Recurrent Neural Network Based Text Generation Techniques By IBM Watson

INTRODUCTION:

1.1 Overview

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing (nlp), speech recognition, and image captioning.Language Modeling is the core problem for a number of natural language processing tasks such as speech to text, conversational system, and text summarization.Text Generation is a type of Language Modeling problem.

1.2 Purpose

A language model allows us to predict the probability of a word in a text given the previous words. Language models are important for various higher level tasks such as machine translation, spelling correction, and so on.The next word that may occur in the sentence is predicted based on the previous words in the sentence. In this project we predicting next hundred characters of a sentence based on the previous characters.

2 LITERATURE SURVEY

2.1 Existing problem

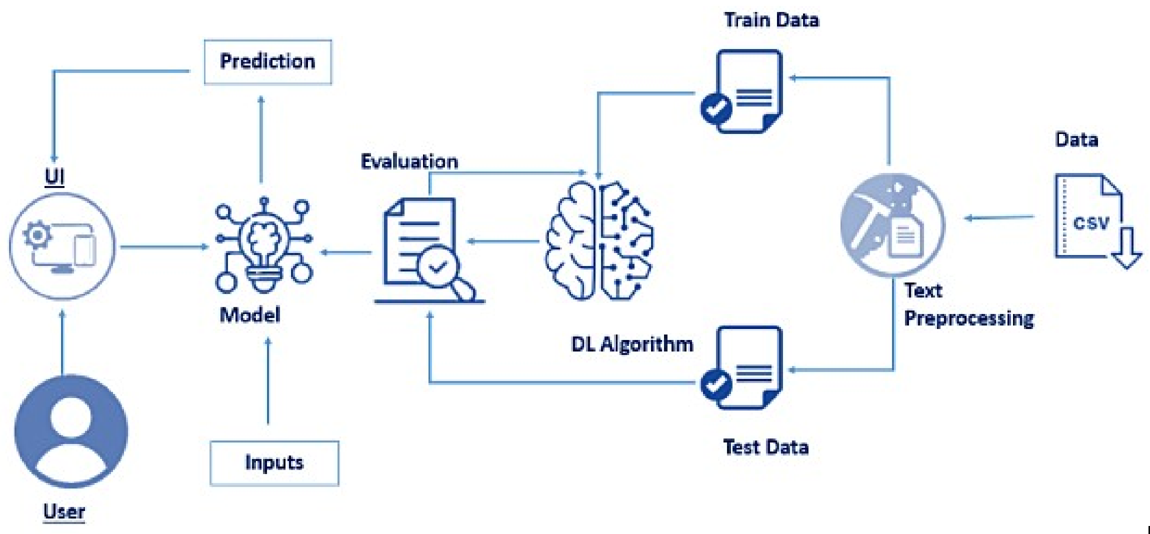
We generally form the next words in a sentence based on the words on our previous experience in usage of words. A sentence is generally formed from what has been learnt about the usage of the words based on our past experiences.However , digitizing the system to predict what might be the next word in the sentence is a
difficult task.

2.2 Proposed Solution

A trained language model learns the likelihood of occurrence of a word based on the previous sequence of words used in the text. Language models can be operated at character level, n-gram level, sentence level or even paragraph level. In this project, we are creating a language model for generating natural language text by implementing and training state-of-the-art Recurrent Neural Network.

THEORITICAL ANALYSIS

3.1 Block Diagram



3.2 Hardware and Software Requirements

This project is deployed in python using Anaconda navigator.
Anaconda Navigator :

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupiter notebook and spyder

The various packages used in the project are :

Tensorflow package:
In Python, if you want to move data through a graph then you can easily use the TensorFlow library for creating dataflow graphs. Basically tensor is a matrix of n dimensions that represents the input type and flows works on the basis of flow graphs that have edges and nodes. It is used as a visualization of graph library for the python.To install the Tensor flow package we can use the command as:

```
pip install tensorflow==2.5.0
```

Keras package:
Keras being a model-level library helps in developing deep learning models by offering high-level building blocks. All the low-level computations such as products of Tensor, convolutions, etc. are not handled by Keras itself, rather they depend on a specialized tensor manipulation library that is well optimized to serve as a backend engine.

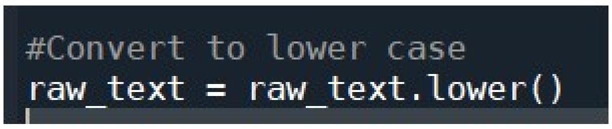
pip install keras

Numpy Package:
NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

pip install numpy

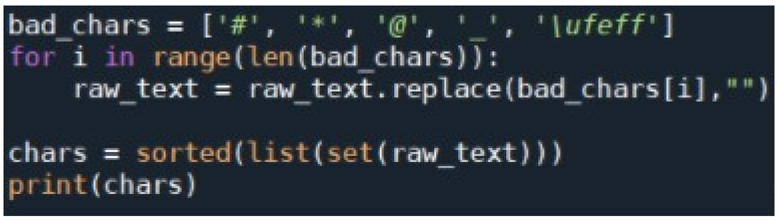
4 EXPERIMENTAL INVESTIGATIONS

This project deals with predicting the occurence of next word based on rnn.Natural Language processing is the process of filtering the text. As text can be diverse and in various forms it is essential to perform text cleaning and preprocessing. The nltk library and re libraries are useful for text preprocessing. Initially we convert all the data into lowercase letters to limit ourselves to the less number of alphabets. However we can also use the text with upper case lettes.We use sub() function from re library to substitute unnecessary characters with empty strings.



```
#Convert to lower case
raw_text = raw_text.lower()
```

we can also use replace() function instead of sub() function for the string. The following code replaces the unncesssary characters in empty string.



```
bad_chars = ['#', '*', '@', '_', '\ufeff']
for i in range(len(bad_chars)):
    raw_text = raw_text.replace(bad_chars[i],"")

chars = sorted(list(set(raw_text)))
print(chars)
```

The following is the flow of action for the project

- User interacts with the UI (User Interface) to enter the input values
- Entered input values are analyzed by the model which is integrated
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities and tasks listed below

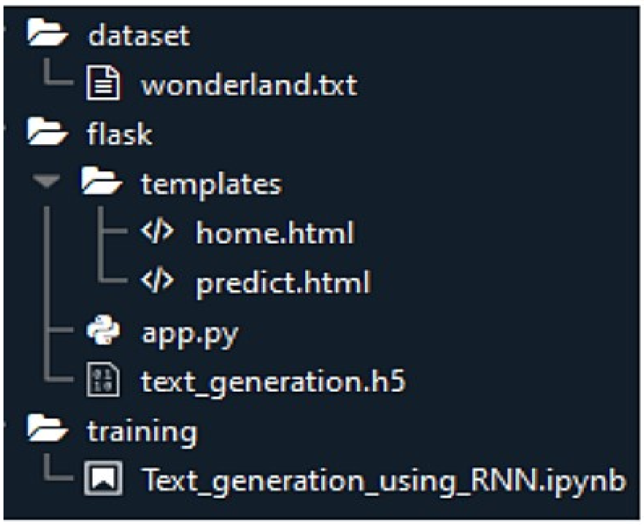
-

- Data Collection:

- Collect the dataset or Create the dataset
- Text Preprocessing:
    - **Import the Libraries.**
    - **importing the dataset**
    - **Clean the data**
    - **Create sliding window**
- Model Building
    - **Import the model building Libraries**
    - **Initializing the model**
    - **Adding LSTM Layers**
    - **Adding Output Layer**
    - **Configure the Learning Process**
    - **Train and Save the Model**
    - **Test the Model**
- Application Building
    - **Create an HTML file**
    - **Build a Python Code**
    - **Run the app**

Project Structure:

Your project structure looks like below



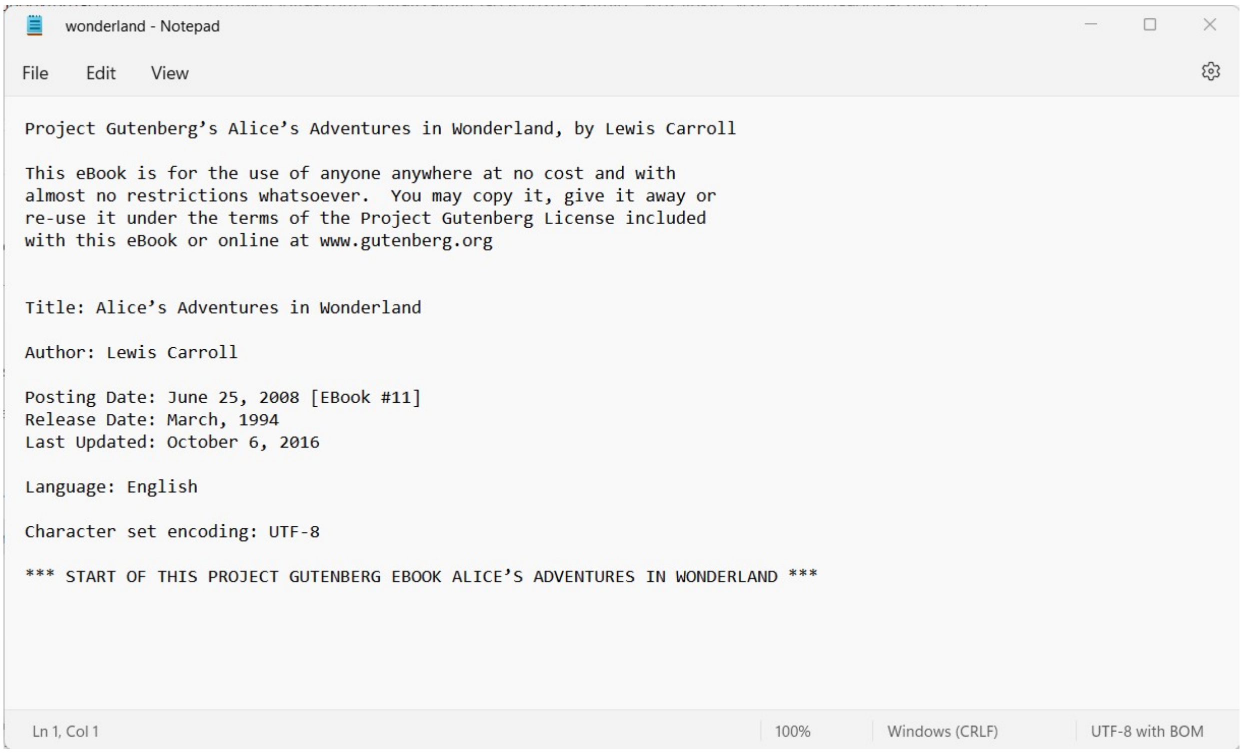
```
dataset
└ wonderland.txt
flask
  templates
  │ ├ home.html
  │ └ predict.html
  ├ app.py
  └ text_generation.h5
training
└ Text_generation_using_RNN.ipynb
```

We are building a Flask Application which needs HTML pages stored in the templates folder and a python script app.py for server-side scripting. we need the model which is saved and the saved model in this content is text_generation.h5

An NLP& RNN file Text_generation_using_RNN.ipynb to train the machine

Data Collection:

ML depends heavily on data, without data, it is impossible for an “AI” to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training **data set.** It is the actual **data set** used to train the model for performing various actions.The data set used here is a text data set named "wonderland.txt".



```
Project Gutenberg’s Alice’s Adventures in Wonderland, by Lewis Carroll

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org


Title: Alice’s Adventures in Wonderland

Author: Lewis Carroll

Posting Date: June 25, 2008 [EBook #11]
Release Date: March, 1994
Last Updated: October 6, 2016

Language: English

Character set encoding: UTF-8

*** START OF THIS PROJECT GUTENBERG EBOOK ALICE’S ADVENTURES IN WONDERLAND ***
```

Milestone 2: Text Preprocessing:

Text Pre-processing includes the following main tasks

- Import the Libraries.
- importing the dataset
- Clean the data
- Create a sliding window

Import the Libraries

The first step is usually importing the libraries that will be needed in the program.

Import NumPy, keras.utils and sys libraries to your Python script:

import numpy as np

from keras.utils import np_utils

import sys

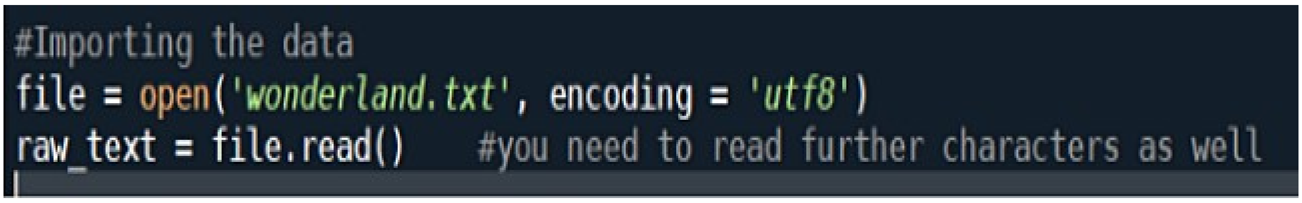
Keras.utils package provides utilities for Keras, such as modified callbacks, generators, etc

Importing the Dataset

Next, we download the training data. The children book "Alice's Adventures in Wonderland" written by Lewis Caroll has been used as the training data in this project. The downloaded book has been stored in the root directory with the name 'wonderland.txt'. We open this book using the open command.

Sometimes, In Python, there is no need for importing an external library to read and write files. Python provides an inbuilt function for creating, writing, and reading files. You can read a file in Python by calling the .txt file in a "read mode"(r)

Let’s load a .txt data file into our program.We will need to locate the directory of the text file at first (it’s more efficient to keep the dataset in the same directory as your program).



```
#Importing the data
file = open('wonderland.txt', encoding = 'utf8')
raw_text = file.read()    #you need to read further characters as well
```

Next, we summarize the entire book to find that the book consists of a total of 163,721 characters To check this we use

print(len(raw_text))

Clean the data

Convert each word into its lower case

Words having the same meaning awesome, Awesome, AWESOME will be considered as non-identical words in the vector space model. So its better to covert all the words into lowercase.This helps in reducing the memory storage.

Storing all the unique characters present in the text -

Next, we store all the distinct characters occurring in the book in the chars variable. We also remove some of the rare characters (stored in bad-chars) from the book. The final vocabulary of the book is printed at the end of code segment.
The chars variable contains all the unique characters as follows

It is observed from the above output, that chars variable contains bad chars like '#', '*', '@', '_', '\ufeff'.

Let's remove bad characters by replacing them with “.

Now we notice that the unwanted characters are removed from the chars variable

Creating a sliding window

Now, we need to create our dataset in the form the model will need. So, we create an input window of 100 characters (SEQ_LENGTH = 100) and shift the window one character at a time until we reach the end of the book. Encoding is used, so as to map each of the characters into it's corresponding location in the vocabulary. enumerate() function takes any iterable as argument and returns enumerate object using which the iterable can be traversed. Such enumerate object with index and value is then converted to a dictionary using dictionary comprehension. Now the char_to_int dictionary contains character and its index value(key value pair).

We have declared two list to store data_Xand data_Y.

Each time the input window contains a new sequence, it is converted into integers, using this encoding and appended to the input list data_X. For all such input windows, the character just following the sequence is appended to the output list data_Y.

Reshaping the data_X and data_Y

Now, the data_X and data_Y lists are converted into a numpy array of required dimensions, so that they can be fed to the model for the training.

For LSTM model , it is necessary to reshape the data_X into 3 dimensional array before building the model.

reshape input to be [samples, time steps, features] which is required for LSTM.

The shape of data_X and data_Y is as follows respectively

Let’s binariase the textual data in data_Y using to_categorical() function which is present in np.utils library.

np.utils.to_categorical is used to convert array of labeled data to one-hot vector.

Model Building:

Import the model building Libraries

Importing the nessacary libraries. The libraries required at this step are keras,keras.models,keras.layes,keras.optimizers,keras.callbacks .

Initializing the model

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define a linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add() method.

model=Sequential()

This initialise the variable "model" to Sequential object.

Adding LSTM Layers

model.add(256,input_shape=(SEQ_LENGTH, 1), return_sequences = True)

model.dropout(0.2)

The model is given an input of 100 character sequences and it outputs the respective probabilities with which a character can succeed the input sequence. The model consists of 3 hidden layers. The first two hidden layers consist of 256 LSTM cells, and the second layer is fully connected to the third layer. The number of neurons in the third layer is same as the number of unique characters in the training set.

return_sequences is True as we need to add another LSTM layer after the current one. input_shape corresponds to the number of time stamps and the number of indicators.

Dropout layer is used to deactivate 20% of the neurons to avoid over fitting and increase the accuracy .

Adding Output Layers

The dense layer is deeply connected neural network layer. It is most common and frequently used layer.

The number of neurons in the third layer is same as the number of unique characters in the training set. The neurons in the third layer, use softmax activation so as to convert their outputs into respective probabilities.

Configure the learning process

The compilation is the final step in creating a model. Once the compilation is done, we can move on to training phase.Loss function is used to find error or deviation in the learning process. Keras requires loss function during model compilation process.

Optimization is an important process which optimize the input weights by comparing the prediction and the loss function. Here we are using adam optimizer

Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in training process

Train and save the module

Now, finally the model is build and then fitted for training.

Model Checkpoint callback is used in combination with training using model.fit() to save a model or weights (in a checkpoint file) at some interval, so the model or weights can be loaded later to continue the training from the state saved.

A few options this callback provides include:

Whether to only keep the model that has achieved the "best performance" so far, or whether to save the model at the end of every epoch regardless of performance.

Definition of 'best'; which quantity to monitor and whether it should be maximized or minimized.The frequency it should save at. Currently, the callback supports saving at the end of every epoch, or after a fixed number of training batches.

Whether only weights are saved, or the whole model is saved.The model is trained for 10 epochs and a batch size of 128 sequences has been used. After every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 10 epochs and probably there is further scope to improve the model.The model is saved with .h5 extension .An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

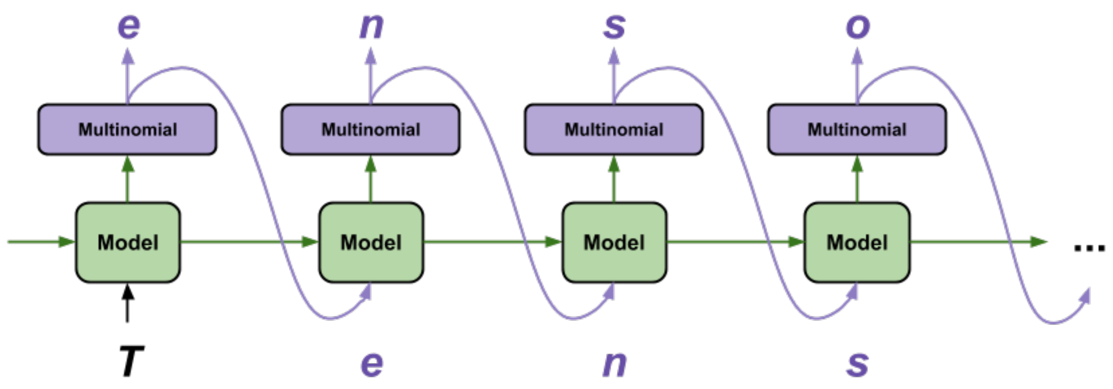
Test the module

The trained model can be used for different applications like generating texts as well as predicting the next word, .The best model with the least loss (as we obtained in the last epoch of training) is loaded and the model is built .
The initial 100 character seed used for generating text is the first few characters of the famous children book 'The Cat in the Hat' by Dr. Seuss.After taking the input, the characters are converted into indexes using char_to_int dictionary.Starting with the initial seed next 100 characters are generated by shifting the 100 character input window for generating the next character(Prediction).

Reshape the inputs and predict using model_predict predefined function. Then argmax function which is a predefined function in NumPy is used to return the indices of the characters and the indexes are converted into respective characters and the text is generated.

```
X = np.reshape(test_text, (1,SEQ_LENGTH,1))
X = X/float(VOCABULARY)
Prediction = model.predict(X)
index = np.argmax(Prediction)
```

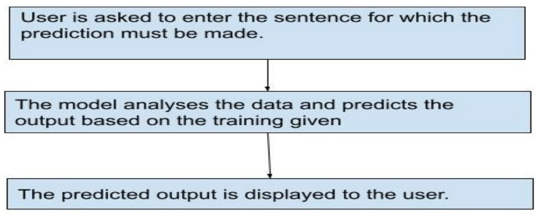
The simplest way to generate text with this model is to run it in a loop, and keep track of the model's internal state as you execute it.



Each time you call the model you pass in some text and an internal state. The model returns a prediction for the next character and its new state. Pass the prediction and state back in to continue generating text.

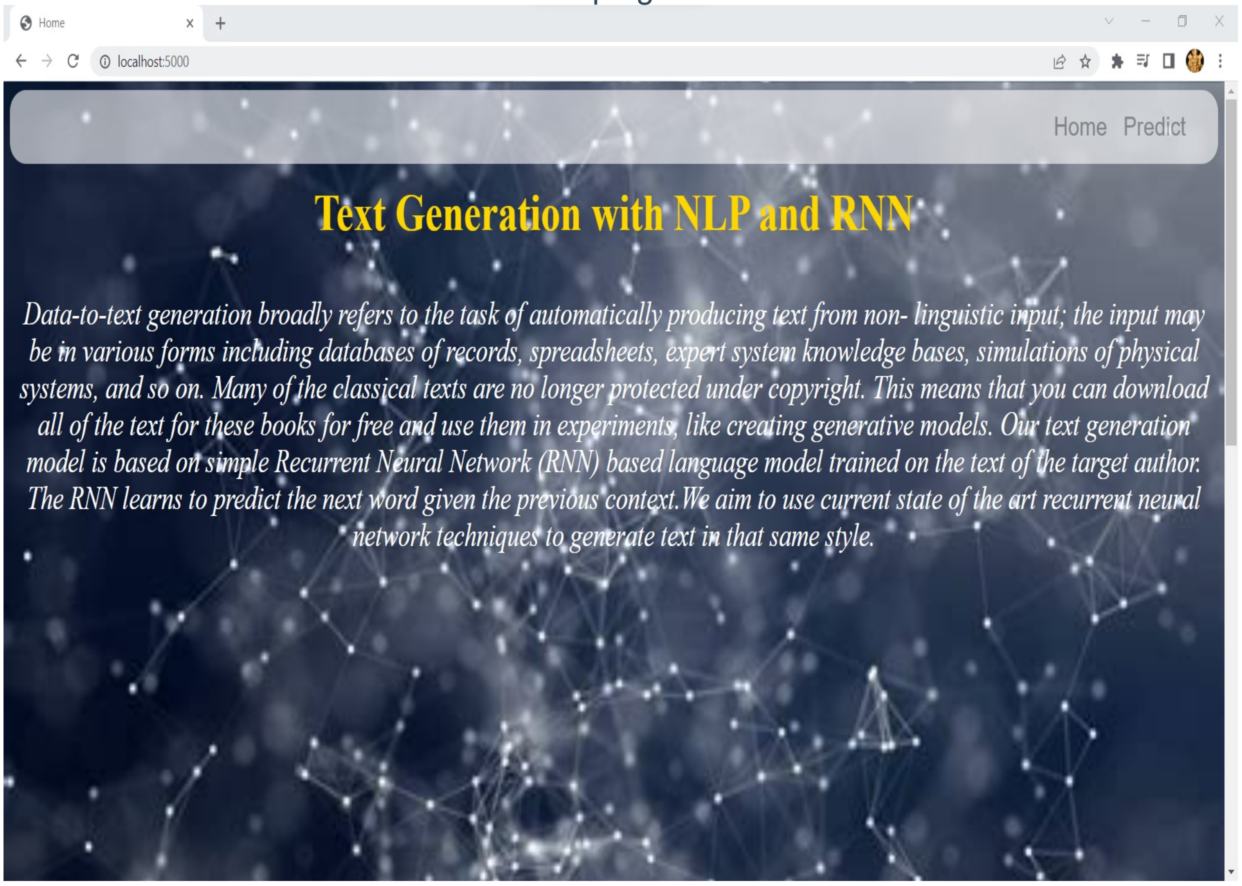
5 FLOW CHART

The project flow will be as follows in the project.User gives the sentence for which the next occurence of word must be predicted. The model analyses the sentence and predicts the next word which will be displayed to user.


User is asked to enter the sentence for which the prediction must be made.
The model analyses the data and predicts the output based on the training given
The predicted output is displayed to the user.

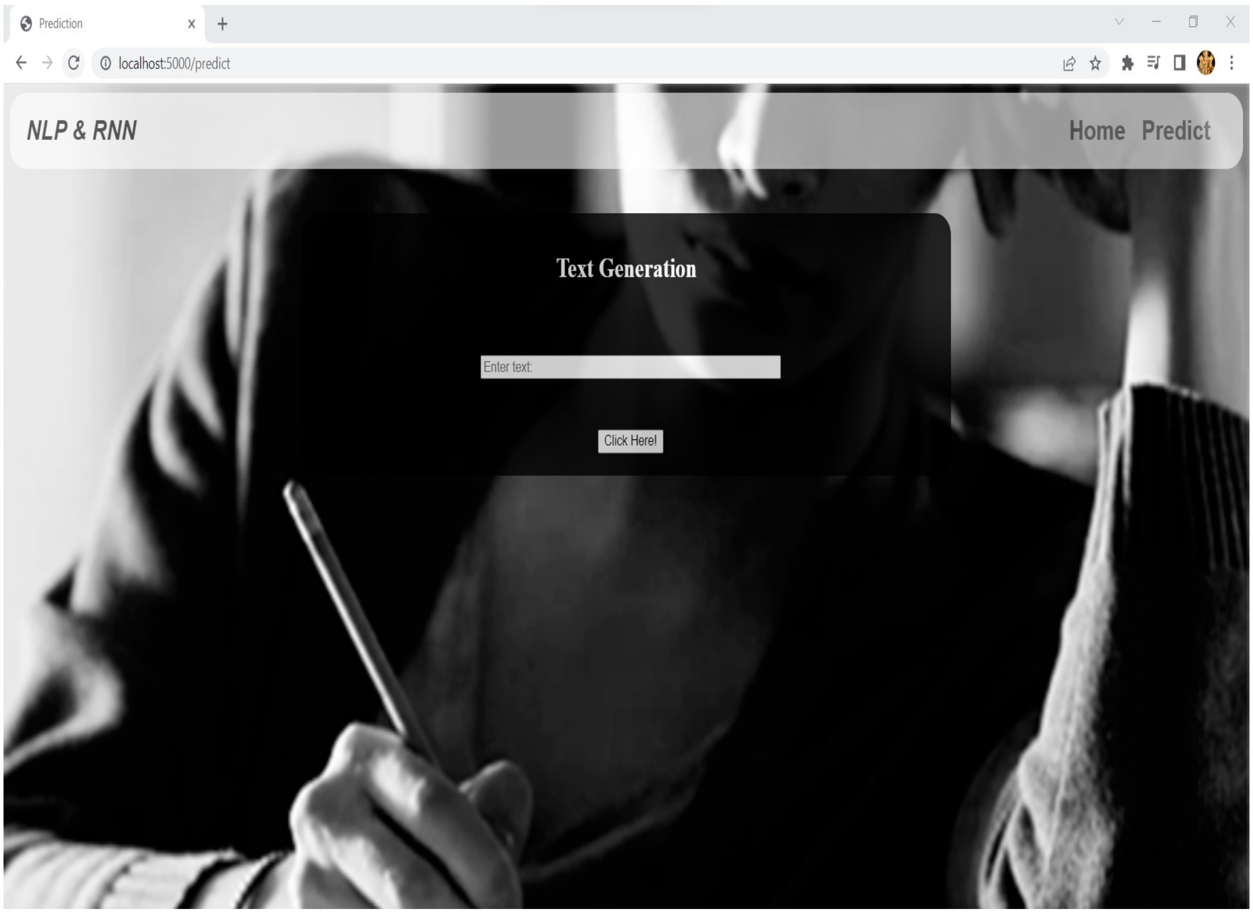
6 RESULT

Initially "home.html" will be as follows.
We need to click on "Predict" button on top right



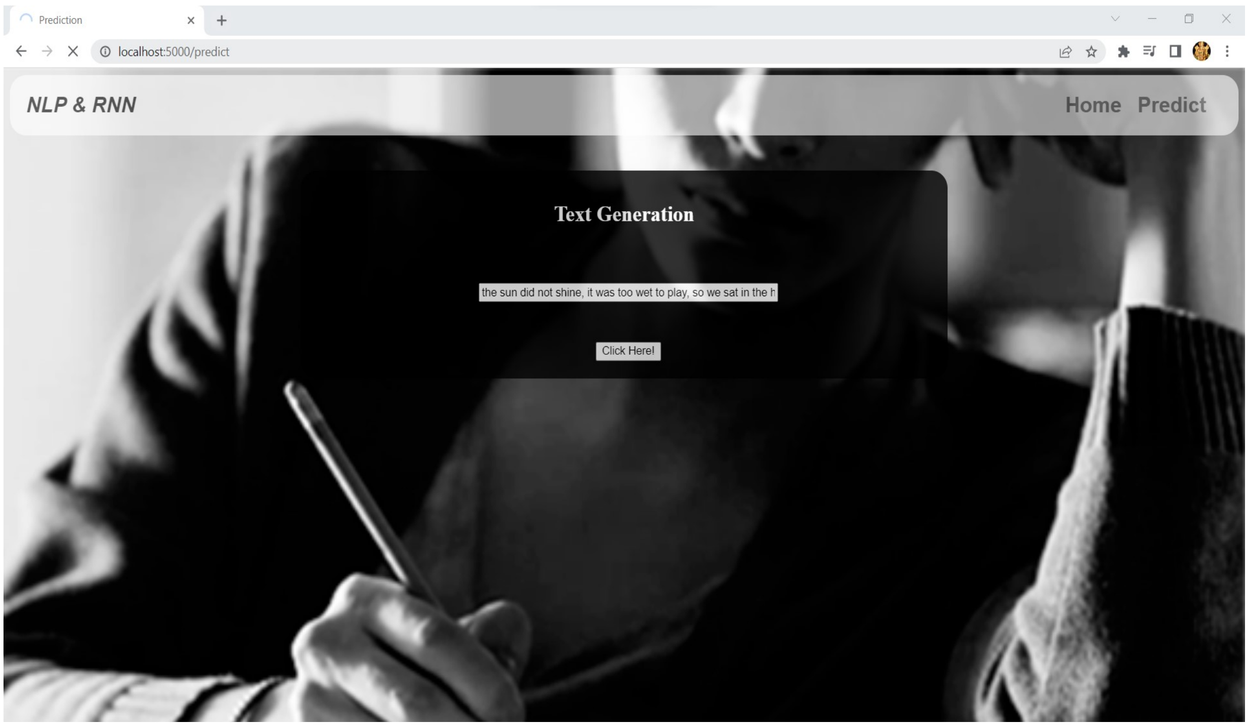
Text Generation with NLP and RNN

Data-to-text generation broadly refers to the task of automatically producing text from non- linguistic input; the input may be in various forms including databases of records, spreadsheets, expert system knowledge bases, simulations of physical systems, and so on. Many of the classical texts are no longer protected under copyright. This means that you can download all of the text for these books for free and use them in experiments, like creating generative models. Our text generation model is based on simple Recurrent Neural Network (RNN) based language model trained on the text of the target author. The RNN learns to predict the next word given the previous context.We aim to use current state of the art recurrent neural network techniques to generate text in that same style.

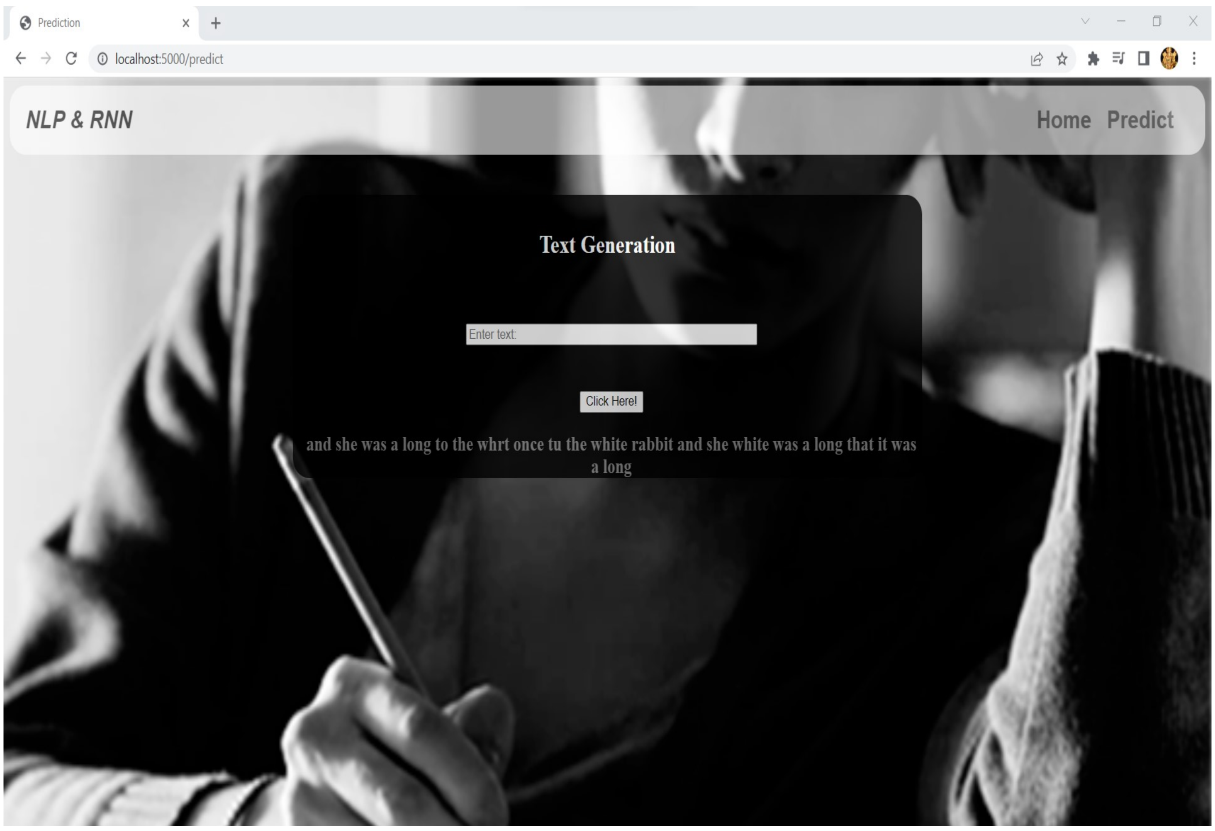


Now from this page enter the text to predict the output on UI

Consider the text we have entered is "the sun did not shine, it was too wet to play, so we sat in the house, all that cold, cold wet day. ". It has produced following output.



the sun did not shine, it was too wet to play, so we sat in the h



and she was a long to the whrt once tu the white rabbit and she white was a long that it was a long

7 ADVANTAGES AND DISADVANTAGES

Advantages:

It is useful to generate next words which may occur in the sentence.

Disadvanatage:

To predict the occurence of next word accurately , it is necessary to understand the previous context.

8 APPLICATIONS

This project uses natural language processing to accomplish the task. It is useful to give suggestion for the occurence of next word

9 CONCLUSION

Given a character, or a sequence of characters, what is the most probable next character? This is the task you're training the model to perform. The input to the model will be a sequence of characters, and you train the model to predict the output—the following character at each time step.

Since RNNs maintain an internal state that depends on the previously seen elements, given all the characters computed until this moment, what is the next character

10 FUTURE SCOPE

The accuracy for the prediction can be further improved by giving dataset that would train the models with sentences that would occur in different contexts.