MINI PROJECT REPORT

Next Word Prediction

by

Abhinav Maurya

Entry No. : 19BCS003

Submitted in partial fulfillment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

SHRI MATA VAISHNO DEVI UNIVERSITY, KATRA

(School of Computer Science & Engineering)

JAMMU & KASHMIR – 182 320

Session 2021-22

ACKNOWLEDGMENTS

We three have made efforts towards this project. However, it would not have been possible without the guidance of Dr. Baijnath Kaushik. At every moment of the project, he had been supporting and made valuable remarks over which we made changes and got better results than what we could have got.

We would also thank all of the faculty members who directly or indirectly inspired, supported, and gave their valuable suggestions. Lastly, we would like to thank all of the classmates who helped and supported us.

DECLARATION

We undersigned solemnly declare that the project report **Next Word Prediction** is based on my work carried out during the course of our study under the supervision of Dr. Baijnath Kaushik**.**

We assert the statements made and conclusions are drawn are an outcome of my research work. I further certify that

I. The work contained in the report is original and has been done by us under the supervision of my supervisor.

II. The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or the any other University of India or abroad.

III. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and given their details in the references.

Abhinav Maurya

19bcs003, B. Tech, 5th Sem

School of Computer Science & Engineering

Shri Mata Vaishno Devi University, Katra

I endorse the above declaration of the Student.

(Name and Signature of the Supervisor)

ABSTRACT

Next Word Prediction is called Language Modeling. next-word prediction, is convenient for users because it helps to type without errors and faster. LSTM and BI-LSTM were chosen for next-word prediction. Their sequential nature (current output depends on previous) helps to successfully cope with the next-word prediction task. After the clients composed 3 words, the model will comprehend 3 words and anticipate impending top 2 words utilizing RNN neural network.

Our Aim of creating this model to predict 2 or more than 2 word as fast as possible utilizing minimum time. As RNN is Long short time memory it will understand past text and predict the words which may be helpful for the user to frame sentences and this technique uses letter to letter prediction means it predict a letter after letter to create a word.

Table of Contents

LIST OF FIGURES

Chapter 1: INTRODUCTION

The English language is one of the famous language in India. English language is spoken in most part the world, countries like United Kingdom, United States of America , New Zealand , Canada and many more .

1.1. NEED OF WORD PREDICTION

People around the world use their mobile devices for email, social networking, banking, and a variety of other purposes.

When typing on *mobile devices, it can be easier if the keyboard provides three options for the next word.

Its main advantage is that it increases the accuracy and efficiency of the model.

1.2 PROBLEM STATEMENT

Mostly we face the problem of writing with speed while we are on chat with someone. This creates a trouble in communication effectively.

So, the motivation behind the project is to build an efficient next word prediction model that accurately predicts various words, Depending upon previous word used in the sentence.

1.3 SOLUTION

LSTMs are being used a huge amount to solve many problems today because Long Short Term Memory (LSTM) network is extension of RNN (Recurrent Neural Network) that extends the memory . LSTM are used as building blocks for the layers of RNN . The same technology was used to solve our problem.

1.4 ORGANIZATION OF REPORT

The report for the project is structured in the form of chapters, wherein each chapter aims to describe in detail a certain aspect of the project. The chapters are broadly divided into 3 parts :

- Related Work
- Libraries used
- Conclusion

1.5 LSTM APPROACH

Gradient descent diminishes when one uses neural networks for backpropagation. We find the model useless, since weight updates are greatly affected. Three gates -forgotten, read, and input -- make up an LSTM's hidden state and memory cell.

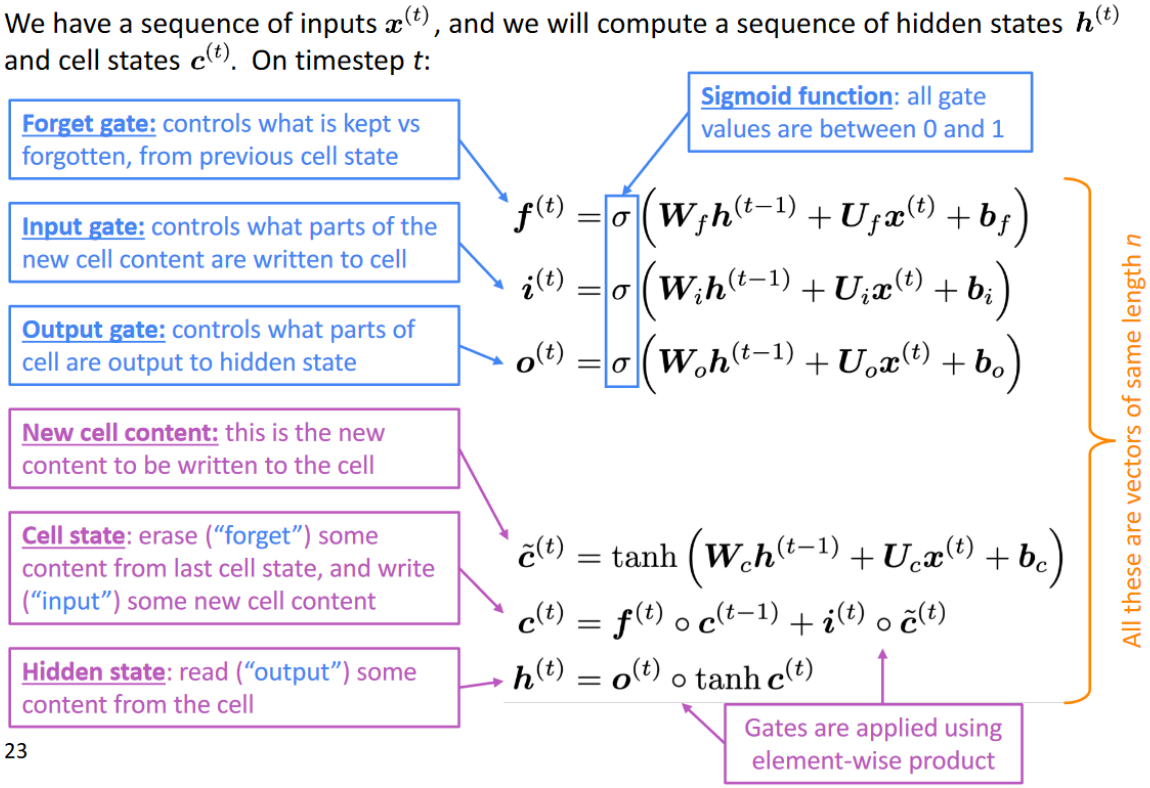

Fig. 1.1 Working of LSTM gate structure [Ref. 9]

Information that is useless can be removed with the forget gate. A gate that adds new information to a cell is an input gate, and a gate that outputs information from the cell to the next hidden state is an output gate. By using the sigmoid function, each gate equation can be reduced to zero or one.

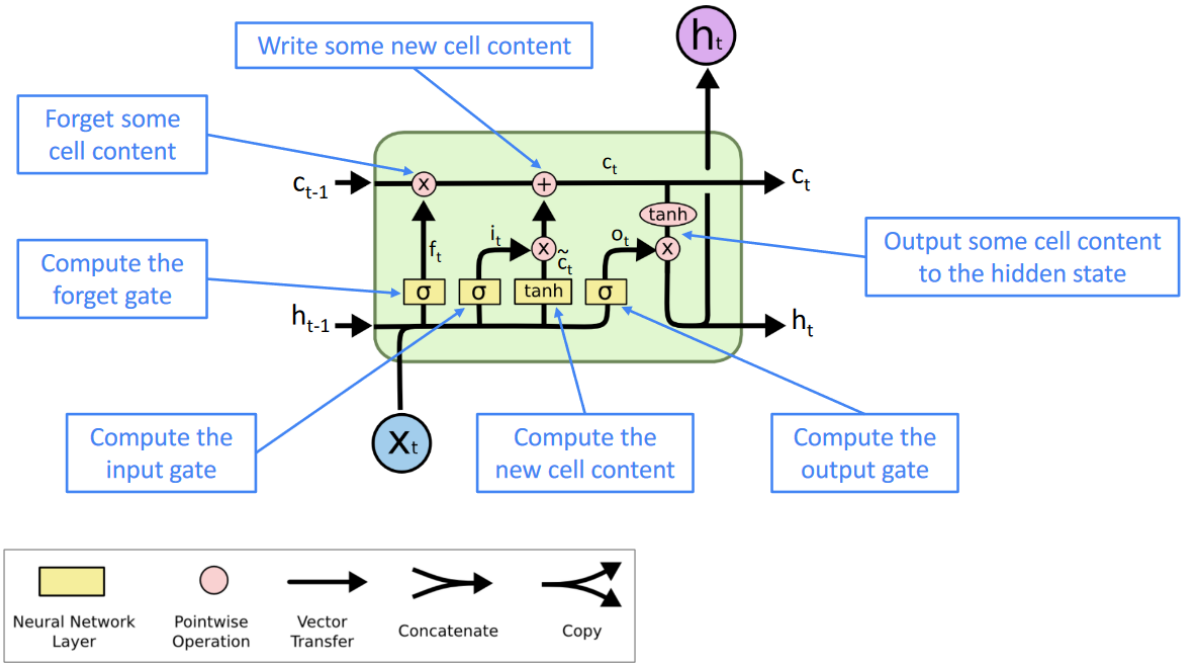

Fig. 1.2 Architecture of LSTM [Ref. 9]

This figure shows the exact architecture of an LSTM. In this example, X represents the word subscript t, which indicates the time instant. C and H are inputs from an earlier time or the last step, as we can see. It has a forget gate that controls the weights, so that it knows exactly what information needs to be removed before proceeding to the next gate. We use sigmoid for this. As soon as I add the input, some new information is written in the cell.

LIMITATIONS OF LSTM:

LSTMs work well for some problems, but they have some drawbacks:

- LSTMs take a longer time to train.
- LSTMs require more memory to be trained.
- LSTMs are easily overfit.
- Dropout is very difficult to implement in LSTMs.
- LSTM is sensitive to various random weight initializations.

1.6 BI LSTM APPROACH

Bi-LSTM neural networks consist of LSTM units that carry out both past and future operations. As Bi-LSTMs are able to learn all dependencies without storing duplicate context information, they have shown excellent performance when dealing with sequential modeling problems and are widely used for text classification. Unlike LSTMs, Bi-LSTMs are characterized by two parallel layers propagating back and forth to capture dependencies between contexts.

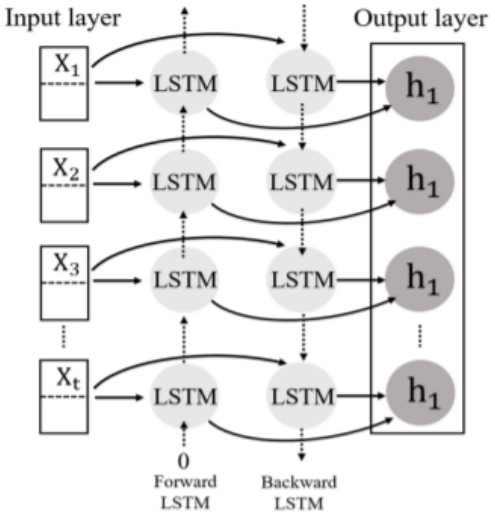

Figure 3. The Bi-long short-term memory (LSTM) process used to capture sequential features. The last hidden layer of the LSTM is extracted as features representing the text.

Fig. 1.3 Bi-LSTM network structure [Ref. 9]

CHAPTER 2 : RELATED WORKS

2.1 PREPROCESSING THE DATA

We use Tokenizer from keras.processing.text to encode our input strings for input to the Embedding layer. What we do in preprocessing is very simple: We first create features dictionary sequences. Then, with the help of the Tokenizer, we encode it into integer form.

```
In [4]:
file= open("data1.txt","r",encoding="UTF-8")

In [5]:
lines = []
for i in file:
      lines.append(i)
data =""
for i in lines:
      data =' '.join(lines)

data = data.replace('\n', '').replace('\r', '').replace('\ufeff', '').replace('“','').repla
print(len(data))

54373
```

Fig. 2.1 Removing unwanted string

I am doing fine, how long has it been since last we met? How are your parents doing?" We clean our corpus and tokenize it with Regular Expressions.

fdfdsqqqd

```
In [6]:

data = data.split()
data = ' '.join(data)
print(len(data))
```

```
53458
```

```
In [28]:

data[:100]
```

```
Out[28]:

'The Project Gutenberg eBook of The Man With the Golden Eyes, by Alexander B
lade This eBook is for th'
```

Fig. 2.2 Merging the cleaned Data

The nltk(Natural Language Toolkit) library provides expressions, and word_tokenize from the sequences dictionary.

```
In [30]:

import nltk
from nltk.tokenize import word_tokenize

tokens = word_tokenize(data)
train_len = 3+1
text_sequences = []
for i in range(train_len,len(tokens)):
    seq = tokens[i-train_len:i]
    text_sequences.append(seq)
print(text_sequences)
print(len(text_sequences))
```

Fig. 2.3 Creating text sequence using word_tokenize

Throughout our corpus of training sequences, they are collected into the 'text_sequences' list, which looks like this::

```
[['The', 'Project', 'Gutenberg', 'eBook'], ['Project', 'Gutenberg', 'eBoo
k', 'of'], ['Gutenberg', 'eBook', 'of', 'The'], ['eBook', 'of', 'The', 'Ma
n'], ['of', 'The', 'Man', 'With'], ['The', 'Man', 'With', 'the'], ['Man',
'With', 'the', 'Golden'], ['With', 'the', 'Golden', 'Eyes'], ['the', 'Gold
en', 'Eyes', ','], ['Golden', 'Eyes', ',', 'by'], ['Eyes', ',', 'by', 'Ale
xander'], [',', 'by', 'Alexander', 'Blade'], ['by', 'Alexander', 'Blade',
'This'], ['Alexander', 'Blade', 'This', 'eBook'], ['Blade', 'This', 'eBoo
k', 'is'], ['This', 'eBook', 'is', 'for'], ['eBook', 'is', 'for', 'the'],
['is', 'for', 'the', 'use'], ['for', 'the', 'use', 'of'], ['the', 'use',
'of', 'anyone'], ['use', 'of', 'anyone', 'anywhere'], ['of', 'anyone', 'an
ywhere', 'in'], ['anyone', 'anywhere', 'in', 'the'], ['anywhere', 'in', 't
he', 'United'], ['in', 'the', 'United', 'States'], ['the', 'United', 'Stat
es', 'and'], ['United', 'States', 'and', 'most'], ['States', 'and', 'mos
t', 'other'], ['and', 'most', 'other', 'parts'], ['most', 'other', 'part
```

Fig. 2.4 output of text sequence

After tokenizing, we have the above sequences in encoded form. The numbers represent the indexes of the respective words from the 'sequences' dictionary before reassignment.

```
sequences = {}
count = 1
for i in range(len(tokens)):
    if tokens[i] not in sequences:
        sequences[tokens[i]] = count
        count += 1
print(sequences)
```

```
[[2, 20, 55, 113], [20, 55, 113, 7], [55, 113, 7, 2], [113, 7, 2, 28], [7,
2, 28, 16], [2, 28, 16, 2], [28, 16, 2, 79], [16, 2, 79, 51], [2, 79, 51,
3], [79, 51, 3, 43], [51, 3, 43, 426], [3, 43, 426, 427], [43, 426, 427, 2
3], [426, 427, 23, 113], [427, 23, 113, 25], [23, 113, 25, 27], [113, 25,
27, 2], [25, 27, 2, 123], [27, 2, 123, 7], [2, 123, 7, 169], [123, 7, 169,
428], [7, 169, 428, 11], [169, 428, 11, 2], [428, 11, 2, 101], [11, 2, 10
1, 80], [2, 101, 80, 8], [101, 80, 8, 225], [80, 8, 225, 66], [8, 225, 66,
574], [225, 66, 574, 7], [66, 574, 7, 2], [574, 7, 2, 226], [7, 2, 226, 4
9], [2, 226, 49, 45], [226, 49, 45, 331], [49, 45, 331, 8], [45, 331, 8, 1
6], [331, 8, 16, 277], [8, 16, 277, 45], [16, 277, 45, 575], [277, 45, 57
5, 576], [45, 575, 576, 1], [575, 576, 1, 10], [576, 1, 10, 68], [1, 10, 6
8, 124], [10, 68, 124, 24], [68, 124, 24, 3], [124, 24, 3, 227], [24, 3, 2
27, 24], [3, 227, 24, 170], [227, 24, 170, 21], [24, 170, 21, 577], [170,
21, 577, 24], [21, 577, 24, 171], [577, 24, 171, 2], [24, 171, 2, 69], [17
1, 2, 69, 7], [2, 69, 7, 2], [69, 7, 2, 20], [7, 2, 20, 55], [2, 20, 55, 9
2], [20, 55, 92, 278], [55, 92, 278, 16], [92, 278, 16, 23], [278, 16, 23,
113], [16, 23, 113, 21], [23, 113, 21, 279], [113, 21, 279, 49], [21, 279,
49, 280], [279, 49, 280, 1], [49, 280, 1, 53], [280, 1, 53, 10], [1, 53, 1
0, 33], [53, 10, 33, 26], [10, 33, 26, 198], [33, 26, 198, 11], [26, 198,
```

Fig 2.5 sequence with index

Once the sequences have been encoded, we then split them into inputs and outputs to define training data and target data. training data and target data. The model will guess the next word based on three previous words, so the first three words serve as inputs, and the last word is the label that the model should predict.

```
In [16]:
xs, labels = input_sequences[:,:-1],input_sequences[:,-1]
ys = tf.keras.utils.to_categorical(labels, num_classes=total_words)

In [17]:
print(xs[5])
print(labels[5])
print(ys[5][14])

[  0   0   0   0   0   0   0   0   0   0   1  14  11 103   3   1]
23
0.0
```

Fig. 2.6 output of input_sequence with x_label and y_label

To do this, we convert the output labels into one-hot vectors, which are combinations of 0's and 1. One-hot vectors in 'train_targets' would look like this:

For the first target label "how", the sequence dictionary index was 1. In encoded form, you'd expect that index 1 would be shifted to '1' in the first one-hot vector of "train_targets".

2.2 BUILDING THE MODEL

We now train our Sequential model that consists of five layers: An Embedding layer, one Bi-LSTM layers, and one Dense layers. The input length of our Embedding layer is set to the size of the sequence, which in this example is 3. As such, we split the inputs and targets for our training data 3 to 1, which means that when we feed our model for prediction we must provide a 3 by 3 vector.

```
In [23]:
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150)))
model.add(Dense(total_words, activation='softmax'))
adam = Adam(lr=0.01)
checkpoint = ModelCheckpoint("next_words.h5", monitor='loss', verbose=1, save_best_only=Tru
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=50, verbose=1,callbacks=[checkpoint])
#print model.summary()
print(model)
```

Fig 2.7 preparing the model for training

2.3 PREDICTING WORDS:

$$\text{Softmax}(z_i) = \exp(z_i) / \sum_j \exp(z_j)$$

Using the softmax function, we can give our model input in encoded form and get the three most probable words as shown below.

Example

2.33 P (Class 1) = exp (2.33) / exp(2.33) + exp(-1.46) + exp(0.56) = 0.83827314

-1.46 P (Class 2) = exp (-1.46) / exp(2.33)+ exp(-1.46)+ exp(0.56) = 0.01894129

0.56 P (Class 3) = exp (0.56) / exp(2.33) + exp(-1.46) + exp(0.56) = 0.14278557

Predicting the three suggested words

In [20]:

```
from keras.preprocessing.sequence import pad_sequences
input_text = input().strip().lower()
encoded_text = tokenizer.texts_to_sequences([input_text])[0]
pad_encoded = pad_sequences([encoded_text], maxlen=max_sequence_len-1, truncating='pre')
print(encoded_text, pad_encoded)
for i in (model.predict(pad_encoded)[0]).argsort()[-3:][::-1]:
    pred_word = tokenizer.index_word[i]
    print("Next word suggestion:",pred_word)
```

```
next word can be
[213, 54, 43] [[  0   0   0   0   0   0   0   0   0   0   0   0   0 213  54
43]]
Next word suggestion: only
Next word suggestion: there
Next word suggestion: copied
```

Fig. 2.8 predicting the output

In the above code we use padding since we trained our model on sequences of length 3, The padding function ensures that the last three words are taken into account when we input five words. Without it, our results wouldn't be as good! The same is true if you enter an unknown word because the one-hot vector will contain 0 at that word's index. In the future, we may consider adding sequences of length 2(inputs) to 1(target label) and 1(input) to 1(target label) for optimal results, This is what we did with 3 (inputs) to 1 (target label). The model predicts the next three words based on the previous three.

Predicting the output (Next two continous words)

localhost:8888/notebooks/final-next-word-prediction.ipynb#

12/23/21, 1:04 PM final-next-word-prediction - Jupyter Notebook

In [18]:

```
from tensorflow.keras.models import load_model
import pickle

# Load the model and tokenizer
model = load_model('next_words.h5')
```

```
Enter your line: the output of the project
Suggested next two word are :  ['gutenberg', 'literary']
the output of the project  gutenberg literary
Enter your line: the output of the project  gutenberg literary
Suggested next two word are :  ['archive', 'foundation']
the output of the project  gutenberg literary archive foundation
Enter your line: the output of the project  gutenberg literary archive fou
ndation
Suggested next two word are :  ['is', 'a']
the output of the project  gutenberg literary archive foundation is a
Enter your line: the output of the project  gutenberg literary archive fou
ndation is a
Suggested next two word are :  ['non', 'profit']
the output of the project  gutenberg literary archive foundation is a non
profit
Enter your line: the output of the project  gutenberg literary archive fou
ndation is a non profit
Suggested next two word are :  ['but', 'a']
the output of the project  gutenberg literary archive foundation is a non
profit but a
Enter your line: the output of the project  gutenberg literary archive fou
ndation is a non profit but a
Suggested next two word are :  ['non', 'profit']
```

Fig. 2.9 predicting the next two continuous words

CHAPTER 3 : LIBRARIES USED

```
In [14]:
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, LSTM, Dense, Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
```

Fig. 3.1 list of libraries used

3.1 TENSORFLOW

This open source artificial intelligence library uses data flow graphs to build models. Programmers can use it to build very complex neural networks. TensorFlow is primarily used for classification, perception, understanding, discovering, hypothesis generation, and prediction.

Main Use Cases of TensorFlow:

Text Based Applications

Other applications of TensorFlow are sentiment analysis (CRM, Social Media), threat detection (Social Media, Government), and fraud detection (Insurance, Finance).

Text-based applications are frequently used for language detection.

It is well known that Google Translate supports over 100 languages. These evolved versions can be used in a variety of situations, such as translating legalese jargon into plain language in contracts.

Text Summarization:

Google summarizes short texts using a technique called sequence-to-sequence learning. Headlines for news stories could be generated using this technique.

3.2 KERAS

For developing and evaluating deep learning models, Keras is a powerful and easy-to-use open-source Python library. Using the efficient numerical computation library TensorFlow, you can define and train neural network models in just a few lines of code.

3.2.1 KERAS PREPROCESSING

In the Keras deep learning library, the Keras Preprocessing module handles the preprocessing and augmentation of data. There are utilities for working with image data, text data, and sequence data.

3.2.2 KERAS LAYERS

Keras is based on layers, which are the basic building blocks for neural networks. In a layer, there is a tensor-in-tensor-out computation function (called the layer's call method) and some state, which resides in TensorFlow variables (called a layer's weight).

Jason Brownlee notes that the first layer is technically composed of two layers, the input layer, specified by input_dim, and the hidden layer

3.3 PAD SEQUENCE

It ensures that each sequence in a list has the same length using pad_sequences. The default padding method to accomplish this is to pad the beginning of every sequence by zero so that all sequences are the same length.

3.4 KERAS OPTIMIZER

In machine learning and deep learning, optimizers are classes or methods used to modify attributes of your models, such as weights and learning rate. These methods help to speed up the learning process.

A stochastic gradient descent method, Adam optimization utilizes an adaptive estimation of first- and second-order moments.

4. PURPOSE OF THE PROJECT

The main goal of this project is to help users of small devices text faster. Also, in this project we pretend to personalize the prediction of the next word using personal writing history.

CHAPTER 5: REQUIREMENT AND SPECIFICATION

5.1 EXPERIMENTAL SETUP

We have used the Windows10′ operating system for our experimental setup. The specifications are as follows:

Processor: Intels Core *M i5-9300H CPU (2.40GHz)

Installed memory (RAM): 8.00 GB

For training of Next word prediction model:

we used jupyter Notebook service from our System.

Jupyter Notebook provided a much more powerful system specification to work upon

without buying them.

5.2 PYTHON:

It is a high-level general-purpose interpreted language. Code readability is enhanced by its use of significant indentation. Its language constructs program and object-oriented approach aim to help programmers write clear, logical code for small and large projects.

5.3 DATA SET:

We have following dataset:

A text file in utf8 encoding downloaded from the website project Gutenberg

5.4 CONCLUSION

With the use of Machine Learning algorithms we are able to suggest next word based upon the last three words of a given input. At present the code implements work for satisfied results.

CHAPTER 6: TRAINING MODEL

Screenshots of the results while training model

```
Epoch 35/50
266/267 [============================>.] - ETA: 0s - loss: 0.2784 - accura
cy: 0.9182
Epoch 00035: loss improved from 0.29133 to 0.27828, saving model to next_w
ords.h5
267/267 [==============================] - 9s 34ms/step - loss: 0.2783 - a
ccuracy: 0.9183
Epoch 36/50
267/267 [==============================] - ETA: 0s - loss: 0.2756 - accura
cy: 0.9183
Epoch 00036: loss improved from 0.27828 to 0.27562, saving model to next_w
ords.h5
267/267 [==============================] - 10s 36ms/step - loss: 0.2756 -
accuracy: 0.9183
Epoch 37/50
267/267 [==============================] - ETA: 0s - loss: 0.2670 - accura
cy: 0.9205
Epoch 00037: loss improved from 0.27562 to 0.26703, saving model to next_w
ords.h5
267/267 [==============================] - 9s 34ms/step - loss: 0.2670 - a
ccuracy: 0.9205
Epoch 38/50
266/267 [============================>.] - ETA: 0s - loss: 0.2689 - accura
cy: 0.9208
Epoch 00038: loss did not improve from 0.26703
267/267 [==============================] - 9s 32ms/step - loss: 0.2688 - a
ccuracy: 0.9208
Epoch 39/50
266/267 [============================>.] - ETA: 0s - loss: 0.2656 - accura
cy: 0.9202
```

Fig 3.2 output of the model training

LEARNING GRAPH :

ACCURACY:

```
In [25]:
plot_graphs(history, 'accuracy')
```

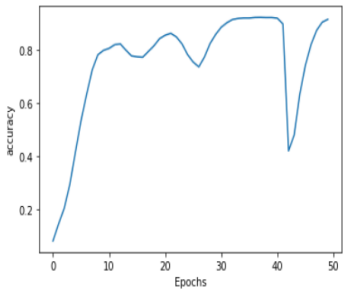

Fig. 6.1 accuracy vs epochs graphs

LOSS GRAPH:

```
In [26]:
plot_graphs(history, 'loss')
```

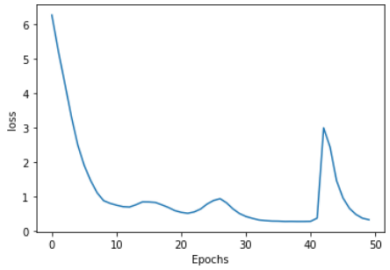

Fig. 6.2 Loss vs epochs graphs

CHAPTER 7 CONCLUSION

Next Word Prediction report presents how a valuable keyboard application that uses predictive text models can be built. It assumes, that the word which would be predicted by the application should belongs to English Language .

Accordingly, humans are able to predict a new word by using prior experience or their own knowledge of the subject, such as after the word yellow predicting either hat or paper, or we use syntactic knowledge such as after the word the predicting that adjective should appear, or we use lexical knowledge to choose potatoes rather than steak after the word baked.For a machine to predict next word in a sentence, in the same fashion as humans, the appraoch in this report assumes that

1. A simple statistical technique can be employed to account for a substantial part of the domain or subject knowledge needed to allow Word Prediction.

Techniques will be based on probabilities of a sequence (of letters, words, etc.) and not on Natural Language Parsing Context Free Grammar to parse words into subjects, verbs, and objects.

One of the biggest challenges in predicting is that infinite sentences or strings are possible with some finite vocabulary of words.

REFERENCES

Websites used for reference:

1. https://medium.com/@antonio.lopardo/the-basics-of-language-modeling-1c8832f21079
2. https://www.kaggle.com/ysthehurricane/next-word-prediction-bi-lstm-tutorial-easy-way
3. https://medium.com/analytics-vidhya/nlp-word-prediction-by-using-bidirectional-lstm-9c01c24b2725
4. https://bansalh944.medium.com/text-generation-using-lstm-b6ced8629b03
5. https://medium.com/geekculture/next-word-prediction-using-lstms-98a1edec8594
6. https://github.com/rajveermalviya/language-modeling
7. https://github.com/kurchi1205/Next-word-Prediction-using-Swiftkey-Data/blob/main/LSTM%20Model.ipynb
8. https://medium.com/geekculture/next-word-prediction-using-lstms-98a1edec8594
9. https://www.analyticsvidhya.com/blog/2021/08/predict-the-next-word-of-your-text-using-long-short-term-memory-lstm/

Books used for reference:

1. Graves, A.; Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw. 2005, 18, 602–610. [CrossRef] [PubMed]
2. Liu, G.; Guo, J. Bidirectional LSTM with attention mechanism and convolutional layer for text classification. Neurocomputing 2019, 337, 325–338. [CrossRef]
3. Liang, D.; Zhang, Y. AC-BLSTM: Asymmetric convolutional bidirectional LSTM networks for text classification. arXiv 2016, arXiv:1611.01884.