

ABSTRACT

One area of natural language processing that can aid in word prediction is next word prediction, often known as language modelling. One application of machine learning is this. It had previously been considered by some researchers utilizing various models, including federated text models and recurrent neural networks. Every researcher made a forecast using their own models, including the researcher in this instance. The Long Short Term Memory (LSTM) model with 200 training epoch was chosen by the researchers to create the model. The researcher employed web scraping to gather the dataset. 180 destinations in Indonesia from nine provinces are included in the dataset. Researchers used tensor flow, Keras, NumPy, and matplotlib for the libraries.

1. INTRODUCTION

The researcher made advantage of tensor flow js to download the model in json format. A subset of machine learning called "deep learning" imitates the way the human brain absorbs information and builds patterns in it to aid in decision-making. In essence, it is an AI function that has networks that can learn from unsupervised, amorphous data. A dataset of texts is used for the succeeding word forecast. A NLP application is Next Word Prediction (Natural Language Processing). It also goes by the name of language modelling. Essentially, it involves figuring out what word will come next in a sentence. The majority of us utilise several of its programs including auto-correct, which is frequently used in emails and messages, as well as MS Word and Google Search, which predicts the next word based on our search history.

Deep learning is a subclass of machine learning, it mimics the functionality of the human brain the way it processes and creates pattern in the facts for choice making. It is basically an AI function that has networks capable of learning unsupervised data that is shapeless. The succeeding word forecast is performed on dataset consisting of texts. Next Word Prediction is an application of NLP. It is also known as Language Modelling. Basically it is the process of predicting the next word in a sentence. It has many applications which are used by most of us such as auto-correct which is mostly used in Emails / Messages; it has also its usage in MS Word google search where forecasts the next word based on our search history or the search we did for globe. In this work we have studied NLP, different deep learning techniques such as LSTM, Bi LSTM and performed a comparative study. We found satisfactory results in Bi LSTM and LSTM. The accuracy received using Bi LSTM and LSTM are: 66.1% and 58.27%.

Machine learning addresses the question of how to build computers that improve automatically through experience. Experience here can be obtained by training the machines using the model that has been built. By that training, the machine will learn through the patterns. As machine learning uses a new programming paradigm, then its concept is different from traditional programming. In traditional programming, the inputs are rules and data, then the output is the answer. While in machine learning, the inputs are answers and data, then the output is rules. This rule is what will be used later to build the model that can recognize certain types of patterns such as in deep learning. Almost every day, we use gadgets such as mobile and computer, then for sure we also

use it for typing, whether it's just browsing on google or sending messages. During those activities, we may see that it recommends the next word based on what we type. Here is what is called next word prediction, because it predicts the next word that may come after our texts. It helped us to increase writing fluency and save time. Next word prediction (NWP) is an acute problem in the arena of natural language processing. It is also called Language Modelling and here it's about mining the text. Many programmers have used it and so have the researchers. Some researchers who discussed it had written their findings as a journal and published it. Two of those journals are such as Next Words Prediction Using Recurrent Neural Networks by Saurabh Ambulgekar et al and Pre Training Federated Text Models for Next Word Prediction by Joel Streammel and Arjun Singh. Each researcher used their own models to make the prediction. So the results are different too between one to the other. But the purpose is the same as to build models. They also focused on getting good accuracy as the greater the accuracy, the better the model will predict the next word.

Deep learning models are capable enough to focus on the accurate features themselves by requiring a little guidance from the programmer and are very helpful in solving out the problem of dimensionality. Deep learning algorithms are used, especially when we have a huge no of inputs and outputs. In human brain approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousands of their neighbors.

2. LITERATURE SURVEY

2.1 Definition of Prediction:

The method used for the literature review is described in this section. In this essay, the contributions that have previously been made in the academic community are explored.

The multi-window convolution (MRNN) algorithm is used, and they have also developed the residual-connected minimum gated unit (MGU), a condensed form of the long-term support vector machine (LSTM). Using several layers of neural networks can cause latency for predicting n numbers of words. For the code completion problem, this work used the RNN method as well as GRU, another type of RNN that helps users forecast the next code syntax. The authors assert that their approach is more accurate than others currently used. Within-vocabulary words and identifier prediction make up the two halves of their next word prediction system. To make predictions inside vocabulary terms, they used the LSTM neural language model.

Writers contributed to the Bangla language. They have suggested a cutting-edge technique for word completion and prediction. They have suggested a language model based on N-grams that predicts a group of words. They got results that were good enough.

Authors have utilized LSTM to anticipate the next word in Assamese. They fed their model transcript language that had been organized using the International Phonetic Association (IPA) chart. They produced a prototype for those who are physically handicapped. This paradigm is based on Unigram, Bigram, and Trigram a strategy for Image restoration is the operation of taking a corrupt/noisy image and estimating the clean, original image. An image worth thousands of words and atmospheric turbulence affects the image quality so it is necessary to restore that degraded images. Generally, main causes of degradation are a blur, noise, and motion. The goal of the image restoration is to recover an image. The purpose is to restore a degraded image to its original content and quality due to various phenomena.

2.1.1 Vector Median-Rational Hybrid filters:

A new class of nonlinear filters called vector median-rational hybrid filters (VMRHF's) for multispectral image processing is introduced and applied to the color image filtering problem. These filters are based on rational functions (RF's) offering a number of advantages. First, a rational function is a universal approximator and a good extrapolator. Second, it can be trained by a linear adaptive algorithm. Third, it produces the best approximation (w.r.t. a given cost function) for some specific functions. The output is the result of a vector rational operation over the output of three sub-filters, such as two vector median (VM) sub-filters and one center weighted vector median filter (CWVMF). These filters exhibit desirable properties, such as edge and details preservation and accurate chromaticity estimation. Index Terms--- Color image filtering, rational functions, vector median filters, vector median-rational hybrid filters, vector rational filters.

It also proposed a novel coding algorithm based on the tree structured segmentation, which achieves oracle like rate-distortion (R-D) behavior for a simple class of signals, namely piecewise polynomials in the high bit rate regime. We consider a R-D optimization framework, which employs optimal bit allocation strategy among different signal segments to achieve the best tradeoff between description complexity and approximation quality. First, we describe the basic idea of the algorithm for the 1D case. It can be shown that the proposed compression algorithm based on an optimal binary tree segmentation achieves the oracle like R-D behavior ($(D(R)/\text{spl sim}/c/\text{sub } 0/2/\text{sup } - c1R/)$) with the computational cost of the order $O(N\log N)$. We then show the extension of the scheme to the 2D case with the similar R-D behavior without sacrificing the computational ease. Finally, we conclude with some experimental results.

Multivariate data ordering and its use in color image filtering are presented. Several of the filters presented are extensions of the single-channel filters based on order statistics. The statistical analysis of the marginal order statistics is presented for the p -dimensional case. A family of multichannel filters based on multivariate data ordering, such as the marginal median, the vector median, the marginal α -trimmed mean, and the multichannel modified trimmed mean filter, is described in detail. The performance of the marginal median and the vector median filters in impulsive noise filtering.

2.1.2 Adaptive Multichannel Filters

Here a new class of filters for multichannel image processing is introduced and analyzed. This class constitutes a generalization of vector directional filters. The proposed filters use fuzzy transformations of the angles among the different vectors to adapt to local data in the image. The principle behind the new filters is explained and comparisons with other popular nonlinear filters are provided. The specific case of color image processing is studied as an important example of multichannel image processing. Simulation results indicate that the new filters offer some flexibility and have excellent performance. New adaptive filters for color image processing are introduced and analyzed. The adaptive methodology constitutes a unifying and powerful framework for multichannel signal processing. Using this methodology, color image filtering problems are treated from a global viewpoint that readily yields and unifies previous, seemingly unrelated, results.

The new filters utilize Bayesian techniques and nonparametric methodologies to adapt to local data in the color image. The principles behind the new filters are explained in detail. Simulation studies indicate that the new filters are computationally attractive and have excellent performance.

This addresses the problem of noise attenuation for multichannel data. The proposed filter utilizes adaptively determined data-dependent coefficients based on a novel distance measure which combines vector directional with vector magnitude filtering. The special case of color image processing is studied as an important example of multichannel signal processing.

2.1.3 An Adaptive nearest Neighbour Multichannel filter

Two nonlinear algorithms for processing vector-valued signals are introduced. The algorithms, called vector median operations, are derived from two multidimensional probability density functions using the maximum-likelihood-estimate approach. The underlying probability densities are exponential, and the resulting operations have properties very similar to those of the median filter. In the vector median approach, the samples of the vector-valued input signal are processed as vectors. The operation inherently utilizes the correlation between the signal components, giving the filters

some desirable properties. General properties as well as the root signals of the vector median filters are studied. The vector median operation is combined with linear filtering, resulting in filters with improved noise attenuation and filters with very good edge response. An efficient algorithm for implementing long vector median filters is presented. The noise attenuation of the filters is discussed, and an application to velocity filtering is shown.

A new class of nonlinear filters called Vector Median Rational Hybrid Filters (VMRHF) for multispectral image processing was introduced and applied to the color image filtering problem. These filters are based on Rational Functions (RF). The VMRHF is a two-stage filter, which exploits the features of the vector median filter (VM) and those of the vector rational operator (VRF) (The output is the result of vector rational operation taking into account three sub-functions, such as two vector median sub-filters and one center weighted vector median filter (CWVMF)). These filters exhibit desirable properties, such as, edge and details preservation and accurate chromaticity estimation. The performances of the proposed filter are compared with those of the vector median and the directional-distance filters (DDF). Vector directional filters (VDF) for multichannel image processing are introduced and studied. These filters separate the processing of vector-valued signals into directional processing and magnitude processing. This provides a link between single-channel image processing where only magnitude processing is essentially performed, and multichannel image processing where both the direction and the magnitude of the image vectors play an important role in the resulting (processed) image.

VDF find applications in satellite image data processing, color image processing, and multispectral biomedical image processing. Results are presented here for the case of color images, as an important example of multichannel image processing. It is shown that VDF can achieve very good filtering results for various noise source models. In the early development of signal and image processing, linear filters were the primary tools. Their mathematical simplicity and the existence of some desirable properties made them easy to design and implement. Moreover, linear filters offered satisfactory performance in many applications. However, linear filters have poor performance in the presence of

noise that is not additive as well as in problems where system nonlinearities or non-Gaussian statistics are encountered. In addition, various criteria, such as the maximum entropy criterion, lead to nonlinear solutions. In image processing applications, linear filters tend to blur edges, do not remove impulsive noise effectively, and do not perform well in the presence of signal dependent noise. It is also known that, although the exact characteristics. For such reasons, nonlinear filtering techniques for Signal/Image processing were considered.

3. SYSTEM ANALYSIS

3.1. EXISTING SYSTEM

- One of the oldest methods used in trying to compute the probability that a given word is the next word in a sentence is using n-gram models.
- N-gram models are a type of predictive model that attempt to predict the next word in a sentence based on the (n - 1) previous words.
- These models make guesses about the probability of a word without any accompanying context.

3.2. N-Gram Language Modelling with NLTK

Language modelling is the way of determining the probability of any sequence of words. Language modelling is used in a wide variety of applications such as Speech Recognition, Spam filtering, etc. In fact, language modelling is the key aim behind the implementation of many state-of-the-art Natural Language Processing models.

Methods of Language Modelling:

Two types of Language Modelling:

1. Statistical Language Modelling: Statistical Language Modelling, or Language Modelling, is the development of probabilistic models that are able to predict the next word in the sequence given the words that precede. Examples such as N-gram language modelling.

2. Neural Language Modelling: Neural network methods are achieving better results than classical methods both on standalone language models and when models are incorporated into larger models on challenging tasks like speech recognition and machine translation. A way of performing a neural language model is through word embedding's.

N-gram can be defined as the contiguous sequence of n items from a given sample of text or speech. The items can be letters, words, or base pairs according to the application. The N-grams typically are collected from a text or speech corpus (A long text dataset).

N-gram Language Model:

An N-gram language model predicts the probability of a given N-gram within any sequence of words in the language. A good N-gram model can predict the next word in the sentence i.e the value of $p(w|h)$

Example of N-gram such as unigram (“This”, “article”, “is”, “on”, “NLP”) or bi-gram (‘This article’, ‘article is’, ‘is on’, ‘on NLP’).

Now, we will establish a relation on how to find the next word in the sentence using

We need to calculate $p(w|h)$, where h is the candidate for the next word. For example in the above example, let's consider, we want to calculate what is the probability of the last word being “NLP” given the previous words.

$p(\text{NLP} | \text{this}, \text{article}, \text{is}, \text{on})$

After generalizing the above equation can be calculated as:

$p(w_5 | w_1, w_2, w_3, w_4)$ or $P(W) = p(w_n | w_1, w_2 \dots w_n)$.

3.2 Drawbacks of Existing System

- Cannot handle the common cause of non-stationary signals and noise.
- These filters have very limited capabilities in structure adaption.
- Accuracy of the predicted words are minimal after preservation.
- While training n-grams on data can save humans time and effort, it is unclear if this approach is appropriate for all genres and sources.
- Different genres have different writing styles, and it can be difficult to combine them when writing in multiple genres.
- This model uses the probabilistic model to learn and store combinations of words dynamically that the user uses.
- This raises questions about the privacy of the user.
- The model would take enormous computing power and a much greater amount of time than the bigram model to compute

3.3 PROPOSED SYSTEM

The model is trained using deep learning algorithms and natural language processing. Deep learning is a technique that uses neural networks to imitate the way human's process information.

The model is trained using the dataset which is more general when compared with the n-gram model giving more normalized and generalized predictions.

The model uses Long short-term memory (LSTM) which has feedback connections, which can process not only single data points, but also entire sequences of data. In this work, a robust structure-adaptive hybrid vector filter (SAHVF) is proposed for color image restoration. At each pixel location, the corrupted image vector (pixel) is first classified into several different signal activity categories by noise-adaptive preprocessing and modified decomposition.

This proposed work in "Fig. 3.3" is an illustration to create flexible model that can help users to detect next word while understanding user vocal in a fast and effective manner so user need to provide 40 letters then it passes this letter to LSTM NN and predicts N number of letters. As shown in the above diagram, like in "Fig. 3.3" providing input data up to 40 letters later this sentence will pass through LSTM Neural Network.

Letter LSTM understand and learn every letter, letter by letter and create a score for the next letter.

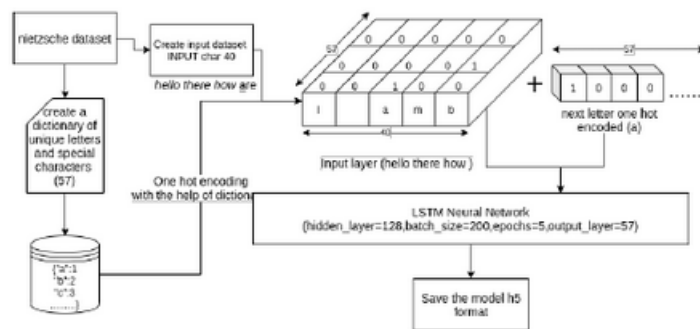


Fig 3.3: LSTM

3.4 Advantages of Proposed System

- Regardless of the genre, the model will be able to predict the most suitable words whose formation is accepted by English grammar. Even if the genre changes, the prediction remains accurate.
- The model is able to predict the words that were not included in the training data, based on Deep Learning and NLP.
- No privacy issue and requires less time for prediction when compared with n-grams model

3.5 Software Development Life Cycle Model

SDLC METHDOLOGIES

This document plays a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, “A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

- The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
- A preliminary design is created for the new system.

- A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.

A second prototype is evolved by a fourfold procedure:

Evaluating the first prototype in terms of its strengths, weakness, and risks. Defining the requirements of the second prototype. Planning a designing the second prototype. Constructing and testing the second prototype.

- At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer's judgment, result in a less-than-satisfactory final product.
- The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
- The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
- The final system is constructed, based on the refined prototype.
- The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time

Advantages

- Estimates(i.e. budget, schedule etc .) become more realistic as work progresses, because important issues discovered earlier.
- It is more able to cope with the changes that are software development generally entails.
- Software engineers can get their hands in and start working on the core of a project earlier.

3.6 Project Implementation Plan

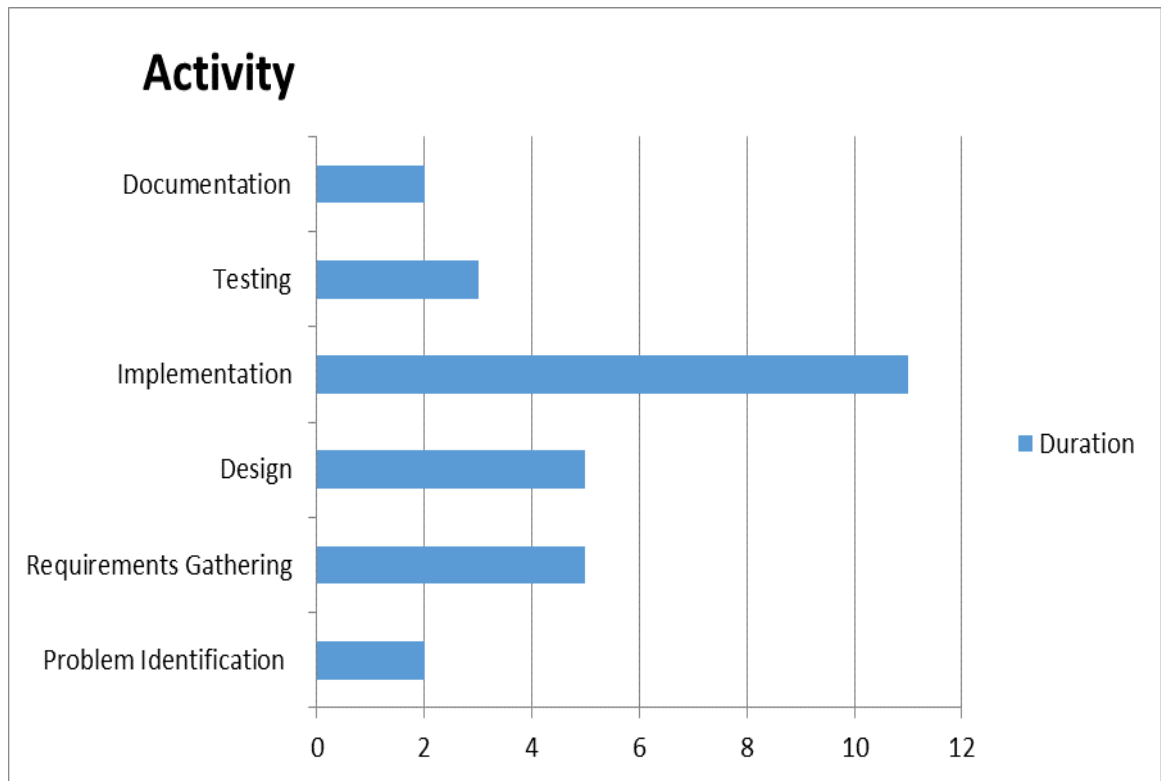


Fig 3.6 –PLAN

3.7 MODULES

LONG SHORT TERM MEMORY NETWORK

Long Short Term Memory Networks is an advanced RNN, a sequential network that allows information to persist. It is capable of handling the vanishing gradient problem faced by RNN. A recurrent neural network is also known as RNN is used for persistent memory.

Let's say while watching a video you remember the previous scene or while reading a book you know what happened in the earlier chapter. Similarly RNNs work, they remember the previous information and use it for processing the current input. The shortcoming of RNN is, they cannot remember Long term dependencies due to vanishing gradient. LSTMs are explicitly designed to avoid long-term dependency.

LSTM Architecture

At a high-level LSTM works very much like an RNN cell. Here is the internal functioning of the LSTM network. The LSTM consists of three parts, as shown in the image below and each part performs an individual function.

LSTM Architecture

The first part chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part, the cell tries to learn new information from the input to this cell. At last, in the third part, the cell passes the updated information from the current timestamp to the next timestamp. These three parts of an LSTM cell are known as gates. The first part is called Forget gate, the second part is known as the Input gate and the last one is the Output gate.

LSTM gates

Just like a simple RNN, an LSTM also has a hidden state where $H(t-1)$ represents the hidden state of the previous timestamp and H_t is the hidden state of the current timestamp. In addition to that LSTM also have a cell state represented by $C(t-1)$ and $C(t)$ for previous and current timestamp respectively.

Here the hidden state is known as Short term memory and the cell state is known as Long term memory. Refer to the following image.

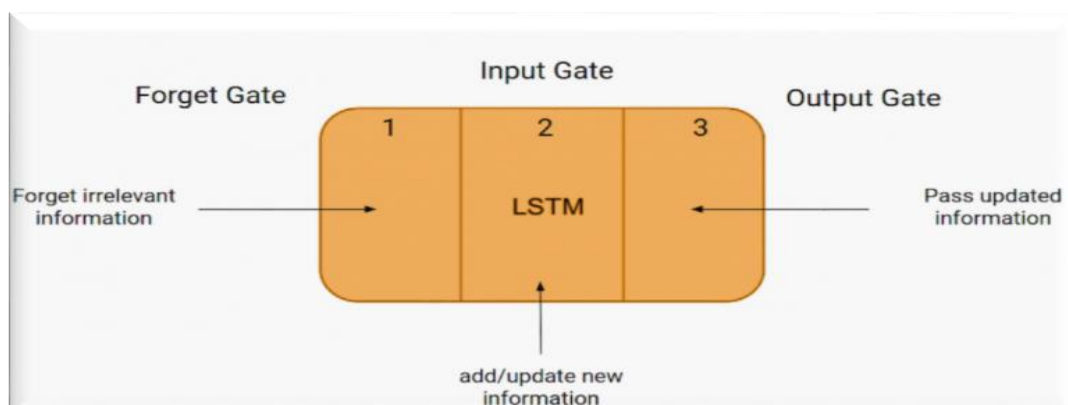


Fig 3.7.1 -Blue Print of LSTM

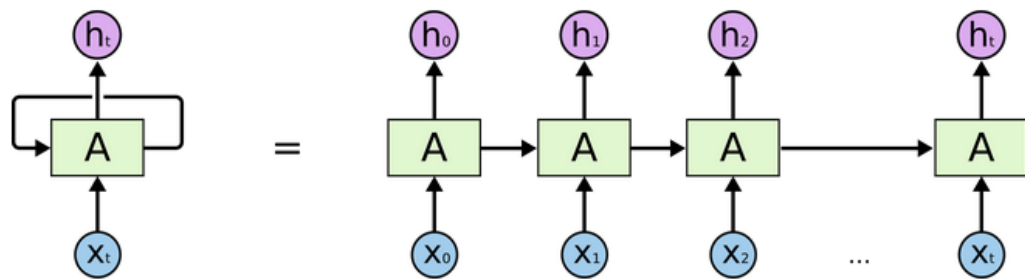
LSTM memory

It is interesting to note that the cell state carries the information along with all the timestamps.

Information along with all the timestamps

Let's take an example to understand how LSTM works. Here we have two sentences separated by a full stop. The first sentence is "Bob is a nice person" and the second sentence is "Dan, on the Other hand, is evil". It is very clear, in the first sentence we are talking about Bob and as soon as we encounter the full stop(.) we started talking about Dan.

As we move from the first sentence to the second sentence, our network should realize that we are no more talking about Bob. Now our subject is Dan. Here, the Forget gate of the network allows it to forget about it. Let's understand the roles played by these gates in LSTM architecture.



An unrolled recurrent neural network.

Fig-3.7.2 - Unrolled recurrent neural network

Forget Gate

In a cell of the LSTM network, the first step is to decide whether we should keep the information from the previous timestamp or forget it. Here is the equation for forget gate.

Forget Gate LSTM

Let's try to understand the equation, here

- x_t : input to the current timestamp.
- U_f : weight associated with the input

- H_{t-1} : The hidden state of the previous timestamp
- W_f : It is the weight matrix associated with hidden state

Later, a sigmoid function is applied over it. That will make f_t a number between 0 and 1. This f_t is later multiplied with the cell state of the previous timestamp as shown below.

Time Stamp

If f_t is 0 then the network will forget everything and if the value of f_t is 1 it will forget nothing. Let's get back to our example, The first sentence was talking about Bob and after a full stop, the network will encounter Dan, in an ideal case the network should forget about Bob.

Input Gate

Let's take another example

“Bob knows swimming. He told me over the phone that he had served the navy for four long years.”

So, in both these sentences, we are talking about Bob. However, both give different kinds of information about Bob. In the first sentence, we get the information that he knows swimming. Whereas the second sentence tells he uses the phone and served in the navy for four years.

Now just think about it, based on the context given in the first sentence, which information of the second sentence is critical. First, he used the phone to tell or he served in the navy. In this context, it doesn't matter whether he used the phone or any other medium of communication to pass on the information. The fact that he was in the navy is important information and this is something we want our model to remember. This is the task of the Input gate.

Input gate is used to quantify the importance of the new information carried by the input. Here is the equation of the input gate

LSTM Input Gate Here,

- X_t : Input at the current timestamp t
- U_i : weight matrix of input

- H_{t-1} : A hidden state at the previous timestamp
- W_i : Weight matrix of input associated with hidden state

Again we have applied sigmoid function over it. As a result, the value of I at timestamp t will be between 0 and 1.

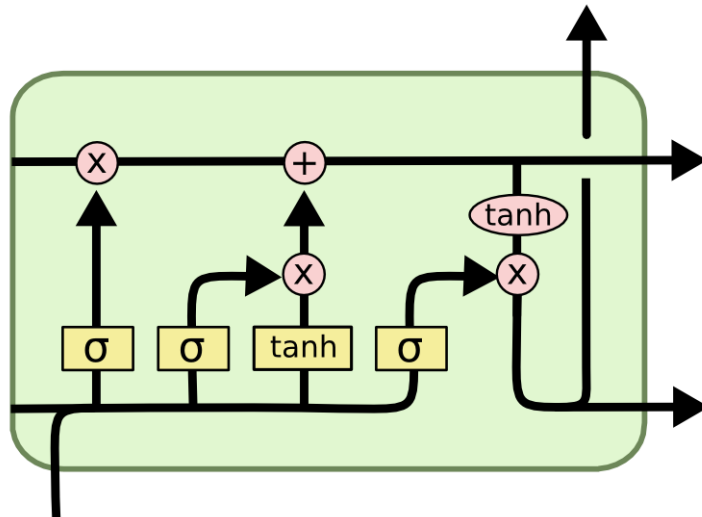


Fig 3.7.3 -LSTM Algorithm

New information LSTM

Now the new information that needed to be passed to the cell state is a function of a hidden state at the previous timestamp $t-1$ and input x at timestamp t . The activation function here is \tanh . Due to the \tanh function, the value of new information will be between -1 and 1. If the value of N_t is negative the information is subtracted from the cell state and if the value is positive the information is added to the cell state at the current timestamp.

However, the N_t won't be added directly to the cell state. Here comes the updated equation

Here, C_{t-1} is the cell state at the current timestamp and others are the values we have calculated previously.

Output Gate

Now consider this sentence

During this task, we have to complete the second sentence. Now, the minute we see the word brave, we know that we are talking about a person. In the sentence only Bob is brave, we cannot say the enemy is brave or the country is brave. So based on the current expectation we have to give a relevant word to fill in the blank. That word is our output and this is the function of our Output gate.

Output Gate

Its value will also lie between 0 and 1 because of this sigmoid function. Now to calculate the current hidden state we will use O_t and \tanh of the updated cell state. As shown below.

O_t and \tanh

It turns out that the hidden state is a function of Long term memory (C_t) and the current output. If you need to take the output of the current timestamp just apply the Soft Max activation on hidden state H_t .

Here the token with the maximum score in the output is the prediction.

This is the more intuitive diagram of the LSTM network.

LSTM network

This diagram is taken from an interesting blog. I urge you all to go through it. Here is the link-

To summarize, in this article we saw the architecture of a sequential



Fig 3.7.4-Gates of LSTM

LSTM Applications

LSTM networks find useful applications in the following areas:

- Language modeling
- Machine translation
- Handwriting recognition
- Image captioning
- Image generation using attention models
- Question answering
- Video-to-text conversion
- Polymorphic music modeling
- Speech synthesis
- Protein secondary structure prediction

Step-by-Step LSTM Walk Through

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at h_{t-1}

and x_t , and outputs a number between 0 and 1 for each number in the cell state C_{t-1} . A 1 represents "completely keep this" while a 0

Represents "completely get rid of this."

Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, \tilde{C}_t , that could be added to the state. In the next step, we'll combine these two to create an update to the state.

In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting. It's now time to update

the old cell state, C_{t-1} , into the new cell state C_t . The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t , forgetting the things we decided to forget earlier. Then we add $i_t * C_{\sim t}$. This is the new candidate values, scaled by how much we decided to update each state value. In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through \tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

Variants on Long Short Term Memory

What I've described so far is a pretty normal LSTM. But not all LSTMs are the same as the above. In fact, it seems like almost every paper involving LSTMs uses a slightly different version. The differences are minor, but it's worth mentioning some of them. One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding "peephole connections." This means that we let the gate layers look at the cell state. The above diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.

Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.

A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014). It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some

other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.

A gated recurrent unit neural network.

These are only a few of the most notable LSTM variants. There are lots of others, like Depth Gated RNNs by Yao, et al. (2015). There's also some completely different approach to tackling long-term dependencies, like Clockwork RNNs by Koutnik, et al. (2014).

5. SOFTWARE REQUIREMENT SPECIFICATIONS

5.1 Functional Requirements

- Type a sentence
- Detection
- Structure activity classification
- Text Summarization
- Tokenization
- Named Entity Recognition
- Parts of Speech Tagging
- Sentiment Analysis
- Dependency graphs
- Topic Modelling
- Aspect Mining

Type a Sentence

The user inputs a sentence from the dataset after giving the input, the dataset is pre trained and the estimation is being done, to check if the words are being corrupted/degraded by the noisy models.

Detection

An adaptive filtering methods will help to detect the Noises or the Noisy Models, which are corrupted in the sentences. A weight-adaptive vector filtering operation with an optimal window is then activated to achieve the best tradeoff between noise suppression and detail preservation.

Structure activity classification

The purpose of the structure activity classification is to partition the local structures so that a proper filtering structure and dimension can be determined for each word position to achieve the best prediction quality. The text channel (sentence) of the preprocessed input sentence, which contains most of sentence structure information, is first tokenized by a modified LSTM technique. Then, a structure activity index is calculated for each word position according to the tokenization results.

Classification is then finally applied to the activity index of each word to partition it into one of three signal activity areas, namely, the high-activity area, the medium-activity area, and the low activity area. Each of them will relate to a well-designed filtering process which provides the optimal restoration solution for the partitioned word.

Text Summarization

As the name implies, NLP approaches may be used to summarize vast amounts of text. The text summary is most commonly employed in news stories and academic papers.

Text summarization can be done in two ways:

- **Extraction:** Extraction techniques extract elements of the text to provide a summary.
- **Abstraction:** Abstraction approaches provide a summary by producing new text that expresses the essence of the original content.

For text summarization, several methods such as LexRank, TextRank, and Latent Semantic Analysis can be utilized. To use LexRank as an example, this algorithm ranks phrases based on their similarity. When a sentence is similar to many sentences, and these sentences are similar to other sentences, it is given a higher ranking.

Tokenization:

If you don't know what tokenization is, computers won't be able to comprehend or interact with us. As a result, we split the language down into tokens, which are effectively words and phrases, and then feed them into the software. Tokenization is the process of breaking down language into tokens.

Let us consider this fragment of a sentence, “NLP information extraction is fun”. This

sentence can be tokenized in the following ways, as per nanonets:

One-word (sometimes called unigram token): NLP, information, extraction, is, fun.

Two-word phrase (bigram tokens): NLP information, information extraction, extraction is, is fun, fun NLP.

Three-word sentence (trigram tokens): NLP information extraction, information extraction is, extraction is fun.

Named Entity Recognition

Extracting the entities in the text is the most fundamental and useful approach in NLP. It emphasizes the text's most important ideas and references.

It's one of the most time-consuming data preparation tasks. It entails identifying important information in a text and categorizing it into a set of predetermined categories. Named entity recognition (NER) extracts entities from text such as individuals, places, organizations, dates, and so on.

Grammar rules and supervised models are commonly used in NER. There are, however, NER systems with pre-trained and built-in NER models, such as open NLP.

Parts of Speech Tagging

When it comes to extracting information from text, tagging sections of speech is critical. It will assist us in comprehending the context of the text data. Text from documents is sometimes referred to as "unstructured data," or data with no specified structure or form.

As a result, we may employ POS tagging techniques to offer the context of words or tokens that are used to categorize them in certain ways.

All tokens in text data are classified into distinct word categories, such as nouns, verbs, adjectives, prepositions, determiners, and so on, in parts of speech tagging.

This extra information associated with words allows for additional processing and analysis, such as sentiment analytics, lemmatization, or any report that allows us to examine a specific class of words in greater detail.

Sentiment Analysis

Sentiment analysis is the most extensively used NLP approach. Sentiment analysis is especially beneficial in situations where individuals express their ideas and feedback, such as customer surveys, reviews, and social media comments.

Both supervised and unsupervised algorithms can be used for sentiment analysis. The Naive Bayes model is the most often used supervised model for sentiment analysis. It requires a sentiment-labelled training corpus, which is used to train a model, which is then used to identify the sentiment.

Different machine learning approaches such as random forest or gradient boosting can also be used instead of Naive Bayes.

Dependency graphs

A dependency graph is a data structure made up of directed graphs that represents how one thing in a system interacts with other elements in the same system. A dependency graph's underlying structure is a directed graph, in which each node points to the node on whom it depends.

Using directed graphs, dependency graphs allow us to uncover links between neighboring words. This link will provide you with information about the dependence type (e.g. Subject, Object, etc.).

A dependency network of a brief phrase is depicted in the diagram below. The arrow pointing from the term faster indicates that faster modifies going, and the label 'advmod' attached to the arrow specifies the dependency's exact nature.

Topic Modelling

Topic modelling, according to Aureus Analytics, is one of the most difficult ways for identifying natural subjects in text. Topic modelling has the benefit of being an unsupervised method. Model training and a labelled training dataset are not necessary. Topic modelling may be accomplished using a variety of approaches, including:

- Latent Semantic Analysis (LSA)
- Latent Semantic Analysis with Probabilistic Constraints (PLSA)
- Correlated Topic Model with Latent Dirichlet Allocation (LDA) (CTM).

Latent Dirichlet allocation is one of the most common approaches. The foundation of

LDA is that each text document is made up of numerous subjects, each of which is made up of several words. LDA just requires text documents and the predicted number of topics as input.

Aspect Mining

Aspect mining is a technique for identifying the many features of a text. It pulls comprehensive information from the text when used in combination with sentiment analysis. Part-of-speech tagging is one of the simplest ways of aspect mining.

5.2 Non-Functional Requirements

The major non-functional Requirements of the system are as follows

- 1. Accuracy:** The system is designed to give the accurate results so that the real time applications can use the output which is provided by the detector or the system.
- 2. Reliability:** The system is more reliable because of the qualities that are inherited from the chosen Python. The code built by using python is more reliable.
- 3. Performance:** This system is developing in the high level languages and using the advanced technologies it will give response to the end user on client system with in very less time.
- 4. Maintainability:** The system is designed to be the cross platform supportable. The system is supported on a wide range of hardware and any software platform.

5.3 Software Requirement Specifications

Operating system	:	Windows XP/7.
Coding Language	:	PYHTON/J2EE
Library	:	Tensor flow

5.4 Hardware Requirement Specifications

System	:	Intel Dual core
Hard Disk	:	40 GB.
Floppy Drive	:	44 Mb.
Monitor	:	15 VGA Color.
Mouse	:	Optical mouse
Ram	:	512 Mb

6. SYSTEM DESIGN

6.1 SYSTEM ARCHITECTURE

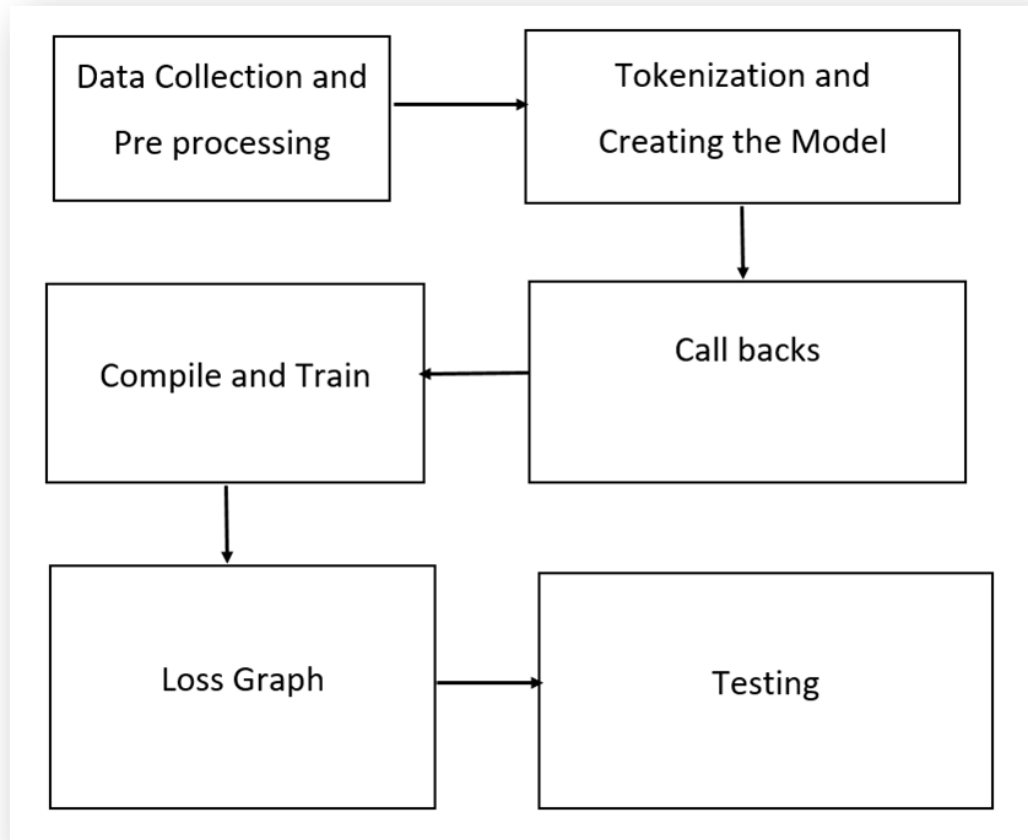


Fig 6.1 -Architecture of LSTM

6.2 UML DIAGRAMS

6.2.1 USECASE DIAGRAM :

It shows the set of use cases, actors & their relationships. In our project we have 2 actors sender and receiver & use cases shows encryption and decryption, login, generating key process.

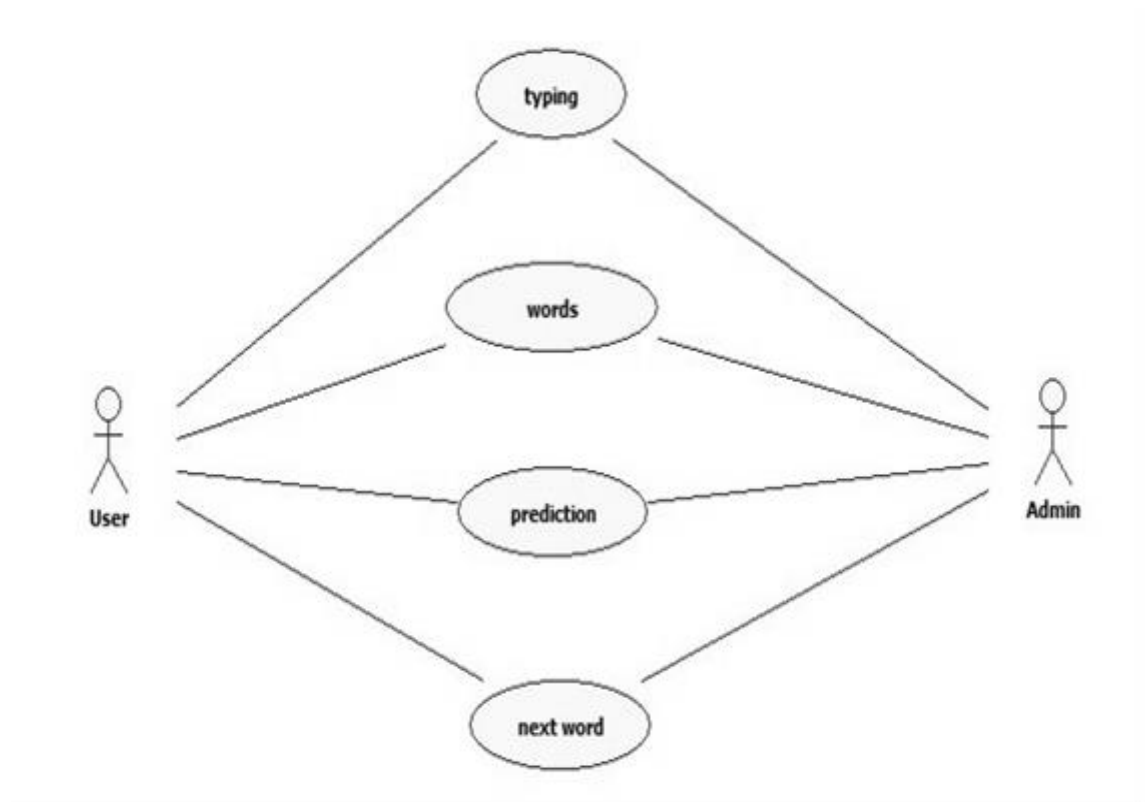


Fig 6.2.1 - Use Case Diagram

6.2.2 CLASS DIAGRAM:

A class diagram shows a set of classes, interfaces, and collaborations and their relationships. These diagrams are the most common diagram found in modeling object-oriented systems. Class diagrams address the static design view of a system. Class diagrams that include active classes address the static process view of a system.

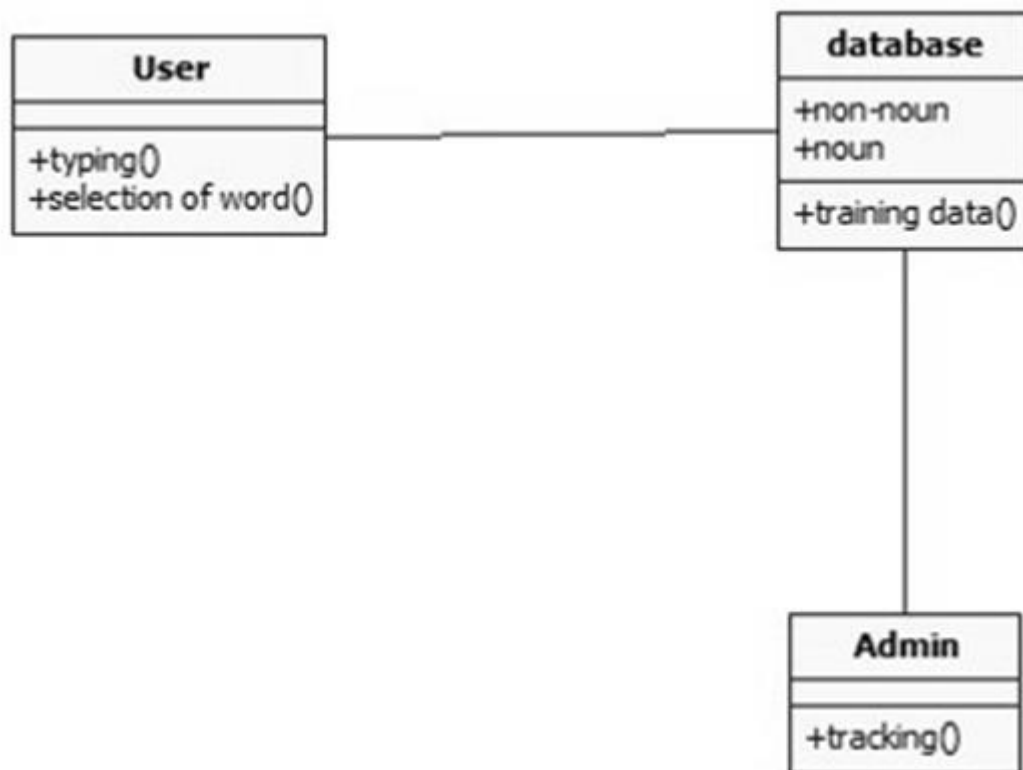


Fig 6.2.2 -Class Diagram

6.2.3 SEQUENCE DIAGRAM:

A sequence diagram is an interaction diagram that emphasizes the time-ordering of messages; a collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages. Sequence diagrams and collaboration diagrams are isomorphic, meaning that you can take one and transform it into the other.

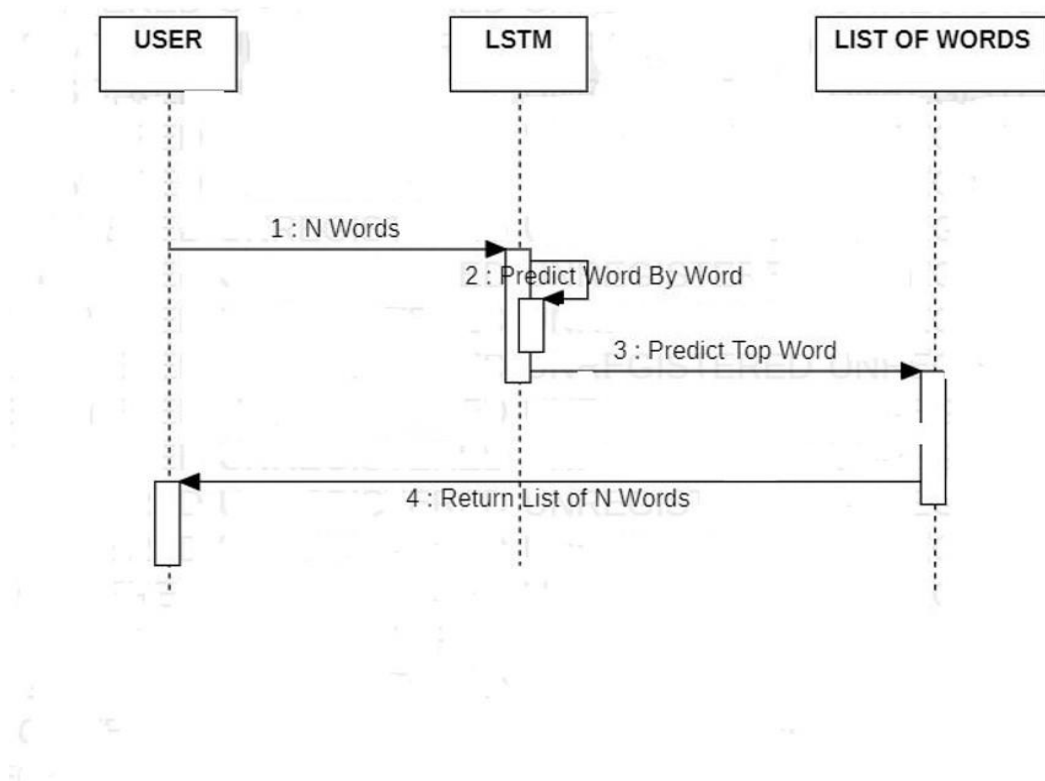


Fig 6.2.3-Sequence Diagram

6.2.4 ACTIVITY DIAGRAM:

An activity diagram is a special kind of a state chart diagram that shows the flow from activity to activity within a system. Activity diagrams address the dynamic view of a system. They are especially important in modeling the function of a system and emphasize the flow of control among objects.

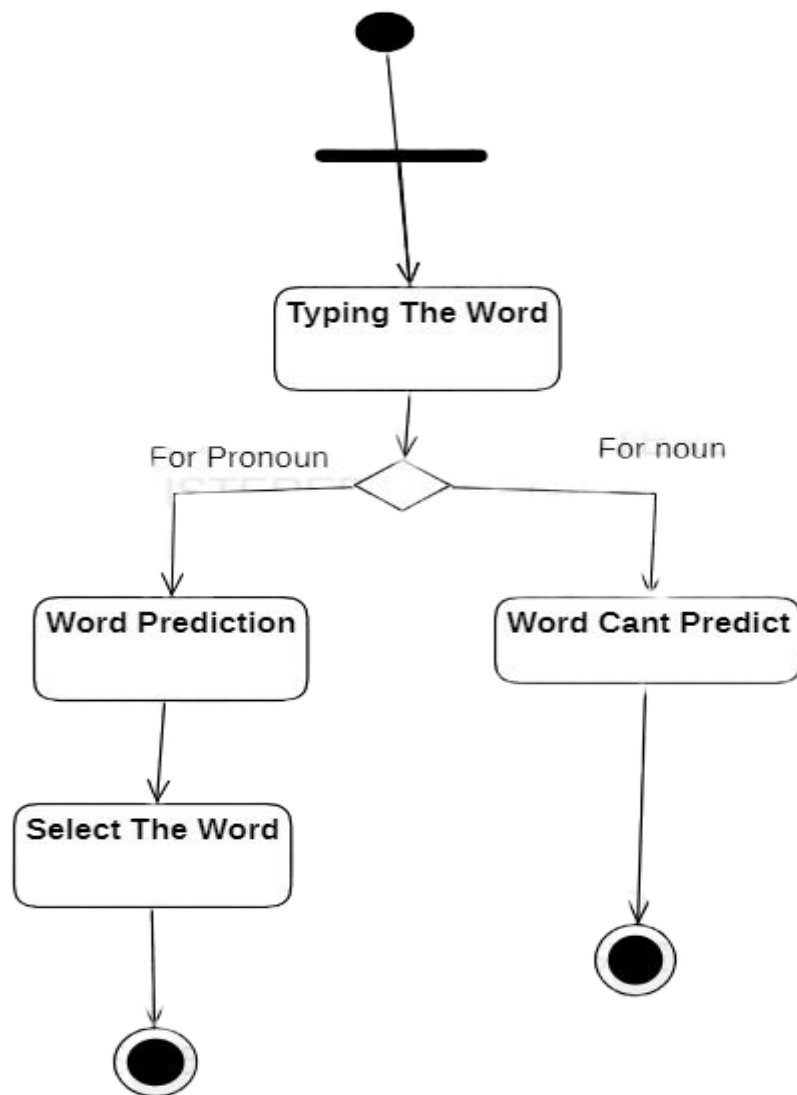


Fig 6.2.4 -Activity Diagram

7. IMPLEMENTATION

7.1 INTRODUCTION

Software Environment:

Python is a programming language. Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python Programming Language:

- **Python is Interpreted** - Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** - You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** - Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** - Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python:

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and UNIX shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features:

Python's features include

- **Easy-to-learn** - Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** - Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** - Python's source code is fairly easy-to- maintain.
- **A broad standard library** - Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** - Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** - Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** - you can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** - Python provides interfaces to all major commercial databases.
- **GUI Programming** - Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of UNIX.
- **Scalable** - Python provides a better structure and support for large programs than shell scripting.

7.2 TECHNOLOGIES USED

7.2.1 Numpy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors.

The Python programming language was not initially designed for numerical

computing, but attracted the attention of the scientific and engineering community early on, so that a special interest group called matrix-sig was founded in 1995 with the aim of defining an array computing package. Among its members was Python designer and maintainer Guido van Rossum, who implemented extensions to Python's syntax (in particular the indexing syntax) to make array computing easier.

An implementation of a matrix package was completed by Jim Fulton, then generalized by Jim Hugunin to become numeric, also variously called Numerical Python extensions or NumPy. Hugunin, a graduate student at Massachusetts Institute of Technology (MIT), joined the Corporation for National Research Initiatives (CNRI) to work on JPython in 1997 leaving Paul Dubois of Lawrence Livermore National Laboratory (LLNL) to take over as maintainer. Other early contributors include David Ascher, Konrad Hinsen and Travis Oliphant.

A new package called Numarray was written as a more flexible replacement for Numeric. Like Numeric, it is now deprecated. Numarray had faster operations for large arrays, but was slower than Numeric on small ones, so for a time both packages were used for different use cases. The last version of Numeric v24.2 was released on 11 November 2005 and numarray v1.5.2 was released on 24 August 2006.

There was a desire to get Numeric into the Python standard library, but Guido van Rossum decided that the code was not maintainable in its state then.

In early 2005, NumPy developer Travis Oliphant wanted to unify the community around a single array package and ported Numarray's features to Numeric, releasing the result as NumPy 1.0 in 2006.

This new project was part of SciPy. To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy. Support for Python 3 was added in 2011 with NumPy version 1.5.0 .

In 2011, PyPy started development on an implementation of the NumPy API for PyPy. It is not yet fully compatible with NumPy. NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted, and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging.

7.2.2 OpenCV

OpenCV (Open-Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the 30 commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces,

identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

7.2.3 Tensorflow

TensorFlow is an open-source software library for machine learning and artificial intelligence. It was developed by Google Brain and released under the Apache 2.0 license in 2015. TensorFlow is designed to be flexible and efficient, allowing developers to build and deploy machine learning models for a wide range of applications, including natural language processing, image and video recognition, and financial forecasting.

TensorFlow is built on the concept of a computation graph, which is a series of operations (called "ops") that are organized into a directed acyclic graph (DAG). This allows TensorFlow to efficiently parallelize the execution of ops across a wide range of hardware, including CPUs, GPUs, and TPUs (tensor processing units). TensorFlow also provides a set of high-level APIs that make it easy to build and train machine learning models using a variety of data sources and machine learning techniques.

One of the key features of TensorFlow is its ability to execute code across a range of platforms, including standalone machines, clusters, and cloud environments. TensorFlow also provides tools for deploying machine learning models to production, including TensorFlow Serving and TensorFlow Lite.

TensorFlow has a large and active community of developers and users, who contribute to the development of the library and share their own experiences and insights through forums, blogs, and other online resources. There are also many resources available for learning about TensorFlow, including documentation, tutorials, and courses.

In summary, TensorFlow is a powerful and flexible platform for building and deploying machine learning models. It is widely used in industry and research, and has a large and active community of developers and users.

7.2.4 Keras

Keras is a high-level, open-source software library that provides a Python interface for creating and training artificial neural networks. It was developed to enable fast experimentation with deep learning models and to make it easier to get started with this type of machine learning.

Keras is built on top of other low-level libraries such as TensorFlow and Theano, which provide the underlying functionality for training and evaluating neural networks. However, Keras abstracts away much of the complexity of these libraries, allowing users to focus on designing and training their models without having to worry about the details of the underlying algorithms.

One of the key features of Keras is its user-friendly API, which makes it easy to build and train deep learning models. It provides a set of high-level building blocks for

defining and training neural networks, such as layers, optimizers, and activation functions. These building blocks can be combined in a modular way to create complex models, and Keras includes a range of pre-trained models that can be fine-tuned for specific tasks.

Another advantage of Keras is that it is designed to be easy to extend and customize. It includes a range of hooks and callbacks that allow users to implement custom functionality, such as early stopping, model checkpointing, and learning rate scheduling. It also supports a range of advanced features such as multi-GPU training, distributed training, and model parallelism.

In addition to its user-friendliness and flexibility, Keras is also widely used in the machine learning community and has a strong ecosystem of tools and resources. It is supported by a large and active community of developers and users, and there are many tutorials, code examples, and other resources available online for learning how to use Keras.

Overall, Keras is a powerful and user-friendly toolkit for building and training deep learning models, and it is well-suited for a wide range of machine learning tasks. Its user-friendly API, extensibility, and strong community support make it a popular choice for researchers and practitioners working with neural networks.

7.3 SAMPLE CODE

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

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

for i in file:
    lines.append(i)

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

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

data = data.replace("\n", "").replace("\r", "").replace("\uffff", "")
data[:360]
import string

translator = str.maketrans(string.punctuation, '*'*len(string.punctuation))
new_data = data.translate(translator)
```

```

new_data[:500]
z = []

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

data = ' '.join(z)
data[:500]
tokenizer = Tokenizer()
tokenizer.fit_on_texts([data])

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

sequence_data = tokenizer.texts_to_sequences([data])[0]
sequence_data[:10]
vocab_size = len(tokenizer.word_index) + 1
print(vocab_size)
sequences = []

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

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

for i in sequences:

```

```

X.append(i[0])
y.append(i[1])
X = np.array(X)
y = np.array(y)
print("The Data is: ", X[:5])
print("The responses are: ", y[:5])
y = to_categorical(y, num_classes=vocab_size)
y[:5]

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

pip install keras

from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import TensorBoard

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

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

logdir='logsnextword1'
tensorboard_Visualization = TensorBoard(log_dir=logdir)
model.compile(loss="categorical_crossentropy", optimizer=Adam(lr=0.001))
model.fit(X, y, epochs=150, batch_size=64, callbacks=[checkpoint, reduce,
    tensorboard_Visualization])

%tensorboard --logdir log

```

8. SYSTEM TESTING

8.1 TESTING PLAN

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirement analysis where the information domain, functions, behavior, performance, constraints and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software we spiral in along streamlines that decrease the level of abstraction on each turn.

A strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Testing progress by moving outward along the spiral to integration testing, where the focus is on the design and the construction of the software architecture. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally we arrive at system testing, where the software and other system elements are tested as a whole.

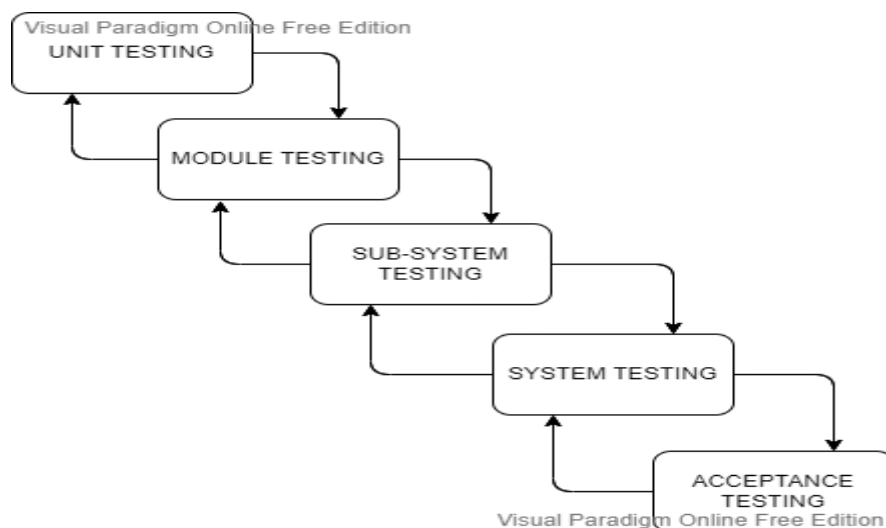


Fig: 8.1 - Testing Plan

8.2 SOFTWARE TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

8.2.1 TESTING METHODOLOGIES

The following are the Testing Methodologies:

- Unit Testing.
- Integration Testing.
- Black box Testing.
- White box Testing.

8.2.2 Unit Testing

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a module's control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is Unit Testing. During this testing, each module is tested individually and the module interfaces are verified for the consistency with design specification. All important processing path are tested for the expected results. All error handling paths are also tested.

8.2.3 Integration Testing

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

8.2.4 White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level.

8.2.5 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

TESTING STRATEGY:

A strategy for system testing integrates system test cases and design techniques into a well- planned series of steps that results in the successful construction of software. The testing strategy must co-operate test planning, test case design, test execution, and the resultant data collection and evaluation. A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level tests that validate major system functions against user requirements. Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing represents an interesting anomaly for the software. Thus, a series of testing are performed for the proposed system before the system is ready for user acceptance testing.

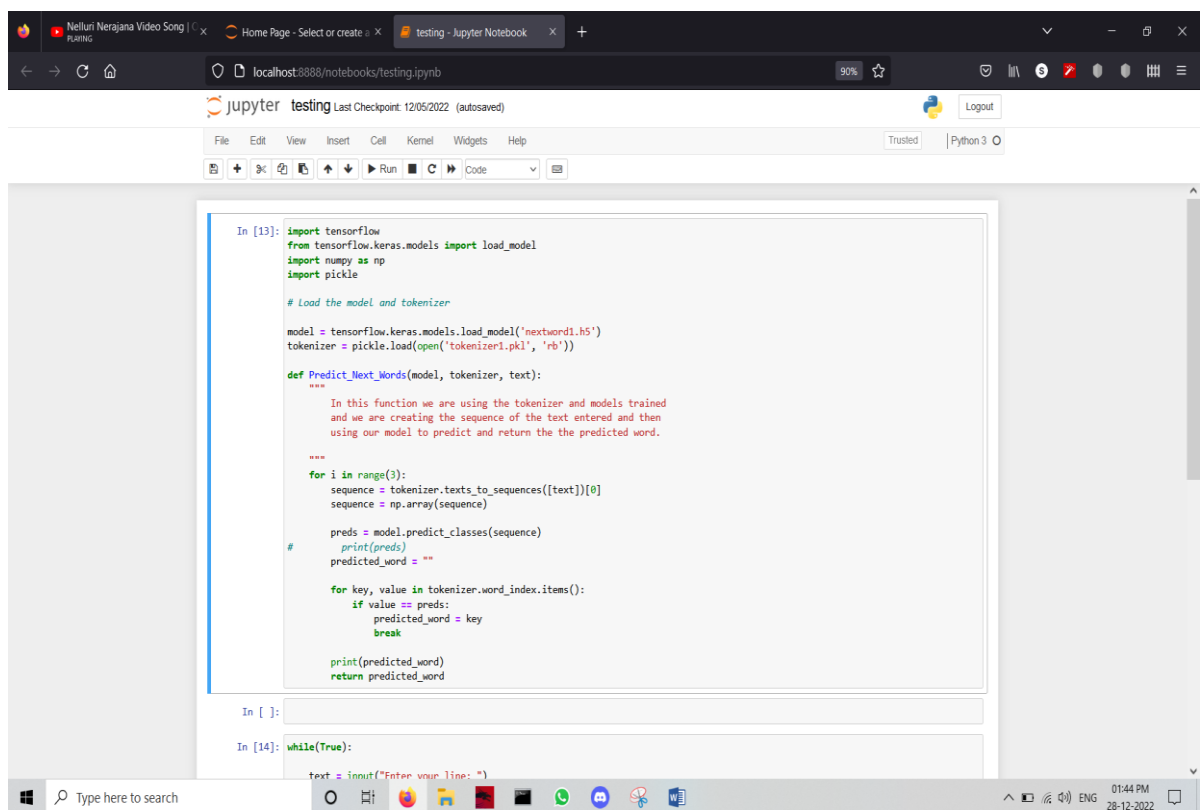
8.3 TEST CASES AND RESULTS:

S.no	Test Case	Excepted Result	Result	Remarks
1	No input	No output	No output	As the user did not enter any data the model would not be able to predict anything
2	One Letter input ("A" and "I")	Next word Prediction	Pass	Next word is predicted based on the letter.
3	One Letter input (other than "A" and "I")	No output	Pass	No prediction is given if the letter is not "A" or "I"
3	One word input	Next word prediction	Pass	If the last word is not a noun, then Prediction is given
4	Meaningful word input	Next word prediction	Pass	Next word is predicted based on the word
5	Meaningless word input	No Output	Pass	No prediction is given as the given input is meaningless
6	A sentence as an input	Next Word Prediction	Pass	Next word is predicted based on the sentence
7	Input string is "stop the script"	Ends the program	Pass	The code stops running

9. OUTPUT SCREENS

In this function we are using the tokenizer and models trained and we are creating the sequence of the text entered and then using our model to predict and return the predicted word.

We are testing our model and we will run the model until the user decides to stop the script. While the script is running we try and check if the prediction can be made on the text. If no prediction can be made we just continue.



The screenshot shows a Jupyter Notebook running in a web browser. The browser's address bar shows the URL `localhost:8888/notebooks/testing.ipynb`. The Jupyter interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The notebook contains two code cells. The first cell, labeled 'In [13]:', contains the following Python code:

```
In [13]: import tensorflow
from tensorflow.keras.models import load_model
import numpy as np
import pickle

# Load the model and tokenizer
model = tensorflow.keras.models.load_model('nextword1.h5')
tokenizer = pickle.load(open('tokenizer1.pkl', 'rb'))

def Predict_Next_Words(model, tokenizer, text):
    """
    In this function we are using the tokenizer and models trained
    and we are creating the sequence of the text entered and then
    using our model to predict and return the the predicted word.
    """
    for i in range(3):
        sequence = tokenizer.texts_to_sequences([text])[0]
        sequence = np.array(sequence)

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

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

        print(predicted_word)
        return predicted_word
```

The second cell, labeled 'In [14]:', contains the following Python code:

```
In [14]: while(True):
    text = input("Enter your line: ")
```

The bottom of the screenshot shows the Windows taskbar with the search bar and several application icons. The system clock in the bottom right corner displays the time as 01:44 PM on 28-12-2022.

Fig 9.1: Prediction Code

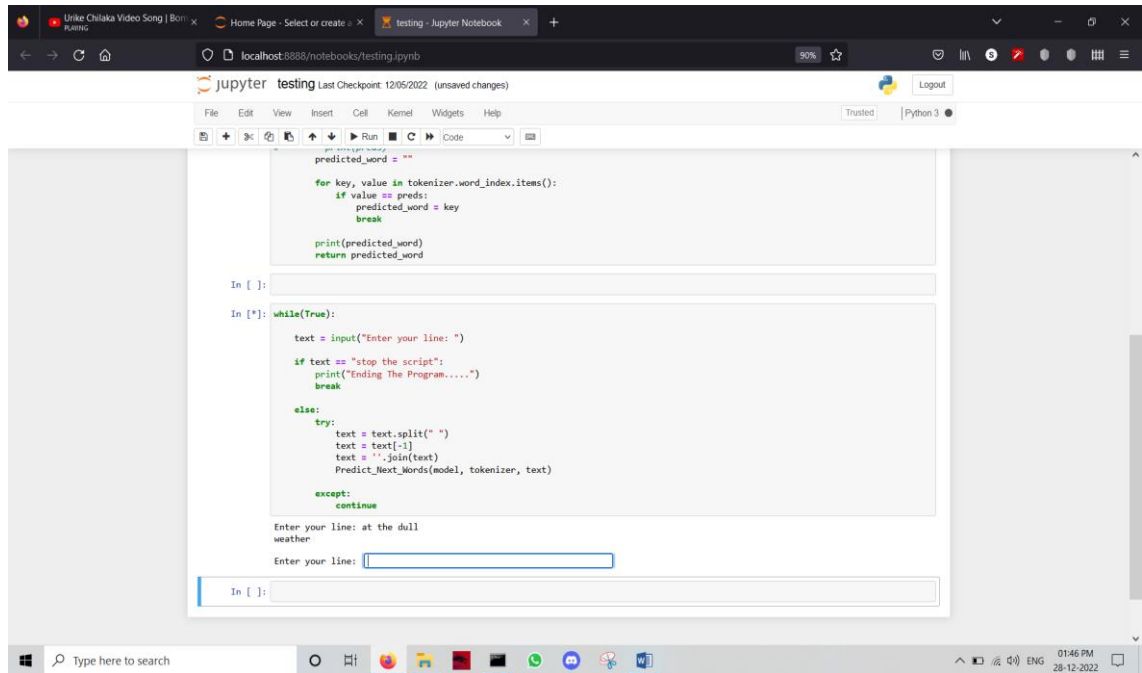


Fig 9.2: Prediction Input Box

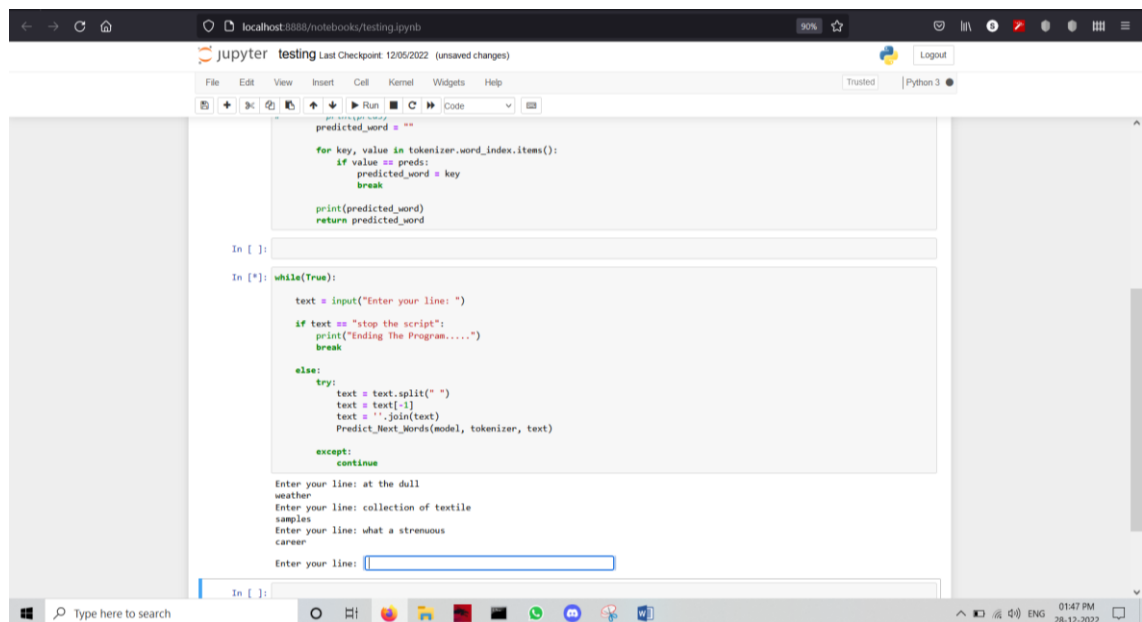


Fig 9.3: Prediction Results

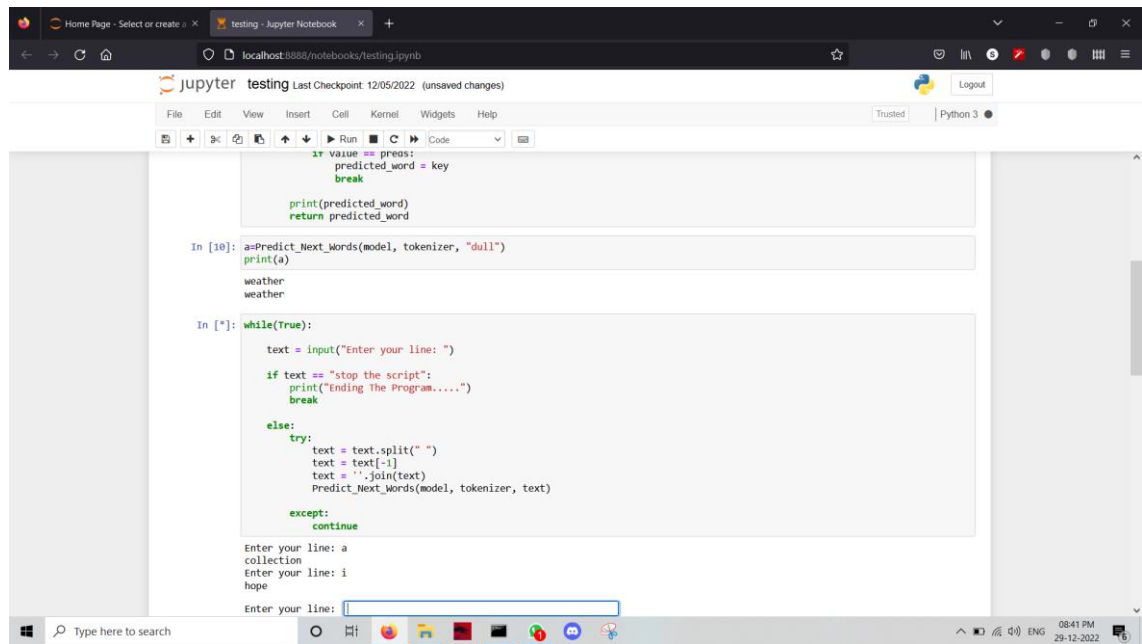


Fig 9.4: One word input

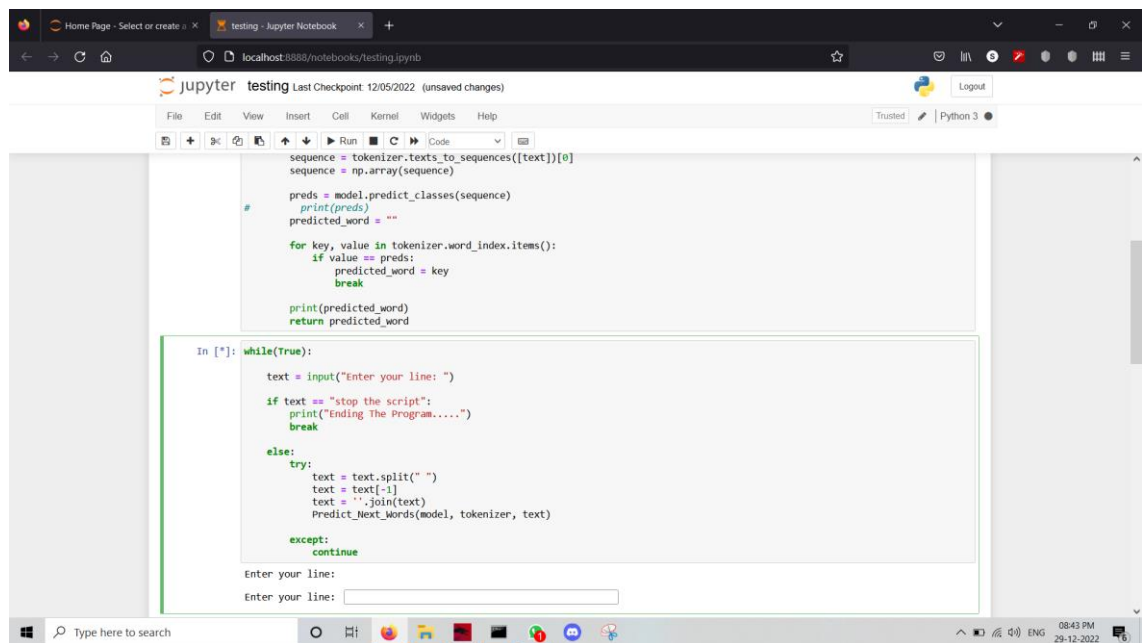


Fig 9.5: Empty Input

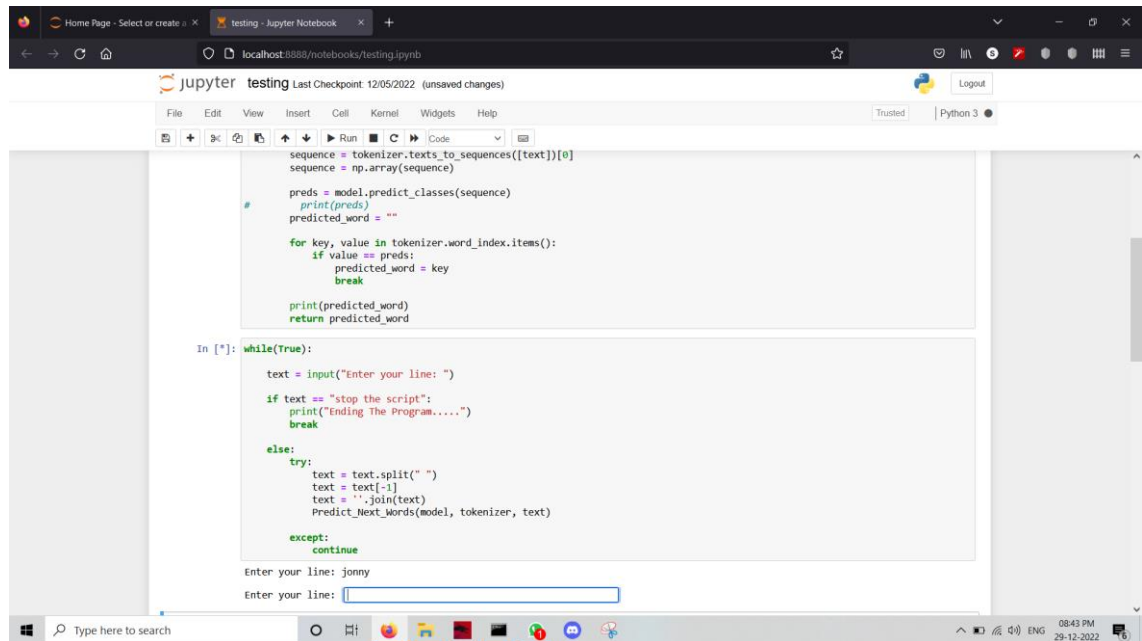


Fig 9.6: Noun as Input

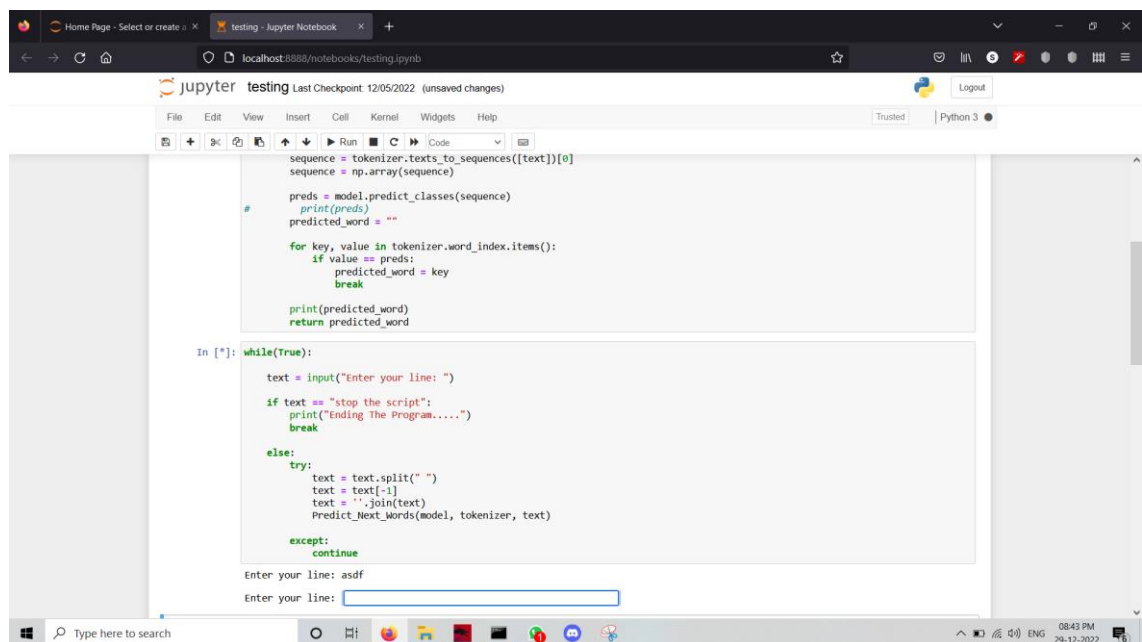


Fig 9.7: Meaningless word input

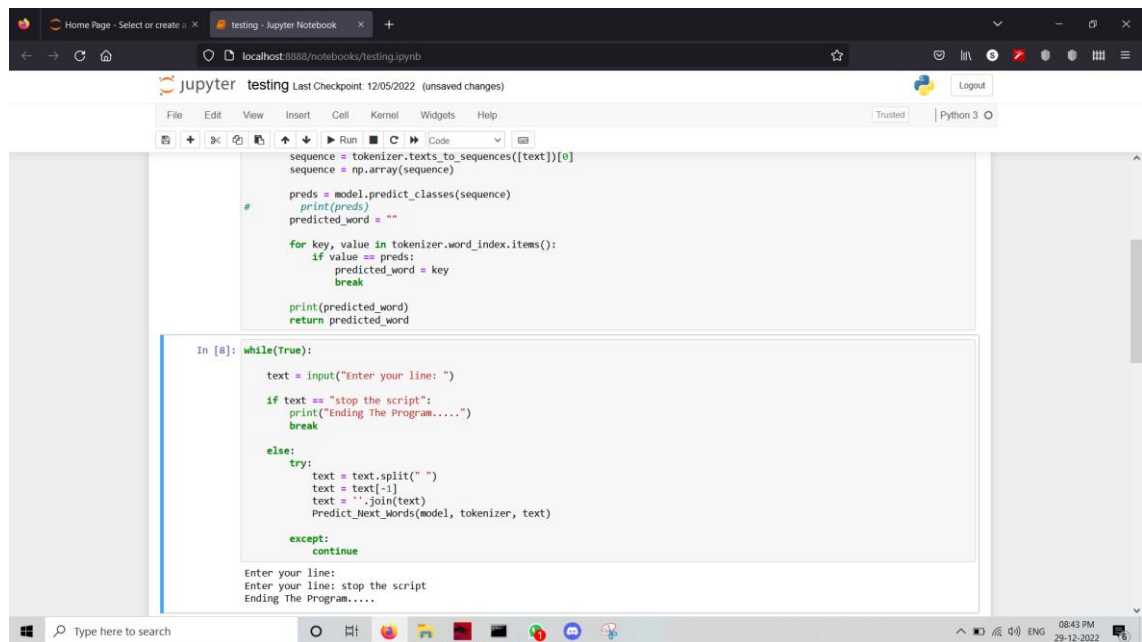


Fig 9.8: Stop the Script

10. CONCLUSION

The subsequent word prediction model that was developed is fairly correct on the provided dataset. NLP requires applying various types of pattern discovery approaches aimed at eliminating noisy data. The loss was considerably reduced in concerning a hundred epochs. Files or dataset that are large to process need still some optimizations. However, bound pre-processing steps and bound changes within the model are often created to boost the prediction of the model. In future, with the rapidly transformation of technology, we may have the technology to get the image with high accuracy and high visual quality. The upcoming filtering methods can provide a better image reconstruction, with less than half of the processing time than those classic vector filters.

Next word prediction is one of NLP fields because it's about mining the text. The researcher here used the LSTM model to make the prediction with 200 epochs. The result showed that it maintained to get accuracy 75% while the loss was 55%. Based on that result, it could be said the accuracy is good enough. As it also showed that it's better than two of the two other researches which used different models. The model could be used to predict the next word by giving the input of the destination.

- We are able to develop a high-quality next word prediction for the dataset.
- We were able to reduce the loss from 7.87 to 0.631 in about 150 epochs.
- The next word prediction model which we have developed is fairly accurate on the provided dataset. The overall quality of the prediction is good.
- However, certain pre-processing steps and certain changes in the model can be made to improve the prediction of the model.

11. REFERENCES

- [1] Jordan, Michael I., and Tom M. Mitchell. "Machine learning: Trends, perspectives, and prospects." *Science* 349, no. 6245 (2015): 255-260.
- [2] Sahoo, Abhaya Kumar, Chittaranjan Pradhan, and Himansu Das. "Performance evaluation of different machine learning methods and deep-learning based convolutional neural network for health decision making." In *Nature inspired computing for data science*, pp. 201- 212. Springer, Cham, 2020.
- [3] Prajapati, Gend Lal, and Rekha Saha. "REEDS: Relevance and enhanced entropy based Dempster Shafer approach for next word prediction using language model." *Journal of Computational Science* 35 (2019): 1-11.
- [4] Ambulgekar, Sourabh, Sanket Malewadikar, Raju Garande, and Bharti Joshi. "Next Words Prediction Using Recurrent NeuralNetworks." In *ITM Web of Conferences*, vol. 40, p. 03034. EDP Sciences, 2021.
- [5] Stremmel, Joel, and Arjun Singh. "Pretraining federated text models for next word prediction." In *Future of Information and Communication Conference*, pp. 477-488. Springer, Cham, 2021.
- [6] Xiaoyun, Qu, Kang Xiaoning, Zhang Chao, Jiang Shuai, and Ma Xiuda. "Short-term prediction of wind power based on deep long short-term memory." In *2016 IEEE PES Asia Pacific Power and Energy Engineering Conference (APPEEC)*, pp. 1148-1152. IEEE, 2016.