COMPARISON OF RNN, LSTM AND GRU ON SPEECH RECOGNITION DATA

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Apeksha Nagesh Shewalkar

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

October 2018

Fargo, North Dakota

North Dakota State University
Graduate School

Title

COMPARION OF RNN, LSTM AND GRU ON SPEECH RECOGNITION DATA

By

Apeksha Nagesh Shewalkar

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig
Chair

Dr. Saeed Salem

Dr. Maria Alfonseca-Cubero

Approved:

October 23, 2018	Dr. Kendall Nygard
Date	Department Chair

ABSTRACT

Deep Learning [DL] provides an efficient way to train Deep Neural Networks [DNN]. DDNs when used for end-to-end Automatic Speech Recognition [ASR] tasks, could produce more accurate results compared to traditional ASR. Normal feedforward neural networks are not suitable for speech data as they cannot persist past information. Whereas Recurrent Neural Networks [RNNs] can persist past information and handle temporal dependencies. For this project, three recurrent networks, standard RNN, Long Short-Term Memory [LSTM] networks and Gated Recurrent Unit [GRU] networks are evaluated in order to compare their performance on speech data. The data set used for the experiments is a reduced version of TED-LIUM speech data. According to the experiments and their evaluation, LSTM performed best among all other networks with a good word error rate at the same time GRU also achieved results close to those of LSTM in less time.

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to my research advisor, Dr. Simone Ludwig at North Dakota State University for trusting me and giving me the opportunity of research in this new area on Deep Learning and Recurrent Neural Networks. I have learnt a lot in this whole process. Her indispensable guidance and encouragement during the research and execution of experiments made this paper possible.

I wish to express my gratitude to Guy Hokanson, System Administrator in Computer Science department for helping with the GPU machine issues while running the long runs. I also like to thank you NVIDIA Corporation for the support.

Lastly, I wish to thank my parents and to my spouse for their continuous support, faith and encouragement throughout my master studies.

DEDICATION

I would like to dedicate this paper to the beginners who are interested in massively growing Machine Learning field.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DL	Deep Learning
DNN	Deep Neural Network
ASR	Automatic Speech Recognition
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
GPU	Graphics Processing Unit
WER	Word Error Rate
CNN	Convolutional Neural Network
DBN	Deep Belief Network
CTC	Connectionist Temporal Classification
GMMs	Gaussian Mixture Models
HMMs	Hidden Marcov Models
MAP	Maximum-A-Posteriori
NLP	Natural Language Processing
PTB	Penn Tree Bank
CFL	Context Free Language
CSL	Context Sensitive Language
FFT	Fast Fourier Transform
ReLu	Rectified Linear Unit
BRNNs	Bidirectional Recurrent Neural Networks

1. INTRODUCTION

Deep Learning is a subset of Machine Learning. One of the architectures of Deep Learning is Deep Neural Networks [DNNs]. These DNNs are nothing but a class of Artificial Neural Networks having many hidden layers as compared to standard neural networks, hence the name Deep Neural Networks. Neural Networks have been around since many decades but being a large network, DNNs require more data to analyze and hence more powerful computers. Thus, because of the sudden rise of powerful computers using GPUs, deep learning has gained popularity in many areas recently.

Deep Learning architectures have been applied in many areas such as machine translation, language modeling, automatic speech recognition, image processing and many more. In the image processing area, one of the architectures of deep learning, Convolutional Neural Network [CNN] is used to recognize images. Specific hidden layers of CNN such as convolutional layers and pooling layers help to encode certain image properties.

Deep Belief Network [DBN] auto-encoder is another type of deep learning architecture [1]. It is similar to DNN, having multiple hidden layers. It consists of connections between the layers but no recurrent connection between units within each layer. DBN has been used for many tasks like natural language understanding or building automated fault detection method for quality inspections [2,3].

Deep learning is gaining huge popularity in the Automatic Speech Recognition area as well. Particularly, the speech recognition is a combination of good acoustic and language model [4,5]. Speech recognition data is continuous data and it can perform better if it has access to both past as well as future information in order to predict the current information.

Feedforward Neural Network is one of the basic architectures where the output of one layer is forwarded to its next layer, thus it works in an unidirectional way. In a particular Feedforward Neural Network architecture, the input layer is connected to the first hidden layer. Each hidden layer is always connected to the next hidden layer and finally the last hidden layer to the output layer. This manner of connecting layers is the reason why they are called 'feedforward'. As there is no connection to previous layers, feedforward neural networks cannot persist past information. Therefore, it makes them less suitable for the speech recognition task. When using DNN architectures for speech recognition tasks there are some problems to be considered like temporal dependencies and different speaking rates [6], [7], [8]. Standard DNNs are unable to handle different speaking rates; they can only work on fixed size acoustic frame windows.

Another architecture of deep learning is Recurrent Neural Networks [RNNs]. RNNs can access past information because of its loop like structure. In RNN, recurrent connections can be formed in three ways; between a neuron and a neuron itself or between a neuron and a neuron in the same layer or with the neuron and a neuron in the previous layer of a neural network architecture. These recurrent connections are formed with hidden and output neurons only and not with input or bias neurons. This type of architecture makes it useful to persist past information in order to predict current information and to deal with different speaking rates [8].

In the process of speech recognition, temporal dependencies play an important part as well. Temporal dependency could be shorter. For example, "The birds are flying in the *sky"*. Whereas in some cases temporal dependency could be longer too. For example, "I am born and brought up in India… I speak fluent *Hindi"*. In the first example, in order to predict the word *sky,* we need no further information as its obvious that the next word will be sky. However, in case of the second example, the actual physical distance between first sentence and the current sentence where it's needed to predict the word *Hindi* is long. RNNs have the limitations of the vanishing/exploding gradient problem, and thus, can work with short-term temporal dependencies only. Thus, the speech recognition problems in which temporal dependencies are shorter, RNNs are very popularly applied. Speech being a dynamic process, RNNs are better over feedforward neural networks [9].

Along with the short-term temporal dependency limitation, RNNs also need pre-segmented training data and conversion of the output into labeled sequences by performing post-processing on the data as well. This problem is solved by combining the Connectionist Temporal Classification [CTC] method with RNN. The CTC method can be used to label data sequences in the training process. It is being used and proven to be best in the case where the input and output label lengths are different and are unknown [10]. Another limitation as mentioned before while working with RNN is that RNNs cannot work efficiently with long-term dependencies in data where the distance between the relevant information and the place where it is required is large. RNNs cannot hold this information from long distance. This limitation has been overcome by a class of RNN, Long Short-Term Memory [LSTM] networks. LSTM networks have special memory cell structure, which is intended to hold long-term dependencies in data. And therefore, makes them perfect for speech recognition tasks [9]. Much later, a decade and half after LSTM, Gated Recurrent Unit [GRU] were introduced by Cho et al. [11] in 2014. They are similar to LSTM networks but with a simpler architecture, suitable to work on long-term dependencies and sequential data.

For this project, I have built three neural network models using standard RNN, LSTM and GRU cell. These models are trained end-to-end using the CTC method for sequence labeling and dropout as a regularization method. Afterwards, the performance of these three different models is evaluated for the speech recognition task on a reduced version of the TED-LIUM dataset [12].

2. RELATED WORK

In early days, the speech recognition task was typically performed based on generative models. These generative models are made up of Gaussian Mixture Models [GMMs], Hidden Markov Models [HMMs] and Maximum-A-Posteriori [MAP] estimation [13], [14]. However, restrictions to these models are; they need expert knowledge about the specific language at hand and in case of Automatic Speech Recognition [ASR] using generative models it requires specific pre-processing of the speech data [14]. Whereas, when ASR is performed end-to-end it does not require expert knowledge as it dependents on a good acoustic and language model used [14]. Due to the advancement in deep learning architectures more discriminative models (sequence to sequence models) have been used for speech recognition tasks [13], [15]. For these discriminative models, audio sequences act as an input and gives corresponding text transcript sequences as an output [13].

Language modeling is the essential element to many Natural Language Processing [NLP] tasks such as machine translation, and speech recognition. Given a particular sequence of words the language model predicts the next word in the sequence with the help of a probabilistic model built to assign probabilities to the words. RNNs have performed very sound in language modeling tasks [16], [17]. The dataset used for performance evaluation here was the Penn Tree Bank [PTB] data set [17].

RNNs and LSTMs have also been applied to sequence to sequence mapping problems. Typically, in sequence-to-sequence models, two RNNs are used, one as an encoder for input processing and another as a decoder at the output end to create output. In translation tasks [18], multilayered RNN cells are used and the performance is evaluated using the WMT'14 data set on English to French translation.

LSTMs and RNNs have performed excellent in speech recognition tasks. In case of learning Context Free Language [CFL] and Context Sensitive Language [CSL], LSTM has been proven to be an efficient choice over standard RNNs [19].

For training RNNs for the speech recognition task, sequence labeling is an important step. HMM is used in the past with the RNN model for sequence labeling [20], [21]. However, currently with DNN, the HMM-RNN framework does not perform efficiently. The Connectionist Temporal Network [CTC] method has been introduced by Graves et al. [10] as an efficient solution for the sequence labeling task. The CTC method could train RNN end-to-end without the need of pre-segmentation of input training data or post processing of output. Further, it is the perfect choice for problems where the input-output label alignment is not known. The Deep LSTM RNN model was also build by Grave et al. [9] and trained with the CTC method end-to-end. The framework was built for the speech recognition task and the performance was evaluated on the TIMIT phone recognition data. They have achieved state-of-the-art results for this task.

RNN, LSTM and DNN were also applied to large vocabulary speech recognition problems - the Google English Voice Search Task by Sak et al. [22]. Here in this problem they used a modified version of a standard LSTM network architecture for optimal use of all model parameters.

The TIMIT speech data set has been used in many experiments where different architecture models like bidirectional LSTM, deep bidirectional LSTM, RNNs, and hybrid are built and evaluated. The Phoneme Classification task is performed using bidirectional LSTMs in [23], [24]. Results of framewise phoneme classification shows that bidirectional LSTMs performed better than unidirectional LSTMs as well as standards RNNs. Results of this experiment shows that bidirectional architectures are a better choice in speech recognition tasks as relevant information can be present in past or future.

The Hybrid bidirectional LSTM-HMM network has been used for the phoneme recognition task and proved to be an improvement compared to unidirectional LSTM-HMM and traditional HMM systems. Bidirectional LSTMs have been proven to be better than state-of-the-art HMM based systems when experimented with the handwriting recognition task with both online and offline data by Graves et al. [25].

Deep bidirectional LSTM architecture are also being used for the speech recognition task. In particular, for the deep bidirectional LSTM network, each hidden layer is a combination of a forward layer and a backward layer. Each hidden layer receives an input from the previous forward and backward layer. They were combined with the CTC objective function to create an end-to-end model for speech recognition in [26]. The performance was evaluated on the Wall Street Journal corpus. This approach of using the objective function helped authors achieve very good results with a word error rate even in the absence of a language model.

A hybrid system of deep bidirectional LSTM and HMM has also been experimented on the speech recognition problem in [27]. The performance was evaluated on the TIMIT data set where it outperformed the GMM deep network benchmark results obtained on the partial Wall Street Journal corpus.

Hundreds of hours of speech data have been used in speech recognition tasks with a variety of DNN architectures. This includes data like Wall Street Journal, Librispeech, Switchboard, TED-LIUM, Fisher corpus [28], [29], [30]. TED-LIUM data in particular has been used for experiments and tasks like for the audio augmentation task [30], for modeling probabilities of pronunciation and silence [31], and also in automatic speech recognition with human correction task both at the word level as well as the lattice level [32].

Cho et al. in 2014 [33] came up with a different version of recurrent neural networks known as Gated Recurrent Unit [GRU]. Being a variation of RNN, they do not have the problem of the vanishing/exploding gradient problem. The GRU architecture is similar to LSTM but simpler than LSTM. Due to its simpler structure and fewer internal gates it is less expensive than LSTM. Both LSTM and GRU networks have been used in speech recognition tasks and in polyphonic music modeling [11], [34].

This MS research work is inspired from Hannun et al. [28]. They have used the standard RNN model with one hidden layer of bidirectional RNN for the speech recognition task. Multiple GPUs [Graphics Processing Unit] were used in parallel to speed up the experiments. However, I have used a single GPU for my experiments to build and evaluate three different bidirectional models RNN, LSTM and GRU in performing the speech recognition task.

3. RECURRENT NEURAL NETWORKS

In this section, three RNN models used for this experiment are explained in brief with the help of the corresponding network architectures and equations.

3.1. Recurrent Neural Network (RNN)

Recurrent Neural Networks are the type of neural networks with loops which allow them to persist information from the past in the network model.

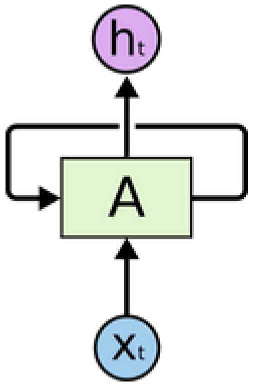

Figure 1. Basic loop structure in RNN [8]

In Figure 1, the center square represents a neural network, which takes input x_t at the current time slice *t* and gives the value h_t as an output. The loop shown in the structure enables it to use information from past time slices to produce output for the current time slice *t*. Thus, we can say that the decision made at time slice *t-1* affects the decision to be made at time slice *t*. So, the response of the network to the new data depends on the current input as well as the output from the recent past data. The RNN output calculation is based on iteratively calculating the output of the following two equations:

$$h_t = H(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \quad (1)$$

$$y_t = W_{hy}\, h_t + b_y \quad (2)$$

In Equations (1) and (2), x_t is the input sequence at the current time slice t, y_t is the output sequence at time slice t, and h represents the hidden vector sequence from time slice 1 to T. W and b represents weight matrices and biases, respectively. Lastly, an activation function used for the hidden layer is H.

3.2. Long Short-Term Memory (LSTM) Network

LSTMs are a special type of recurrent neural networks with memory cells. These memory cells are the essential part in handling long-term temporal dependencies in the data. To remember information over a long period is their default behavior and they do not struggle to learn it. LSTMs also deal with the vanishing/exploding gradient problem during backpropagation. Thus, they overcome both of the shortcomings that RNNs face.

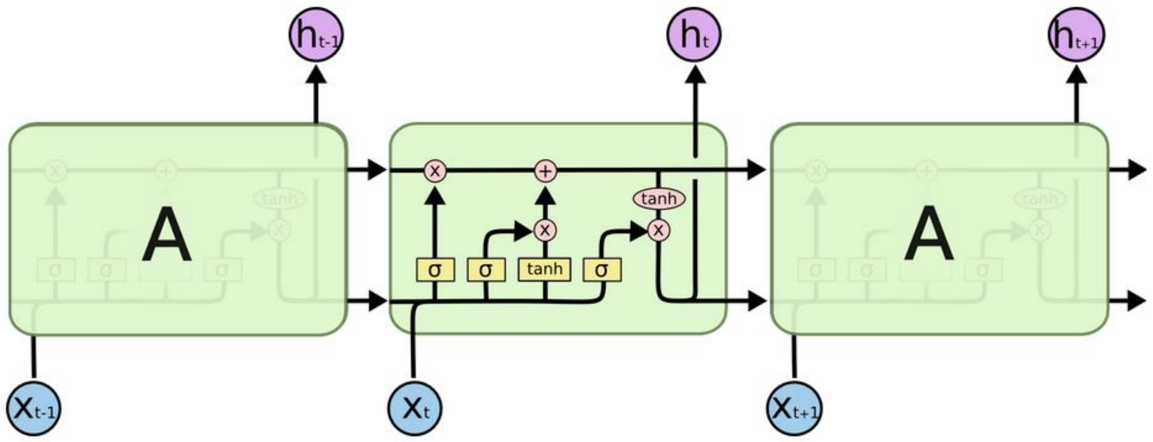

Figure 2. LSTM cell with four interacting layers [8]

Figure 2 shows the chain like structure of LSTM and a particular memory cell in LSTM. Each big square block represents a memory cell here. The cell state is the vital part of LSTM and is shown by the horizontal line going through the top of the cell in the figure. It runs from every cell in the chain of the LSTM network. LSTM has the option to add or delete information from this cell state. This operation is done by another structure in LSTM called gates. Gates are formed by the sigmoid activation function (shown by σ in Figure 2) and pointwise multiplication operation (shown by $\otimes$ in Figure 2). As shown in above diagram there are three gates which control information to pass through the cell state. They are as follows:

- Forget gate – decides what information to throw away
- Input gate – decides what new information to save in the cell state
- Output gate – decides what information of the cell state to output

Originally, Hochreiter and Schmidhuber first came up with LSTM networks in 1997 [35]. Since then, there have been variations in the memory cell architecture by people for experimenting in different application areas. The calculations in standard single LSTM cell can be stated by the following equations:

$$f_t = \sigma\left(W_f \,.\, [h_{t-1}, x_t] + b_f\right) \tag{3}$$

$$i_t = \sigma\,(W_i \,.\, [h_{t-1}, x_t] + b_i) \tag{4}$$

$$\tilde{C}_t = tanh\,(W_c \,.\, [h_{t-1}\,, x_t] + b_c) \tag{5}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{6}$$

$$o_t = \sigma\,(W_o \,.\, [h_{t-1}, x_t] + b_o) \tag{7}$$

$$h_t = o_t * tan\,h\,(C_t) \tag{8}$$

where the activation functions used are sigmoid function (σ) and hyperbolic tangent function ($\tan h$), i_t, f_t, o_t, C_t, $\widetilde{C_t}$ represents the input gate, forget gate, output gate, memory cell content and new memory cell content, respectively. As mentioned earlier, three gates are made up of the sigmoid function, and the output of the particular cell is scaled up by using the hyperbolic tangent function.

3.3. Gated Recurrent Unit (GRU)

GRUs are another type of RNNs with memory cells. They are similar to LSTM but with simpler cell architecture. GRU also has gating mechanism to control the flow of information through cell state but has fewer parameters and does not contain an output gate.

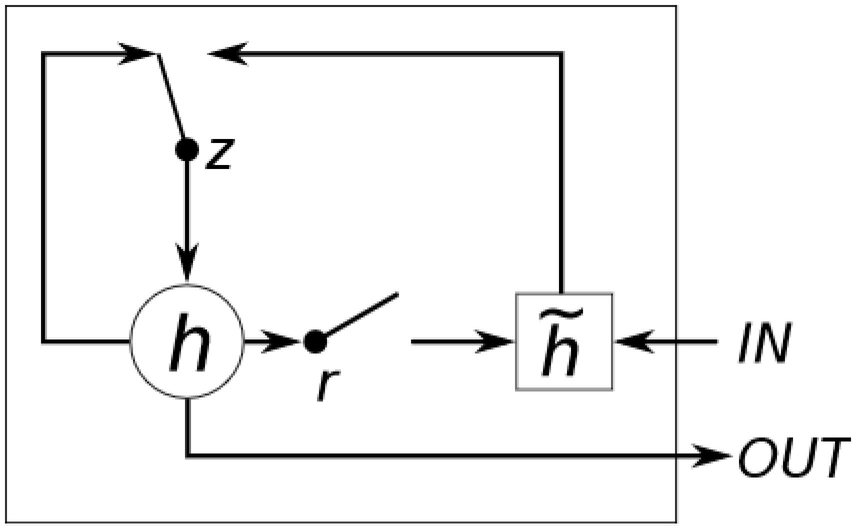

Figure 3. Gated Recurrent Unit (GRU) single cell structure [11]

Figure 3 shows a particular single cell structure of GRU. It consists of two gates, *r* is a reset gate, and *z* an update gate. The reset gate regulates the flow of new input to the previous memory, and the update gate determines how much of the previous memory to keep. If we compare GRU with LSTM, the update gate is the combination of the input and forget gate and the previous hidden state (*h* in Figure 3) is connected to the reset gate directly. Another difference is in the exposure of memory content. As GRUs do not have an output gate, it exposes all of its memory content, whereas in LSTM the memory content to be used or seen by other units/cells in the network is managed by the output gate [11]. The following equations are used in the GRU output calculations:

$$r_t = sigm\,(W_{xr}\,x_t + W_{hr}\,h_{t-1} + b_r) \tag{9}$$

$$z_t = sigm\,(W_{xz}\,x_t + W_{hz}\,h_{t-1} + b_z) \tag{10}$$

$$\tilde{h}_t = \tanh(W_{xh}\, x_t + W_{hh}\,(r_t \odot h_{t-1}) + b_h) \tag{11}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \tag{12}$$

In Equations (9)-(12), x_t, h_t, r_t, z_t represent the input vector, output vector, reset gate and an update gate, respectively. All W variables denote the weight matrices, and b are biases. Activation functions used are the same as LSTM, sigmoid function ($sigm$) and hyperbolic tangent ($\tan h$) function.

The gating mechanism in both LSTM and GRU cells makes them the perfect choice for long-term dependencies. Bahdanau et al. in [36] experimented with both of these cells for the machine translation task. Their preliminary experiments proved both LSTM and GRU comparable to each other. Chung et al. [11] performed the evaluation of LSTM and GRU cells on the sequence modeling task using data sets like raw speech signal data and polyphonic music data. Although both, LSTM and GRU networks, performed well they were unable to conclude which one was better than other. These experiments motivated me to include GRU along with LSTM for their performance evaluation in this research for the speech recognition task.

4. EXPERIMENT ARCHITECTURE

The architecture used for this experiment is based on the RNN architecture in [28]. For the models in this experiment, preprocessed speech spectrogram is the input and it gives the corresponding English plain text as an output. In the preprocessing step, a small window of raw audio waveform (typically 20ms) is selected. Next, the Fast Fourier Transform [FFT] is calculated and the magnitude (power) is taken to describe the frequency content in a particular window of the audio waveform selected initially. This is how one frame is computed. After computing all frames similarly, frames from adjacent windows are concatenated to form a spectrogram. This spectrogram acts as input features for the RNN model architecture. This pre-processing step is explained in visual format on the simple “Hello World” example in Figures 4 and 5.

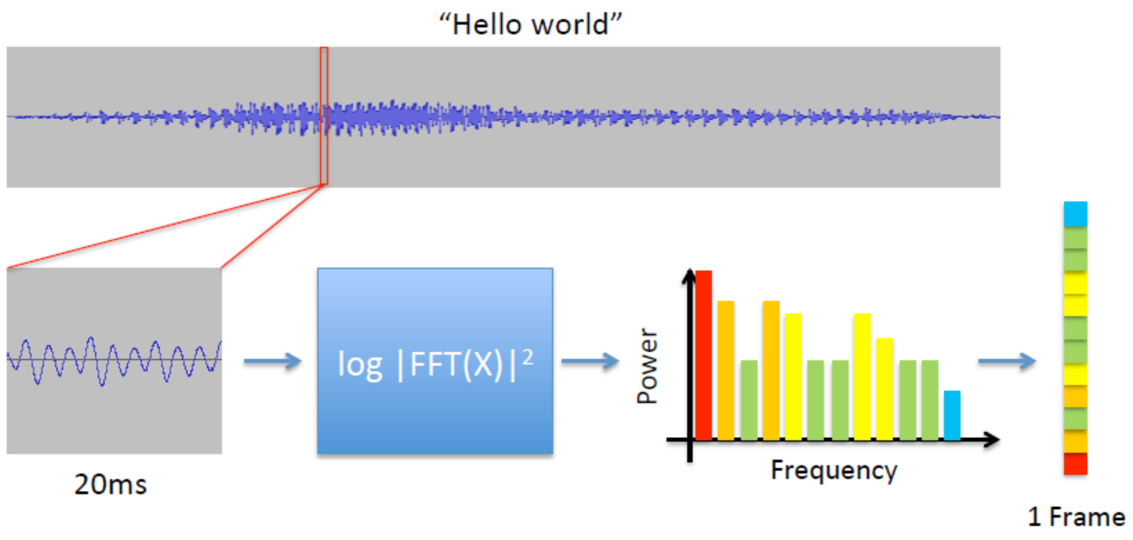

Figure 4. Pre-processing of a small window of “Hello World” raw audio waveform to corresponding one frame of spectrogram [37]

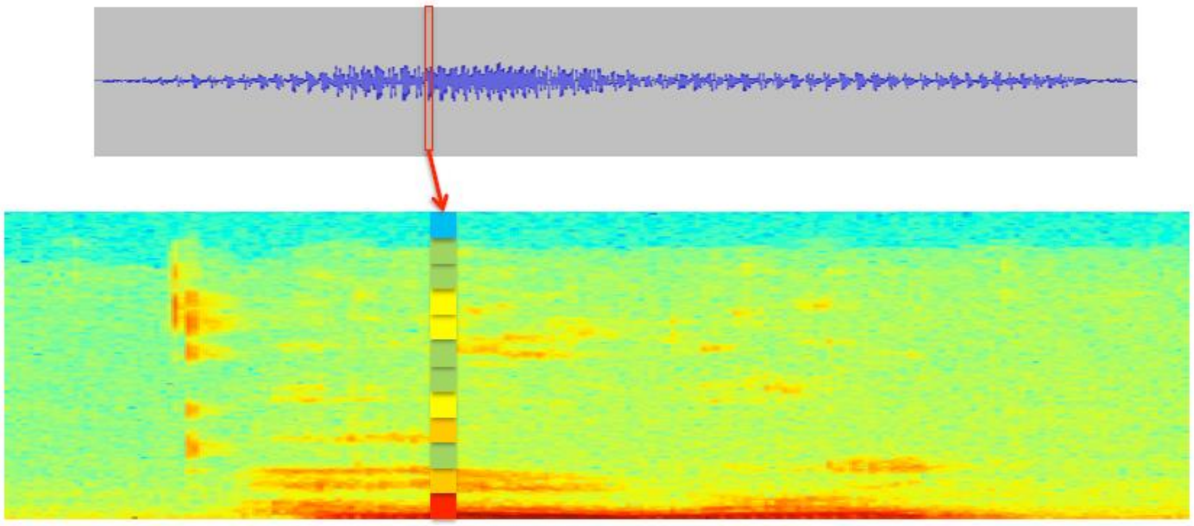

Figure 5. Full spectrogram from raw audio waveform with one frame showed in zoom [37]

Assume X = $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots\}$ is a training set and a single utterance *x* with label *y* is sampling from this training set X. In this training set, every utterance $x^{(i)}$ is a time series of $T^{(i)}$ length. Here, the $T^{(i)}$ time slice is a vector representation of audio features $x_t^{(i)}$ where *t* = 1 to $T^{(i)}$. The ultimate aim is, given a transcription y, convert an input sequence x into a sequence of character probabilities using $\hat{y}_t = \mathbb{P}(c_t|x)$ where $c_t \in \{a, b, c, \ldots, z, space, apostrophe, blank\}$.

The RNN model shown in Figure 6 is a deep network with five hidden layers, one input and one output layer. The hidden units of layer *l* are written as $h^{(l)}$ and that makes $h^{(0)}$ an input. The output of the first layer (the input layer) at each time *t* is dependent on the spectrogram frame x_t and the context of C frames on each side. For this experiment, context values C ∈ {5,7,9} are considered. In the model shown in Figure 6, layer one, two and three are normal feedforward layers and for each time *t*, they are calculated as follows:

$$h_t^{(l)} = g\left(W^{(l)}\, h_t^{(l-1)} + b^{(l)}\right) \tag{13}$$

where, g(z) is the clipped rectified-linear unit [ReLu] activation function to calculate the output at the respective hidden layers. $W^{(l)}$, $b^{(l)}$ are the weight matrix and bias vector used at layer *l*, respectively. To overcome the vanishing gradient problem, ReLu is preferred instead of normal sigmoid activation function.

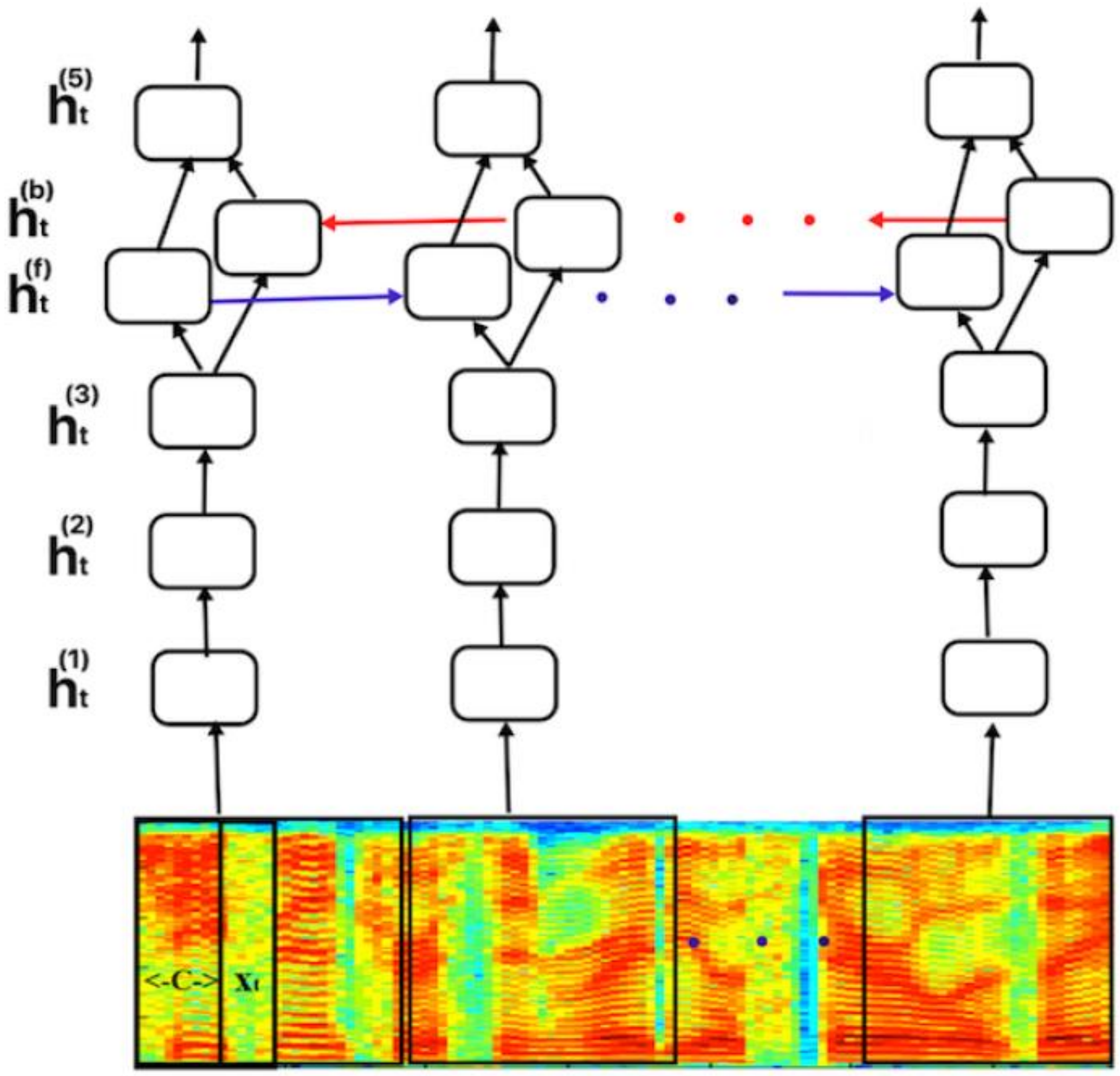

Figure 6. The RNN architecture with speech spectrogram as an initial input and five hidden layers

Standard RNN could persist information from the past only. In case of the speech recognition task, speech training data is recorded all at once. Thus, in order to predict current data, future as well as past data plays an important role here. For this purpose, Bidirectional Recurrent Neural networks [BRNNs] are useful. Therefore, the next hidden layer (layer four) in the model is a Bidirectional Recurrent layer which is made up of forward hidden sequence and backward hidden sequence [9]. These two hidden sequence layers in BRNN, one forward hidden sequence $h^{(f)}$ and one backward hidden sequence $h^{(b)}$ are calculated by following equations:

$$h_t^{(f)} = g\left(W^{(4)}\, h_t^{(3)} + W_r^{(f)}\, h_{t-1}^{(f)} + \, b^{(4)}\right) \tag{14}$$

$$h_t^{(b)} = g\left(W^{(4)}\, h_t^{(3)} + W_r^{(b)}\, h_{t+1}^{(b)} + \, b^{(4)}\right) \tag{15}$$

In Equations (14) and (15), the forward hidden sequence is calculated sequentially from *t* =1 to *t*= $T^{(i)}$ and the backward hidden sequence is calculated sequentially from *t* = $T^{(i)}$ to *t* = 1 for the i^{th} utterance. Once the bidirectional layer processes the data in both directions, it feed forward the output to the next layer (layer five), which is again a normal feedforward layer. Layer five which takes input from layer four can be calculated as:

$$h_t^{(5)} = g\left(W^{(5)}\, h_t^{(4)} + \, b^{(5)}\right) \tag{16}$$

where, $h_t^{(4)} = \, h_t^{(f)} + \, h_t^{(b)}$

After hidden layer five, the last layer is the output layer. The standard softmax function at this layer evaluates the predicted character probabilities in each time slice *t* and character *k* in the alphabet. Equation (17) calculates these probabilities at the output layer:

$$h_{t,k}^{(6)} = \, \hat{y}_{t,k} = \, \mathbb{P}\,(c_t = k|x) = \frac{exp\left(W_k^{(6)} h_t^{(5)} + b_k^{(6)}\right)}{\sum_j exp\left(W_j^{(6)} h_t^{(5)} + b_j^{(6)}\right)} \tag{17}$$

where $W_k^{(6)}$ and $b_k^{(6)}$ are the k^{th} column in the weight matrix and the k^{th} bias, respectively. Once the predicted character probabilities $\mathbb{P}\,(c_t = k|x)$ are computed, the next step is to calculate the CTC loss [28] $\mathcal{L}\,(\hat{y}, y)$ to measure the prediction error. When provided a network output, the CTC loss function computes the error in the predicted output. This error is the negative log likelihood of the target probability. The input for this CTC loss function is the output of the predicted character probabilities for each time slice obtained from Equation (17). Once the CTC loss is calculated, the corresponding loss gradient has to be calculated given the actual character sequence *y* and the network outputs. Afterwards this loss is backpropagated from the output layer to the weights in the network layer by layer. For this experiment, out of different backpropagation algorithms available, the ADAM optimization algorithm [38] is used. The reason behind selecting this algorithm is that it is very tolerant to the learning rate and also to other training parameters which lead to less fine-tuning.

In this project, experiments are performed on three models. The only difference between three models is the kind of bidirectional recurrent layer used in layer four of the architecture explained above. The first model, which is described above, uses a Bidirectional RNN layer. The second and third model make use of the bidirectional LSTM layer and bidirectional GRU layer, respectively. The formulation of these second and third model is similar to the first model.

5. EXPERIMENTS AND RESULTS

In this section, the speech data set used for the experiments, the evaluation measures and lastly the results are discussed in detail.

5.1. Data Set

For these experiments, a subset of the TED-LIUM release 2 corpus [12], which is available publicly is used. This is the second version of the TED-LIUM dataset with enhanced language model. Rousseau et al. [4] were able to achieve more accurate results in terms of word error rate [WER] compared to the older version (TED-LIUM release 1) of this data. The data set has filtered data from the TED website with audio files and their corresponding transcriptions. This data is specifically designed to train and evaluate acoustic models. For these experiments, I have reduced the data from 34.3 GB to 11.7 GB. This reduced data set can be found at [39]. The data is already separated in train, test and validation folders. The data has the following contents:

- 378 audio talks in NIST sphere format (SPH files)
- 378 corresponding transcripts (STM format files)
- Dictionary with pronunciations (152k entries)
- Language Model improved with selected monolingual data from WMT12 corpus [4]

5.2. Evaluation Measures

Normally, speech recognition task performance can be measured based on two different parameters, the first is accuracy and the second is speed [40]. The accuracy based evaluation measures are WER, loss, and mean edit distance.

Most research work done in the ASR area have used WER as their error measurement. WER is derived from the Levenshtein distance [41] and is formulated by [42], [43]:

$$WER = \left(\frac{S+I+D}{N}\right) \times 100 \tag{18}$$

where *N* denotes the number of total words present in the actual transcript, S, I, D are the number of substitutions, number of insertions, and number of deletions, respectively. WER is taken as the lower the WER value the better is the speech recognition [42], [43].

The loss term is also mentioned as Expected Transcription Loss [26]. This expected transcription loss function is given by:

$$\mathcal{L}(x) = \sum_y P_r\,(y|x)\,\mathcal{L}(x,y) \tag{19}$$

where *x* is the input sequence given, $P_r\,(y|x)$ is the distribution over transcript sequence y given by CTC, and $\mathcal{L}(x,y)$ is a transcription loss function.

The edit distance is explained with the help of example below. Assume that $d(A,B)$ is the normalized edit distance between two words or strings *A* and *B* [44]. Then, the mean edit distance can be calculated by:

$$d(A,B) = \min\left(\frac{W(P)}{N}\right) \tag{20}$$

where *P* denotes the editing path between *A* and *B*, *W(P)* is the total sum of weights of all edited operations of editing path *P*, and *N* denotes the total number of edited operations (the total length of editing path, *P*) [44].

5.3. Hyperparameter Setup

The hyperparameter values used while training the speech data in the experiment are stated below:

- Total epochs = 10
- Training batch size = 16
- Testing batch size = 8
- Dropout rate = 30%

- Activation function = ReLu
- Total number of neurons in each hidden layer = 500 or 1000 (as specified)
- Backpropagation technique = ADAM Optimizer with:
    - $\beta_1 = 0.9$
    - $\beta_2 = 0.999$
    - $\epsilon = 1e-8$
    - learning rate $\alpha = 0.0001$

5.4. Results

Experiments were ran using three models as explained earlier, standard RNN, LSTM and GRU with two different configuration architectures. First architecture has 500 nodes in each hidden layer whereas the second architecture has 1000 nodes in each hidden layer.

Table 1 shows the results of the three models each using 500 nodes architecture. The WER is measured in percentage. RNN achieved 87.02% WER whereas LSTM and GRU achieved closer values of WER 77.55% and 79.39%, respectively. As a part of the neural network optimization the loss was measured. Loss values for RNN, LSTM and GRU are 186.61, 160.51 and 162.22, respectively. We can say that the loss values show a similar trend as WER. Lastly, the mean edit distance values are mentioned in the table out of which LSTM and GRU achieved better values of 0.3853 and 0.3939, respectively.

Table 1. 500 nodes layer architecture results

Model	WER (%)	Loss	Mean edit distance
Standard RNN	87.02	186.61	0.4484
LSTM	77.55	160.51	0.3853
GRU	79.39	162.22	0.3939

Table 2 shows the results of the three models with 1000 nodes layer architecture. These results show similar trends as with 500 nodes but with better results. In terms of WER, LSTM achieved a value of 65.04 compared to the other two models. The loss values for RNN, LSTM and GRU models are 164.60, 134.35 and 136.89, respectively. In terms of mean edit distance also LSTM achieved the best result of 0.3222. As we can see, LSTM and GRU model achieved close results in terms of all three measures, WER, loss, and mean edit distance.

Table 2. 1000 nodes layer architecture results

Model	**WER (%)**	**Loss**	**Mean edit distance**
Standard RNN	78.66	164.60	0.3991
LSTM	65.04	134.35	0.3222
GRU	67.42	136.89	0.3308

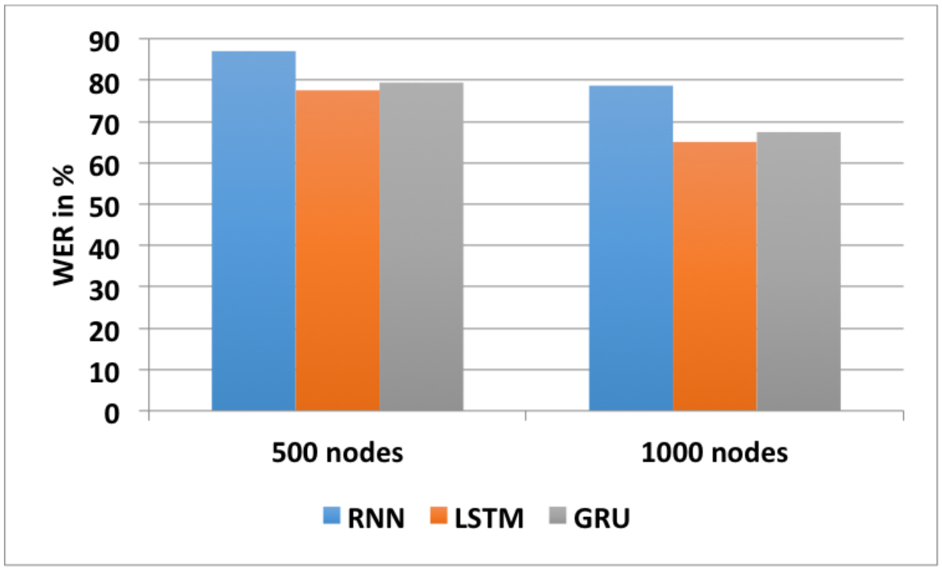

Figure 7. WER values in percentage for all three models, RNN, LSTM and GRU considering 500 nodes and 1000 nodes layer architecture

Figure 7 shows the WER values for both the 500 nodes and the 1000 nodes architecture. We can see that the LSTM achieved the lowest WER compared to the GRU and RNN models for both architectures.

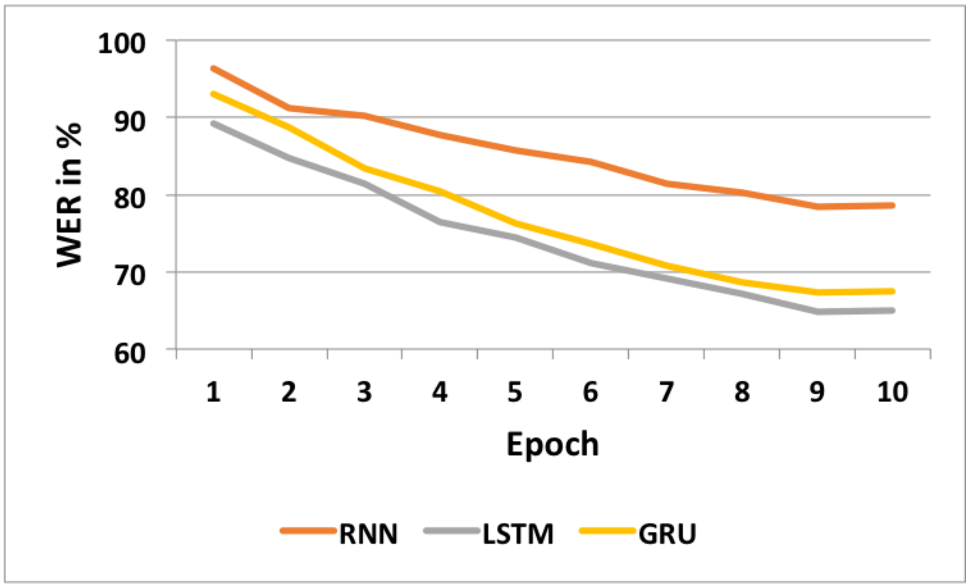

Figure 8. WER values in percentage for each epoch for three models with 1000 nodes

Figure 8 shows the WER values per epoch for all three models with the 1000 nodes layer architecture. At the end of each epoch, the test data was applied to test the model obtained at the end of each epoch and WER values were recorded. The pattern is the same as we observed so far. LSTM has achieved lower a WER compared to other two (RNN and GRU) models. We can also observe from the figure that the models started converging after Epoch 9 and the best WER values were recorded at Epoch 9 as 78.43% for RNN 64.76% for LSTM and 67.34% for GRU model.

If we observe the models in terms of running time, as shown in Figure 9, the RNN model has the shortest running time in terms of days and it beats the other two models. However, the WER values of LSTM and GRU are way better than RNN and we cannot compare RNN with them. In case of the 500 nodes layer architecture, LSTM took more than two days and GRU model took approximately 1.5 days. With the 1000 nodes layer architecture the values are more significant. LSTM model ran for more than 7 days whereas GRU run was finished in 5 days and 5 hours.

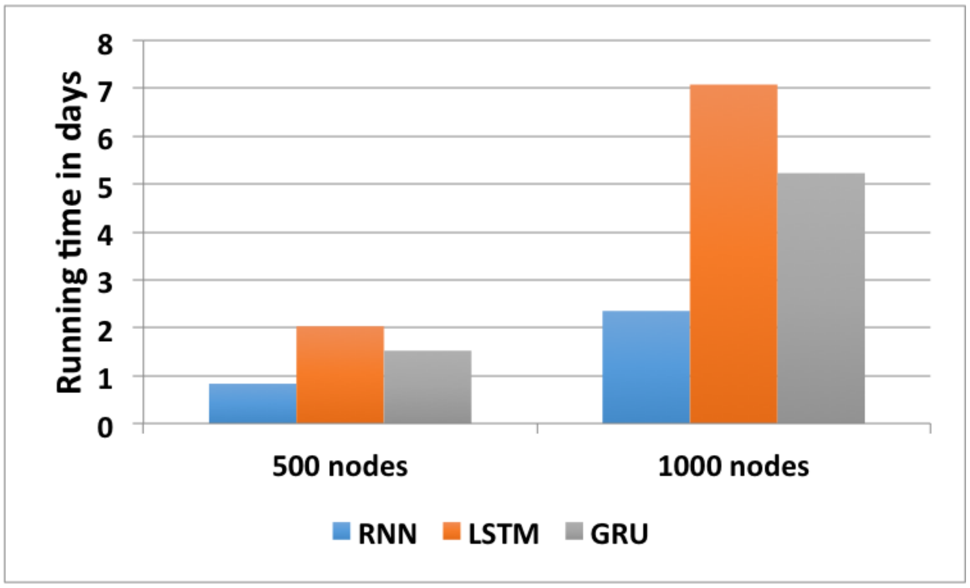

Figure 9. Running time of three models in days with 500 nodes and 1000 nodes layer architecture

6. CONCLUSION

A simple feedforward neural network is not capable of handling reverse connections. They cannot persist past information to make predictions at the current time which is important for tasks like speech recognition. For this purpose, RNNs were introduced. RNNs have a loop like structure and can persist short-term past information. Due to limitations of RNNs to the vanishing/exploding gradient problem and not being able to work on long-term temporal dependencies in the data, LSTMs were introduced which overcame these limitations with the help of memory cells in their structure. Recently, GRU were also introduced which could be used to solve similar type of problems such as LSTMs with simpler architecture.

This project evaluated these three networks, standard RNN, LSTM and GRU, and compared their performances. The data set used for comparison was the reduced TED-LIUM speech data. The networks were trained and evaluated with two architectures. The one with 500 nodes in each hidden layer and another with 1000 nodes in each hidden layer. WER, loss and mean edit distance are the evaluation measures used for these experiments. While observing the results it is obvious that there is a tradeoff between accuracy in the results and the run time of the model. Though, LSTM achieved better results in all the runs, its run time is highest among all of the models. We could see that the GRU values are also close to LSTM values with lesser run time. Thus, the recommendation would be to use GRU cell neural network while working on smaller data like in this case the reduced TED-LIUM data.

REFERENCES

1. “Deep Belief Network.” *Wikipedia*, Wikimedia Foundation, en.wikipedia.org/wiki/Deep_belief_network.
2. R. Sarikaya, G. E. Hinton and A. Deoras, "Application of Deep Belief Networks for Natural Language Understanding," in *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 4, pp. 778-784, April 2014.
3. J. Sun, A. Steinecker, P. Glocker, Application of Deep Belief Networks for Precision Mechanism Quality Inspection. In: Ratchev S. (eds) Precision Assembly Technologies and Systems. IPAS 2014. IFIP Advances in Information and Communication Technology, vol 435. Springer, Berlin, Heidelberg, 2014.
4. A. Rousseau, P. Delglise, Y. Estve, Enhancing the TED-LIUM Corpus with Selected Data for Language Modeling and More TED Talks. In LREC, 3935-3939, May 2014.
5. Y. Gaur, F. Metze, J. P. Bigham, Manipulating Word Lattices to Incorporate Human Corrections, Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 2016.
6. E. Busseti, I. Osband, S. Wong, Deep Learning for Time Series Modeling, Seminar on Collaborative Intelligence in the TU Kaiserslautern, Germany, 2012.
7. Deep Learning for Sequential Data - Part V: Handling Long Term Temporal Dependencies, https://prateekvjoshi.com/2016/05/31/deeplearning-for-sequential-data-part-v-handling-long-term-temporaldependencies/, last retrieved July 2017.
8. Understanding LSTM Networks, http://colah.github.io/posts/2015-08-Understanding-LSTMs/, last retrieved July 2017.

9. A. Graves, A. R. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks. In 2013 IEEE International Conference on Acoustics, speech and signal processing (ICASSP), 6645-6649, 2013.
10. A. Graves, S. Fernndez, F. Gomez, J. Schmidhuber, Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In Proceedings of the 23rd international conference on Machine learning, 369-376, ACM, June 2006.
11. J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.
12. TED-LIUM Corpus, http://www-lium.univ-lemans.fr/en/content/ted-lium-corpus, last retrieved July 2017.
13. C. C. Chiu, D. Lawson, Y. Luo, G.Tucker, K. Swersky, I. Sutskever, N. Jaitly, An online sequence-to-sequence model for noisy speech recognition, arXiv preprint arXiv:1706.06428, 2017.
14. T. Hori, S. Watanabe, Y. Zhang, W. Chan, Advances in Joint CTC-Attention based End-to-End Speech Recognition with a Deep CNN.
15. W. Chan, N. Jaitly, Q. V. Le, O. Vinyals, Listen, attend and spell. arXiv preprint arXiv:1508.01211, 2015.
16. T. Mikolov, Statistical language models based on neural networks, PhD thesis, Brno University of Technology, 2012.
17. W. Zaremba, I. Sutskever, O. Vinyals, Recurrent neural network regularization. arXiv preprint arXiv:1409.2329, 2014.
18. I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks. In Advances in neural information processing systems, 3104-3112, 2014.

19. F. A. Gers, E. Schmidhuber, LSTM recurrent networks learn simple context-free and context-sensitive languages. IEEE Transactions on Neural Networks, 12(6), 1333-1340, 2001.

20. O. Vinyals, S. V. Ravuri, D. Povey, Revisiting recurrent neural networks for robust ASR. In 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 4085-4088, 2012.

21. A. L. Maas, Q. V. Le, T. M. O'Neil, O. Vinyals, P. Nguyen, A. Y. Ng, Recurrent neural networks for noise reduction in robust ASR. In Thirteenth Annual Conference of the International Speech Communication Association, 2012.

22. H. Sak, A. Senior, F. Beaufays, Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. arXiv preprint arXiv:1402.1128, 2014.

23. A. Graves, J. Schmidhuber, Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks, 18(5), 602-610, 2005.

24. A. Graves, S. Fernndez, J. Schmidhuber, Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition. In: Duch W., Kacprzyk J., Oja E., Zadrony S. (eds) Artificial Neural Networks: Formal Models and Their Applications ICANN, Lecture Notes in Computer Science, vol. 3697, Springer, Berlin, Heidelberg, 2005.

25. A. Graves, M. Liwicki, S. Fernndez, R. Bertolami, H. Bunke, J. Schmidhuber, A novel connectionist system for unconstrained handwriting recognition, IEEE transactions on pattern analysis and machine intelligence, 31(5), 855-868, 2009.

26. A. Graves, N. Jaitly, Towards end-to-end speech recognition with recurrent neural networks. In Proceedings of the 31st International Conference on Machine Learning (ICML-14), 1764-1772, 2014.

27. A. Graves, N. Jaitly, A. R. Mohamed, Hybrid speech recognition with deep bidirectional LSTM. In 2013 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), 273-278, December 2013.

28. A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates A. Y. Ng (2014). Deep speech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567, 2014.

29. H. Xu, G. Chen, D. Povey, S. Khudanpur, Modeling phonetic context with non-random forests for speech recognition. In Sixteenth Annual Conference of the International Speech Communication Association, 2015.

30. T. Ko, V. Peddinti, D. Povey, S. Khudanpur, Audio augmentation for speech recognition. In INTERSPEECH, 3586-3589, 2015.

31. G. Chen, H. Xu, M. Wu, D. Povey, S. Khudanpur, Pronunciation and silence probability modeling for ASR. In Sixteenth Annual Conference of the International Speech Communication Association, 2015.

32. Y. Gaur, F. Metze, J. P. Bigham, Manipulating Word Lattices to Incorporate Human Corrections. In INTERSPEECH, 3062-3065, 2016.

33. K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, On the properties of neural machine translation: Encoder-decoder approaches, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, 2014.

34. D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, Y. Bengio, End-toend attention-based large vocabulary speech recognition. In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 4945-4949, March 2016.

35. S. Hochreiter, J. Schmidhuber, Long Short-Term Memory. Neural Comput. 9, 8, 1735-1780, November 1997.
36. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. Technical report, arXiv preprint arXiv:1409.0473, 2014.
37. A. Coates, and V. Rao. “Speech Recognition and Deep Learning.” cs.stanford.edu/~acoates/ba_dls_speech2016.pdf.
38. D. Kingma, J. Ba, Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.
39. Reduced TED-LIUM release 2 corpus (11.7 GB), http://www.cs.ndsu.nodak.edu/_siludwig/data/TEDLIUM release2.zip, last retrieved July 2017.
40. Speech recognition performance, https://en.wikipedia.org/wiki/Speech recognition#Performance, last retrieved July 2017.
41. Levenshtein distance, https://en.wikipedia.org/wiki/Levenshtein distance, last retrieved July 2017.
42. A. C. Morris, V. Maier, P. Green, From WER and RIL to MER and WIL: improved evaluation measures for connected speech recognition. In Eighth International Conference on Spoken Language Processing, 2004.
43. Word error rate, https://en.wikipedia.org/wiki/Word error rate, last retrieved July 2017.
44. A. Marzal, E. Vidal, Computation of normalized edit distance and applications, IEEE transactions on pattern analysis and machine intelligence, 15(9), 926-932, 1993.