Speech recognition by Neural Networks

Vitoantonio Bevilacqua **Giuseppe Mastronardi** **Pierluigi Fasiello**

Politecnico di Bari, Dipartimento di Elettrotecnica ed Elettronica,
Via Orabona, 4, 70125 - Bari - Italy
Tel: +39 080 596 3252, E-mail: bevilacqua@poliba.it

Abstract

Human voice analysis, based on its acoustic properties compared to phonological categories of speech, allows to study those phonemes, as vowels, that can be used in the recognition of a speaker. This work describes a particular application of vocal recognition by Artificial Neural Network (ANN) which allows a discrimination out of different speakers in order to highlight an alternative method to check and/or to identify a human being. To make use of computational characteristics of ANN, the three Italian vowels /a/, /e/ and /o/ have been considered. The frequencies extracted from the signals of these phonemes, spoken by six different speakers (three males and three females), have been used to train and to test a neural network. Experimental results show that an ANN is able to classify the speakers in six classes and to recognise them as different ones.

Keywords:

Artificial Neural Network (ANN), Back-Propagation.

Introduction

Taking into consideration that voice was one of the first ways used by man to communicate and to get recognised by his similar, we tried to compare the vocal mark to fingerprints in order to value if voice allows the recognition of a person. Therefore, the target of this work is to find out if it's possible to identify a human subject using his voice's physical characteristics processed by particular learning and classification's systems: the neural networks. Basing on acoustic properties of human voice, we have characterised a speaker with some parameters (the frequencies of his vocal signal) by means of that it's possible to train a neural network and to evaluate its learning ability. To allow speaker's recognition we used two different types of neural network: a multi-layer network with Back-Propagation algorithm and a Kohonen's network in order to decide which one was able to solve the problem in a better manner. To simulate the two networks we used Matlab 6.5 and to analyse the vocal signal we used WaveLab 1.60 and Cool Edit 96. Finally, to extract the frequencies we used the software "Analisi Vocale (1999)" [2].

Approach and Methods

Analysis of acoustic parameters

One of the voice's analysis process is to study the spectrum and time parameters which represent the objective information of the vocal signal. This process consists of three different steps. The first step concerns the collection of voice's samples to analyse. We chose the Italian vowels /a/, /e/ and /o/, which are the right phonemes to select those characteristics of a person that allow a different classification out of different ones. The second step concerns the extraction of acoustic parameters. We extracted the fundamental frequency f_0 and the following four formant frequencies f_1, f_2, f_3 and f_4 by the vocal signal of our six speakers. Then we characterised them with the follow parameters f_1 / f_0, f_2 / f_0, f_3 / f_0, f_4 / f_0. Finally, in the third step these parameters are used to compare the analysed voice's samples and to classify them. This recognition phase was undertaken by neural network.

Neural networks

Neural networks are composed of simple elements called neurons, operating in parallel. These elements are inspired by biological nervous systems. As in nature, the network function is determined largely by the connections between elements. We can train a neural network to perform a particular function by updating the values of the connections (weights) between elements [1]. Commonly neural networks are trained, so that a particular input leads to a specific target output. Such a situation is shown below. There, the network is adjusted, based on a comparison of the output and the target, until the network output matches the target. Typically many such input/target pairs are used, in this supervised learning, to train a network.

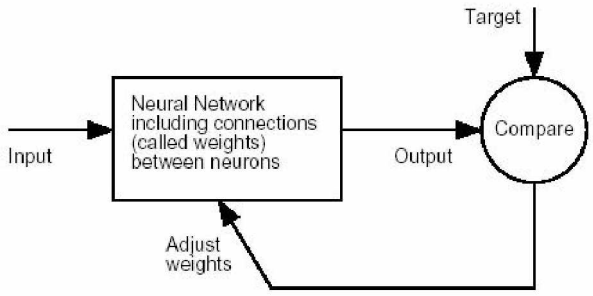

Figure 1 – Way to train an ANN in supervised learning

Batch training of a network proceeds by making weights' and biases' changes based on an entire set (batch) of input vectors. Incremental training changes the weights and biases of a network after presentation of each individual input vector. Incremental training is sometimes referred to as "on line" or "adaptive" training.

The supervised training methods are commonly used, but other networks can be obtained from unsupervised training techniques or from direct design methods. Unsupervised networks can be used, for instance, to identify groups of data. A one-layer network with R input elements and S neurons follows.

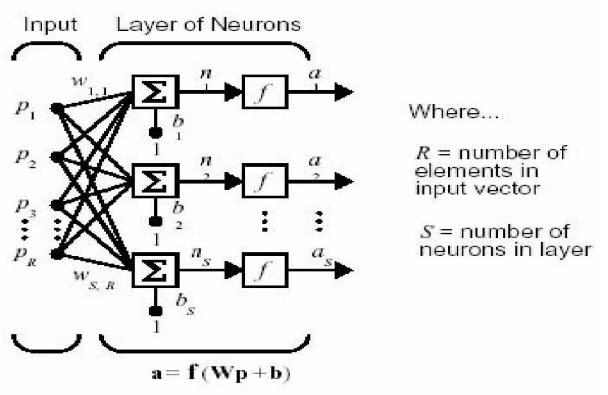

Figure 2 –One-layer neural network's architecture

In this network, each element of the input vector **p** is connected to each neuron input through the weight matrix **W**. The ith neuron has an adder that gathers its weighted inputs and biases to form its own scalar output n(i). The various n(i) taken together form an S-element net input vector **n**. Finally, the neuron layer outputs form a column vector **a**. The input vector elements enter the network through the weight matrix **W**:

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1R} \\ w_{21} & w_{22} & \cdots & w_{2R} \\ \cdots & \cdots & \cdots & \cdots \\ w_{S1} & w_{S2} & \cdots & w_{SR} \end{bmatrix} \quad (1)$$

Note that the row indexes on the elements of matrix **W** indicate the destination neuron of the weight, and the column indexes indicate which source is the input for that weight.

A network can have several layers. Each layer has a weight matrix **W**, a bias vector **b**, and an output vector **a**. The layers of a multilayer network play different roles. A layer that produces the network output is called an output layer. All other layers are called hidden layers. There are two basic types of input vectors that affect the simulation of networks: those that occur concurrently (at the same time, or in no particular time sequence), and those that occur sequentially in time. For concurrent vectors, the order is not important, and if we had a number of networks running in parallel, we could present one input vector to each of the networks. For sequential vectors, the order in which the vectors appear is important. More details can be found in [1]. Here we can only describe the architecture and the operation way of the two neural networks simulated to reach our target: the multilayer feedforward network with Back-Propagation algorithm and the self-organising Kohonen's network. Back-Propagation was created by generalising the Widrow-Hoff learning rule to multiple-layer networks and non-linear differentiable transfer functions. Standard Back-Propagation is a gradient descent algorithm, as is the Widrow-Hoff learning rule, in which the network weights are moved along the negative of the gradient of the performance function. The term Back-Propagation refers to the manner in which the gradient is computed for non-linear multilayer networks. Each input is weighted with an appropriate w. The sum of the weighted inputs and the bias forms the input to the transfer function f. Neurons may use any differentiable transfer function f to generate their output. Multilayer networks often use the log-sigmoid transfer function logsig (see [1]). Feedforward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear or sigmoid transfer function neurons. The network can be trained for function approximation (non-linear regression), pattern association, or pattern classification. The training process requires a set of examples of proper network behaviour-network inputs **p** and target outputs **t**. During training the weights and biases of the network are iteratively adjusted to minimise the network performance function. There are many variations of the Back-Propagation algorithm. The simplest implementation of Back-Propagation learning updates the network weights and biases in the direction in which the performance function decreases most rapidly - the negative of the gradient. There are two different ways in which this gradient descent algorithm can be implemented: incremental mode and batch mode. In the incremental mode, the gradient is computed and the weights are updated after each input is applied to the network. In the batch mode all of the inputs are applied to the network before the weights are updated. The batch steepest descent training function is trained. Self-organising in networks is one of the most fascinating topics in the neural network field. Such networks can learn to detect regularities and correlations in their input and adapt their future responses to that input accordingly. The neurons of competitive networks learn to recognise groups of similar input vectors. Self-organising maps learn to recognise groups of similar input vectors in such a way that neurons physically near each other in the neuron layer respond to similar input vectors. A basic reference is [2]. The architecture for a competitive network is shown below.

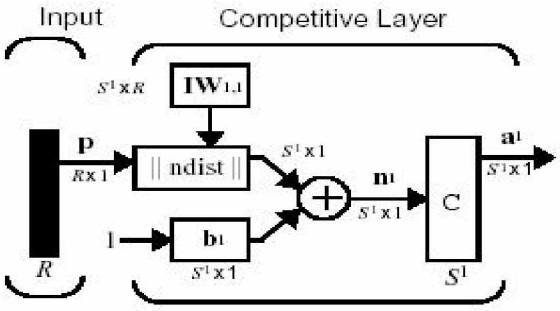

Figure 3 –Architecture of a competitive neural network

The box in this figure accepts the input vector **p** and the input weight matrix $\mathbf{IW}^{1,1}$, and produces a vector having $\mathbf{S}^1$ elements. The elements are the negative of the distances between the input vector and vectors i$\mathbf{IW}^{1,1}$ formed from the rows of the input weight matrix. ,The net input $\mathbf{n}^1$ of a competitive layer is computed by finding the negative distance between input vector **p** and the weight vectors and adding the biases **b**. If all biases are zero, the maximum net input a neuron can have is 0. This occurs when the input vector **p** equals that neuron's weight vector. The competitive transfer function accepts a net input vector for a layer and returns neuron outputs of 0 for all neurons except for the winner, the neuron associated with the most positive element of net input $\mathbf{n}^1$. The winner's output is 1. If all biases are 0, then the neuron whose weight vector is closest to the input vector has the least negative net input and, therefore, wins the competition to output a 1. The weights of the winning neuron (a row of the input weight matrix) are adjusted with the Kohonen learning rule [1]. Thus, the neuron whose weight vector was closest to the input vector is updated to be even closer. The result is that the winning neuron is more likely to win the competition the next time a similar vector is presented, and less likely to win when a very different input vector is presented. As more and more inputs are presented, each neuron in the layer closest to a group of input vectors soon adjusts its weight vector toward those input vectors. Eventually, if there are enough neurons, every cluster of similar input vectors will have a neuron that outputs 1 when a vector in the cluster is presented, while outputting a 0 at all other times. Thus, the competitive network learns to categorise the input vectors it sees.

Collection of voice's samples and extraction of frequencies

Because of variability of the frequencies extracted from the vocal signal around their mean value, we have collected 20 pronunciation of the three vowels and, for each of them, we have calculated the f_0, f_1, f_2, f_3 and f_4 frequencies to get the parameters: f_1 / f_0, f_2 / f_0, f_3 / f_0 e f_4 / f_0. These values have been used to obtain the training set of the ANN. Then the same vowels have been pronounced by the six speakers for other 10 times to get other values for the parameters above so that the test set remains defined. The process taken to extract the frequencies is the follow. From the signal we have selected a range of 120 ms around the stable area of the examined vowel. Then we have converted it into a 16 bit PCM mono signal sampled @ 9600 Hz. Finally, with the application "Analisi Vocale (1999)", we have obtained the so called formgraph by means of it, using the Cepstrum and Linear Predictive Code (LPC), we calculated the desired frequencies. Consequently, each speaker has been characterised by 30 parameters (20 for the training set and 10 for the test set) as shown in Table 1 for the vowel /a/. Similar tables have been obtained for the other vowels (/e/ and /o/). In fig. 4 is presented an example of formgraph:

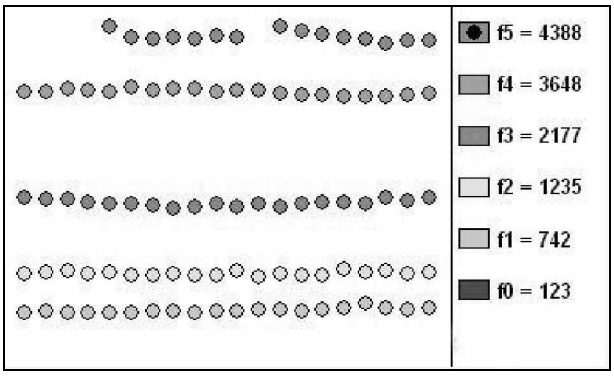

Figure 4 –Example of formgraph

Table 1 – Parameters for ANN's input, relative to vowel /a/

Speaker #1	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
a1	5,908333	10,35	18,06667	31,56667
a2	6,277311	10,37815	16,94118	32,58824
a3	5,983607	10,09016	17,59836	30,85246
…	…	…	…	…
a30	6,344538	10,67227	16,66387	30,08403
Speaker #2	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
a1	5,455285	10,8374	18,53659	29,90244
a2	5,965812	10,88889	18,92308	32,82051
a3	6,598214	11,77679	20,75	34,11607
…	…	…	…	…
a30	5,922414	11,41379	22,09483	33,12069
Speaker #3	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
a1	5,030612	6,77551	15,07653	19,72449
a2	4,983425	7,546961	15,56354	20,56906
a3	5	7,877193	14,85965	20,93567
…	…	…	…	…
a30	5,102041	6,969388	15,08673	17,79592
Speaker #4	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
a1	6,445255	8,350365	19,22628	26,0073
a2	6,262774	8,635036	19,09489	26,92701
a3	6,208633	9,071942	18,91367	26,19424
…	…	…	…	…
a30	6,841667	9,358333	21,46667	29,68333
Speaker #5	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
a1	6,394595	8,459459	15,6973	21,94054
a2	7,006024	8,76506	16,83735	25,24096
a3	6,19883	8,725146	18,09942	23,54971
…	…	…	…	…
a30	6,071429	7,452381	18,42262	24,19643
Speaker #6	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
a1	6,12973	8,437838	14,31351	22,52432
a2	6,885714	9,628571	16,08	24,4
a3	6,270718	8,530387	14,63536	23,54144
…	…	…	…	…
a30	6,07027	8,281081	14,58378	22,49189

Training set and test set

To obtain a good classification by simulated ANN, we decided to use only particular combinations of the parameters above. We used 8 input vectors as mean, element by element, of groups of 10 random vectors taken from the first twenty ones of Table 1 out of all the possible combinations equal to $C = \binom{20}{10}$. The obtained training set is shown in Table 2. At the same time we have calculated as mean of the last ten vectors of Table 1 the 6 input vectors of the test set (Table 3). Both Tables are relative to vowel /a/.

Table 2 – Training set for ANN's input, relative to vowel /a/

Speaker #1	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
Mean 1	6,28	10,17	18,75	32,12
Mean 2	6,23	10,09	18,26	31,02
…	…	…	…	…
Mean 8	6,39	10,24	19,21	31,99
Speaker #2	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
Mean 1	6,49	11,61	20,77	34,54
Mean 2	6,63	11,66	22,12	34,37
…	…	…	…	…
Mean 8	6,69	11,72	21,54	35,12
Speaker #3	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
Mean 1	4,99	7,31	15,60	20,06
Mean 2	4,95	7,34	15,28	19,93
…	…	…	…	…
Mean 8	4,94	7,32	15,32	19,97
Speaker #4	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
Mean 1	6,55	9,39	20,29	28,46
Mean 2	6,64	9,55	20,73	28,78
…	…	…	…	…
Mean 8	6,66	9,56	20,78	29,17
Speaker #5	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
Mean 1	6,24	8,54	17,08	22,61
Mean 2	5,81	8,07	16,90	22,38
…	…	…	…	…
Mean 8	6,18	8,45	17,19	22,72
Speaker #6	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
Mean 1	6,26	8,74	15,00	23,54
Mean 2	6,09	8,57	14,94	22,95
…	…	…	…	…
Mean 8	6,27	8,78	15,26	23,61

In this way we have a training matrix with 24x8 elements. After training the ANN, we have tested its classification ability supplying as input, one by one, the vectors of the test set. The results are discussed in the next two paragraphs, where we have simulated the two neural networks shown in the Introduction.

Table 3 – Test set for ANN's input, relative to vowel /a/

Speaker #1	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
Mean	6,25	10,30	18,29	30,34
Speaker #2	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
Mean	6,38	11,64	21,55	33,98
Speaker #3	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
Mean	4,90	7,28	16,05	19,90
Speaker #4	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
Mean	6,61	9,30	20,35	28,86
Speaker #5	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
Mean	5,99	8,41	17,30	22,69
Speaker #6	**f1/f0**	**f2/f0**	**f3/f0**	**f4/f0**
Mean	6,23	8,73	15,26	23,25

Speech recognition by Kohonen's neural network

Kohonen's neural network used in this section for the speech recognition is an unsupervised self-organising network. It learns the regularities of input vectors and classifies them into different classes without the use of a target output. The input matrix is shown below.

P_a= [6.2800 6.2300 … 6.1900 6.1200 6.3900
10.1700 10.0900 … 10.0300 10.0200 10.2400
18.7500 18.2600 … 18.7500 17.8000 19.2100
32.1200 31.0200 … 31.9600 31.1400 31.9900

6.4900 6.6300 … 6.7000 6.4300 6.6900
11.6100 11.6600 … 11.7700 11.5500 11.7200
20.7700 22.1200 … 21.5800 21.3500 21.5400
34.5400 34.3700 … 35.5500 33.8000 35.1200
…
…
…

6.2600 6.0900 … 6.0900 6.0800 6.2700
8.7400 8.5700 … 8.6000 8.5300 8.7800
15.0000 14.9400 … 14.9000 14.6800 15.2600
23.5400 22.9500 … 22.9400 22.8900 23.6100]

Successively, to make the network able to classify the speakers into six different classes, a competitive neural network with six neurons must be created. This can be done using the Matlab 6.5 toolbox on neural network with the command newc:

Net=newc ([3.00 8.00; 6.00 13.00; 13.00 24.00; 18.00 36.00], 6);

The first argument represents the variability range of input vector's elements. The second one indicates the number of neurons of the network.
The training phase is obtained with the two follow commands:

net.trainParam.epochs = 2000;
net = train(net, P_a);

The first command shows the number of epochs in the training step, the second runs the ANN's learning. During this phase, for every epoch, a single vector is presented as input to the network which changes the weights of its neurons according to the received input. The neuron whose weight vector is near to the input vector wins the competition and outputs 1. The others output 0. The same neuron will win every time is presented a similar input vector. In this way, on the basis of neuron's outputs, the network is able to assign a different class to every speaker.

To simulate the ANN and to show the speaker's classes indexes we used the commands:

Y = sim(net, P_a);
Yc = vec2ind(Y);

The training phase and simulation brought the results, for the output Y_c , shown in Table 4. For each speaker is presented the number class corresponding to his input vector:

Table 4 – ANN's output Y_c corresponding to input matrix P_a

Speaker #1							
6	6	6	6	6	6	6	6
Speaker #2							
3	3	3	3	3	3	3	3
Speaker #3							
1	1	1	1	1	1	1	1
Speaker #4							
2	2	2	2	2	2	2	2
Speaker #5							
5	5	5	5	5	5	5	5
Speaker #6							
4	4	4	4	4	4	4	4

We can see how ANN was able to correctly assign to the same class the input vectors of the generic speaker.
Now, to test the learning ability of the network we have to present as input the vectors of Table 3 (the test set). In this way we obtained:

Speaker #1:
>> p1=[6.25 10.30 18.29 30.34];
>> Y = sim(net, p1');
>> Yc = vec2ind(Y);
>> Yc
Yc = 6
Speaker #2:
>> p1=[6.38 11.64 21.55 33.98];
>> Y = sim(net, p1');
>> Yc = vec2ind(Y);
>> Yc
Yc = 3
Speaker #3:
>> p1=[4.90 7.28 16.05 19.90];
>> Y = sim(net, p1');
>> Yc = vec2ind(Y);
>> Yc
Yc = 1
Speaker #4:
>> p1=[6.61 9.30 20.35 28.86];
>> Y = sim(net, p1');
>> Yc = vec2ind(Y);
>> Yc
Yc = 2
Speaker #5:
>> p1=[5.99 8.41 17.30 22.69];
>> Y = sim(net, p1');
>> Yc = vec2ind(Y);
>> Yc
Yc = 5
Speaker #6:
>> p1=[6.23 8.73 15.26 23.25];
>> Y = sim(net, p1');
>> Yc = vec2ind(Y);
>> Yc
Yc = 4

So, the ANN has assigned the vectors of the test set, ever used before, to the respective classes. We can say that Kohonen's neural network carried out the correct recognition.

Speech recognition by multilayer feedforward network with Back-Propagation algorithm

In this section is simulated a supervised multilayer feedforward network with Back-Propagation algorithm. It is made of an input layer of 28 neurons, two hidden layers of 32 neurons and an output layer with 1 neuron. The commands used are shown below (the input matrix P_a is the same of the previous paragraph):

T = [1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6]/6;
net=newff(minmax(P_a),[28,32,32,1],{'tansig','tansig','tansig','logsig'},'traingd');
net.trainParam.show = 25;
net.trainParam.lr = 0.05;
net.trainParam.epochs = 30000;
net.trainParam.goal = 2e-5;
[net, tr] = train(net, P_a, T);
a = sim(net, P_a);
where :

- T is the target vector. It gets the desired output;
- 'tansig' and 'logsig' are the transfer functions of the layers;
- 'traingd' is the training function in the B-P mode;

- show is used for the visualisation of training;
- lr is used to establish changes for the weights of neurons;
- epochs and goal cause the training to stop;
- train starts the training phase;
- sim simulates the network. It returns the output value.

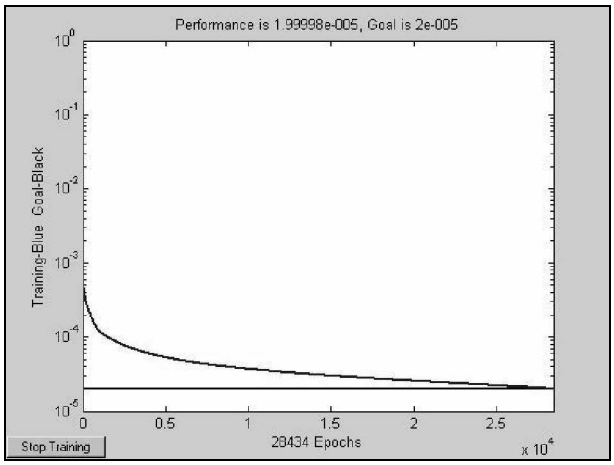

Figure 5 –Training of a supervised ANN with B-P algorithm

Table 5 – ANN's output a corresponding to input matrix P_a

Speaker #1							
0,160	0,163	0,168	0,173	0,165	0,160	0,171	0,174
Speaker #2							
0,332	0,337	0,335	0,332	0,333	0,333	0,332	0,333
Speaker #3							
0,501	0,497	0,499	0,503	0,502	0,499	0,506	0,494
Speaker #4							
0,661	0,676	0,668	0,662	0,674	0,660	0,664	0,668
Speaker #5							
0,834	0,826	0,837	0,828	0,840	0,832	0,836	0,834
Speaker #6							
0,996	0,993	0,995	0,995	0,996	0,993	0,994	0,995

So, the network has learnt to assign the output given by the target vector T to similar input vectors. Using the input vectors of the test set we can observe the classification capability of the ANN. Here are the results:

Speaker #1:
>> p1=[6.25 10.30 18.29 30.34];
>> a = sim(net, p1');
>> a
a = 0.1655
Speaker #2:
>> p1=[6.38 11.64 21.55 33.98];
>> a = sim(net, p1');
>> a
a = 0.3423
Speaker #3:
>> p1=[4.90 7.28 16.05 19.90];
>> a = sim(net, p1');
>> a
a = 0.4803
Speaker #4:
>> p1=[6.61 9.30 20.35 28.86];
>> a = sim(net, p1');
>> a
a = 0.6697
Speaker #5:
>> p1=[5.99 8.41 17.30 22.69];
>> a = sim(net, p1');
>> a
a = 0.8143
Speaker #6:
>> p1=[6.23 8.73 15.26 23.25];
>> a = sim(net, p1');
>> a
a = 0.9936
We can see how this network is able to correctly classify every speaker of our test.

Conclusion

The analysis of ANN here processed to try the recognition of different speakers has proved how both the simulated architecture are able to reach our objective. The only limit is the follow. If a speaker belongs to the reference group used to train the network he's correctly classified as shown above; on the contrary, the speaker is mistakenly assigned to one of the classes created in the previous learning phase. In this case, Kohonen's network doesn't work well because it assigns the speaker to the class corresponding to an input vector similar to new speaker's one. Instead, multilayer feedfoward B-P ANN could work well because the new speaker's input vector could get an output whose value is far from those desired and represented by the target vector, so it could be thought that he doesn't belong to the reference group. In any case, we think that it's possible to use ANN to operate an individual identification, for example, to log on a secret database or to focus on a suspect of a threatening call phone.
Our results have been reached using only simple phonemes, such as the vowels /a/, /e/ and /o/; but new developments could supply a better recognition by using phonemes more complex [3], words or even a full phrase.

References

[1] H. Demuth, M. Beale, 1992-2002. Neural Network Toolbox For Use with MATLAB®. The MathWorks: Natick (MA).

[2] T. Kohonen, 1987. Self-Organisation and Associative Memory, 2nd Edition. Springer-Verlag: Berlin.

[3] V. Amoruso, G. Mastronardi, 23-24 Novembre 1995. Fonemi.cla: classificazione di fonemi per la caratterizzazione del parlatore mediante formogramma. Roma, Italy: Atti delle 6 giornate di studio del GFS.