

## 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 behavior 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”

The screenshot shows a Jupyter Notebook titled "week 10 - Text Generation with RNNs" with a last checkpoint of 27/02/2020. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code cell contains the following Python code:

```
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")  
checkpoint_callback=tf.keras.callbacks.ModelCheckpoint(  
    filepath=checkpoint_prefix,  
    save_weights_only=True)
```

Below the code cell, the notebook displays the output of the training process. It starts with the command `EPOCHS=5` and then shows the results of `model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])`. The output shows the progress of the training over 5 epochs, with the loss decreasing from 2.6453 to 1.4523.

**Start the Training**

```
In [102]: EPOCHS=5  
In [103]: history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])
```

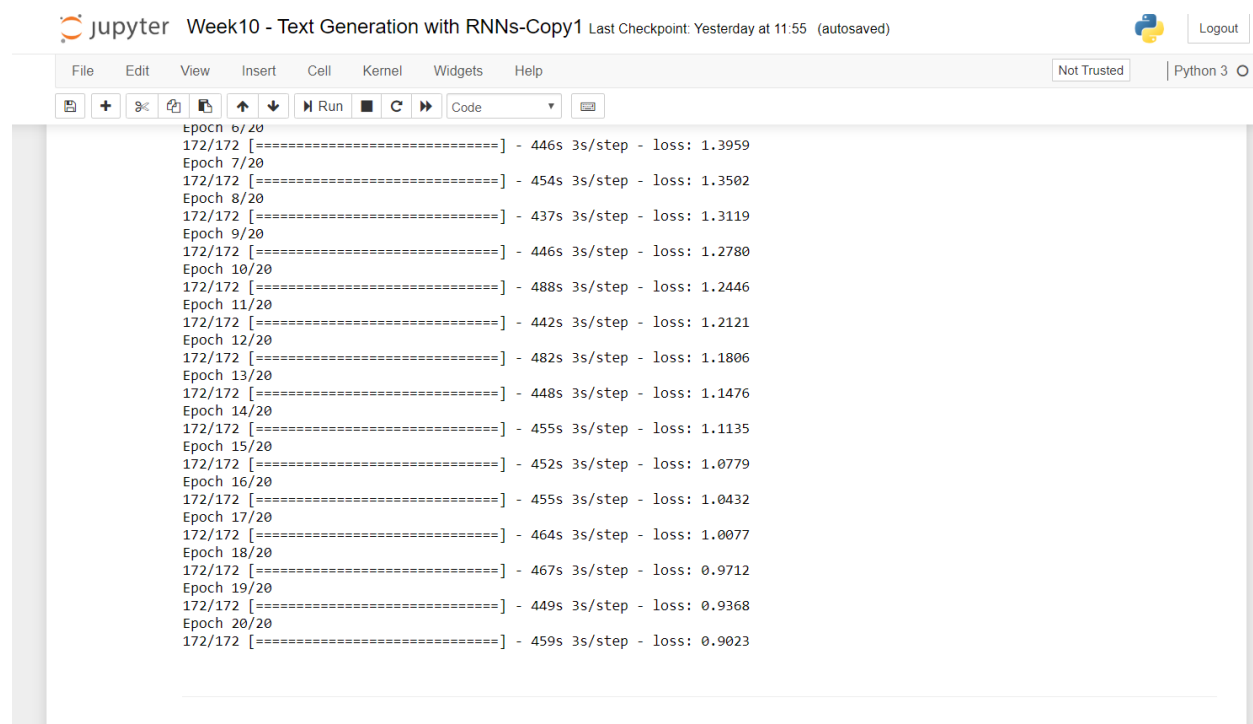
Epoch 1/5  
172/172 [=====] - 449s 3s/step - loss: 2.6453  
Epoch 2/5  
172/172 [=====] - 503s 3s/step - loss: 1.9551  
Epoch 3/5  
172/172 [=====] - 502s 3s/step - loss: 1.6873  
Epoch 4/5  
172/172 [=====] - 484s 3s/step - loss: 1.5393  
Epoch 5/5  
172/172 [=====] - 502s 3s/step - loss: 1.4523

**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”



The screenshot shows a Jupyter Notebook interface with the title "Week10 - Text Generation with RNNs-Copy1". The top bar indicates the last checkpoint was yesterday at 11:55 (autosaved). The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code cell displays the output of a training process over 20 epochs. Each epoch shows a progress bar (172/172), time taken (3s/step), and the loss value. The loss starts at 1.3959 in epoch 6 and decreases steadily to 0.9023 by epoch 20.

Epoch	Progress	Time	Loss
6/20	172/172	446s	1.3959
7/20	172/172	454s	1.3502
8/20	172/172	437s	1.3119
9/20	172/172	446s	1.2780
10/20	172/172	488s	1.2446
11/20	172/172	442s	1.2121
12/20	172/172	482s	1.1806
13/20	172/172	448s	1.1476
14/20	172/172	455s	1.1135
15/20	172/172	452s	1.0779
16/20	172/172	455s	1.0432
17/20	172/172	464s	1.0077
18/20	172/172	467s	0.9712
19/20	172/172	449s	0.9368
20/20	172/172	459s	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.

