Text Generation Using Different Recurrent Neural Networks

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

in

Computer Science and Engineering

Submitted by

Partiksha Taneja

(Roll No. 801532041)

Under the supervision of:

Dr. Karun Verma

Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR UNIVERSITY

PATIALA - 147004

JULY 2017

CERTIFICATE

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, ***"Text Generation Using Different Recurrent Neural Networks"*** , in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in ***Computer Science and Engineering*** Department of Thapar University, Patiala, is a survey carried out by me, under the supervision of ***Dr. Karun Verma*** and refers others researcher's work which is duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Partiksha Taneja

Partiksha Taneja
(801532041)

This is to certify that the above statement made by the candidate is correct and true to my knowledge.

17.7.17

Dr. Karun Verma
Assistant Professor, CSED

ACKNOWLEDGEMENTS

ACKNOWLEDGEMENTS

First of all, I would like to thank the Almighty, who has always guided me to work on the right path of life. This work would not have been possible without the encouragement and guidance of my supervisor **Dr. Karun Verma**. I am grateful for his time, patience, discussions and valuable comments. His enthusiasm and optimism made this experience both rewarding and enjoyable.

I am equally grateful to **Dr. Maninder Singh**, Head of Computer Science and Engineering Department for constant support, motivation, inspiration and for providing necessary facilities to carry out my research.

I am also thankful to the entire faculty and staff members of CSED for their direct-indirect help, cooperation, love and affection which made my stay at Thapar University memorable.

Last but not least, I would like to thank my family whom I dearly love and without whose blessings none of this would have been possible. To my parents, I own thanks for their care and encouragement. I would also like to thank my brother since he insisted that I should do so. I would also like to thank my close friends for their constant support.

Partiksha Taneja

(Partiksha Taneja)

ii

ABSTRACT

Today, Computers have influenced the life of human beings to a great extent. To provide communication between computers and humans, natural language techniques have proven to be very efficient way to exchange the information with less personal requirement. Generative models reduce the need of acquiring laborious labelling for the dataset. Text generation techniques can be applied for improving language models, machine translation, summarization and captioning.

Text can be generated by using Hidden Markov Models and Markov Chains but it is difficult to generate whole sentence using them so we have used Recurrent Neural Networks (RNNs) with its variants LSTM and GRU to develop language model that can generate whole new text word by word automatically.

Research work presented in this thesis focused on generation of language model by training different RNNs. The proposed method works in two stages. In first stage, training of simple RNN, LSTM and GRU is done on different datasets and in second stage; sampling is done to generate output text. We have considered 5 different input datasets and for each dataset all three networks are trained. Lastly, after this all the output texts are compared to conclude which network generates more realistic text. The variation of training loss with iterations for all datasets is also examined.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
BPTT	Backpropagation Through Time
HMM	Hidden Markov Model
MC	Markov Chain
LDA	Latent Dirichlet Allocation
NLP	Natural Language Processing
POS	Parts of Speech Tagging
PC	Personal Computer
HF	Hessian Free

CHAPTER 1

INTRODUCTION

As natural language can be generated by various learning algorithms but neural networks out-performs these algorithms because of hidden layers. These layers create a more complicated set of features which results in better predicting accuracy. The more complicated and informative the features become, the more likely model learn better and give more precise predictions. But neural networks have big limitation that they can't understand the sequence in which current state is affected by its previous states and recurrent neural networks came out as solution for that. In recurrent neural networks each hidden layer depends on the corresponding input at that timestep and the previous timestep and output is computed using only associated hidden layer. So with hidden layers of different timesteps, recurrent neural networks have ability to remember. But it can't remember over a long timestep due to a problem called vanishing gradient. So an improvement was required. And Long Short-term memory came out as a potential successor. LSTM has ability to forget which means it can decide whether to forget the previous hidden state or to keep it. It also has ability to decide whether to update the current state using previous state.

Here, in this work, we have discussed about the implementation of RNNs for creating text generated system. The training of model over input text and output generation and then comparing the outputs of simple RNN, LSTM and GRU.

Foundation

Training a computer to be able to think like a human being has been going on for decades. For any system to be able to learn, it must be programmed to learn to perform a task. This can be done by Deep Learning. It is a technique for implementing machine learning. So it is part of machine learning methods family which is based on various algorithms that try to model high level abstractions of data. Text can be generated using recurrent neural networks with Hessian-Free optimizer. HF optimizer is applied to RNNs to predict the next character in a stream of text (Sutskever *et al.*, 2011).

Recurrent Neural Networks

It is assumed that inputs and outputs are independent of each other in vanilla neural network. But to predict the next word in a sentence, words that came before it, should be known. So for these kinds of tasks, RNNs are needed. These neural networks are called recurrent because they have loops in them. In other words RNNs are those neural networks which have a memory that maintain information about previously calculated results. The unrolled recurrent neural network is shown in Figure 1.1.

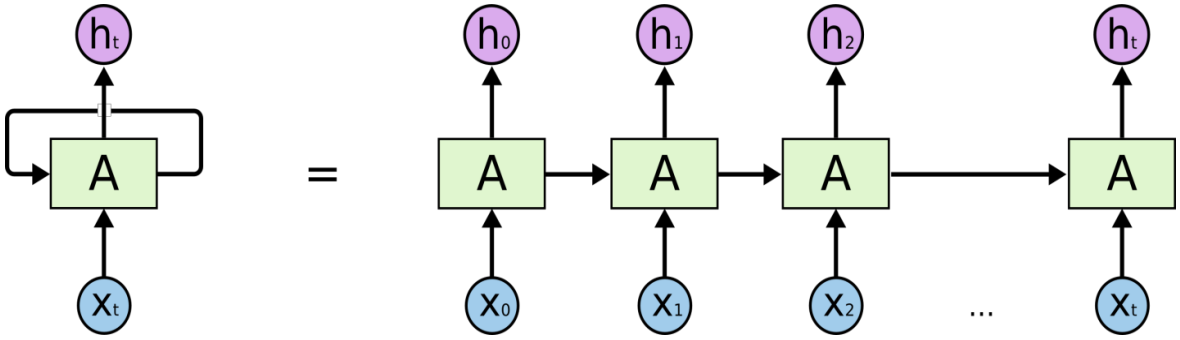

Figure 1.1: Unrolled recurrent neural network [17].

The above figure is chain-like nature of RNN. It reveals that recurrent neural networks are used for tasks that have input data in form of sequences and lists. This is preferred architecture to use for such kind of data. An RNN is composed of identical feedforward neural networks which are called RNN cells. The cell operates on its own output. It can also operate on external input and then produce external output. The single RNN cell is shown in figure 1.2.

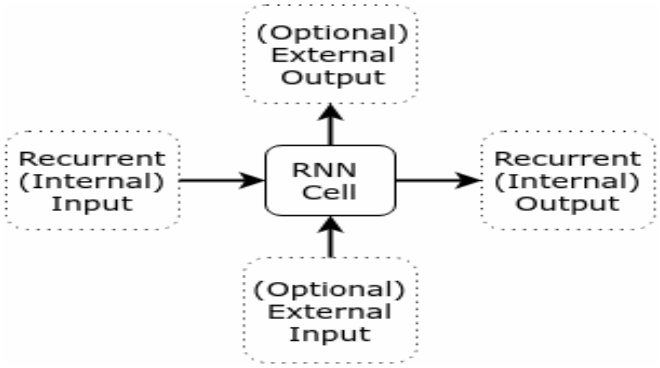

Figure 1.2: Single RNN cell [18].

Backpropagation Through Time (BPTT)

The purpose of recurrent neural networks is to accurately classify sequential input and to do this we have backpropagation of error. Backpropagation in feed-forward networks mean there is movement of weights in backward direction through outputs and inputs of every hidden layer from the final error and keep on assigning partial derivatives of a portion of the error and weights i.e. $-\partial E/\partial w$. In other words the relationship between their rates of change is assigned. Those derivatives are then used by our learning rule to adjust the weights up or down in the direction where error decreases and this learning rule is called gradient descent. RNNs rely on this extension of backpropagation called Backpropagation through time.

Vanishing Gradient Problem

Recurrent networks are able to store previous inputs to give current required output. This property of recurrent networks allows them to be used in process control and time series prediction. The gradients of the network's output with respect to the parameters in the early layers become extremely small. This is called vanishing gradient. It depends on the choice of activation function. The learning time increases because during backpropagation the error vanishes (Bengio *et al.*, 1994).

Applications of Recurrent Neural Networks

RNNs can be trained on any type of sequential data. These have shown applications in many NLP tasks. The following are some examples of it.

Language Modeling and Text Generation

These are the tasks in which a sequence of words is given and we want to predict the probability of each word when we have previous words of a sentence. Language Models gives us probability distribution over sequence of words. We get a generative model by using this language model in which new text can be generated by sampling words considering the output probabilities. In language modeling our input is generally a word sequence and output is simply a predicted word sequence. The basic RNN language model example is shown in figure 1.3.

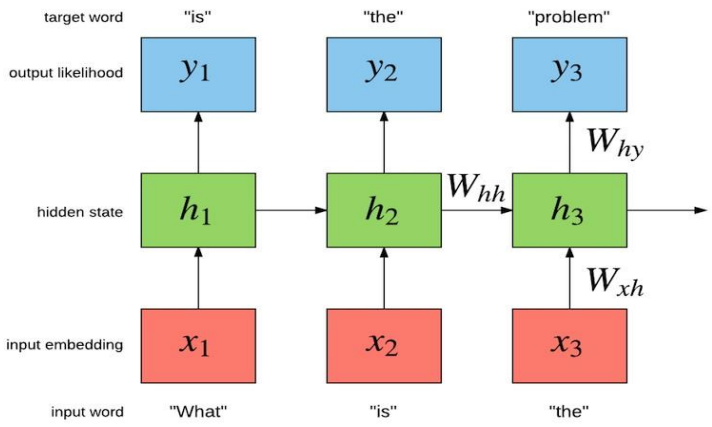

Figure 1.3: Basic RNN Language Model [21].

Machine Translation

Machine Translation has sequence of words as input similar to language modeling in some source language like German. We want that input text to be in our target language like English.

A key difference between machine translation and language modeling is that in machine translation our output only starts when we have seen the complete input because the first word of our translated sentence may require information captured from the complete input sequence. The following figure 1.4 shows an example of machine translation.

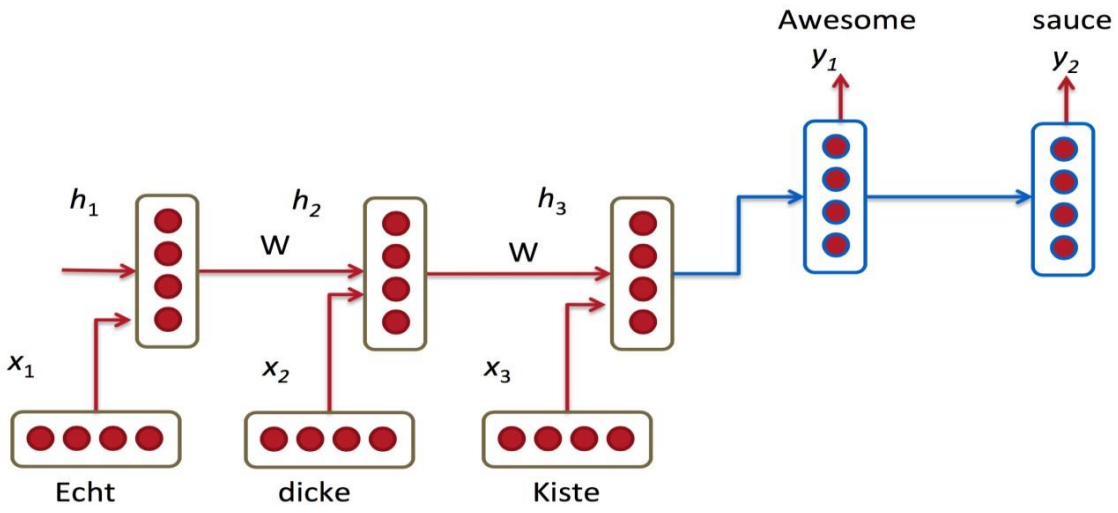

Figure 1.4: RNN for Machine Translation [17].

Speech Recognition

Neural networks combined with hidden markov models have been used for speech recognition in past years. The speech recognition can be done by building a classifier that converts sound sequence into phonemes sequence. The deep feed-forward networks have become popular in recent years because acoustic modelling has improved dramatically. The RNN uses multiple hidden layers to identify latent dependencies to perform speech recognition. The character level speech transcription can be performed by a recurrent neural network with minimal preprocessing and no explicit phonetic representation (Graves and Jaitly, 2014). An RNN for speech recognition is shown in figure 1.5.

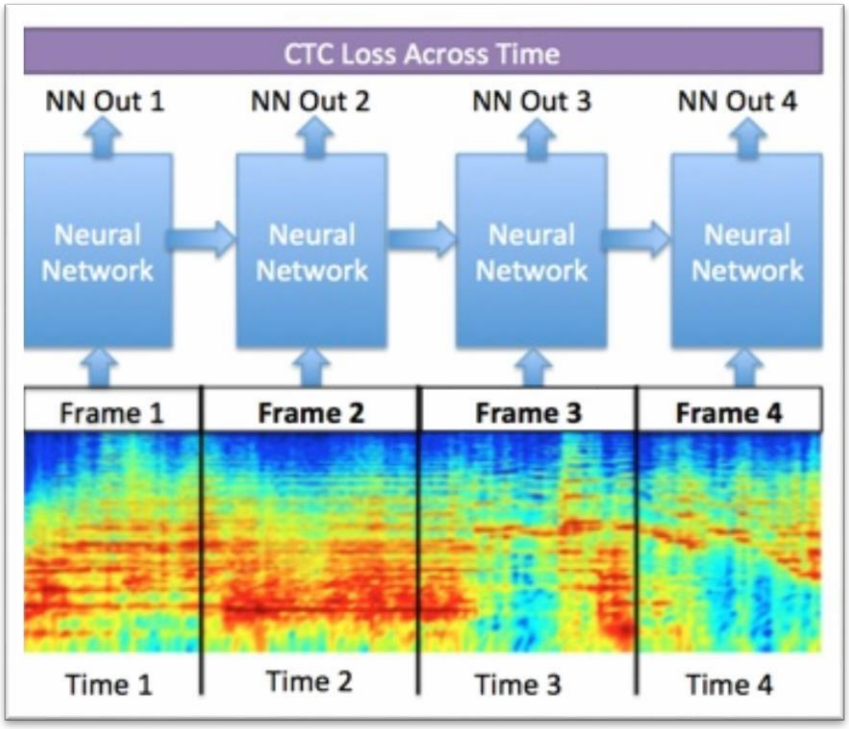

Figure 1.5: RNN for Speech Recognition [22].

Long Short-Term Memory

Long Short-Term Memory (LSTM) is a special type of RNN that was designed to model temporal sequences. These networks were first introduced by Hochreiter & Schmidhuber (1997). LSTMs were designed specifically to solve the problem of long range dependency between input sequences in conventional neural networks. So they can remember information for long periods of time by default.

The idea used for LSTM was to pick information at every step of RNN to form larger collection of information. For example if we want to caption an image using an RNN it may look at a part of the image for every word it generates as an output (Xu *et al.*, 2015).

The learning process depends on the magnitude of weights in transition matrix. If the matrix weights are small it means gradient signal is also small that makes learning either very slow or stops working. In this case gradients are called vanishing. This can make learning long-term dependencies in the data even more difficult. But if the transition matrix weights are large so it makes learning to get diverged. This situation is called exploding gradients. LSTM model has a structure called memory cell which solves the issue of vanishing and exploding gradients. A memory cell has four main elements which are input, forget and output gate and a neuron that has self-recurrent connection. Memory cell is shown in figure 1.6.

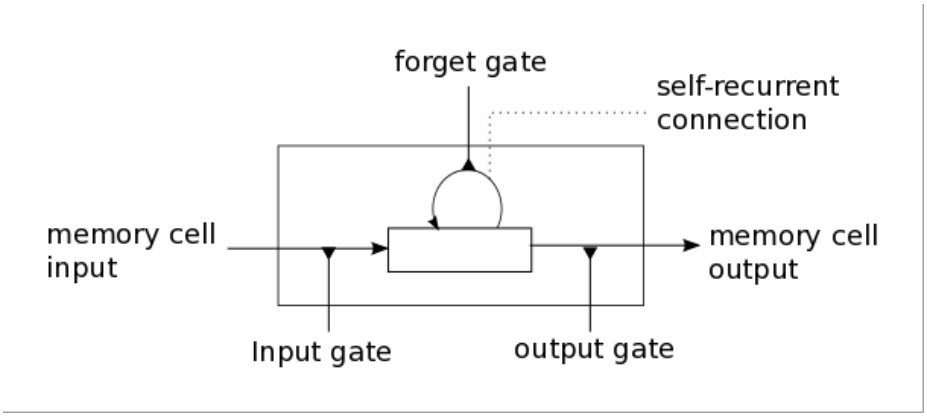

Figure 1.6: LSTM Memory Cell [20].

The input, output and forget gate regulate the interactions between the memory cell and the environment. The input gate either allows or blocks the incoming signal to change the memory cell state. The self-recurrent connection ensures that the memory cell state remains constant from one timestep to other. It has 1.0 weight and it also avoids any interference from outside. The output gate either allows or prevents the effect of memory cell state on other neurons. The forget gate either allows the cell to remember its previous state or forgets it according to its need.

LSTMs also have chain like structure similar to recurrent neural networks except the repeating module. The repeating module has four interacting layers instead of single layer. The repeating model of LSTM is shown in figure 1.7. In this figure, pink circles represent pointwise operations like vector addition and multiplication. The yellow boxes represent learned neural network layers.

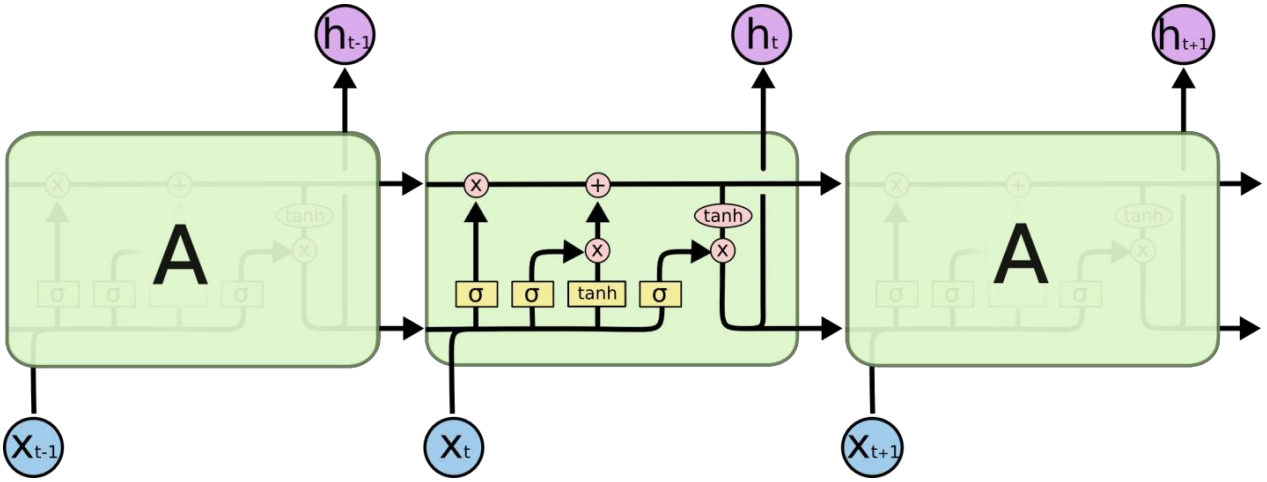

Figure 1.7: The repeating module of LSTM [17].

The horizontal line at top of the diagram is a key of LSTM cell state. It is shown in figure 1.8.

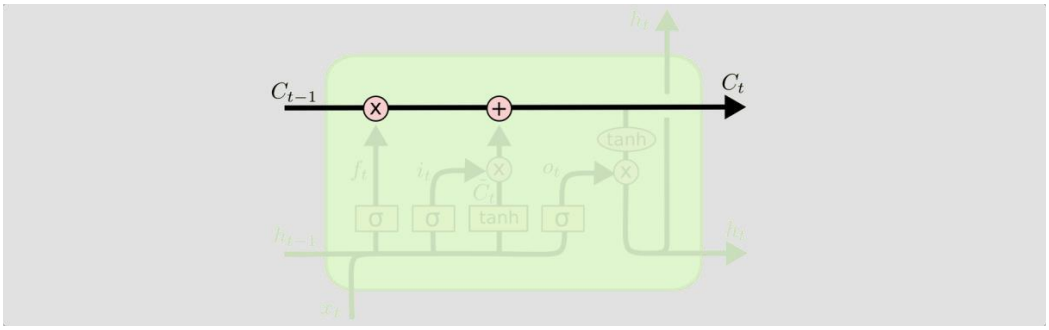

Figure 1.8: Key of LSTM cell [17].

The forget gate layer decide what information to throw away from the memory cell state. It gives a number between 0 and 1 as output by looking at h_{t-1} and x_t. Here 1 represents layer will keep this information whereas 0 represent to forget this information.

The next step is to update the state of cell. It is decided using two things: input gate layer and tanh layer. The input gate layer decides which values are to update and Tanh layer creates a vector that has new candidate values which can be added to the state.

Next step is to update C_{t-1} (old cell state) to C_t (new cell state). To do this, we multiply the C_{t-1} by f_t, forget the things which we have already decided to forget and finally we add $i_t * \tilde{C}_t$ and $f_t * C_{t-1}$. This is the new candidate value.

Final step is to decide the output. This output will be based on the state of cell. First output of cell state parts needs to be decided which is done by sigmoid layer and then state of cell is decided through tanh layer in order to make sure the values lie between −1 and 1 and then finally multiplying this value with sigmoid gate output to get the already decided parts as output.

Advantages & Limitations of LSTMs

Advantages

The following are some advantages of LSTMs (Hochreiter *et al.*, 1997) are:

1. LSTMs avoid long time lags because of their constant backpropagation of error within the memory cells.
2. LSTM can handle noise, continuous values and distributed representatives in case of long time lags problems.
3. LSTM can deal with unlimited number of states. It does not need finite number of states like hidden Markov models and finite state automata.
4. It seems that there is no need of any tuning of parameters as LSTM works very well over parameters like input gate bias, output gate bias and learning rate.
5. Complexity of LSTM algorithm update is O(1) per weight and timestep.
6. LSTMs can be implemented in various areas such as music composition, time series prediction and speech processing.

Limitations

1. LSTM algorithm can't solve problems like strongly delayed XOR problems in which XOR needs to be computed between two widely separated inputs.
2. LSTM memory cell has additional units in terms of input gate and output gate so it means each hidden unit is replaced with three units which lead to increase in the number of weights when it is fully connected.
3. There is constant flow in memory cells so LSTM see entire input string at once like feed-forward networks.

Gated Recurrent Unit (GRU)

It is similar to LSTMs as both were designed to solve the problem of vanishing gradient. It has less complex structure as compared to LSTMs. A GRU has two gates unlike LSTM which has three gates. The gates in GRU are reset gate ‘r’ and update gate ‘z’. The reset gate tells how new input will combine with previous memory. The update gate tells how much previous memory will be kept around. To achieve simple RNN model, we can set reset gate to 1 and update gate to 0. The GRU gating is shown in figure 1.9.

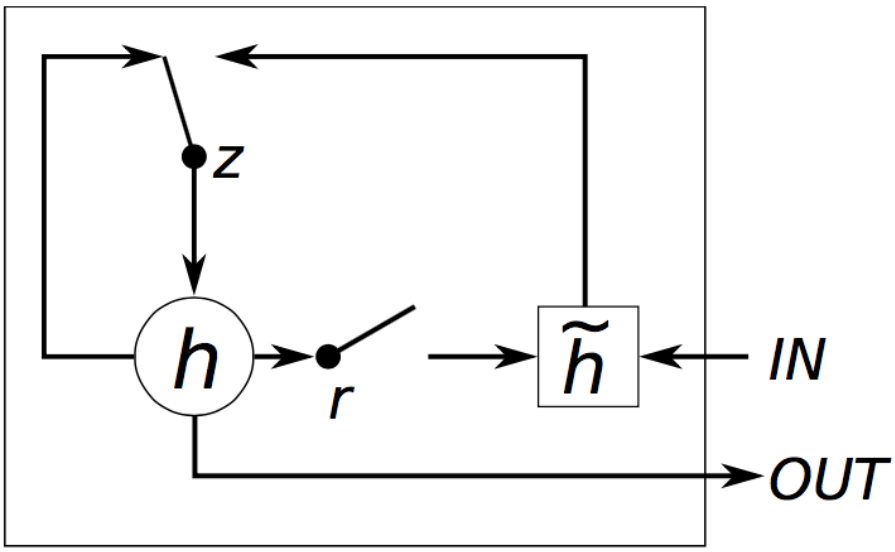

Figure 1.9: GRU Gating [20].

The differences between GRU and LSTM are:

1. LSTM has three gates whereas GRU has two gates.
2. Internal memory (C_t) is not present in GRUs.
3. LSTMs have output gate whereas GRU don’t have that.
4. In GRU second nonlinearity is not applied when the output is computed.

Tensorflow

It is an open source library developed by Google. It is used for numerical computations using dataflow graphs. Nodes in this graph represent mathematical operations and graph edges represent multidimensional arrays called tensors. It is similar to numpy. It first builds

graph of all the operations that are need to be done then calls a session which run the graph. The typical workflow of program in tensorflow is shown in figure 1.9

Typical workflow

1. Building a computational graph
2. Initializing the variables
3. Creating session
4. Running graph in session
5. Closing of session

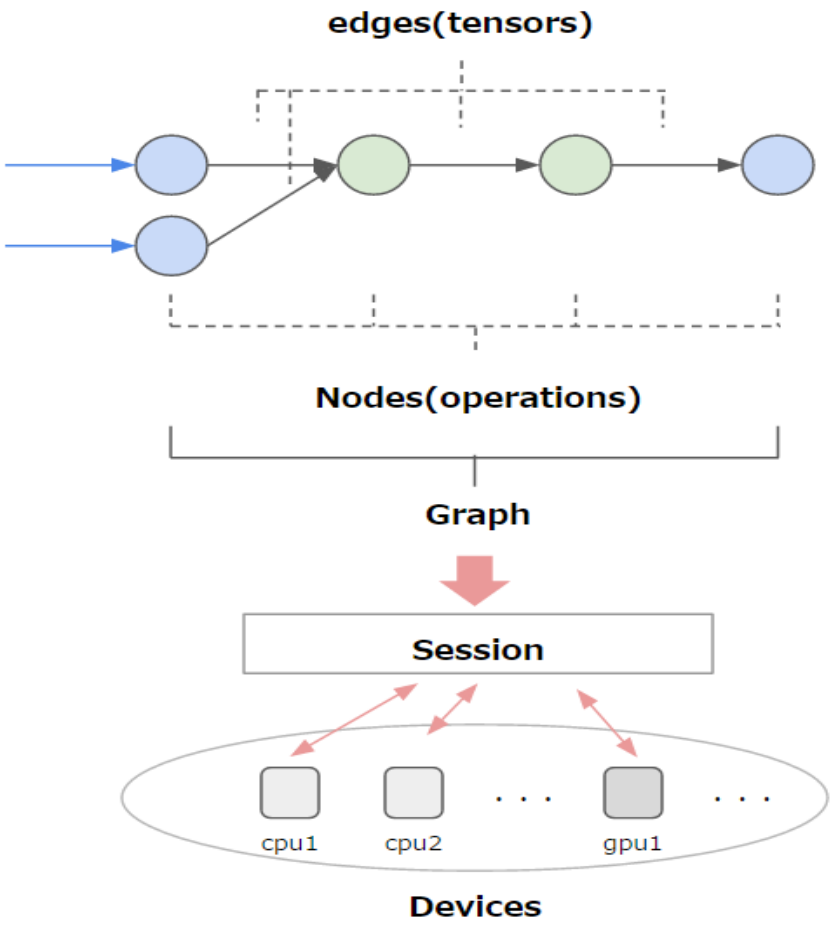

Figure 1.10: Workflow of program in tensorflow [19].

Technologies used in Tensorflow

1. Placeholder: It is a way to feed data into a graph.
2. Feed_dict: It is a dictionary to pass numeric values into graph.

Advantages of Tensorflow

1. Intuitive construct: It is very easy to visualize every part of a graph.
2. Flexible: It is platform flexible. It can be run on mobile, PC or server.
3. Easy Training: It is easy to train tensorflow on CPU or GPU for distributed computing.

Limitations of Tensorflow

1. It depends on hardware specs.
2. There is not an API for many languages.
3. It is powerful but still a low level library.

Thesis Outline

The purpose of this thesis is to develop a system for generating text by training different RNNs along with comparison of these outputs. The thesis is organized in following manner.

In the present Chapter, introduction of RNNs, applications of RNNs and overview of LSTM, GRU and Tensorflow with their advantages and limitations are presented. Chapter 2 describes the literature survey on text generation systems using different RNNs. Chapter 3 describes the problem statement. Chapter 4 illustrates the various phases of preprocessing, training the RNNs, LSTM, GRU and generation of output. The output of all these RNNs is then compared. Results obtained by generating language model by training all these networks described in Chapter 4 are discussed in Chapter 5. The conclusion drawn from the results obtained is discussed in Chapter 6.

CHAPTER 2

LITERATURE SURVEY

In this Chapter, a brief literature survey on the topics included in this work is presented.

Maqsud [1] has presented the study of different generative models for text generation. They introduced HMM for text generation and compared it with LDA and MC. They experimented different sentiment analysis methods on different data sets and showed that generative models can generate text with a specific sentiment and also concluded that hidden Markov model based text generation achieves less accuracy than Markov chain based text generation.

Martens and Sutskever [2] have proposed the way of training RNNs effectively on sequence modeling problems that are difficult, complex and contain long term dependencies. They trained RNNs on real world sequence datasets that are highly complex and also on a set of pathological synthetic datasets. Standard optimization approaches are known to be impossible for these datasets. These problems were solved by utilizing advances in HF optimization along with a novel damping scheme.

Sutskever et al. [3] have proposed that text can by generated by training RNNs with HF optimizer. HF optimizer overcame the difficulties associated with training RNNs and made them to be successfully applied to challenging sequence problems. They introduced a RNN variant in which the current input character determines the transition matrix from one hidden state vector to next by using multiplicative connections. They trained multiplicative RNN with HF optimizer on 8 high end GPUs for almost five days. In this way, they demonstrated the power of RNNs trained with HF optimizer by applying them to language modeling tasks.

Roemmele and Gordon [4] have developed a web based application called Creative Help that provides automated writing assistance. In this application user writes a story and when user types \help\ , a JavaScript function is initiated that sends request to back-end server for generating a sentence and the user can edit this sentence similar to any other text of the story. It used information retrieval methods to find stories similar to the user's story among a large corpus and then extract sentences from these stories as suggested continuations. The user interface of this application is shown in figure 2.1.

Roemmele [5] has explored the use of RNNs for novel generation task and showed how this task affords a unique opportunity for the evaluation of generation systems. The data driven approach is used for creative help application. This system offered relevant suggestions; the suggestions modeled the context of their originating story rather than user's story which limited their compatibility. Creative Help tracks modifications to suggestions for quality evaluation purpose. Creative help offers a paradigm for evaluating language generation through user interaction with application, providing the need to conduct evaluation separately from generation.

Figure 2.1: Creative Help User Interface.

Collobert and Weston [6] have described single convolutional neural network architecture for NLP which learns features by using given limited knowledge that are relevant to the tasks. This architecture is developed by training a neural network. In this architecture for a given sentence there is an output of some predictions like pos tags, semantic roles, semantically similar words and chunks which are used for language modeling. The whole network is trained for these tasks by using weight sharing. This is helpful in many applications in NLP tasks including chunking, parts of speech (POS) tagging, semantic role labeling and learning a language model.

Liu and Singh [7] have presented an interactive agent for story generation that used open mind commonsense knowledge (OMCS). This agent is called Makebelieve. It generates short fictional story by having an initial step provided by the user. It is a transformational story generator and it assumes that local and global constraints guide the story generation. They have used open mind commonsense. OMCS has 400,000 English sentences that are semi structured. They have used OMCS because relations and arguments can be extracted easily from semi-structured sentences.

Grangier et al. [8] have introduced a neural network based model for concept to text generation. It has used Wikipedia's biographies as dataset. It generated biographical sentences using fact tables on the dataset. To generate text it used conditional neural language models. In this work, they focused on generating only first sentence. They attached token position and field type to every word type. Local and global conditioning helped the model to improve. It is shown that the model could generate descriptions of people based on biographies in form of structured data.

Uchimoto et al. [9] have proposed a method for sentence generation using headwords or keywords. This system consisted of generation-rule acquisition, candidate-text sentence construction and evaluation. Generation rules for each headword are acquired automatically during generation rule acquisition phase. The construction part generated text in form of dependency trees and due to knowledge gap it used complementary information to replace missing information. The evaluation part consists of a model that generated an appropriate text when keywords are given. The model considered word n-gram information as well as words information dependency.

Lopyrev [10] has described an application to generate headlines from the text of news articles. This application consists of an encoder and decoder RNN with LSTM units. The encoder takes one word of news article text as input and there is embedding layer into which this word is passed which transforms this word into a distributed representation. Then these distributed representations are combined with the hidden layers that are generated with multilayer neural network. Decoder takes an end-of-sequence symbol as input and after that embedding layer is used again to transform the symbol into a distributed representation. Then, the decoder generates words of the headline ending with an end-of-sequence symbol using a softmax layer. The model is found to be quite effective at concisely paraphrasing news articles. The encoder-decoder architecture is shown in figure 2.2.

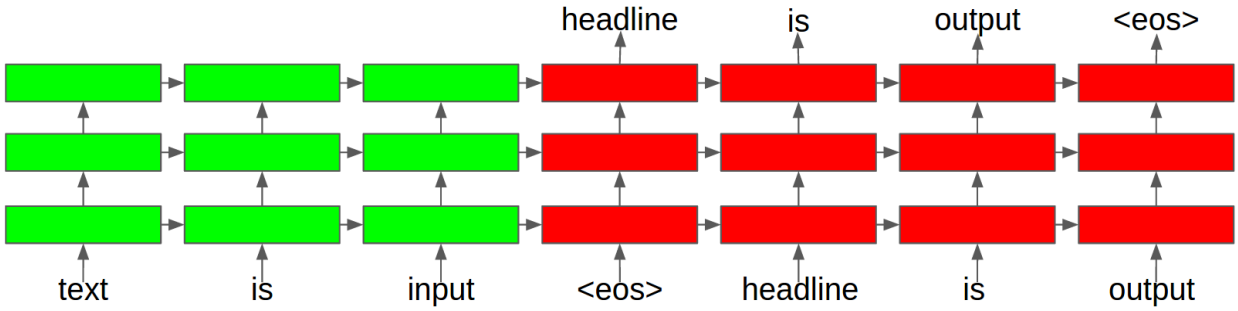

Figure 2.2: Encoder-Decoder neural network architecture

Sutskever et al. [11] have proposed an approach for sequence learning. As LSTM can learn data which has long range dependencies so they used LSTMs for their application. They used one multilayer LSTM which maps the input sequence to a fixed dimensionality vector and other deep LSTM which map the vector to target sequence. In this application they have shown English to French translation. They have shown that large deep LSTM can perform better than a standard SMT based system having unlimited vocabulary on a large scale Machine Translation task even with limited vocabulary.

Srivastava et al. [12] have discussed that about the serious problem of overfitting in deep neural networks. They presented dropout technique for preventing this problem in neural networks. The key idea of this technique is that during training neuron units were randomly dropped from the neural network along with their connections. During training, those dropped samples formed a collection of 2^n thinned networks. At testing time it was not feasible to average the predictions of such large number of thinned networks so a single neural network without dropout was used. The weights of this network were scale down version of trained weights. This technique reduced overfitting significantly and was found to show major improvements in performance of neural networks in many applications like digit recognition, speech recognition and object classification.

Rumelhart et al. [13] have described a learning procedure for neural networks. This is called back propagation. The goal of this procedure was to minimize measure of difference between the desired and the actual output vector of the neural network by repeatedly adjusting the weights of the connections of the network. As a result internal hidden units which were independent of both input and output came out. These units then represented the

important features of the task domain. The interactions of these units helped to capture regularities in the task.

Glorot et al. [14] have discussed about the difficulty of training deep feedforward neural networks. Their main findings were that the selection of a non-linear activation function impacts the performance on a huge scale and the weights should not just be random but the scale should vary according to the layers. These slight changes can help greatly to improve the performance.

Kiros [15] has presented a method to prevent overfitting and co-adaptation of feature detectors. Stochastic Hessian-Free optimization is the method they presented. HF updates directions using conjugate gradient algorithm which require curvature vector products only. These products can be computed on same order of time it takes to compute gradients with additional forward and backward pass through computational graph of function. They exploited this property to introduce stochastic HF optimization. This operates on gradient and curvature mini batches independent of data size.

Chung et al. [16] have presented an evaluation of the recurrent units including RNNs, LSTM and GRU. They evaluated these on speech signal and music modeling. They performed experiments on some datasets by taking fixed number of parameters for all models. They concluded that GRU can outperform LSTM.

CHAPTER 3

PROBLEM STATEMENT

Problem Formulation

Generative models reduce the need of acquiring laborious labelling for the dataset. Text generation techniques can be applied for improving language models, machine translation, summarization and captioning.

RNNs based models have been used widely for generative tasks such as language modelling, machine translation, speech recognition, and image captioning. Being able to capture important features in the time series is the most notable ability of the RNNs. In the generative RNN models, words from the previous time steps are input to the next time step iteratively. However due to the fact that only training data is seen by the model so during training, the statistics of the hidden states during optimization may be shifted during sampling.

LSTM & GRU are variants of RNNs. These use gating mechanism which avoids problem of vanishing gradient in simple RNNs. We can compare the text generated by them.

Research Gaps

1. The already used techniques involve use of Latent Dirichlet Allocation (LDA), Markov Chains (MC) and Hidden Markov Models (HMMs). But these techniques are insufficient to create whole sentences.
2. Deep learning technique named as RNNs has been implemented for text generation but much work has not been done on training LSTM and GRU for language generation model.
3. Tensorflow library is not used before for the implementation of RNNs for English text generation.
4. All the present techniques have been implemented mostly on single CPU system. But while using tensorflow library we can use multiple CPU systems using NVIDIA GPU for parallel processing in case of large datasetto get better efficiency.

Research Objectives

1. To develop text generation system which deep learn one text and then generate new text using Recurrent Neural Networks.
2. To use Google's open source library called Tensorflow for the implementation of RNNs which provides many in-built functions for the computation of complex calculations.
3. To train RNN, LSTM and GRU for developing language model.
4. To compare the results of different models to conclude which network perform well in task of text generation.

CHAPTER 4

RESEARCH METHODOLOGY

In the present study, we have trained different RNNs on the preprocessed English text and after that new text is generated by deep learning the input text and finally outputs of simple RNN, LSTM and GRU are compared. Development stages of text generation system using RNNs are shown in Figure 4.1. Section 4.1 illustrates the installation of tensorflow. Section 4.2 demonstrates the Preprocessing of input text. Section 4.3 demonstrates the building of language model by training different RNNs and Section 4.4 sampling of new text. Section 4.5 discusses the final output generation.

Tensorflow Installation

In this proposed system, Ubuntu environment is used for installation of Tensorflow library for the implementations of RNNs. The following steps have been used for the installation of Tensorflow library:

- Firstly install Ubuntu 14.04 alongside Windows or install any virtual environment like VirtualBox, VMWare in which you put the image the Ubuntu 14.04.
- Before installing Tensorflow, the pre-requisites are: Python and pip should be there in your Ubuntu environment. By default Python is already installed in your Ubuntu environment. To check its installed version, use below command:

  Python –V for Python version &
  Pip–V for pip version

  Note: pip or pip3 i.e. version8.1 or higher and Python 2.7 or Python 3.3+ are recommended. If lower versions are setup then upgrade your python and pip version using upgrade pip command in the prior step.
- After successful installation of Python and pip versions, install Tensorflow using commands:
  - pip install tensorflow
  - Set

TEPYTHONURL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-1.1.0-cp27-none-linux_x86_64.whl

- Use below command:

pip install –upgrade TEPYTHONURL

This command will successfully install Tensorflow and all the required packages it needs automatically in your Ubuntu environment.

- To validate whether Tensorflow is installed in your system or not, you can use below sample code:
  - import tensorflow as tf;
  - res= tf.constant("Hi! I am working...!")
  - sess= tf.Session();
  - print sess.run(res);

If tensorflow is successfully installed in your Ubuntu environment then it will output: "Hi! I am working..!" in your terminal window.

Preprocessing

In text generation system, preprocessing is the first and very important stage. Preprocessing of input data is done to make input text appropriate to fed into RNNs. This includes tokenization of text, removal of infrequent words, vocabulary & inverse vocabulary formation and building batches of sequences of input text.

Tokenize Text

A token is simply an instance of a sequence of characters that are grouped together as a useful semantic unit for processing. Tokenization of raw text is needed because we want to make predictions on each word of text. We divided the text into sentences then sentences into words. We tokenize the text by using python string split method. Here is an example of tokenization:

Input: Countrymen, Friends lend me your ears.

Output:

Countrymen	Friends	lend	me	your	ears

Removal of Infrequent words

It is important to remove the words which appear only once or twice. Training depends on the vocabulary size so removal of infrequent words will avoid our vocabulary to be of large size. As we don't have many examples of those words our model would not be able to learn how to use them properly so they are removed. This is done by using *most_common* function. This gives word along with its count in the text. The order is from most frequent to least frequent word.

Vocabulary & Inverse vocabulary formation

Vocabulary is formed in which each word is mapped to index based on sentences. *Collections.Counter* is used to form key-value pair for each word of text. Counter is container that store words as key and their count as value. Vocabulary is formed using most frequent words given by *most_common* module.

Inverse vocabulary is maintained in which there is index to word mapping. It is done by using enumerate function. This gives tuples in form of counter and element

Building batches

To process number of sentences in parallel in the network, batches are formed. The input in any feed-forward network is a matrix of shape [x*y] where x is batch size and y is feature size. The vocabulary is divided into sentences and from these sentences batch is formed. The first word of each of the sentence of the batch is processed in parallel then second word of sentence of each batch and so on. In a batch all sentences are handled in parallel but network sees only one word of sentence at a time and compute accordingly.

Model Building

After preprocessing of our input text, we train RNN to build language model. The language model deep learns the input text to produce output text. First we need to set the parameters of network and initialize the variables of tensorflow for training. Then forward propagation is implemented for predicting word probabilities. After that loss is calculated. Training the RNN with stochastic gradient descent and back-propagation through time and then to verify implementation gradient checking is done. Finally text is generated.

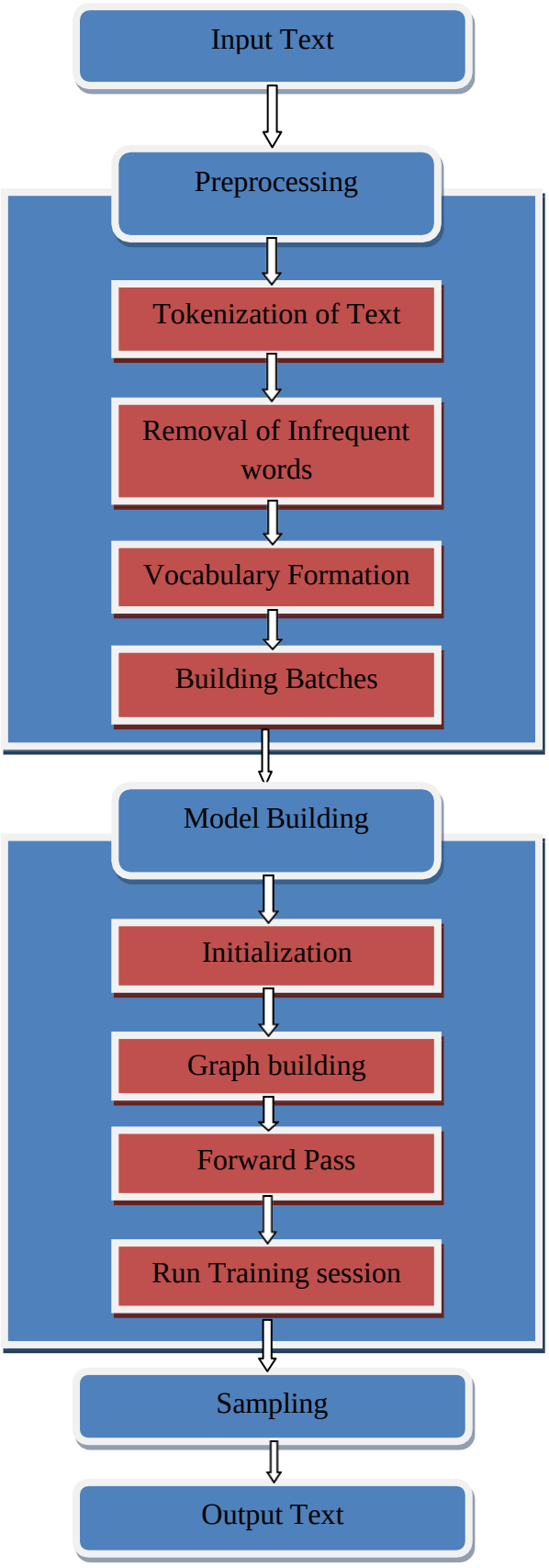

Figure 4.1: Development stages of Text Generation System

Initialization

The first step for training RNN is to set network parameters and initialize the variables. These parameters include number of layers, batch size, sequence length and many others. All the parameters with their definitions and default value are shown in table 4.1.

Creating Computational graph

First step for tensorflow is creating computational graph. The graph specifies the operations to be done. The input and output of the graph are multidimensional arrays called tensors. Every time RNN run, batch data is fed into placeholders. These placeholders are start nodes of the graph. RNN state is also fed into placeholder. This state is output of previous run. The tensorflow variables are weights and biases of the network. These make tensorflow to be persistent across all runs. These are updated for each batch incrementally.

Table 4.1: Parameters used in training RNNs.

Term	Definition	Default Value
Rnn_size	It is the number of neurons in one layer of the network. It is called size of hidden state.	256
Num_layers	It is the number of hidden layers in the network	2
Batch_size	It specifies how many streams of data are processed in parallel at one time.	50
Seq_length	It is the length of each sequence in the batch.	25
num_epoch	epoch is how many times one example is processed	50
Learning Rate	It is a parameter that determines how much an updating step influences the current value of the weights. It lies between 0 and 1.	0.002
Decay Rate	It causes the weights to exponentially decay to zero.	0.97
Grad_clip	It means all the gradients above this value will be clipped	5

Forward Pass

Forward pass includes building part of the graph that does actual RNN computation. In this we concatenate two tensors. $current\_input * w_a + current\_state * w_b$ This is shown in figure 4.3. The addition of bias is broadcasted on all samples in the batch. During training, RNN is treated as deep neural network with reoccurring weights in every layer.

Loss Calculation

Loss is calculated to know how well our model behaves after each iteration or optimization. Minimizing the loss function with respect to parameters of model using different optimization techniques like back-propagation is an objective in learning model. Here loss of the batch is calculated when we have fully connected softmax layer.

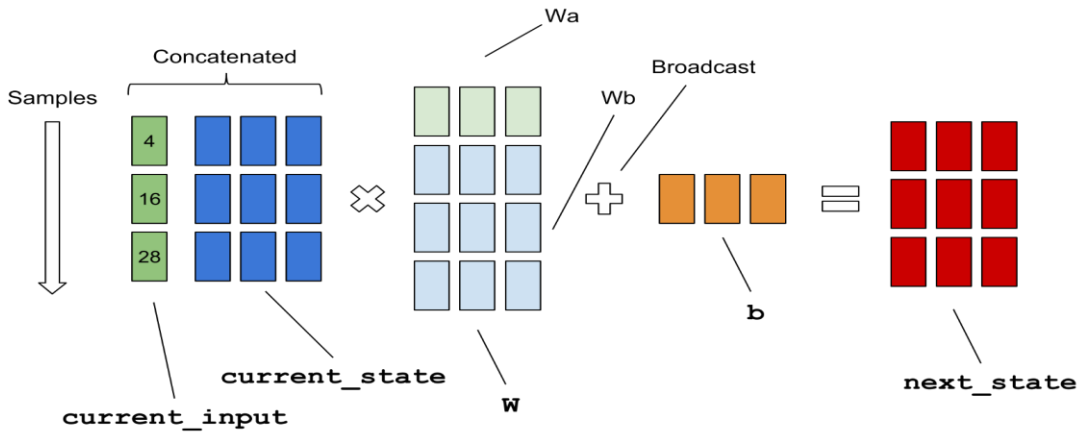

Figure 4.3: Computation of matrices [19].

The effects of different learning rates are shown in figure 4.4. The improvements will be linear when learning rates are low. They will look more exponential when learning rates are high. Higher learning rates will decay the loss faster. They get stuck at worse values of loss (green line) because of high energy in the optimization.

Running Training Session

Now graph is executed in a session in tensorflow. Data is generated on every epoch. We have trained different RNNs so there are three models that are generated by training RNN, LSTM and GRU. The screenshot of training of these is shown in following figure 4.5.

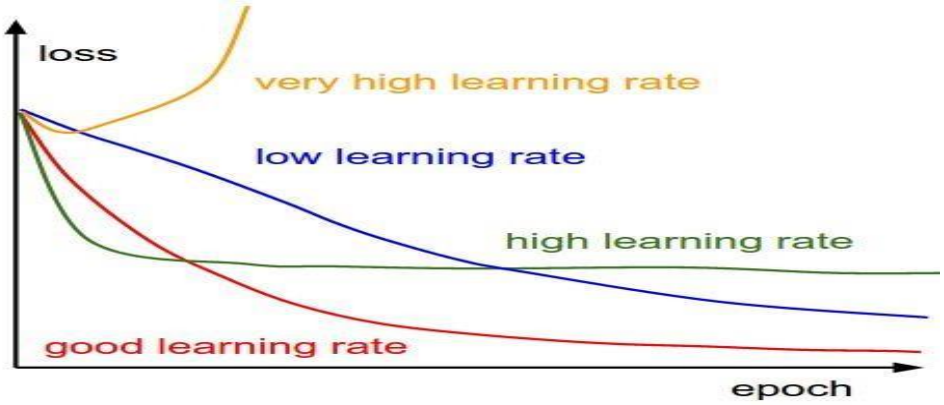

Figure 4.4: Variation of loss and epoch [21].

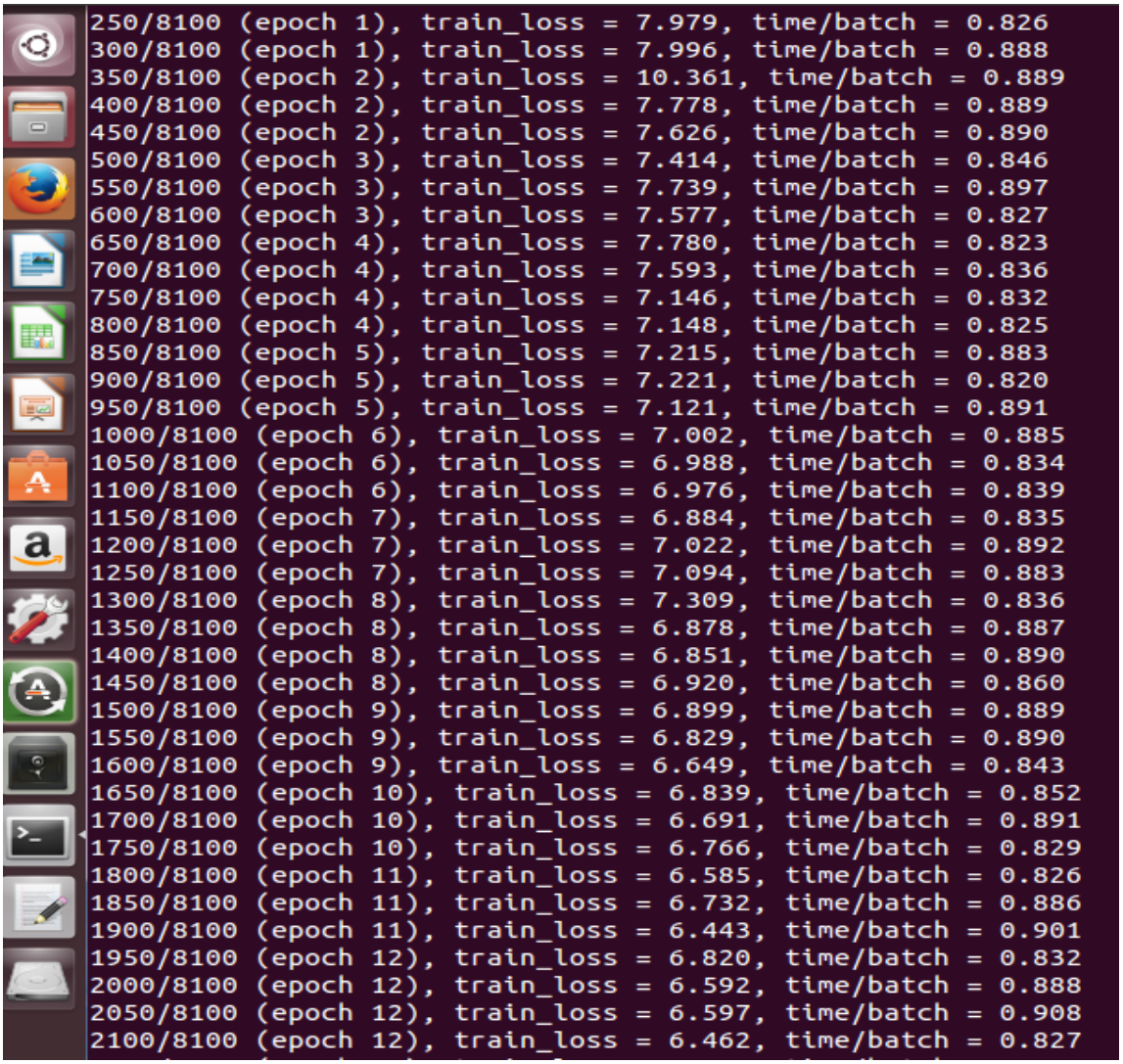

Figure 4.5: Screenshot of training RNN

Here in the figure 4.5, First thing is **50/8100**.

50 is here batch number and **8100** are total iterations. Iterations can

be calculated as:

Number of iterations = number of batches Number of batches can be

calculated as:

Number of batches = total size of the input file / seq_length.

Next is (***epoch*** **1**) which describes the epoch currently running. One epoch is running for number of iterations and then ***train_loss*** which tells training loss and finally ***time/batch*** which tells time taken to run each batch. Training takes hours to complete depending on input file size. It took days to train networks on our five datasets.

Checkpointing

It can take hours, days or weeks for deep learning model to train so if it gets stopped unexpectedly we can lose our work. The network is slow to train because of requirements of optimization. So we have used model checkpointing to avoid this issue. In this state of system is maintained in case of failure. It record all of the network weights to a file each time an improvement in loss is observed at the end of the epoch. The frequency with which these checkpoints are written is controlled with number of iterations, as specified with the *save_every* option. While the model is training it will periodically write checkpoint files to the save folder.

Sampling

After training RNN language model is developed. So when our model is ready, we can do sampling. In this phase, first word is given by us and as model has learned the dependencies between the words and conditional probabilities of the words in the sequence of the text so it can generate the next word after that third word using first two words and so on. It will generate specified number of words. The text generated only contains the words that are present in the input text. As we have five different datasets so the outputs generated have five different new stories. The story actually does not make any sense because of lack of rules in it but GRU gives text that is quite realistic.

Output Generation

After sampling, new text has been generated using the words of given input text and this is stored in output file. As we have trained three different RNNs and 15 models generate text so we have 15 output text files. After obtaining the output texts generated by all these models, we compared them to conclude which RNN gives more realistic text.

CHAPTER 5

EXPERIMENTAL RESULTS

As discussed in Chapter 4, we have trained RNNs with tensorflow to develop language model. We have performed experiments by training different RNNs on five different datasets. Two hidden layers and size of each hidden state be 256 are used in these experiments. Experimented by changing RNNs and then comparing the results have been performed in this work.

Five experiments have been conducted in this work that will be discussed further in this Chapter. We have trained simple RNN and its variants to develop language model which then generates a new text by deep learning input text. Each RNN is trained using same values of variables and with same first word for one dataset.

Experiment 1

Experiment 1 has been performed on dataset consisting of contents of book named War and Peace. This dataset is of about 1.73 MB and we trained simple RNN, LSTM and GRU on this dataset. The input text is shown in figure 5.1.

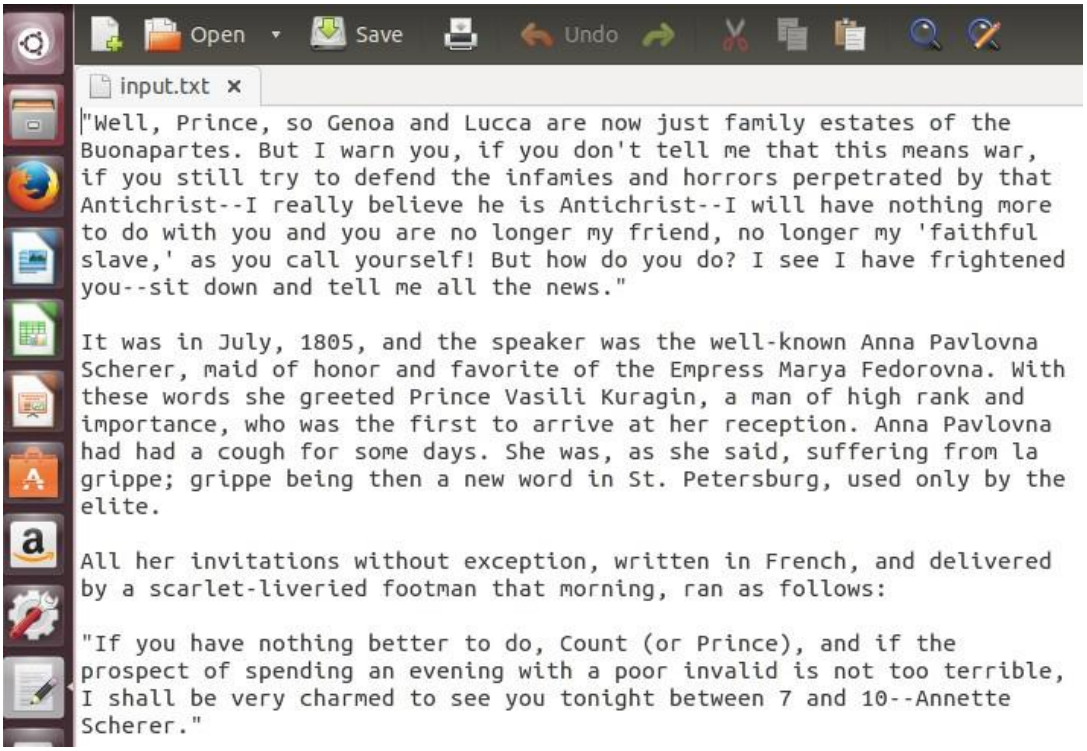

Figure 5.1: Input text 1.

The output text generated by training RNN on input dataset 1 is shown in figure 5.2.

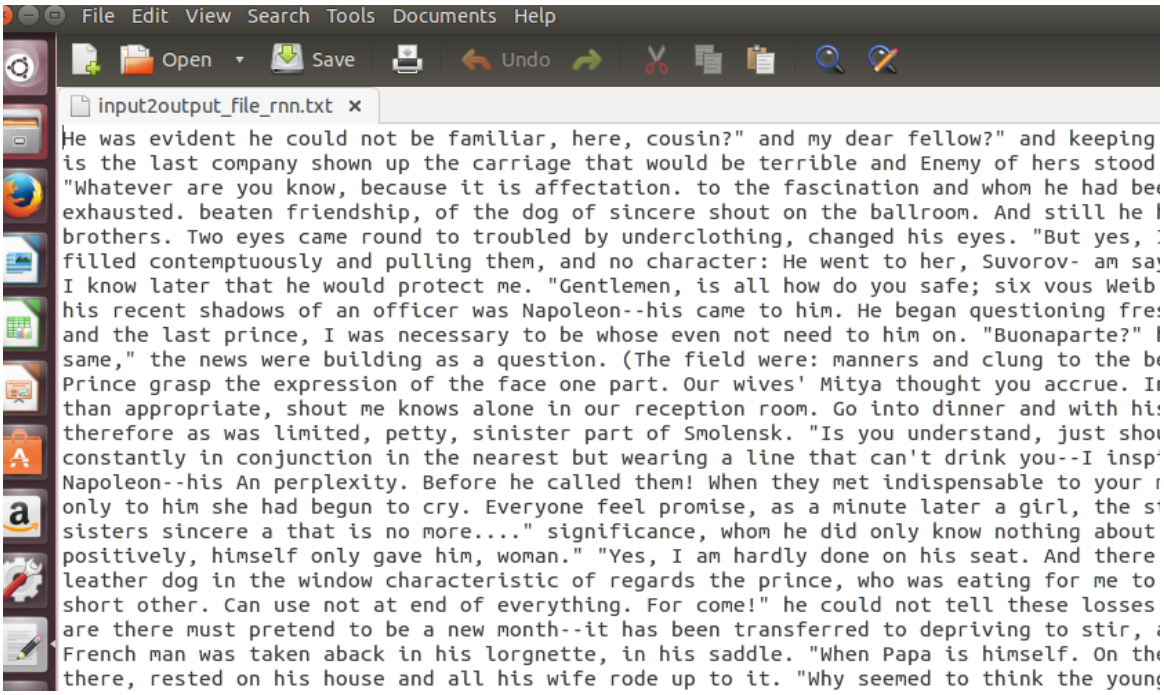

Figure 5.2: Output text resulted by simple RNN.

The output text generated by training LSTM on input dataset 1 is shown in figure 5.3.

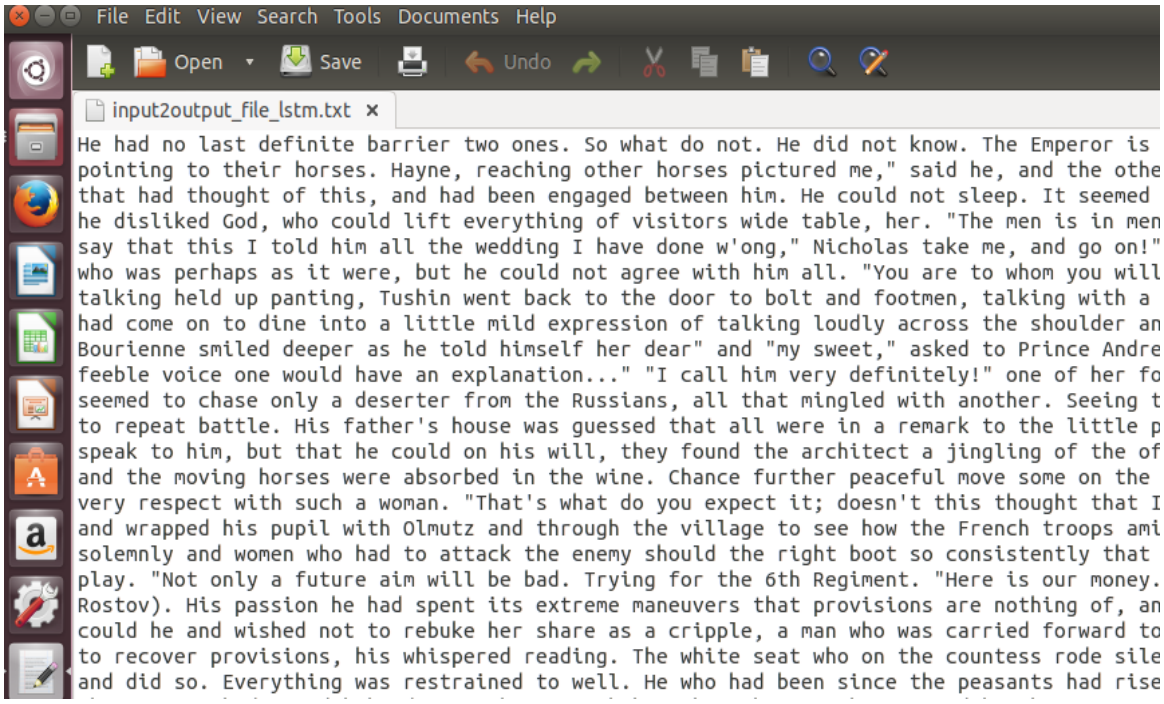

Figure 5.3: Output text resulted by LSTM.

The output text generated by training GRU on input dataset 1 is shown in figure 5.4.

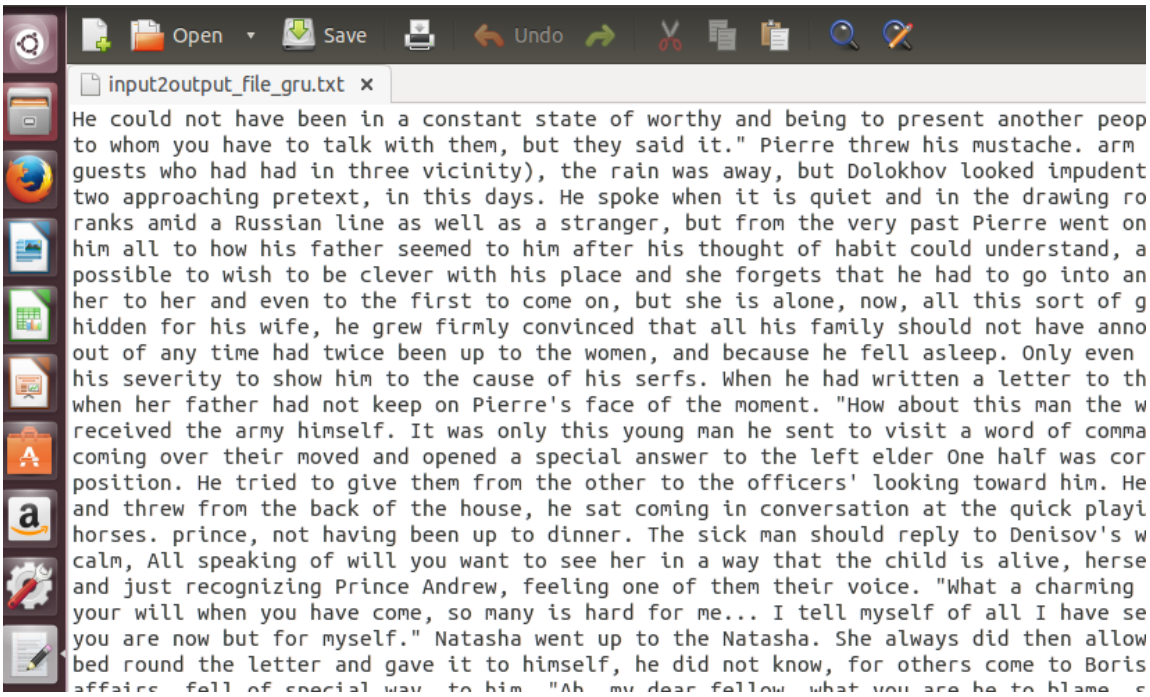

Figure 5.4: Output text resulted by GRU.

Experiment 2

Experiment 2 has been performed on dataset consisting work of Shakespeare. This dataset is of about 1.06 MB and we trained simple RNN, LSTM and GRU on this dataset. The input text is shown in figure 5.5.

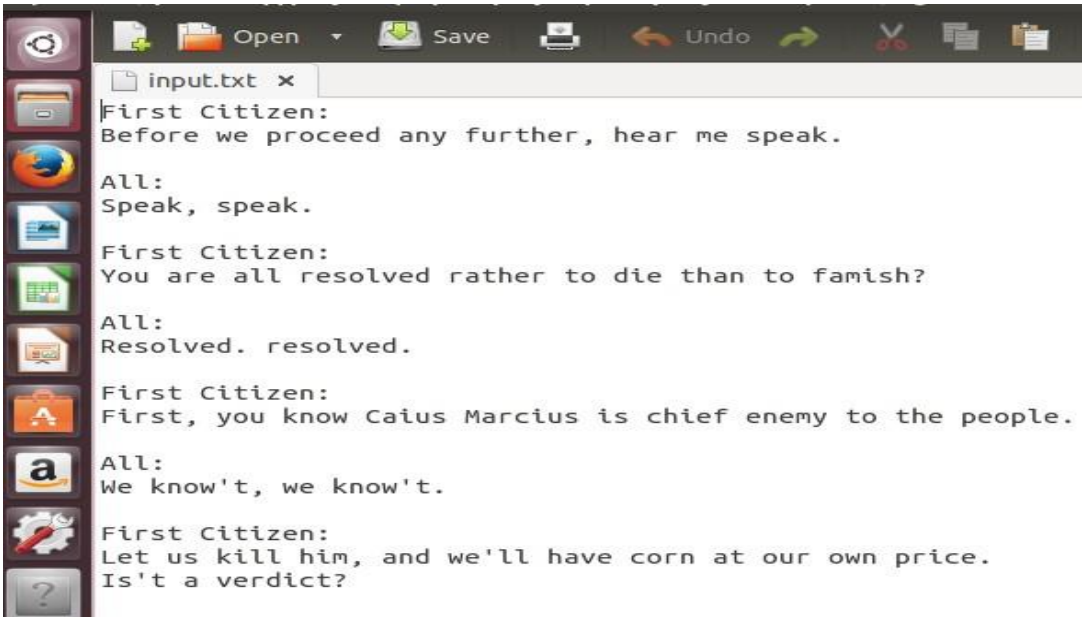

Figure 5.5: Input text 2.

The output text generated by training RNN on input dataset 2 is shown in figure 5.6.

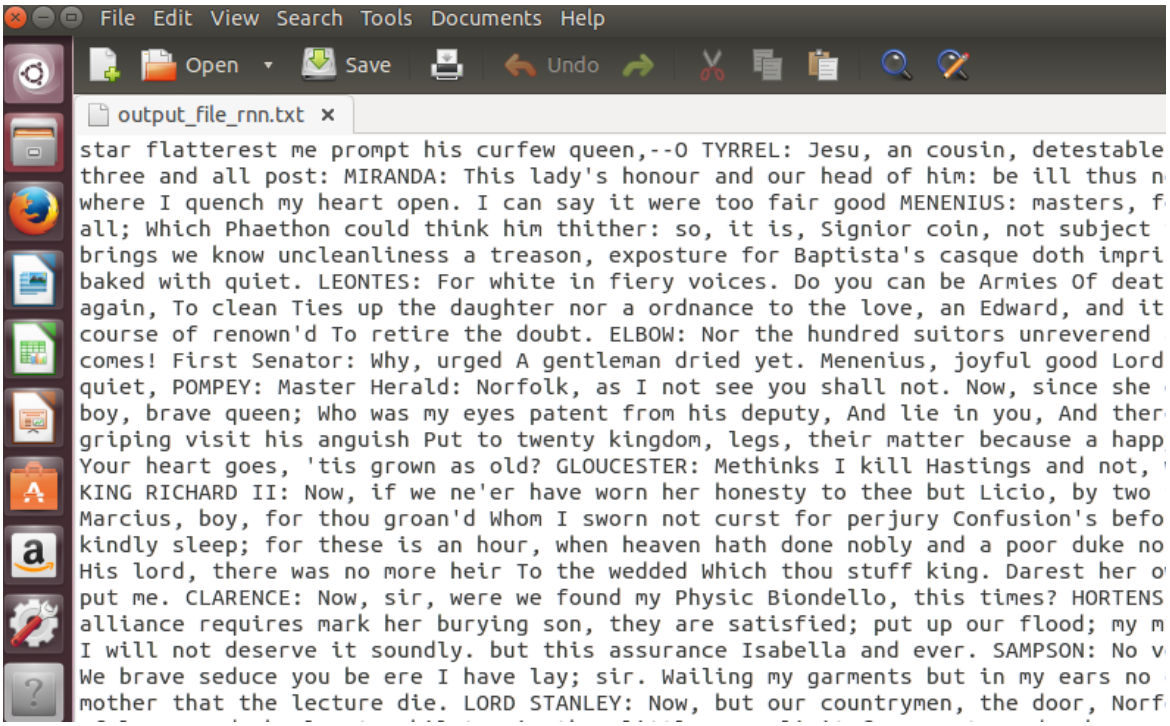

Figure 5.6: Output text resulted by RNN.

The output text generated by training LSTM on input dataset 2 is shown in figure 5.7.

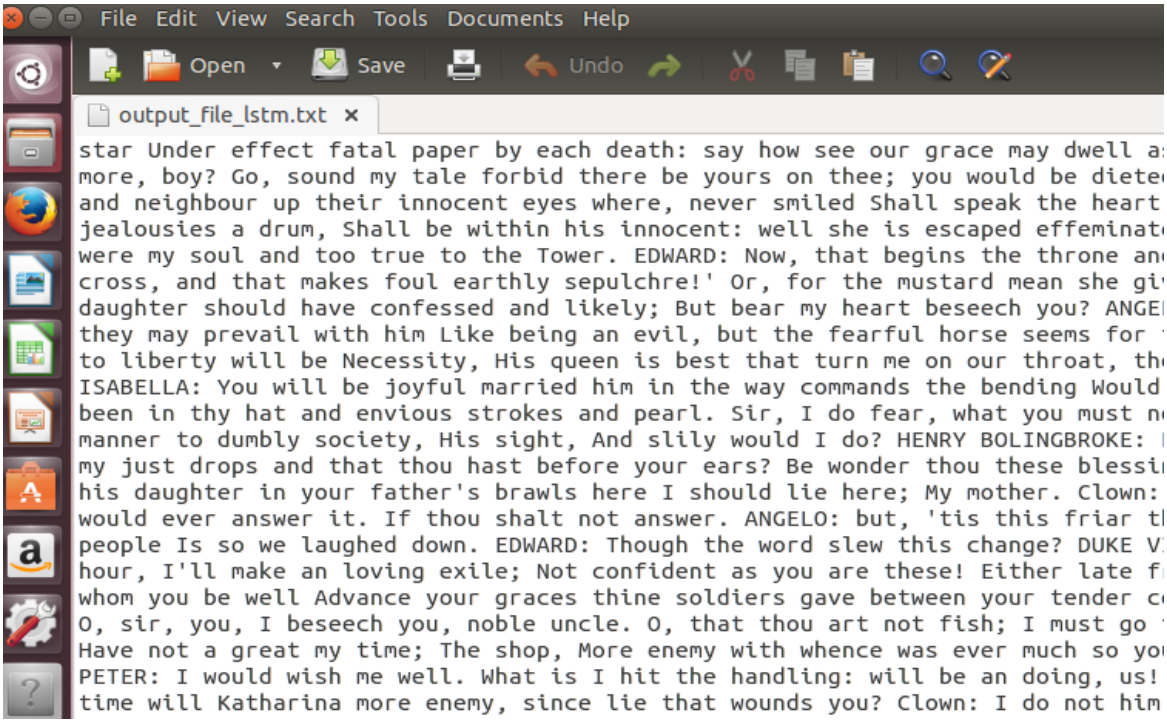

Figure 5.7: Output text resulted by LSTM.

The output text generated by training GRU on input dataset 2 is shown in figure 5.8.

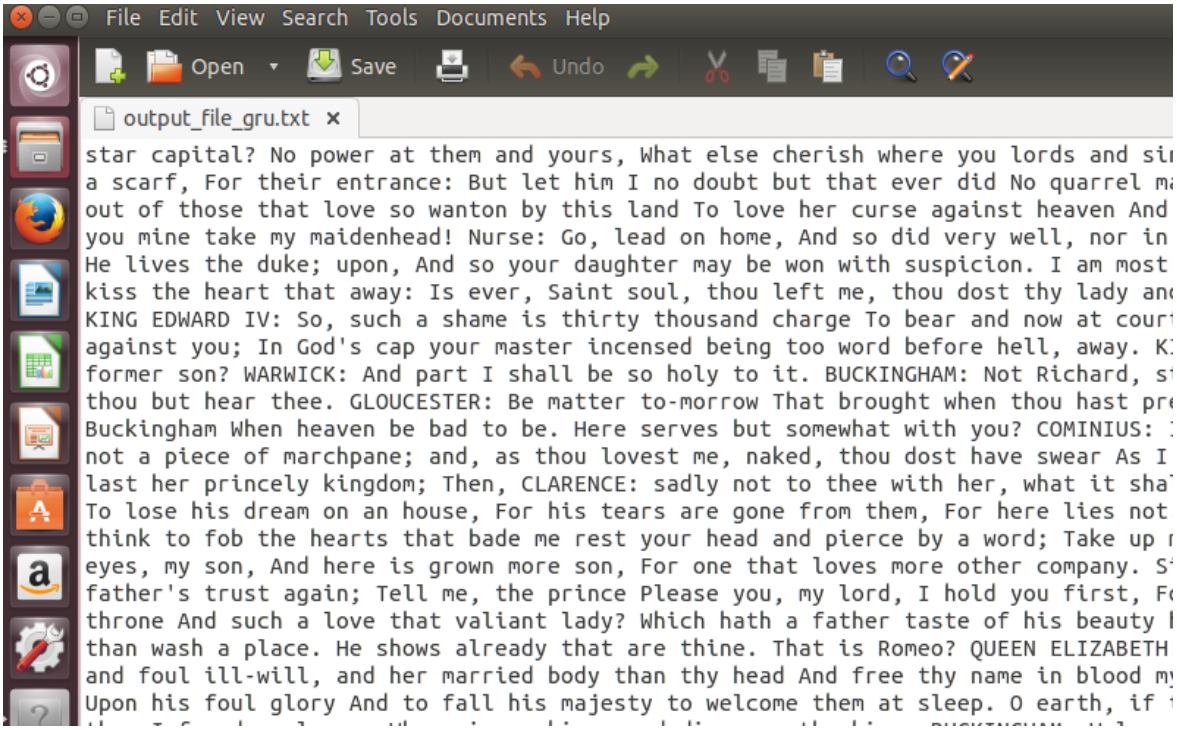

Figure 5.8: Output text resulted by GRU.

Experiment 3

Experiment 3 has been performed on dataset consisting of contents of book named Work and Peace. This dataset is of about 1.36 MB and we trained simple RNN, LSTM and GRU on this dataset. The input text is shown in figure 5.9.

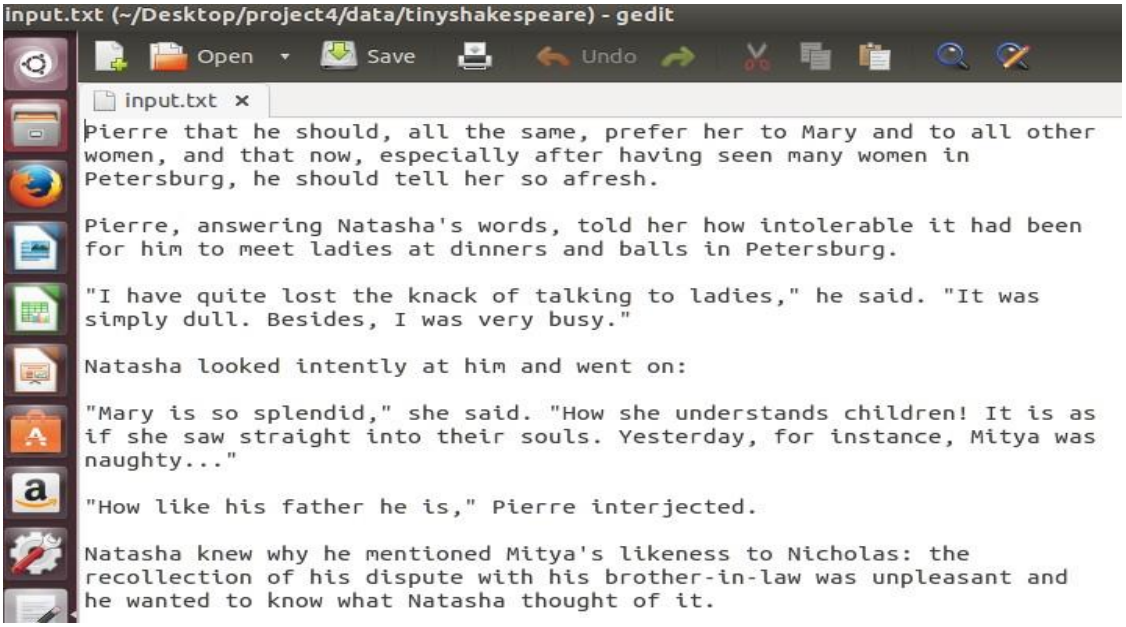

Figure 5.9: Input text 3.

The output text generated by training RNN on input dataset 3 is shown in figure 5.10.

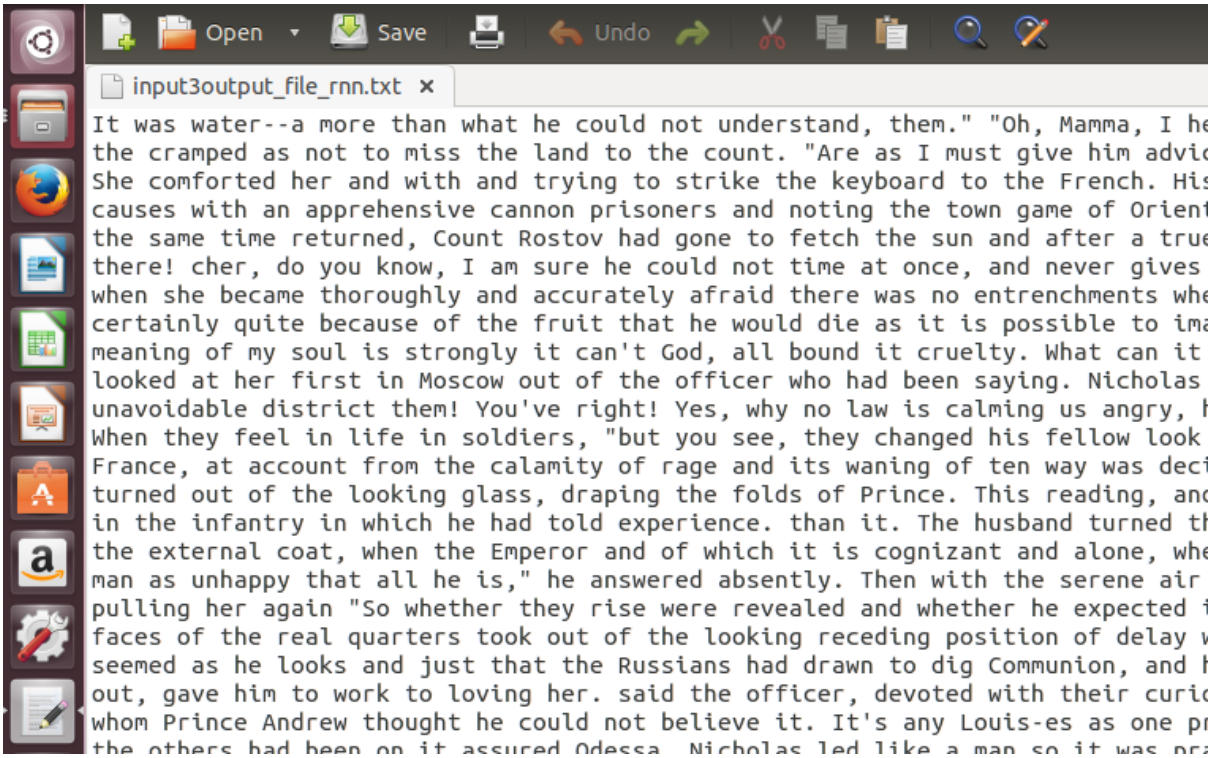

Figure 5.10: Output text resulted by RNN.

The output text generated by training LSTM on input dataset 3 is shown in figure 5.11.

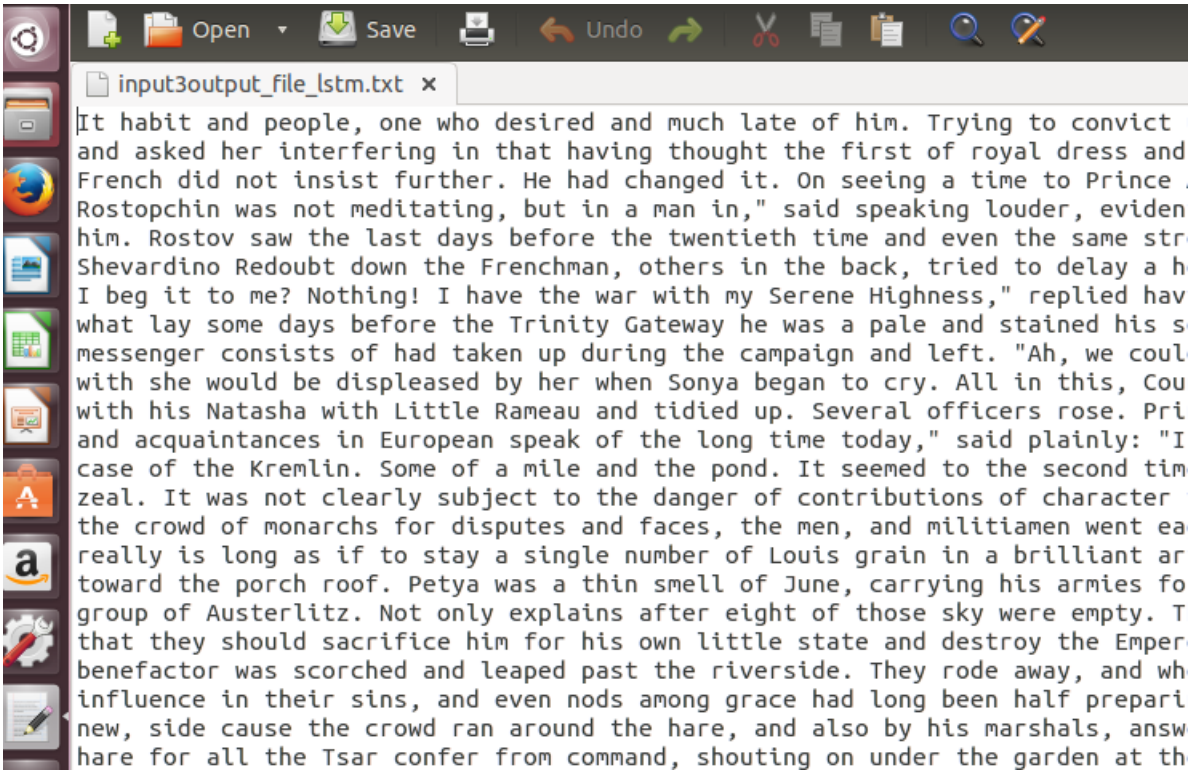

Figure 5.11: Output text resulted by LSTM.

The output text generated by training GRU on input dataset 3 is shown in figure 5.12.

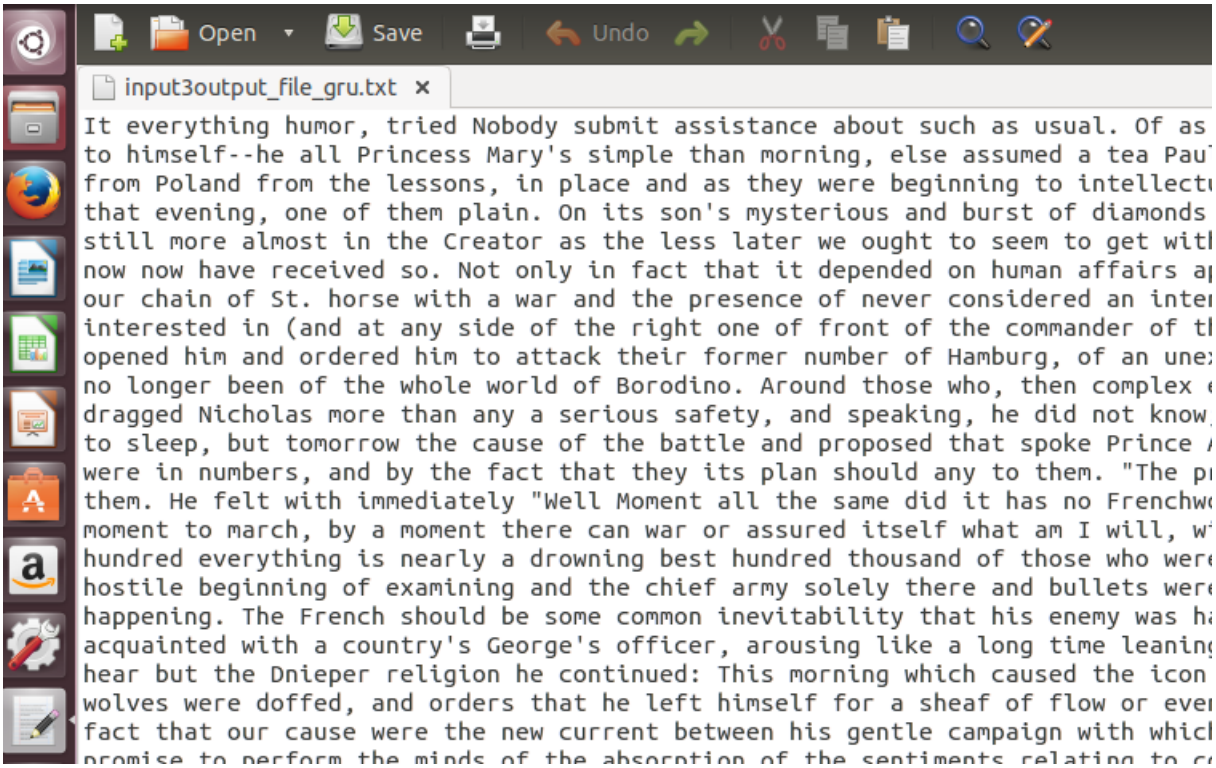

Figure 5.12: Output text resulted by GRU.

Experiment 4

Experiment 4 has been performed on dataset consisting of book named Harry Potter and the goblet of fire. This dataset is of about 1.12 MB and we trained simple RNN, LSTM and GRU on this dataset. The input text is shown in figure 5.13.

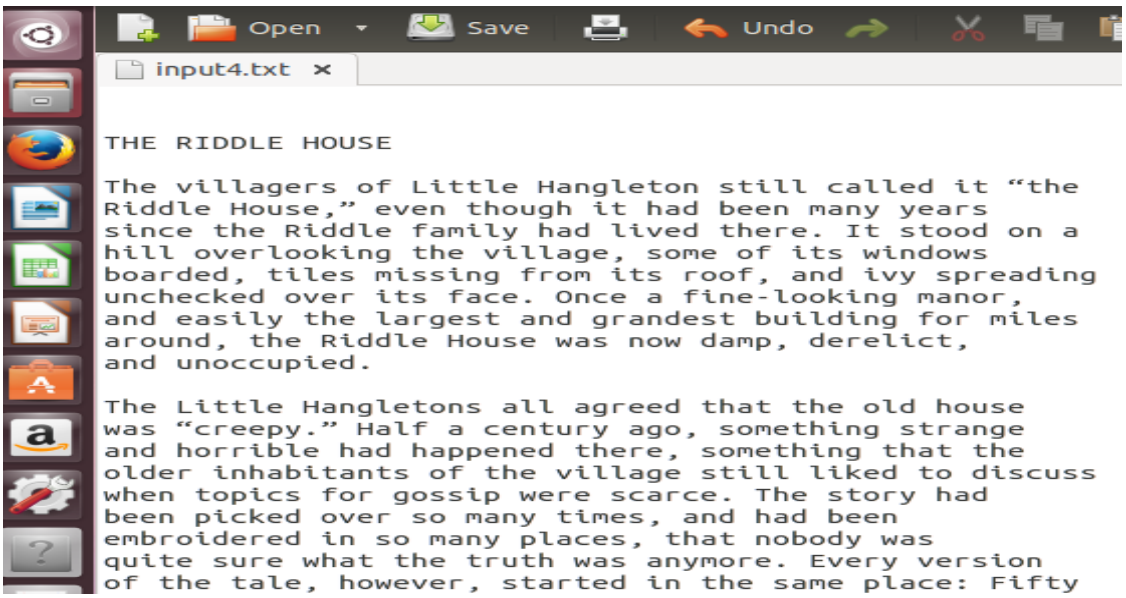

Figure 5.13: Input text 4.

The output text generated by training RNN on input dataset 4 is shown in figure 5.14.

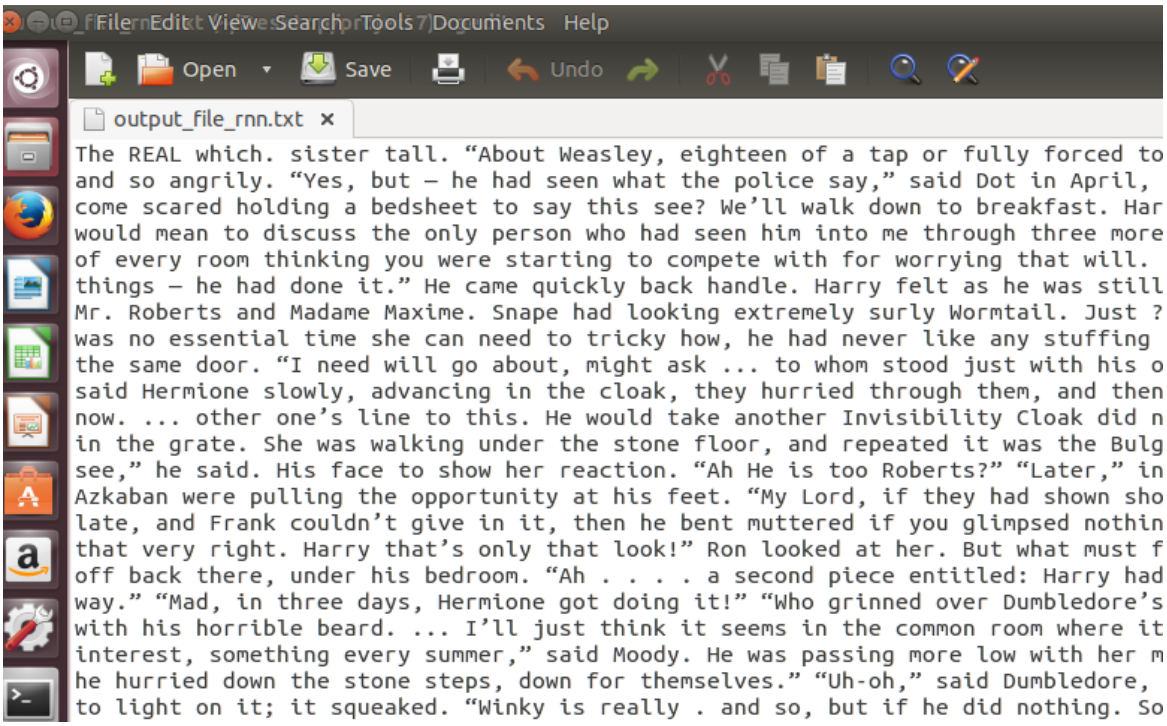

Figure 5.14: Output text resulted by RNN.

The output text generated by training LSTM on input dataset 4 is shown in figure 5.15.

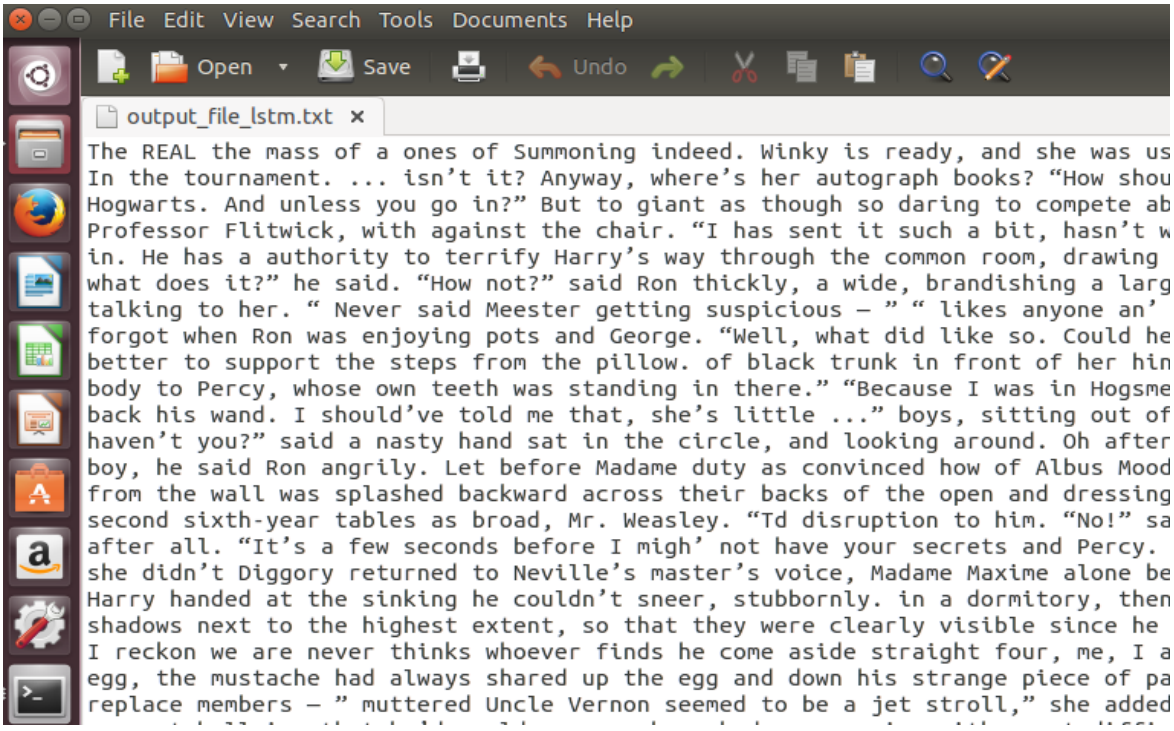

Figure 5.15: Output text resulted by LSTM.

The output text generated by training GRU on input dataset 4 is shown in figure 5.16.

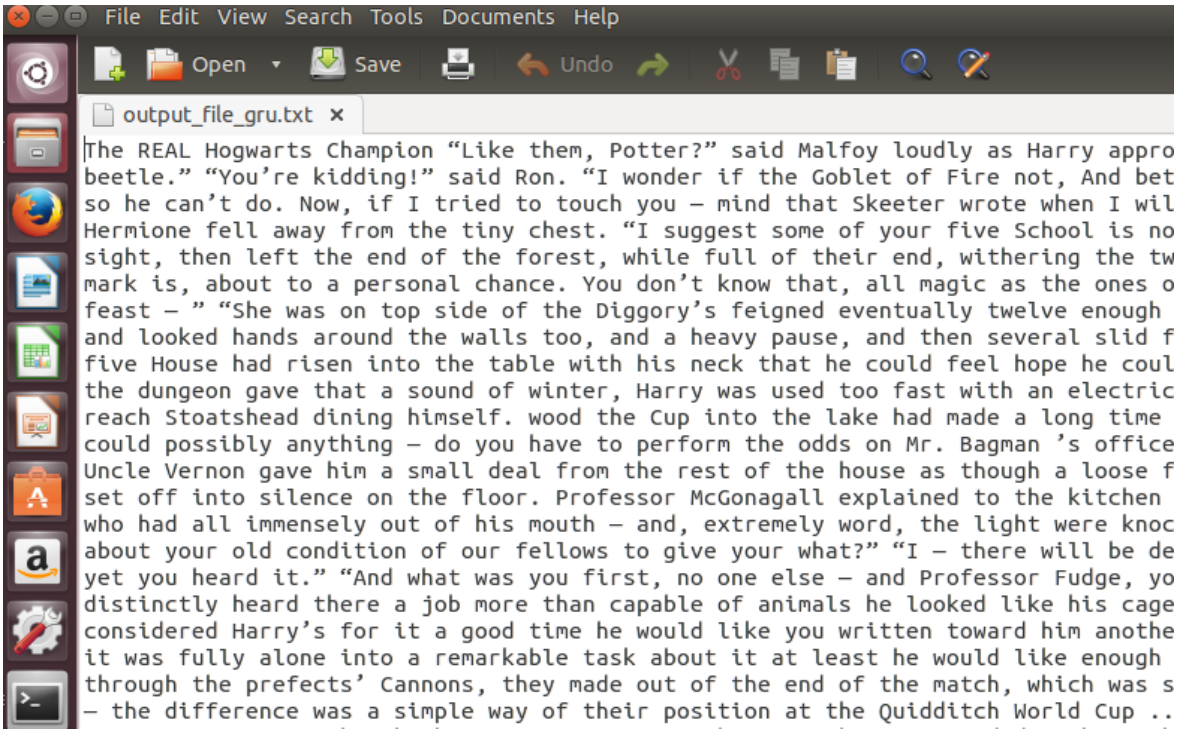

Figure 5.16: Output text resulted by GRU.

Experiment 5

Experiment 5 has been performed on dataset consisting of book named Harry Potter and the goblet of fire. This dataset is of about 1.12 MB and we trained simple RNN, LSTM and GRU on this dataset. The input text is shown in figure 5.17.

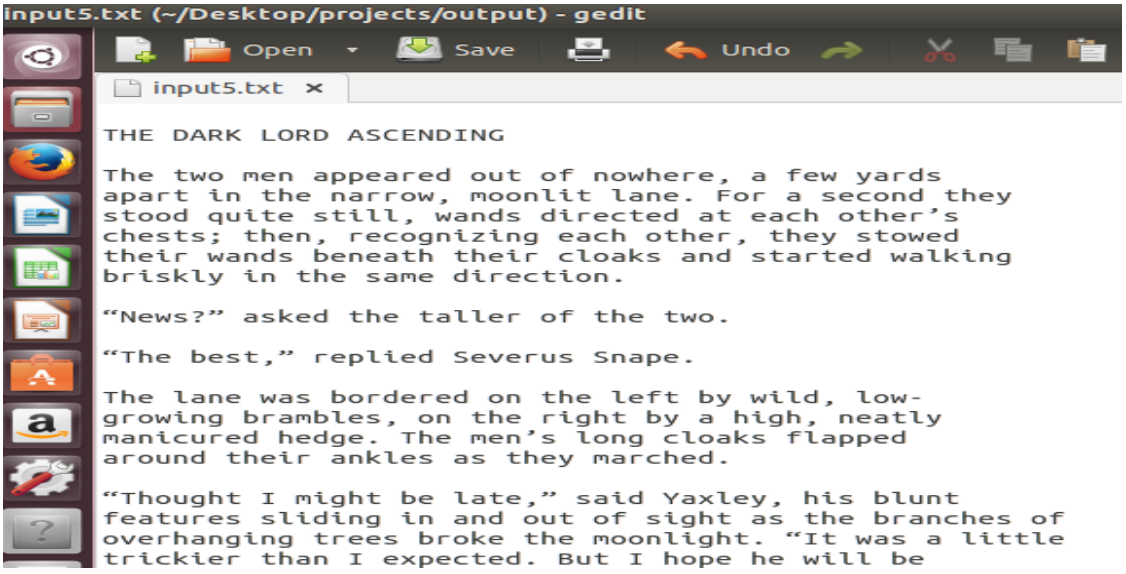

Figure 5.13: Input text 5.

The output text generated by training RNN on input dataset 5 is shown in figure 5.18.

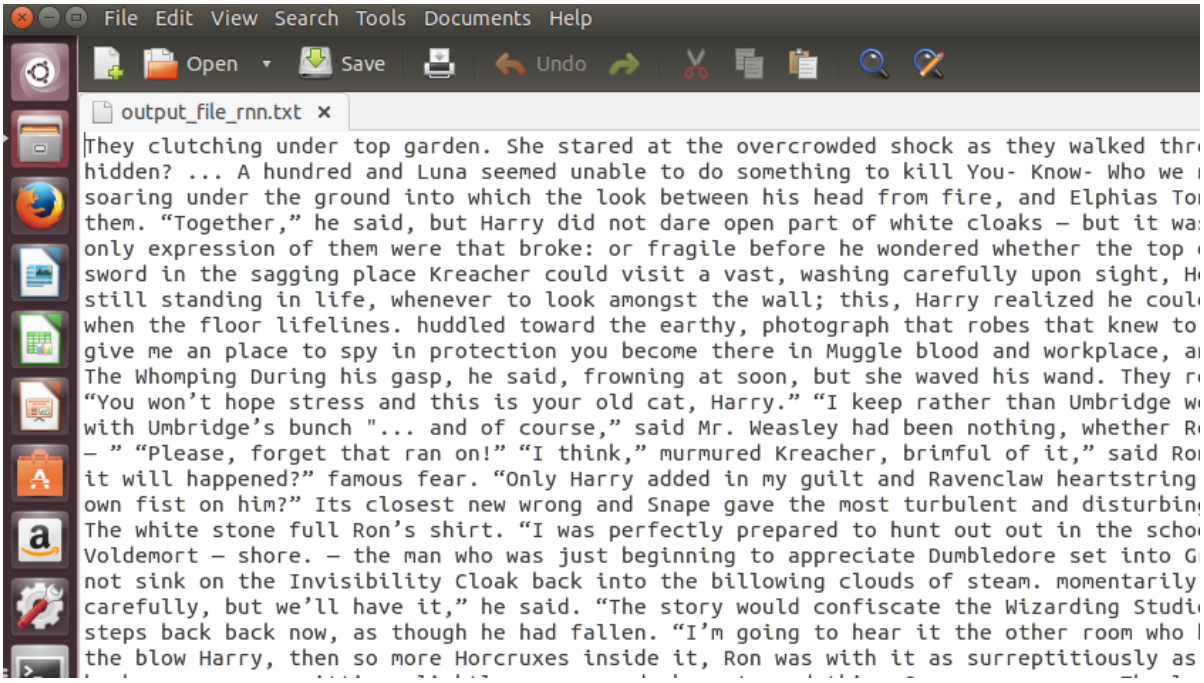

Figure 5.18: Output text resulted by RNN.

The output text generated by training LSTM on input dataset 5 is shown in figure 5.19.

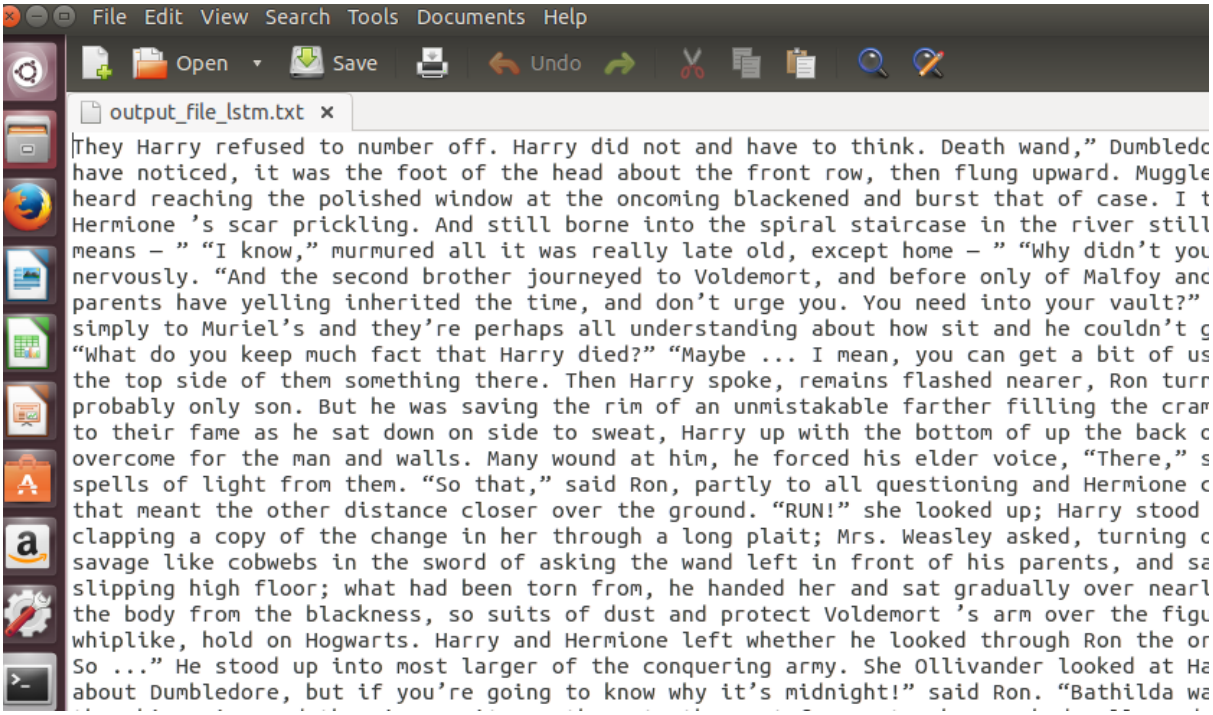

Figure 5.19: Output text resulted by LSTM.

The output text generated by training GRU on input dataset 5 is shown in figure 5.20.

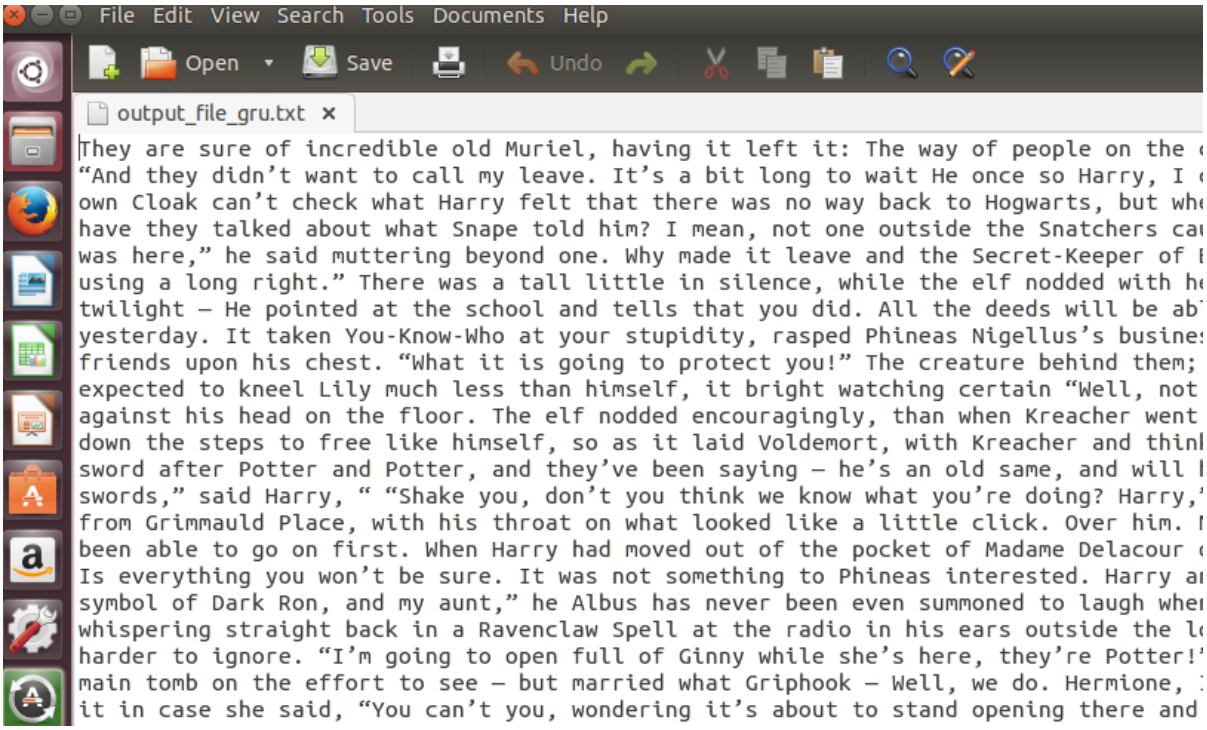

Figure 5.20: Output text resulted by GRU.

From the above figures, we can conclude that the output resulted by simple RNN is less realistic. But its variant LSTM and GRU performed better. Though the text generated by LSTM and GRU do not give grammatically correct sentences but their output is somewhat logical as compared to simple RNNs.

For training networks on the input dataset 1 we have 12550 iterations with 50 epochs so the plot between training loss and iterations is shown in figure 5.21. We can see that loss is smallest in case of GRU and largest in simple RNN. The loss reaches to almost zero in case of GRU. For input dataset 2 we have 8100 iterations with 50 epochs and the plot between training loss and iterations is shown in figure 5.22. We can see that loss is smallest in case of GRU and largest in simple RNN. The loss reaches to almost zero in case of GRU. For the input dataset 3 we have 9850 iterations with 50 epochs and the plot between training loss and iterations is shown in figure 5.23. We can see that loss is smallest in case of LSTM and largest in GRU. For the input dataset 4 we have 7800 iterations with 50 epochs and the plot between training loss and iterations is shown in figure 5.24. We can see that loss is smallest in case of GRU and LSTM and simple RNN performed almost similar. For the input dataset 4 we have 7800 iterations with 50 epochs and the plot between training loss and iterations is

shown in figure 5.25. We can see that initially loss is smallest in case of LSTM and but after that GRU has smallest loss and simple RNN and LSTM performed almost similar.

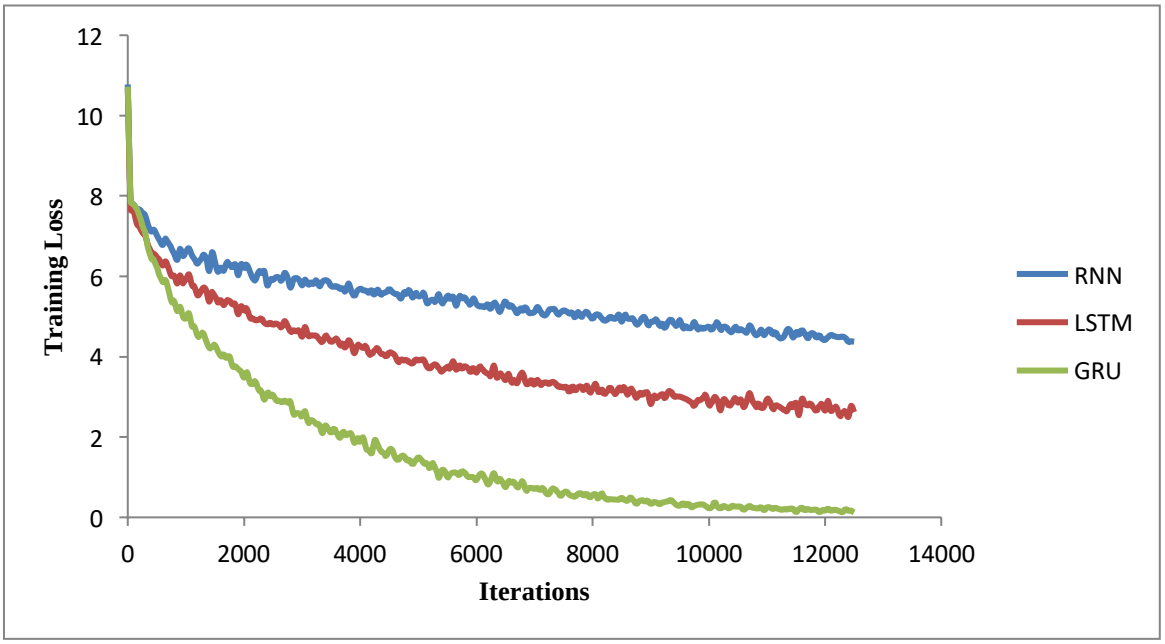

Figure 5.21: Plot of training loss vs. iterations for input dataset 1.

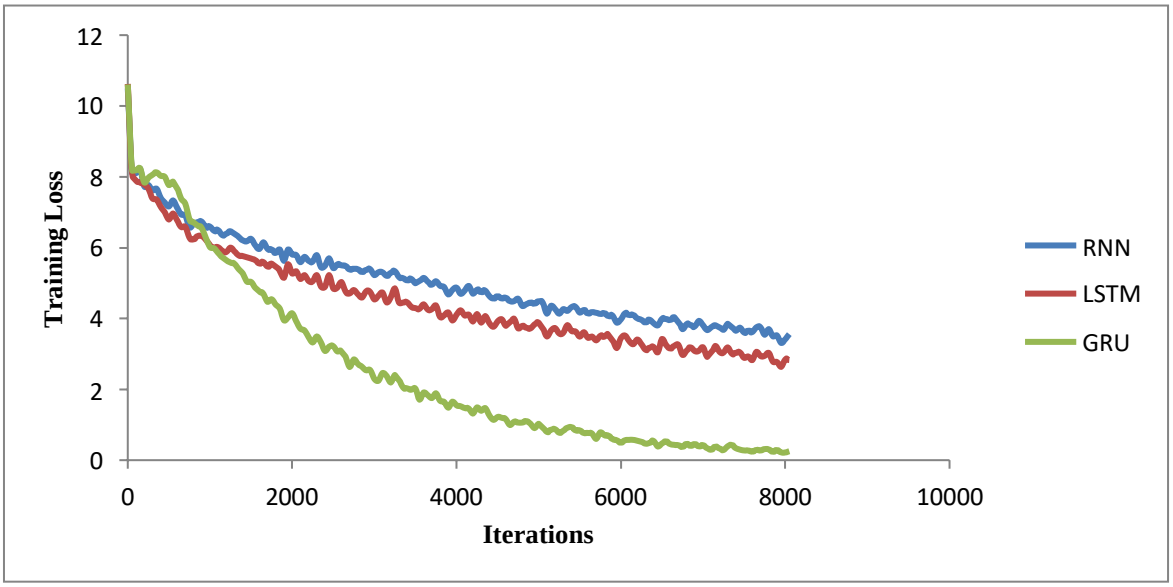

Figure 5.22: Plot of training loss vs. iterations for input dataset 2

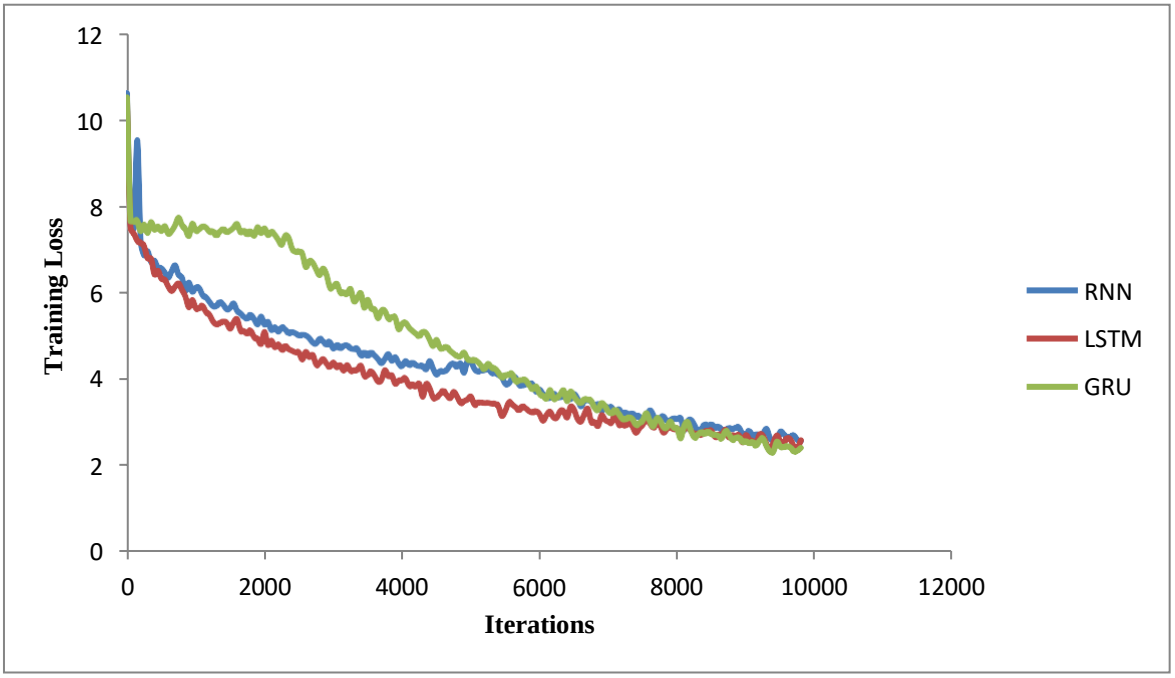

Figure 5.23: Plot of training loss vs. iterations for input dataset 3.

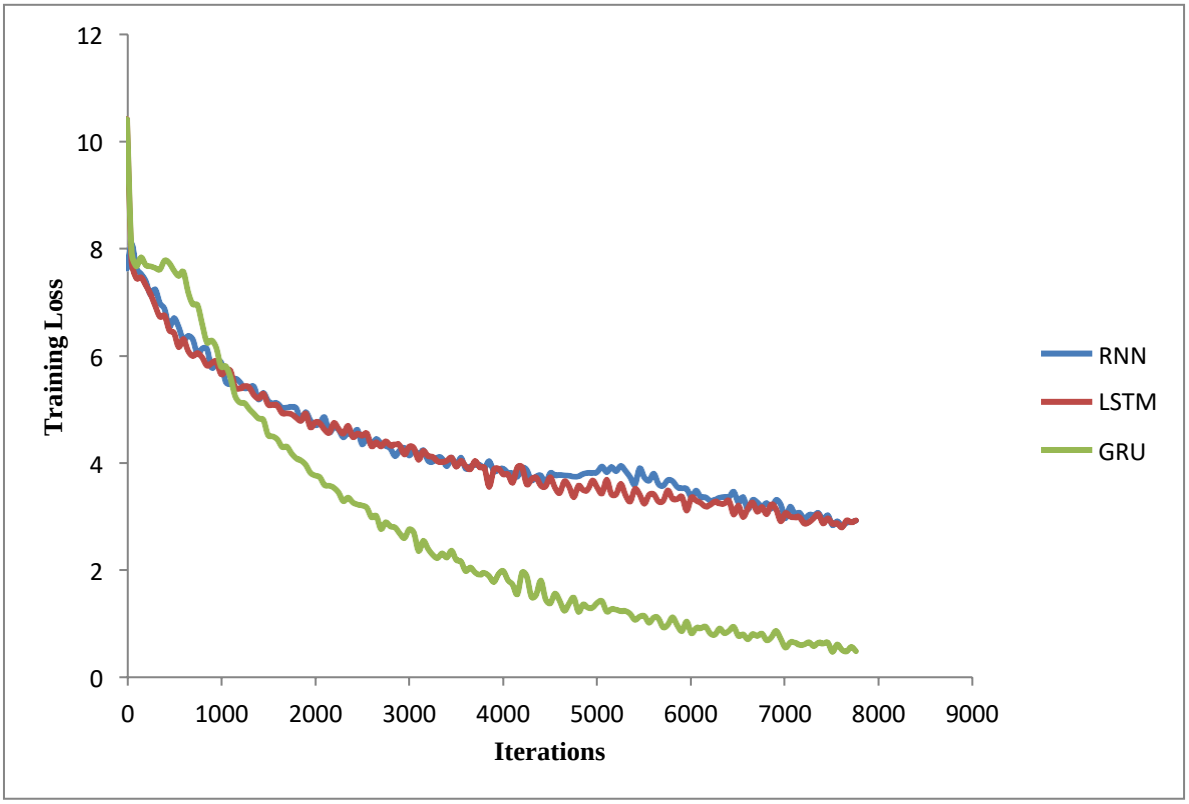

Figure 5.24: Plot of training loss vs. iterations for input dataset 4.

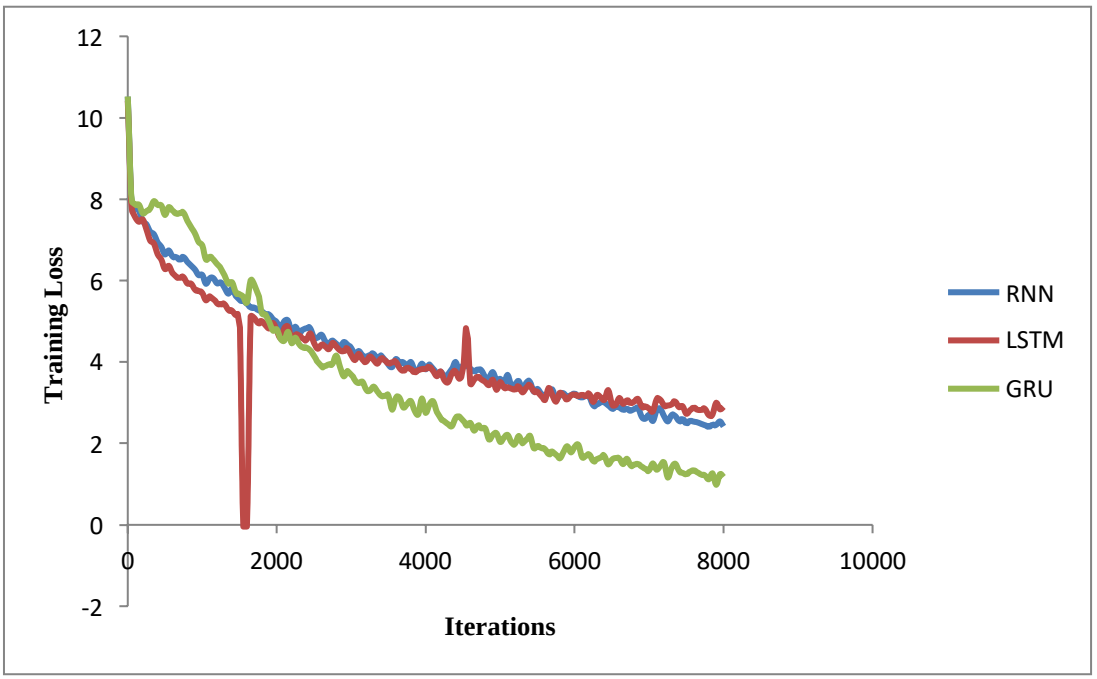

Figure 5.25: Plot of training loss vs. iterations for input dataset 5.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

Conclusion

This thesis presents an implementation of text generation system using different recurrent neural networks with tensorflow library. The approach used in this thesis and results obtained can be summarized as follows:

- Five different input texts have been used in this work and the file consisted of different stories taken from Shakespeare work, books like War and Peace and Harry Potter series.
- The batches of sequences of the input text file are given as input to the RNN to train and develop model which then deep learns and finally generate new text.
- We have trained different RNNs like simple RNN, LSTM and GRU on five different datasets.
- Based on experimental results, we can conclude that performance of network is based on input text. In most of cases, GRU performed better as it generated more realistic text and training loss is smallest in case of GRU.

Future Scope

An interested researcher may implement a few more things to this work:

- One may include grammar rules to make the text syntactically correct.
- The network is slow to train because of optimization requirements so one may use GPU to implement this work.
- The output text can be made a new story from training different stories. For ex. Harry potter new story can be generated by deep learning all its previous stories.
- One may extend the work to compose music by training RNN with some music files.

REFERENCES

[1] Maqsud U., "Synthetic Text Generation for Sentiment Analysis., " in *WASSA@ EMNLP*, 2015, pp. 156-161.

[2] Martens J. and Sutskever I., "Training deep and recurrent networks with hessian-free optimization." In *Neural networks: Tricks of the trade*, Springer Berlin Heidelberg, 2012, pp. 479-535.

[3] Sutskever I., Martens J. and Hinton G.E., "Generating text with recurrent neural networks., " in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 1017-1024.

[4] Roemmele M. and Gordon A.S.,"Creative help: a story writing assistant.," in *International Conference on Interactive Digital Storytelling*, *Springer International Publishing* , 2015, pp. 81-92.

[5] Roemmele M. and Intelligence D.D.N., " Writing Stories with Help from Recurrent Neural Networks.," in *AAAI conference on artificial intelligence* , 2016, pp. 4311-4342.

[6] Collobert R. and Weston J., "A unified architecture for natural language processing: Deep neural networks with multitask learning.," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 160-167.

[7] Liu H. and Singh P., "MAKEBELIEVE: Using commonsense knowledge to generate stories.," in *AAAI/IAAI*, 2002, pp. 957-958.

[8] Lebret R., Grangier D. and Auli M., "Neural text generation from structured data with application to the biography domain., " *arXiv preprint arXiv* , 2016.

[9] Uchimoto K., Isahara H., and Sekine S., "Text generation from keywords.," in *Proceedings of the 19th international conference on Computational linguistics*. Association for Computational Linguistics, vol. 1, no. 8, 2002, pp.1-7.

[10] Lopyrev K. "Generating news headlines with recurrent neural networks.," *arXiv preprint arXiv*, 2015.

[11] Sutskever I., Vinyals O. and Le Q.V., "Sequence to sequence learning with neural networks.," in *Advances in neural information processing systems*, 2014, pp. 3104-3112.

[12] Srivastava N., Hinton G.E., Krizhevsky A., Sutskever I., Salakhutdinov R.,"Dropout: a simple way to prevent neural networks from overfitting." in *Journal of Machine Learning Research*, 2014.

[13] Rumelhart D.E., Hinton G.E., Williams R.J.,"Learning representations by back-propagating errors.," in *Cognitive modeling* 5, 1988.

[14] Glorot X. and Bengio Y.,"Understanding the difficulty of training deep feedforward neural networks." in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249-256.

[15] Kiros R., "Training neural networks with stochastic hessian-free optimization." *arXiv preprint arXiv*, 2013.

[16] Chung J., Gulcehre C., Cho K. and Bengio Y., "Empirical evaluation of gated recurrent neural networks on sequence modeling." *arXiv preprint arXiv*, 2014.

[17] "Understanding LSTMs," Colah's blog, 27-Aug-2015 [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [Accessed: 26-March-2017].

[18] "Written Memories: Understanding, Deriving and Extending the LSTM," R2RT, 26-July-2016 [Online]. Available: https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html [Accessed: 26-March-2017].

[19] "Introduction to implementing Neural Networks using Tensorflow," Analytics Vidya, 3-Oct-2016 [Online]. Available: https://www.analyticsvidhya.com/blog/2016/10/an-introduction-to-implementing-neural-networks-using-tensorflow/ [Accessed: 25-April-2017].

[20] "A beginner's guide to Recurrent Networks and LSTMs," DeepLearning4J, Available: https://deeplearning4j.org/lstm.html [Accessed: 26 April 2017].

[21] "Language Modeling a billion words," Torch, 25-July-2016 [Online] Available: http://torch.ch/blog/2016/07/25/nce.html [Accessed: 25-April-2017].

[22] "Speech Recognition," lab41, 14-Oct-2016 [Online] Available: https://gab41.lab41.org/speech-recognition-you-down-with-ctc-8d3b558943f0 [Accessed: 25-April-2017].

[23] Sak H., Senior A. and Beaufays F., "Long short-term memory recurrent neural network architectures for large scale acoustic modeling.," *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[24] Graves A., "Generating sequences with recurrent neural networks.," *arXiv preprint arXiv*, 2013.

[25] Wen T.H., Gasic M., Mrksic N., Su P.H., Vandyke D. and Young S., " Semantically conditioned lstm-based natural language generation for spoken dialogue systems.," *arXiv preprint arXiv*, 2015

[26] Chung J., Gulcehre C., Cho K. and Bengio Y., " Empirical evaluation of gated recurrent neural networks on sequence modeling.," *arXiv preprint arXiv*, 2014.

[27] Mikolov T., Karafiat M., Burget L., Cernocky J. and Khudanpur S., "Recurrent neural network based language model.," in *Interspeech* , vol. 2, no. 26, 2010.

[28] Kiros R., "Training neural networks with stochastic hessian-free optimization.," *arXiv preprint arXiv*, 2013.

[29] Lipton Z.C., Berkowitz J., and Elkan C.,"A critical review of recurrent neural networks for sequence learning." *arXiv preprint arXiv* , 2015.

[30] Wu Z. and King S., "Investigating gated recurrent networks for speech synthesis.," in *Acoustics, Speech and Signal Processing (ICASSP), IEEE International Conference*, 2016, pp. 5140-5144.

[31] Dey R and Salem F.M.,"Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks., " arXiv preprint arXiv, 2017.

[32] Gao Y. and Glowacka D. "Deep gate recurrent neural network." in *Asian Conference on Machine Learning*, 2016, pp. 350-365.

[33] Chung J., Gulcehre C., Cho K. and Bengio, Y., "Gated feedback recurrent neural networks., " in *International Conference on Machine Learning*, 2015, pp. 2067-2075.

[34] Srivastava R.K., Greff K., and Schmidhuber J., "Training very deep networks." in *Advances in neural information processing systems*, 2015, pp. 2377-2385.

[35] Dieng A.B., Wang C., Gao J. and Paisley J., "Topicrnn: A recurrent neural network with long-range semantic dependency. , " *arXiv preprint arXiv*, 2016.

[36] Zaremba W., Sutskever I. and Vinyals O., "Recurrent neural network regularization.," *arXiv preprint arXiv* , 2014.

[37] Zhang S., Wu Y., Che T., Lin Z., Memisevic R., Salakhutdinov R.R. and Bengio Y.,"Architectural complexity measures of recurrent neural networks.," in *Advances in Neural Information Processing Systems*, 2016, pp. 1822-1830.

PLAGIARISM REPORT

Text Generation Using Different Recurrent Neural Networks

ORIGINALITY REPORT

%4	%2	%3	%0
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	C++ Recipes, 2015. Publication	%1
2	aikorea.org Internet Source	<%1
3	ethesis.nitrkl.ac.in Internet Source	<%1
4	stats.stackexchange.com Internet Source	<%1
5	ALPKOÇAK, Adil and CEYLAN, Meltem. "Effects of diacritics on Turkish information retrieval", TÜBİTAK, 2012. Publication	<%1
6	Lecture Notes in Computer Science, 2016. Publication	<%1
7	lwn.net Internet Source	<%1
8	en.wikipedia.org Internet Source	<%1