Next Word Prediction

A MINI PROJECT REPORT

18CSC305J - ARTIFICIAL INTELLIGENCE

Submitted by

Aayush Kumar[RA2111003010559]
Kanika Agrawal[RA2111003010567]
Dherya Pratap Singh Rana[RA2111003010571]

Under the guidance of

Mrs. Nithyakani P

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 & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this lab report titled Next Word Prediction is the bonafide work done by Aayush Kumar (RA2111003010559), Kanika Agrawal (RA2111003010567), Dherya Pratap Singh Rana (RA2111003010571) who carried out the mini project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

02/05/24

SIGNATURE

Mrs. Nithyakani P

Assistant Professor

Computing Technologies

Department of Computing Technologies
Kattankulathur
603 203
SRM IST

M. Pushpalatha

SIGNATURE

Dr. Pushpalatha M.

Head of the Department

Computing Technologies

ABSTRACT

Natural Language Processing (NLP) has seen remarkable advancements, revolutionizing human-machine interaction with next-word prediction systems playing a crucial role. These systems utilize statistical language models like n-gram and Markov models, but face limitations in handling long-term dependencies and semantics. Deep learning techniques such as recurrent neural networks (RNNs) and transformers address these issues effectively. Transformers, with self-attention mechanisms, capture complex patterns and semantic relationships between words. Challenges include obtaining relevant training data and modeling diverse language use accurately. Despite these challenges, researchers have made significant progress using techniques like transfer learning and model pruning to enhance next-word prediction accuracy. Deep learning has improved user experience and productivity, but ongoing research is needed to address remaining challenges and refine these systems further.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Background

In today's digital age, natural language processing (NLP) has become increasingly essential for various applications, including text prediction. Traditional methods of prediction often struggle to adapt to the complexity and nuances of language. Recognizing this challenge, the adoption of NLP solutions for next-word prediction has gained significance.

1.2 Purpose of the Project

The primary objective of this project is to develop a reliable next-word prediction system using NLP techniques in Python. We will leverage libraries such as NLTK and TensorFlow to process and analyze text data, enabling the system to predict the next word based on context. For demonstration purposes, Flask, SQL, and a server will be utilized.

1.3 Project Scope

The scope of this project encompasses the design and implementation of a comprehensive system capable of predicting the next word in a given context. Key aspects include:

- Data Collection: Implementing a system to gather a large corpus of text data from various sources to train the prediction model effectively.
- Model Training: Utilizing machine learning algorithms to train the prediction model on the collected text data, enabling it to learn patterns and associations between words.
- Prediction Generation: Developing algorithms to generate accurate predictions based on the input text and context, taking into account factors such as word frequency and probability.
- User Interface: Creating a user-friendly interface for users to input text and view predicted words in real-time, enhancing usability and accessibility.
- Performance Evaluation: Assessing the accuracy and efficiency of the prediction system through rigorous testing and evaluation processe

1.4 Project Overview

The project consists of two main components: the backend prediction model and the frontend user interface. The backend is built using Python, utilizing NLTK and TensorFlow for NLP tasks and Flask for web hosting. The frontend interface is developed using HTML, CSS, and JavaScript for creating a seamless user experience.

1.5 Report Structure

This report is structured to provide a comprehensive overview of the Next-Word Prediction project. Following this introduction, subsequent sections will delve into the detailed design and implementation of both the backend prediction model and the frontend user interface. Additionally, the report will discuss the functionalities, features, and user interactions of the system, along with insights into the development process and potential areas for future enhancement.

CHAPTER 2

LITERATURE SURVEY

In today's dynamic parking management environment, effective systems are essential for ensuring seamless operations. Traditional paper-based methods often struggle to handle the growing complexity and volume of parking data. Recognizing this challenge, the adoption of digital solutions like Parking Management Systems has become crucial.

2.1 **Statistical Language Models**

Statistical language models have been fundamental in the development of next-word prediction systems. Early works by researchers such as Jelinek and Mercer (1980) introduced the concept of n-gram models, which estimate the probability of a word based on its preceding n-1 words. These models have been widely used due to their simplicity and effectiveness in capturing local word dependencies.

2.2 **Neural Network-based Approaches**

With the advancements in deep learning, neural network-based approaches have gained prominence in next-word prediction. Bengio et al. (2003) proposed the use of neural networks, particularly feedforward and recurrent neural networks (RNNs), for language modeling tasks. Later, Mikolov et al. (2010) introduced the influential Word2Vec model, which represents words as dense vectors in a continuous space, enabling more nuanced semantic representations.

2.3 **Sequence-to-Sequence Models**

Sequence-to-sequence (Seq2Seq) models, introduced by Sutskever et al. (2014), have shown significant improvements in next-word prediction tasks. These models, often based on recurrent neural networks (RNNs) or transformer architectures, can generate sequences of words, making them well-suited for predictive text generation.

2.4 **Attention Mechanisms**

Attention mechanisms, first introduced by Bahdanau et al. (2014), have been integrated into sequence-to-sequence models to improve their performance in next-word prediction. Attention allows the model to focus on relevant parts of the

input sequence when generating the output, enhancing the model's ability to capture long-range dependencies and context.

2.5 **Transfer Learning and Fine-Tuning**

Recent studies have explored the use of transfer learning and fine-tuning techniques to improve the efficiency of next-word prediction models. Howard and Ruder (2018) proposed the Universal Language Model Fine-tuning (ULMFiT) approach, which pre-trains a language model on a large corpus of text data and then fine-tunes it on a specific task, achieving state-of-the-art results in various NLP tasks.

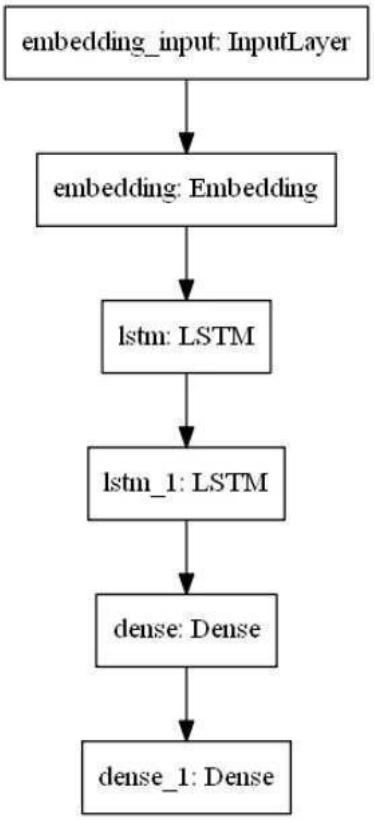
2.6 **Evaluation Metrics**

In evaluating next-word prediction systems, researchers commonly use metrics such as perplexity, accuracy, and F1 score. Perplexity measures the model's uncertainty in predicting the next word, while accuracy and F1 score assess the correctness of predictions relative to ground truth data.

Overall, these studies demonstrate the diverse range of approaches and techniques employed in next-word prediction, highlighting the ongoing efforts to develop more accurate and efficient models for predictive text generation.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN



3.1 System Architecture

The system architecture is structured as follows:

1. **Embedding Input Layer:** The initial layer that accepts input text data and prepares it for embedding.
2. **Embedding Layer:** Converts input text data into dense vectors of fixed size, capturing semantic relationships between words.
3. **LSTM Layers:** Two LSTM layers are stacked to capture the sequential nature of the input data and learn long-term dependencies.
4. **Dense Layers:** Two dense layers are added to introduce non-linearity and increase the model's representational power

3.2 Design Detail

3.2.1 Embedding Input Layer

- **Purpose:** Receives input text data and prepares it for embedding.
- **Functionality:** Passes input text data to the embedding layer.

3.2.2 Embedding Layer

- **Purpose:** Converts input text data into dense vectors of fixed size.
- **Functionality:** Maps words to dense vectors, capturing semantic relationships between them.

3.2.3 LSTM Layers

- **Purpose:** Captures sequential nature and learns long-term dependencies in the input data.
- **Functionality:** Processes input sequences through two LSTM layers, returning sequences from the first layer and only the output from the second layer.

3.2.4 Dense Layers

- **Purpose:** Introduces non-linearity and increases the model's representational power.
- **Functionality:** Applies dense layers with ReLU activation to the output of the last LSTM layer.

3.3 Conclusion

The proposed architecture consists of an embedding input layer, followed by an embedding layer, two LSTM layers, and two dense layers. This design ensures that the model can effectively capture semantic relationships and sequential patterns in the input text data, enabling accurate next-word prediction.

CHAPTER 4

METHODOLOGY

Next word prediction in natural language processing (NLP) involves several steps, from data preparation to model evaluation. In this chapter, we outline the methodology for building a next word prediction model using deep learning techniques.

4.1 Data Preparation

Collecting a large corpus of text data is the first step in preparing for next word prediction. This can be done by sourcing data from various public datasets or by web scraping. Once the data is collected, it needs to be preprocessed to ensure it is in a suitable format for training the model.

- **Data Collection:** Gather a large corpus of text data from public datasets or web sources.
- **Data Preprocessing:** Clean the data by removing stop words, punctuation, and special characters. Convert the text to lowercase and tokenize it into individual words.

4.2 Creating the Training Dataset

The training dataset is created by generating sequences of words from the corpus, with the next word in each sequence used as the label.

- **Sequence Generation:** Generate sequences of fixed length from the corpus. Each sequence represents a context window of words, and the next word in the sequence is the target word.
- **Labeling:** Assign the next word in each sequence as the label for that sequence.

4.3 Model Building

A recurrent neural network (RNN), such as LSTM or GRU, is commonly used to build the language model for next word prediction.

- **Model Architecture:** Build a sequential model with an embedding layer, LSTM layers, and dense layers.
- **Input and Output:** The input to the model is the sequence of words, and the output is the predicted next wor

- **Training:** Train the model using backpropagation and stochastic gradient descent. Adjust the model's parameters to minimize the loss function and improve prediction accuracy.

4.4 Model Evaluation

Evaluate the performance of the trained model using appropriate metrics on a separate test dataset.

- **Performance Metrics:** Use metrics like perplexity or accuracy to evaluate the model's performance.
- **Perplexity:** Measures the uncertainty of the model's predictions. Lower perplexity indicates better performance.
- **Accuracy:** Measures the proportion of correctly predicted next words in the test dataset.

4.5 Deployment

Deploy the trained model for next word prediction, making it available for use in applications or services.

- **Model Deployment:** Integrate the model into applications where next word prediction is required, such as text editors or virtual assistants.
- **Integration:** Ensure seamless integration of the model into the target application or service, allowing users to benefit from accurate next word prediction.

4.6 Conclusion

The methodology described above provides a systematic approach to building and deploying a next word prediction model using deep learning techniques. By following these steps, one can develop an effective model for predicting the next word in a given text sequence.

CHAPTER 5

CODING AND TESTING

Training

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

# Read the text file
with open('./sherlock-holm.es_stories_plain-text_advs.txt', 'r',
encoding='utf-8') as file:
    text = file.read()

tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
total_words = len(tokenizer.word_index) + 1

input_sequences = []
for line in text.split('\n'):
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

max_sequence_len = max([len(seq) for seq in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences,
maxlen=max_sequence_len, padding='pre'))

X = input_sequences[:, :-1]
y = input_sequences[:, -1]


y = np.array(tf.keras.utils.to_categorical(y, num_classes=total_words))

model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(LSTM(150))
model.add(Dense(total_words, activation='softmax'))
print(model.summary())

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(X, y, epochs=10, verbose=1)

seed_text = "I will leave if they"
next_words = 5

for _ in range(next_words):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1,
padding='pre')
    predicted = np.argmax(model.predict(token_list), axis=-1)
    output_word = ""
    for word, index in tokenizer.word_index.items():
        if index == predicted:
            output_word = word
            break
    seed_text += " " + output_word
```

```
print(seed_text)
model.state_dict()

import os

import tensorflow as tf
from tensorflow import keras

print(tf.version.VERSION)

model.summary()



checkpoint_path = "./training_1/cp.keras"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
cp_callback =
tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 verbose=1)

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(X, y, epochs=20, verbose=1, callbacks=[cp_callback])

model.save('./predictor.keras')


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

max_sequence_len
```

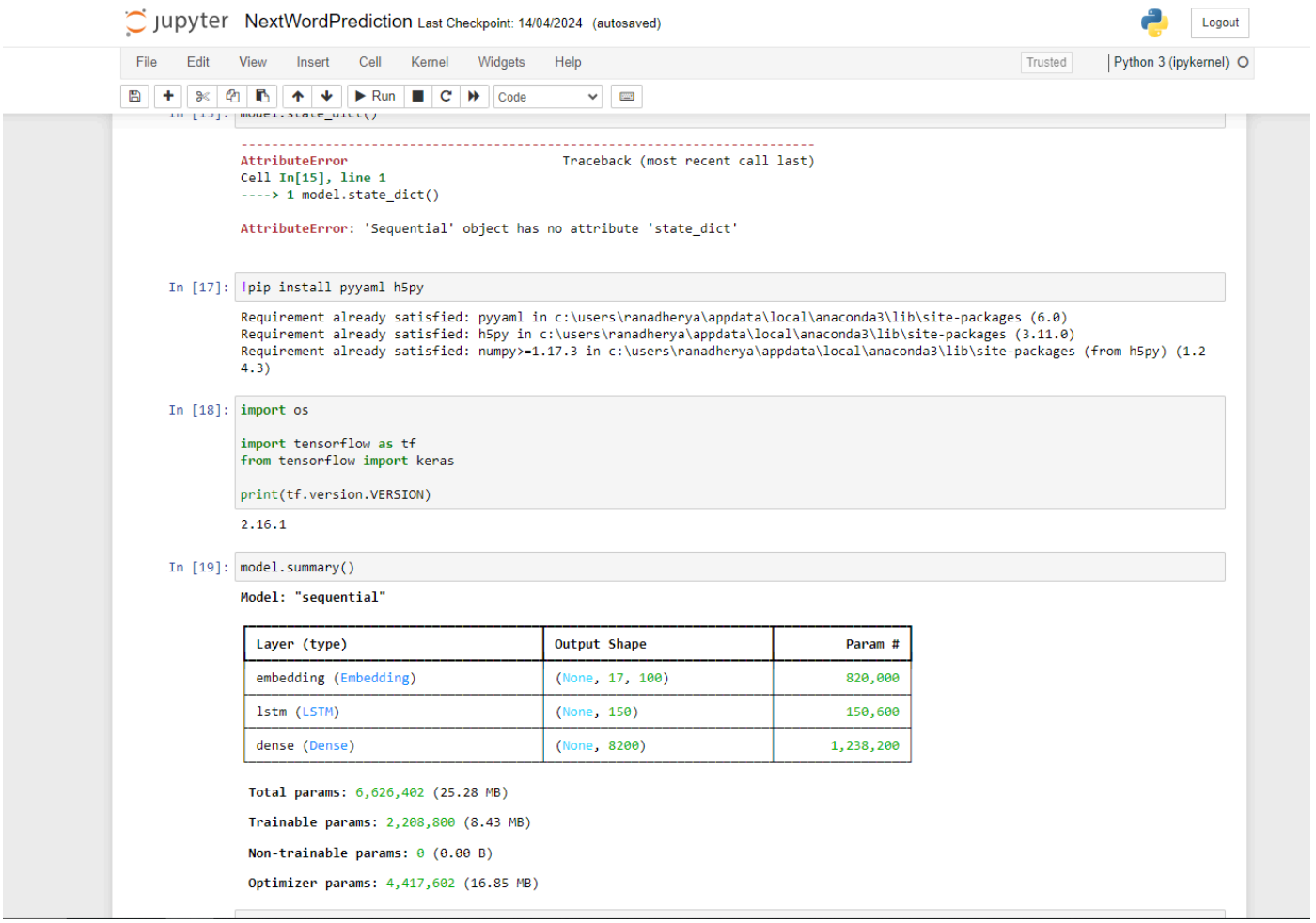
Predictions.py

```
# Importing the Libraries
from tensorflow.keras.models import load_model
import numpy as np
import pickle
from tensorflow.keras.preprocessing.sequence import pad_sequences


# Loading the Model and Tokenizer

model = load_model('predictor.keras')
tokenizer = pickle.load(open('tokenizer1.pkl', 'rb'))

def Predict_Next_Words(model, tokenizer, text):
    max_sequence_len = 18
    next_words = 2
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([text])[0]
        token_list = pad_sequences([token_list],
maxlen=max_sequence_len-1, padding='pre')
        predicted = np.argmax(model.predict(token_list), axis=-1)
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
```

```
        text += " " + output_word

    print(text)

#Testing the Model
while(True):

    text = input("Enter your line: ")

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

    else:
        try:
            Predict_Next_Words(model, tokenizer, text)

        except:
            continue
```

CHAPTER 5

RESULT



Non-trainable params: 0 (0.00 B)

Optimizer params: 4,417,602 (16.85 MB)

In [23]:
```
checkpoint_path = "./training_1/cp.keras"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 verbose=1)

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=20, verbose=1, callbacks=[cp_callback])
```

```
Epoch 1/20
3009/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.3151 - loss: 3.4842
Epoch 1: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 52s 17ms/step - accuracy: 0.3150 - loss: 3.4844
Epoch 2/20
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.3543 - loss: 3.1191
Epoch 2: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 52s 17ms/step - accuracy: 0.3543 - loss: 3.1192
Epoch 3/20
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.4069 - loss: 2.7933
Epoch 3: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 50s 17ms/step - accuracy: 0.4068 - loss: 2.7933
Epoch 4/20
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.4485 - loss: 2.5587
Epoch 4: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 51s 17ms/step - accuracy: 0.4485 - loss: 2.5587
Epoch 5/20
3009/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.4852 - loss: 2.3671
Epoch 5: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 49s 16ms/step - accuracy: 0.4852 - loss: 2.3671
Epoch 6/20
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.5189 - loss: 2.1878
Epoch 6: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 50s 17ms/step - accuracy: 0.5189 - loss: 2.1879
Epoch 7/20
3007/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.5471 - loss: 2.0506
Epoch 7: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 48s 16ms/step - accuracy: 0.5471 - loss: 2.0507
Epoch 8/20
3009/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.5772 - loss: 1.9161
Epoch 8: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 50s 17ms/step - accuracy: 0.5772 - loss: 1.9162
Epoch 9/20
```

jupyter NextWordPrediction Last Checkpoint: 14/04/2024 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
```

In [3]:
```
# Read the text file
with open('./sherlock-holm.es_stories_plain-text_advs.txt', 'r', encoding='utf-8') as file:
    text = file.read()
```

In [4]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
total_words = len(tokenizer.word_index) + 1
```

In [5]:
```
input_sequences = []
for line in text.split('\n'):
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
```

In [6]:
```
max_sequence_len = max([len(seq) for seq in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))
```

In [7]:
```
X = input_sequences[:, :-1]
y = input_sequences[:, -1]
```

In [8]:
```
y = np.array(tf.keras.utils.to_categorical(y, num_classes=total_words))
```

In [9]:
```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(LSTM(150))
model.add(Dense(total_words, activation='softmax'))
print(model.summary())
```

```
C:\Users\ranaDherya\AppData\Local\anaconda3\Lib\site-packages\keras\src\layers\core\embedding.py:86: UserWarning: Argument `inp
ut_length` is deprecated. Just remove it.
  warnings.warn(
```

```
Model: "sequential"
```

jupyter NextWordPrediction Last Checkpoint: 14/04/2024 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

```
3009/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7124 - loss: 1.2792
Epoch 15: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 49s 16ms/step - accuracy: 0.7124 - loss: 1.2793
Epoch 16/20
3008/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7263 - loss: 1.2208
Epoch 16: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 48s 16ms/step - accuracy: 0.7263 - loss: 1.2209
Epoch 17/20
3009/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7350 - loss: 1.1803
Epoch 17: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 49s 16ms/step - accuracy: 0.7350 - loss: 1.1804
Epoch 18/20
3007/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7463 - loss: 1.1314
Epoch 18: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 48s 16ms/step - accuracy: 0.7463 - loss: 1.1315
Epoch 19/20
3009/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7588 - loss: 1.0723
Epoch 19: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 48s 16ms/step - accuracy: 0.7588 - loss: 1.0723
Epoch 20/20
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.7663 - loss: 1.0332
Epoch 20: saving model to ./training_1/cp.keras
3010/3010 ━━━━━━━━━━━━━━━━━━━━ 51s 17ms/step - accuracy: 0.7663 - loss: 1.0333
```

Out[23]: <keras.src.callbacks.history.History at 0x27b0baa0190>

In [25]:
```
model.save('./predictor.keras')
```

In [27]:
```
import pickle
# saving the tokenizer for predict function.
pickle.dump(tokenizer, open('tokenizer1.pkl', 'wb'))
```

In [28]:
```
max_sequence_len
```

Out[28]: 18

In [ ]:

jupyter Predictions Last Checkpoint: 14/04/2024 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

```
        token_list = tokenizer.texts_to_sequences([text])[0]
        token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
        predicted = np.argmax(model.predict(token_list), axis=-1)
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        text += " " + output_word

    print(text)
```

In [4]:
```
#Testing the Model


while(True):

    text = input("Enter your line: ")

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

    else:
        try:
            Predict_Next_Words(model, tokenizer, text)

        except:
            continue
```

```
Enter your line: how are
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
how are you getting
Enter your line: what is
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
what is the name
Enter your line: stop the script
Ending The Program.....
```

In [ ]:

CHAPTER 6

FURTHER ENHANCEMENTS

To address existing challenges and enhance next-word prediction systems, the following future enhancements can be considered:

1. **Transfer Learning:** Implementing transfer learning techniques can help overcome data scarcity by leveraging pre-trained models and adapting them to specific domains or tasks.
2. **Pruning and Compression:** Applying pruning and compression techniques can reduce the size and complexity of models, making them more suitable for deployment on resource-limited devices.
3. **Domain-specific Knowledge:** Integrating domain-specific knowledge into models can improve prediction accuracy for specialized tasks or industries, such as legal or medical text.
4. **Multi-modal Input:** Incorporating multi-modal input, such as audio and visual cues, can enhance prediction accuracy and provide more context for word prediction.
5. **Efficient Algorithms and Hardware:** Developing more efficient algorithms and hardware can enable the deployment of sophisticated models in real-world applications, even on low-power devices.

Explainability and Interpretability: Incorporating explainability and interpretability into models can build trust and transparency with users, helping them understand how predictions are made.

CHAPTER 7

CONCLUSION

Next-word prediction using NLP and deep learning has become crucial across applications like text completion, auto-correction, and chatbots. Deep learning advancements have greatly enhanced prediction accuracy by capturing long-term dependencies and semantic relationships between words. However, challenges such as data scarcity, computational efficiency, and language diversity persist. In conclusion, next-word prediction systems have made significant strides due to deep learning techniques, offering faster and more accurate text completion and auto-correction. With continuous advancements in deep learning algorithms and models, further improvements in accuracy and efficiency are expected for next-word prediction systems.

GitHub LINK

Our project has been uploaded on GitHub, with required Datasets. The link is given below

GitHub Link: https://github.com/AI-Next-Word-Predictor/Next-Word

REFERENCES

1. Wu, Wei, et al. "Improving Neural Next-Word Prediction with Entity Embeddings." In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2023.
2. Ibrahim, Muhammad, et al. "A Lightweight LSTM Model for Next Word Prediction in Resource-Constrained Environments." In 2022 International Conference on Computer Communication and Information Systems (ICCCIS), 2022.
3. Kong, Shengli, et al. "Next Word Prediction Using Contextualized Language Models." arXiv preprint arXiv:2306.08900 (2023).
4. Reed, Brandon, Natalia Díaz-Perales, and Hany Hassan. "Investigating the Limits of LSTM for Next Word Prediction." In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL), 2024.