INF FAKULTÄT FÜR INFORMATIK

Fuzzy Systems

Neuro-Fuzzy Systems

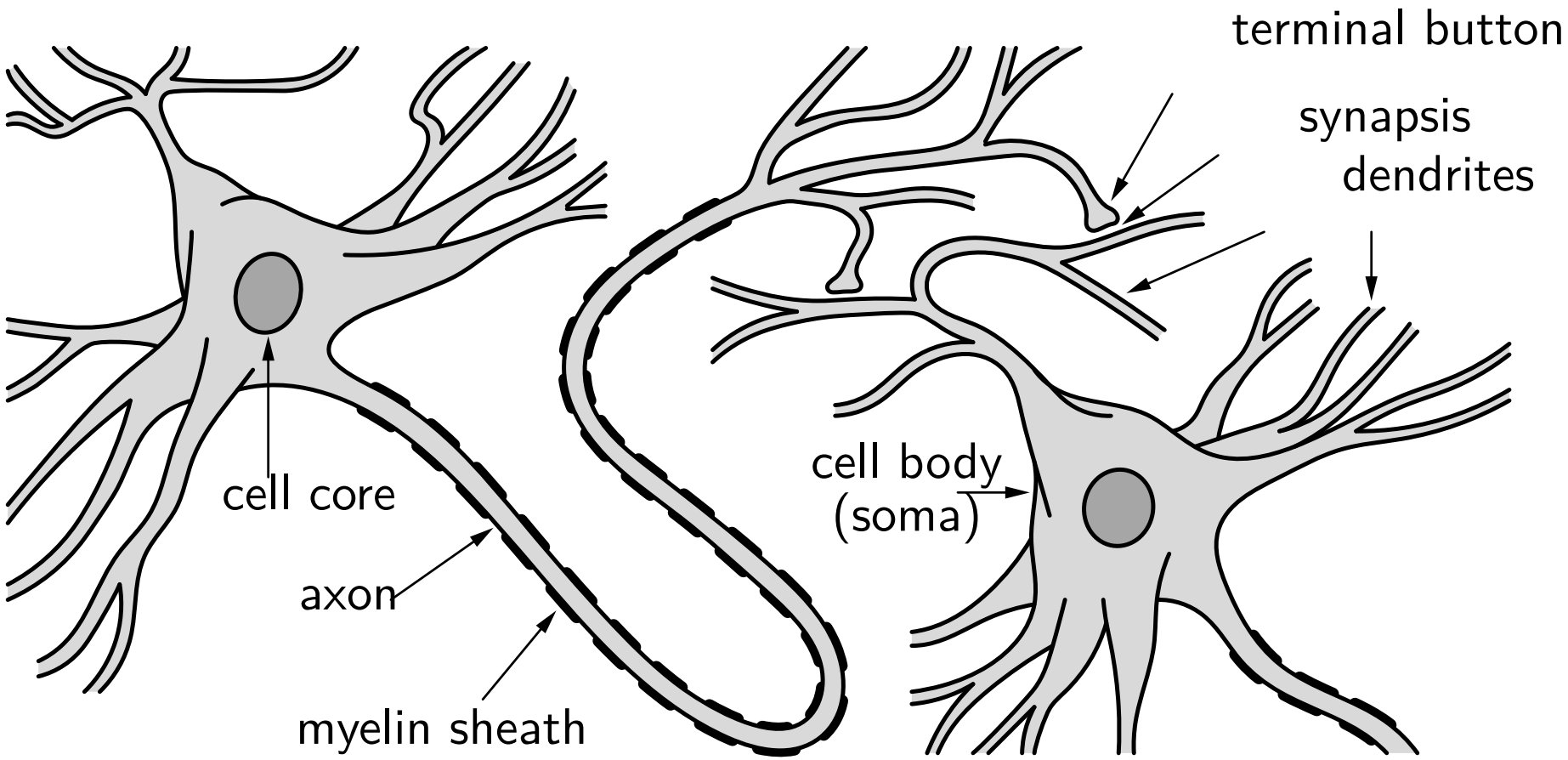
Prof. Dr. Rudolf Kruse **Christoph Doell**

{kruse,doell}@iws.cs.uni-magdeburg.de
Otto-von-Guericke University of Magdeburg
Faculty of Computer Science
Department of Knowledge Processing and Language Engineering

Outline

Biological Background

Structure of a prototypical biological neuron



Biological Background

Simplified description of neural information processing

The axon terminal releases chemicals, called neurotransmitters.

They act on the membrane of the receptor dendrite to change its polarization (inside is usually 70 mV more negative than outside).

- decrease in potential difference: *excitatory* synapse
- increase in potential difference: *inhibitory* synapse

If there is enough net excitatory input, the axon is depolarized.

The resulting *action potential* travels along the axon (the speed depends on the degree to which the axon is covered with myelin).

When the action potential reaches terminal buttons,
it triggers the release of neurotransmitters.

The Perceptron Model

Rosenblatt (1962) described a neuronal model as a computer program, which he called *perceptron*.

It was constructed to solve pattern recognition problems.

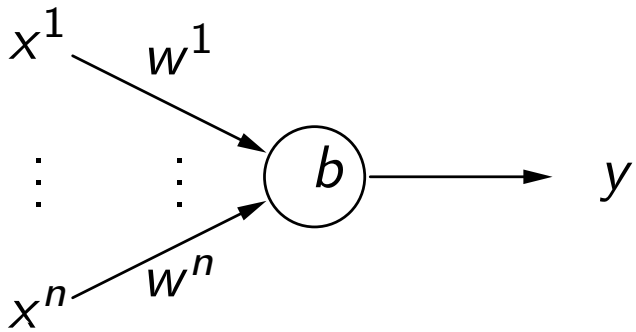
It constructs a rule to separate data of 2 classes using examples.

Each neuron is a *threshold logic unit*.

- n inputs $\boldsymbol{x} = (x^1, \ldots, x^n) \in X \subset \mathbb{R}^n$
- one output $y \in \{-1, +1\}$
- output is connected with inputs by

$$y = \operatorname{sgn}\{\langle \boldsymbol{w}, \boldsymbol{x} \rangle - b\}$$

- inner product $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$, threshold $b \in \mathbb{R}$
- weights $\boldsymbol{w} = (w^1, \ldots, w^n) \in X \subset \mathbb{R}^n$



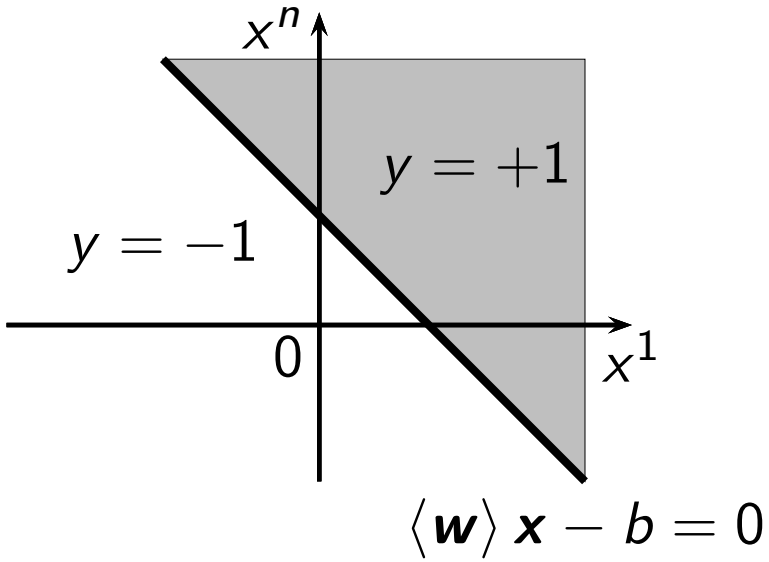
The Perceptron Model: Geometrical Representation

A neuron defines two regions in X where it takes values -1 and $+1$.

These regions are separated by

$$\langle \boldsymbol{w}, \boldsymbol{x} \rangle - b = 0$$

During the learning process, the perceptron chooses appropriate coefficients $\boldsymbol{w}$



The Perceptron Model: Net of Neurons

Rosenblatt considered the composition of several neurons

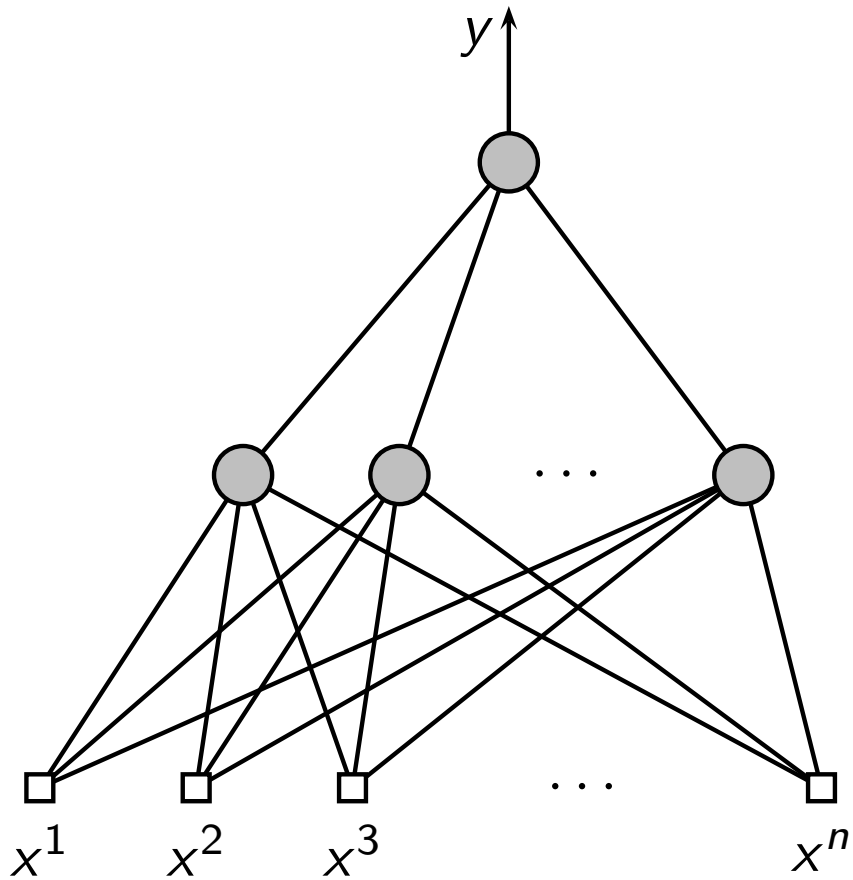
- arranged in some levels,
- last level with only one neuron.

Choosing appropriate coefficients for all neurons of the net: perceptron specifies two regions in X.

The regions are separated by piecewise linear surfaces.

Learning: Find coefficients for all neurons using the training data.

In the 1960s it wasn't clear how to choose coefficients simultaneously.



The Perceptron Model: Learning Algorithm

Therefore Rosenblatt suggested the following scheme:

1. Fix the coefficients of all neurons, except for the last one.
   X is transformed into a new space $\mathcal{Z}$.
2. During training, find the coefficients of the last neuron,
   i.e. construct a separating hyperplane in $\mathcal{Z}$.

He suggested to learn with reward and punishment stimuli.

- Training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l)$
- Corresponding training data $(\boldsymbol{z}_1, y_1), \ldots, (\boldsymbol{z}_l, y_l)$ in $\mathcal{Z}$
- At each time step k, Perceptron processes one example.
- $\boldsymbol{w}(k)$ denotes the coefficient vector of the last neuron at time k.

The Perceptron Model: Learning Algorithm (cont.)

The initial vector $\boldsymbol{w}$ is zero, *i.e.* $\boldsymbol{w}(0) = \mathbf{0}$.

If the next example of the training data $(\boldsymbol{z}_k, y_k)$ is classified correctly, *i.e.*

$$y_k \cdot \langle \boldsymbol{w}(k-1), \boldsymbol{z}_k \rangle > 0,$$

the coefficient vector of the hyperplane is not changed,
i.e. $\boldsymbol{w}(k) = \boldsymbol{w}(k-1)$.

If, however, the next example is misclassified, *i.e.*

$$y_k \cdot \langle \boldsymbol{w}(k-1), \boldsymbol{z}_k \rangle < 0,$$

the coefficient vector is changed by $\boldsymbol{w}(k) = \boldsymbol{w}(k-1) + y_k \cdot \boldsymbol{z}_k$

The Idea of Neural Networks

In 1986 several researchers independently proposed a method

- to simultaneously find the coefficients for all neurons of a perceptron
- using the so-called *back-propagation*.

It replaces the discontinuous $\operatorname{sgn}\{\langle \boldsymbol{w}, \boldsymbol{x} \rangle - b\}$ by a *sigmoid function*

$$y = S\left(\langle \boldsymbol{w}, \boldsymbol{x} \rangle - b\right)$$

S is a monotonic function with $S(-\infty) = -1$ and $S(+\infty) = +1$, *e.g.* $S(u) = \tanh(u)$.

The composition of neurons is a continuous function which, for any fixed $\boldsymbol{x}$, has a gradient *w.r.t.* all coefficients of all neurons.

The back-propagation method solves this gradient.

It only guarantees to find one of the local minima.

Outline

Neuro-Fuzzy Systems

Building a fuzzy system requires both

- prior knowledge (fuzzy rules, fuzzy sets),
- manual tuning which is time-consuming and error-prone.

This process can be supported by learning, *e.g.*

- learning fuzzy rules (structure learning),
- learning fuzzy sets (parameter learning).

How to use approaches from artificial neural networks for that?

Comparison of Neural Networks and Fuzzy System

Neural Networks	Fuzzy Systems
are low-level computational structures	deal with reasoning on a higher level
perform well when enough data are present	use linguistic information from domain experts
can learn	neither learn nor adjust themselves to new environment
are black-boxes for the user	are based on natural language

Neuro-fuzzy systems shall combine the parallel computation and learning abilities of neural networks with the human-like knowledge representation and explanation abilities of fuzzy systems.

As a result, neural networks become more transparent, while fuzzy systems become capable of learning.

Hybridization of Both Techniques

The idea of hybrid methods is to map fuzzy sets and fuzzy rules to a neural network structure.

For that, we consider the fuzzy rules R_i of a Mamdani controller

$$R_i: \quad \text{If } x_1 \text{ is } \mu_i^{(1)} \text{ and } \ldots \text{ and } x_n \text{ is } \mu_i^{(n)} \text{ then } y \text{ is } \mu_i,$$

or the fuzzy rules R_i' of a TSK controller

$$R_i': \quad \text{If } x_1 \text{ is } \mu_i^{(1)} \text{ and } \ldots \text{ and } x_n \text{ is } \mu_i^{(n)}, \text{ then } y = f_i(x_1, \ldots, x_n).$$

The activation $\tilde{a}_i$ of these rules can be calculated by a t-norm.

With input $\boldsymbol{x}$ and the minimum t-norm, we get

$$\tilde{a}_i(x_1, \ldots, x_n) = \min\{\mu_i^{(1)}(x_1), \ldots, \mu_i^{(n)} x_n\}.$$

Hybridization of Both Techniques

Main idea: We replace each connection weight $w_{ji} \in \mathbb{R}$ from an input neuron u_j to an inner neuron u_i by a fuzzy set $\mu_i^{(j)}$.

Thus u_i represents a rule and the connections from the input units represent the fuzzy sets of the antecedents of the rules.

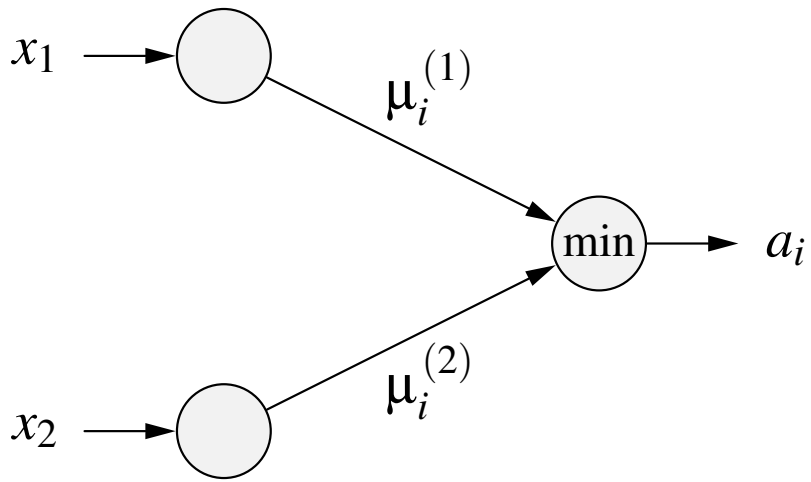
To calculate the rule activation of u_i, we must modify their network input functions.

For example, with minimum t-norm we obtain

$$\mathrm{net}_i = \min\{\mu_i^{(1)}(x_1), \ldots, \mu_i^{(n)} x_n\}$$

as network input function.

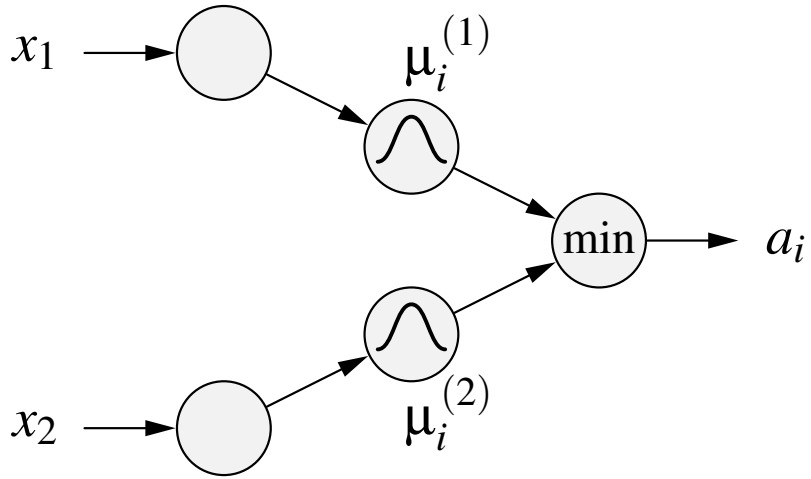
Fuzzy Sets as Weights



If we replace the activation function of the neuron by the identity, then it corresponds to the rule activation $\tilde{a}_i$.

So, the neuron can be used directly to compute the rule activity of any fuzzy rule.

Fuzzy Sets as Activation Functions



Another representation: The fuzzy sets of the antecedent are modeled as separate neurons.

The network input function is here the identity and the activation function is the fuzzy membership function.

We need 2 neuron layers to model the antecedent of a fuzzy rule.

Advantage: The fuzzy sets can be directly used in several rules (ensures interpretability).

Neuron Output Computation

For TSK each rule we get one more unit for evaluating the output function f_i.

It will be connected to all of the input units $(x_1, \ldots, x_n)$.

In the output layer, the outputs are be combined with the rule activations $\tilde{a}_i$.

This output neuron will finally calculate the output by the network input function

$$\text{out} = \frac{\sum_{i=1}^{r} \tilde{a}_i \cdot f_i(x_i, \ldots, x_n)}{\sum_{i=1}^{r} \tilde{a}_i}.$$

For Mamdani rules, it depends on the chosen t-conorm and the defuzzification method.

Also, a common output neuron combines the activations of the rule neurons and calculates a crisp output value.

Summary of Hybridization Steps

1. For every input x_i, create a neuron in the input layer.
2. For every fuzzy set $\mu_i^{(j)}$, create a neuron and connect it to the corresponding x_i.
3. For every output variable y_i, create one neuron.
   For every fuzzy rule R_i, create an inner (rule) neuron and specify a t-norm for calculating the rule activation.
4. Every R_i is connected according to its fuzzy rule to the "antecedent" neurons.

5.a) Mamdani: Every rule neuron is connected to the output neuron according to the consequent. A t-conorm and the defuzzification method have to be integrated into the output neurons.

6.b) TSK: For every rule unit, one more neuron is created for the output function. These neurons are connected to the corresponding output neuron.

Advantages and Problems of this Approach

Now learning algorithms of artificial neural networks can be applied to this structure.

Usually the learning methods have to be modified due to some reasons:

- The network input and activation functions changed.
- Not the real-valued network weights but the parameter of the fuzzy sets have to be learned.

In the following, we discuss 2 hybrid neuro-fuzzy systems,
i.e. ANFIS [Jang, 1993] and NEFCLASS [Nauck and Kruse, 1997].

Models with Supervised Learning Methods

NFS with supervised learning optimize the fuzzy sets of a given rule base by observed input-output tuples.

Requirement: An existing (fuzzy) rule base must exist.

Convenient for replacing a standard controller by a fuzzy controller.

If no initial rule base is available, we might apply fuzzy clustering to the input data for that.

In the following, we discuss a typical example for a neuro-fuzzy system with supervised learning, *i.e.* the ANFIS model

Several other approaches are discussed, *e.g.* in [Nauck et al., 1997].

The ANFIS Model

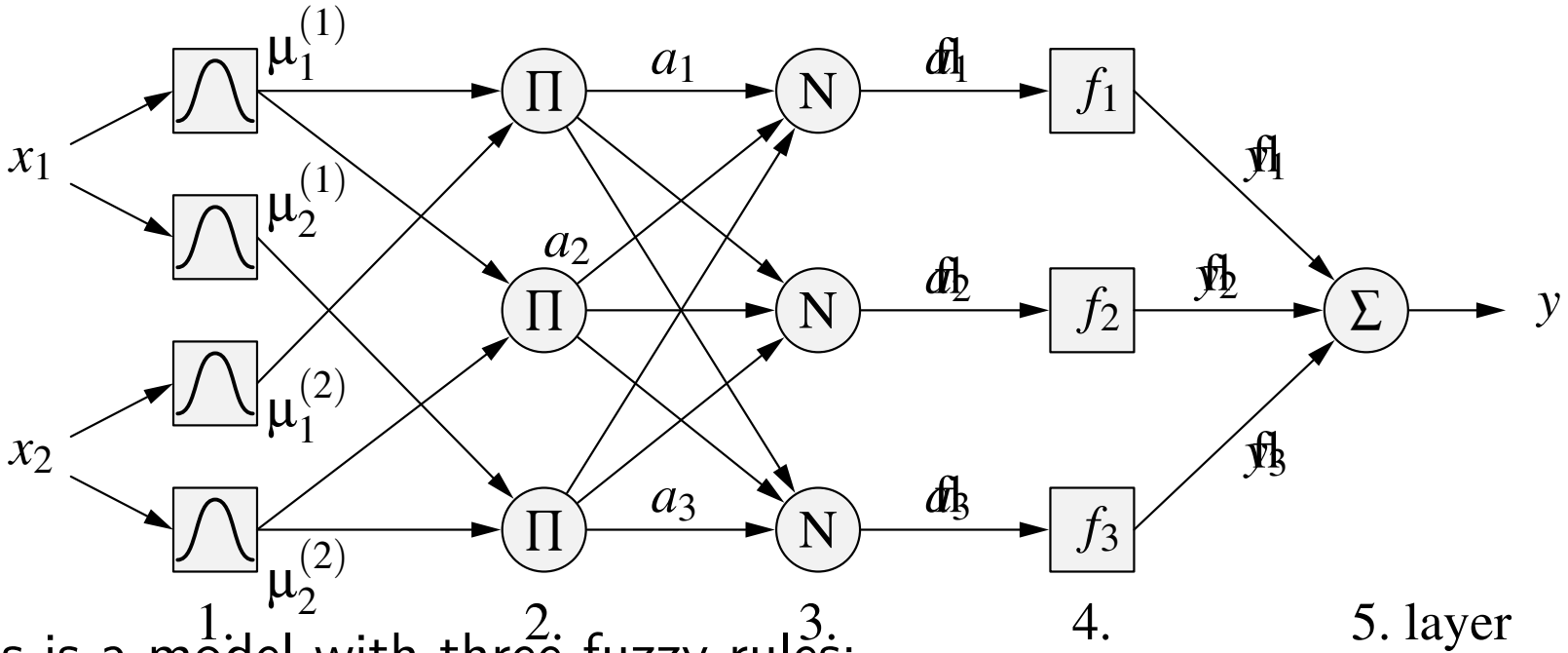
In [Jang, 1993] the neuro-fuzzy system *ANFIS* (Adaptive-Network-based Fuzzy Inference System) was developed.

By now it has been integrated in many controllers and simulation tools, *e.g.* Matlab.

The ANFIS model is based on a hybrid structure, *i.e.* it can be interpreted as neural network and as fuzzy system.

The model uses the fuzzy rules of a TSK controller.

Example of an ANFIS Model



This is a model with three fuzzy rules:

$$R_1: \quad \text{If } x_1 \text{ is } A_1 \text{ and } x_2 \text{ is } B_1 \text{ then } y = f_1(x_1, x_2)$$

$$R_2: \quad \text{If } x_1 \text{ is } A_1 \text{ and } x_2 \text{ is } B_2 \text{ then } y = f_2(x_1, x_2)$$

$$R_3: \quad \text{If } x_1 \text{ is } A_2 \text{ and } x_2 \text{ is } B_2 \text{ then } y = f_3(x_1, x_2)$$

with linear output functions $f_i = p_i x_1 + q_i x_2 + r_i$ in the antecedent part.

ANFIS: Layer 1 – The Fuzzification Layer

Here, neurons represent fuzzy sets of the fuzzy rule antecedents.

The activation function of a membership neuron is set to the function that specifies the neuron's fuzzy set.

A fuzzification neuron receives a crisp input and determines the degree to which this input belongs to the neuron's fuzzy set.

Usually bell-curved functions are used, *e.g.*

$$\mu_i^{(j)}(x_j) = \frac{1}{1 + \left(\frac{x_j - a_i}{b_i}\right)^{2c_i}}$$

where a_i, b_i, c_i are parameters for center, width, and slope, resp.

The output of a fuzzification neuron thus also depends on the membership parameters.

ANFIS: Layer 2 – The Fuzzy Rule Layer

Each neuron corresponds to a single TSK fuzzy rule.

A fuzzy rule neuron receives inputs from the fuzzification neurons that represent fuzzy sets in the rule antecedents.

It calculates the firing strength of the corresponding rule.

In NFS, the intersection is usually implemented by the product.

So, the firing strength $\tilde{a}_i$ of rule R_i is

$$\tilde{a}_i = \prod_{j=1}^{k} \mu_i^{(j)}(x_j).$$

ANFIS: Layer 3 – The Normalization Layer

Each neuron in this layer receives the firing strengths from all neurons in the rule layer.

The normalised firing strength of a given rule is calculated here.

It represents the contribution of a given rule to the final result.

Thus, the output of neuron i in layer 4 is determined as

$$\bar{a}_i = a_i = \mathrm{net}_i = \frac{\tilde{a}_i}{\sum_j \tilde{a}_j}.$$

ANFIS: Layers 4 and 5 – Defuzzification and Summation

Each neuron in layer 4 is connected to the respective normalisation neuron, and also receives the raw input values $\boldsymbol{x}$.

A defuzzification neuron calculates the weighted consequent value of a given rule as

$$\bar{y}_i = a_i = \mathrm{net}_i = \bar{a}_i f_i(x_1, \ldots, x_n).$$

The single neuron in layer 5 calculates the sum of outputs from all defuzzification neurons and produces the overall ANFIS output:

$$y \quad = \quad f(\boldsymbol{x}_i) = a_{\mathrm{out}} = \mathrm{net}_{\mathrm{out}} = \sum_i \bar{y}_i = \frac{\sum_i \tilde{a}_i f_i(x_1, \ldots, x_n)}{\sum_i \tilde{a}_i}.$$

How does ANFIS learn? – The Forward Pass

ANFIS uses a hybrid learning algorithm that combines least-squares and gradient descent [Jang, 1993].

Each learning epoch is composed of one forward and one backward pass.

In the **forward pass**, a training set of input-output tupples $(\boldsymbol{x}_k, y_k)$ is presented to the ANFIS, neuron outputs are calculated on the layer-by-layer basis, and rule consequent parameters are identified by least squares.

Goal: minimize mean squared error $e = \sum_{i=1}^{m} |y(k) - f(\boldsymbol{x}(k))|^2$

How does ANFIS learn? – The Forward Pass

r_{ij}: parameters of output function f_i, $x_i(k)$: input values, $y(k)$: output value of k-th training pair, $\bar{a}_i(k)$: relative control activation

Then we obtain

$$y(k) = \sum_i \bar{a}_i(k) y_i(k) = \sum_i \bar{a}_i(k) \left(\sum_{j=1}^{n} r_{ij} x_j(k) + r_{i0} \right), \qquad \forall i, k.$$

Therefore, with $\hat{x}_i(k) := [1, x_1(k), \ldots, x_n(k)]^T$ we obtain the overdetermined linear equation system

$$\mathbf{y} = \mathbf{aRX}$$

for $m > (n+1) \cdot r$ with m number of training points, r number of rules, n number of input variables.

The consequent parameters are adjusted while the antecedent parameters remain fixed.

How does ANFIS learn? – The Backward Pass

In the **backward pass**, the error is determined in the output units based on the new calculated output functions.

Also, with the help of gradient descent, the parameters of the fuzzy sets are optimized.

Back propagation is applied to compute the "error" of the neurons in the hidden layers

It updates the parameters of these neurons by the chain rule.

ANFIS: Summary

Forward and backward passes improves convergence.

Reason: Least squares already has an optimal solution for the parameters of the output function *w.r.t.* the initial fuzzy sets.

Unfortunately ANFIS has no restrictions for the optimization of the fuzzy sets in the antecedents. So, after optimization the input range might not be covered completely with fuzzy sets.

Thus definition gaps can appear which have to be checked afterwards.

Fuzzy sets can also change, independently form each other, and can also exchange their order and so their importance, too.

We have to pay attention to this, especially if an initial rule base was set manually and the controller has to be interpreted afterwards.

Learning Fuzzy Sets

Gradient descent procedures are only applicable, if a differentiation is possible, *e.g.* for Sugeno-type fuzzy systems.

Applying special heuristic procedures that do not use any gradient information can facilitate the learning of Mamdani-type rules.

Learning algorithms such as NEFCLASS [Nauck and Kruse, 1997] are based on the idea of backpropagation but constrain the learning process to ensure interpretability.

Learning Fuzzy Sets

Mandatory constraints: Fuzzy sets must ...

- stay normal and convex,
- not exchange their relative positions (they must not "pass" each other),
- always overlap.

Optional constraints:

- Fuzzy sets must stay symmetric.
- The membership degrees must add up to 1.

A learning algorithm must enforce these constraints.

Example: Medical Diagnosis

The *Wisconsin Breast Cancer Dataset* stores results from patients tested for breast cancer.

These data can be used to train and evaluate classifiers.

For instance, decision support systems must tell if unseen data indicate malignant or benign case?

A surgeon must be able to check this classification for plausibility.

We are looking for a simple and interpretable classifier.

Example: WBC Data Set

699 cases (16 cases have missing values).

2 classes: benign (458), malignant (241).

9 attributes with values from $\{1, \ldots, 10\}$ (ordinal scale, but usually interpreted numerically).

In the following, x_3 and x_6 are interpreted as nominal attributes.

x_3 and x_6 are usually seen as "important" attributes.

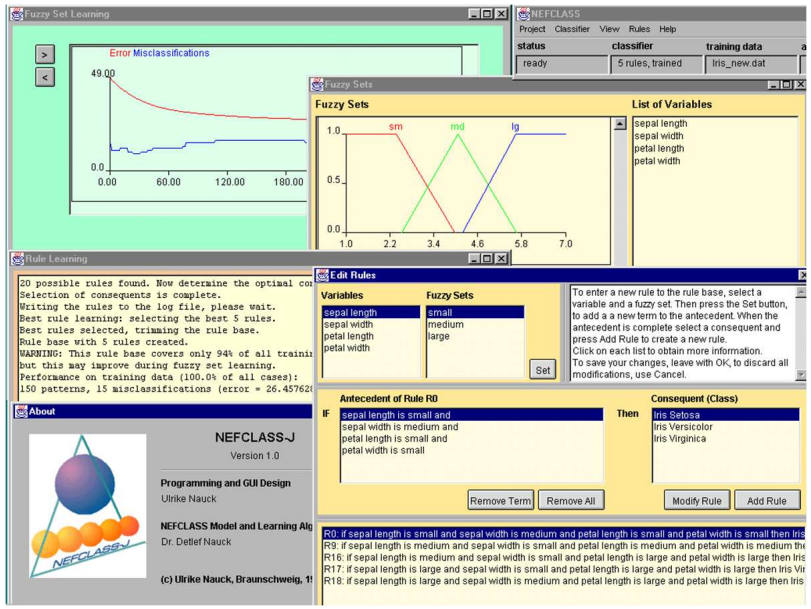
Applying NEFCLASS-J

A tool for developing neuro-fuzzy classifiers.

It is written in Java.

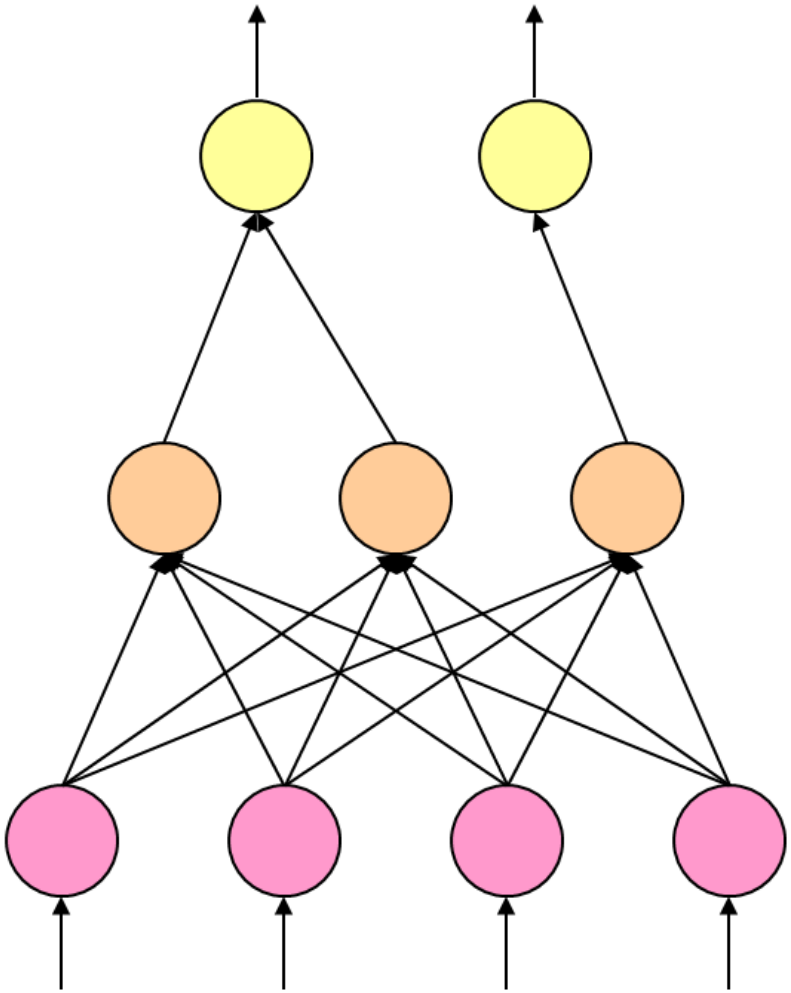
A free version for research is available.

This project started at our group.



`http://fuzzy.cs.ovgu.de/nefclass/nefclass-j/`

NEFCLASS: Neuro-Fuzzy Classifier



output variables

unweighted connections

fuzzy rules

fuzzy sets (antecedents)

input attributes (variables)

NEFCLASS: Features

It automatically induces a fuzzy rule base from data.

It can handle several shapes of fuzzy sets.

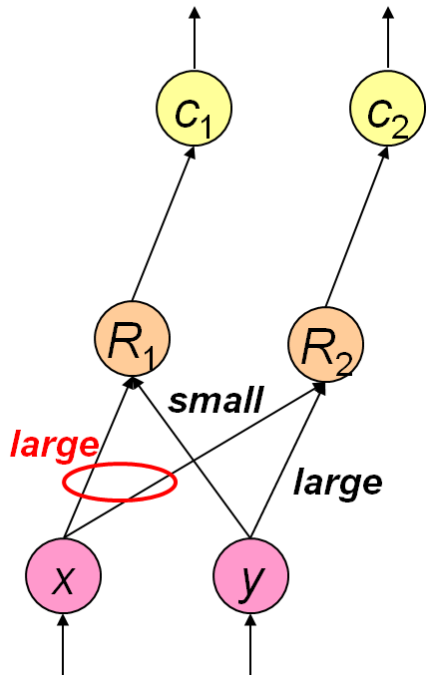
It processes numeric and symbolic attributes.

It treats missing values (no imputation).

It automatically prunes the rule base.

It fuses expert knowledge and rules obtained from data.

Representation of Fuzzy Rules



Example: 2 rules

R_1 : if x is *large* and y is *small*, then class is c_1

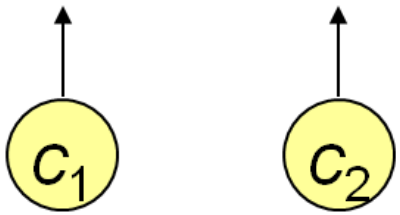
R_2 : if x is *large* and y is *large*, then class is c_2

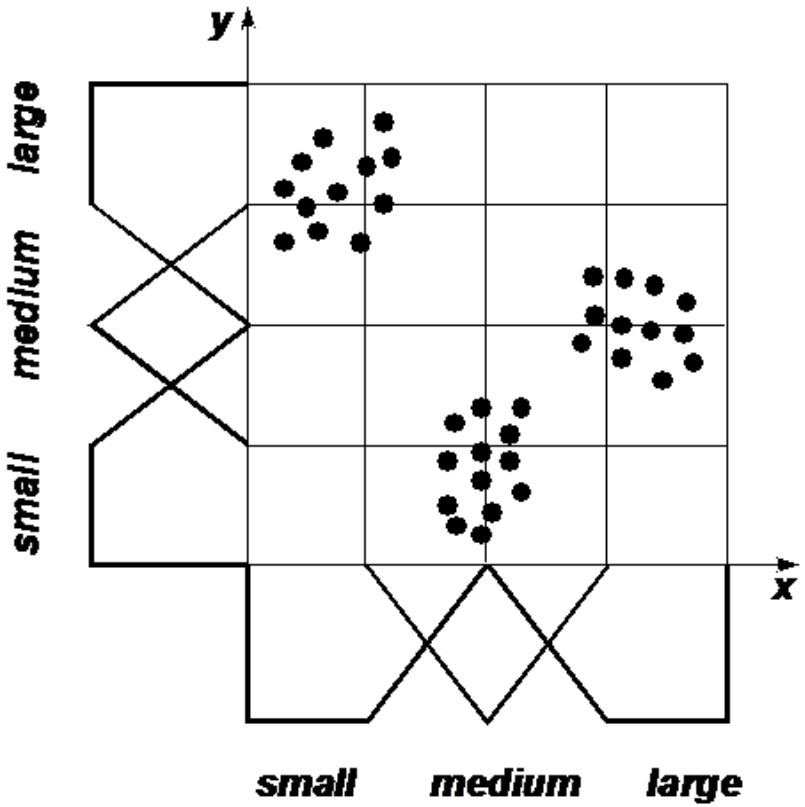
Connections $x \rightarrow R_1$ and $x \rightarrow R_2$ are linked.

Fuzzy set *large* is a shared weight,
i.e. the term *large* has always the same meaning in both rules.

1. Training Step: Initialization

Specify initial fuzzy partitions for all input variables.





1. Training Step: Rule Base

for each pattern p {
  find antecedent A such that $A(p)$ is maximal
  if $A \notin L$ {
    add A to L
  }
}
for each antecedent $A \in L$ {
  find best consequent C for A
  create rule base candidate $R = (A, C)$
  determine performance of R
  add R to B
}
return one rule base from B

Fuzzy rule bases can also be created by using prior knowledge, fuzzy cluster analysis, fuzzy decision trees, evolutionary algorithms, ...

Selection of a Rule Base

The performance of a rule is evaluated by

$$P_r = \frac{1}{N} \sum_{i=1} N(-1)^c R_r(\boldsymbol{x}_i)$$

with

$$c = \begin{cases} 0 & \text{if class}(x_i) = \text{con}(R_r), \\ 1 & \text{otherwise} \end{cases}$$

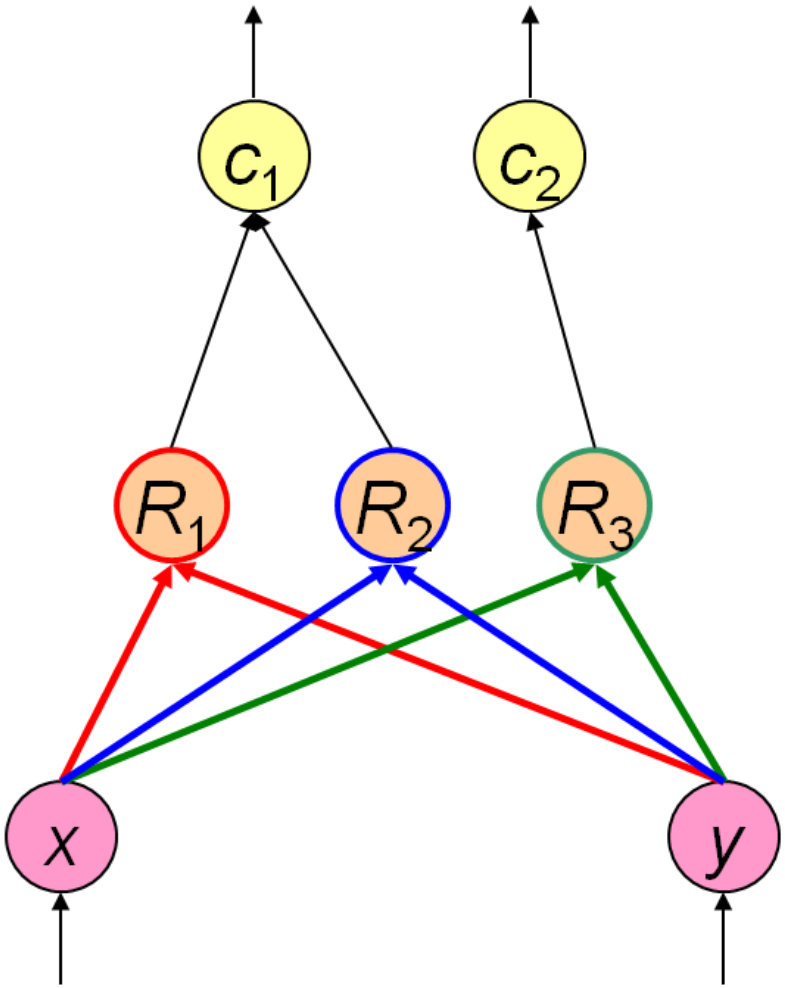
Then the rules are sorted by performance.

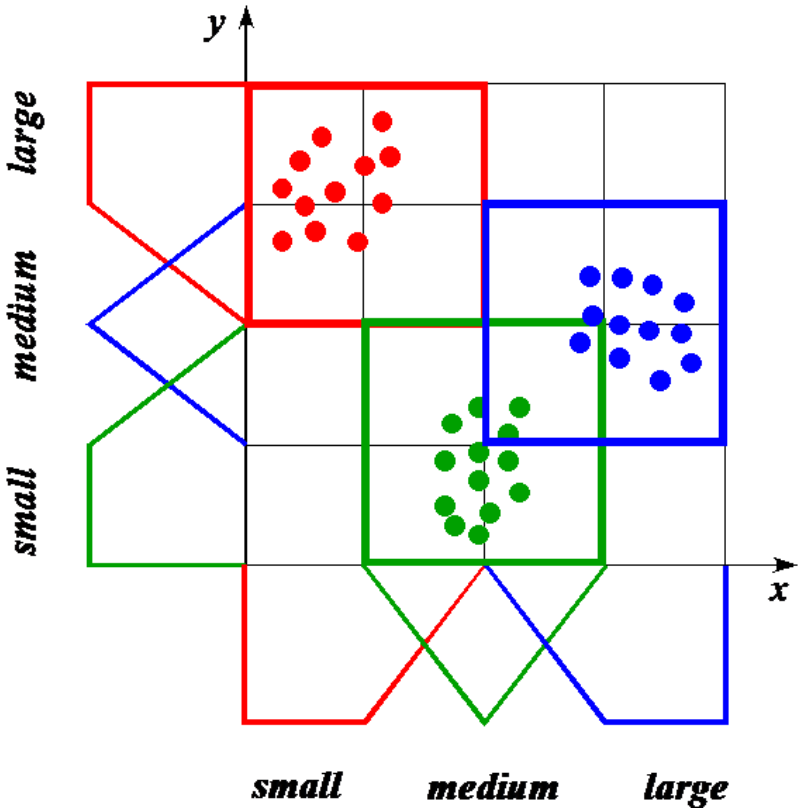
Either the best r rules or the best r/m rules per class are selected.

r is either given or is determined automatically such that all patterns are covered.

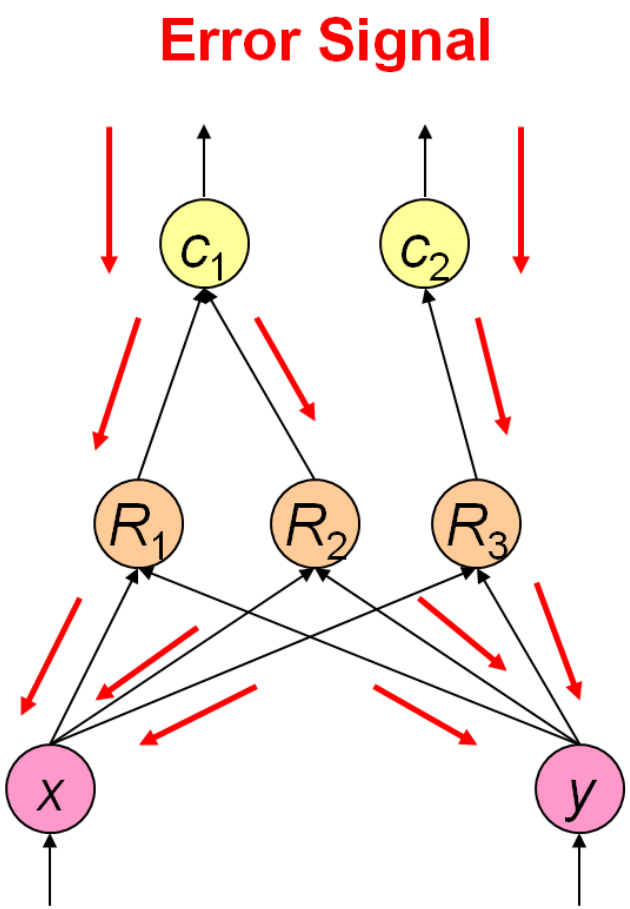
Rule Base Induction

NEFCLASS uses modified Wang-Mendel procedure.





Computing the Error Signal



fuzzy error (jth output):

$$E_j = \text{sgn}(d)(1 - \gamma(d))$$

with $d = t_j - o_j$ and $\gamma : \mathbb{R} \to [0, 1], \gamma(d) = \exp - \left(\frac{a \cdot d}{d_{\max}}\right)^2$ (t: correct output, o: actual output)

rule error:

$$E_r = (\tau_r(1 - \tau_r) + \varepsilon)E_{\text{con}(R_r)}$$

with $0 < \varepsilon \ll 1$

3. Training Step: Fuzzy Sets

e.g. triangular membership function:

$$\mu_{a,b,c} : \mathbb{R} \to [0,1], \quad \mu_{a,b,c}(x) = \begin{cases} \frac{x-a}{b-a} & \text{if } x \in [a,b), \\ \frac{c-x}{c-b} & \text{if } x \in [b,c], \\ 0 & \text{otherwise} \end{cases}$$

Parameter updates for an antecedent fuzzy set:

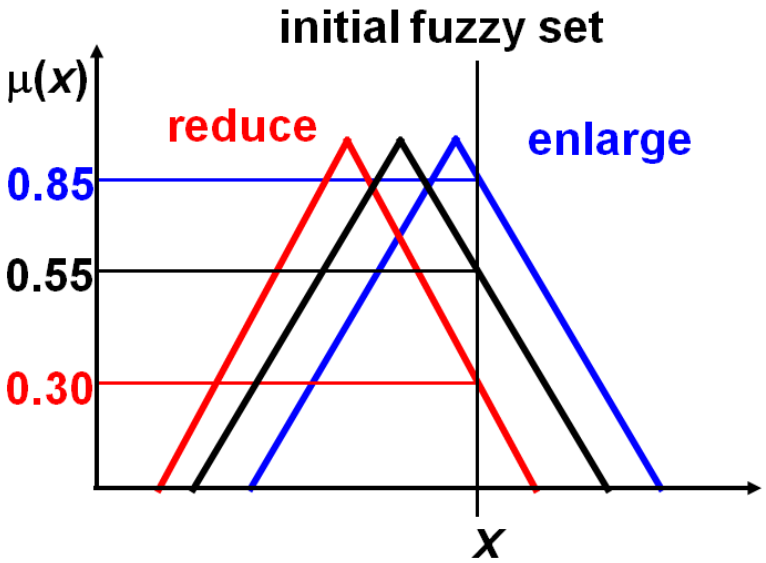
$$f = \begin{cases} \sigma\mu(x) & \text{if } E < 0, \\ \sigma(1-\mu(x)) & \text{otherwise} \end{cases}$$

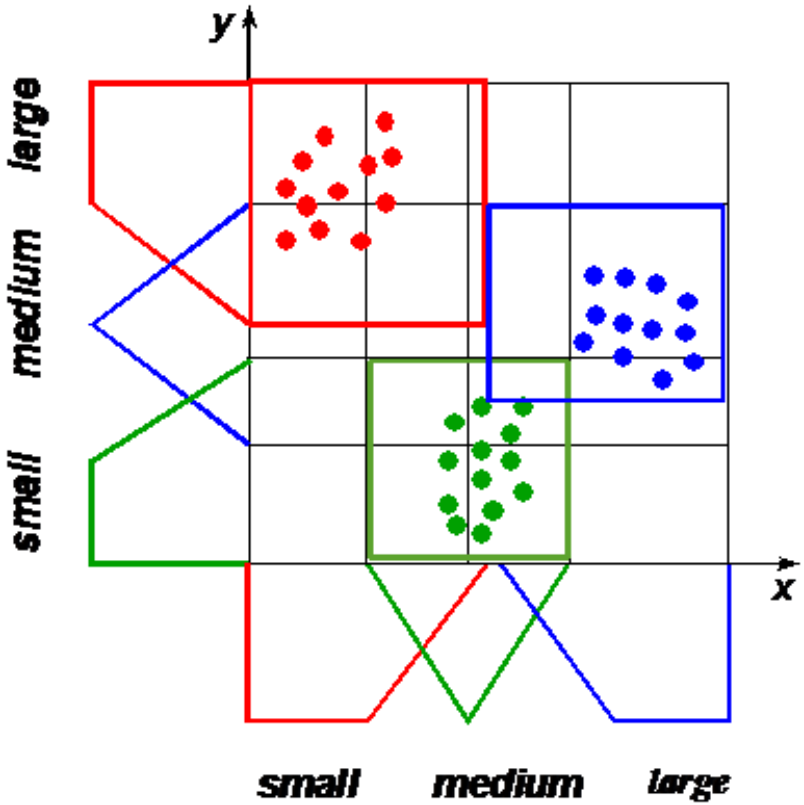
$$\Delta b = f \cdot E \cdot (c-a)\,\mathrm{sgn}(x-b)$$

$$\Delta a = -f \cdot E \cdot (b-a) + \Delta b$$

$$\Delta c = f \cdot E \cdot (c-b) + \Delta b$$

Training of Fuzzy Sets

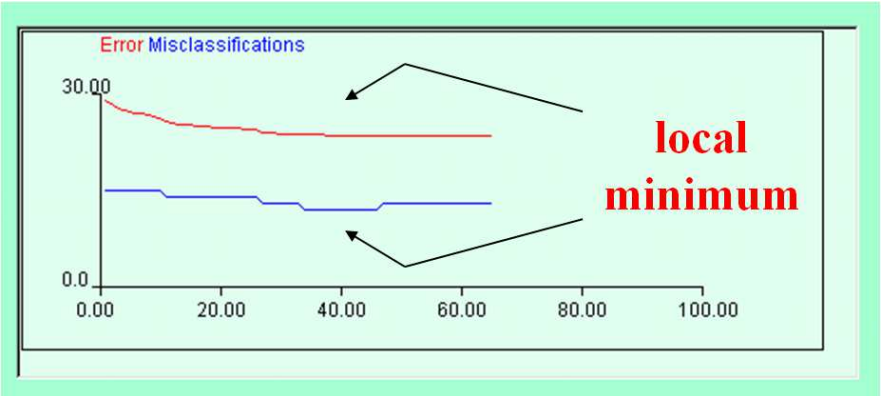




Heuristics: A fuzzy set is moved away from x (towards x) and its support is reduced (enlarged) in order to reduce (enlarge) the degree of membership of x.

Training of Fuzzy Sets

do {
  for each pattern {
    accumulate parameter updates
    accumulate error
  }
  modify parameters
} **while** change in error



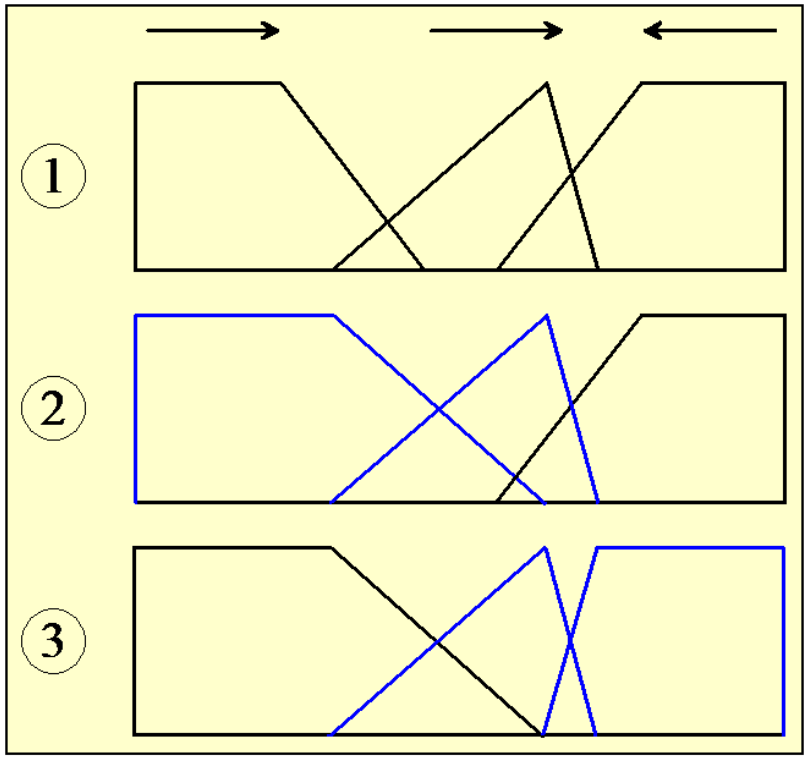
observing the error on validation set

variations:

- adaptive learning rate
- online/batch learning
- optimistic learning (n step look ahead)

Constraints for Training Fuzzy Sets

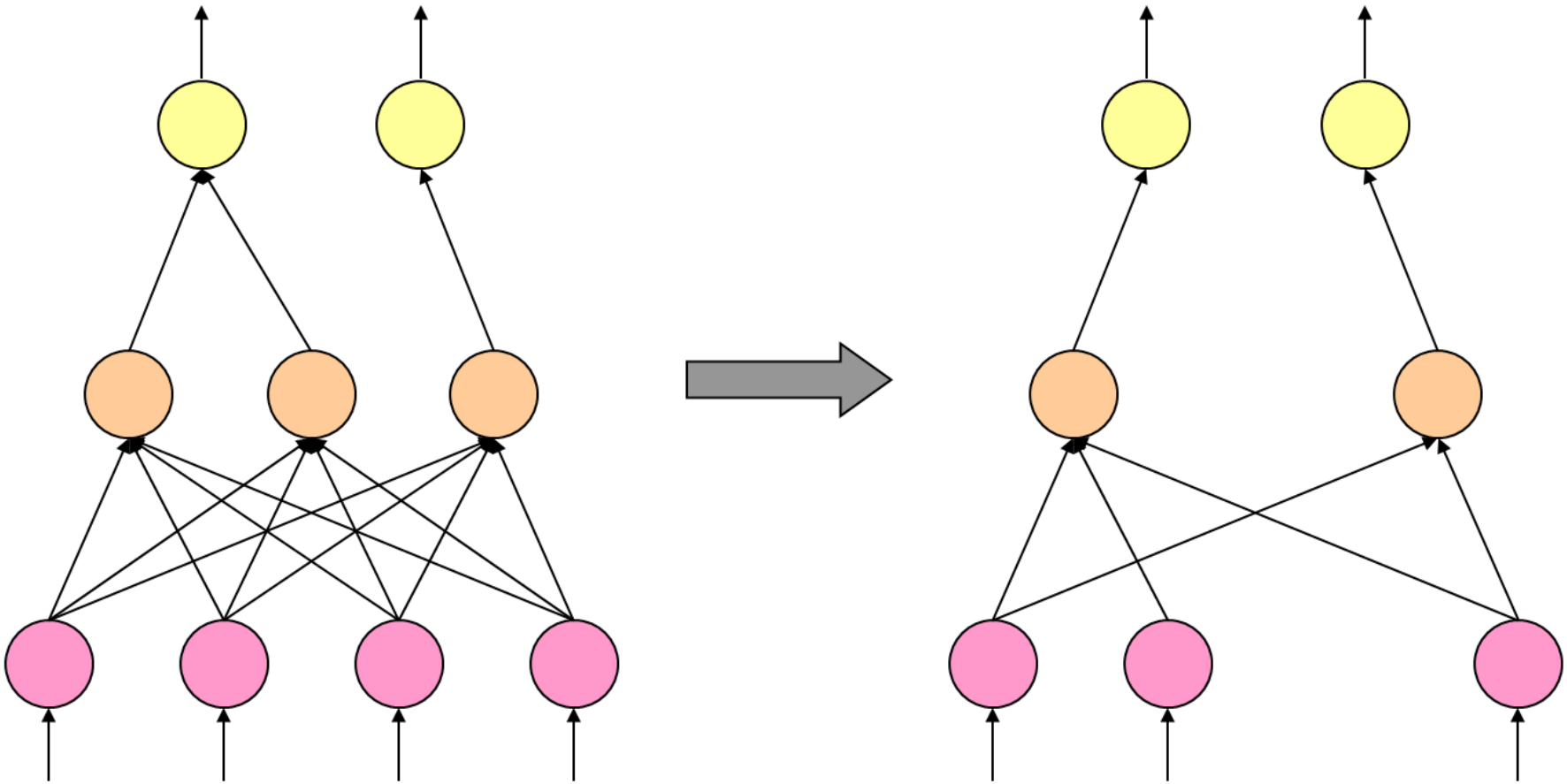
- valid parameter values
- non-empty intersection of adjacent fuzzy sets
- keep relative positions
- maintain symmetry
- complete coverage (degrees of membership add up to 1 for each element)



Correcting a partition after modifying the parameters

4. Training Step: Pruning

Goal: Remove variables, rules, and fuzzy sets in order to improve the interpretability and generalization.



Pruning Methods

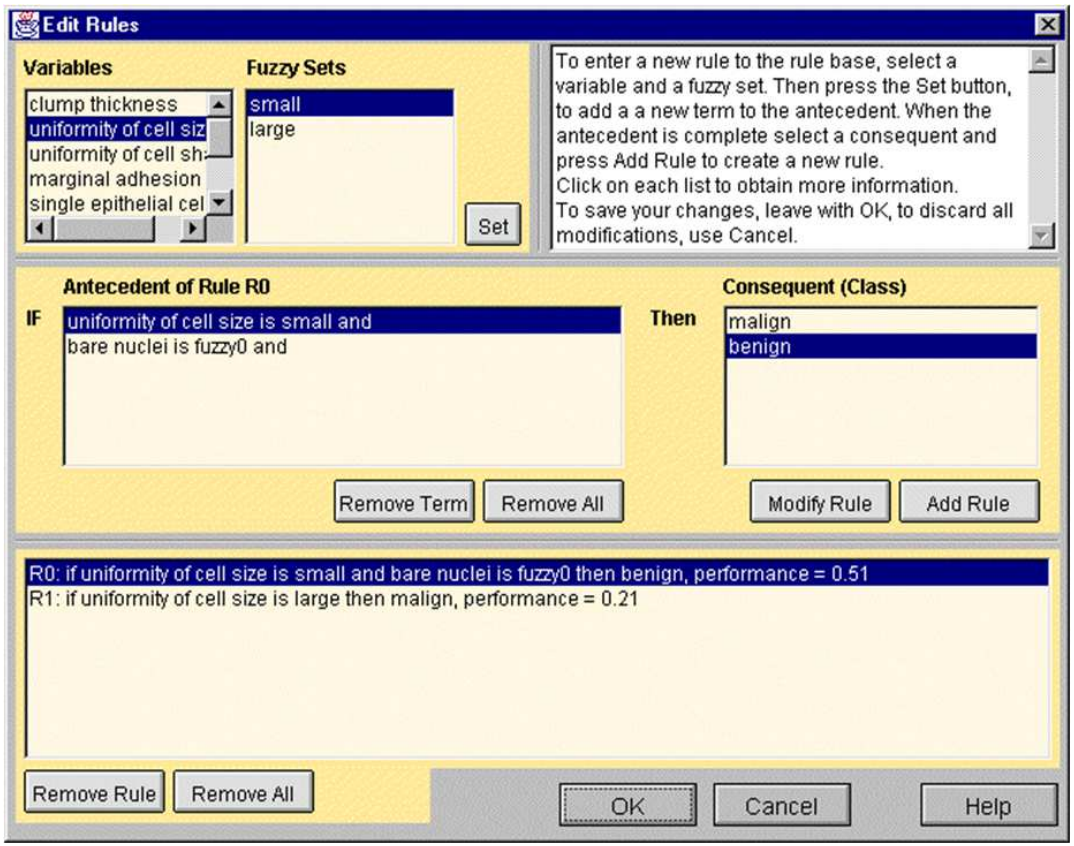
do {
  select pruning method
  do {
    execute pruning step
    train fuzzy sets
    if no improvement {
      undo step
    }
  } **while** there is improvement
} **while** there is further method

1. Remove variables (use correlations, information gain etc.).
2. Remove rules (use rule performance).
3. Remove terms (use degree of fulfillment).
4. Remove fuzzy sets (use fuzziness).

Example: WBC Fuzzy Rules

R_1: if uniformity of cell size is *small* and bare nuclei is fuzzy0 then *benign*

R_2: if uniformity of cell size is *large* then *malignant*



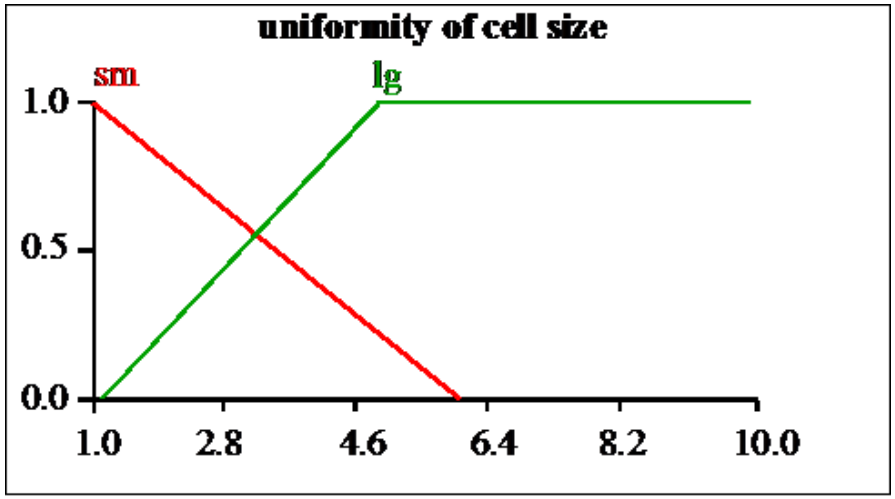
Example: WBC Classification Performance

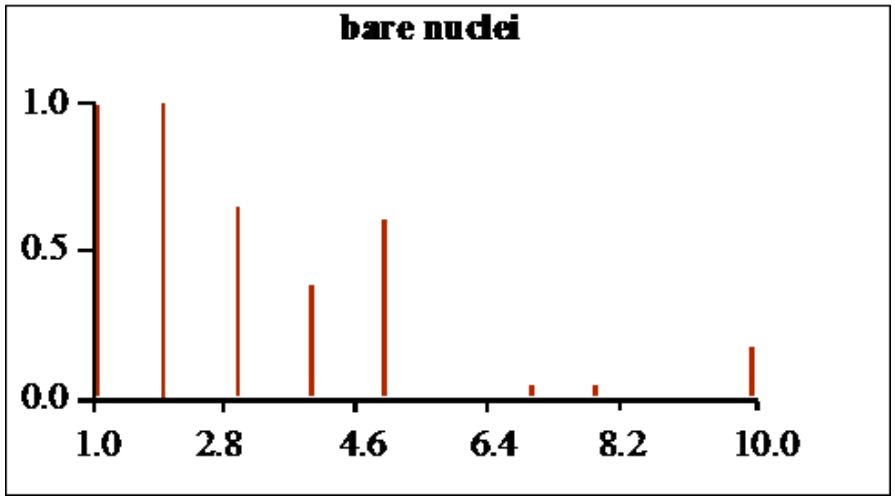
	predicted class					
	malign		benign		$\sum$	
malign	228	(32.62%)	13	(1.86%)	241	(34.99%)
benign	15	(2.15%)	443	(63.38%)	458	(65.01%)
$\sum$	243	(34.76)	456	(65.24)	699	(100.00%)

estimated performance on unseen data (cross validation):

NEFCLASS-J:	95.42%	NEFCLASS-J (numeric):	94.14%
Discriminant Analysis:	96.05%	Multilayer Perceptron:	94.82%
C 4.5:	95.10%	C 4.5 Rules:	95.40%

Example: WBC Fuzzy Sets





Example: Stock Market Prediction

[Kruse et al., 1998]

Prognosis of the daily proportional changes of the German stock exchange DAX (Siemens).

Database: Time series from 1986 to 1997.

DAX	Composite-DAX
German 3 months interest rate	Return Germany
German Morgan-Stanley index	Dow Jones industrial index
DM / US-$	US treasure bonds
gold price	Japanese Nikkei-Index
European Morgan-Stanley-Index	Price earning ratio

Fuzzy Rules in Finance

trend rule:

IF DAX is decreasing AND US-$ is decreasing
THEN DAX prediction is decreasing
WITH high certainty

turning point rule:

IF DAX is decreasing AND US-$ is increasing
THEN DAX prediction is increasing
WITH low certainty

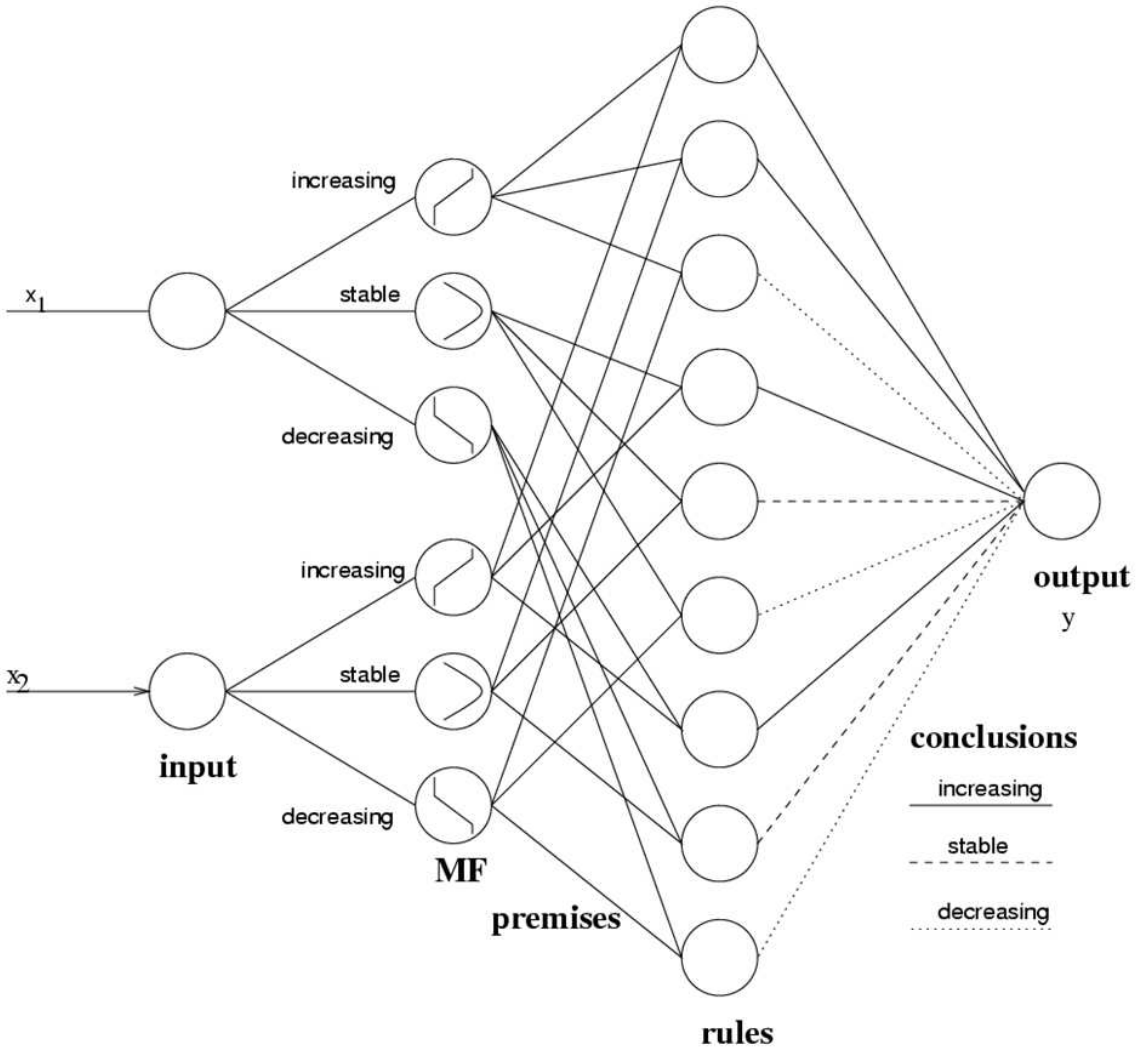
delay rule:

IF DAX is stable AND US-$ is decreasing
THEN DAX prognosis is decreasing
WITH very high certainty

in general:

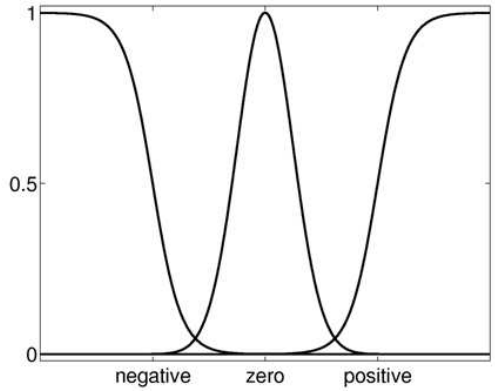
IF x_1 is μ_1 AND x_2 is μ_2 AND $\ldots$ AND x_n is μ_n
THEN $y = \eta$
WITH weight k

Neuro-Fuzzy Architecture



From Rules to Artificial Neural Networks

Evaluation of the membership degrees:



Evaluation of the rules (rule activity):

$$\mu_l = \mathbb{R}^n \to [0,1]^r, \quad \underline{x} \Rightarrow \prod_{j=1}^{D_l} \mu_{c,s}^{(j)}(x_i)$$

Accumulation of rule inputs and normalization:

$$\mathrm{NF} : \mathbb{R}^n \to \mathbb{R}, \quad \underline{x} \Rightarrow \sum_{l=1}^{r} w_l \frac{k_l \mu_l(\underline{x})}{\sum_{j=1}^{r} k_j \mu_j(\underline{x})}$$

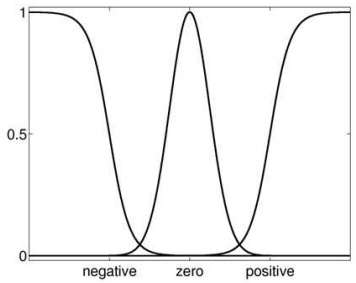
Dimension Reduction of the Weight Space

Semantics-Preserving Learning Algorithm

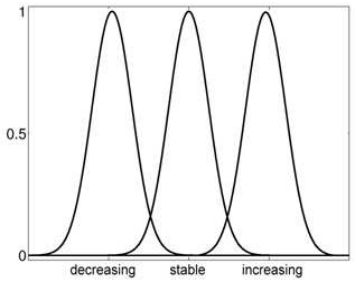
Membership functions of different inputs share their parameters, *e.g.*

$$\mu_{\text{DAX}}^{\text{stabil}} = \mu_{\text{C-DAX}}^{\text{stabil}}.$$

Membership functions of same input variable are not allowed to pass each other, they must keep their original order, *e.g.*



$$\mu_{\text{decreasing}} < \mu_{\text{stable}} < \mu_{\text{increasing}}$$



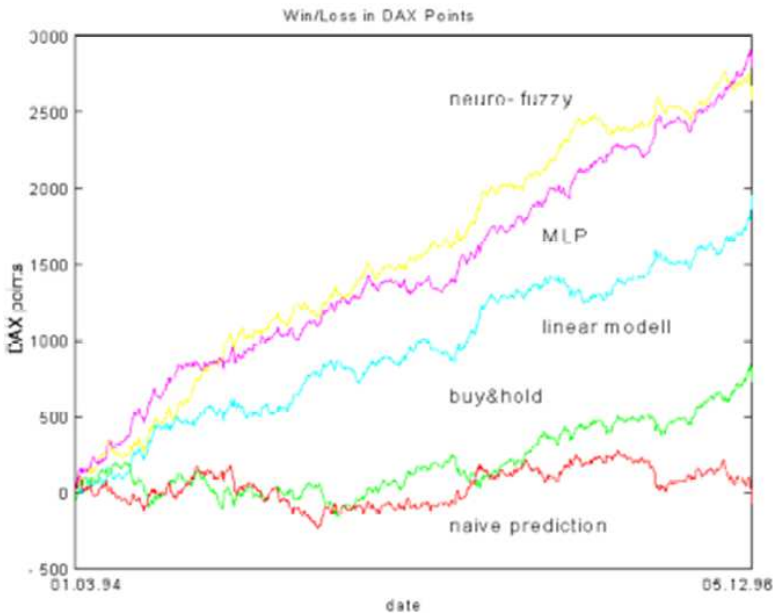
Benefits:

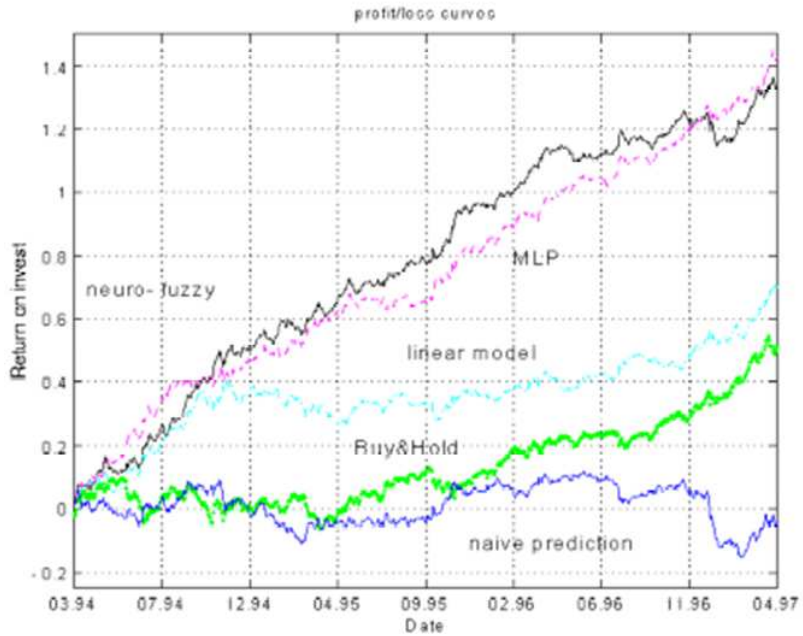
- The optimized rule base can still be interpreted.
- The number of free parameters has been reduced.

Return-on-Investment Curves

Different models have been evaluated.

Validation data: from 1 March 1994 until April 1997.





Summary

Neuro-fuzzy systems can be very useful for knowledge discovery.

The interpretability enables plausibility checks and improves the acceptance.

(Neuro-)fuzzy systems exploit the tolerance for sub-optimal solutions.

Neuro-fuzzy learning algorithms must observe constraints not to jeopardize the semantics of the model.

There is no automatic model creator! The user must **work** with the tool!

Such simple learning techniques support an explorative data analysis.

Literature about Neuro-Fuzzy Systems

Borgelt, C., Klawonn, F., Kruse, R., and Nauck, D. (2003).
Neuro-Fuzzy-Systeme: Von den Grundlagen künstlicher Neuronaler Netze zur Kopplung mit Fuzzy-Systemen.
Vieweg, Wiesbaden, Germany, 3rd edition.

Jang, J.-S. R. (1993).
Anfis: Adaptive-network-based fuzzy inference system.
IEEE Transactions on Systems, Man, and Cybernetics, 23(3):665–685.

Kruse, R., Borgelt, C., Klawonn, F., Moewes, C., Steinbrecher, M., and Held, P. (2013).
Computational Intelligence: A Methodological Introduction.
Texts in Computer Science. Springer, New York.

Kruse, R., Siekmann, S., Neuneier, R., and Zimmermann, H. G. (1998).
Neuro-fuzzy methods in finance applied to the german stock index DAX.
In Bol, G., Nakhaeizadeh, G., and Vollmer, K.-H., editors, *Risk Measurement, Econometrics and Neural Networks*, Contributions to Economics, pages 69–82.
Physica-Verlag.

Nauck, D., Klawonn, F., and Kruse, R. (1997).
Foundations of Neuro-Fuzzy Systems.
John Wiley & Sons Ltd, Chichester, United Kingdom.

Nauck, D. and Kruse, R. (1997).
A neuro-fuzzy method to learn fuzzy classification rules from data.
Fuzzy Sets and Systems, 89(3):277–288.