RECURRENT NEURAL NETWORK BASED TEXT GENERATION TECHNIQUES BY IBM WATSON

1. INTRODUCTION

1.1 Overview

Text Generation is a type of Language Modeling problem. Language Modeling is the core problem for a number of natural language processing tasks such as speech to text, conversational system, and text summarization. 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.

1.2 Purpose

The text generation task to predict the next character given its previous characters. It employs a recurrent neural network with LSTM layers to achieve the task. The deep learning process will be carried out using Tensor Flow's, Keras, and a high-level API. Generative models like recurrent neural networks are useful not only to study how well a model has learned a problem, but to learn more about the problem domain itself.

2. LITERATURE SURVEY

2.1 Existing problem

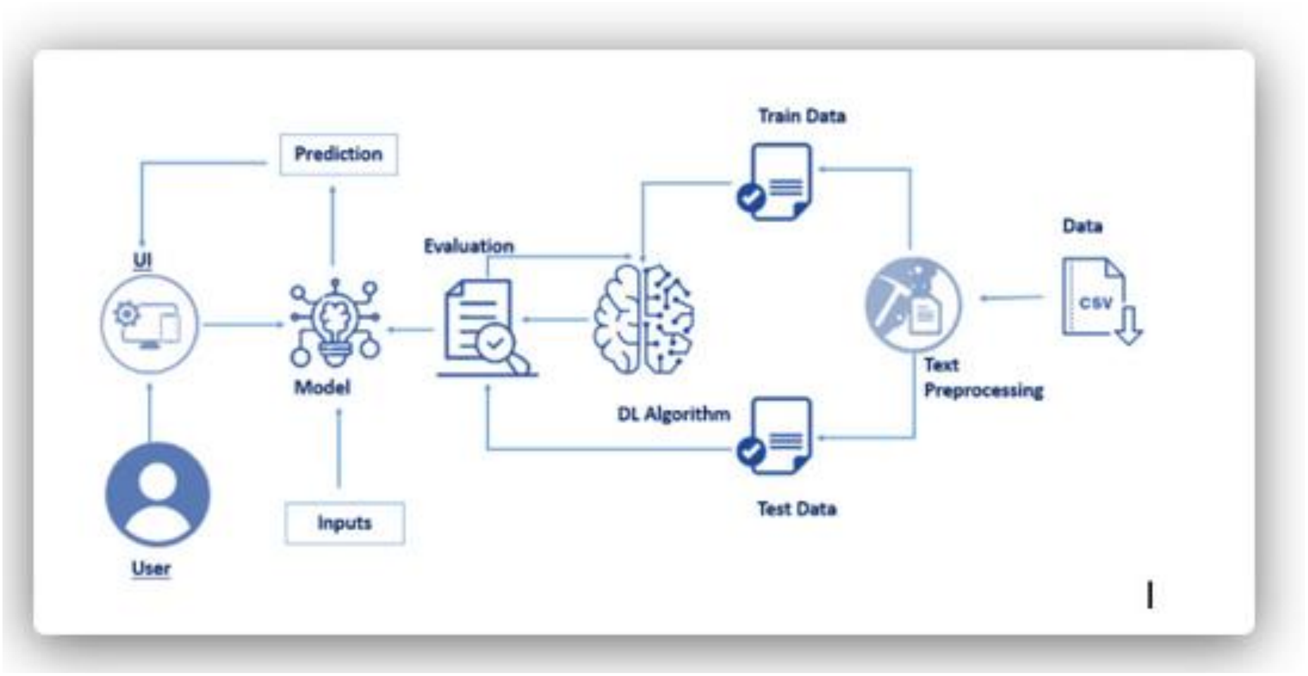
It takes us, humans, many years of learning to communicate in a language. This is simply because languages are complex and constantly evolving. Rules of spelling and grammar aren't standard, there are multiple ways to communicate the same thing, and the same word might mean different things in different contexts. As people, we understand communication through a complex process which takes into account several tangible and intangible aspects. Sarcasm, lingo, hashtags, etc. are processed by the human brain easily, which can be difficult for machines to replicate. But we have to try.

2.2 Proposed solution

A recurrent neural network (RNN) is an upgraded version of the neural network, where connections between nodes are treated as sequential signals. We can build language models across many levels — word-level, phrase-level or what we are going to do today, which is corrector-level. Basically, a corrector is a set of alphabets, punctuation, etc., which helps predict the next corrector. We are doing this primarily because corrector-level language modelling will give you a finite and manageable vocabulary. You can go forth and build word-level models on the same principles as well. We will use Tensor Flow 2.0 with Keras as the high-level library.

3. THEORITICAL ANALYSIS

3.1 Block Diagram



3.2 Hardware / Software designing

Software Requirements:

- Anaconda Navigator
- Keras
- Flask
- IBM Watson

Hardware Requirements:

• Processor	: Intel Core i3
• Hard Disk Space	: Min 100 GB
• Ram	: 8 GB
• Display	: 14.1 “Color Monitor(LCD, CRT or LED)
• Clock Speed	: 1.67 GHz

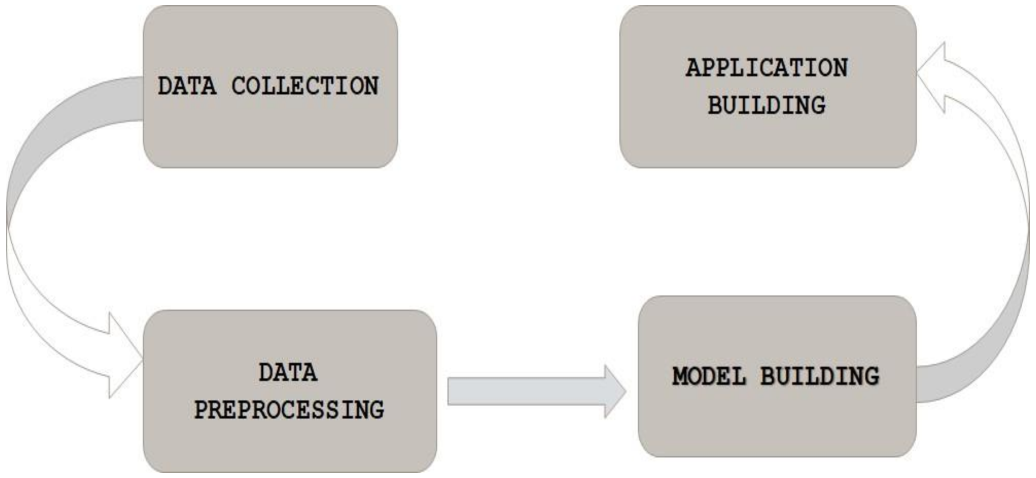
4. EXPERIMENTAL INVESTIGATIONS

Deep learning provides us a way to strap huge amount of computation and data with small efforts done by hand. With the help of word embedding’s and popular models of deep learning like RNNs, CNNs, VAEs, GANs have made NLP problems a lot easier.The enormous increase in available data and computational power, also the developments in deep learning, have provided many new possibilities for researchers to analyse new applications for text generation. We expect such a trend to continue with more and better model designs. We expect to see more NLP applications to employ reinforcement learning methods to explore the research on text generation.

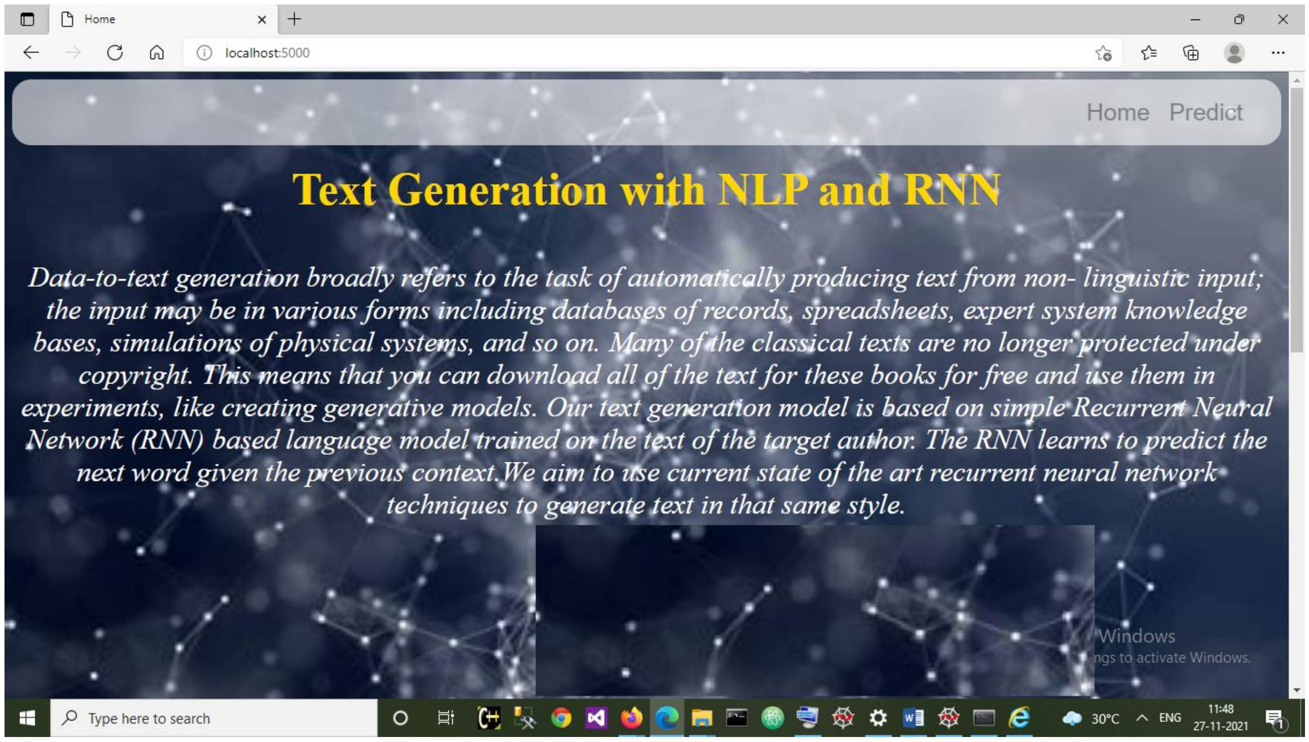
Generating natural language is a chalice of text generation research. The current deep learning models have not yet fully captured the nuances, technicalities, and interpretation of natural language, which aggregates when generating longer text. The evaluation progress of text generation models requires a better metric carefully designed by the human study.

.

5. FLOWCHART

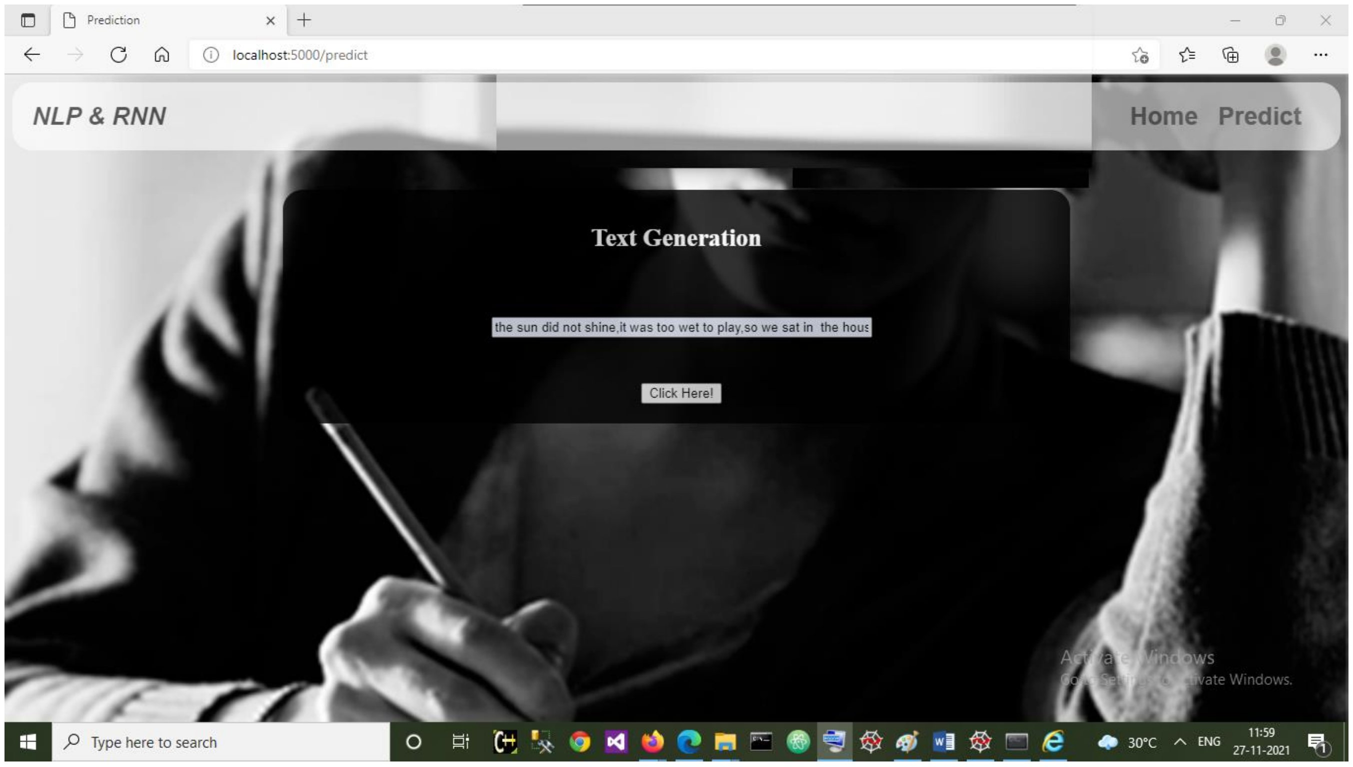


6. RESULT



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.



7. ADVANTAGES & DISADVANTAGES

Advantages:

The advantages of Natural Language Generation go beyond the usual perception that people have when it comes to AI adoption.

- Automated Content Creation (Automation of content generation &Data delivery in the expected format**)**
- Significant Reduction in Human Involvement
- Predictive Inventory Management
- Performance Activity Management at Call Centre

Disadvantages:

• How to capture long-term dependencies (beyond a brief discussion of attention) or maintain global coherence. This remains a challenge due to the curse of dimensionality as well as neural networks failing to learn more abstract concepts from the predominant next-step prediction training objective.
• How to interface models with a knowledge base, or other structured data that cannot be supplied in a short piece of text. Some recent work has used pointer mechanisms towards this end.
• Consequently, while we focus on natural language, to be precise, this guide does not cover natural language generation (NLG), which entails generating documents or longer descriptions from structured data. The primary focus is on tasks where the target is a single sentence—hence the term "text generation" as opposed to "language generation

8. APPLICATIONS

- Text Classification
- Language Modeling
- Speech Recognition
- Caption Generation
- Machine Translation
- Document Summarization
- Question Answering

9. CONCLUSION

Recurrent Neural Networks are a technique of working with sequential data, because they can remember the last inputs via an internal memory. They achieve state of the art performance on pretty much every sequential problem and are used by most major companies. A RNN can be used to generate text in the style of a specific author.

The steps of creating a text generation RNN are:

1. Creating or gathering a dataset
2. Building the RNN model
3. Creating new text by taking a random sentence as a starting point

10. FUTURE SCOPE

Few suggestions for building text generation techniques are: plan to use a deep deterministic policy gradient in the future to better train the generator. It will also choose other models such as recurrent neural networks and recurrent neural networks as discriminators.

There are also a lot of things you can improve about the model to get better outputs. A few of them are:

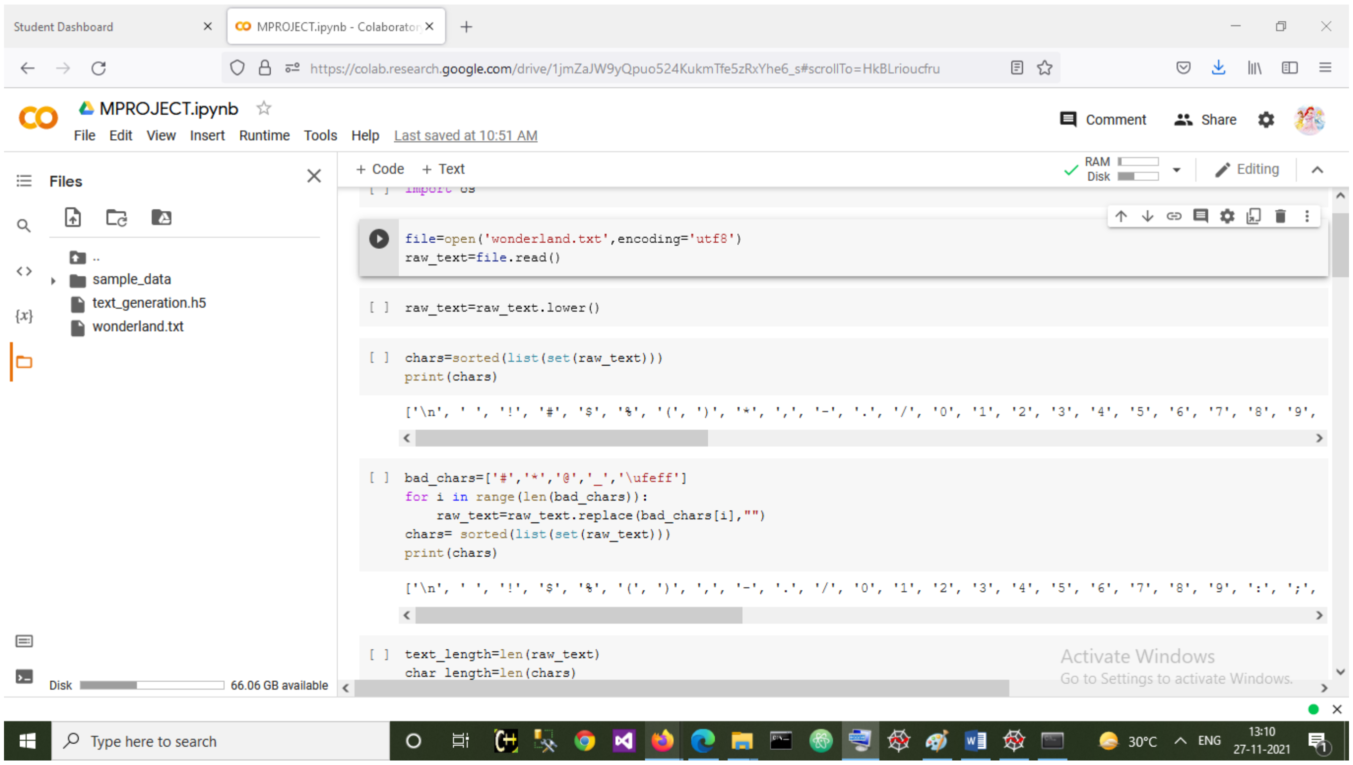
1. Using a more sophisticated network structure (more LSTM-, Dense Layers)
2. Training for more epochs
3. Playing around with the batch size

11. BIBILOGRAPHY

- https://gilberttanner.com/blog/generating-text-using-a-recurrent-
- neuralnetworkhttps://mobcoder.com/blog/natural-language-processing/
- https://towardsdatascience.com/text-generation-using-rnns-fdb03a010b9f
- https://stackabuse.com/text-generation-with-python-and-tensorflow-keras/
- https://www.tensorflow.org/text/tutorials/text_generation

APPENDIX

Source Code



```
import os
```

```
file=open('wonderland.txt',encoding='utf8')
raw_text=file.read()
```

```
raw_text=raw_text.lower()
```

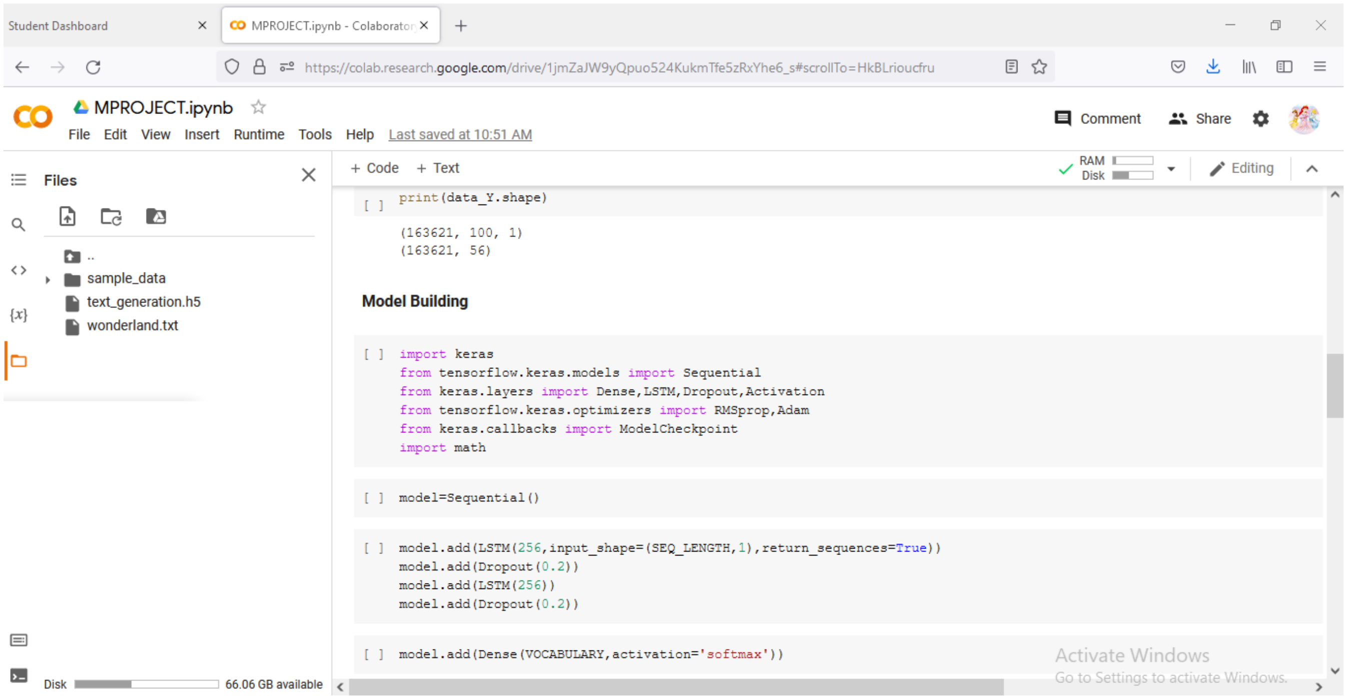
```
chars=sorted(list(set(raw_text)))
print(chars)
```

```
['\n', ' ', '!', '#', '$', '%', '(', ')', '*', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
```

```
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)
```

```
['\n', ' ', '!', '$', '%', '(', ')', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';',
```

```
text_length=len(raw_text)
char_length=len(chars)
```



```
print(data_Y.shape)
```

```
(163621, 100, 1)
(163621, 56)
```

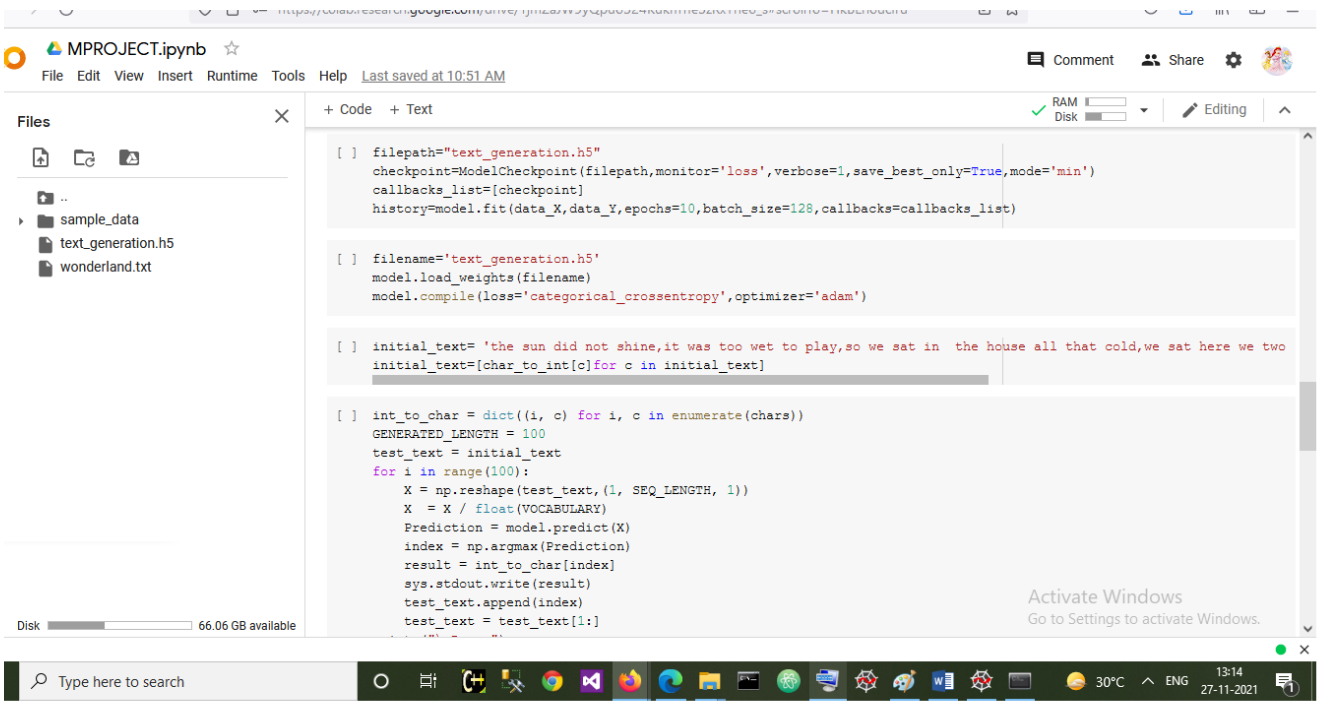
Model Building

```
import keras
from tensorflow.keras.models import Sequential
from keras.layers import Dense,LSTM,Dropout,Activation
from tensorflow.keras.optimizers import RMSprop,Adam
from keras.callbacks import ModelCheckpoint
import math
```

```
model=Sequential()
```

```
model.add(LSTM(256,input_shape=(SEQ_LENGTH,1),return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(256))
model.add(Dropout(0.2))
```

```
model.add(Dense(VOCABULARY,activation='softmax'))
```

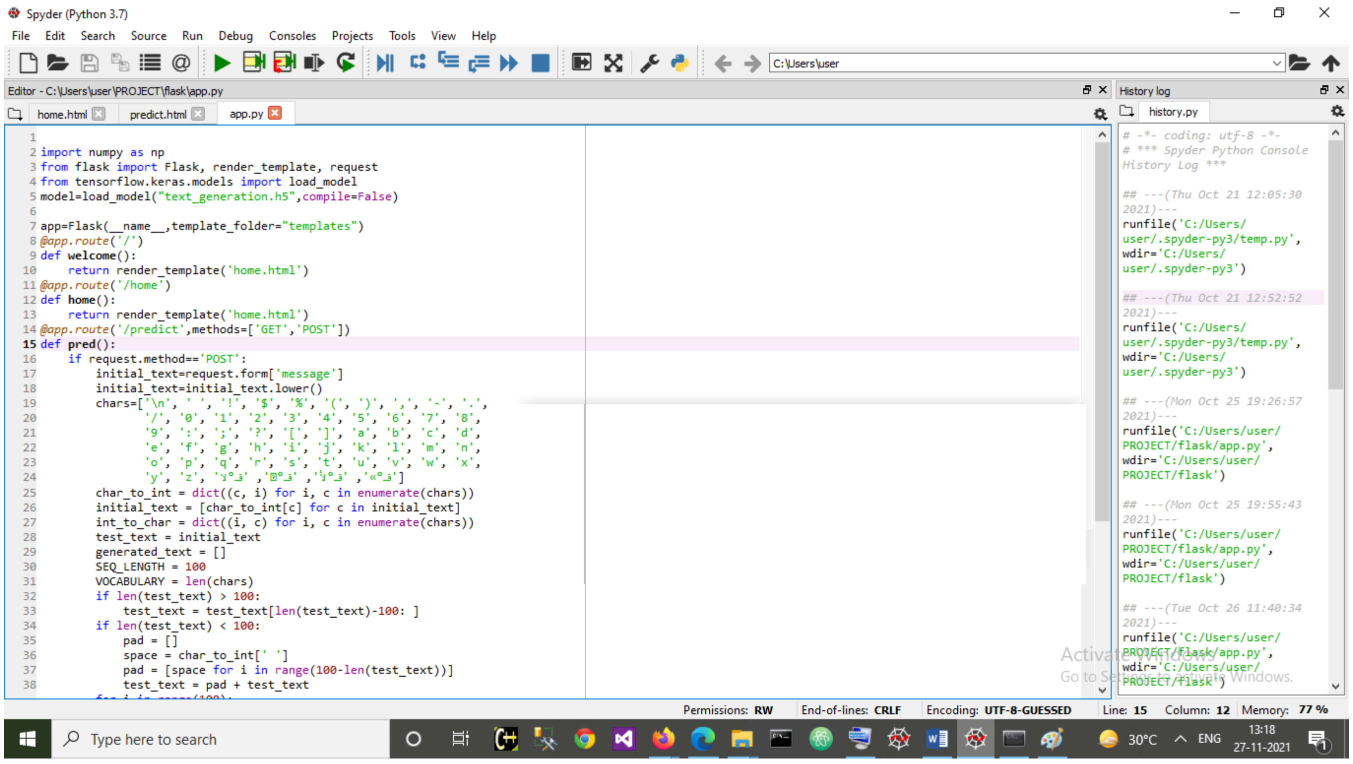

MPROJECT.ipynb

```
filepath="text_generation.h5"
checkpoint=ModelCheckpoint(filepath,monitor='loss',verbose=1,save_best_only=True,mode='min')
callbacks_list=[checkpoint]
history=model.fit(data_X,data_Y,epochs=10,batch_size=128,callbacks=callbacks_list)
```

```
filename='text_generation.h5'
model.load_weights(filename)
model.compile(loss='categorical_crossentropy',optimizer='adam')
```

```
initial_text= 'the sun did not shine,it was too wet to play,so we sat in  the house all that cold,we sat here we two
initial_text=[char_to_int[c]for c in initial_text]
```

```
int_to_char = dict((i, c) for i, c in enumerate(chars))
GENERATED_LENGTH = 100
test_text = initial_text
for i in range(100):
    X = np.reshape(test_text,(1, SEQ_LENGTH, 1))
    X  = X / float(VOCABULARY)
    Prediction = model.predict(X)
    index = np.argmax(Prediction)
    result = int_to_char[index]
    sys.stdout.write(result)
    test_text.append(index)
    test_text = test_text[1:]
```

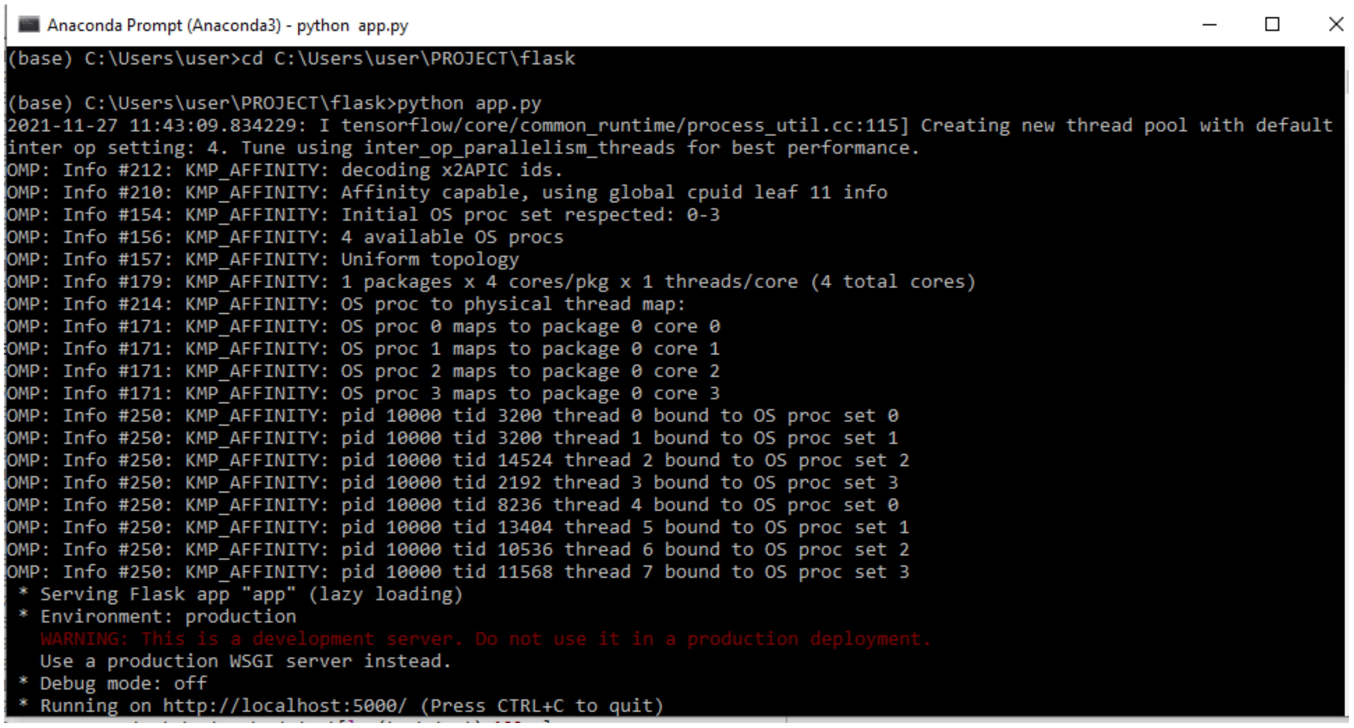

Spyder (Python 3.7)

Editor - C:\Users\user\PROJECT\flask\app.py

```

import numpy as np
from flask import Flask, render_template, request
from tensorflow.keras.models import load_model
model=load_model("text_generation.h5",compile=False)

app=Flask(__name__,template_folder="templates")
@app.route('/')
def welcome():
    return render_template('home.html')
@app.route('/home')
def home():
    return render_template('home.html')
@app.route('/predict',methods=['GET','POST'])
def pred():
    if request.method=='POST':
        initial_text=request.form['message']
        initial_text=initial_text.lower()
        chars=['\n', ' ', '!', '$', '%', '(', ')', ',', '-', '.',
               '/', '0', '1', '2', '3', '4', '5', '6', '7', '8',
               '9', ':', ';', '?', '[', ']', 'a', 'b', 'c', 'd',
               'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
               'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
               'y', 'z', 'ي°ق' ,'▯°ق' ,'ئ°ق' ,'«°ق']
        char_to_int = dict((c, i) for i, c in enumerate(chars))
        initial_text = [char_to_int[c] for c in initial_text]
        int_to_char = dict((i, c) for i, c in enumerate(chars))
        test_text = initial_text
        generated_text = []
        SEQ_LENGTH = 100
        VOCABULARY = len(chars)
        if len(test_text) > 100:
            test_text = test_text[len(test_text)-100: ]
        if len(test_text) < 100:
            pad = []
            space = char_to_int[' ']
            pad = [space for i in range(100-len(test_text))]
            test_text = pad + test_text
```

History log — history.py

```
# -*- coding: utf-8 -*-
# *** Spyder Python Console History Log ***

## ---(Thu Oct 21 12:05:30 2021)---
runfile('C:/Users/user/.spyder-py3/temp.py', wdir='C:/Users/user/.spyder-py3')

## ---(Thu Oct 21 12:52:52 2021)---
runfile('C:/Users/user/.spyder-py3/temp.py', wdir='C:/Users/user/.spyder-py3')

## ---(Mon Oct 25 19:26:57 2021)---
runfile('C:/Users/user/PROJECT/flask/app.py', wdir='C:/Users/user/PROJECT/flask')

## ---(Mon Oct 25 19:55:43 2021)---
runfile('C:/Users/user/PROJECT/flask/app.py', wdir='C:/Users/user/PROJECT/flask')

## ---(Tue Oct 26 11:40:34 2021)---
runfile('C:/Users/user/PROJECT/flask/app.py', wdir='C:/Users/user/PROJECT/flask')
```



Anaconda Prompt (Anaconda3) - python app.py

```
(base) C:\Users\user>cd C:\Users\user\PROJECT\flask

(base) C:\Users\user\PROJECT\flask>python app.py
2021-11-27 11:43:09.834229: I tensorflow/core/common_runtime/process_util.cc:115] Creating new thread pool with default
inter op setting: 4. Tune using inter_op_parallelism_threads for best performance.
OMP: Info #212: KMP_AFFINITY: decoding x2APIC ids.
OMP: Info #210: KMP_AFFINITY: Affinity capable, using global cpuid leaf 11 info
OMP: Info #154: KMP_AFFINITY: Initial OS proc set respected: 0-3
OMP: Info #156: KMP_AFFINITY: 4 available OS procs
OMP: Info #157: KMP_AFFINITY: Uniform topology
OMP: Info #179: KMP_AFFINITY: 1 packages x 4 cores/pkg x 1 threads/core (4 total cores)
OMP: Info #214: KMP_AFFINITY: OS proc to physical thread map:
OMP: Info #171: KMP_AFFINITY: OS proc 0 maps to package 0 core 0
OMP: Info #171: KMP_AFFINITY: OS proc 1 maps to package 0 core 1
OMP: Info #171: KMP_AFFINITY: OS proc 2 maps to package 0 core 2
OMP: Info #171: KMP_AFFINITY: OS proc 3 maps to package 0 core 3
OMP: Info #250: KMP_AFFINITY: pid 10000 tid 3200 thread 0 bound to OS proc set 0
OMP: Info #250: KMP_AFFINITY: pid 10000 tid 3200 thread 1 bound to OS proc set 1
OMP: Info #250: KMP_AFFINITY: pid 10000 tid 14524 thread 2 bound to OS proc set 2
OMP: Info #250: KMP_AFFINITY: pid 10000 tid 2192 thread 3 bound to OS proc set 3
OMP: Info #250: KMP_AFFINITY: pid 10000 tid 8236 thread 4 bound to OS proc set 0
OMP: Info #250: KMP_AFFINITY: pid 10000 tid 13404 thread 5 bound to OS proc set 1
OMP: Info #250: KMP_AFFINITY: pid 10000 tid 10536 thread 6 bound to OS proc set 2
OMP: Info #250: KMP_AFFINITY: pid 10000 tid 11568 thread 7 bound to OS proc set 3
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://localhost:5000/ (Press CTRL+C to quit)
```