ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

Performance Study of TDNN Training Algorithm for Speech Recognition

[I]Rubya Shaharin, [II]Uzzal Kumar Prodhan, [III]Dr. Md. Mijanur Rahman
[I]MS Student, [II,III]Assistant Professor
[I,II,III]Dept. of CSE, Jatiya Kabi Kazi Nazrul Islam University, Mymensingh, Bangladesh

Abstract
Time delay neural network is special types of architecture of artificial neural network whose functionality is to work with continuous data. It permits a classification of the phoneme by taking into account the dynamic aspect of speech and consequently to overcome problems of co articulation phenomenon. This paper concern about the performance of training algorithm used to train TDNN. Set of speech data are used to analyze the performance of training algorithm. The feature of speech data are extracted from speech spectrogram by using Otsu's thresholding method. The extracted features are fed into TDNN with 8 different training algorithms to adjust the synaptic weights in order to minimize the error between the computed output and the desired output for all samples. Fifty percent of speech data are used to training and other fifty percent of data are used to test the network. From the experiment It have been seen that scaled conjugate gradient outperform over other algorithm. Conjugate Fletcher-Reeves Update Achieved average accuracy 95% for unknown and 99% of known speech word.

Keywords
Artificial Neural Network, Isolated word, MLP, TDNN and Training Algorithm

I. Introduction

For 40 years, Artificial Neural Networks (ANNs) have been used for difficult problems in pattern recognition [Viglione, 1970]. In 1989 waibel et al introduce special neural network architecture for phoneme recognition which is known as Time Delay Neural Network (TDNN) [1]. Since then TDNN is vastly used for different types of pattern recognition in various platform(ex. Turbo C, C++, python, MATLAB etc).The architectural properties is almost same to the MLP only TDNN has an extra properties is known as tapped delay line. For the training criteria artificial neural network is different from the conventional computing system. By the Training facility ANN can learn as like human being. For training the network it use some well known training algorithm such as Back-propagation, gradient decent, delta rule etc. The best training algorithm can adjust weight efficiently, and enhance the recognition accuracy. TDNN is the best recognition system for speech recognition. The problems arise to choice of best training algorithm. The aim of this paper is to identify the best TDNN training algorithm for speech recognition. To evaluate the TDNN system for speech recognition we use 10 isolated speech words as input.

This paper organized as follows: Section I describes the introduction and arrangements of this paper. Section II describes about the artificial neural networks, MLP, TDNN etc. Section III introduces the training of TDNN and some well known training algorithm. Section IV discusses the methodological steps of this work. Section V shows the experimental result and Section VI concludes this paper.

II. Artificial Neural Network

Artificial neural network organized as layers, each layers contains of interconnected node called neuron and activation function. The first layer is called input layer which is directly attached to the input. The layer which produces output is known as output layer. And the layers between input and output are known as hidden layers. Each layer has its own weight matrix. Figure 1 shows the architecture of a single neuron:

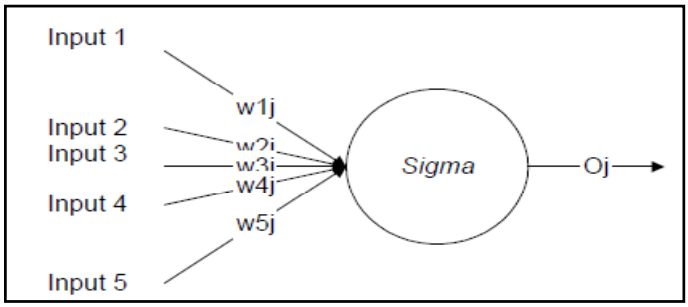

Fig. 1: A single layer perceptron model

In this figure input 1 …input 5 are the inputs of this neuron w1j… w5j are the weights, sigma is the transfer function and Oj is the output of this neuron. For the single layer architecture training doesn't require. The output of this neuron is produce by following equation:

Output = sigma(∑input*weight)

Multilayer perceptron (MLP) constructed by one input layer one output layer and one or more hidden layer. Inputs of this network are connected to the input layer with input weight for each neuron. Weighted sum of inputs known as outputs of input layer are going to the hidden layer as input of hidden unit. The outputs of hidden layer are connected to the output layer with hidden layer weights. And finally we get output from output layer.

Each layer has its own set of weights and an activation function. It is a static process. The final output is compare to its desired output. When the distance between the actual and desired output is near zero then stop the process. Otherwise change the weight to find optimal solution. The process of changing the weight according to the desired output is known as training the network. Figure 2 represent the basic architecture for MLP with one hidden unit.

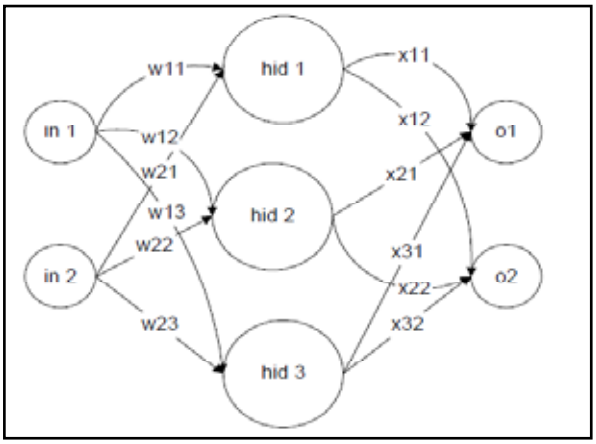

Fig. 2: Multilayer Perceptron

Here in 1 and in 2 represent the input, w11, w12,….,w23 represent weight and o1, o2 are the output. It has one hidden unit, with three neuron and 2 input neurons and 2 output neuron.

A. Time Delay Neural Network

Time delay neural network (TDNN) [13] is a specialized ANN architecture which primary purpose is to deal with sequential or speech data. TDNN was first introduced by Waibel et al in 1989 for phoneme recognition. TDNN is outperformed over Hidden markov model (HMM) for phoneme recognition.
TDNN has same architecture as simple feed forward neural network associates with tapped delay line in its input neuron and hidden neuron. The basic TDNN architecture [1] shown in fig. 3.

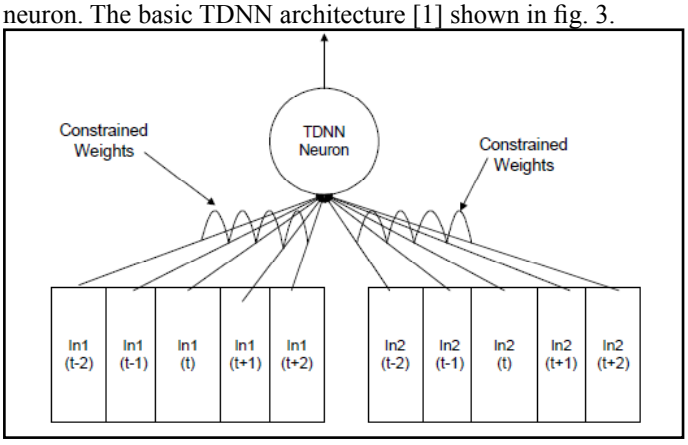

Fig. 3: TDNN neuron

The primary feature of TDNNs is the time-delayed inputs to the nodes. Each time delay is connected to the node via its own weight, and represents input values in past instances in time.
In Figure 3-we see a TDNN unit with 2 inputs. Not only is the current value of each input fed into the network (In1(t) and In2(t)), but the past two and future two values are also fed in (In1(t-2), In1(t-1), In1(t+1), In1(t+2) etc.). The purpose of this is to force the TDNN to generalize not only on the current input, but also across past and future inputs.

III. Training of TDNN

Although the dynamic networks can be trained using the same algorithm that's used for static network training. The performance of the algorithms on dynamic network can be quite difference and the gradient must be computed in complex way.
First apply regular forward and backward pass to all time shifted as if they were separate events. This yields different error derivatives for corresponding time shift connection rather than changing the weights on time shift separately. However we actually update each weight on corresponding connections by the same value namely average of all corresponding Time delayed weight changes.
The training procedure is same to the MLP. TDNN uses same training algorithm for train, such as Levenberg-Marquardt backpropagation (trainlm), Bayesian regularization (trainbr), Fletcher-Powell conjugate gradient backpropagation(traincgf), Polak-Ribiere conjugate gradient backpropagation (traincgp), Powell -Beale conjugate gradient backpropagation (traincgb), Scaled conjugate gradient backpropagation (trainscg) etc.

1. Backpropagation Algorithm

Back propagation [8] is a common method of teaching artificial neural networks how to perform a given task. It is a supervised learning method, and is a generalization of the delta rule. It requires a teacher that knows, or can calculate, the desired output for any input in the training set. It is most useful for feed-forward networks. The term is an abbreviation for "backward propagation of errors". Back propagation [12] requires that the activation function used by the artificial neurons (or "nodes") be differentiable [9].

Algorithm:

Notation:

x_j^l : input to node j of layer l

w_{ij}^l: weight from layer l-1 node I to layer l node j

∂(n): sigmoid function

θ_j^l: bias of j^{th} node of layer l

t_j: target value of node j of the output layer

o_j^l: output of node j at layer l

Given a set of training data points or target data t_j and output o_j.We can write error function E as:

$$E=\frac{1}{2}\sum_{k\in k}(o_k - t_k)^2$$

We let the error of the network for a single training iteration is denoted by E. We want to calculate $\frac{\Delta E}{\Delta w_{jk}^l}$, the rate of changes of the error with respect to the given connective weight. So we can minimize it.

Now we consider two cases. The node is an output node or it is in hidden layer node.

For Output layer:

$$\frac{\Delta E}{\Delta w_{jk}} = \frac{\Delta}{\Delta w_{jk}} \frac{1}{2} \sum_{k\in k}(o_k - t_k)^2$$

$$= (o_k - t_k) \frac{\Delta}{\Delta w_{jk}} o_k$$

$$=(o_k - t_k) \frac{\Delta}{\Delta w_{jk}} \partial(x_k)$$

$$=(o_k - t_k)\, \partial(x_k)\, (1\text{-}\partial(x_k)) \frac{\Delta}{\Delta w_{jk}} x_k$$

$$= (o_k - t_k)\, o_k(1 - o_k) o_j$$

For notation purpose I will define Δ_k to be the expression $(o_k - t_k)\, o_k(1 - o_k)$so we can rewrite the equation

$$\frac{\Delta E}{\Delta w_{jk}} = \Delta_k\, o_j \text{ Where } \Delta_k = (o_k - t_k)\, o_k(1 - o_k)$$

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

2. Resilient Backpropagation (Rprop)

Resilence backpropogation (trainrp) training algorithm eliminates the effects of the magnitudes of the partial derivatives [10]. In this sign of the derivative is used to determine the direction of the weight update and the magnitude of the derivative have no effect on the weight update. The size of the weight change is determined by a separate update value. The update value for each weight and bias is increased by a factor whenever the derivative of the performance function with respect to that weight has the same sign for two successive iterations [11]. The update value is decreased by a factor whenever the derivative with respect that weight changes sign from the previous iteration. If the derivative is zero, then the update value remains the same. Whenever the weights are oscillating weight change will be reduced.

Algorithm [7]:

Repeat:

Compute the gradient $\frac{\delta E}{\delta w}(t)$

For all weights and biases{

If { $(\frac{\delta E}{\delta w_{ij}}(t-1) * \frac{\delta E}{\delta w_{ij}}(t) > 0)$ then

$\Delta_{ij}(t)$ = minimum $\left(\Delta_{ij}(t-1) * \eta^{+}, \Delta max\right)$

$\Delta w_{ij}(t)$ = -sign$\left(\frac{\delta E}{\delta w_{ij}}(t)\right) * \Delta_{ij}(t)$

$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$

$\frac{\delta E}{\delta w_{ij}}(t-1) = \frac{\delta E}{\delta w_{ij}}(t)$

}

Else if { $(\frac{\delta E}{\delta w_{ij}}(t-1) * \frac{\delta E}{\delta w_{ij}}(t) < 0)$ then

$\Delta_{ij}(t)$ = maximum $\left(\Delta_{ij}(t-1) * \eta^{-}, \Delta min\right)$

$\frac{\delta E}{\delta w_{ij}}(t-1) = 0$

}

Else if { $(\frac{\delta E}{\delta w_{ij}}(t-1) * \frac{\delta E}{\delta w_{ij}}(t) = 0)$ then

$\Delta w_{ij}(t)$ = -sign$\left(\frac{\delta E}{\delta w_{ij}}(t)\right) * \Delta_{ij}(t)$

$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$

$\frac{\delta E}{\delta w_{ij}}(t-1) = \frac{\delta E}{\delta w_{ij}}(t)$

}

}

3. Levenberg-Marquardt backpropagation (trainlm)

Levenberg Marquardt algorithm is fastest supervised learning algorithm. It updates weight and bias values according to the Levenberg Marquardt optimization method. The algorithm is:

1. Present all input the network and compute the corresponding network output and errors. Compute the mean square error over all inputs:

   $$E = \frac{1}{2}\sum_{k=0}^{p-1}\sum_{l=0}^{n0-1}(d_{kl} - a_{kl})^2$$

   Where d_{kl} and a_{kl} are the desired output and the actual output at the output neuron l for the input k of the network respectively. P is the total number of training pattern and n0 represent the total number of neuron at the output layer of the network.

2. Compute the Jacobian matrix j(w) where the weights and biases of the network.

3. Solve the Levenberg-Marquardt weight update equation

   $$\Delta w = [\,J^{T}(w)J(w)+\mu I]^{-1}J^{T}(w)R$$

   Where, $R = (d_{kl} - a_{kl})$

4. Re compute the error using $w+\Delta w$. If this new error is smaller than that computed in step 1 then reduce the training parameter μ by μ^{-} , let $w+\Delta w$, and go back to step 1. If the error is not reduced then increase μ by μ^{+} and go back to step 3. μ is the predefined value set by the user.

5. The algorithm is assumed to have converged when the norm of gradient is less than some predetermined value or when error has been reduced to some error goal.

Although levenberg-Marquardt is one of the fastest algorithm but it does require more memory than other algorithm.

4. Conjugate Gradient Algorithms

The basic back propagation algorithm adjusts the weights in the steepest descent direction (negative of the gradient). This is the direction in which the performance function is decreasing most rapidly. It turns out that, although the function decreases most rapidly along the negative of the gradient, this does not necessarily produce the fastest convergence. In the conjugate gradient algorithms a search is performed along conjugate directions, which produces generally faster convergence than steepest descent directions. In this section, we present four different variations of conjugate gradient algorithms.

Basic conjugate gradient algorithm [6]:

$x^{(0)} = 0,\ r_0 = b$

For k= 1, 2, ...

1. Return $x^{(k-1)}$ if $\|r_{k-1}\|_2 \leq \in \|b\|_2$
2. If k = 1, $p_{k=} r_0$; otherwise $p_{k=} r_{k-1} + \beta_{p_{k-1}}$

   Where $\beta = \frac{p_{k-1}^{T} A r_{k-1}}{p_k^{T} A\, p_{k-1}}$
3. Compute $x^{(k)} = x^{(k-1)} + \alpha p_k$

   Where $\alpha = \frac{\|r_{k-1}\|_2^2}{p_k^{T} A\, p_k}$ and $r_k = b - A_x(k)$

The conjugate gradient algorithms are usually much faster than variable learning rate back propagation, and are sometimes faster than trainrp, although the results will vary from one problem to another. The conjugate gradient algorithms require only a little more storage than the simpler algorithms, so they are often a good

choice for networks with a large number of weights.

5. Fletcher-Reeves Update (traincgf)

All of the conjugate gradient algorithms start out by searching in the steepest descent direction (negative of the gradient) on the first iteration.

$$p_0 = -g_0$$

A line search is then performed to determine the optimal distance to move along the current search direction:

$$x_{k+1} = x_k + \alpha_k p_k$$

Then the next search direction is determined so that it is conjugate to previous search directions. The general procedure for determining the new search direction is to combine the new steepest descent direction with the previous search direction:

$$p_k = -g_k + \beta_k p_{k-1}$$

The various versions of conjugate gradient are distinguished by the manner in which the constant β_k is computed. For the Fletcher-Reeves update the procedure is

$$\beta_k = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}}$$

Its need little more storage.

6. Polak-Ribiére Update (traincgp)

Another version of the conjugate gradient algorithm was proposed by Polak and Ribiére. As with the Fletcher-Reeves algorithm, the search direction at each iteration is determined by

$$p_k = -g_k + \beta_k p_{k-1}$$

For the Polak-Ribiére update, the constant β_k is computed by

$$\beta_k = \frac{\Delta g_{k-1}^T g_k}{g_{k-1}^T g_{k-1}}$$

This is the inner product of the previous change in the gradient with the current gradient divided by the norm squared of the previous gradient. The `traincgp` routine has performance similar to `traincgf`. It is difficult to predict which algorithm will perform best on a given problem. The storage requirements for Polak-Ribiére (four vectors) are slightly larger than for Fletcher-Reeves (three vectors).

7. Powell-Beale Restarts (traincgb)

For all conjugate gradient algorithms, the search direction will be periodically reset to the negative of the gradient. The standard reset point occurs when the number of iterations is equal to the number of network parameters (weights and biases), but there are other reset methods that can improve the efficiency of training. One such reset method was proposed by Powell [Powe77], based on an earlier version proposed by Beale [Beal72]. For this technique we will restart if there is very little orthogonality left between the current gradient and the previous gradient. This is tested with the following inequality.

$$\left|g_{k-1}^T g_k\right| \geq 0.2 \left\|g_k\right\|^2$$

If this condition is satisfied, the search direction is reset to the negative of the gradient.The `traincgb` routine has performance that is somewhat better than `traincgp` for some problems, although performance on any given problem is difficult to predict. The storage requirements for the Powell-Beale algorithm (six vectors) are slightly larger than for Polak-Ribiére (four vectors).

8. Scaled Conjugate Gradient (trainscg)

Scaled Conjugate Gradient (trainscg) does not require line search at each iteration step like other conjugate training functions. Step size scaling mechanism is used which avoids a time consuming line search per learning iteration. This mechanism makes the algorithm faster than any other second order algorithms.

Algorithm [5]:

mechanism makes the algorithm faster than any other second order algorithms.

Algorithm [5]:

1. Choose initial weight vector w_1

   Set $p_{1 = r_1 = -E'(w_1)}$,k=1
2. Calculate second order information

   $s_k = E''(w_k) = p_k$

   $\delta_k = p_k^T s_k$
3. Calculate step size

   $\mu_k = p_k^T r_k$

   $\propto_k = \frac{\mu_k}{\delta_k}$
4. Update weight vector

   $w_{k+1} = w_k + \propto_k p_k$

   $r_{k+1} = -E'(w_{k+1})$
5. If K mod N = 0 then restart algorithm

   $p_{k+1} = r_{k+1}$

   Else create new conjugate direction

   $\beta_k = \frac{|r_{k+1}|^2 - r_{k+1} r_k}{\mu_k}$

   $p_{k+1} = r_{k+1} + \beta_k p_k$

If the steepest decent direction $r_k \neq 0$ then set k= k+1 and go to step 2 else terminate and return w_{k+1} as the desired minimum

The trainscg function requires more iteration to converge than the other conjugate gradient algorithms, but the number of computations at each iteration is significantly reduced because no line search is performed [3].

9. BFGS Algorithm (trainbfg)

Newton's method is an alternative to the conjugate gradient methods for fast optimization. The basic step of Newton's method is:

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

$$x_{k+1} = x_k - A_k^{-1} g_k$$

Where A_k is the Hessian matrix (second derivatives) of the performance index at the current values of the weights and biases. Newton's method often converges faster than conjugate gradient methods. Unfortunately, it is complex and expensive to compute the Hessian matrix for feed forward neural networks. There is a class of algorithms that is based on Newton's method, but which doesn't require calculation of second derivatives. These are called quasi-Newton (or secant) methods. They update an approximate Hessian matrix at each iteration of the algorithm. The update is computed as a function of the gradient. The quasi-Newton method that has been most successful in published studies is the Broyden, Fletcher, Goldfarb, and Shanno (BFGS) update. This algorithm has been implemented in the trainbfg routine. This algorithm requires more computation in each iteration and more storage than the conjugate gradient methods, although it generally converges in fewer iterations. The approximate Hessian must be stored, and its dimension is , where *n* is equal to the number of weights and biases in the network.
For smaller networks, however, trainbfg can be an efficient training function.

10. One Step Secant Algorithm (trainoss)

Since the BFGS algorithm requires more storage and computation at each iteration than the conjugate gradient algorithms, there is need for a secant approximation with smaller storage and computation requirements. The one step secant (OSS) method is an attempt to bridge the gap between the conjugate gradient algorithms and the quasi-Newton (secant) algorithms. This algorithm does not store the complete Hessian matrix; it assumes that at each iteration, the previous Hessian was the identity matrix. This has the additional advantage that the new search direction can be calculated without computing a matrix inverse.
This algorithm requires less storage and computation per epoch than the BFGS algorithm. It requires slightly more storage and computation per epoch than the conjugate gradient algorithms. It can be considered a compromise between full quasi-Newton algorithms and conjugate gradient algorithms.

IV. Methodology

For this experiment we record 10 isolated Bengali digits from a single speaker. 20 samples are collected with the respect to each digit. The feature of speech words are extracted by binary feature extraction [14] method using Otsu's thresholding method.

A. Speech Data Processing

We record 10 Bengali digit speeches from a single speaker with 8KHz sampling rate by using MATLAB R2010a software as *.wav format. We collect 20 samples for each digit word. Our speech database has 200 samples for 10 digits. After getting speech files we detect end point that is identify the voiced part of speech signal and then calculate spectrogram for each speech file. Figure 4 and 5 shows the original speech signal and voiced part of original speech signal respectively. We get voiced signal by end point detection.

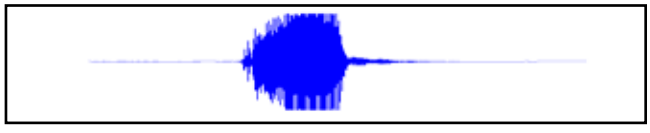

Fig. 4: Original speech signal

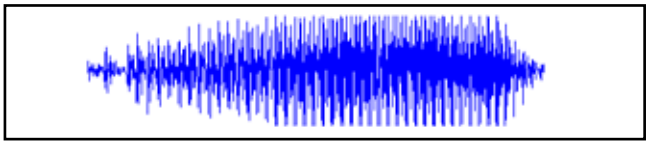

Fig. 5: Voiced part of speech signal

Spectrogram

The spectrogram is calculated by using the Short Time Fourier Transform (STFT) of audio signal. The Short Time Fourier Transform (STFT) can be expressed by:

$$\mathrm{STFT}\{x[n]\}(m,w) \equiv \mathrm{X}(m,w) = \sum_{-\infty}^{\infty} x[n]w[n-m]\, e^{-jwn}$$

Where X[n] = signal, W[n]= window function m is discrete and w is continuous. The spectrogram of signal x[n] can be estimated by computing the squares magnitude of Short Time Fourier Transform (STFT) of the signal and given by following equation:

$$\text{Spectrogram (n, w)} = |STFT(n,w)|^2$$

Spectrograms of speech files are stored as image with the extension (*.png). The image files are stored in a file. We use MATLAB 'imread'built in function to read the image files. Then we use mat2gray function that return matrix contain value 0.0(full black) and value 1.0(full white). Figure 6 shows the spectrogram of speech signal.

Fig. 6: Spectrogram image of speech signal

B. Convert image to binary image

For converting gray image to binary image its need to compute global threshold. That can be used to convert an intensity image to binary image. The threshold is computed by using MATLAB graythresh function. Graythresh function uses Otsu's method, which chooses the threshold to minimize the variance of the black and white pixel. Threshold is containing with a variable level.

A. Normalization

After choosing global threshold we convert binary format with the use of it. For convert binary format we use MATLAB im2bw function. Im2bw normalized intensity value that lies in the range [0,1]. Im2bw replace all pixel in the input image with luminance greater than level with the value 1(white) and replace all the pixel less than level with the value 0(black). Binary image of figure 6 is shown in the figure 7.

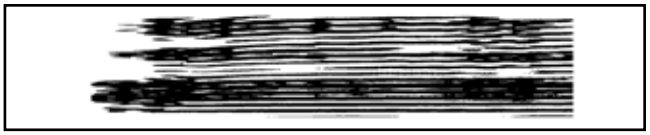

Fig. 7: Binary image of spectrogram image

B. Resize image

We want to resize image by 8x6. For this we use imresize function. After resizing we get 48x1 matrix for each input image. And this matrix is our desired input of Time Delay Neural Network (TDNN).

C. Training and test vector preparation

We have total ten input pattern or ten words. Each word has ten different samples. So we have total 200 samples of ten words. We use 50% that means 10 sample of each word to training the network and other 10 are used as testing pattern.

V. Experimental Result

To evaluate each algorithm for Time Delay Neural Network we create 3 layers distributed TDNN with 100 input neurons and 40 hidden neurons with the delay 0 to 3 for input layer and 0 to 5 for hidden layer. To examine the developed system we use 10 patterns of each isolated word as training pattern and rest of 10 patterns for each word as testing pattern. That is total 100 pattern are for training and other 100 pattern for testing the system. The Mean Squared Error (mse) is used as a performance function, tangent sigmoid are used as a transfer function for hidden layer and pure linear used as transfer function for output layer. The performance of the developed system for speech recognition is given by table 1. The table shows the word recognition accuracy (%), required time, epochs needed and performance for each algorithm.

Table 1: Performance of different training algorithm

Algorithm	Epoch	Time (s)	Performance (mse)	Accuracy (%) Known	Accuracy (%) Unknown
Trainlm	X	X	X	X	X
Trainscg	49	3.00	0.02	97	93
Traincgb	45	3.50	0.006	97	94
Traincgf	58	3.00	0.005	99	95
Traincgp	54	3.00	0.004	98	92
Trainrp	6	0.11	0.342	10	10
Trainbfg	X	X	X	X	X
Trainoss	64	3.30	0.029	96	95

In the table 1 ‘X’ symbol indicates that the algorithm does not train the network. “Out of memory” error has been occurred in this case. That is this algorithm is not preferred for large volume of data, those need extra memory for processing than others. The highest accuracy’s are marked bold.

VI. Conclusion

From this experiment we can see that levenberge-Marquardt algorithm and BFGS algorithm are not efficient for the large network or large volume of data, its need more memory than others algorithm.

For the large network we can use one of Conjugate gradient algorithm such as ‘traincgf’, ‘traincgp’, traincgb’, trainscg’. Although One Step Secant (trainoss) shows good accuracy for unknown pattern but it shows poor accuracy for known pattern. Good choice of training algorithm for large TDNN network is conjugate gradient Fletcher-Reeves Update algorithm for speech recognition.

VII. AcknowledgEment

The author would like to thanks to the Ministry of Information and Communication Technology of Bangladesh for giving financial support by providing MOICT-Fellowship.

References

[1] *Colin Keng-Yan TAN and Kim-Teng LUA,"Learning of Word Boundaries In Continuous Speech Using Time Delay Neural Networks".*

[2] *http://matlab.izmiran.ru/help/toolbox/nnet/backpr59.html*

[3] *Bhavna Sharma, and Prof. K. Venugopalan, "Comparison of Neural Network Training Functions for Hematoma Classification in Brain CT Images",IOSR Journal of Computer Engineering (IOSR-JCE);e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 1, Ver. II (Jan. 2014), PP 31-35 www.iosrjournals.org.*

[4] *N. N. R. Ranga Suri and Dipti Deodhare,P. Nagabhushan,"Parallel Levenberg-Marquardt-based Neural Network Training on Linux Clusters - A Case Study".*

[5] *Martin F. Møller,"A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning", November 13, 1990.*

[6] *L. Vandenberghe, "conjugate gradient method" EE236C (Spring 2013-2014).*

[7] *Martin Riedmiller,"Rprop-Description and Implementation Details" Technical Report, January 1994.*

[8] *Simon Haykin, “Neural Networks – A Comprehensive foundation”, 2nd Ed., Pearson Education, 2004.*

[9] *Ch. Jyosthna Devi, B. Syam Prasad Reddy, K. Vagdhan Kumar, N. Raja Nayak, "ANN Approach for Weather Prediction using Back Propagation",International Journal of Engineering Trends and Technology- Volume3Issue1-2012.*

[10] *A.D. Anastasiadis, G.D. Magoulas, and M.N. Vrahatis, “New globally convergent training scheme based on the resilient propagation algorithm”,Neurocomputing, 64, 2005, pp.253–270.*

[11] *http://www.rohan.sdsu.edu/doc/matlab/toolbox/nnet/backpr57.html*

[12] *Raúl Rojas,” The back propagation algorithm of Neural Networks - A Systematic Introduction, “chapter 7, ISBN 978-3540605058*

[13] *A. Waibel, T. Hanazawa, G. Hinton, K. Shikano and K. Lang, "Phoneme Recognition Using Time-Delay Neural Networks", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 37, No. 3, March 1989.*

[14] *Md. Mijanur Rahman and Md. Al-Amin Bhuiyan,"DYNAMIC THRESHOLDING ON SPEECH SEGMENTATION",IJRET: International Journal of Research in Engineering and Technology eISSN: 2319-1163 | pISSN: 2321-7308*