Experiments with Deep Neural Networks for Automatic Speech Recognition - DT2118

Akash Kumar Dhaka
Thomai Stathopoulou

June 7, 2015

Abstract

For this project we attempt to implement Automatic Speech Recognition (ASR). With the use of implemented tools for the training and application of Deep Neural Networks, we conduct a series of experiments, while evaluating and commenting on the performance of the different networks.

1 Introduction

Deep neural networks have become very popular in the field of Machine Learning. With the computational capabilities increasing more and more every day, deep neural networks become more approachable and it is more feasible to train networks with many layers and/or many nodes per layer.

The subject of this project is to experiment with deep neural networks, in order to achieve automatic speech recognition. Using a dataset of different people uttering numbers, we intend to train neural networks of varying numbers of layers and nodes in each layer, in order to be able to evaluate and compare their performances.

2 Method

The concepts of Neural Networks in general and more importantly Deep Neural Networks, are considered to be known in the scope of this report. However, follows a description summarizing the basic structure of them.

2.1 Artificial Neural Networks (ANN)

In the field of Machine Learning an ANN is a statistical learning model which is inspired by the way that a biological neural network (nervous system) processes information [1]. It imitates a biological neural network, by simulating the way that the different neurons are connected with each other intercommunicating information back and forth.

The ANNs are consisted of a number of nodes (representing the neurons). These nodes are divided into layers, such as the input layer, the output layer and one or more hidden layers in between. The nodes of every layer are connected with all nodes of the next and previous layer and communication goes both ways. The goal of an ANN is to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Every node contains a weight, which is applied to the data that is fed to this node. Through the back-forth communication, it is possible to adjust these weights, based on the correct/incorrect results that the ANN produces.

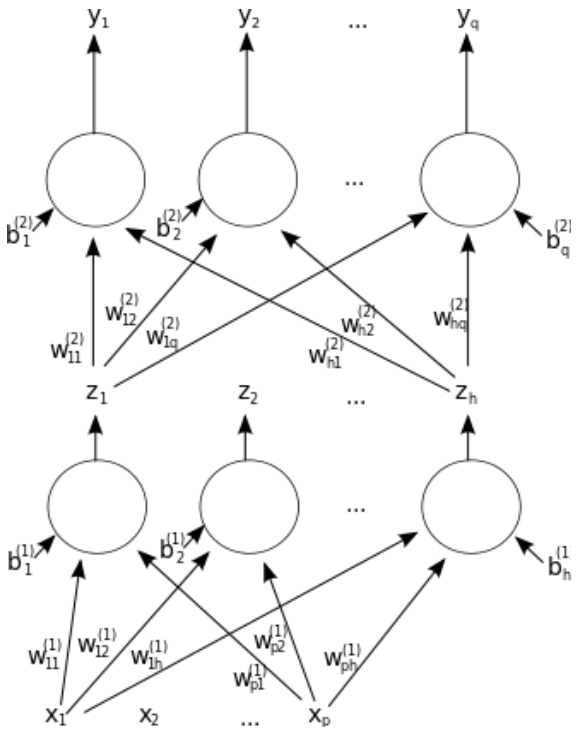

Figure 1: A two-layer feedforward artificial neural network

At the end, the output of the output layer, is usually filtered using a transfer function, so that the result of the ANN is one single value (eg. the label of the class to which the input data point is assigned).

ANNs have been very popular in different areas of Artificial Intelligence and Machine Learning, such as Computer Vision and Automatic Speech Recognition.

2.2 Deep Neural Networks

With the recent advances in deep learning field, and the emergence of GPU, deep learning approaches have produced significant improvements over previous benchmarks in different fields. Deep Learning is about learning multiple hierarchies of representations and abstractions. The difference here is that the features are not hand crafted as in the old approach of pattern recognition and classification, there is an additional step of unsupervised learning before the training which gives better feature representations. Another important difference between the deep neural networks and traditional neural networks is that they use the ReLu function in place of the sigmoid function, and this helps in avoiding the problem of vanishing gradients in the back propagation algorithm.

Table 1: The phonemes encountered in the TIDIGITS dataset and their labels

Phoneme	**Label**	**Phon.(cont.)**	**Label(cont.)**
ah	0	ow	11
ao	1	r	12
ay	2	s	13
eh	3	sil	14
ey	4	sp	15
f	5	t	16
hh	6	th	17
ih	7	uw	18
iy	8	v	19
k	9	w	20
n	10	z	21

3 Data

The data that are going to be used are taken from a large set called TIDIGITS [4]. This dataset contains recordings of different people (adults and children) uttering numbers. More specifically the dataset contains 326 speakers (111 men, 114 women, 50 boys and 51 girls). Every speaker pronounces 77 digit sequences (recordings). Additionally, the recordings of each speaker group is divided into training and testing subset.

The recordings are then split into phonemes. Table 1 shows all the possible phonemes found in the dataset, as well as their assigned labels (an integer number) for the purposes of the experiments.

3.1 Data Transformation and Feature Extraction

The TIDIGITS database is provided in the form of multiple `.wav` files. Every `.wav` corresponds to one utterance. Furthermore, after applying some preprocessing functions from the Hidden Markov Model Toolkit (HTK) [6], we are provided with a new file (`.mfc`), which displays the contents of all the recordings (shown in Figure 2). This file basically expresses when exactly each phonemes starts and ends within a recording (time measured in milliseconds).

Using the `.mfc` files it is possible to produce MFCC features for the different phonemes. Depending on the length of the frames extracted from the recordings, varying numbers of features are produced for every phoneme. For example for a frame length of 10ms, for the `.mfc` of Figure 2 we will get 17 features for the "sil" phoneme, 7 features for "hh", 8 features for "w" etc. These features are written in a text file, where every line corresponds to one frame and contains the feature elements followed by the label assigned to the phoneme.

The final step is to convert the text file into a format that is accepted by PDNN. That is, either a `.pfile` or a pickle file. With the use of a simple Python script, the text files are converted into pickle files.

For the experiments conducted for this project, only the utterances produced by men and women were used. The training set contains ∼1,500,000 utterances (of which 70% is used for the actual training and 30% is used for validating). For the testing phase, even though the original dataset contained the same amount of utterances as the training set, it seemed for appropriate to use only 30% of that data, resulting in ∼500,000 utterances. This was mostly decided because of the computational complexity, which would result in much slower experiments.

"path/to/recording/recording.wav"
0 1700000 sil
1700000 2400000 hh
2400000 3200000 w
3200000 4500000 ah
4500000 4800000 n
4800000 4800000 sp
4800000 5700000 s
5700000 6400000 eh
6400000 7800000 v
7800000 8700000 ah
8700000 9100000 n

Figure 2: Example of the contents of a recording

4 Experiments

For the training and testing of the deep neural networks, we used two implemented toolkits, Kaldi and PDNN [5]. These are toolkits implemented for use in speech recognition tasks (Kaldi) and deep neural networks in particular (PDNN). We conducted a series of experiments by varying the network parameters such as number of nodes in the hidden layers, number of hidden layers and learning rate.

We used the DNN implementation for neural networks in the aforementioned PDNN library. The library is written in python, currently in active development and built on top of Theano [2] [3]. Since these libraries are still in development there have been some minor scripting issues in some of the shell files, which were confronted individually.

The type of features used for the experiments is MFCC including the zeroth coefficient. Therefore all the networks that were trained had a first layer

with 13 nodes and a final layer of 22 nodes (the number of phonemes).

Finally after training and evaluating the performance of several networks, using the data produced as described in Section 3.1, the data was then normalized in order to be used for the training of new networks.

When training each network, the data is assume to be independent and identically distributed based on a Gaussian distribution $\mathcal{N}(0, 1)$. Therefore, in order to produce the normalized dataset, we calculated the mean and standard deviation of all input vectors. The normalized vectors are calculated:

$$X_i = \frac{X_i - \mu_i}{\sigma_i} \tag{1}$$

5 Results

Figures 3, 4, 5 and 6, show the training and validation error over 100 epochs of the networks trained with non-normalized and normalized data. The number of hidden layers used was 2, 3 and 4 with 128, 256 and 512 nodes per layer. A learning rate of 0.3 was used for most of the networks, while for the normalized data there were some networks that were trained with a learning rate of 0.2.

All networks display a similar behavior with very close to each other errors. All training errors are decreasing throughout the entire training process. However, we observed from the validation and testing errors (Table 2) that the phenomenon of over-fitting is very prominent in these experiments.

When using the normalized data, one important observation is that the graphs for the training and validation errors a smoother than the ones for non-normalized data, especially for the validation errors. However, this makes the over-fitting much more visible. For example, the network $512 \times 512 \times 512$ for normalized data produces the lowest training error of 15%. At the same time it also produces the highest validation error of 35% and the over-fitting for this network is the most intense. Finally the testing error of this network is 41.61%, which is quite far from the lowest error 41.06%, which was produced by the 512×512 network.

The normalized data have in general a better performance, which can be verified by Table 2. Just the use of the normalized data improves the performance $2 - 7\%$.

6 Conclusions & Discussion

As already mentioned in Section 5, the use of normalized data improves the performance of the networks. However, the general performance of the networks was very similar for all different trainings and no particular patterns emerged, concerning e.g. number of layers or number of nodes per layer.

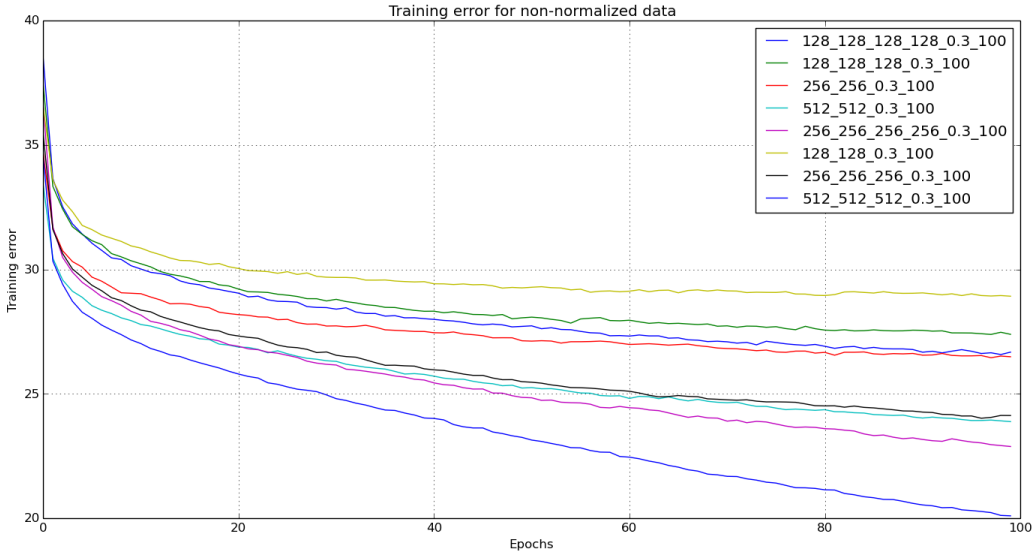

Figure 3: Training error over 100 epochs of the non-normalized data

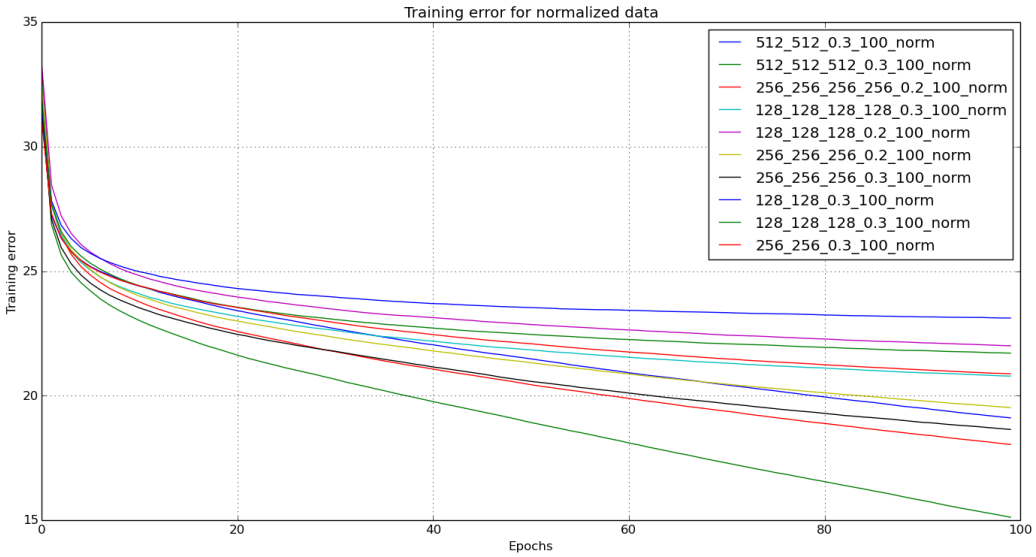

Figure 4: Training error over 100 epochs of the normalized data

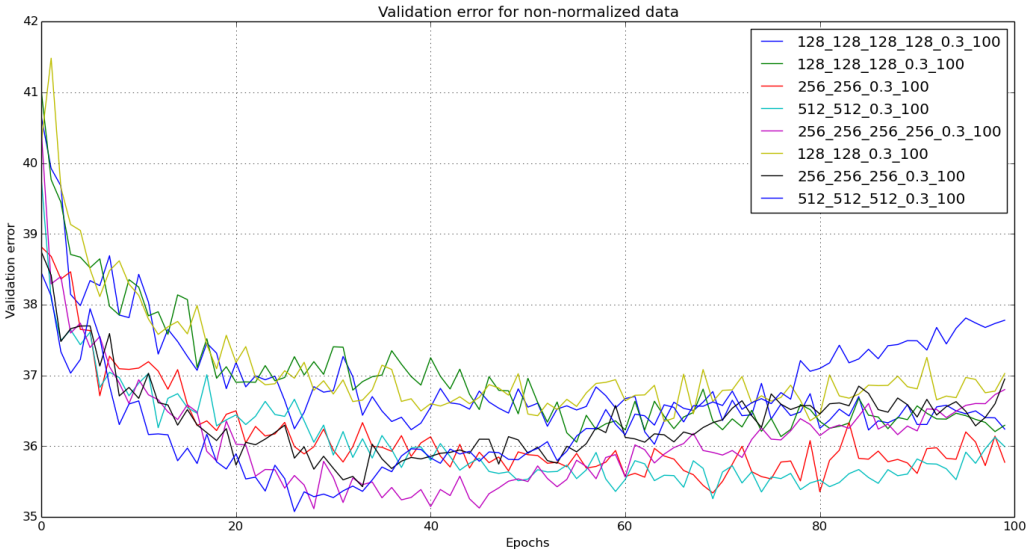

Figure 5: Validation error over 100 epochs of the non-normalized data

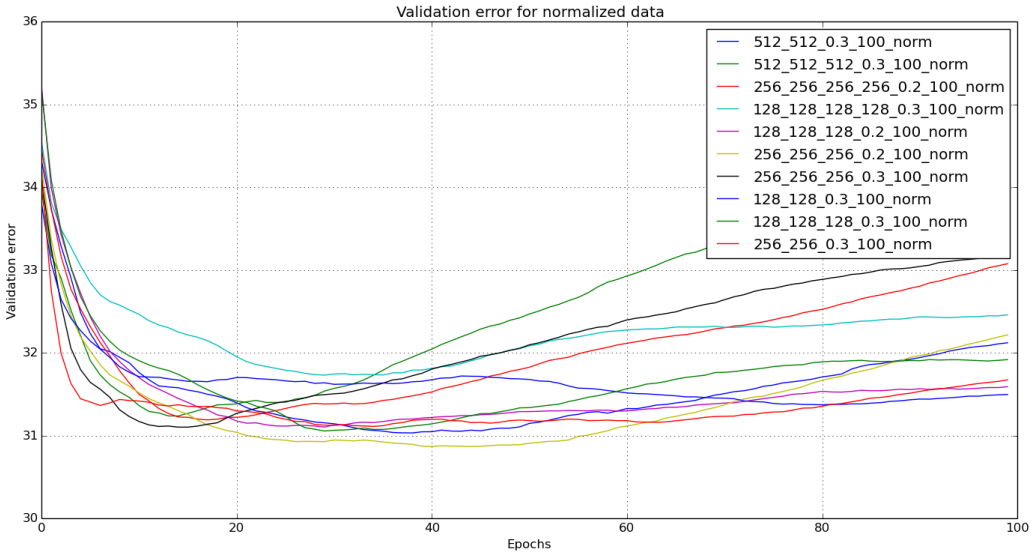

Figure 6: Validation error over 100 epochs of the normalized data

Table 2: Testing errors of different networks

Layers	Nodes	Learning Rate	Normalized	Error
2	256	0.3	No	44.49%
2	512	0.3	No	45.74%
3	128	0.3	No	47.97%
3	256	0.3	No	45.87%
3	512	0.3	No	46.81%
4	128	0.3	No	46.64%
4	256	0.3	No	48.16%
2	128	0.3	Yes	41.17%
2	256	0.3	Yes	41.28%
2	512	0.3	Yes	41.06%
3	128	0.3	Yes	42.69%
3	267	0.3	Yes	42.49%
3	512	0.3	Yes	41.61%
4	128	0.3	Yes	42.51%
3	128	0.2	Yes	41.4%
3	256	0.2	Yes	41.87%
4	256	0.2	Yes	42.06%

It is our belief that, in order to be able to see significant changes, one needs to train much larger networks (6 layers with 2048 nodes is one example). However, due to time and computational power constraints, it was not possible to train networks of this size in the scope of the project.

There are many different approaches to this task, some of which are mentioned in the following paragraphs.

Training of large networks As already mentioned it is important, given the necessary resources, to train very large networks, in order to see improvement, or at least to be able to draw conclusions about this task. Furthermore these network need not have the same number of nodes in all layers, but one can experiment with different combinations. This is basically due to the complex nature and size of the data.

Detect over-fitting It is already commented in Section 5, that there is very intense over-fitting of the networks. Even though the exact number of epochs, after which the networks start to over-fit, has not been determined and is probably affected by the specifications and size of the network, it can be possible, to detect the moment that the networks start to over-fit (by observing the validation error) and stop the training, in order to keep the best network possible.

Preprocess data The observation that the use of normalized data improves the performance of the network is an indication that the form of the used data is always important in such experiments. There are many different features that can be used, as well as different methods of pre-processing the data. The features and normalization used for this project is just an indication and there are many more paths to explored yet.

Convolutional networks Finally it is very interesting to try and complete this classification using convolutional networks with varying number of convolutional layers of different filters.

References

[1] Various Authors. Artificial neural network. `http://en.wikipedia.org/wiki/Artificial_neural_network`. [Online; accessed 7-June-2015].

[2] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[3] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[4] R. Gary Leonard and George Doddington. Tidigits ldc93s10. *Philadelphia: Linguistic Data Consortium*, 1993.

[5] Yajie Miao. Kaldi+pdnn: Building dnn-based ASR systems with kaldi and PDNN. *CoRR*, abs/1401.6984, 2014.

[6] S.J. Young and Sj Young. The htk hidden markov model toolkit: Design and philosophy. *Entropic Cambridge Research Laboratory, Ltd*, 2:2–44, 1994.