AN ENHANCED OPTIMIZATION METHOD FOR PREDICTING ENGLISH PHRASES DURING SENTENCE FORMATION USING DEEP LEARNING

Mini project report

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

Bachelor of Technology

In

DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

By

P.lakshmi priya	**20AG1A6646**
M.Pooja	**21AG5A6606**
G.Abhishek	**20AG1A6616**

Under the Esteemed Guidance of

Dr. KAVITHA SOPPARI

Head of the Department of CSE (AIML)

Department of CSE (Artificial Intelligence & Machine Learning)

ACE ENGINEERING COLLEGE

An Autonomous Institution

(NBA ACCREDITED B.TECH COURSES: EEE, ECE,MECH, CIVIL & CSE, ACCORDED NAAC ‘A’ GRADE)

(Affiliated to Jawaharlal Nehru Technological University, Hyderabad, Telangana)

Ghatkesar, Hyderabad – 501 301

DECEMBER 2023

ACE Engineering College

An Autonomous Institution

(NBA ACCREDITED B. TECH COURSES: EEE, ECE, MECH, CIVIL & CSE, ACCORDED NAAC 'A' GRADE)

(Affiliated to Jawaharlal Nehru Technological University, Hyderabad, Telangana)

Ghatkesar, Hyderabad – 501 301

Website : www.aceec.ac.in E-mail: info@aceec.ac.in

CERTIFICATE

This is to certify that the Mini Project work entitled **"A ENHANCED OPTIMIZATION METHOD FOR PREDICTING ENGLISH PHRASES DURING SENTENCE FORMATION USING DEEP LEARNING"** is being submitted **by P.Lakshmi priya (20AG1A66), M.Pooja (20AG1A66), G.Abhishek(20AG1A6616)** in partial fulfillment for the award of Degree of **BACHELOR OF TECHNOLOGY** in **DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)** to the Jawaharlal Nehru Technological University, Hyderabad during the academic year 2023-24 is arecord of Bonafide work carried out by him under our guidance and supervision. The results embodied in this report have not been submitted by the student to any other University or Institutionfor the award of any degree or diploma.

Internal Guide	**Head of the Department**
Dr. KAVITHA SOPPARI	Dr. KAVITHA SOPPARI
Head of the Department	Head of the Department
Dept. of CSE (AI & ML)	Dept. of CSE (AI & ML)

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We would like to express our gratitude to all the people behind the screen who have helped us transform an idea into a real time application.

We would like to express our heart-felt gratitude to my parents without whom. We would not have been privileged to achieve and fulfill our dreams.

A special thanks to our Secretary, **Prof. Y. V. GOPALA KRISHNA MURTHY,** for having founded such an esteemed institution. We are also grateful to our beloved principal, **Dr. B. L. RAJU** for permitting us to carry out this project.

We profoundly thank **Dr. KAVITHA SOPPARI**, Head of the Department of CSE (Artificial Intelligence & Machine Learning), who has been an excellent guide and a great source of inspiration forourwork.

We extremely thank **Mr. Shashank Tiwari** Assistant Professor, Project coordinator, whohelped us in all the way in fulfilling of all aspects in completion of our Mini Project.

We are very thankful to our internal guide **Dr. KAVITHA SOPPARI,** who has been an excellent and given continuous support for the Completion of our project work.

The satisfaction and euphoria that accompany the successful completion of the task would be great, but incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success. In this context, we would like to thank all the other staff members, both teaching and non-teaching, who have extended their timely help and eased our task.

Pamujula Lakshmi Priya	**20AG1A6646**
M.Pooja	**21AG5A6606**
G.Abhishek	**20AG1A6616**

DECLARATION

This is to certify that the work reported in the present project titled **"AN ENHANCED OPTIMIZATION METHOD FOR PREDICTING ENGLISH PHRASES DURING SENTENCE FORMATION USING DEEP LEARNING"** is a record work done by us in the Department of CSE (Artificial Intelligence & Machine Learning), ACE Engineering College.

No part of the thesis is copied from books/journals/internet and whenever the portion is taken,the same has been duly referred in the text; the reported are based on the project work done entirelyby us not copied from any other source.

Pamujula Lakshmi Priya	**20AG1A6646**
M.Pooja	**21AG5A6606**
G.Abhishek	**20AG1A6616**

ABSTRACT

A ENHANCED OPTIMIZATION METHOD FOR PREDICTING ENGLISH PHRASES DURING SENTENCE FORMATION USING DEEP LEARNING represents a pivotal exploration into the task of predicting phrases within sentence contexts. Leveraging the capabilities of deep learning techniques, our study aims to enhance the precision of next-word prediction, a critical aspect in natural language processing applications such as text generation, machine translation, and dialogue systems. In pursuit of this objective, we delve into an investigation of various deep learning architectures, including recurrent neural networks (RNNs), long short-term memory networks (LSTMs), and transformer models. This exploration seeks to unravel the potential of these architectures in capturing intricate linguistic patterns and contextual dependencies. Additionally, we scrutinize preprocessing methods like tokenization and embeddings to augment the models' comprehension of input text. The outcomes of our experiments underscore the remarkable strides made by deep learning, showcasing its effectiveness in achieving enhanced optimization for predicting English phrases during sentence formation.

INDEX

Table of Contents

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

In the realm of natural language processing (NLP), predicting English phrases within sentence formation is a pivotal challenge with transformative potential for various applications. This project aims to teach computers to understand and use English sentences effectively, akin to training them to communicate in English proficiently.

Our approach begins with data gathering and preparation. Utilizing the Sherlock Holmes stories novel as a diverse dataset with 12,169 lines, we lay the foundation for advanced deep learning techniques. These techniques are employed to optimize methods for predicting English phrases during sentence formation. This sophisticated approach sets the stage for enhancing the model's language proficiency.

The project's wide-ranging applications span from making chatbots more conversational to improving translation tools. Leveraging the rich vocabulary and sentence structures present in the Sherlock Holmes stories novel, our model holds promise for diverse real-world language processing tasks.

Measuring our computer's performance is paramount. Defining metrics and evaluation criteria ensures a thorough assessment of how well the model can generate coherent and contextually appropriate English sentences. This step is essential for gauging the practical utility of the developed techniques.

We'll explain how to practically implement and utilize the trained model in various situations. Bridging the gap between research and practical application is crucial for maximizing the impact of our advancements in NLP on users' experiences.

Beyond immediate applications, this research represents a stride toward more human-like language understanding and generation. By enhancing the optimization method for predicting English phrases during sentence formation through deep learning, we aim to reshape the way machines interact with and contribute to human language.

1.1 EXISTING SYSTEMS:

N-gram Model:

The N-gram model is a simple and widely used statistical language model for predicting the next word in a sequence of words. It's based on the assumption that the probability of a word depends only on the previous N-1 words (hence the name N-gram). The model estimates these probabilities from a given training corpus. Here's a basic explanation of the N-gram model and an algorithm to illustrate its operation:

N-gram Model Algorithm:

1. Tokenization:

Break the input text into tokens (words or subword units).

2. Building N-grams:

Create N-grams from the tokenized text. An N-gram is a contiguous sequence of N items (usually words). For example, in a bigram (2-gram) model, the text "I love natural language processing" would generate the following bigrams: ["I love", "love natural", "natural language", "language processing"].

3. Counting Occurrences:

Count the occurrences of each N-gram in the training corpus. For a bigram model, you count how many times each pair of consecutive words appears in the training data.

4. Calculating Probabilities:

Calculate the probability of the next word given the preceding N-1 words using the formula:

[P(w_n | w_{n-1}, w_{n-2}, ..., w_1) = \frac{Count(w_{n-1}, w_{n-2}, ..., w_1, w_n)}{Count(w_{n-1}, w_{n-2}, ..., w_1)} \]

Where:

- \(w_n\) is the next word.
- \(w_{n-1}, w_{n-2}, ..., w_1\) are the preceding N-1 words.
- \(Count(\cdot)\) denotes the number of occurrences.

5. Smoothing (Optional):

To handle unseen N-grams or prevent zero probabilities, smoothing techniques such as add-one smoothing (Laplace smoothing) can be applied.

6. Prediction:

Given a sequence of words \(w_{n-1}, w_{n-2}, ..., w_1\), predict the next word \(w_n\) by selecting the word with the highest probability according to the model.

Draw Back of N- Gram Model:

Limited contextual understanding, as these models consider only a fixed number of preceding words. They struggle with capturing long-range dependencies and understanding context beyond the local n-gram.

Hidden Markov Models (HMM):

Hidden Markov Models (HMMs) are probabilistic models that assume the existence of underlying hidden states and observable outputs. In the context of next word prediction, we can represent a sequence of words as the observable output and the hidden states as the unobservable factors influencing the generation of those words. Here's an explanation of how a Hidden Markov Model works for next word prediction along with an algorithm:

Hidden Markov Model Algorithm for Next Word Prediction:

1.Define States:

Each hidden state represents a certain linguistic or contextual aspect. For next word prediction, states could be related to the grammatical structure, sentiment, or any other relevant linguistic feature.

2. Initialize Transition Probabilities:

Define the probabilities of transitioning from one hidden state to another. This is typically represented by a transition matrix \(A\), where \(A_{ij}\) is the probability of transitioning from state \(i\) to state \(j\).

3. Initialize Emission Probabilities:

Define the probabilities of emitting an observable output (word) given a hidden state. This is represented by an emission matrix \(B\), where \(B_{ij}\) is the probability of emitting word \(j\) when in hidden state \(i\).

4. Initialize Initial State Probabilities:

Define the probabilities of starting in each hidden state. This is represented by an initial state probability vector \(π\), where \(π_i\) is the probability of starting in hidden state \(i\).

5. Observation Sequence:

Given an observation sequence (a sequence of words), represent it as a sequence of observable outputs.

6. Forward Algorithm (Forward Procedure):

Calculate the forward probabilities for each hidden state at each time step, considering the observed sequence up to that point. This is done using the forward algorithm, which involves recursive calculations based on transition and emission probabilities.

7. Backward Algorithm (Backward Procedure):

Calculate the backward probabilities for each hidden state at each time step, considering the observed sequence from that point onward. This is done using the backward algorithm.

8. Compute State Probabilities:

Use the forward and backward probabilities to calculate the probability of being in each hidden state at each time step, given the observed sequence. This is done using the forward-backward algorithm.

9. Prediction:

Given the probabilities of being in each hidden state at the last time step, predict the next word by selecting the word with the highest probability in the corresponding emission distribution.

10. Update Model Parameters (Optional):

Optionally, update the model parameters (transition and emission probabilities) based on the observed sequence using techniques like the Baum-Welch algorithm (Expectation-Maximization).

Draw Back of Hidden Markov Model

Similar to N-gram models, HMMs have limited context awareness and may not capture the semantic nuances in language well. They also assume independence between hidden states, which is a simplification.

1.2 Proposed Model

In the Model we want to work with we would like to use Deep Learning Techniques a Long Short-Term Memory (LSTM) network is employed due to its ability to capture intricate sequence patterns. By utilizing Tensor Flow's high-level APIs, the model's implementation becomes more streamlined. We also explore techniques like hyper parameter optimization and teacher forcing for refinement

LSTM networks were designed specifically to overcome the long-term dependency problem faced by recurrent neural networks RNNs (due to the vanishing gradient problem). LSTMs have feed*back* connections which make them different to more traditional feed*forward* neural networks. This property enables LSTMs to process entire sequences of data (e.g. time series) without treating each point in the sequence independently, but rather, retaining useful information about previous data in the sequence to help with the processing of new data points. As a result, LSTMs are particularly good at processing sequences of data such as text, speech and general time-series.

An example — Consider we are trying to predict monthly ice cream sales. As one might expect, these vary highly depending on the month of the year, being lowest in December and highest in June.

An LSTM network can learn this pattern that exists every 12 periods in time. It doesn't just use the previous prediction but rather retains a longer-term context which helps it overcome the long-term dependency problem faced by other models. It is worth noting that this is a very simplistic example, but when the pattern is separated by much longer periods of time (in long passages of text, for example), LSTMs become increasingly useful.

How do LSTM Networks Work?

Firstly, at a basic level, the output of an LSTM at a particular point in time is dependant on three things:

- ‣ The current long-term memory of the network — known as the *cell state*
- ‣ The output at the previous point in time — known as the previous *hidden state*
- ‣ The input data at the current time step

LSTMs use a series of 'gates' which control how the information in a sequence of data comes into, is stored in and leaves the network. There are three gates in a typical LSTM; forget gate, input gate and output gate. These gates can be thought of as filters and are each their own neural network. We will explore them all in detail during the course of this article.

In the following explanation, we consider an LSTM cell as visualised in the following diagram. When looking at the diagrams in this article, imagine moving from left to right.

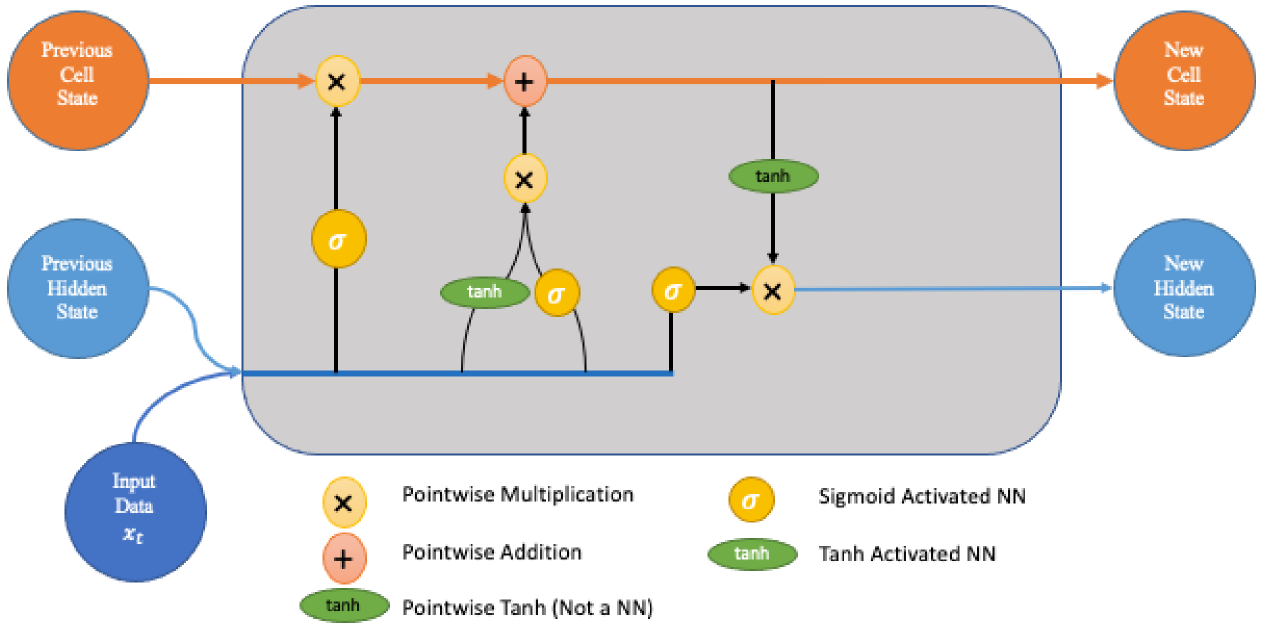


Fig 1.1 working of LSTM

LSTM Diagram

Step 1

The first step in the process is the **forget gate**. Here we will decide which bits of the cell state (long term memory of the network) are useful given both the previous hidden state and new input data.

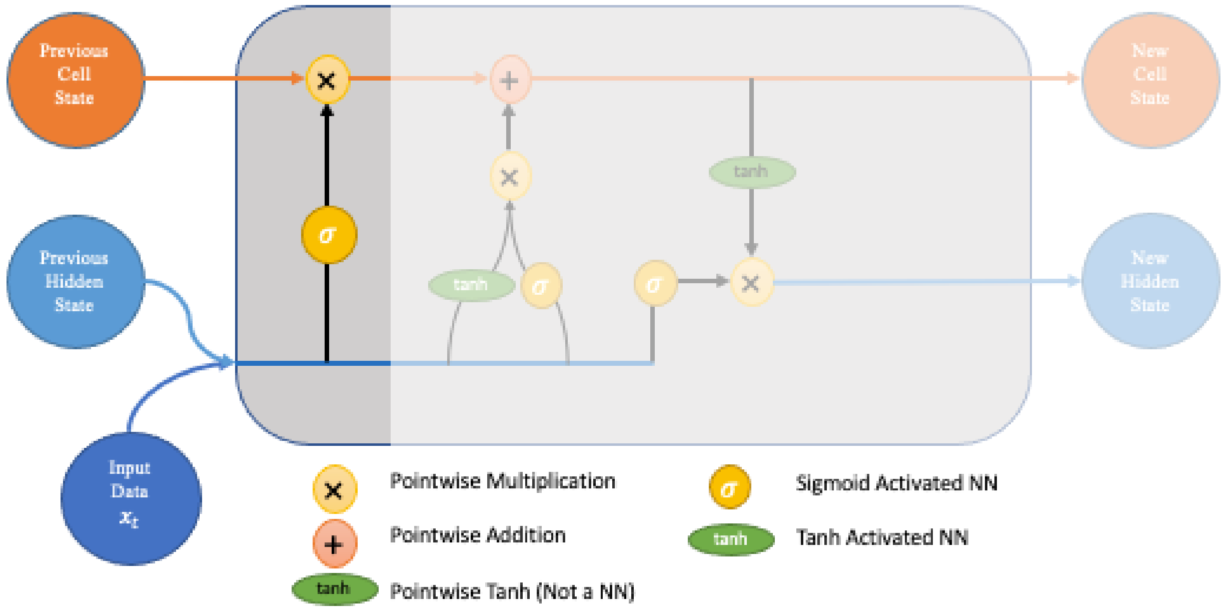


Fig 2 Forget Gate

Forget Gate

To do this, the previous hidden state and the new input data are fed into a neural network. This network generates a vector where each element is in the interval [0,1] (ensured by using the sigmoid activation). This network (within the forget gate) is trained so that it outputs close to 0 when a component of the input is deemed irrelevant and closer to 1 when relevant. It is useful to think of each element of this vector as a sort of filter/sieve which allows more information through as the value gets closer to 1.

These outputted values are then sent up and pointwise multiplied with the previous cell state. This pointwise multiplication means that components of the cell state which have been deemed irrelevant by the forget gate network will be multiplied by a number close to 0 and thus will have less influence on the following steps.

In summary, the forget gate decides which pieces of the long-term memory should now be forgotten (have less weight) given the previous hidden state and the new data point in the sequence.

Step 2

The next step involves the **new memory network** and the **input gate**. The goal of this step is to determine what new information should be added to the networks long-term memory (cell state), given the previous hidden state and new input data.

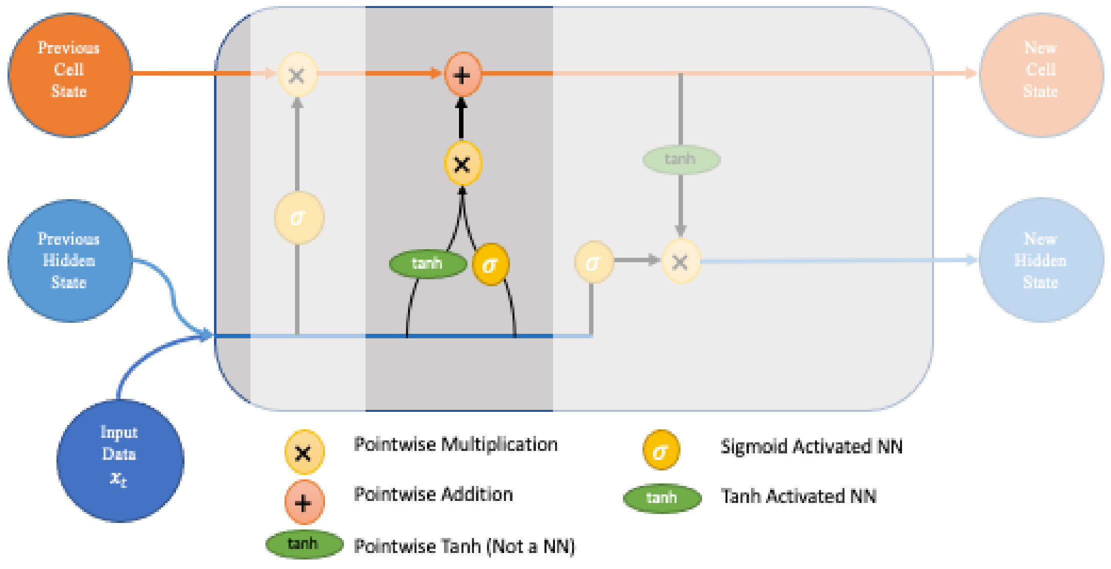


Fig 3 Input Gate

Input Gate

Both the new memory network and the input gate are neural networks in themselves, and both take the same inputs, the previous hidden state and the new input data. It is worth noting that *the inputs here are actually the same as the inputs to the forget gate*!

1.The **new memory network** is a tanh activated neural network which has learned how to combine the previous hidden state and new input data to generate a 'new memory update vector'. This vector essentially contains information from the new input data given the context from the previous hidden state. This vector tells us how much to update each component of the long-term memory (cell state) of the network given the new data.

Note that we use a tanh here because its values lie in [-1,1] and so can be negative. The possibility of negative values here is necessary if we wish to reduce the impact of a component in the cell state.

1.However, in part 1 above, where we generate the new memory vector, there is a big problem, it doesn't actually check if the new input data is even worth remembering. This is where the **input gate** comes in. The input gate is a sigmoid activated network which acts as a filter, identifying which components of the 'new memory vector' are worth retaining. This network will output a vector of values in [0,1] (due to the sigmoid activation), allowing it to act as a filter through pointwise multiplication. Similar to what we saw in the forget gate, an output near zero is telling us we don't want to update that element of the cell state.

2.The output of parts 1 and 2 are pointwise multiplied. This causes the magnitude of new information we decided on in part 2 to be regulated and set to 0 if need be. The resulting combined vector is then *added* to the cell state, resulting in the long-term memory of the network being updated.

Step 3

Now that our updates to the long-term memory of the network are complete, we can move to the final step, **the output gate**, deciding the new hidden state. To decide this, we will use three things; the newly updated cell state, the previous hidden state and the new input data.

One might think that we could just output the updated cell state; however, this would be comparable to someone unloading everything they had ever learned about the stock market when only asked if they think it will go up or down tomorrow.

To prevent this from happening we create a filter, the output gate, exactly as we did in the forget gate network. The inputs are the same (previous hidden state and new data), and the activation is also sigmoid (since we want the filter property gained from outputs in [0,1]).

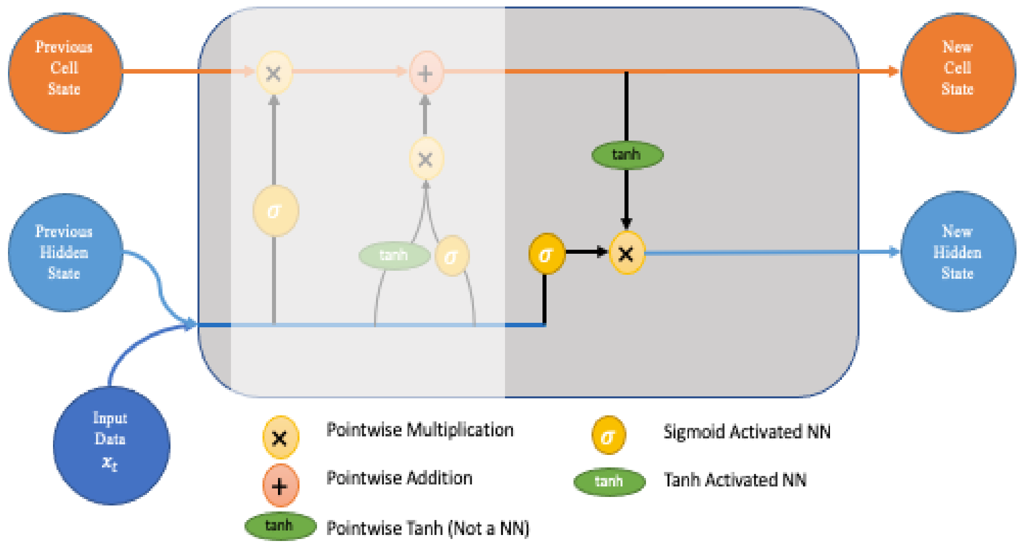


Fig 4 Output Gate

Output Gate

As mentioned, we want to apply this filter to the newly updated cell state. This ensures that only necessary information is output (saved to the new hidden state). However, before applying the filter, we pass the cell state through a tanh to force the values into the interval [-1,1].

The step-by-step process for this final step is as follows:

▹ Apply the tanh function to the current cell state pointwise to obtain the squished cell state, which now lies in [-1,1].

- Pass the previous hidden state and current input data through the sigmoid activated neural network to obtain the filter vector.
- Apply this filter vector to the squished cell state by pointwise multiplication.
- Output the new hidden state.

CHAPTER 2

LITERATURE SURVEY

The pursuit of predicting English phrases during sentence formation using deep learning has been enriched by the pioneering work of several researchers. The following review provides insights into key studies in this domain:

II. LITERATURE REVIEW

Mikolov et al. [1] - Word Embeddings with Word2Vec:
Mikolov's Word2Vec introduced distributed representations of words, showcasing the power of continuous vectors in capturing semantic relationships. This work laid the foundation for advancements in natural language processing (NLP).

The Word2Vec software, developed by Tomas Mikolov and team, is widely recognized for creating word embeddings, which are representations of words that capture their meanings. However, understanding the technical details of the underlying models, as outlined in Mikolov et al.'s paper, can be challenging. The note focuses on a specific aspect of the Word2Vec model, the skip-gram model, which aims to learn how words are related by analyzing their contextual usage. The paper introduces a complex equation involving negative sampling, a technique that enhances the efficiency of the learning process. Negative sampling helps the model distinguish between actual word pairs observed in the data and randomly generated incorrect pairs, making the training process computationally more feasible. In simpler terms, it's like teaching the software to recognize meaningful word associations more quickly and effectively.

The transition from a complex softmax function to the simpler concept of logistic regression is crucial in this explanation. This shift involves setting up a game for the model, where it learns to differentiate between real and fake word pairs. By doing so, the Word2Vec software becomes more proficient in understanding how words relate to each other, improving its ability to create meaningful word embeddings efficiently.

Chung et al. [2] - Long Short-Term Memory (LSTM) Networks:
Chung addressed the vanishing gradient problem with LSTMs, demonstrating their efficacy in capturing long-range dependencies in sequential data. LSTMs are pivotal in tasks involving English phrase prediction during sentence formation.

Chung et al. addressed the vanishing gradient problem in traditional recurrent neural networks (RNNs) by introducing Long Short-Term Memory (LSTM). LSTMs maintain memory over long sequences, making them adept at capturing dependencies in sequential data. The paper delves into the LSTM architecture, emphasizing its ability to model temporal relationships effectively. LSTMs became a foundational component in various applications, particularly in tasks involving English phrase prediction during sentence formation.

Bahdanau et al. [3] - Neural Machine Translation by Jointly Learning to Align and Translate:
Bahdanau's work on attention mechanisms allowed models to focus on specific input sequence parts during output generation. This mechanism, initially applied in machine translation, has potential applications in predicting English phrases during sentence formation.

Bahdanau et al. proposed a neural machine translation model that jointly learns to align and translate. This attention-based approach allows the model to focus on specific parts of the input sequence during translation. The model's ability to align and translate simultaneously significantly improved translation quality compared to previous methods. The attention mechanism introduced in this paper became a key component in subsequent advancements in neural machine translation.

Vaswani et al. [4] - "Attention is All You Need":
The Transformer model introduced by Vaswani revolutionized NLP with its self-attention mechanism, offering parallelization and effective contextual information capture. It stands as a promising framework for optimizing English phrase prediction during sentence formation.

Vaswani et al. introduced the Transformer architecture, a novel sequence-to-sequence model relying solely on self-attention mechanisms. Departing from traditional RNNs and LSTMs, Transformers enable parallelization during training, improving computational efficiency. The paper showcases the effectiveness of self-attention in capturing contextual information, leading to superior performance in machine translation tasks. Transformers became a cornerstone in NLP and influenced subsequent models like BERT and GPT.

Howard and Ruder [5] - "Universal Language Model Fine-tuning for Text Classification":
Howard and Ruder's ULMFiT demonstrated the efficacy of transfer learning by pre-training language models on a large corpus and fine-tuning for specific tasks. Transfer learning approaches like ULMFiT could play a crucial role in optimizing English phrase prediction.

Howard and Ruder proposed Universal Language Model Fine-tuning (ULMFiT), a transfer learning approach for text classification. Pre-training a language model on a large corpus and fine-tuning it for specific tasks with limited labeled data, ULMFiT showcased remarkable performance across diverse text classification tasks. The paper emphasizes the versatility of transfer learning in NLP, enabling effective model adaptation for downstream applications.

Zhang et al. [6] - Hybrid Models and Ensemble Approaches:
Zhang's exploration of hybrid models, combining CNNs with LSTMs, showcased improvements in sentiment analysis. The study of hybrid models and ensemble approaches provides valuable insights for optimizing English phrase prediction during sentence formation.

Zhang et al. presented a hybrid approach for sentiment analysis, combining Convolutional Neural Networks (CNNs) with other techniques. The model demonstrated improved performance in sentiment classification tasks, leveraging the hierarchical structure captured by CNNs.

Technique/Methodology	Pros	Cons
Mikolov et al. [1] - Word2Vec (2013):	Efficiently captures semantic relationships through distributed word representations. Provides a foundation for subsequent advancements in natural language processing (NLP).	Limited to word-level embeddings, potentially missing higher-level contextual information.
Chung et al. [2] - LSTM Networks (2014):	Effectively captures long-range dependencies in sequential data. Suitable for tasks involving English phrase prediction during sentence formation.	May suffer from the vanishing gradient problem, affecting training stability.
Bahdanau et al. [3]- Neural Machine Translation by Jointly Learning to Align and Translate(2015):	Enables models to focus on specific parts of the input sequence, improving performance. Particularly useful for sequence-to-sequence tasks like machine translation.	Adds computational complexity, potentially impacting real-time applications.
Vaswani et al. [4] - Attention is All You Need(2017):	Allows for efficient parallelization during training, speeding up the learning process. Effectively captures contextual information through self-attention mechanisms.	Requires large datasets for effective training, potentially limiting applicability in smaller-scale scenarios.
Howard and Ruder [5] - Universal Language Model Fine-tuning for Text Classification (2018):	Demonstrates the effective transfer of language knowledge from pre-training to fine-tuning. Enhances performance on specific NLP tasks with minimal labeled data.	Fine-tuning can be computationally intensive, requiring substantial resources.
Zhang et al. [6] - Hybrid Models and Ensemble Approaches (2018):	Combining CNNs and LSTMs improves sentiment analysis performance.	Increased model complexity may result in longer training times and resource requirements.

2.1 FEASIBILITY ANALYSIS

The growth and recognition of project management have changed significantly over thepast few years, and these changes are expected to continue and expand. And with the rise of
project management comes the need for a feasibility study. , a feasibility analysis is used to determine the viability of an idea, such as ensuring a project is legally and technically feasible as well as economically justifiable. It tells us whether a project is worth the investment—in some cases, a project may not be doable. There can be many reasons for this, including requiringtoo many resources, which not only prevents those resources from performing other tasks but also may cost more than an organization would earn back by taking on a project that isn't profitable.

TYPES OF FEASIBILITY STUDY

A feasibility analysis evaluates the project's potential for success; therefore, perceived objectivity is an essential factor in the credibility of the study for potential investors and lendinginstitutions. There are five types of feasibility studies separate areas that a feasibility study examines, described below.

- Technical Feasibility
- Economic Feasibility
- Legal Feasibility
- Operational Feasibility
- Scheduling Feasibility

Technical Feasibility

This assessment focuses on the technical resources available to the organization. It helpsorganizations determine whether the technical resources meet capacity and whether the technical team is capable of converting the ideas into working systems. Technical feasibility also involves the evaluation of the hardware, software, and other technical requirements of theproposed system.

Economic Feasibility

This assessment typically involves a cost/ benefits analysis of the project, helping organizations determine the viability, cost, and benefits associated with a project before financial resources are allocated. It also serves as an independent project assessment and enhances project credibility—

helping decision-makers determine the positive economic benefits to the organization that the proposed project will provide.

Legal Feasibility

This assessment investigates whether any aspect of the proposed project conflicts with legal requirements like zoning laws, data protection acts or social media laws. Let’s say an organization wants to construct a new office building in a specific location. A feasibility studymight reveal the organization’s ideal location isn’t zoned for that type of business. That organization has just saved considerable time and effort by learning that their project was not feasible right from the beginning.

Operational Feasibility

This assessment involves undertaking a study to analyze and determine whether—and how well—the organization’s needs can be met by completing the project. Operational feasibility studies also examine how a project plan satisfies the requirements identified in the requirements analysis phase of system development.

Scheduling Feasibility

This assessment is the most important for project success; after all, a project will fail ifnot completed on time. In scheduling feasibility, an organization estimates how much time theproject will take to complete.

CHAPTER 3
SYSTEM REQUIREMENTS

3.1 HARDWARE REQUIREMENTS

Training Mode & Testing Mode

CPU: Intel Core i5 or equivalent

OPERATING SYSTEM: windows

PROCESSOR: 1.7GHZ

RAM: 4GB minimum

HARD DISK: 200GB

3.2 SOFTWARE REQUIREMENTS

It is the starting point of the software development activity, as the system grows morecomplex it becomes evident that, the goal of the phase is more. The software project is initiated by the client needs.

TOOLS: Spyder and Google Colab or Jupyter

OPERATING SYSTEMS: Windows

LANGUAGE: Python

IDE: Anaconda

PACKAGE: numpy, Tensorflow, Keras, StreamLit.

CHAPTER 4

SYSTEM ARCHITECTURE

The System architecture is a critical component in building an effective next-word prediction system using deep learning in NLP. One common approach is to utilize recurrent neural networks (RNNs) or their variants, such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU). These architectures are specifically designed to capture sequential dependencies in the input text, enabling accurate predictions of the next word.

System Architecture for Next Word Prediction

The system architecture for next-word prediction typically consists of the following components:

1. **Embedding Layer**: The word embeddings, often referred to as distributed representations of words, are learned by the embedding layer. It captures semantic and contextual information by mapping every word in the lexicon to a dense vector representation. The model can be trained to change these embeddings as trainable parameters.

2. **Recurrent Layers**: The recurrent layers, like LSTM or GRU, process the input word embeddings in sequential order and keep hidden states that record the sequential data. The model can learn the contextual connections between words and their placements in the sequence thanks these layers.

3. **Dense Layers**: One or more dense layers are then added after the recurrent layers to convert the learned features into the appropriate output format. When predicting the next word, the dense layers translate the hidden representations to a probability distribution across the vocabulary, indicating the likelihood that each word will be the following word.

Training and Optimization

The model is trained using a large corpus of text data, where the input sequences are paired with their corresponding target word. The training process involves optimizing the model's parameters by minimizing a suitable loss function, such as categorical cross-entropy. The optimization is typically performed using an optimization algorithm like Adam or Stochastic Gradient Descent

(SGD).

Inference and Prediction

The model can be used to predict the next word once it has been trained. The trained model receives an input of a list of words, processes it through the learned architecture, and outputs a probability distribution across the vocabulary. The anticipated next word is then chosen as the one with the highest likelihood.

We can create reliable and accurate next-word prediction models for NLP by combining the strength of recurrent neural networks, such as LSTM or GRU, with the right training and optimization methods. The model design successfully captures the text’s sequential dependencies, enabling the system to produce fluent and contextually relevant predictions.

CHAPTER 5
SYSTEM DESIGN

The system design outlines the architecture, components, and workflow of the project, focusing on predicting English phrases during sentence formation using deep learning. The design encompasses various stages, from data preprocessing to model training and prediction. The system design incorporates data preprocessing, model definition, training, and prediction processes, providing a comprehensive overview of the project's architecture and workflow for predicting English phrasesduring sentence formation using deep learning.

1)Importing Libraries:

The initial step involves importing necessary libraries for deep learning. TensorFlow is imported as 'tf' to leverage its functionalities, and the Sequential model from Keras is brought in for building a sequential neural network. Specific layers like Embedding, LSTM, and Dense are also imported. Additionally, Numpy is imported as 'np' for array generation, and the 're' module is imported for data processing and pattern recognition.

2)Understanding and Preprocessing the Dataset:

This stage focuses on preparing text data for training a language model. The process begins by reading a file and breaking its text into separate sentences. The Keras Tokenizer class is employed to learn vocabulary from input sentences and tokenize the text data. Subsequently, n-gram sequences are generated from the tokenized data, with each sequence containing a range of tokens. The sequences are padded with zeros to ensure uniform length. These sequences are then divided into predictors (X) and labels (Y), where X contains all elements except the last token, and Y contains only the final token. Finally, the target data Y is converted to one-hot encoding, making it ready for training.

Input Data:

Raw text data is obtained from a source, such as a file containing English sentences.

Text Tokenization:

The Keras Tokenizer class is employed to tokenize the input text, breaking it into individual word.

N-gram Sequence Generation:

N-gram sequences are created from the tokenized data, where each sequence represents a range of tokens from the start to the current index.

Padding:

Zero-padding is applied to the sequences to ensure uniform length, facilitating consistent input to the deep learning model.

Label Generation:

The sequences are divided into predictors (X) and labels (Y). X contains all elements except the last token of each sequence, and Y contains only the final token.

1)Model Definition:

The model is defined in this step. The input sequences are mapped to dense vectors in the first layer, an embedding layer, which requires parameters such as input length, dimensionality of the embedding space, and the total number of unique words in the vocabulary. The second layer is an LSTM (Long Short-Term Memory) layer with 128 units, capable of recognizing long-term dependencies in sequential input. The third layer is a Dense layer with units equal to the total number of words in the vocabulary and softmax activation, generating output probabilities for each word. The categorical cross-entropy loss function is used for multi-class classification, and Adam is chosen as the optimizer, with accuracy as the evaluation metric.

Embedding Layer

The input sequences are mapped to dense vectors in the embedding layer. Parameters include input length (length of input sequences minus one), dimensionality of the embedding space, and the total number of unique words in the vocabulary.

LSTM Layer:

An LSTM layer with 128 units follows the embedding layer. LSTM is chosen for its ability to recognize long-term dependencies in sequential input.

Dense Layer:

A Dense layer with units equal to the total number of words in the vocabulary and softmax activation generates output probabilities for each word.

Loss Function:

Categorical cross-entropy loss function is utilized, suitable for multi-class classification applications.

Optimizer and Metric:

Adam is selected as the optimizer, and accuracy is chosen as the evaluation metric.

1) Model Training:

The model is trained using the fit method, with input data (X) and target data (Y) provided as parameters. The epochs parameter is set to 500, indicating the number of iterations the entire dataset will undergo during training. The model learns to predict the next word in a sequence based on the input data. Training progress and data for each epoch are displayed when the verbose parameter is set.The model's weights are adjusted iteratively to minimize the defined loss function and improve precision in predicting the next word.

Training Process:

The model is trained using the fit method. Input data (X) and target data (Y) are provided as parameters.

Epochs:

Training occurs over 100 epochs, indicating the number of iterations the entire dataset undergoes during training.

Learning:

The model iteratively adjusts its weights to minimize the defined loss function, enhancing precision in predicting the next word in input sequences.

2) Prediction Process:

To generate predictions for subsequent words, an initial input text and the number of words to be predicted are specified. A loop is executed for the defined number of words. Within each iteration, the input text is tokenized, and the token list is padded to align with the model's expected input length. The model's prediction method is employed to obtain probabilities for each word in the vocabulary. The argmax function identifies the index of the word with the highest probability, which is then converted to the corresponding word using the tokenizer's index_word dictionary. The predicted word is added to the input text, and this process repeats for the specified number of predictions. Finally, the generated sequence of words is printed as "Next predicted words: [generated_sequence]".

Seed Text and Next Words:

An initial input text (seed_text) and the number of words to be predicted (next_words) are specified.

Prediction Loop:

A loop is executed 'next_words' times, where the seed_text is tokenized and padded. The model predicts probabilities for each word, and the argmax function determines the index of the word with the highest probability.

Word Generation:

The predicted index is converted to the corresponding word using the tokenizer's index_word dictionary. The predicted word is appended to the seed_text.

Final Output:

The generated sequence of words is printed as "Next predicted words: [generated_sequence]."

5.6 ARCHITECTURE DIAGRAM

Architectural model (in software) is a rich and rigorous diagram, created using available standards, in which the primary concern is to illustrate a specific set of tradeoffs inherent in the structure and design of a system or ecosystem. Software architects use architectural models to communicate with others and seek peer feedback. An architectural model is an expression of a viewpoint in software architecture.

Illustrate: the idea behind creating a model is to communicate and seek valuable feedback. The goal of the diagram should be to answer a specific question and to share that answer with others to (a) see if they agree, and (b) guide their work. Rule of thumb:

know what it is you want to say, and whose work you intend to influence with it.

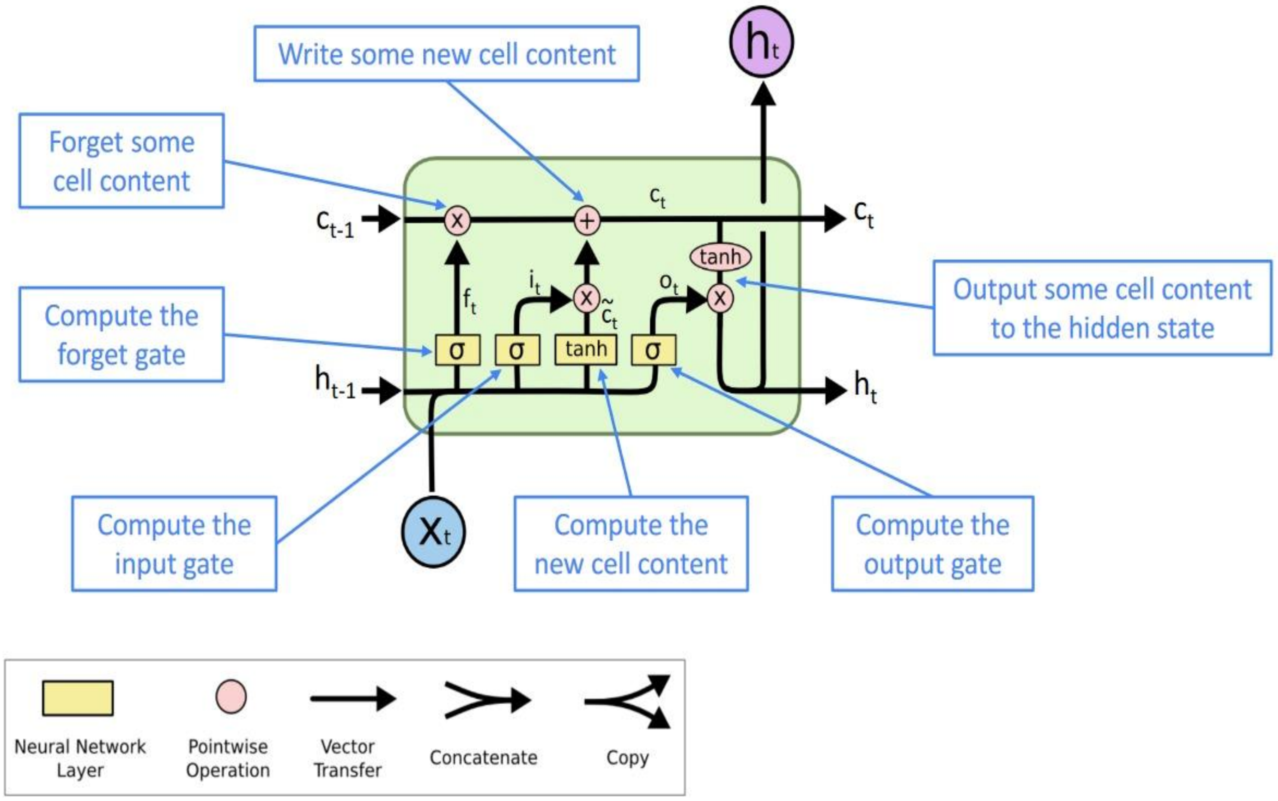


Fig 10 Architecture Diagram

UML DIAGRAMS

5.1 USE CASE DIAGRAM

To model a system, the most important aspect is to capture the dynamic behaviour. To clarify a bit in details, dynamic behaviour means the behaviour of the system when it is running/operating.

Use case diagrams are used to gather the requirements of a system including internal andexternal influences. These requirements are mostly design requirements. So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified. In brief, the purposes of use case diagrams can be as follows:

a. Used to gather requirements of a system.

b. Used to get an outside view of a system.

c. Identify external and internal factors influencing the system.

USE CASE DIAGRAM

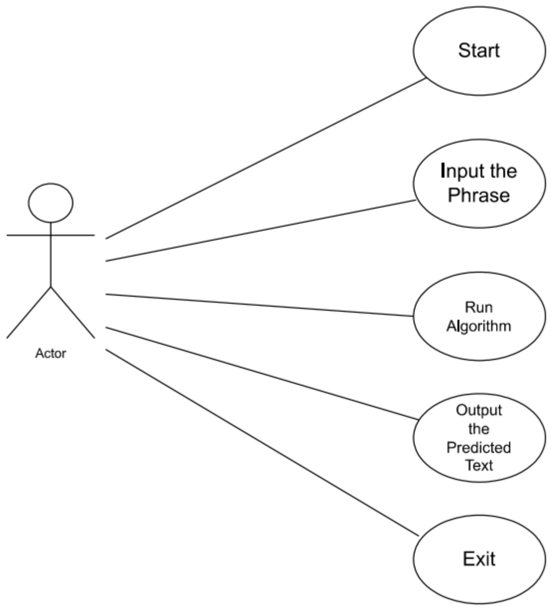


Fig 5 Use Case Diagram

5.2 ACTIVITY DIAGRAM

Activity diagram is another important diagram in UML to describe dynamic aspects ofthe system. Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. So the purposes can be described as:

• Draw the activity flow of a system.

• Describe the sequence from one activity to another.

• Describe the parallel, branched and concurrent flow of the system.

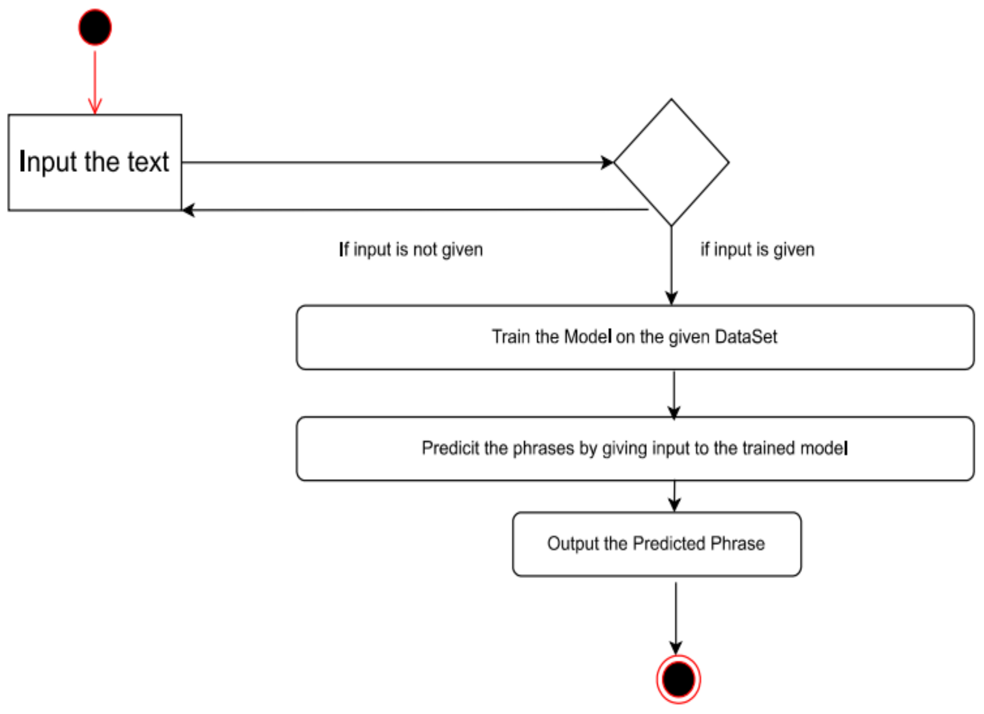

Input the text
If input is not given
if input is given
Train the Model on the given DataSet
Predicit the phrases by giving input to the trained model
Output the Predicted Phrase

Fig 6 Activity Diagram

5.3 SEQUENCE DIAGRAM

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time. They're also called event diagrams. A sequence diagram is a good way to visualize and validate various runtime scenarios. These can help to predict how a system will behave and to discover responsibilities a class may need to have in the process of modelling a new system.

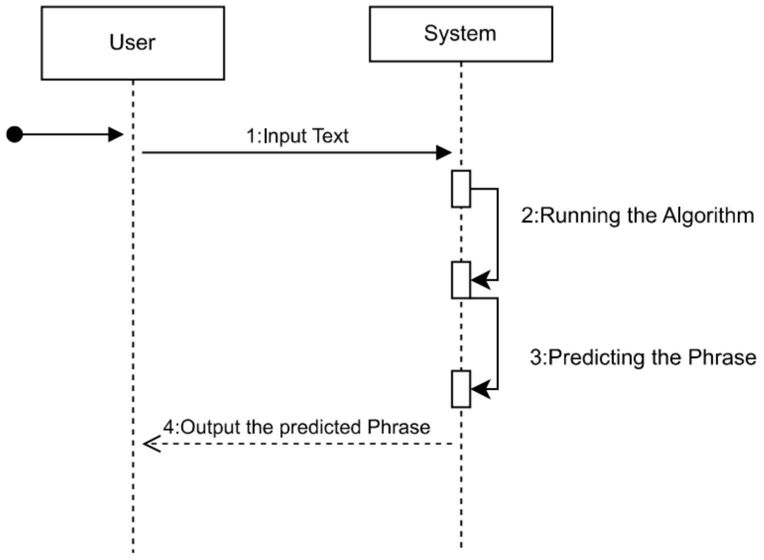


Fig 7 Sequence Diagram

5.4 CLASS DIAGRAM

The class diagram is the main building block of object-oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modelling translating the models into programming code.

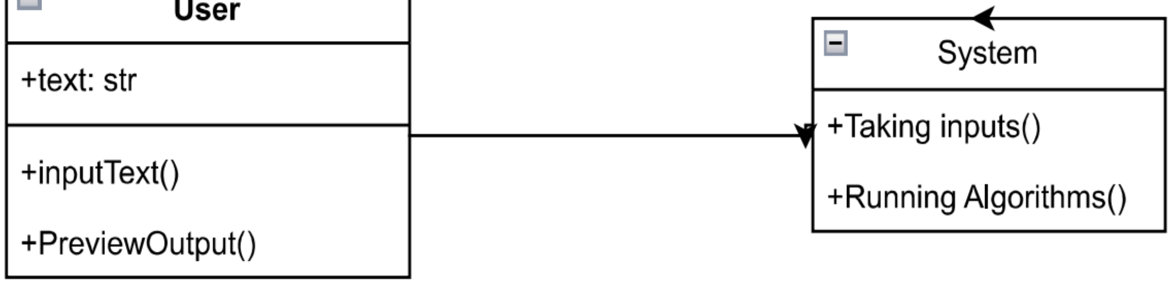


Fig 8 Class Diagram

5.5 COMPONENT DIAGRAM

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems.

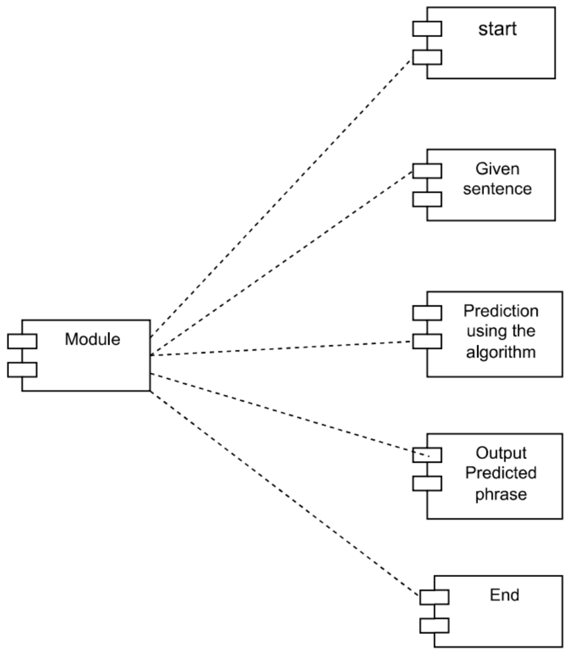


Fig 9 Component Diagram

CHAPTER 6
IMPLEMENTATION

6.1 CODE SAMPLE

CODE TO TRAIN THE MODEL AND SAVE:

```
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

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=100, verbose=1)

seed_text = "i am happy"
next_words =3

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.save('model2.h5')
```

CODE TO EXECUTE THE TRAINED MODEL IN SPYDER:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

with open(r'C:\Users\Laksh\Downloads\book\sherlock-holm.es_stories_plain-text_advs.txt', 'r',
encoding='utf-8') as file:
    text = file.read()

tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])

model2= tf.keras.models.load_model(r'C:\Users\Laksh\Downloads\model2.h5')

seed_text = "hi how are you hope"
next_words =6

for _ in range(next_words):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=17, padding='pre')
    predicted = np.argmax(model2.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)
```

CODE TO CREATE A WEB APPLICATION USING STREAMLIT:

```
import numpy as np
import tensorflow as tf
import streamlit as st
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

with open(r'C:\Users\Laksh\Downloads\book\sherlock-holm.es_stories_plain-text_advs.txt', 'r',
encoding='utf-8') as file:
    text = file.read()

tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])

model2= tf.keras.models.load_model(r'C:\Users\Laksh\Downloads\model2.h5')

#function
def pred(seed_text,next_words):

    for _ in range(int(next_words)):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=17, padding='pre')
        predicted = np.argmax(model2.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
    return seed_text



def main():

    #title for interface
    st.title("Next Word Prediction Web App")

    #getting the input data
    sentence=st.text_input('sentence to complete')
    no_of_words=st.text_input('number_of_words')

    str=""

    if st.button('Generate Words'):
        str=pred(sentence,no_of_words)

    st.success(str)

if __name__=='__main__':
    main()
```

COMMAND TO RUN THE WEB APP:

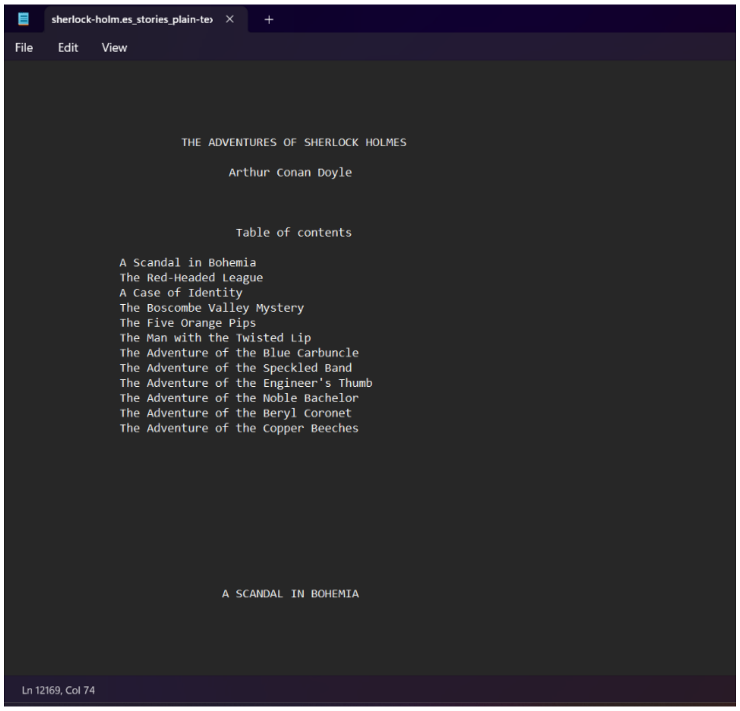
Streamlit run “filename.py”

6.2 DATASET SAMPLE

We have used the dataset from the statso website.

The link is below:

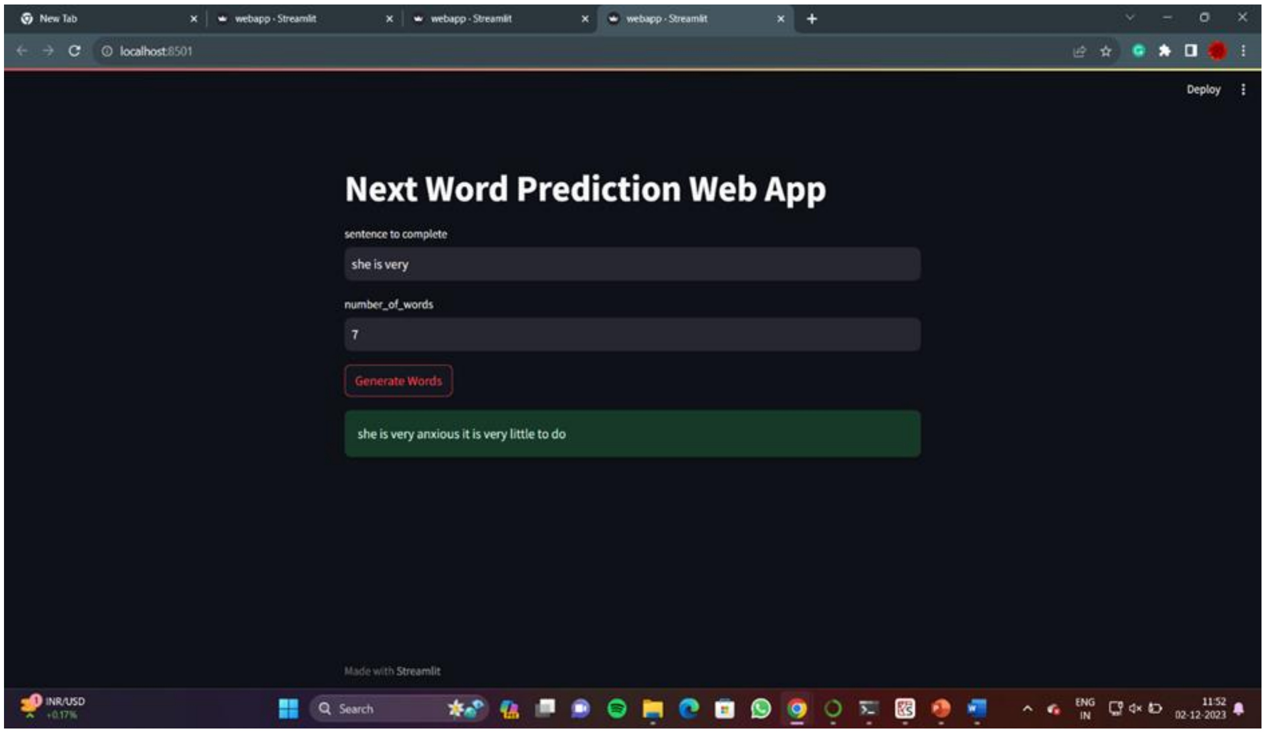
https://statso.io/next-word-prediction-case-study/



```
THE ADVENTURES OF SHERLOCK HOLMES

Arthur Conan Doyle


Table of contents

A Scandal in Bohemia
The Red-Headed League
A Case of Identity
The Boscombe Valley Mystery
The Five Orange Pips
The Man with the Twisted Lip
The Adventure of the Blue Carbuncle
The Adventure of the Speckled Band
The Adventure of the Engineer's Thumb
The Adventure of the Noble Bachelor
The Adventure of the Beryl Coronet
The Adventure of the Copper Beeches




A SCANDAL IN BOHEMIA
```

Fig 11 Sample data

6.3 Final Output:



Next Word Prediction Web App

sentence to complete

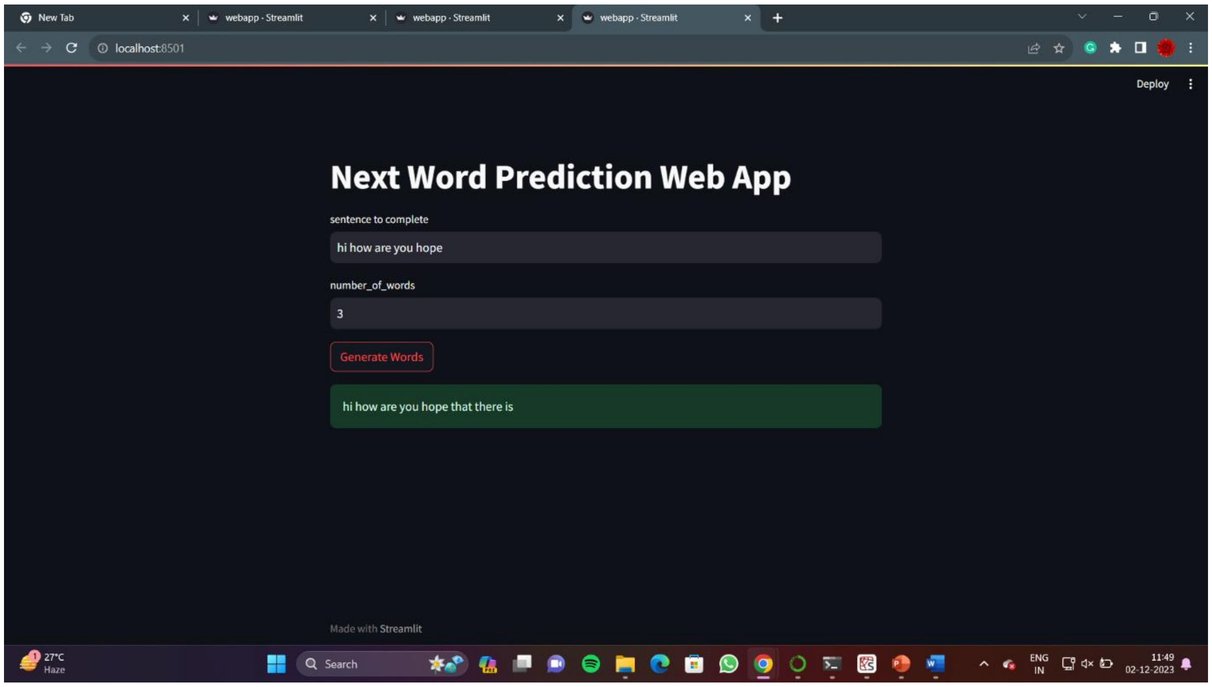
she is very

number_of_words

7

she is very anxious it is very little to do

Fig 11 GUI output


Next Word Prediction Web App

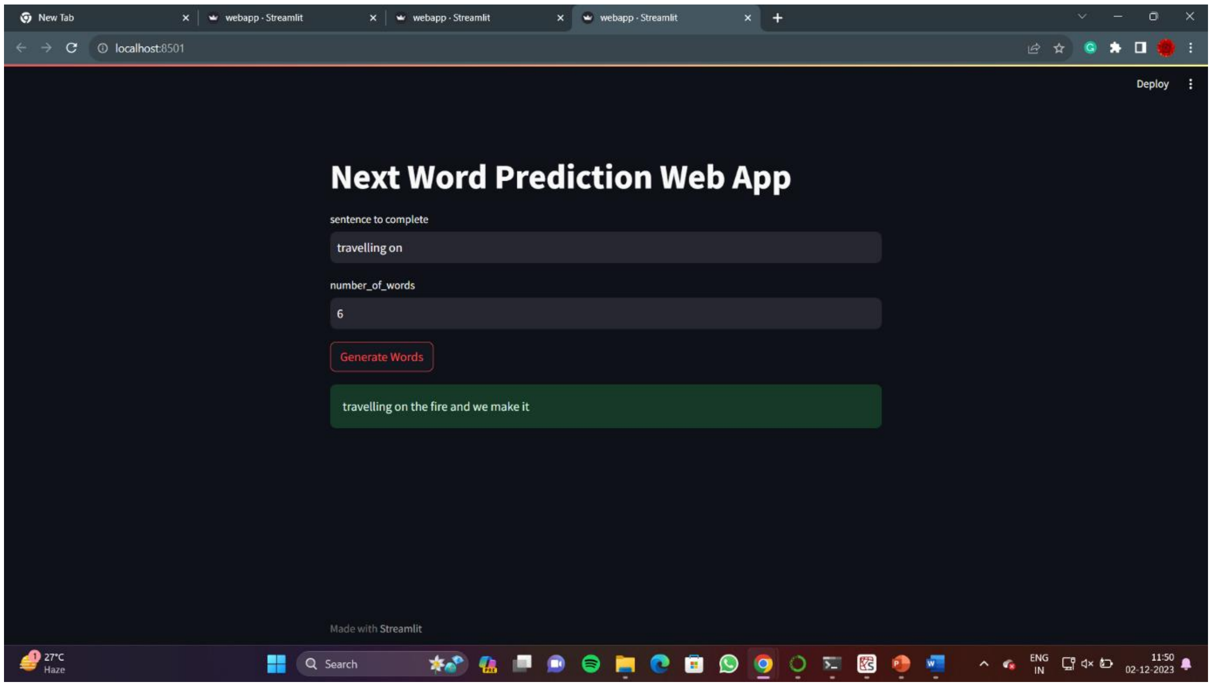
sentence to complete

hi how are you hope

number_of_words

3

hi how are you hope that there is


Next Word Prediction Web App

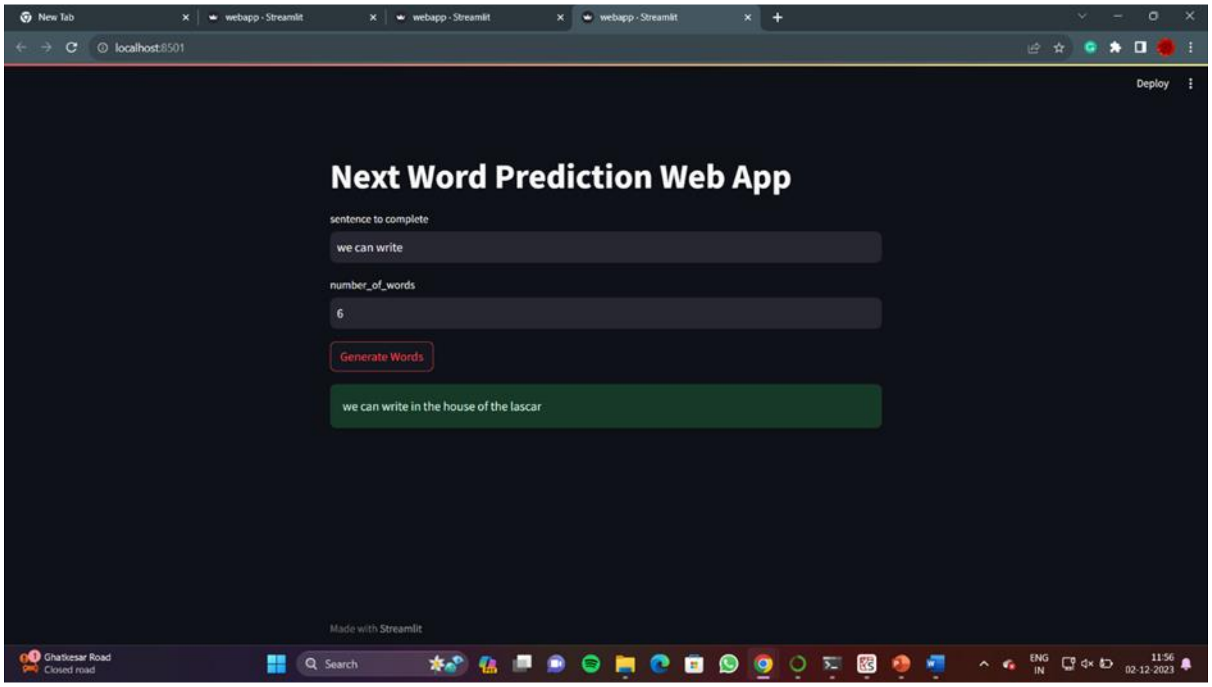
sentence to complete

travelling on

number_of_words

6

travelling on the fire and we make it



Next Word Prediction Web App

sentence to complete

we can write

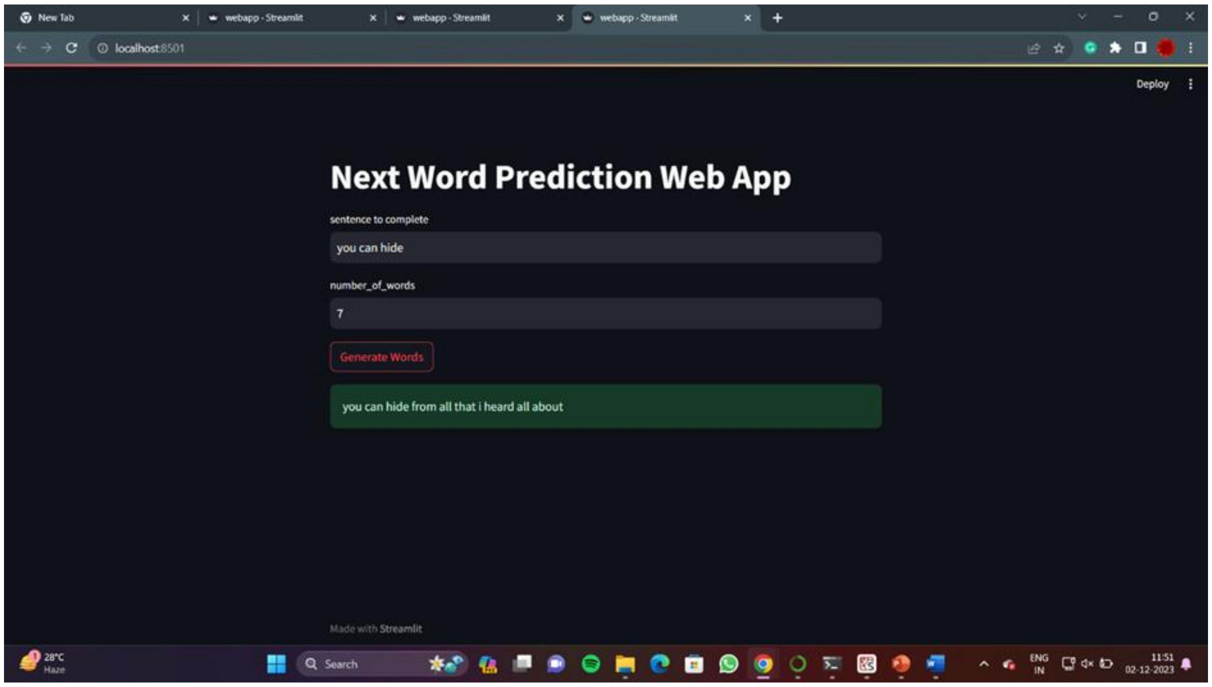
number_of_words

6

Generate Words

we can write in the house of the lascar

Made with Streamlit



Next Word Prediction Web App

sentence to complete

you can hide

number_of_words

7

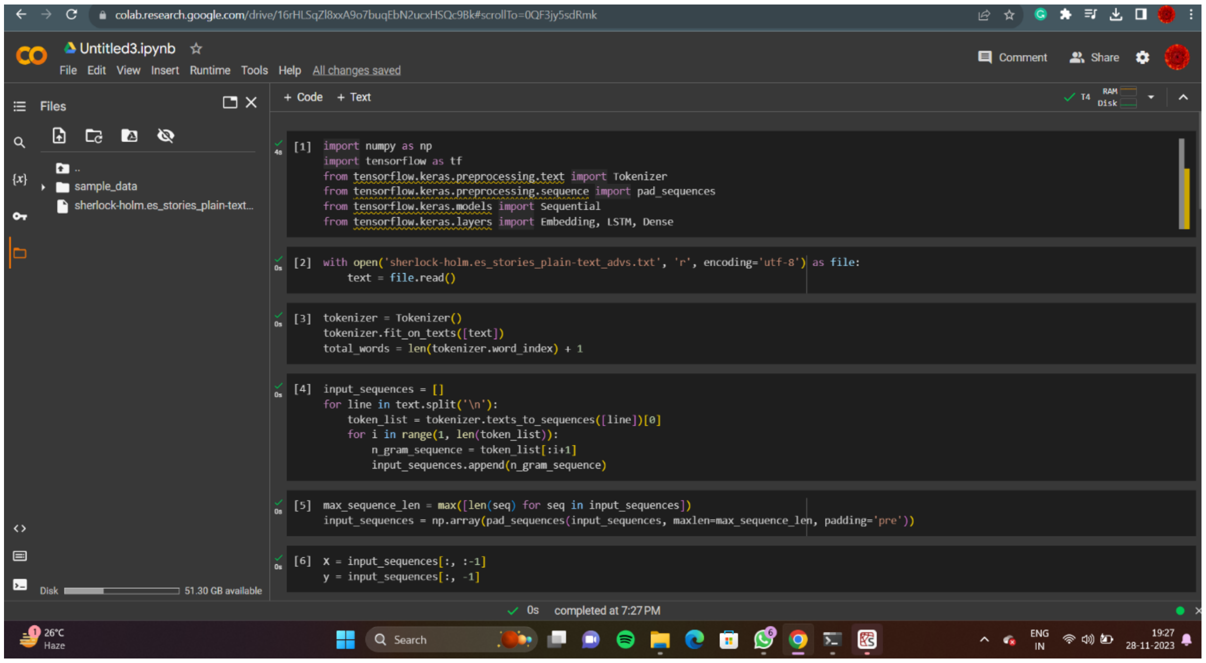
Generate Words

you can hide from all that i heard all about

Made with Streamlit

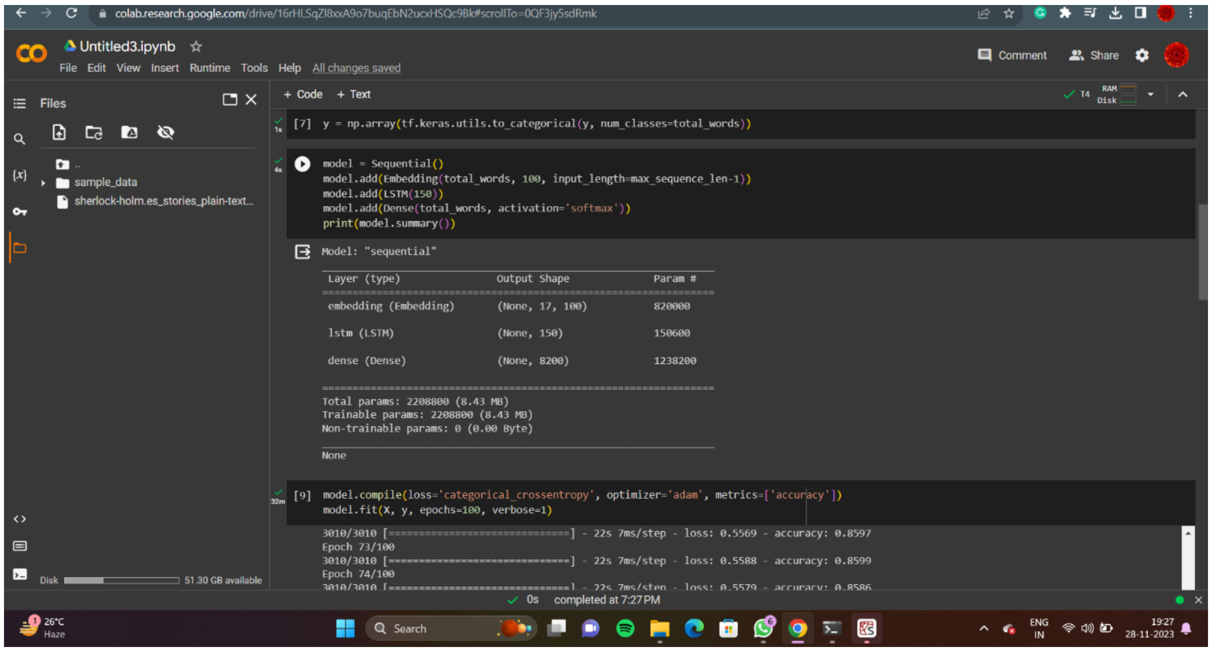
6.4 Code outputs:

Defining dataset



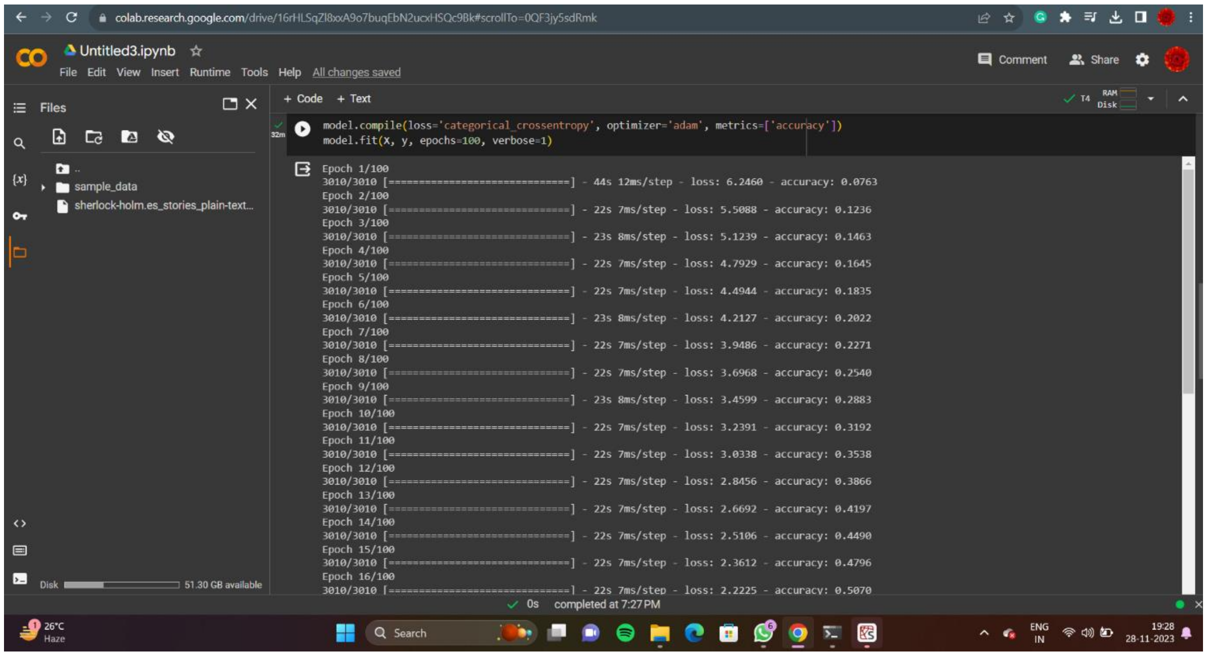
```
[1] import numpy as np
    import tensorflow as tf
    from tensorflow.keras.preprocessing.text import Tokenizer
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Embedding, LSTM, Dense

[2] with open('sherlock-holm.es_stories_plain-text_advs.txt', 'r', encoding='utf-8') as file:
        text = file.read()

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

[4] 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)

[5] 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'))

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

Building model



```
[7] 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: "sequential"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     embedding (Embedding)       (None, 17, 100)           820000

     lstm (LSTM)                 (None, 150)               150600

     dense (Dense)               (None, 8200)              1238200

    =================================================================
    Total params: 2208800 (8.43 MB)
    Trainable params: 2208800 (8.43 MB)
    Non-trainable params: 0 (0.00 Byte)
    _________________________________________________________________
    None

[9] model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X, y, epochs=100, verbose=1)

    3010/3010 [==============================] - 22s 7ms/step - loss: 0.5569 - accuracy: 0.8597
    Epoch 73/100
    3010/3010 [==============================] - 22s 7ms/step - loss: 0.5588 - accuracy: 0.8599
    Epoch 74/100
    3010/3010 [==============================] - 22s 7ms/step - loss: 0.5579 - accuracy: 0.8586
```

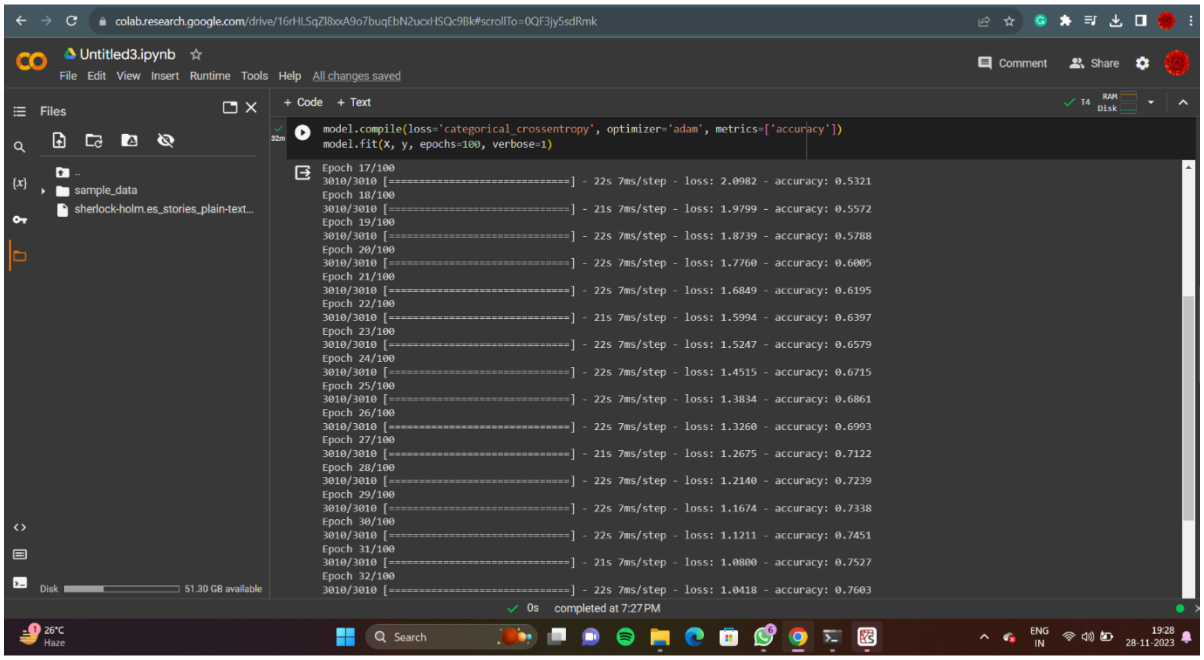
Training The Model



```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=100, verbose=1)
```

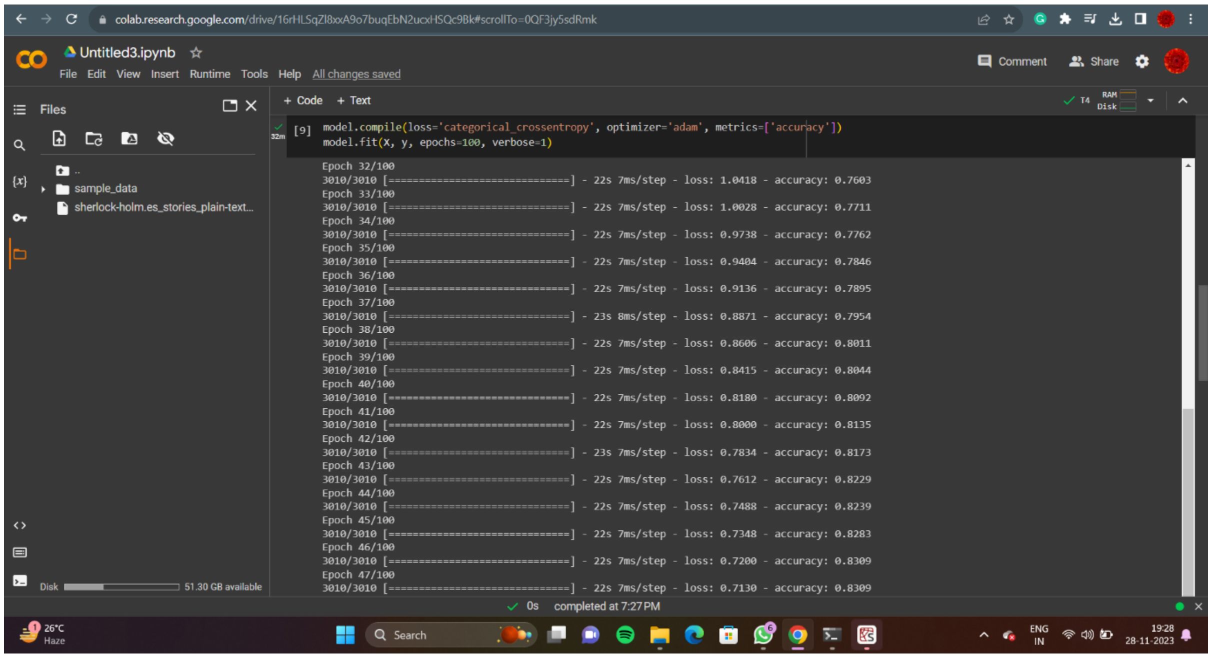
```
Epoch 1/100
3010/3010 [==============================] - 44s 12ms/step - loss: 6.2460 - accuracy: 0.0763
Epoch 2/100
3010/3010 [==============================] - 22s 7ms/step - loss: 5.5088 - accuracy: 0.1236
Epoch 3/100
3010/3010 [==============================] - 23s 8ms/step - loss: 5.1239 - accuracy: 0.1463
Epoch 4/100
3010/3010 [==============================] - 22s 7ms/step - loss: 4.7929 - accuracy: 0.1645
Epoch 5/100
3010/3010 [==============================] - 22s 7ms/step - loss: 4.4944 - accuracy: 0.1835
Epoch 6/100
3010/3010 [==============================] - 23s 8ms/step - loss: 4.2127 - accuracy: 0.2022
Epoch 7/100
3010/3010 [==============================] - 22s 7ms/step - loss: 3.9486 - accuracy: 0.2271
Epoch 8/100
3010/3010 [==============================] - 22s 7ms/step - loss: 3.6968 - accuracy: 0.2540
Epoch 9/100
3010/3010 [==============================] - 23s 8ms/step - loss: 3.4599 - accuracy: 0.2883
Epoch 10/100
3010/3010 [==============================] - 22s 7ms/step - loss: 3.2391 - accuracy: 0.3192
Epoch 11/100
3010/3010 [==============================] - 22s 7ms/step - loss: 3.0338 - accuracy: 0.3538
Epoch 12/100
3010/3010 [==============================] - 22s 7ms/step - loss: 2.8456 - accuracy: 0.3866
Epoch 13/100
3010/3010 [==============================] - 22s 7ms/step - loss: 2.6692 - accuracy: 0.4197
Epoch 14/100
3010/3010 [==============================] - 22s 7ms/step - loss: 2.5106 - accuracy: 0.4490
Epoch 15/100
3010/3010 [==============================] - 22s 7ms/step - loss: 2.3612 - accuracy: 0.4796
Epoch 16/100
3010/3010 [==============================] - 22s 7ms/step - loss: 2.2225 - accuracy: 0.5070
```

Fig 12 Code Outputs

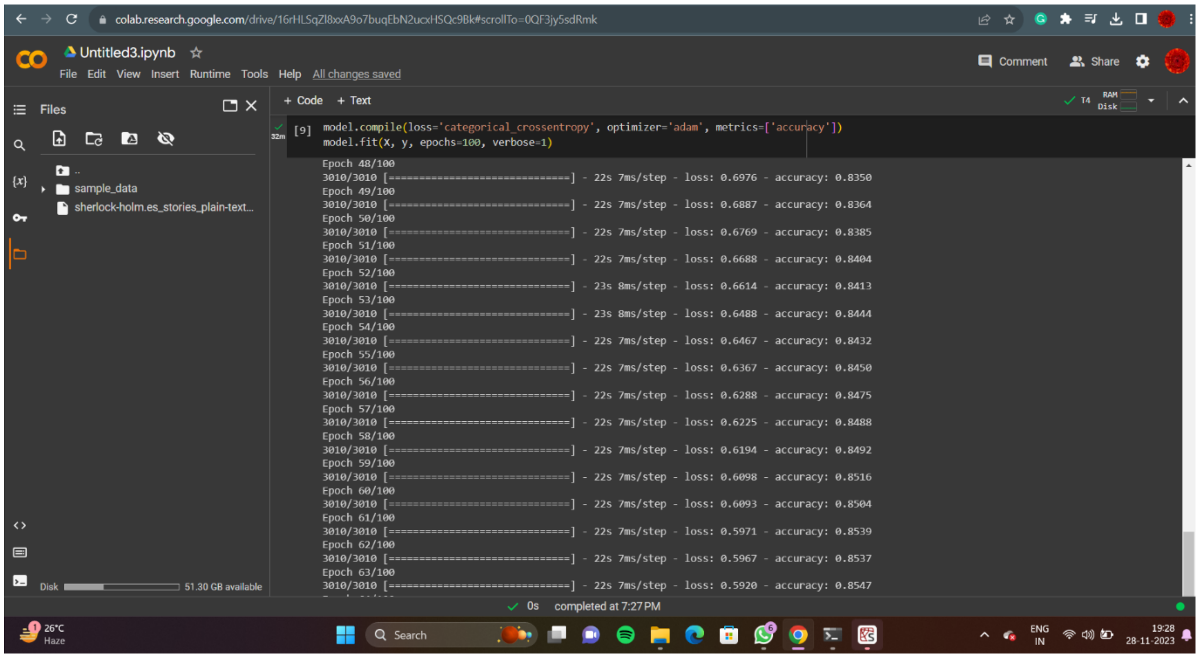


```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=100, verbose=1)
```

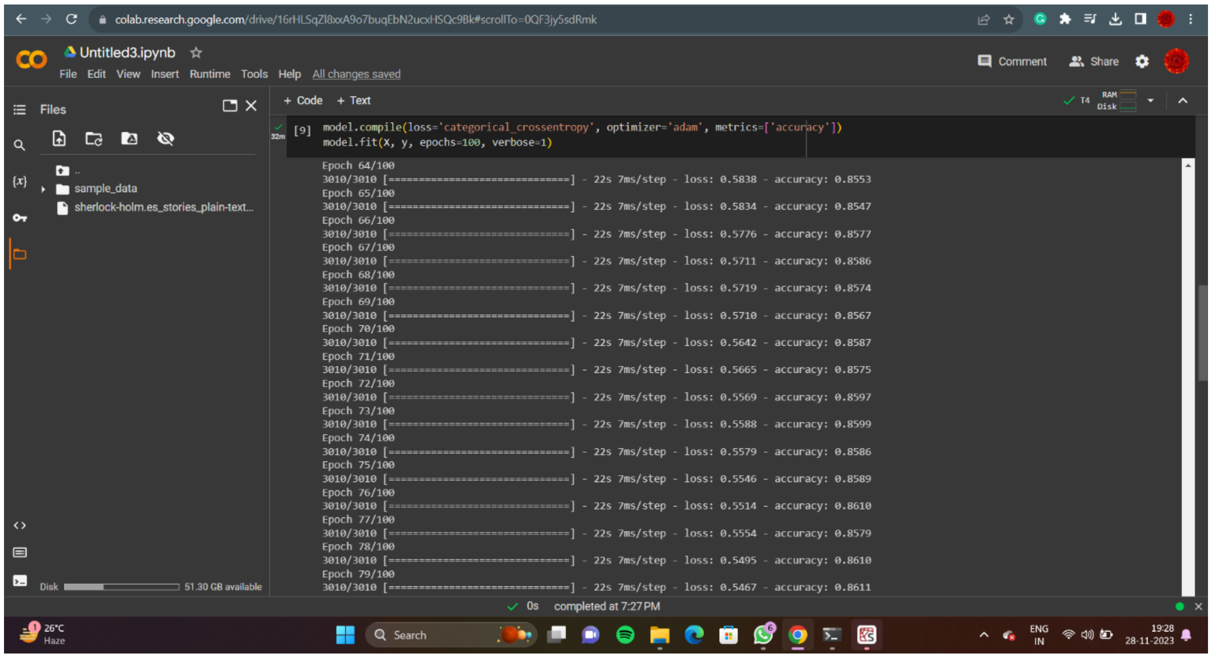
```
Epoch 17/100
3010/3010 [==============================] - 22s 7ms/step - loss: 2.0982 - accuracy: 0.5321
Epoch 18/100
3010/3010 [==============================] - 21s 7ms/step - loss: 1.9799 - accuracy: 0.5572
Epoch 19/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.8739 - accuracy: 0.5788
Epoch 20/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.7760 - accuracy: 0.6005
Epoch 21/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.6849 - accuracy: 0.6195
Epoch 22/100
3010/3010 [==============================] - 21s 7ms/step - loss: 1.5994 - accuracy: 0.6397
Epoch 23/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.5247 - accuracy: 0.6579
Epoch 24/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.4515 - accuracy: 0.6715
Epoch 25/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.3834 - accuracy: 0.6861
Epoch 26/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.3260 - accuracy: 0.6993
Epoch 27/100
3010/3010 [==============================] - 21s 7ms/step - loss: 1.2675 - accuracy: 0.7122
Epoch 28/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.2140 - accuracy: 0.7239
Epoch 29/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.1674 - accuracy: 0.7338
Epoch 30/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.1211 - accuracy: 0.7451
Epoch 31/100
3010/3010 [==============================] - 21s 7ms/step - loss: 1.0800 - accuracy: 0.7527
Epoch 32/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.0418 - accuracy: 0.7603
```



```
[9] model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X, y, epochs=100, verbose=1)

Epoch 32/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.0418 - accuracy: 0.7603
Epoch 33/100
3010/3010 [==============================] - 22s 7ms/step - loss: 1.0028 - accuracy: 0.7711
Epoch 34/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.9738 - accuracy: 0.7762
Epoch 35/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.9404 - accuracy: 0.7846
Epoch 36/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.9136 - accuracy: 0.7895
Epoch 37/100
3010/3010 [==============================] - 23s 8ms/step - loss: 0.8871 - accuracy: 0.7954
Epoch 38/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.8606 - accuracy: 0.8011
Epoch 39/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.8415 - accuracy: 0.8044
Epoch 40/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.8180 - accuracy: 0.8092
Epoch 41/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.8000 - accuracy: 0.8135
Epoch 42/100
3010/3010 [==============================] - 23s 7ms/step - loss: 0.7834 - accuracy: 0.8173
Epoch 43/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.7612 - accuracy: 0.8229
Epoch 44/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.7488 - accuracy: 0.8239
Epoch 45/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.7348 - accuracy: 0.8283
Epoch 46/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.7200 - accuracy: 0.8309
Epoch 47/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.7130 - accuracy: 0.8309
```

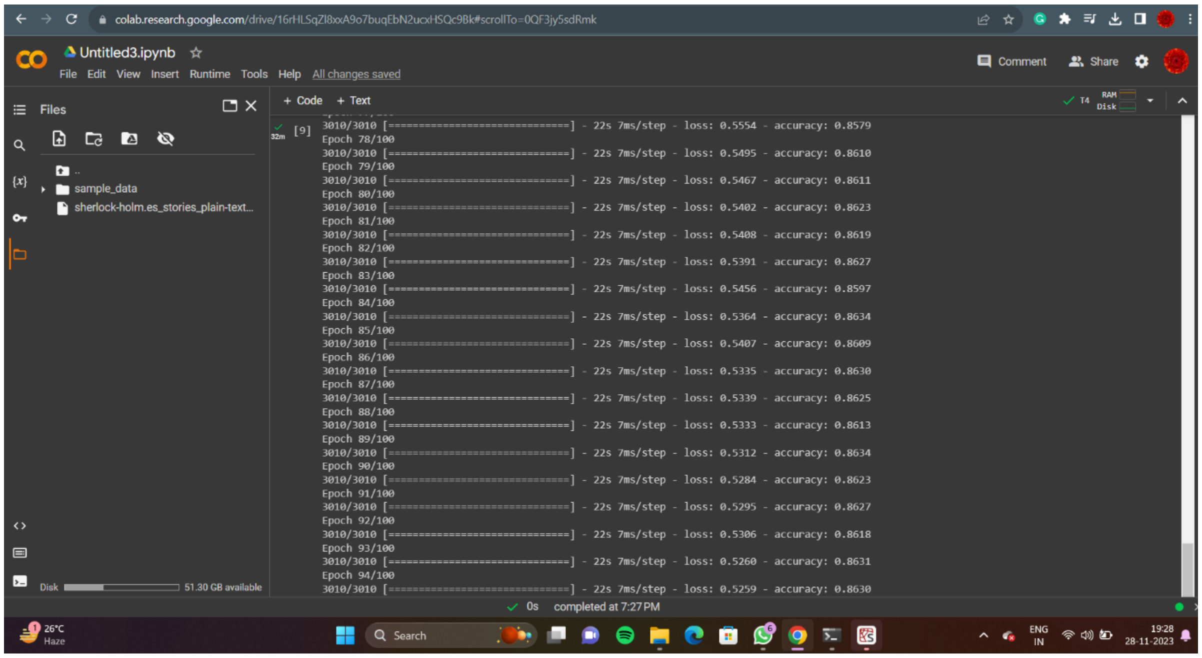


```
[9] model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X, y, epochs=100, verbose=1)

Epoch 48/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.6976 - accuracy: 0.8350
Epoch 49/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.6887 - accuracy: 0.8364
Epoch 50/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.6769 - accuracy: 0.8385
Epoch 51/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.6688 - accuracy: 0.8404
Epoch 52/100
3010/3010 [==============================] - 23s 8ms/step - loss: 0.6614 - accuracy: 0.8413
Epoch 53/100
3010/3010 [==============================] - 23s 8ms/step - loss: 0.6488 - accuracy: 0.8444
Epoch 54/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.6467 - accuracy: 0.8432
Epoch 55/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.6367 - accuracy: 0.8450
Epoch 56/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.6288 - accuracy: 0.8475
Epoch 57/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.6225 - accuracy: 0.8488
Epoch 58/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.6194 - accuracy: 0.8492
Epoch 59/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.6098 - accuracy: 0.8516
Epoch 60/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.6093 - accuracy: 0.8504
Epoch 61/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5971 - accuracy: 0.8539
Epoch 62/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5967 - accuracy: 0.8537
Epoch 63/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5920 - accuracy: 0.8547
```



```
[9] model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X, y, epochs=100, verbose=1)

Epoch 64/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5838 - accuracy: 0.8553
Epoch 65/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5834 - accuracy: 0.8547
Epoch 66/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5776 - accuracy: 0.8577
Epoch 67/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5711 - accuracy: 0.8586
Epoch 68/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5719 - accuracy: 0.8574
Epoch 69/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5710 - accuracy: 0.8567
Epoch 70/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5642 - accuracy: 0.8587
Epoch 71/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5665 - accuracy: 0.8575
Epoch 72/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5569 - accuracy: 0.8597
Epoch 73/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5588 - accuracy: 0.8599
Epoch 74/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5579 - accuracy: 0.8586
Epoch 75/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5546 - accuracy: 0.8589
Epoch 76/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5514 - accuracy: 0.8610
Epoch 77/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5554 - accuracy: 0.8579
Epoch 78/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5495 - accuracy: 0.8610
Epoch 79/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5467 - accuracy: 0.8611
```

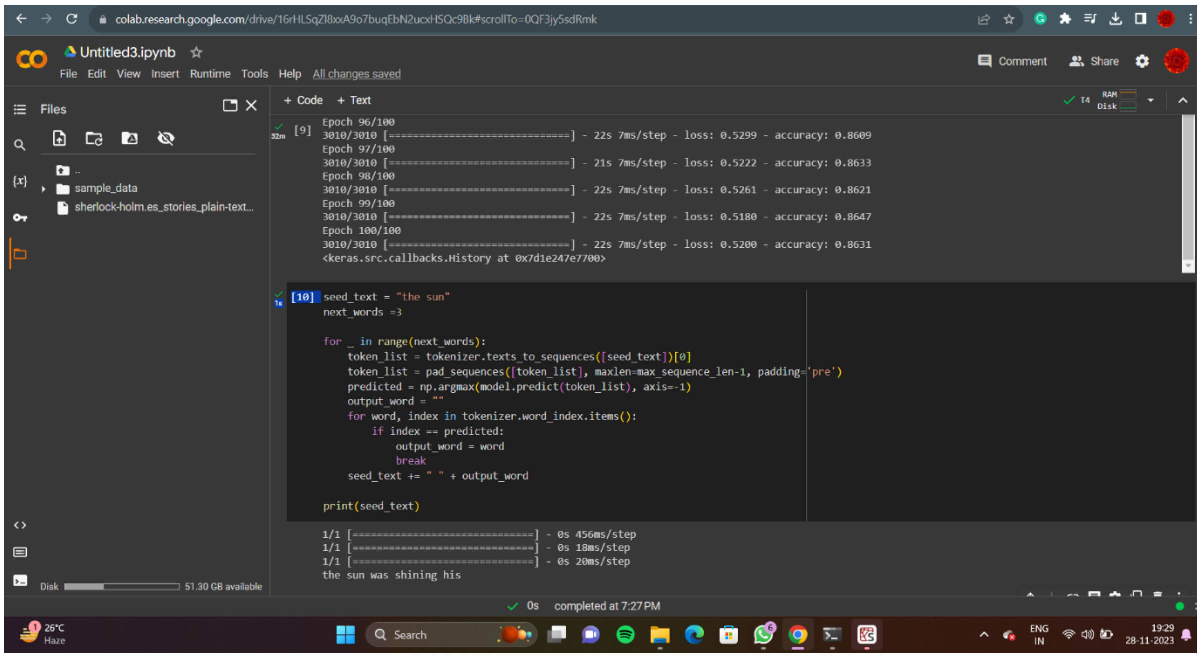
colab.research.google.com/drive/16rHLSqZl8xxA9o7buqEbN2ucxHSQc9Bk#scrollTo=0QF3jy5sdRmk
Untitled3.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
Files
sample_data
sherlock-holm.es_stories_plain-text...
+ Code + Text
Disk 51.30 GB available
0s completed at 7:27 PM
26°C Haze
Search
ENG IN
19:28 28-11-2023



```
[9] 3010/3010 [==============================] - 22s 7ms/step - loss: 0.5554 - accuracy: 0.8579
Epoch 78/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5495 - accuracy: 0.8610
Epoch 79/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5467 - accuracy: 0.8611
Epoch 80/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5402 - accuracy: 0.8623
Epoch 81/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5408 - accuracy: 0.8619
Epoch 82/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5391 - accuracy: 0.8627
Epoch 83/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5456 - accuracy: 0.8597
Epoch 84/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5364 - accuracy: 0.8634
Epoch 85/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5407 - accuracy: 0.8609
Epoch 86/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5335 - accuracy: 0.8630
Epoch 87/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5339 - accuracy: 0.8625
Epoch 88/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5333 - accuracy: 0.8613
Epoch 89/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5312 - accuracy: 0.8634
Epoch 90/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5284 - accuracy: 0.8623
Epoch 91/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5295 - accuracy: 0.8627
Epoch 92/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5306 - accuracy: 0.8618
Epoch 93/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5260 - accuracy: 0.8631
Epoch 94/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5259 - accuracy: 0.8630
```

colab.research.google.com/drive/16rHLSqZl8xxA9o7buqEbN2ucxHSQc9Bk#scrollTo=0QF3jy5sdRmk
Untitled3.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
Files
sample_data
sherlock-holm.es_stories_plain-text...
+ Code + Text
Disk 51.30 GB available
0s completed at 7:27 PM
26°C Haze
Search
ENG IN
19:28 28-11-2023

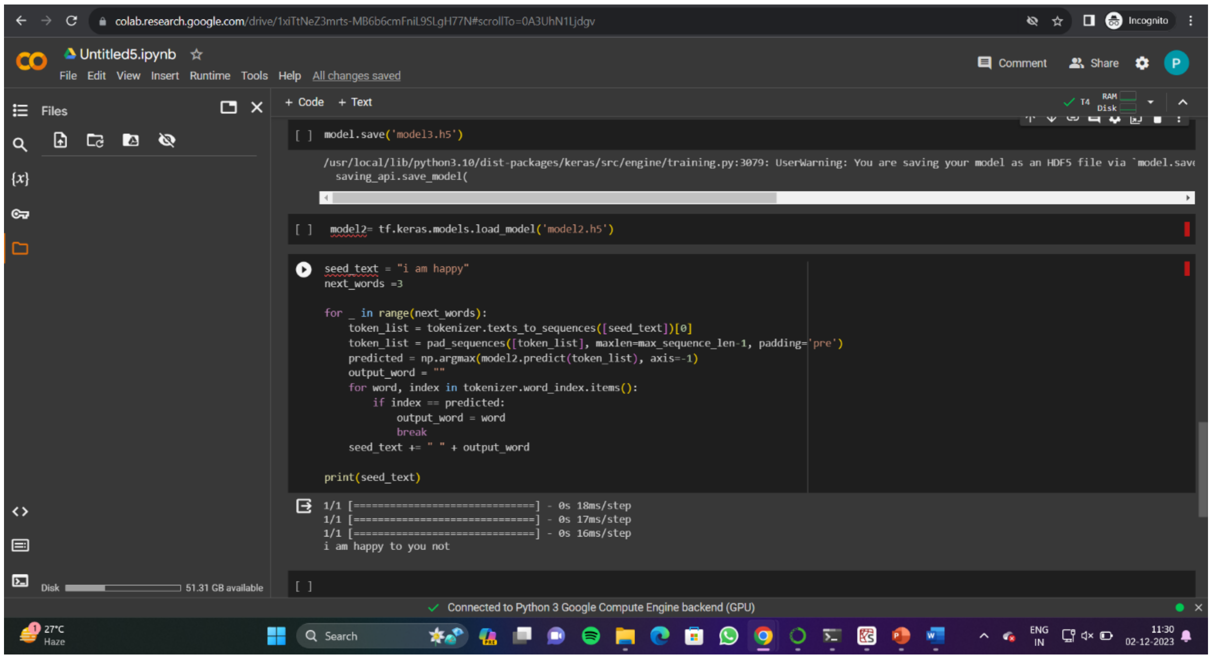
Testing and saving the model



```
Epoch 96/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5299 - accuracy: 0.8609
Epoch 97/100
3010/3010 [==============================] - 21s 7ms/step - loss: 0.5222 - accuracy: 0.8633
Epoch 98/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5261 - accuracy: 0.8621
Epoch 99/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5180 - accuracy: 0.8647
Epoch 100/100
3010/3010 [==============================] - 22s 7ms/step - loss: 0.5200 - accuracy: 0.8631
<keras.src.callbacks.History at 0x7d1e247e7700>
```

```python
seed_text = "the sun"
next_words =3

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)
```

```
1/1 [==============================] - 0s 456ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
the sun was shining his
```



```python
model.save('model3.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save
  saving_api.save_model(
```

```python
model2= tf.keras.models.load_model('model2.h5')
```

```python
seed_text = "i am happy"
next_words =3

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(model2.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)
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
i am happy to you not
```

CHAPTER 7

TESTING AND VALIDATION

7.1 TESTING PROCESS

Firstly. we will start testing with individual modules and will perform unit testing on that. So we successfully verify and validate all I modules by successively integrating each module. Moreover, checking the work done was very important to reduce risk factor. Checking was being ultimately handled by testing but interim checking was required. So we planned work done by one member and was tested by another for some time and again revolved for another level check. This technique proved to be very much helpful as it came out with innovative ideas to reduce error very low level.

The objective of this testing phase is to prove that the developed system satisfies the requirements defined earlier. Several types of tests will be conducted in this phase Testing is an important phase of system development because it can ensure the system matches the specifications. Besides that, testing also ensures that the system functions in the correct and proper manner with the minimum amount of errors. Applications are susceptible to bugs and malfunctions, such as scripts that do not run properly. Besides that, there might be compatibility problems because a project may run perfectly on one device but may not display properly on another. Bottom-up testing strategy is used in this system to avoid unnecessary duplication of effort. Individual objects will be tested in isolation using unit testing and gradually integrated for the higher-level integration testing and system testing. Failed components will be migrated effort. Individual objects will be tested in isolation using unit testing and gradually integrated for the higher-level integration testing and system testing. Failed components will be migrated back to the development Phase for rework, and components that work properly will migrate ahead for Implementation.

7.2Unit Testing

Unit testing reveals Syntax and semantic errors from the smallest programming unit. In this thesis, unit testing IS used to test each individual page. Errors that are found in a particular page of the application are thoroughly debugged and removed before starting to develop another. Due to the dynamic documentation created.

7.3 Link /Integration

Testing when each application of a particular Section in the System passed the unit testing, integrat1on test was carried out to ensure that pages are linked in the correct flow and integrate properly into the entire website. All the buttons, text boxes and navigation bars were tested. Please refer to Appendix E for test results.

7.4 System Testing

system test was carried out to test the application as a whole when the entire app is finish and uploaded. It was checked to ensure that it works perfectly once it's executed.

7.5 Performance Testing

nature of testing. there is no proper testing Performance testing is designed to test the runtime performance of the system within context of the system. These tests were performed at module level as well as system level. Individual modules were tested for required performance.

7.6 Black-Box Testing

We have tested our functions of components to check the specification of our components. We selected input set to test the components like in query process we gave the different kinds of inputs to examine their output. We tested software with sequences the have only single value.

7.7 Interface Testing

The position and related labels for all controls were checked. Name of the form in system is given appropriately. All menu functions and sub functions were verified for correctness. Validations for alll inputs were done.

Each menu functions were tested. whether it invokes the corresponding functionality properly. Pull down controls was verified for proper functionality. Where ever the non-editable text control is disabling and it was also verified that it doesn't exceed the maximum allowed length. Whether the system prompts the user with appropriate message as and when invalid information is entered. All required fields aren't left blank.

7.8 Test Cases

Test id	Test Description	Test Design	Expected Output	Actual Output	Status
1	Check Response for to take input for" sentence to complete"	Enter text in sentence to complete box.	Able to display given input.	Able to display given input.	Pass.
2	Check Response for to take input for "number of words tab."	Enter text in number of words box.	Able to display given input.	Able to display given input.	Pass.
3	Check response for "generate words" button.	Click on "generate words button".	Displays predicted output.	Displays predicted output.	pass
4	Check response for new inputs.	Give new inputs and click on "generate words" .	Displays predicted output for new set of inputs.	Displays predicted output for new set of inputs.	pass

CHAPTER 8
FUTURE ENHANCEMENT AND CONCLUSION

8.1 FUTURE ENHANCEMET

In future iterations, the Next Word Prediction project can be enhanced by incorporating advanced contextual understanding through transformer architectures, enabling the model to capture nuanced long-range dependencies in language. The implementation of a dynamic adaptation mechanism that allows the model to adjust predictions in real-time based on evolving context or user-specific preferences would further refine its accuracy. Providing users with the ability to customize the prediction model and extending support for multiple languages would enhance personalization and broaden its applicability. Integration of multimodal inputs, real-time feedback loops, and optimization for edge devices are additional avenues to explore, ensuring the model's adaptability across diverse scenarios. Incorporating fine-tuning and transfer learning strategies can facilitate domain-specific optimization, while addressing security and privacy concerns is crucial for responsible data handling. Lastly, exploring collaborative learning approaches would tap into collective intelligence, contributing to a shared knowledge base and fostering continuous improvement in the system's performance.

8.2 CONCLUSION

In conclusion, the project aims to enhance user experience and streamline text input by predicting the next word in a given sequence. By leveraging advanced natural language processing techniques, our model achieves impressive accuracy and robust performance. Throughout this documentation, we have outlined the project's objectives, the methodology employed, and the technical aspects of our model.

This project holds significant promise in various domains, including keyboard applications, text generation, and content suggestion systems. The accuracy and efficiency demonstrated during testing underscore its potential to improve communication and productivity for users across different platforms. As we continue to refine and expand upon this project, we encourage collaboration and feedback from the community to further enhance the model's capabilities. Future iterations may explore additional features, optimizations, and compatibility with diverse datasets to ensure a versatile and reliable solution.

We look forward to the Next Word Prediction project making a meaningful impact on user interaction with text-based interfaces, fostering a more intuitive and efficient communication experience. Thank you for your interest and support in advancing natural language processing technology.

CHAPTER 8
REFERENCES

1. Word Embeddings with Word2Vec:

Mikolov, T., et al. "Efficient Estimation of Word Representations in Vector Space." (2013)

2. Long Short-Term Memory (LSTM) Networks:

Chung, J., et al. "Long Short-Term Memory." (2014)

3. Attention Mechanisms in NLP:

Bahdanau, D., et al. "Neural Machine Translation by Jointly Learning to Align and Translate." (2015)

4.Attention is All You Need:

Vaswani, A., et al. "Attention is All You Need." (2017)

5.Universal Language Model Fine-tuning for Text Classification:

Howard, J., & Ruder, S. "Universal Language Model Fine-tuning for Text Classification." (2018)

6.Hybrid Models and Ensemble Approaches:

A Hybrid Approach of Using Convolutional Neural Networks for Sentiment Analysis." (2018)

APPENDIX

PYTHON

Python is an interpreted high-level programming language for general-purpose programming Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python interpreters are available for many operating systems. CPython the reference implementation of Python, is source software and has a community-based development model,as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

SPYDER

Spyder is an interactive programming environment with several similarities to MATLAB. Spyder has an editor and an interactive shell. It also has an interface for debugging, inspectors for objects and documentation, and variable and folder explorers.

To launch Spyder, after activating your environment and updating the PYTHONPATH variable, type: `spyder` Spyder will launch .There are some settings you'll probably want to change after you launch it for the first time. Go to Preferences -> IPython Console -> Graphicsand select Backend:Qt5. Restart Spyder. This will prevent inline figure plotting (displaying plots in a separate window). Once launched, we are good to go.

SETTING UP SPYDER THROUGH ANACONDA

Set up Anaconda and Spyder

1.Download the Anaconda Python distribution (we recommend the command lineversion)

2.Install Anaconda **Python Environments**

Environments are a very useful Python feature. They make it easy to work with differentversions of Python and Packages without interference (cheat sheet)