Performance Enhancement of Phoneme Recognition using GPUs

Tamizharasan P., Karthikeyan M, Ramasubramanian N. and A. Joshi

National Institute of Technology, Tiruchirappalli, India
{tamizh5500@gmail.com; karthik1992nms@gmail.com; nrs@nitt.edu; adj.comp@coep.ac.in }

Abstract: Phoneme recognition is a vital task in the automatic speech recognition process. The set of algorithms used for phoneme recognition requires effective handling of raw acoustic input data and demands huge amount of computations. New measures proposed in the pre-processing step and back propagation of artificial neural networks attains improved accuracy and performance benefit, on its implementation on Graphics Processing Unit (GPU). The implementation has been tested on audio files with an average size of 250 KB. It is observed that there is a considerable performance enhancement of around 10x in the GPU implementation against CPU which portray considerable improvement in the learning accuracy.

Keywords: ***Neural Networks, Back propagation, GPU.***

1 Introduction

Automatic speech recognition research community has contributed to the improvement of the accuracy and speed consistently when implemented with models Hidden Markov Model (HMM), deep learning and modern multi or manycore architectures. Multi-user speech recognition is still having the gap to get such improvement [1]. Deep Learning models have been adopted to solve many scientific problems like speech recognition, image classification which require careful handling of data representations. The term deep learning models are derived from the combination of neural networks and learning algorithms. The combinations of deep learning algorithms such as supervised learning, feed forward algorithm, back propagation, convolution neural networks, recurrent neural networks etc., improve the performance of speech recognition [2][3][4]. Preprocessing the acoustic input is an important task that needs to be handled carefully to represent the data for the improvement of the quality. Representation learning reduces the complexities that are involved in the acoustic and language modeling to attain the expected results. If the target of representation learning is well-defined, then the given system is trained successfully [5][6][7].

Proper feature learning or representation learning procedure yields very good performance in multi-user speech recognition tasks also. These procedures of deep architectures pave an enhanced way to find the gender information and its accuracy to yield considerably better result [8]. These learning procedures can also be used in noisy environment such as in-car speech recognition [9], background noise from television and radio. At present, speech recognition whether irrespective of mixed bilingual or multi-lingual is computationally intensive and hence arise the scope for parallelization also.

The stimulus for this work is predominantly driven by the recent developments in the performance of GPUs. It is also driven by the ability to make use of these powerful GPUs for conventional computing purposes. The GPUs have lot of processing cores when compared to a normal CPU. The GPUs can pull off these many cores because each core has a reduced clock speed. Nevertheless, the sheer increase in the number of cores increases the possibilities of parallelizing algorithms to achieve a very high speed up value.

The objective of this work is to implement a speech processing algorithm based on artificial neural network that runs on the graphics processing unit rather than the central processing unit. Further the performance of the algorithm is compared to the normal implementation of the algorithm to arrive at better conclusions for improving on the performance of the CUDA based implementation.

Section 2 deals with how preprocessing steps and feature extraction helps speech recognition tasks. The importance of back propagation in handling errors at the time of phoneme recognition convergence is given in section 3. The implementation details of phoneme recognition on GPU and CPU are given in the section 4. Results and performance analysis are discussed in section 5. Conclusion and future work are given in section 6.

B. Iyer, S. Nalbalwar and R.Pawade(Eds.)
ICCASP/ICMMD-2016. Advances in Intelligent Systems Research.
Vol. 137, Pp. 531-537.

2 Audio Pre-Processing & Feature Extraction

Any raw audio input data recorded in a system is subject to handle pre-processing steps such as smoothening, noise reduction etc. The pre-processing is mandatory because the file size of a single audio file is too large to give as an input to the neural network. Fig. 1 shows sample audio waveform. Even if a large enough neural network is used for processing these audio files directly, it will be completely inefficient [10]. The reason that it becomes inefficient is because, not all parts of the audio file is required for extracting the features of the speech. Moreover, the audio files tend to contain noises as well as silent segments [11][12]. Because of this pre-processing the audio file is required to remove the noise and silent segments and extract features from it so that the neural network can learn and predict the speech in a much effective manner.
It is proposed to include one more normalization step in pre-processing and analyze its impact on manycore architecture like GPU. The extracted feature is also in wave format. This feature wave is again normalized on a scale of 0 to 1. This is done so that the amount of computation done by each node is reduced drastically. The accuracy does not take a hit because of the normalization. Normalized audio data of a sample is shown in Fig. 2.
The audio file pre-processing and feature extraction contain the following steps,

- Noise reduction.
- Remove silent parts.
- Extract the values from initial audio stream and form a normalized data.
- Feed the normalized data to the neural network.

2.1 Normalizing function

```
double normalize(double value)
{
        double val=(value – minValue)/(maxValue – minValue);
        return val;
}
```

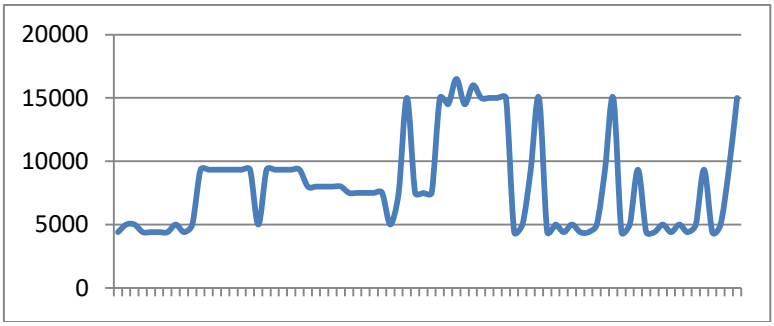

Fig. 1. Sample Audio File.

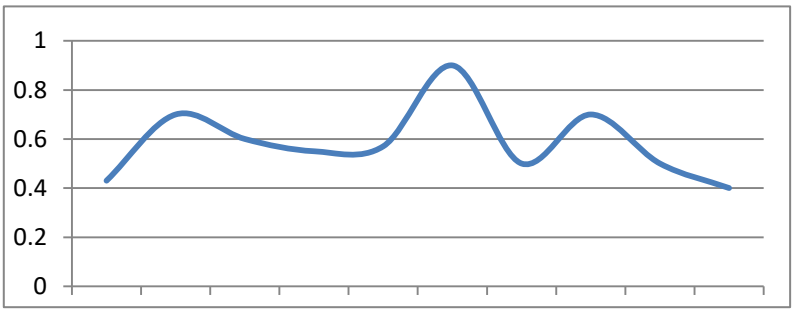

Fig. 2. Normalized Audio Data

3 Back Propagation

Typical feed forward neural network model is shown in Fig. 3 which depicts two nodes in input layer, three nodes in hidden layer and two nodes in output layer. An activation function will take an input or set of inputs, weights and produces an output at each node. The output will be feed forwarded to other nodes till the output layer. Actual output and target output will be compared and total error will be calculated.
Back propagation is the backbone of the neural network design. There are lots of ways to make a neural network to learn. The most widely used and efficient algorithm is the back propagation algorithm. The back propagation is very simple to understand and implement and at the same time very powerful. The back propagation algorithm attempts to correct the weights of each and every edge of the neural network. First the neural network is fed with the input and the corresponding output generated by it is stored. Now the neural network tries to correct itself by performing back propagation. It does that by first finding out the error between the actual output and the obtained output. Then, for each edge it finds the individual contribution towards the error. The error values can be represented in confusion matrix or error matrix.
Each edge weight is corrected by the following formula:

$$W = W - (CE \times LR)$$

In this formula, weight of each edge is given by "W", contribution to error of each edge is given by "CE", rate at which to learn is given by "LR". Usually the "LR" value is very low. This is because lower rates of learning increases accuracy, but require more iterations to complete the learning phase [13].

$$CE = \Delta \times activation\ (current\ node)$$
$$\Delta = -(Expected\ output - Actual\ output)$$

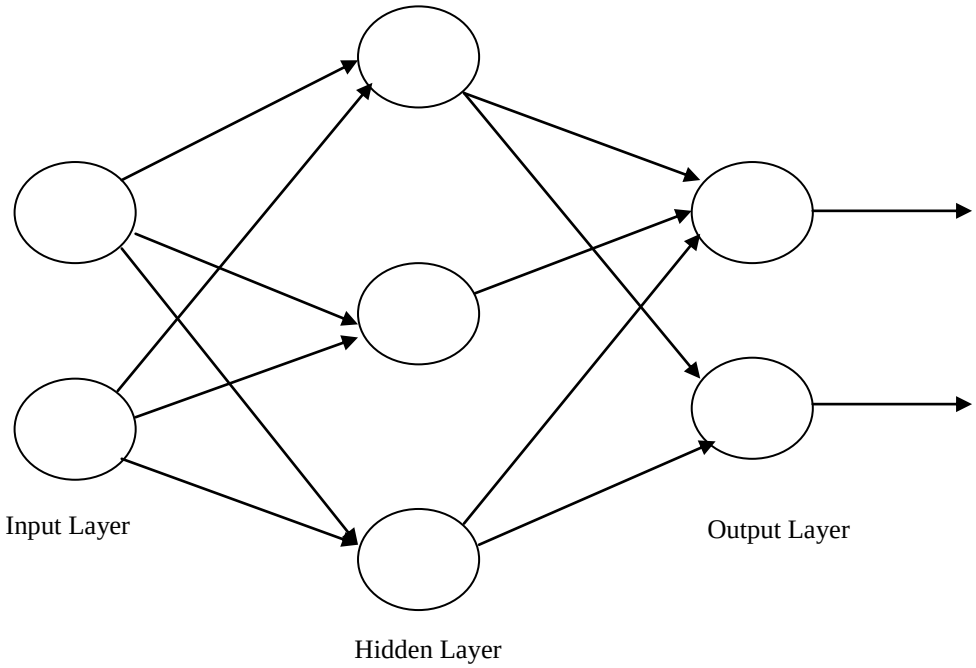

Fig. 3. Simple Artificial Neural Network Model

Higher learning rate can be achieved but it may lead to sudden increase in the learning curve. This is desirable in some cases where performance is preferred over accuracy, but if we prefer accuracy then a lower rate is the way to go.

4 Implementation of Phoneme Recognition

An Artificial Neural Network (ANN) is to be used as the learning part of the speech processing algorithms. The design of the ANN should be done initially. Factors such as number of hidden layers, number of nodes in each

layer, the activation function to be used, the initial weights of all the edges are to be considered. These factors are critical and are to be chosen properly in order to achieve the maximum efficiency.

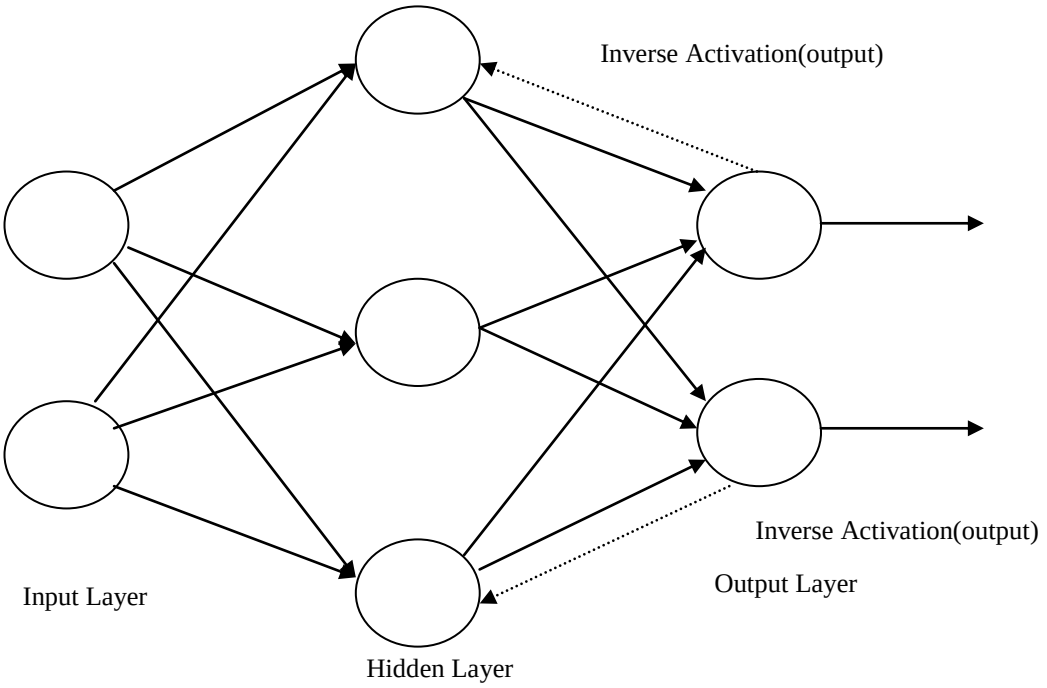

Fig 4: Back propagation with inverse activation

The design details of the neural network are as follows: input layer with 10 nodes, two hidden layers with 20 nodes each and an output layer with 44 nodes. The input layer gets the data from the pre-processed audio data. The pre-processed data is split into 10 parts so as to get a much better precision in the output. The increase in number of nodes in the first hidden layer is to increase the learning capacity of the neural network. It facilitates the neural network in mapping the input data to the correct output.The second hidden layer also serves the same purpose of the first one. It increases the efficiency as well as the correctness of the prediction.
The reason for using 44 nodes is that, there are 44 phonemes in the English language. Each node corresponds to one phoneme. The output of the network will be the probability that the user actually spoke that particular phoneme.
The forward phase of the propagation is done to learn and the output is compared with the actual output to find the error and apply the back propagation algorithm on it.
The forward phase of the algorithm is calculated for each node using the given formula:

$$output = activation\ \Sigma(weight_i \times output_i)$$

In this formula, Weight of input edge and corresponding output of previous layer is given by $weight_i$ & $output_i$.

4.1 Activation Function

The activation function is used to produce the output at that node. The activation function used in this neural network is given by,

$$activation(x) = \frac{1}{(1 + e^{-x})}$$

4.2 Inverse Activation Function

The inverse of the activation function is given by the partial derivative of the activation function. The inverse activation function is used in the back propagation phase in order to find the contribution of error for each edge. The inverse activation function is given by,

$$inverse\ (activation\ (x)) = x \times (1 - x)$$

1. Sigmoid function:

```
__device__ double sigmoid(double value)
{
        double val=1/(1+exp(-value));
        return val;
}
```

2. Activation function:

```
__device__ double activation(double value)
{
        return sigmoid(value);
}
```

3. Inverse Activation function:

```
__device__ double inverseActivation(double value)
{
        return value*(1-value);
```

The proposed work is to design and implement a specific set of speech processing algorithms over a GPU. An implementation over the CPU is also developed so as to compare both the implementations and arrive at a performance comparison between them both. The implementation will be done in C++ language with the help of CUDA libraries. Both the implementations (CPU & GPU) are to be tested on the same set of inputs to provide a sense of fairness. The details of hardware and software specifications are as follows: GPU model is NVidia GeForce GT610 and CUDA Toolkit Version 7.5 is used. Number of cores available in GT610 GPU is 48.

5 Results & Performance Analysis

Performance speed up is achieved by processing the individual nodes in each layer simultaneously using separate threads. However the layers must be processed one by one. This simultaneous execution of the threads gives an average speed up of around 10. But the figures in the table do not reflect that because of the increased overhead brought about by the transfer of data to and from the device memory and host memory.

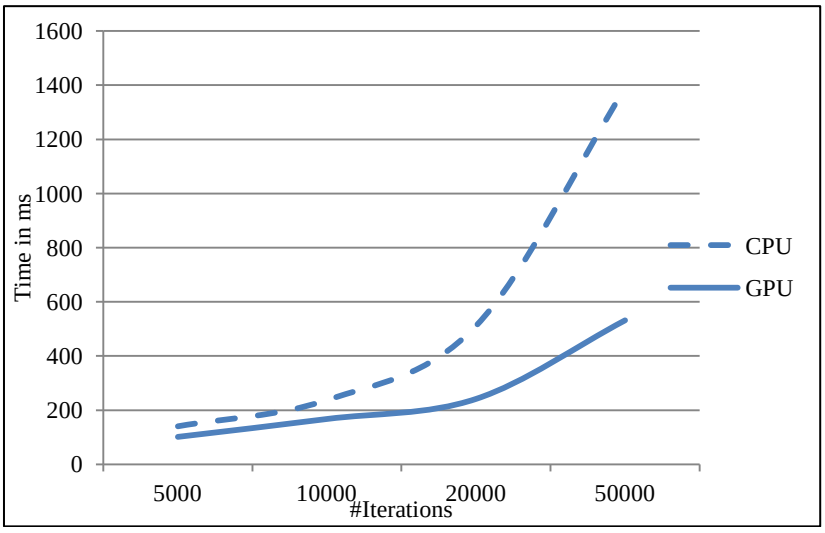

Fig. 5. Performance Comparison of GPU against CPU

Each node of each layer in the neural network is processed sequentially in a normal implementation of the algorithm. But in the case of CUDA implementation, each node of a layer is be processed simultaneously by an individual CUDA thread specifically created to process that particular node. This thread runs in parallel to other similar threads in one of the GPU cores allocated dynamically during runtime. Table 1 shows the performance analysis of CPU and GPU with increasing number of iterations of the input audio file of size 250KB

Table 1. Performance Analysis

#Iterations	5000	10000	20000	50000
File Size (250KB)	1250MB	2500MB	5000MB	12500MB
CPU Runtime(seconds)	140	237	511	1389
GPU Runtime(seconds)	102	168	242	532

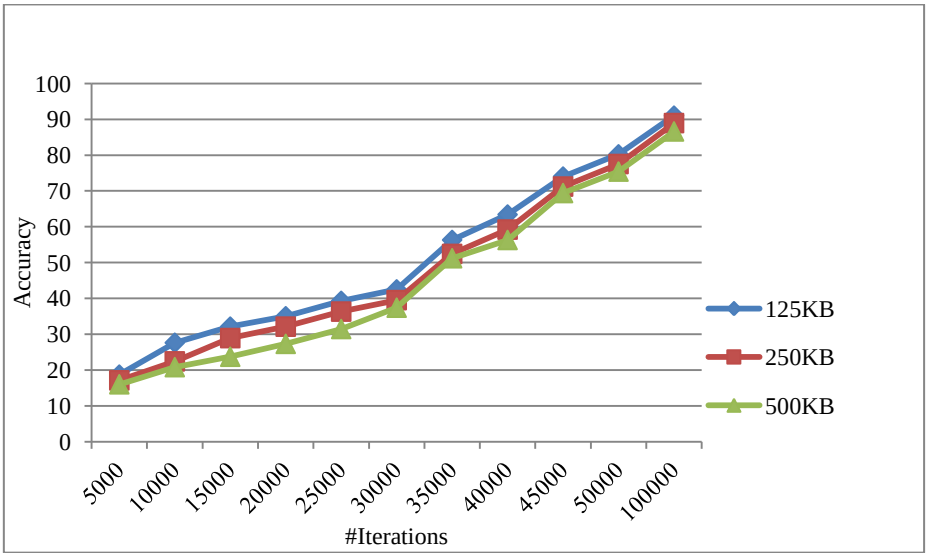

Fig. 6. Learning Accuracy

Fig. 5 shows the graphical representation of the performance of the CPU and GPU implementations. The curve of CPU takes an exponential increase as the total iterations of the algorithm increases. The reason for plotting the graph based on the number of iterations is because, with increase in the total iterations the neural network learns and predicts results accurately. So increasing the number of iterations fine tunes the efficiency of the neural network. The GPU implementation on the other hand handles the increase in number of iterations much efficiently. The graph clearly shows that the GPU implementation out performs the CPU implementation in regards of execution time. Fig. 6 shows the learning accuracy of GPU. The accuracy improves when number of iterations is increased.

6 Conclusion and Future Work

The newly introduced normalization procedure reduces the vast amount of computations that is required for phoneme recognition. Around 10x improvement in speedup and more than 90 percentage of accuracy achieved even in the basic model of GPU. The use of GPU instead of conventional CPU has its advantage in the perspective of speedup. But the model that was used being basic does not have the capability of “Unified Memory Access” which is present in newer models of GPUs in production. This feature reduces the overhead of copying to and from the GPU memory and RAM (CPU memory). This will prove to be very advantageous as more time is spent on transferring data to and from the two memories. Hence, future work will be on attempting to optimize memory access.

This alone would have a large impact on the processing time of the neural network. Another perspective to improve would be to implement the same algorithm for continuous speech. The continuous speech would be a challenge as the neural network should be re-modeled to support the extra features that are present in continuous speech but not in phoneme recognition.

References

[1] Jungsuk Kim and Ian Lane. Accelerating multi-user large vocabulary continuous speech recognition on heterogeneous cpu-gpu platforms. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 5330-5334. IEEE, 2016.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. Nature, 521(7553):436-444, 2015.

[3] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. Neural computation, 18(7):1527-1554, 2006.

[4] Wouter Gevaert, Georgi Tsenov, and Valeri Mladenov. Neural networks used for speech recognition. Journal of Automatic Control, 20(1):1-7, 2010.

[5] Abdel-rahman Mohamed, Tara N Sainath, George Dahl, Bhuvana Ramabhadran, Geo_rey E Hinton, and Michael A Picheny. Deep belief networks using discriminative features for phone recognition. In 2011 IEEE international conference on acoustics, speech and signal processing (ICASSP), pp. 5060-5063. IEEE, 2011.

[6] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine, 29(6):82-97, 2012.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence, 35(8):1798-1828, 2013.

[8] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In Advances in neural information processing systems, pp. 1096-1104, 2009.

[9] Weifeng Li, Yicong Zhou, Norman Poh, Fei Zhou, and Qingmin Liao. Feature denoising using joint sparse representation for in-car speech recognition. IEEE Signal Processing Letters, 20(7):681-684, 2013.

[10] Senaka Buthpitiya, Ian Lane, and Jike Chong. A parallel implementation of viterbi training for acoustic models using graphics processing units. In Innovative Parallel Computing (InPar), 2012, pp. 1-10. IEEE, 2012.

[11] Paul Lamere, Philip Kwok, Evandro Gouvea, Bhiksha Raj, Rita Singh, William Walker, Manfred Warmuth, and Peter Wolf. The cmu sphinx-4 speech recognition system. In IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP2003), Hong Kong, volume 1, pp. 2-5. Citeseer, 2003.

[12] David Huggins-Daines, Mohit Kumar, Arthur Chan, Alan W Black, Mosur Ravishankar, and Alexander I Rudnicky. Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In 2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings, volume 1, pp. I-I.IEEE, 2006.

[13] Shunlu Zhang, Pavan Gunupudi, and Qi-Jun Zhang. Parallel back-propagation neural network training technique using cuda on multiple gpus. In 2015 IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO), pp. 1-3. IEEE, 2015.