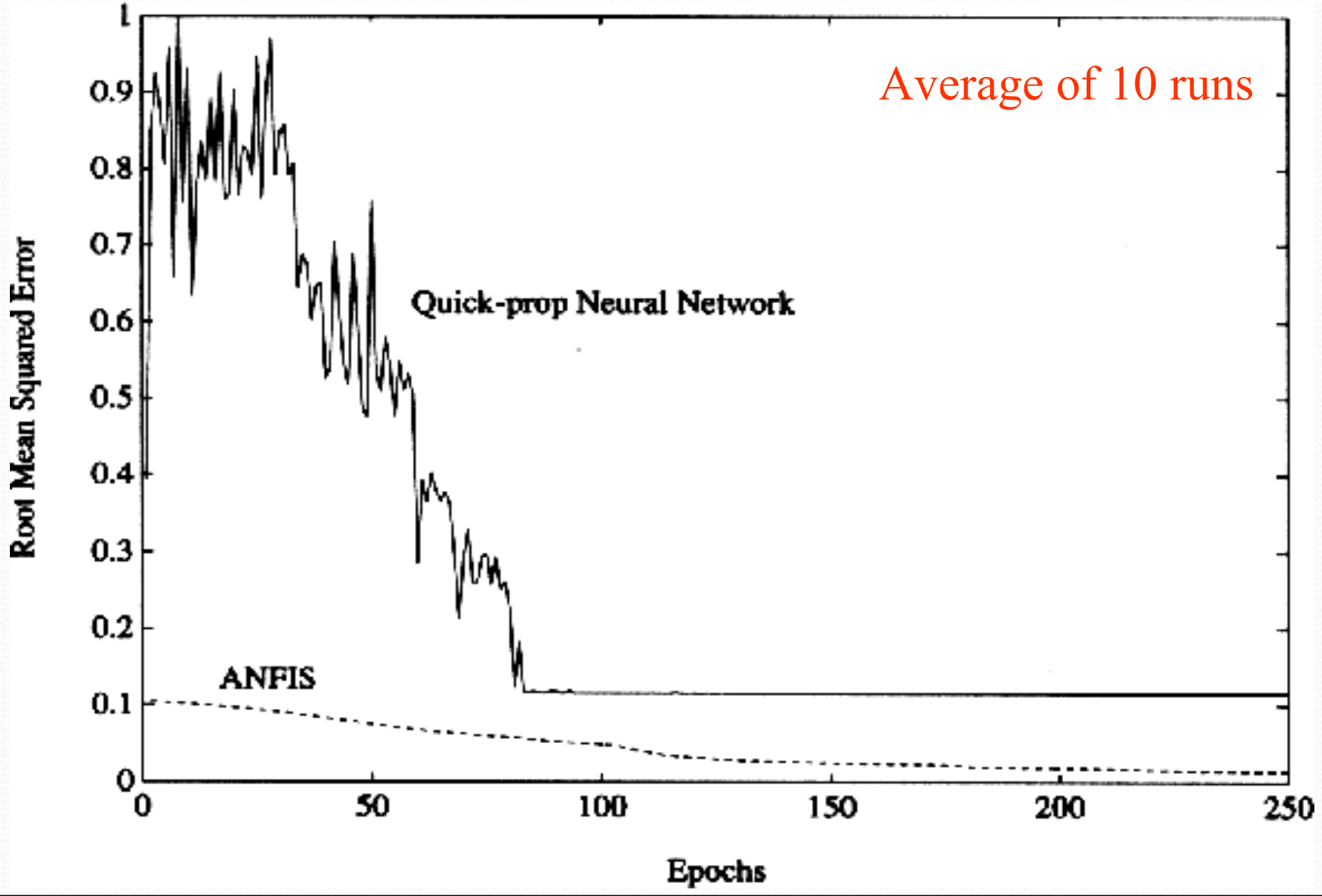
Two-input sinc function

- Input range: [-10,10]x[-10,10]
  121 data points

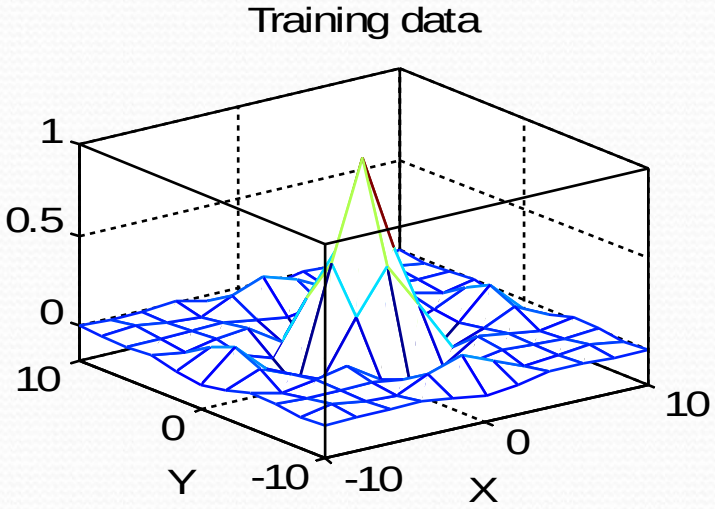
$$z = \mathrm{sinc}(x, y) = \frac{\sin(x)\sin(y)}{xy}$$

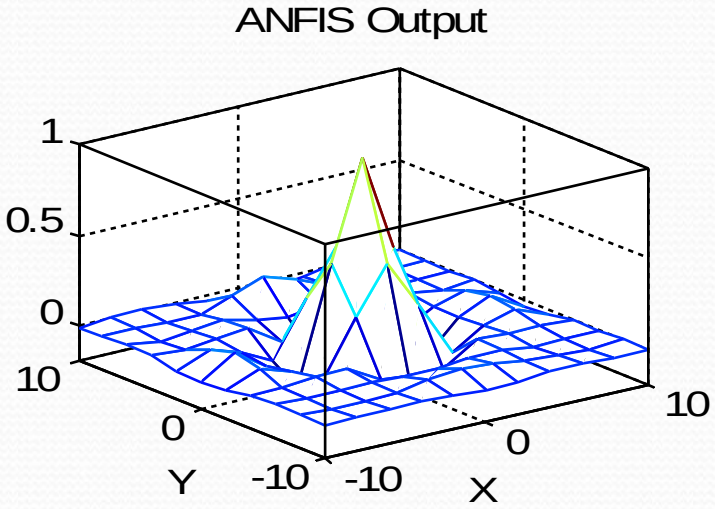
- ANFIS: 16 rules, 4 membership functions per variable, 72 parameters (48 consequence, 24 premise)
- MLP: 18 neurons in hidden layer, 73 parameters, quick propagation
- ANFIS takes longer for same number of epochs

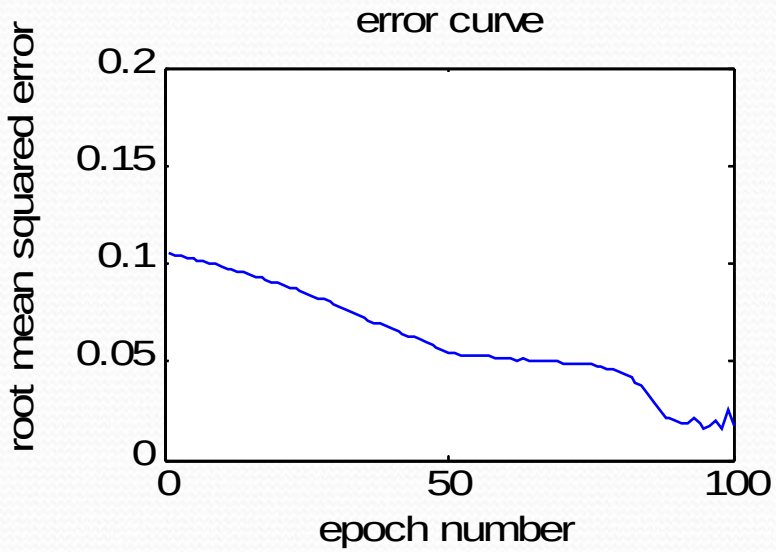
MLP vs. ANFIS results

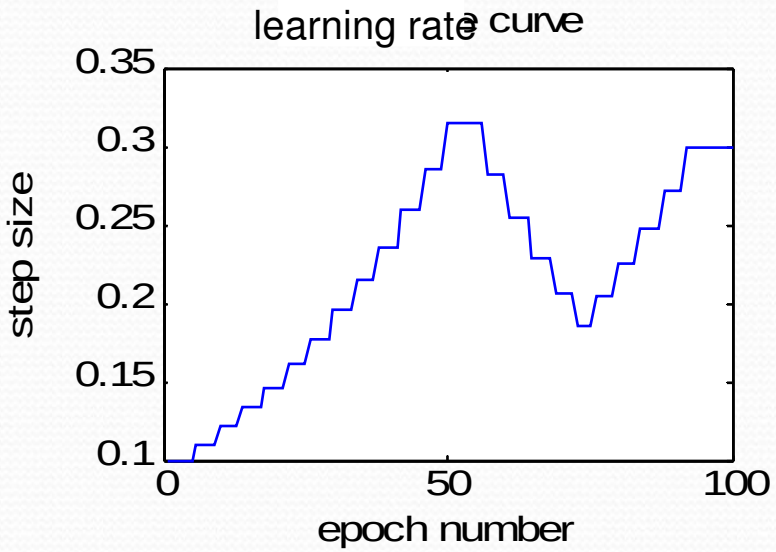


ANFIS output

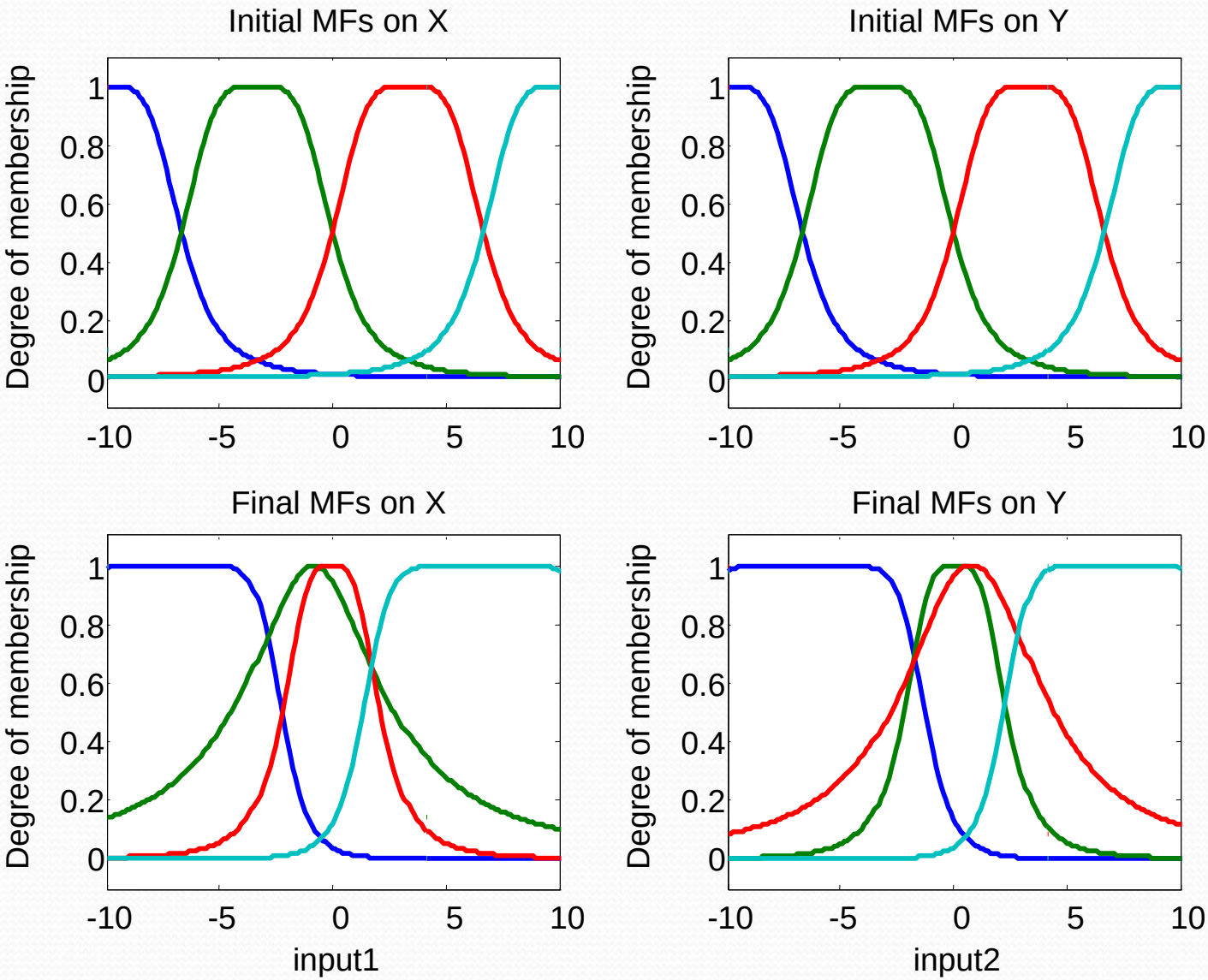








ANFIS model


Initial MFs on X
Initial MFs on Y
Final MFs on X
Final MFs on Y
Degree of membership
1
0.8
0.6
0.4
0.2
0
-10
-5
0
5
10
input1
input2

Modeling dynamic systems

- f(.) has the following form

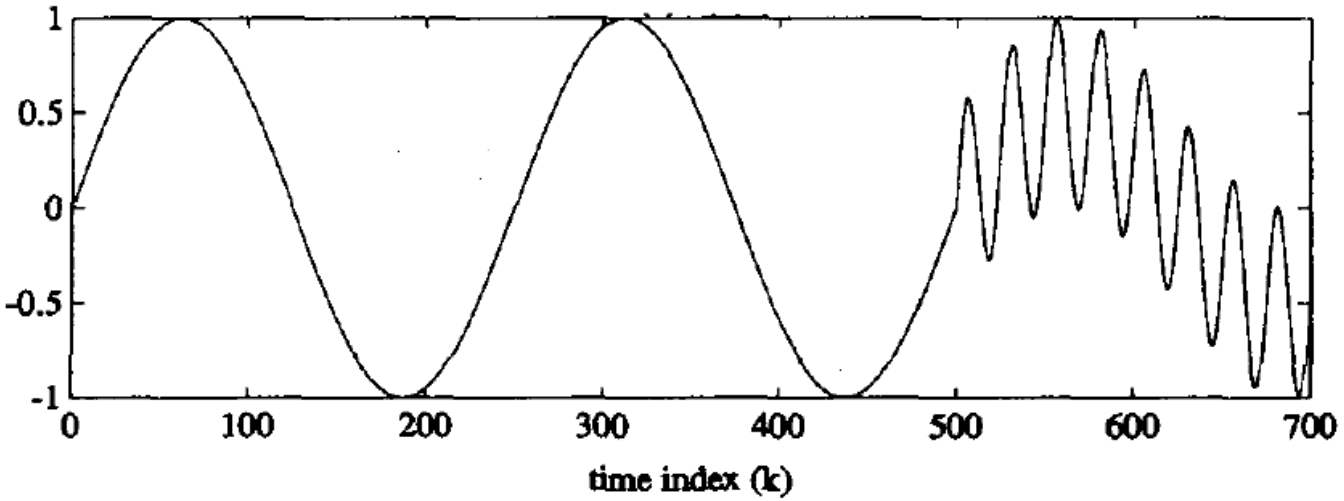
$$f(u) = 0.6\sin(\pi u) + 0.3\sin(3\pi u) + 0.1\sin(5\pi u)$$

where

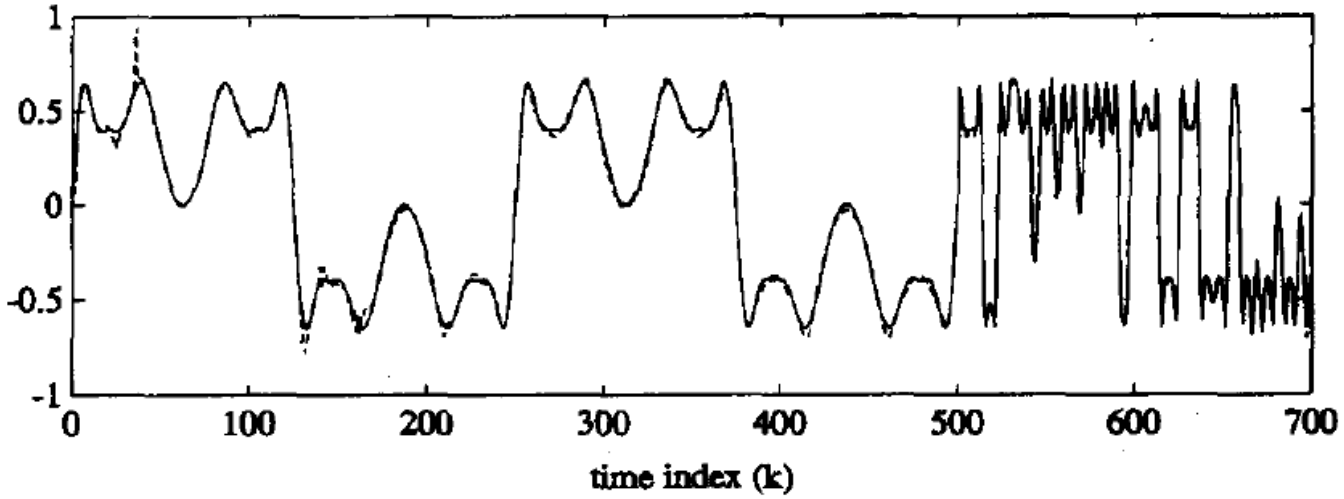
$$u(k) = \sin(2\pi k / 250)$$

- ANFIS parameters updated at each step
- Learning rate: 0.1; forgetting factor: 0.99
- ANFIS can adapt even after the input changes

Plant and model outputs



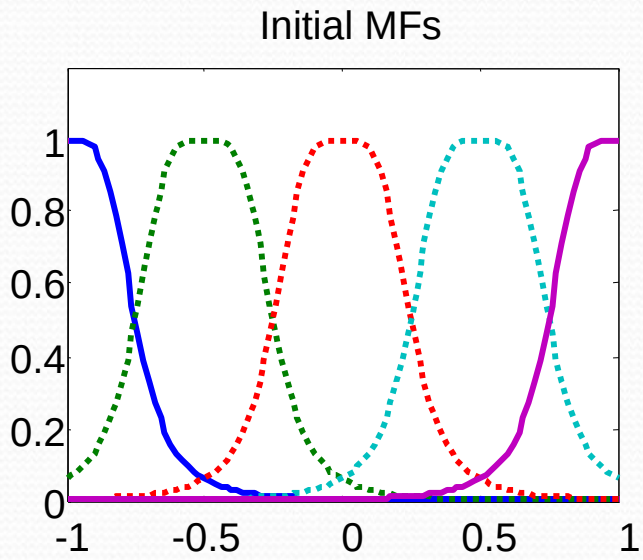
(a)

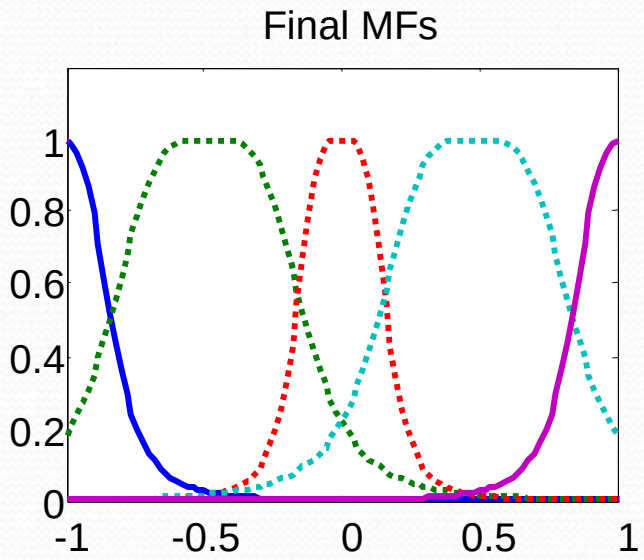


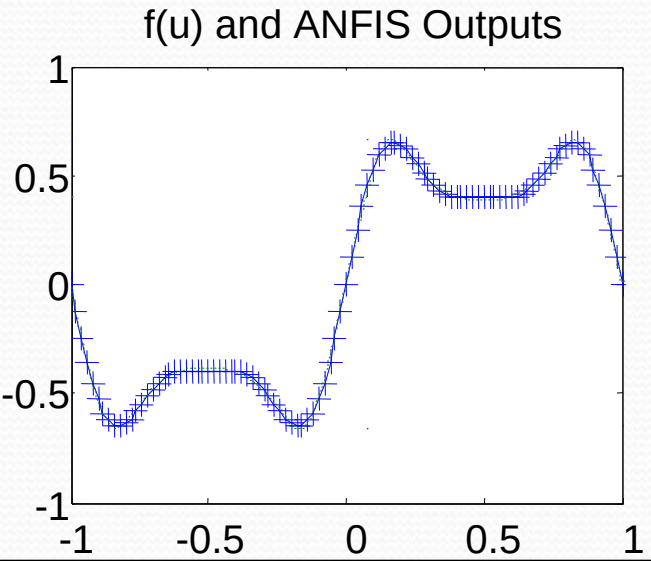
(b)

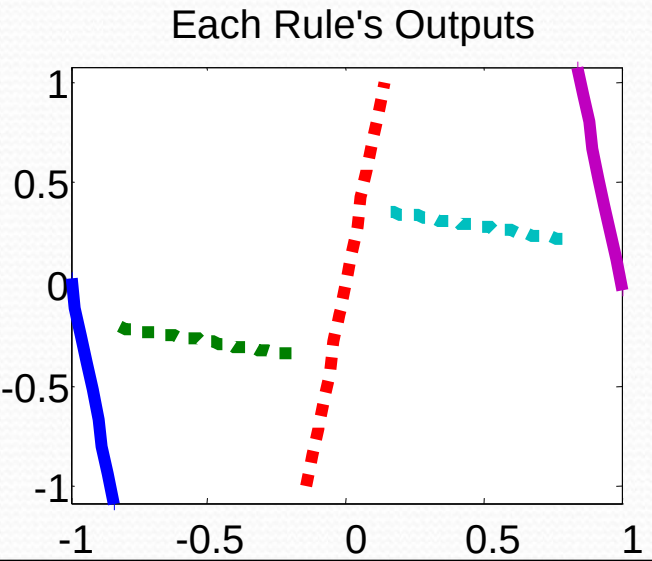
Effect of number of MFs

5 membership functions



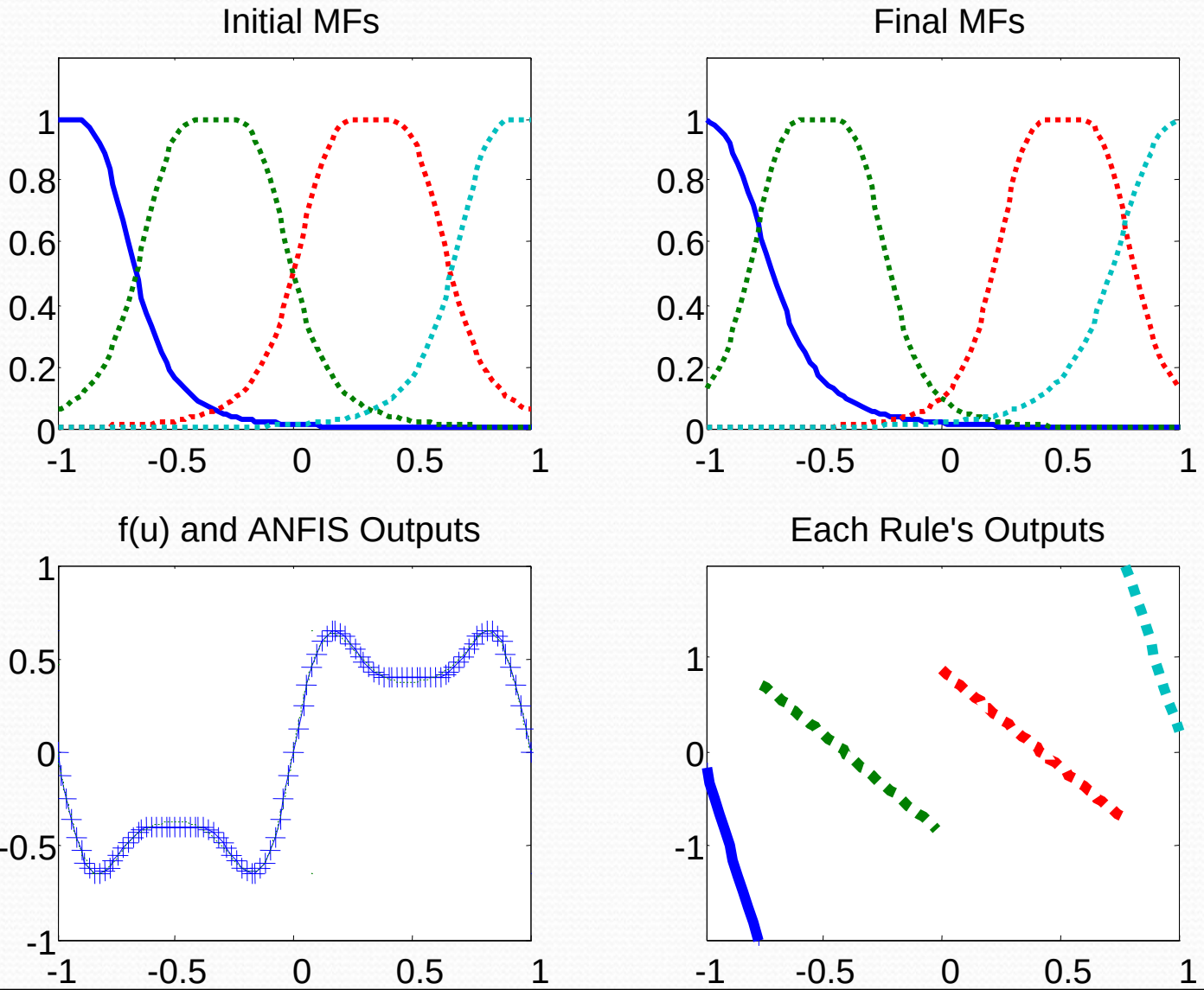






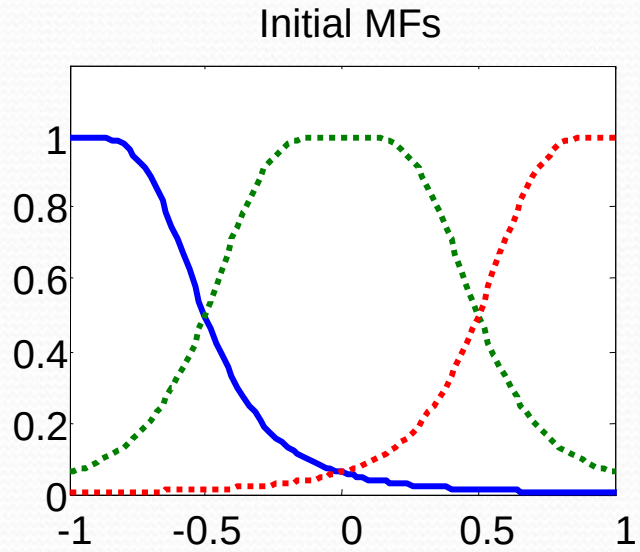
Effect of number of MFs

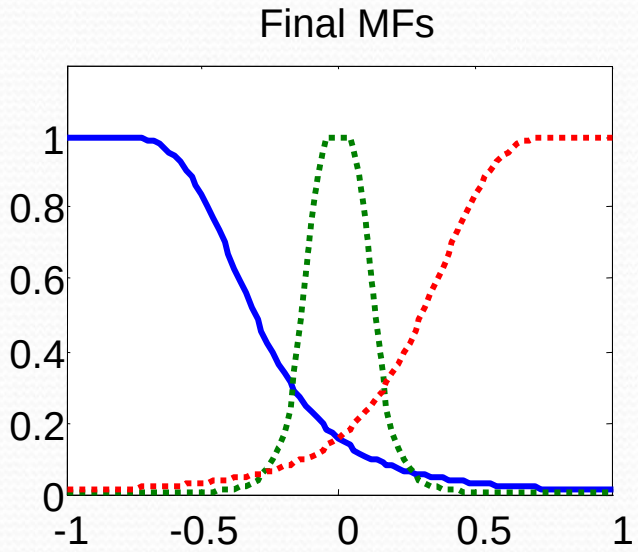
4 membership functions

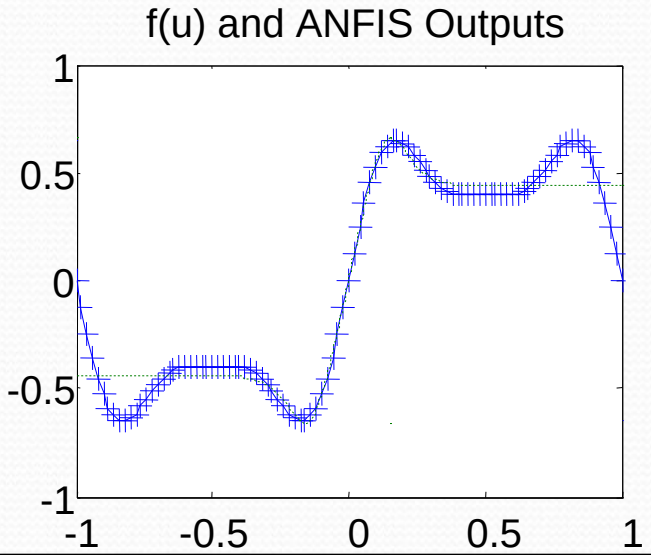


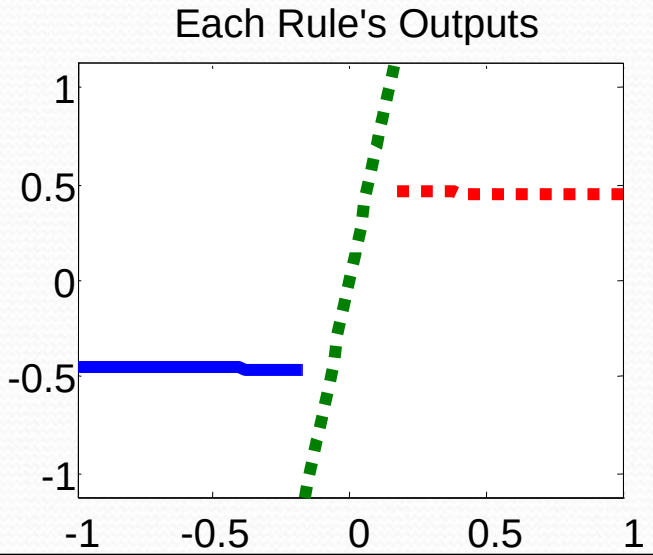
Effect of number of MFs

3 membership functions







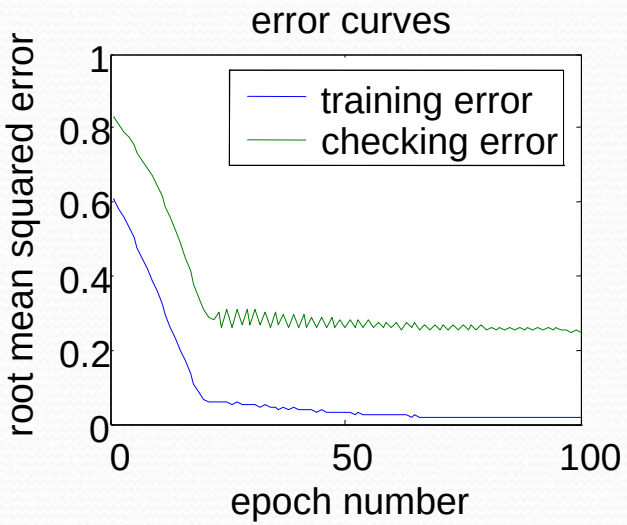


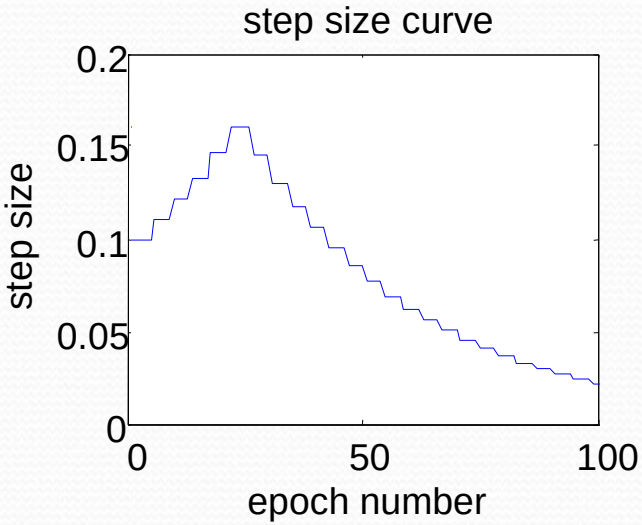
Three-input nonlinear function

- System mapping is given by

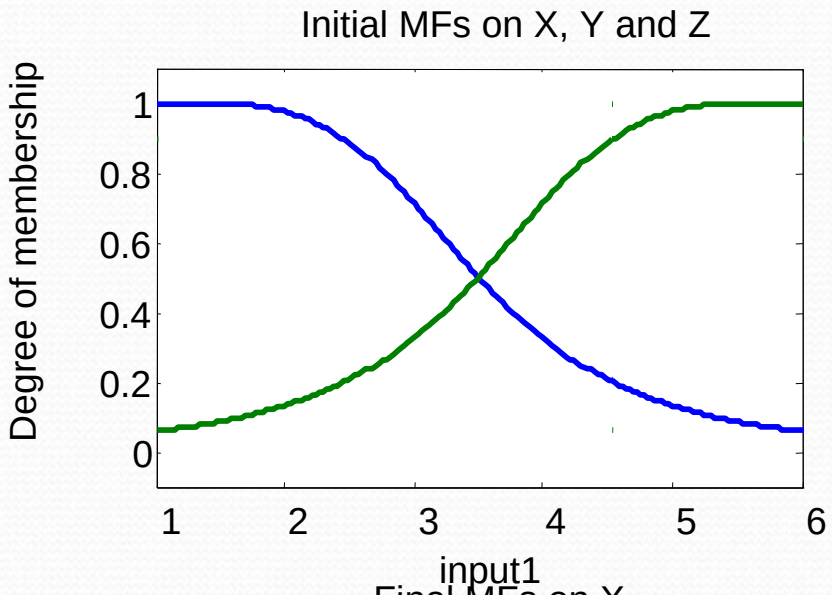
$$o = \left(1 + \sqrt{x} + \frac{1}{y} + \frac{1}{\sqrt{z^3}}\right)^2$$

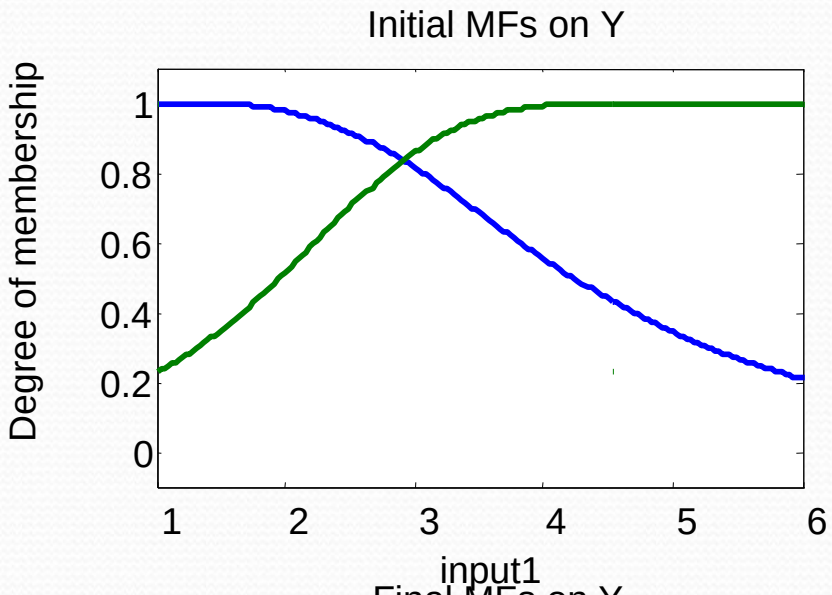
- Two membership functions per variable, 8 rules
- Input ranges: [1,6]x[1,6]x[1,6]
- 215 training data, 125 validation data

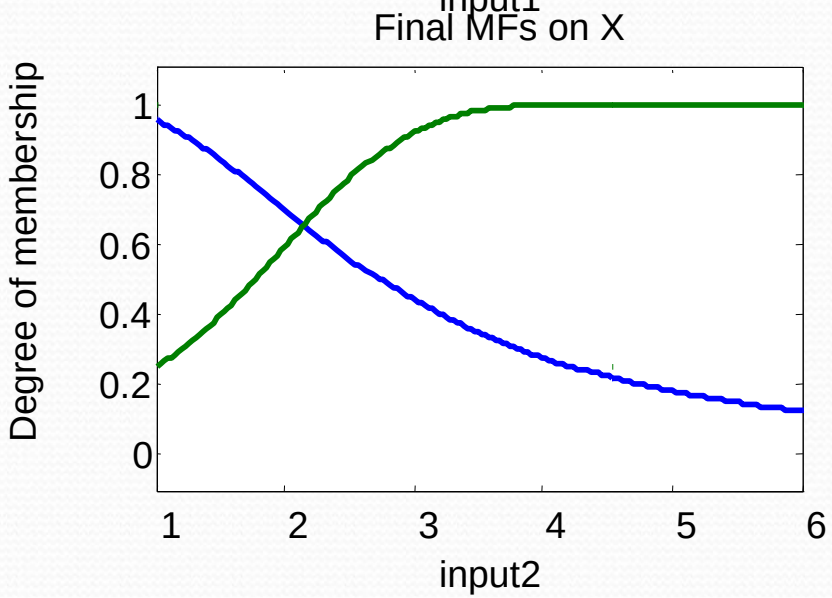


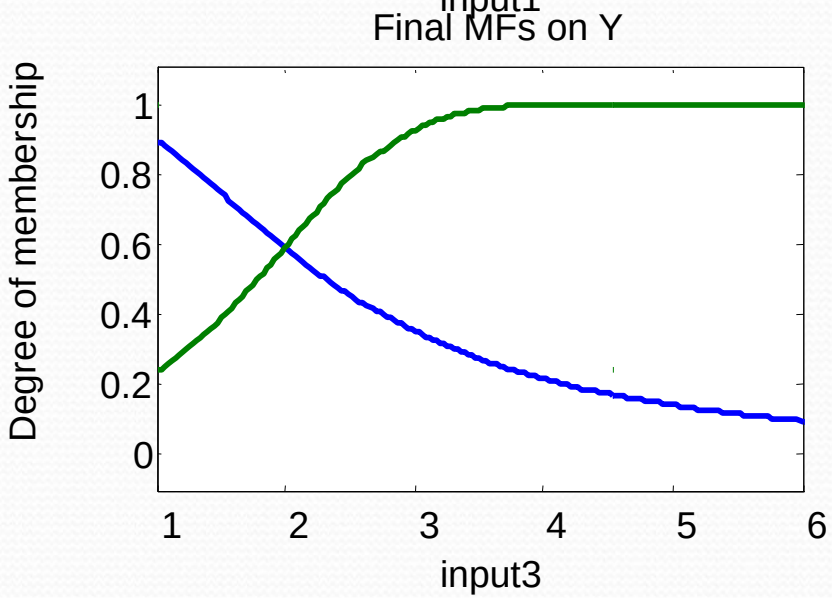


ANFIS model


Initial MFs on X, Y and Z
Degree of membership
1
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6
input1


Initial MFs on Y
Degree of membership
1
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6
input1


Final MFs on X
Degree of membership
1
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6
input2


Final MFs on Y
Degree of membership
1
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6
input3

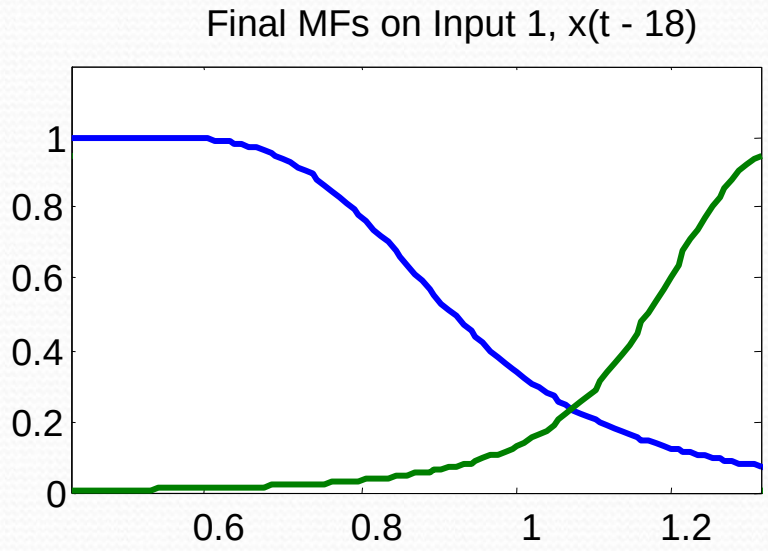
Predicting chaotic time series

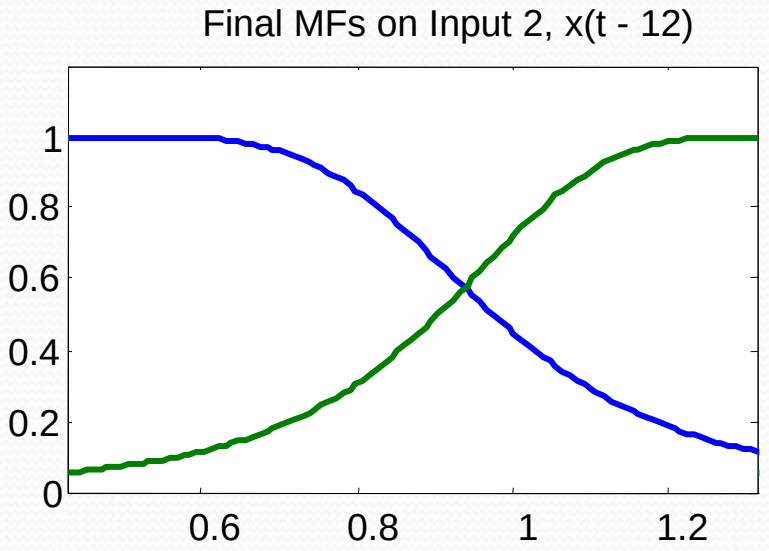
- Consider a chaotic time series generated by

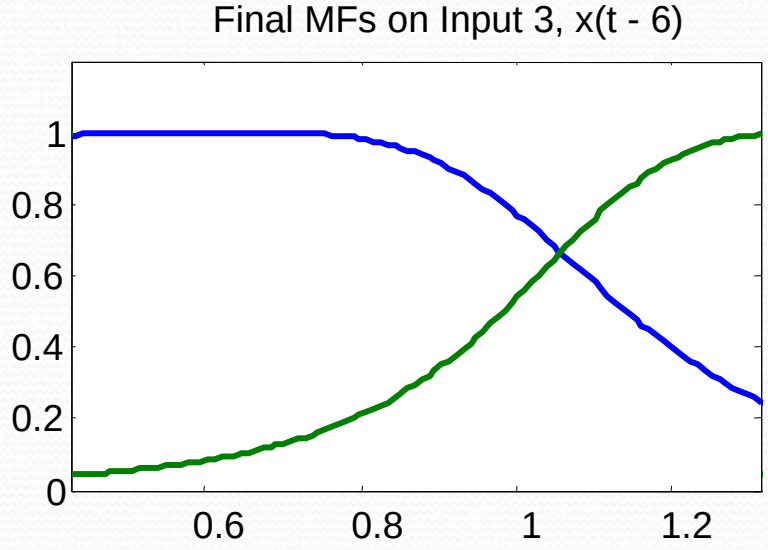
$$dx(t)/dt = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t)$$

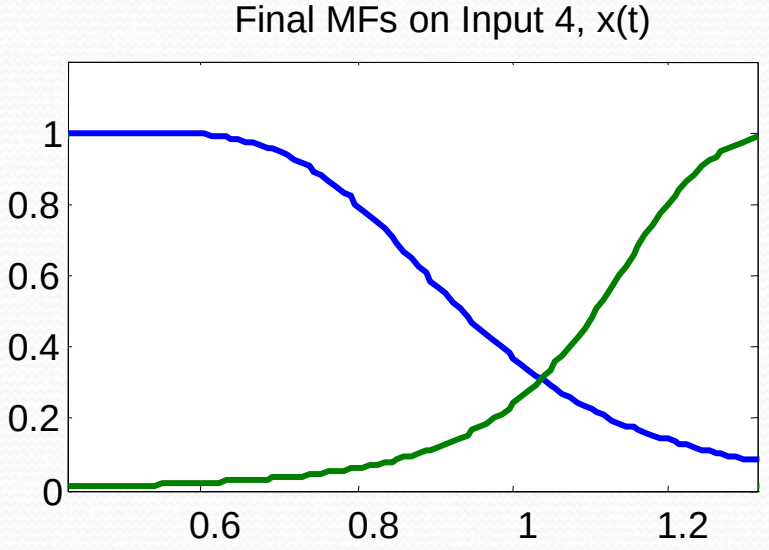
- Task: predict the output of system at some future instance t+P by using past output
- 500 training data, 500 validation data
- ANFIS input: [x(t-18),x(t-12),x(t-6),x(t)]
- ANFIS output: x(t+6)
- Two MFs per variable, 16 rules
- 104 parameters (24 premise, 80 consequent)
- Data generated from t=118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

ANFIS model

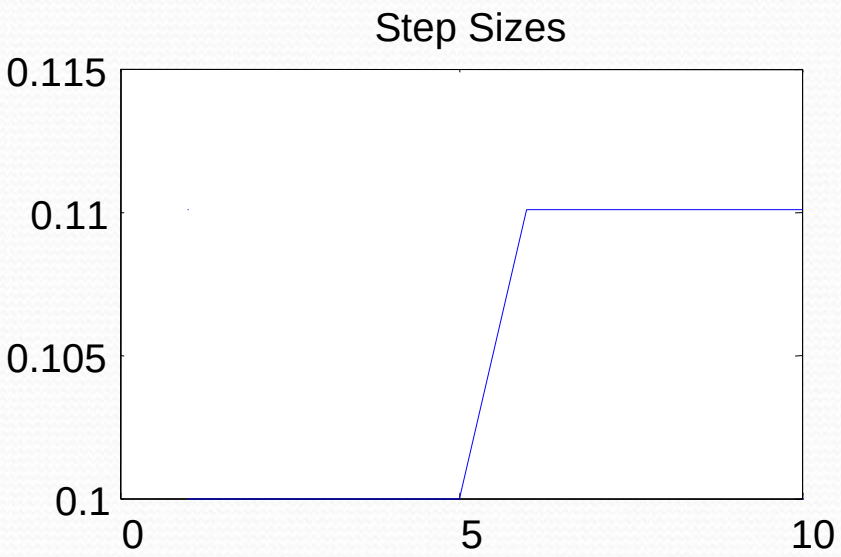

Final MFs on Input 1, x(t - 18)
1
0.8
0.6
0.4
0.2
0
0.6
0.8
1
1.2

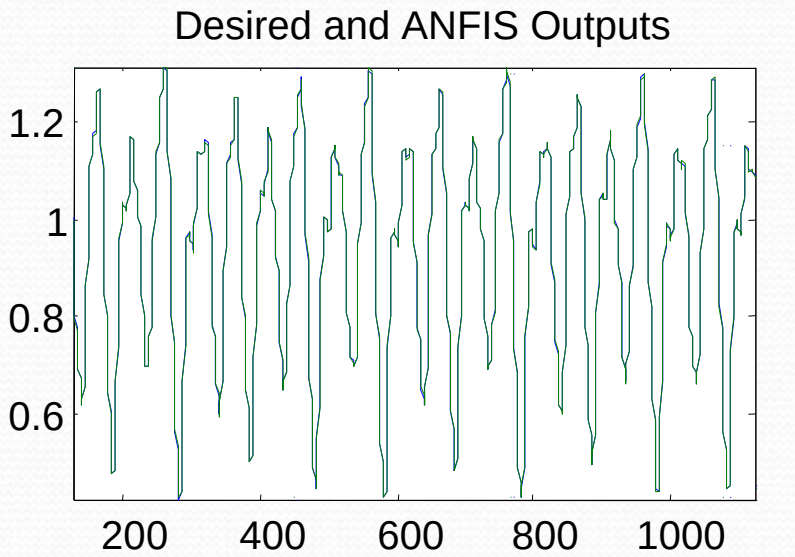

Final MFs on Input 2, x(t - 12)
1
0.8
0.6
0.4
0.2
0
0.6
0.8
1
1.2

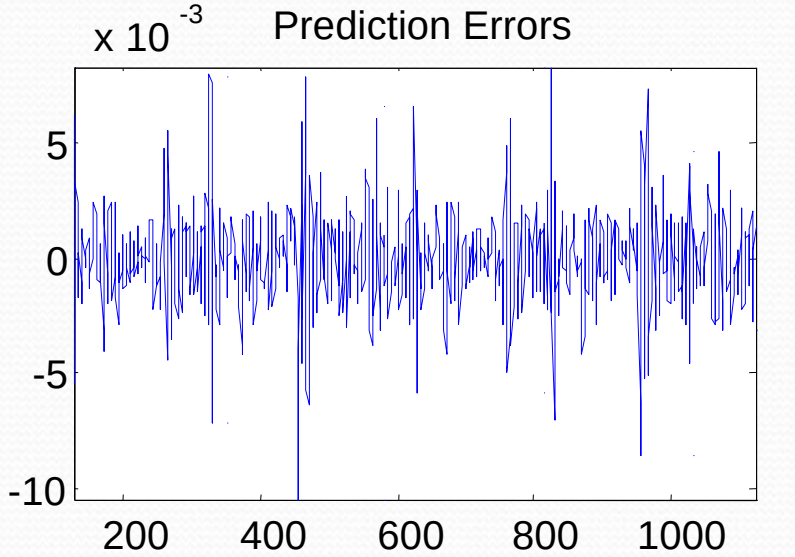

Final MFs on Input 3, x(t - 6)
1
0.8
0.6
0.4
0.2
0
0.6
0.8
1
1.2


Final MFs on Input 4, x(t)
1
0.8
0.6
0.4
0.2
0
0.6
0.8
1
1.2

Model output







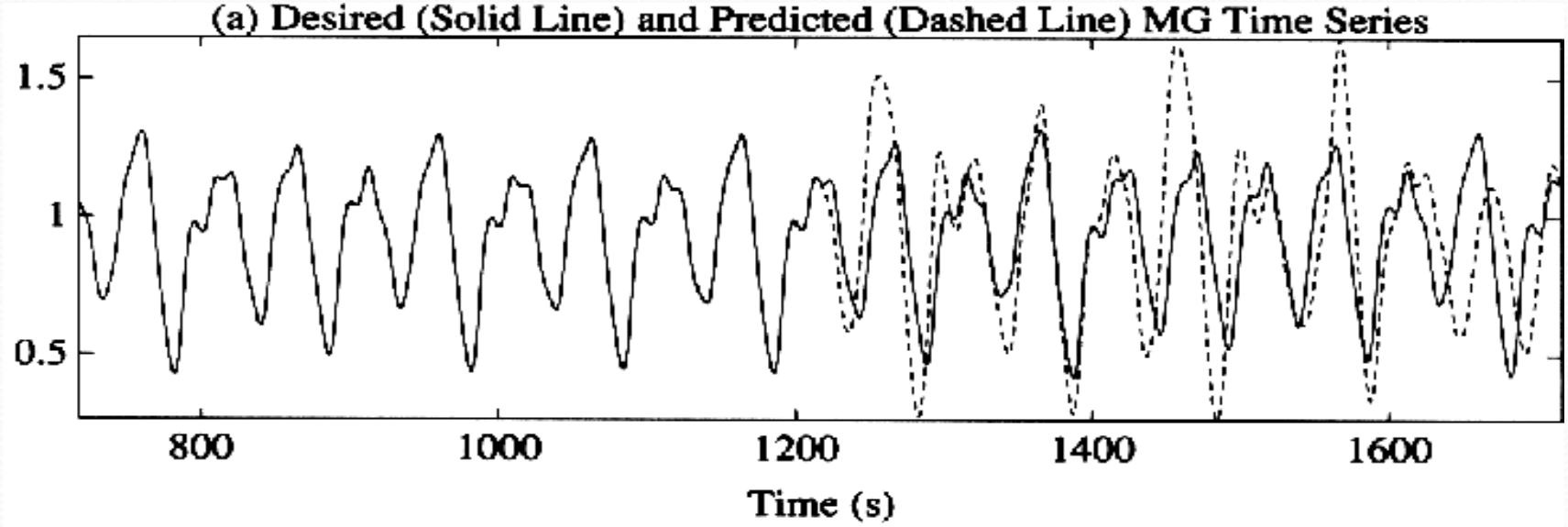
Auto-Regression (AR) Model

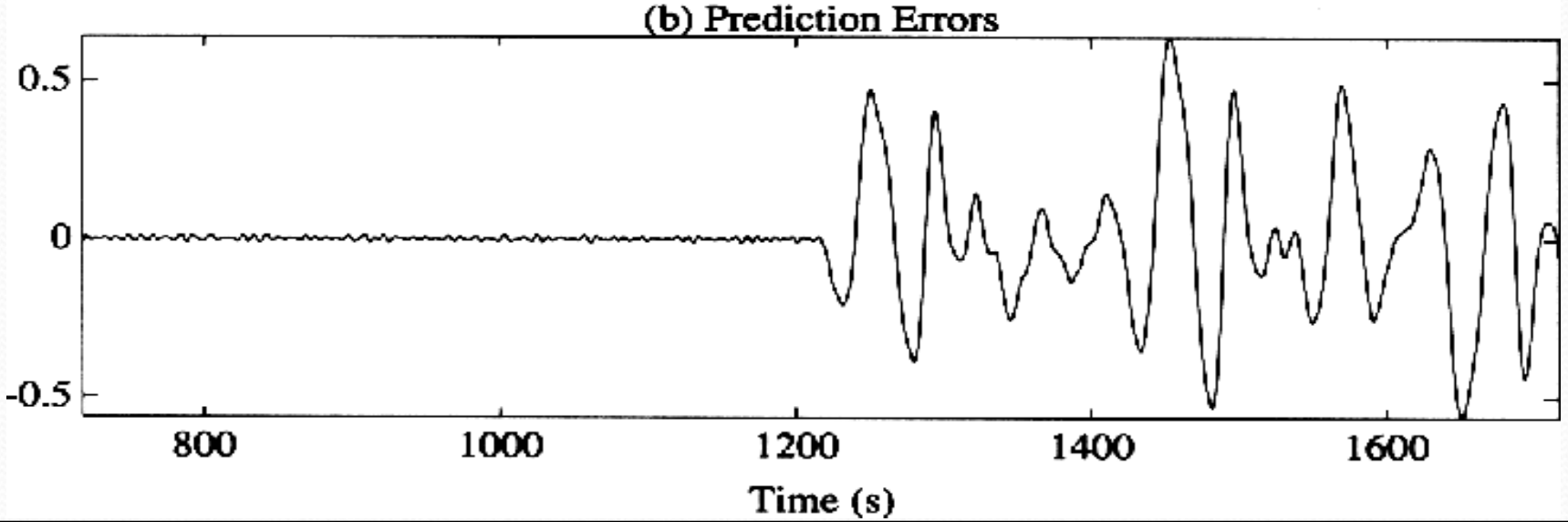
- Linear prediction:

$$x(t+6) = a_0 + a_1 x(t) + a_2 x(t-6) + a_3 x(t-12) + \cdots + a_{103} x(t-102*6)$$

- Order = number of terms

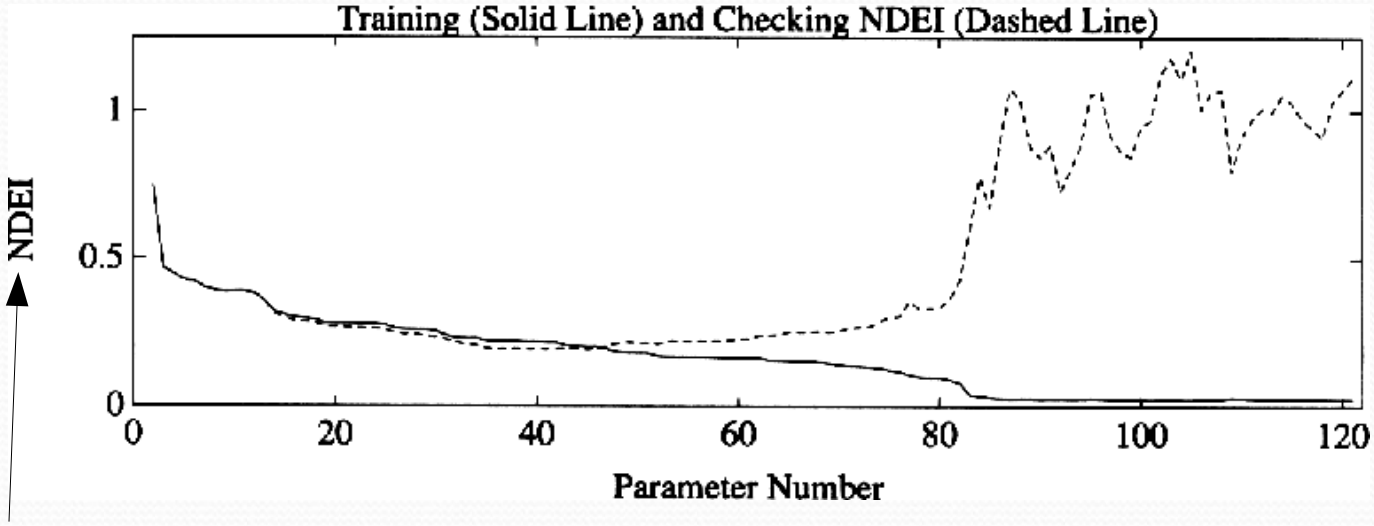
103rd order AR model





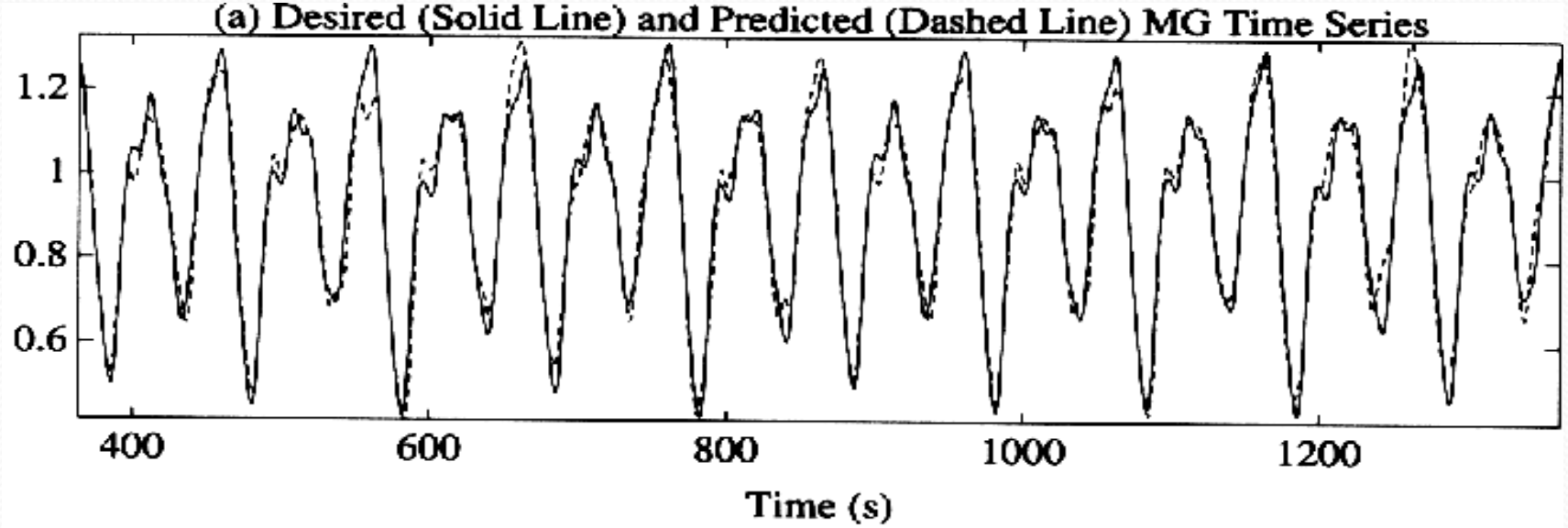
Order selection

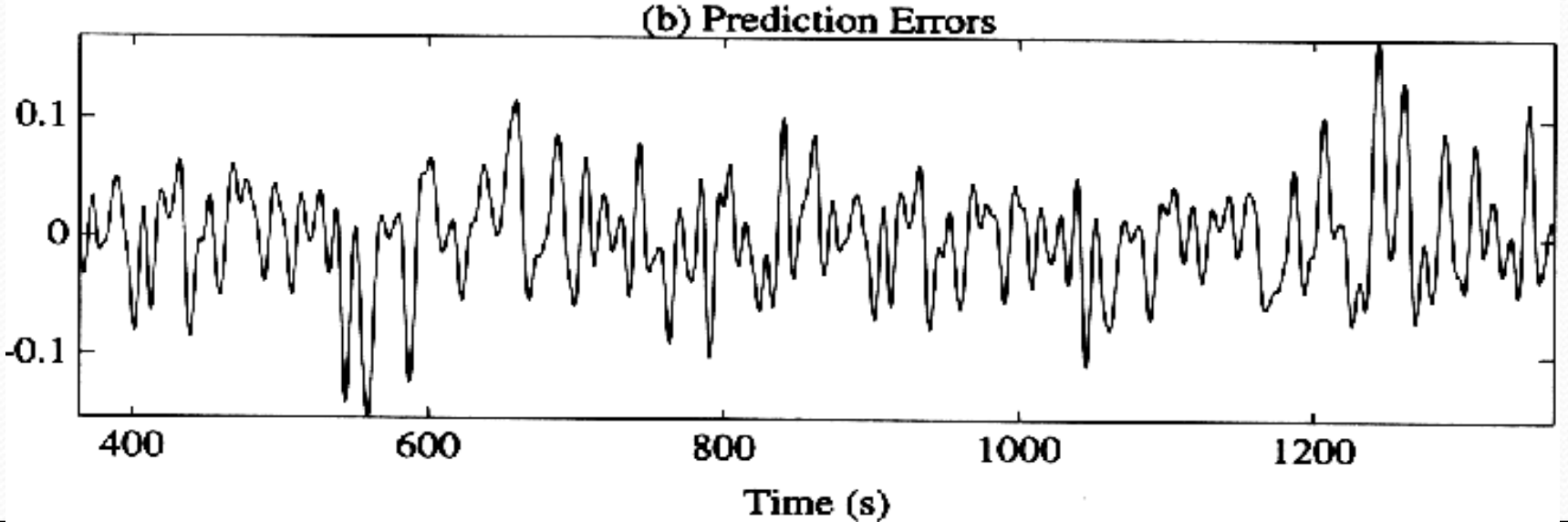
- Select optimal order of AR model in order to prevent overfitting
- Select the order that minimizes the error on a test set



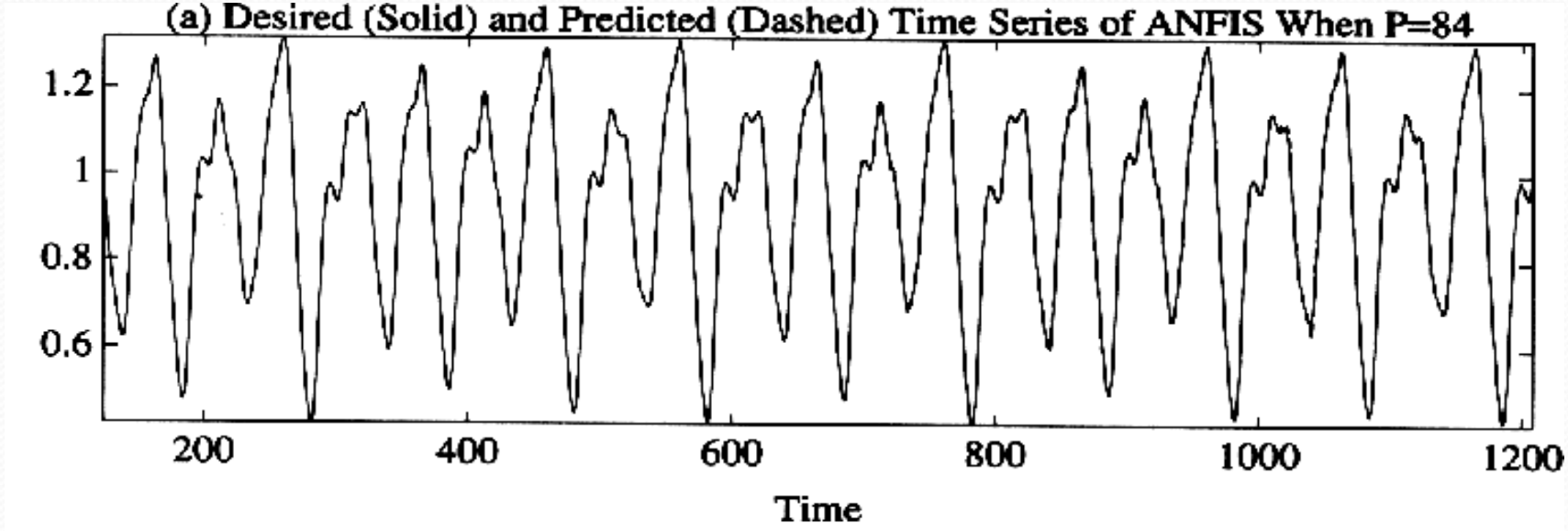
Root mean square error / standard deviation target

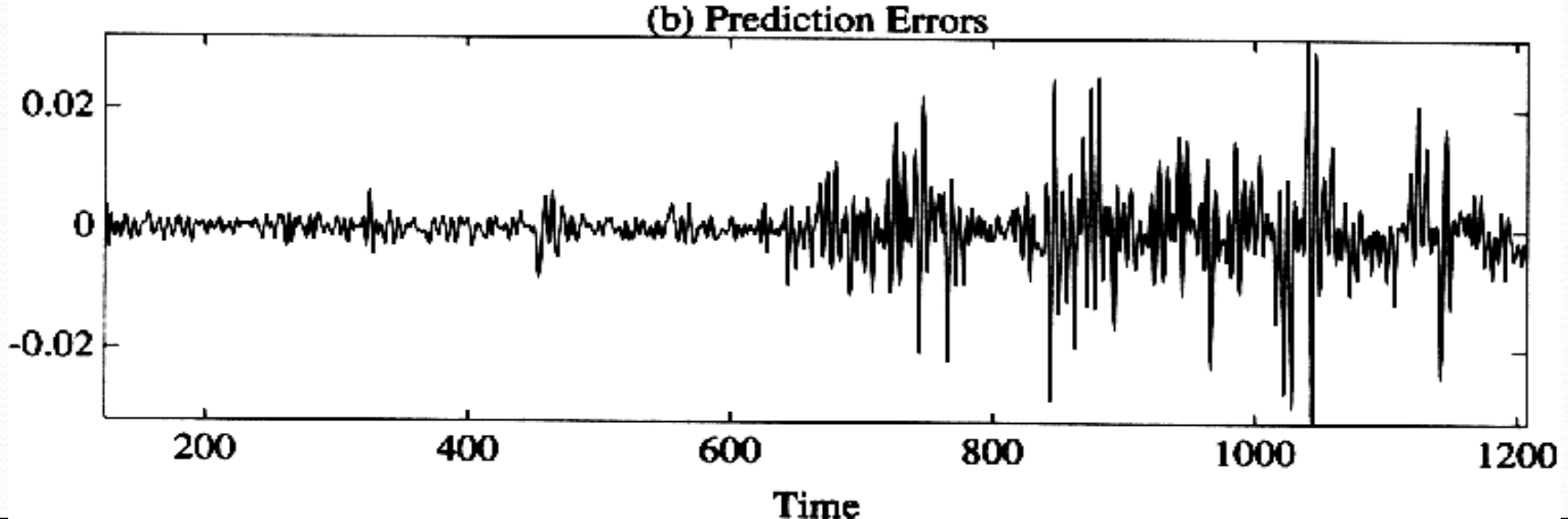
44th order AR model





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