

# **WORD WISE PREDICTION USING DEEP LEARNING**

A Project Report on Submitted in partial fulfillment of  
the requirements for the award of the degree of

**BACHELOR OF TECHNOLOGY**

IN

**COMPUTER SCIENCE AND ENGINEERING**

**SUBMITTED BY**

**NAME**

**J. AAKANKSHA**

**ROLL NO**

**20NE1A0564**

Under the Esteemed Guidance of

**Mr. S. ANIL KUMAR**, M. Tech (Ph. D)

Assoc. Professor



**Department of Computer Science and Engineering**

**TIRUMALA ENGINEERING COLLEGE**

(Approved by AICTE & Affiliated to JNTU KAKINADA, Accredited by NAAC A+ & NBA)  
Jonnalagadda, Narasaraopet, GUNTUR (Dt), A.P.

2020-2024

# **TIRUMALA ENGINEERING COLLEGE**

(Approved by AICTE & Affiliated to JNTUK, Accredited by NAAC & NBA)

Jonnalagadda, Narasaraopet, GUNTUR (Dt), A.P.

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



## **CERTIFICATE**

This is to certify that the project report entitled “**WORD WISE PREDICTION USING DEEP LEARNING**” is the bonafied work done by **J. AAKANKSHA (Roll No: 20NE1A0564)** in partial fulfillment of the requirements for the award of “Bachelor of Technology” degree in the **Department of CSE** from J.N.T.U KAKINADA during the year 2023 – 2024 under our guidance and supervision and worth of acceptance of requirements of the university.

**Project Guide**

**Mr. S. Anil Kumar** M. Tech, (Ph. D)

**Head of the Department**

**Dr. N. Gopala Krishna** M. Tech, Ph.D, MISTE.,

**Project Coordinator**

**Mr. S. Anil Kumar** M. Tech, (Ph. D)

**External Examiner**

## ACKNOWLEDGEMENT

We wish to express our thanks to various personalities who are responsible for the completion of the project. We are extremely thankful to our principal sir **Dr. Y. V. Narayana M.E., Ph. D, FIETE** for his kind attention and valuable guidance throughout the course.

We express our deep felt gratitude to our **H.O.D Dr. N. Gopala Krishna M. Tech, PhD, MISTE** and **Mr. S. Anil Kumar M. Tech, (Ph.D)**, coordinator of the project for extending their encouragement. Their profound knowledge and willingness have been a constant source of inspiration for us throughout the project work.

We wish to express our sincere deep sense of gratitude to our guide **Mr. S. Anil Kumar M. Tech, (Ph.D)** for significant suggestions and help in every respect to accomplish the project work. His persisting encouragement, everlasting patience and keen interest in discussions have benefited to us.

We affectionately acknowledge the encouragement received from my friends and those who involved in giving valuable suggestions and clarifying out doubts which had really helped us in successfully completing my project.

### PROJECT ASSOCIATES

**J. AAKANKSHA**

**20NE1A0564**

## **INDEX**

<b>CONTENTS</b>	<b>PAGE NO</b>
<b>1. INTRODUCTION</b>	<b>1</b>
<b>2. LITERATURE SURVEY</b>	<b>16</b>
<b>3. SYSTEM ANALYSIS</b>	
3.1 Existing System	19
3.2 Proposed System	23
3.3 Feasibility Study	37
3.3.1 Technical Feasibility	
3.3.2 Operational Feasibility	
3.3.3 Economic Feasibility	
3.3.4 Social Feasibility	
3.4 Requirements Specification	39
<b>4. DESIGN</b>	
4.1 UML Diagrams	40
4.1.1 Use Case Diagram	41
4.1.2 Sequence Diagram	42
4.1.3 Class Diagram	43
4.1.4 Activity Diagram	44
<b>5. IMPLEMENTATION</b>	<b>45</b>
5.1 Sample Code	47
<b>6. OUTPUT SCREENS</b>	<b>54</b>
<b>7. SYSTEM TESTING</b>	
7.1 Testing Objective	58
<b>8. CONCLUSION</b>	<b>61</b>
<b>9. FUTURE SCOPE</b>	<b>62</b>
<b>10. BIBLIOGRAPHY</b>	<b>63</b>

# **ABSTRACT**

## **ABSTRACT**

Word wise prediction is an input technology that simplifies the process of typing by suggesting the next word for a user to select, as typing in a conversation consumes time. You might be using it daily when you write texts or emails without realizing it.

Most of the keyboards in smartphones suggest next word prediction features. Google also uses next word prediction based on our browsing history. So, preloaded data is also stored in the keyboard function of our smartphones to predict the next word correctly.

By predicting the next word in a sequence, the number of keystrokes of the user can be reduced. In this project, we have used a deep learning approach by using Long Short Term Memory(LSTM) technique which gives better prediction accuracy than the machine learning approach.

# INTRODUCTION

# 1. INTRODUCTION

Wouldn't it be cool for your device to predict what could be the next word that you are planning to type? This is similar to how a predictive text keyboard works on apps like What's App, Facebook Messenger, Instagram, e-mails, or even Google searches. It will consider the last word of a particular sentence and predict the next possible word. You might be using it daily when you write texts or emails without realizing it. Most of the keyboards in smartphones give next word prediction features; google also uses next word prediction based on our browsing history. So, preloaded data is also stored in the keyboard function of our smartphones to predict the next word correctly. By predicting the next word in a sequence, the number of keystrokes of the user can be reduced. Three deeplearning techniques namely Long Short Term Memory (LSTM), Bi-LSTM and BERT have been explored for the task of predicting the next word. In this project we have used the LSTM technique for predicting the next word.

Word prediction tools were developed which can help to communicate and also to help the people with less typing speed. The research on word prediction has been performing well. Word prediction technique does the task of guessing the preceding word that is likely to continue with few initial text fragments. Existing systems work on a word prediction model, which suggests the next immediate word based on the current available word. These systems work using machine learning algorithms which have limitations to create accurate sentence structure. Developing technologies have been producing more accurate outcomes than the existing system technologies, models developed using deep learning concepts are capable of handling more data efficiently.



## **Deep Learning**

Deep learning is a type of machine learning and artificial intelligence (AI) that imitates the way humans gain certain types of knowledge. Deep learning is an important element of data science, which includes statistics and predictive modelling, it is extremely beneficial to data scientists who are tasked with collecting, analyzing and interpreting large amounts of data; deep learning makes this process faster and easier.

Deep learning can be thought of as a way to automate predictive analytics. While traditional machine learning algorithms are linear, deep learning algorithms are stacked in a hierarchy of increasing complexity and abstraction.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labelled data and neural network architectures that contain many layers.

### **How deep learning works**

Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks. The term "deep" usually refers to the number of hidden layers in the neural network. Traditional Neural Networks only contain 2-3 hidden layers, while deep networks can have as many as 150.

### **Deep Learning Algorithms**

#### **How Deep Learning Algorithms Work?**

While Deep Learning algorithms feature self-learning representations, they depend upon ANNs that mirror the way the brain computes information. During the training process, algorithms use unknown elements in the input distribution to extract features, group objects, and discover useful data patterns.

## **Deep learning models use several algorithms**

While no one network is considered perfect, some algorithms are better suited to perform specific tasks. To choose the right ones, it's good to gain a solid understanding of all primary algorithms.

- **Algorithms used in Deep Learning**
  - Convolutional Neural Network (CNN)
  - Recurrent Neural Networks (RNNs).
  - Long Short-Term Memory Networks (LSTMs)
  - Stacked Auto-Encoders
  - Deep Belief Networks (DBN)

### **Convolutional Neural Network (CNN)**

CNN is also known as Conv. Nets consist of multiple layers and are mainly used for image processing and object detection. Yann LeCun developed the first CNN in 1988 when it was called LeNet. It was used for recognizing characters like ZIP codes and digits.

CNNs have several advantages over traditional neural networks for image-related tasks. They can automatically learn hierarchical features from raw input data, reducing the need for manual feature extraction. Additionally, CNNs leverage parameter sharing and local connectivity, making them computationally efficient and capable of handling large input data, such as high-resolution images.

### **Recurrent Neural Networks (RNN)**

- Recurrent Neural Networks are a versatile tool that can be used in a variety of situations. They're employed in a variety of methods for language modelling and text generators. This type of neural network is used to create labels for images that aren't tagged when paired with Convolutional Neural Networks.

- However, there is one flaw with recurrent neural networks. They have trouble learning long-range dependencies, which means they don't comprehend relationships between data that are separated by several steps.
- The output from the LSTM becomes an input to the current phase and can memorize previous inputs due to its internal memory. RNNs are commonly used for image captioning, time-series analysis, natural- language processing, handwriting recognition, and machine translation.

## **Long Short-Term Memory Networks (LSTMs)**

LSTMs are a type of Recurrent Neural Network (RNN) that can learn and memorize long term dependencies. Recalling past information for long periods is the default behavior. LSTMs retain information over time. They are useful in time-series prediction because they remember previous inputs. LSTMs have a chain-like structure where four interacting layers communicate in a unique way.

## **Stacked Auto-Encoders**

Autoencoders are a specific type of feed forward neural network in which the input and output are identical. Geoffrey Hinton designed autoencoders in the 1980s to solve unsupervised learning problems. They are trained neural networks that replicate the data from the input layer to the output layer. Autoencoders are used for different cases in our daily life. It is mainly used in the field of medicine and used for the prediction and then processing.

## **Deep Belief Networks (DBN)**

DBNs are generative models that consist of multiple layers of stochastic, latent variables. The latent variables have binary values and are often called hidden units.

DBNs are a stack of Boltzmann Machines with connections between the layers, and each RBM layer communicates with both the previous and subsequent layers.

Deep Belief Networks (DBNs) are used for image recognition, video recognition, and motion capture data. As for the above algorithms we use LSTM algorithm for the implementation of next word prediction using deep learning.

## Long Short-Term Memory Networks (LSTMs)

LSTMs are a type of Recurrent Neural Network (RNN) that can learn and memorize long term dependencies. Recalling past information for long periods is the default behavior.

LSTMs retain information over time. They are useful in time-series prediction because they remember previous inputs. LSTMs have a chain-like structure where four interacting layers communicate in a unique way. Besides time-series predictions, LSTMs are typically used for speech recognition, music composition, and pharmaceutical development.

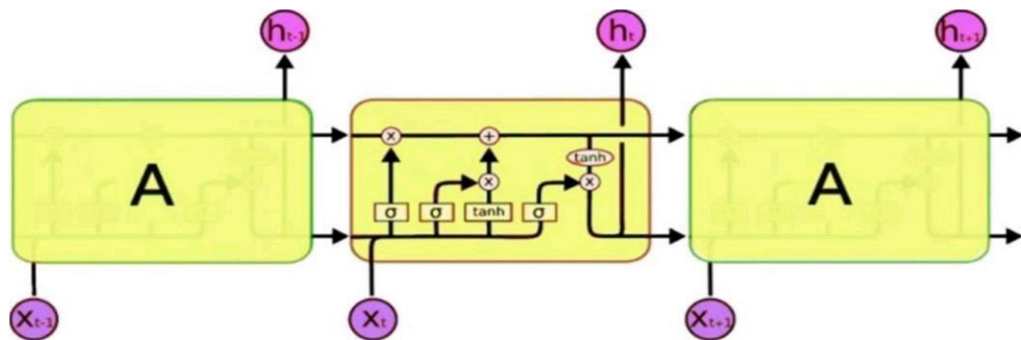


Fig: Architecture of LSTM

A typical LSTM network is composed of different memory blocks called cells. There are two states that are being transferred to the next cell; the cell state and the hidden state. The memory blocks are responsible for remembering things and manipulations to this memory are done through three major mechanisms, called gates. Each of them is being discussed below.

- **Forget Gate**

A forget gate is responsible for removing information from the cell state. The information that is no longer required for the LSTM to understand things or the information that is of less importance is removed via the multiplication of a filter. This is required for optimizing the performance of the LSTM network.

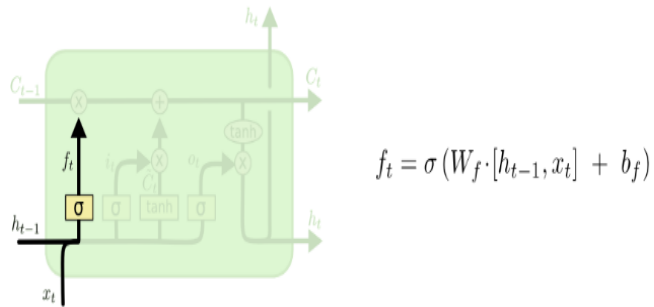


Fig: Forget Gate

- **Input Gate**

The input gate plays a critical role in controlling the flow of information into the memory cell. It determines how much of the new input information should be stored in the cell state. The input gate accomplishes this by using a sigmoid activation function to produce a value between 0 and 1 for each component of the input data, indicating the proportion.

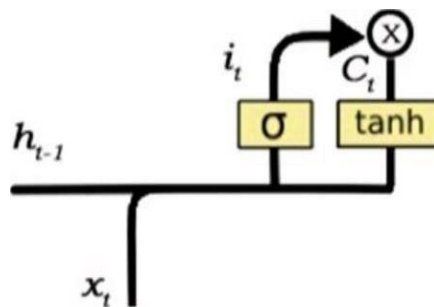


Fig: Input Gate

- Regulating what values need to be added to the cell state by involving a sigmoid function. This is basically very similar to the forget gate and acts as a filter for all the information from  $h_{t-1}$  and  $x_t$ .
- Creating a vector containing all possible values that can be added (as perceived from  $h_{t-1}$  and  $x_t$ ) to the cell state. This is done using the  $\tanh$  function, which outputs values from -1.
- Multiplying the value of the regulatory filter (the sigmoid gate) to the created vector (the  $\tanh$  function) and then adding this useful information to the cell state via an additional operation.
- Once this three-step process is done, we ensure that only that information is added to the cell state that is important and is not redundant.

- **Output Gate**

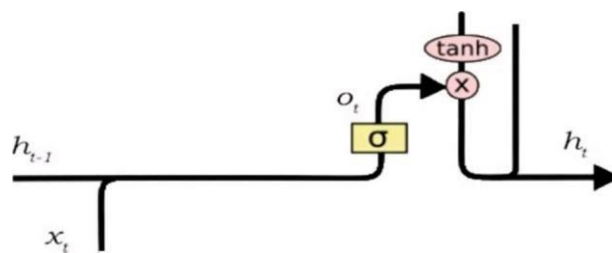


Fig: Output Gate

The functioning of an output gate can again be broken down to three steps:

- Creating a vector after applying the  $\tanh$  function to the cell state, thereby scaling the values to the range -1 to +1.
- Making a filter using the values of  $h_{t-1}$  and  $x_t$ , such that it can regulate the values that need to be output from the vector created above. This filter again employs a sigmoid function.
- Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as an output and also to the hidden state of the next cell.
- This mechanism contributes to the network's ability to capture long-term dependencies and make accurate predictions, making LSTM networks highly effective for tasks involving sequential data processing such as language modeling, speech recognition, time.

## **Applications of LSTM**

LSTM models need to be trained with a training dataset prior to its employment in real-world applications. Some of the most demanding applications are discussed below

- Robot control
- Human action recognition
- Time series prediction
- Speech recognition
- Rhythm learning
- Music composition
- Grammar learning
- Handwritten recognition

## **Applications of Deep Learning**

Deep Learning applications may seem disillusioning to a normal human being, but those with the privilege of knowing the machine learning world understand the dent that deep learning is making globally by exploring and resolving human problems in every domain.

### **Self-Driving Cars**

Deep Learning is the force that is bringing autonomous driving to life.

A million sets of data are fed to a system to build a model, to train the machines to learn, and then test the results in a safe environment. The Uber Artificial Intelligence Labs at Pittsburgh is not only working on making driverless cars humdrum but also integrating several smart features such as food delivery options with the use of driverless cars. The major concern for autonomous car developers is handling unprecedented scenarios.

### **Virtual Assistants**

The most popular application of deep learning is virtual assistants ranging from Alexa

to Siri to Google Assistant. Each interaction with these assistants provides them with an opportunity to learn more about your voice and accent, thereby providing you a secondary human interaction experience. Virtual assistants use deep learning to know more about their subjects ranging from your dine-out preferences to your most visited spots or your favorite songs. They learn to understand your commands by evaluating natural human language to execute them.

## **Entertainment**

Netflix and Amazon are enhancing their deep learning capabilities to provide a personalized experience to its viewers by creating their personas factoring in show preferences, time of access, history, etc. to recommend shows that are of liking to a particular viewer. VEVO has been using deep learning to create the next generation of data services for not only personalized experiences for its users and subscribers, but also artists, companies, record labels, and internal business groups to generate insights based on performance and popularity.

## **Health care**

Helping early, accurate and speedy diagnosis of life-threatening diseases, augmented clinicians addressing the shortage of quality physicians and healthcare providers, pathology results and treatment course standardization, and understanding genetics to predict future risk of diseases and negative health episodes are some of the Deep Learning projects picking up speed in the Healthcare domain.

## **Fraud Detection**

Another domain benefitting from Deep Learning is the banking and financial sector that is plagued with the task of fraud detection with money transactions going digital. Autoencoders in Keras and TensorFlow are being developed to detect credit card frauds, saving billions of dollars of cost in recovery and insurance for financial institutions. Fraud prevention and detection are done based on identifying patterns.



## **Natural Language Processing**

### **What is Natural Language Processing?**

Natural language processing (NLP) is the ability of a computer program to understand human language as it is spoken and written referred to as natural language. It is a component of artificial intelligence (AI).

NLP plays a crucial role in bridging the gap between humans and machines by enabling computers to understand, process, and generate natural language data. As AI and machine learning advancements continue, NLP is expected to play an increasingly important role in various domains, revolutionizing the way we interact with technology and information.

### **How Natural Language Processing work?**

There are two main phases to natural language processing: data pre-processing and algorithm development. Data pre-processing involves preparing and "cleaning" text data for machines to be able to analyze it. pre-processing puts data in workable form and highlights features in the text that an algorithm can work with. There are several ways this can be done, including

#### **Tokenization**

This is when text is broken down into smaller units to work with. Stop word removal. This is when common words are removed from text so unique words that offer the most information about the text remain.

#### **Part-of-speech tagging**

This is when words are marked based on the part-of speech they are such as nouns, verbs and adjectives. Once the data has been pre-processed, an algorithm is developed to process it. There are many different natural language processing algorithms.

## **Rules-based system**

^ This system uses carefully designed linguistic rules. This approach was used early on in the development of natural language processing, and is still used.

## **Machine learning-based system**

Machine learning algorithms use statistical methods. They learn to perform tasks based on training data they are fed, and adjust their methods as more data is processed. Using a combination of machine learning, deep learning and neural networks, natural language processing algorithms hone their own rules through repeated processing and learning.

## **Natural Language Generation**

This uses a database to determine semantics behind words and generate new text. Example: An algorithm could automatically write a summary of findings from a business intelligence platform, mapping certain words and phrases to features of the data in the BI platform.

Earlier approaches to natural language processing involved a more rules-based approach, where simpler machine learning algorithms were told what words and phrases to look for in text and given specific responses when those phrases appeared. But deep learning is a more flexible, intuitive approach in which algorithms learn to identify speakers' intent from many examples almost like how a child would learn human language.

Three tools used commonly for natural language processing include Natural Language Toolkit (NLTK), Genism and Intel natural language processing Architect. NLTK is an open-source Python module with data sets and tutorials. Genism is a Python library for topic modelling and document indexing. Intel NLP Architect is another Python library for deep learning topologies and techniques.

## **What is Natural Language Processing used for?**

Some of the main functions that natural language processing algorithms perform are:

### **Text classification**

This involves assigning tags to texts to put them in categories. This can be useful for sentiment analysis, which helps the natural language processing algorithm determine the sentiment, or emotion behind a text.

For example, when brand A is mentioned in X number of texts, the algorithm can determine how many of those mentions were positive and how many were negative. It can also be useful for intent detection, which helps predict what the speaker or writer may do based on the text they are producing.

### **Text extraction**

This involves automatically summarizing text and finding important pieces of data. One example of this is keyword extraction, which pulls the most important words from the text, which can be useful for search engine optimization.

Doing this with natural language processing requires some programming; it is not completely automated. However, there are plenty of simple keyword extraction tools that automate most of the process - the user just has to set parameters within the program. For example is named entity recognition, which extracts the names of people, places and other entities from text.

### **Machine Translation**

This is the process by which a computer translates text from one language, such as English, to another language, such as French, without human intervention.

### **Natural Language Generation**

This involves using natural language processing algorithms to analyze unstructured data and automatically produce content based on that data. One example of this is in

language models such as GPT3, which are able to analyze an unstructured text and then generate believable articles based on the text.

The functions listed above are used in a variety of real-world applications, including:

- Customer feedback analysis where AI analyses social media reviews.
- Customer service automation where voice assistants on the other end of a customer service phone line are able to use speech recognition to understand what the customer is saying, so that it can direct the call correctly.
- Academic research and analysis where AI is able to analyze huge amounts of academic material and research papers not just based on the metadata of the text, but the text itself.
- Analysis and categorization of medical records where AI uses insights to predict, and ideally prevent, disease.

## **Benefits of Natural Language Processing**

The main benefit of NLP is that it improves the way humans and computers communicate with each other. The most direct way to manipulate a computer is through code the computer's language. By enabling computers to understand human language, interacting with computers becomes much more intuitive for humans.

Other benefits include:

- Improved accuracy and efficiency of documentation;
- Ability to automatically make a readable summary of a larger, more complex original text.
- Useful for personal assistants such as Alexa, by enabling it to understand spoken word.
- Enables an organization to use chatbots for customer support.
- Easier to perform sentiment analysis.
- Provides advanced insights from analytics that were previously unreachable due to data volume.

## **Text Generation by using NLP**

### **What are Text Generators?**

Nowadays, there is a huge amount of data that can be categorized as sequential. It is present in the form of audio, video, text, time series, sensor data, etc. A special thing about this type of data is that if two events are occurring in a particular time frame, the occurrence of event A before event B is an entirely different scenario as compared to the occurrence of event A after event B.

Text, a stream of characters lined up one after another, is a difficult thing to crack. This is because when handling text, a model may be trained to make very accurate predictions using the sequences that have occurred previously, but one wrong prediction has the potential to make the entire sentence meaningless.

### **Different Steps of Text Generation**

Text generation usually involves the following steps:

- Importing Dependencies.
- Loading of Data.
- Creating Character/Word mappings.
- Data Pre-processing.
- Modelling
- Generating text

## **Deep Learning for NLP**

For a long time, the majority of methods used to study NLP problems employed shallow machine learning models and time-consuming, hand-crafted features. This led to problems such as the curse of dimensionality since linguistic information was represented with sparse representations (high-dimensional features).

However, with the recent popularity and success of word embeddings (low dimensional, distributed representations), neural-based models have achieved superior results on various language related tasks as compared to traditional machine learning models like SVM or logistic regression.

## **Distributed Representations**

As mentioned earlier, hand-crafted features were primarily used to model natural language tasks until neural methods came around and solved some of the problems faced by traditional machine learning models such as curse of dimensionality.

## **Convolutional Neural Network (CNN)**

A CNN is basically a neural-based approach which represents a feature function that is applied to constituting words or n-grams to extract higher-level features. The resulting abstract features have been effectively used for sentiment analysis, machine translation, and question answering, among other tasks.

Colbert and Weston were among the first researchers to apply CNN- based frameworks to NLP tasks. The goal of their method was to transform words into a vector representation via a look-up table, which resulted in a primitive word embedding approach that learn weights during the training.

In order to perform sentence modelling with a basic CNN, sentences are first tokenized into words, which are further transformed into a word embedding matrix (i.e., input embedding layer) of  $d$  dimension.

## **RNN Variants**

An LSTM consist of three gates (input, forget, and output gates), and calculate the hidden state through a combination of the three. GRUs are similar to LSTMs but consist of only two gates and are more efficient because they are less complex.

A study shows that it is difficult to say which of the gated RNNs are more effective, and they are usually picked based on the computing power available.

# **LITERATURE SURVEY**

## 2. LITERATURE SURVEY

- **S. Lai, et. al. [1]** have proposed the context based information classification; RCNN is very useful. The performance is best in several datasets, particularly on document-level datasets. Depending on the words used in the sentences, weights are assigned to it and are pooled into minimum, average and the max pools. Here, max pooling is applied to extract the keywords from the sentences which are most important. RNN, CNN and RCNN when compared with other traditional methods such as LDA, Tree Kernel and logistic regression generate highly accurate results.
- **Hassan, et. al. [9]** have proposed RNN for the structure sentence representation. This tree like structure captures the semantics of the sentences. The text is analyzed word by word by using RNN then the semantics of all the previous texts are preserved in a fixed size hidden layer. For the proposed system LSTM plays an important role, being a memory storage, it holds the characters which helps in predicting the next word.
- **J. Y. Lee, et. al. [7]** have proposed that text classification is an important task in natural language processing. Many approaches have been developed for classification such as SVM (Support Vector Machine), Naive Bayes and so on. Usually short text appears in sequence (sentences in the document) hence using information from preceding text may improve the classification.

This paper introduced RNN (Recurrent Neural Network) and CNN (Convolutional Neural Network) based models for text classification. V. Tran, et. al. [5] have proposed that n-gram is a contiguous sequence of 'n' items from a given sequence of text. If the given sentence is 'S', we can construct a list of n-grams from 'S', by finding pairs of words is used to derive the probability of sentences using the chain rule of probability.



- **Z. Shi, et. al. [4]** have defined that recurrent neural network have input, output and hidden layers. The current hidden layer is calculated by the current input layer and previous hidden layer. LSTM is a special Recurrent Neural Network. The repeating module of ordinary RNN has a simple structure; instead, LSTM uses a more complex function to replace it for more accurate results. The key element in the LSTM is the cell state which is also called the hiddenlayer state.
- **J. Shin, et. al. [10]** have defined that understanding the contextual aspects of a sentence is very important and reveals significant contributions in the field of natural language processing (NLP) and deep learning. Their work likely explores advancements.
- **W. Yin, et. al. [11]** have defined various classification tasks as important for Natural language processing applications. Nowadays CNN is increasingly used as they are able to model long range dependencies in sentences, the systems used are with fixed-sized filters. But, the proposed MVCNN approach breaks this barrier and yields better results when applied to multiple datasets: binary with 89.4%, Sentiment140 with 88.2% and Subjectivity classification dataset (Subj.) with 93.9% accuracy.

Multichannel initialization brings two advantages:

- 1) Frequent words can have c representations in the beginning (instead of only one), which means it has more available information to leverage.
- 2) A rare word missed in some embedding versions made.

- **Sutskever, et. al. [12]** have defined deep learning as being the newest technology in the era that has advanced in many fields. One of the techniques called Deep Neural Networks are very powerful machine learning models and have achieved efficient and excellent performance on many problems like speech recognition, visual object detection etc. due to its ability to perform parallel computation for the modest number of steps.

Many attempts have been made to address the problems with neural networks. The results showed that a large deep LSTM with a limited vocabulary can outperform a standard SMT-based system.

- **W. yin, et. al. [3]** have defined that deep neural network have revolutionized the field of natural language processing. Convolutional Neural Network and Recurrent Neural Network, the two main types of DNN architectures, are widely explored to handle various NLP tasks.

CNN is supposed to be good at extracting position invariant features and RNN at modelling units in sequence. RNNs are well suited to encode order information and long-range context dependency. CNNs are considered good at extracting local and position in variant features and therefore should perform well on TextC, but in experiments they are outperformed by RNNs.

- **K. C. Arnold, et. al. [6]** have proposed an approach that presents phrase suggestion instead of word predictions. It says phrases were interpreted as suggestions that affect the context of what the user. write more than then the conventional single word suggestion. The proposed system uses statistical language modelling capable of accurately predicting phrases and sentences. System used n-gram sequence model and KenLM for language model queries which used Kneser-Ney smoothing. It pruned the n grams that repeated less thantwo times in the dataset, by marking the start-of-sentence t token with some additional flags to indicate the start of the sentence.

# SYSTEM ANALYSIS

### 3.1 EXISTING SYSTEM

An n-gram model is a type of probabilistic language model for predicting the next item in such a sequence in the form of a  $(n - 1)$ —order Markov model. The N-gram model predicts the occurrence of a word based on the occurrence of its  $N - 1$  previous words.

You can think of an N-gram as the sequence of N words, by that notion, a 2-gram (or bigram) is a two-word sequence of words like "please turn", "turn your", or "your homework", and a 3-gram (or trigram) is a three-word sequence of words like "please turn your", or "turn your homework".

## This is Big Data AI Book

<b>Uni-Gram</b>	This	Is	Big	Data	AI	Book
<b>Bi-Gram</b>	This is	Is Big	Big Data	Data AI	AI Book	
<b>Tri-Gram</b>	This is Big	Is Big Data	Big Data AI	Data AI Book		

Given a sequence of  $N-1$  words, an N-gram model predicts the most probable word that might follow this sequence. It's a probabilistic model that's trained on a corpus of text. Such a model is useful in many NLP applications including speech recognition, machine translation and predictive text input.

An N-gram model is built by counting how often word sequences occur in corpus text and then estimating the probabilities. Since a simple N-gram model has limitations, improvements are often made via smoothing, interpolation, and back off.

An N-gram model is one type of a Language Model (LM), which is about finding the probability distribution over word sequences. It involves breaking down text into sequences of  $n$  contiguous words or characters, known as n-grams, and analyzing the frequency of occurrence of these n-grams in a corpus of text data. Let's understand N-

gram with an example. Consider the following sentence:

Example: "I love reading blogs about data science on Analytics Vidhya."

A 1-gram (or unigram) is a one-word sequence. For the above sentence, the unigrams would simply be: I "love", "reading", "blogs", "about", "data", "science" "on" "Analytics", "Vidhya"

A 2-gram (or bigram) is a two-word sequence of words, like "I love", "love reading", or "Analytics Vidhya".

And a 3-gram (or trigram) is a three-word sequence of words like "I love reading", "about data science" or "on Analytics Vidhya"

### Various cases of input for our n-grams model

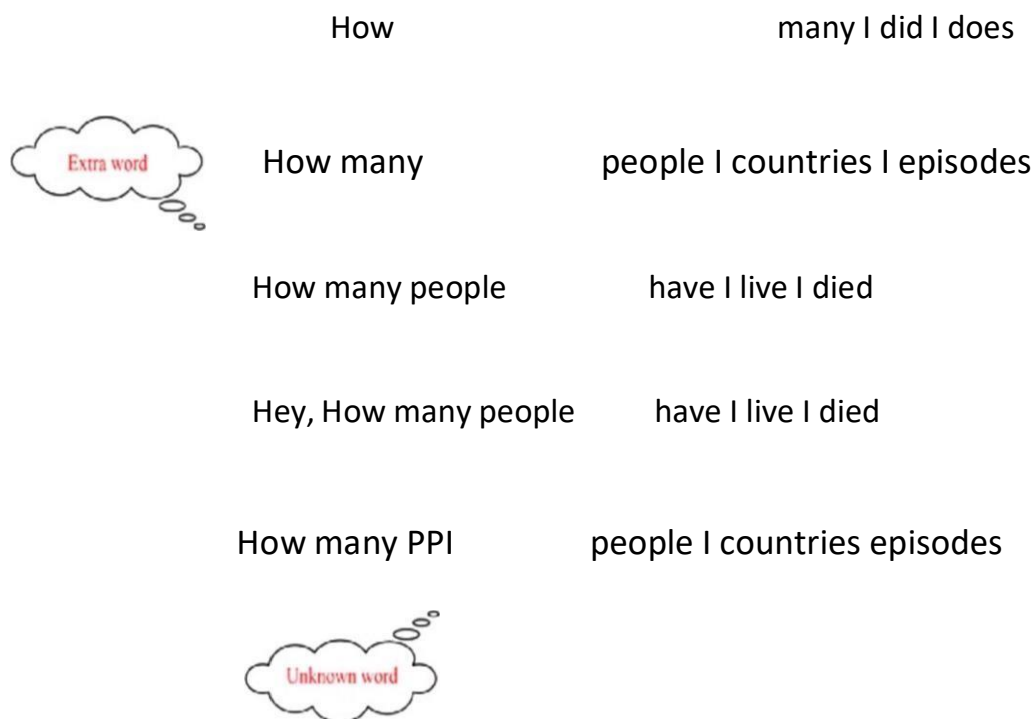


Fig: Example of N-grams Approach

## **How do we evaluate an N-gram model?**

The best way to evaluate a model is to check how well it predicts in end-to-end application testing. This approach is called extrinsic evaluation but it's time consuming and expensive. An alternative approach is to define a suitable metric and evaluate it independent of the application. This is called intrinsic evaluation. This doesn't guarantee application performance but it's a quick first step to check algorithmic performance.

## **Software tools are available to do N-gram modelling**

In Python, NLTK has the function `nltk.util.ngrams()`. A more comprehensive package is `nltk.lm`. Outside of NLTK, the `n gram` package can compute n-gram string similarity. N gram Viewer is a useful research tool by Google. It's based on material collected for Google Books. Given a sequence of words, it shows how the N-gram counts have changed over the years.

## **Applications**

In speech recognition, input may be noisy and this can lead to wrong speech-to-text conversions. N-gram models can correct this based on their knowledge of the probabilities. Likewise, N-gram models are used in machine translation to produce more natural sentences in the target language.

When correcting for spelling errors, sometimes dictionary lookups will not help. For example, in the phrase "in about fifteen minutes" the word 'minutes' is a valid dictionary word but it's incorrect in this context. N-gram models can correct such errors.

N-grams models can be used for spelling correction by suggesting corrections based on the frequency of occurrence of n-grams in a corpus of correctly spelled text. statistics, we could also classify languages or differentiate between US and UK spellings. For example, 'sz' is common in Czech; 'gb' and 'kp' are common in Igbo.

In general, many NLP applications benefit from N-gram models including part-of-speech tagging, natural language generation, word similarity, sentiment extraction and predictive text input.

### **Disadvantages**

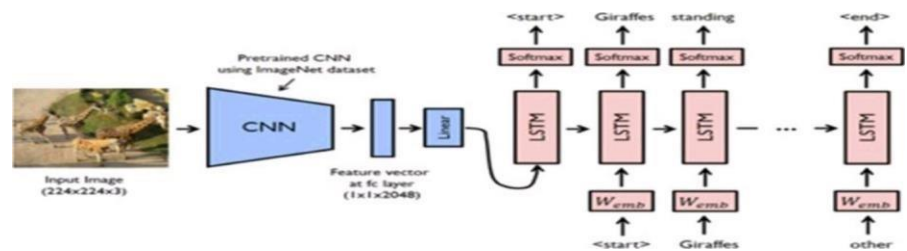
- Markov chains don't have memory as we are suggesting the next word based on frequency.
- Sparsity problem occurs with increasing values of  $n$ .
- Storage problems occurs due to large number of  $n$ -grams from large vocabulary size.
- N-grams only consider a fixed number of preceding words ( $N$ ) to predict the next word.
- It results in a shallow understanding of the context, as they fail to capture long-range dependencies and semantic relationships between words.
- N-grams treat each word sequence as a distinct entity, without considering similarities or variations in meaning.

## 3.2 PROPOSED SYSTEM

RNNs suffer from the vanishing gradient problem which makes it difficult to learn and tune the parameters in the earlier layers. In the proposed system, long short-term memory (LSTM) networks were later introduced to overcome this limitation.

An LSTM, Long Short-Term Memory, model was first introduced in the late 90s by Hoch Reiter and Schmid Huber. Since then, many advancements have been made using LSTM models and its applications are seen from areas including time series analysis to connected handwriting recognition. An LSTM network is a type of RNN which learns dependence on historic data for a sequence prediction task. What allows LSTMs to learn these historic dependencies are its feedback connections. For a common LSTM cell, we usually see an input gate, an output gate, and a forget gate. The weights of these gates control the flow of information in an LSTM model and thus are the parameters learnt during the training process.

Some variants of the LSTM model include a Convolution LSTM (or CNN- LSTM) and a Bidirectional LSTM (or Bi-LSTM). For our task, these variants correspond to different encodings of our input sequence. A CNN-LSTM model is typically used for image captioning tasks where the CNN portion is used to recognize features of the image and the LSTM is used to generate a suitable caption based on the features. Experimenting with using a CNN as the encoding layer yielded some interesting results; however, with the large number of parameters and even bigger model sizes, it was not fit for this task.





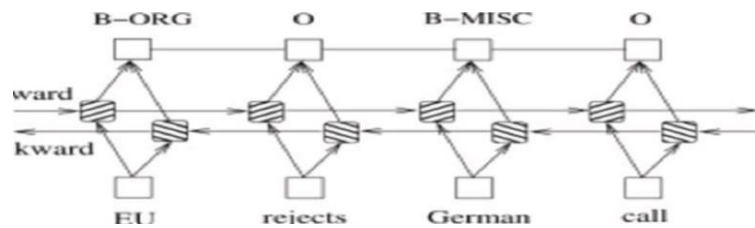


Fig 4(a): Top: CNN-LSTM, Bottom: Bi-LSTM

A more advanced approach, using a neural language model, is to use Long Short-Term Memory (LSTM). The LSTM model uses Deep learning with a network of artificial "cells" that manage memory, making them better suited for text prediction than traditional neural networks and other models.

Example: "The grass is always ..."

The next word is simply "green" and could be predicted by most models and networks. But for the sentence, "It's winter and there has been little sunlight, the grass is always ...", we need to know the context from further back in the sentence to predict the next word "brown".

Standard RNNs and other language models become less accurate when the gap between the context and the word to be predicted increases. Here's when LSTM comes in use to tackle the long-term dependency problem because it has memory cells to remember the previous context. You can learn more about LSTM networks [here](#).

Keras offers an embedding layer that can be used for neural networks on text data. The Embedding layer is initialized with random weights and learns embeddings for all of the words in the training dataset.

It requires the input data in an integer encoded form. This data preparation step can be performed with the help of the Tokenizer API also provided by Keras.

## Pre-Processing the data

For input to the Embedding layer, we first have to use a Tokenizer from keras. processing. Text to encoder input strings. What we are doing in pre-processing is simple: We first create feature dictionary sequences. Then we encode it into the integer form with the help of the Tokenizer.

How many days have passed since we last met? How are your parents?". We first clean our corpus and tokenize it with the help of Regular expressions, and word\_tokenize from the nltk library. What does the 'sequences' dictionary do? Below is the 'sequences' dictionary before using the tokenizer.{'how': 1, 'are': 2, 'you': 3, 'many': 4, 'days': 5, 'since': 6, 'we': 7, 'last': 8, 'met': 9, 'your': 10, 'parents': 11}

Our 'text sequences' list keeps all the sequences in our training corpus and it would be: [L 'how', 'are', 'you', 'how'], ['are', 'you', 'how', 'many'], ['you', 'how', 'many', 'days'], [how', 'many', 'days', 'since'], ['many', 'days', 'since', 'we'], ['days', 'since', 'we', 'last'], ['since', 'we', 'last', 'met'], ['we', 'last', 'met', 'how'], ['last', 'met', 'how', 'are'], ['met', 'how', 'are', 'your']]

After using the tokenizer, we have the above sequences in the encoded form. The numbers are nothing but the indexes of the respective words from the 'sequences' dictionary before re-assignment.

```
[[1,2, 9, 1], [2, 9, 1, 3], [9, 1, 3, 4], [1, 3, 4, 5], [3, 4, 5, 6], [4, 5, 6, 7], [5, 6, 7, 8], [6, 7, 8, 1], [7, 8, 1, 2], [8, 1, 2, 10]]
```

Once we have our sequences in encoded form, training data and target data is defined by splitting the sequences into the inputs and output labels. As for this example, we are going to predict the next word based on three previous words so in training we use the first three words as input and the last word as a label that is to be predicted by the model. Our 'training\_inputs' would now be:

```
1 [[1 0 1] [9 1 3] [1 3 4] [3 4 5] [4 5 6] [5 6 7] [6 7 8] [7 8 1]
  [8 1 2]]
```

Then, we convert our output labels into one-hot vectors i.e. into combinations of 0's and 1's.

1. One-hot vectors in 'train targets' would look like this:

```
[0,0,0,0,1,1,0,1]
```

```
[1,1,0,1,0,0,1,1,0]
```

For the first target label "how", the index was '1' in the sequence dictionary so in the encoded form you'd expect '1' at the place of index 1 in the first one-hot vector of 'train targets'.

## Building the model

Building a Long Short-Term Memory (LSTM) model involves a systematic process starting with data preprocessing and culminating in model evaluation. Initially, data must be prepared by tokenizing text into sequences of integers, potentially padding sequences to ensure uniform length. This preprocessing facilitates the conversion of raw text data into a format suitable for model training.

Now we train our Sequential model that has 5 layers: An Embedding layer, two LSTM layers, and two Dense layers. In the input layer of our model i.e., the Embedding layer, the input length is set to the size of a sequence that is 3 for this example. (Note: We split the data for training inputs and training targets as 3 to 1, so when we give input to our model for prediction, we will have to provide 3 length vectors.)

## Predicting Words

After our model is trained, we can give input in the encoded form and get the three most probable words. we use padding because we trained our model on sequences of length 3, so when we input 5 words, padding will ensure that the last three words are taken as an input to our model.

What happens when we input less than 3 words? We will not get the best results! The same happens when we input an unknown word as the one-hot vector will contain 0 in that word's index. What we can do in the future is we add sequences of length 2(inputs) to 1(target label) and 1(input) to 1(target label) as we did here 3(inputs) to 1(target label) for best results.

Below is the final output of our model predicting the next 3 words based on the previous words.

---

```
how many people
[[6 4 3]]
next word suggestion: died
next word suggestion: visit
Next word suggestion: live
```

Fig 4.3: Output of LSTM model

The above output shows the vector form of the input along with the suggested words. [6,4,3] is the 'encoded text' and [ [6, 4, 3]] is the 'pad\_encoded'.

Note: Here we split our data as 3(inputs) to 1(target label). Therefore, we must input three words.

## Phases of Proposed Method

A corpus is a collection of authentic text or audio organized into datasets. In natural language processing, a corpus contains text and speech data that can be used to train natural machine learning systems.

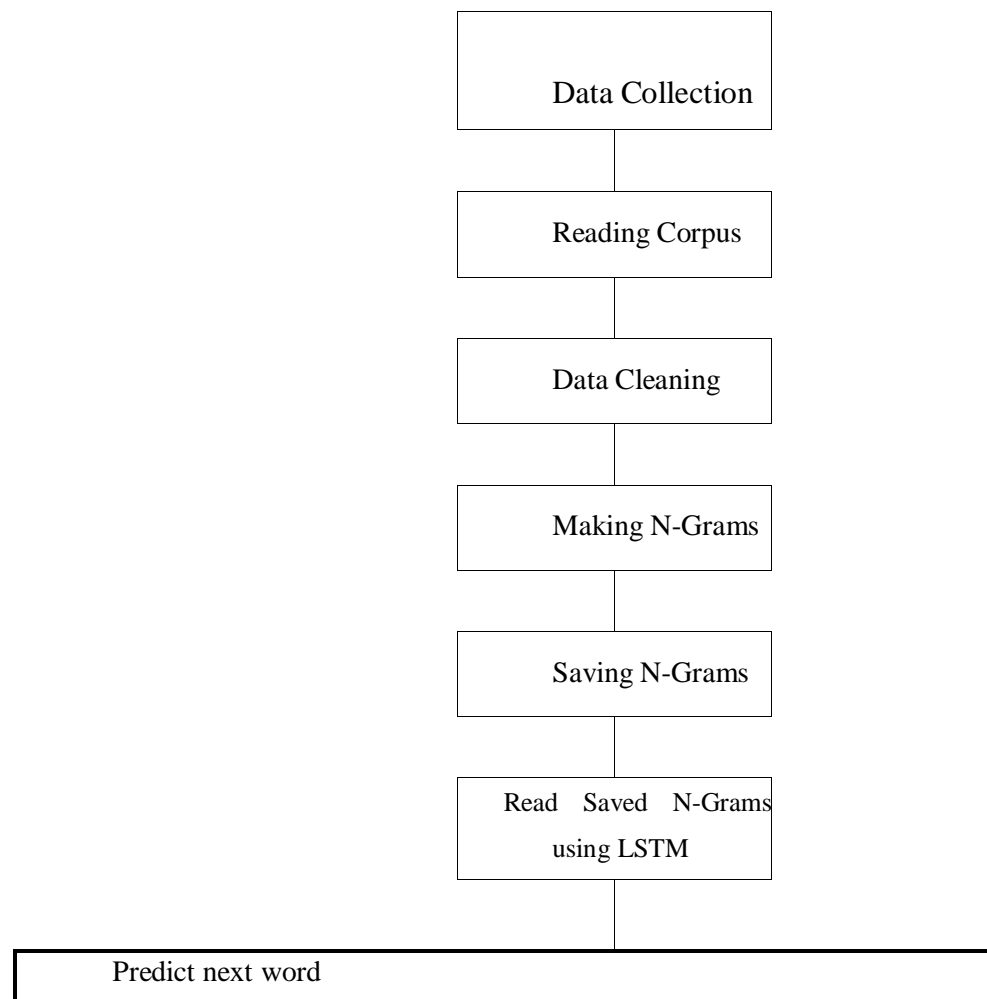


Fig 4.4: Phases of Proposed System

## Data cleaning

Removing null records, dropping unnecessary columns, treating missing values, rectifying junk values or otherwise called outliers, restructuring the data to modify it to a more readable format, is known as data cleaning.

One of the most common data cleaning examples is its application in data warehouses. A data warehouse stores a variety of data from numerous sources and optimizes it for analysis before any model fitting can be done.

## **Making N-Grams**

An N-gram means a sequence of N words. So, for example, "Medium blog" is a 2-gram (a bigram), "A Medium blog post" is a 4-gram, and "Write on Medium" is a 3-gram(trigram).

## **Saving N-Grams**

We save the data present in making N-Grams that are unigram, Bi-Gram and Trigram representations.

## **Read Saved N-Grams using LSTM**

We train our LSTM model by considering the number of LSTM classifiers and read our saved N-grams.

## PROPOSED SYSTEM ARCHITECTURE

System architecture describes. "The overall structure of the system and the ways in which the structure provides conceptual integrity'. Architecture is the hierarchy structure of a program component, the way these components interact and the structure of data that are used by the components.

The architecture for next word prediction using deep learning is designed to effectively capture the contextual nuances and dependencies within textual data. At its core, the system begins by transforming words into numerical representations through techniques like word embeddings.

These embeddings encode semantics similarities between words, providing a dense representation that preserves essential linguistic information. Additionally, incorporating supplementary features such as part-of-speech tags or word frequencies enriches the input representation, facilitating a more comprehensive understanding of the text.

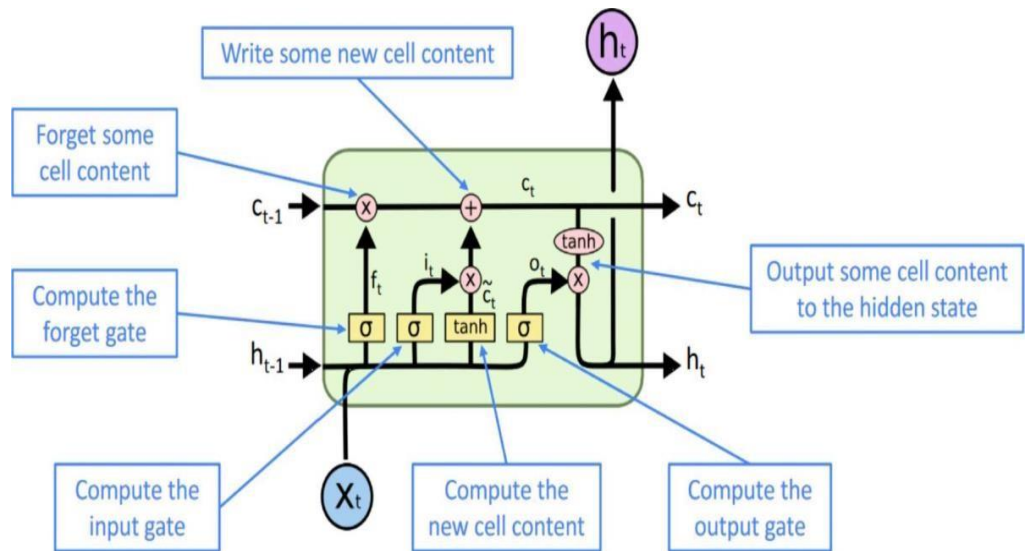


Fig : Block Diagram of Next Word Predictor Using LSTM

Step-1: First consider text corpus.

Step-2: From the text corpus Modelling will take place.

Step-3: We need to model N-Grams like uni-gram, bi-gram, tri-gram models.

Step-4: Then suggestions will be made.

Step-5: Next, we need to train our model, i.e. LSTM model. Depending upon the number of inputs, we require a number of LSTM classifiers.

Step-6: Here we require a number of LSTM classifiers.

Step-7: After training our model we will estimate the next word.

Step-8: Finally, we will get the predicted next word for the given input.

## **Stupid Backoff**

If the input text is more than 4 words or if it does not match any of the n-grams in our dataset, a "stupid backoff" algorithm will be used to predict the next word. The basic idea is it reduces the user input to n-1 gram and searches for the matching term and iterates this process until it finds the matching term.

At its core, the algorithm begins by estimating the probabilities of n-grams based on their occurrences within a training corpus. For instance, in a trigram model, it computes the probabilities associated with all observed trigrams within the training data.



## PROPOSED ALGORITHM

Algorithm: Next word Prediction Using LSTM

Input: Text Corpus data.

For 1 to P // LSTM classifiers

For 1 to N // suggesting unigram or Bigram or Trigram

1. Generating the Subset n
    - i) Generate N-grams instances from the whole training set.
    - ii) Randomly choose n.
  2. Training the individual
    - i) LSTM classifier Train up to  $p^{\text{th}}$  classifier.
  3. Making a prediction For the given Input Predict the outcome with Next word End
- Predict the Next word Based on N-gram.

## SYSTEM FRAMEWORK

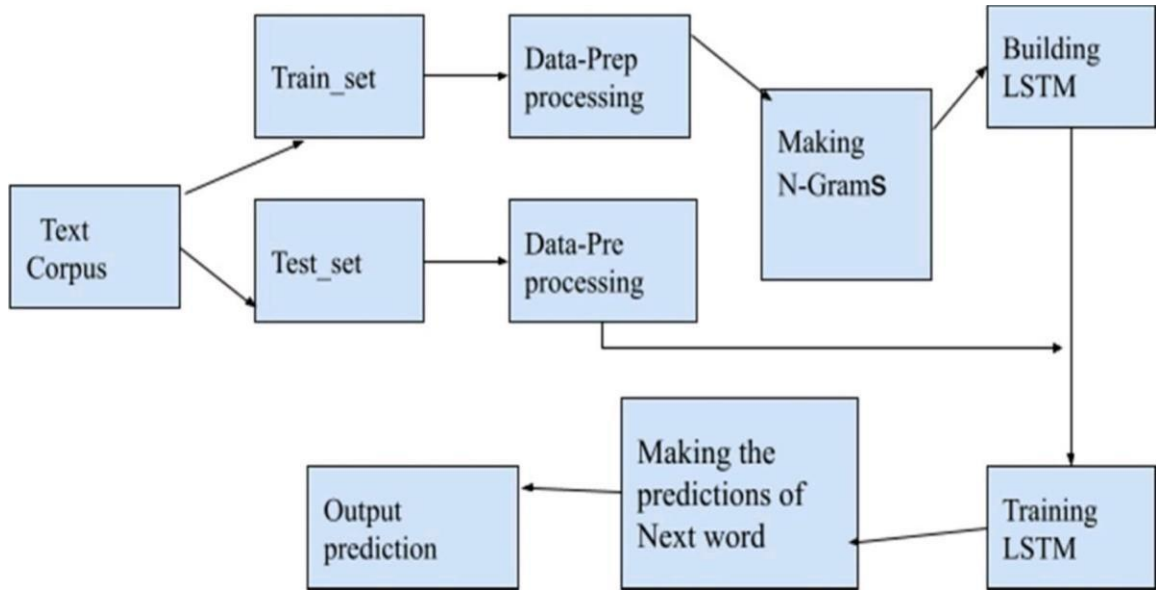


Fig : System Framework

## DATA PROCESSING

### Exploratory analysis

The input files had garbage text such as repeated letter words (e.g., "aaaa", "qqqqqqqq"). The single-word, Document Term Matrix (DTM) showed a high number of sparse terms. Although the texts are mostly in English, they contain words from other languages.

### Understanding word frequencies

It was not possible to produce a DTM for each n-gram because it kept reporting errors. For this reason, discovering bigram and trigram frequencies was also not possible. After some research, as well as trial and error attempts that were consuming too much time, the strategy was changed to create each n-gram with repetitions removed, and creating a term frequency vector for all individual words.

Increasing coverage for terms not found in the corpora. An initial (learning) idea is to have the "next word" prediction model capture the missing word (obviously typed by the user), and report it to a backend system that integrates the resulting phrase to the Corpus. At a scheduled time, the backend system will process the Corpus again, this time with the new phrase included, and update mobile devices with a new version of the n-gram model.

### **Basic N- gram model Patterns**

- FILTER PATTERN\_OI: characters not matching "a" through "z" (lower and uppercase), digits ranging from 0 to 9, the blank space, and the single quote character (').
- FILTER PATTERN\_02: text pattern starting with a single quote, followed by any number of letters ranging from "a" through "z" (lower and uppercase), and then followed by a blank space.
- FILTER PATTERN 03: text pattern matching truncated forms of the verbs 'to do', 'to be', and 'to have'.
- FILTER PATTERN 04: "stand-alone" numbers, such as 2000.
- SINGLE\_CHARACTER PATTERN: terms composed of one character, excluding a , i ", and I".

### **PROCESS**

Words are converted to lowercase. Although capitalized proper names will be lost, there are many more portions of the texts that contain unnecessary capitalized words.

All characters not forming part of FILTER\_PATTERN\_01 are removed. The blank space is included in the pattern just so that it is ignored. The single quote, usually located in between words (e.g., "don't" and "they've") is left in the texts. This way, these words can be processed later, that is, once the rest of the non-pattern characters have been removed.

All word portions matching FILTER PATTERN\_02 are removed. A first attempt at keeping words like "don't", and "you'll" was made and actually worked fine on the Corpus text files.

However, when creating the n-grams the N-Gram Tokenizer() function removed the single quote characters anyway, leaving such words as "don t", and "you ll". Thus, the resulting n-grams contained garbage text (e.g., bigrams like "you ll" and "ll see").

FILTER PATTERN\_04: Numbers are removed only when they appear as single words (e.g., 2000). Numbers embedded within alphabetical characters are left (e.g., r2d2, c3po, u2) as they might be used in certain communication contexts.

Stop words are not removed as they may be necessary to complete a given phrase. Profanity words are not removed (for now.. as resulting n-grams end up not making much sense. All single characters not forming part of SINGLE\_CHARACTER PATTERN is removed.

## **Memory Issues**

To handle memory issues and prolonged processing time (some runs took approximately 24 hrs. using the entire dataset), the aforementioned text samples were created. Additionally, data structures are deleted from memory as soon as they are no longer necessary, and the garbage collection function is executed immediately after.

## **Model to handle unseen N-Grams**

The planned strategy for the "next word" prediction model is as follows:

- If a three-word input does not yield a quad gram, the input is reduced by removing the first word, and n-gram search is resumed with a two-word input.
- If a two-word input does not yield a trigram, the input is reduced by removing the first word, and n-gram search is resumed with a one-word input.
- If a one-word input does not yield a bigram, the model returns an empty string.

## **Building LSTM Model**

We train our Sequential model that has 5 layers: An Embedding layer, two LSTM layers, and two Dense layers. In the input layer of our model i.e., Embedding layer, the input length is set to the size of a sequence that is 3 for this example. (Note: We split the data for training inputs and training targets as 3 to, so when we give input to our model for prediction, we will have to provide a 3-length vector.)

## **Predicting words**

After our model is trained, we can give input in the encoded form and get the three most probable words (if N-gram=3).

We use padding because we trained our model on sequences of length 3, so when we give input of 5 words, padding will ensure that the last 3 words are taken as input to our model. What happens when we input less than 3 words? We will not get the best results! The same happens when we input an unknown word as the one hot vector will contain 0 in that word's index.

What we can do in the future is we add sequences of length 2(inputs) to I(target label) and 1 (input) to 1 (target label) as we did here 3(inputs) to 1(target label) for best results. Below is the final output of our model prediction the next 2 words based on the previous words.

## 3.3 Feasibility Study

### 3.3.1 Technical Feasibility

**Data Availability:** Ensure that a sufficient amount of training data is available for the languages and domains you intend to support. Consider the feasibility of collecting and preprocessing the data, including any legal or ethical considerations related to data usage and privacy.

**Model Architecture:** Select a suitable deep learning architecture for next word prediction, considering factors such as model size, computational efficiency, and performance. Transformer-based architectures like GPT have shown promising results for natural language generation tasks but may require significant computational resources.

**Scalability:** Design the system to scale efficiently as the user base grows or the demand for predictions increases. Consider distributed computing techniques and load balancing strategies to handle high volumes of user requests.

**Hardware Requirements:** Deep learning models, especially large ones like transformers, can be computationally intensive to train and deploy. Assess the hardware requirements for training the model, including the need for GPUs or TPUs for faster computation. Also, consider the hardware requirements for deploying the model in real-time, especially if it will run on resource-constrained devices like mobile phones.

**Software Frameworks:** Choose appropriate deep learning frameworks such as TensorFlow, PyTorch, or JAX for developing and training the model. Consider the availability of pre-trained models and libraries for natural language processing tasks, which can accelerate development.

### **3.3.2 Operational Feasibility**

In our project, operational feasibility is favourable. Operation feasibility encompasses various aspects related to the effective integration, usability, and maintenance of the proposed next word prediction system using deep learning. Firstly, seamless integration with existing systems is paramount. Whether the system is intended for mobile keyboards or desktop word processors, its integration should be intuitive and seamless, enhancing rather than disrupting the user experience. This requires careful consideration of compatibility with popular platforms and applications, ensuring that users can easily access and utilize the prediction features without encountering technical barriers.

### **3.3.3 Economic Feasibility**

Assessing the economic feasibility of implementing a next word prediction system using deep learning involves a thorough examination of various factors. Initially, the development costs need to be carefully considered. This encompasses the expenses associated with acquiring and preprocessing training data, selecting and fine-tuning the deep learning model architecture, and developing the necessary software infrastructure. Additionally, investment in skilled personnel, such as data scientists, machine learning engineers, and software developers, contributes significantly to the initial investment.

### **3.3.4 Social Feasibility**

The social feasibility of word wise prediction using deep learning encompasses various factors that influence its acceptance and adoption within society. One aspect is its potential to enhance accessibility and inclusivity in digital communication platforms. By providing predictive text suggestions, especially for individuals with disabilities or those who face challenges in typing, such technology can promote equal participation in online conversations.

### **3.4 REQUIREMENTS SPECIFICATION**

#### **SOFTWAREE REQUIREMENTS**

- Operating System Windows 11
- Coding language Python 3.10
- IDE Jupyter Notebook, Google Colab

#### **HARDWARE REQUIREMENTS**

- System Intel or 86-64 Processor
- Hard Disk 10gb hdd (ssd recommended)
- RAM 8GB ddr 4 RAM



**DESIGN**

## **4.1 UML DIAGRAMS**

UML (Unified Modeling Language) is a general-purpose, graphical modeling language in the field of Software Engineering. UML is used to specify, visualize, construct, and document the artifacts (major elements) of the software system. It was initially developed by Grady Booch, Ivar Jacobson, and James Rumbaugh in 1994-95 at Rational software, and its further development was carried out through 1996. In 1997, it got adopted as a standard by the Object Management Group.

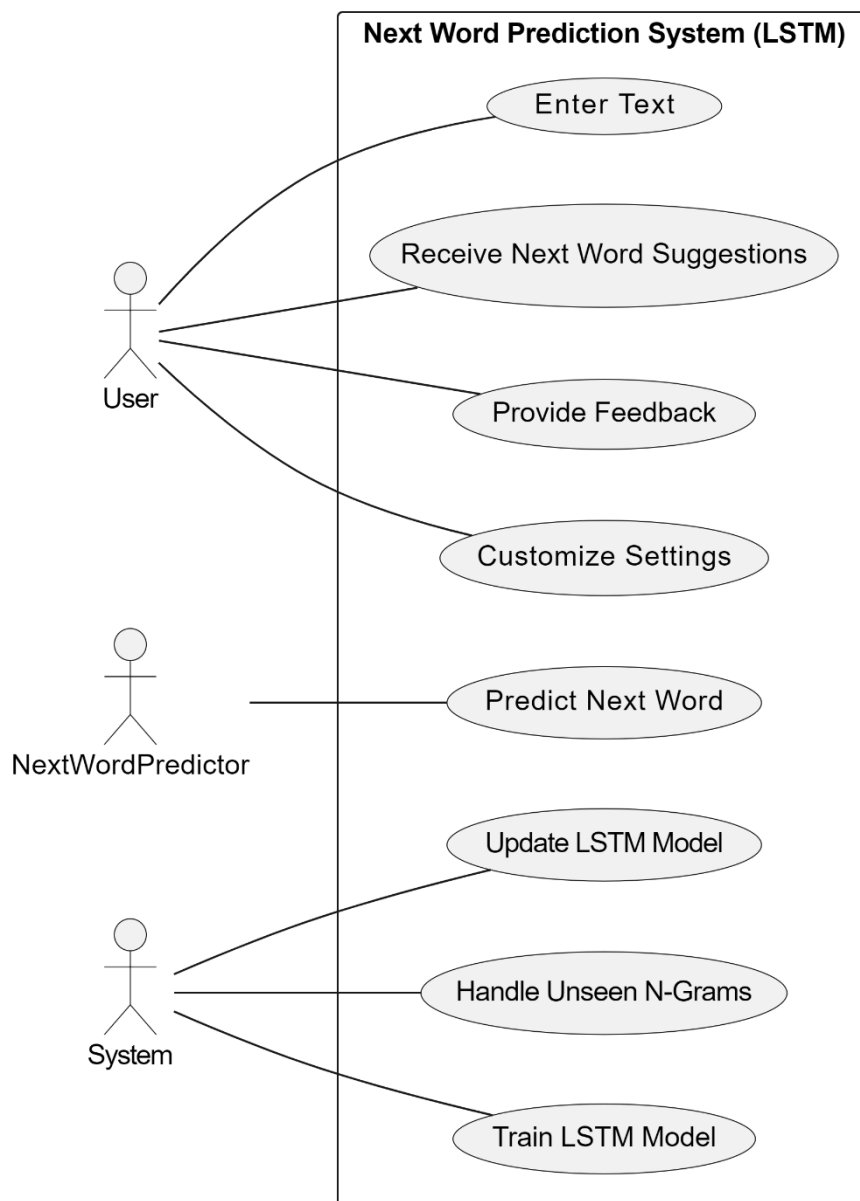
Since it is a general-purpose modelling language, it can be utilized by all the modelers. UML came into existence after the introduction of object-oriented concepts to systemize and consolidate the object-oriented development, due to the absence of standard methods at that time.

The UML diagrams are made for business users, developers, ordinary people, or anyone who is looking forward to understand the system, such that the system can be software or non-software. Thus it can be concluded that the UML is a simple modelling approach that is used to model all the practical systems.

Unified Modeling Language (UML) diagrams are graphical representations used in software engineering to visually depict different aspects of a system, its components, and their relationships. There are several types of UML diagrams, each serving a specific purpose in the software development lifecycle.

### 4.1.1 Use Case Diagram

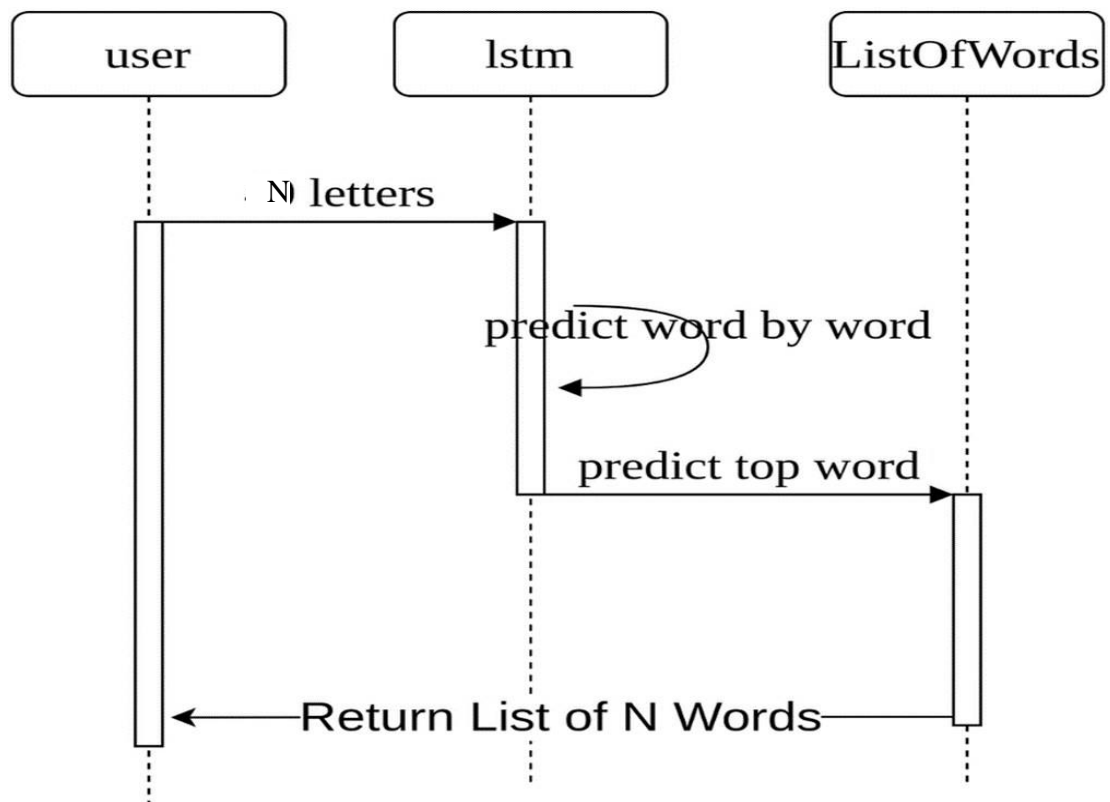
A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses. A Use Case Diagram is a vital tool in system design, it provides a visual representation of how users interact with a system. It serves as a blueprint for understanding the functional requirements.



### 4.1.2 Sequence Diagram

The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios. It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time. In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branching.

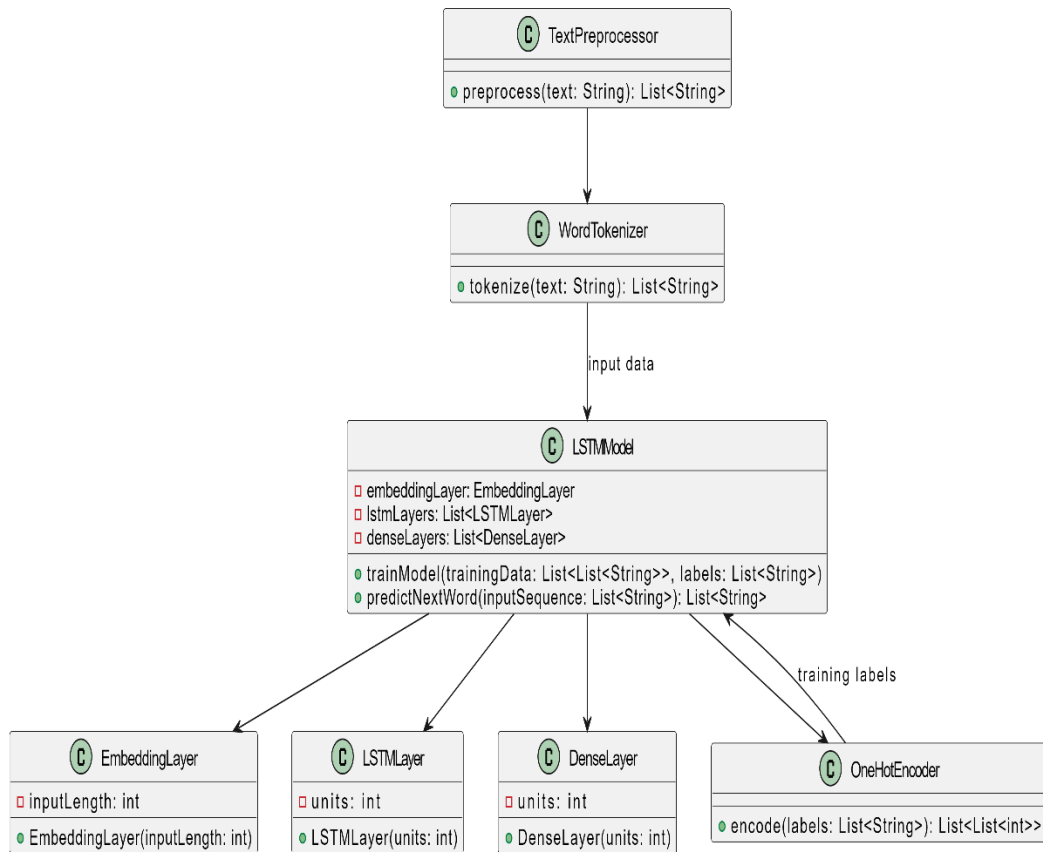
#### Sequence Diagram



### 4.1.3 Class Diagram

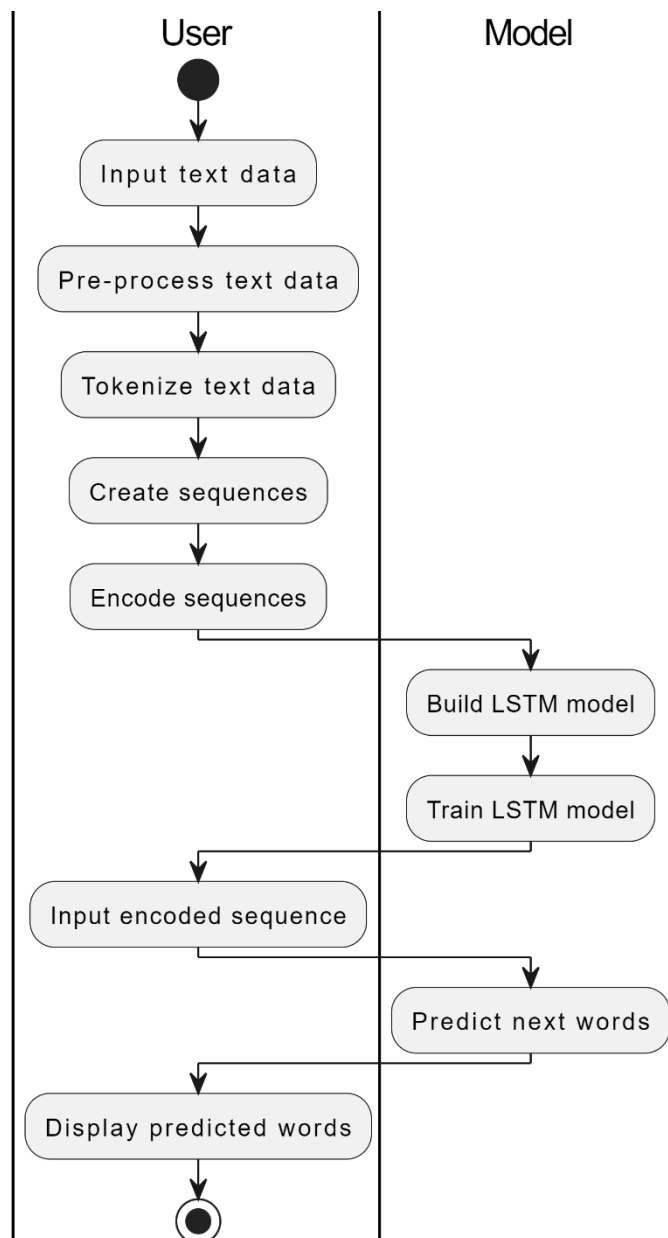
Class diagrams are a type of UML (Unified Modeling Language) diagram used in software engineering to visually represent the structure and relationships of classes in a system. UML is a standardized modeling language that helps in designing and documenting software systems. They are an integral part of the software development process, helping in both the design and documentation phases.

Class diagrams provide a high-level overview of a system's design, helping to communicate and document the structure of the software. They are a fundamental tool in object-oriented design and play a crucial role in the software development lifecycle.



#### 4.1.4 Activity Diagram

Activity Diagrams are used to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We can depict both sequential processing and concurrent processing of activities using an activity diagram i.e. an activity diagram focuses on the condition of flow and the sequence in which it happens.



# **IMPLEMENTATION**

## **5. IMPLEMENTATION**

### **Keras**

Keras is a powerful and ease-to-use free open source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code.

Keras was created to be user friendly, modular, easy to extend, and to work with Python. The API was "designed for human beings, not machines," and "follows best practices for reducing cognitive load."

Neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes are all standalone modules that you can combine to create new models. New modules are simple to add, as new classes and functions. Models are defined in Python code, not separate model configuration files.

### **Google Colab**

If you have used Jupyter notebook previously, you would quickly learn to use Google Colab. To be precise, Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook.



## **What Colab Offers You**

As a programmer, you can perform the following using Google Colab.

- Write and execute code in Python
- Document your code that supports mathematical equations
- Create/Upload/Share notebooks
- Import/Save notebooks from/to Google Drive
- Import/Publish notebooks from GitHub
- Import external datasets e.g. from Kaggle
- Integrate PyTorch, TensorFlow, Keras, OpenCV
- Free Cloud service with free GPU

## **Python**

Python is a high level general purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. It contains various libraries which are useful to implement the logic like NumPy, Keras, OpenCV, etc.. It provides easy compilation and provides efficient output.

## **TensorFlow**

TensorFlow stands as a leading open-source machine learning library, developed by Google Brain. It boasts a robust ecosystem tailored to streamline the development and deployment of machine learning models, with a particular focus on neural networks. TensorFlow's strength lies in its flexible architecture, which supports diverse hardware platforms such as CPUs, GPUs, and TPUs (Tensor Processing Units), making it adaptable to a wide array of computational environments, from desktops to large-scale distributed systems.

## 5.1 SAMPLE CODE

```
\nimport tensorflow as tf\n\n#easily convert text to sequence and vice-versa\n\nfrom tensorflow.keras.preprocessing.text import Tokenizer\nfrom tensorflow.keras.layers import Embedding, LSTM, Dense\nfrom tensorflow.keras.models import Sequential\nfrom tensorflow.keras.utils import to_categorical\nfrom tensorflow.keras.optimizers import Adam\nimport pickle\nimport numpy as np\nimport os\n\nfrom google.colab import files\nuploaded = files.upload()\n\n#opening file in read mode\nfile = open("Pride And Prejudice.txt", "r", encoding = "utf8")\n\n# store file in list\nlines = []\nfor i in file:\n    lines.append(i)
```

`

```
# Convert list to string
```

```
data = ""
```

```
for i in lines:
```

```
    data = ' '.join(lines)
```

```
#replace unnecessary stuff with space
```

```
data=data.replace('\n','').replace('\r','').replace('\u0000',  
'').replace('“',').replace('”',')
```

```
#new line, carriage return, 49nicode character →replace by space
```

```
#remove unnecessary spaces
```

```
data = data.split()
```

```
data = ' '.join(data)
```

```
data[:500]
```

```
#printing preprocessed data
```

```
len(data)
```

```
#object
```

```
tokenizer = Tokenizer()
```

```
#passing preprocessed data
```

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

```
#passing preprocessed data
```

```

# saving the tokenizer for predict function
pickle.dump(tokenizer, open('token.pkl', 'wb'))
#conversion of string/text into numeric representation

sequence_data = tokenizer.texts_to_sequences([data])[0]

#here data is more cleared
sequence_data[:15]

len(sequence_data)

# The index starts from 0

vocab_size = len(tokenizer.word_index) + 1
#printing unique words in our text corpus
print(vocab_size)

#Empty list
sequences = []

for i in range(3, len(sequence_data)):

#using 3 words of input for predicting the next word and the last 4th word will be output
    words = sequence_data[i-3:i+1]
    #appending 4 words to list
    sequences.append(words)

```

```

print("The Length of sequences are: ", len(sequences))
# list into arrays
sequences = np.array(sequences)
#printing first 10 sequences
sequences[:10]

#for input 3 words
X = []
#for output 4th word
y = []

for i in sequences:
    X.append(i[0:3])
    y.append(i[3])

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

print("Data: ", X[:10])

print("Response: ", y[:10])
#covert class vectors to binary class matrix for using loss function as categorical cross
entropy

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

```

```

#using Sequential as model
model = Sequential()
#Embedding layer
model.add(Embedding(vocab_size, 10, input_length=3))
#LSTM layer
model.add(LSTM(1000, return_sequences=True))
#Another LSTM layer
model.add(LSTM(1000))
#dense layer
model.add(Dense(1000, activation="relu"))
#another dense layer
model.add(Dense(vocab_size, activation="softmax"))

model.summary()

!pip install pydotplus
import pydotplus
from tensorflow import keras

from keras.utils import plot_model

keras.utils.plot_model(model, to_file='plot.png', show_layer_names=True)

from tensorflow.keras.callbacks import ModelCheckpoint
#used to save and evaluate weighs at some interval
checkpoint=ModelCheckpoint("next_words.h5",monitor='loss',verbose=1,
save_best_only=True)

```

```

、
model.compile(loss="categorical_crossentropy",optimizer=Adam(learning_rate=0.001)
)

#fitting model

model.fit(X, y, epochs=70, batch_size=64, callbacks=[checkpoint])


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

# Load the model and tokenizer

model=load_model('next_words.h5')
tokenizer = pickle.load(open('token.pkl', 'rb'))

def Predict_Next_Word(model, tokenizer):
    #text into sequence
    sequence = tokenizer.texts_to_sequences([text])
    sequence = np.array(sequence)#sequence into array
    preds = np.argmax(model.predict(sequence))

    #It returns max value

    predicted_word = ""

    #Iterating each items of dictionary that are in tokenizer file
    for key, value in tokenizer.word_index.items():
        if value==preds:

```

```

    predicted_word = key
    break
print(predicted_word)
return predicted_word

while(True):

    text = input("Enter your line: ")
    #Taking input from user
    if text == "0":

        print("Execution completed.....")
        break
    else:
        try:
            text = text.split(" ")
            text = text[-3:]
            print(text)
            def Predict_Next_Words(model, tokenizer, text)

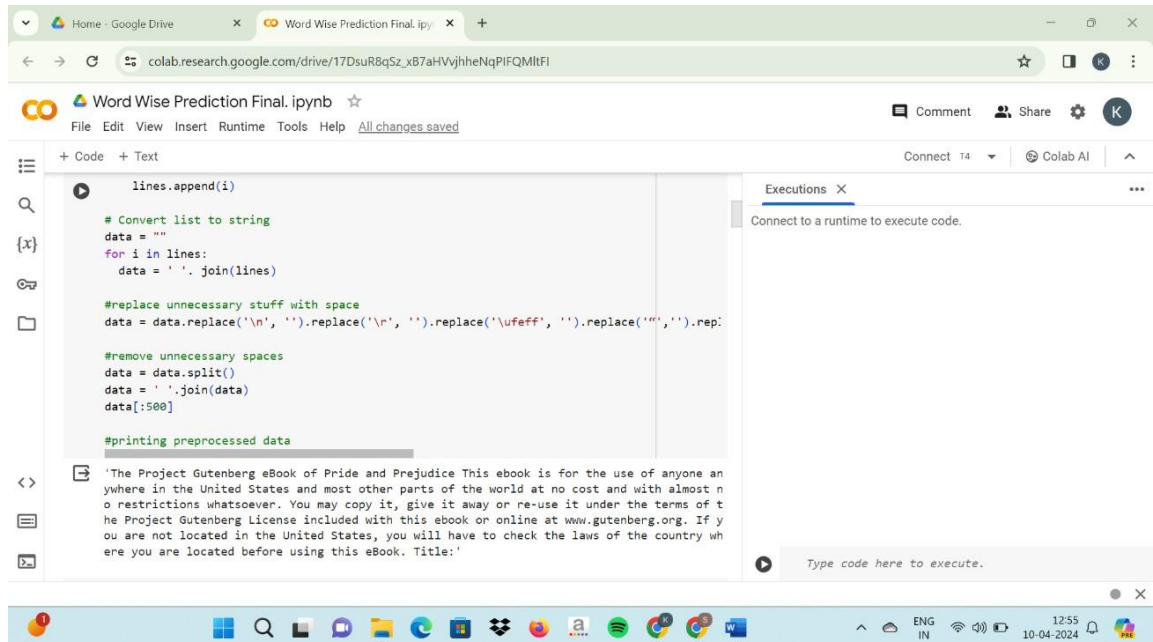
        except Exception as e:
            print("Error occurred: ",e)
        continue

```



# **OUTPUT SCREENS**

## 6. OUTPUT SCREENS



The screenshot shows a Google Colab notebook titled "Word Wise Prediction Final. ipynb". The code in the cell is as follows:

```
lines.append(i)

# Convert list to string
data = ""
for i in lines:
    data = ' '.join(lines)

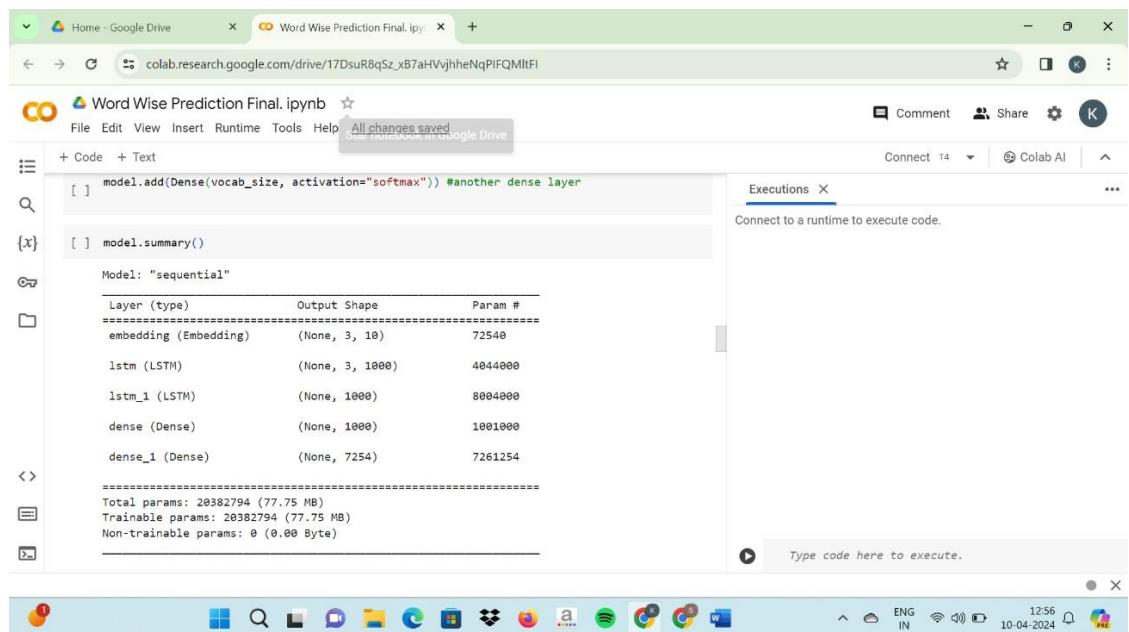
#replace unnecessary stuff with space
data = data.replace('\n', '').replace('\r', '').replace('\u00ff', '').replace(' ','').rep:

#remove unnecessary spaces
data = data.split()
data = ' '.join(data)
data[:500]

#printing preprocessed data
```

The output of the code is a text snippet from the Project Gutenberg eBook of Pride and Prejudice:

```
'The Project Gutenberg eBook of Pride and Prejudice This ebook is for the use of anyone an
ywhere in the United States and most other parts of the world at no cost and with almost n
o restrictions whatsoever. You may copy it, give it away or re-use it under the terms of t
he Project Gutenberg License included with this ebook or online at www.gutenberg.org. If y
ou are not located in the United States, you will have to check the laws of the country wh
ere you are located before using this eBook. Title:'
```



The screenshot shows the same Google Colab notebook with the following code:

```
model.add(Dense(vocab_size, activation="softmax")) #another dense layer

[ ] model.summary()
```

The output of the `model.summary()` call is a detailed summary of the model's architecture and parameters:

```
Model: "sequential"

Layer (type)                 Output Shape                 Param #
=====
embedding (Embedding)        (None, 3, 10)                72540
lstm (LSTM)                   (None, 3, 1000)              4844000
lstm_1 (LSTM)                 (None, 1000)                 8004000
dense (Dense)                 (None, 1000)                 1001000
dense_1 (Dense)               (None, 7254)                 7261254
=====
Total params: 20382794 (77.75 MB)
Trainable params: 20382794 (77.75 MB)
Non-trainable params: 0 (0.00 Byte)
```

Word Wise Prediction Final. ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[ ] !pip install pydotplus
import pydotplus
from tensorflow import keras
from keras.utils import plot_model

keras.utils.plot_model(model, to_file='plot.png', show_layer_names=True)
```

Requirement already satisfied: pydotplus in /usr/local/lib/python3.10/dist-packages (2.0.2)  
Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.10/dist-packages

```

graph TD
    embedding_input --> InputLayer
    InputLayer --> embedding
    embedding --> lstm
  
```

Executions X

Connect to a runtime to execute code.

Type code here to execute.

Word Wise Prediction Final. ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[ ]
```

```

graph TD
    lstm --> lstm_1
    lstm_1 --> dense
    dense --> dense_1
  
```

```
[ ] from tensorflow.keras.callbacks import ModelCheckpoint #used to save and elvaluate weighs
checkpoint = ModelCheckpoint("next words.h5", monitor='loss', verbose=1, save best only=True)
```

Executions X

Connect to a runtime to execute code.

Type code here to execute.

```
checkpoint = ModelCheckpoint("next_words.h5", monitor='loss', verbose=1, save_best_only=True,
                             model_compare=lambda m1, m2: m1.get('loss') < m2.get('loss'))
model.compile(loss="categorical_crossentropy", optimizer=Adam(learning_rate=0.001))
model.fit(X, y, epochs=70, batch_size=64, callbacks=[checkpoint]) #fitting model
```

Epoch 1/70  
2050/2050 [=====] - ETA: 0s - loss: 6.2199  
Epoch 1: loss improved from inf to 6.21988, saving model to next\_words.h5  
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning:   
saving\_api.save\_model(  
2050/2050 [=====] - 48s 21ms/step - loss: 6.2199  
Epoch 2/70  
2050/2050 [=====] - ETA: 0s - loss: 5.5905  
Epoch 2: loss improved from 6.21988 to 5.59052, saving model to next\_words.h5  
2050/2050 [=====] - 35s 17ms/step - loss: 5.5905  
Epoch 3/70  
2050/2050 [=====] - ETA: 0s - loss: 5.2680  
Epoch 3: loss improved from 5.59052 to 5.26800, saving model to next\_words.h5  
2050/2050 [=====] - 33s 16ms/step - loss: 5.2680  
Epoch 4/70  
2050/2050 [=====] - ETA: 0s - loss: 5.0426  
Epoch 4: loss improved from 5.26800 to 5.04262, saving model to next\_words.h5  
2050/2050 [=====] - 34s 17ms/step - loss: 5.0426  
Epoch 5/70  
2050/2050 [=====] - ETA: 0s - loss: 4.8371  
Epoch 5: loss improved from 5.04262 to 4.83706, saving model to next\_words.h5

```
2048/2050 [=====] - ETA: 0s - loss: 0.4791  
Epoch 59: loss improved from 0.48167 to 0.47909, saving model to next_words.h5  
2050/2050 [=====] - 34s 17ms/step - loss: 0.4791  
Epoch 60/70  
2049/2050 [=====] - ETA: 0s - loss: 0.4779  
Epoch 60: loss improved from 0.47909 to 0.47779, saving model to next_words.h5  
2050/2050 [=====] - 38s 19ms/step - loss: 0.4778  
Epoch 61/70  
2050/2050 [=====] - ETA: 0s - loss: 0.4709  
Epoch 61: loss improved from 0.47779 to 0.47088, saving model to next_words.h5  
2050/2050 [=====] - 35s 17ms/step - loss: 0.4709  
Epoch 62/70  
2050/2050 [=====] - ETA: 0s - loss: 0.4683  
Epoch 62: loss improved from 0.47088 to 0.46830, saving model to next_words.h5  
2050/2050 [=====] - 38s 19ms/step - loss: 0.4683  
Epoch 63/70  
2050/2050 [=====] - ETA: 0s - loss: 0.4653  
Epoch 63: loss improved from 0.46830 to 0.46534, saving model to next_words.h5  
2050/2050 [=====] - 39s 19ms/step - loss: 0.4653  
Epoch 64/70  
2050/2050 [=====] - ETA: 0s - loss: 0.4637  
Epoch 64: loss improved from 0.46534 to 0.46365, saving model to next_words.h5  
2050/2050 [=====] - 39s 19ms/step - loss: 0.4637  
Epoch 65/70  
2047/2050 [=====] - ETA: 0s - loss: 0.4610  
Epoch 65: loss improved from 0.46365 to 0.46112, saving model to next_words.h5
```

Home - Google Drive Word Wise Prediction Final. ipynb

colab.research.google.com/drive/17DsUR8qSz\_xB7aHVvjheNqPIFQMltFI#scrollTo=3OUF0KPtj4sw

Word Wise Prediction Final. ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

continue

```
Enter your line: The Project Gutenberg
['The', 'Project', 'Gutenberg']
1/1 [=====] - 1s 1s/step
literary
Enter your line: The Project Gutenberg of
['Project', 'Gutenberg', 'of']
1/1 [=====] - 0s 21ms/step
e
Enter your line: The Project Gutenberg eBook of
['Gutenberg', 'eBook', 'of']
1/1 [=====] - 0s 22ms/step
pride
Enter your line: how can you abuse your own
['abuse', 'your', 'own']
1/1 [=====] - 0s 18ms/step
children
Enter your line: He was quite
['He', 'was', 'quite']
1/1 [=====] - 0s 18ms/step
young
Enter your line: He could not help seeing that you were about five times as
['five', 'times', 'as']
1/1 [=====] - 0s 18ms/step
pretty
```

Executions X

Connect to a runtime to execute code.

Type code here to execute.

12:58 10-04-2024

Home - Google Drive Word Wise Prediction Final. ipynb

colab.research.google.com/drive/17DsUR8qSz\_xB7aHVvjheNqPIFQMltFI#scrollTo=3OUF0KPtj4sw

Word Wise Prediction Final. ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
Enter your line: The Project Gutenberg eBook of
['Gutenberg', 'eBook', 'of']
1/1 [=====] - 0s 22ms/step
pride
Enter your line: how can you abuse your own
['abuse', 'your', 'own']
1/1 [=====] - 0s 18ms/step
children
Enter your line: He was quite
['He', 'was', 'quite']
1/1 [=====] - 0s 18ms/step
young
Enter your line: He could not help seeing that you were about five times as
['five', 'times', 'as']
1/1 [=====] - 0s 18ms/step
pretty
Enter your line: and her sister
['and', 'her', 'sister']
1/1 [=====] - 0s 19ms/step
allowed
Enter your line: however, it may all come to
['all', 'come', 'to']
1/1 [=====] - 0s 27ms/step
nothing
Enter your line: 0
Execution completed.....
```

Executions X

Connect to a runtime to execute code.

Type code here to execute.

12:59 10-04-2024

# **SYSTEM TESTING**

## **7. SYSTEM TESTING**

### **7.1 TESTING OBJECTIVE**

Software testing is a process used to help identify the correctness, completeness and quality of developed computer software. Software testing is the process used to measure the quality of developed software. Testing is the process of executing a program with the intent of finding errors. Software testing is often referred to as verification & validation.

STLC (Software Testing Life Cycle):

Testing itself has many phases i.e., is called as STLC. STLC is part of SDLC

- Test Plan
- Test Development
- Test Execution
- Analyze Result
- Defect Tracking

### **TEST PLAN**

It is a document which describes the testing environment, purpose, scope, objectives, test strategy, schedules, mile stones, testing tool, roles and responsibilities, risks, training, staffing and who is going to test the application, what type of tests should be performed and how it will track the defects.

### **TEST DEVELOPMENT**

Preparing test cases, test data, Preparing test procedure, Preparing test scenario, Writing test script.

### **TEST EXECUTION**

In this phase we execute the documents those are prepared in test development phase.

## **ANALYZE RESULT**

Once executed documents will get results either pass or fail. We need to analyze the results during this phase.

## **TYPES OF TESTING**

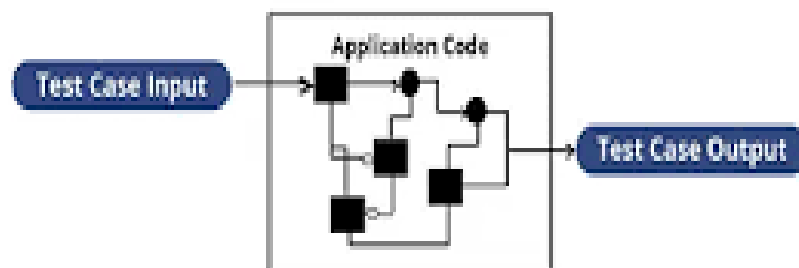
1. White Box Testing
2. Black Box Testing
3. Grey Box Testing

### **1. WHITE BOX TESTING**

White box testing as the name suggests gives the internal view of the software. This type of testing is also known as structural testing or glass box testing as well, as the interest lies in what lies inside the box.

The term 'white box' is used because of the internal perspective of the system. The clear box or white box or name denote the ability to see through the software's outer shell into its inner workings.

### **WHITE BOX TESTING APPROACH**





## 2. BLACK BOX TESTING

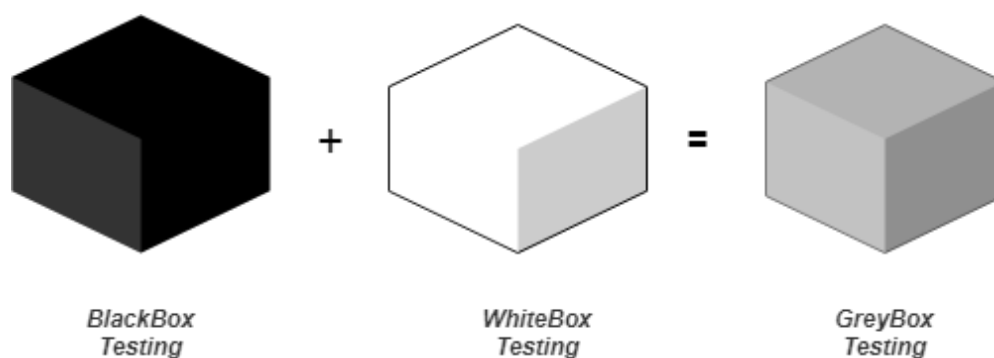
It is also called as behavioral testing. It focuses on the functional requirements of the software. Testing either functional or nonfunctional without reference to the internal structure of the component or system is called black box testing.



## 3. GREY BOX TESTING

Grey Box testing is a software testing method to test the software application with partial knowledge of the internal working structure. It is a combination of black box and white box testing because it involves access to internal coding to design test cases as white box testing and testing practices are done at functionality level as black box testing.

Grey Box testing commonly identifies context-specific errors that belong to web systems. For Instance while testing, if tester encounters any defect then he makes changes in code to resolve the defect and then test it again in real time.



# CONCLUSION

## 8. CONCLUSION

Word wise prediction powered by deep learning stands as a transformative technology, offering remarkable accuracy and fluency in anticipating the next word in a sequence of text. Through sophisticated algorithms and extensive training on vast datasets, deep learning models capture complex patterns and dependencies in language, enhancing user experience and productivity across various domains.

Furthermore, the continuous evolution of deep learning techniques and the availability of large-scale datasets have contributed to the ongoing improvement of next word prediction systems. Techniques such as transfer learning, fine-tuning, and ensemble methods enable models to leverage knowledge from pre-trained models and adapt to specific tasks or domains with minimal data requirements.

Additionally, the integration of attention mechanisms and contextual embeddings further enhances the models' ability to understand and generate contextually relevant predictions, leading to more precise and contextually coherent outcomes.

As deep learning continues to advance and researchers explore novel approaches to modeling language, the future holds great promise for further improvements in word wise prediction systems, ultimately shaping the way we interact with and leverage textual information in various applications and domains.

This Project shows Text sequence prediction can be implemented through deep learning techniques which can change the scenario of typing whole sentences. The study shows that deep learning might provide better results when compared with other techniques. Previously, machine learning was used in prediction but deep learning models produced better predictions.

# **FUTURE SCOPE**

## 9. FUTURE SCOPE

Future enhancements for word wise prediction using deep learning can be directed towards several key areas to elevate the performance and versatility of the models. Firstly, refining the models' contextual understanding capabilities stands as a crucial aspect. Leveraging advanced contextual embeddings and attention mechanisms, such as those found in transformer-based architectures, can enable the models to grasp intricate contextual dependencies more effectively, thereby refining the accuracy of predictions.

Exploring the integration of multimodal information also presents an exciting avenue for future enhancement. By incorporating visual or auditory cues alongside textual input, models can leverage complementary information from multiple modalities to enrich prediction accuracy and contextual understanding. This holistic approach can facilitate more comprehensive language modeling and prediction, especially in applications where textual information is complemented by other modalities.

Efforts towards efficient model deployment are also crucial, particularly for real-time applications or resource-constrained environments. Streamlining model architectures and deployment strategies through techniques like model quantization, pruning, and compression can ensure efficient inference without compromising prediction quality, enabling seamless integration of next word prediction capabilities into various applications and devices.

In this project, we predicted the next word with a sequence of words. In the future, we may predict the next sentence with a sequence of sentences. One more application of sequence modeling is the translation of language. In the future, we may translate the English language into our own native language and find the nextword prediction in our own native language.

# **BIBLIOGRAPHY**

## 10. BIBLIOGRAPHY

1. R. Kneser and H. Ney, "Improved backing-off for n-gram language modeling", International Conference on Acoustics Speech and Signal Processing, pp. 181-184, 1995.
2. S.F. Chen and J.T. Goodman, "An empirical study of smoothing techniques for language modeling", Computer Speech and Language, vol. 13, no. 4, pp. 359-393, 1999.
3. J. Goodman, "A bit of progress in language modeling", Microsoft Research, 2001.
4. Yoshua Bengio, Rejean Ducharme, Pascal Vincent and Christian Jauvin, "A neural probabilistic language model", Journal of Machine Learning Research, vol.
5. J. Allen, Natural Language Understanding, Benjamin/Cummings Publishing, 1995.
6. Fu-Lian Yin, Xing-Yi Pan, Xiao-Wei Liu and Hui-Xin Liu, Deep neural network language model research and application overview, 2015.
7. Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu and Charles Sutton, A survey of machine learning for big code and naturalness, 2017.
8. Mohd. Majid and Piyush Kumar, Language Modelling: Next word Prediction, 2019.
9. "The list of books "Beyond Good and Evil by Friedrich Wilhelm Nietzsche" from the official Library" in The Project Gutenberg.
10. The list of Free eBooks - Project Gutenberg books" from the official Library' The Project Gutenberg.

DOI: 10.55041/IJSREM31940



ISSN: 2582-3930

Impact Factor: 8.448

INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING & MANAGEMENT

An Open Access Scholarly Journal || Index in major Databases & Metadata

### CERTIFICATE OF PUBLICATION

International Journal of Scientific Research in Engineering & Management is hereby awarding this certificate to

**Jetti Aakanksha**

in recognition to the publication of paper titled

**WORD WISE PREDICTION USING DEEP LEARNING**

published in IJSREM Journal on Volume 08 Issue 04 April, 2024

www.ijsrem.com

  
Editor-in-Chief  
IJSREM Journal

e-mail: editor@ijsrem.com



# WORD WISE PREDICTION USING DEEP LEARNING

S. Anil Kumar<sup>1</sup>, Kakumanu Lakshmi Sireesha<sup>2</sup>, Jeti Aakanksha<sup>3</sup>,

Kothapalli Nikita<sup>4</sup>, Mummadi Sessa Sai<sup>5</sup>

<sup>1</sup> Professor, Department of Computer Science and Engineering, Tirumala Engineering College

<sup>2,3,4,5</sup> Student, Department of Computer Science and Engineering, Tirumala Engineering College

\*\*\*

**Abstract** - Word wise prediction is an input technology that simplifies the process of typing by suggesting the next word for a user to select, as typing in a conversation consumes time. You might be using it daily when you write texts or emails without realizing it.

Most of the keyboards in smart phones suggest next word prediction features. Google also uses next word prediction based on our browsing history. So, preloaded data is also stored in the keyboard function of our smart phones to predict the next word correctly.

By predicting the next word in a sequence, the number of keystrokes of the user can be reduced. In this project, we have used a deep learning approach by using Long Short Term Memory (LSTM) technique which gives better prediction accuracy than the machine learning approach.

**Key Words:** Word, technology, smart phones, keystrokes, long short term memory.

## 1. INTRODUCTION

Word prediction tools were developed which can help to communicate and also to help the people with less typing speed. The research on word prediction has been performing well. Word prediction technique does the task of guessing the preceding word that is likely to continue with few initial text fragments. Existing systems work on a word prediction model, which suggests the next immediate word based on the current available word. These systems work using machine learning algorithms which have limitations to create accurate sentence structure. Developing technologies have been producing more accurate outcomes than the existing system technologies, models developed using deep learning concepts are capable of handling more data efficiently.

This is similar to how a predictive text keyboard works on apps like What's App, Facebook Messenger, Instagram, e-mails, or even Google searches. It will consider the last word of a particular sentence and predict the next possible word. You might be using it daily when you write texts or emails without realizing it. Most of the keyboards in smartphones give next word prediction features; Google also uses next word prediction based on our browsing history. So, preloaded data is also stored in the keyboard function of our smartphones to predict the next word correctly. By predicting the next word in a sequence, the number of keystrokes of the user can be reduced. Three deep learning techniques namely Long Short Term Memory (LSTM), Bi-LSTM and BERT have been explored for the task of predicting the next word. In this project we have used the LSTM technique for predicting the next word.

## 2. LITERATURE REVIEW

- **S. Lai, et. al. [1]** have proposed the context based information classification; RCNN is very useful. The performance is best in several datasets, particularly on document-level datasets. Depending on the words used in the sentences, weights are assigned to it and are pooled into minimum, average and the max pools. Here, max pooling is applied to extract the keywords from the sentences which are most important. RNN, CNN and RCNN when compared with other traditional methods such as LDA, Tree Kernel and logistic regression generate highly accurate results.
- **Hassan, et. al. [9]** have proposed RNN for the structure sentence representation. This tree like structure captures the semantics of the sentences. The text is analyzed word by word by using RNN then the semantics of all the previous texts are preserved in a fixed size hidden layer. For the proposed system LSTM plays an important role, being a memory storage, it holds the characters which helps in predicting the next word.
- **J. Y. Lee, et. al. [7]** have proposed that text classification is an important task in natural language processing. Many approaches have been developed for classification such as SVM (Support Vector Machine), Naive Bayes and so on. Usually short text appears in sequence (sentences in the document) hence using information from preceding text may improve the classification.
- **Z. Shi, et. al. [4]** have defined that recurrent neural network have input, output and hidden layers. The current hidden layer is calculated by the current input layer and previous hidden layer. LSTM is a special Recurrent Neural Network. The repeating module of ordinary RNN has a simple structure; instead, LSTM uses a more complex function to replace it for more accurate results. The key element in the LSTM is the cell state which is also called the hidden layer state.
- **J. Shin, et. al. [10]** have defined that understanding the contextual aspects of a sentence is very important and reveals significant contributions in the field of natural language processing (NLP) and deep learning. Their work likely explores advancements.

### 3. METHODOLOGY

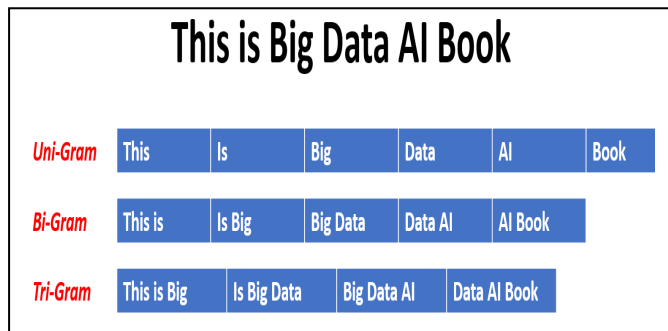
#### 3.1 EXISTING SYSTEM

An n-gram model is a type of probabilistic language model for predicting the next item in such a sequence in the form of a  $(n - 1)$ —order Markov model. The N-gram model predicts the occurrence of a word based on the occurrence of its  $N - 1$  previous words.

You can think of an N-gram as the sequence of N words, by that notion, a 2-gram (or bigram) is a two-word sequence of words like "please turn", "turn your", or "your homework", and a 3-gram (or trigram) is a three-word sequence of words like "please turn your", or "turn your homework".

#### Disadvantages

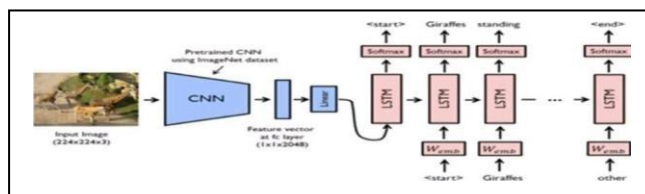
- Markov chains don't have memory as we are suggesting the next word based on frequency.
- Sparsity problem occurs with increasing values of n.
- Storage problems occur due to large number of n-grams from large vocabulary size.
- N-grams only consider a fixed number of preceding words (N) to predict the next word.



#### 3.2 PROPOSED SYSTEM

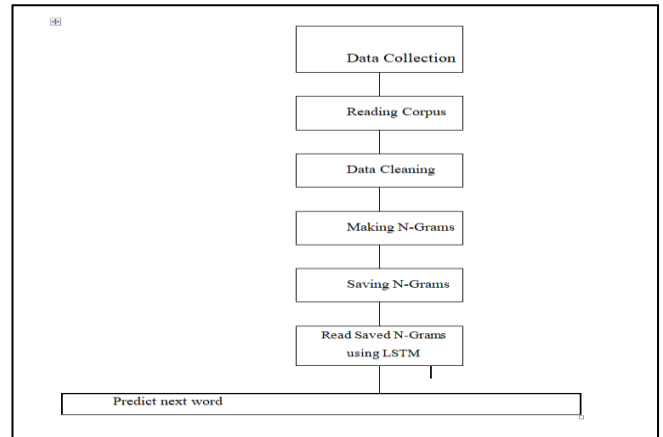
RNNs suffer from the vanishing gradient problem which makes it difficult to learn and tune the parameters in the earlier layers. In the proposed system, long short-term memory (LSTM) networks were later introduced to overcome this limitation.

An LSTM, Long Short-Term Memory, model was first introduced in the late 90s by Hoch Reiter and Schmid Huber. Since then, many advancements have been made using LSTM models and its applications are seen from areas including time series analysis to connected handwriting recognition. An LSTM network is a type of RNN which learns dependence on historic data for a sequence prediction task. What allows LSTMs to learn these historic dependencies are its feedback connections. For a common LSTM cell, we usually see an input gate, an output gate, and a forget gate. The weights of these gates control the flow of information in an LSTM model and thus are the parameters learnt during the training process.

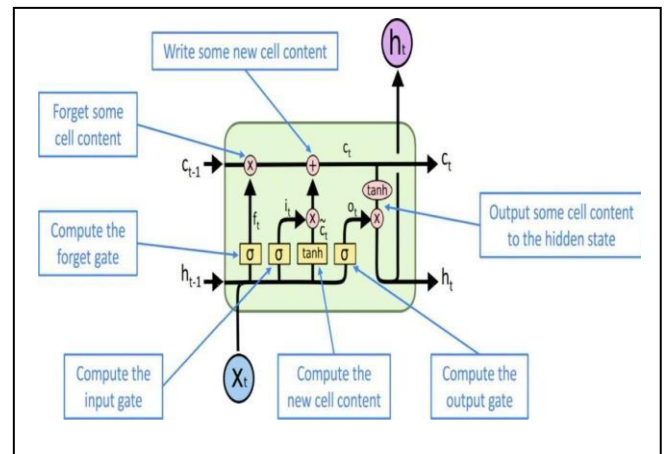


#### Phases of Proposed Method

A corpus is a collection of authentic text or audio organized into datasets. In natural Language processing, a corpus contains text and speech data that can be used to train AI machine learning systems.



#### Block diagram of Next Word Predictor Using LSTM



#### PROPOSED ALGORITHM

Algorithm: Next word Prediction Using LSTM

**Input:** Text Corpus data.

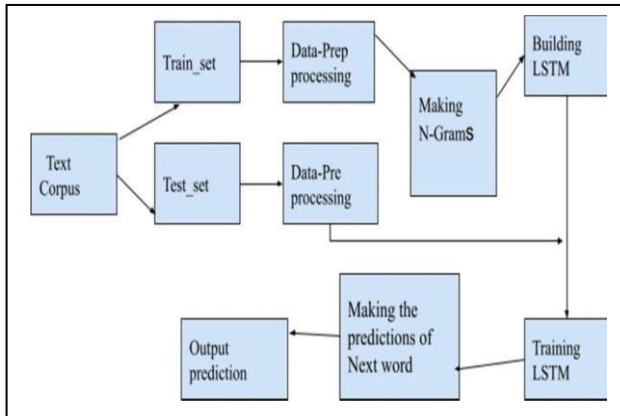
For 1 to P // LSTM classifiers

For 1 to N // suggesting unigram or Bigram or Trigram

1. Generating the Subset n
2. Generate N-grams instances from the whole training set.
3. Randomly choose n.
4. Training the individual
5. LSTM classifier Train up to  $p^{\text{th}}$  classifier.
6. Making a prediction For the given Input
7. Predict the outcome with Next word End

8. Predict the Next word Based on N-gram.

## SYSTEM FRAMEWORK



## DATA PROCESSING

### Exploratory analysis

The input files had garbage text such as repeated letter words (e.g., "aaaa", "qqqqqqqq"). The single-word, Document Term Matrix (DTM) showed a high number of sparse terms. Although the texts are mostly in English, they contain words from other languages.

### Understanding word frequencies

It was not possible to produce a DTM for each n-gram because it kept reporting errors. For this reason, discovering bigram and trigram frequencies was also not possible. After some research, as well as trial and error attempts that were consuming too much time, the strategy was changed to create each n-gram with repetitions removed, and creating a term frequency vector for all individual words.

### Basic N-gram model Patterns

- FILTER PATTERN\_OI: characters not matching "a" through "z" (lower and uppercase), digits ranging from 0 to 9, the blank space, and the single quote character (').
- FILTER PATTERN\_O2: text pattern starting with a single quote, followed by any number of letters ranging from "a" through "z" (lower and uppercase), and then followed by a blank space.
- FILTER PATTERN\_O3: text pattern matching truncated forms of the verbs 'todo', 'to be', and 'to have'.
- FILTER PATTERN\_O4: "stand-alone" numbers, such as 2000.
- SINGLE\_CHARACTER PATTERN: terms composed of one character, excluding a, i, ' and l'.

## PROCESS

Words are converted to lowercase. Although capitalized proper names will be lost, there are many more portions of the texts that contain unnecessary capitalized words.

All characters not forming part of Fiter\_Pattern\_1 are

removed. The blank space is included in the pattern just so that it is ignored. The single quote, usually located in between words (e.g., "don't" and "they've") is left in the texts. This way, these words can be processed later, that is, once the rest of the non-pattern characters have been removed.

## 4. IMPLEMENTATION

### Keras

Keras is a powerful and ease-to-use free open source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code.

Keras was created to be user friendly, modular, easy to extend, and to work with Python. The API was "designed for human beings, not machines," and "follows best practices for reducing cognitive load."

Neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes are all standalone modules that you can combine to create new models. New modules are simple to add, as new classes and functions. Models are defined in Python code, not separate model configuration files.

### Google Colab

If you have used Jupyter notebook previously, you would quickly learn to use Google Colab. To be precise, Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook.

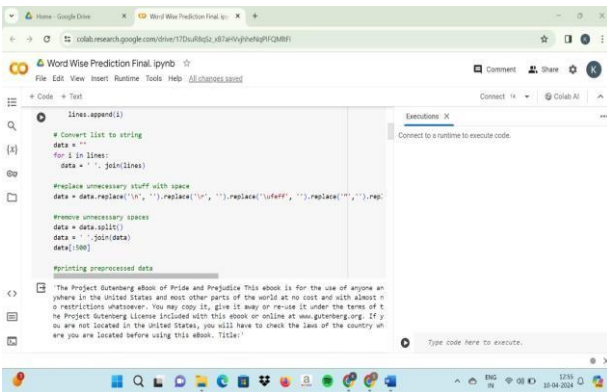
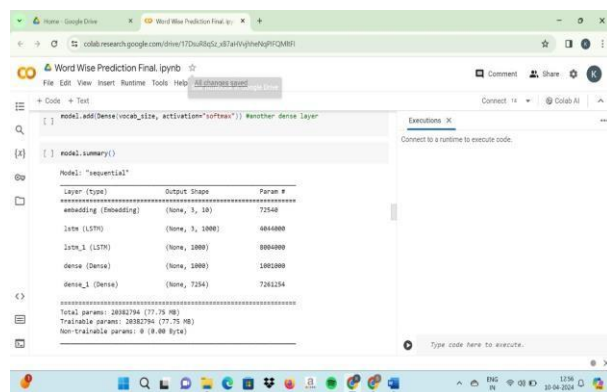
### Python

Python is a high level general purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. It contains various libraries which are useful to implement the logic like NumPy, Keras, OpenCV, etc.. It provides easy compilation and provides efficient output.

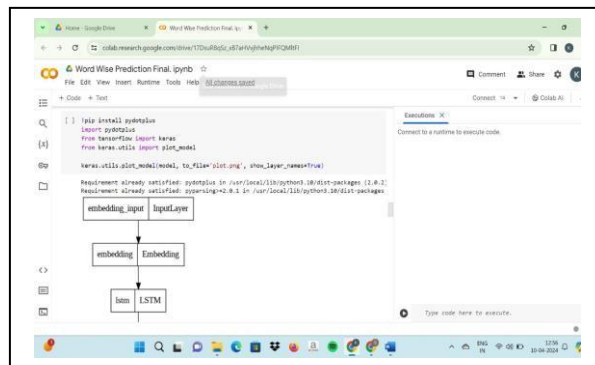
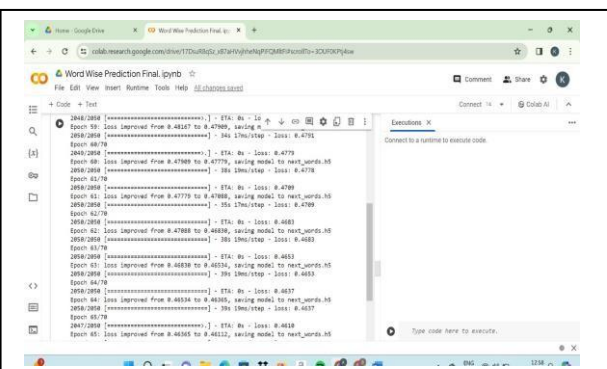
### TensorFlow

TensorFlow stands as a leading open-source machine learning library, developed by Google Brain. It boasts a robust ecosystem tailored to streamline the development and deployment of machine learning models, with a particular focus on neural networks. TensorFlow's strength lies in its flexible architecture, which supports diverse hardware platforms such as CPUs, GPUs, and TPUs (Tensor Processing Units), making it adaptable to a wide array of computational environments, from desktops to large-scale distributed systems.

## 5.RESULTS

Layer (Type)	Output Shape	Param #
embedding (Embedding)	(None, 3, 16)	72340
lstm_1 (LSTM)	(None, 3, 1000)	4004000
lstm_2 (LSTM)	(None, 1000)	8004000
dense (Dense)	(None, 1000)	1001000
dense_1 (Dense)	(None, 7234)	7261234

## 6.TESTING

Software testing is a process used to help identify the correctness, completeness and quality of developed computer software. Software testing is the process used to measure the quality of developed software. Testing is the process of executing a program with the intent of finding errors. Software testing is often referred to as verification & validation. STLC (Software Testing Life Cycle):

Testing itself has many phases i.e., is called as STLC. STLC is part of SDLC

- Test Plan
- Test Development
- Test Execution
- Analyze Result
- Defect Tracking

### TYPES OF TESTING

- 1.White Box Testing
- 2.Black Box Testing
- 3.Grey Box Testing

### WHITE BOX TESTING

White box testing as the name suggests gives the internal view of the software. This type of testing is also known as structural testing or glass box testing as well, as the interest lies in what lies inside the box.

### BLACK BOX TESTING

It is also called as behavioral testing. It focuses on the functional requirements of the software. Testing either functional or nonfunctional without reference to the internal structure of the component or system is called black box testing

### GREY BOX TESTING

Grey Box testing is a software testing method to test the software application with partial knowledge of the internal working structure. It is a combination of black box and white box testing because it involves access to internal coding to design test cases as white box testing and testing practices are done at functionality level as black box testing.



## 7. CONCLUSION

Word wise prediction powered by deep learning stands as a transformative technology, offering remarkable accuracy and fluency in anticipating the next word in a sequence of text. Through sophisticated algorithms and extensive training on vast datasets, deep learning models capture complex patterns and dependencies in language, enhancing user experience and productivity across various domains.

Furthermore, the continuous evolution of deep learning techniques and the availability of large-scale datasets have contributed to the ongoing improvement of next word prediction systems. Techniques such as transfer learning, fine-tuning, and ensemble methods enable models to leverage knowledge from pre-trained models and adapt to specific tasks or domains with minimal data requirements.

Additionally, the integration of attention mechanisms and contextual embeddings further enhances the models' ability to understand and generate contextually relevant predictions, leading to more precise and contextually coherent outcomes.

As deep learning continues to advance and researchers explore novel approaches to modeling language, the future holds great promise for further improvements in word wise prediction systems, ultimately shaping the way we interact with and leverage textual information in various applications and domains.

## 8. REFERENCES

1. R. Kneser and H. Ney, "Improved backing-off for n-gram language modeling", International Conference on Acoustics Speech and Signal Processing, pp. 181-184, 1995.
2. S.F. Chen and J.T. Goodman, "An empirical study of smoothing techniques for language modeling", Computer Speech and Language, vol. 13, no. 4, pp. 359-393, 1999.
3. J. Goodman, "A bit of progress in language modeling", Microsoft Research, 2001.
4. Yoshua Bengio, Rejean Ducharme, Pascal Vincent and Christian Jauvin, "A neural probabilistic language model", Journal of Machine Learning Research, vol.
5. J. Allen, Natural Language Understanding, Benjamin/Cummings Publishing, 1995.
6. Fu-Lian Yin, Xing-Yi Pan, Xiao-Wei Liu and Hui-Xin Liu, Deep neural network language model research and application overview, 2015.
7. Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu and Charles Sutton, A survey of machine learning for big code and naturalness, 2017.
8. Mohd. Majid and Piyush Kumar, Language Modelling: Next word Prediction, 2019.
9. "The list of books "Beyond Good and Evil by Friedrich Wilhelm Nietzsche" from the official Library" in The Project Gutenberg.
10. The list of Free eBooks - Project Gutenberg books" from the official Library' The Project Gutenberg.