**Department of Computer Engineering and Applications**

**GLA University, Mathura**

**17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,**

**Mathura – 281406**



***Declaration***

*We hereby declare that the work which is being presented in the Mini Project “****Copy Cat Bot”,*** *in partial fulfillment of the requirements for Mini-Project LAB, is an authentic record of our own work carried under the supervision of* ***Mr. Piyush Khandelwal, Assistant Professor, GLA University, Mathura****.*

**Name of Students with signature**



**Department of Computer Engineering and Applications**

**GLA University, Mathura**

**17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,**

**Mathura – 281406**

**CERTIFICATE**

*This is to certify that the project entitled* ***“Copy Cat Bot ”*** *carried out in Mini Project – ILab is a bonafide work done by* ***Rishabh Sengar (161500451), Ram Kamra (161500437) ,Priyank Kaushik (161500414) and Prashant Raghav (161500401)*** *and is submitted in partial fulfillment of the requirements for the award of the degree Bachelor of Technology (Computer Science & Engineering).*

**Signature of Supervisor:**

**Name of Supervisor:**

**Date:**

**ACKNOWLEDGEMENT**

*It gives us a great sense of pleasure to present the report of the B. Tech Mini Project undertaken during B. Tech. Third Year. This project in itself is an acknowledgement to the inspiration, drive and technical assistance contributed to it by many individuals. This project would never have seen the light of the day without the help and guidance that we have received.*

*Our heartiest thanks to* ***Dr. (Prof). Anand Singh Jalal,*** *Head of Dept., Department of CEA for providing us with an encouraging platform to develop this project, which thus helped us in shaping our abilities towards a constructive goal.*

*We owe special debt of gratitude to* ***Mr****.* ***Piyush Khandelwal,*** *Assistant Professor Department of CEA, for his constant support and guidance throughout the course of our work. His sincerity, thoroughness and perseverance have been a constant source of inspiration for us. He has showered us with all his extensively experienced ideas and insightful comments at virtually all stages of the project & has also taught us about the latest industry-oriented technologies.*

*We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind guidance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project.*

Rishabh Sengar

Priyank Kaushik

Ram Kamra

Prashant Raghav

**Abstract**

The purpose of this project is to discuss about text generation, using Deep learning approaches, especially Recurrent Neural Networks (RNN). It is not the first project about it, and probably not the last. Actually, there is a lot of literature about text generation using “AI” techniques, and some codes are available to generate texts from existing novels, trying to create new chapters for great success like **”Game of Thrones”**, **”Harry Potter”**, or a complete new theater scene in the style of **Shakespeare**. Sometimes with interesting results.

Mainly, these approaches are using standard RNN such as LSTM (Long Short-Term Memory), and they are pretty fun to be experimented. However, generated texts have a taste of unachievable. Generated sentences seem to be quite right, with correct grammar and syntax, as if the neural network was understanding correctly the structure of a sentence. But the whole new text does not have a great sense. And sometimes, has complete nonsense.

This result could come from the approach itself, using only LSTM to generate text, word by word. But how can we improve them? In this project, I will try to investigate a *slightly* different way to generate sentences in a text generator solution. It does not mean that. We will use something completely different than LSTM: We will use LSTM networks to generate sequences of words. However. We will try to go further than a classic LSTM neural network and we will use an additional neural network (LSTM again), to select the best phrases in the text generation.

**Table of Contents**

|  |  |
| --- | --- |
| Declaration | ii |
| Certificate | iii |
| Acknowledgments | iv |
| Abstract | v |
| Table of Contents | vi |
| 1. **Introduction**(This chapter must describe introduction about your project) | **1** |
| 1.1 Motivation and Overview | 1 |
| 1.3 Objective | 2 |
| 1. **Software Requirement Analysis** | **7** |
| 2.1Define the problem | 7 |
| 2.2 Define the modules and their functionalities (SRS) ……………..………………………………… | 7 |
| 1. **Software Design** | **16** |
| * 1. Use Case Diagram | 16 |
| * 1. Class\ Object Diagram | 17 |
| * 1. Sequence Diagram |  |
| 1. **Testing** | **22** |
| 4.1 Black- Box Testing | 22 |
| 4.2 White- Box Testing | 22 |
| 1. **Implementation and User Interface** | **27** |
| 5.1 Output | 27 |
| **References / Bibliography** | **28** |
| 1. **Appendices** |  |

**1. Introduction**

Recurrent neural networks can also be used as generative models.

This means that in addition to being used for predictive models (making predictions) they can learn the sequences of a problem and then generate entirely new plausible sequences for the problem domain.

Generative models like this are useful not only to study how well a model has learned a problem, but to learn more about the problem domain itself.

In this project you will discover how to create a generative model for text, character-by-character using LSTM recurrent neural networks in Python with Keras.

The idea is to train the RNN with many sequences of words and the target *next\_word.* As a simplified example, if each sentence is a list of five words, then the target is a list of only one element, indicating which is the following word in the original text:

We don’t actually send the strings, but a vectorized representation of the word inside a dictionary of possible words (more on that later). The idea is that after many epochs the RNN will learn “the style” of how the corpus is written, trying to adjust the weights of the network to predict the next word given a sequence of the N previous words.

As We explained in the other story, the corpus contains more than 5 million characters in more than 1 million words. Getting the text is just the beginning of the problem, because as any other machine learning project, it was necessary to analyze, clean and perform some pre-processing of this data.

We won’t enter into details now (probably another story), but to say the least, the data was **dirty.** Thousands of typos, misspellings, slang, incorrect punctuation, etc.

**2. Software Requirement Analysis**

Different modules in our project are :-

**Natural Language Processing:**

NLP is a branch of data science that consists of systematic processes for analyzing, understanding, and deriving information from the text data in a smart and efficient manner. By utilizing NLP and its components, one can organize the massive chunks of text data, perform numerous automated tasks and solve a wide range of problems such as – automatic summarization, machine translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation etc.

**Machine Learning**

Machine learning (ML) is the study of algorithms and mathematical models that computer systems use to progressively improve their performance on a specific task. Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.

**Web Scraping :**

Web scraping a web page involves fetching it and extracting from it. Fetching is the downloading of a page (which a browser does when you view the page). Therefore, web crawling is a main component of web scraping, to fetch pages for later processing. Once fetched, then extraction can take place. The content of a page may be parsed, searched, reformatted, its data copied into a spreadsheet, and so on. Web scrapers typically take something out of a page, to make use of it for another purpose somewhere else. An example would be to find and copy names and phone numbers, or companies and their URLs, to a list.

**Data Analysis**

Data Analysis is a process of inspecting, cleansing, transforming, and modeling data with the goal of discovering useful information, informing conclusions, and supporting decision-making. Data Analysis can easily be performed with the help of different libraries like pandas, matplotlib, seaborn etc.

**Pandas** is a **Python** package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in **Python**. **Pandas** is a software library written for the **Python** programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. **pandas** is free software released under the three-clause BSD license.

**Matplotlib** is a plotting library for the **Python** programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like wxPython, Qt, or GTK+.

Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics.

**Model :**

**Long short-term memory** (**LSTM**) units (or blocks) are a building unit for layers of a recurrent neural network (RNN). A RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate.

**3. Software Design**

**3.1 USE CASE Diagram**

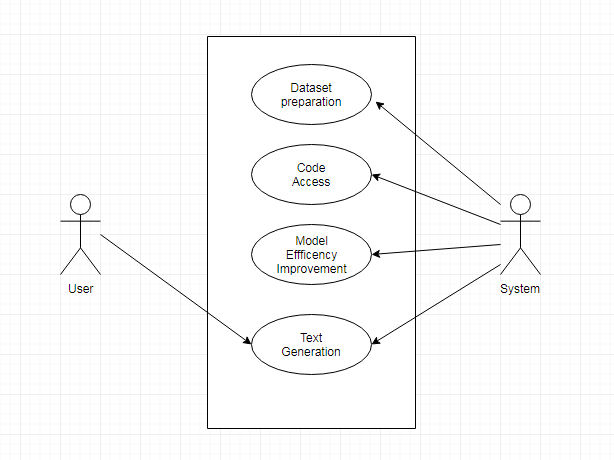
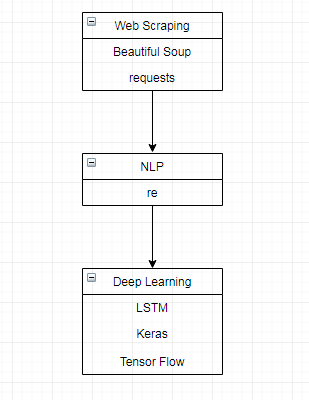
****

Fig1: Use case diagram

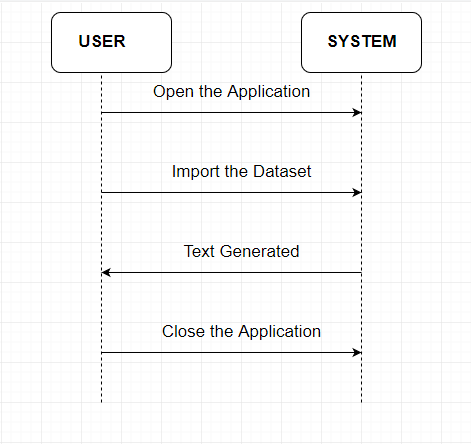
|  |  |
| --- | --- |
| **USE CASE** | **DESCRIPTION** |
| Update dataset | Admin can update dataset. |
| Code access | Admin can update/modify the code as per the requirements. |
| Model Efficiency Improvement | Admin can increase the efficiency by updating the models |
| Text Generation | User gets the generated text. |

**3.3Class / Object Diagram**

****

|  |  |
| --- | --- |
| **Class** | **Description** |
| Web Scrapping | Data extraction from different sources to prepare dataset |
| NLP(Natural Language Processing) | Manipulating text data. |
| Deep learning | A machine learning method that stimulates the neural network in human brain. |
| LSTM(Long Short Term Memory) | A building unit for layers of a recurrent neural network (RNN). |

**3.4 Sequence Diagram**

****

**4. Testing**

“Testing is the process of executing a program with the intent of finding errors.”

Although software testing is itself an expensive activity, yet launching of software without testing may lead to cost potentially much higher than that of testing, specially in systems where human safety is involved.

In the software life cycle the earlier the errors are discovered and removed, the lower is the cost of their removal.

Test and Test case terms are used interchangeably. In practice, both are same and are treated as synonyms. Test case describes an input description and an expected output description. The set of test cases is called a test suite. Hence any combination of test cases may generate a test suite.

**Verification** is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

**Validation** is the process of evaluating a system or component during or at the end of development process to determine whether it satisfies the specified requirements.

**Testing= Verification + Validation**

The term Acceptance Testingis used when the software is developed for a specific customer. A series of tests are conducted to enable the customer to validate all requirements. These tests are conducted by the end user / customer and may range from adhoc tests to well planned systematic series of tests.

The terms alpha and beta testing are used when the software is developed as a product for anonymous customers.

Alpha Testsare conducted at the developer’s site by some potential customers. These tests are conducted in a controlled environment. Alpha testing may be started when formal testing process is near completion.

Beta Testsare conducted by the customers / end users at their sites. Unlike alpha testing, developer is not present here. Beta testing is conducted in a real environment that cannot be controlled by the developer.

**Functional Testing**

BLACK BOX TESTING, also known as Behavioral Testing, is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These **tests** can be functional or non-functional, though usually functional.

Fig: Black box testing

**Structural Testing**

A complementary approach to functional testing is called structural / white box testing. It permits us to examine the internal structure of the program.

**Path Testing**

Path testing is the name given to a group of test techniques based on judiciously selecting a set of test paths through the program. If the set of paths is properly chosen, then it means that we have achieved some measure of test thoroughness.

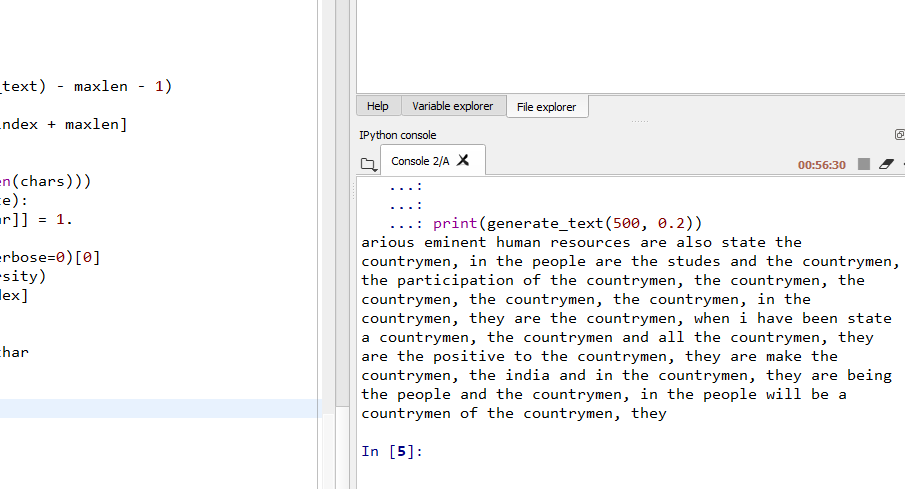
This type of testing involves:

1. generating a set of paths that will cover every branch in the program.
2. finding a set of test cases that will execute every path in the set of program paths.

**WHITE BOX TESTING**(also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure / design / implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system.

**5 . Implementation and User Interface**

* 1. **OUTPUT :-**



1. **References/Bibliography**

* SuperDataScience Team : - https://www.superdatascience.com/
* NLKT : - <https://www.nltk.org/>
* Scipy Lecture Notes : - <http://www.scipy-lectures.org/>
* Naïve Bayes :-<https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/>
* Ensemble Methods :-<https://scikit-learn.org/stable/modules/ensemble.html>
* Scraping :-<https://medium.freecodecamp.org/how-to-scrape-websites-with-python-and-beautifulsoup-5946935d93fe>
* LSTM :- https://keras.io/
* Kaggle :-<https://www.kaggle.com/>
* ResearchGate :- https://www.researchgate.net/

\

1. **Appendices**

**Code Template 1.**

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Importing Pakages

from \_\_future\_\_ import print\_function

import sys

import numpy as np

import pandas as pd

import string, os

import random

import warnings

warnings.filterwarnings("ignore")

# Importing keras

from keras.callbacks import LambdaCallback

from keras.models import Sequential

from keras.layers import Dense, Activation

from keras.layers import LSTM

from keras.optimizers import RMSprop

# Importing the dataset

file=pd.read\_csv("C:/Users/RISHABH/Documents/GitHub/Mini/Copy CatBot/mann\_ki\_baat.csv",encoding="unicode\_escape")

# Dropping of columns like month and year

file.drop('month', axis=1, inplace=True)

file.drop('year', axis=1, inplace=True)

**Explanation :**

In this template we are importing our dataset using pandas library and various other libraries. After that we are dropping our ‘month’ and ‘year’ column so that we can work on remaining features.

**Code Template 2.**

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Merging of all the rows to generate raw text

#raw\_text=list()

raw\_text=''

for i in range(47):

raw\_text=raw\_text+file.iloc[i,0]

#raw\_text.append(file.iloc[i,0])

# Lower casing all the words.

raw\_text = raw\_text.lower()

print('text length', len(raw\_text))

print(raw\_text[:300])

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Explanation :**

Merging of all the rows to generate raw text and Lower casing all the words.

**Code Template 3.**

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# create mapping of unique chars to integers, and a reverse mapping

# Since we are training on character level, therefore we have to relate

# each unique character to a number

chars = sorted(list(set(raw\_text)))

print('total chars: ', len(chars))

character\_to\_integer = dict((c, i) for i, c in enumerate(chars))

integer\_to\_character = dict((i, c) for i, c in enumerate(chars))

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Explanation:-**

Create mapping of unique chars to integers, and a reverse mapping

Since we are training on character level, therefore we have to relate

Each unique character to a number

**Code Template 4.**

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

maxlen = 40

step = 3

sentences = []

next\_character = []

for i in range(0, len(raw\_text) - maxlen, step):

sentences.append(raw\_text[i: i + maxlen])

next\_character.append(raw\_text[i + maxlen])

print('nb sequences:', len(sentences))

print(sentences[:10], "\n")

print(next\_character[:10])

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Explanation :**

This next cell step gives us an array, sentences, made up of maxlen (40) character sequences chunked in steps of 3 characters from our corpus user, and next\_chars, an array of single characters from user at i + maxlen for each i. I've printed out the first 10 strings in the array so you can see we're chunking the corpus into partially overlapping, equal length "sentences."

**Code Template 5.**

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

x = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)

y = np.zeros((len(sentences), len(chars)), dtype=np.bool)

# Encoding character level features from "sentences" and "next\_character".

for i, sentence in enumerate(sentences):

for t, char in enumerate(sentence):

x[i, t, character\_to\_integer[char]] = 1

y[i, character\_to\_integer[next\_character[i]]] = 1

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Explanation :**

Reshape our data in a format we can pass to the Keras LSTM

The shape look like [samples, time steps, features]

Create a sparse boolean tensors x & y.

Encoding character level features from "sentences" and "next\_character".

**Code Template 6.**

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Define the LSTM model

# Sequential modeling( used to define a linear stack of network layer )

model = Sequential()

# Defining Single Hidden LSTM layer with,

# units -> Dimensionality O/P space

# Input\_shape -> shape of the I/P

model.add(LSTM(units = 128, input\_shape=(maxlen, len(chars))))

# Adding Dropout Layer

model.add(Dropout = 0.2)

# Adding O/P layer which is Dense & activation function is softmax

model.add(Dense(len(chars), activation = 'softmax'))

# Compile the network with the loss function and optimizer function.

# This will allow our network to change weights and minimize the loss.

model.compile (loss='categorical\_crossentropy', optimizer='adam')

print (model.summary())

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Explanation :**

In the cell below, we define the model. We start with a sequential model and add an LSTM as an input layer. The shape we define for our input is identical to our data by this point which is exactly what we need. I’ve selected a batch\_size of 128 which is the number of samples, or sequences, our model looks at during training before updating. You can experiment with different numbers here if you want. I'm also adding a dense output layer. Finally, we'll use add an activation layer with softmax as our activation function as we're in essence doing multiclass classification to predict the next character in a sequence.

Now we can compile our model. We’ll use RMSprop with a learning rate of 0.1 to optimize the weights in our model (you can experiment with different learning rates here) and categorical\_crossentropy as our loss function. Cross entropy is the same as log loss commonly used as the evaluation metric in binary classification competitions on Kaggle (except in our case there are more than two possible outcomes).

**Code Template 7.**

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Samples an index from a probability array with some temperature.

# Temperature is the scaling factor applied to our O/P of our dense layer before

## softmax function is applied.

# In short, it defines how "conservative/creative" our model's guesses can be

## for our next character in the sequence.

def sample(preds, temperature=1.0):

# helper function to sample an index from a probability array

preds = np.asarray(preds).astype('float64')

preds = np.log(preds) / temperature

exp\_preds = np.exp(preds)

preds = exp\_preds / np.sum(exp\_preds)

probas = np.random.multinomial(1, preds, 1)

return np.argmax(probas)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Explanation :**

Now our model is ready. Before we feed it any data, the cell below defines a couple of helper functions with code modified from this script. The first one, sample, samples an index from an array of probabilities with some temperature. Quick pause to ask, what is temperature exactly ?

**Temperature** is a scaling factor applied to the outputs of our dense layer before applying the softmax activation function. In a nutshell, it defines how conservative or "creative" the model's guesses are for the next character in a sequence. Lower values of temperature (e.g., 0.2) will generate "safe" guesses whereas values of temperature above 1.0 will start to generate "riskier" guesses. Think of it as the amount of surpise you'd have at seeing an English word start with "st" versus "sg". When temperature is low, we may get lots of " the’s and and’s ” . When temperature is high, things get more unpredictable.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Code Template 8.**

# Callback function to print predicted text generated by our LSTM

# Diversity is for values of Temperature.

def on\_epoch\_end(epoch, logs):

# Function invoked at end of each epoch. Prints generated text.

print()

print('----- Generating text after Epoch: %d' % epoch)

start\_index = random.randint(0, len(raw\_text) - maxlen - 1)

for diversity in [0.2, 0.5, 1.0, 1.2]:

print('----- diversity:', diversity)

generated = ''

sentence = raw\_text[start\_index: start\_index + maxlen]

generated += sentence

print('----- Generating with seed: "' + sentence + '"')

sys.stdout.write(generated)

for i in range(400):

x\_pred = np.zeros((1, maxlen, len(chars)))

for t, char in enumerate(sentence):

x\_pred[0, t, character\_to\_integer[char]] = 1.

preds = model.predict(x\_pred, verbose=0)[0]

next\_index = sample(preds, diversity)

next\_char = integer\_to\_character[next\_index]

generated += next\_char

sentence = sentence[1:] + next\_char

sys.stdout.write(next\_char)

sys.stdout.flush()

print()

print\_callback = LambdaCallback(on\_epoch\_end=on\_epoch\_end)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Explanation:**

Anyway, so the second is defining a callback function to print out predicted text generated by our trained LSTM at the first and then every subsequent fifth epoch with five different settings of temperature each time (see the line for diversity in [0.2, 0.5, 1.0, 1.2]: for the values of temperature; feel free to tweak these, too!). This way we can fiddle with the temperature knob to see what gets us the best generated text ranging from conservative to creative. Note that we're using our model to predict based on a random sequence, or "seed", from our original subsetted data, user: start\_index = random.randint(0, len(user) - maxlen - 1).

Finally, we name our callback function generate\_text which we'll add to the list of callbacks when we fit our model in the cell after this one.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Code Template 9.**

# Here, We will use model check pointing to record all of the network weights to

## a file each time an improvement in loss is observed at the end of the epoch.

from keras.callbacks import ModelCheckpoint

filepath = "weights.hdf5"

checkpoint = ModelCheckpoint(filepath, monitor='loss',

verbose=1, save\_best\_only=True,

mode='min')

# Defining Callbacks

from keras.callbacks import ReduceLROnPlateau

reduce\_lr = ReduceLROnPlateau(monitor='loss', factor=0.2,

patience=1, min\_lr=0.001)

callbacks = [print\_callback, checkpoint, reduce\_lr]

model.fit (x, y, batch\_size=128, epochs=15, callbacks=callbacks)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Explanation:**

Finally we’ve made it! Our data is ready (x for sequences, y for next characters), we've chosen a batch\_size of 128, and we've defined a callback function which will print generated text using model.predict() at the end of the first epoch followed by every fifth epoch with five different temperature setting each time. We have another callback, ModelCheckpoint, which will save the best model at each epoch if it's improved based on our loss value (find the saved weights file weights.hdf5 in the "Output" tab of the kernel).

Let’s fit our model with these specifications and epochs = 15 for the number of epochs to train. And of course, let's not forget to put our GPU to use! This will make training/prediction much faster than if we used a CPU. In any case, you will still want to grab some lunch or go for a walk while you wait for the model to train and generate predictions if you're running this code interactively.

P.S. If you’re running this interactively in your own notebook on Kaggle, you can click the blue square “Stop” button next to the console at the bottom of your screen to interrupt the model training.

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Code Template 10.**

# Generate new text

def generate\_text(length, diversity):

# Get random starting text

start\_index = random.randint(0, len(raw\_text) - maxlen - 1)

generated = ''

sentence = raw\_text[start\_index: start\_index + maxlen]

generated += sentence

for i in range(length):

x\_pred = np.zeros((1, maxlen, len(chars)))

for t, char in enumerate(sentence):

x\_pred[0, t, character\_to\_integer[char]] = 1.

preds = model.predict(x\_pred, verbose=0)[0]

next\_index = sample(preds, diversity)

next\_char = integer\_to\_character[next\_index]

generated += next\_char

sentence = sentence[1:] + next\_char

return generated

print(generate\_text(500, 0.2))

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**EXPLANATION :**

Generate new text

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Code Template 11.**

import requests

import html.parser

from bs4 import BeautifulSoup

import re

from urllib.request import Request, urlopen

file=open('F:\\tdemo\\Desktop\\nm\_code.txt','r')

page1=file.read()

file.close()

#Parsering the html code

soup=BeautifulSoup(page1,'html.parser')

#print(soup.prettify())

title=list()

links=list()

content=list()

subcontent=list()

for div in soup.find\_all('div',class\_='speechesItemLink left\_class '):

#content.append(div.get\_text())

for anch in div.find\_all('a',class\_='left\_class'):

title.append(anch.get\_text())

links.append(anch.get('href'))

print(title)

print(len(title))

print(links)

print(len(links))

for i in range(900,945):

page = Request(links[i], headers={'User-Agent': 'Mozilla/5.0'})

web\_byte = urlopen(page).read()

soup1=BeautifulSoup(web\_byte,'html.parser')

print(page)

print(web\_byte)

print(soup1)

article=soup1.find('article',class\_='articleBody main\_article\_content')

for para in article.find\_all('p'):

subcontent.append(para.get\_text())

content.append(subcontent)

print(content)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**EXPLANATION :**

Web Scraping is the process of extracting meaningful information from a website .Here, we are extracting the speeches of Modi from [www.narendramodi.in](http://www.narendramodi.in) with the help of Beautiful soup, urllib, selenium , requests . We are storing these speeches, title and links into a list and then convert that into a dataset.