VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

A project report on

LYRICS GENERATION USING NEURAL NETWORK

Submitted towards J component of course

Artificial Intelligence

CSE3013

For

Review 3

Under the guidance of

PROF. Vasantha WB

Submitted by

NO.	NAME	REGISTRATION NO.
1	Ujjwal Saharia	19BCE0487
2	Karan Rochlani	19BCE0383
3	Kushagra Gupta	19BCE0760
4	Suraj Kumar	19BCE0369

INTRODUCTION

Music can be shown as a sequence of events and thus it can also be modelled as conditional probabilities between various musical events. In many frameworks that are programmed, these connections are streamlined. The probability of a present state to occur relies upon the probabilities of the previous states. A succession of melodic effects such as notes, harmonies, musicality designs is made by anticipating the accompanying occasion given a seed grouping.

In a Feed-Forward neural system, the data just moves in a single heading, from the input layer, through the hidden layers to the output layer. The data moves straight through the system. Hence, the data never contacts a hub twice. In Feed-Forward Neural Networks there is no memory of the info they got already and are therefore bad in anticipating what's coming straight away. Since a feedforward organizer just thinks about the present information, it has no idea of the request in time.

Recurrent Neural Network (RNN) takes into account fusing long term dependency. In a RNN, information cycles through loops. When it settles on a choice, it mulls over the present information from the data sources it got previously. Recurrent Neural Network has two information sources, the present and the ongoing past. The grouping of information contains vital data about what is coming straightaway, which is the reason a RNN can do things different algorithms can't. RNN's apply weights to the current and previous input. They additionally change their weights for both through gradient descent and Backpropagation Through Time.

Long-Short Term Memory(LSTM) systems are an expansion for recurrent neural networks, which fundamentally broadens their memory. The units of a LSTM are utilized as structure units for the layers of a RNN called the LSTM network. LSTM empowers RNN to recollect their contributions over an extensive stretch of time. This is on the grounds that LSTM contains their data in a memory that is much similar to the memory of a PC in light of the fact that the LSTM can peruse, compose and erase data from its memory. This memory can be viewed as a gated cell. In a LSTM you have three gateways: input, forget and output gate. These gates decide if to input a gateway, either forget the gateway or the output gateway. LTSM is helpful for lyric generation as it removes the problem of vanishing gradients.

In this paper we will use Long Short-term memory(LSTM) that will help the RNN to remember about their inputs for a long period of time. Through this program(Lyrics Generator) we will try to generate lyrics for even difficult songs. We will train the neural network on lots of words to generate the lyrics of songs based on a given input word or sentence. This project can be further extended to identify other forms of music like Jazz, Rock, Pop, Metallic etc. It has other applications like Auto fillers in industries, Chatbots , Automated email generating system etc. At the end of this project we will try to develop a program with as much high accuracy as possible in order to be a successful product.

The lyrics of a song are auto generated using the concept of Neural Networks. A lyricist has a very challenging job that tests his creativity on every level. He must go through different aspects before giving his final track the complete lyrics. These aspects include choosing words that ideally rhyme with regard to the title of the song and that can give more meaning and sense in order to add to the theme of the song. Another important aspect of writing song lyrics is that it is quite difficult to create a song whose lyrics enthral the public and that can

generate a positive response in the majority of the crowd. These aspects make writing lyrics for a song hard, quite a challenging task for a singer. Keeping this problem in mind, we have decided to create a program called - Lyrics Generator. The concept of Natural Language Generation techniques are used in this program. A corpus of the desired song is given as input to the program and the program then gives its output as a song containing words from the corpus. This generator program is also capable of giving us words that rhyme with our song. We wish to simplify the task of lyricists and help them get their much needed inspiration to create the best quality of music. This project uses a dataset from Kaggle which includes Artist Specific Song Verses (The number of songs and hence the size of dataset depends on the artist specific dataset that we choose).

We have kept in mind to not let the size of the dataset become very large by including a wide number of songs in it as this will increase the training time of the program. Also, an optimizing algorithm known as Adam Optimizer is used to reduce the loss and the execution time of the model so as to gain better and faster results. We wish to create a generator program which is successfully and opportunely able to generate the desired lyrics of the song with ninety percent accuracy along with bringing melody to the music and building the theme of the song in order to touch the hearts of the public

LITERATURE REVIEW SUMMARY TABLE:

Authors and Year	Title	Concept and Framework	Methodology	Analysis	Limitations and Future Research
Eric Malmi, Pyry Takala, Hannu Toivonen, Tapani Raiko, Aristides Gionis (2016)	Dope Learning: A Computational Approach to Rap Lyrics Generation	Created a lyrics generation model using two techniques: Rank Support Vector Machine and Deep Neural Network	Next line prediction is the method used. After feature extraction, both bag-of-words and Latent semantic analysis is used to find similarity and then passed through a neural network to train.	10980 songs from 104 different artists were used as the dataset.The dataset was passed through pre-processing techniques like removal of stop words, tokenization, etc.	Since a lot of operations were done on the data, the execution time is very large and in accordance with that, the accuracy obtained is not enough.
Peter Potash, Alexey Romanov, Anna Rumshisk (2015)	GhostWriter : Using an LSTM for Automatic Rap Lyric Generation	Created a lyrics generation model using LSTM.	The dataset is passed through an LSTM model which is used to generate lyrics. The results are tested for their similarity with original ones and their rhyming density. These values are compared with a simple n-gram model to test the accuracy	Songs from 14 artists from The Original Hip-Hop (Rap) Lyrics Archive were used as dataset	The value obtained for similarity index and rhyme density comes out to be smaller than n-gram because of repetition of words. This happened because pre-processing techniques were not implemented on the data.
Olga Vechtomova, Hareesh Bahuleyan, Amirpasha Ghabussi, Vineet John (2018)	Generating lyrics with variational autoencoder and multi- modal artist embeddings	Lyrics generation using CNN classifier and Variational Autoencoder (VAE) along with LSTM.	CNN classifies artists on the basis of spectrogram images, then VAE is trained to construct lines from song lyrics from the artist embeddings obtained from CNN which uses LSTM for encoding and decoding. Finally the lyrics are decoded to generate the final lyrics.	Dataset contains songs from 7 artists, one from each genre: Art Rock, Industrial, Classic Rock, Alternative, Hard Rock, Electronic, and Psychedelic Rock making a	The accuracy is lower when compared to other models. Also, the execution time for the model is high because of Variational Autoencoding, LSTM and then finally decoding the lyrics.

				total of 34000 lines.	
Yin Fan, Xiangju Lu, Dian Li, Yuanliu Liu (2016)	Video-Based Emotion Recognition using CNN- RNN and C3D Hybrid Networks	The main aim of the paper is to find the emotion related to a particular video. RNN takes appearance features transferred by implementing CNN on particular video and C3D models appearance and motion of video.	The system is a hybrid network which has 2 core parts: RNN-CNN and C3D. Audio classifiers are also added to the system so that the system can develop the ability to differentiate between sounds to complete specific tasks. LSTM is also introduced in the system to deal with the vanishing gradient problem and later C3D structure is used as a spatial-temporal model. As far as implementation goes all the faces of video frames are extracted and aligned using a similarity transform according to the facial key points and a CNN filter is applied to filter out the mistakes of faces	AFEW 6.0 database is taken into consideration which contains 1750 video clips that were further divided into three parts: 774 for training, 383 for validation, and 593 for test.	With the use of improved models, the accuracy of the system can be improved. In this paper only 2 models are combined C3D and CNN-RNN which produces effective results, but the combination of more than 2 models the accuracy can be improved.
Malte Loller-Andersen and Björn Gambäck (2018)	Deep Learning-based Poetry Generation Given Visual Input	The main idea of the framework is to generate poetry from the given visual input. The system consists of CNN for image classification and a LSTM neural network trained on a song lyrics data set.	The basic idea of the architecture is, once the system gets the input it will go through inception for classification. Then inception will produce 5 top scoring class and if these 5 classes are below the threshold class then the input will be discarded otherwise the image is carried to the next stage where the related words are discovered related to the image using the concept called Concept Net Before displaying the poetry 2-3 steps are carried out to make the process optimal and efficient.	The dataset was collected from www.mldb.org, an online song lyrics database. To connect to the site the authors had to write a python script and sort through the HTML files of the site and find the song text, artist and album.	One thing that was clear from the output was that the system was not able to generate meaningful poetry i.e. it lacked sense. To enhance the predictions sequence to sequence model can be used instead of LSTM. To get the more accurate result, training of LTSM can be done on poetry rather than song lyrics.

OBJECTIVES OF THIS PROJECT:

1. Generating artist-like lyrics that are needed to prepare new songs
2. Making use of neural networks and corpus in order to train and generate the lyrics
3. Giving the user the facility to generate lyrics by
4. providing only one word related to his/her topic.
5. Generate a self-made rap using our model.

INNOVATION COMPONENT IN THE PROJECT:

The innovative part of this project is the use of LSTM (Long Short Term Memory). There are several reasons for this. This becomes an important factor when we use recurrent neural networks as all our previous outputs are being stored. The LSTM is when the sequence is long. However, this does not apply to our code where we use a sequence length of one. This means that, first we predict what letters generally come after others, and then, after a few iterations (epochs) we predict which words follow others. This is the innovative part of our project, as most codes implement lyrics generators using LSTMs.

WORK DONE AND IMPLEMENTATION

Collecting Data

We downloaded a few datasets from Kaggle and used them to generate the lyrics. The artist specific datasets which we used were of Justin Bieber's, Lorde's, etc. we can even use any dataset according to user specific need.

Methodology

The raw data collected first needs to be processed before we can train it. We remove all the errors of white spaces and commas. We also make a list of all the unique words that were used, and convert it to an array. Then, we encode all of doing this, like creating dummy variables, but this way of encoding gives the them using the one-hot-encoding method. There are several other methods of highest efficiency of them all.

Defining the Model Class

We define a class for our Recurrent Neural Network, containing methods to build the basic structure of our model. This includes, the input and output layers, the nonlinear function applied to the input layer and hidden layers, the forward propagation algorithm to get the final output and the linear function the Gated Recurrent Units to keep track of what words have we used up till this used in the transition from last hidden layer to the output layer. We also define points. This better predicts the next words to come.

Training the Data

We use the array of words to find the probability that one word follows another. These probabilities are used as our training dataset. And it also generates the weights for our neural network. These weights are tuned to minimise our Loss function to give the optimum values. Once this is done, we are ready to test our model on new training set data.

SOFTWARE COMPONENTS:

- Python IDE
- Jupyter Notebook
- Torch
- Kaggle
- LSTM Model

HARDWARE COMPONENTS:

- A Windows/Linux based system
- 4GB RAM
- I3 Processor

TOOLS USED:

- Jupyter Notebook
- Kaggle
- Google Colab

CODE AND OUTPUT:

Coding:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.models import Sequential

from tensorflow.keras.optimizers import Adam

from tensorflow.keras import regularizers

import tensorflow.keras.utils as ku

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import tensorflow as tf

lyric=open('/content/lorde.txt').read()

lyric[:2000]

corpus=lyric.lower().split('\n')

for i in range(40,60):

  print(corpus[i])

from google.colab import drive

drive.mount('/content/drive')

tokenizer = Tokenizer()

tokenizer.fit_on_texts(corpus)

total_words = len(tokenizer.word_index) + 1

total_words

input_sequences = []

for line in corpus:

 token_list = tokenizer.texts_to_sequences([line])[0]

 for i in range(1, len(token_list)):
```

```
    n_gram_sequence = token_list[:i+1]
    input_sequences.append(n_gram_sequence)
for i in range(20):
    print(input_sequences[i])
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len,
padding='pre'))
print(max_sequence_len)
print(input_sequences)
model = Sequential()
model.add(Embedding(1372, 160, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(200, return_sequences = True)))
model.add(Dropout(0.2))
model.add(LSTM(100))
model.add(Dense(1372/2, activation='relu', kernel_regularizer=regularizers.l2(0.001
)))
model.add(Dense(1372, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
predictors, label = input_sequences[:,:-1],input_sequences[:,-1]
label = ku.to_categorical(label, num_classes=total_words)
history = model.fit(predictors, label, epochs=50, verbose=1)
acc = history.history['accuracy']
loss = history.history['loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.title('Training accuracy')
plt.figure()
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.title('Training loss')
plt.legend()
plt.show()
model.save('lyrics_generator.h5')
```

```
from keras.models import load_model
loaded_model=load_model('lyrics_generator.h5')
import time
saved_model_path = "./{}.h5".format(int(time.time()))
model.save(saved_model_path)
!pip install tensorflow js
import tensorflow js
!tensorflowjs_converter --input_format=keras {saved_model_path} ./
model_json = model.to_json()
with open("model.json", "w") as json_file:
  json_file.write(model_json)
model.save_weights("model.h5")
print("Saved model to disk")
from keras.models import model_from_json
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights("model.h5")
print("Loaded model from disk")
loaded_model.summary()

#web app code
!pip install jupyter-dash
import plotly.express as px
from jupyter_dash import JupyterDash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
df = px.data.tips()# Build App
app = JupyterDash(__name__)

app.layout = html.Div([
```

```
    html.H1('Lyrics Generator'),
    html.Div(["Next_words: ",
              dcc.Input(id='my-input', type='number')]),
    html.Div(["Seed_text: ",
              dcc.Input(id='my-input1', type='text')]),
    html.Button('Generate Lyrics', id='show-lyrics'),
    html.Div(id='body-div')
])

@app.callback(
    Output(component_id='body-div', component_property='children'),
    Input(component_id='show-lyrics', component_property='n_clicks'),
    Input(component_id='my-input', component_property='value'),
    Input(component_id='my-input1', component_property='value'),
)
def update_output(n_clicks,input_value,input_value1):
    if n_clicks is None:
        raise PreventUpdate
    else:
        return lyrics(input_value,input_value1)
def lyrics(next_words,seed_text):
 for _ in range(next_words):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-
 1, padding='pre')
    predicted = np.argmax(loaded_model.predict(token_list, verbose=0), axis=-1)
    output_word = ""
    for word, index in tokenizer.word_index.items():
      if index == predicted:
        output_word = word
        break
    seed_text += " " + output_word
 return seed_text
```

```
app.run_server(mode='external', port=8060)

app._terminate_server_for_port("localhost", 8060)
```

Result Snapshots

Lyrics Generator

Is an NLP model that is used to create the lyrics of the words given as input using LSTM in it on Tensorflow.

1.Aim

The aim is to create a lyrics inducer for any word given as th input and output as the set of words which which will be the lyrics of the song.

2.Setup

```
[ ] from tensorflow.keras.preprocessing.sequence import pad_sequences
    from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
    from tensorflow.keras.preprocessing.text import Tokenizer
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras import regularizers
    import tensorflow.keras.utils as ku
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import tensorflow as tf
```

3. Importing the dataset

```
[ ] lyric=open('/content/lorde.txt').read()
    lyric[:2000]
```

```
'Well, summer slipped us underneath her tongue\nOur days and nights are perfumed with obsession\nHalf of my wardrobe
n your bedroom floor\nUse our eyes, throw our hands overboard I am your sweetheart psychopathic crush\nDrink up your
ments, still I can't get enough\nI overthink your p-punctuation use\nNot my fault, just a thing that my mind do A rus
the beginning\nI get caught up, just for a minute\nBut lover, you're the one to blame, all that you're doing\nCan you
r the violence?\nMegaphone to my chest Broadcast the boom boom boom boom\nAnd make 'em all dance to it\nBroadcast the
m boom boom boom\nAnd make 'em all dance to it\nBroadcast the boom boom boom boom\nAnd make 'em all dance to it\nBroa
t the boom boom boom boom\nAnd make 'em all dance to it\nBroadcast the boom boom boom boom\nAnd make 'em all dance to
Our thing progresses, I call and you come through\nBlow all my friendships to sit in hell with you\nBut we're the gre
t, they'll hang us in the Louvre…'
```

4.Pre processing

```
corpus=lyric.lower().split('\n')
for i in range(40,60):
    print(corpus[i])
```

```
watch the wasters blow the speakers
spill my guts beneath the outdoor light
it's just another graceless night
i hate the headlines and the weather
```

5.Code mounting drive

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

6.Tokenizing

```
[ ] tokenizer = Tokenizer()
    tokenizer.fit_on_texts(corpus)
    total_words = len(tokenizer.word_index) + 1
    total_words
```

```
1372
```

7.Creating Sequences

```
input_sequences = []
for line in corpus:
  token_list = tokenizer.texts_to_sequences([line])[0]
  for i in range(1, len(token_list)):
    n_gram_sequence = token_list[:i+1]
    input_sequences.append(n_gram_sequence)
```

```
[ ] for i in range(20):
        print(input_sequences[i])
```

```
[112, 196]
[112, 196, 703]
[112, 196, 703, 42]
[112, 196, 703, 42, 568]
[112, 196, 703, 42, 568, 180]
[112, 196, 703, 42, 568, 180, 569]
[22, 165]
[22, 165, 4]
[22, 165, 4, 181]
[22, 165, 4, 181, 36]
[22, 165, 4, 181, 36, 704]
[22, 165, 4, 181, 36, 704, 21]
[22, 165, 4, 181, 36, 704, 21, 705]
[706, 13]
[706, 13, 9]
[706, 13, 9, 707]
[706, 13, 9, 707, 40]
[706, 13, 9, 707, 40, 27]
[706, 13, 9, 707, 40, 27, 19]
[706, 13, 9, 707, 40, 27, 19, 439]
```

8.Padding

```
[ ]  max_sequence_len = max([len(x) for x in input_sequences])
     input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len,
     padding='pre'))
     print(max_sequence_len)
     print(input_sequences)
```

```
29
[[  0   0   0 ...   0 112 196]
 [  0   0   0 ... 112 196 703]
 [  0   0   0 ... 196 703  42]
 ...
 [  0   0   0 ... 207 140 140]
 [  0   0   0 ... 140 140 140]
 [  0   0   0 ... 140 140 140]]
```

9.Building the model

```
[ ]  model = Sequential()
     model.add(Embedding(1372, 160, input_length=max_sequence_len-1))
     model.add(Bidirectional(LSTM(200, return_sequences = True)))
     model.add(Dropout(0.2))
     model.add(LSTM(100))
     model.add(Dense(1372/2, activation='relu', kernel_regularizer=regularizers.l2(0.001
     )))
     model.add(Dense(1372, activation='softmax'))
     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
     print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 28, 160)           219520
_________________________________________________________________
bidirectional_1 (Bidirection (None, 28, 400)           577600
_________________________________________________________________
dropout_1 (Dropout)          (None, 28, 400)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               200400
_________________________________________________________________
dense_2 (Dense)              (None, 686)               69286
_________________________________________________________________
dense_3 (Dense)              (None, 1372)              942564
=================================================================
Total params: 2,009,370
Trainable params: 2,009,370
Non-trainable params: 0
_________________________________________________________________
None
```

```
None
```

```
[ ] predictors, label = input_sequences[:,:-1],input_sequences[:,-1]
    label = ku.to_categorical(label, num_classes=total_words)
```

10. Training the model

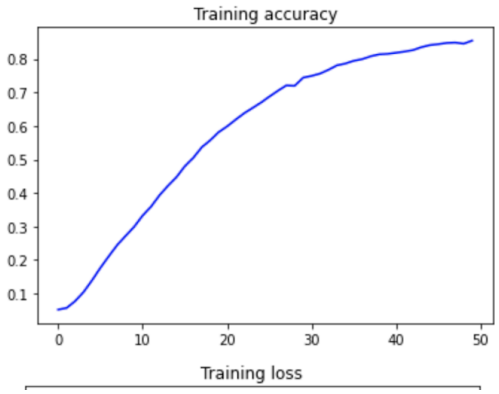
```
[ ] history = model.fit(predictors, label, epochs=50, verbose=1)
```

```
Epoch 1/50
380/380 [==============================] - 95s 237ms/step - loss: 5.9840 - accuracy: 0.0
Epoch 2/50
380/380 [==============================] - 90s 236ms/step - loss: 5.4261 - accuracy: 0.0
Epoch 3/50
380/380 [==============================] - 90s 236ms/step - loss: 5.0919 - accuracy: 0.0
Epoch 4/50
380/380 [==============================] - 90s 236ms/step - loss: 4.7865 - accuracy: 0.1
Epoch 5/50
380/380 [==============================] - 90s 236ms/step - loss: 4.5150 - accuracy: 0.1
Epoch 6/50
380/380 [==============================] - 90s 237ms/step - loss: 4.2447 - accuracy: 0.1
Epoch 7/50
380/380 [==============================] - 90s 237ms/step - loss: 4.0093 - accuracy: 0.2
Epoch 8/50
380/380 [==============================] - 90s 236ms/step - loss: 3.7785 - accuracy: 0.2
Epoch 9/50
380/380 [==============================] - 90s 236ms/step - loss: 3.5769 - accuracy: 0.2
Epoch 10/50
380/380 [==============================] - 90s 236ms/step - loss: 3.3912 - accuracy: 0.2
Epoch 11/50
380/380 [==============================] - 90s 237ms/step - loss: 3.1986 - accuracy: 0.3
Epoch 12/50
380/380 [==============================] - 90s 236ms/step - loss: 3.0489 - accuracy: 0.3
Epoch 13/50
```

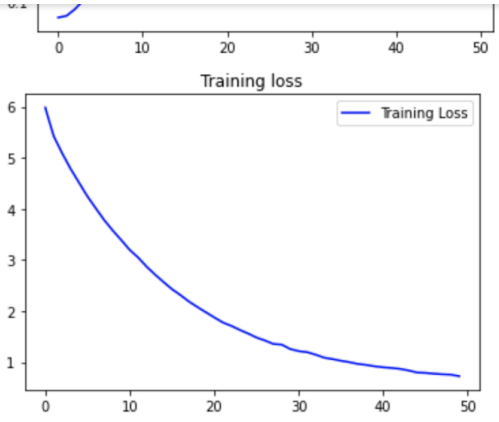
```
380/380 [==============================] - 90s 236ms/step - loss: 1.1479 - accuracy: 0.7674
Epoch 34/50
380/380 [==============================] - 90s 238ms/step - loss: 1.0898 - accuracy: 0.7808
Epoch 35/50
380/380 [==============================] - 90s 238ms/step - loss: 1.0618 - accuracy: 0.7861
Epoch 36/50
380/380 [==============================] - 90s 237ms/step - loss: 1.0290 - accuracy: 0.7944
Epoch 37/50
380/380 [==============================] - 90s 237ms/step - loss: 1.0002 - accuracy: 0.7994
Epoch 38/50
380/380 [==============================] - 91s 239ms/step - loss: 0.9675 - accuracy: 0.8077
Epoch 39/50
380/380 [==============================] - 91s 239ms/step - loss: 0.9470 - accuracy: 0.8136
Epoch 40/50
380/380 [==============================] - 90s 237ms/step - loss: 0.9207 - accuracy: 0.8150
Epoch 41/50
380/380 [==============================] - 90s 236ms/step - loss: 0.9006 - accuracy: 0.8184
Epoch 42/50
380/380 [==============================] - 90s 237ms/step - loss: 0.8872 - accuracy: 0.8219
Epoch 43/50
380/380 [==============================] - 90s 236ms/step - loss: 0.8691 - accuracy: 0.8261
Epoch 44/50
380/380 [==============================] - 90s 236ms/step - loss: 0.8389 - accuracy: 0.8350
Epoch 45/50
380/380 [==============================] - 90s 236ms/step - loss: 0.7994 - accuracy: 0.8413
Epoch 46/50
380/380 [==============================] - 89s 235ms/step - loss: 0.7918 - accuracy: 0.8439
Epoch 47/50
380/380 [==============================] - 89s 235ms/step - loss: 0.7760 - accuracy: 0.8476
Epoch 48/50
380/380 [==============================] - 89s 235ms/step - loss: 0.7640 - accuracy: 0.8487
Epoch 49/50
380/380 [==============================] - 90s 237ms/step - loss: 0.7567 - accuracy: 0.8453
Epoch 50/50
380/380 [==============================] - 90s 236ms/step - loss: 0.7266 - accuracy: 0.8543
```

11.Analysing the results

```
acc = history.history['accuracy']
loss = history.history['loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.title('Training accuracy')
plt.figure()
plt.plot(epochs, loss, 'b', label='Training Loss')
plt.title('Training loss')
plt.legend()
plt.show()
```



[ ]



12.Saving the model

```
[ ] model.save('lyrics_generator.h5')
```

13.Loading the model

```
[ ] from keras.models import load_model
    loaded_model=load_model('lyrics_generator.h5')
```

Saving the model in json

```
[ ] import time
    saved_model_path = "./{}.h5".format(int(time.time()))
    model.save(saved_model_path)
```

```
[ ] !pip install tensorflowjs
```

```
[ ] import tensorflowjs
```

```
[ ] !tensorflowjs_converter --input_format=keras {saved_model_path} ./
```

```
2021-05-30 16:24:11.786981: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynam
```

```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("model.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

Loading the model from json

```
from keras.models import model_from_json
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights("model.h5")
print("Loaded model from disk")
```

```
Loaded model from disk
```

```
loaded_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 28, 160)           219520
_________________________________________________________________
bidirectional (Bidirectional (None, 28, 400)           577600
_________________________________________________________________
dropout (Dropout)            (None, 28, 400)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               200400
_________________________________________________________________
dense (Dense)                (None, 686)               69286
_________________________________________________________________
dense_1 (Dense)              (None, 1372)              942564
=================================================================
Total params: 2,009,370
Trainable params: 2,009,370
Non-trainable params: 0
_________________________________________________________________
```

Testing the model

```
[22] next_words = 50
     seed_text = "i am a king"
```

```
def lyrics(next_words,seed_text):
  for _ in range(next_words):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-
  1, padding='pre')
    predicted = np.argmax(loaded_model.predict(token_list, verbose=0), axis=-1)
    output_word = ""
    for word, index in tokenizer.word_index.items():
      if index == predicted:
        output_word = word
        break
    seed_text += " " + output_word
  print(seed_text)
```

```
[27] lyrics(next_words,seed_text)
```

```
'i am a king in our dreams but everybody's like a fire beneath our love is a ghost lead the violence boys the radio up we
had and sing and it it it it it didn't love it it it it it it alone it kissed a level beneath the fun go it it'
```

Web app creation

```
[15] !pip install jupyter-dash
```

```
[16] import plotly.express as px
     from jupyter_dash import JupyterDash
     import dash_core_components as dcc
     import dash_html_components as html
     from dash.dependencies import Input, Output
```

```
[25] df = px.data.tips()# Build App
     app = JupyterDash(__name__)

     app.layout = html.Div([
         html.H1('Lyrics Generator'),
         html.Div(["Next_words: ",
                   dcc.Input(id='my-input', type='number')]),
         html.Div(["Seed_text: ",
                   dcc.Input(id='my-input1', type='text')]),
         html.Button('Generate Lyrics', id='show-lyrics'),
         html.Div(id='body-div')
     ])

     @app.callback(
         Output(component_id='body-div', component_property='children'),
         Input(component_id='show-lyrics', component_property='n_clicks'),
         Input(component_id='my-input', component_property='value'),
         Input(component_id='my-input1', component_property='value'),
     )
```

```
def update_output(n_clicks,input_value,input_value1):
    if n_clicks is None:
        raise PreventUpdate
    else:
        return lyrics(input_value,input_value1)
def lyrics(next_words,seed_text):
  for _ in range(next_words):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-
  1, padding='pre')
    predicted = np.argmax(loaded_model.predict(token_list, verbose=0), axis=-1)
    output_word = ""
    for word, index in tokenizer.word_index.items():
      if index == predicted:
        output_word = word
        break
    seed_text += " " + output_word
  return seed_text
```

```
[28] app.run_server(mode='external', port=8060)
```

```
Dash app running on:
http://127.0.0.1:8060/
```

```
[ ] app._terminate_server_for_port("localhost", 8060)
```

WEB APP IMPLEMENTATION

RESULT AND DISCUSSION:

So basically through this project we discussed the use and advantages of Long Short-term memory that will help the RNN to remember about their inputs for a long period of time. Through this program(Lyrics Generator) we also tried to generate lyrics for even difficult songs. We will train the neural network on lots of words to generate the lyrics of songs based on a given input word or sentence. At the end of this project we have tried to develop a program with as much high accuracy as possible in order to be a successful product.

CONCLUSION:

The lyrics generation was successfully implemented using Recursive Neural Network along with LSTM. The dataset taken was enough to train the model and generate a good quality of rap lyrics. The accuracy of the model was increased by the use of Adam Optimizer which also reduced the loss caused during the training. A self-generated rap was created using this model. The future scope of this model could involve providing it with a web interface that makes it easier for the artist to generate the required lyrics. Also, a melody can be designed using the generated lyrics by making use of some packages from python library. Another scope can include generating paragraph wise lyrics so that the artist can add various themes to different verses in his lyrics.

REFERENCES:

1. E. Malmi, P. Takala, H. Toivonen, T. Raiko, and A. Gionis, “Dopelearning: A computational approach to rap lyrics generation,” in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 195–204, 2016.

2. P. Potash, A. Romanov, and A. Rumshisky, “Ghostwriter: Using an lstm for automatic rap lyric generation,” in Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 1919–1924, 2015.

3. O. Vechtomova, H. Bahuleyan, A. Ghabussi, and V. John, “Generating lyrics with variational autoencoder and multi-modal artist embeddings,” arXiv preprint arXiv:1812.08318, 2018.

4. Y. Fan, X. Lu, D. Li, and Y. Liu, “Video-based emotion recognition using cnn-rnn and c3d hybrid networks,” in Proceedings of the 18th ACM International Conference on Multimodal Interaction, pp. 445–450, 2016.

5. M. Loller-Andersen and B. Gamb¨ack, “Deep learning-based poetry generation given visual input.,” in ICCC, pp. 240–247, 2018.

6. A. C. T. Fernandez, K. J. M. Tarnate, and M. Devaraj, “Deep rapping: Character level neural models for automated rap lyrics composition,” International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN, pp. 2278–3075, 2018.

7. A. Lambert, T. Weyde, and N. Armstrong, “Perceiving and predicting expressive rhythm with recurrent neural networks,” in Proceedings of the 12th International Conference in Sound and Music Computing, SMC 2015, SMC15, 2015.

8. B. F. Bhaukaurally, M. H. A. Didorally, and S. Pudaruth, “A semi-automated lyrics generation tool for mauritian sega,” IAES International Journal of Artificial Intelligence (IJ-AI), vol. 1, no. 4, 2012.

9. J. P. Mahedero, A. Martinez, P. Cano, M. Koppenberger, and F. Gouyon, “Natural language processing of lyrics,” in Proceedings of the 13th annual ACM international conference on Multimedia, pp. 475–478, 2005.