Fuzzy Systems
Learning Fuzzy Systems

Prof. Dr. Rudolf Kruse  Jonas Schulze
{rudolf.kruse,jonas.schulze}@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

If $x$ is large then $y$ is large

Output value

Current input value
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

Calculation:
- Sufficient classification?
  - No
  - Calculation of prototypes
    - Calculation of membership degrees
- Yes
  - Extraction of fuzzy rules

Evaluation:
Distance Measure

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

\[
d^2(\mathbf{c}, \mathbf{x}) = \|\mathbf{c} - \mathbf{x}\|^2
\]

\[
d^2_T(\mathbf{c}, \mathbf{x}) = \frac{1}{\tau^p} \|\mathbf{c} - \mathbf{x}\|^2
\]
Distance Measure with Size Adaption

\[ d_{ij}^2 = \frac{1}{\tau_i^p} \cdot \| c_i - x_j \|^2 \]

\[ c_i = \frac{\sum_{j=1}^{n} u_{ij}^m x_j}{\sum_{j=1}^{n} u_{ij}^m} \]

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

\[ \tau = \sum_{i=1}^{c} \tau_i \]

\( p \) 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 $\omega_j$ is attached to each datum $x_j$

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

![Membership Degrees and Weighting Factors](image-url)
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.

\[ u_{ij} = \frac{d_{ij}^{1-m}}{\sum_{k=1}^{c} d_{kj}^{1-m}}, \quad \omega_j = \frac{\left( \sum_{i=1}^{c} u_{ij}^m d_{ij}^2 \right)^{\frac{1}{q+1}}}{\sum_{k=1}^{n} \left( \sum_{i=1}^{c} u_{ik}^m d_{ik}^2 \right)^{\frac{1}{q+1}}} \cdot \omega \]
Determining the Number of Clusters

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

<table>
<thead>
<tr>
<th>rule</th>
<th>max. no. of pax</th>
<th>De st.</th>
<th>depart.</th>
<th>% transfer pax</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>paxmax1</td>
<td>R1</td>
<td>time1</td>
<td>tpax1</td>
</tr>
<tr>
<td>2</td>
<td>paxmax2</td>
<td>R1</td>
<td>time2</td>
<td>tpax2</td>
</tr>
<tr>
<td>3</td>
<td>paxmax3</td>
<td>R1</td>
<td>time3</td>
<td>tpax3</td>
</tr>
<tr>
<td>4</td>
<td>paxmax4</td>
<td>R1</td>
<td>time4</td>
<td>tpax4</td>
</tr>
<tr>
<td>5</td>
<td>paxmax5</td>
<td>R5</td>
<td>time1</td>
<td>tpax5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**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

cell core
axon
myelin sheath
cell body (soma)
terminal button
synapsis
dendrites
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** \( \theta \) and each input \( x_i \) is associated with a **weight** \( w_i \). A threshold logic unit computes the function

\[
y = \begin{cases} 
1, & \text{if } xw = \sum_{i=1}^{n} w_i x_i \geq \theta, \\
0, & \text{otherwise.}
\end{cases}
\]

![Diagram of a TLU](image)
Threshold Logic Units: Examples

Threshold logic unit for the conjunction \( x_1 \land x_2 \).

Threshold logic unit for the implication \( x_2 \rightarrow x_1 \).
Threshold Logic Units: Geometric Interpretation

Threshold logic unit for $x_1 \land x_2$.

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

R. Kruse, J. Schulze  
FS – Learning Fuzzy Systems  
Part 5 23 / 91
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.
The biimplication problem $x_1 \leftrightarrow x_2$: There is no separating line.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Formal proof by *reductio ad absurdum*:

since $(0, 0) \rightarrow 1$: $0 \geq \theta$, (1)
since $(1, 0) \rightarrow 0$: $w_1 < \theta$, (2)
since $(0, 1) \rightarrow 0$: $w_2 < \theta$, (3)
since $(1, 1) \rightarrow 1$: $w_1 + w_2 \geq \theta$. (4)

(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

\[ x_1 \leftrightarrow x_2 \equiv (x_1 \rightarrow x_2) \land (x_2 \rightarrow x_1) \]

computes \( y_1 = x_1 \rightarrow x_2 \)

computes \( y_2 = x_2 \rightarrow x_1 \)

computes \( y = y_1 \land y_2 \)

\( y = x_1 \leftrightarrow x_2 \)
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 $\text{sgn}\{\langle \mathbf{w}, \mathbf{x} \rangle - b\}$ by a sigmoid function

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

$S$ 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 $\mathbf{x}$, 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 pixels, 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

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

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 $x$ 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

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

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.
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 \( \mu^{(j)}_i \), 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:

\[ R_1 : \text{If } x_1 \text{ is } A_1 \text{ and } x_2 \text{ is } B_1 \text{ then } y = f_1(x_1, x_2) \]
\[ R_2 : \text{If } x_1 \text{ is } A_1 \text{ and } x_2 \text{ is } B_2 \text{ then } y = f_2(x_1, x_2) \]
\[ R_3 : \text{If } x_1 \text{ is } A_2 \text{ and } x_2 \text{ is } B_2 \text{ then } y = f_3(x_1, x_2) \]

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

$$\bar{a}_i = a_i = \text{net}_i = \frac{\tilde{a}_i}{\sum_j \tilde{a}_j}.$$
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 \( x \).

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

\[
\bar{y}_i = a_i = \text{net}_i = \bar{a}_i f_i(x_1, \ldots, x_n).
\]

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

\[
y = f(x_i) = a_{\text{out}} = \text{net}_{\text{out}} = \sum_i \bar{y}_i = \frac{\sum_i \bar{a}_i f_i(x_1, \ldots, x_n)}{\sum_i \bar{a}_i}.
\]
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 tuples $(x_k, y_k)$ 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 $e = \sum_{i=1}^{m} |y(k) - f(x(k))|^2$
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

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

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

$$y = \bar{a}RX$$

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.
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

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

\[
\text{do } \{ \\
\text{for each pattern } \{ \\
\quad \text{accumulate parameter updates} \\
\quad \text{accumulate error} \\
\} \\
\text{modify parameters} \\
\} \text{ while change in error}
\]

variations:

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

observing the error on validation set
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*
### Example: WBC Classification Performance

<table>
<thead>
<tr>
<th>predicted class</th>
<th>malign</th>
<th>benign</th>
<th>∑</th>
</tr>
</thead>
<tbody>
<tr>
<td>malign</td>
<td>228 (32.62%)</td>
<td>13 (1.86%)</td>
<td>241 (34.99%)</td>
</tr>
<tr>
<td>benign</td>
<td>15 (2.15%)</td>
<td>443 (63.38%)</td>
<td>458 (65.01%)</td>
</tr>
<tr>
<td>∑</td>
<td>243 (34.76%)</td>
<td>456 (65.24%)</td>
<td>699 (100.00%)</td>
</tr>
</tbody>
</table>

**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

- **Uniformity of Cell Size**
  - Two fuzzy sets: "sm" and "lg"
  - Graph showing membership values for cell size values from 1.0 to 10.0.

- **Bare Nuclei**
  - Bar chart showing membership values for cell size values from 1.0 to 10.0.
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

<table>
<thead>
<tr>
<th>Term</th>
<th>Biology</th>
<th>Computer Science</th>
</tr>
</thead>
<tbody>
<tr>
<td>individual</td>
<td>living creature</td>
<td>candidate solution</td>
</tr>
<tr>
<td>chromosome</td>
<td>DNA histone protein string</td>
<td>character string</td>
</tr>
<tr>
<td></td>
<td>defines “blueprint” and (partly)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>properties of individual in coded form</td>
<td></td>
</tr>
<tr>
<td></td>
<td>usually several chromosomes per individual</td>
<td>usually one chromosome per individual</td>
</tr>
<tr>
<td>gene</td>
<td>part of chromosome</td>
<td>one character</td>
</tr>
<tr>
<td></td>
<td>fundamental unit of inheritance that (partly)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>defines property of an individual</td>
<td></td>
</tr>
<tr>
<td>allele</td>
<td>characteristic of a gene</td>
<td>value of a character</td>
</tr>
<tr>
<td>(allomorph)</td>
<td>there is only one characteristic of a gene per chromosome</td>
<td></td>
</tr>
<tr>
<td>locus</td>
<td>position of a gene</td>
<td>position of a character</td>
</tr>
<tr>
<td></td>
<td>exactly just one gene on every position in chromosome</td>
<td></td>
</tr>
</tbody>
</table>
### Fundamental Terms and their Meanings II

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

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

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>1</th>
<th>2</th>
<th>0</th>
<th>3</th>
<th>4</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

R. Kruse, J. Schulze
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 $[-90^\circ, 90^\circ]$.

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 shrunk 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

<table>
<thead>
<tr>
<th>( \dot{\theta} ) ( \theta )</th>
<th>nb</th>
<th>nm</th>
<th>ns</th>
<th>az</th>
<th>ps</th>
<th>pm</th>
<th>pb</th>
</tr>
</thead>
<tbody>
<tr>
<td>pb</td>
<td>az</td>
<td>ps</td>
<td>ps</td>
<td>pb</td>
<td>pb</td>
<td>pb</td>
<td>pb</td>
</tr>
<tr>
<td>pm</td>
<td>ns</td>
<td>az</td>
<td>ps</td>
<td>pm</td>
<td>pm</td>
<td>pb</td>
<td>pb</td>
</tr>
<tr>
<td>ps</td>
<td>nm</td>
<td>ns</td>
<td>az</td>
<td>ps</td>
<td>pm</td>
<td>pm</td>
<td>pb</td>
</tr>
<tr>
<td>az</td>
<td>nb</td>
<td>nm</td>
<td>ns</td>
<td>az</td>
<td>ps</td>
<td>pm</td>
<td>pb</td>
</tr>
<tr>
<td>ns</td>
<td>nb</td>
<td>nm</td>
<td>nm</td>
<td>ns</td>
<td>az</td>
<td>ps</td>
<td>pm</td>
</tr>
<tr>
<td>nm</td>
<td>nb</td>
<td>nb</td>
<td>nm</td>
<td>nm</td>
<td>ns</td>
<td>az</td>
<td>ps</td>
</tr>
<tr>
<td>nb</td>
<td>nb</td>
<td>nb</td>
<td>nb</td>
<td>nb</td>
<td>ns</td>
<td>ns</td>
<td>az</td>
</tr>
</tbody>
</table>
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” ⟷ “approximately zero” or “positive medium”,
- “negative big” ⟷ “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)

<table>
<thead>
<tr>
<th>...</th>
<th>nm</th>
<th>ns</th>
<th>az</th>
<th>ps</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>-45</td>
<td>-30</td>
<td>-15</td>
<td>0</td>
<td>15</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th></th>
<th>-45</th>
<th>-15</th>
<th>15</th>
<th>-15</th>
<th>15</th>
<th>30</th>
<th>15</th>
<th>30</th>
<th>45</th>
<th>30</th>
<th>45</th>
<th>60</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-45</td>
<td>-30</td>
<td>-20</td>
<td>-30</td>
<td>-20</td>
<td>-10</td>
<td>-20</td>
<td>-10</td>
<td>0</td>
<td>-10</td>
<td>10</td>
<td>30</td>
<td>...</td>
</tr>
</tbody>
</table>

R. Kruse, J. Schulze
Encoding Fuzzy Partitions

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

\[
\begin{align*}
\text{Gene 1:} & & \mu_1(x_1) & \mu_2(x_1) & \cdots & \mu_n(x_1) \\
\vdots & & \vdots & \vdots & \ddots & \vdots \\
\text{Gene m:} & & \mu_1(x_m) & \mu_2(x_m) & \cdots & \mu_n(x_m)
\end{align*}
\]

Encoding with \(m \times n\) numbers \(\in [0, 1]\)

<table>
<thead>
<tr>
<th>pb</th>
<th>0 0 0 0 0 0 0 0 0 0 0 0 0 .5 1 1 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>pm</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 .5 1 1 .5 0 0 0</td>
</tr>
<tr>
<td>ps</td>
<td>0 0 0 0 0 0 0 0 0 0 0 .5 1 .5 0 0 0 0 0</td>
</tr>
<tr>
<td>az</td>
<td>0 0 0 0 0 0 .5 1 .5 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>ns</td>
<td>0 0 0 0 0 .5 1 .5 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>nm</td>
<td>0 0 0 .5 1 .5 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>nb</td>
<td>1 1 1 .5 1 .5 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

-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 $\geq 1$ local maximum.
- Multimodal fuzzy sets are harder to interpret than unimodal ones.

Therefore, fuzzy sets are repaired (made unimodal).

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)