# Question 3: Lyrics Generation using RNN

# Importing Dependencies

import tensorflow as tf  
import numpy as np  
import os  
import time  
import pandas as pd

# Saving lyrics in a text file

df = pd.read\_csv('LYRICS\_DATASET.csv')  
df.head()

---------------------------------------------------------------------------  
NameError Traceback (most recent call last)  
<ipython-input-1-1073ba51982d> in <module>()  
----> 1 df = pd.read\_csv('LYRICS\_DATASET.csv')  
 2 df.head()  
  
NameError: name 'pd' is not defined

df['Lyrics'].to\_csv(r'all\_lyrics.txt', header=None, index=None, sep='\n', mode='a')

# Part 1

### Reading all\_lyrics.txt

path\_to\_file = 'lyrics\_dataset.txt'

text = open(path\_to\_file, 'rb').read().decode(encoding='utf-8')  
  
print ('{} characters'.format(len(text)))

389962 characters

print(text[:50])

I hate you for what you did  
And I miss you like a

### Getting a list of unique characters

vocab = sorted(set(text))  
print ('{} unique characters'.format(len(vocab)))

97 unique characters

#### Pre-processing

The model doesn't understand characters. So we need to convert it into **numbers**. Here we'll going to build a dictionary for character=>number and number=>character.

char2idx = {u:i for i, u in enumerate(vocab)}  
idx2char = np.array(vocab)  
  
text\_as\_int = np.array([char2idx[c] for c in text])

for char,\_ in zip(char2idx, range(20)):  
 print(' {:4s}: {:3d},'.format(repr(char), char2idx[char]))

'\n': 0,  
 ' ' : 1,  
 '!' : 2,  
 '"' : 3,  
 '&' : 4,  
 "'" : 5,  
 '(' : 6,  
 ')' : 7,  
 '\*' : 8,  
 ',' : 9,  
 '-' : 10,  
 '.' : 11,  
 '/' : 12,  
 '0' : 13,  
 '1' : 14,  
 '2' : 15,  
 '3' : 16,  
 '4' : 17,  
 '5' : 18,  
 '6' : 19,

Let's say we want to map this string to intergers:

I hate you for what

print ('{} {}'.format(repr(text[:20]), text\_as\_int[:20]))

'I hate you for what ' [34 1 60 53 72 57 1 77 67 73 1 58 67 70 1 75 60 53 72 1]

### Creating training examples and targets

Here we're going to break the text into sequence of seq\_length + 1 , for example let' say the string is Hello:

* Then input sequence becomes : "Hell"
* The output sequence becomes : "ello"

seq\_length = 100  
examples\_per\_epoch = len(text)//(seq\_length+1)  
  
char\_dataset = tf.data.Dataset.from\_tensor\_slices(text\_as\_int)  
  
for i in char\_dataset.take(5):  
 print(idx2char[i.numpy()] , end = "")

I hat

sequences = char\_dataset.batch(seq\_length+1, drop\_remainder=True)  
  
for item in sequences.take(5):  
 print(repr(''.join(idx2char[item.numpy()])))

"I hate you for what you did\nAnd I miss you like a little kid\nI faked it every time\nBut that's alright"  
'\nI can hardly feel anything\nI hardly feel anything at all\nYou gave me fifteen hundred\nTo see your hyp'  
'notherapist\nI only went one time\nYou let it slide\nFell on hard times a year ago\nWas hoping you would '  
'let it go, and you did\nI have emotional motion sickness\nSomebody roll the windows down\nThere are no w'  
"ords in the English language\nI could scream to drown you out\nI'm on the outside looking through\nYou'r"

### Mapping function

This function will split a string into input and target format:

def split\_input\_target(chunk):  
 input\_text = chunk[:-1]  
 target\_text = chunk[1:]  
 return input\_text, target\_text  
  
dataset = sequences.map(split\_input\_target)

### Creating training batches

# Batch size  
BATCH\_SIZE = 64  
BUFFER\_SIZE = 10000  
dataset = dataset.shuffle(BUFFER\_SIZE).batch(BATCH\_SIZE, drop\_remainder=True)  
dataset

<BatchDataset shapes: ((64, 100), (64, 100)), types: (tf.int64, tf.int64)>

vocab\_size = len(vocab)  
  
embedding\_dim = 256  
  
rnn\_units = 1500

### Define the model

**1. Embedding layer :** The input layer. A trainable lookup table that will map the numbers of each character to a vector with embedding\_dim dimensions

**2. GRU layer :** A type of RNN with size units=rnn\_units (LSTM could also be used here.)

**3. Dense layer :** The output layer, with vocab\_size outputs and 'RELU' as the activation fuction

**4. Dropout layer :** Benifits regularisation and prevents overfitting

def build\_model(vocab\_size, embedding\_dim, rnn\_units, batch\_size):  
 model = tf.keras.Sequential([  
   
 tf.keras.layers.Embedding(vocab\_size, embedding\_dim,  
 batch\_input\_shape=[batch\_size, None]),  
   
 tf.keras.layers.GRU(rnn\_units,  
 return\_sequences=True,  
 stateful=True,  
 recurrent\_initializer='glorot\_uniform'),  
  
 tf.keras.layers.Dense(vocab\_size,activation='relu'),  
   
 tf.keras.layers.Dropout(0.2),  
 ])  
 return model

model = build\_model(  
 vocab\_size = len(vocab),  
 embedding\_dim=embedding\_dim,  
 rnn\_units=rnn\_units,  
 batch\_size=BATCH\_SIZE)

for input\_example\_batch, target\_example\_batch in dataset.take(1):  
 example\_batch\_predictions = model(input\_example\_batch)  
 print(example\_batch\_predictions.shape, "# (batch\_size, sequence\_length, vocab\_size)")

(64, 100, 97) # (batch\_size, sequence\_length, vocab\_size)

model.summary()

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param #   
=================================================================  
 embedding (Embedding) (64, None, 256) 24832   
   
 gru (GRU) (64, None, 1500) 7911000   
   
 dense (Dense) (64, None, 97) 145597   
   
 dropout (Dropout) (64, None, 97) 0   
   
=================================================================  
Total params: 8,081,429  
Trainable params: 8,081,429  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

def loss(labels, logits):  
 return tf.keras.losses.sparse\_categorical\_crossentropy(labels, logits, from\_logits=True)  
  
example\_batch\_loss = loss(target\_example\_batch, example\_batch\_predictions)  
print("Prediction shape: ", example\_batch\_predictions.shape, " # (batch\_size, sequence\_length, vocab\_size)")  
print("scalar\_loss: ", example\_batch\_loss.numpy().mean())

Prediction shape: (64, 100, 97) # (batch\_size, sequence\_length, vocab\_size)  
scalar\_loss: 4.574761

model.compile(optimizer='adam', loss=loss)

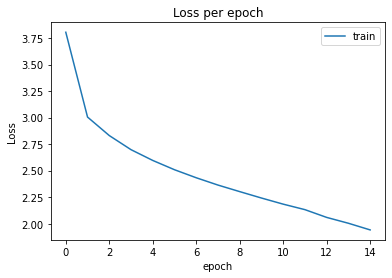
### Training

EPOCHS=15

history = model.fit(dataset, epochs=EPOCHS)

Epoch 1/15  
60/60 [==============================] - 16s 226ms/step - loss: 3.8047  
Epoch 2/15  
60/60 [==============================] - 14s 230ms/step - loss: 3.0052  
Epoch 3/15  
60/60 [==============================] - 14s 226ms/step - loss: 2.8308  
Epoch 4/15  
60/60 [==============================] - 14s 230ms/step - loss: 2.6977  
Epoch 5/15  
60/60 [==============================] - 14s 229ms/step - loss: 2.5962  
Epoch 6/15  
60/60 [==============================] - 14s 226ms/step - loss: 2.5086  
Epoch 7/15  
60/60 [==============================] - 14s 230ms/step - loss: 2.4329  
Epoch 8/15  
60/60 [==============================] - 15s 230ms/step - loss: 2.3640  
Epoch 9/15  
60/60 [==============================] - 14s 231ms/step - loss: 2.3023  
Epoch 10/15  
60/60 [==============================] - 14s 228ms/step - loss: 2.2419  
Epoch 11/15  
60/60 [==============================] - 14s 227ms/step - loss: 2.1842  
Epoch 12/15  
60/60 [==============================] - 14s 230ms/step - loss: 2.1323  
Epoch 13/15  
60/60 [==============================] - 15s 231ms/step - loss: 2.0592  
Epoch 14/15  
60/60 [==============================] - 14s 226ms/step - loss: 2.0040  
Epoch 15/15  
60/60 [==============================] - 14s 229ms/step - loss: 1.9413

import matplotlib.pyplot as plt  
plt.plot(history.history['loss'])  
plt.title('Loss per epoch')  
plt.ylabel('Loss')  
plt.xlabel('epoch')  
plt.legend(['train'], loc='upper right')  
plt.show()



### Part B: Trying different loss functions

### 1. Sparse Categorical Crossentropy

In part A, we use sparse\_categorical\_crossentropy as loss function.

**loss = 1.2670**

### 2. Mean Squared Error(MSE)

**loss = 6.6186**

model = build\_model(  
 vocab\_size = len(vocab),  
 embedding\_dim=embedding\_dim,  
 rnn\_units=rnn\_units,  
 batch\_size=BATCH\_SIZE)

model.compile(optimizer='adam', loss='mean\_squared\_error')

history = model.fit(dataset, epochs=EPOCHS)

Epoch 1/15  
60/60 [==============================] - 17s 251ms/step - loss: 5.6347  
Epoch 2/15  
60/60 [==============================] - 15s 248ms/step - loss: 5.3683  
Epoch 3/15  
60/60 [==============================] - 15s 248ms/step - loss: 5.4342  
Epoch 4/15  
60/60 [==============================] - 16s 249ms/step - loss: 5.4868  
Epoch 5/15  
60/60 [==============================] - 16s 249ms/step - loss: 6.8124  
Epoch 6/15  
60/60 [==============================] - 15s 247ms/step - loss: 6.5915  
Epoch 7/15  
60/60 [==============================] - 15s 248ms/step - loss: 6.5700  
Epoch 8/15  
60/60 [==============================] - 15s 248ms/step - loss: 6.5898  
Epoch 9/15  
60/60 [==============================] - 15s 246ms/step - loss: 6.5908  
Epoch 10/15  
60/60 [==============================] - 15s 246ms/step - loss: 6.6023  
Epoch 11/15  
60/60 [==============================] - 15s 246ms/step - loss: 6.6229  
Epoch 12/15  
60/60 [==============================] - 15s 245ms/step - loss: 6.6020  
Epoch 13/15  
60/60 [==============================] - 15s 247ms/step - loss: 6.6196  
Epoch 14/15  
60/60 [==============================] - 15s 246ms/step - loss: 6.5961  
Epoch 15/15  
60/60 [==============================] - 15s 247ms/step - loss: 6.6186

### 3. Mean Absolute Error(MAE)

**loss = 1.2670**

model = build\_model(  
 vocab\_size = len(vocab),  
 embedding\_dim=embedding\_dim,  
 rnn\_units=rnn\_units,  
 batch\_size=BATCH\_SIZE)

model.compile(optimizer='adam', loss='mean\_absolute\_error')

history = model.fit(dataset, epochs=EPOCHS)

Epoch 1/15  
60/60 [==============================] - 17s 251ms/step - loss: 6.0171  
Epoch 2/15  
60/60 [==============================] - 15s 248ms/step - loss: 5.5608  
Epoch 3/15  
60/60 [==============================] - 16s 250ms/step - loss: 5.3678  
Epoch 4/15  
60/60 [==============================] - 15s 248ms/step - loss: 5.5201  
Epoch 5/15  
60/60 [==============================] - 16s 248ms/step - loss: 5.6483  
Epoch 6/15  
60/60 [==============================] - 15s 246ms/step - loss: 5.5909  
Epoch 7/15  
60/60 [==============================] - 15s 246ms/step - loss: 5.4458  
Epoch 8/15  
60/60 [==============================] - 15s 245ms/step - loss: 5.5188  
Epoch 9/15  
60/60 [==============================] - 15s 246ms/step - loss: 5.5100  
Epoch 10/15  
60/60 [==============================] - 15s 248ms/step - loss: 5.6828  
Epoch 11/15  
60/60 [==============================] - 15s 246ms/step - loss: 5.5550  
Epoch 12/15  
60/60 [==============================] - 15s 247ms/step - loss: 5.6820  
Epoch 13/15  
60/60 [==============================] - 15s 248ms/step - loss: 5.5684  
Epoch 14/15  
60/60 [==============================] - 15s 248ms/step - loss: 5.5939  
Epoch 15/15  
60/60 [==============================] - 15s 248ms/step - loss: 5.5650

### Trying different optimizer

### 1. Adam

model = build\_model(  
 vocab\_size = len(vocab),  
 embedding\_dim=embedding\_dim,  
 rnn\_units=rnn\_units,  
 batch\_size=BATCH\_SIZE)

model.compile(optimizer='adam', loss=loss)

history = model.fit(dataset, epochs=EPOCHS)

Epoch 1/15  
60/60 [==============================] - 17s 244ms/step - loss: 3.8536  
Epoch 2/15  
60/60 [==============================] - 15s 244ms/step - loss: 2.9930  
Epoch 3/15  
60/60 [==============================] - 15s 244ms/step - loss: 2.8297  
Epoch 4/15  
60/60 [==============================] - 15s 244ms/step - loss: 2.6899  
Epoch 5/15  
60/60 [==============================] - 15s 244ms/step - loss: 2.5896  
Epoch 6/15  
60/60 [==============================] - 15s 243ms/step - loss: 2.5030  
Epoch 7/15  
60/60 [==============================] - 15s 243ms/step - loss: 2.4300  
Epoch 8/15  
60/60 [==============================] - 15s 243ms/step - loss: 2.3572  
Epoch 9/15  
60/60 [==============================] - 15s 244ms/step - loss: 2.2932  
Epoch 10/15  
60/60 [==============================] - 15s 243ms/step - loss: 2.2325  
Epoch 11/15  
60/60 [==============================] - 15s 243ms/step - loss: 2.1666  
Epoch 12/15  
60/60 [==============================] - 15s 242ms/step - loss: 2.1070  
Epoch 13/15  
60/60 [==============================] - 15s 242ms/step - loss: 2.0527  
Epoch 14/15  
60/60 [==============================] - 15s 243ms/step - loss: 1.9859  
Epoch 15/15  
60/60 [==============================] - 15s 243ms/step - loss: 1.9184

### 2. SGD

model = build\_model(  
 vocab\_size = len(vocab),  
 embedding\_dim=embedding\_dim,  
 rnn\_units=rnn\_units,  
 batch\_size=BATCH\_SIZE)

model.compile(optimizer='SGD', loss=loss)

history = model.fit(dataset, epochs=EPOCHS)

Epoch 1/15  
60/60 [==============================] - 17s 241ms/step - loss: 4.5277  
Epoch 2/15  
60/60 [==============================] - 15s 241ms/step - loss: 4.4182  
Epoch 3/15  
60/60 [==============================] - 15s 240ms/step - loss: 4.2954  
Epoch 4/15  
60/60 [==============================] - 15s 239ms/step - loss: 4.1525  
Epoch 5/15  
60/60 [==============================] - 15s 239ms/step - loss: 4.0031  
Epoch 6/15  
60/60 [==============================] - 15s 239ms/step - loss: 3.8829  
Epoch 7/15  
60/60 [==============================] - 15s 239ms/step - loss: 3.7986  
Epoch 8/15  
60/60 [==============================] - 15s 240ms/step - loss: 3.7425  
Epoch 9/15  
60/60 [==============================] - 15s 239ms/step - loss: 3.7182  
Epoch 10/15  
60/60 [==============================] - 15s 239ms/step - loss: 3.7059  
Epoch 11/15  
60/60 [==============================] - 15s 241ms/step - loss: 3.6980  
Epoch 12/15  
60/60 [==============================] - 15s 239ms/step - loss: 3.6984  
Epoch 13/15  
60/60 [==============================] - 15s 239ms/step - loss: 3.6976  
Epoch 14/15  
60/60 [==============================] - 15s 239ms/step - loss: 3.6969  
Epoch 15/15  
60/60 [==============================] - 15s 239ms/step - loss: 3.6936

### 3. Nadam

model = build\_model(  
 vocab\_size = len(vocab),  
 embedding\_dim=embedding\_dim,  
 rnn\_units=rnn\_units,  
 batch\_size=BATCH\_SIZE)

model.compile(optimizer='Nadam', loss=loss)

history = model.fit(dataset, epochs=EPOCHS)

Epoch 1/15  
60/60 [==============================] - 18s 251ms/step - loss: 4.1755  
Epoch 2/15  
60/60 [==============================] - 16s 250ms/step - loss: 3.4419  
Epoch 3/15  
60/60 [==============================] - 16s 251ms/step - loss: 2.9531  
Epoch 4/15  
60/60 [==============================] - 16s 249ms/step - loss: 2.7780  
Epoch 5/15  
60/60 [==============================] - 16s 249ms/step - loss: 2.5475  
Epoch 6/15  
60/60 [==============================] - 16s 249ms/step - loss: 2.4509  
Epoch 7/15  
60/60 [==============================] - 16s 251ms/step - loss: 2.3709  
Epoch 8/15  
60/60 [==============================] - 16s 249ms/step - loss: 2.3048  
Epoch 9/15  
60/60 [==============================] - 16s 249ms/step - loss: 2.2328  
Epoch 10/15  
60/60 [==============================] - 16s 249ms/step - loss: 2.1723  
Epoch 11/15  
60/60 [==============================] - 16s 250ms/step - loss: 2.1030  
Epoch 12/15  
60/60 [==============================] - 16s 249ms/step - loss: 2.0375  
Epoch 13/15  
60/60 [==============================] - 16s 250ms/step - loss: 1.9693  
Epoch 14/15  
60/60 [==============================] - 16s 251ms/step - loss: 1.8957  
Epoch 15/15  
60/60 [==============================] - 16s 249ms/step - loss: 1.8315

### Part C: Generating new lyrics

def generate\_text(model, chars\_to\_generate , temp , start\_string):  
 # Evaluation step (generating text using the learned model)  
  
 # Number of characters to generate  
 num\_generate = chars\_to\_generate  
  
 # Converting our start string to numbers (vectorizing)  
 input\_eval = [char2idx[s] for s in start\_string]  
 input\_eval = tf.expand\_dims(input\_eval, 0)  
  
 # Empty string to store our results  
 text\_generated = []  
  
 # Low temperatures results in more predictable text.  
 # Higher temperatures results in more surprising text.  
 # Experiment to find the best setting.  
 temperature = temp  
  
 # Here batch size == 1  
 model.reset\_states()  
 for i in range(num\_generate):  
 predictions = model(input\_eval)  
 # remove the batch dimension  
 predictions = tf.squeeze(predictions, 0)  
  
 # using a categorical distribution to predict the character returned by the model  
 predictions = predictions / temperature  
 predicted\_id = tf.random.categorical(predictions, num\_samples=1)[-1,0].numpy()  
  
 # We pass the predicted character as the next input to the model  
 # along with the previous hidden state  
 input\_eval = tf.expand\_dims([predicted\_id], 0)  
  
 text\_generated.append(idx2char[predicted\_id])  
  
 return (start\_string + ''.join(text\_generated))

from numpy import arange  
  
# Number of characters to generate (keep between 250 to 500)  
chars\_to\_generate = 500   
  
# Printing the generated text  
# Temperature 1.0 gives the craziest output and 0.1 gives the lowest varience  
# Keeping the temperature 0.35 gives best meaningful / coherent text.  
  
# Give the seed string as the first word of generate text  
print(generate\_text(model , chars\_to\_generate , 0.35 , start\_string=u"Baby "))  
  
# Uncomment below to check the variences ==>  
  
# for i in arange(0.1,1.1,0.1):  
# print("==============")  
# print("FOR TEMP : {} ".format(i))  
# print("==============")  
# print(generate\_text(model , chars\_to\_generate , i , start\_string=u"Love "))  
# print()

Baby out to the sturt gives to make it because I believe in us  
Tell me how long  
There's a fire too much  
I think I love you better now  
I'm in love with your body  
  
  
Oh honey  
I see the stars are the floor  
Masier the summer  
That you can see the stars that we say  
  
  
But I will never be the same  
  
  
But the fell in love with your body  
Oh many times, how many times, how much I wanna see you now  
I see the sumeching back a shoulders  
I'm sunch in the back of my through  
The sun line the lovers we need  
Like the sta