OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

INF FAKULTÄT FÜR INFORMATIK

Fuzzy Systems

Learning Fuzzy Systems

Prof. Dr. Rudolf Kruse **Alexander Dockhorn**

`{kruse,dockhorn}@ovgu.de`

Otto-von-Guericke University of Magdeburg

Faculty of Computer Science

Institute of Intelligent Cooperating Systems

Example: Automatic Gear Box

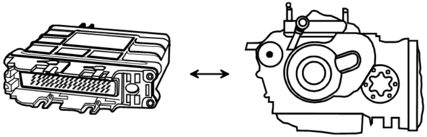
7 Mamdani fuzzy rules

Optimized program

- 24 byte RAM
- 702 byte ROM

Runtime 80 ms

- 12 times per second new sport factor is assigned



How to find these fuzzy rules?

Learning from Examples

There are lots of methods for learning:

In Statistics:

Parameter fitting, structure identification, model selection

Machine learning:

Deep learning

Neural networks: supervised learning with backpropagation

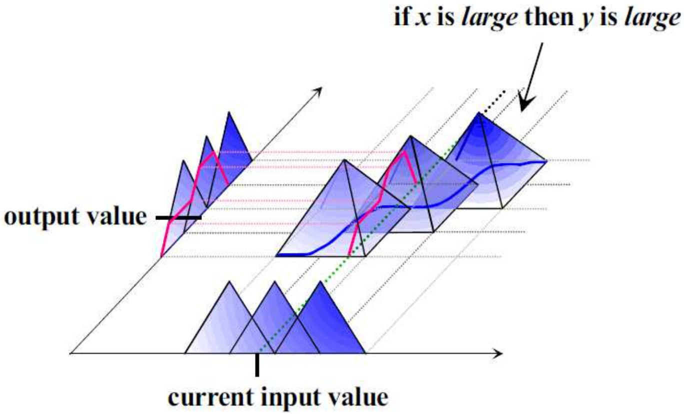
Cluster analysis: unsupervised learning

The learning problem becomes an optimization problem.

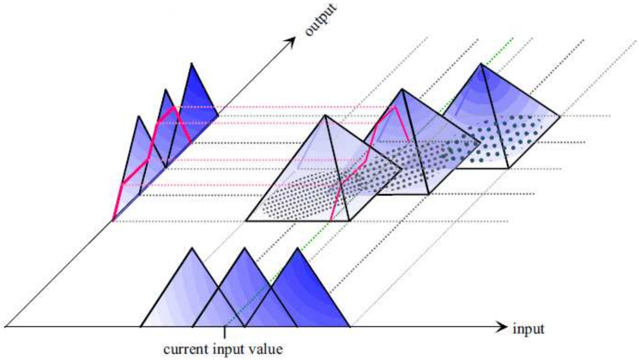
How to use these methods in fuzzy systems for learning?

Learning Fuzzy Systems by Clustering

Function Approximation with Fuzzy Rules



Learning Fuzzy Rules from Data



Perform fuzzy cluster analysis of the input-output data.

Then project the clusters.

Finally, obtain fuzzy rules, *e.g.* “if x is small, then y is medium”.

Example: Transfer Passenger Analysis

German Aerospace Center (DLR) developed macroscopic passenger flow model for simulating passenger movements on airport's land side

For passenger movements in terminal areas: distribution functions are used today

Goal: build fuzzy rule base describing transfer passenger amount between aircrafts

These rules can be used to improve macroscopic simulation

Idea: find rules based on probabilistic fuzzy *c*-means (FCM)

Attributes for Passenger Analysis

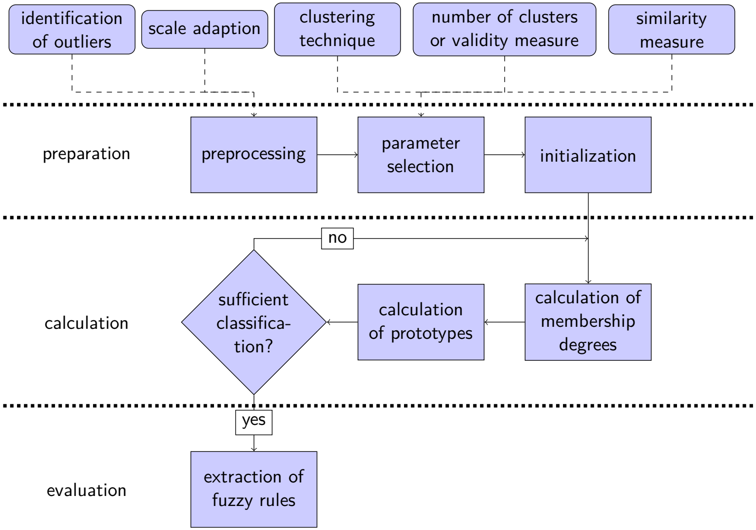
Maximal amount of passengers in certain aircraft (depending on type of aircraft)

Distance between airport of departure and airport of destination (in three categories: short-, medium-, and long-haul)

Time of departure

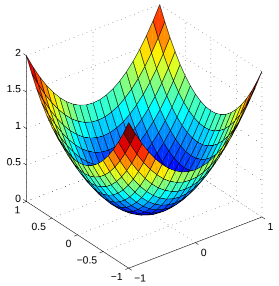
Percentage of transfer passengers in aircraft

General Clustering Procedure

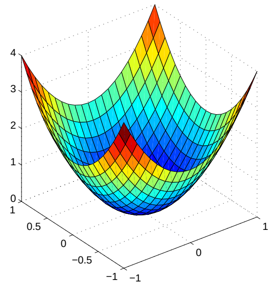

identification of outliers
scale adaption
clustering technique
number of clusters or validity measure
similarity measure
preparation
preprocessing
parameter selection
initialization
no
calculation
sufficient classification?
calculation of prototypes
calculation of membership degrees
yes
evaluation
extraction of fuzzy rules

Distance Measure

distance between $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{c} = (0, 0)$



$$d^2(\boldsymbol{c}, \boldsymbol{x}) = \|\boldsymbol{c} - \boldsymbol{x}\|^2$$



$$d^2_\tau(\boldsymbol{c}, \boldsymbol{x}) = \frac{1}{\tau^p} \|\boldsymbol{c} - \boldsymbol{x}\|^2$$

Distance Measure with Size Adaption

$$d_{ij}^2 = \frac{1}{\tau_i^p} \cdot \|\boldsymbol{c}_i - \boldsymbol{x}_j\|^2$$

$$\boldsymbol{c}_i = \frac{\sum_{j=1}^n u_{ij}^m \boldsymbol{x}_j}{\sum_{j=1}^n u_{ij}^m}$$

$$\tau_i = \frac{\left(\sum_{j=1}^n u_{ij}^m d_{ij}^2\right)^{\frac{1}{p+1}}}{\sum_{k=1}^c \left(\sum_{j=1}^n u_{kj}^m d_{kj}^2\right)^{\frac{1}{p+1}}} \cdot \tau$$

$$\tau = \sum_{i=1}^c \tau_i$$

p determines emphasis put on size adaption during clustering

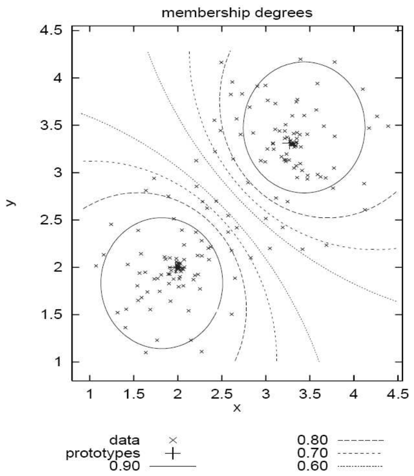
Constraints for the Objective function

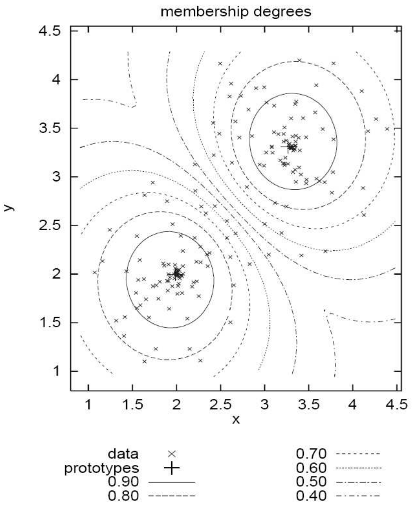
Probabilistic clustering

Noise clustering

Influence of outliers

Probabilistic and Noise Clustering





Influence of Outliers

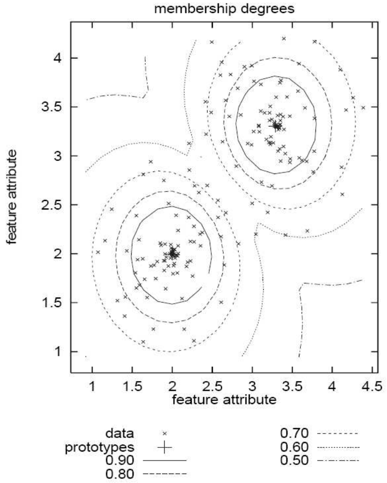
A weighting factor ω_j is attached to each datum $\boldsymbol{x}_j$

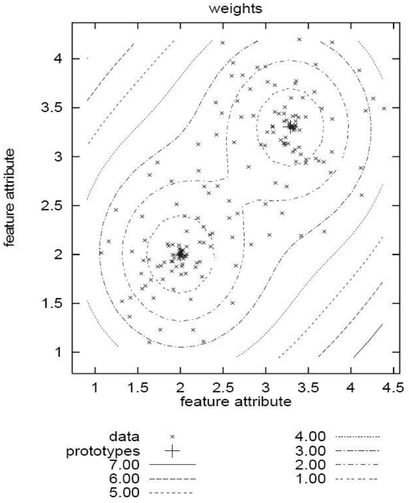
Weighting factors are adapted during clustering

Using concept of weighting factors:

- outliers in data set can be identified and
- outliers' influence on partition is reduced

Membership Degrees and Weighting Factors





Influence of Outliers

Minimize objective function

$$J(X, U, C) = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{m} \cdot \frac{1}{\omega_j^q} \cdot d_{ij}^2$$

subject to

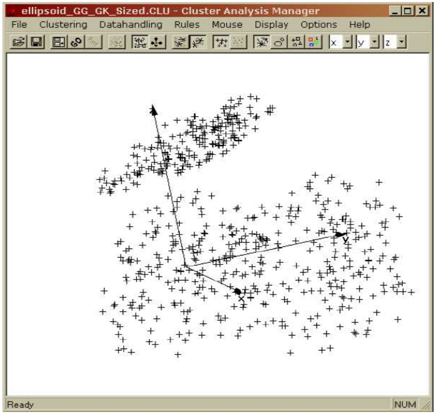
$$\forall j \in [n] : \sum_{i=1}^{c} u_{ij} = 1, \quad \forall i \in [c] : \sum_{j=1}^{n} u_{ij} > 0, \quad \sum_{j=1}^{n} \omega_j = \omega$$

q determines emphasis put on weight adaption during clustering

Update equations for memberships and weights, resp.

$$u_{ij} = \frac{d_{ij}^{\frac{2}{1-m}}}{\sum_{k=1}^{c} d_{kj}^{\frac{2}{1-m}}}, \qquad \omega_j = \frac{\left(\sum_{i=1}^{c} u_{ij}^{m} d_{ij}^2\right)^{\frac{1}{q+1}}}{\sum_{k=1}^{n} \left(\sum_{i=1}^{c} u_{ik}^{m} d_{ik}^2\right)^{\frac{1}{q+1}}} \cdot \omega$$

Determining the Number of Clusters


ellipsoid_GG_GK_Sized.CLU - Cluster Analysis Manager
File Clustering Datahandling Rules Mouse Display Options Help
x y z
Ready
NUM

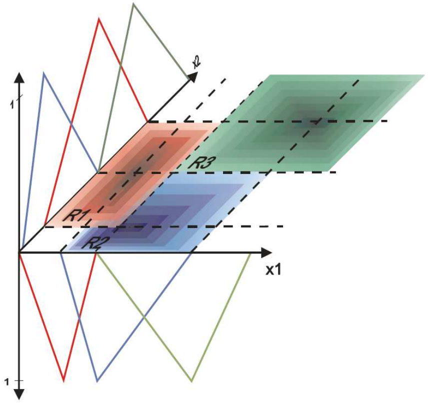
Here, validity measures evaluating whole partition of data

Getting: global validity measures

Clustering is run for varying number of clusters

Validity of resulting partitions is compared

Fuzzy Rules and Induced Vague Areas

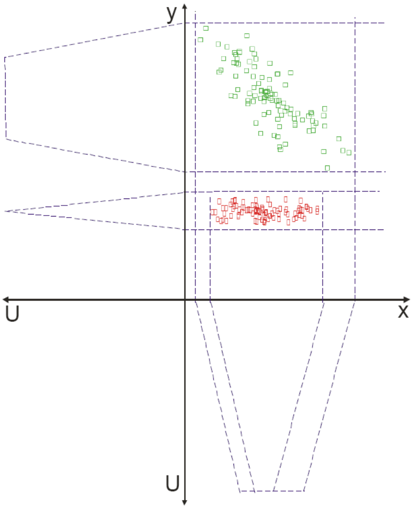


Intensity of color indicates firing strength of specific rule

Vague areas = fuzzy clusters where color intensity indicates membership degree

Tips of fuzzy partitions in single domains = projections of multidimensional cluster centers

Simplification of Fuzzy Rules



Similar fuzzy sets are combined to one fuzzy set

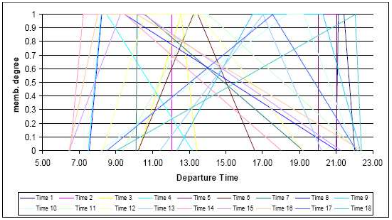
Fuzzy sets similar to universal fuzzy set are removed

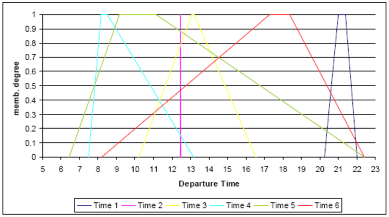
Rules with same input sets are

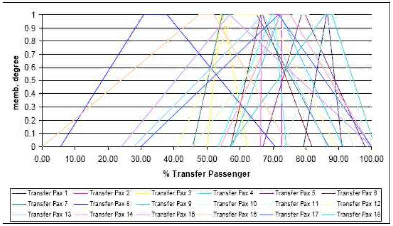
- Combined if they also have same output set(s) or
- Otherwise removed from rule set

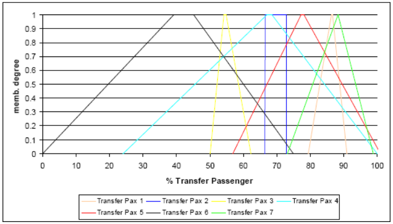
Results

FCM with $c = 18$, outlier and size adaptation, Euclidean distance:









resulting fuzzy sets simplified fuzzy sets

Evaluation of the Rule Base

rule	max. no. of pax	De st.	depart.	% transfer pax
1	paxmax1	R1	time1	tpax1
2	paxmax2	R1	time2	tpax2
3	paxmax3	R1	time3	tpax3
4	paxmax4	R1	time4	tpax4
5	paxmax5	R5	time1	tpax5
...	...	...	...	...

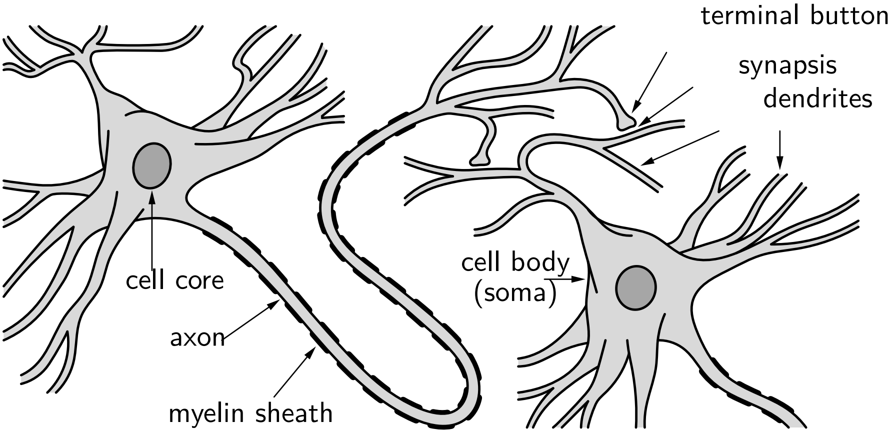
rules 1 and 5: aircraft with relatively small amount of maximal passengers (80-200), short- to medium-haul destination, and departing late at night usually have high amount of transfer passengers (80-90%)

rule 2: flights with medium-haul destination and small aircraft (about 150 passengers), starting about noon, carry relatively high amount of transfer passengers (ca. 70%)

Learning Fuzzy Systems by using Neural Networks

Biological Background

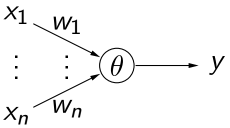
Structure of a prototypical biological neuron



Threshold Logic Units

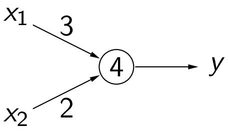
A **Threshold Logic Unit (TLU)** is a processing unit for numbers with n inputs $x_1, \ldots, x_n$ and one output y. The unit has a **threshold** θ and each input x_i is associated with a **weight** w_i. A threshold logic unit computes the function

$$y = \begin{cases} 1, & \text{if} \quad \mathbf{x}\mathbf{w} = \sum_{i=1}^{n} w_i x_i \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$$



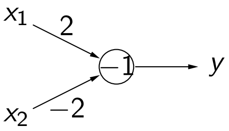
Threshold Logic Units: Examples

Threshold logic unit for the conjunction $x_1 \wedge x_2$.



x_1	x_2	$3x_1 + 2x_2$	y
0	0	0	0
1	0	3	0
0	1	2	0
1	1	5	1

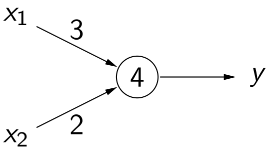
Threshold logic unit for the implication $x_2 \rightarrow x_1$.

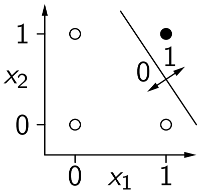


x_1	x_2	$2x_1 - 2x_2$	y
0	0	0	1
1	0	2	1
0	1	-2	0
1	1	0	1

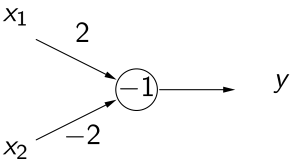
Threshold Logic Units: Geometric Interpretation

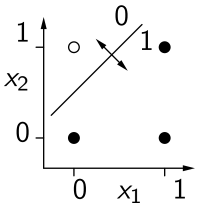
Threshold logic unit for $x_1 \wedge x_2$.





A threshold logic unit for $x_2 \rightarrow x_1$.





Threshold Logic Units: Linear Separability

Two point sets in an-dimensional space are called **linear separable** if and only if they can be separated by an (n-1)-dimensional hyperplane.

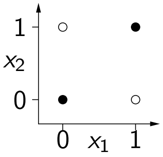
Points of one set are allowed to lie exactly on the hyperplane.

A boolean function is called linear separable iff the set of the preimages of 0 and the set of the preimages of 1 are linear separable.

Threshold Logic Units: Limitations

The biimplication problem $x_1 \leftrightarrow x_2$: There is no separating line.

x_1	x_2	y
0	0	1
1	0	0
0	1	0
1	1	1



Formal proof by *reductio ad absurdum*:

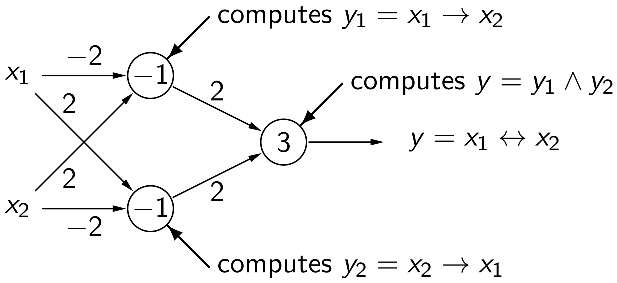
$$\begin{array}{lrcl}
\text{since } (0,0) \mapsto 1: & 0 & \geq \theta, & (1)\\
\text{since } (1,0) \mapsto 0: & w_1 & < \theta, & (2)\\
\text{since } (0,1) \mapsto 0: & w_2 & < \theta, & (3)\\
\text{since } (1,1) \mapsto 1: & w_1 + w_2 & \geq \theta. & (4)
\end{array}$$

(2) and (3):$w_1 + w_2 < 2\theta$. With (4): $2\theta > \theta$, or $\theta > 0$.Contradiction to (1).

Networks of Threshold Logic Units

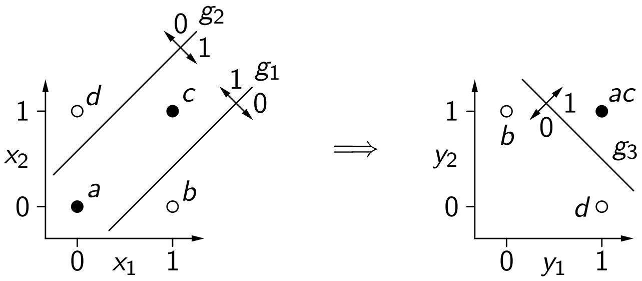
Solving the biimplication problem with a network.

Idea: logical decomposition $\quad x_1 \leftrightarrow x_2 \quad \equiv \quad (x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_1)$


computes $y_1 = x_1 \rightarrow x_2$
x_1
-2
-1
2
2
computes $y = y_1 \wedge y_2$
3
$y = x_1 \leftrightarrow x_2$
2
2
x_2
-1
-2
computes $y_2 = x_2 \rightarrow x_1$

Networks of Threshold Logic Units

Solving the biimplication problem: Geometric interpretation



The first layer computes new Boolean coordinates for the points.

After the coordinate transformation the problem is linearly separable.

Backpropagation

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.

Object Recognition in Pictures

Imagenet Large Scale Visual Recognition Challenge (LSVRC) since 2010

Look for 200 classes (chair, table, person, bike, ...)

Pictures with ca. 500 x 400 pxels, 3 color channels

Neural network with ca. 600.000 neurons in the first layer

200 neurons in the last layer

Winner 2015: ResNet, uses more than 150 layers, classification error 3.6%

Neuro-Fuzzy Systems

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

$$\begin{aligned} R_i: \quad & \text{If } x_1 \text{ is } \mu_i^{(1)} \text{ and } \dots \text{ and } x_n \text{ is } \mu_i^{(n)} \\ & \text{then } y \text{ is } \mu_i, \end{aligned}$$

or the fuzzy rules R'_i of a TSK controller

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

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

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

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.

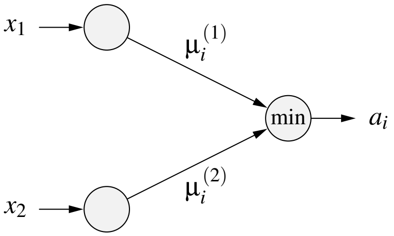
To calculate the rule activation of u_i, we must modify their network input functions.

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.

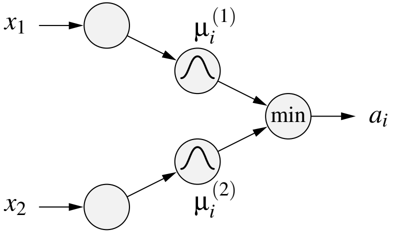
Fuzzy Sets as Weights



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 and NEFCLASS.

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

The ANFIS Model

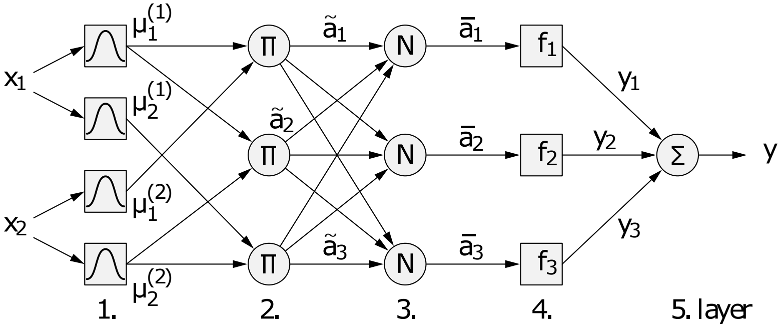
The neuro-fuzzy system *ANFIS* (Adaptive-Network-based Fuzzy Inference System)

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:

$$\begin{aligned} R_1 &: \quad \text{If } x_1 \text{ is } A_1 \text{ and } x_2 \text{ is } B_1 \text{ then } y = f_1(x_1, x_2) \\ R_2 &: \quad \text{If } x_1 \text{ is } A_1 \text{ and } x_2 \text{ is } B_2 \text{ then } y = f_2(x_1, x_2) \\ R_3 &: \quad \text{If } x_1 \text{ is } A_2 \text{ and } x_2 \text{ is } B_2 \text{ then } y = f_3(x_1, x_2) \end{aligned}$$

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

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

$$\mathbf{y} = \bar{\mathbf{a}}\mathbf{R}\mathbf{X}$$

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 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.

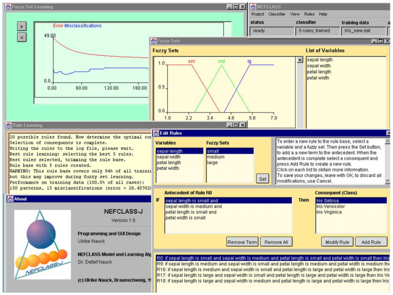
Applying NEFCLASS-J

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.



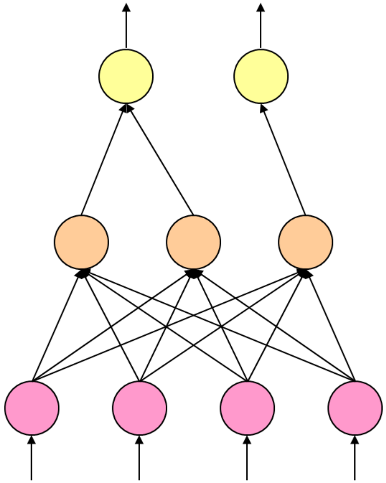
```
20 possible rules found. Now determine the optimal cor
Selection of consequents is complete.
Writing the rules to the log file, please wait.
Best rule learning: selecting the best 5 rules.
Best rules selected, trimming the rule base.
Rule base with 5 rules created.
WARNING: This rule base covers only 94% of all trainir
but this may improve during fuzzy set learning.
Performance on training data (100.0% of all cases):
150 patterns, 15 misclassifications (error = 26.457628
```

To enter a new rule to the rule base, select a variable and a fuzzy set. Then press the Set button, to add a a new term to the antecedent. When the antecedent is complete select a consequent and press Add Rule to create a new rule.
Click on each list to obtain more information.
To save your changes, leave with OK, to discard all modifications, use Cancel.

sepal length is small and
sepal width is medium and
petal length is small and
petal width is small

R0: if sepal length is small and sepal width is medium and petal length is small and petal width is small then Iris
R9: if sepal length is medium and sepal width is small and petal length is medium and petal width is medium the
R16: if sepal length is medium and sepal width is small and petal length is large and petal width is large then Iris
R17: if sepal length is large and sepal width is small and petal length is large and petal width is large then Iris Vir
R18: if sepal length is large and sepal width is medium and petal length is large and petal width is large then Iris

NEFCLASS: Neuro-Fuzzy Classifier



output variables

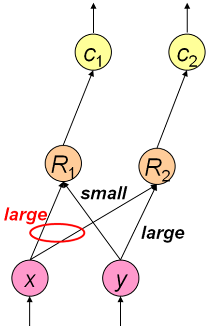
unweighted connections

fuzzy rules

fuzzy sets (antecedents)

input attributes (variables)

Representation of Fuzzy Rules



Example: 2 rules

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

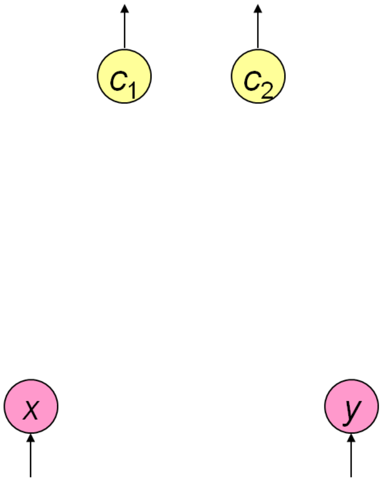
R_2 : if x is *large* and y is *large*, then class is c_2

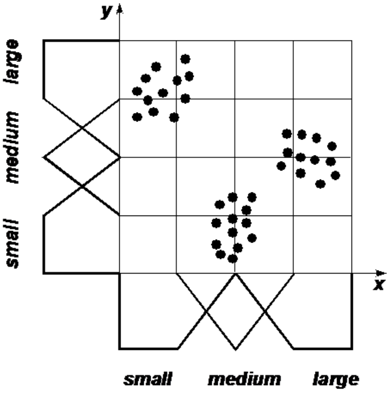
Connections $x \rightarrow R_1$ and $x \rightarrow R_2$ are linked.

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

1. Training Step: Initialization

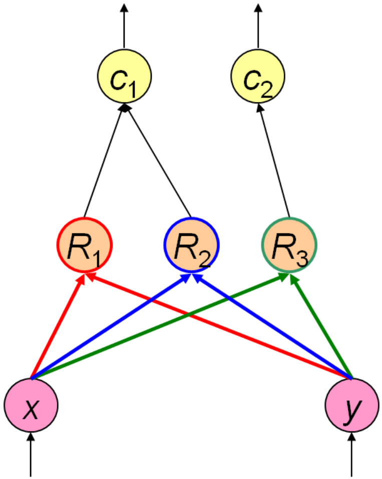
Specify initial fuzzy partitions for all input variables.

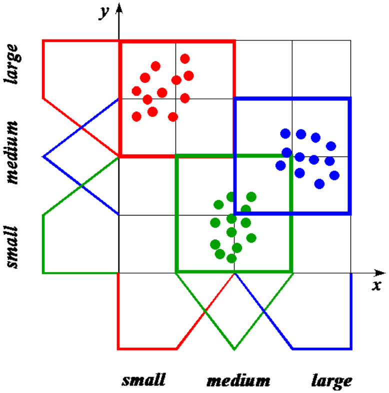




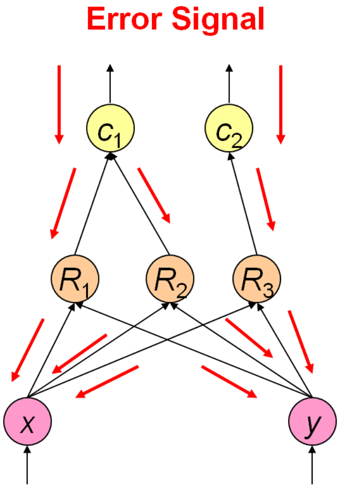
3. Training Step: Rule Base Induction

NEFCLASS uses modified Wang-Mendel procedure.

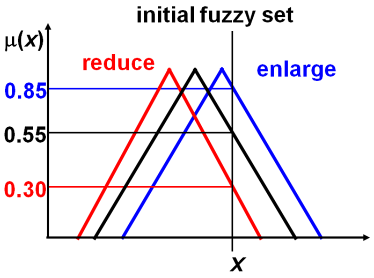


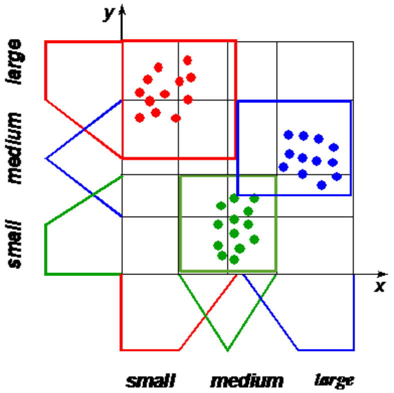


3. Improving Rules by Backpropagation of the Fuzzy Error Signal



4. Improving Fuzzy Sets by Heuristics

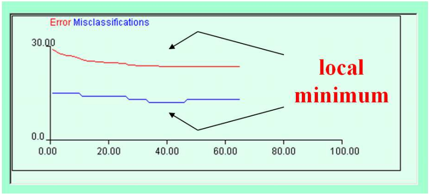




Heuristics: A fuzzy set is moved away from x (towards x) and its support is reduced (enlarged) in order to reduce (enlarge) the degree of membership of x.

4. Improving Fuzzy Sets by Heuristics

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



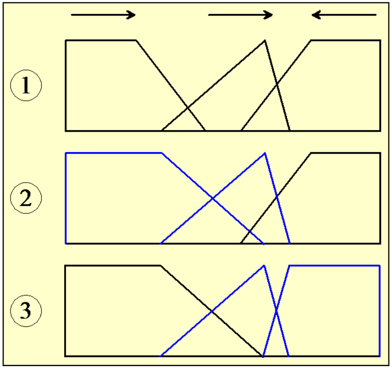
observing the error on validation set

variations:

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

Constraints for Training Fuzzy Sets

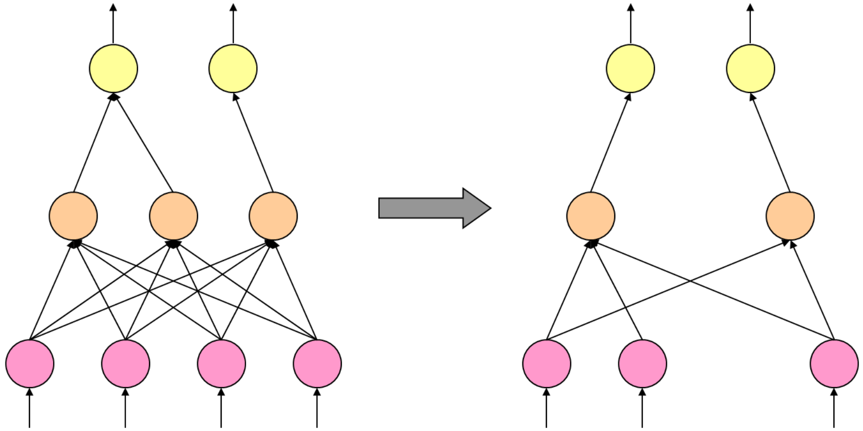
- valid parameter values
- non-empty intersection of adjacent fuzzy sets
- keep relative positions
- maintain symmetry
- complete coverage (degrees of membership add up to 1 for each element)



Correcting a partition after modifying the parameters

5. Training Step: Pruning

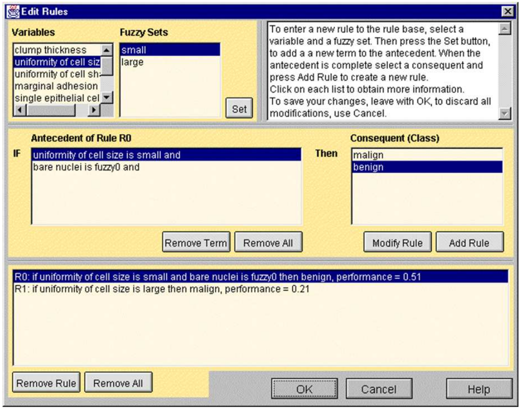
Goal: Remove variables, rules, and fuzzy sets in order to improve the interpretability and generalization.



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*



To enter a new rule to the rule base, select a variable and a fuzzy set. Then press the Set button, to add a a new term to the antecedent. When the antecedent is complete select a consequent and press Add Rule to create a new rule.
Click on each list to obtain more information.
To save your changes, leave with OK, to discard all modifications, use Cancel.

uniformity of cell size is small and
bare nuclei is fuzzy0 and

R0: if uniformity of cell size is small and bare nuclei is fuzzy0 then benign, performance = 0.51
R1: if uniformity of cell size is large then malign, performance = 0.21

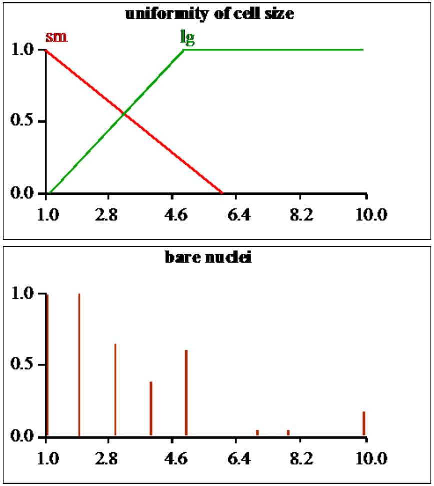
Example: WBC Classification Performance

	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



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.

Optimization of Fuzzy Systems with Evolutionary Algorithms

Evolutionary Algorithms

. . . are based on *biological theory of evolution*.

Fundamental principle:

- Beneficial properties created by random variation are chosen by natural selection.
- Differential reproduction: Individuals with beneficial properties have better chances to reproduce themselves.

The theory of evolution describes the diversity and complexity of species.

It allows to unify all disciplines of biology.

Fundamental Terms and their Meanings I

Term	Biology	Computer Science
individual	living creature	candidate solution
chromosome	DNA histone protein string	character string
	defines "blueprint" and (partly) properties of individual in coded form	
	usually several chromosomes per individual	usually one chromosome per individual
gene	part of chromosome	one character
	fundamental unit of inheritance that (partly) defines property of an individual	
allele (allomorph)	characteristic of a gene	value of a character
	there is only one characteristic of a gene per chromosome	
locus	position of a gene	position of a character
	exactly just one gene on every position in chromosome	

Fundamental Terms and their Meanings II

Term	**Biology**	**Computer Science**
phenotype	outer appearance of a living creature	implementation of a candidate solution
genotype	genetic constitution of a living creature	encoding of a candidate solution
population	all living creatures	family/multiset of chromosomes
generation	population at given time	population at given time
reproduction	creation of offspring from one or two (if, then usually two) creatures	creation of (child) chromosomes from one or more (parent) chromosomes
fitness	capability/conformity of a living creature	capability/goodness of a candidate solution
	determines chances of both survival and reproduction	

Elements of an Evolutionary Algorithm I

An EA consists of

a **codebook** for the candidate solutions

- This codebook is problem-specific (*i.e.* no general rules exist).

a method to generate an **initial population**

- Usually, it generates random character strings.
- Depending on the encoding, sophisticated approaches can be necessary.

a **weighting function** (fitness function) for individuals

- It acts as an environment and indicates the goodness of the individuals.

a **technique of sampling** based on the fitness function

- It defines both which individuals are used to generate offspring and which ones reach the next generation without any change.

Elements of an Evolutionary Algorithm II

Furthermore, an EA consists of

genetic operators that change candidate solutions

- *mutation* (random modification of particular genes)
- *crossover* (recombination of chromosomes; actually "crossing over" due to process in meiosis (period of cell division) where chromosomes are divided and then joined crossed-over again)

values for several **parameters**

- *e.g.* population size, mutation probability, etc.

termination criterion, *e.g.*

- fixed number of generations has been processed
- for fixed number of generations, no improvement occurred
- predefined minimal goodness of solution has been reached

Generic Basic EA

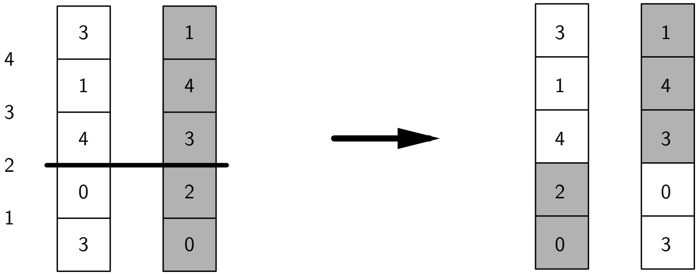
Algorithm 1 EA Schema

```
$t \leftarrow 0$
pop($t$) $\leftarrow$ create population of size $\mu$
evaluate pop($t$)
**while** termination criterion not fulfilled {
    pop$_1$ $\leftarrow$ select parents for $\lambda$ offspring from pop($t$)
    pop$_2$ $\leftarrow$ generate offspring by recombination from pop$_1$
    pop$_3$ $\leftarrow$ variate individuals in pop$_2$
    evaluate pop$_3$
    $t \leftarrow t + 1$
    pop($t$) $\leftarrow$ select $\mu$ individuals from pop$_3$ $\cup$ pop($t - 1$)
}
**return** best individual from pop($t$)
```

Genetic Operators: Crossover

Exchange of one part of chromosome (or subset of selected genes) between two individuals, *e.g.* so-called **one-point crossover**:

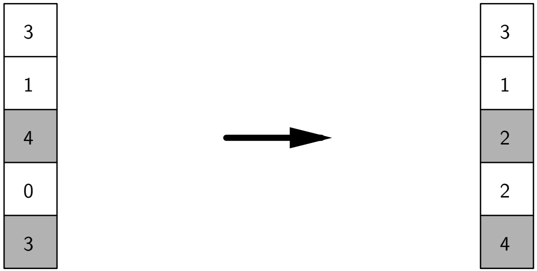
1. Choose a random point of division between two genes.
2. Exchange the gene sequences of one side of the division point.



Genetic Operators: Mutation

Randomly chosen genes are randomly replaced (alleles change).

The number of replaced genes maybe random, too (should be small).



Most mutations are damaging/harmful (they worsen the fitness).

Initially non-existent alleles can (only) be generated by mutation.

Optimization of Fuzzy Controllers with GAs

Mamdani controller can be induced/optimized as follows:

- rule base (which rules, which outputs)
- fuzzy sets/fuzzy partitions (shape, location, width, number of fuzzy sets
- t-norm or t-conorm for rule evaluation (rarely)
- parameters of defuzzification method (if present, rarely)
- inputs used in the rules (feature selection)

Here we talk about the optimization of the rule base and the fuzzy sets with a fixed choice of input values.

Rule evaluation: minimum and maximum

Defuzzification: center of gravity (COG)

Optimization: Three Possible Approaches

1. Rule base and fuzzy partitions are optimized simultaneously:
   - Disadvantage: Many parameters must be optimized at same time.
2. First, the fuzzy partitions are optimized *w.r.t.* given rule base, then rule base is optimized with best fuzzy partitions:
   - Disadvantage: Expert knowledge needed to create rule base (starting with random rule base is not promising).
3. First, rule base is optimized *w.r.t.* given fuzzy sets, then fuzzy partitions are optimized with best rule base:
   - Fuzzy sets may be distributed, *e.g.* equidistantly.
   - Here, the user must specify the number of fuzzy sets for input and output.
   - We only consider this approach here.

Optimization: Fitness Function

A good controller should exhibit several properties:

- The target value should be reached from any (initial) situation.
- The target value should be reached quickly.
- The target value should be reached with minimal effort (energy).

The controller is applied multiple times to target system:

- Here, simulation of inverted pendulum problem.
- Several randomly chosen starting points.
- A score is assigned to the controller according to its success (number of situations, duration of successful control, energy costs).

Here, the evaluation of individuals is by far the most expensive operation:

- Every individual must be put in control for at least certain number of time steps in order to yield reliable fitness score.

Assessing the Controller Success

If the deviation of the actual value to the target is big, it's a failure, *e.g.* inverted pendulum: It must stay within $[-90°, 90°]$.

After some time, the actual value should be close to the target value and remain within its proximity (range of tolerance). Otherwise the process is aborted also (failure).

The range of tolerance is shrinked during generations (towards target).

- During the first generations, it suffices if the pendulum doesn't topple over.
- Later the pendulum must stand upright within a shrinking angle interval.

The abs. values of adjusting values are added up as penalty value.

In balance, a fast switch betw. large/small forces effects the controller. Thus large forces need to be avoided.

Optimizing the Rule Base: Encoding

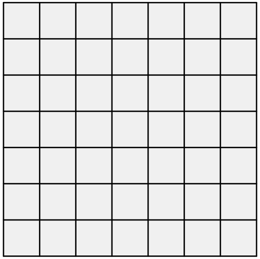
We only consider complete rule bases,
that is, one rule for every combination of input fuzzy sets.

For every combination of input fuzzy sets, we need to determine a linguistic term of the control variable (by filling a table).

Example rule base for inverted pendulum controller

$\dot{\theta}\backslash\theta$	nb	nm	ns	az	ps	pm	pb
pb	az	ps	ps	pb	pb	pb	pb
pm	ns	az	ps	pm	pm	pb	pb
ps	nm	ns	az	ps	pm	pm	pb
az	nb	nm	ns	az	ps	pm	pb
ns	nb	nm	nm	ns	az	ps	pm
nm	nb	nb	nm	nm	ns	az	ps
nb	nb	nb	nb	nb	ns	ns	az

schematic



Representing the Rule Base as Chromosome

Linearization (transformation into a vector):

The table is traversed in an arbitrary but fixed order.

Entries are listed in a vector, *e.g.*

- listing row by row.
- So, neighboring relations between cells are lost.

Adjacent entries should contain similar linguistic terms (this is important, *e.g.* during crossover).

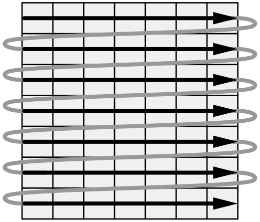


Table (direct usage of scheme)

for two- or higher dimensional chromosomes.

Thus special genetic operators are needed.

Genetic Operators for Rule Base: Mutation

1. The rule/table entry is chosen randomly.
2. The linguistic term of output is altered randomly.
3. Multiple table entries may be altered.

It might be beneficial to restrict the mutation of the rule base:

An entry is only changed to a linguistic term similar to the current one, *e.g.*

- "positive small" $\mapsto$ "approximately zero" or "positive medium",
- "negative big" $\mapsto$ "negative medium" or "negative small".

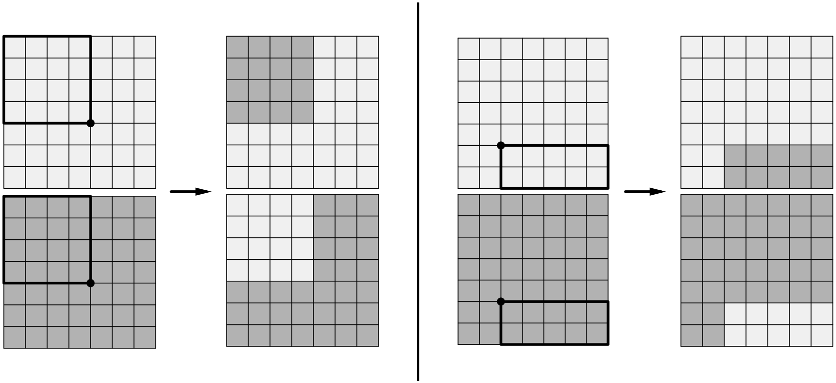
This prevents a too fast "depletion" of the collected information.

Also, the rule bases are only modified "carefully".

Genetic Operators: One-Point Crossover

Choose both interior grid point and corner randomly.

This defines a subtable that will be exchanged between two parents.



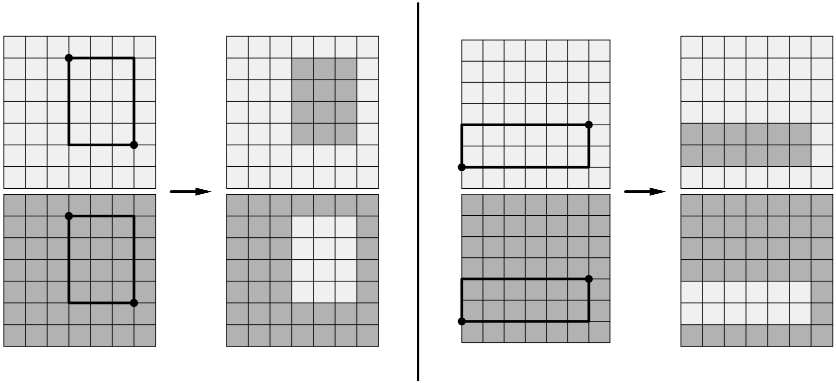
The crossover should exhibit a location-dependent bias.

So, it prefers to inherit adjacent rules.

Genetic Operators: Two-Point Crossover

Choose two grid points randomly (border points allowed).

This defines a subtable that will be exchanged between two parents.



Partial solutions can be exchanged in a more flexible way.

The two-point crossover performs better than one-point crossover.

Optimizing Fuzzy Sets

Given: optimized rules with fixed and unchanged equidistant fuzzy sets.

Goal: further improvement of rule behavior by adjusting fuzzy sets with fixed rule base (“fine tuning”).

Encoding fuzzy sets (first possibility):

1. Choose the shape of the fuzzy sets (*e.g.* triangle, trapezoid, Gaussian, parabola, spline, etc.).
2. List the defining parameters of the fuzzy sets (*e.g.* triangle: left border, center, right border).

Example: controller with triangular fuzzy sets (excerpt)

...		nm			ns			az			ps		...
...	-45	-30	-15	-30	-15	0	-15	0	15	0	15	30	...

Optimizing Fuzzy Sets: Disadvantages

In advance one must specify that, *e.g.* triangles or trapezoids are used.

So, the encoding is quite “rigid” *w.r.t.* the shape of the fuzzy sets.

Genetic operators can violate the order of parameters, *e.g.*

- considering triangles $\mu_{l,m,r}$, $l \leq m \leq r$ must hold.

A possible “overtaking” between fuzzy sets, *i.e.*

- the meaningful order of the fuzzy sets may be destroyed by mutation/crossover, *e.g.* it should hold true that *ns* lies left of *ps*.

The condition $\forall x : \sum_i \mu_i(x) = 1$ might be violated:

- This can be treated by representing identical parameters only once.

...	-45	-15	15	-15	15	30	15	30	45	30	45	60	...
...	-45	-30	-20	-30	-20	-10	-20	-10	0	-10	10	30	...

Encoding Fuzzy Partitions

Distinct membership degrees of fuzzy sets are specified for a predefined and equidistant set of sampling points:

$$\underbrace{\overbrace{\begin{pmatrix} \mu_1(x_1) \\ \vdots \\ \mu_n(x_1) \end{pmatrix}}^{\text{Gene 1}} \cdots \overbrace{\begin{pmatrix} \mu_1(x_i) \\ \vdots \\ \mu_n(x_i) \end{pmatrix}}^{\text{Gene i}} \cdots \overbrace{\begin{pmatrix} \mu_1(x_m) \\ \vdots \\ \mu_n(x_m) \end{pmatrix}}^{\text{Gene m}}}_{\text{Encoding with } m \times n \text{ numbers} \in [0,1]}$$

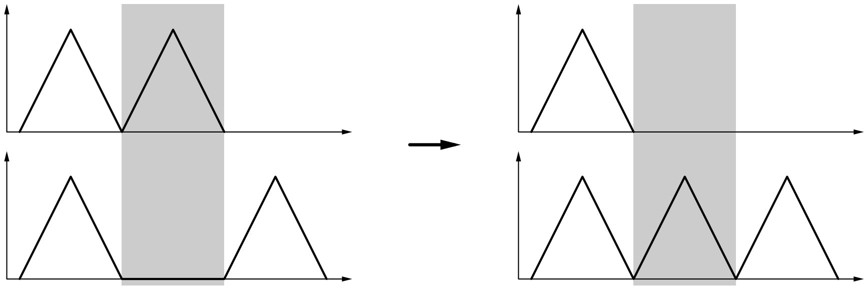
pb	0	0	0	0	0	0	0	0	0	0	0	0	0	.5	1	1	1
pm	0	0	0	0	0	0	0	0	0	0	0	.5	1	.5	0	0	0
ps	0	0	0	0	0	0	0	0	0	.5	1	.5	0	0	0	0	0
az	0	0	0	0	0	0	0	.5	1	.5	0	0	0	0	0	0	0
ns	0	0	0	0	0	.5	1	.5	0	0	0	0	0	0	0	0	0
nm	0	0	0	.5	1	.5	0	0	0	0	0	0	0	0	0	0	0
nb	1	1	1	.5	0	0	0	0	0	0	0	0	0	0	0	0	0
	-60		-45		-30		-15		0		15		30		45		60

Genetic Operators

Mutation (analogous to standard mutation):

- A randomly chosen entry is altered randomly.
- It is reasonable to restrict the magnitude of alteration, *e.g.* by specifying an interval or by a normal distribution.

Crossover (basic one-/two-point crossover):
Note that fuzzy sets may be erased by crossover.

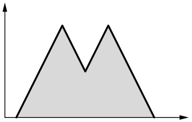


Repairing Fuzzy Sets

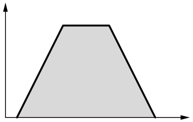
Mutation/crossover may violate the *unimodality* of the fuzzy set,

- *i.e.* the membership function may have ≥ 1 local maximum.
- Multimodal fuzzy sets are harder to interpret than unimodal ones.

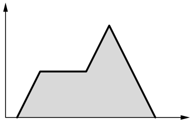
Therefore, fuzzy sets are repaired (made unimodal).



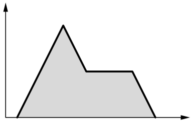
bimodal fuzzy set



join the maxima



cut off left maximum



cut of right max.

Fuzzy sets may be widened or cut off such that the entire domain is covered, but without having too many fuzzy sets covering the same areas.

Summary

Optimization in two steps:

1. Optimize the rule base with fixed fuzzy partitions.
2. Optimize the fuzzy partitions with an induced rule base.

It was possible to generate a working rule set for the inverted pendulum problem with evolutionary algorithms.

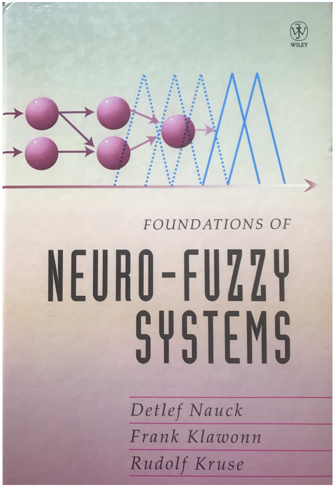
The approach of inducing a rule base is quite expensive in time.

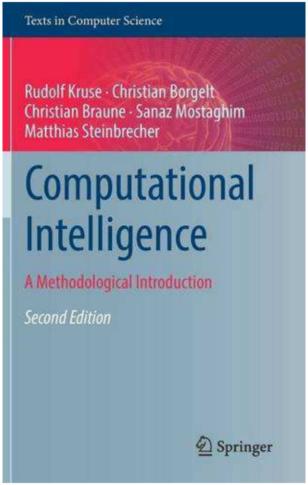
However, it does not need any background knowledge.

Additional requirements might be considered for fitness:

- *compactness* (small number of rules and fuzzy sets)
- *completeness* (coverage of relevant areas of input space)
- *consistency* (no rules with similar antecedents and different consequents)
- *interpretability* (limited overlapping of fuzzy sets)

References


Foundations of Neuro-Fuzzy Systems, Detlef Nauck, Frank Klawonn, Rudolf Kruse, Wiley


Texts in Computer Science. Rudolf Kruse · Christian Borgelt · Christian Braune · Sanaz Mostaghim · Matthias Steinbrecher. Computational Intelligence: A Methodological Introduction. Second Edition. Springer