VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

KLASIFIKACE VZORŮ POMOCÍ FUZZY NEURONOVÝCH SÍTÍ
FUZZY NEURAL NETWORKS FOR PATTERN CLASSIFICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE AUTHOR	Bc. Tamás Ollé
VEDOUCÍ PRÁCE SUPERVISOR	Ing. Jitka Svobodová

BRNO, 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor
Elektronika a sdělovací technika

Student:	Bc. Tamás Ollé	***ID:***	83398
Ročník:	2	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Klasifikace vzorů pomocí fuzzy neuronových sítí

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte principy fuzzy logiky a umělých neuronových sítí. Navrhněte neuronovou síť, která je kombinací fuzzy systému a zvoleného typu neuronové sítě.

Vytvořte učební množinu pro neuronovou síť. Navržený systém naprogramujte v prostředí MATLAB s použitím toolboxu Parallel Computing Toolbox a otestujte na úloze klasifikace vzorů.

Získané výsledky porovnejte s výsledky získanými pomocí vybraných typů neuronových sítí bez použití fuzzy logiky.

DOPORUČENÁ LITERATURA:

[1] VASILIC, S. Fuzzy neural network pattern recognition algorithm for classification of the events in power system networks [online]. Texas A&M University, 2004 – [cit. 18.12.2009]. Dostupné na www: http://handle.tamu.edu/1969.1/436.

[2] DRÁBEK, O., SEIDL, P., TAUFER, I. Umělé neuronové sítě – základy teorie a aplikace. Chemagazín. 2005 (4) s. 32-34

Termín zadání:	6.2.2012	***Termín odevzdání:***	18.5.2012

Vedoucí práce: Ing. Jitka Svobodová
Konzultanti diplomové práce:

prof. Dr. Ing. Zbyněk Raida
Předseda oborové rady

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení:	Bc. Tamás Ollé
Bytem:	Tajovského 9, Komárno, 945 01, Slovenská Republika
Narozen/a (datum a místo):	7. května 1986 v Komárne

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Technická 3058/10, Brno, 616 00
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
prof. Dr. Ing. Zbyněk Raida, předseda rady oboru Elektronika a sdělovací technika
(dále jen „nabyvatel“)

Čl. 1

Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):

   ☐ disertační práce
   ☒ diplomová práce
   ☐ bakalářská práce
   ☐ jiná práce, jejíž druh je specifikován jako ....
   (dále jen VŠKP nebo dílo)

Název VŠKP:	Klasifikace vzorů pomocí fuzzy neuronových sítí
Vedoucí/ školitel VŠKP:	Ing. Jitka Svobodová
Ústav:	Ústav radioelektroniky
Datum obhajoby VŠKP:	__________________

   VŠKP odevzdal autor nabyvateli[*]:

   ☒ v tištěné formě – počet exemplářů: 2
   ☒ v elektronické formě – počet exemplářů: 2

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.

3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

[*] hodící se zaškrtněte

Článek 2

Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.

2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.

3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti

   - [✓ ihned po uzavření této smlouvy]
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy
     (z důvodu utajení v něm obsažených informací)

4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3

Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.

2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.

3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.

4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: 18. května 2012

……………………………………….. ……………………………………………

Nabyvatel Autor

ABSTRAKT

Práce popisuje základy principu funkčnosti neuronů a vytvoření umělých neuronových sítí. Je zde důkladně popsána struktura a funkce neuronů a ukázán nejpoužívanější algoritmus pro učení neuronů. Základy fuzzy logiky, včetně jejich výhod a nevýhod, jsou rovněž prezentovány. Detailněji je popsán algoritmus zpětného šíření chyb a adaptivní neuro-fuzzy inferenční systém. Tyto techniky poskytují efektivní způsoby učení neuronových sítí.

KLÍČOVÁ SLOVA

neuron, umělé neuronové sítě, akční potenciál, algoritmus zpětného šíření chyb, fuzzy logika, fuzzy-neuronová síť, adaptivní neuro-fuzzy inferenční systém

ABSTRACT

This work describes the principle of operation of neurons and how they form artificial neural networks. The structure and the operation of neurons are thoroughly described and the most widely used algorithm for neuron training is shown as well as the basics of fuzzy logic including its advantages and disadvantages. This work fully describes the backpropagation algorithm and the adaptive neuro-fuzzy inference system. These techniques provide effective methods of neural network learning.

KEYWORDS

neuron, artificial neural networks, action potential, backpropagation algorithm, fuzzy logic, fuzzy neural network, adaptive neuro-fuzzy inference system

OLLÉ, T. *Klasifikace vzorů pomocí fuzzy neuronových sítí*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav radioelektroniky, 2012. 46 s., 6 s. příloh. Diplomová práce. Vedoucí práce: Ing. Jitka Svobodová

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma Klasifikace vzorů pomocí fuzzy neuronových sítí jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny uvedeny v seznamu literatury na konci práce.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Jitky Svobodové za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne ....

(podpis autora)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

A fuzzy system is an alternative to traditional concepts of set membership and logic. Although its basics originate from the ancient Greek philosophy, it is a relatively new field, and as such, leaves much room for development and applications at the leading edge of artificial intelligence. Within this work, I try to present the foundations of neural networks along with some of the more remarkable difficulties to its use with examples from the field of artificial intelligence.

Modern techniques of artificial intelligence can be found in almost all fields of the human science, however, the biggest usage is in engineering field. The “neuro-fuzzy” approach was born as a combination of artificial neural networks and fuzzy logic. These two techniques are often used together for solving engineering problems, where classic methods are not able to provide a straightforward or correct solution. Generally, the neuro-fuzzy term means a type of system characterized for a similar structure of a fuzzy controller where the fuzzy sets and rules are adjusted using neural networks’ tuning techniques in an iterative way with data vectors (input and output system data) [1].

Two different processes take place in such systems. The first is called the learning phase, where neural networks adjust their internal parameters. The second, implementation phase behaves like a fuzzy logic system. The combination of these two techniques is likely to produce better results than the two techniques applied separately.

Within this work, an own neural network will be built in Matlab, using the presented techniques. A neural network for voice recognition will be programmed. The goal of the project is to apply these specific techniques on particular examples, and to analyze and present the differences between them.

2. NEURAL NETWORKS

The basic conception behind the neural net is to simulate the biological functions of the human brain. The human brain consists of about 100 billion processing units connected together in just such a network. These processing units are called “brain cells” or “neurons” and each one is a living cell [2]. The main characteristic of the neural network is the fact, that these structures can learn with examples (training vectors, input and output samples of the system). The neural networks modifies its internal structure and the weights of the connections between its artificial neurons to make the mapping, with a level of acceptable error for the application of the relation input/output that represent the behavior of the modeled system [1].

The advantages of the neural networks are:

- learning capacity
- generalization capacity
- robustness in relation to disturbances

The disadvantages of the neural networks are:

- impossible interpretation of the functionality
- difficulty in determining the number of layers and number of neurons

2.1 Real brains

Real neurons are much too small to see directly, but we can have a look at it under a microscope (Figure 2.1).

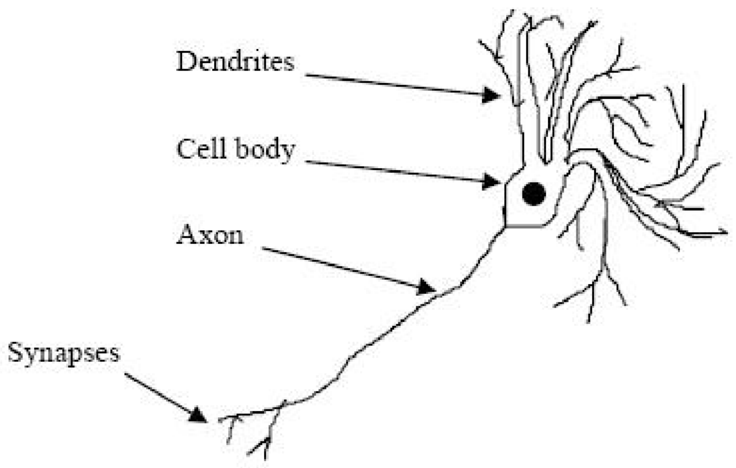


Fig. 2.1 A biological neuron ([2])

The main component parts of the biological neuron are:

- Dendrites – short tips of the neuron with centripetal type, which receives information from the outside world (if the neuron is a sensory one)
- Cell body – the bulbous end of a neuron, which contains the cell nucleus (mechanism that keep the cell alive)
- Axon – conducts electrical signals to other neurons, or to muscles or glands

The input information to the body is processed by neurons. The light sensors in our eyes (called rods and cones) are neurons in which the dendrites are stimulated by light. Under our skin, there are pressure sensing neurons, heat sensors, pain sensors and a bunch of other neurons, which help us to detect the outside world around us. The moving of our muscles is also stimulated by motor neurons. By looking at the Figure 2.2 you can get a closer look at the process.

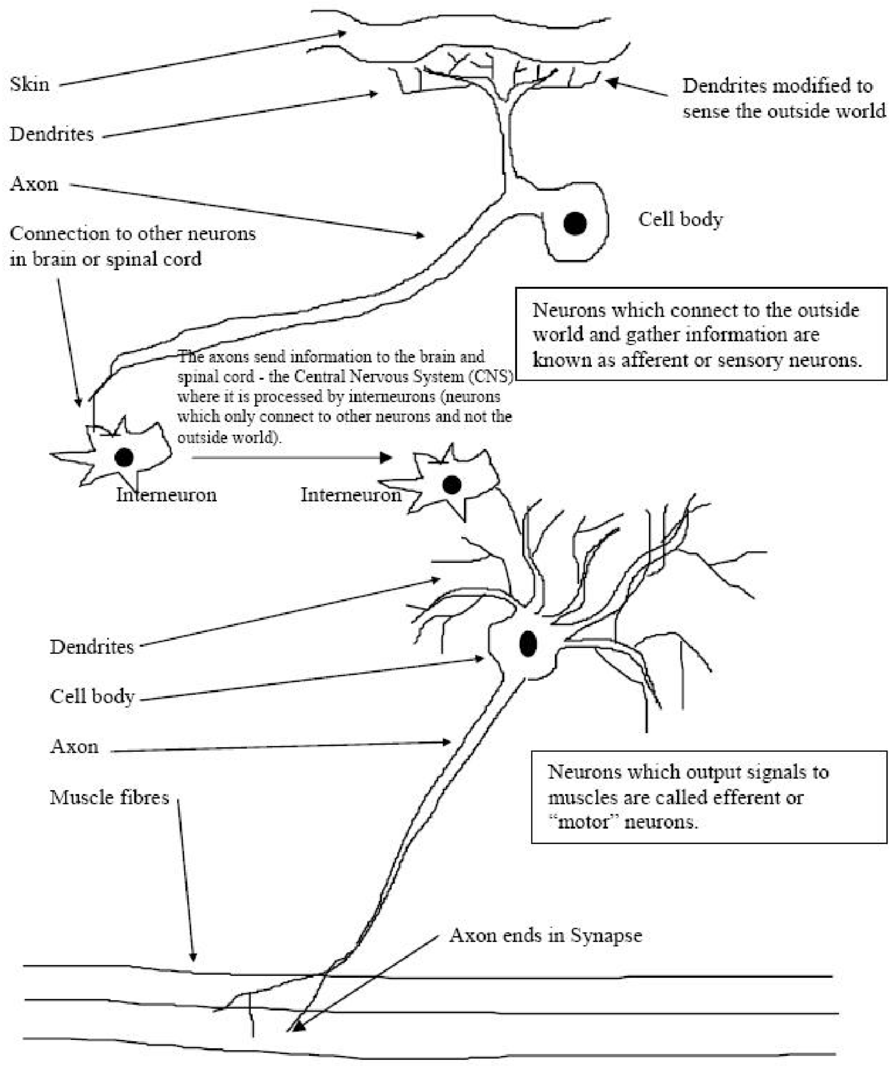

Skin
Dendrites
Axon
Connection to other neurons in brain or spinal cord
Dendrites modified to sense the outside world
Cell body
Neurons which connect to the outside world and gather information are known as afferent or sensory neurons.
The axons send information to the brain and spinal cord - the Central Nervous System (CNS) where it is processed by interneurons (neurons which only connect to other neurons and not the outside world).
Interneuron
Interneuron
Dendrites
Cell body
Axon
Muscle fibres
Neurons which output signals to muscles are called efferent or "motor" neurons.
Axon ends in Synapse

Fig. 2.2 Body function control by neurons ([2])

The input information goes through the long axons of the sensory neurons into the spinal cord and brain. There they are connected to other neurons (called interneurons). Finally, the result of the processing is passed to the output neurons which stimulate muscles or glands to affect the outside world. This mechanism is responsible for all our actions from simple reflexes to consciousness itself [2].

2.2 Operation of neurons

After reviewing how the neurons form a network, we need to understand the function of each individual neuron. When a neuron is stimulated by another neuron (or by outside influences in case of sensory neurons), it produces pulses, called "action potentials".

Before a neuron becomes stimulated (at its poise), it is polarized. This means that, neuron is charged up and ready to produce electrical pulse. Each neuron has associated with it a level of stimulus, above which a nerve pulse or action potential will be generated. Only when it receives enough stimulation, from one or more sources it will initiate a pulse – which travels a couple of hundred meters per second [2].

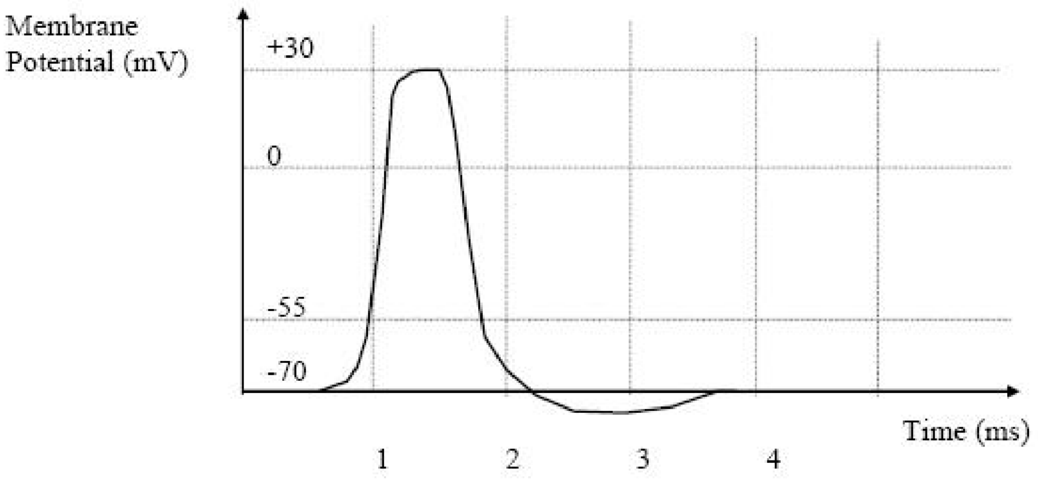


Fig. 2.3 The action potential ([2])

With the help of an oscilloscope, we are able to monitor these pulses. Each pulse is only a couple of milliseconds wide. By increasing the stimulation, the density of impulses will increase as well. It means more pulses per second.

2.3 Learning

Spot where the end of the axon meets the dendrites of the next neuron is called the Synapse, and it is important to the functioning of the neuron and to learning [2]. The enlargement of this area is illustrated in Figure 2.4.

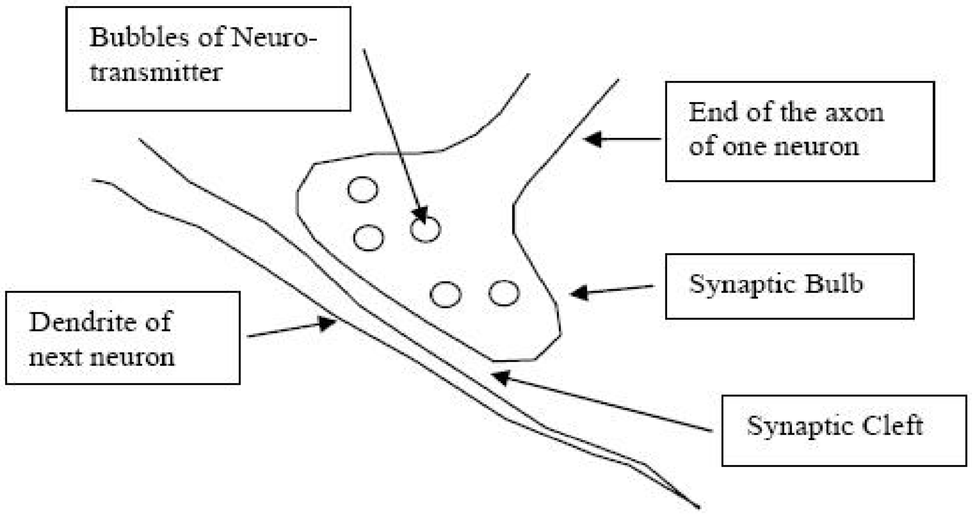


Fig. 2.4 The synapse ([2])

The end of the axon is called the synaptic bulb. Between this and the next cell is a few tens of nanometers wide gap, called the synaptic cleft. When the action potential reaches the end of the axon, it stimulates the release of chemicals called neurotransmitters, which are present in the synaptic bulb. These cross the cleft and stimulate the next cell [2]. As more often the synapse is used, the stronger it gets.

2.4 Artificial neural networks

The history of artificial neural networks goes back to 1943, when Warren McCulloch and Walter Pitts designed a simple artificial model of neuron. Most of the artificial neural networks are based on their model up to this day.

The Artificial Neural Network (neural net or ANN) is a collection of simple processors connected together [2]. It is actually a simplified mathematical model of brain-like systems. Each processor can only perform a very simple mathematical function by its own, but with a large network of them, we can achieve much greater capabilities and do many things. The basic conception is presented in Figure 2.5.

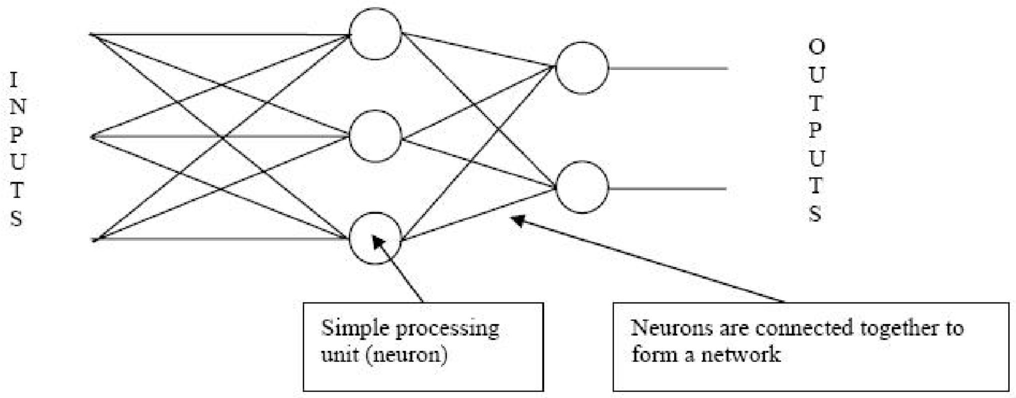

INPUTS
OUTPUTS
Simple processing unit (neuron)
Neurons are connected together to form a network

Fig. 2.5 A neural net with simple processors connected together ([2])

The most important advantage of neural networks is probably their adaptivity, which allows to perform well even at situations when the system or the environment being controlled varies over time.

2.4.1 The basic Artificial Neuron

A basic artificial neuron is shown in Figure 2.6. Individual markings have the following meaning:

i	…inputs to the neuron
w	…represents the strength of the synaptic connection of its dendrite
S	…activity or activation of the neuron (sum of the inputs and their weights)

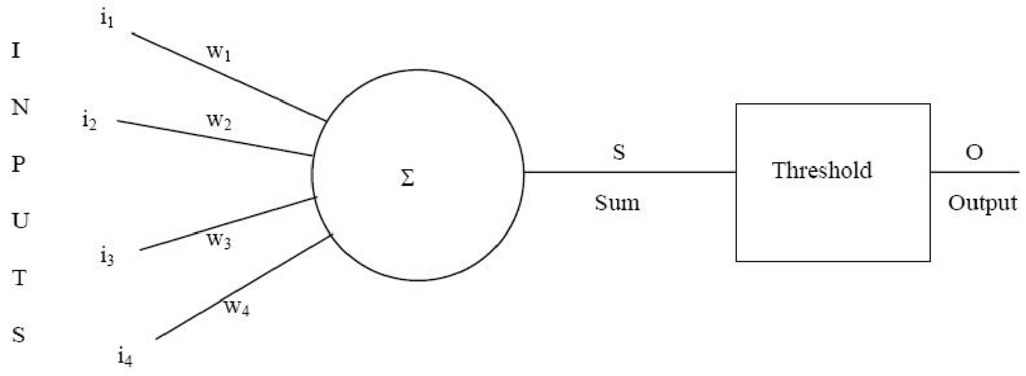


Fig. 2.6 A basic artificial neuron ([3])

Mathematical expression of artificial neuron is the following:

$$S = i_1 w_1 + i_2 w_2 + i_3 w_3 + i_4 w_4 \tag{2.1}$$

After the summary, a threshold (set at 0.5) is applied in a simple binary level:

$$\text{if } S > 0.5 \text{ then } O = 1$$
$$\text{if } S < 0.5 \text{ then } O = 0 \tag{2.2}$$

Described in words: the neuron takes its inputs and weights them according to the strength of connection. If the total sum of the weighted inputs is more than the previously defined threshold, the neuron produces a pulse (just like the biological one).

Artificial Neural Networks used simple binary outputs at an early stage, but later than switched to continuous output function, because it was more flexible. One example is the Sigmoid function:

$$O = \frac{1}{1+e^{-S}} \tag{2.3}$$

This function always produces an output between 0 and 1 that is why it is often called activation function. Other activation functions (linear, logarithmic, and tangential) are also used sometimes; however, the Sigmoid function is probably the most common. The biggest difference between threshold and Sigmoid function is that in the threshold case, the output changes suddenly from 0 to 1. In sigmoid case, the change from 0 to 1 happens gently – this helps the neuron to express uncertainty. Figure 2.7 compares the difference.

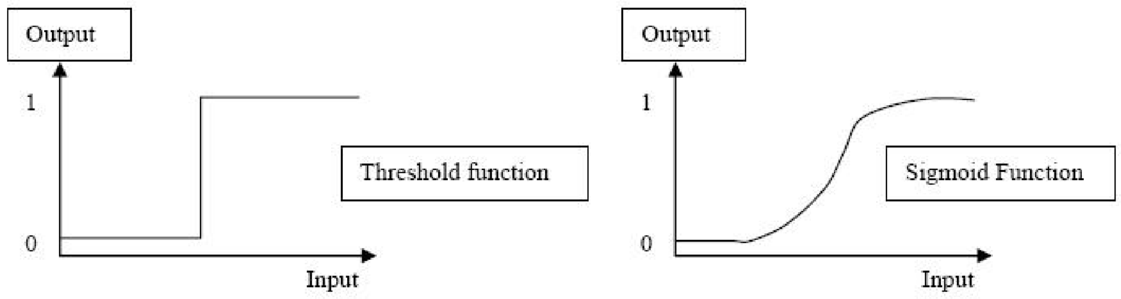


Fig. 2.7 Threshold and Sigmoid function ([3])

Earlier formula (2.1) may be formalized for a neuron of *n* inputs:

$$S = i_1 w_1 + i_2 w_2 + \ldots + i_n w_n \tag{2.4}$$

Generally:

$$S = \sum_{x=1}^{x=n} w_x i_x \tag{2.5}$$

Or, if the inputs are considered as forming a vector $\overline{I}$, and the weights a vector or matrix $\overline{W}$ [3]:

$$S = \overline{I} \cdot \overline{W} \qquad (2.6)$$

3. BACKPROPAGATION ALGORITHM

After overviewing the basics of neural networks in the previous chapters, let´s have a look at some practical networks, their applications and how they are trained.

Many hundreds of neural network types have been suggested over the years; however, there are only a small group of widely uses, so-called “classic” networks, on which many others are based. These networks are: backpropagation, Hopfield networks, competitive networks and networks using spiky neurons. There are even more variations on these themes. This chapter will deal with the algorithm called backpropagation.

3.1 The algorithm

Probably the most common way to connect neurons with sigmoid activation function are multilayer nets. Multilayer neural network with one inner neural layer (neurons are marked Z_j, $j = 1,...,p$) is shown in Figure 3.1. Output neurons (neurons are marked Y_k, $k = 1,...,m$). Neurons in output and inside layers must have a defined bias. Typical marking of the bias of the k^th neuron (Y_k) in the output layer is w_{0k} and typical marking of bias of the j^th neuron (Z_j) in the inside layer is v_{0j}. Bias (e.g. j^th neuron) matches weighted value of the assigned connection between the given and fictional neuron, whose activation is always 1. From the displayed picture then ensue, that a multilayer neural network is created minimally by three layers of neurons: input, output and at least one inside layer. Between two neighbour layers can always be found a so called *complete neural connection*, so each neuron of lower layer is connected with each neurons of higher layer.

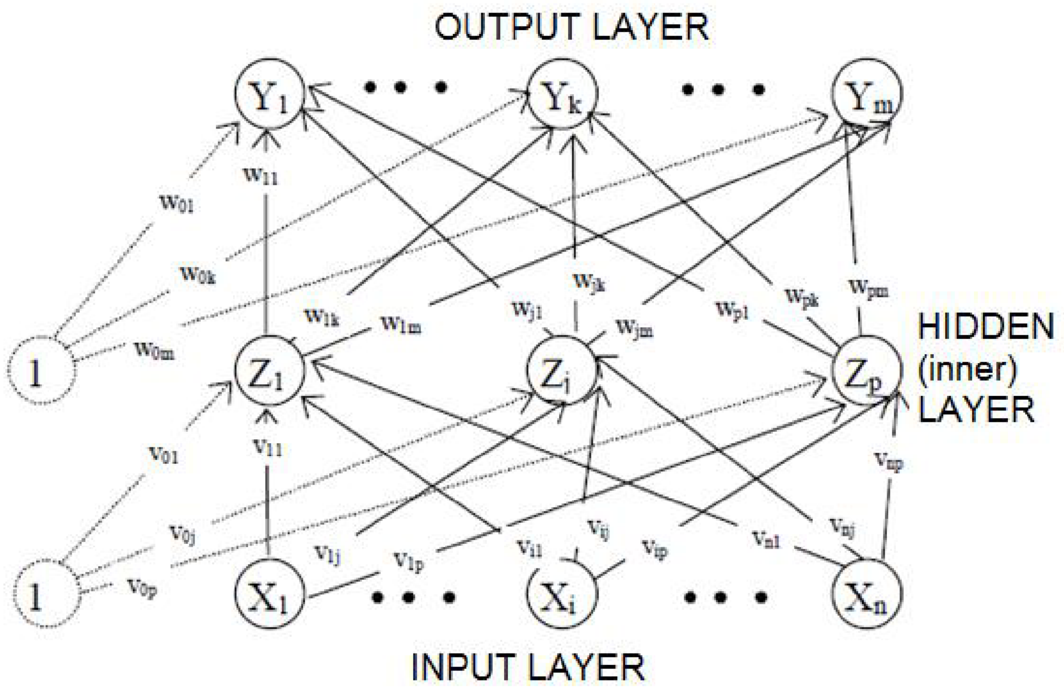


Fig. 3.1 Neural network with one inner neural layer ([9])

Backpropagation algorithm is used in approximately 80% of all neural network applications. Algorithm itself includes three periods: feedforward spreading of the input signal of training pattern, backward spreading of errors and actualization of weighted values on connections.

During feedforward signal spreading, each neuron in the input layer (X_i, i = 1,...,n) receives input signal (x_i) and mediates its transfer to all neurons in the inner layer (Z_1..., Z_p). Each neuron in the inner layer calculates its activation (z_j) and sends this signal to all the neurons in the output layer. Each neuron in the output layer calculates its activation (y_k), which matches its real output (kth neuron) after submission of the input sample.

In principle, in this way, we obtain the response of neural net on the input stimulus given by excitation of input layer neuron. Signal spreading in biological system proceeds in such a way too, where input layer can be created e.g. with visual cells and in the output layer of the brain are then identified individual objects of watching. The question then will be, how synaptic weights leading to correct response on the input signal are defined. The process of determining the synaptic weights is linked again with the concept of learning the neural networks.

Another issue is the ability of *generalization* over the learned material, in other words, how the neural network is able to deduce on the basis of learned phenomea that were not part of the learning process, but can somehow be deduced from the learned.

What is needed for learning the neural network? It is both the *training set* containing elements describing the solved problem and then a method that can fix these samples in the form of neural network synaptic weight values, including the already mentioned ability to generalize, if possible. Stop first at the training set. Each training set pattern describes, how neurons are excited in the input and output layers. Formally, for the training set T we can consider set of elements (patterns) that are arranged in pairs defined as follows:

$$\begin{aligned} T &= \{\{S_1, T_1\}\{S_2, T_2\} \dots \{S_q, T_q\}\} \\ S_i &= [s_1\ s_2 \dots s_n] \qquad s_j \in \langle 0,1 \rangle \\ T_i &= [t_1\ t_2 \dots t_m] \qquad t_j \in \langle 0,1 \rangle \end{aligned} \tag{3.1}$$

where	q	number of training set patterns
	S_i	excitation vector of the input layer consisting of *n* neurons
	T_i	excitation vector of the output layer consisting of *m* neurons
	s_j, t_j	excitation of the jth neuron of the input, respectively the output layer

The method that allows the adaptation of the neural network training set is called backpropagation. This method is an adaptation in the opposite direction of the spread of information from higher layers to lower layers.

During the neural network adaptation with backpropagation method, calculated activation y_k with defined output values t_k for each neuron in the output layer and for each training pattern are compared. Based on this comparison, the neural network error is defined, for which factor δ_k (k = 1, ..., m) is calculated. δ_k is, as it was already mentioned, the part of error that spreads back from the neuron Y_k to all the neurons of previous layers which are defined with neuron connections. Factor δ_j (j = 1, ..., p) can be defined similarly, which is a part of errors spreads back from neuron Z_j to all the input layer neurons, which are defined with the neuron connections.

Weight value adjustment w_{jk} on the connections between neurons in the inner and output layers depends on factor δ_k and the activation of Z_j neuron in the inner layer. Weight value adjustment v_{ij} on the connections between neurons in the input and inner layers depends on factor δ_j and the activation of X_i neuron in the input layer.

The activation function for neural neworks with adaptive backpropogation method must have the following characteristics: it must be continuous, differentiable and monotonically nondecreasing. The most commonly used activation function is therefore standard (logical) sigmoid and hyperbolic tangent. Network error E(w) is due to the training set defined as the sum of the partial network error $E_l(w)$ due to individual training patterns and depends on the network confugiration w:

$$E(w) = \sum_{l=1}^{q} E_l(w) \tag{3.2}$$

Partial network error $E_l(w)$ for the lth training pattern (l=1, ...,q) is proportional to the sum of squared deviations of actual output values of the network input for l-training pattern from the required output values for this example:

$$E_l(w) = \frac{1}{2}\sum_{k \in Y}(y_k - t_k)^2 \tag{3.3}$$

The aim of adaptation is to minimize network errors in the weight space. Since the fault of the network directly depends on a complicated nonlinear complex function of a multilayer network, the goal presents a non-trivial optimalization problem. For its solution, the basic model uses the simplest version of gradient method, which requires differentiability of the error function. Geometric conception will help us in better understanding.

The error function E(w) is schematically shown in Figure 3.2 – configuration, which is a multidimensional vector of weights w, is projected on the axis of x. Error function determines the network error due to fixed training set, depending on network configuration. During the network adaptation, we are looking for a configuration, for which the error function is minimal. We start with a randomly chosen configuration $w^{(0)}$, where the corresponding network error from the desired network will probably be large. In analogy with human learning, it corresponds to the initial settings of synaptic weights of the newborn, who instead of the desired behaviors such as walking, talking, etc. performs random movements and makes vague noises. During the adaptation, we frame at this point $w^{(0)}$ tangent vector (gradient) $\frac{\partial E}{\partial w}(w^{(0)})$ and move in the direction of this vector down by ε. For sufficiently small ε then we obtain the new configuration $w^{(1)} = w^{(0)} + \quad w^{(1)}$, for which the error function is smaller than for the original configuration $w^{(0)}$, i.e. $E(w^{(0)}) \geq E(w^{(1)})$. The entire process is repeated for $w^{(1)}$ and so we get $w^{(2)}$ such that $E(w^{(1)}) \geq E(w^{(2)})$ etc., until we get to the local minimum of the error function. In a multidimensional weighted space, this procedure exceeds our imagination. Although with appropriate choice of the learning rate (α) this method always converges to some local minimum from any initial configuration, there is no guarantee that this happens in real time. Usually this process is very time-consuming (several days of calculation with PC) for small multilayer networks (tens of neurons) as well.

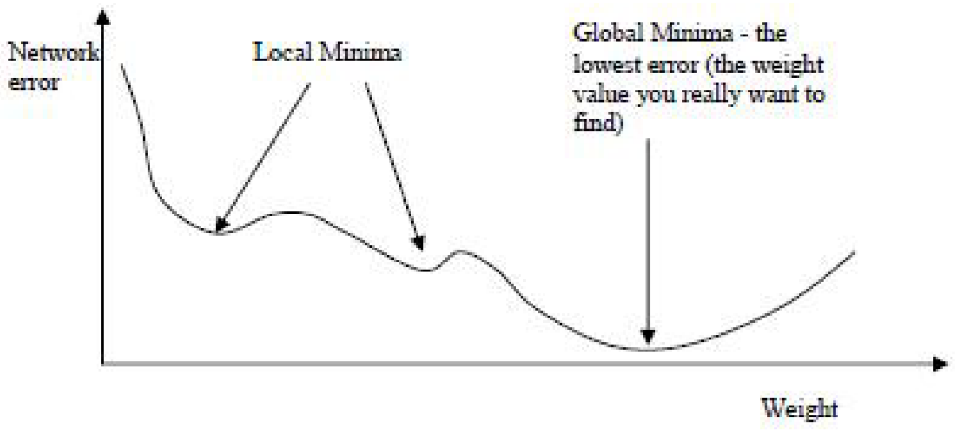

Network error
Local Minima
Global Minima - the lowest error (the weight value you really want to find)
Weight

Fig. 3.2 Gradient method ([4])

The main problem with gradient method is that when it finds a local minimum, then this minimum does not need to be the global minimum (see Figure 3.2). Presented adaptation process stops at this low level (zero gradient) and the network error does not decrease further.

There are a number of solutions to solve this problem. The simplest and most effective (can also solve several other problems) is to reset the weights to different random numbers and try training again. Another solution is to add „momentum“ to the weight change. This means that the weight change this interpretation depends not just on the current error, but also on previous changes. For example $W^+ = W +$ Current change + (change on previous iteration*constant), where constant is < 1 [4].

3.1.1 Description of the backpropagation algorithm

Step 0. The weighting values and the bias are initialized by small random numbers. Assigning the initialization values of the learning coeficient α.

Step 1. Repeat steps (2 to 9) until the condition of calculation termination is not executed.

Step 2. Perform steps (3 to 8) for each (bipolar) training pair s:t.

Feedforward:

Step 3. Activate the input neurons (X_i, $i=1, ...,n$)

$x_i = s_i$

Step 4. Calculate the input values of internal neurons (Z_j, $j=1, ...,p$):

$$z_{in_j} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij} \tag{3.4}$$

Determintation of internal neuron output values

$$z_j = f(z\_in_j) \tag{3.5}$$

Step 5. Determination of the actual output values of neural network signal (Y_k, $k=1, ...,m$):

$$y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j w_{jk} \tag{3.6}$$

$$y_k = f(y\_in_k) \tag{3.7}$$

Backpropagation:

Step 6. Value of the expected output for the input training pattern is assigned to each neuron in the output layer (*Y_k, k=1, ...,m*). Furthermore $\delta_k = (t_k - y_k)f'(y\_in_k)$ is calculated, which is a part of the weight correction $\Delta w_{jk} = \alpha\delta_k z_j$ and bias correction $\Delta w_{0k} = \alpha\delta_k$.

Step 7. A summation of its delta inputs (i.e. from neurons located in the following layer), $\delta\_in_j = \sum_{k=1}^{m}\delta_k w_{jk}$) is assigned to each neuron in the inner layer (*Z_j, j=1, ...,p).* By multiplying the obtained values with derivation of activation function, we get $\delta_j = \delta\_in_i f'(z\_in_j)$, which is a part of the weight correction $\Delta v_{ij} = \alpha\delta_j x_i$ and bias correction $\Delta v_{0j} = \alpha\delta_j$.

Update weights and thresholds:

Step 8. Each neuron in the output layer (*Y_k, k=1, ...,m*) updates on their connections weight values including its bias (*j=0, ...,p*):

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}. \tag{3.8}$$

Each neuron in the inner layer (*Z_j, j=1, ...,p*) updates on their connections weight values including its bias (*i=0, ...,n*):

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}. \tag{3.9}$$

Step 9. Termination condition:
if any changes in weight values do not occur, or if there was performed maximally defined amount of weight changes, stop; otherwise continue.

Although the description of backpropagation learning algorithm is formulated for classic von Neumann computer model, despite it is clear that it can be implement in the distributed way. For each training pattern, the active mode for its input runs firstly so that the information in the neural network spreads from the input to its output. Then based on external information about the required output, i.e. the error of individual inputs, partial derivation of error function are calculated so that the signal spreads back from the output to the input. Network calculation at reverse run proceeds sequentially in layers, while in one layer can proceed paralelly.

3.2 Running the algorithm

Now, after we have reviewed the algorithm in detail, let´s take a look how it works with a large data set. We will trying to teach a network to recognise the first four letters of the alphabet on a 5x7 grid, see below.

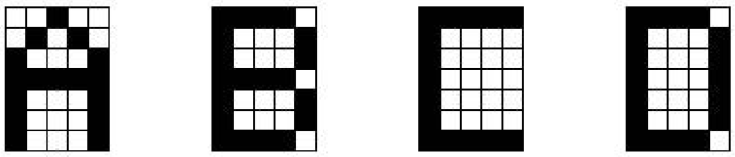


Fig. 3.3 The first four letters of the alphabet ([4])

The first step to train the network is to apply the first letter and change all the weights on the network once. Next do the same for the second letter, then the third, etc. After you have done this for all four letters, return to the first one, and repeat the whole process until the error becomes small (see Figure 3.4).

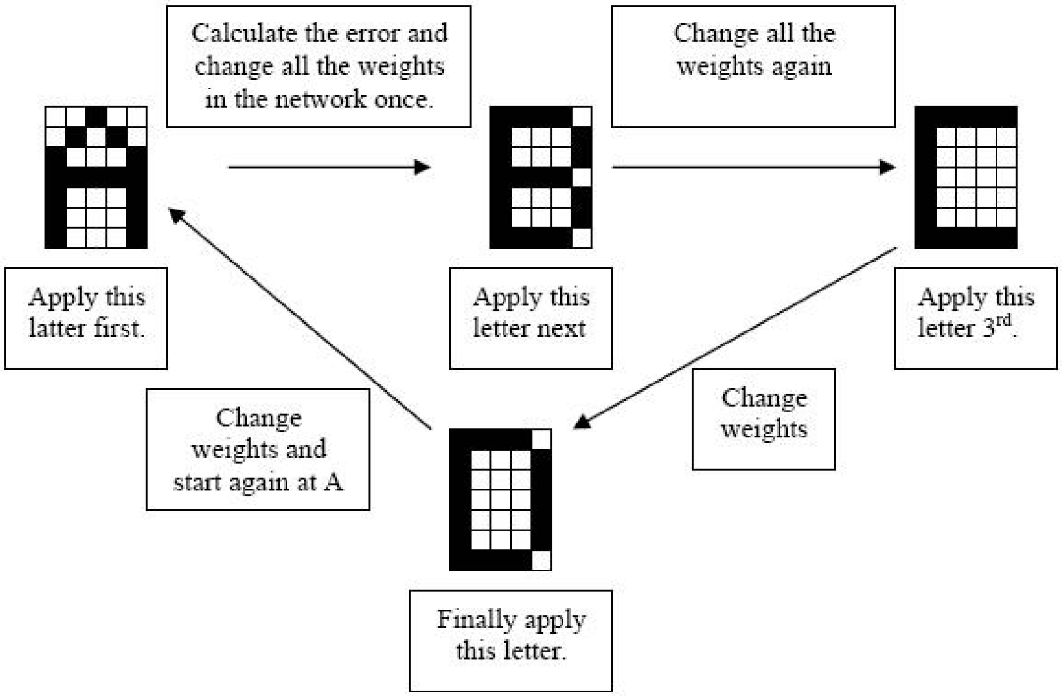
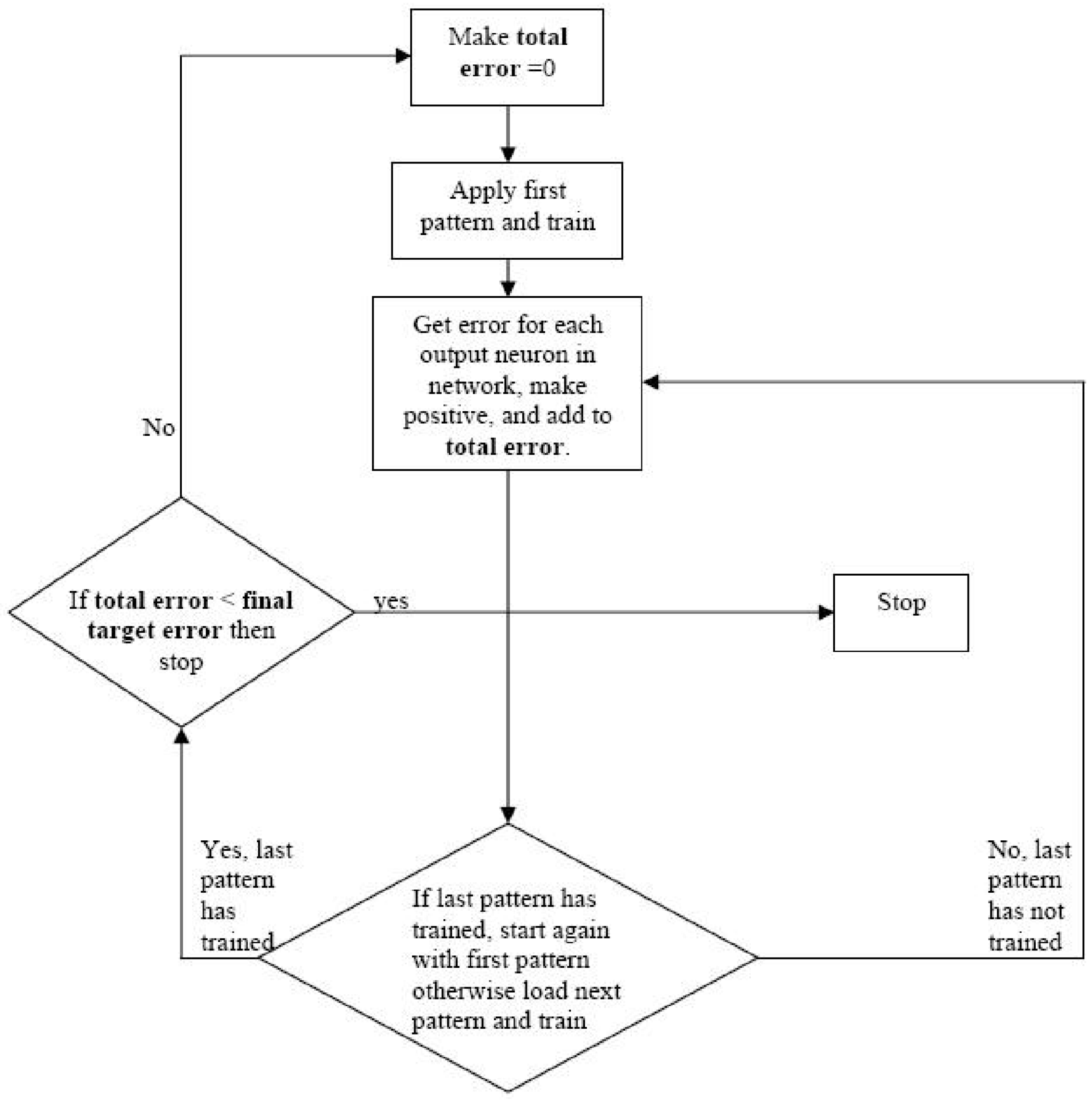

Calculate the error and change all the weights in the network once.
Change all the weights again
Apply this latter first.
Apply this letter next
Apply this letter 3rd.
Change weights
Change weights and start again at A
Finally apply this letter.

Fig. 3.4 The first correctly working algorithm ([4])

Beginners often make a mistake by reducing the errors for each letters individually (apply the first letter to the network, run the algorithm and then repeat it until the error reduces, then apply the second letter, do the same, and so on). In such a way, the network learns to recognize the first letter, then forget it and learn the second letter, etc. and at the end the network would remember only the last letter.

3.3 Stop the training

An important question is: when do we need to stop the training? In practice, it is usual to let the error fall to a lower value, then wait until the network recognizes all the letters successfully. In this case, the network keeps training all the patterns repeatedly until the total error falls to some pre-determined low target value and then it stops [4]. Let´s not forget that we need to make all errors positive. Figure 3.5 shows us the calculation method.



Make **total error** =0
Apply first pattern and train
Get error for each output neuron in network, make positive, and add to **total error**.
No
If **total error** < **final target error** then stop
yes
Stop
Yes, last pattern has trained
If last pattern has trained, start again with first pattern otherwise load next pattern and train
No, last pattern has not trained

Fig. 3.5 Total error for network ([4])

A trained network can recognize not just the perfect patterns, but also the corrupted or noisy ones. Using a validation set is a better way of working out when to stop network training – this helps us to eliminate network overtraining. The idea behind this method is to have a second set of patterns – noisy versions of the training set. Validation set is used to calculate the error, after the network has trained. In case of a fully trained network, the validation set error reaches a minimum, in case of overtraining this error starts rising.

4. FUZZY SYSTEMS

Fuzzy logic was first developed in 1965 by Lotfi Zadeh. It provides an approximate but effective means of describing behavior of systems that are too complex, ill-defined or not easily analyzed mathematically. Its development was motivated by the need for a conceptual framework, which can help in addressing the issue of uncertainty and lexical imprecision. With the help of fuzzy logic we can mathematically express the uncertainties of human cognitive processes like thinking and reasoning. Fuzzy logic uses graded statements rather than ones that are strictly true or false. Some significant characteristics of the fuzzy logic are:

- In fuzzy logic, exact reasoning is viewed as a limiting case of approximate reasoning [6]
- In fuzzy logic, everything is a matter of degree [6]
- In fuzzy logic, knowledge is interpreted a collection of elastic or, equivalently, fuzzy constrain on a collection of variables [6]
- Inference is viewed as a process of propagation of elastic constraints [6]
- Any logical system can be fuzzified [6]

The function of such systems can be described by a set of fuzzy rules, like 'if-then' (premise-consequent). If-then rules use linguistics variables with symbolic terms. Each term represents a fuzzy set. The terms of the input space (typically 5-7 for each linguistic variable) compose the fuzzy partition [1]. The fuzzy interference mechanism consists of three stages:

1. stage – conversion a numerical input value to a fuzzy value – fuzzyfication
2. stage – definition of the rules according to the firing strengths of the inputs
3. stage – retransformation of the resultant fuzzy values into numerical values - defuzzyfication

Main advantages of the fuzzy systems:

- ability to represent uncertainties of the human knowledge with linguistic variables
- easy interpretation of the results
- easy expansion of the base of knowledge by addition of new rules
- robustness in relation of the possible disorders in the system

Main disadvantages are:

- unable to universalize, only answers to what is written in its rule base
- topological changes of the system would demand alternation in the rule base
- definition of the inference logical rules needs expert

4.1 Fuzzy Neural Networks

A marriage between fuzzy logic and neural networks can attenuate the problems of these technologies. Neural net technology can be used to learn system behavior based on system input-output data. This learned knowledge can be used to generate fuzzy logic rules and membership functions, significantly reducing the development time. This provides a more cost effective solution as fuzzy implementation is typically a less expensive alternative than neural nets for embedded control applications. Expressing the weights of the neural net using fuzzy rules helps to provide greater insights into the neural nets, thus leading to a design of better neural nets [5].

Every intelligent technique has some computational qualities (explanation of decisions, learning ability, etc.) making them suited for individual problems. For example, while neural networks are good at recognizing patterns, they are not good at explaining how they reach their decisions [6]. Fuzzy logic systems are good in decision explanations but the rules they use to make those decisions they cannot acquire automatically.

The main reason behind the creation of intelligent hybrid systems have been these limitations. With the combination of two or more techniques, we are able to overcome the limitations of individual techniques. If there is a complex application with two different sub-problems, then a neural network and an expert system can be used separately for solving these individual tasks. A short comparison between the operation of fuzzy systems and neural networks is presented in the following table:

Skills		**Fuzzy Systems**	**Neural Nets**
Knowledge acquisition	Inputs	Human experts	Sample sets
	Tools	Interaction	Algorithms
Uncertainty	Information	Quantitive and Qualitive	Quantitive
	Cognition	Decision making	Perception
Reasoning	Mechanism	Heuristic search	Parallel computat.
	Speed	Low	High
Adaptation	Fault-tolerance	Low	Very high
	Learning	Induction	Adjusting weights
Natural language	Implementation	Explicit	Implicit
	Flexibility	High	Low

Table 4.1 Properties of fuzzy systems and neural networks (based on [6])

Neural network learning techniques can automate the process of design and tune of the membership functions and reduce the development time and cost in a large measure. The behavior of fuzzy systems can be explained with the help of fuzzy rules and their performance can be adjusted by tuning the rules. However, fuzzy system applications are limited to the fields where expert knowledge is available and the number of input variables is small.

To overcome the problem of knowledge acquisition, neural networks are extended to automatically extract fuzzy rules from numerical data [6]. The computational process for fuzzy neural systems starts with the development of fuzzy neuron, based on the understanding of biological neuron and the learning mechanisms. This leads to the following steps:

- development of fuzzy neural models motivated by biological neurons [6]
- models of synaptic connections which incorporates fuzziness into neural network [6]
- development of learning algorithms (that is the method of adjusting the synaptic weights) [6]

Two possible models of fuzzy neural networks are:

- In response to linguistic statements, the fuzzy interface block provides an input vector to a multi-layer neural network. The neural network can be adapted (trained) to yield desired command outputs or decisions [8].

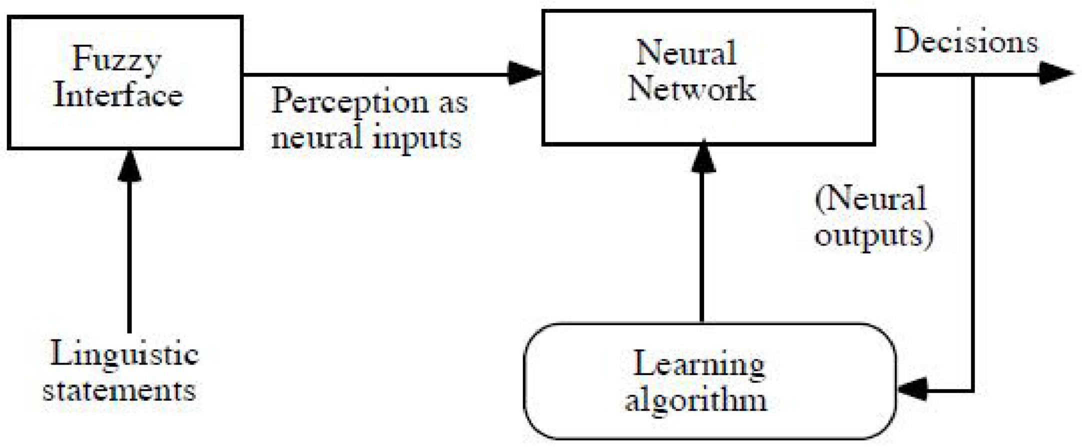


Fig. 4.1 The first model of fuzzy neural network ([8])

- A multi-layered neural network drives the fuzzy inference mechanism [8].

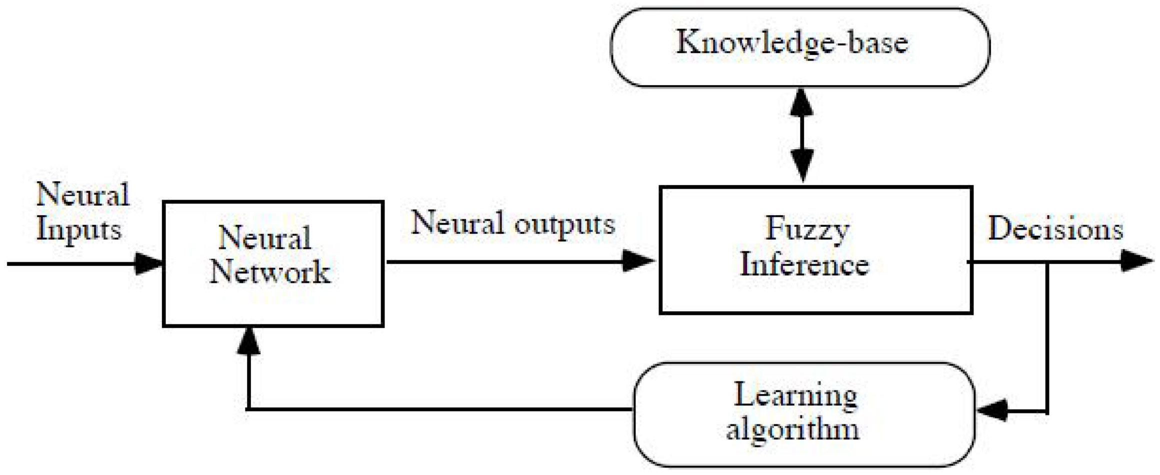


Fig. 4.2 The second model of fuzzy neural network ([8])

A typical fuzzy neural network is Barenji´s ARIC (Approximate Reasoning Based Intelligent Control) architecture. It is a neural network model of a fuzzy controller and learns by updating its prediction of the physical system´s behavior and fine tunes a predefined control knowledge base [8].

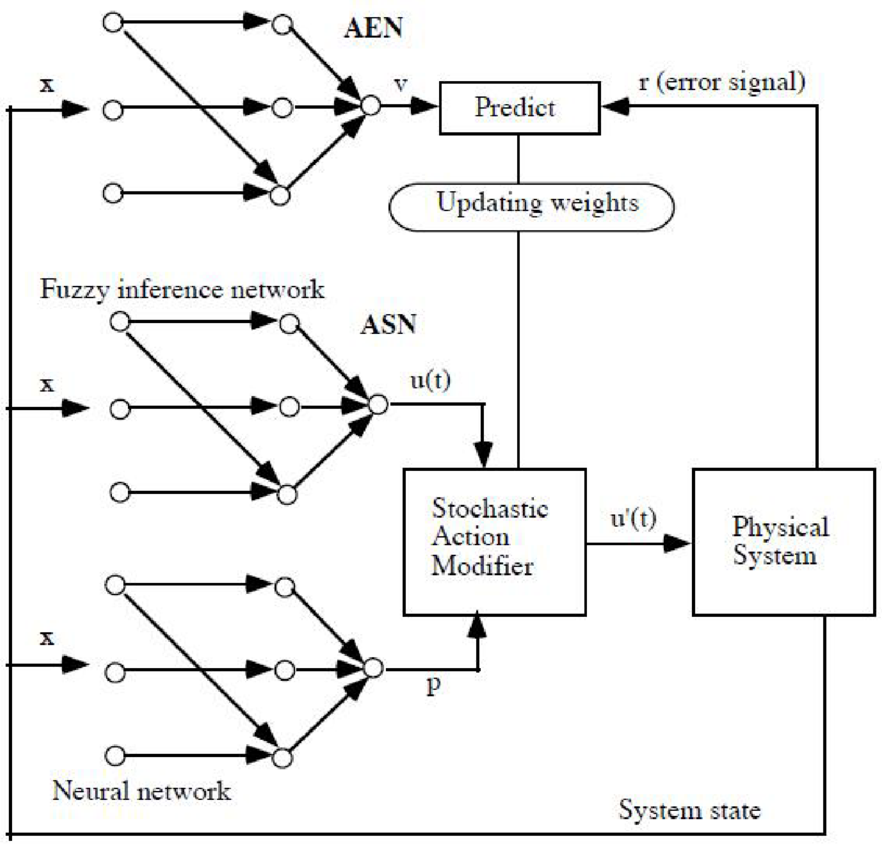


Fig. 4.3 Berenji´s ARIC architecture ([8])

With this architecture we have the opportunity to combine the advantages of both neural networks and fuzzy controllers. By predefining the fuzzy IF-THEN rules the system learns faster than a standard neural control system, because it has not to learn from scratch. ARIC is made up of feedforward neural networks, the Action-State Evaluation Network (AEN) and the Action Selection Network (ASN).

ASN is a multilayer neural network representation of a fuzzy controller. In fact, it consists of two separated nets, where the first one is the fuzzy inference part and the second one is a neural network that calculates p[t, t + 1], a measure of confidence associated with the fuzzy inference value u(t + 1), using the weights of time t and the system state of time t + 1. A stochastic modifier combines the recommended control value u(t) of the fuzzy inference part and the so called „probability“ value p and determines the final output value of the ASN [8]:

$$u'(t) = o(u(t), p[t, t+1]) \tag{4.1}$$

The hidden unit z_i of the fuzzy inference network represent the fuzzy rules, the input units x_j the rule antecedents, and the output unit u represents the control action, that is the defuzzified combination of the conclusions of all rules (output of hidden units). In the input layer, the system state variables are fuzzified [8]. ARIC uses monotonic membership functions only. The fuzzy labels of control rules are set for each rule locally. The membership values are then multiplied by weights attached to the connection of the input unit to the hidden unit. The minimum of those values is its final input [8].

A special monotonic membership function which represents the conclusion of the rule is stored in each hidden unit. The crisp output value belonging to the minimum membership value can be easily calculated by the inverse function (thanks to the monotonicity of this function). This value is multiplied with the connection weight between the hidden unit and the output unit. The output value is then calculated as a weighted avarage of all rule conclusions [8].

The AEN tries to forecast the behavior of the system. It is a feedforward neural network with one hidden layer, which receives the system state as its input and an error signal r from the physical system as additional information [8]. The network output v[t, t´] is viewed as a prediction of future reinforcement that depends of the weights of time t and the system state of time t´ (which can be t or t+1). Better state have characteristically higher reinforcements.

The weight changes are determined by a reinforcement procedure that uses the output of the ASN and the AEN. The ARIC architecture was applied to cart-pole balancing and it was shown that the system is able to solve this task [8].

5. ADAPTIVE NEURO FUZZY INFERENCE SYSTEM

Adaptive Neuro Fuzzy Inference System (ANFIS) as developed by Jang et al. (1997) is a class of adaptive networks that are functionally equivalent to fuzzy inference systems (FIS), where the parameters of fuzzy inference systems are updated by neural networks from a set of training data. An adaptive network, as its name implies, is a network structure consisting of nodes and directional links through which the nodes are connected. Moreover, part of all of the nodes are adaptive, which means their outputs depend on the parameters pertaining to these nodes, and the learning rule specifies how these parameters should be changed to minimize a prescribed error measure. ANFIS enjoys many of the advantages claimed by neural networks (NNs) and the linguistic interpretability of fuzzy inference systems, wherein both NNs and FIS play active roles in an effort to reach specific goals [10], [11].

Thanks to its capability and because it can perform the same function, almost any neural network can be replaced by ANFIS. Its primary advantages are non-linearity and structural knowledge representation.

ANFIS consists of a self-tuning Sugeno-type inference system and calculates its outputs as a weighted linear combination of the consequents. The hybrid learning algorithm includes two stages, which are:

- forward pass – identifies the consequent parameters with the help of FIS learning mechanism and least-squares estimator (LSE)
- backward pass – propagates backward the error rates (error backpropagation) and updates the premise parameters by the gradient descent method

In ANFIS, the membership functions (gaussian functions) are expected to map all inputs by changing their parameters. It is desired that all inputs can be mapped to produce the desired outputs. Unfortunately, in the case that there occur variations in the inputs, the desired outputs will be poorly approximated by the actual outputs because of limitations in finding the parameters of the fixed finite number of fuzzy membership functions [10].

The fuzzy membership function is the basic block of fuzzy logic systems and has many possible interpretations [10]. It can define the richness of the extracted information from the given data in case of highly nonlinear systems and the form of the membership functions can be extended to cover this richness.

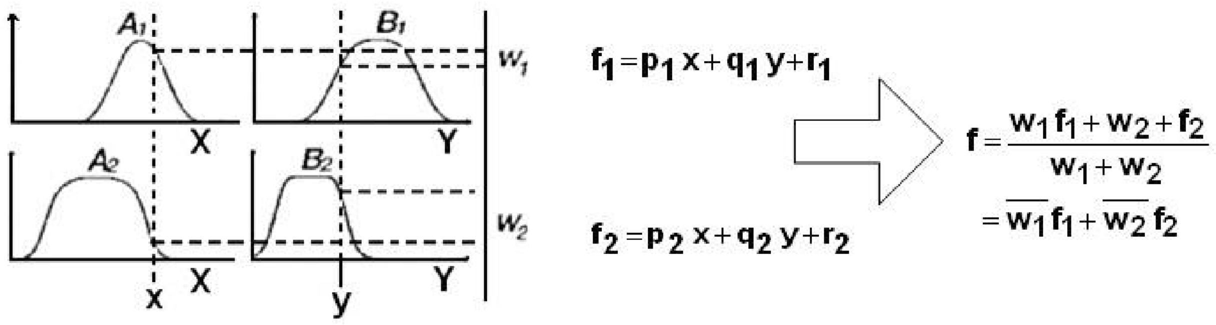
5.1 Learning algorithm of ANFIS

The standard ANFIS uses the Sugeno-type fuzzy model to generate fuzzy rules from a given input-output data set. For easy understanding, let's take a simple version of fuzzy inference system with two inputs *x*, *y* and one output *f*. A rule set for a typical first-order Sugeno fuzzy with two fuzzy if-then rules has the following form (based on [10]):

Rule 1: If *x* is A_1 and *y* is B_1, then $f_1 = p_1 + q_1 y + r_1$ (5.1)

Rule 2: If *x* is A_2 and *y* is B_2, then $f_2 = p_2 + q_2 y + r_2$ (5.2)

Figure 5.1 shows the reasoning mechanism for the Sugeno model. The corresponding standard ANFIS architecture where nodes in the same layer have similar functions is shown in Figure 5.2. The important part of the presented ANFIS is the modification of the error correction rules of error backpropagation (EBP) by using a mapping function to replace the membership function in the standard ANFIS [10].



A_1
B_1
A_2
B_2
X
Y
x
y
w_1
w_2
$f_1 = p_1 x + q_1 y + r_1$
$f_2 = p_2 x + q_2 y + r_2$
$f = \frac{w_1 f_1 + w_2 + f_2}{w_1 + w_2}$
$= \overline{w_1} f_1 + \overline{w_2} f_2$

Fig. 5.1 A two-input first-order Sugeno fuzzy model ([10])

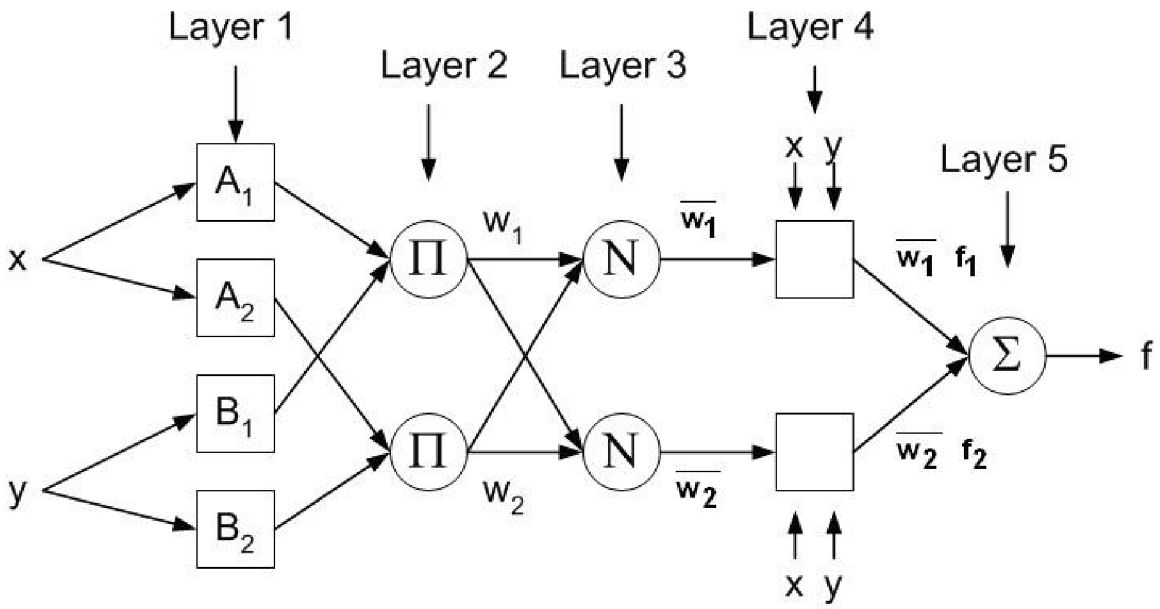


Fig. 5.2 A standard ANFIS architecture ([10])

5.1.1 Forward pass

The forward pass is based on the architecture presented in Figure 5.2. It uses two inputs and one output. For convenience, a different notation is introduces as shown in Figure 5.3 [10].

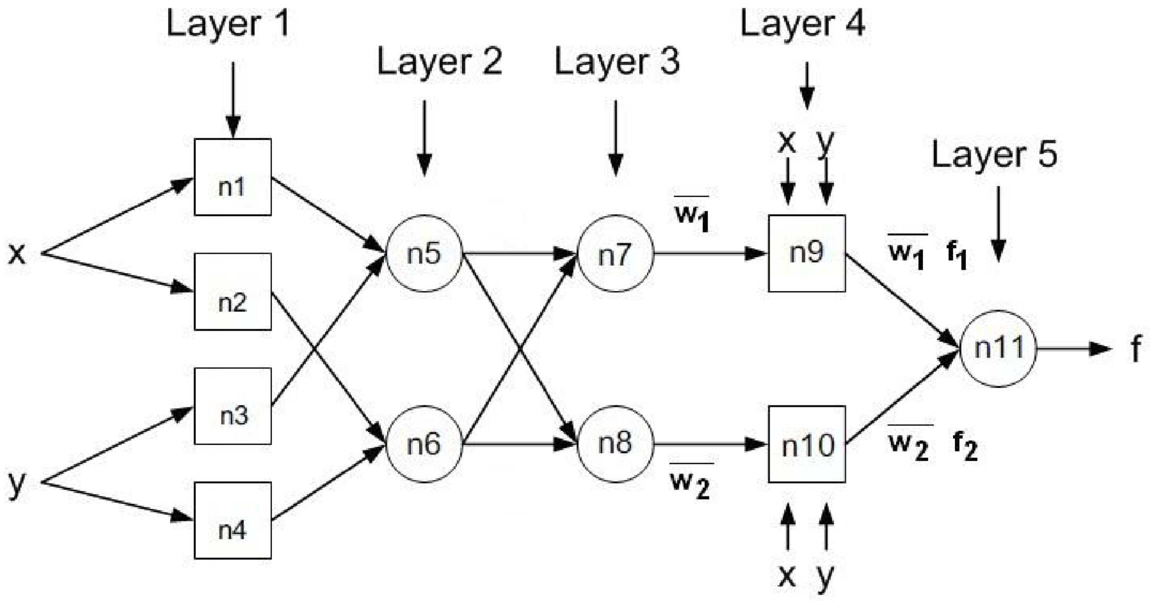


Fig. 5.3 The forward pass (based on [10])

The functions of the individual layers are the following:

Layer 1:
This layer is the so-called *fuzzification layer*. The bell activation function is used as the membership function, which has a regular bell shape and is specified as

$$\mu A(x) = \frac{1}{1 + \left| \frac{x - c_i}{a_i} \right|^{2b_i}} \qquad (5.3)$$

The membership function has parameters {a_i, b_i, c_i}, i = 1, 2, 3, 4 which are predetermined by selecting parameter values. Each output of this node is labeled by *a*. Accordingly, the outputs are denoted by *n1a*, *n2a*, *n3a*, and *n4a*. The symbol *a* is used in order to differentiate with new symbol *b* (after the correction) that will be used later in the backward pass [10].

Layer 2:
This layer is the *rule layer*, where fuzzy logic AND is used in the node function. The output of this layer can be obtained as

$$n5a = \min(n1a, n3a)$$
$$n6a = \min(n2a, n4a) \tag{5.4}$$

Layer 3:
This layer is the *normalization layer*. Let *ntot_a = n5a + n6a*, then the normalization is given by [10]

$$n7a = n5a / ntot\_a$$
$$n8a = n6a / ntot\_a \tag{5.5}$$

Layer 4:
This layer is the *deffuzification layer.* By arranging the incoming signals, matrix *A* can be obtained which has the form

$$A = [(n7a\ x)\ (n7a\ y)\ n7a\ (n8a\ x)\ (n8a\ y)\ n8a] \tag{5.6}$$

By means of the LSE method, we obtain the consequent parameter *P* = [p_1, q_1, r_1, p_2, q_2, r_2] by using the following equation

$$P = [A^T A]^{-1} A^T U \tag{5.7}$$

where *U* is the desired output of the controller. The consequent parameter *P* is then used to compute f_1 and f_2 by using the following equation

$$f_1 = p_1 x + q_1 y + r_1$$
$$f_2 = p_2 x + q_2 y + r_2 \tag{5.8}$$

After that, the output of the node *n9* and *n10* are calculated by the equation [3]

$$n9a = n7a\ f_1$$
$$n10a = n8a\ f_2 \tag{5.9}$$

Layer 5:
This layer is represented by a single *summation neuron*. This layer produces the overall ANFIS output with a simple summation of the layer input signals given by

$$n11a = n9a + n10a \tag{5.10}$$

5.1.2 Backward pass

After running the forward pass, we get the resulted error. Within the backward pass, this error is propagated back to the system by using error correction rule of the modified error back propagation (EBP), see Figure 5.4.

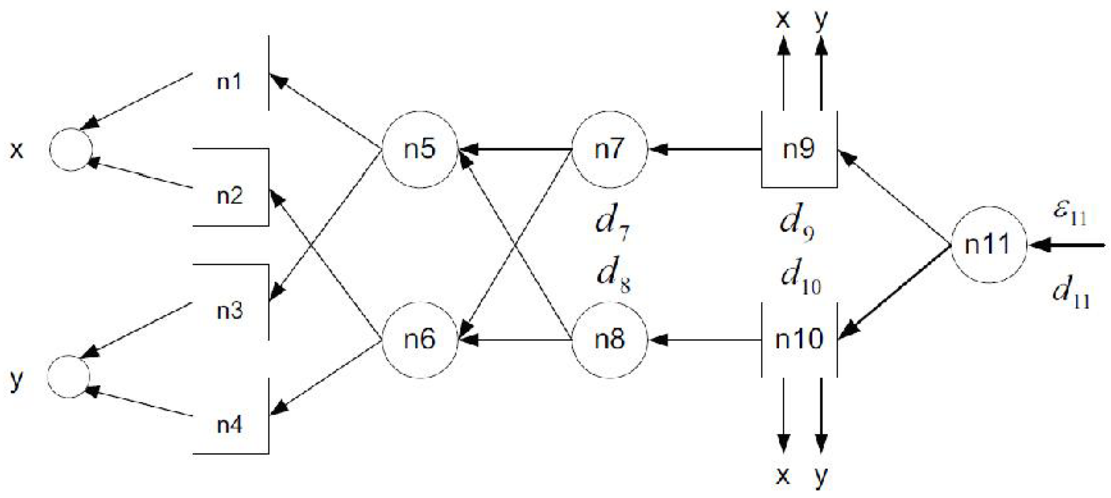


Fig. 5.4 The backward pass ([10])

Symbol *ε₁₁* defines the error between the desired output d_k and the actual output. The sum of the squared error is given by [10]

$$E_p = \sum_{k=1}^{N(l)} (d_k^p - x_{l,k}^p)^2 \qquad (5.11)$$

In our case the sum of the squared error defines the difference between the desired and the actual output, $E_p = \varepsilon_{11}$. The value x_l in this layer is given by *n11* and $d_k =$ *U*, then the error is defined as [10]

$$\varepsilon_{11} = -2(U - n11a) \qquad (5.12)$$

Next, d_{11} is defined as follows [10]

$$d_{11} = -\varepsilon_{11}/2 = U - n11a \qquad (5.13)$$

The output of the node *n11* then becomes [10]

$$n11b = n11a + d_{11} \qquad (5.14)$$

According to formula 5.10, we have

$$n11b = n9b + n10b$$

Based on formula 5.14, we can define

$$n9b = n9a + d_9$$
$$n10b = n10a + d_{10}$$

then we can appoint

$$d_{11} = d_9 + d_{10} \tag{5.15}$$

Multiplying the left side of formula 5.15 by (f_1 + f_2)/(f_1 + f_2) leads to [10]

$$\frac{d_{11} f_1}{f_1 + f_2} + \frac{d_{11} f_2}{f_1 + f_2} = d_9 + d_{10} \tag{5.16}$$

Since *n9a = n7a* f_1 and *n10a = n8a* f_2 , after correction we have *n9b = n7b* f_1 and *n10b = n8b* f_2 . As a result, we obtain [10]

$$n9a + d_9 = (n7a + d_7) f_1$$
$$n10a + d_{10} = (n8a + d_8) f_2$$

Next, from the *ntot_a* of the forward pass, we write the new *ntot_b* as follows [10]

$$ntot\_b = ntot\_a + d\_tot \tag{5.17}$$

where *d_tot* is arbitrary and obtained from the experiment data. Suppose *d_tot = 0*, this implies *ntot_b = ntot_a*. Then the output nods in Layer 2 has the form [10]

$$n5b1 = (n7a + d_7) ntot\_b$$
$$n6b1 = (n8a + d_8) ntot\_b \tag{5.18}$$

In this layer, the minimum value of input signals are selected - the logic AND function is applied to process the outputs of Layer 1. As in Layer 2, we already have *n5a1* and *n6b1*, it is important that the outputs of this node must satisfy *n5b = n5b1* and *n6b = n6b1.* A simple way is to split n5b1 and n6b1 into two parts. We then add an arbitrary value to the one part, so that it has higher value than the other part. As a result, this part will not be chosen in Layer 2 [10]. After adding the arbitrary value which belongs to the output node in Layer 1, as a result we get the original value of *n1b*, *n2b*, *n3b* and *n4b*. The next step is mapping all the inputs to the corrected output of Layer 1. The mapping function then becomes the membership function of the learning mechanism of the modified ANFIS.

6. DATA ACQUISITION

After overviewing the basic theory of neural networks and the theory of backpropagation algorithm and fuzzy systems, the next step is to acquire the necessary data for further processing – the speech signal.

Speech/voice recognition is a difficult task to be performed by a computer system [12]. Although a wide range of commercial products were launched in the last decade, an absolute solution has not been found out yet, and many research areas have still remained opened in the field.

Speech is a sequence of waves which are transmitted through a medium and are characterized by some features, including characteristic frequencies and corresponding intensities [13]. The vibrations of sound waves are perceived by eardrums in the inner ear, and these oscillations are forwarded to a specific part of brain for further processing.

The three deciding factors when talking about human-like perception of speech are *loudness*, *pitch* and *quality*. Loudness represents the energy (intensity) of the sound. The greater the amplitude is, the louder the sound appears. Pitch is responsible for the tone of the sound. Higher pitch issues higher tone and against, lower pitches lower tone. The quality of sound is a perceptual correlate of its spectral content related to the fundamental frequency of the vocal vibration of the speaker organ [13].

The most important factor of the recording is the clarity of the recorded signal. It should be as clear and noise free as possible. For this reason, the recording took place in a quite environment using a Tascam DR-40 portable digital recorder for the best possible sound quality. The recorded continuous signal was then split into separate words with WavePad Sound Editor software.

Five words in three languages (English, Czech and Hungarian) were created for further neural network training purposes, i.e. a total of 60 words (every single word 4 times) and another 15 words for testing the system. Each signal has been converted into mono and was saved uncompressed with a bit depth of 16 bits and a sample rate of 44.1 kHz. The recorded words are listed in the table below.

	Language		
	English	**Czech**	**Hungarian**
Words	Andrew	Ondřej	András
	apple	jablko	alma
	grape	hrozno	szőlő
	orange	pomeranč	narancs
	strawberry	jahoda	eper

Table 6.1 List of recorded words

The next pictures show the differences between individual words with the same meaning in different languages. Axis X represents the time in seconds, while axis Y the amplitude of the signal.

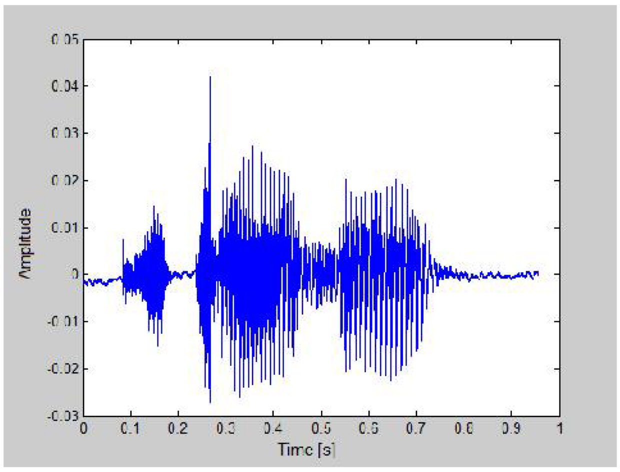


Fig. 6.1 Sound wave of the word ‘strawberry’

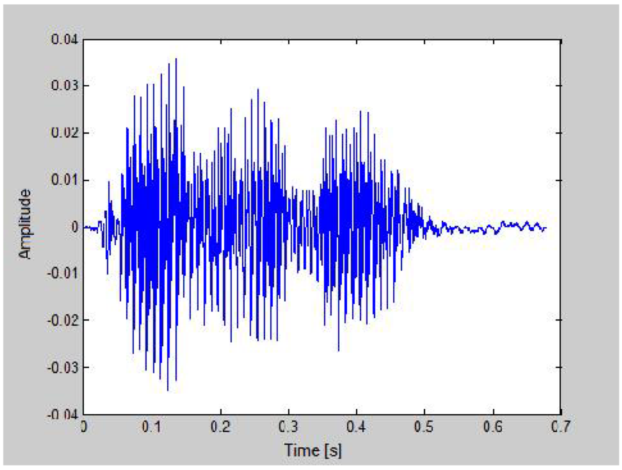


Fig. 6.2 Sound wave of the word ‘jahoda’

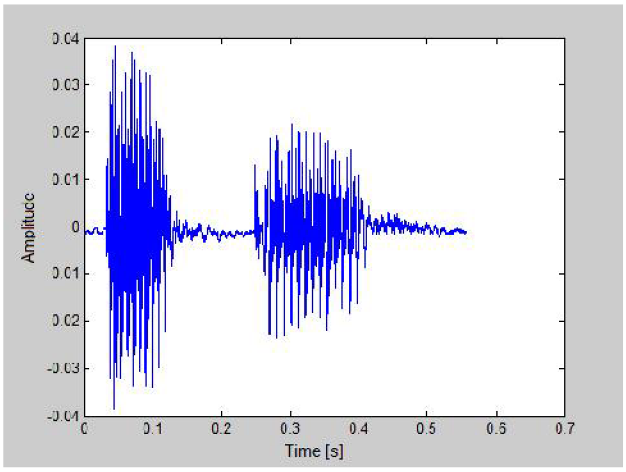


Fig. 6.3 Sound wave of the word 'eper'

Each spelled character has its very own composition, which can be described with pairs of parameters:

- frequencies – the rate at which the sound wave passes a given point
- amplitudes – represents the amount of energy of a given frequency in a sound wave
- wavelength – distance between the crest of one wave to another

By taking a closer look at the signals, we can easily count the number of syllables and one of the most important phonetic characteristic – where the word is emphasized (accented). While the words 'strawberry' and 'jahoda' consists of three syllables (straw-be-rry, ja-ho-da), the word 'eper' consists of only two syllables (e-per). Words 'jahoda' and 'eper' reaches its maximum amplitude right at the near beginning, while the word 'strawberry' somewhere around the letter 'a'. These are the places where the given words are emphasized.

7. THE REALIZATION OF THE PROGRAM

This chapter serves to demonstrate the program built in Matlab. The program itself can be separated into 3 parts: ANFIS using the Matlab’s Fuzzy Logic Toolbox (’anfis’), neural network using Matlab’s Neural Network Toolbox (’nn’) and the individually built neural network based on the backpropagation algorithm presented in Chapter 3 (’nnv’).

The main program is the script file `spust.m`. Its listing is included in Appendix. The purpose of the first part is to read the parameters of the test voice recordings. This program serves for probing ANFIS as well as neural networks. For the selection of the operation mode, the variable `mode` has to be set to ’anfis’, ’nn’ or ’nnv’ by commenting and uncommenting the individual lines.

```
mode='anfis';
%mode='nn';
%mode='nnv';
```

The data reading is implemented in the `wavload` function, which receives as input parameters the path to the directory containing training files and the number of output parameters. At the beginning of this script, the parallel processing toolbox is initialized by the command `matlabpool` open. The usage of this toolbox greatly increases the processing speed in case of the processor is multi-cored or there are more computers available.The next part of the code brings into effect the actual learning of the network.

In the case of the mode is set to ’anfis’, the parfor cycle is used for the creation and learning of three ANFIS networks, each for one output variable. Parfor is part of the parallel processing toolbox. Its iterations are run in parallel increasing the computing speed. Firstly, the given network has to be created. For the purpose of this work, the practical usage of ANFIS is heavily limited by its high demands on processing power for the case of higher number of inputs and second level neurons. The basic task of network creation takes into account all combinations of inputs and membership functions. In this case it means a very high number of created membership functions and second level neurons. Therefore, a special function was used for the creation of these functions and network nodes which analyses the input data and searches for existing clusters in it. These clusters are used for simplification of the input side of the network. This approach significantly increases the maximal number of usable inputs of the system.

The function `genfis2` creates a Sugeno-type FIS structure. For the creation of input rules, the subtractive cluster analysis method is used. This method tries to make use of existing patterns to simplify the input part of the network. The subtractive clustering initially assumes all data points as clusters. Subsequently, some clusters are merged together based on preset distance criterion, then the new cluster centers are calculated.

The learning itself is realized by the function `anfis` that executes the learning algorithm individually for each network. The number of ANFIS networks equals to the number of output variables (columns in matrix `tgt`). It utilizes a hybrid learning technique, what is a combination of the least-squares estimator (LSE) method and the error backpropagation (EBP) algorithm. Afterwards, the network is tested for correctness with the same data as used for training using the function `evalfis`. The result of each network is saved to the corresponding column in matrix `res`. For the case of usage of neural network, the function `feedforwardnet` is used, which creates a neural network suitable for classification tasks. The number of neurons in each layer is also set here. The function `train` trains the network for the given training data.

In the case, the mode is set to 'nnv', the neural network functions created within the frame of Semestral Project MM2E (netinit, netlearn and neteval) are in use. These functions can create a simple neural network structure, and are able to train and evaluate it.

Loading of audio files – `wavload.m`

This function is used for audio file loading and parameter calculation (see Appendix). Firstly, the file names are determined in the given directory that has the wav extension. After that, all files are processed sequentially, as is described herein. At the beginning, the given file is read into a vector and is normalized to have maximal amplitude of 1. Subsequently, the parameters are calculated using the `fftparams` function. The file names are prepared to contain information about the language of the recording. The first letter of it corresponds to the first letter of the used languages (i.e. 'c' means Czech, 'e' means English and the prefix 'h' is for Hungarian). This information is used for creating the target matrix (`tgt`) that is used for training the network. The target matrix and the matrix of FFT parameters are returned as return variables of the function.

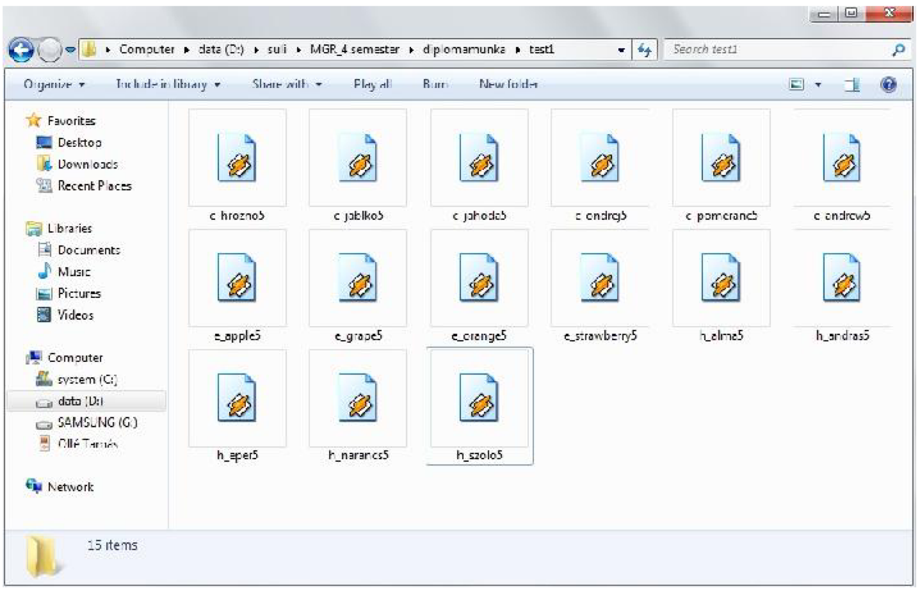


Fig. 7.1 The test words

Analysis parameters – `fftparams.m`

The signals in their raw form are not suitable as inputs to a network because these contain extremely large amount of information. However, parameters can be used instead of the original signals that describe the signal shape at an appropriate level. The signal is divided into a constant number of sections. In our case the lengths of these sections are set to approximately 50 ms (depending on the length of the actual signal, each one is split into 14 pieces). For each section, the spectral composition is calculated using the fast Fourier transformation (FFT). In each of these spectra, the five highest spectral components are determined. The frequencies and amplitudes of these components are used as the analyzed signal parameters. Before use, signals are filtered by a bandpass filter of boundaries 100 and 2000 Hz. The Fourier transformation is also smoothed to limit the influence of noise. This function is also listed in Appendix. Figure 7.2 displays the process. The top of the figure shows the portion of the signal wave, the second one is the signal after the Fourier transformation process and the last one illustrates the peak points of the transformed signal that were used as input parameters.

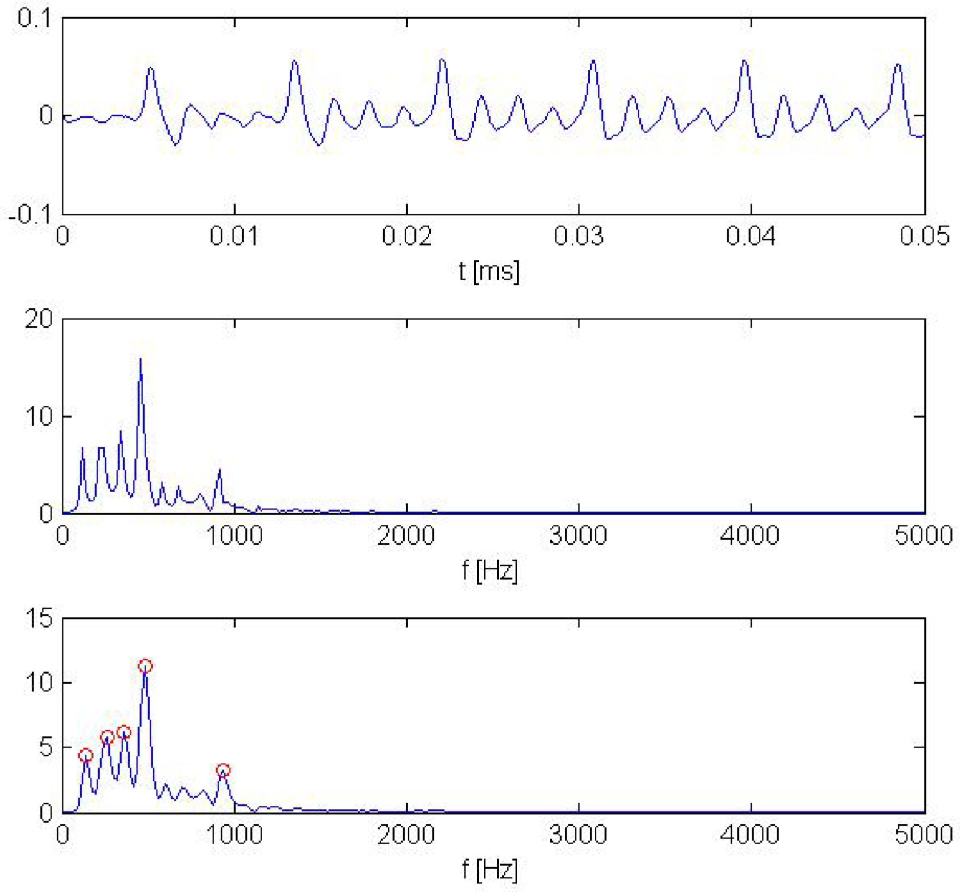


Fig. 7.2 The fast Fourier transformation (FFT)

Neural network creation – `netinit.m`

This function creates a simple structure that contains the necessary information and weights of each neuron input. The weights are initialized with small random numbers. This structure variable is returned by the function.

Neural network training – `netlearn.m`

This function implements the classical backpropagation algorithm for training the neural network. The network coefficients are updated on each run as many times as the number of input-target pairs. The number of runs (training epochs) has to be set manually. The function returns the trained network.

Neural network simulation – `neteval.m`

This function calculates the output of each neuron gradually in each layer and, finally, the output of the whole network for the given input sets. The result is returned as a matrix, where the corresponding outputs are organized in rows. Each row corresponds to one input set.

The following figures (Fig. 7.3 and Fig. 7.4) show the workflow of the program where the first four blocks represents the training, while the last three parts the testing/evaluation part.

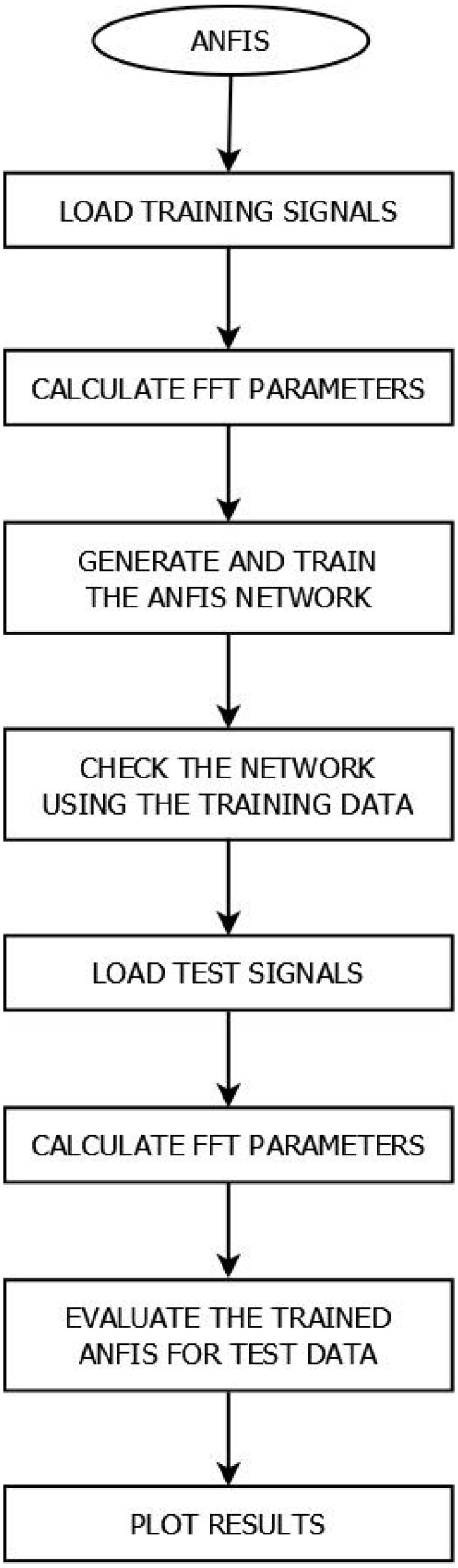

ANFIS
LOAD TRAINING SIGNALS
CALCULATE FFT PARAMETERS
GENERATE AND TRAIN THE ANFIS NETWORK
CHECK THE NETWORK USING THE TRAINING DATA
LOAD TEST SIGNALS
CALCULATE FFT PARAMETERS
EVALUATE THE TRAINED ANFIS FOR TEST DATA
PLOT RESULTS

Fig. 7.3 Flow chart of ANFIS

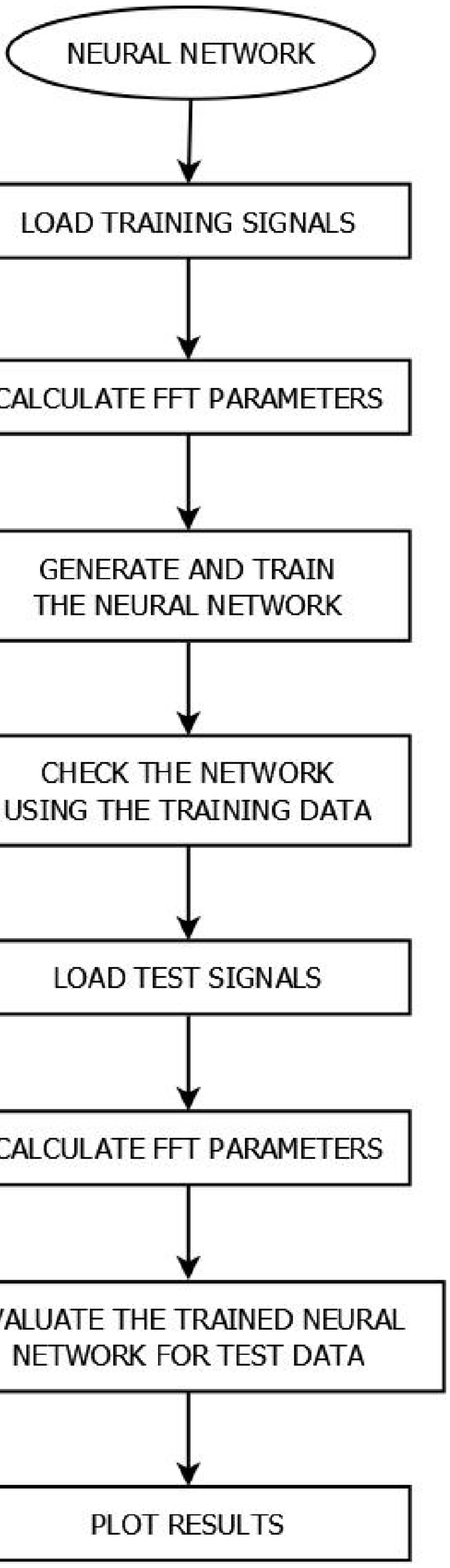

NEURAL NETWORK
LOAD TRAINING SIGNALS
CALCULATE FFT PARAMETERS
GENERATE AND TRAIN THE NEURAL NETWORK
CHECK THE NETWORK USING THE TRAINING DATA
LOAD TEST SIGNALS
CALCULATE FFT PARAMETERS
EVALUATE THE TRAINED NEURAL NETWORK FOR TEST DATA
PLOT RESULTS

Fig. 7.4 Flow chart of the Neural Network

8. RUNNING THE SIMULATION

After the program was made and its adequate functioning was tested, the next step is experimenting with it and fine tuning the simulation parameters for optimal results. A number of input values are created for every network the following way: each input signal is cut up into 14 pieces and 10 parameters are calculated from each segment for a total number of 140 input values per input file.

8.1 The ANFIS network

The ANFIS is a very complex structure; its implementation is extremely time-consuming. The ANFIS network created by the Fuzzy Logic toolbox has clearly the same advantages over an own implementation and have the Neural Network toolbox over the implemented simple network. These include flexibility and wide range of possibilities of configuration.

The membership function for the ANFIS network is calculated by the `genfis2` function. This function generates the structure of the Fuzzy Inference System from data using subtractive clustering.

The subtractive clustering is a one-pass algorithm for estimating the number of clusters and the cluster centers through the training data. This method partitions the training data into groups called clusters and generates the cluster centers until the maximum potential value in the current iteration is equal to or less than the threshold δ. By the end of the clustering process, a set of fuzzy rules are obtained [2].

8.1.1 Network parameters

For the training and the testing process the following parameters were set within the ANFIS network:

Number of layers:	5
Output function of the neuron:	see Chapter 7
Training function:	combination of the least-squares method and the backpropagation gradient descent
Number of epochs:	3
Threshold:	the biggest output of the three networks indicates the recognized language
Number of outputs:	1 for each network (for a total amount of 3)

The training is done by the function `anfis` while the testing is done by `evalfis`.

8.1.2 Simulation results

A total number of 5 simulations were run to determine approximately the average error. The result of the program is very stable, which means that just slight differences can be seen in the simulation results (if any). Average time of simulation is around 450 seconds. The memory demand is very high. The simulation needs approximately 6 GB of RAM to run „smoothly“. The simulation result of the ANFIS network is shown in Figure 8.1.

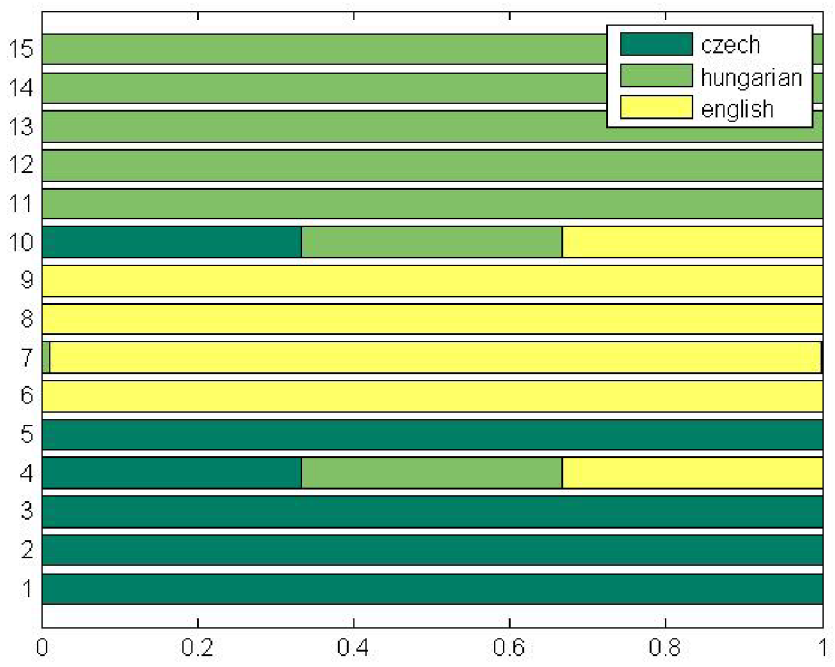


Fig. 8.1 Example of the simulation results in ANFIS

The next table shows the actual meaning assigned to numbers. This table is valid for every single figure with simulation results hereafter.

	LIST OF THE TEST WORDS	
	No.	**Word**
Hungarian	15	szőlő
	14	narancs
	13	eper
	12	András
	11	alma
English	10	strawberry
	9	orange
	8	grape
	7	apple
	6	Andrew
Czech	5	pomeranč
	4	Ondřej
	3	jahoda
	2	jablko
	1	hrozno

Table 8.1 List of the test words

The average simulation results for ANFIS network are listed in Table 8.2. Each row contains three outputs of the neural network. These values can be considered as an approximate “probability”. Since their sum can be different than one, they cannot represent real probabilities.

	Czech	**Hungarian**	**English**
hrozno	0.9835	0.0000	0.0000
jablko	1.1189	0.0000	0.0000
jahoda	1.0003	0.0000	0.0000
Ondřej	0.5000	0.5000	0.5000
pomeranč	0.8924	0.0000	0.0000
Andrew	0.0000	0.0000	0.8268
apple	0.0000	0.0093	0.9459
grape	0.0000	0.0000	1.0149
orange	0.0000	0.0000	1.0405
strawberry	0.5000	0.5000	0.5000
alma	0.0000	0.9523	0.0000
András	0.0000	0.8743	0.0000
eper	0.0000	1.2376	0.0000
narancs	0.0000	1.0792	0.0000
szőlő	0.0000	1.1724	0.0000

Table 8.2 The results of the simulation of ANFIS in numbers

As the table shows, the network is unable to decide the language of two words. These words are “Ondřej” and the word “strawberry” (both 0.5000). The most precisely allocated word is “eper” (1.2376) while the least precisely allocated word is “Andrew” (0.8268).

8.2 The ’NN’ network

This section was created using the Matlab’s Neural Network Toolbox by the following commands:

```
nnet = feedforwardnet([10 10 8 8]);
nnet = configure(nnet, inp', tgt');
```

The input parameter of the `feedforwardnet` function determines the number of neurons in each hidden layer and implicitly the number of hidden layers. The `configure` function sets the network input and output sizes and ranges and initializes the weights.

8.2.1 Network parameters

For the training and the testing process the following parameters were set within the NN network:

Number of layers:	5
Output function of the neuron:	hyperbolic tangent sigmoid transfer function
Training function:	Levenberg-Marquardt
Number of epochs:	50, however, the training might be finished by additional criteria
Threshold:	the biggest output of the network corresponds to the recognized language
Number of outputs:	3

As soon as one of the progress bars (each representing a criterion) reaches its maximum, the training stops.

8.2.2 Simulation results

A total number of 5 simulations were run to determine the average error. The result of this method is very unstable, which means that the simulation results vary unacceptably by each simulation. Average time of simulation is approximately 10 seconds. A better result of the simulations is illustrated in Figure 8.2.

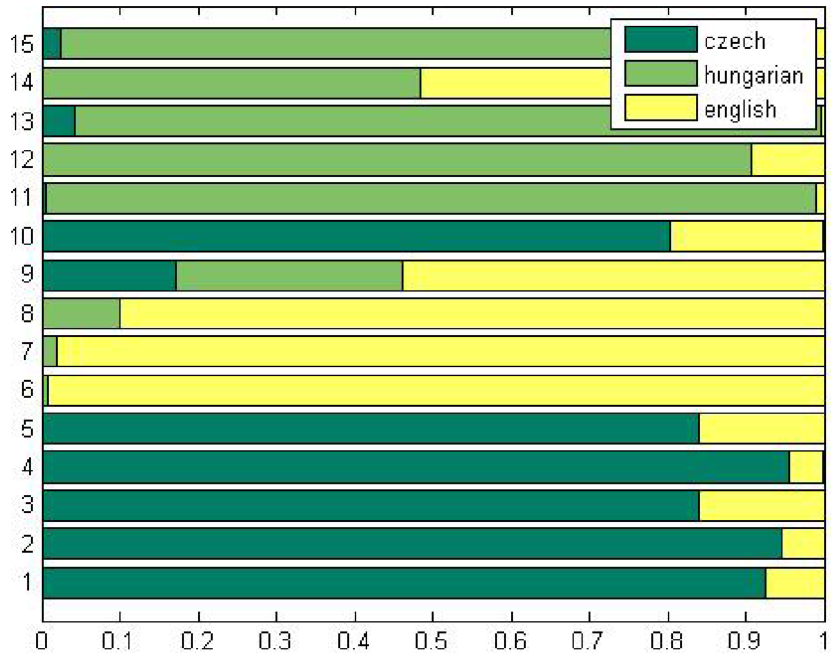


Fig. 8.2 Example of the simulation results in ’NN’

The average of the 5 simulation results for ’NN’ network is listed in Table 8.3.

	Czech	Hungarian	English
hrozno	0,8197	0,1170	0,0779
jablko	1,0296	0,0012	0,0136
jahoda	0,9488	0,0305	0,0456
Ondřej	0,9730	0,0150	0,0541
pomeranč	0,6045	0,3666	0,0588
Andrew	0,0088	0,1829	0,8350
apple	0,0935	0,1437	0,7977
grape	0,1073	0,1172	0,7927
orange	0,0508	0,0757	0,9078
strawberry	0,2427	0,3560	0,4233
alma	0,0105	0,9734	0,0116
András	0,2058	0,7902	0,0202
eper	0,0618	0,8912	0,1093
narancs	0,0771	0,7836	0,1383
szőlő	0,1062	0,7893	0,1357

Table 8.3 The results of the simulation of ’NN’ in numbers

As the table shows, the network is able to recognize the language of all words, however, it cannot allocate the words as precisely as the above presented ANFIS network. The most precisely allocated word is “jablko” (1.0296) which has almost zero chance to be determined as a Hungarian word (0,0012), while the least precisely allocated word this time is “strawberry” (0.4233).

8.3 The ’NNV’ network

This section was created using the mentioned algorithms in Chapter 3.

8.3.1 Network parameters

For the training and the testing process the following parameters were set within the NNV network:

Number of layers:	3
Output function of the neuron:	sigmoid function
Training function:	error backpropagation
Number of epochs:	2000
Threshold:	the biggest output of the three networks indicates the recognized language
Number of outputs:	3

The training was done by the function `netlearn` while the testing is done by `neteval`.

8.3.2 Simulation results

A total number of 5 simulations were run to allocate the average error rate. The result of the program is mostly stable. The average time of simulation is approximately 30 seconds. The simulation results of the 'NNV' network are shown in Figure 8.3.

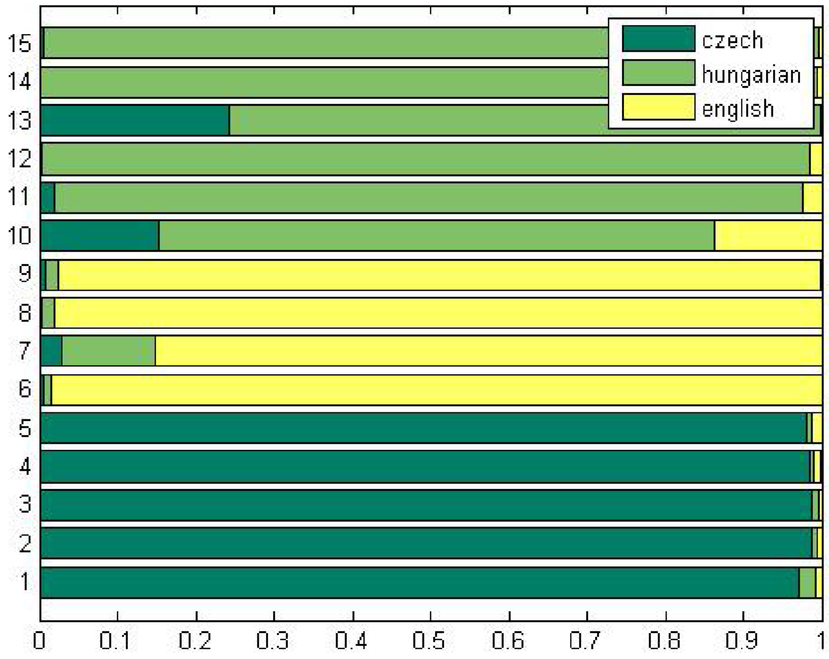


Fig. 8.3 Example of the simulation results in 'NNV'

The average simulation results for 'NN' network are listed in Table 8.4.

	Czech	**Hungarian**	**English**
hrozno	0,9798	0,0174	0,0085
jablko	0,9916	0,0064	0,0072
jahoda	0,9885	0,0136	0,0047
Ondřej	0,9869	0,0043	0,0157
pomeranč	0,9777	0,0090	0,0123
Andrew	0,0078	0,0069	0,9914
apple	0,0224	0,0860	0,7960
grape	0,0052	0,0180	0,9846
orange	0,0120	0,0171	0,9711
strawberry	0,0553	0,3941	0,1727
alma	0,0235	0,9682	0,0230
András	0,0044	0,9709	0,0246
eper	0,5571	0,8879	0,0003
narancs	0,0025	0,9967	0,0044
szőlő	0,0049	0,9921	0,0049

Table 8.4 The results of the simulation of 'NNV' in numbers

As the table shows, the network is able to recognize the language of all words except the word "strawberry", which is identified as a Hungarian word. The most precisely recognized word is "narancs" (0.9967) while the least precise result had the already mentioned "strawberry" (0.1727). The word "eper" has the smallest possibility to be identified as an English word (0,0003).

9. CONCLUSION

Within the scope of this master´s thesis, I tried to give a deep insight into the function of neural networks, starting with the base of the whole concept – real neurons. The first half of this paper describes the structure and the operation of real and artificial neurons including the description of the learning process and the manner and topology of their interconnections. The backpropagation algorithm is also described which is one of the basic types of neural network training. A detailed insight is given into fuzzy systems and fuzzy neural networks including the main advantages and disadvantages of fuzzy systems and the properties of both systems and clearly describes the problems which can be solved by combining these two techniques. The model of Fuzzy Neural Network and Barenji's ARIC (Approximate Reasoning Based Intelligent Control) architecture is also presented.

After introducing the Fuzzy Systems and Fuzzy Neural Networks, the Adaptive Neuro-Fuzzy Inference System (ANFIS) was presented which effectively combines both neural networks and fuzzy logic reasoning in order to achieve the best possible results. This type of network can be exceptionally suitable for the language recognition task too.

A prerequisite of network training is to acquire training data. In our case these were recordings of individual words. Five different words in three languages (English, Czech and Hungarian) were recorded for further network training and testing purposes for a total of 15 acquired words. For the training method, 60 words were used (every single word 4 times) and another 15 words for testing the system. In the framework of Matlab, a language recognition software was built, which has three separate network that can be used – the ANFIS network and a neural network based on toolbox functions and an own implementation of neural network trained by the backpropagation algorithm. Each network was fine-tuned for optimal functionality.

The goal of the work was to train the networks with the training words to gain the ability of recognizing the language of the words and, subsequently, test these trained networks. Each network was able to recognize all the languages. The best results were obtained using the ANFIS network. This network uses a hybrid learning algorithm, an effective combination of neural networks and fuzzy inference system while the other two networks are simple neural networks without the benefits of fuzzy logic reasoning. The ANFIS network was not able to decide two words; nevertheless, the recognition results of other words were superior to these ones. The ANFIS network has especially high demands on processing power and memory size.

The second network ('NN') produced very unsteady results. It was able to recognize all languages, however, the results have to be selected and averaged to achieve good recognition. The results of recognition were not as clear as in the other cases. On the other side, this method was the less time-consuming.

The third network was a custom-built neural network ('NNV') which utilized the error backpropagation learning algorithm. The network is able to decide the language of all words except of one. On the other hand it produces fairly good results comparable to ANFIS.

REFERENCES

[1] VIERA, J., DIAS, F.M. a MOTA, A. Neuro-Fuzzy Systems: A Survey. *WSEAS TRANSACTIONS on SYSTEMS.* April 2004, vol. 3, issue 2, s. 414-419. ISSN 1109-2777.

[2] MACLEOD, Christopher. *An Introduction to Practical Neural Networks and Genetic Algorithms For Engineers and Scientists*, 2004. Chapter 1, An introduction to Neural Networks, s. 1-5.

[3] MACLEOD, Christopher. *An Introduction to Practical Neural Networks and Genetic Algorithms For Engineers and Scientists*, 2004. Chapter 2, Artificial Neural Networks, s. 6-15.

[4] MACLEOD, Christopher. *An Introduction to Practical Neural Networks and Genetic Algorithms For Engineers and Scientists*, 2004. Chapter 3, The Back Propagation Algorithm, s. 16-27.

[5] JAIN, L.C.; MARTIN, N.M. *Fusion of Neural Networks, Fuzzy Systems and Genetic Algorithms: Industrial Applications.* CRC Press, CRC Press LLC, 1998., 368 s. ISBN 0849398045.

[6] FULLÉR, Robert. *Introduction to Neuro-Fuzzy Systems.* Advances in Soft Computing Series, Springer-Verlag, Berlin/Heildelberg, 2000., 289 s. ISBN 3-7908-1256-0.

[7] LIU, Puyin; LI, Hongxing. *Fuzzy Neural Network Theory and Application.* Series in Machine Perception and Artificial Intelligence – Vol. 59, World Scientific Publishing Co. Pte. Ltd., 2004., 376 s. ISBN 981-238-786-2.

[8] FULLÉR, Robert. *Neural Fuzzy Systems.* Åbo Akademis tryckeri, Åbo, ESF Series A:443, 1995., 249 s. ISBN 951-650-624-0, ISSN 0358-5654.

[9] VOLNÁ, Eva. *Neuronové sítě 1.* Ostrava, 2002. Studijní materiály pro distanční kurz: Neuronové sítě 1. Ostravská univerzita v Ostravě, Přírodovědecká fakulta.

[10] RAHMAT, Basuki; JOELIANTO, Endra. *Adaptive Neuro Fuzzy Inference System (ANFIS) with Error Backpropagation Algorithm using Mapping Function.* International Journal of Artificial Intelligence. Autumn 2008, Vol. 1, Number A08, s. 3-8. ISSN 0974-0635.

[11] KASABOV, Nikola K. *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering.* The MIT Press, 1996. ISBN 0-262-11212-4.

[12] ELWAKDY, A. M., ELSEHELY, B. E., ELTOKHY, C. M., ELHENNAWY, D. A. Speech Recognition using a Wavelet Transform to Establish Fuzzy Inference System through Substractive Clustering and Neural Network (ANFIS). *INTERNATIONAL JOURNAL of CIRCUITS, SYSTEMS and SIGNAL PROCESSING* [online]. 2008, vol. 2, issue 1 [cit. 2012-04-11]. ISSN 1998-4464. Dostupný z: http://www.naun.org/journals/circuitssystemssignal/2008.htm

[13] JANG, Jyh-Shing R. ANFIS: Adaptive-Network-Based Fuzzy Inference System. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS* [online]. 1993, vol. 23, no. 3 [cit. 2012-04-15].
Dostupný z: http://ece.ut.ac.ir/Classpages/S86/ECE406/Papers/ANFIS.pdf

LIST OF SYMBOLS, ABBREVIATIONS AND VARIABLES

ANN	Artificial Neural Net
MPL	Multi-Layer Perceptron
ARIC	Approximate Reasoning Based Intelligent Control
AEN	Action-State Evaluation Network
ASN	Action Selection Network
FIS	Fuzzy Inference System
ANFIS	Adaptive Neuro Fuzzy Inference System
LSE	Least-Squares Estimator
EBP	Error Backpropagation
FFT	Fast Fourier Transformation

LIST OF INSERTS

A MATLAB PROGRAMS

A.1 spust.m

```
clear; clc;
if (matlabpool('size')==0)
  matlabpool open;
end;

num_anfis_inputs=140;

mode='anfis';
%mode='nn';^
%mode='nnv';

tic;
fprintf('Load train data...\n');
traindir = 'train1';  %nacita treningove vzorky do traindir

[inp,tgt]=wavload(traindir,num_anfis_inputs);

%% train
fprintf('Train...\n');

switch mode
    case 'anfis'
        epoch_n = 3;
        parfor i=1:size(tgt,2)
            fprintf(' %d...\n',i);
            in_fis(i) = genfis2(inp,tgt(:,i),.20);
            %in_fis(i) = genfis2(inp,tgt(:,i),.3);
            out_fis(i) = anfis([inp
tgt(:,i)],in_fis(i),epoch_n,zeros(1,4));
            res(:,i) = evalfis(inp,out_fis(i));
        end
    case 'nn'
        nnet = feedforwardnet([10 10 8 8]);
        nnet = configure(nnet, inp', tgt');
        nnet.trainParam.epochs = 50;
        nnet.trainParam.goal = 0.0005;
        nnet.trainParam.min_grad=1e-10;
        nnet.trainParam.max_fail=10;
        nnet = train(nnet,inp',tgt');
        res = sim(nnet, inp')';
    case 'nnv'
        nnet = netinit(size(inp,2), size(tgt,2));
        nnet = netlearn(nnet, inp, tgt, 2000);
        res = neteval(nnet, inp);
end
%% test data

fprintf('Eval. test data...\n');
testdir = 'test1';  %nacita treningove vzorky do traindir

[tinp,ttgt]=wavload(testdir,num_anfis_inputs);
```

```
switch mode
    case 'anfis'
        warning('off', 'Fuzzy:evalfis:InputOutOfRange');
        tres=[];
        for i=1:size(tgt,2)
            fprintf(' %d...\n',i);
            tres(:,i)=evalfis(tinp,out_fis(i));
        end
    case 'nn'
        tres = sim(nnet', tinp')';
    case 'nnv'
        tres = neteval(nnet, tinp);
end

figure(1);
colormap(summer);
tres(tres<0)=0;
barh(tres./(sum(tres,2)*[1 1 1]),'stacked','DisplayName','tres ratios');
xlim([0 1]);
legend('czech','hungarian','english');
%%
toc;
fprintf('End.\n');
```

A.2 wavload.m

```
function [ inp, tgt ] = wavload( traindir, num_anfis_input_params)
% load wavs from a dir and convert to parameters
%   Detailed explanation goes here

files = dir([traindir '\*.wav']);

inp=[]; %vstupni signaly v radcich
tgt=[]; %nastavi jazyky (podle predpony 'c,h,e') - target
for i=1:length(files)   %nacita kazdy soubor
    [inp_tmp,Fs]=wavread([traindir '\' files(i).name]);

    inp_tmp = inp_tmp ./ max(max(abs(inp_tmp)));
    %inp = [inp; inp_tmp(round(linspace(1, length(inp_tmp),
num_anfis_input_params)),1)'];  %vybira 100 prvek z prvniho sloupce / vlozi
do radku
    inp = [inp; fftparams( inp_tmp, 14, Fs ) ];

    lang=files(i).name(1);
    if (lang == 'c')
        tgt = [tgt; 1 0 0];
    elseif (lang == 'h')
            tgt = [tgt; 0 1 0];
    elseif (lang == 'e')
            tgt = [tgt; 0 0 1];
    else
            tgt = [tgt; 0 0 0];
    end
end
end
```

A.3 fftparams.m

```
function [ params ] = fftparams( sa, nsecs, Fs )
%fft parameters, sa: input signal, tsec: sectioning time,
%   Detailed explanation goes here
params=[];

bandpassfilter_struct =...
 design(fdesign.bandpass('n,f3dB1,f3dB2',8,100,2000,Fs),'butter');

fftfilter=[.25 .5 .25];
fmax=3000;

sa= filter(bandpassfilter_struct, sa);

salen = length(sa);
slen = floor(salen/nsecs);

for i=0:nsecs-1

    sbgn = i*slen+1;
    send = (i+1)*slen;
    s = sa(sbgn:send);

    f = 0 : Fs/(length(s)-1) : fmax;
    sfft = abs(fft(s));
    fsfft=filter(fftfilter,1,sfft);

    [pks, locs] = findpeaks(fsfft(1:length(f)), 'SORTSTR',
'descend','minpeakdistance',3);
    try
        locs = locs(1:5);
        pks = pks(1:5);
    catch
        locs = [0 0 0 0 0];
        pks = [0 0 0 0 0];
    end

    params = [params reshape([locs;pks],1,[]) ];
    %params = [params locs'];
end
```

A.4 netinit.m

```
function [ network ] = netinit( inputs, outputs )  %vstupy, vystupy

    %krok 0

    n=struct;   %struktura, obsahuje celou sit

    n.numin=inputs;
    n.numout=outputs;
    n.numz=inputs;

    n.v=(rand(n.numin,n.numz)-0.5)/1000;
    n.w=(rand(n.numz,n.numout)-0.5)/1000;

    n.v0 = (rand(n.numz,1)-0.5)/1000;
    n.w0 = (rand(n.numout,1)-0.5)/1000;

    n.alfa = 0.0001;


    network=n;

end
```

A.5 netlearn.m

```
function [ onet ] = netlearn( net, input, target, runs )

%input - vstupni signaly ve radcich

f = @(x) sigmf(x, [10 .5]); %sigmoid funkce
fa = @(x) (f(x)-f(x-0.001))/0.001;  %derivace funkci sigmoid

%krok 1
while (1)

%krok 2
    for i = 1:size(input,1)

        %krok 3
        x = input(i,:)';
        t = target(i,:)';

        %krok 4
        z_in = net.v'*x + net.v0;
        z = f(z_in);

        %krok 5
        y_in = net.w0 + net.w'*z;
        y = f(y_in);

        %krok 6
        delta = (t-y).*fa(y_in);
        deltaw = net.alfa.*(z*delta');
        deltaw0 = net.alfa.*delta;

        %krok7
        delta_in = net.w*delta;
        delta = delta_in.*fa(z_in);
        deltav = net.alfa.*(x*delta');
        deltav0 = net.alfa.*delta;

        %krok8
        net.w = net.w + deltaw;
        net.v = net.v + deltav;
        net.w0 = net.w0 + deltaw0;
        net.v0 = net.v0 + deltav0;

    end
    runs=runs-1;
    if (runs<=0)
        break;
    end
end

onet=net;
end
```

A.6 neteval.m

```
function [ out ] = neteval(network, input)

%input - vstupni signaly ve radcich

f = @(x) sigmf(x, [10 .5]);
fa = @(x) (f(x)-f(x-0.001))/0.001;

out=[];
%krok 2
for i = 1:size(input,1)

    %krok 3
    x = input(i,:)';

    %krok 4
    z_in = network.v'*x + network.v0;
    z = f(z_in);

    %krok 5
    y_in = network.w0 + network.w'*z;
    y = f(y_in);

    out = [out; y'];
end
```