**WIKI REPORT**

**What I had learnt from ICP:**

* From this ICP I’ve learned about the various concepts of Bigdata such as Data augmentation and batch normalization.
* For programming I had been used google colab to import all the dependencies and work according to the program
* From this ICP I have learnt different activation functions available for hidden and output layers in neural networks.
* Usage of keras in the model has helped me to know more about the deep learning concepts.
* Hence by using all the libraries and programming logic I had obtained the model and learnt about model building.

**ICP Description:**

* In this ICP the task performed is the text generationon dataset, using a new and good dataset.
* Used a different data and used the model provided in ICP6 to perform text generation. We made 4 changes and added LSTM layers to model, changed hyperparameters in the source code.

**Challenges Faced During ICP:**

* Found difficulty in understanding types of parameters and in identifying the parameters that are required for the classification.
* Found difficulty in identifying correct combination of parameters to be passed to the classifiers.
* The number of epochs has been changed many times, which is why the classifier took so long.
* While loading the Wikipedia dataset I felt difficult.
* Choosing the best optimizer made me feel difficult among the adam algorithms.
* Resolving all the challenges and difficulties the final output of code is obtained as expected.

**LSTM LAYERS:**

We added LSTM layers to model

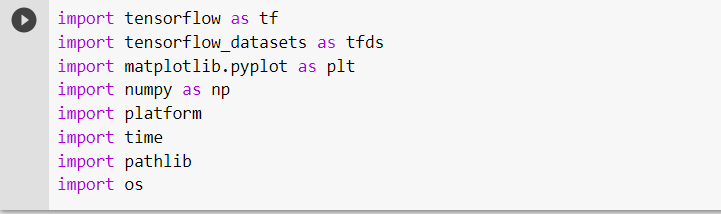
* tf.keras.layers.Embedding: The input layer. A trainable lookup table that will map the numbers of each character to a vector with embedding\_dim dimensions.
* tf.keras.layers.GRU:  A type of RNN with size units=rnn\_units (You can also use a LSTM layer here.)
* tf.keras.layers.Dense: The output layer, with vocab\_size outputs.

**The 4 hyper parameters changed in the source code are all related to the classifier.**

1. LSTM is a particular type of RNN with a mechanism to avoid the vanishing gradient problem and learn long term. LSTM is also capable of learning short term dependencies. This LSTM is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. This can be used to make predictions based on time series data.
2. Plotting - For each character the model looks up the embedding, runs the GRU one time step with the embedding as input, and applies the dense layer to generate logits predicting the log-likelihood of the next character.
3. Here, we changed the total number of steps (batches of samples) to yield from generator before declaring one epoch finished and starting the next epoch. steps\_per\_epoch is batches of samples to train. It is used to define how many batches of samples to use in one epoch. It is used to declaring one epoch finished and starting the next epoch.
4. We used prefectch function,Prefetching overlaps the preprocessing and model execution of a training step. Doing so reduces the step time to the maximum of the training and the time it takes to extract the data. The tf.data API provides the tf.data.Dataset.prefetch transformation. It can be used to decouple the time when data is produced from the time when data is consumed. In particular, the transformation uses a background thread and an internal buffer to prefetch elements from the input dataset ahead of the time they are requested.

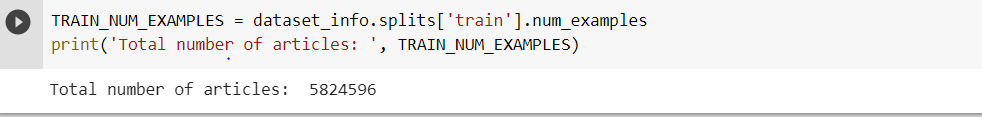
**Steps of Execution:**

* In the first step, import all the dependencies required for the code, such as tensorflow, tensorflow\_datasets, matplotlib.pyplot, numpy, platform, time, pathlib and os.



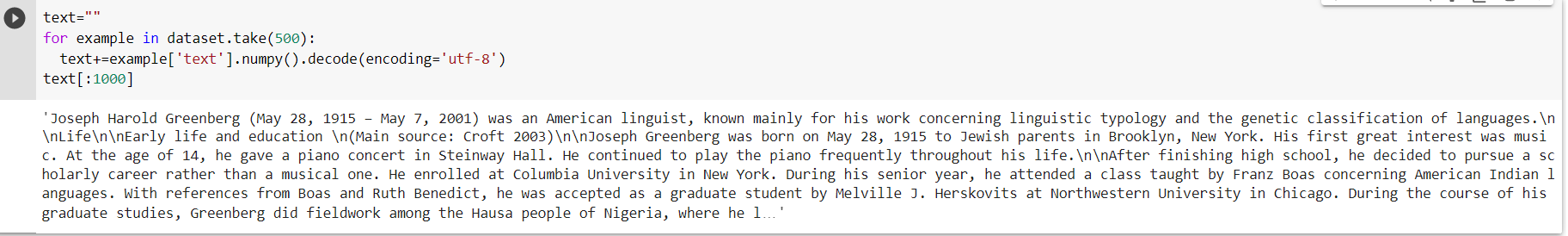
* Downloaded the data from the Wikipedia dataset. The data from dataset is loaded into training and printed total number of articles in it.



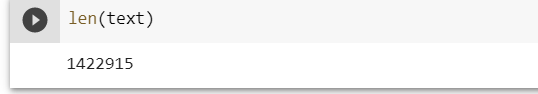


* Read the data

Taken 500 characters in text



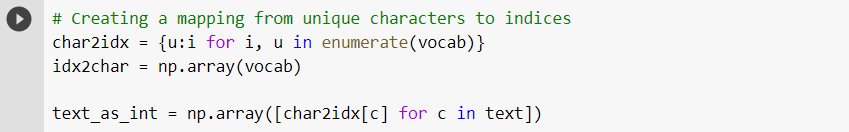
* Length of the text



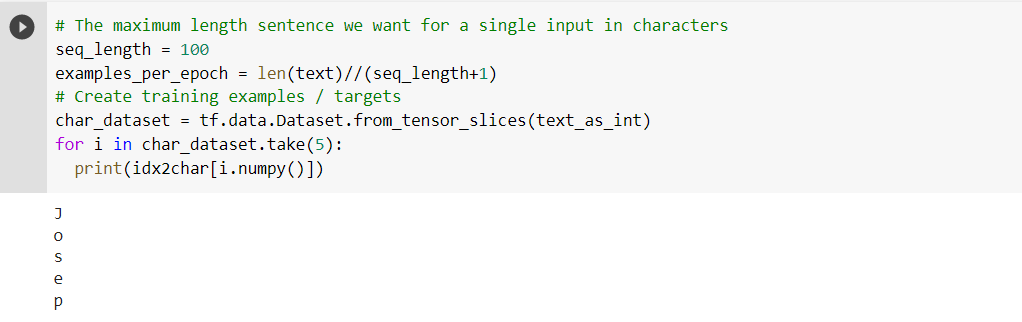
* Printed the number of unique characters



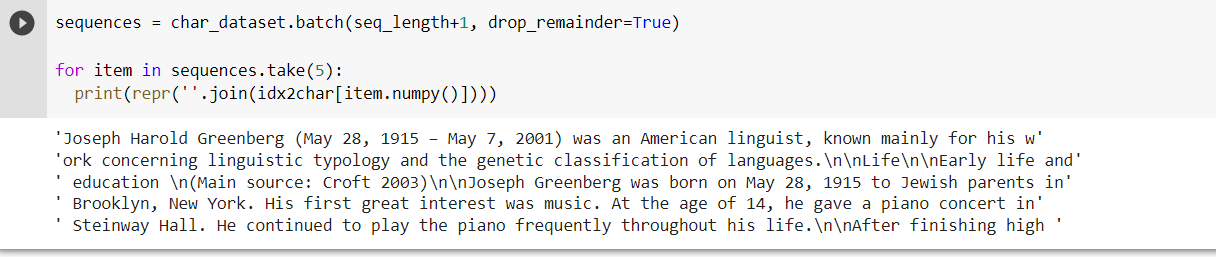
* Processed and vectorized text



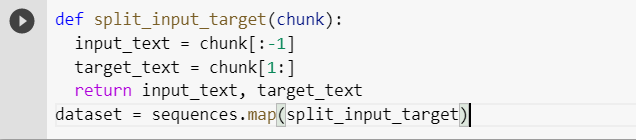
* The maximum length sentence we want for a single input in characters and created training examples/targets.



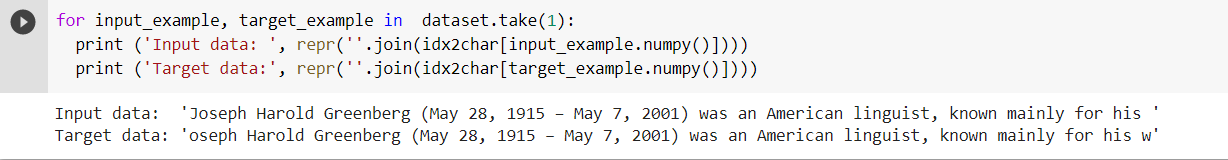
* By using batch method we converted these individual characters to sequences of the desired size.



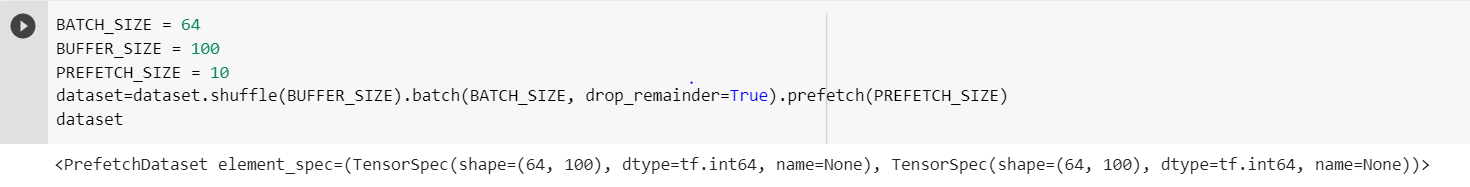
* Map method is applied to a simple function to each batch for duplicate and shift it to form the input and target text for each sentence.



* Print Input data and Target data



* We created training batches, before that we shuffled the data and packed it into batches.

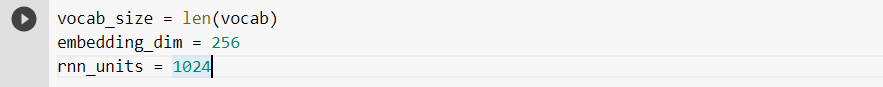


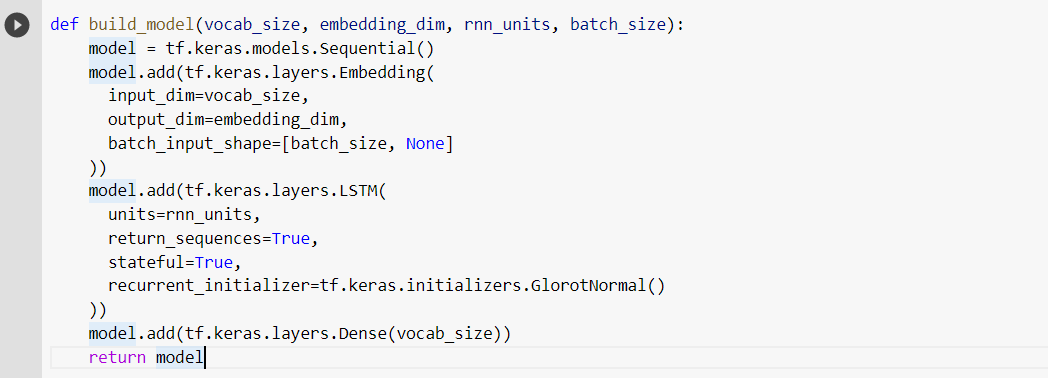
* For defining the model we used tf.keras.Sequential.

tf.keras.layers.Dense

tf.keras.layers.Embedding

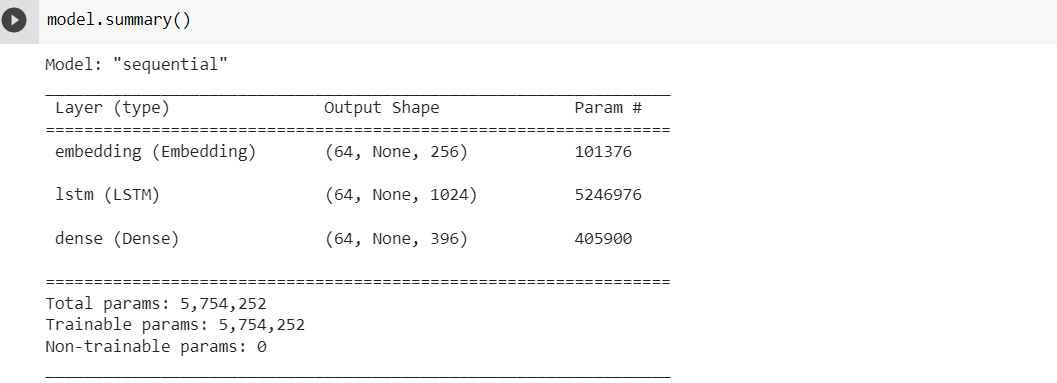
tf.keras.layers.GRU



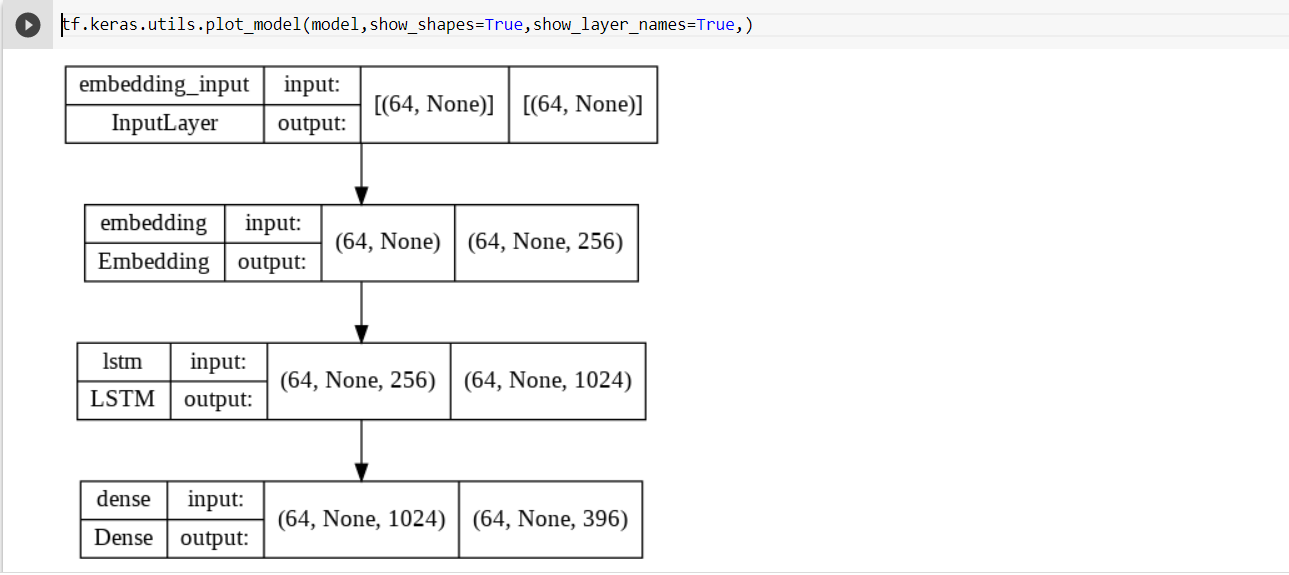




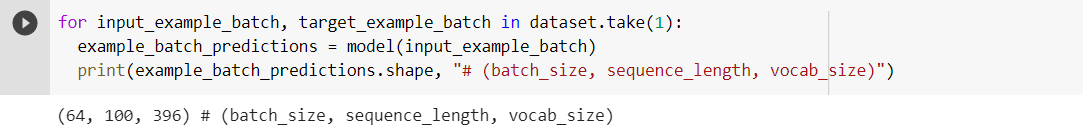
* Added Layers



* Used tf.keras.utils.plot\_model, It allowed to create a visualization of our Keras neural network.

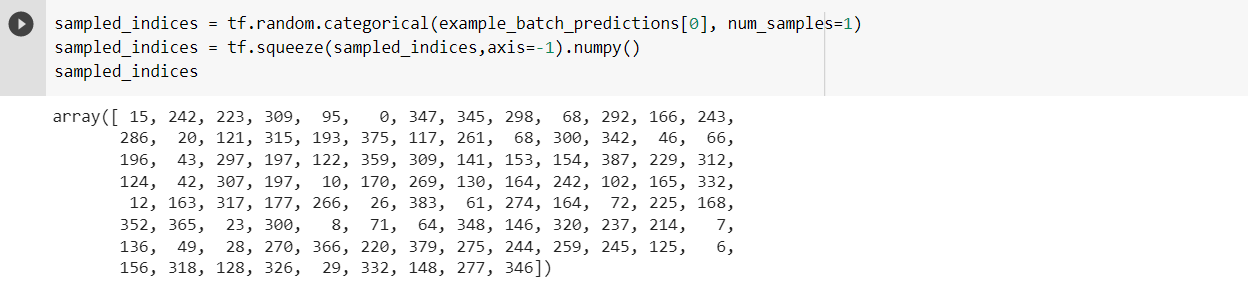


* Now run the model to see that it behaves as expected.



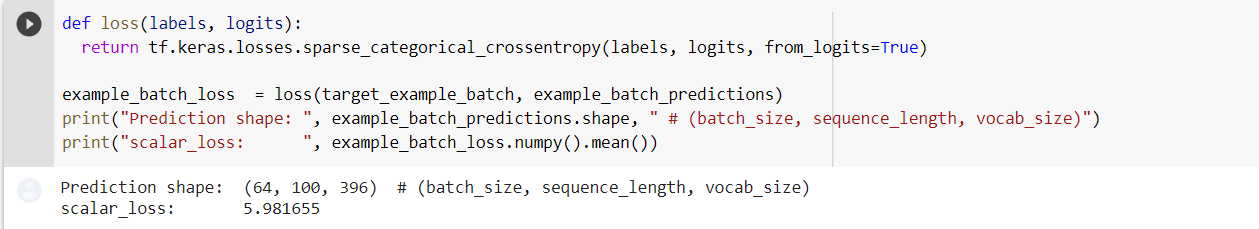
* tf.random categorical: It is used to create a random seed for the distribution. See tf. random. behaviour.

tf. squeeze() function returns a tensor with the same value as its first argument, but a different shape.

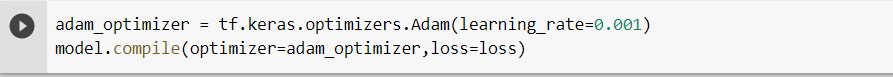


* Attach an optimizer, and a loss function

The standard tf.keras.losses.sparse\_categorical\_crossentropy loss function works in this case because it is applied across the last dimension of the predictions.

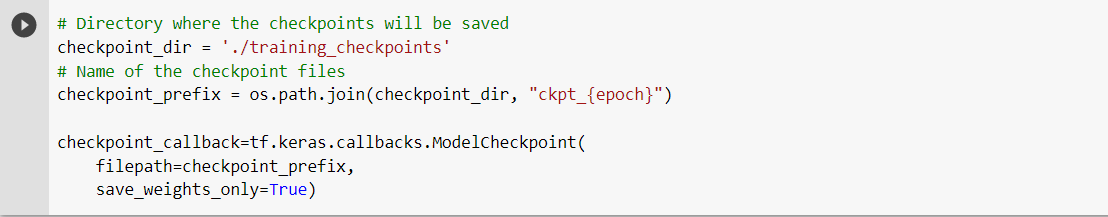


* Adam Optimizer: It is used to accelerate the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients with default arguments and the loss function.

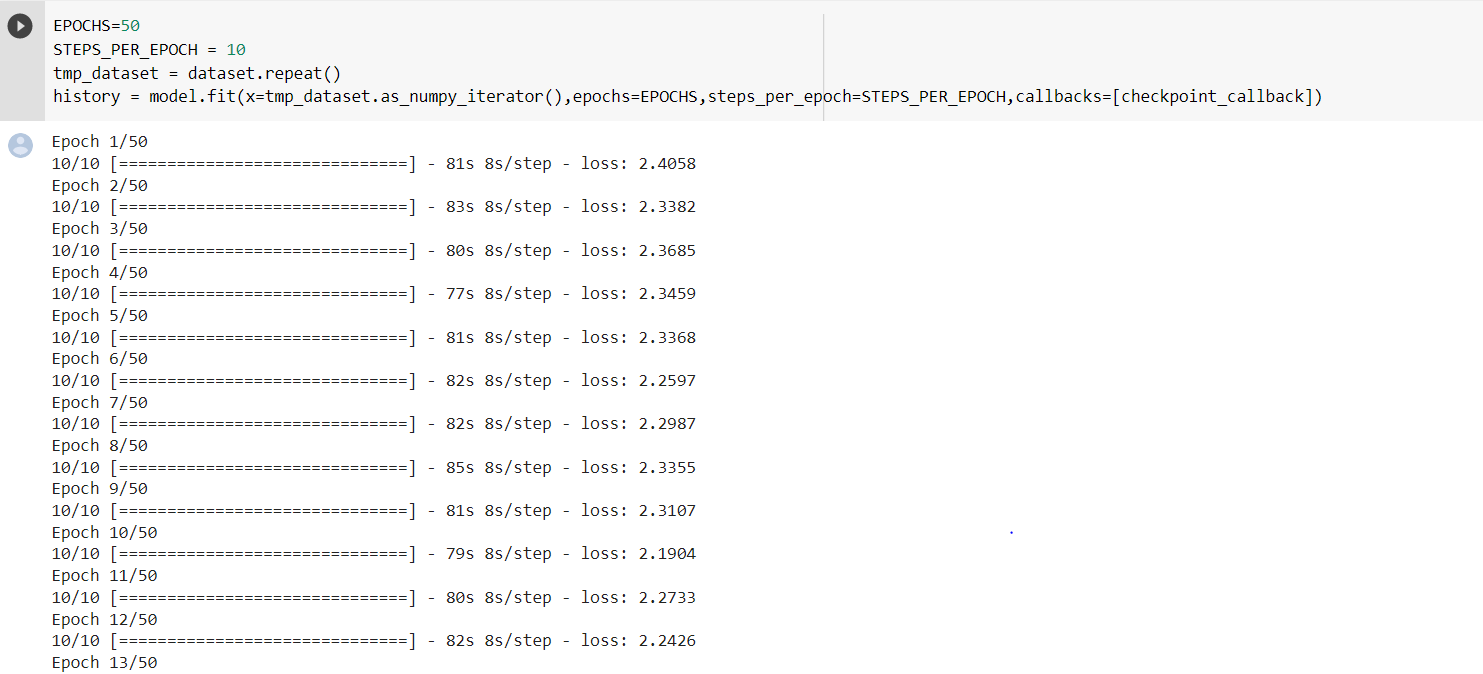


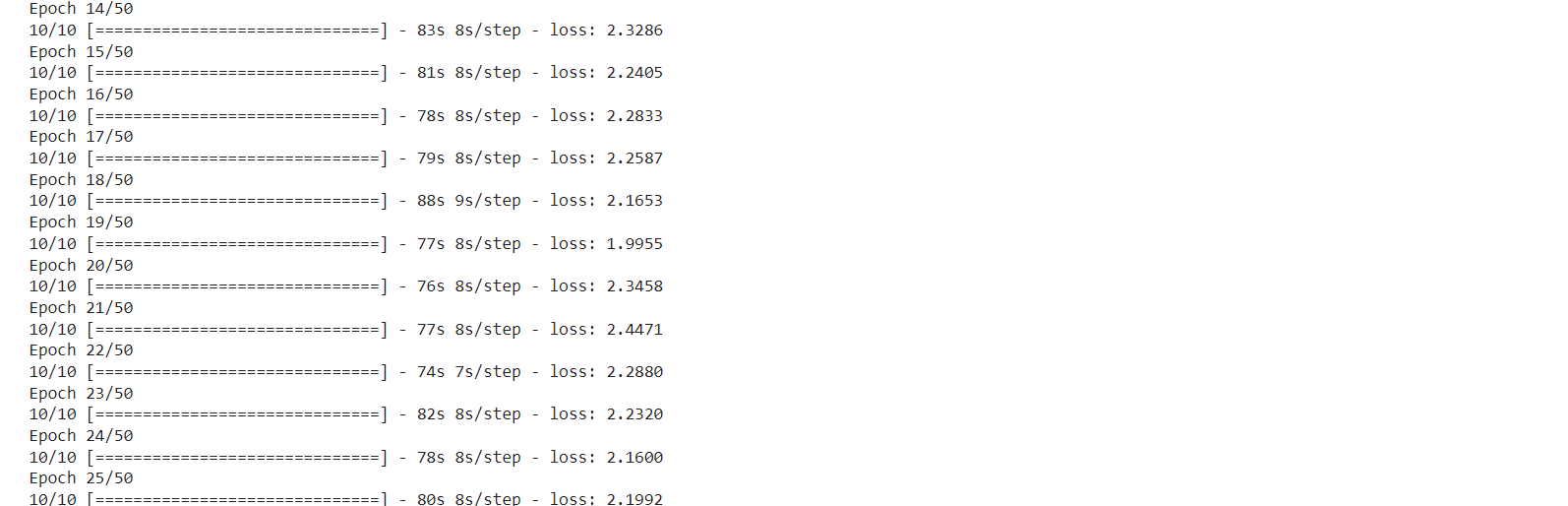
* The checkpoint may be used directly, or used as the starting point for a new run, picking up where it left off.

ModelCheckpoint callback is used in conjunction with training using model.the callback supports saving at the end of every epoch, or after a fixed number of training batches.

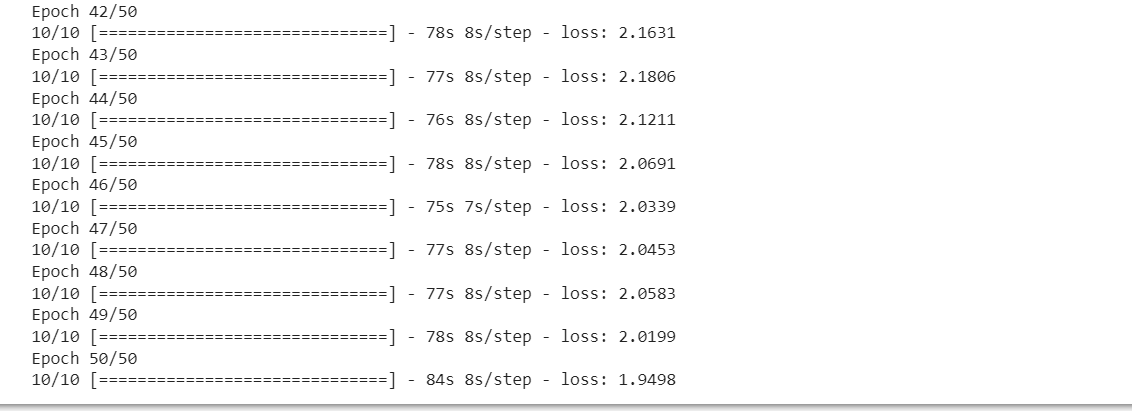


* To train the model we are using 50 epochs to keep training and also we used 10 steps\_per\_epoch.







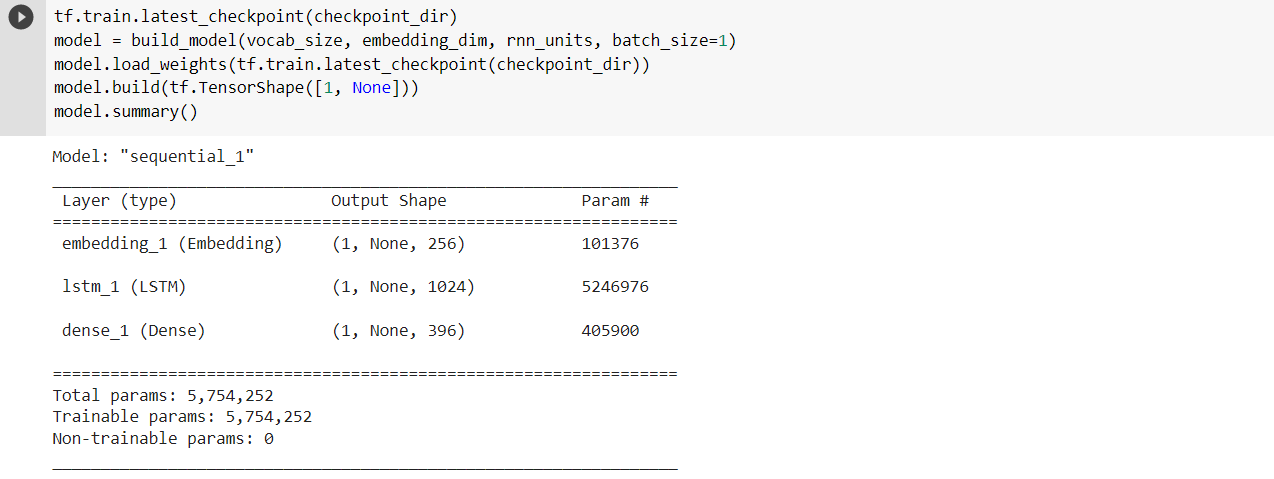


* Using a Keras Long Short-Term Memory (LSTM) Model to Predict Stock Prices.It is a variation of a recurrent neural network (RNN) that is quite effective in predicting the long sequences of data like sentences and stock prices over a period of time.

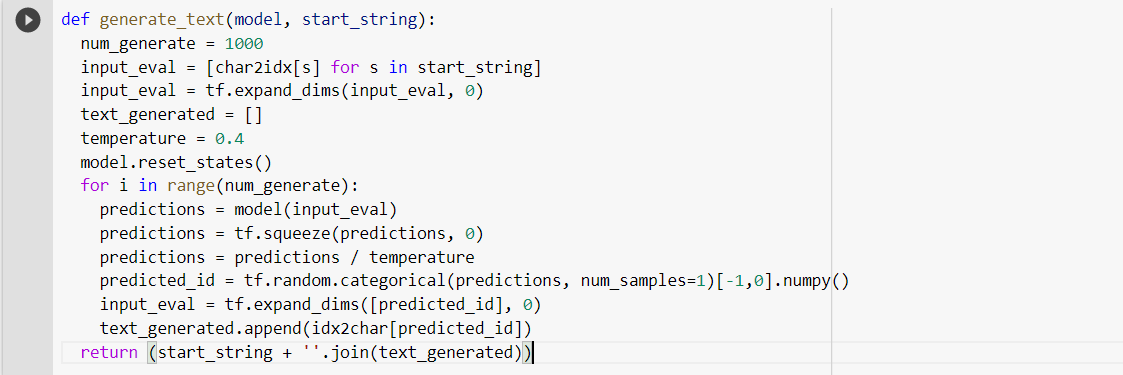
RNN layer uses a for loop to iterate over the timesteps of a sequence, while maintaining an internal state that encodes information about the timesteps it has seen so far.

Embedding layer: It enables us to convert each word into a fixed length vector of defined size.

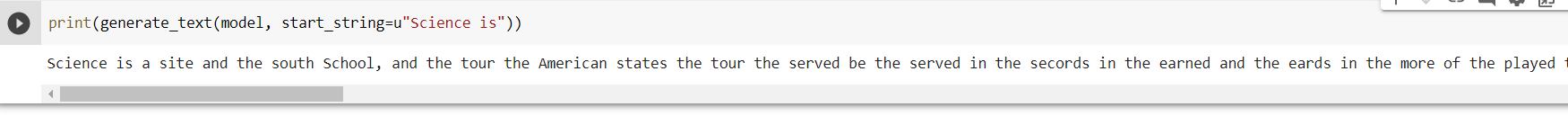
Dense Layer: It is a normal fully connected layer in a neuronal network.It is used to classify image based on output from convolutional layers.



* Number of characters are generated then converted our start string to numbers. we used temperature=0.4 to get results in most predictable text. Categorical distribution is used to predict the character. Here,predicted character passed as the next input to the modelalong with the previous hidden state.



* Print the text-output



**Video link: https://youtu.be/7gZ9WKlHHdQ**

**Report By:**

**Shirisha Baddam – 16331297 – sbfcc@unsystem.edu**

**Maanukonda Muralikrishna – 16330766 - mm5hd@umsystem.edu**