A Project Report

On

Deep Learning Architectures for Speech Recognition: A Comparative Study

BY

Shikhar Bharadwaj

2014A7PS0113H

Under the supervision of

Dr. Aruna Malapati

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

CS F376: DESIGN ORIENTED PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(NOVEMBER 2017)

Acknowledgements

Foremost, I extend my deepest gratitude towards Prof. G. Sundar, the Director of BITS Pilani Hyderabad Campus for giving me the opportunity to receive education in his prestigious institute.

I am highly indebted to Dr. Aruna Malapati for her guidance, motivation, enthusiasm and immense support. I thank her for the guidance she gave me in our weekly discussions about the direction of the project and for providing me an opportunity to work with her in her busy schedule.

I am thankful to BITS infrastructure for providing me necessary resources.

Lastly, I would like to extend my thanks towards my family who were always there to help and give their best suggestions.

-Shikhar Bharadwaj

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "**Deep Learning Architectures for Speech Recognition: A Comparative Study**" submitted by Mr. Shikhar Bharadwaj (ID No. 2014A7PS0113H) in partial fulfillment of the requirements of the course CS F376, Study Oriented Project Course, embodies the work done by him under my supervision and guidance.

Date: 27-11-17 **(Dr. Aruna Malapati)**

BITS- Pilani, Hyderabad Campus

ABSTRACT

Speech Recognition has traditionally been handled using elaborate statistical models such as GMM-HMM. With the increase in computation power and deep learning, new deep learning architectures have been proposed to perform end to end speech recognition. This report explains the classical pipeline for deep learning contrasting it with the more modern end to end deep learning and hybrid architectures. The report also tries to explain the primitives used in deep learning architectures like Recurrent Neural Nets , Long Short Term Memory and the training algorithms. Then, the report explains an implementation of the Language Model and the results obtained from it. Following this is the description of Connectionist Temporal Classification model and its implementation. Two models for the simpler task of Keyword Recognition are also considered at the end.

CONTENTS

Title page…………………………………………………………….1

Acknowledgements…………………………………………………..2

Certificate…………………………………………………………....3

Abstract……………………………………………………………....4

1. Introduction….……………………………………….…………6

2.Recurrent Neural Networks……………………………………..….9

3.Language Model……………………………………………13

4.Results…………………………………………………………....15

5.Connectionist Temporal Classification….………………………16

6.Implementation and Results………………………………………...18

7.Keyword Recognition……………………………………………….20

References……………………………………………………………..23

Introduction

Automatic Speech Recogntion (ASR) has been an active area of research since 1960s. It has been an important technology for computer-human interaction.This project focusses on the speech transcription problem which, defined simply , is getting text from spoken words. In the past most of the research was focussed on breaking the speech recognition problem into subtasks and developing models to handle these subtasks. The results from these tasks were finally combined to get the transcribed text. This method, though state of art today, wasn't much developed back in the 60s when keyboard and mouse based input methods significantly outperformed speech in efficiency and accuracy. But with the growth in computing power, new deep learning models have been proposed that are relatively simple to implement. At the same time these models provide comparable, if not better, accuracy to the classical speech pipeline.

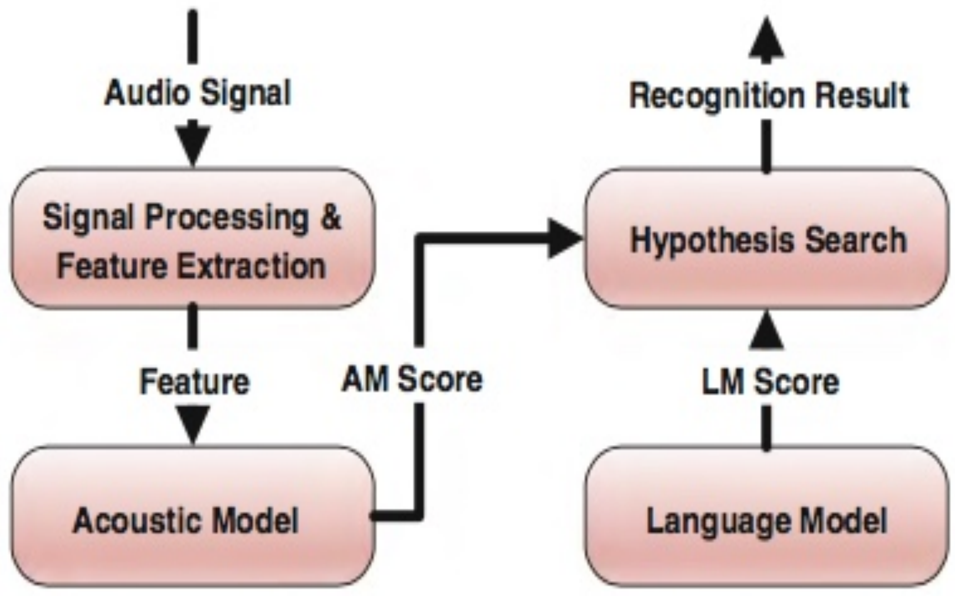
The advent of deep learning has been due to Moor's law. The computational power of multi core processors, GPUs and computer clusters has enabled the training of powerful models. Secondly, the large amount of data available today has helped researchers to do away with the practical but limiting assumptions needed for the models to work. This makes the current systems more robust. For example, the early speech recognition research was focussed on keyword detection because of the paucity of data. Currently large vocabulary speech recognition systems have also been shown to work with good results. A surge in Speech Recognition has also resulted from the increase in popularity of mobile and wearable devices that render keyboard and mouse based input methods obsolete. Speech being the natural mode of communication is thus an important area to study.

Basic Architecture of ASR Systems

An automatic speech recognition system has four main components:-

- Signal Processing and feature extraction
- Acoustic Model (AM)
- Language Model (LM)

- Hypothesis Search



Architecture of ASR systems

The signal processing and feature extraction module takes audio signal and enhances it by reducing noise and channel distortions. Features are then extracted from the signal by converting it from time domain to frequency domain. The feature vector that are suitable for deep learning are MFCC (Mel frequency ceptral coefficients)[1].

Acoustic model uses knowledge of acoustics and phonetics to give an AM score for the feature sequence from previous module. The main issue in acoustic modelling is handling variable length feature vectors. This arises from the fact that speech is variable in length, thus generating variable length vectors to be decoded. To address such problems techniques such as Dynamic Time Warping and Hidden Markov Models are used.

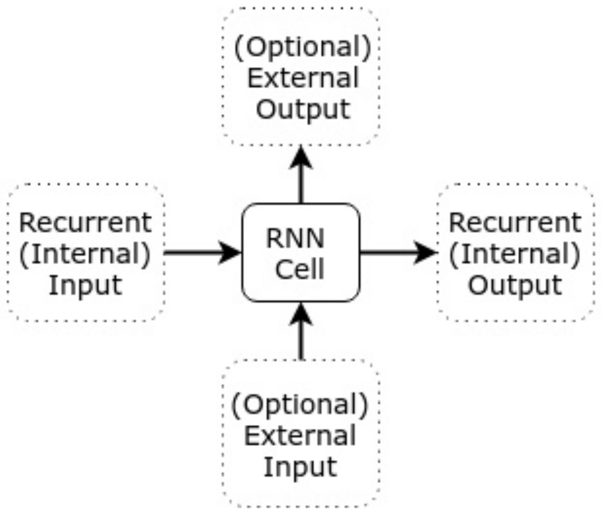
Language Model estimates the probability of hypothesized decoded sequence. This estimate is presented as an LM score that is combined with AM score to do a hypothesis search. The Language Model is comparatively easier to train because it just has to learn the correlation between words for which textual data is enough. Prior knowledge about the domain or task where speech recognition system would be used is important for increasing accuracy of the system.

The hypothesis search combines AM and LM score given the feature vector sequence and hypothesized decoded sequence. The sequence with highest score is generated as a result of the recognition process.

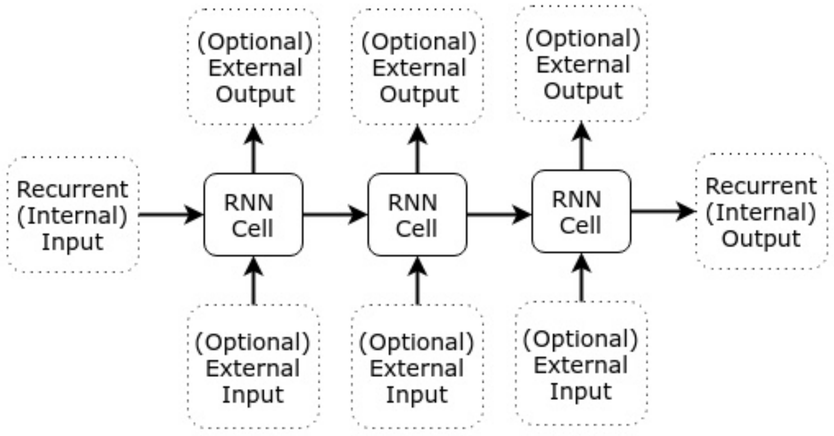
Recurrent Neural Network

RNN is a special type of neural net in which the output from a layer at a previous timestep is fed again into the layer as an input for the next timestep. The intuition for RNN arises from the fact that neural networks are good at discrimination based on the current data(one shot task) but lack memory. To incorporate memory into the network structure the hidden layers' ouputs are fed back as input. This particularly helps with time sequence data like speech. The network is able to understand a context for the current data that increases its accuracy.

An RNN is a compostion of identical feedforward neural networks, one for each step in time, referred to as 'RNN Cells'. These cells operate on their own output, allowing them to be composed. They can also operate on external input and produce external output. Here is a diagram of a single RNN cell:


(Optional) External Output
Recurrent (Internal) Input
RNN Cell
Recurrent (Internal) Output
(Optional) External Input

Here is a diagram of three RNN cells:


(Optional) External Output
(Optional) External Output
(Optional) External Output
Recurrent (Internal) Input
RNN Cell
RNN Cell
RNN Cell
Recurrent (Internal) Output
(Optional) External Input
(Optional) External Input
(Optional) External Input

The recurrent outputs can be thought of as “State” that is passed on to the next timestep. Thus an RNN cell accepts prioir state and a current input to produce the current state. This current state can be optionally further processed by applying a function to get the current output. Here is the algebraic description of RNN cell:

$$\begin{pmatrix} s_t \\ o_t \end{pmatrix} = f \begin{pmatrix} s_{t-1} \\ x_t \end{pmatrix}$$

where:

- s_t and s_{t-1} are our current and prior states,
- o_t is our (possibly empty) current output,
- x_t is our (possibly empty) current input, and
- f is our recurrent function.

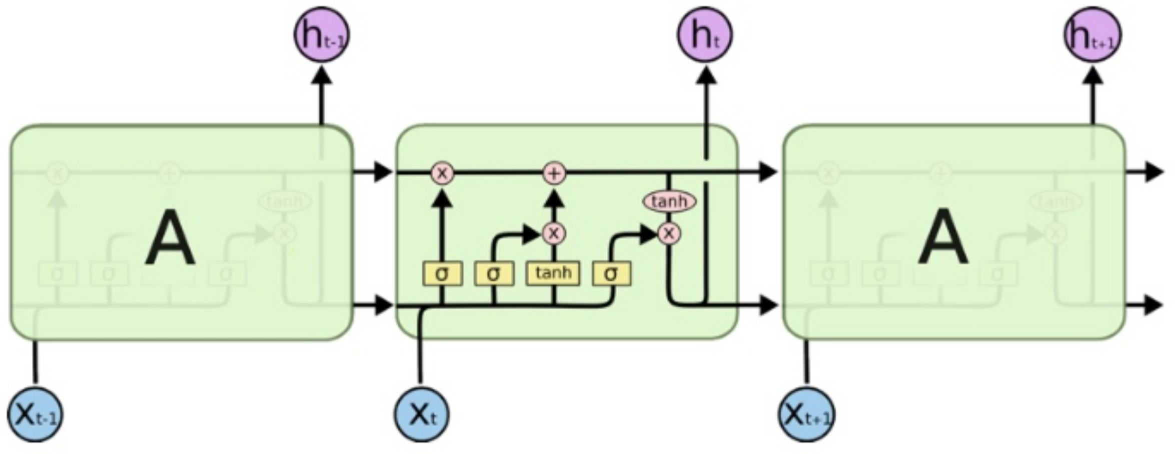
Instead of creating a separate cell for each timestep as in the previous diagram the process can be carried in place by looping the current state back as an input. In the diagram above only one of the cells was used in each timestep but now the same cell will be used for each timesteps, albeit with a new input/state each time. The main consideration while using RNNs is designing the RNN in terms of inputs and outputs. Put more simply, defining what the timesteps should be. For example consider the problem of translation, say from English to French. One option is to consider the whole sentence in a timestep. This leads to the reduction of RNN to a simple feedforward network. This is so because there is no need for an intermediate state when translating a single sentence at once. Hence even the final state does not matter. For the state to matter, there must be more data that follows and uses the context gained from the translation done at previous timesteps. For example, if a paragraph has to be translated it might use the context from previous sentences. This context is gained from the ‘state’ that we have talked about previously.

Problems with Vanilla RNN

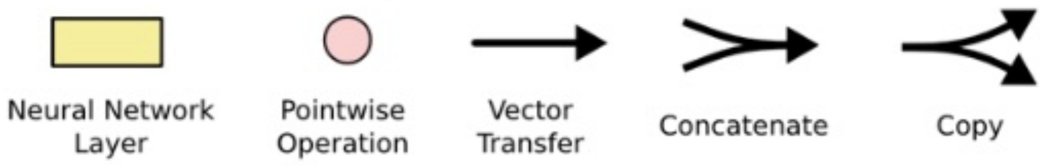
Consider a RNN network that predicts the next term of a sentence. For ex, “I grew up in France….I speak fluent *French*”.Recent information suggests that the next word has to be the name of a language but if we want to know which language we have to get the context from further back. In theory, RNN are expected to learn such long term dependencies but in practice RNN don’t seem to be able to learn them. The problem was explored in depth by Hochreiter(1991) and Bengii, et al. (1994) , who found some fundamental reasons why it might be difficult.

LSTM Networks

As a solution to this problem LSTM Networks were proposed by Hochreiter and Schmidhuber (1997). Long Short Term Memory networks are designed to avoid the long term dependency problem. Remembering information for long periods of time is practically their default behaviour. LSTM, like RNN, have a chain like structure when expanded in time. The following figure shows the structure of LSTM Cell.



The repeating module in an LSTM contains four interacting layers.



The key to LSTM is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

An LSTM has three of these gates, to protect and control the cell state. The gates are defined by the following equations:

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma\left(W_o \ [\, h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Here C is the cell state that is calculated by multiplying f (forget gate value) with the previous state and i(input gate value) with new state. What this basically means is that the LSTM decides how much of the previous information it has to keep and how much new information it must add on. This combined with the optimisation algorithm helps the model to decide which information is needed for prediction. Hence it is able to learn generalisations for predicting the next word. In the next section I explain an application of LSTM to perform Language Modelling on a character level.

Language Model

A language model enables one to predict the probability of observing a sentence as :

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1})$$

Where w(i) is ith word in the sentence. The language model implemented as an experiment was a character level model. Characters instead of words were focussed on because lesser number of characters allowed better training times than a word vocabulary would have. Also, the changes in the results produced by the model with number of iterations resulted in a better understanding of the LSTM training process.

Problem Statement

The problem statement was to generate a probability distribution for the next character given a sequence of characters. More formally, each term on right in the equation given above. After generating this distribution a character stream of Shakespearean text was sought by repeating the process of generating distribution with new characters.

Training Data and Preprocesssing

The training data for the task was obtained from Kaggle contest. The data consisted of text of all the works of Shakespeare. All the upper case characters were ignored to lower the number of total characters. After this preprocessing a total of 52 characters remained. These included punctuations and space. No other preprocessing was employed to train the model.

Network Architecture

The model was implemented using Tensorflow. An embedding matrix was created that represented each of the 52 characters as real vectors in 100 dimensions. This matrix was learnt by the model while training. The character vector was provided to three layers of LSTM network where each LSTM cell had a state size of 100. The batch size was chosen to be 32. From the final layer a softmax layer was applied. This turned LSTM state into a vector of 52 real values which represented a probability distribution over the characters. The loss for this network was cross entropy. It was trained using Adam Optimizer.

Training and Sampling

For generating Shakespearean text, a random character was given as input to the Language Model. The model then generated a probability distribution for the next character. Based on a sampling method one character was chosen from this distribution and given back to the model as input. The process was repeated to generate a stream of text.

For training the model the data was broken into n length character groups where n was specified by the user. For the ith iteration i till i+n characters were provided as input to the model. The expected output was i+1 to i+n+1 characters.Essentially what this means is that for each character in the input the model expects next character from the text. With the definition of LSTM that we saw before this was an easy task for the network since it can remember the information from far back and the information it had to predict the next character was an amalgamation of previous information along with the current input character. The network was trained for 10k, 20k and 100k iterations and the results obtained.

Training time

The model took approximately 1 hour for running 20k iterations on a 16 core Xeon processor with 32GB RAM. Running 10k iterations took 2 hours on an i5 processor with 4GB RAM.

Results

The model trained for 5k iterations produced no understandable text. The only observation from the model was that the random text it threw appeared to be well segmented into word length sequences of three to seven character lengths. So, what it learnt in first 5k iterations was to put spaces to segment random text into words. The model trained on 10k iterations also produced disheartening results. It repeated certain high frequency characters like ‘i’, ’e’, ‘t’, ‘a’ etc. This was probably due to the fact that the model saw these characters max number of times in 10k iterations and could only learn how they were placed, ie the correlation between the spacing of these characters.

When trained on 20k iterations the model started giving interesting results. Here is an example of the text the model produced-

```
“at i falge tot mut in the fon me toure fount tort anlitherte and tour
than wathitt tath angill tour then thens, to an me whut hes to thetter,
the seasens the fou at thate he tore for whore to me fore ine that thorand
ind thou af wanless wore hallt, wore the sand, ferang thith one thut, as
and and a millen wher, wour to that in wouste the that hont ast in menther
tor, the sor woun hot to merthe hind to mire is tou and tors”
```

As can be seen in this text, the model learnt to recognize small words like ‘at’, ‘in’, ‘the’, ‘me’, ‘to’ and certain high frequency words like ‘that’. Although the rest of the text is gibberish. The results improved upon training for 100k iterations-

```
“me thinksheft to the word and mark of your good the warth this soul stay
me, to the chowing of your shall in the seef a clast a shilisor,well as
marrow thoush how, which shility would and the serve to the propony serve
your stain and hell they but our had speak all their strace our state and
heng, which to my lood they the sown a my lord”
```

In the text above, the model has learnt many new words. This has lead to lesser number of misspellings than the previous version. The words don’t make any sense when seen in context but the model finally outputs proper words instead of random characters. This proves that with enough data and training time the model can learn the correlations between the various words in the text just as it has learnt the correlation between the characters by increasing the number of iterations from 20k to 100k.

As a conclusion the model has successfully learnt syntax, probably in the first 15-20k iterations. As the number of iterations were increased it had more time to focus on the less frequent characters and thus was better able to model the language.

Connectionist Temporal Classification

Historically, the problem of Speech Recognition has been tackled using Hidden Markov Models. The HMM produce good results but have several problems-

- They require a significant amount of task specific knowledge. For ex. For designing the state model for HMMs.
- They require implicit independence assumptions to make decoding and training tractable.
- For standard HMM, training is generative, even though sequence labelling is discriminative.

The first approach to use Neural Nets in Speech Recognition was mainly focussed on developing an ANN-HMM hybrid system (Bourlard & Morgan, 1994; Bengio., 1999) that relied on HMM for segmentation of input audio into phoneme frames and hence its accuracy was limited by the accuracy of HMM. The neural nets were used to provide localised classification and HMMs to model the long range sequential nature of data. Such methods, in addition to being prone to error due to HMM segmentation, do not fully utilise the potential of Recurrent Neural Nets.

Connectionist Temporal Classification

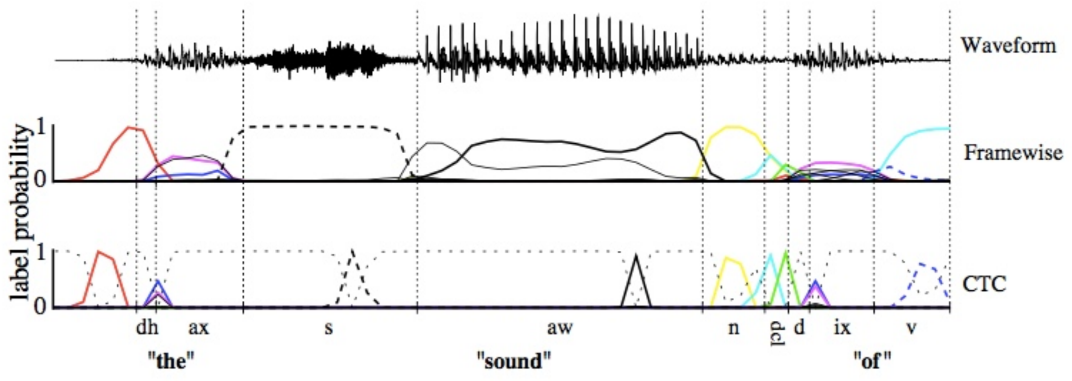
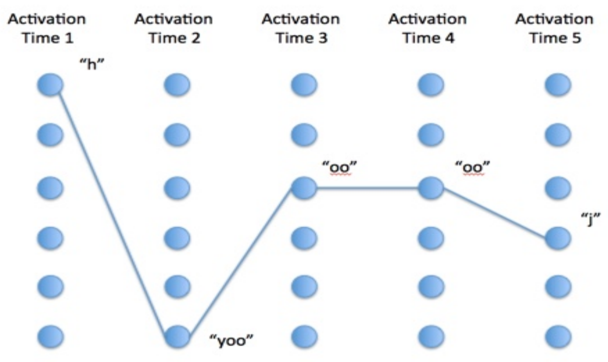


Figure 1. **Framewise and CTC networks classifying a speech signal.** The shaded lines are the output activations, corresponding to the probabilities of observing phonemes at particular times. The CTC network predicts only the sequence of phonemes (typically as a series of spikes, separated by 'blanks', or null predictions), while the framewise network attempts to align them with the manual segmentation (vertical lines). The framewise network receives an error for misaligning the segment boundaries, even if it predicts the correct phoneme (e.g. 'dh'). When one phoneme always occurs beside another (e.g. the closure 'dcl' with the stop 'd'), CTC tends to predict them together in a double spike. The choice of labelling can be read directly from the CTC outputs (follow the spikes), whereas the predictions of the framewise network must be post-processed before use.

In [5] Alex Graves introduced a new loss function that enables the use of Recurrent Neural Networks and its modifications for recognizing speech. The method designed by Alex Graves allows sequence to sequence conversion of unsegmented data. The basic idea is that the RNN outputs are a probability distribution over various phonemes. An objective function is then defined that maximises the probability of correct labellings given the input sequence. Since this objective function is differentiable, Backpropagation Through Time can be used to train the network.

The network outputs the probability distribution over the labels at each timestep and a path through the outputs that has the maximum probability is considered the final encoding of given speech.



What makes this method interesting is that it accounts for the rate of speech(dynamic time warping). The paper proposes that in addition to the labellings an additional blank is required for accounting the silences and speed of speech. It then defines a many to one mapping *B* that removes all duplicate labellings and blanks along the path. Since the probability is computed with the final labellings and hence the mapping helps with dynamic time warping.

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}).$$

where π is the path of labellings.

Since the computation of this equation requires exponentially many computations of paths, a recursive dynamic programming based computation method has been proposed that is similar to the forward-backward algorithm of Hidden Markov Model. The proposed method computes the log likelihood of correct labelling from the network's ouputs which has to be maximised.

For decoding purposes, in the implementation described in the next section we have used a beam search algorithm that maintains a fixed number of top local best paths and works out the global best path from them. Another approach would be to use a greedy algorithm that considers only the label with maximum probability at each time step.

Implementation and Results

Architecture

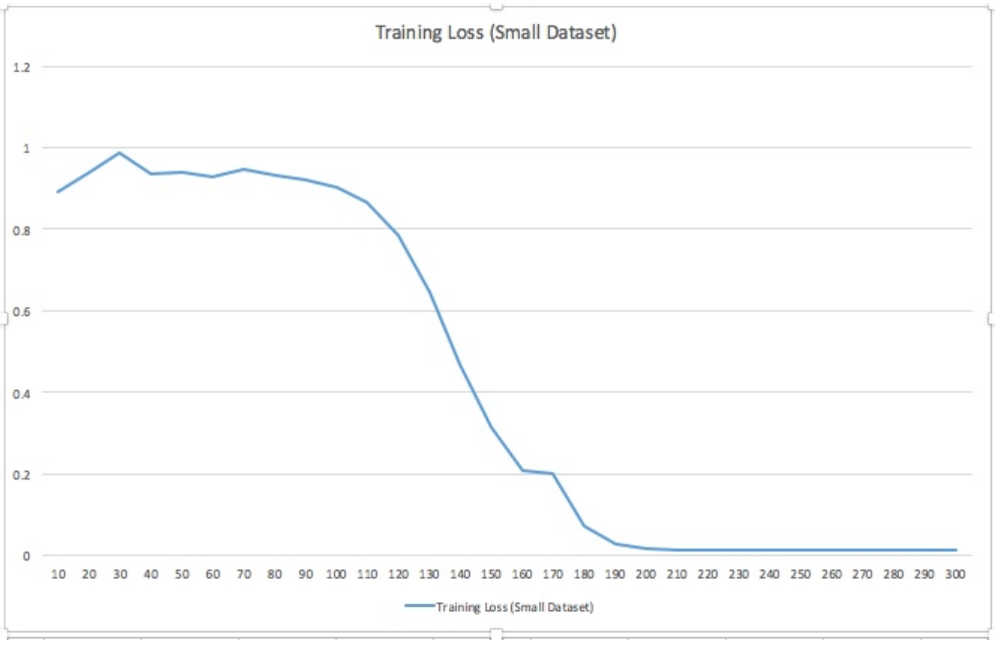
A Bidirectional Recurrent Neural Network with LSTM cells in both direction was used. The input to the network consists of 13 features, namely the 12 Mel Frequency Ceptral Coefficients and total energy in the audio signal.The B-RNN's outputs then pass through a hidden layer followed by a softmax layer to generate probabilities over labels. 27 labels were chosen, 26 of them representing capital English letters and a space. An extra label is used for the "blank". Hence there are 28 outputs of the hidden layer.The loss function is as described in the previous section- the CTC loss. The optimizer takes into account momentum and learning rate of 0.9 and 0.001 respectively.

Data

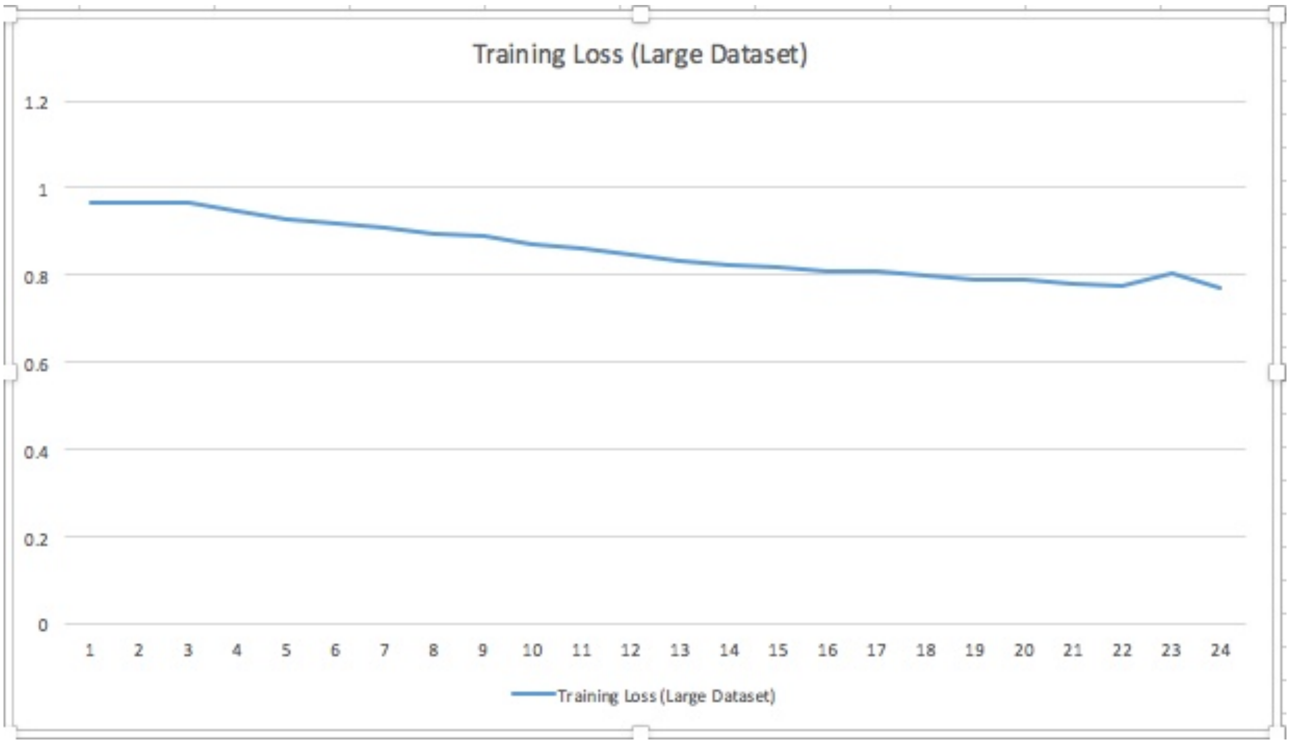
The model was trained on two dataset. The smaller dataset was provided by Tensorflow library and consists of a mere 300KB of audio. The bigger dataset was obtained from LibriSpeech [6]. This consists of audio book recordings of 370MB.

Results

The training loss for the small dataset is shown in the figure:



It took nearly 5 minutes for the model to train on the small dataset. When trained on the larger dataset, the model yielded only a little decay in loss. The graph for that is:



The model produced the above result by running continuously for 7 hours on an i5 processor. The graph above shows only a slight reduction in training loss with each epoch because this is only a part of the complete graph. A larger dataset has more variations that must be captured by the network. This would require more training time and hence larger number of iterations than shown in the graph above.

Keyword Recognition

Task

Keyword Recognition is the task of understanding simple spoken words from a given set. This task is more feasible than the previous task of transcripting the audio. Given an audio signal, the word spoken in it can be from a set of 10 classes or unknown. The task involves predicting the correct class of the word. The set of classes are `yes`, `no`, `up`, `down`, `left`, `right`, `on`, `off`, `stop` and `go.`Anything else has to be predicted as unknown.

Dataset

The dataset was obtained from [7]. The training data consists of 25,000+ audio files in wav format that contains a mixture of all 10 classes and 2300 files that contain random spoken words not found in the test set. Each file is about 1 second long. Test set consists of 150,000+ files whose class has to be predicted.

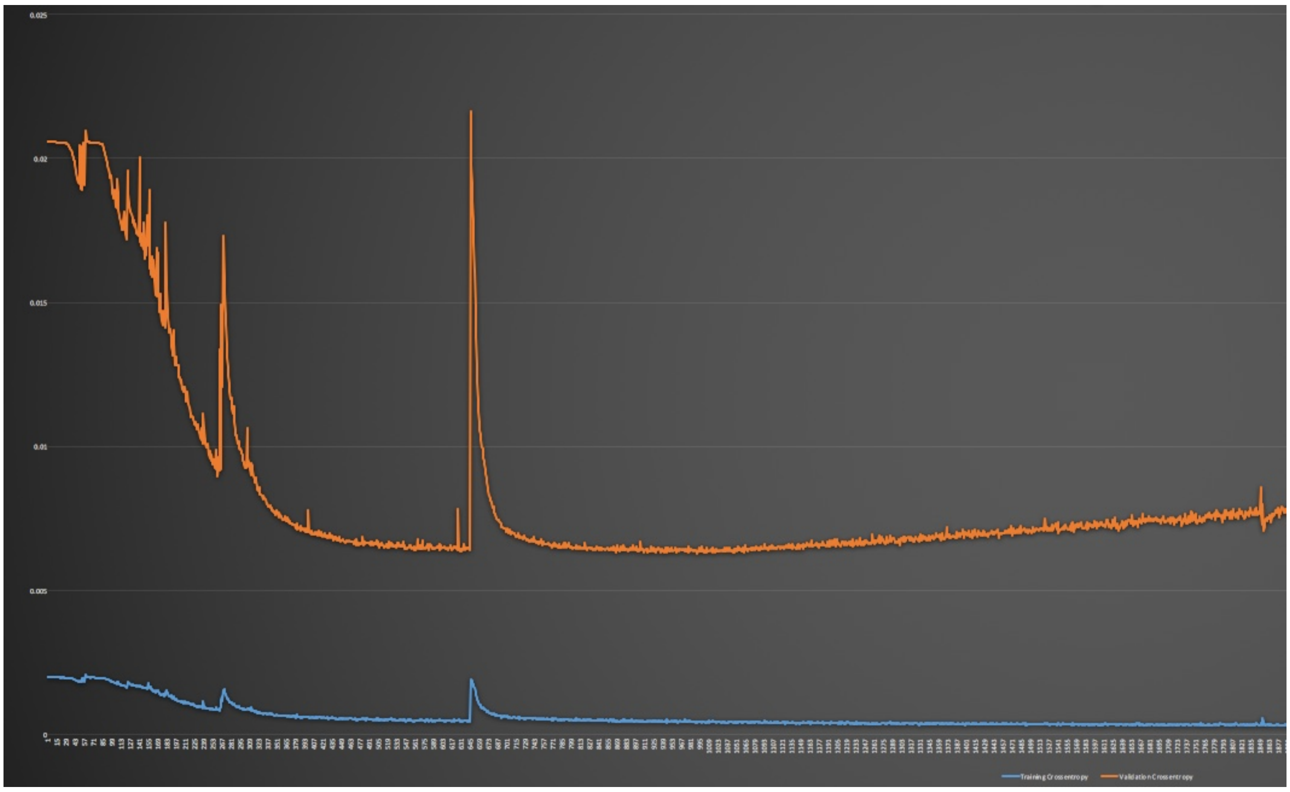
Preprocessing

The files were preprocessed to obtain 13 features. 12 MFCC coefficients and 1 total energy in the signal. These values were then normalized. MFCC coefficients were found with window length of 25 milliseconds and window step of 10 milliseconds. Thus a numpy array of (13,99) was obtained to represent each audio file.

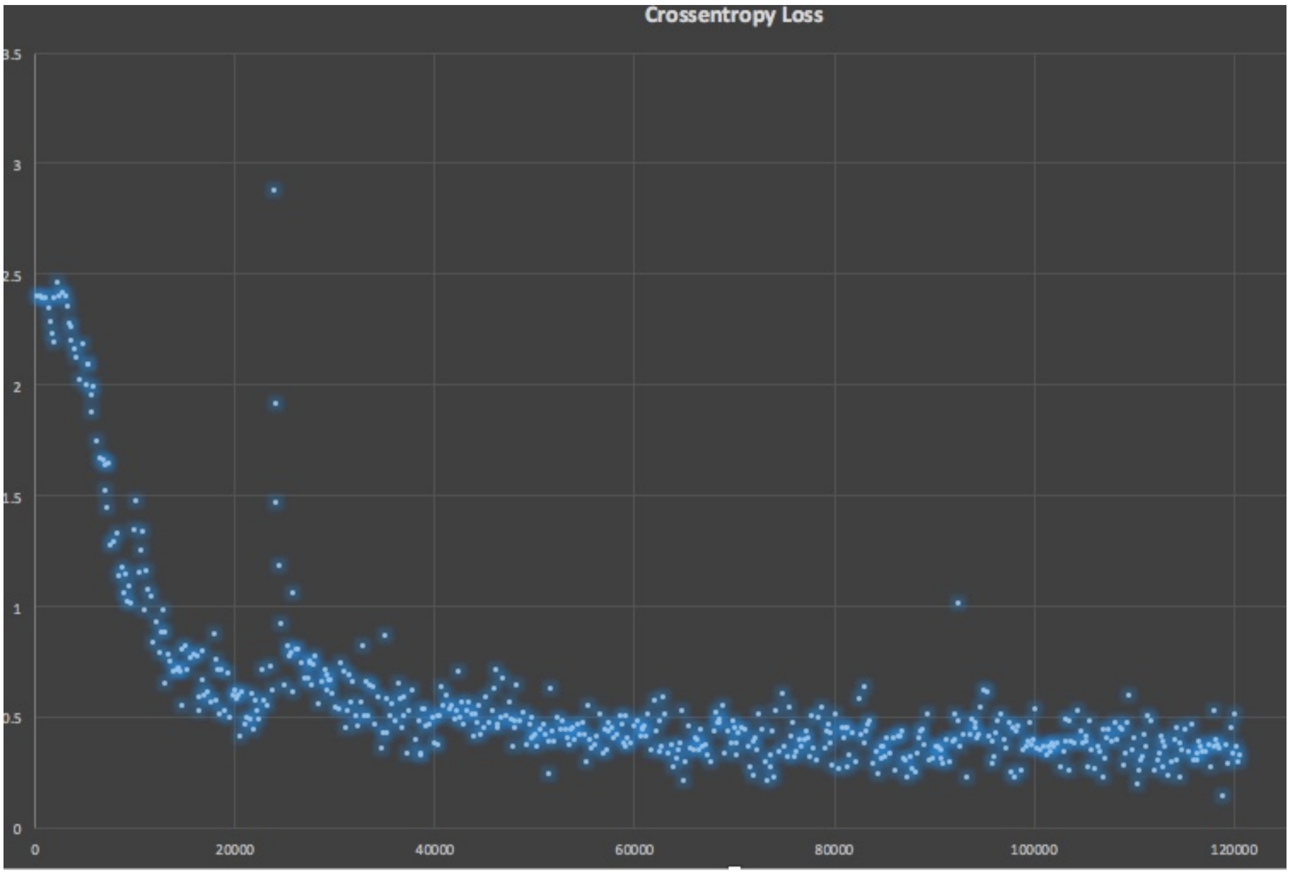
Architecture-1

A bidirectional LSTM net was used to predict the correct class. The LSTM layer had a hidden state of 32 and a batch size of 128 was chosen. The final softmax layer converted the 32 LSTM outputs to 11 classes. Crossentropy was used as a loss function.

The architecture was run with a 80-20 split of training and validation to obtain the optimal number of epochs. The results show that after about 600 epochs the model tends to overfit on the data. Here is a graph showing the same.

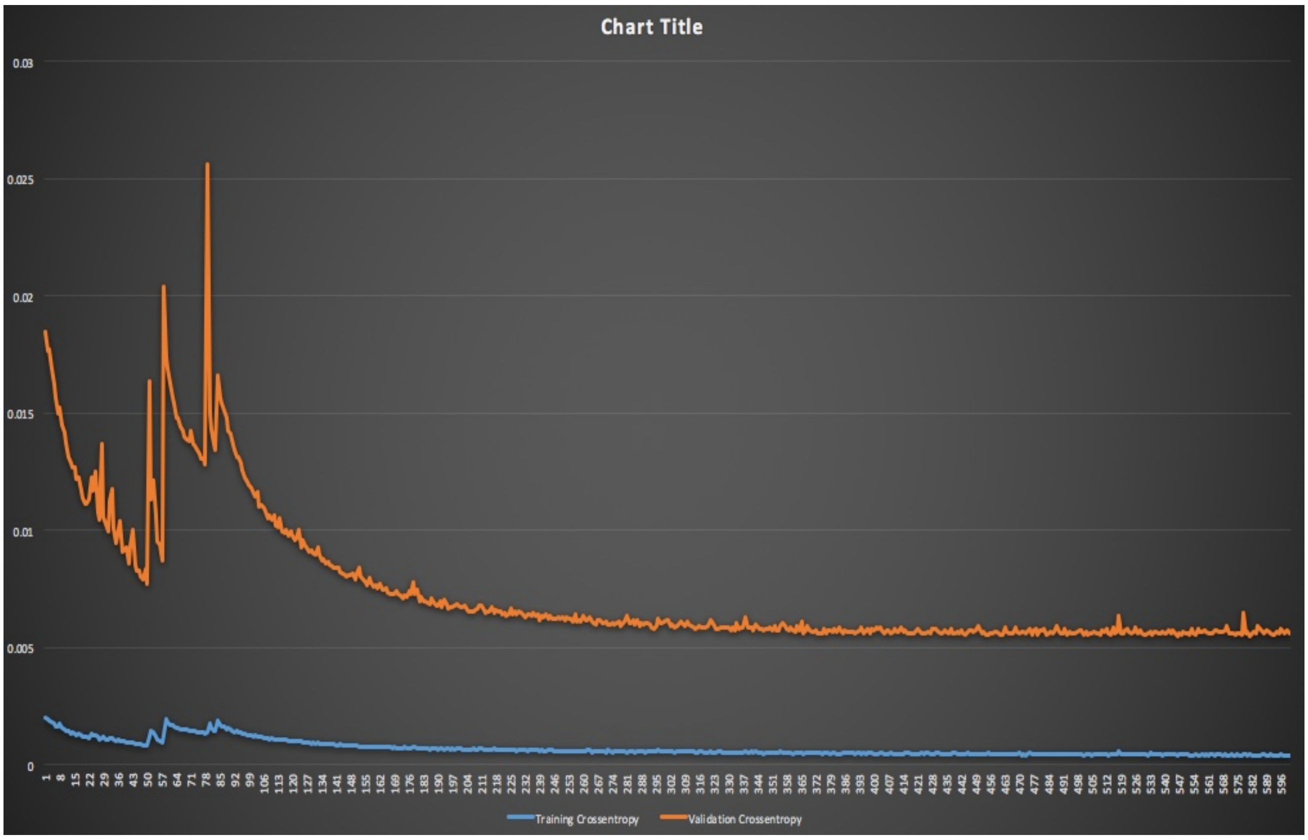


The big spike in validation loss at 600 epochs is the optimal point to break out of training to avoid overfitting. The loss for minibatches (per 100) is as follows:



Architecture-2

Two layers of unidirectional LSTM based RNN were used, followed by a softmax layer. The LSTM layers had hidden state sizes of 32 and 16 respectively. The softmax layer converted the 16 value output of the net to 11 class predictions. This model gives an accuracy of 78.8% on clean audio and 64% on audio with background noise.



Conclusion

The experiments make it clear that an accuracy of nearly 80%(on clean audio) can be achieved with RNNs. Bidirectional RNN achieve almost the same result as unidirectional LSTM based RNN, the only difference being that the training time of latter is much less than the former.

References

[1] Davis, S., Mermelstein, P.: Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE Trans. Acoust. Speech Signal Process. 28(4), 357–366 (1980)

[2] http://colah.github.io/posts/2015-08-Understanding-LSTMs/ as accessed on 5th Ocotber, 2017

[3] https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html as accessed on 5th October, 2017

[4] Dong Yu, Li Deng : Automatic Speech Recognition – A deep learning approach

[5] Graves et. Al. : Connectionist Temporal Classification: Labelling Unsegmented SequenceData with Recurrent Neural Networks, Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, 2006

[6] Panayatov et al : Librispeech: An ASR corpus based on public domain audio books), IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015

[7] Warden P. Speech Commands: A public dataset for single-word speech recognition, 2017. Available from http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz