World Academy of Science, Engineering and Technology
International Journal of Psychological and Behavioral Sciences
Vol:2, No:7, 2008

A Cognitive Model for Frequency Signal Classification

Rui Antunes, Fernando V. Coito

***Abstract*—**This article presents the development of a neural network cognitive model for the classification and detection of different frequency signals. The basic structure of the implemented neural network was inspired on the perception process that humans generally make in order to visually distinguish between high and low frequency signals. It is based on the dynamic neural network concept, with delays. A special two-layer feedforward neural net structure was successfully implemented, trained and validated, to achieve minimum target error. Training confirmed that this neural net structure descents and converges to a human perception classification solution, even when far away from the target.

***Keywords*—** Neural Networks, Signal Classification, Adaptative Filters, Cognitive Neuroscience

I. Introduction

THE traditional processing methods for classifying signal frequencies, such as the digital and analog filters are yet quite dependent on the signal's shape, amplitude and noise, and very often have to be completely re-calculated when the signal frequencies changes.

This work is about using a "common sense" human inspired perception model, in order to create a simple and small dynamic neural network structure, which will be able to classify the high from the low frequency signals.

Neural networks are inspired from the biological nervous systems, like the brain (with axons, dendrites, nucleus and synapses), being composed of simple parallel operating elements.

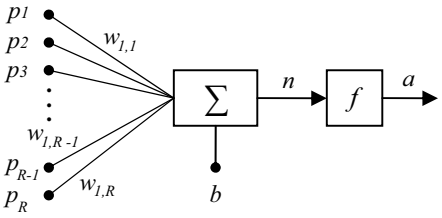

Fig. 1 The generic neuron

Manuscript received June 30, 2008. This work was done in Faculdade de Ciências e Tecnologia - New University of Lisbon (FCT-UNL).

Rui Antunes is with the Electrical Engineering Department of Escola Superior de Tecnologia de Setúbal, at the Setúbal Polytechnic Institute, Rua Vale de Chaves, Estefanilha, 2910-761 Setúbal, Portugal (phone: 351-265-790000; fax: 351-265-721869; e-mail: rui.antunes@estsetubal.ips.pt).

Fernando V. Coito is with the Electrical Engineering Department of Faculdade de Ciências e Tecnologia, at the New University of Lisbon, Quinta da Torre, 2829-516, Caparica, Portugal (phone: 351-21-2948300; fax: 351-21-2954461; e-mail: fjvc@fct.unl.pt).

Fig. 1 shows the internal structure of a generic neuron, where p_i represents the i^{th} input and $w_{1,i}$ a correspondent weight. The f function will be the neuron activation function.

Thus, the inputs to a generic neuron include its bias and the sum of its weighted inputs. The output response (a) of a neuron will rely on the neuron's inputs, and also on its activation function, which implements:

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \end{bmatrix} \qquad \mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix} \tag{1}$$

$$n = w_{1,1} \cdot p_1 + w_{1,2} \cdot p_2 + ... + w_{1,R} \cdot p_R + b = \mathbf{W} \cdot \mathbf{p} + b \tag{2}$$

$$a = \mathrm{f}(n) = \mathrm{f}(\mathbf{W} \cdot \mathbf{p} + b) \tag{3}$$

Other neurons may also be combined into one or more layers. So a neural network structure will be a formal mathematical representation of the total number of layers, the number of neurons for each layer, and its activation functions, and also the way layers are connected.

The choice for the neural network architecture relies mainly on the kind of the problem addressed.

II. Dynamic Networks With Sequential Inputs

When a neural network has no feedback or delays, it is commonly called a static network. Its inputs are concurrent and it doesn't matter if they occur in a particular time sequence.

A neural network which contains delays is called a dynamic network. In such cases, the input to the network is actually a sequence of input vectors in a certain time order (note that incremental and batch training techniques can be done both on dynamic and static networks).

A. The Tapped Delay Line

A tapped delay line (TDL) is used in a sequential inputs dynamic network, as shown in Fig. 2. An input signal $p[k]$ will enter and pass through (R-1) delays. Thus, the tapped delay line output will be an R-dimensional array, made with the current time input signal $p[k]$, and the previous

input signals ($p[k-1]$, $p[k-2]$, ... $p[k-R+1]$). This kind of dynamic network, when combined with a linear activation function can also be referred as a finite impulse response (FIR) digital adaptative filter.

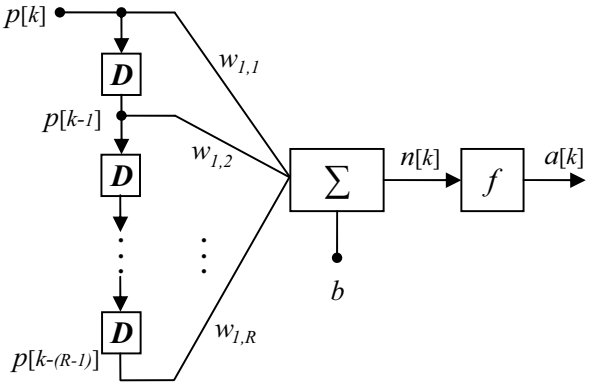

Fig. 2 A generic dynamic neural network with *R* delays

$$\mathbf{p} = \begin{bmatrix} p[k] \\ p[k-1] \\ \vdots \\ p[k-(R-1)] \end{bmatrix} \tag{4}$$

And the neural network output will be:

$$a[k] = \mathrm{f}(\mathbf{W} \cdot \mathbf{p} + b) = \mathrm{f}(\sum_{i=1}^{R} (w_{1,i} \cdot p[k-i+1]) + b) \tag{5}$$

B. Multiple Generic-Neurons Dynamic Network

One can have more than one neuron in the adaptive system, and the TDL may still be used when several (S) neurons are present:

$$\mathbf{W}_{\mathbf{L_m},\mathbf{L_{m-1}}} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix} \quad \mathbf{b}_{\mathbf{L_m}} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \tag{6}$$

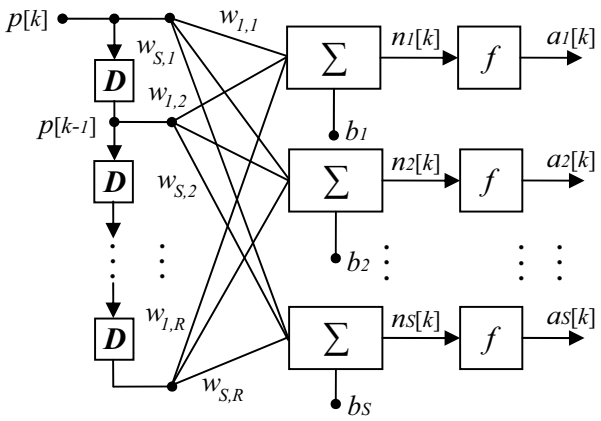

Fig. 3 Multiple generic-neurons dynamic network (an input layer of generic neurons - L_1)

For a given L_m layer, the correspondent network output ($a_{\mathrm{L_m}}$) can be expressed as:

$$a_{\mathrm{L_m}\,j}[k] = \mathrm{f}_{\mathrm{L_m}}(\sum_{i=1}^{R} (w_{\mathrm{L_m},\mathrm{L_{m-1}}\,j,i} \cdot p[k-i+1]) + b_{\mathrm{L_m}\,j}) \tag{7}$$

$$\mathbf{a}_{\mathbf{L_m}}[k] = \begin{cases} \mathrm{f}_{\mathbf{L_m}}(\mathbf{W}_{\mathbf{L_m},\mathbf{L_{m-1}}} \cdot \mathbf{p} + \mathbf{b}_{\mathbf{L_m}}) & \Leftarrow \mathrm{m} > 1 \\ \mathrm{f}_{\mathbf{L_1}}(\mathbf{W}_{\mathbf{L_1},\mathbf{L_1}} \cdot \mathbf{p} + \mathbf{b}_{\mathbf{L_1}}) & \Leftarrow \mathrm{m} = 1 \end{cases} \tag{8}$$

where $p[k]$ is the layer input.

A given neural network can also have one or more of such layers. All are usually named hidden layers, except for the layer that generates the network output (which is called the output layer).

III. Cognitive Neural Net Model

In this work, a special feedforward neural net cognitive model was proposed, inspired on a simple human perception approach, that first starts to amplify the high frequencies from the low frequencies, by using the derivative operator:

$$n_1(t) = \mathrm{I}_{\mathrm{W}_{1,1}} \frac{d}{dt} n_1(t) \tag{9}$$

expressed in discrete time as the difference equation:

$$n_1[k] = \mathrm{I}_{\mathrm{W}_{1,1}} \cdot (p[k] - p[k-1]) \tag{10}$$

which corresponds to

$$n_1[k] = \mathrm{I}_{\mathrm{W}_{1,1}} \cdot p[k] + \mathrm{I}_{\mathrm{W}_{1,2}} \cdot p[k-1]) + b_{\mathrm{L}_1 1} \tag{11}$$

with:

$$\begin{cases} \mathrm{I}_{\mathrm{W}_{1,2}} = -\mathrm{I}_{\mathrm{W}_{1,1}} \\ b_{\mathrm{L}_1 1} = 0 \end{cases} \tag{12}$$

A. First Hidden (Input) Layer

The first neuron of the neural net structure will work as a real-time derivative operator, able to amplify the high from the low frequencies. As a human doesn't really focus on the correspondent output sign, we proposed a non-linear activation function (the square function), able to further amplify the derivative output, and able also to turn it always positive.

That's in fact what a human first expects to do when he percepts [1] the high frequency signals - often he will simply start to try to amplify (greatly) the higher variations of its module.

So, the first layer output $a_{\mathrm{L}_1 1}[k]$ will be expressed as:

$$a_{\mathrm{L}_1 1}[k] = \mathrm{f}_{\mathrm{L}_1}(n_{\mathrm{L}_1 1}[k]) = (n_1[k])^2 \tag{13}$$

leading to:

$$a_{L_1 1}[k] = (\mathrm{Iw}_{1,1})^2 \cdot (p[k] - p[k-1])^2 = u[k] \quad (14)$$

B. Second (Output) Layer

Now, a human operator is able to visually integrate and threshold a part of the first amplified differenced output. Thus, an additional two-neuron output layer was proposed, to work as an integrator operator, along with a hard-limit output activation function ("hardlim"), which returns "0" or "1", and with two fixed bias values, for respectively classifying the high from the low frequencies.

In the continuous domain, we can write for the two integrated outputs:

$$x_1(t) = C_1 \int_{t_0}^{t} u(t)dt + b1 \quad (15)$$

$$x_2(t) = C_2 \int_{t_0}^{t} u(t)dt + b2 \quad (16)$$

And with F_S as the sampling frequency, we can obtain, for discrete time:

$$t - t_0 = R \cdot T_S = \frac{R}{F_S} \quad (17)$$

$$x_1[k] = \sum_{i=1}^{R} (L\mathrm{w}_{1,i} u[k-i+1]) + b1 \quad (18)$$

$$x_2[k] = \sum_{i=1}^{R} (L\mathrm{w}_{2,i} u[k-i+1]) + b2 \quad (19)$$

which will give, for the (two) outputs $a_j[k]$:

$$a_j[k] = \mathrm{f}_{L_2}(x_j[k]) = \mathrm{f}_{L_2}(\sum_{i=1}^{R} (L\mathrm{w}_{j,i} u[k-i+1]) + b_j) \quad (20)$$

$$a_j[k] = \mathrm{f}_{L_2}(\sum_{i=1}^{R} (L\mathrm{w}_{j,i} u[k-i+1]) + b_j) = \mathrm{hardlim}(x_j[k]) \quad (21)$$

$$a_{j=1,2}[k] = \begin{cases} 1 \Leftarrow \sum_{i=1}^{R} (L\mathrm{w}_{j,i} u[k-i+1]) + b_j) \geq 0 \\ 0 \Leftarrow \sum_{i=1}^{R} (L\mathrm{w}_{j,i} u[k-i+1]) + b_j) < 0 \end{cases} \quad (22)$$

Assuming a symmetrical condition between the high and the low frequencies, we can expect that the integrator gain relation might just be:

$$C_1 = -C_2 \quad (23)$$

and that will give:

$$L\mathrm{w}_{\substack{1,i \\ i=1..R}} = -L\mathrm{w}_{2,i} \quad (24)$$

C. Complete Neural Net Structure

The two-layer neural net complete structure is shown below:

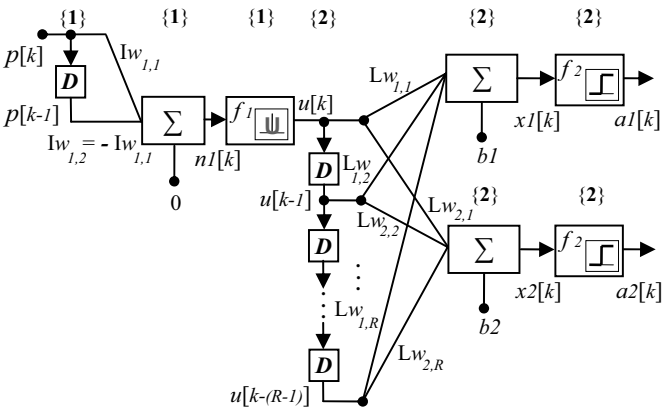

Fig. 4 Complete two-layer ({1} and {2}) neural network structure

The tapped delay lines have one delay for the first layer and eleven delays for the second layer, generating respectively, $p[k\text{-}1]$ and $u[k\text{-}1]$ to $u[k\text{-}10]$.

The second-layer hard-limit activation function makes the neural net to produce only output digital values. Nevertheless, one should remind that layers of neurons with nonlinear transfer functions (as the first one) will give further ability for the network to learn nonlinear and linear relationships.

The MATLAB sample code for creating this neural net structure is:

```
%global network creation
net=network; %create a neural network
net.numInputs=1; %one input
net.numLayers=2; %two layers
net.biasConnect=[0; 1];  %net structure
net.inputConnect=[1; 0];  %net structure
net.layerConnect=[0 0;1 0]; %net structure
net.layers{1}.size=1; %net structure
net.layers{1}.transferFcn='fquadrado'; %pre-defined square function
net.layers{2}.size=2; %net structure
net.layers{2}.transferFcn='hardlim'; %hard limit
net.outputConnect=[1 1]; %net structure
net.targetConnect=[1 1]; %net structure
net.inputWeights{1,1}.delays=[0 1]; %two delays
net.layerWeights{2,1}.delays=[0 1 2 3 4 5 6 7 8 9 10];
%eleven delays
net.IW{1,1}=[4 -4]; %Iw1,1=4   Iw1,2=-4
net.LW{2,1}=[1 1 1 1 1 1 1 1 1 1 1 ; -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1];
%Lw1,1=...=Lw1,11=1   Lw2,1=...=Lw2,11=-1
net.b{1}=[ ]; %layer1 bias b=0
net.b{2}=[-4 ;1]; %layer2 bias b1=-4   b2=1
```

IV. Training The Neural Net

To achieve the desired output target, the neural net can be trained by supervised learning [2], [3]. If the error goal or the maximum number of epochs is reached, the training stops, giving the net new parameters. Otherwise it will execute another epoch.

A. Input Signal Generation

The input for our experiment was a Simulink generated, one Volt amplitude sinusoidal signal, consisted by two different time-phased frequencies, respectively with one and four Hertz. The sampling rate F_S was 100Hz, and the signal's duration twenty seconds (2000 samples). A low-pass output filter $H(s)$ avoided any input signal discontinuities:

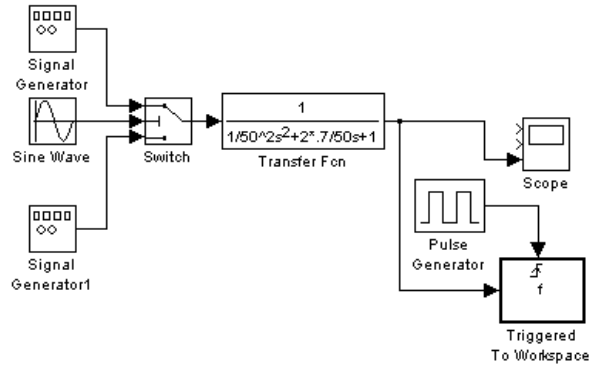

Fig. 5 Simulink input signal generation

$$H(s) = \frac{1}{4\times10^{-4}s^2 + 0.028s + 1} \tag{25}$$

B. The LMS Learning Algorithm (LEARNWH)

The Widrow-Hoff least-mean-squares weight/bias learning rule [4] uses an approximate steepest descent method to minimize the mean square error, by applying the squared error (at each epoch), in order to estimate it:

$$\begin{aligned}\mathbf{W}[k+1] &= \mathbf{W}[k] + 2\alpha\mathbf{e}[k]\mathbf{p}^T[k]\\ \mathbf{b}[k+1] &= \mathbf{b}[k] + 2\alpha\mathbf{e}[k]\end{aligned} \tag{26}$$

Therein ***b*** and ***e*** are, respectively, the bias and the error vectors, and α is a learning rate. The learning process could be faster when α is higher. Yet, if α is too high, it may increase the error and generate fully unstable training.

Although the non-linear characteristics of these neural net activation functions, the Widrow-Hoff LMS rule still performed (as will be confirmed) a simple, reliable and fast learning algorithm.

C. Training Results

The MATLAB sample code for training the neural net structure is show above:

```
%zero initial net conditions
net.IW{1,1}=0;
net.LW{2,1}=[0 0 0 0 0 0 0 0 0 0 ; 0 0 0 0 0 0 0 0 0 0];
net.b{2}=[0; 0];
%train parameters
net.trainFcn='trainb';
%Batch training with weight and bias learning rules
net.performFcn='mse';  %mean square error
net.inputWeights{1,1}.learnFcn='learnwh'; %Widrow-Hoff learning
net.layerWeights{2,1}.learnFcn='learnwh'; %Widrow-Hoff learning
net.biases{2}.learnFcn='learnwh'; %Widrow-Hoff learning
net.inputWeights{1,1}.learnParam=struct('lr',0.1); %learning rate
net.layerWeights{2,1}.learnParam=struct('lr',0.1); %learning rate
net.biases{2}.learnParam=struct('lr',0.1); %learning rate
net.trainParam.epochs=100; %max. number of epochs
net.trainParam.show=5;
net.trainParam.goal=0.0001; %goal
%net batch train
T1=A; %target
net=train(net,P1,T1);
```

The network was trained, starting with all weights and bias values from zero, for both layers, and taking a pre-defined *lr* learning rate. Given the correct output target (in this case two square waves, one for each frequency) for the $p[k]$ input signal, we obtained for the mean square error (MSE):

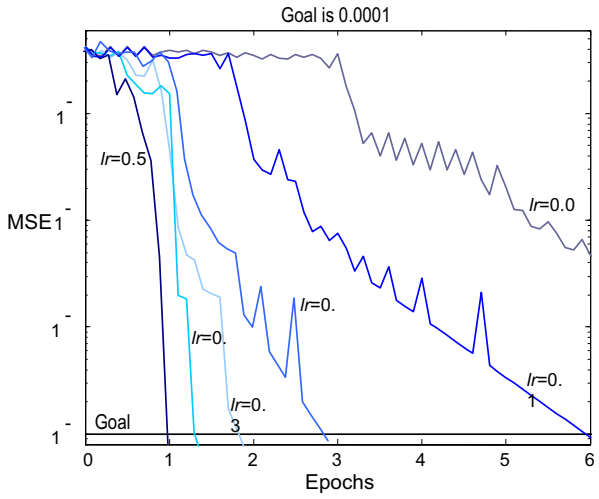

Fig. 6 Neural net training results at different learning rates

It should be stressed first that when the learning rate is above 0.52, the training will no longer converge to the goal (unstable learning). All other applied learning rates converged, however when the learning rate (*lr*) is lower, there are considerably more local minima on the descent to the goal, and the number of epochs must be higher to achieve it.

Trainb function trains the net (according to the weight and bias learning rules) with batch updates, and the weights and biases at the end of an entire epoch through the input data are updated. With $lr = 0.1$, the goal was reached within 60 epochs and the network ended up with:

$$\text{net.IW}\{1\}=[-3.99 \quad 3.96] \Rightarrow I_{W_{1,1}} = -3.99,\ I_{W_{1,2}} = 3.96$$

net.b{1}=[ ] $\Rightarrow b = 0$

net.LW{2}=

[36.22 35.78 34.25 32.39 30.49 29.03 28.48 29.12 30.97 33.72 36.80 ;
-50.56 -45.59 -40.59 -37.05 -35.08 -33.82 -32.65 -31.62 -31.13 -31.68 -33.54]

$\Rightarrow \mathbf{W}_{L_2,L_1}$ = net.LW{2}

net.b{2}=$\begin{bmatrix} -135.40 & 45.30 \end{bmatrix}^T \Rightarrow b1 = -135.40,\ b2 = 45.30$

One should remind that the bias value for the first layer is always null.

Fig. 7 shows the layer's weighs *Lw1,i* and *Lw2,i*, for each of the correspondent learning rate:

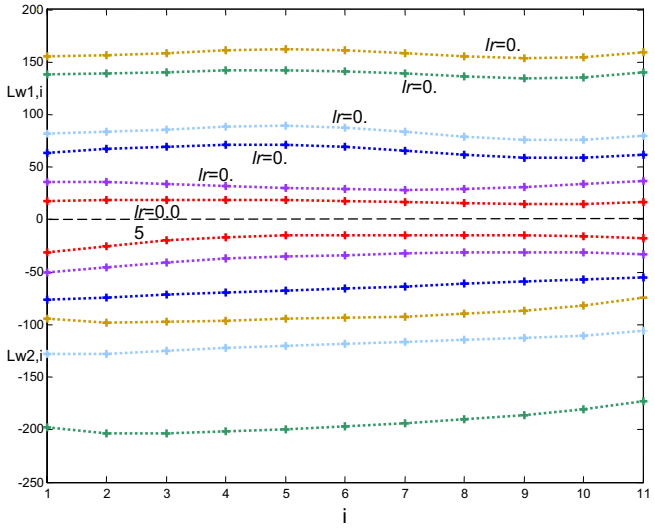

Fig. 7 Second layer weights for different learning rates

These weights (L*w1,i* and L*w2,i*) have always different sign (i.e. the two output neurons have an opposite sign relation for its inputs, as expected). Also, we can see that the trained weighs values associated with a same output neuron will vary

TABLE I
FIRST LAYER WEIGHTS VERSUS LEARNING RATE

	lr=0.05	lr=0.1	lr=0.2	lr=0.3	lr=0.4	lr=0.5
Iw1,1	-3.993	-3.993	-3.994	-3.995	-3.996	-3.994
Iw1,2	3.959	3.960	3.961	3.965	3.970	3.961

little. This means (and confirms) that the output neural

TABLE II
SECOND LAYER BIASES VERSUS LEARNING RATE

	lr=0.05	lr=0.1	lr=0.2	lr=0.3	lr=0.4	lr=0.5
b1	-68.8	-135.4	-254.8	-325.5	-554.0	-622.0
b2	24.0	45.3	82.2	155.1	232.0	115.5

As the values of *Iw1,1* and *Iw1,2* are almost symmetrical (for all the learning rates), it can be confirmed that the input layer neural network structure starts first to execute, as expected, a discrete instant derivative operation for the input signal *p*[*k*].

The second layer biases vary with the learning rate although *b1* and *b2* have always different signs, and its ratio module (for each of the learning rate value) varies between 2.10 and 5.39.

V. RESULTS

To test the neural network, it was introduced the filtered input signal already described in Chapter IV (Section A). The net weights and biases where obtained after training with *lr*=0.1, reaching (at 60 epochs) the goal (0.0001).

Figs. 8, 9 and 10 shows the first 700 samples for the *p*[*k*] input and also for all the *u*[*k*], *x1*[*k*], *x2*[*k*], *a1*[*k*] and *a2*[*k*] outputs:

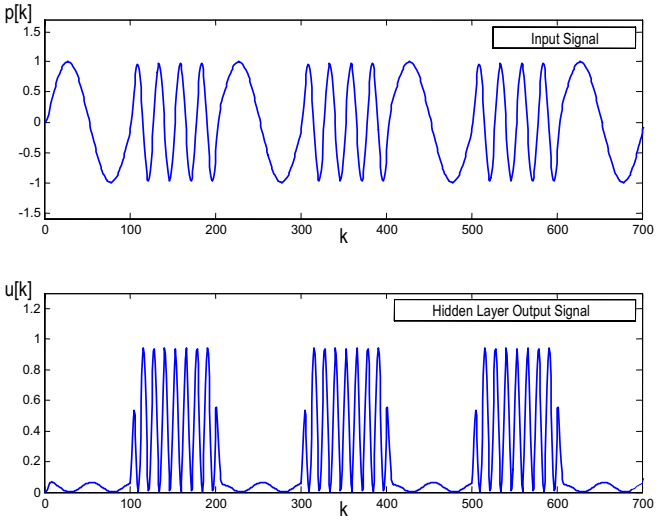

Fig. 8 Input signal *p*[*k*] and layer one output signal *u*[*k*]

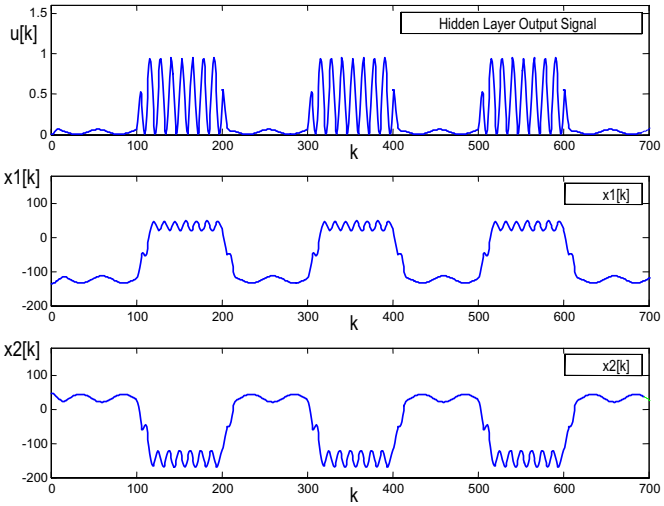

Fig. 9 First layer output signal *u*[*k*], and second layer *x*1[*k*] and *x*2[*k*] outputs (before the activation function)

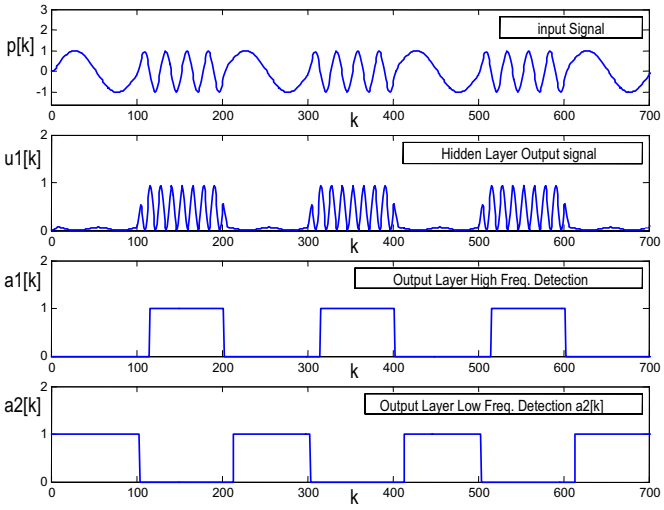

Fig. 10 Resulting final network outputs ($a1[k]$ and $a2[k]$), obtained from the trained input signal $p[k]$

It can be confirmed that the neural network successfully classifies the high and the low-frequency signal intervals.

For a cross validation we used two different amplitude, time-phased, sawtooth signals (respectively with 0.63Hz and 8Hz), at a 100Hz sampling rate, and with 20s duration. Again, it was applied the same low-pass output filter $H(s)$ (25):

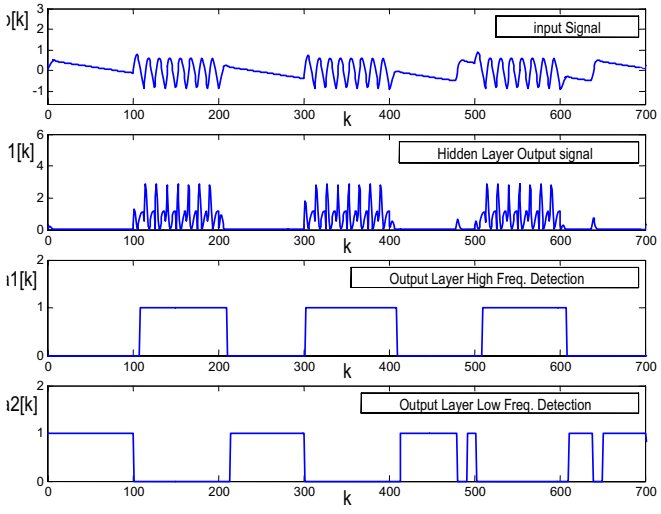

Fig. 11 Final outputs ($a1[k]$ and $a2[k]$) obtained from a filtered time-phased sawtooth signal input (new $p[k]$)

This neural net model is able to distinguish from the two different frequency signals, but as the net structure has only eleven delays, it might still generate some potential mismatches, specially in small time intervals with high amplitude variations - see $a2[k]$ between k=475 (4.75s) and k=485 (4.85s).

Nevertheless, a neural network with more delays in the second layer tapped delay line will necessarily increase the classification output delay (can be observed, with $a1[k]$ and $a2[k]$ near 1s, 2s, 3s, etc.). A possible solution for this dilemma lies in increasing the sample rate (and the correspondent number of acquired samples), i.e. to increase the array size of $p[k]$.

VI. Conclusions

Building a neural net model by first defining its basic structure, using a "common sense" cognitive heuristic, proved to be a good method before the network training, as both layer weights will converge to an expected perception solution. Keep in mind thought that when using neural net structures with tapped delay lines there are always inherent output classification delays, which can be reduced by increasing the signals acquisition sample rate.

The proposed model was successfully tested, and could be easily further developed also to classify more than two frequency signals. We only need to maintain the same neural model cognitive structure, and implement an additional output neuron for each frequency to classify.

References

[1] Bruce Goldstein, "Sensation and Perception", Sixth Edition, WADSWORTH, 2002.
[2] Paulo Gil, "Redes Neuronais Artificiais na Modelação e Controlo de Sistemas Dinâmicos", Controlo Inteligente, DEE/FCT/UNL.
[3] Leslie Smith, "An Introduction to Neural Networks", Department of Computing Mathematics, Centre for Cognitive and Computational Neuroscience, University of Stirling, UK, 2003. Available: http:/www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html
[4] Howard Demuth, Mark Beale, "Neural Network Toolbox For Use with MATLAB - User's Guide Version 3.0", The MathWorks, Inc, 1992.

Rui Antunes was born in Lisbon, in 1970. He received the degree in Electrical Engineering and Computers from the Technical University of Lisbon, Instituto Superior Técnico (IST), Portugal, in 1993, and the MSc degree in Electrical Engineering and Computers, in 1999, from the Technical University of Lisbon, Instituto Superior Técnico (IST), Portugal. Currently he is working towards his PhD degree in Electrical Engineering at Faculdade de Ciências e Tecnologia - New University of Lisbon (FCT-UNL), Portugal.
From 1994 up to 1996 he has worked as a Process and Product Engineer at Ford Electronics (Palmela), and he is currently Adjunct Teacher with the Electrical Engineering Department at Escola Superior de Tecnologia de Setúbal, in the Setúbal Polytechnic Institute (Portugal).

Fernando V. Coito was born in Lisbon, in 1961. He received the degree in Electrical Engineering from the Technical University of Lisbon, Instituto Superior Técnico (IST), Portugal, in 1986, the MSc degree in Electrical Engineering from the Technical University of Lisbon, Instituto Superior Técnico (IST), Portugal, in 1990, and the PhD in Electrical Engineering and Computers from the Technical University of Lisbon, Instituto Superior Técnico (IST), Portugal, in 1996.
From 1986 up to 1998 he performed his research activity at the IST Signal Analysis and Processing Center (CAPS) and at INESC. He is now a researcher at UNINOVA. His main scientific interests are dynamic process modeling, adaptive and fault tolerant control, and optimization.
He is currently Associate Professor with the Electrical Engineering Department at Faculdade de Ciências e Tecnologia - New University of Lisbon (FCT-UNL), Portugal.