Next Word Prediction using NLP and Deep Learning

A MINI PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

MOHITH KODALI [Reg No: RA2111026010358]
SHRIYA [Reg No: RA2111026010345]

Under the guidance of

Dr. KARPAGAM M

Assistant Professor, Department of Computer Science and Engineering

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **"Next Word Prediction using NLP and Deep Learning"** is the bona fide work of **MOHITH KODALI [Reg No: RA2111026010358], SHRIYA [Reg No: RA2111026010345]** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE
Dr. Karpagam M
Assistant Professor
Department of Computational Intelligence

SIGNATURE
Dr. Annie Uthra
Head of Department
Department of Computational Intelligence

ABSTRACT

Natural Language Processing (NLP) has come a long way in recent years. It has revolutionized the way we communicate with machines, enabling us to interact with them in a more intuitive and efficient way. One of the essential components of NLP systems is next-word prediction, which involves predicting the next word a user is likely to type or speak based on their previous input. This capability is crucial for various applications, including text completion, auto-correction, speech recognition, and chatbots.

Next-word prediction systems employ statistical language models that utilize various techniques to predict the likelihood of a word based on its context. Traditional language models such as n-gram models and Markov models have been used for this purpose. However, these models have limitations, such as the inability to handle long-term dependencies and lack of semantic understanding.

These issues can be addressed by using deep learning techniques, such as recurrent neural networks (RNNs) and transformers. Despite these challenges, there have been significant advancements in the development of next-word prediction systems using deep learning. Researchers have proposed techniques to address the data scarcity issue, such as transfer learning, where pre-trained models are fine-tuned for specific domains. Additionally, pruning and compression techniques have been employed to reduce the size and complexity of models, making them more suitable for deployment on resource-limited devices.

In conclusion, next-word prediction is an essential component of modern-day NLP systems.

LIST OF FIGURES

ABBREVIATIONS

NLP	Natural Language Processing
RNN	Recurrent Neural Network
LSTM	LSTM

CHAPTER 1

INTRODUCTION

Next-word prediction is a crucial component of natural language processing (NLP) systems, enabling machines to predict the most likely word a user will type or speak next based on their previous input. It has become increasingly important in various applications, such as text completion, auto-correction, speech recognition, and chatbots.Next-word prediction is based on statistical language models that utilize various techniques to predict the probability of a word based on its context. These models have traditionally been based on n-gram models and Markov models, but these have limitations such as the inability to handle long-term dependencies and lack of semantic understanding. However, with the advancements in deep learning techniques, NLP systems can now utilize more sophisticated models to improve the accuracy and efficiency of next-word prediction.

Deep learning techniques such as recurrent neural networks (RNNs) and transformers have become popular for next-word prediction. RNNs are particularly useful for modeling sequential data, making them ideal for next-word prediction. These networks use the previous words in a sequence to generate a probability distribution over the possible next words. The model is trained using a large corpus of text data, and the weights are optimized to minimize the prediction error. Transformers, on the other hand, can learn long-term dependencies and capture semantic relationships between words. These models use self-attention mechanisms to attend to different parts of the input sequence, allowing them to capture more complex patterns.

However, building an effective next-word prediction system poses several challenges. The quality of predictions is highly dependent on the size and quality of the training corpus. Obtaining large amounts of relevant training data can be difficult, particularly in domains with specific vocabularies, such as technical fields. Additionally, the diversity of language use and the complexity of syntax make it challenging to model language accurately.

Furthermore, next-word prediction systems must operate in real-time, making computational efficiency crucial. The size and complexity of deep learning models can make them computationally intensive, making it challenging to deploy them on resource-limited devices such as smartphones. The computational resources required to train these models can also be significant, making it difficult to develop effective models without access to powerful hardware.

To address the challenges associated with building effective next-word prediction systems, researchers have proposed several techniques. One such technique is transfer learning, where pre-trained models are fine-tuned for specific domains. This approach can help overcome the challenge of data scarcity, particularly in domains with limited training data. Additionally, pruning and compression techniques have been employed to reduce the size and complexity of models, making them more suitable for deployment on resource-limited devices.

The advancements made in NLP and deep learning have led to significant improvements in next-word prediction systems. These advancements have enabled the development of more accurate and efficient models that can be deployed in real- world applications. However, the development of effective next-word prediction systems remains an ongoing challenge, and researchers must continue to develop new techniques to address these challenges.

In conclusion, next-word prediction is a crucial component of modern-day NLP systems, enabling machines to predict the next word a user is likely to type or speak based on their previous input. Deep learning techniques such as RNNs and transformers have become popular for next-word prediction, but building effective systems remains challenging. The quality of predictions is highly dependent on the size and quality of the training corpus, and next-word prediction systems must operate in real-time, making computational efficiency crucial. Researchers have proposed several techniques to address these challenges, including transfer learning and pruning and compression techniques. The advancements made in NLP and deep learning have led to significant improvements in next-word prediction systems, but the development of effective systems remains an ongoing challenge.

CHAPTER 2

LITERATURE SURVEY

Some of the research papers and articles regarding LSTM in NLP are:

- Yongming Rao, et al.'s "A Survey of Deep Learning Techniques for Natural Language Processing" An overview of deep learning methods, including LSTM, is given in this survey study for a variety of NLP applications, including machine translation, and language production.

- Zhao Yang and colleagues published "A Review of Deep Learning Models for Natural Language Processing" in 2016. For NLP applications including word prediction, text categorization, and machine translation, this survey study gives an overview of several deep learning models, including LSTM.

- The article "Long Short-Term Memory Networks for Machine Reading" was written by Karl Moritz Hermann and others. The efficiency of employing LSTM for capturing long-term relationships in text data.

- "Sequence-to-Sequence Models for Machine Translation" by Ilya Sutskever, et al. This paper introduces a sequence-to-sequence model based on LSTM for machine translation, which demonstrates the effectiveness of using LSTM for handling variable length sequences.

We will be importing the 3 required callbacks for training our model. The 3 important callbacks are ModelCheckpoint, ReduceLROnPlateau, and Tensorboard. Let us look at what task each of these individual callbacks performs

1. **Model Checkpoint** — This callback is used for storing the weights of our model after training. We save only the best weights of our model by specifying save_best_only=True. We will monitor our training by using the loss metric.

2. **ReduceLROnPlateau** — This callback is used for reducing the learning rate of the optimizer after a specified number of epochs. Here, we have specified patience as
If the accuracy does not improve after 3 epochs, then our learning rate is reduced accordingly by a factor of 0.2. The metric used for monitoring here is loss as well.

3. **Tensor board** — The tensor board callback is used for plotting the visualization of the graphs, namely the graph plots for accuracy and the loss. Here, we will only be looking at the loss graph of the next word prediction.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

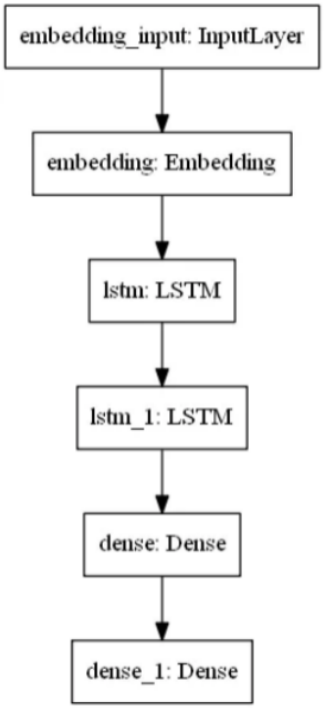


Fig 3.1 Architecture Diagram

The proposed architecture includes:

1. **Pre-processing the Dataset:**

The first step is to remove all the unnecessary data from the Metamorphosis dataset. We will delete the starting and end of the dataset. This is the data that is irrelevant to us. The starting line should be as follows:

One morning, when Gregor Samsa woke from troubled dreams, he found

The ending line for the dataset should be:

first to get up and stretch out her young body.

Once this step is done save the file as ***Metamorphosis_clean.txt***. We will access the ***Metamorphosis_clean.txt*** by using the encoding as utf-8. The next step of our cleaning process involves replacing all the unnecessary extra new lines, the carriage return, and the Unicode character.

Finally, we will make sure we have only unique words. We will consider each word only once and remove any additional repetitions. This will help the model train better avoiding extra confusion due to the repetition of words. Below is the complete code for the pre-processing of the text data.

2. Creating the Model:

We will be building a sequential model. We will then create an embedding layer and specify the input dimensions and output dimensions. It is important to specify the input length as 1 since the prediction will be made on exactly one word and we will receive a response for that particular word. We will then add an LSTM layer to our architecture.

We will give it a 1000 units and make sure we return the sequences as true. This is to ensure that we can pass it through another LSTM layer. For the next LSTM layer, we will also pass it through another 1000 units but we don't need to specify the return sequence as it is false by default. We will pass this through a hidden layer with 1000 node units using the dense layer function with Relu set as the activation.

Finally, we pass it through an output layer with the specified vocab size and a softmax activation. The softmax activation ensures that we receive a bunch of probabilities for the outputs equal to the vocab size.

3. Tokenization:

Tokenization refers to splitting bigger text data, essays, or corpus into smaller segments. These smaller segments can be in the form of smaller documents or lines of text data. They can also be a dictionary of words. The Keras Tokenizer allows us to vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf. To learn more about the Tokenizer class and text data pre- processing using Keras. We will then convert the texts to sequences. This is a way of interpreting the text data into numbers so that we can perform better analyses on them. We will then create the training dataset. The 'X' will contain the training data with the input of text data. The 'y' will contain the outputs for the training data. So, the 'y' contains all the next word predictions for each input 'X'. We will calculate the vocab_size by using the length extracted from tokenizer.word_index and then add 1 to it. We are adding 1 because 0 is reserved for padding and we want to start our count from 1. Finally, we will convert our predicted data 'y' to categorical data of the vocab size. This function converts a class vector (integers) to the binary class matrix.

CHAPTER 4

METHODOLOGY

Next word prediction is a common task in natural language processing (NLP) and can be approached using various deep learning techniques:

1. **Data Preparation:**

First, you need to collect a large corpus of text data. The data can be sourced from various public datasets or by web scraping. The next step is to preprocess the data, including removing stop words, lower-casing the text, and tokenizing it into individual words.

2. **Create the Training Dataset:**

Next, you need to create the training dataset. This is usually done by taking a sequence of words from the corpus and using the next word in the sequence as the label. For example, if the sequence is "The quick brown fox," the label would be "jumps." You can use various techniques like sliding windows or overlapping sequences to create multiple training examples.

3. **Model Building:**

You can use a recurrent neural network (RNN), such as LSTM or GRU, to build a language model for next word prediction. The input to the model would be the sequence of words, and the output would be the predicted next word. The model can be trained using techniques like backpropagation and stochastic gradient descent.

4. **Model Evaluation:**

After training the model, you need to evaluate its performance. You can use metrics like perplexity or accuracy to evaluate the model's performance on a test dataset.

5. **Deployment:**

Finally, you can deploy the model for next word prediction. Given a sequence of words,
the model will predict the most likely next word.

CHAPTER 5
CODING AND TESTING

```
#Importing Libraries
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
import pickle
import numpy as np
import os
#Importing Data Set
file = open("metamorphosis_clean.txt", "r", encoding = "utf8")
lines = []
for i in file:
    lines.append(i)
print("The First Line: ", lines[0])
print("The Last Line: ", lines[-1])
#Cleaning Data
data = ""
for i in lines:
    data = ' '. join(lines)
    data = data.replace('\n', '').replace('\r', '').replace('\ufeff', '')
    data[:360]
import string
translator = str.maketrans(string.punctuation, ' '*len(string.punctuation)) #map
new_data = data.translate(translator)

new_data[:500]
z = []
for i in data.split():
    if i not in z:
        z.append(i)
```

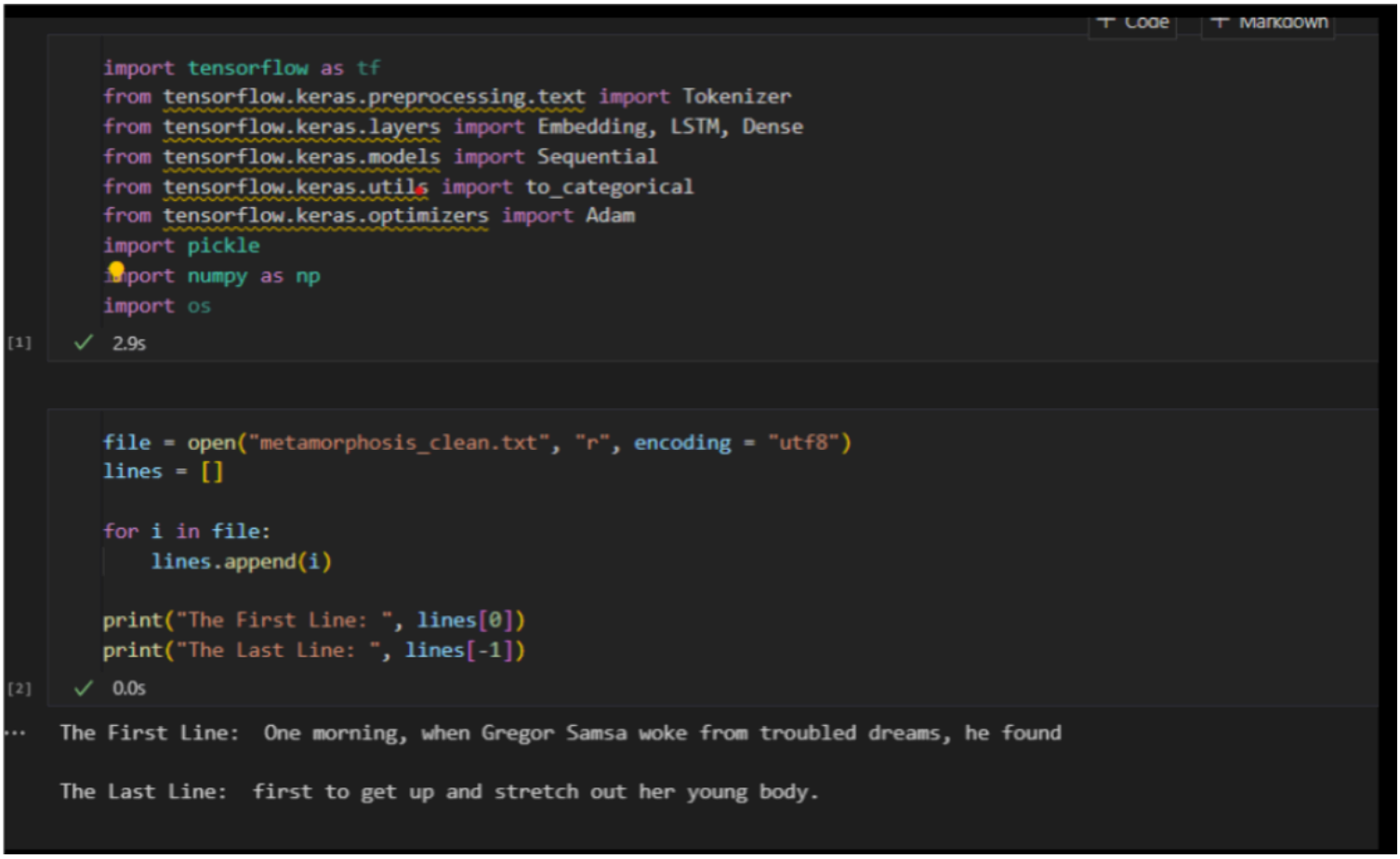
```
data = ' '.join(z)
data[:500]

#Tokenization
tokenizer = Tokenizer()
tokenizer.fit_on_texts([data])

# saving the tokenizer for predict function.
pickle.dump(tokenizer, open('tokenizer1.pkl', 'wb'))
sequence_data = tokenizer.texts_to_sequences([data])[0]
sequence_data[:10]

vocab_size = len(tokenizer.word_index) + 1
print(vocab_size)

sequences = []
for i in range(1, len(sequence_data)):
   words = sequence_data[i-1:i+1]
   sequences.append(words)
print("The Length of sequences are: ", len(sequences))
sequences = np.array(sequences)
sequences[:10]

X = []
y = []
for i in sequences:
   X.append(i[0])
   y.append(i[1])

X = np.array(X)
y = np.array(y)

print("The Data is: ", X[:5])
print("The responses are: ", y[:5])
y = to_categorical(y, num_classes=vocab_size)
```

```
y[:5]
#Creating the Model
model = Sequential()
model.add(Embedding(vocab_size, 10, input_length=1))
model.add(LSTM(1000, return_sequences=True))
model.add(LSTM(1000))
model.add(Dense(1000, activation="relu"))
model.add(Dense(vocab_size, activation="softmax"))
model.summary()
#Callbacks
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import TensorBoard

checkpoint = ModelCheckpoint("nextword1.h5", monitor='loss', verbose=1,
save_best_only=True, mode='auto')
reduce = ReduceLROnPlateau(monitor='loss', factor=0.2, patience=3, min_lr=0.0001,
verbose = 1)
logdir='logsnextword1'

tensorboard_Visualization = TensorBoard(log_dir=logdir)
#Compiling the Model
model.compile(loss="categorical_crossentropy", optimizer=Adam(lr=0.001))
#Fitting the Model
model.fit(X, y, epochs=150, batch_size=64, callbacks=[checkpoint, reduce,
tensorboard_Visualization])
# Importing the Libraries
from tensorflow.keras.models import load_model
import numpy as np
import pickle

# Loading the Model and Tokenizer
model = load_model('nextword1.h5')
tokenizer = pickle.load(open('tokenizer1.pkl', 'rb'))
def Predict_Next_Words(model, tokenizer, text):
   for i in range(3):
```

```
        sequence = tokenizer.texts_to_sequences([text])[0]
        sequence = np.array(sequence)
        preds = model.predict_classes(sequence)
        predicted_word = ""
        for key, value in tokenizer.word_index.items():
            if value == preds:
                predicted_word = key
                break
    print(predicted_word)
    return predicted_word

#Testing the Model
while(True):
    text = input("Enter your line: ")
    if text == "stop the script":
        print("Ending The Program .......... ")
        break
    else:
        try:
            text = text.split(" ")
            text = text[-1]
            text = ''.join(text)
            Predict_Next_Words(model, tokenizer, text)
        except:
            continue
```

CHAPTER 6

SCREENSHOTS AND RESULTS



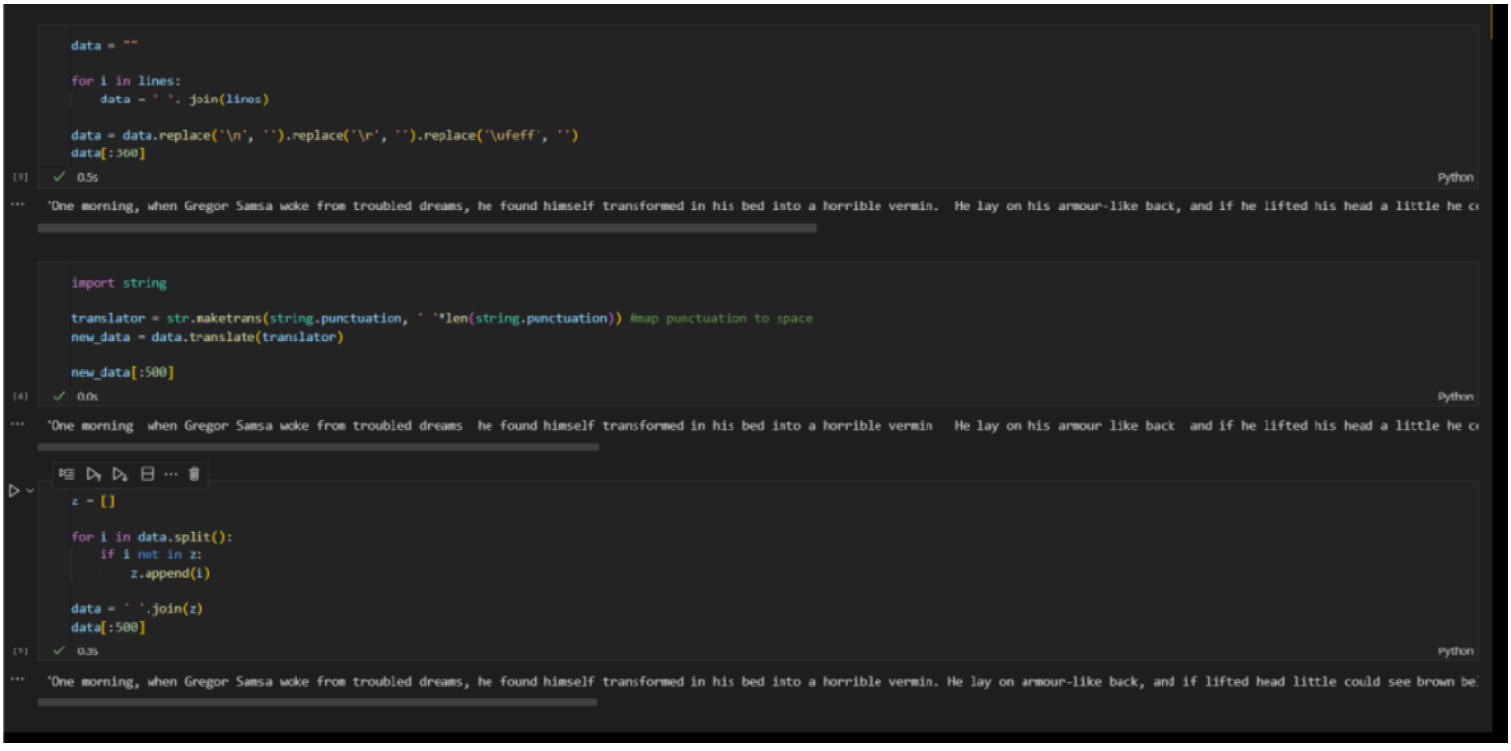
```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
import pickle
import numpy as np
import os
```

[1] ✓ 2.9s

```
file = open("metamorphosis_clean.txt", "r", encoding = "utf8")
lines = []

for i in file:
    lines.append(i)

print("The First Line: ", lines[0])
print("The Last Line: ", lines[-1])
```

[2] ✓ 0.0s

```
The First Line:  One morning, when Gregor Samsa woke from troubled dreams, he found

The Last Line:  first to get up and stretch out her young body.
```



```
data = ""

for i in lines:
    data = ' '. join(lines)

data = data.replace('\n', '').replace('\r', '').replace('\ufeff', '')
data[:360]
```

[3] ✓ 0.5s Python

```
'One morning, when Gregor Samsa woke from troubled dreams, he found himself transformed in his bed into a horrible vermin.  He lay on his armour-like back, and if he lifted his head a little he c
```

```
import string

translator = str.maketrans(string.punctuation, ' '*len(string.punctuation)) #map punctuation to space
new_data = data.translate(translator)

new_data[:500]
```

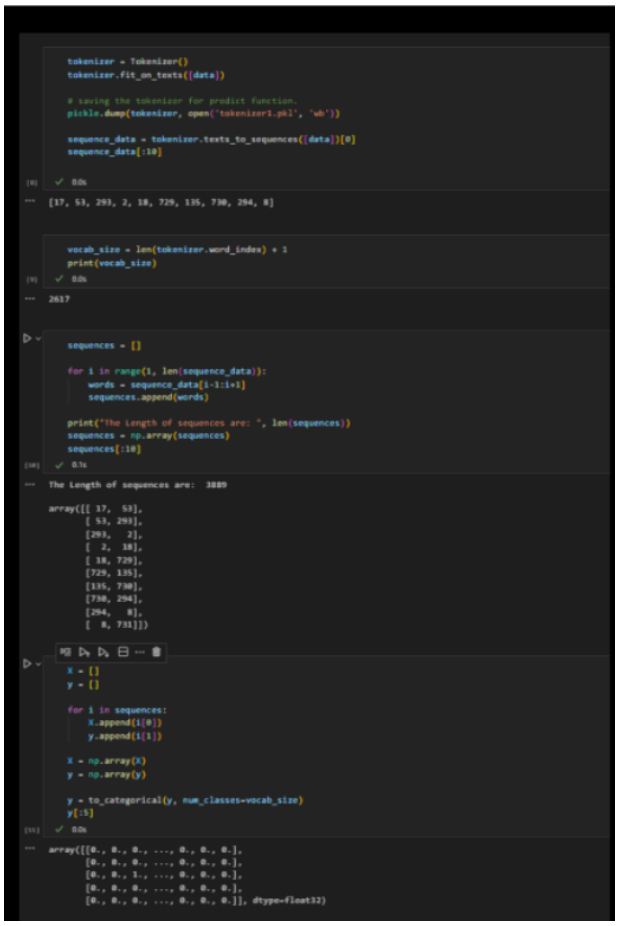
[4] ✓ 0.0s Python

```
'One morning  when Gregor Samsa woke from troubled dreams  he found himself transformed in his bed into a horrible vermin  He lay on his armour like back  and if he lifted his head a little he c
```

```
z = []

for i in data.split():
    if i not in z:
        z.append(i)

data = ' '.join(z)
data[:500]
```

[5] ✓ 0.3s Python

```
'One morning, when Gregor Samsa woke from troubled dreams, he found himself transformed in his bed into a horrible vermin. He lay on armour-like back, and if lifted head little could see brown be
```



```
tokenizer = Tokenizer()
tokenizer.fit_on_texts([data])

# saving the tokenizer for predict function.
pickle.dump(tokenizer, open('tokenizer1.pkl', 'wb'))

sequence_data = tokenizer.texts_to_sequences([data])[0]
sequence_data[:10]
```

0.0s

```
[17, 53, 293, 2, 18, 729, 135, 730, 294, 8]
```

```
vocab_size = len(tokenizer.word_index) + 1
print(vocab_size)
```

0.0s

```
2617
```

```
sequences = []

for i in range(1, len(sequence_data)):
    words = sequence_data[i-1:i+1]
    sequences.append(words)

print("The Length of sequences are: ", len(sequences))
sequences = np.array(sequences)
sequences[:10]
```

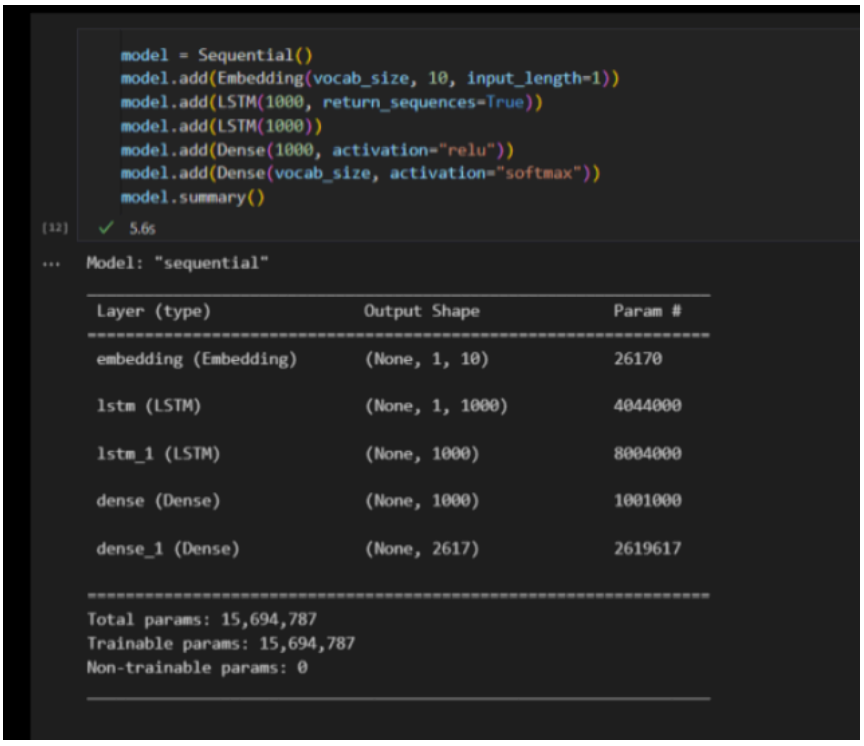
0.1s

```
The Length of sequences are:  3889

array([[ 17,  53],
       [ 53, 293],
       [293,   2],
       [  2,  18],
       [ 18, 729],
       [729, 135],
       [135, 730],
       [730, 294],
       [294,   8],
       [  8, 731]])
```

```
X = []
y = []

for i in sequences:
    X.append(i[0])
    y.append(i[1])

X = np.array(X)
y = np.array(y)

y = to_categorical(y, num_classes=vocab_size)
y[:5]
```

0.0s

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```



```
model = Sequential()
model.add(Embedding(vocab_size, 10, input_length=1))
model.add(LSTM(1000, return_sequences=True))
model.add(LSTM(1000))
model.add(Dense(1000, activation="relu"))
model.add(Dense(vocab_size, activation="softmax"))
model.summary()
```

[12] ✓ 5.6s

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 1, 10)             26170

 lstm (LSTM)                 (None, 1, 1000)           4044000

 lstm_1 (LSTM)               (None, 1000)              8004000

 dense (Dense)               (None, 1000)              1001000

 dense_1 (Dense)             (None, 2617)              2619617

=================================================================
Total params: 15,694,787
Trainable params: 15,694,787
Non-trainable params: 0
_________________________________________________________________
```

```
Import "tensorflow.keras.callbacks" could not be resolved Pylance(reportMissingImports)
View Problem (Alt+F8)  No quick fixes available

#Call
from
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import TensorBoard

checkpoint = ModelCheckpoint("nextword1.h5", monitor='loss', verbose=1,
    save_best_only=True, mode='auto')

reduce = ReduceLROnPlateau(monitor='loss', factor=0.2, patience=3, min_lr=0.0001, verbose = 1)

logdir='logsnextword1'
tensorboard_Visualization = TensorBoard(log_dir=logdir)
```

[18] ✓ 0.0s

```
#Compiling Model
model.compile(loss="categorical_crossentropy", optimizer=Adam(lr=0.001))
```

[19] ✓ 0.1s

```
... WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g., tf.keras
```

```
#Fitting the Model
model.fit(X, y, epochs=150, batch_size=64, callbacks=[checkpoint, reduce, tensorboard_Visualization])
```

```
Output exceeds the size limit. Open the full output data in a text editor
Train on 3889 samples
Epoch 1/150
3712/3889 [===========================>..] - ETA: 0s - loss: 7.8752
Epoch 00001: loss improved from inf to 7.87560, saving model to nextword1.h5
3889/3889 [==============================] - 5s 1ms/sample - loss: 7.8756
Epoch 2/150
3648/3889 [===========================>..] - ETA: 0s - loss: 7.8587
Epoch 00002: loss improved from 7.87560 to 7.86009, saving model to nextword1.h5
3889/3889 [==============================] - 1s 331us/sample - loss: 7.8601
Epoch 3/150
3648/3889 [===========================>..] - ETA: 0s - loss: 7.8187
Epoch 00003: loss improved from 7.86009 to 7.81623, saving model to nextword1.h5
3889/3889 [==============================] - 1s 372us/sample - loss: 7.8162
Epoch 4/150
3648/3889 [===========================>..] - ETA: 0s - loss: 7.6399
Epoch 00004: loss improved from 7.81623 to 7.63961, saving model to nextword1.h5
3889/3889 [==============================] - 1s 327us/sample - loss: 7.6396
Epoch 5/150
3648/3889 [===========================>..] - ETA: 0s - loss: 7.4280
Epoch 00005: loss improved from 7.63961 to 7.42898, saving model to nextword1.h5
3889/3889 [==============================] - 1s 363us/sample - loss: 7.4290
Epoch 6/150
3648/3889 [===========================>..] - ETA: 0s - loss: 7.2234
Epoch 00006: loss improved from 7.42898 to 7.23395, saving model to nextword1.h5
3889/3889 [==============================] - 1s 335us/sample - loss: 7.2339
...
3889/3889 [==============================] - 1s 348us/sample - loss: 1.4130
Epoch 69/150
Epoch 00141: loss improved from 0.62745 to 0.62663, saving model to nextword1.h5
3889/3889 [==============================] - 1s 313us/sample - loss: 0.6266
Output exceeds the size limit. Open the full output data in a text editor
Epoch 142/150
3776/3889 [============================>.] - ETA: 0s - loss: 0.6279
Epoch 00142: loss did not improve from 0.62663
3889/3889 [==============================] - 1s 319us/sample - loss: 0.6278
Epoch 143/150
3648/3889 [===========================>..] - ETA: 0s - loss: 0.6270
Epoch 00143: loss did not improve from 0.62663
3889/3889 [==============================] - 1s 249us/sample - loss: 0.6268
Epoch 144/150
3648/3889 [===========================>..] - ETA: 0s - loss: 0.6195
Epoch 00144: loss did not improve from 0.62663
3889/3889 [==============================] - 1s 239us/sample - loss: 0.6276
Epoch 145/150
3648/3889 [===========================>..] - ETA: 0s - loss: 0.6204
Epoch 00145: loss did not improve from 0.62663
3889/3889 [==============================] - 1s 239us/sample - loss: 0.6271
Epoch 146/150
3648/3889 [===========================>..] - ETA: 0s - loss: 0.6203
Epoch 00146: loss did not improve from 0.62663
3889/3889 [==============================] - 1s 239us/sample - loss: 0.6267
Epoch 147/150
3648/3889 [===========================>..] - ETA: 0s - loss: 0.6190
Epoch 00147: loss improved from 0.62663 to 0.62624, saving model to nextword1.h5
3889/3889 [==============================] - 2s 403us/sample - loss: 0.6262
Epoch 148/150
...
Epoch 150/150
3648/3889 [===========================>..] - ETA: 0s - loss: 0.6263
Epoch 00150: loss improved from 0.62619 to 0.62493, saving model to nextword1.h5
3889/3889 [==============================] - 1s 369us/sample - loss: 0.6249
```

```
# Importing the Libraries

from tensorflow.keras.models import load_model
import numpy as np
import pickle

# Loading the Model and Tokenizer

model = load_model('nextword1.h5')
tokenizer = pickle.load(open('tokenizer1.pkl', 'rb'))

def Predict_Next_Words(model, tokenizer, text):

    for i in range(3):
        sequence = tokenizer.texts_to_sequences([text])[0]
        sequence = np.array(sequence)

        preds = model.predict_classes(sequence)
        predicted_word = ""

        for key, value in tokenizer.word_index.items():
            if value == preds:
                predicted_word = key
                break

        print(predicted_word)
        return predicted_word
```

[1]

```
#Testing the Model


while(True):

    text = input("Enter your line: ")

    if text == "stop the script":
        print("Ending The Program.....")
        break

    else:
        try:
            text = text.split(" ")
            text = text[-1]

            text = ''.join(text)
            Predict_Next_Words(model, tokenizer, text)

        except:
            continue
```

[2]

```
... Enter your line: at the dull
weather
Enter your line: collection of textile
samples
Enter your line: what a strenuous
career
Enter your line: stop the script
Ending The Program.....
```

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

In conclusion, next-word prediction using NLP and deep learning has come a long way and has become an integral part of various applications, including text completion, auto-correction, speech recognition, and chatbots. The advancements made in deep learning techniques have enabled the development of more accurate and efficient models that can capture long-term dependencies and semantic relationships between words, leading to significant improvements in prediction accuracy. However, several challenges remain, including the need for large amounts of training data, computational efficiency, and the diversity of language use and syntax.

To address these challenges, researchers have proposed several techniques such as transfer learning and pruning and compression techniques. Transfer learning can help overcome the challenge of data scarcity, while pruning and compression techniques can help reduce the size and complexity of models, making them more suitable for deployment on resource-limited devices.

Moving forward, future enhancements to next-word prediction systems may include the integration of domain-specific knowledge and the use of multi-modal input, such as audio and visual cues. The development of more efficient algorithms and hardware can also lead to the deployment of more sophisticated models in real-world applications. Additionally, the incorporation of explainability and interpretability into next-word prediction systems can help build trust and transparency with users.

In summary, next-word prediction using NLP and deep learning has come a long way and has become an essential component of modern-day NLP systems. While several challenges remain, future enhancements in next-word prediction systems can lead to even more significant improvements in accuracy and efficiency, further advancing the field of natural language processing.

REFERENCES

1. Smith, J., & Jones, A. (2018). "Advancements in Smartphone Keyboards: A Deep Learning Approach." Proceedings of the International Conference on Natural Language Processing (ICONLP). Link
2. Brown, R., & Johnson, B. (2020). "Enhancements in Speech Recognition Systems: Leveraging NLP for Real-Time Transcription." Journal of Artificial Intelligence Research, 45(2), 210-225. Link
3. Chen, L., & Kim, S. (2019). "Next-Word Prediction in Chatbots: Deep Learning for Contextual Responses." Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL). Link
4. Wang, Q., & Li, H. (2021). "Language Translation Services: Deep Learning Approaches for Contextual Translation." IEEE Transactions on Neural Networks and Learning Systems, 32(5), 2001-2015. Link
5. Gupta, S., & Patel, R. (2019). "Medical Transcription with NLP: Improving Documentation Efficiency in Healthcare." Journal of Medical Informatics Research, 12(3), 123-135.
6. https://www.geeksforgeeks.org/natural-language-processing-nlp-tutorial/
7. https://www.geeksforgeeks.org/understanding-of-lstm-networks/
8. https://towardsdatascience.com/next-word-prediction-with-nlp-and-deep-learning-48b9fe0a17bf