Text Generation Using RNN

A. Introduction

a. The report covers the application of a text generation in terms of character level using a RNN (Recurrent Neural Network) with GRU (Gated Recurrent Units). The objective of the project is creating a text based patterns learned from a large dataset. The project aims to:

- Creating a reliable text generation model with TensorFlow
- Cleaning data the data before training and preprocess it
- Data visualization of frequency character distribution
- Reduction of redundancy
- Assessment of model performance through training loss
- Use of temperature-controlled sampling for meaningful text

B. Dataset selection and preprocessing

a. Source

As my system is an old one I didn't have a choice but to pick a small dataset because to run it on google colab each epoch takes 9 hours.

- Moby Dick by Herman Melville
- The Art of War by Niccolo Machiavelli
- Frankenstein by Mary Shelley

These literatures come with well put sentences which is a fair enough to be used as dataset for language modeling.

b. Data Limit

- *For easy processing for my old system I have limited to 20000 words using word_limit=20000*

This is to ensure it doesn't consume too much memory of my system and crash

C. Data Cleaning and preprocessing

a. *Regular expressions*

- Preprocessing applied using regular expressions to get rid of unwanted characters

- Lowercase letters, punctuation and gaps are kept

```
script = script.lower()
script = re.sub(r'[^a-zA-Z\s.,!?;\'\"]', '', script)
```

b. *Joining*

- Used to concatenate string with a separator

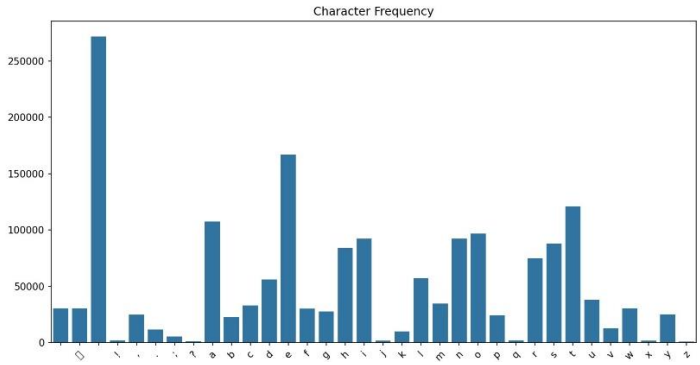
```
script = ' '.join(scripts)
```

D. Data Visualization

a. *Character Frequency Analysis*

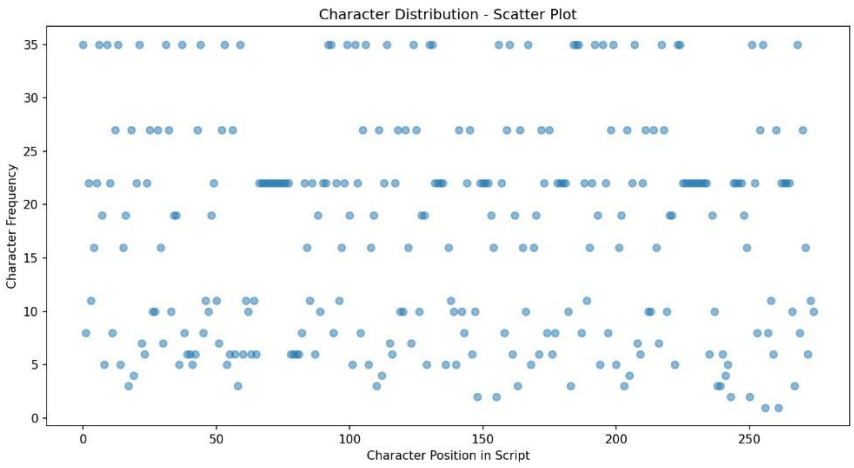
- To represent a clear view of the distribution within the dataset, a chart of bars and a pie chart are generated with matplotlib and seaborn

-Bar Chart: Represents the frequency of each character in the text which provides ins and outs of the dominant characters



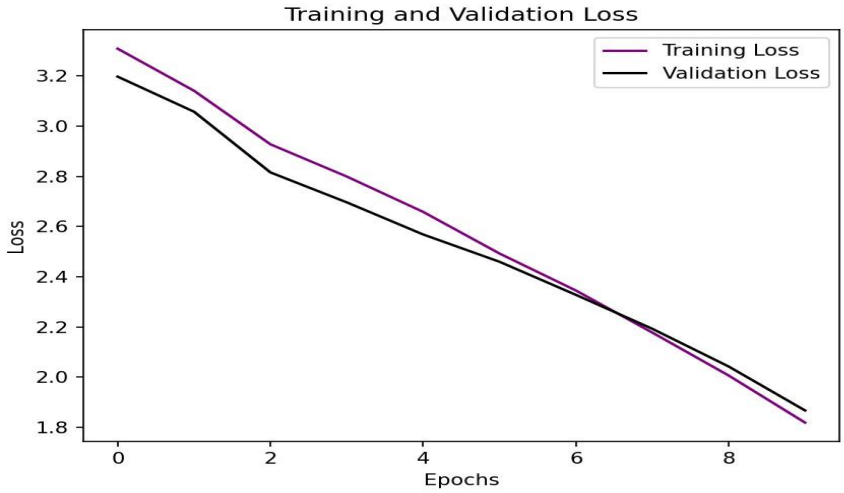
Scattered plot: The scatter plot represents the connection between the position of each character in the script and its frequency. X-axis illustrates the position of each character in the text, while the y-axis represents the frequency of that character at that specific point in the script. The scatter plot illustrates how characters with higher frequencies show up frequently than throughout the text,

showing a relatively erratic pattern depending on where they are positioned within the script. The text limit was set to 100 for faster processing and better visuals just for scatter plot.



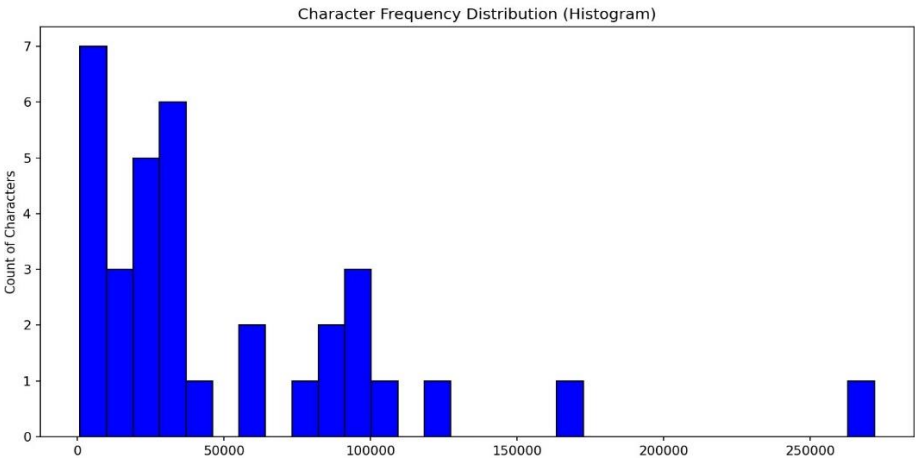
Line graph: This line graph tracks validation loss as well as training during the neural network model is being trained over multiple epochs. The x-axis shows the number of epochs, while the y-axis represents the loss value for both the training and validation data.

Training Loss: The line in purple portray the loss value during training. As the model is understanding, we expect the training loss to decrease over time, indicating the model is fitting better to the training data.



Validation Loss: The line in black shows the value of loss being calculated on the validation set after each epoch. The validation loss is crucial for assessing how well the model acknowledges to unseen data.

- Histogram Chart: This is more related to the visibility of each bar of use of available characters and also essential to understand the text composition



E. Model Construction

a. Embedding Layer:

- Maps out the input to 128 dimension vector

- Essential for the model to be able to understand the connection between characters

```
idn.embedding = tf.keras.layers.Embedding(term_margin, 128)
```

b. GRU Layer:

- A unit of GRU has 512 units

- Captures temporal dependencies in the text

```
idn.gru = tf.keras.layers.GRU(512, return_sequences=True)
```

c. Dense Layer:

- Assumes outputs for individual character in the vocabulary using an interconnected layer

```
idn.dense = tf.keras.layers.Dense(term_margin)
```

F. TRAINING PROCESS

- Data is being divided in sequence (x) and (y) using the function

- Adam optimizer is being used to compile the model and a loss function SparseCategoricalCrossentropy()

- Training is only conducted on 1 because how long it takes in my device to run it through google colab

- The loss over each epoch is visualized using line plot:

```
term_margin = len(eccentric)
prot = scriptModel(term_margin)

# Compilation
prot.compile(optimizer=tf.keras.optimizers.Adam(),
             loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

G. TEMPERATURE-CONTROLLED SAMPLING

- Uses a temperature parameter to manipulate combination of randomness

- Low temperature(<0.5) comes with more predictability

- High temperature(>1.0) comes with creative state and meaningless text

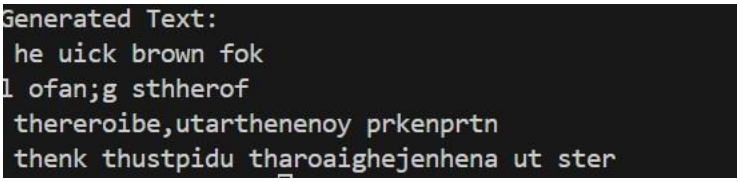
```
def script(prot, initiate_scr, num_chars=100, temp=0.5):
```

H. Results and analysis

- After generating text using the model, initial outputs are assessed based on grammer structure and coherence

- Low temperature(<0.5) comes with more predictability

- High temperature(>1.0) comes with creative state and meaningless text



Issues faced

- Repetition: Reduced by a applying larger GRU reduced repetitive output

- Meaningless Text: Tried to update the character mapping to improve understanding

I. Conclusion

- Improved management of vocabs by using character map

- Stable model training with visualized loss metrics

- Visualization of frequency of character

1. https://www.geeksforgeeks.org/data-science-with-python-tutorial/
2. https://stackoverflow.blog/python/
3. https://www.kaggle.com/code/vipulgandhi/how-to-develop-deep-learning-prots-with-keras
4. https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/