Business Requirements Document

AI Music Lyrics Composition Assistant

Author: Uche Othniel Osakwe

Table of Contents

1 Executive Summary

This Business Requirements Document (BRD) outlines the requirements for the AI Music Lyrics composition assistant. It contains both functional and non-functional requirements, an overview of the current status, as well as the proposed process once the solution is implemented. It is used to determine what needs to be done, and as a starting point for solution design.

2 Project Description

“Writer’s Block” as the name implies, is one major pitfall that most creative today face while creating literature, poems, movie scripts, and song lyrics. It takes a lot of time, effort and motivation to get all the ideas from the mind, to pen and paper before any production can be done.

Previously creatives have been known to hide the fact that they hire ghost writers in order to help create lyrics and writeups so as to stay consistent in the industry, and sometimes if there is no synergy the creatives tend to drift off their original styles and brand due to external influences in the creative process.

After these two facts have been established, I decided to carry out this project in order to build a user interface in which creatives can feed their previous compositions and write ups of whatever kind or media into an AI text generator model, and the machine will be able to predict and generate ideas in form of texts and sentences in order to assist the creatives in reproducing those billboard topping music composition, consistently. Consistency is the key to success in the entertainment industry!

3 Project Scope

3.1 In Scope

The following areas are in scope for this project:

- NLP text generation model
- GUI creation
- Text Processing

3.2 Out of Scope

The following areas are out of scope for this project:

- Audio generation
- Musical generation
- Audio engineering

4 Business Drivers

- *Improve efficiency in the music production process*
- *Reduce time or costs of song writing process*
- *Avoid legal and copyright issues*

.

4.1 Business Driver 1

Improve efficiency in the music production process

- This new innovation is going to be a breakthrough not just for experienced music lyrics composers but will be a very great advantage to new comer musicians trying to find a grip around lyrics composition.

4.2 Business Driver 2

Reduce time and cost of song writing

- It cost a lot to get ghost writers to project an artiste’s creativity in form or words. This AI solution which can be retrained over and over again with more original lyrics of a certain composer can help create ideas which will be in line with a composer’s style, and this will reduce the time it takes to create a song and also the cost of hiring another individual to do that for the composer.

4.3 Business Driver 3

Avoid Legal and copyright issues.

- A lot of copyright issues are involved in hiring individuals to assist in the creative process of music production. This AI solution will definitely cut down legal issues in the aspect of song writers copyright when applied appropriately.

5 Proposed Process

This project will entail 5 major processes

- Initiation and approval
- Data set availability search
- Training an NLP model
- Testing the Model
- Creating the GUI
- Integration into a music production workflow

6 Functional Requirements

6.1 Priority

The requirements in this document are divided into the following categories:

Value	Rating	Description
1	Critical	This requirement is critical to the success of the project. The project will not be possible without this requirement.
2	High	This requirement is high priority, but the project can be implemented at a bare minimum without this requirement.
3	Medium	This requirement is somewhat important, as it provides some value but the project can proceed without it.
4	Low	This is a low priority requirement, or a “nice to have” feature, if time and cost allow it.
5	Future	This requirement is out of scope for this project, and has been included here for a possible future release.

6.2 Requirements Category 1 (RQC)

ID	Requirement	Priority	Raised By
RQC 1	Data	Critical	
RQC 2	RNN algorithm	Critical	
RQC 3	Error free Text generation Model	Critical	
RQC 4	Fast processing computer system	Critical	
RQC 5	GUI creation and deployment tool	Critical	

7 Non-Functional Requirements

ID	Requirement
NFR 1	Human music Lyrics composer
NFR 2	Music Instrumentals
NFR 3	

8 References

Name	Link
Roger B. Dannenberg	http://www.cs.cmu.edu/~rbd
Dan Nelson	https://stackabuse.com/text-generation-with-python-and-tensorflow-keras/
Jiovan Lin	https://jovianlin.io/keras-models-sequential-vs-functional/

9 Document History

Version	Date	Changes	Author
0.1	24/01/2020	First document	Uche Osakwe
0.2	7/02/2020	Data Acquisition and understanding	Uche Osakwe
0.3	21/02/2020	Modelling Part1 and Part2	Uche Osakwe
0.4	06/03/2020	Evaluation and Quality Assurance/ Text Generator Prototype	Uche Osakwe
0.5			

10 Data Acquisition and Understanding

For this project I have decided to make use of a dataset which contains a large amount of texts written by William Shakespeare. This data set is a very valid one to train our model with because it contains a large amount of words and characters which have already be decoded and cleaned up for immediate use. The source of this dataset is also a reliable one. Python download the dataset directly from a goggle api storage link, so its very secure and loads completely without errors. Subsequently as the project is being executed, we will attempt to feed in just any set of conversations or texts from sources like WhatsApp, preprocess and utilize on our model. The idea is that when the model is fully created, any artiste can feed in their original composition and the machine would learn to compose lyrics very similar to what they would originally compose.

The link is provided here:
https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt

In the next phase I will be talking about the Data Assumptions, Limitations and Constraints

10.1 Data Assumptions:

- This dataset will be elaborate enough to help us train our model properly
- The dataset has already been cleaned up and can be used without issues or errors
- The dataset will be successfully tokenized to keywords based on different genre’s of music

10.2 Limitations

- Although the dataset consists of numerous texts which helps us train a good model with a real life scenario of text input, not much exploratory data analysis can be performed because its not a tabularly defined dataset with distinct feature columns and axis.
- Considering the kind of vocabulary built in to the dataset texts, which is an ancient form of writing and vocabulary. A model created wont be able to generate words and lyrics outside this vocabulary, therefore it wont be able to be in-sync with just any type of music artiste.

10.3 Constraints

- The major constraint so far is being able to embed or tokenize our working dataset according to genre or mood of the music, as our reference generator key words

- Inputting any desired text dataset especially encrypted WhatsApp chat conversation, requires a lot of pre-processing and additionally decoding in order to be read and explored by the python IDE.

In conclusion, with the available data set I have, I can move on to the next stage of building the model and validating it for text generation in this project.

10.4 Methodology

- The major AI methodology to be applied in this project is the NLP(Natural language processing)
- The concept behind this project is Text generation using RNN. This is going to be the backbone of our model.

10.5 Procedures

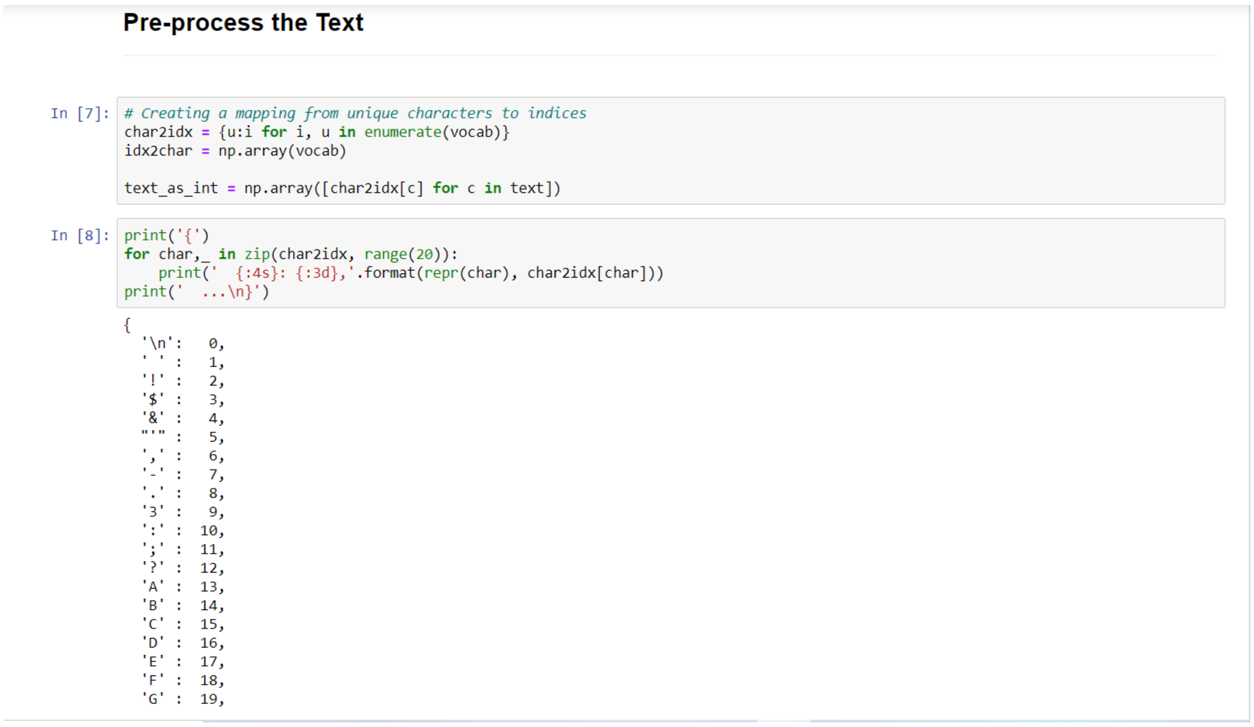
- The first stage of the project is to analyze and explore our dataset in order to understand all the necessary features and required pre-processing steps.
- Secondly pre-processing of the dataset in order to make it ready and fit to apply into a model
- Third phase will be to create the model with a well-established amount of epochs in order to produce a very accurate model.
- Fourth phase would be to create an end point GUI in which users can interact with to generate lyrics per time. Depending on the number of characters we wish to set our model to generate per iteration.
- Final stage would be the testing stage.

Kindly find attached the link to my Exploratory Data Analysis Jupiter notebook.

https://github.com/ucheothniel/uche-osakwe-/blob/master/Exploratory%20Data%20Analysis.ipynb

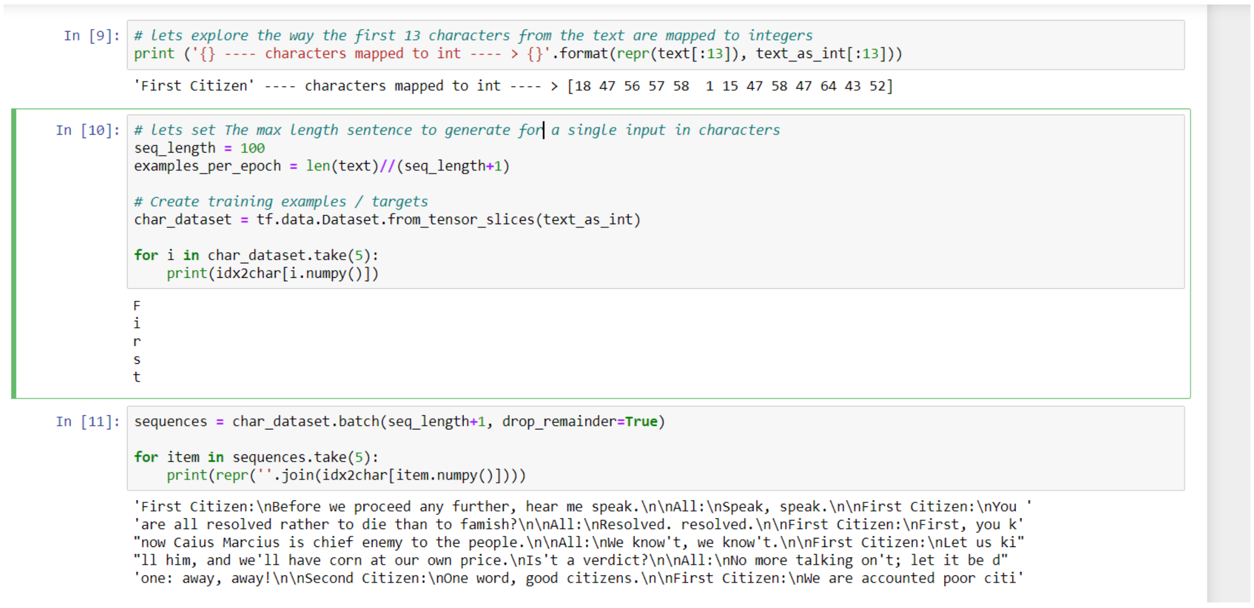
11 Modelling Part 1

11.1 Data Pre-processing Pipeline

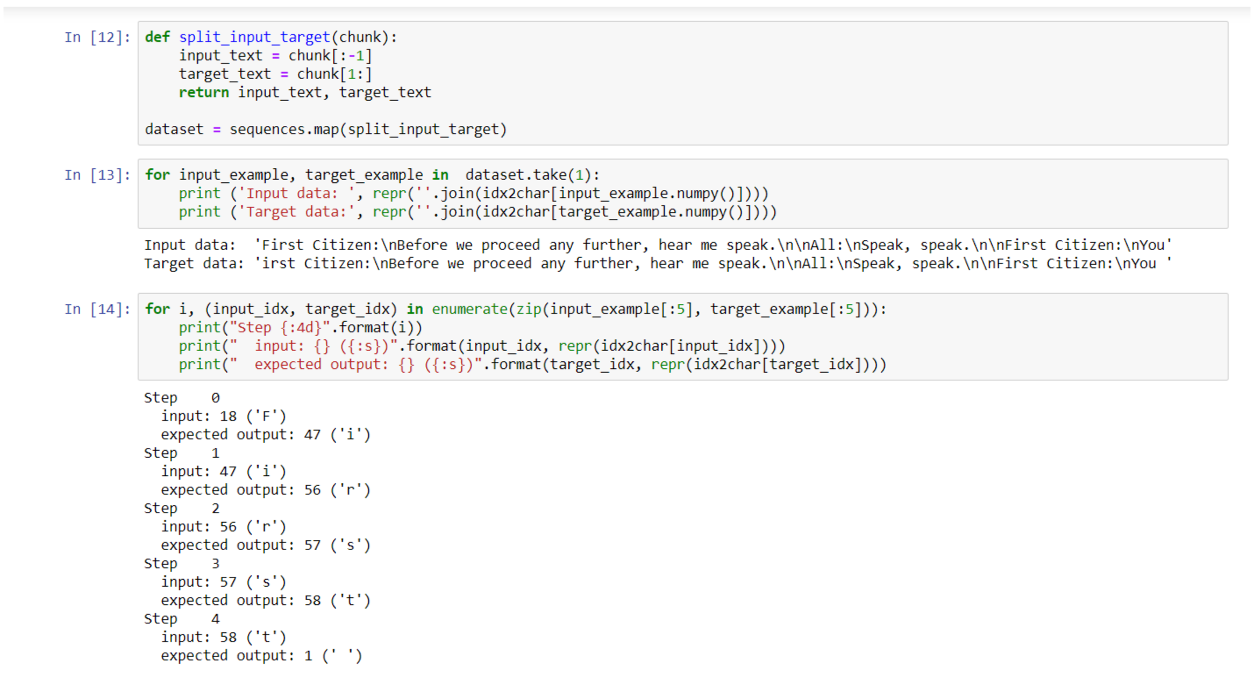


Pre-process the Text

```
In [7]: # Creating a mapping from unique characters to indices
        char2idx = {u:i for i, u in enumerate(vocab)}
        idx2char = np.array(vocab)

        text_as_int = np.array([char2idx[c] for c in text])

In [8]: print('{')
        for char,_ in zip(char2idx, range(20)):
            print('  {:4s}: {:3d},'.format(repr(char), char2idx[char]))
        print('  ...\n}')

        {
          '\n':   0,
          ' ' :   1,
          '!' :   2,
          '$' :   3,
          '&' :   4,
          "'" :   5,
          ',' :   6,
          '-' :   7,
          '.' :   8,
          '3' :   9,
          ':' :  10,
          ';' :  11,
          '?' :  12,
          'A' :  13,
          'B' :  14,
          'C' :  15,
          'D' :  16,
          'E' :  17,
          'F' :  18,
          'G' :  19,
```



```
In [9]: # lets explore the way the first 13 characters from the text are mapped to integers
        print ('{} ---- characters mapped to int ---- > {}'.format(repr(text[:13]), text_as_int[:13]))

        'First Citizen' ---- characters mapped to int ---- > [18 47 56 57 58  1 15 47 58 47 64 43 52]

In [10]: # lets set The max length sentence to generate for a single input in characters
         seq_length = 100
         examples_per_epoch = len(text)//(seq_length+1)

         # Create training examples / targets
         char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)

         for i in char_dataset.take(5):
             print(idx2char[i.numpy()])

         F
         i
         r
         s
         t

In [11]: sequences = char_dataset.batch(seq_length+1, drop_remainder=True)

         for item in sequences.take(5):
             print(repr(''.join(idx2char[item.numpy()])))

         'First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou '
         'are all resolved rather to die than to famish?\n\nAll:\nResolved. resolved.\n\nFirst Citizen:\nFirst, you k'
         "now Caius Marcius is chief enemy to the people.\n\nAll:\nWe know't, we know't.\n\nFirst Citizen:\nLet us ki"
         "ll him, and we'll have corn at our own price.\nIs't a verdict?\n\nAll:\nNo more talking on't; let it be d"
         'one: away, away!\n\nSecond Citizen:\nOne word, good citizens.\n\nFirst Citizen:\nWe are accounted poor citi'
```



```
In [12]: def split_input_target(chunk):
             input_text = chunk[:-1]
             target_text = chunk[1:]
             return input_text, target_text

         dataset = sequences.map(split_input_target)
```

```
In [13]: for input_example, target_example in  dataset.take(1):
             print ('Input data: ', repr(''.join(idx2char[input_example.numpy()])))
             print ('Target data:', repr(''.join(idx2char[target_example.numpy()])))
```

```
Input data:  'First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou'
Target data: 'irst Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou '
```

```
In [14]: for i, (input_idx, target_idx) in enumerate(zip(input_example[:5], target_example[:5])):
             print("Step {:4d}".format(i))
             print("  input: {} ({:s})".format(input_idx, repr(idx2char[input_idx])))
             print("  expected output: {} ({:s})".format(target_idx, repr(idx2char[target_idx])))
```

```
Step    0
  input: 18 ('F')
  expected output: 47 ('i')
Step    1
  input: 47 ('i')
  expected output: 56 ('r')
Step    2
  input: 56 ('r')
  expected output: 57 ('s')
Step    3
  input: 57 ('s')
  expected output: 58 ('t')
Step    4
  input: 58 ('t')
  expected output: 1 (' ')
```

12 Modelling Part 2

12.1 Model Library selection:

For this section we will be iterating how our model will be built using the RNN deep learning text generator model with Keras and Tensorflow. These two libraries have been selected due to their seamless operation with NLP processing and they also possess a large community support for developers and this fact has been very substantial in providing solutions to errors and stumbling blocks while creating this model.

Pros of TensorFlow

- Debugging of codes is easy using TensorFlow.
- Library Management: Tensorflow is backed by google which gives it high support on updates, issue resolution and seamless operation.
- Pipelining: TensorFlow is designed to use various backend software (GPUs, ASIC), etc and it is also highly parallel.
- Its performance is high and matching the best in the industry.

Cons of TensorFlow

- Benchmark tests: TensorFlow lacks in both speed and usage when it is compared to its competitors.
- No support for OpenCL.
- It is a very low level with a steep learning curve.
- There is no need for any super low-level matter.

12.2 Model Selection

The Two Models to be considered for our project are The Keras Sequential Model and the Keras Functional Model.

The sequential API allows you to create models layer-by-layer for most problems. It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs.

For example, it is not straightforward to define models that may have:

1. multiple different input sources,
2. produce multiple output destinations, or
3. models that re-use layers.

Alternatively, the functional API allows you to create models that have a lot more flexibility as you can easily define models where layers connect to more than just the previous and next layers. In fact, you can connect layers to (literally) any other layer. As a result, creating complex networks such as siamese networks and residual networks become possible. (Lin, 2017)

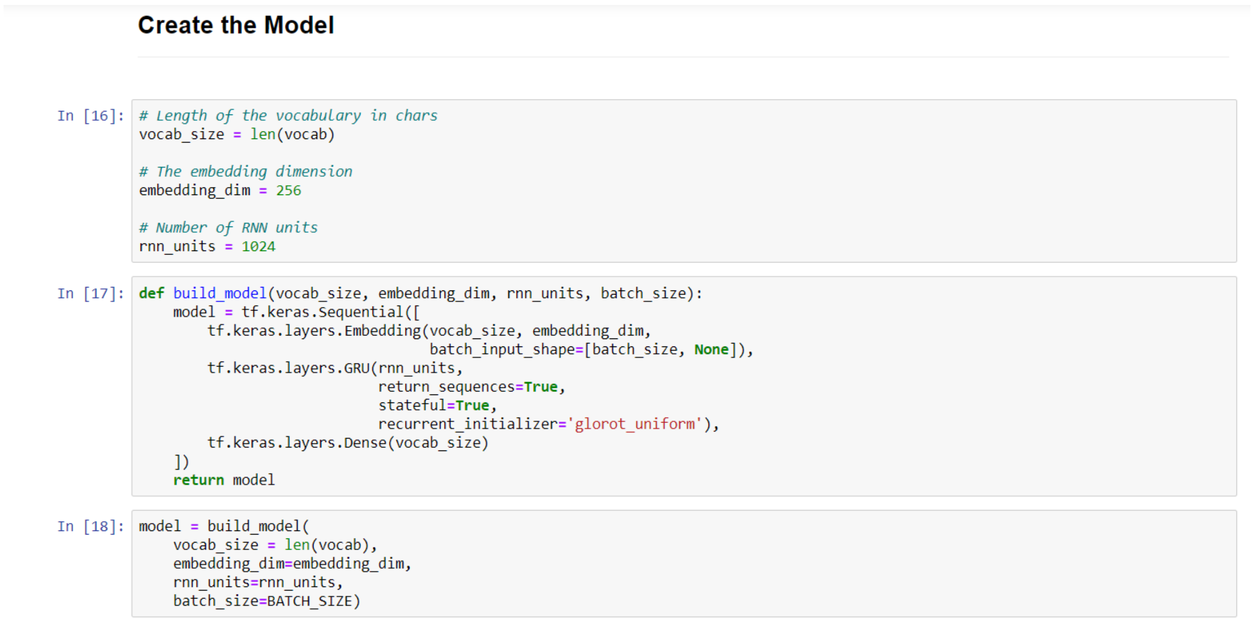
Based on all my research concerning NLP text generation the Keras Sequential model and the keras functional model are the two keras models that fit our project and I've decided to move forward in selecting Tensorflow Keras Sequential model ahead of the keras functional model.

Rationale:

- The lack of complexity of our project does not require us to compulsorily make use of the functional model as it consumes more computational power.
- The sequential model works optimally for single input and single output models and in this case that is the framework of the music lyrics generation project.
- The final rationale for selecting the sequential model over the functional model is that our text generation model will be made to predict and generate texts based on already trained sequences of conversations and texts in the input data in order to generate similarly structured sentences compared to the structure of the input data.

12.3 Model Creation

Screenshots of the Keras Sequential Model creation.

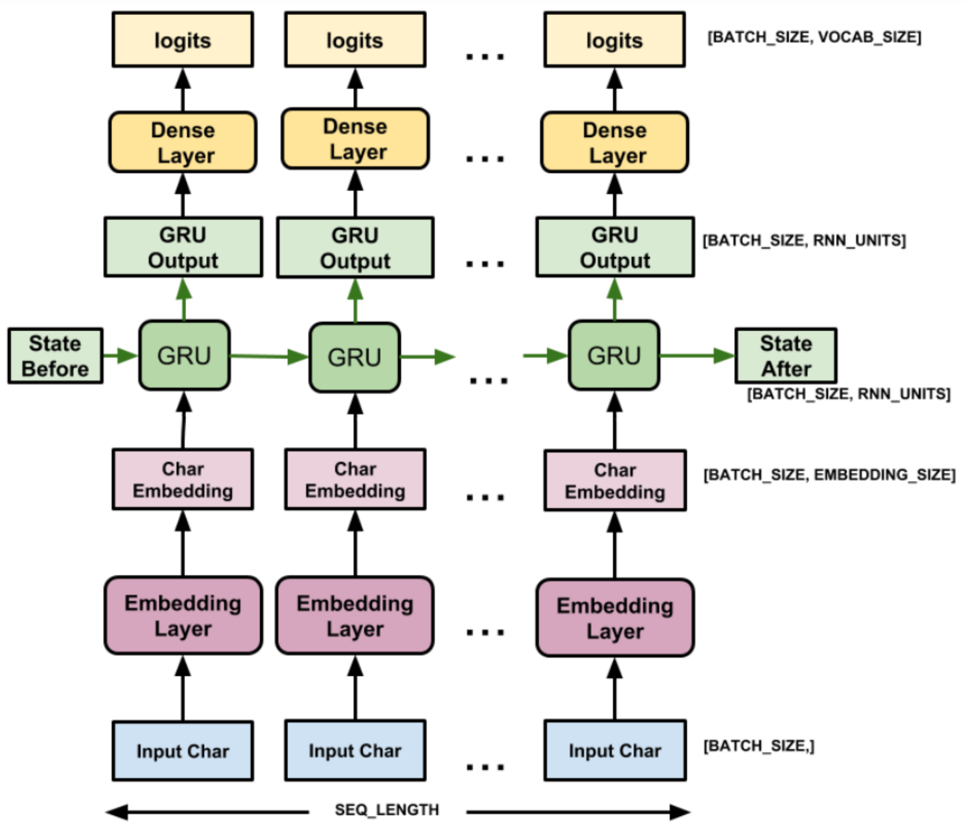


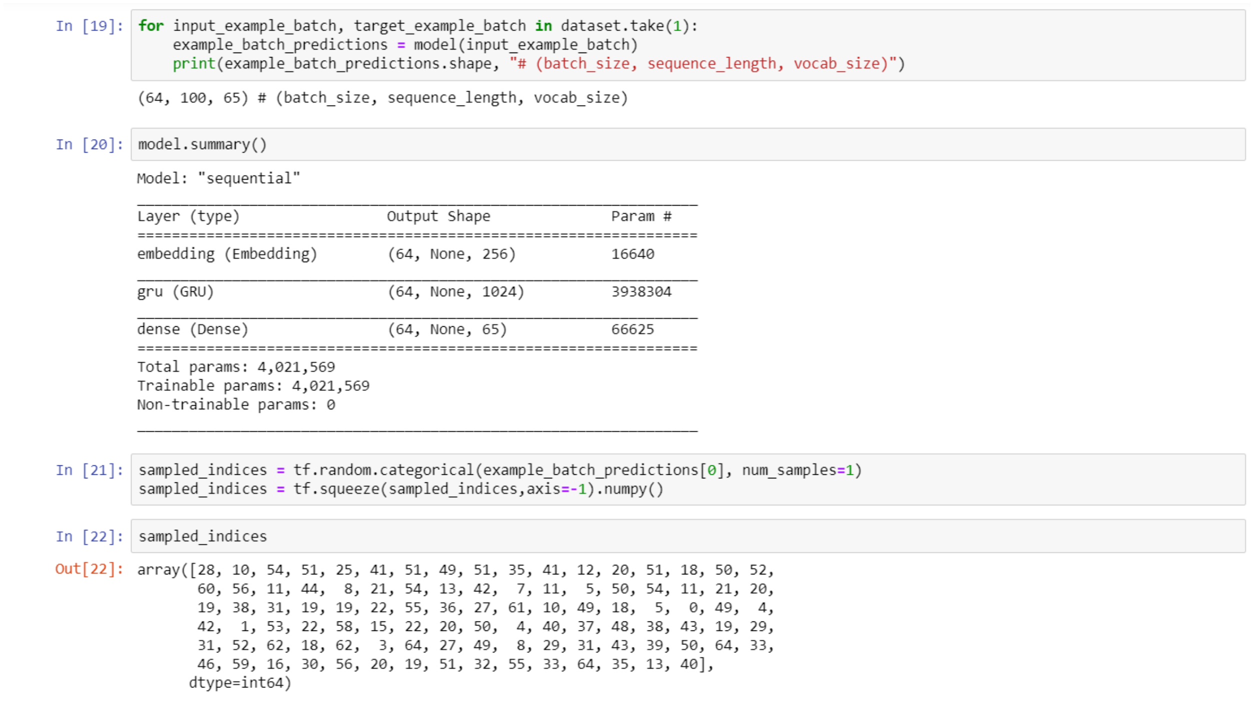
Create the Model

```
In [16]: # Length of the vocabulary in chars
         vocab_size = len(vocab)

         # The embedding dimension
         embedding_dim = 256

         # Number of RNN units
         rnn_units = 1024
```

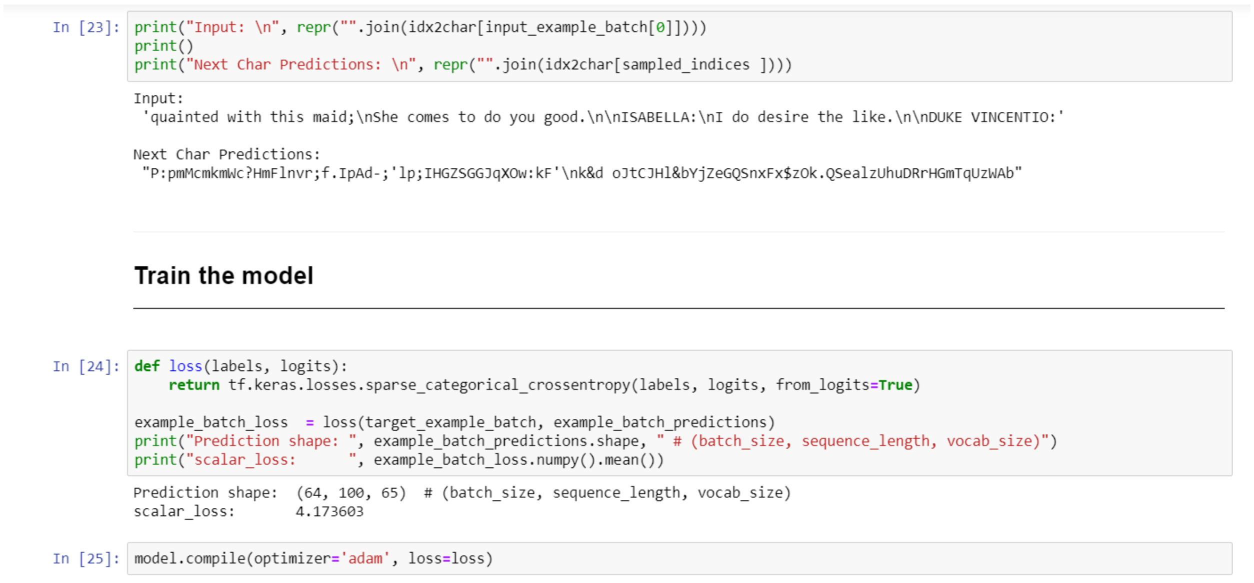
```
In [17]: def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
             model = tf.keras.Sequential([
                 tf.keras.layers.Embedding(vocab_size, embedding_dim,
                                           batch_input_shape=[batch_size, None]),
                 tf.keras.layers.GRU(rnn_units,
                                     return_sequences=True,
                                     stateful=True,
                                     recurrent_initializer='glorot_uniform'),
                 tf.keras.layers.Dense(vocab_size)
             ])
             return model
```

```
In [18]: model = build_model(
             vocab_size = len(vocab),
             embedding_dim=embedding_dim,
             rnn_units=rnn_units,
             batch_size=BATCH_SIZE)
```





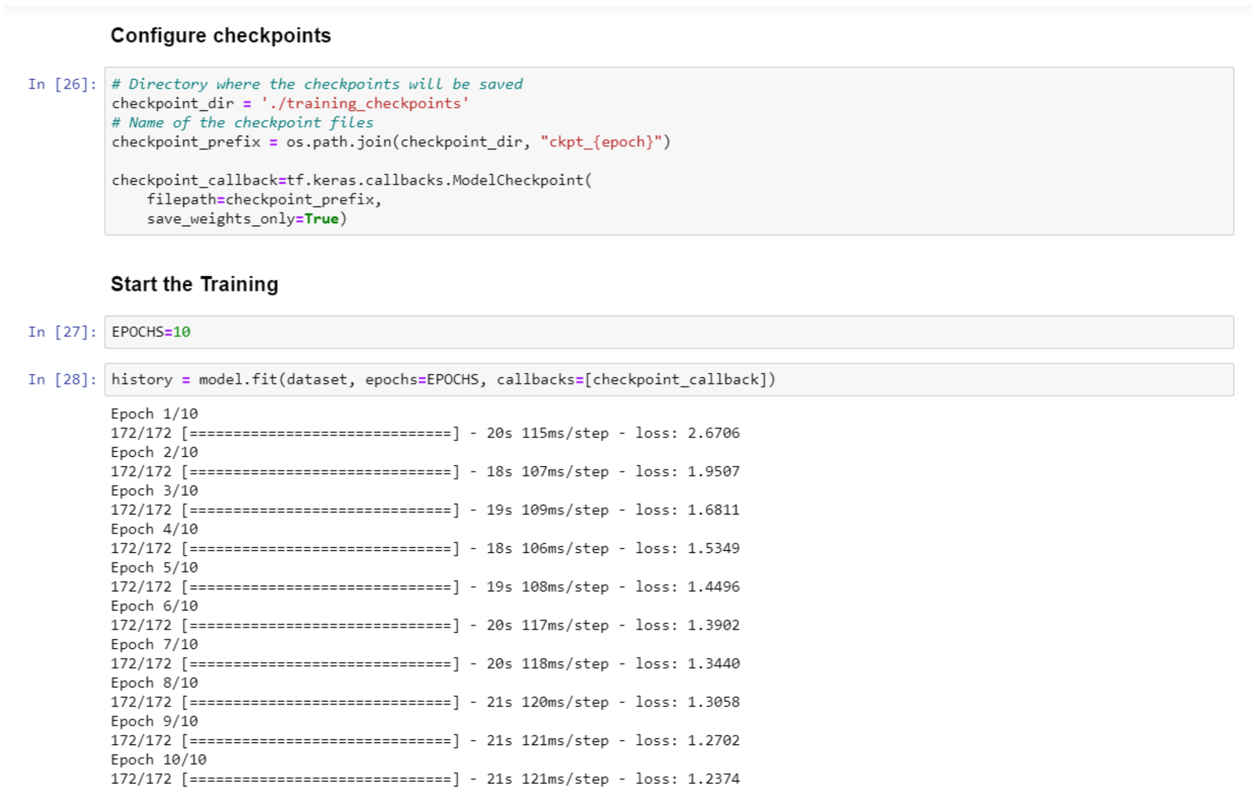
```
In [19]: for input_example_batch, target_example_batch in dataset.take(1):
             example_batch_predictions = model(input_example_batch)
             print(example_batch_predictions.shape, "# (batch_size, sequence_length, vocab_size)")

         (64, 100, 65) # (batch_size, sequence_length, vocab_size)

In [20]: model.summary()

         Model: "sequential"
         _________________________________________________________________
         Layer (type)                 Output Shape              Param #
         =================================================================
         embedding (Embedding)        (64, None, 256)           16640
         _________________________________________________________________
         gru (GRU)                    (64, None, 1024)          3938304
         _________________________________________________________________
         dense (Dense)                (64, None, 65)            66625
         =================================================================
         Total params: 4,021,569
         Trainable params: 4,021,569
         Non-trainable params: 0
         _________________________________________________________________

In [21]: sampled_indices = tf.random.categorical(example_batch_predictions[0], num_samples=1)
         sampled_indices = tf.squeeze(sampled_indices,axis=-1).numpy()

In [22]: sampled_indices

Out[22]: array([28, 10, 54, 51, 25, 41, 51, 49, 51, 35, 41, 12, 20, 51, 18, 50, 52,
                60, 56, 11, 44,  8, 21, 54, 13, 42,  7, 11,  5, 50, 54, 11, 21, 20,
                19, 38, 31, 19, 19, 22, 55, 36, 27, 61, 10, 49, 18,  5,  0, 49,  4,
                42,  1, 53, 22, 58, 15, 22, 20, 50,  4, 40, 37, 48, 38, 43, 19, 29,
                31, 52, 62, 18, 62,  3, 64, 27, 49,  8, 29, 31, 43, 39, 50, 64, 33,
                46, 59, 16, 30, 56, 20, 19, 51, 32, 55, 33, 64, 35, 13, 40],
               dtype=int64)
```



```
In [23]: print("Input: \n", repr("".join(idx2char[input_example_batch[0]])))
         print()
         print("Next Char Predictions: \n", repr("".join(idx2char[sampled_indices ])))

         Input:
          'quainted with this maid;\nShe comes to do you good.\n\nISABELLA:\nI do desire the like.\n\nDUKE VINCENTIO:'

         Next Char Predictions:
          "P:pmMcmkmWc?HmFlnvr;f.IpAd-;'lp;IHGZSGGJqXOw:kF'\nk&d oJtCJHl&bYjZeGQSnxFx$zOk.QSealzUhuDRrHGmTqUzWAb"
```

Train the model

```
In [24]: 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, 65)  # (batch_size, sequence_length, vocab_size)
         scalar_loss:       4.173603

In [25]: model.compile(optimizer='adam', loss=loss)
```



Configure checkpoints

```
In [26]: # Directory where the checkpoints will be saved
         checkpoint_dir = './training_checkpoints'
         # Name of the checkpoint files
         checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

         checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(
             filepath=checkpoint_prefix,
             save_weights_only=True)
```

Start the Training

```
In [27]: EPOCHS=10
```

```
In [28]: history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])
```

```
Epoch 1/10
172/172 [==============================] - 20s 115ms/step - loss: 2.6706
Epoch 2/10
172/172 [==============================] - 18s 107ms/step - loss: 1.9507
Epoch 3/10
172/172 [==============================] - 19s 109ms/step - loss: 1.6811
Epoch 4/10
172/172 [==============================] - 18s 106ms/step - loss: 1.5349
Epoch 5/10
172/172 [==============================] - 19s 108ms/step - loss: 1.4496
Epoch 6/10
172/172 [==============================] - 20s 117ms/step - loss: 1.3902
Epoch 7/10
172/172 [==============================] - 20s 118ms/step - loss: 1.3440
Epoch 8/10
172/172 [==============================] - 21s 120ms/step - loss: 1.3058
Epoch 9/10
172/172 [==============================] - 21s 121ms/step - loss: 1.2702
Epoch 10/10
172/172 [==============================] - 21s 121ms/step - loss: 1.2374
```

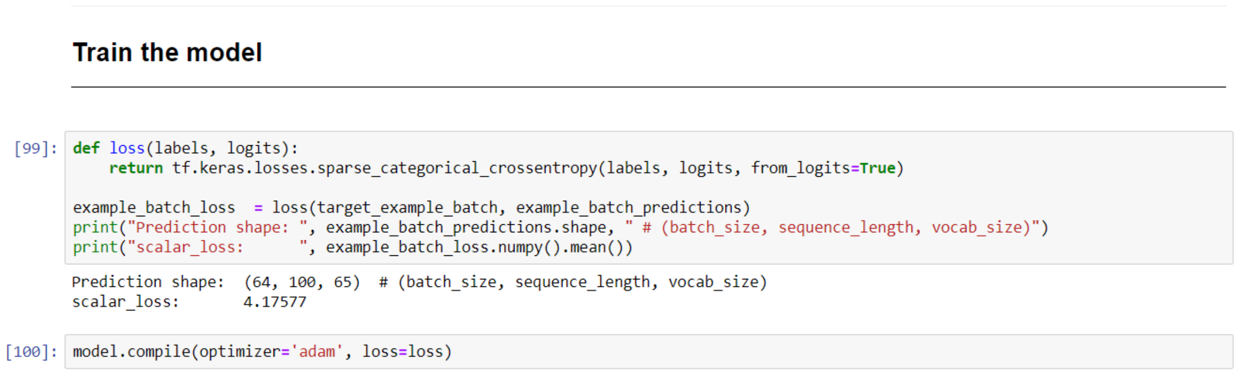
Link to Modelling Part 1 and 2 codes github

https://github.com/ucheothniel/uche-osakwe-/blob/master/Modelling%20part%201%20and%20part%202.ipynb

13 Evaluation and Quality Assurance

- The RNN model has been created and tested using different number of epochs in order to validate and calculate the Loss performance of our model on our sample training batch dataset.

Initial Mean loss value of our dataset batch has been evaluated at 4.17577 Which is a very high value and this loss needs to be reduced minimally by setting an appropriate Epochs.

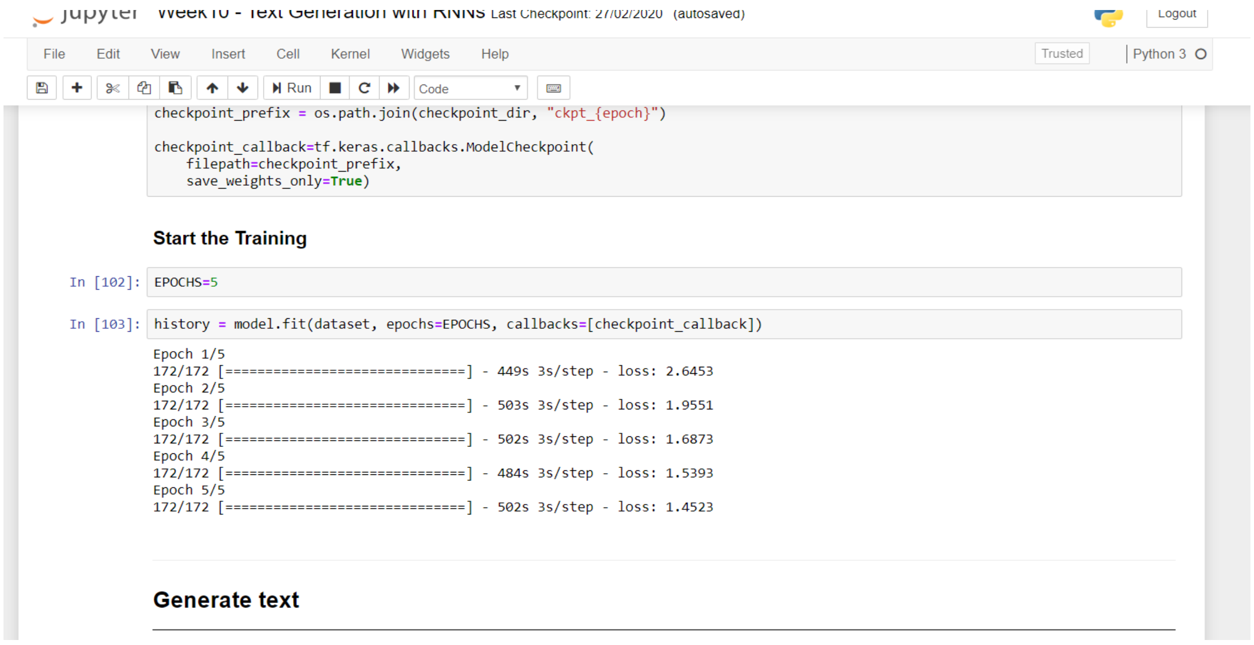


Train the model

```
[99]: 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, 65)  # (batch_size, sequence_length, vocab_size)
      scalar_loss:       4.17577

[100]: model.compile(optimizer='adam', loss=loss)
```

Two epochs numbers have been utilized and the loss values have been recorded for observation.

Working with “Epochs = 5”


jupyter
Week 10 - Text Generation with RNNs Last Checkpoint: 27/02/2020 (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code

```
            checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

            checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(
                filepath=checkpoint_prefix,
                save_weights_only=True)
```

Start the Training

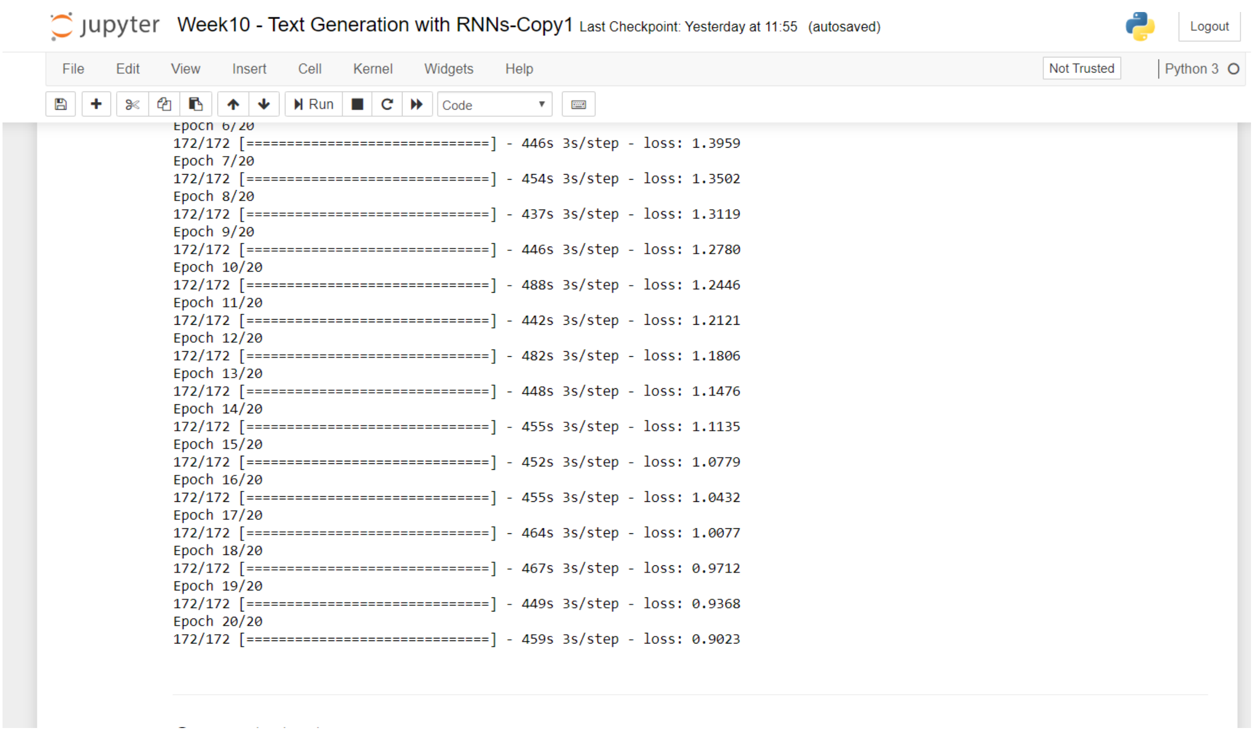
```
In [102]: EPOCHS=5

In [103]: history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])

          Epoch 1/5
          172/172 [==============================] - 449s 3s/step - loss: 2.6453
          Epoch 2/5
          172/172 [==============================] - 503s 3s/step - loss: 1.9551
          Epoch 3/5
          172/172 [==============================] - 502s 3s/step - loss: 1.6873
          Epoch 4/5
          172/172 [==============================] - 484s 3s/step - loss: 1.5393
          Epoch 5/5
          172/172 [==============================] - 502s 3s/step - loss: 1.4523
```

Generate text

After running 5 Epochs, the model was able to iterate really fast, but ended with a loss of 1.4523 which is on the high side for a loss value. At least we aim to have a loss value below “1”

After much research online I observed that most neural networks although usually have large datasets, require to be run on an epochs of at least 15 – 30, but not too high as this leads to vanishing gradient problem.

Therefore, I decided to work with “Epochs = 20”


jupyter Week10 - Text Generation with RNNs-Copy1 Last Checkpoint: Yesterday at 11:55 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code

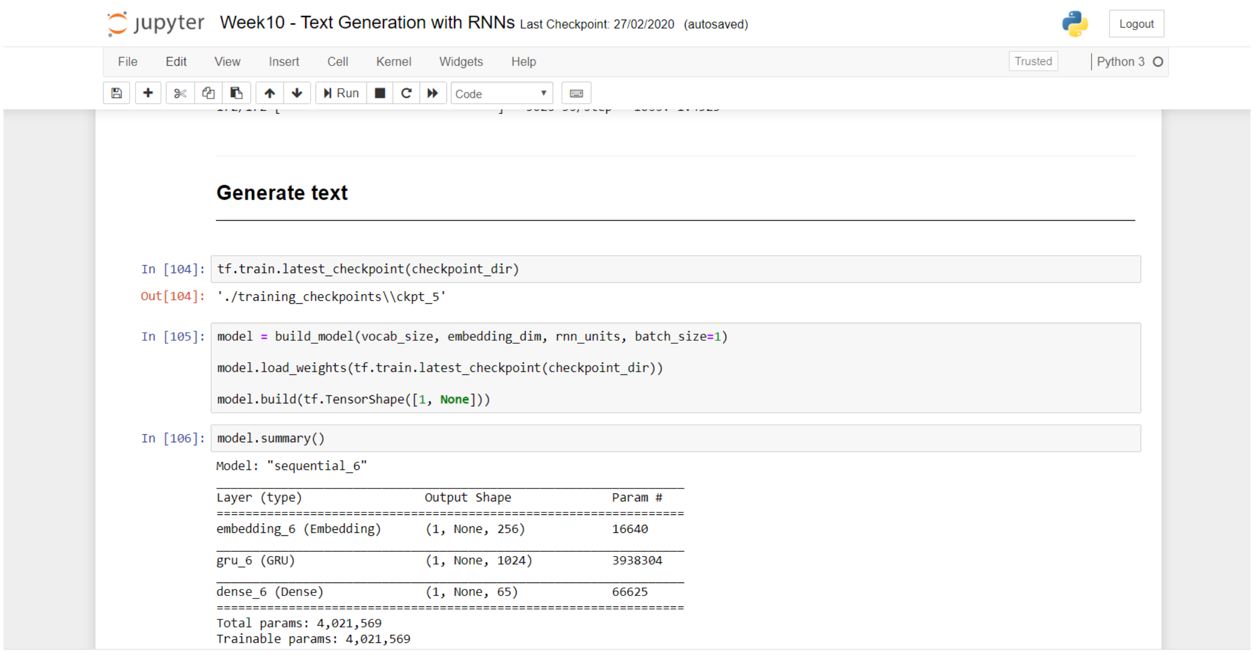
```
Epoch 6/20
172/172 [==============================] - 446s 3s/step - loss: 1.3959
Epoch 7/20
172/172 [==============================] - 454s 3s/step - loss: 1.3502
Epoch 8/20
172/172 [==============================] - 437s 3s/step - loss: 1.3119
Epoch 9/20
172/172 [==============================] - 446s 3s/step - loss: 1.2780
Epoch 10/20
172/172 [==============================] - 488s 3s/step - loss: 1.2446
Epoch 11/20
172/172 [==============================] - 442s 3s/step - loss: 1.2121
Epoch 12/20
172/172 [==============================] - 482s 3s/step - loss: 1.1806
Epoch 13/20
172/172 [==============================] - 448s 3s/step - loss: 1.1476
Epoch 14/20
172/172 [==============================] - 455s 3s/step - loss: 1.1135
Epoch 15/20
172/172 [==============================] - 452s 3s/step - loss: 1.0779
Epoch 16/20
172/172 [==============================] - 455s 3s/step - loss: 1.0432
Epoch 17/20
172/172 [==============================] - 464s 3s/step - loss: 1.0077
Epoch 18/20
172/172 [==============================] - 467s 3s/step - loss: 0.9712
Epoch 19/20
172/172 [==============================] - 449s 3s/step - loss: 0.9368
Epoch 20/20
172/172 [==============================] - 459s 3s/step - loss: 0.9023
```

From this iteration we can observe that the loss value of our RNN model got reduced over the 20 epochs and we ended at a value of 0.9023. This shows us that our model performs better when trained on larger epochs values than a smaller number.

14 Text Generator Prototype

In this section of the report, I have successfully trained my model and I have performed my first text generation operation.

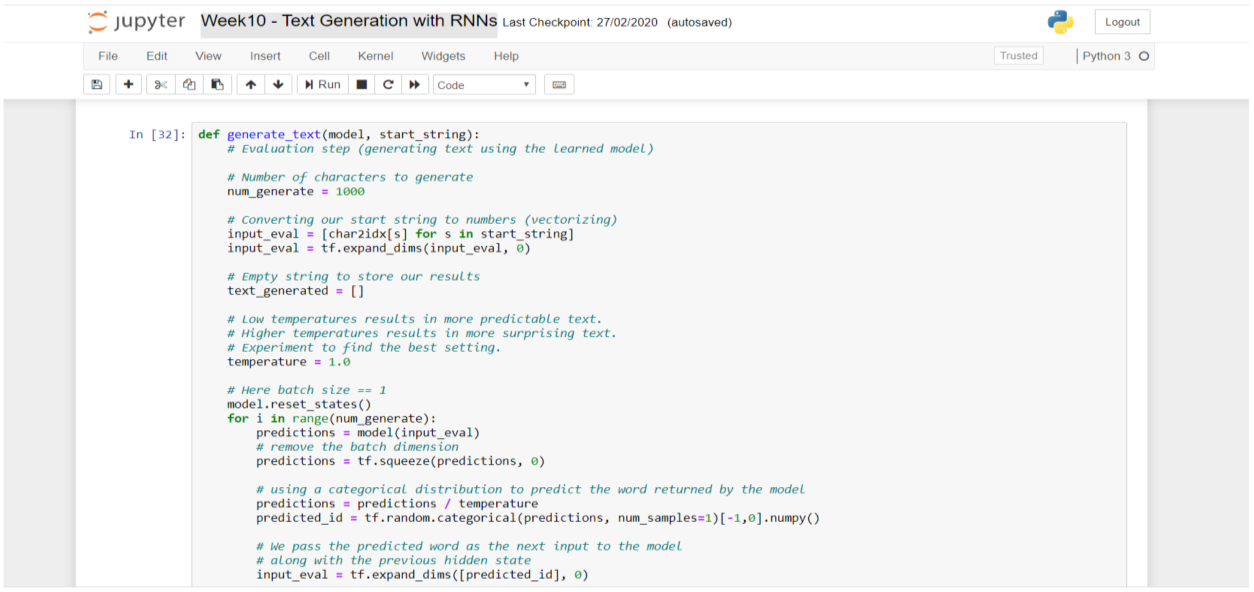
Step 1: Load the model training checkpoint and model weights


jupyter Week10 - Text Generation with RNNs Last Checkpoint: 27/02/2020 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Generate text

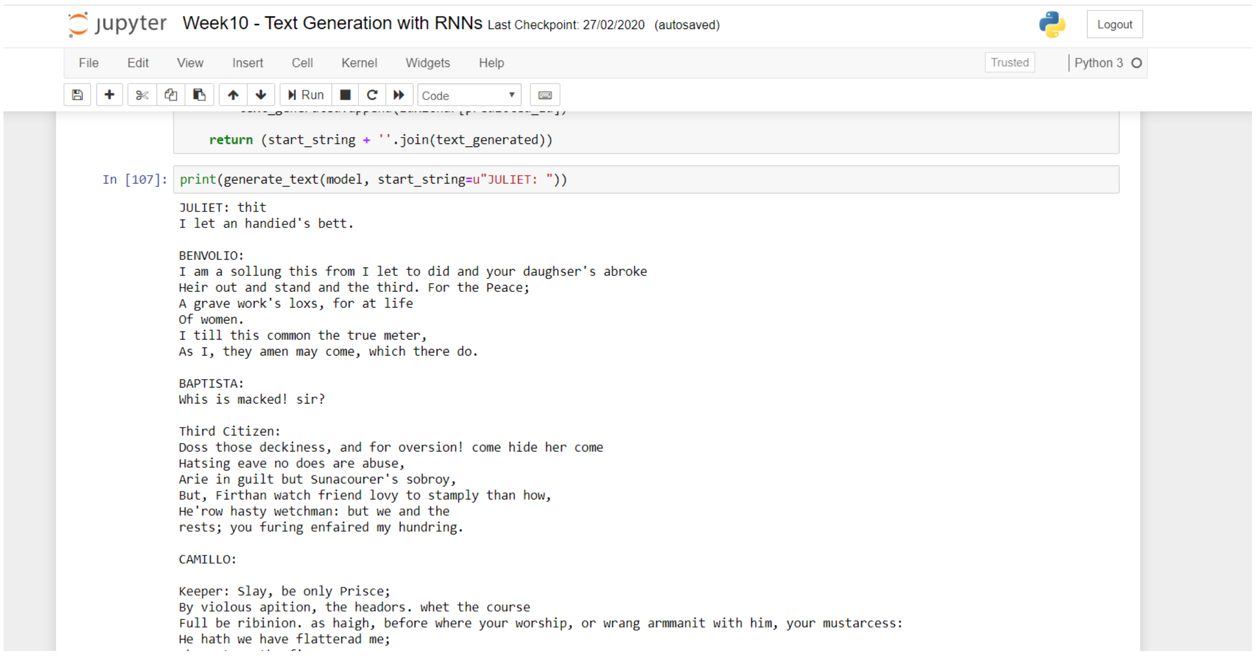
```
In [104]: tf.train.latest_checkpoint(checkpoint_dir)
Out[104]: './training_checkpoints\\ckpt_5'

In [105]: model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=1)

          model.load_weights(tf.train.latest_checkpoint(checkpoint_dir))

          model.build(tf.TensorShape([1, None]))

In [106]: model.summary()

          Model: "sequential_6"
          _________________________________________________________________
          Layer (type)                 Output Shape              Param #
          =================================================================
          embedding_6 (Embedding)      (1, None, 256)            16640
          _________________________________________________________________
          gru_6 (GRU)                  (1, None, 1024)           3938304
          _________________________________________________________________
          dense_6 (Dense)              (1, None, 65)             66625
          =================================================================
          Total params: 4,021,569
          Trainable params: 4,021,569
```

Step 2: Define the text generation function


jupyter Week10 - Text Generation with RNNs Last Checkpoint: 27/02/2020 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [32]: def generate_text(model, start_string):
           # Evaluation step (generating text using the learned model)

           # Number of characters to generate
           num_generate = 1000

           # 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 = 1.0

           # 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 word returned by the model
               predictions = predictions / temperature
               predicted_id = tf.random.categorical(predictions, num_samples=1)[-1,0].numpy()

               # We pass the predicted word as the next input to the model
               # along with the previous hidden state
               input_eval = tf.expand_dims([predicted_id], 0)
```

Step 3: Run the text generator with a token word “JULIET”


jupyter
Week10 - Text Generation with RNNs Last Checkpoint: 27/02/2020 (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code

```
    return (start_string + ''.join(text_generated))
```

```
In [107]: print(generate_text(model, start_string=u"JULIET: "))
```

```
JULIET: thit
I let an handied's bett.

BENVOLIO:
I am a sollung this from I let to did and your daughser's abroke
Heir out and stand and the third. For the Peace;
A grave work's loxs, for at life
Of women.
I till this common the true meter,
As I, they amen may come, which there do.

BAPTISTA:
Whis is macked! sir?

Third Citizen:
Doss those deckiness, and for oversion! come hide her come
Hatsing eave no does are abuse,
Arie in guilt but Sunacourer's sobroy,
But, Firthan watch friend lovy to stamply than how,
He'row hasty wetchman: but we and the
rests; you furing enfaired my hundring.

CAMILLO:

Keeper: Slay, be only Prisce;
By violous apition, the headors. whet the course
Full be ribinion. as haigh, before where your worship, or wrang armmanit with him, your mustarcess:
He hath we have flatterad me;
```

We have been able to successfully generate sentences based on our initial dataset input.

Furthermore some other concepts such as LSTM layers can be introduced into this type of RNN model, although it would require more work, a different pre-processing technique, and addition of layers to the model architecture, this would be a possibly good improvement to the model.