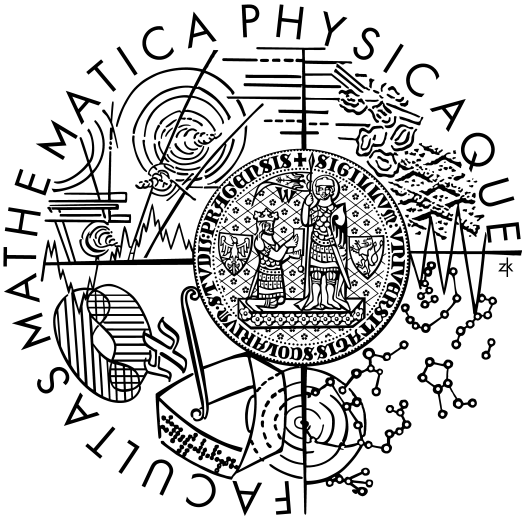
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Jiří Iša

Artificial neural networks for clustering and rule extraction
(Umělé neuronové sítě pro klastrování a extrakci pravidel)

Katedra softwarového inženýrství

Vedoucí diplomové práce: *RNDr. Iveta Mrázová, CSc.*

Studijní program: *Informatika, Teoretická informatika, Umělá inteligence*

I would like to thank the giants for offering their shoulders, my parents for pushing me upwards and my nearests for being patient with me when being angry of falling down.

This thesis would not exist without the constant careful supervision and valuable advice of Dr. Iveta Mrázová - my sincere thanks.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne: 18. dubna 2007 Jiří Iša

Contents

Title:	Artificial neural networks for clustering and rule extraction
Author:	Jiří Iša
Department:	Department of Software Engineering
Supervisor:	RNDr. Iveta Mrázová, CSc.
Supervisor's e-mail:	mrazova@ksi.ms.mff.cuni.cz
Keywords:	fuzzy, rules, extraction, neural network

Abstract: Rule extraction with neural networks has been a common research topic over the last decades. This master thesis proposes a novel growing fuzzy inference neural network, based on the principle of growing neural structures [5]. This allows the network to adjust iteratively its number of hidden neurons. For the purpose of this network an existing clustering algorithm is enhanced to improve the sensitivity to the requested output. A novel fast weights adaptation, inspired by the fuzzy set theory, is also suggested. The characteristics of the proposed model and a new method of the selection of significant input features support the induction of a relatively small amount of simple fuzzy rules. The introduced techniques have been experimentally tested on real-world data describing the relationship between various types of housing in the Boston area and its price. The data was obtained from the "Boston housing" dataset.

Název práce:	Umělé neuronové sítě pro klastrování a extrakci pravidel
Autor:	Jiří Iša
Katedra (ústav):	Katedra softwarového inženýrství
Vedoucí práce:	RNDr. Iveta Mrázová, CSc.
E-mail vedoucí:	mrazova@ksi.ms.mff.cuni.cz
Klíčová slova:	fuzzy, pravidla, extrakce, neuronová síť

Abstrakt: Problematika extrakce pravidel pomocí neuronových sítí byla během uplynulých desetiletí častým námětem výzkumných prací. Tato diplomová práce navrhuje nový model rostoucí fuzzy inferenční neuronové sítě, vycházejících z principu rostoucích neuronových struktur [5]. To síti umožňuje postupný nárůst počtu skrytých neuronů až do velikosti potřebné pro dané úlohy. Pro účely této sítě byla zvýšena citlivost již existujícího shlukovacího algoritmu vůči požadovaným výstupním hodnotám. Je také představen nový rychlý algoritmus adaptace vah, inspirovaný teorií fuzzy množin. Vlastnosti navrženého modelu i nová metoda výběru signifikantních vstupních příznaků podporuje extrakci relativně malého množství jednoduchých fuzzy pravidel. Navržené techniky jsou experimenálně ověřeny na reálných datech popisujících vztah mezi různým typem bydlení v okolí Bostonu a jeho cenou. Data byla získána z databáze "Bostonské ceny bydlení".

Chapter 1

Introduction

Over the last decades artificial neural networks have been used as a very useful tool to solve many tasks. Their powerful learning capabilities predetermine them for difficult and not fully understood problems. Once a training set is available for the task at hand, a very common approach is to train one or more neural network models for it and then let the best model work. No matter how, just let it work. Unfortunately, for many tasks the black-box behavior may pose a serious problem. For example in medicine nobody can take the risk of using a model, which can misjudge in some unknown case.

The behavior and application of neural networks stand in the opposition to the classical if-then rules. If-then rules are well understood by every software developer. Once there is a domain expert available, it is often feasible to provide an initial set of rules. Later on, these rules can be refined.

On the other hand, the problem is how to obtain the set of rules for a given task in a general case. An expert may not always be accessible or cooperative. Furthermore, the size of the set of rules may increase dramatically with the complexity of the task. The expert may not be able to communicate all rules. She may not even know them all consciously.

Another problem is the "sharpness" of if-then rules. If the input changes slightly, a completely different rule may become active and the output can change significantly. This may lead to large undesired system oscillations.

A well-known solution to these if-then rules problems represent fuzzy if-then rules. The central point of view shifts from the classical crisp logic to the fuzzy logic. In fuzzy logic a term may be satisfied to a certain degree instead of only being true or false. When using fuzzy logic, a rule may become active to a certain degree. The consequents of rules are then combined together with regard to the rule activations.

Yet another advantage of the fuzzy theory approach may be its stability with no extreme oscillations. Furthermore, it still provides a set of rules and as

such it can be created and verified by a domain expert. Although the presence of fuzziness may allow to express the same in less rules, when compared with their "crisp" if-then counterparts, the fuzzy rules interaction can become very difficult to understand. While it remains possible for a human expert to verify the rules (at least to a certain level), it may become extremely tedious, if not impossible, to construct a fuzzy rule base just manually.

For these reasons author of this thesis expects that fuzzy neural networks represent the right direction to eliminate the weaknesses of both neural networks and if-then rules. In this thesis, several models of neural networks, including existing fuzzy neural networks, are discussed. Their advantages and disadvantages are used to motivate the introduction of a new growing fuzzy inference neural network (GFINN). The nature of the model provides a growing architecture for the task being solved. The constructed network is considered to be sufficiently large in order to achieve the desired performance and sufficiently small in order to yield adequate generalization.This new model is based on structure of so-called growing neural gas. The basic model is, however, substantially extended and improved. The final neural structure can be transformed into a set of fuzzy rules, making the function of the network verifiable. The GFINN structure supports an easy selection of significant input features. As a result, the extracted rules will be simpler and easier to understand, compared to other existing fuzzy neural networks. A simple FOPT algorithm will be introduced for quick training of GFINN networks. Supporting experiments performed so far have confirmed that FOPT is capable of achieving an almost optimal performance, despite of the relatively simple computations of the output weights.

The capabilities of the new model were tested both on an artificial dataset and on real-world data - the Boston housing dataset, publicly available from the UCI repository. Despite of a large amount of incorrectness in the data, GFINN has been shown to extract a reasonably small set of fuzzy rules, which are easy to understand.

Several different models of self-organizing neural networks, including the growing structures, are described in the second chapter. The point of view of the fuzzy set and fuzzy logic theory is brought in in the third chapter. This chapter also describes existing fuzzy-neural techniques. The GFINN model is introduced in the fourth chapter and experimentally verified in the fifth one.

Chapter 2

Self-organizing neural networks

2.1 Motivation

In the nature, self-organization can be seen as a process in which an order emerges spontaneously. In such a process a large collection of simple microscopic objects interacts locally. Few simple rules can give a raise to a very complex macroscopic structure.

Self-organization in the human brain has been linked, among others, to the evolution of speech [19, 18] and to human localization in space [21, 16]. In the hippocampus only few place cells fire for every space location. Nearby locations activate close groups of neurons.

In the literature, there can be found several formal models resembling this kind of natural brain processes [10, 1].

Few necessary terms such as clustering and vector quantization are introduced in this chapter first. Later on, the basic self-organizing maps (SOM) and related models are described and several improvements are suggested.

2.2 Background

2.2.1 Clustering

In the real world we can observe that certain objects may form groups based on their similarity. A group of boxes will differ from a group of spheres. A group of third world countries may be distinct from a group of highly industrial countries.

Methods which aim at automatical discovering such groups of mutually similar objects are called clustering techniques. The goal is to divide a finite set of input patterns $T = \{\overrightarrow{t_i} : \ i = 1, 2, \ldots, k\}$, $\overrightarrow{t_i} \in \Re^n$ into k pairwise

Figure 2.1: A natural example of self-organization - A complex regular snowflake structure is determined by local molecule interaction.

disjoint subsets T_j, $j = 1, \ldots, k$. At the same time, the subsets are required to contain only mutually similar elements, while different subsets should contain mutually dissimilar items. In general, the chosen similarity measure should be considered to be domain specific, although though few general measures, such as Euclidean metric, are used very often [10].

2.2.2 Vector quantization

Vector quantization is a classical signal-processing method that usually forms a quantized approximation to the distribution of the input patterns $\overrightarrow{x} \in \Re^n$ [10]. A finite number of adequate reference vectors $\overrightarrow{m_i} \in \Re^n$, $i = 1, \ldots, k$ is to be found during vector quantization. Once the reference vectors are chosen, the approximation of $\overrightarrow{x}$ means finding the reference vector $\overrightarrow{m_b}$ closest to $\overrightarrow{x}$ (in the input space), usually using the Euclidean metric:

$$\overrightarrow{m_b} = \text{argmin}\{\|\overrightarrow{x} - \overrightarrow{m_i}\|\} \tag{2.1}$$

2.2.3 Artificially generated sample data

The following sections deal with the supervised and unsupervised variants of self-organizing maps. To visualize their properties a sample dataset shown in Figure 2.2 is used.

When used to illustrate the function of unsupervised methods, the input data patterns are considered to come from the grey areas with a uniform probability distribution. In such a case, white areas have a zero data density.

When used to illustrate supervised learning, the situation is slightly different. In this case the grey areas represent the data from the class labeled

Figure 2.2: An artificially generated sample dataset.

with '+1', while the white areas represent the class labeled with '0'. For supervised learning, the data patterns are uniformly distributed over the whole input space $\langle 0, 1\rangle \times \langle 0, 1\rangle$.

2.3 Unsupervised variants

2.3.1 K-means clustering

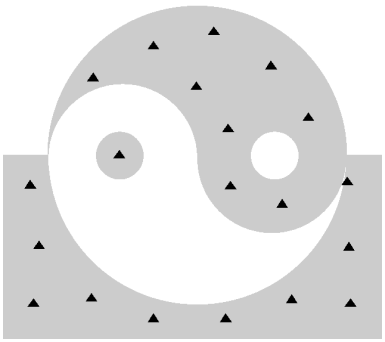


Figure 2.3: Cluster centers obtained by k-means clustering ($k = 20$)

The k-means clustering algorithm is generally not considered as belonging to the SOM-family. This method is, however, very similar to the described SOMs and brings an important insight into their function.

K-means is widely used as one of the most straightforward clustering algorithms. For the given value of k and the training set $T = \{\overrightarrow{x_i} : \ i = 1, 2, \ldots, n\}$, $\overrightarrow{x_i} \in \Re^n$ it finds k centers $\overrightarrow{m_1}, \ldots, \overrightarrow{m_k}$ of the clus-

ters and divides the training set T into k disjoint subsets corresponding to these clusters.

K-means algorithm [20]

Input: Training set $T = \{\overrightarrow{x_i}\}$, $k \in N$
Output: k cluster centers $\overrightarrow{m_1}, \ldots, \overrightarrow{m_k}$
Algorithm:

1. Initialize the cluster centers $\overrightarrow{m_i}$, $1 \leq i \leq k$ randomly.
2. For every center $\overrightarrow{m_i}$ find the set T_i of all the training patterns from T, for which $\overrightarrow{m_i}$ is the nearest cluster center using the Euclidean metric.
3. Use the mean of each group T_i as a new cluster center $\overrightarrow{m_i}$:
$$\overrightarrow{m_i}(t+1) = \frac{1}{|T_i|} \sum_{\overrightarrow{x} \in T_i} \overrightarrow{x} \tag{2.2}$$
4. Repeat steps 2 and 3 for a fixed number of steps.

Discussion

Figure 2.3 shows the result of the k-means algorithm with $k = 20$ on the sample dataset.

Learning can be stopped if the cluster centers do not change from one learning cycle to the next one. In this case, all cluster centers remain stable in the future.

Although k-means is used for clustering very often, it actually performs vector quantization. Both these tasks are similar in that they both map input patterns to a small set of representative vectors. It is, however, expected, that in clustering coherent groups of patterns are mapped all together, onto a single representant, no matter how big the coherent group of patterns is. But it can be shown, that the k-means algorithm attempts to equally distribute the input patterns over the representants with respect to their distance, which is vector quantization task (see Figure 2.4).

The biggest difference between k-means clustering and self-organizing maps is the lack of structure over cluster centers. The cluster centers are completely self-sufficient. An input pattern $\overrightarrow{x} \in T_i$ does not affect the adaptation of another center $\overrightarrow{m_j}$, $i \neq j$.

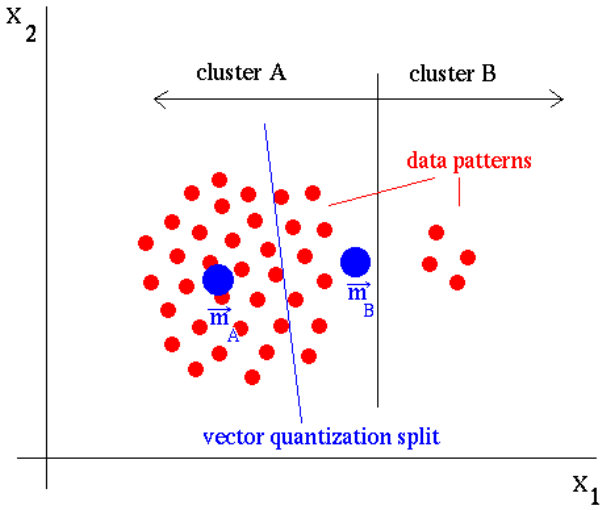


Figure 2.4: An illustration of the difference between vector quantization and clustering. Both the expected outcome of a clustering algorithm and of the 2-means algorithm are depicted on the figure.

2.3.2 Self-organizing feature maps - the standard model

In this text "standard" denotes the widely used SOM version described by Kohonen [10]. Self-organizing feature maps represent a type of unsupervised artificial neural networks. It has been inspired by biological processes similar to those ones mentioned in the introduction. The goal is to map the presented input patterns to the corresponding small region of the network. In the most common case, a single neuron should be active for the presented input pattern. It is also desired that the SOM preserves the topological relationships present among the input data - similar input patterns should be mapped onto close network areas. This makes the method particularly suitable for low-dimensional visualization of high-dimensional data. In the past it has been successfully used in various areas - for example to group syllables [18], similar groups of pictures [11], texts [12], countries [9] and many other objects. Furthermore it may also show, which data clusters appear to be similar.

The SOM consists of neurons $N = \{n_1, \ldots, n_k\}$ and a set of edges $E = \{e_1, \ldots, e_m\}$ interconnecting these neurons. Every neuron n_i has an attached reference vector $\overrightarrow{m_i} \in \Re^n$ with the same dimensionality as the input patterns. During unsupervised learning the reference vectors are adapted to

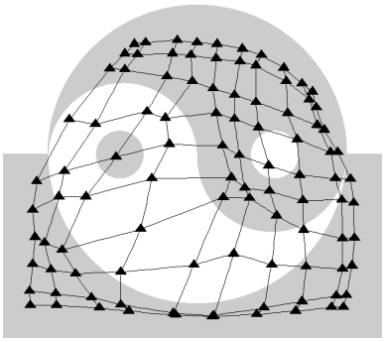


Figure 2.5: SOM grid structure trained on the sample dataset

produce low-dimensional representations of the training samples, while preserving their topological structure present in the input space.

Kohonen suggests using a regular two-dimensional graph structure for SOM. The most common one is the rectangular grid structure. The network is organized into rows and columns of neurons. Every neuron is connected by an edge with neurons in the next rows and columns. Another option is to use a hexagonal structure.

The main advantage of these two-dimensional structures is that they can be used for visualization easily. When the input patterns are used to calibrate their best matching neurons during the drawing, the underlying data relationships can be shown nicely. The resulting structure is easy to understand and verify.

Recall

Let us consider the most common SOM model with a single neuron activation. After an input pattern $\overrightarrow{x}$ is presented to the network, the best matching neuron n_b is found. The best matching neuron n_b is the closest one to the input pattern $\overrightarrow{x}$. The index b can be determined according to:

$$b = \underset{i}{\operatorname{argmin}} \| \overrightarrow{x} - \overrightarrow{m_i} \| \tag{2.3}$$

$\overrightarrow{m_i}$ denotes the reference vector of neuron n_i.

All input patterns mapped on the same neuron are considered to form one data cluster.

The metric used throughout this thesis is the general Euclidean metric:

$$\|\overrightarrow{x} - \overrightarrow{y}\| = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.4}$$

Training

Basic SOMs are trained by self-organization. A training sample $\overrightarrow{x}$ is presented to the network. Afterwards the network adjusts the reference vectors in order to reflect the occurrence of such a sample vector better.

SOM algorithm

Repeat the following steps for a fixed amount of iterations or until another stop condition is met:

1. Present the next training pattern $\overrightarrow{x}$ from the training set T.
2. Find the best matching neuron n_b for the presented input pattern $\overrightarrow{x}$ according to:

$$b = \underset{i}{\operatorname{argmin}} \|\overrightarrow{x} - \overrightarrow{m_i}\|$$

3. Adjust the reference vectors using the following rule:

$$\overrightarrow{m_i}(t+1) = \overrightarrow{m_i}(t) + h_{bi}(t)[\overrightarrow{x} - \overrightarrow{m_i}(t)] \tag{2.5}$$

where

t ... discrete time step

$h_{bi}(t)$ ... neighborhood function

The trained network should preserve the topological structure of the input data. This is supported by the neighborhood function h_{bi}. Not only the best matching neuron is adapted for the presented input pattern $\overrightarrow{x}$, but also the neighboring neurons in the network. One of the common approaches to the definition of the neighborhood function h_{bi} is to use the neighborhood set $N_b(t)$:

$$h_{bi}(t) = \begin{cases} \alpha(t) & \text{if } i \in N_b(t) \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

where $\alpha(t)$ denotes the learning rates depending on time t. Both $\alpha(t)$ and the size of the neighborhood set $N_b(t)$ should decrease in time. The neighborhood set N_i contains neurons that are close to the neuron n_i in the network structure. Typically $N_i(t) = \{n_j | \Delta(n_i, n_j) <= d(t)\}$, where $\Delta(n_i, n_j)$ is the graph distance of neurons n_i and n_j and $d(t)$ is a distance threshold in time t.

Another approach to the definition of h_{bi} is to use reference vectors instead of the network structure. The often used *Gaussian neighborhood* is defined by means of:

$$h_{bi}(t) = \alpha(t) * e^{-\frac{\|\vec{m_b} - \vec{m_i}\|^2}{2\sigma^2(t)}} \tag{2.7}$$

where $\alpha(t)$ is the learning rate and the parameter $\sigma(t)$ controls the neighborhood width [10].

To guarantee the convergence of the algorithm, the neighborhood function h_{bi} should approach 0 with increasing time. This property can be achieved by decreasing the distance threshold $d(t)$ with time in the first approach, or by lowering the $\sigma(t)$ parameter in the second approach.

Discussion

Figure 2.5 shows a regular 10×10 self-organizing map grid covering the sample dataset.

One of the disadvantages of the standard approach is, that it should be decided beforehand, which size of the network to use. If it is too small, the network may not be able to handle the data appropriately. On the other hand, a too big size can lead to overfitting.

A planar (2D) network structure represents an advantage for visualization, but it may make it more difficult to reflect complex relationships present in multi-dimensional data. Then the two-dimensional network needs to "bend" in the multi-dimensional input space. Similar input patterns may then be represented by neurons located very far one from the other on the SOM grid.

2.3.3 Growing grid

Growing grid extends the standard self-organizing map by an ability to change its structure based on the incoming training patterns. This removes one of the mentioned problems of the standard SOMs- the predetermination of their size. This method is suggested by Fritzke in [4].

Initially, the network is organized into a $p \times r$ grid of neurons:

$$N = [n_{ij}], 1 \leq i \leq p, 1 \leq j \leq r$$

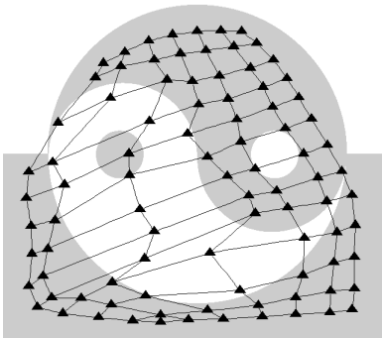


Figure 2.6: The growing grid for the sample dataset

Each neuron $n_{ij} \in N$ has an associated reference vector $\overrightarrow{m_{ij}} \in \Re^n$ as described in 2.3.2. There is also a *resource variable* τ_{ij} associated with every neuron n_{ij}. It stores a statistical information used to determine where to insert a new row or column of neurons.

Algorithm

1. Start with a small grid (for example 1×2).
2. Initialize all resource variables τ_{ij} to 0.
3. Repeat the following steps for a fixed amount of cycles or until another stop condition is met:
   (a) Present the next training pattern $\overrightarrow{x}$.
   (b) Find the best matching neuron n_b.

3. (c) Adapt all reference vectors according to:

$$\overrightarrow{m_{ij}}(t+1) = \overrightarrow{m_{ij}}(t) + \alpha(t) e^{-\frac{\Delta^2(n_{ij},n_b)}{2\sigma(t)^2}} (\overrightarrow{x} - \overrightarrow{m_{ij}}) \qquad (2.8)$$

where $\Delta(n_{gh}, n_{ij})$ is the Manhattan distance:

$$\Delta(n_{gh}, n_{ij}) = |g - i| + |h - j| \qquad (2.9)$$

[4] advises to keep both the learning rate $\alpha(t)$ and the width parameter $\sigma(t)$ constant during the whole adaptation process to improve the on-line learning capability.

(d) Increase the resource variable of the best matching neuron n_b:

$$\tau_b(t+1) = \tau_b(t) + 1 \qquad (2.10)$$

(e) If the current time t is a multiple of $p(t) * r(t) * \lambda$ (λ is an integer value influencing the growth of the network), the network is extended by adding a column or a row:

i. Find the neuron n_g with the highest value of resource variable τ_g. This is the most often used (i.e. best-matching neuron) in the grid. In order to distribute the input signals more equally among the neurons, the grid shall be changed around this neuron.

ii. Find the neuron n_f, that is neighbor of n_g with the highest distance of its reference vector $\|m_f - m_g\|$. This neuron is supposed to indicate the direction of the highest variance in the underlying data.

iii. Without loss of generality let us assume for $n_g = n_{ij}$ that $n_f = n_{i(j+1)}$, i.e. n_f and n_g share the same row in the grid. The case of shared columns is analogical. A new column j' of additional neurons is placed between the columns j and $j+1$:

$$\overrightarrow{m_{kj'}} = \frac{1}{2}(\overrightarrow{m_{kj}} + \overrightarrow{m_{k(j+1)}}) \qquad 1 \leq k \leq p \qquad (2.11)$$

3. (e) iv. Using this operation the number of columns increases:

$$r(t+1) = r(t) + 1$$

v. Reset all resource variables τ_s to zero.

The growing grid algorithm constructs a 7×13 structure for the sample dataset (see Fig. 2.6).

2.3.4 Growing neural gas

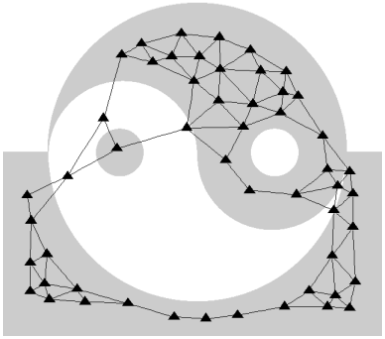


Figure 2.7: A growing neural gas used for the sample dataset

The *growing neural gas* (GNG) [5, 14] employs a much more loose structure than the growing grid and the standard SOMs do. The network graph does not have to be two-dimensional any more. During the learning phase this algorithm creates new neurons, new edges can be added and the old ones deleted.

The resource variables τ_i are replaced by variables $error_i$. A new age_e variable is attached to every edge e. This variable counts the time since the last moment, when the two neurons on this edge were the two best-matching neurons for a training pattern $\overrightarrow{x}$. If it has been a long time since both neurons on an edge were the best-matching pair, the two involved neurons are likely to be located one too far from the other in the input space, and thus they should not be connected by an edge.

Growing neural gas algorithm

1. Start with two neurons placed randomly in the input space and mutually inter-connected by an edge.
2. Take the next training pattern $\overrightarrow{x}$.
3. Find the nearest neuron n_{b_1} and the second-nearest neuron n_{b_2}. The neuron n_{b_2} will be used for the new edge insertion or age_e variable reset in step 7.
4. Increment age of all edges emanating from n_{b_1}:

$$age_e(t+1) = age_e(t) + 1$$

5. Add the squared distance between the input $\overrightarrow{x}$ and the reference vector $\overrightarrow{m_{b_1}}$ to the local error counter of the neuron n_{b_1}:

$$error_{n_{b_1}}(t+1) = error_{n_{b_1}}(t) + \|\overrightarrow{m_{b_1}} - \overrightarrow{x}\|^2 \tag{2.12}$$

6. Move n_{b_1} in the direction of $\overrightarrow{x}$:

$$\overrightarrow{m_{b_1}}(t+1) = \overrightarrow{m_{b_1}}(t) + \alpha_b(\overrightarrow{x} - \overrightarrow{m_{b_1}}) \tag{2.13}$$

   And similarly for all the neighbors n_k of n_{b_1}:

$$\overrightarrow{m_k}(t+1) = \overrightarrow{m_k}(t) + \alpha_k(\overrightarrow{x} - \overrightarrow{m_k}) \tag{2.14}$$

7. If n_{b_1} and n_{b_2} are connected by an edge, reset the age of this edge to zero. Otherwise create such an edge with an initial value 0.
8. Remove all edges with the age larger than age_{max}. If there is a neuron without any emanating edge after this operation, remove it as well.

9. If the number of steps performed so far is a multiple of the integral growing parameter λ, a new neuron is inserted:

   (a) Determine a neuron n_g with the largest accumulated local error.

   (b) Find n_f, a direct neighbor of n_g with the largest accumulated local error.

   (c) Insert a new neuron n_r halfway between n_g and n_f:

$$\overrightarrow{m_r} = \frac{1}{2}(\overrightarrow{m_g} + \overrightarrow{m_f})$$

   (d) Replace the edge $n_g \leftrightarrow n_f$ by two edges: $n_g \leftrightarrow n_r$ and $n_r \leftrightarrow n_f$.

   (e) Decrease the error variable $error_g$ and $error_f$ by a multiplication by a constant α. Initialize the error variable $error_r$ with the new $error_g$ value.

10. Decrease all error variables by the multiplication by a constant $d \in (0, 1)$ (forgetting).

11. If the stopping criterion has not been met yet (network size, fixed umber of steps, ...), repeat from the second step.

Figure 2.7 shows the growing neural gas covering the sample dataset.

2.4 Supervised extensions

Although the basic self-organizing neural networks (SOMs) were designed as an unsupervised model, it is very easy to extend them for the supervised learning. One can just assign an output value y_i to every neuron n_i. After the best matching neuron n_b is found for the input data pattern $\overrightarrow{x}$, the corresponding output value y_b can be used as the network output.

This section deals with such extensions of the previously described SOM methods. Several approaches to obtain the output values y_i are shown. All models in this section are described from the point of view of data classification.

2.4.1 LVQ1

Similarly like k-means, LVQ also does not belong to the SOM family. However, it is, again, a very straightforward method similar to standard SOM methods, that can be used to obtain some useful insight.

Before further processing the training set T is split into disjoint training sets $T_c = \{\overrightarrow{x} \in T : class(\overrightarrow{x}) = c\}$. For every T_c a separate k-means clustering algorithm is run. The algorithm results in a set of cluster centers $C_c = \{\overrightarrow{m_{c1}}, \ldots, \overrightarrow{m_{ck}}\}$ [10]. In every cluster set C_c the output value y_{ci} of the respective center m_i is set to the class label c.

For an input pattern $\overrightarrow{x}$ the output value y_b of the best matching center $\overrightarrow{m_b}$, with $b = \text{argmin}_i \|\overrightarrow{x} - \overrightarrow{m_i}\|$, is taken as the output of the network.

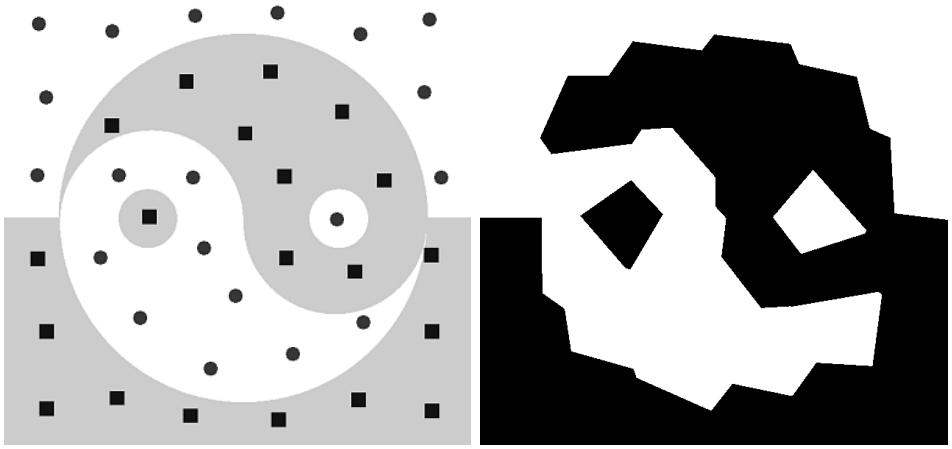


Figure 2.8: LVQ1 cluster centers for the sample data (left) and the classification (right).

Figure 2.8 shows the cluster centers obtained for the sample data. The centers for the "+1" class are squared, while the centers for the "0" class are circular.

It is very important to note, that the clustering (resp. quantization) is run for every data class c_i completely independently of the data from different classes c_j, $i \neq j$. If two different classes have nearby located areas with high data density, there are two different cluster centers assigned to them. If the training set had not been divided, only a single cluster could have been created. Unfortunately, this single cluster, representing data from different classes could have a high classification error.

2.4.2 Supervised standard self-organizing map

The crucial point is how to obtain an appropriate output value for every neuron. Because the network is self-organizing anyway, the self-organization can be used for learning of the outputs as well. Kohonen suggests to use extended data patterns $\overrightarrow{x_k} = (\overrightarrow{x_k^{in}}, \overrightarrow{x_k^{out}})$, $\overrightarrow{x_k} \in \Re^{m+n}$ for this purpose [10]. The $\overrightarrow{x_k^{in}} \in \Re^m$ part of $\overrightarrow{x_k}$ corresponds to the m-dimensional input space of the data and $\overrightarrow{x_k^{out}} \in \Re^n$ to the n-dimensional output space. The standard self-organizing map with such (m+n)-dimensional reference vectors can be trained using the "complete" training set $T = \{\overrightarrow{x_k}\}$.

To classify an input pattern $\overrightarrow{x^{in}} \in \Re^m$, the best matching neuron n_b is found based only on the distance in the input space $\|\overrightarrow{x^{in}} - \overrightarrow{m_b^{in}}\|$. The $\overrightarrow{m_b^{out}}$ part of the reference vector may then be considered to represent the output of the network.

Discussion

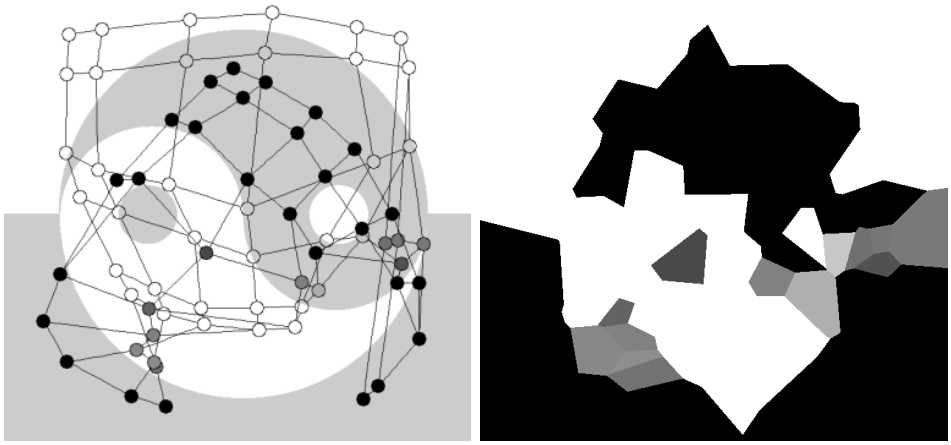


Figure 2.9: 10×10 grid used to classify the sample data (left) and the classification clusters (right).

Figure 2.9 shows the 10×10 grid used for the sample data classification. The color of the neurons reflects the output value used for classification later on. The darker, the closer is the "class label" to +1; the lighter, the closer to 0.

Figure 2.9 illustrates a problem as well. Although the input data are two-dimensional, the network has to work with three-dimensional patterns

(including the output). As a result, there may occur limitations mentioned in section 2.3.2. The two-dimensional structure of the grid forces it to bend over in the third coordinate in order to fill the whole extended pattern space. Some neurons learn then a wrong output, just because they happen to have "wrong" neighbors. The resulting classification is obviously far from being perfect.

2.4.3 Supervised growing grid

The supervised growing grid algorithm also does not differ too much from the unsupervised version. Only the resource variable τ_i is replaced by the error variable $error_i$, similar to the one in the growing neural gas [3]. Its value accumulates the classification error. The grid grows in the direction of wrong classification instead of a higher data density.

An approach different from the Kohonen's "extra coordinates" may be taken for the classification. It is suggested in this thesis, that the ratio of classes of the data, for which the neuron n_i is the best-matching neuron, may be remembered for all the classes and used as an output class estimate of the class c:

$$\eta_i(c) = \frac{count_{ci}}{count_i} \in \langle 0, 1 \rangle \tag{2.15}$$

where c is a class, n_i is a neuron, $count_{ci}$ is the amount of patterns with best-matching neuron n_i and class c. $count_i$ is the amount of all training patterns with the best-matching neuron n_i. The class c with the maximum $\eta_i(c)$ estimate is used as the output y_i of the neuron n_i. This is inspired by the maximum-likelihood approach in probabilistic models.

Figure 2.10 shows the structure developed by growing grid for the sample data. The darkness corresponds to the estimate $\eta_i(+1)$ of the class labeled "+1". This figure also shows the weakness avoided by LVQ (as mentioned in 2.4.1). Because the learning reflects the data density and not the data class, in the result the neuron positions may be inappropriate for the classification.

2.4.4 Supervised growing neural gas

The supervised growing neural gas algorithm combines the unsupervised growing neural gas version with the error accumulation described in section 2.4.3 for the supervised growing grid.

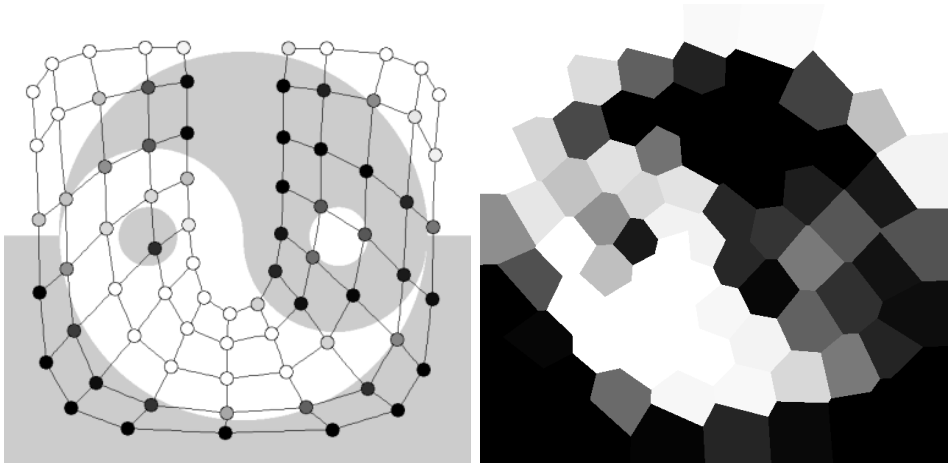


Figure 2.10: The growing grid structure for the sample data (left) and the classification clusters (right).

Proposed improvements

In the original version of the algorithm described by Fritzke [3], there is no limit for the size of the network. The only way to restrict its size is to put it as a stop condition. Although the new neurons are inserted into areas with the highest misclassification rates, they are naturally attracted to areas with higher data density due to the standard self-organizing learning rule afterwards.

To restrict this situation two suggestions are proposed in this thesis. First, the applied learning rule should be changed from

$$\overrightarrow{m_i}(t+1) = \overrightarrow{m_i}(t) + h_{bi}(t)(\overrightarrow{x} - \overrightarrow{m_i}(t)) \tag{2.16}$$

to

$$\overrightarrow{m_i}(t+1) = \overrightarrow{m_i}(t) + h_{bi}(t)\eta_i(c)(\overrightarrow{x} - \overrightarrow{m_i}(t)) \tag{2.17}$$

adding an extra sensitivity to the class density of the data similarly to the LVQ1 learning mechanism. $\overrightarrow{m_i}$ is a reference vector of the neuron n_i, n_b is the best-matching neuron for input $\overrightarrow{x}$, h_{bi} is the neighborhood function, t is a discrete time step and c denotes the class the sample $\overrightarrow{x}$ belongs to. The class estimates $\eta_i(c)$ are determined according to equation 2.15. This way the neuron does not get attracted to another class data, than it represents, so strongly and it supports convergence to the center of the own class cluster.

The other extension suggested here is to prune redundant neurons. As already mentioned the network grows infinitely. Furthermore, the new neurons converge often to the "flat" areas of the input space - areas with a

constant class membership and high data density. To identify redundant neurons, let us define the ε-insignificance of a neuron: A neuron n_i will be called *ε-insignificant*, if $\forall n_j \in neighbors(n_i); \|\overrightarrow{\eta_i} - \overrightarrow{\eta_j}\| < \varepsilon$, $\overrightarrow{\eta_i} = (\eta_i(c_1), \ldots \eta_i(c_k))$, $c_1, \ldots, c_k$ are the data classes. The insignificance of a neuron is computationally easy to verify. If 0-insignificant neuron n_r should be removed from the network, the classification error of the entire network does not change. All the data previously mapped to the currently removed neuron n_r would be mapped to one of its topological neighbors $n_j \in neighbors(n_r)$ with exactly the same class estimates $\overrightarrow{\eta_r} = \overrightarrow{\eta_j}$. If an ε-insignificant neuron with $\varepsilon > 0$ is removed, the new network error is affected not only by the ε value, but also by the probability of the removed neuron to be the best-matching one for the considered data inputs.

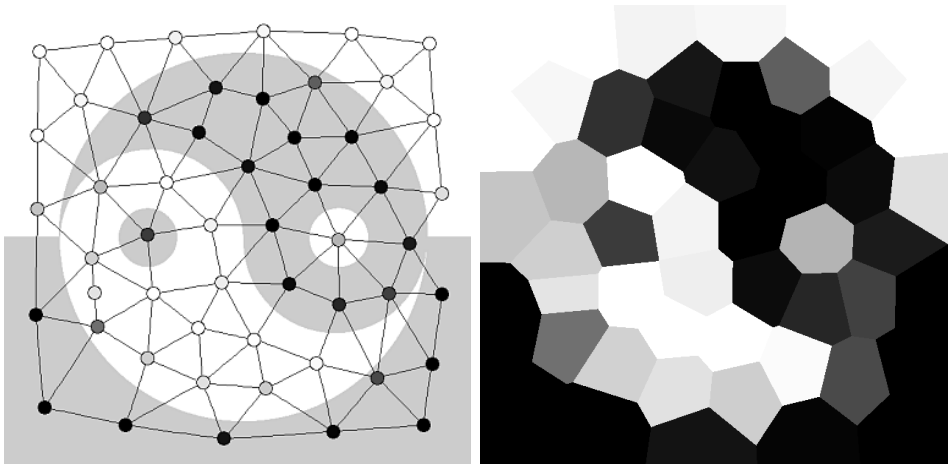


Figure 2.11: The supervised growing neural gas with pruning applied to the sample dataset. The formed network structure is shown on the left, classification clusters can be seen on the right.

Discussion

Figure 2.11 shows the network structure obtained with the extended supervised growing neural gas algorithm. The color of the neurons corresponds to the maximum-likelihood class estimate $\eta_i(+1)$.

Figure 2.12 shows (left), how the size of the network stabilizes at about fifty neurons using the 0-insignificance pruning, although, according to the algorithm, it had time enough to grow up to four hundred neurons.

As shown on Figure 2.12 (right) the crisp classification error of the network keeps decreasing even when the network size is being reduced.

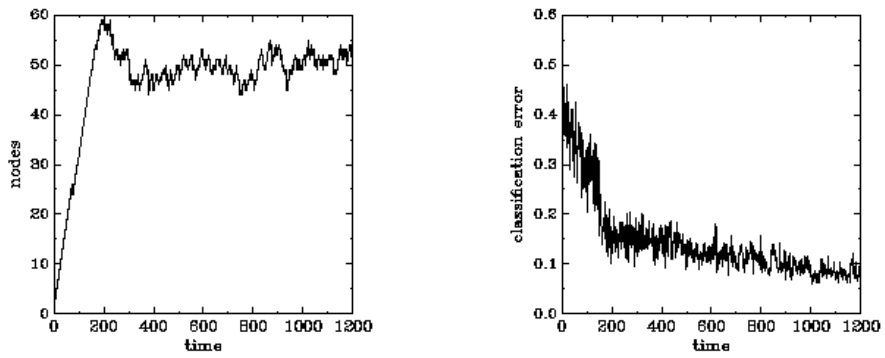


Figure 2.12: The growth of the extended growing neural gas network (left), the drop in the error values for crisp classification for the extended supervised growing neural gas algorithm (right).

Despite the effort to reflect the class membership of the data more carefully, the neurons get attracted to areas with a higher data density. For the sample dataset, the neurons were moved out of the narrow peaks of the data classes into flat areas. This does not only decrease the chance of having a correct class representant where it is needed, but it can impact wrong classification too. Both facts contribute to a higher classification error.

Chapter 3

Fuzzy rules and fuzzy neural networks

3.1 Motivation

In the self-organizing maps described in the previous chapter, an input pattern is projected onto a single neuron in the network. In the supervised scenario this neuron has an output value, which is used as the output of the network. This approach, although straightforward and easy to understand, has some drawbacks. One of them is, that the network has only a limited set of possible outputs, which corresponds to the number of its neurons. This renders continuous output of the network impossible. Moreover, an even small change of an input pattern may cause a different neuron activation and a completely different output of the network. Such a behavior may be totally unsuitable in many situations, such as motion control. Rash convulsions can physically destroy the motion device.

An alternative solution to these problems represent fuzzy logic-based models, motivated by the assumption, that some facts may be partially true. Not just true or false. When combined with a neural network of a SOM type, this point of view introduces a partial neuron activation, while retaining an understandable theoretical background of fuzzy logic. The resulting models of artificial neural networks may even resemble the biological counterparts more, than the standard SOM models. The reason is that partial activation of neurons makes distributed representation of data as well as continuous output of the network possible.

3.2 Fuzzy sets

In the classical set theory a *membership* of an item to the set is a *"crisp"* function. An item either belongs to the set, or it does not. In the fuzzy set theory the membership function is not so strict [25]. For example John, being 178 cm tall, does not fully belong to the set of tall people. But we can also hardly say, that he is not tall at all. We could say that he belongs to the set of tall people with a membership value 0.8.

Definition 1. *Membership function μ_S of a set S assigns to every object a value representing how much it belongs to the set.*

$$\begin{aligned} \mu_S : & \quad x \to [0,1] && (3.1)\\ \mu_\emptyset(x) &= 0 && (3.2)\\ \mu_{\Omega\setminus S}(x) &= 1 - \mu_S(x) && (3.3)\\ \mu_{S\cap T}(x) &= min\{\mu_S(x), \mu_T(x)\} && (3.4) \end{aligned}$$

3.3 Fuzzy logic

Similarly to the fuzzy membership a fuzzy proposition can also have values between *true* and *false*.

The valuation of a compound formula is combined from the valuation of its elementary items using the following rules:

$$\begin{aligned} \nu(x) : & \quad x \to [0,1] && (3.5)\\ \nu(false) &= 0 && (3.6)\\ \nu(\neg x) &= 1 - \nu(x) && (3.7)\\ \nu(x \wedge y) &= min\{\nu(x), \nu(y)\} && (3.8) \end{aligned}$$

or with an alternative conjunction (*t-norm*):

$$\nu(x \wedge y) = \nu(x)\nu(y) \quad (3.9)$$

There are two commonly used approaches how to handle equality of real numbers in fuzzy logic (see fig. 3.3) - the Gaussian equality and its partially linear approximation.

3.4 Fuzzy rules

Fuzzy rule base is essentially a collection of fuzzy if-then rules, in which the preconditions and consequents involve linguistic variables [13]. A collection of

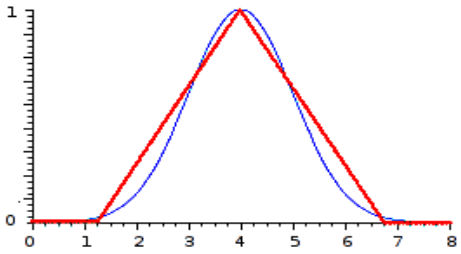


Figure 3.1: Gaussian fuzzy equality $e^{\frac{(x-4)^2}{2}}$ and its partially linear approximation as fuzzy equality $x =_F 4$.

fuzzy rules characterizes a simple input-output relation of the studied system. For multi-input-single-output system, the general form (Zadeh-Mamdani form) of the fuzzy rules is [26]:

$$IF \ \wedge_{F_j} (x_j \ op \ v_j^i) \ THEN \ y = val_i$$

- x_i ... input variables
- op ... fuzzy operator, commonly one of the following: $<_F$, $\leq_F$, $=_F$, $\geq_F$, $>_F$
- v_j^i ... linguistic parameters of the model (for example *low*, *medium* or *very high* with a known corresponding numeric value)
- val_i ... linguistic output value

A variant of this type are rules where the consequent is represented as a function of the input variables:

$$If \ \wedge_F (x_i \ op \ v_i) \ Then \ y = f(x_1, \ldots, x_n$$

having Takagi-Sugeno form of the rules as a special case [22]:

$$If \ \wedge_F (x_i \ op \ v_i) \ Then \ y = \sum_i p_i x_i$$

Where

- $\wedge_F$ ... fuzzy t-norm
- x_i ... input variables

- op ... fuzzy operator, commonly one of the following: $<_F$, $\leq_F$, $=_F$, $\geq_F$, $>_F$
- $v_i, p_i \in \mathcal{R}$ ... parameters of the model

The above rules evaluate the input of the system, compute the output and decide the corresponding action as a function of the input variables.

Both types of rules have linguistic values as their inputs and either linguistic or "crisp" values as their outputs.

3.4.1 Weighted average defuzzification

In usual crisp expert systems, only a single rule "fires" for every input. In fuzzy expert systems, due to the fuzzy set membership, all rules can potentially fire. Some fire stronger, some weaker, depending on the degree of membership of the input to the fuzzy set specified by the antecedent part of the rule.

There are several widely used methods how to combine the consequent outputs of the rules and activation levels of their antecedents [23]. The method used in this work is the *weighted average*:

$$y = \frac{1}{\sum_{r \in Rules} a_r} \sum_{r \in Rules} a_r y_r = \sum_{r \in Rules} \nu_r y_r \tag{3.10}$$

$$a_r = \wedge_{F_j}(x_j \ op \ v_j^r) \tag{3.11}$$

- a_r ... antecedent activation of the rule r
- y_r ... consequent value of the rule r
- ν_r ... normalized activation of the rule r:

$$\nu_k = \frac{a_k}{\sum_{r \in Rules} a_r} \tag{3.12}$$

3.5 Adaptive-network-based fuzzy inference system (ANFIS)

Figure 3.2 shows the ANFIS network [8] for the following Takagi-Sugeno fuzzy rules:

$$Rule\ 1: \quad If\ x_1\ op\ v_{11} \wedge_F x_2\ op\ v_{21}\ Then\ y = \textstyle\sum_i p_{1i} x_i$$
$$Rule\ 2: \quad If\ x_1\ op\ v_{12} \wedge_F x_2\ op\ v_{22}\ Then\ y = \textstyle\sum_i p_{2i} x_i$$

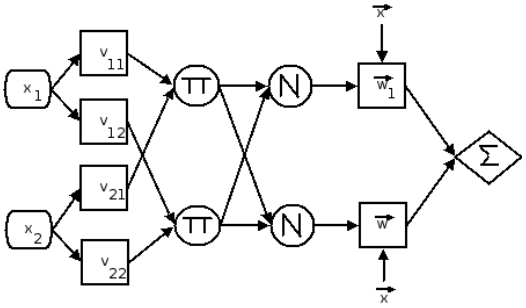


Figure 3.2: ANFIS network structure

Comparing the above rules with the structure of the network, it is easy to recognize the correspondence. The structure is designed to match the Takagi-Sugeno rules, which makes the extraction of the rules from the network trivial.

3.5.1 Recall

Algorithm

1. Calculate the fuzzy memberships of inputs $\overrightarrow{x}$ to the fuzzy partitioning of input values $\overrightarrow{v_{ij}}$ in the first layer (the input layer is not counted). Gaussian function is used if op is equality, sigmoidal function might be used if op is $<_F$ or $>_F$.

2. The second layer calculates the antecedents of the rules. Π or min are mostly used as $\wedge_F$.

3. The third layer normalizes the sum of the second layer outputs $\rightarrow w_k$.

4. The fourth layer calculates fuzzy rules consequents:

$$f_k = \sum_i p_{ki} x_i \tag{3.13}$$

where p_{ki} are the Takagi-Sugeno rule consequent parameters.

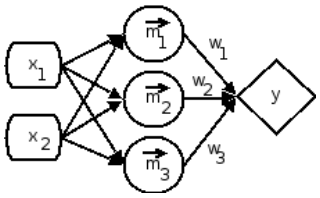


Figure 3.3: FINN network structure example

5. Output neuron defuzzifies the consequents of the rules using the weighted average:

$$y = \frac{\sum_k w_k f_k}{\sum_k w_k} \tag{3.14}$$

3.5.2 Training

A gradient descent algorithm, similar to backpropagation, and complex hybrid learning are described by Jang in [8].

3.5.3 Discussion

The network structure is designed to closely resemble the operations on fuzzy rules it represents. That is the strong and the weak point in the same time. It is very easy to read out the network functionality. On the other it can be extremely difficult to find right the structure of the network during training. The original adaptation algorithms suggested by [8] suffers from many common weaknesses of multilayer artificial neural networks trained with gradient descent. An optimal size of the network layer is not obvious in advance. And gradient descent can converge to local optima, instead of the global one. This risk increases with the size of the network.

Both mentioned problems can be at least partially avoided, if a rough version of the fuzzy rules is known in advance. This knowledge can be encoded into the network structure and the adaptation just fine-tunes the parameters afterwards.

3.6 Fuzzy inference neural networks (FINN)

While ANFIS is designed primarily for the fine-tuning of rules, Fuzzy inference neural network (FINN) is designed to automatically extract fuzzy if-then rules from the data [15].

Figure 3.3 shows the structure of the network. FINN is a RBF-like neural network with local units in a single hidden layer. Every neuron n_i in the hidden layer has a reference vector $\overrightarrow{m_i} = (m_{i1}, \ldots, m_{in})$, where n is the dimensionality of input patterns $\overrightarrow{x_i} = (x_{i1}, \ldots . x_{in})$.

3.6.1 Recall

Algorithm

1. The hidden layer calculates the fuzzy rules antecedents activation ρ_j for every neuron n_j:

$$\rho_j = \min_i e^{-\frac{(x_i - m_{ij})^2}{\sigma_{ij}^2}} \tag{3.15}$$

2. The output layer defuzzifies the outputs using the weighted average defuzzification:

$$y = \frac{\sum_j w_j \rho_j}{\sum_j \rho_j} \tag{3.16}$$

3.6.2 Training

The entire training process consists of two independent phases. In the first phase the parameters for rule antecedents are to be found by the standard self-organizing maps algorithm. Then those antecedent parameters close one to another (closer than a given threshold) are merged together. After the merge, the weights to the output neuron will be obtained using a supervised learning algorithm. In the trained network, every neuron n_j in the first layer together with its weight w_j to the output of the network represents a Zadeh-Mamdami linguistic rule:

$$If\ \overrightarrow{x} =_F \overrightarrow{m_j}\ Then\ y = w_j$$

3.6.3 Discussion

Jang and Sun have proved [7] that a functional equivalence between radial basis function neural networks (RBF) and fuzzy inference systems can be established under the following conditions:

1. The number of RBF local neurons is equal to the number of the fuzzy rules.

2. The output of each fuzzy rule is a constant (zero order Takagi-Sugeno rule).

3. The membership functions within each rule are chosen as Gaussian functions with the same variance.

4. The t-norm operator used to compute the strength of the rule antecedent is multiplication.

5. Both the RBF network and the fuzzy inference system use the same method to derive the overall output (i.e. the weighted average).

The FINN network is actually only one step away from satisfying these conditions. One only needs to replace the fuzzy min-conjunction by the fuzzy product-conjunction in the first network layer. Then the same approach and methods known from the RBF networks research could be applied to the representation of the FINN-like fuzzy rules.

Chapter 4

Growing fuzzy inference neural network (GFINN)

4.1 Motivation

Using insight about FINN almost satisfying Jang&Sun conditions (section 3.6.3), together with the extensions of the SOM-model proposed in earlier chapters, a novel growing fuzzy inference neural system (GFINN) is derived in this thesis. The ANFIS model is adaptive, but requires to pre-specify a fixed number of rules to be found. It is possible, though, to start with all possible rule antecedents and prune those, that are not used, later, but this brute-force approach remains computationally infeasible for many problem domains. The FINN model, starts with a pre-specified size of the network and similar rules are merged together later on. However, the maximum number of the rules still has to be given in advance. The FINN model employs the standard SOM model in the hidden layer. As it has been already mentioned in this thesis, the standard SOMs were designed mainly for data visualization and not for data classification or function approximation.

For these reasons, in this thesis it is suggested to use the growing neural gas with the proposed enhanced output sensitivity during learning and with the developed pruning strategy to form appropriate hidden layers in RBF-like networks. When using the weighted average to compute the output of the network, the Jang&Sun conditions will be satisfied. As a consequence, the network will be equivalent to a fuzzy inference system. Networks combining all these features can grow if necessary, but remain of a stable size when the growth is considered to be completed. Furthermore, they can be interpreted in terms of fuzzy rules easily. To simplify the fuzzy rules, a method for significant input feature selection is also proposed in the following text.

Due to the nature of the fuzzy-rules, GFINN should preferably be used for function approximation or binary-valued classification.

4.2 GFINN structure

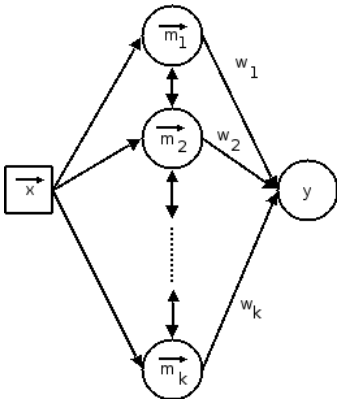


Figure 4.1: GFINN structure

GFINN is a RBF-like structure - it consists of one hidden layer and one output neuron. The hidden layer is formed by an extended growing neural gas structure (with an improved output sensitivity and with pruning applied). The output y_k known from the extended supervised GNG corresponds in this case to the weight w_j to the output neuron. Similarly like for the standard GNG model, every neuron n_k in the hidden layer has a reference vector $\overrightarrow{m_k} \in \Re^n$ and the error accumulation variable $error_k$. Every edge e has an associated age variable age_e. To support an easy features selection, every hidden neuron n_k has also a restriction set $\chi_k = \{\chi_k^1, \chi_k^2, \ldots\}$. This restriction set denotes the significant input features used by the respective neuron. In the beginning, the restriction set contains all available input features. In the last step of training, restriction sets of all neurons in the hidden layer will be reduced to contain only significant input features.

4.3 Recall

The activity of the hidden neurons corresponds to the activation of the antecedent of a fuzzy rule. The computation of the output of the network reflects the weighted average defuzzification.

GFINN recall algorithm

1. For the presented input pattern $\overrightarrow{x}$ compute the activation of hidden neurons:

$$a_k(\overrightarrow{x}) = e^{-\frac{\|\overrightarrow{x}-\overrightarrow{m_k}\|^2_{\chi_k}}{2\sigma}} \tag{4.1}$$

   using the restricted Euclidean distance computed on input features from the restriction set:

$$\|\overrightarrow{x}-\overrightarrow{m_k}\|^2_{\chi_k} = \sum_{j\in\chi_k}(x_j - m_j)^2 \tag{4.2}$$

2. Compute normalized activations on hidden neurons:

$$\nu_k(\overrightarrow{x}) = \frac{a_k(\overrightarrow{x})}{\sum_{k'} a_{k'}(\overrightarrow{x})} \tag{4.3}$$

3. Compute the output of the network using the weighted average defuzzification:

$$y(\overrightarrow{x}) = \sum_k \nu_k w_k \tag{4.4}$$

4.4 Training

The entire training process of the GFINN model consists of two phases - similarly to RBF networks and FINN model. In the GFINN model, however, they are not supposed to follow independently one another only once, but they should rather be repeated iteratively until the model is considered to be trained. This supports an improved sensitivity of the activations of neurons in the hidden layer to the output values.

GFINN training algorithm

1. Initialize the network with a very small number of hidden neurons (e.g. two neurons mutually inter-connected by an edge). Set all the output weights to the average output value computed over the training set - these weights need to be pre-set to some value, because they are involved in the own adaptation procedure. Pre-set all $\chi_k = \{1, \ldots, n\}$.

2. Train the hidden layer using the improved growing neural gas algorithm (section 2.4.4) - repeat the following steps for all training patterns:

   (a) Take the next training pattern $(\overrightarrow{x}, \widehat{y}) \in T$.

   (b) Find the nearest neuron n_{b_1} and the second-nearest neuron n_{b_2}. They are both involved in the manipulation with edges later.

   (c) Increment value of the age variable for all edges emanating from n_{b_1}:

   $$age_e(t+1) = age_e(t) + 1$$

   (d) Add the squared distance between the input $\overrightarrow{x}$ and the reference vector $\overrightarrow{m_{b_1}}$ to the local error counter of the neuron n_{b_1}:

   $$error_{n_{b_1}}(t+1) = error_{n_{b_1}}(t) + \|\overrightarrow{m_{b_1}} - \overrightarrow{x}\|^2 \qquad (4.5)$$

   (e) Move n_{b_1} and its direct neighbors n_k in the direction of $\overrightarrow{x}$:

   $$\overrightarrow{m_{b_1}}(t+1) = \overrightarrow{m_{b_1}}(t) + \alpha_b(1 - |\widehat{y} - w_{b_1}|^2)(\overrightarrow{x} - \overrightarrow{m_{b_1}}) \quad (4.6)$$

   $$\overrightarrow{m_k}(t+1) = \overrightarrow{m_k}(t) + \alpha_k(1 - |\widehat{y} - w_k|^2)(\overrightarrow{x} - \overrightarrow{m_k}) \quad (4.7)$$

   The similarity factor $(1-|\widehat{y}-w_k|^2)$ assumes $\widehat{y}, w_k \in \langle 0, 1\rangle$.

   (f) If n_{b_1} and n_{b_2} are connected by an edge, reset the age of this edge to zero. Otherwise create such an edge with an initial $age_e = 0$ value.

2. (g) Remove all edges with the age larger than age_{max}. If there is a neuron without any emanating edge after this operation, remove it as well.

   (h) If the number of steps performed so far is a multiple of the integer growing parameter λ, a new neuron is inserted:

      i. Determine the neuron n_g with the largest accumulated local error $error_g$.

      ii. Find n_f, a direct neighbor of n_g, which has the largest local error $error_f$ among the neighbors of neuron n_g.

      iii. Insert a new neuron n_r halfway between n_g and n_f:

$$\overrightarrow{m_r} = \frac{1}{2}(\overrightarrow{m_g} + \overrightarrow{m_f})$$

      iv. Replace the edge between n_g and n_f by two edges: an edge between n_g and n_r and the other edge between n_r and n_f.

      v. Decrease the error variable $error_g$ and $error_f$ by a multiplicative constant α, because the structure of the network has been altered between these two neurons and it is time for a limited ‘‘forgetting’’ of the accumulated errors. Initialize the error variable $error_r$ with the new $error_g$ value.

   (i) Prune the ε-insignificant neurons in the hidden layer (section 2.4.4).

   (j) Decrease all error variables by a multiplicative constant $d \in (0, 1)$ - the network ‘‘forgets’’ old error.

   (k) If the stopping criterion has not been met yet (network size, fixed umber of steps, ...), repeat from the second step.

3. Compute the output weights by one of the algorithms described in the following section 4.4.1.

4. Repeat steps two and three until the stop condition is met.

5. Select significant input features (section 4.4.2) - the restriction sets χ_k are altered.

A fixed amount of training cycles or a threshold of an average approximation error or classification error can be used as the stop condition.

4.4.1 Computation of output weights

To compute output weights in RBF-like networks, many methods aim at minimizing some error function, often defined as an output error of the network over the training set. The form of the objective function actually used strongly influences the computation of output weights in the network.

Least mean square error (LMSE)

One of the most common error functions is the sum of squared errors between the desired output and the output provided by the network:

$$E = \frac{1}{2} \sum_{(\overrightarrow{x},\widehat{y})\in T} \left(\widehat{y} - y(x)\right)^2 \tag{4.8}$$

Where $(\overrightarrow{x},\widehat{y})$ denotes the training pattern with the input part $\overrightarrow{x}$ and the desired output part $\widehat{y}$, $y(x)$ is the actual network output for the input $\overrightarrow{x}$ (described in the section 4.3).

For this error function it is possible to obtain the partial derivative $\frac{\partial E}{\partial w_k}$ of the error function E with respect to the output weights w_k in the following way:

$$\frac{\partial E}{\partial w_k} = -\sum_{(\overrightarrow{x},\widehat{y})\in T} \left(\widehat{y} - y(x)\right) \frac{\partial y(x)}{\partial w_k} = -\sum_{(\overrightarrow{x},\widehat{y})\in T} \left(\widehat{y} - y(x)\right) \nu_k(\overrightarrow{x}) \tag{4.9}$$

One can put this term to be equal to zero for all output weights w_k as a necessary condition of the optimal solution, yielding a set of linear equations of the following form, one equation for every w_k:

$$0 = -\sum_{(\overrightarrow{x},\widehat{y})\in T} \left(\widehat{y} - \sum_{k'} \nu_{k'} w_{k'}\right) \nu_k(\overrightarrow{x}) \tag{4.10}$$

Solving this set of linear equation by one of the standard methods gives a solution that is guaranteed to be (locally) optimal.

Gradient descent

Solving a set of linear equations can, however, be computationally expensive. On the other hand, gradient descent approach could be applied instead. The respective output weights do not have to be adapted in batch after every cycle of the hidden layer training, but after presenting every training pattern for the hidden layer adaptation.

In such a case, the output weights shall be changed in the direction opposite to the gradient in every cycle, with the aim to find the (local) minimum of the output error function:

$$w_k(t+1) = w_k(t) + (\widehat{y} - y(\overrightarrow{x}))\nu_k(\overrightarrow{x}) \tag{4.11}$$

w_k denotes the adapted weight, t is a discrete time step, $\widehat{y}$ is the desired output for the input pattern $\overrightarrow{x}$ and $y(\overrightarrow{x})$ is the actual output of the network.

Unfortunately, convergence to a minimum of the error function cannot be guaranteed because of the enduring structural changes of the network.

Fuzzy optimal output weights computation (FOPT)

To circumvent the above drawbacks, a new approach is proposed in this thesis. Similarly to standard SOM models, the hidden neurons can be considered to form clusters in the input space. However, unlike in SOMs, GFINN is based on the assumption of the fuzzy set theory - a pattern membership to a set can be expressed using a membership function. In the case of GFINN, the normalized activation $\nu_j(\overrightarrow{x})$ of the hidden neuron n_j denotes the membership of an input pattern $\overrightarrow{x}$ to a cluster corresponding to this hidden neuron. From this point of view, the output weight w_i can be considered to be independent of other weights w_j, $i \neq j$. Every hidden neuron n_j shall adapt its output weight w_j independently, as an output value attached to the respective cluster of input patterns. The input patterns with a higher value of the membership function to this cluster shall have higher influence, than the input patterns with a lower value of the membership function. The local fuzzy error E_k^{FOPT} (hence *fuzzy optimal*) of a hidden neuron n_k is computed as:

$$E_k^{FOPT} = \frac{1}{2} \sum_{(\overrightarrow{x},\widehat{y}) \in T} \nu_k(\overrightarrow{x})(\widehat{y} - w_k)^2 \tag{4.12}$$

where $\widehat{y}$ is a desired output and w_k is the output weight attached to neuron n_k.

Then the partial derivative of E_k^{FOPT} with respect to w_k can be computed

easily:

$$\frac{\partial E_k^{FOPT}}{\partial w_k} = - \sum_{(\overrightarrow{x},\widehat{y})\in T} \nu_k(\overrightarrow{x})(\widehat{y} - w_k) \tag{4.13}$$

Requiring this partial derivative $\frac{\partial E_k^{FOPT}}{\partial w_k}$ to equal to zero, it can be concluded with the following equation to determine w_k:

$$w_k = \frac{1}{\sum_{(\overrightarrow{x},\widehat{y})\in T} \nu_k(\overrightarrow{x})} \sum_{(\overrightarrow{x},\widehat{y})\in T} \nu_k(\overrightarrow{x})\widehat{y} \tag{4.14}$$

From this formula, it an be seen, that it may be feasible to use weighted averages of the output values of the training patterns as the output weights. The weights in each average correspond to the activation of the respective hidden neuron. The $O(np)$ time complexity of FOPT puts itself between the one-step gradient descent and the possibly time consuming $O(n^p + n^3)$ LMSE solution (n is a number of hidden neurons, p is a number of input patterns).

The resulting scheme differs in effect from the previous computations of output weights, because a different error function is used. The local fuzzy error functions E_k^{FOPT} do not have the global impact of the global error function E. Due to the weighted average output computation, the GFINN model can also be considered to represent a voting scheme, where every voter (hidden neuron) also estimates its own vote strength and specializes itself for a subset of the input space.

4.4.2 Selection of input features

One of the great benefits of the GFINN representation is, that it supports a straightforward method for the selection of significant input features. As a consequence, the induced rules do not have to use all the input features. The features they actually use may vary for different rules.

Algorithm

1. Compute the overall activation Θ_k of every hidden neuron n_k:

$$\Theta_k = \sum_{\overrightarrow{x} \in T} \nu_k(\overrightarrow{x}) \tag{4.15}$$

and sort the hidden neurons according to their overall activation. The most active neuron becomes n_1, the least active neuron becomes n_k. This is a heuristic used to prefer the simplification of the most commonly used rules to the rarely used ones.

2. Try the following features pruning for every hidden neuron n_k (in their sorted order):
Check for every input feature $f \in \{1, \ldots, n\}$, whether it can be omitted from the restriction set χ_k:

   (a) Compute the output of the network $y(\overrightarrow{m_k})$.

   (b) Temporarily remove feature f from the restriction set χ_k.

   (c) Create a set of verification patterns from the reference vector $\overrightarrow{m_k}$. There is one verification pattern $\overrightarrow{v_{k'}}$ for every other hidden neuron $n_{k'}$. They are same as $\overrightarrow{m_k}$, except that in every pattern the f-th coordinate of $\overrightarrow{m_k}$ is replaced by the f-th coordinate of the vector $\overrightarrow{m_{k'}}$: $\overrightarrow{v_{k'}} = (m_k^1, \ldots, m_{k'}^f, \ldots, m_k^n)$. Because of the Gaussian local nature of the hidden neurons, these verification patterns are expected to activate the other neurons most.

   (d) Compute the network output $y(\overrightarrow{v_{k'}})$ for every verification pattern $\overrightarrow{v_{k'}}$. If the difference $|y(\overrightarrow{v_{k'}}) - y(\overrightarrow{m_k})|$ exceeds a given threshold δ, the feature f cannot be pruned. If the output difference never exceeds the threshold, the feature f can be removed from the restriction set χ_k as δ-irrelevant.

Chapter 5

Supporting experiments

5.1 Artificial data

5.1.1 Data

First, the proposed methods were tested on an artificially generated data set. This data set was formed according to a few assumed rules stated below in Table 5.1. The goal of the first experiment is to verify, if the network finds rules similar to those actually used to generate the training patterns.

5.1.2 Results

Accuracy

Figure 5.1 shows the comparison of the correct output values and their approximation achieved by the GFINN network.

Extracted rules

The rules discovered by GFINN for the artificial data are in table 5.2.

1.	IF	$x_1 = low$			THEN	$y = high$
2.	IF	$x_1 = medium$	AND	$x_2 = low$	THEN	$y = high$
3.	IF	$x_1 = medium$	AND	$x_2 = medium$	THEN	$y = medium$
4.	IF	$x_1 = medium$	AND	$x_2 = high$	THEN	$y = low$
5.	IF	$x_1 = high$			THEN	$y = low$

Table 5.1: The hand-written rules used to generate the artificial dataset.

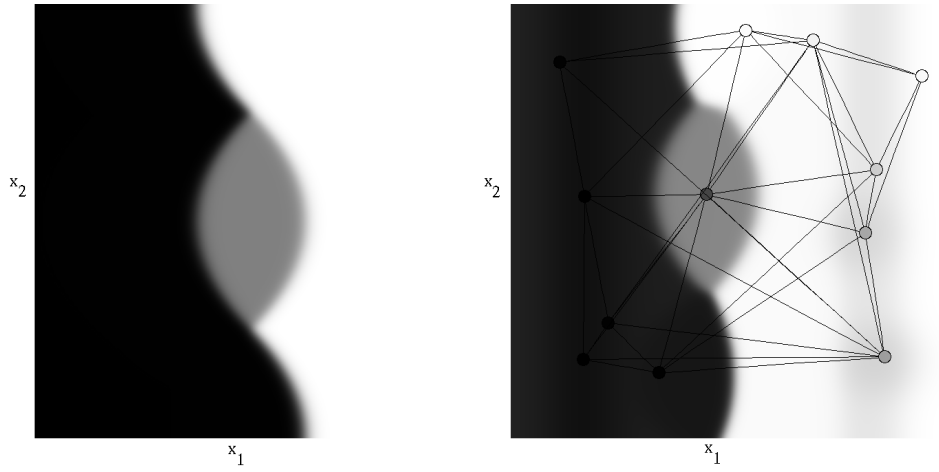


Figure 5.1: The generated artificial data (left), and their approximation by the network (right).

5.1.3 Discussion

From the comparison of the tables 5.1 and 5.2 it is obvious, that there are more discovered rules than rules used to create the dataset. On the other hand, it is also easy to spot the duplicate discovered rules. For example the second, seventh, tenth and twelfth rule in the table 5.2 - they all correspond to the first rule used to generate the dataset (tab. 5.1) - $IF\ x_1 = low\ THEN\ y = high$. After analyzing the similarity of the rules, it is possible to conclude, that the discovered rules indeed correspond to the rules underlying the data.

5.2 Real-world data

5.2.1 Data

Even though experiment performed for artificial data shows promising results, it is also very important to check the network behavior on real world data. In the real world the method needs to handle noise and incorrect values.

The GFINN network is tested on the *housing dataset* available from the UCI machine learning repository [2]. The dataset contains 506 patterns of housing values in Boston Area. It is based on the U.S. census in the year 1978. Every pattern has thirteen numerical (including the housing value) and one binary attribute:

1.	IF	$x_1 = high$			THEN	$y = low$
2.	IF	$x_1 = low$			THEN	$y = high$
3.	IF	$x_1 = medium$	AND	$x_2 = medium$	THEN	$y = medium$
4.	IF	$x_1 = medium$	AND	$x_2 = medium$	THEN	$y = medium$
5.	IF	$x_1 = medium$	AND	$x_2 = low$	THEN	$y = high$
6.	IF	$x_1 = high$			THEN	$y = low$
7.	IF	$x_1 = low$			THEN	$y = high$
8.	IF	$x_1 = medium$	AND	$x_2 = high$	THEN	$y = low$
9.	IF	$x_1 = high$			THEN	$y = low$
10.	IF	$x_1 = low$			THEN	$y = high$
11.	IF	$x_1 = high$			THEN	$y = low$
12.	IF	$x_1 = low$			THEN	$y = high$

Table 5.2: The discovered fuzzy rules for the artificial dataset.

1. CRIM ... per capita crime rate by town
2. ZN ... proportion of residential land zoned for lots over 25,000 sq. ft.
3. INDUS ... proportion of non-retail business acres per town
4. CHAS ... Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX ... nitric oxides concentration (parts per 10 million)
6. RM ... average number of rooms per dwelling
7. AGE ... proportion of owner-occupied units built prior to 1940
8. DIS ... weighted distances to five Boston employment centers
9. RAD ... index of accessibility to radial highways (lower value reflects better accessibility)
10. TAX ... full-value property-tax rate per $10,000
11. PTRATIO ... pupil-teacher ratio by town
12. B ... $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town
13. LSTAT ... % lower status of the population
14. **MEDV** ... **Median value of owner-occupied homes in $1000's**

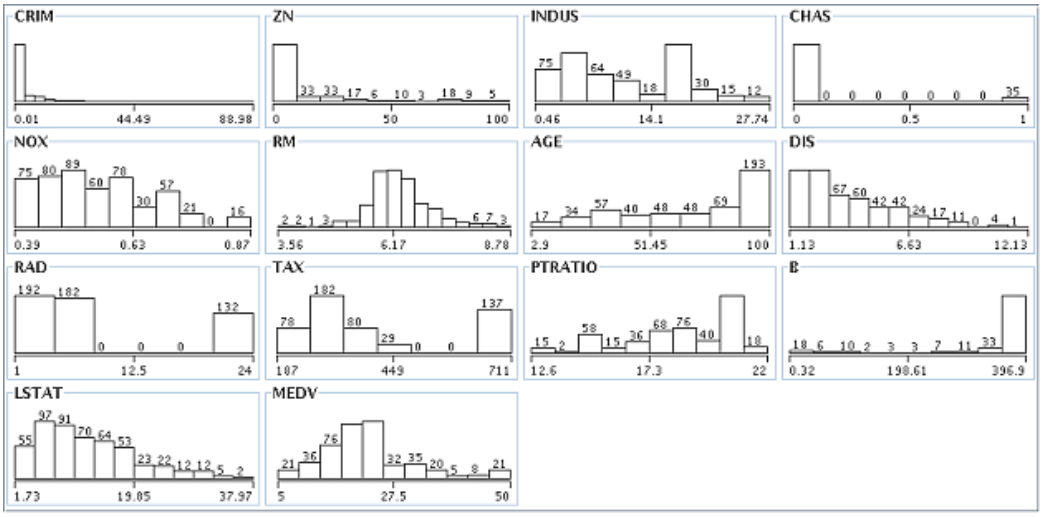


Figure 5.2: The Boston housing data values distributions

The data distribution

Figure 5.2 shows the distributions of the values in the Boston housing dataset. It is very important to note the extraordinarily high amount of houses with the $50.000 price as opposed to just slightly cheaper houses. It comes from the original data censoring, as explained in [6]. All houses costing $50.000 or more are set an artificial price of $50.000. This fact is met again later in this thesis.

[6] also discovered, that there are eight miscoded variables in the dataset. They do not correspond to the values actually collected by the census bureau. In this experiment these wrong values are left untouched, because they just represent noise in the data.

The housing dataset exhibits all properties of real world data:

- real-world data distribution
- noise (including the miscoded variables)
- utterly wrong patterns (censoring)

Correlations

Some features of the data can seem to be correlated to other. Table A in Appendix A shows the input features correlations calculated on all the patterns in the dataset.

There is a very high correlation between taxes (TAX) and the index of accessibility to radial highways (RAD) - 0.91. A high correlation can be observed between concentration of nitric oxides (NOX) and the industrialization level ($INDUS$) - 0.76 - together with a high negative correlation between the nitric oxides (NOX) and distance to employment centers (DIS) - -0.77. The predicted housing value ($MEDV$) is highly correlated with the average number of rooms (RM) - 0.71 - and highly negatively correlated with a lower status of population ($LSTAT$) - -0.74.

5.2.2 Methodology

Goal

The goal of this experiment is to predict the $MEDV$ value based on the other thirteen attribute values. The proposed growing fuzzy inference neural system is also requested to discover the fuzzy rules describing the approximation method.

Data preprocessing

Before any other processing all values are normalized into the $\langle 0, 1\rangle$ interval:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{5.1}$$

For every input variable its highest and lowest values in the dataset are used as the normalization boundaries.

Error calculation

Both the root mean squared error:

$$RMSE(F_i) = \frac{1}{|F_i|}\sqrt{\sum_{(\overrightarrow{x},\widehat{y})\in F_i}(y(\overrightarrow{x}) - \widehat{y})^2},\ F_i \subseteq T \tag{5.2}$$

and the mean absolute error:

$$MAE(F_i) = \frac{1}{|F_i|}\sum_{(\overrightarrow{x},\widehat{y})\in F_i}|y(\overrightarrow{x}) - \widehat{y}|,\ F_i \subseteq T \tag{5.3}$$

are computed for the the 10-fold cross-validation. The dataset is randomly split into ten mutually disjoint folds F_i of about the same size. In every of the ten rounds one of the folds is used as a test set while the other nine folds form together a training set.

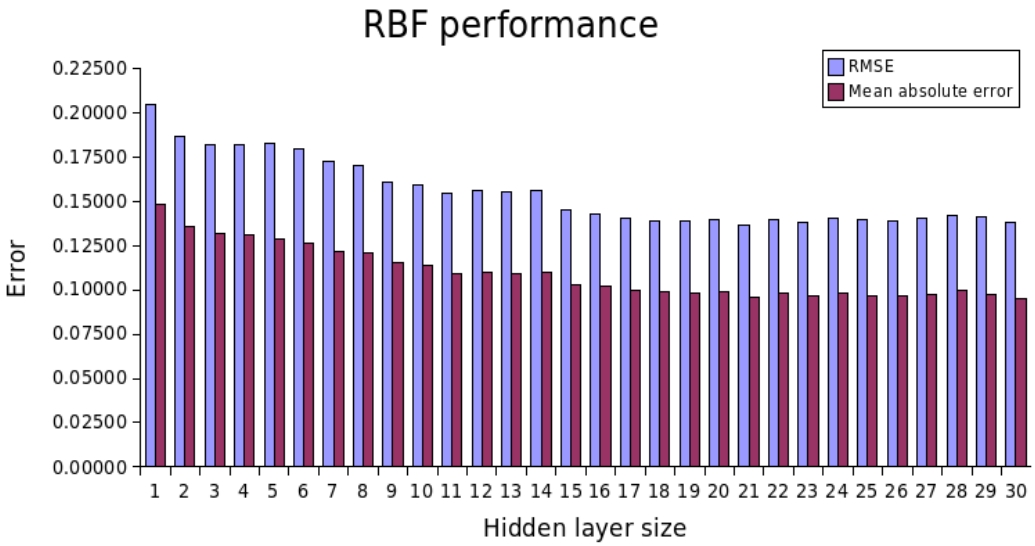


Figure 5.3: Root mean square error and mean absolute error of various sized RBF networks for the housing dataset.

5.2.3 Results

RBF performance

Because GFINN is a type of RBF-networks, which is modified for a rule extraction, it is important to realize, how much of a performance is lost or gained in comparison to the basic RBF model. Figure 5.3 shows the approximation error of normalized RBF networks with different numbers of hidden neurons. These results were obtained using the RBF network implementation in the Weka toolkit (`http://www.cs.waikato.ac.nz/ml/weka/`). This RBF network reaches a maximum performance with about thirty neurons in the hidden layer.

GFINN with LMSE and FOPT output weights computation

Parameters Both tested GFINN variants share the basic parameters. They grow after every third pass through the training set ($\lambda = 3$). The learning speed starts at $\alpha(0) = 0.9$ and decreases in every step according to $\alpha(t+1) = 0.997 * \alpha(t)$. In all cases the Gaussian widths were experimentally set to 0.05. The ε parameter of the ε-insignificance pruning is varied, while the δ parameter of the δ-irrelevant input feature pruning is set to a low value od 0.001. This δ value was selected because it allows to remove a feature from the rule, only if the change of the network output is smaller

ε	LMSE			FOPT		
	RMSE	MAE	Size	RMSE	MAE	Size
0.01	**0.1389**	**0.0989**	37	0.1399	0.1013	**36**
0.03	**0.1437**	**0.1018**	**25**	0.1458	0.1048	**25**
0.05	0.1482	0.1082	**21**	**0.1454**	**0.1035**	226
0.07	**0.1507**	**0.1071**	**18**	0.1580	0.1155	**18**
0.09	**0.1627**	**0.1204**	**10**	0.1629	**0.1204**	11
0.11	**0.1675**	**0.1229**	**8**	0.1711	0.1260	**8**
0.13	**0.1715**	**0.1263**	**7**	0.1740	0.1286	**7**
0.15	**0.1927**	**0.1392**	**4**	0.2368	0.1886	5
0.17	**0.2018**	**0.1462**	3	0.3583	0.3089	**2**

Table 5.3: GFINN performance with the least mean square error (LMSE) and FOPT output weights computation for various ε-insignificance pruning levels. The *Size* column shows the average final network size over ten folds in cross-validation.

than the resolution of the $MEDV$ values in the dataset. Both variants use the N_{bi}-based neighborhood h_{bi} with the constant distance threshold $d = 1$. The neighbors are adapted using the $\alpha_k(t) = \alpha^2(t)$ learning speed, while the best matching neuron is adapted using the $\alpha_b(t) = \alpha(t)$ learning speed.

Performance Table 5.3 shows the performance achieved by the GFINN variants with several ε values for the ε-insignificance pruning. For an easier visual comparison the result of the better GFINN variant is always highlighted by bold font. The obtained performance is fully comparable with the performance of the standard RBF network with the same size - the pressure on the GFINN model to express itself in terms of fuzzy rules does not have any negative impact.

Rules The discovered rules are visualized in Figure 5.5. Because the rules can become quite complex and unreadable, the following visualization approach is used:

- Every column shows one rule.
- Every row corresponds to a feature.
- Each feature value is shown as a height of a filled bar inside a unit-sized square.

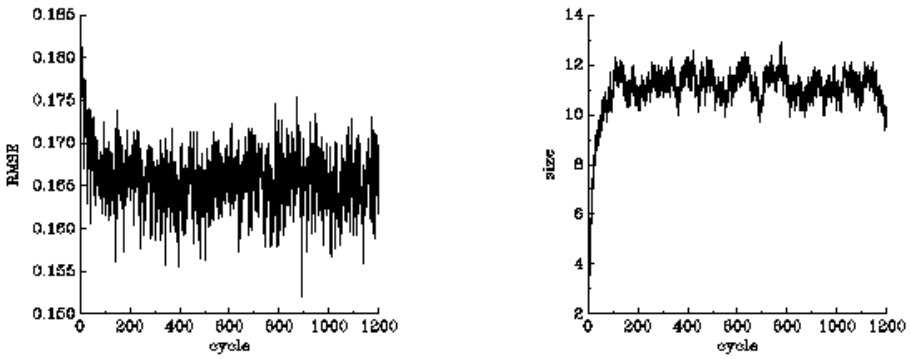


Figure 5.4: Progress of GFINN (LMSE) with 0.09-insignificance pruning learning averaged over the ten folds.

- No bar for a feature in a column means this feature is not used in that particular rule - it is considered to be insignificant.
- Because not all the rules become activated equally over the whole training set, the overall rule activation is also shown. The rules are ordered from the most active one on the left, to the least active one on the right. The background color of every rule also corresponds to this activation frequency. The more green background signifies the higher overall activation. The activation of the most active rule (the left-most one) forms a unit of measurement.

The first four rules expressed in the form of rules sound as:

1. IF $INDUS = medium$ AND $CHAS = false$ AND $NOX = low$ AND $RAD = low$ THEN $MEDV = medium$
2. IF $CRIM = low$ AND $ZN = medium$ AND $INDUS = very\ low$ AND $AGE = medium$ AND $DIS = high$ AND $RAD = very\ low$ AND $TAX = low$ AND $PTRATIO = high$ AND $B = very\ high$ AND $LSTAT = low$ THEN $MEDV = high$
3. IF $CHAS = false$ AND $RAD = very\ high$ AND $B = very\ high$ THEN $MEDV = low$
4. IF $RAD = very\ high$ AND $TAX = very\ high$ AND $B = low$ THEN $MEDV = low$

Figure 5.5: Fuzzy rules discovered for the housing dataset

5.2.4 Discussion

There is no other way how to verify the discovered rules than to use a common-sense. From this point of view the fifth rule - "If it is by the river, it is expensive" - is the easiest one. The first rule, according to its overall activity, seems to capture the most common case. This observation is further supported by its predicted lower medium $MEDV$ value. This rule says that the most "common" housing areas with a low medium housing price are away from the river, with a medium industrialization and low nitric oxides concentration, while still being easily accessible from a highway (low index of accessibility to radial highways).

The second and the third rule complement one another. The rule 2 describes areas with atypical proportion of black inhabitants, with a medium proportion of large mansions, away from the main employment centers, but accessible from a highway, with only a small proportion of lower status inhabitants. Houses in these areas are proposed to be among the more expensive. On the opposite, rule 3 describes areas with a typical proportion of black inhabitants, far away from highways and the Charles river as cheap for housing.

This "far from highways $\Rightarrow$ cheap" view is also expressed by the rule 4. According to the overall activation, the rules 5 to 9 seem to capture rare and special cases, including the most expensive living by rule 9.

Counterintuitively and unexpectedly, the average number of rooms per dwelling (RM) is never used (except for the special-case rule 9). Although the correlation shows and the intuition suggests a strong bind between the flat size and its price, the network seems to ignore the fact. Why is it so remains an open question. One common-sense hypothesis suggests that the flat price is really affected by the location more than by the size. This hypothesis does not, however, explain the high correlation between RM and $MEDV$.

An interesting fact is also revealed, when the patterns from the dataset are sorted according to the network prediction error: The set of the fifteen worst approximated housing patterns is almost identical to the set of housing patterns with the upper bound housing value ($MEDV = \$50.000$). In all cases the neural network underestimates the housing value. It may be caused by the fact, that these values are both extreme and rare. There are, however, other patterns in the dataset, that have only a slightly lower $MEDV$ value, are even rarer, but still do not appear in the worst prediction set. Another hypothesis may be, that the network may have troubles with the fact, that these upper-scaled housing values were censored [6]. They may lack regularities underlying other patterns. Other authors also stumbled upon this problem with their methods, some of them had even removed these patterns from the considered data, claiming them "wrong" [24].

An interesting question remains, whether the discovered rules will be the same or at least similar for slightly changed training sets. Unfortunately, already the first visual inspections of the rules obtained during 10-fold cross-validation reveals, that the extracted rules can differ - for example no rule similar to the fifth one appears for the second fold.

Only the results obtained for LMSE and FOPT are shown in Table 5.3. Results for the gradient descent method computation of the output weights are omitted because of a very low performance. The algorithm was very sensitive to the choice of the learning speed parameter α. The underlying hidden layer changes much faster, than the gradient descent can react to. It could, however, be possible to use the batch mode for gradient descent.

When comparing the two methods - LMSE and FOPT, LMSE usually yields better results. But while providing slightly worse results, FOPT requires much less processing time. Another advantage of the FOPT approach consists in the form of values for the output weights, which can be used immediately as the consequents of the fuzzy rules. LMSE, while being optimal with respect to the training set, sometimes offers output weights that are out of the interval $\langle 0, 1 \rangle$ allowed for the output values. For the considered training set, these "out-of-bounds" consequents may not become fully activated and they may be compensated for by other rules. Unfortunately, this behavior cannot be guaranteed outside of the training set and may be considered to be unacceptable for many tasks. The FOPT method, on the other hand, can never induce rule consequents outside of the interval allowed for the output values.

Chapter 6

Summary

6.1 Summary

In this thesis, a new model of the growing fuzzy inference neural network has been proposed. It is based on the idea of growing neural gas networks, improved by a strategy for nodes pruning and in increased sensitivity to adequate output classes. Further, this system can be treated as a fundament inducing a transparent system of fuzzy rules. This combination provides the network with an unmatched capability of finding the size necessary for the task at hand autonomously first and explaining how the network works in terms of fuzzy rules later on. The fuzzy-set-theory point of view also motivates the new FOPT algorithm for the computation of the output weights, which has been shown to yield almost optimal results in much shorter time, than e.g. the computation of the optimal weights performed by the least mean squares estimations. The output weights computed by FOPT are guaranteed to stay within the output values interval, unlike the output weights computed by LMSE. The FOPT method can be also parallelized easily, supporting the ultimate idea of distributed information processing in neural networks. A new technique for pruning of irrelevant features in the antecedents of the rules has been introduced too.

The proposed GFINN model has been tested both on the artificial and the real-world datasets. The experiments withe the artificial dataset experiment reveal, that the new model of neural networks is capable of discovering the knowledge and structure underlying the training data and that the extracted knowledge can be interpreted in the form of fuzzy rules. The experiments with the Boston housing data tested the ability of the model to deal with larger and noisy data. Also in this case, the network succeeded in discovering a small amount of understandable fuzzy rules describing the housing data.

6.2 Future work

Despite of several advantages of the introduced model summarized in the preceding section, important areas worth of improvements can be also seen - namely the correct position of hidden neurons and the form of the induced fuzzy rules. New neurons, which are created in the areas of high output error, tend to migrate to areas with high data density. This artifact has been inherited from the vector quantization property of original self-organizing maps. The techniques introduced in this thesis restrict this behavior, but are not capable of avoiding it completely. On the other hand, it also remains an open question, whether it is a good idea to get rid of vector quantization. Even small discrepancies in the input space areas with a high data density may impact a high overall error at the output of the network.

When the housing rules obtained for different runs of the model during cross-validation are compared, they clearly differ from run to run completely. Obviously, this network belongs to "high-variance" models, such as decision trees. From this point of view, it might be worth it to compare the GFINN model with fuzzy decision trees [17] with regard to training speed and the sensitivity to the noise in training patterns.

Appendices

Appendix A

Correlations of the housing data features

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
MEDV	-0.39	0.36	-0.48	0.18	-0.43	**0.70**	-0.38	0.25	-0.38	-0.47	-0.51	0.33	**-0.74**
LSTAT	0.46	-0.41	0.60	-0.05	0.59	-0.61	0.60	-0.50	0.49	0.54	0.37	-0.37	
B	-0.39	0.18	-0.36	0.05	-0.38	0.13	-0.27	0.29	-0.44	-0.44	0.18		
PTRATIO	0.29	-0.39	0.38	-0.12	0.19	-0.36	0.26	-0.23	0.46	0.46			
TAX	0.58	-0.31	**0.72**	-0.04	0.67	-0.28	0.51	-0.53	**0.91**				
RAD	0.63	-0.31	0.60	-0.01	0.61	-0.21	0.46	-0.49					
DIS	-0.38	0.66	**-0.71**	-0.01	**-0.77**	0.21	**-0.75**						
AGE	0.35	-0.57	0.64	0.09	**0.73**	-0.24							
RM	-0.22	0.32	-0.39	0.09	-0.30								
NOX	0.42	-0.52	**0.76**	0.09									
CHAS	-0.06	-0.04	0.06										
INDUS	0.41	-0.53											
ZN	-0.2												

Appendix B

Contents of the enclosed CD

Directory	Content
`animations`	animations of the training of growing grid and growing neural gas for the sample data
`data`	artificial and housing datasets
`source`	source codes of the neural networks (not a part of this thesis)

Bibliography

[1] G. A. Carpenter and S. Grossberg. *Pattern Recognition by Self-Organizing Neural Networks, 1st edition*. MIT Press, Cambridge, MA, USA, 1991.

[2] C. L. Blake D. J. Newman, S. Hettich and C. J. Merz. UCI repository of machine learning databases, 1998.

[3] B. Fritzke. Fast learning with incremental RBF networks. *Neural Processing Letters*, 1(1):2–5, 1994.

[4] B. Fritzke. Growing grid - a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, 2(5):9–13, 1995.

[5] B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, Cambridge MA, 1995.

[6] O. W. Gilley and R. Kelley Pace. On the Harrison and Rubinfeld data. *Journal of Environmental Economics and Management*, 31:403–405, 1996.

[7] J.-S. Roger Jang and C.-T. Sun. Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Transactions on Neural Networks*, 4:156–159, 1993.

[8] Jyh-Shing Roger Jang. Anfis: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man & Cybernetics*, 23:665–685, 1993.

[9] S. Kaski and T. Kohonen. Exploratory data analysis by the self-organizing map: Structures of welfare and poverty in the world. In

Apostolos-Paul N. Refenes, Yaser Abu-Mostafa, John Moody, and Andreas Weigend, editors, *Neural Networks in Financial Engineering. Proceedings of the Third International Conference on Neural Networks in the Capital Markets, London, England, 11-13 October, 1995*, pages 498–507. World Scientific, Singapore, 1996.

[10] T. Kohonen. *Self-organizing maps, Third extended edition.* Springer, 2001.

[11] J. Laaksonen, M. Koskela, and E. Oja. PicSOM: Self-organizing maps for content-based image retrieval. In *Proc. of International Joint Conference on Neural Networks (IJCNN'99), Washington, D.C., USA, July 10–16*, 1999.

[12] K. Lagus, T. Honkela, S. Kaski, and T. Kohonen. WEBSOM for textual data mining. *Artificial Intelligence Review*, 13(5/6):345–364, December 1999.

[13] Chin-Teng Ling and C. S. G. Lee. *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems.* Prentice Hall; Bk&Disk edition, May 1996.

[14] M. Martinetz and K. J. Schulten. A "neural-gas" network learns topologies. 1:397–402, 1991.

[15] Takatoshi Nishina and Masafumi Hagiwara. Fuzzy inference neural network. *Neural Computing*, 14(3):223–239, 1997.

[16] J. O'Keefe and J. Dostrovsky. The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, 34:171–175, 1971.

[17] C. Olaru and L. Wehenkel. A complete fuzzy decision tree technique. *Fuzzy Sets and Systems*, 138:221–254, September 2003.

[18] P.-Y. Oudeyer. The self-organization of speech sounds. *Journal of Theoretical Biology*, 233:435–449, 2005.

[19] P.-Y. Oudeyer. *Self-Organization in the Evolution of Speech.* Oxford University Press, 2006.

[20] S. Russel and P. Norvig. *Artificial Intelligence - A Modern Approach, Second Edition.* Prentice Hall, 2003.

[21] T. Strösslin, D. Sheynikovich, R. Chavarriaga, and W. Gerstner. Robust self-localisation and navigation based on hippocampal place cells. *Neural Networks*, 18:1125–1140, November 2005.

[22] T. Takagi and M. Sugeno. Derivation of fuzzy control rules from human operators control actions. *Proc. of the IFAC Symp. on Fuzzy Information, Knowledge Representation and Decision Analysis*, pages 55–60, July 1983.

[23] Constantin von Altrock. *Fuzzy logic and neurofuzzy applications explained*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

[24] B. Weber. `http://www.kellogg.northwestern.edu/faculty/weber/decs-439B/Boston.htm`.

[25] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[26] L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3:28–44, January 1973.