Impact of Varying Neurons and Hidden Layers in Neural Network Architecture for a Time Frequency Application

Imran Shafi[1], Jamil Ahmad[2], MIEEE, Syed Ismail Shah[2], Sr. MIEEE, and Faisal M Kashif[3]

[1]Centre for Advance Studies in Engineering, Islamabad, Pakistan

Email: imran.shafi@gmail.com

[2]Iqra University Islamabad Campus, H-9 Islamabad, Pakistan

Email :{ jamil,ismail}@iqraisb.edu.pk

[3]Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge MA 02139, USA

Email: fmkashif@mit.edu

Abstract

In this paper, an experimental investigation is presented, to know the effect of varying the number of neurons and hidden layers in feed forward back propagation Neural Network Architecture, for a Time Frequency application. Varying the number of neurons and hidden layers has been found to greatly affect the performance of Neural Network (NN), trained via various blurry spectrograms as input over highly concentrated Time Frequency Distributions (TFDs) as targets, of the same signals. Number of neurons and hidden layers are varied during training and the impact is observed over test spectrograms of unknown multi component signals. Entropy and Mean Square Error (MSE) is the decision criteria for the most optimum solution.

Key words: Neural Networks, Back propagation, hidden layer, Time Frequency Analysis, Neurons

1. Introduction

The brain is a very efficient tool. Having about 100,000 times slower response time than computer chips, it (so far) beats the computer in complex tasks, such as image and sound recognition, motion control and so on. It is also about 10,000,000,000 times more efficient than the computer chip in terms of energy consumption per operation. An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way, the brain process information [1]. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well [6].

1.1 Human Vs Artificial Neuron. A typical human neuron collects signals from others through a host of fine structures called *dendrites*. The neuron sends out spikes of electrical activity through a long, thin stand known as an *axon*, which splits into thousands of branches. At the end of each branch, a structure called a *synapse* converts the activity from the axon into electrical effects that inhibit or excite activity from the axon into electrical effects that inhibit or excite activity in the connected neurons. When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes.

The essential features of human's neurons and their interconnections are estimated. We then typically program a computer to simulate these features. However because our knowledge of neurons is incomplete and our computing power is limited, our models are necessarily gross idealizations of real networks of neurons. A model of human's neuron Vs Artificial Neuron is presented in figure 3.

1.2 ANN Layers. The commonest type of ANN consists of three groups, or layers, of units: a layer of "**input**" units is connected to a layer of "**hidden**" units, which is connected to a layer of "**output**" units. The activity of the input units represents the raw information

that is fed into the network. The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units. The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units.

This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents. We also distinguish single-layer and multi-layer architectures. The single-layer organization, in which all units are connected to one another, constitutes the most general case and is of more potential computational power than hierarchically structured multi-layer organizations. In multi-layer networks, units are often numbered by layer, instead of following a global numbering.

The most widely used architecture in ANNs has been the Multiple Layer Perceptron (MLP), trained with the Back Propagation (BP) error learning algorithm. However, the MLP suffers from fundamental problems like convergence time, local minima and absence of a simple rule to obtain the right number of neurons and hidden layers.

In this paper we have used Feed forward Back Propagation NN to find the solution of the last problem. An ANN is trained with various blurry spectrograms as input to be mapped over highly concentrated TFDs of same signals [2]. Test spectrograms of multi component signals are then presented to trained NN. Optimum solution is explored for the application as far as number of neurons and hidden layers are concerned, to get the best concentration along Instantaneous Frequencies (IFs) for resultant images. The concentration is measured in terms of entropies [4] of the resultant TFDs. The lower the entropy, higher is the concentration along IF. In this paper entropy [4] of $Q(n,\omega)$ is considered as measure of concentration given by:

$$E_Q = -\sum_{n=0}^{N-1} \int_{-\pi}^{\pi} Q(n,\omega) \log_2 Q(n,\omega)\, d\omega \geq 0; \qquad (1)$$

Rest of the paper is organized as follows. Section 2 & 3 describes the NN architecture and procedural detail. Section 4 covers the simulation results and Section 5 concludes the paper.

2. The NN Architecture

The brain basically learns from experience. NNs are sometimes called machine learning algorithms, because changing of its connection weights (training) causes the network to learn the solution to a problem. The strength of connection between the neurons is stored as a weight-value for the specific connection. The system learns new knowledge by adjusting these connection weights. The learning ability of a neural network is determined by its architecture and by the algorithmic method chosen for training.

2.1 How BP Algorithm works?

In order to train a NN to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the neural network compute the error derivative of the weights (**EW**). In other words, it must calculate how the error changes as each weight is increased or decreased slightly. The BP algorithm is the most widely used method for determining the **EW**.

The BP algorithm is easiest to understand if all the units in the network are linear. The algorithm computes each **EW** by first computing the **EA**, the rate at which the error changes as the activity level of a unit is changed. For output units, the **EA** is simply the difference between the actual and the desired output. To compute the **EA** for a hidden unit in the layer just before the output layer, we first identify all the weights between that hidden unit and the output units to which it is connected. We then multiply those weights by the **EA**s of those output units and add the products. This sum equals the **EA** for the chosen hidden unit. After calculating all the **EA**s in the hidden layer just before the output layer, we can compute in like fashion the **EA**s for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives back propagation its name. Once the **EA** has been computed for a unit, it is straight forward to compute the **EW** for each incoming connection of the unit. The **EW** is the product of the EA and the activity through the incoming connection.

2.1.1 Various Steps. The BP algorithm consists of four steps:

1. Compute how fast the error changes as the activity of an output unit is changed. This error derivative (EA) is the difference between the actual and the desired activity.

$$\mathrm{EA}_j = \frac{\partial E}{\partial y_j} = y_j - d_j \qquad (2)$$

2. Compute how fast the error changes as the total input received by an output unit is changed. This quantity (EI) is the answer from step 1 multiplied by the rate at which the output of a unit changes as its total input is changed.

$$EI_j = \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \times \frac{dy_j}{dx_j} = EA_j y_j (1 - y_j) \quad (3)$$

3. Compute how fast the error changes as a weight on the connection into an output unit is changed. This quantity (EW) is the answer from step 2 multiplied by the activity level of the unit from which the connection emanates.

$$EW_{ij} = \frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{\partial W_{ij}} = EI_j y_i \quad (4)$$

4. Compute how fast the error changes as the activity of a unit in the previous layer is changed. This crucial step allows back propagation to be applied to multilayer networks. When the activity of a unit in the previous layer changes, it affects the activites of all the output units to which it is connected. So to compute the overall effect on the error, we add together all these seperate effects on output units. But each effect is simple to calculate. It is the answer in step 2 multiplied by the weight on the connection to that output unit.

$$EA_i = \frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{\partial y_i} = \sum_j EI_j W_{ij} \quad (5)$$

By using steps 2 and 4, we can convert the EAs of one layer of units into EAs for the previous layer. This procedure can be repeated to get the EAs for as many previous layers as desired. Once we know the EA of a unit, we can use steps 2 and 3 to compute the EWs on its incoming connections.

2.2 NN Topology

In this paper grayscale blurry TFDs are considered and Levenberg-Marquardt Back propagation (LMB) training algorithm with feed forward back propagation architecture is used. No of hidden layer and neurons are varied to find the optimum solution. The 'tansig' and 'poslin' transfer functions are used in between input-hidden layers and hidden-output layers respectively. Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear and linear relationships between input and output vectors. The linear output layer lets the network produce values outside the range -1 to +1.

3. The Procedural Details

Here we have targeted a Time Frequency application to find the most optimum topology/architecture of NN. To achieve the objective, we proceeded as under:

a. Number of sub spaces are decided for clustering the available data.

b. We select normalized sub space direction vectors that will best represent the subspaces. The directional vectors are used to characterize different types of edges in the image. The choice is dictated by the problem of deblurring. Here are few issues that are considered:

   (1) Edges are important image characteristics.

   (2) Blurring results in loss of edge information from images.

   (3) The process of deblurring may produce a more useful image.

c. Input data is vectorized and correlation between each input vector and directional vectors is calculated to assign it to the correct subspace. This creates a certain clustering effect on the input vectors since a vector will lie in the subspace represented by directional vector that is most similar to this vector with respect to its information content.

d. For each cluster, NNs are trained by varying the nunber of neurons and hidden layers. Test TFDs are given to the trained NNs to find the most optimum solution in terms of number of neurons and hidden layers for the application under consideration. Best topology/architecture is finalized on the basis of performance measured as entropies of the resultant images.

3.1 Training/Test TFDs

To train the NNs with algorithm described earlier, the spectrogram of the two parallel chirps signal is used as input. The grayscale spectrogram of this signal is shown as figure 4. The respective target time-frequency plane image of same signal is shown in figure 4.

3.1.1 Parallel Chirps Signal It is given by:

$$Y(n) = x_1(n) + x_2(n) \qquad (5)$$

Where $x_1(n) = e^{j\omega_1(n)n}$ with $\omega_1(n) = \pi n / 4N$ and $x_2(n) = e^{j\omega_2(n)n}$ with $\omega_2(n) = \frac{\pi}{3} + \frac{\pi n}{4N}$;

Here N represents the total number of points in the signal.

3.1.2 Test Signal we have fed spectrogram of single chirp signal as test image (figure 5) to the trained NN. Discussion of experimental results is presented in next section.

4. Simulation Results

There are a lot of factors that affect the performance of NN such as number of hidden layers, neurons in the hidden layer, learning rate and momentum term etc. We have carried out simulation for the first two factors which are presented below:

4.1 Effect of number of neurons in the hidden layers

We have studied the effect of the number of neurons in the hidden layer. The network was tested with 2,3,4,5,10,15,20,30,40 and 50 neurons in single/multiple hidden layer(s). The network never converged to a stable point when we tried the network with neurons upto 30. The reason being that the less number of neurons take the data from the input grid, and hence fails to convey the correct information to the next layers. The results were satisfactory with 35 neurons in the hidden layer, but by increasing the number of neurons further, no improvement was observed in the reduction of error in last epoch as shown in figure 1. Entropy values are also minimum for 40 or more neurons irrespective of number of hidden layers, as given in table I.

4.2 Effect of number of hidden layers

Number of hidden layers is the most important criteria while studying the architecture of the NNs. We have varied the number of hidden layers for the given input sets and the results are shown in figures 6 to 12. It is noted that the result even deteriorated if we use more then single hidden layer. Our study also verifies that the complex non linear problem at hand can be solved with single hidden layer, so there is no significant need for 2 or more layer architecture. The same fact is strengthened by the entropy values mentioned in table I.

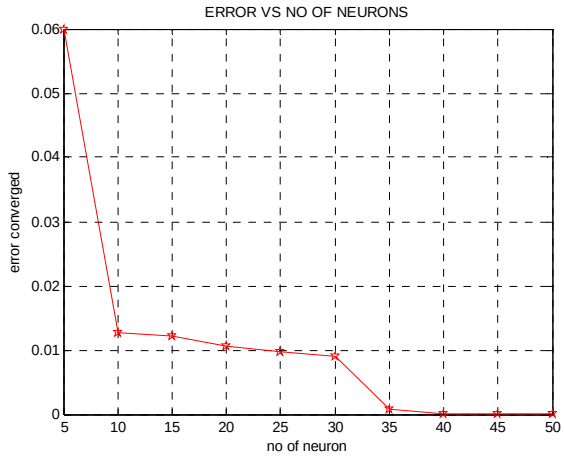


Figure 1: Error Vs Number of neurons in single hidden layers

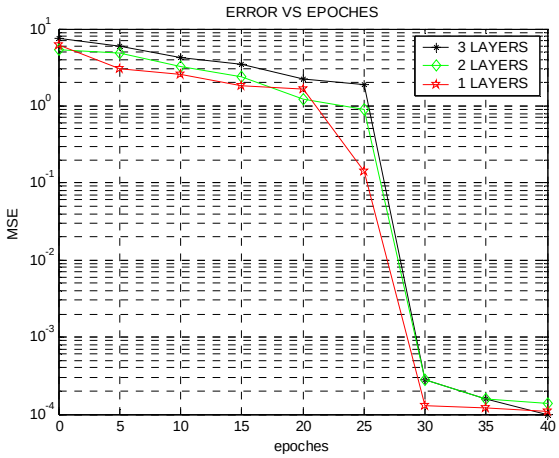


Figure 2: Error Vs epochs performance for various number of hidden layers

5. Conclusions

The simulation results presented in the paper indicated that NN architecture composed of single hidden layer with 40 neurons is able to remove the blur from the unknown spectrograms effectively with minimum MSE in last epoch and lowest entropy values as given in Table I. Increasing the number of neurons/hidden layers further only seems to increase the complexity of the network, and is found to be unsuitable manifested by both visual (figures 6-12) and mathematical findings (Table I). Studying the effect of these parameters in other applications will be a major work for future research.

6. References

[1] K. Jain, J. Mao and K. M. Mohiddin, "Artificial Neural Network: A tutorial", IEEE Trans. on Computers, pp. 31-44, 1996.

[2] I. Shafi, J. Ahmad, S.I. Shah, FM. Kashif, " Evolutionary Denoised and Concentrated Time Frequency Distributions (TFDs) using Bayesian Regularized Neural Network Model", Under Review at Journal of IEEE Transactions on Neural Networks, 2nd Draft submitted on 21 Aug 2006.

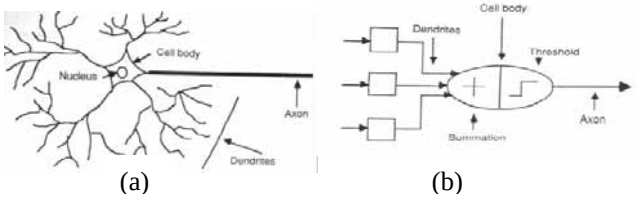
[3] I. Shafi, J. Ahmad, S.I. Shah, FM. Kashif, " Time Frequency Distribution using Neural Networks", Proceeding of IEEE International Conf on Emerging Technologies, pp. 32-35, Pakistan, 2005.

[4] R.M. Gray, "Entropy and Information Theory". New York Springer-Verlag, 1990.

[5] L. Cohen, "Time Frequency Analysis", Prentice-Hall, NJ, 1995.

[6] M.T. Hagan, H.B. Demuth & M. Beale, "Neural Network Design", Thomson Learning USA, 1996.

[7] J. Ahmad, I. Shafi, S.I. Shah, FM. Kashif, "Analysis and Comparison of Neural Network Training Algorithms for the Joint Time-Frequency Analysis", Proceeding of IASTED International Conf on Artificial Intelligence and application, pp. 193-198, Austria, Feb 2006.



(a) (b)

Figure 3: (a) Human Neuron (b) Artificial Neuron

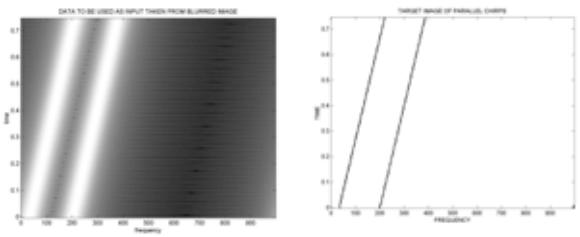


Figure 4: Input training/target images of parallel chirps signal

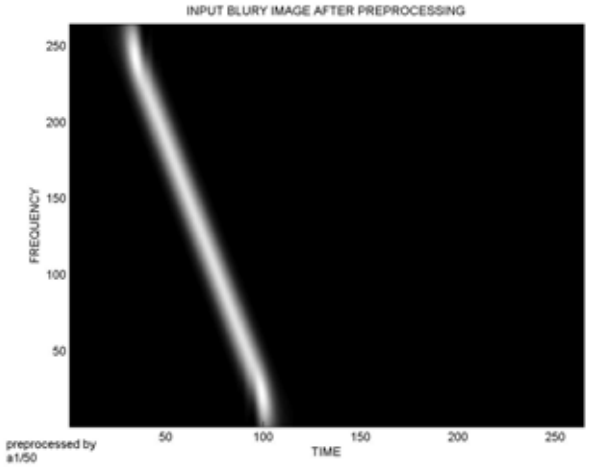


Figure 5: Test image of single chirp signal

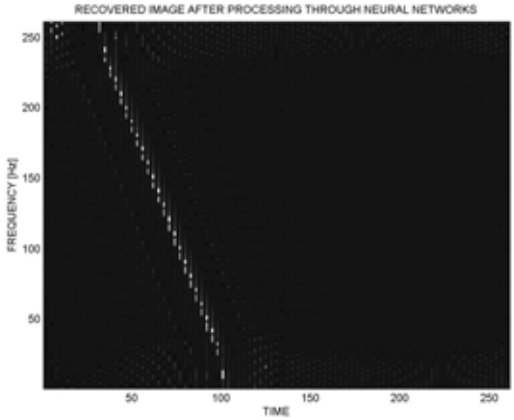


Figure 6: Resultant image with 2 layers, 50 neurons

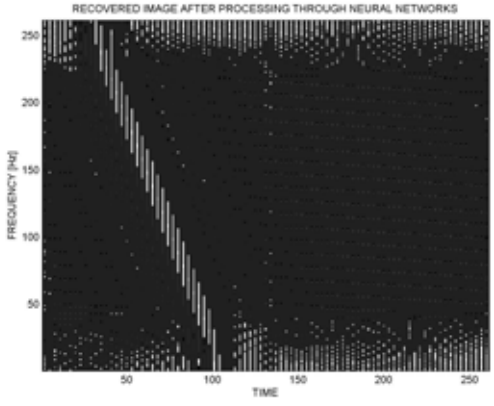


Figure 7: Resultant image with 2 layers 5 neurons

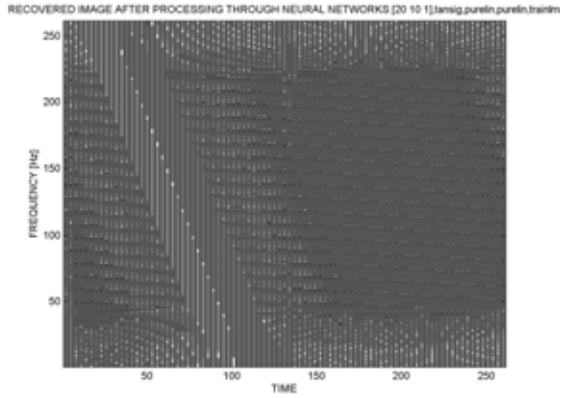


Figure 8: Resultant image with 3 layers 5 neurons

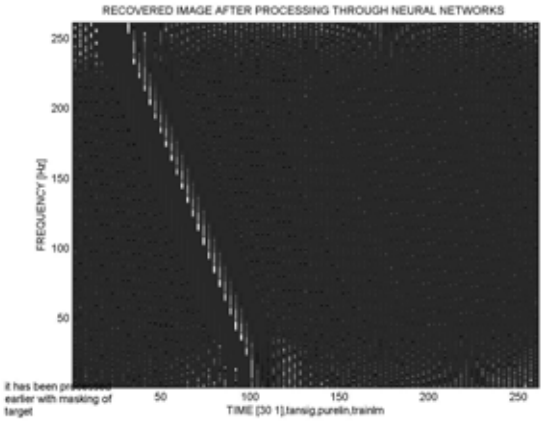


Figure 9: Resultant image with 3 layers 20 neurons

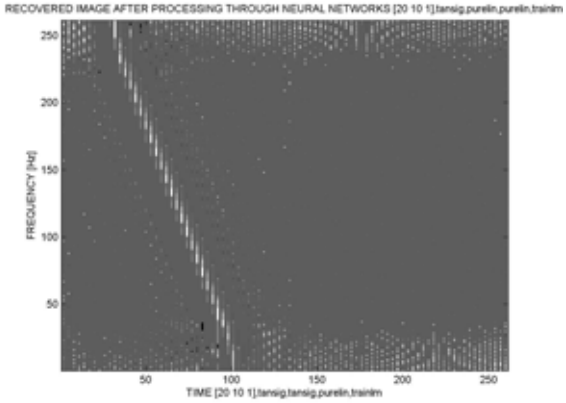


Figure 10: Resultant image with 2 layer 15 neurons

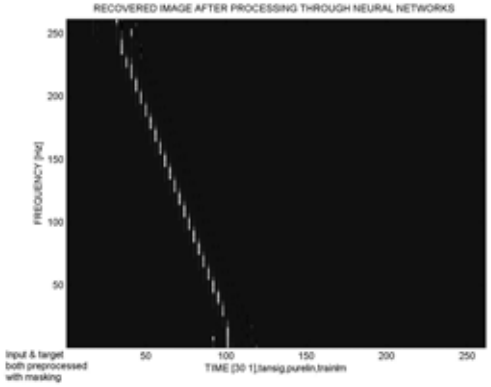


Figure 11: Resultant image with 1 layer 20 neurons

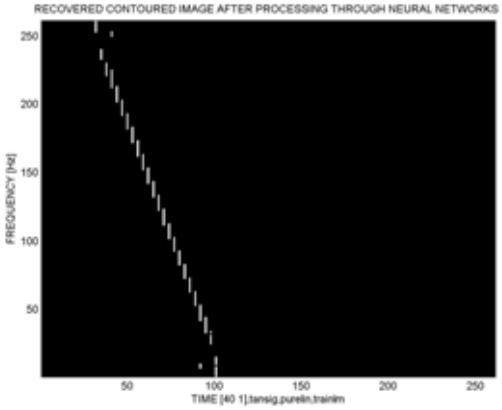


Figure 12: Resultant image with single hidden layer having 40 neurons

TABLE I

Impact of varying neurons and hidden layers over entropy of resultant image

Description	**Number of Neurons**				
	10	**20**	**30**	**40**	**50**
E_Q **bits for single layer**	10.20	9.31	9.01	8.20	8.20
E_Q **bits for 2 layers**	20.41	16.31	15.60	11.21	11.21
E_Q **bits for 3 layer**	22.56	14.30	12.10	10.24	10.24