CS5542 BIG DATA APPS AND ANALYTICS

In Class Programming – 6

Recurrent Neural Networks (RNN)

Kalyani Nikure
Kmn6ng@umkc.edu

Table of Contents

Description

Use a different data and use the model provided in ICP6 to perform Text generation. You must make 4 changes (for example adding LSTM layers to model, changing hyperparameters etc) in the source code. Report your findings in detail.

Note: please indicate in your reports which 4 changes you made in the source code and why in your opinion these changes are logical.

Detailed Steps Explanation

1. Model Evaluation which is already provided in the source code

- Importing all the required dependencies

```
import os
import time
import pandas as pd
import numpy as np
import re
import tensorflow as tf
from keras.preprocessing.text import Tokenizer
from sklearn.preprocessing import LabelEncoder
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential, load_model
from keras.layers import Dense, Embedding, LSTM, Dropout
from keras.callbacks import TensorBoard, ModelCheckpoint
from keras.utils.np_utils import to_categorical
from matplotlib import pyplot
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings('ignore')
```

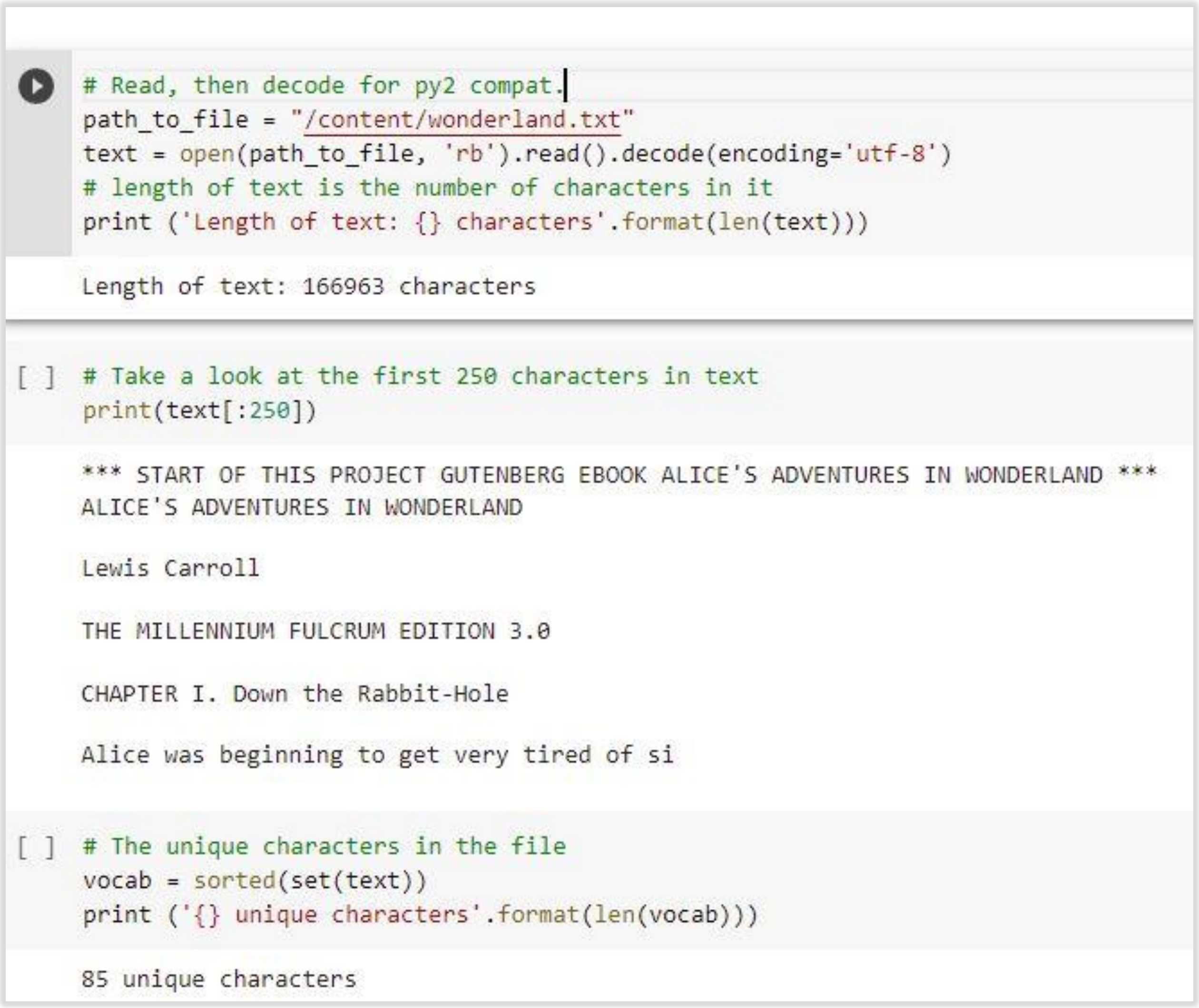
- About dataset

  In this coding challenge I am going to use a favorite book from childhood as the dataset: Alice's Adventures in Wonderland by Lewis Carroll.

  We are going to learn the dependencies between characters and the conditional probabilities of characters in sequences so that we can in turn generate wholly new and original sequences of characters. The text file has about 3,330 lines of text.

  Download Source: https://www.gutenberg.org/cache/epub/11/pg11.txt

- Reading the data from the .txt file and visualize the data



```
# Read, then decode for py2 compat.
path_to_file = "/content/wonderland.txt"
text = open(path_to_file, 'rb').read().decode(encoding='utf-8')
# length of text is the number of characters in it
print ('Length of text: {} characters'.format(len(text)))
```

```
Length of text: 166963 characters
```

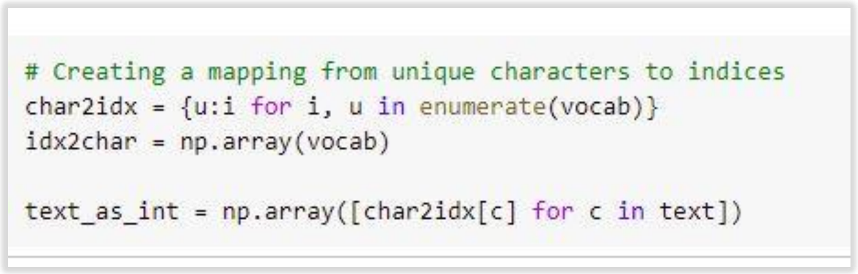
```
[ ] # Take a look at the first 250 characters in text
print(text[:250])
```

```
*** START OF THIS PROJECT GUTENBERG EBOOK ALICE'S ADVENTURES IN WONDERLAND ***
ALICE'S ADVENTURES IN WONDERLAND

Lewis Carroll

THE MILLENNIUM FULCRUM EDITION 3.0

CHAPTER I. Down the Rabbit-Hole

Alice was beginning to get very tired of si
```

```
[ ] # The unique characters in the file
vocab = sorted(set(text))
print ('{} unique characters'.format(len(vocab)))
```

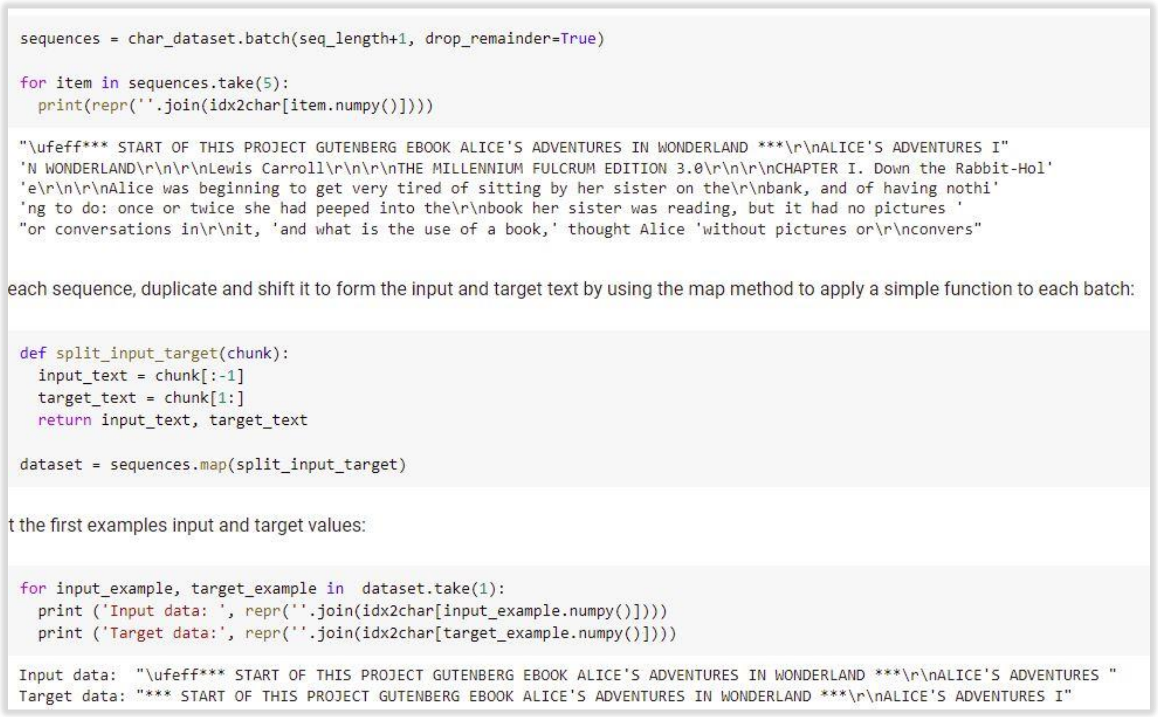
```
85 unique characters
```

- Create mapping from unique characters to indices. This text formatting will help us in modelling the data



```
# 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])
```

- Create sequences from the input text



```
sequences = char_dataset.batch(seq_length+1, drop_remainder=True)

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

```
"\ufeff*** START OF THIS PROJECT GUTENBERG EBOOK ALICE'S ADVENTURES IN WONDERLAND ***\r\nALICE'S ADVENTURES I"
'N WONDERLAND\r\n\r\n\r\nLewis Carroll\r\n\r\n\r\nTHE MILLENNIUM FULCRUM EDITION 3.0\r\n\r\n\r\nCHAPTER I. Down the Rabbit-Hol'
'e\r\n\r\n\r\nAlice was beginning to get very tired of sitting by her sister on the\r\nbank, and of having nothi'
'ng to do: once or twice she had peeped into the\r\nbook her sister was reading, but it had no pictures '
"or conversations in\r\nit, 'and what is the use of a book,' thought Alice 'without pictures or\r\nconvers"
```

each sequence, duplicate and shift it to form the input and target text by using the map method to apply a simple function to each batch:

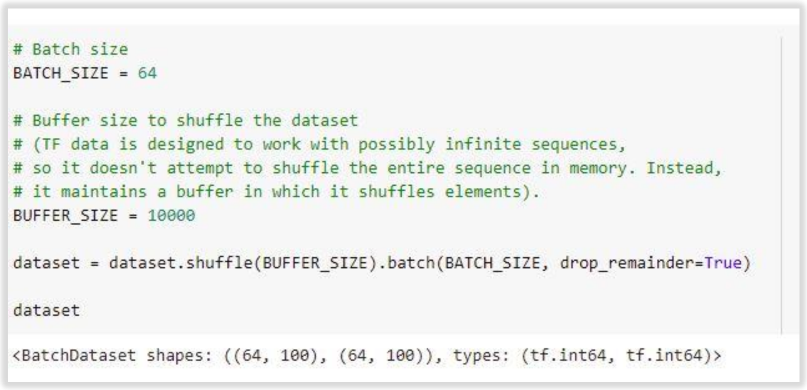
```
def split_input_target(chunk):
  input_text = chunk[:-1]
  target_text = chunk[1:]
  return input_text, target_text

dataset = sequences.map(split_input_target)
```

t the first examples input and target values:

```
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:  "\ufeff*** START OF THIS PROJECT GUTENBERG EBOOK ALICE'S ADVENTURES IN WONDERLAND ***\r\nALICE'S ADVENTURES "
Target data: "*** START OF THIS PROJECT GUTENBERG EBOOK ALICE'S ADVENTURES IN WONDERLAND ***\r\nALICE'S ADVENTURES I"
```

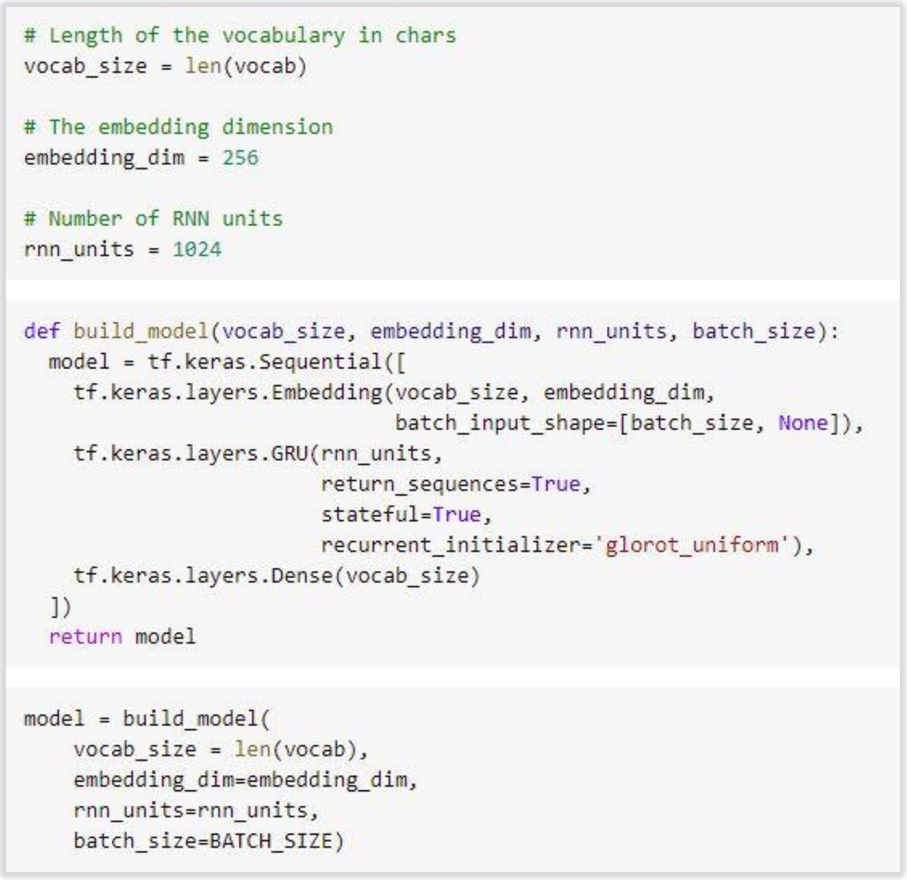
- Batch the datasets in the shape of (64, 100)



```
# Batch size
BATCH_SIZE = 64

# Buffer size to shuffle the dataset
# (TF data is designed to work with possibly infinite sequences,
# so it doesn't attempt to shuffle the entire sequence in memory. Instead,
# it maintains a buffer in which it shuffles elements).
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)>
```

- Build the model using exact same layers provided in the use case

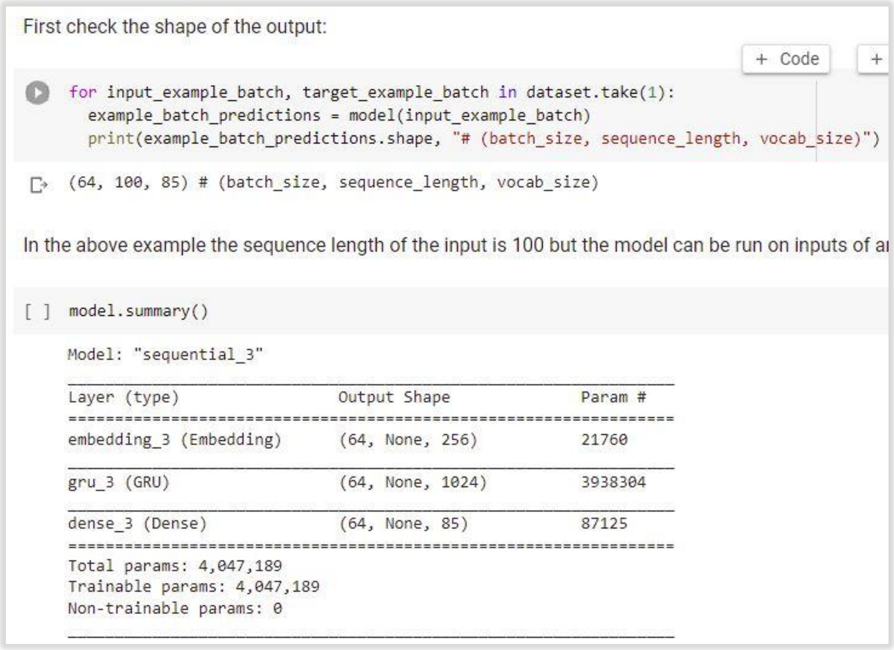


```
# Length of the vocabulary in chars
vocab_size = len(vocab)

# The embedding dimension
embedding_dim = 256

# Number of RNN units
rnn_units = 1024
```

```
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
```

```
model = build_model(
    vocab_size = len(vocab),
    embedding_dim=embedding_dim,
    rnn_units=rnn_units,
    batch_size=BATCH_SIZE)
```

- Model Summary with layers embedded and total parameters trained.



First check the shape of the output:

```
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)")
```

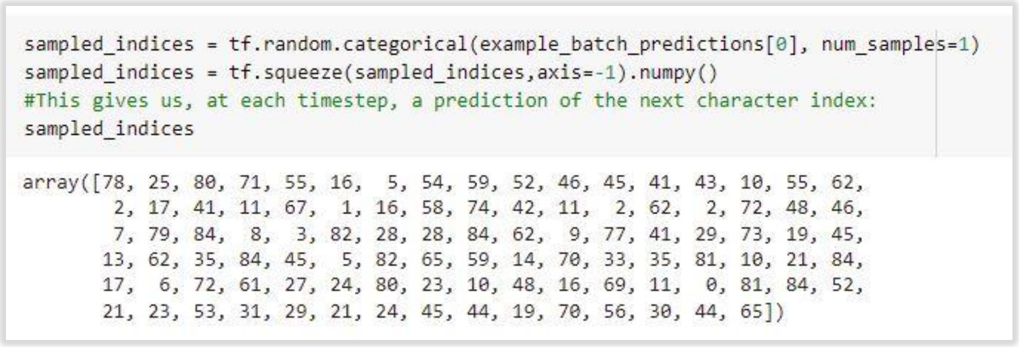
```
(64, 100, 85) # (batch_size, sequence_length, vocab_size)
```

In the above example the sequence length of the input is 100 but the model can be run on inputs of a

```
[ ] model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (64, None, 256)           21760
_________________________________________________________________
gru_3 (GRU)                  (64, None, 1024)          3938304
_________________________________________________________________
dense_3 (Dense)              (64, None, 85)            87125
=================================================================
Total params: 4,047,189
Trainable params: 4,047,189
Non-trainable params: 0
_________________________________________________________________
```

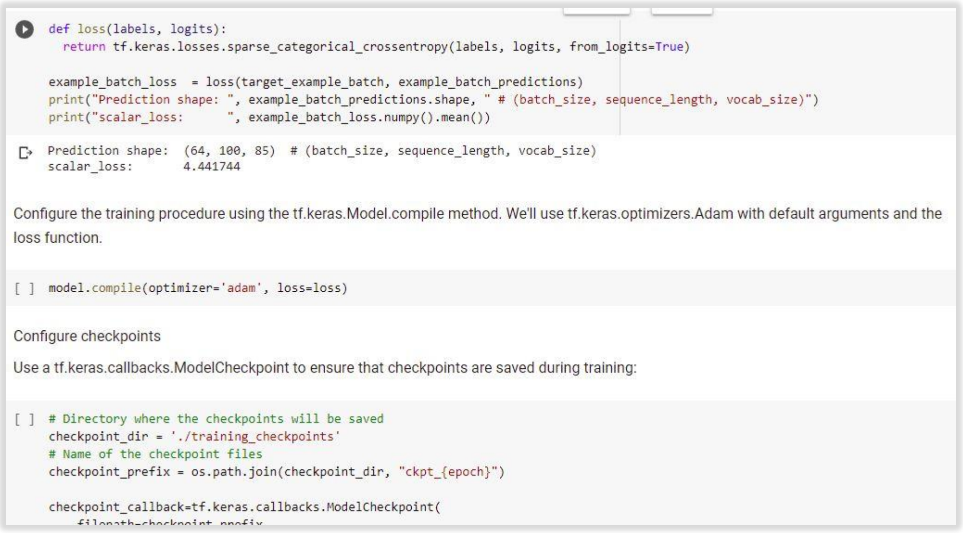
- This the output of sampled indices from the text



```
sampled_indices = tf.random.categorical(example_batch_predictions[0], num_samples=1)
sampled_indices = tf.squeeze(sampled_indices,axis=-1).numpy()
#This gives us, at each timestep, a prediction of the next character index:
sampled_indices
```

```
array([78, 25, 80, 71, 55, 16,  5, 54, 59, 52, 46, 45, 41, 43, 10, 55, 62,
        2, 17, 41, 11, 67,  1, 16, 58, 74, 42, 11,  2, 62,  2, 72, 48, 46,
        7, 79, 84,  8,  3, 82, 28, 28, 84, 62,  9, 77, 41, 29, 73, 19, 45,
       13, 62, 35, 84, 45,  5, 82, 65, 59, 14, 70, 33, 35, 81, 10, 21, 84,
       17,  6, 72, 61, 27, 24, 80, 23, 10, 48, 16, 69, 11,  0, 81, 84, 52,
       21, 23, 53, 31, 29, 21, 24, 45, 44, 19, 70, 56, 30, 44, 65])
```

- Define loss function and print the prediction shape and scalar loss values. Finally compile the model and create ModelCheckpoint to record best training weights.



```
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, 85)  # (batch_size, sequence_length, vocab_size)
scalar_loss:       4.441744
```

Configure the training procedure using the tf.keras.Model.compile method. We'll use tf.keras.optimizers.Adam with default arguments and the loss function.

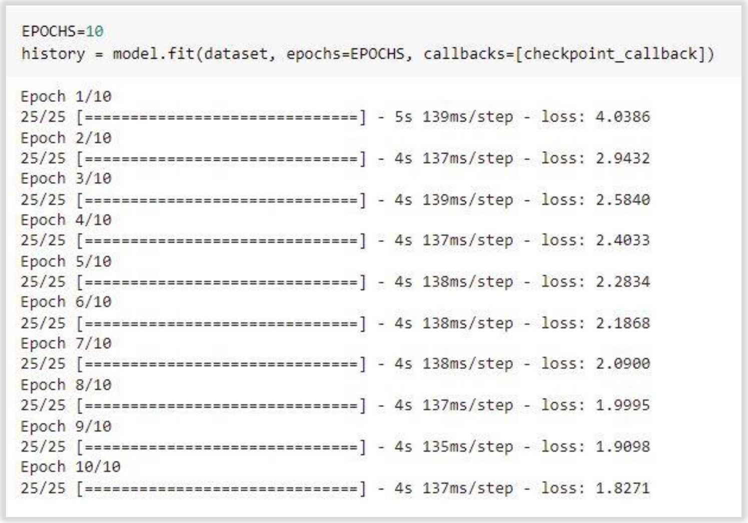
```
[ ]  model.compile(optimizer='adam', loss=loss)
```

Configure checkpoints

Use a tf.keras.callbacks.ModelCheckpoint to ensure that checkpoints are saved during training:

```
[ ]  # 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,
```

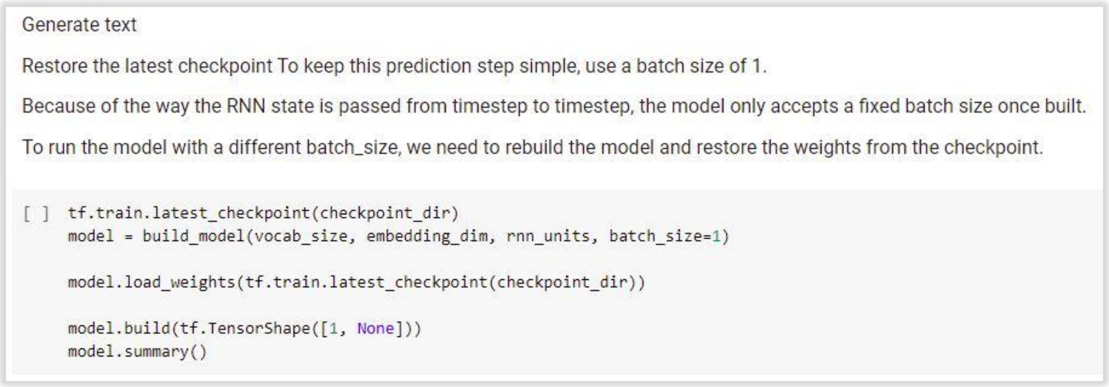
- Fit the model



```
EPOCHS=10
history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])
```

```
Epoch 1/10
25/25 [==============================] - 5s 139ms/step - loss: 4.0386
Epoch 2/10
25/25 [==============================] - 4s 137ms/step - loss: 2.9432
Epoch 3/10
25/25 [==============================] - 4s 139ms/step - loss: 2.5840
Epoch 4/10
25/25 [==============================] - 4s 137ms/step - loss: 2.4033
Epoch 5/10
25/25 [==============================] - 4s 138ms/step - loss: 2.2834
Epoch 6/10
25/25 [==============================] - 4s 138ms/step - loss: 2.1868
Epoch 7/10
25/25 [==============================] - 4s 138ms/step - loss: 2.0900
Epoch 8/10
25/25 [==============================] - 4s 137ms/step - loss: 1.9995
Epoch 9/10
25/25 [==============================] - 4s 135ms/step - loss: 1.9098
Epoch 10/10
25/25 [==============================] - 4s 137ms/step - loss: 1.8271
```

- Finally, now that we have recorded weights for each epoch. Recreate the model using latest checkpoint as it has the best loss value.



Generate text

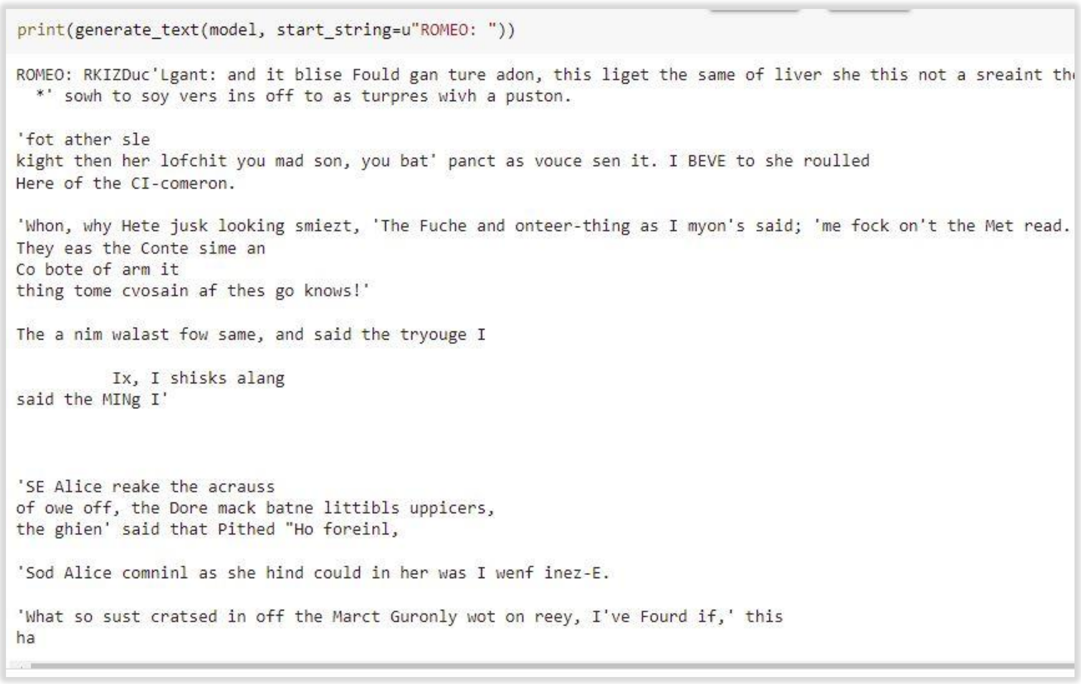
Restore the latest checkpoint To keep this prediction step simple, use a batch size of 1.

Because of the way the RNN state is passed from timestep to timestep, the model only accepts a fixed batch size once built.

To run the model with a different batch_size, we need to rebuild the model and restore the weights from the checkpoint.

```
[ ]  tf.train.latest_checkpoint(checkpoint_dir)
     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]))
     model.summary()
```

- Call Generate text function for text generation using the trained model provided in use case.



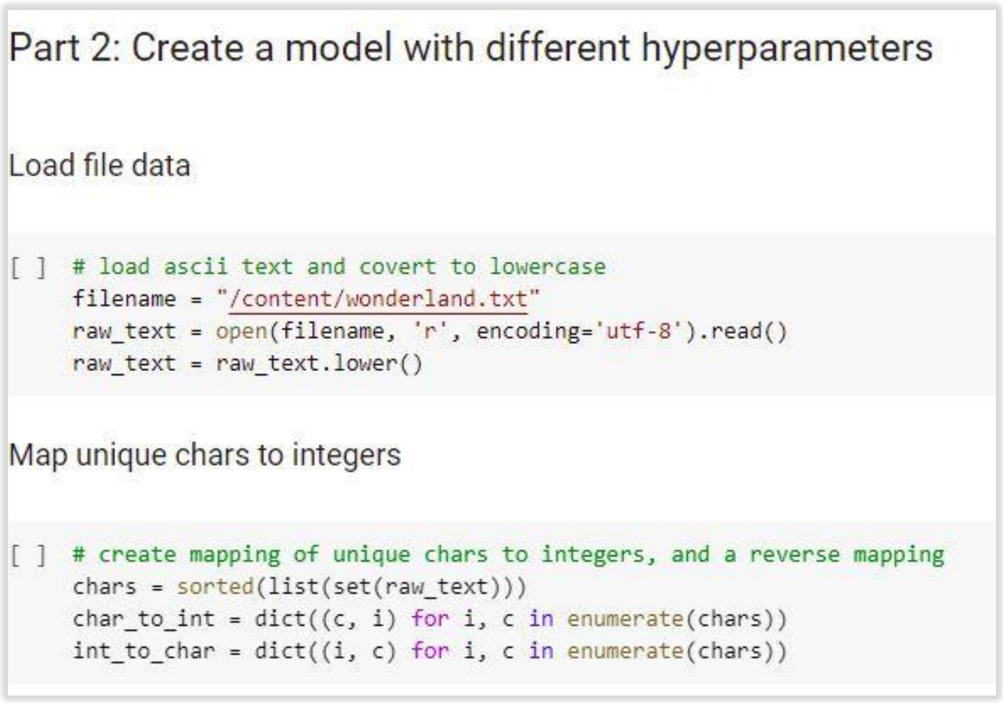
```
print(generate_text(model, start_string=u"ROMEO: "))

ROMEO: RKIZDuc'Lgant: and it blise Fould gan ture adon, this liget the same of liver she this not a sreaint th
  *' sowh to soy vers ins off to as turpres wivh a puston.

'fot ather sle
kight then her lofchit you mad son, you bat' panct as vouce sen it. I BEVE to she roulled
Here of the CI-comeron.

'Whon, why Hete jusk looking smiezt, 'The Fuche and onteer-thing as I myon's said; 'me fock on't the Met read.
They eas the Conte sime an
Co bote of arm it
thing tome cvosain af thes go knows!'

The a nim walast fow same, and said the tryouge I

          Ix, I shisks alang
said the MINg I'



'SE Alice reake the acrauss
of owe off, the Dore mack batne littibls uppicers,
the ghien' said that Pithed "Ho foreinl,

'Sod Alice comninl as she hind could in her was I wenf inez-E.

'What so sust cratsed in off the Marct Guronly wot on reey, I've Found if,' this
ha
```

- Final output generated from first model:

```
ROMEO: RKIZDuc'Lgant: and it blise Fould gan ture adon, this liget the
same of liver she this not a sreaint the turk I door than to founs for off
trie Minch Hartel wentwhtrgain the was sien), of coint Thet to see tly
tome toot,
  *' sowh to soy vers ins off to as turpres wivh a puston.

'fot ather sle
kight then her lofchit you mad son, you bat' panct as vouce sen it. I BEVE
to she roulled
Here of the CI-comeron.

'Whon, why Hete jusk looking smiezt, 'The Fuche and onteer-thing as I
myon's said; 'me fock on't the Met read.
They eas the Conte sime an
Co bote of arm it
thing tome cvosain af thes go knows!'

The a nim walast fow same, and said the tryouge I

          Ix, I shisks alang
said the MINg I'



'SE Alice reake the acrauss
of owe off, the Dore mack batne littibls uppicers,
the ghien' said that Pithed "Ho foreinl,

'Sod Alice comninl as she hind could in her was I wenf inez-E.

'What so sust cratsed in off the Marct Guronly wot on reey, I've Fourd
if,' this
ha
```

2. Model Evaluation after changing the hyperparameters

- Reading the data from txt file and mapping unique chars to integers



Part 2: Create a model with different hyperparameters

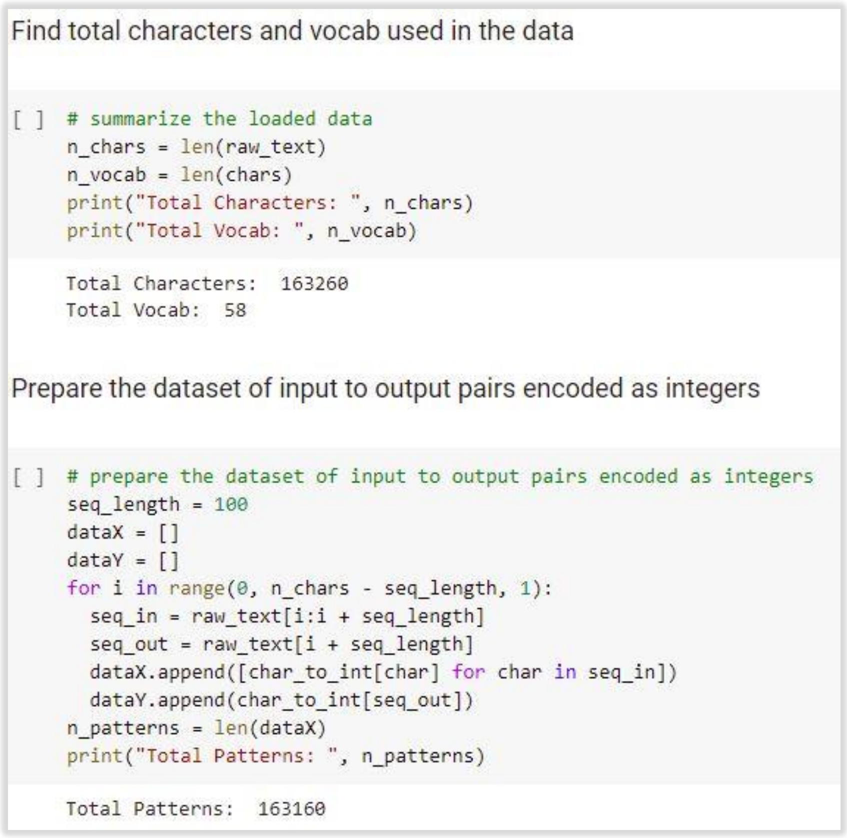
Load file data

```
[ ] # load ascii text and covert to lowercase
    filename = "/content/wonderland.txt"
    raw_text = open(filename, 'r', encoding='utf-8').read()
    raw_text = raw_text.lower()
```

Map unique chars to integers

```
[ ] # create mapping of unique chars to integers, and a reverse mapping
    chars = sorted(list(set(raw_text)))
    char_to_int = dict((c, i) for i, c in enumerate(chars))
    int_to_char = dict((i, c) for i, c in enumerate(chars))
```

- Find the total number of vocab used in the data with word counts



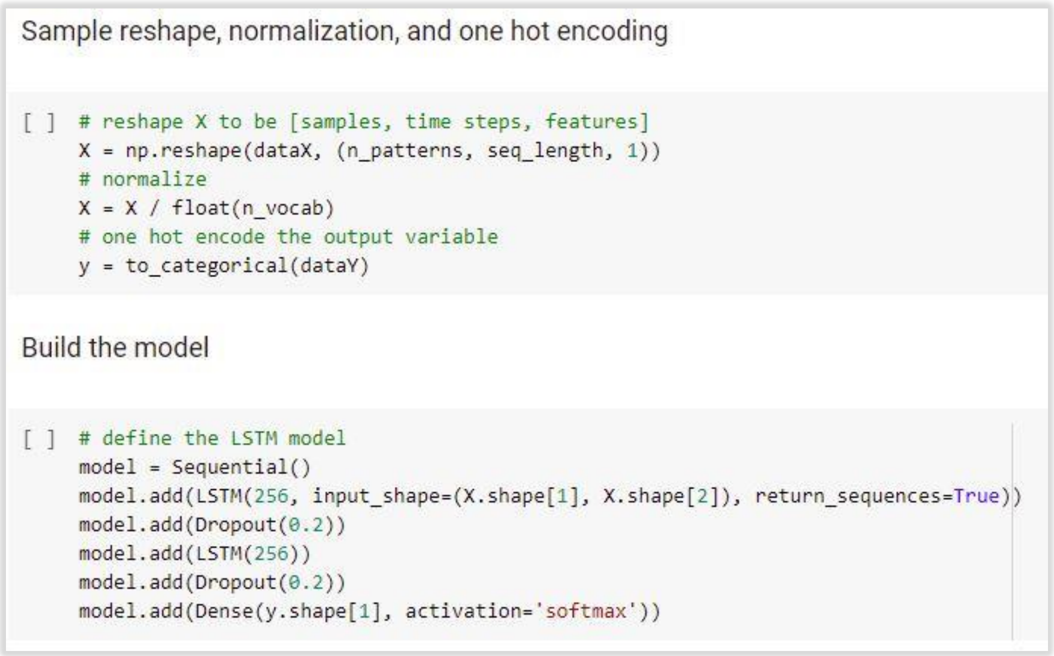
Find total characters and vocab used in the data

```
[ ] # summarize the loaded data
    n_chars = len(raw_text)
    n_vocab = len(chars)
    print("Total Characters: ", n_chars)
    print("Total Vocab: ", n_vocab)

    Total Characters:  163260
    Total Vocab:  58
```

Prepare the dataset of input to output pairs encoded as integers

```
[ ] # prepare the dataset of input to output pairs encoded as integers
    seq_length = 100
    dataX = []
    dataY = []
    for i in range(0, n_chars - seq_length, 1):
      seq_in = raw_text[i:i + seq_length]
      seq_out = raw_text[i + seq_length]
      dataX.append([char_to_int[char] for char in seq_in])
      dataY.append(char_to_int[seq_out])
    n_patterns = len(dataX)
    print("Total Patterns: ", n_patterns)

    Total Patterns:  163160
```

- Create reshape samples, normalization, and one-hot coding. Finally build the model



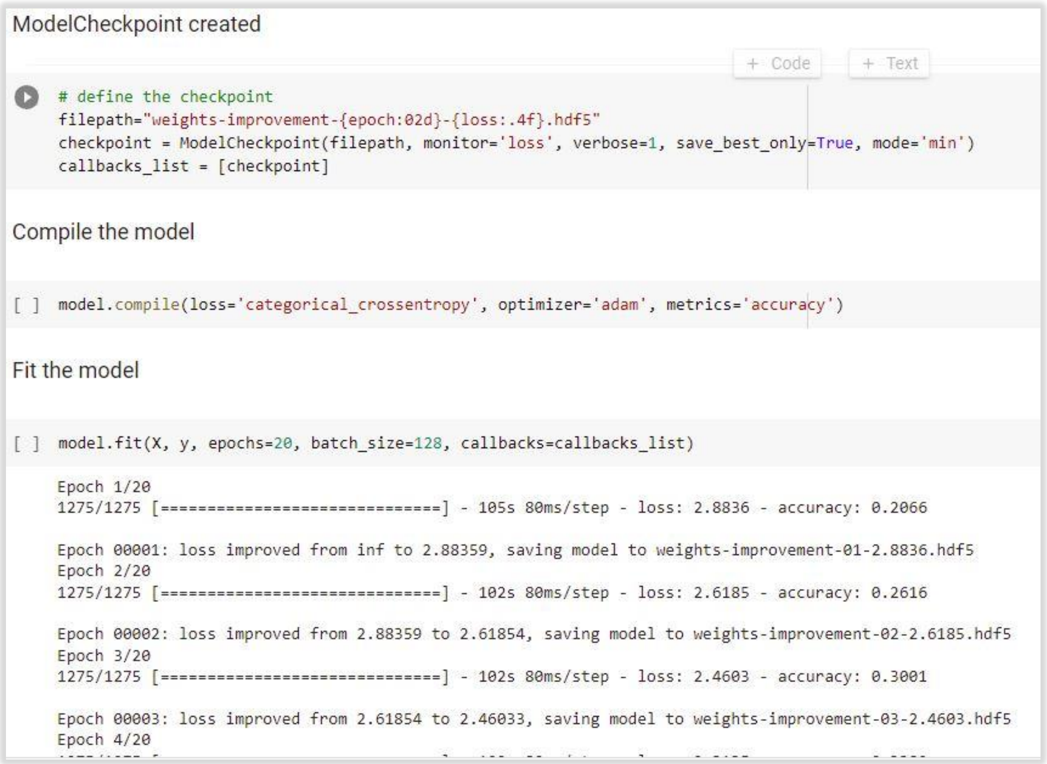
Sample reshape, normalization, and one hot encoding

```
[ ] # reshape X to be [samples, time steps, features]
    X = np.reshape(dataX, (n_patterns, seq_length, 1))
    # normalize
    X = X / float(n_vocab)
    # one hot encode the output variable
    y = to_categorical(dataY)
```

Build the model

```
[ ] # define the LSTM model
    model = Sequential()
    model.add(LSTM(256, input_shape=(X.shape[1], X.shape[2]), return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(256))
    model.add(Dropout(0.2))
    model.add(Dense(y.shape[1], activation='softmax'))
```

- Create model checkpoint to record the weights along the progress of epochs. Fit the model after compiling it.



ModelCheckpoint created

```
# define the checkpoint
filepath="weights-improvement-{epoch:02d}-{loss:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint]
```

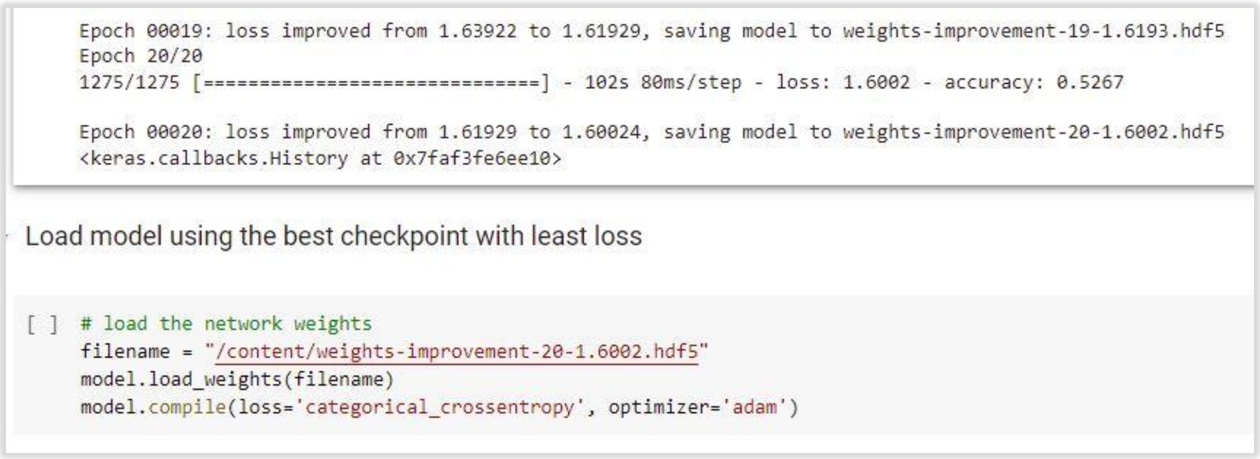
Compile the model

```
[ ] model.compile(loss='categorical_crossentropy', optimizer='adam', metrics='accuracy')
```

Fit the model

```
[ ] model.fit(X, y, epochs=20, batch_size=128, callbacks=callbacks_list)

    Epoch 1/20
    1275/1275 [==============================] - 105s 80ms/step - loss: 2.8836 - accuracy: 0.2066

    Epoch 00001: loss improved from inf to 2.88359, saving model to weights-improvement-01-2.8836.hdf5
    Epoch 2/20
    1275/1275 [==============================] - 102s 80ms/step - loss: 2.6185 - accuracy: 0.2616

    Epoch 00002: loss improved from 2.88359 to 2.61854, saving model to weights-improvement-02-2.6185.hdf5
    Epoch 3/20
    1275/1275 [==============================] - 102s 80ms/step - loss: 2.4603 - accuracy: 0.3001

    Epoch 00003: loss improved from 2.61854 to 2.46033, saving model to weights-improvement-03-2.4603.hdf5
    Epoch 4/20
```

- Below screenshot shows the best checkpointed weight saved on the 20th epoch. So, once again load the model using the same weights and compile it.

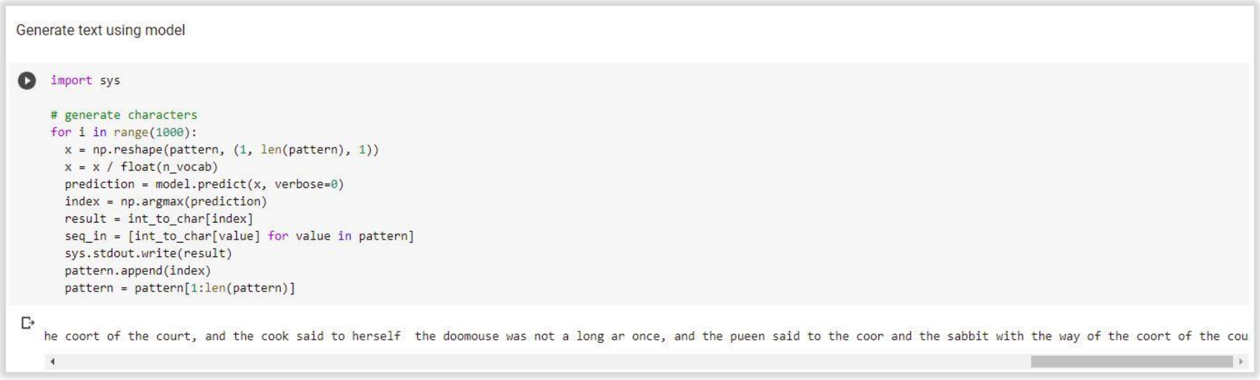


```
Epoch 00019: loss improved from 1.63922 to 1.61929, saving model to weights-improvement-19-1.6193.hdf5
Epoch 20/20
1275/1275 [==============================] - 102s 80ms/step - loss: 1.6002 - accuracy: 0.5267

Epoch 00020: loss improved from 1.61929 to 1.60024, saving model to weights-improvement-20-1.6002.hdf5
<keras.callbacks.History at 0x7faf3fe6ee10>
```

Load model using the best checkpoint with least loss

```
[ ] # load the network weights
    filename = "/content/weights-improvement-20-1.6002.hdf5"
    model.load_weights(filename)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
```

- Now use the same model with best performance, to generate text.



Generate text using model

```
import sys

# generate characters
for i in range(1000):
  x = np.reshape(pattern, (1, len(pattern), 1))
  x = x / float(n_vocab)
  prediction = model.predict(x, verbose=0)
  index = np.argmax(prediction)
  result = int_to_char[index]
  seq_in = [int_to_char[value] for value in pattern]
  sys.stdout.write(result)
  pattern.append(index)
  pattern = pattern[1:len(pattern)]
```

```
he coort of the court, and the cook said to herself  the doomouse was not a long ar once, and the pueen said to the coor and the sabbit with the way of the coort of the cou
```

- Below is the output of Text Generation from the second improved model.

```
'would not a courte,' she mock turtle raid to herself, 'i don't know the
doomouse was the doom and then so her to the sabbit! would not a
cortersation, and the pueen said to the coor and the sabbit with the way
of the coort of the court, and the cook said to herself  the doomouse was
not a long ar once, and the pueen said to the coor and the sabbit with the
way of the coort of the court, and the cook said to herself  the doomouse
was not a long ar once, and the pueen said to the coor and the sabbit with
the way of the coort of the court, and the cook said to herself  the
doomouse was not a long ar once, and the pueen said to the coor and the
sabbit with the way of the coort of the court, and the cook said to
herself  the doomouse was not a long ar once, and the pueen said to the
coor and the sabbit with the way of the coort of the court, and the cook
said to herself  the doomouse was not a long ar once, and the pueen said
to the coor and the sabbit with the way of the coort of the cou
```

3. Final observation comments

We can successfully conclude that our model 2 performed very well as compared to the existing use case model 1. We got text generation with less loss value and also the text generated doesn't seem to be so random but meaningful.

The reasons are as below:

1. For building the Model – We used **denser LSTM layers to memorize the context in the text for short time.**
2. For Activation Function – We used **categorical_crossentropy** loss function of (most useful function for multi-class problems and text processing), **Softmax activation function(for final classification)for the output layer.**
3. For handling Overfitting (Regularizing) – **We used DropOut Layers** to get rid of extra neurons.
4. **Use of a greater number of epochs to train the data accurately** proved to be the best fit.
5. We used the callbacks such as ModelCheckpoint to use the best model performance and generate text using it.

Video Link

- https://youtu.be/HJzqFEcXiek

Conclusion

1. Lessons Learnt

- I developed a deep understanding of the text generation models after doing this ICP.
- The LSTM and RNN model implementation vary a little bit. The choice of both varies with respect to context of the data.

2. Challenges Faced

- To improve the accuracy of existing model, use of correct hyperparameters like activation function, convolutional layers, and number of epochs was a challenge. I did build the model a few times and understood which can be a better fit.
- I developed understanding of LSTM function which is mostly used for short term context in the text.