De La Salle University – Manila

Gokongwei College of Engineering

Department of Electronics and Computer Engineering

CpE Elective 3 Laboratory

LBYCPC4

Laboratory Report #4

Text Generation Models

by

Boncodin, Carl Patrick Q.

Chua, Kendrick Dayle J.

Concepcion, Edwin Jr. S.

Evaristo, Gier Bryant J.

Ong, Bryce Erwin D.

LBYCPC4 – EQ1

Introduction and Objectives

Advanced generative models have revolutionized the creation of text that closely resembles human writing. Earlier models like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) were effective for text generation, but they had limitations in capturing relationships across entire sequences, despite their ability to handle some long-range dependencies.

A significant breakthrough came with the introduction of the Transformer model, presented in the “Attention Is All You Need” by Vaswani et. al (2017). This model expanded on the attention mechanism concept and brought about a fundamental shift in how deep learning approaches sequence-based tasks. The Transformer’s innovative design eliminated the need for recurrent networks such as RNNs or LSTMs, instead relying solely on self-attention mechanisms. Since its introduction, the Transformer architecture has become the foundation for cutting-edge models like BERT and GPT, which represent the current state of the art in natural language processing.

This laboratory activity touches on NLP concepts such as the metrics for gauging the performance of the language model like perplexity, or hyperparameters like temperature. Perplexity is a measurement of how well a probability model predicts a sample (Boston University, n.d.). Temperature, on the other hand, is a hyperparameter that controls the randomness when picking words during text prediction. Low values are more deterministic, good for tasks requiring accuracy or factual responses, while high values are more random but could be useful for creative tasks or generating varied responses (De La Vega, 2023).

This activity is composed of two parts: A. Text Generation with RNN Models and B. Text Generation with Transformer Model. The activity is composed of exercises that would provide hands-on training with LSTM or GRU and Transformer-based language models which would help students to familiarize with NLP concepts and compare the differences between the NLP models.

Objectives:

- Familiarize with the architecture of LSTM and GRU models
- Familiarize with the transformer architecture
- Build and train various text generation models using deep learning framework
- Assess the performance of the model's ability to generate text accurately

Procedure/Methodology

A. Text Generation with RNN Models

```
# Get the wiki-auto dataset. Load the wiki_auto/manual configuration
# https://www.tensorflow.org/datasets/catalog/wiki_auto
# Save the dev split to variable ds_train
# Save the test split to variable ds_test
# Save the dataset info to variable ds_info
### --YOUR CODE HERE-- ###
ds_dev, ds_info = tfds.load('wiki_auto/manual', with_info=True, split= ['dev', 'test'])
ds_train = ds_dev[0]
ds_test = ds_dev[1]
# Preprocess the dataset to include only the 'simple_sentence' feature
ds_train = ds_train.map(lambda x: x["simple_sentence"])
ds_test = ds_test.map(lambda x: x["simple_sentence"])

ds_train_text = str()
for text in ds_train:
  ds_train_text += text.numpy().decode() + "\n"

ds_test_text = str()
for text in ds_test:
  ds_test_text += text.numpy().decode() + "\n"

# Display at least five (5) sample texts from the dataset, one per row
### --YOUR CODE HERE-- ###
for sample in ds_train.take(5):
    print(sample.numpy().decode())
```

Figure 1. Data Acquisition and Preprocessing of Text Generation with RNN Models

The first step in the process is acquiring and preparing the WikiAuto dataset, which contains simplified sentences for use in training the text generation model. The dataset is loaded using TensorFlow Datasets (TFDS), and split into development (training) and test sets. The dataset is further preprocessed by extracting the specific feature, simple_sentence, which will be used for training. The dataset is filtered to only include simplified sentences, which will be used for training the model to predict meaningful next-word or character sequences.

```
# Import functions and classes from Keras library
from keras.utils import to_categorical

# Specify the sequence length
seq_len = 50

# Define the vocabulary dictionaries
vocabulary = sorted(list(set(ds_train_text + ds_test_text)))
chr2idx = {c: i for i, c in enumerate(vocabulary)}
idx2chr = {i: c for i, c in enumerate(vocabulary)}
vocab_size = len(vocabulary)

# Build the training dataset
slide_size = 10
input_tokens, target_tokens = [], []
for i in range(0, len(ds_train_text) - seq_len, slide_size):
  input_tokens.append([chr2idx[ch] for ch in ds_train_text[i:i + seq_len]])
  target_tokens.append(chr2idx[ds_train_text[i + seq_len]])

seqX = tf.convert_to_tensor(input_tokens)
seqy = to_categorical(tf.convert_to_tensor(target_tokens),
                      num_classes=vocab_size)

# Define the Tensorflow dataset for training
ds_train = tf.data.Dataset.from_tensor_slices((seqX, seqy))

# Build the testing dataset
input_tokens, target_tokens = [], []
for i in range(0, len(ds_test_text[:1000000]) - seq_len, slide_size):
  input_tokens.append([chr2idx[ch] for ch in ds_test_text[i:i + seq_len]])
  target_tokens.append(chr2idx[ds_test_text[i + seq_len]])

seqX = tf.convert_to_tensor(input_tokens)
seqy = to_categorical(tf.convert_to_tensor(target_tokens),
                      num_classes=vocab_size)

# Define the Tensorflow dataset for testing
ds_test = tf.data.Dataset.from_tensor_slices((seqX, seqy))
```

Figure 2. Preprocessing and Tokenization of Text Generation with RNN Models

The preprocessed sentences are then tokenized to convert the text data into a numerical format. A vocabulary is built by extracting all the unique characters from the dataset and mapping each character to an index. This process results in two dictionaries which are for mapping characters to indices (for encoding) and another mapping indices back to characters (for decoding model predictions). Using these mappings, the text is split into input sequences and target sequences. The input sequences represent a window of characters that the model will read, while the target sequences contain the next character that the model is expected to predict.

```
# Import functions and classes from Keras library
from keras import Input, Sequential
from keras.layers import Embedding, LSTM, Dropout, Dense

# Create the RNN that uses LSTM. Save it to rnn_lstm variable
# The Embedding layer should have 128-dimensional output vector
# Yuu may use a reasonable rate for the Dropout layer
### --YOUR CODE HERE-- ###

rnn_lstm = Sequential([
    Input(shape=(50,)),
    Embedding(input_dim=vocab_size, output_dim=128),
    LSTM(units=256, activation='tanh', return_sequences=False),
    Dropout(rate=0.5),
    Dense(units=vocab_size, activation='softmax')
])

rnn_lstm.summary()
```

```
# Configure the network for training
# Use the appropriate loss function
# You may use Adam as optimizer
### --YOUR CODE HERE-- ###
from keras.optimizers import Adam
from keras.losses import BinaryCrossentropy

rnn_lstm.compile(
    optimizer=Adam(learning_rate=0.0003),
    loss=BinaryCrossentropy(from_logits=False),
)

# Train the model. Set the number of epochs accordingly
# Set the dataset batch size of your choice
# Assign the output to rnn_lstm_hist variable
### --YOUR CODE HERE-- ###
batch_size = 100
rnn_lstm_hist = rnn_lstm.fit(ds_train.batch(batch_size), epochs=10)
```

Figure 3. Development and Training of RNN Model Using LSTM

After the data is prepared, the model is built using a RNN with LSTM layers. These LSTM layers are designed to understand patterns in the sequence of data. The model includes an embedding layer, which converts the input into a form the model can work with, and an LSTM layer that processes the sequence and keeps important information as it goes along. A dropout layer is also included to prevent overfitting by randomly turning off certain parts of the network during training. Finally, a dense layer generates the predicted next character. Once the model is set up, it is compiled using the Adam optimizer and a binary cross-entropy loss function. The model is then trained over several rounds, where the training data is fed in small groups. During training, the model's progress is monitored by measuring how well it predicts the next character using a loss function.

```
# Define the function to convert model output to character
def predictions_to_char(predictions, temperature=1.0):
  predictions = np.asarray(predictions).astype("float64")
  exp_predictions = np.exp(np.log(predictions) / temperature)
  predictions = exp_predictions / np.sum(exp_predictions)
  probabilities = np.random.multinomial(1, predictions, 1)
  return idx2chr[np.argmax(probabilities)]

# Get at least (5) seed sentences from the test dataset
# Obtain model output for each sentence and use the function defined above
# Generate a sentence that is at least 100 characters in length
# after the seed sentence.
# Save the seed sentences to a list (not TFDS) named seed_sentences
# Save the next character for each sentence to a list named next_chars
### --YOUR CODE HERE-- ###

seed_sentences = []
next_chars = []

for sentence in ds_test.take(5):
    seed = sentence.numpy().decode()
    seed_sentences.append(seed)
    next_chars.append(seed[1])


# Iterate through selected seed sentences and show the generated sentences
for idx, seed_sentence in enumerate(seed_sentences):
  print(f"Sample sentence {idx + 1}:")
  print(seed_sentence, end="")
  for _ in range(100):
    seed_tokens = tf.convert_to_tensor([[chr2idx[ch] for ch in seed_sentence]])
    predictions = rnn_lstm.predict(seed_tokens, verbose=0)
    next_char = predictions_to_char(predictions[0], temperature=0.5)
    seed_sentence = seed_sentence[1:] + next_char
    print(next_char, end="")
  print("\n")
```

Figure 4. Generation of Text Samples

After training the LSTM model, the next step is to evaluate its ability to generate text based on a seed sentence. Given an initial input, the model predicts the next 100 characters, having them one at a time. The prediction process involves sampling from the probability distribution output by the model to determine the most likely next character. The use of a temperature parameter is to control the randomness of the generated text.

```
# Compute the perplexity metric for each seed sentence as you selected above
# Use the saved next character as ground truth
# Import Perplexity metric from Keras NLP
from keras_nlp.metrics import Perplexity
### --YOUR CODE HERE-- ###

perplexity = Perplexity()

for idx, (seed_sentence, next_char) in enumerate(zip(seed_sentences, next_chars)):
    seed_tokens = tf.convert_to_tensor([[chr2idx[ch] for ch in seed_sentence]])
    predictions = rnn_lstm.predict(seed_tokens, verbose=0)
    predictions = tf.reshape(predictions, (1, -1))
    ground_truth = chr2idx[next_char]
    true_char = tf.constant([[ground_truth]], dtype=tf.int64)
    perplexity.update_state(true_char, predictions)
    print(f"Perplexity for seed sentence {idx + 1}: {perplexity.result().numpy()}")
    perplexity.reset_state()
```

Figure 5. Evaluation of Model Using Perplexity Metric

The LSTM model evaluation is measured using perplexity, which determines how well the model predicts sequences. Perplexity is used to evaluate language models, showcasing how uncertain the model is when predicting the next character. It is calculated by comparing the character the model predicts to the actual next character in the sequence. This is done for several starting sentences, and the average perplexity score gives an overall view of the model's performance.

B. Text Generation with Transformer Models

```
# Get the race dataset. Load the race/high configuration
# https://www.tensorflow.org/datasets/catalog/race
# Do not split the dataset. Combine all splits
# Save the dataset into race_ds variable
# Save the dataset info to variable ds_info
### --YOUR CODE HERE-- ###
race_ds, ds_info = tfds.load('race/high', with_info=True, split='train')
# Preprocess the dataset to include only the 'article' feature
### --YOUR CODE HERE-- ###
race_ds = race_ds.map(lambda x: x['article'])
# Display a sample text from the dataset
### --YOUR CODE HERE-- ###
for example in race_ds.take(1):
    print(example)
```

```
# Import functions and classes from Keras library
from keras.layers import TextVectorization

# Define vectorization parameters
vocab_size = 10000
seq_len = 50

# Define the text tokenizer layer and update
vectorizer = TextVectorization(
    max_tokens=vocab_size - 1,
    output_mode="int",
    output_sequence_length=seq_len + 1,
)
vectorizer.adapt(race_ds)
vocab = vectorizer.get_vocabulary()
```

```
# Define the function that will preprocess the dataset
def ds_preprocess(text):
  text = tf.expand_dims(text, -1)
  tokenized_text = vectorizer(text)
  x = tokenized_text[0, :-1]
  y = tokenized_text[0, 1:]
  return x, y

# Apply preprocessing to the dataset
race_ds = race_ds.map(ds_preprocess, num_parallel_calls=tf.data.AUTOTUNE)
race_ds = race_ds.prefetch(tf.data.AUTOTUNE)
```

Figure 6. Data Acquisition and Preprocessing of Text Generation with Transformer Models

In this step, the RACE dataset, a large-scale reading comprehension dataset, is acquired and prepared for training the transformer model. The dataset is loaded from TensorFlow Datasets and preprocessed by extracting the relevant text articles. These articles will be used as input data for the transformer model, which relies on large amounts of text data to learn how to generate sequences. Similar to the RNN model, the text is tokenized, converting words or characters into numerical sequences that can be fed into the model for training.

```python
# Import functions and classes from Keras library
from keras import ops, Sequential
from keras.layers import Layer, Embedding, Dense, Dropout
from keras.layers import MultiHeadAttention, LayerNormalization

# Define the TokenAndPositionEmbedding layer
# This layer creates two separate embeddings for tokens and token index
class TokenAndPositionEmbedding(Layer):
    def __init__(self, maxlen, vocab_size, embed_dim):
        super().__init__()
        self.token_emb = Embedding(input_dim=vocab_size, output_dim=embed_dim)
        self.pos_emb = Embedding(input_dim=maxlen, output_dim=embed_dim)

    def call(self, x):
        maxlen = ops.shape(x)[-1]
        positions = ops.arange(0, maxlen, 1)
        positions = self.pos_emb(positions)
        x = self.token_emb(x)
        return x + positions

# Define the TransformerBlock layer
# This layer is designed to capture long-range dependencies in sequential data
class TransformerBlock(Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super().__init__()
        self.att = MultiHeadAttention(num_heads, embed_dim)
        self.ffn = Sequential(
            [
                Dense(ff_dim, activation="relu"),
                Dense(embed_dim),
            ]
        )
        self.layernorm1 = LayerNormalization(epsilon=1e-6)
        self.layernorm2 = LayerNormalization(epsilon=1e-6)
        self.dropout1 = Dropout(rate)
        self.dropout2 = Dropout(rate)

    def call(self, inputs):
        input_shape = ops.shape(inputs)
        batch_size = input_shape[0]
        seq_len = input_shape[1]
        causal_mask = self.causal_attention_mask(batch_size, seq_len,
                                                 seq_len, "bool")
        attention_output = self.att(inputs, inputs,
                                    attention_mask=causal_mask)
        attention_output = self.dropout1(attention_output)
        out1 = self.layernorm1(inputs + attention_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output)
        return self.layernorm2(out1 + ffn_output)

    def causal_attention_mask(self, batch_size, n_dest, n_src, dtype):
      i = ops.arange(n_dest)[:, None]
      j = ops.arange(n_src)
      m = i >= j - n_src + n_dest
      mask = ops.cast(m, dtype)
      mask = ops.reshape(mask, [1, n_dest, n_src])
      mult = ops.concatenate(
          [ops.expand_dims(batch_size, -1), ops.convert_to_tensor([1, 1])], 0
      )
      return ops.tile(mask, mult)
```

Figure 7. Preprocessing and Tokenization of Transformer Model

The development of the transformer model marks a shift from the recurrent architecture used in RNNs to a more advanced structure based on self-attention mechanisms.The Token and Position Embedding layer creates two separate embeddings which are one for the tokens (representing the words) and another for their positions in the sequence. The Token and Position Embedding layer converts the input token indices into dense vectors that capture the meanings of words. Position embeddings are then added to these token embeddings to give the model information about the position of each word within the sequence. Following the embedding layer, the Transformer Block layer is implemented. This layer is designed to capture long-range dependencies in the input data through the multi-head attention mechanism. The Transformer Block includes a feedforward neural network that processes the outputs from the attention mechanism.

```
# Import functions and classes from Keras library
from keras import Model, Input

# Define the model parameters
embed_dim = 256
num_heads = 2
feed_forward_dim  = 256

# Create the GPT model. Save it to gpt_model variable
# The input shape is the same as the maximum sequence length
# Save the input layer to gpt_input variable
# Save the TransformerBlock layer output to gpt_layer variable
# The last Dense layer has 50 output each having the same size as the vocabulary
# Save the last layer output to gpt_output variable
### --YOUR CODE HERE-- ###
gpt_input = Input(shape=(50,), dtype="int32", name="input_layer")
token_and_position_embedding = TokenAndPositionEmbedding(seq_len, vocab_size, embed_dim)(gpt_input)
gpt_layer = TransformerBlock(embed_dim, num_heads, feed_forward_dim)(token_and_position_embedding)
gpt_output = Dense(vocab_size, activation="linear", name="dense_2")(gpt_layer)

# Create the GPT model using the layers defined above
gpt_model = Model(gpt_input, [gpt_output, gpt_layer])
gpt_model.summary()
```

```
# Import functions and classes from Keras library
from keras.losses import SparseCategoricalCrossentropy

# Configure the model for training
gpt_loss_fn = SparseCategoricalCrossentropy(from_logits=True)
gpt_model.compile(loss=[gpt_loss_fn, None], optimizer="adam")

# Train the model. Set the dataset batch size of your choice
# Assign the output to gpt_hist variable
### --YOUR CODE HERE-- ###
gpt_hist = gpt_model.fit(race_ds.batch(32), epochs=10)
```

Figure 8. Development and Training of Transformer Model

After the model architecture development, compilation and training were done using the preprocessed RACE dataset. During training, the model learns to predict the next word or character in a sequence by reducing the difference between its predictions and the actual next word. The loss function is monitored over multiple training rounds to ensure the model is learning effectively from the data.

```
# Import functions and classes from Keras library
from keras.activations import softmax

# Set five (5) seed sentences
### --CHANGE CODE BELOW-- ###
seed_sentences = [ # provide starting sentences here
    "I am currently working on a new project.",
    "She is dancing gracefully on stage.",
    "They are enjoying their vacation in Hawaii.",
    "We are learning to play the guitar.",
    "He is studying for his upcoming exams."
]

# Define the function to sample from model predictions
def sample_from(logits, k):
  logits, indices = ops.top_k(logits, k, sorted=True)
  indices = np.asarray(indices).astype("int32")
  preds = softmax(ops.expand_dims(logits, 0))[0]
  preds = np.asarray(preds).astype("float32")
  return np.random.choice(indices, p=preds)

# Define the function to generate text
def gpt_output_from_prompt(start_tokens, max_tokens=seq_len, top_k=10):
  start_tokens = [_ for _ in start_tokens]
  num_tokens_generated = 0
  tokens_generated = []
  while num_tokens_generated <= max_tokens:
    pad_len = seq_len - len(start_tokens)
    sample_index = len(start_tokens) - 1
    if pad_len < 0:
      x = start_tokens[:seq_len]
      sample_index = seq_len - 1
    elif pad_len > 0:
      x = start_tokens + [0] * pad_len
    else:
      x = start_tokens
    x = np.array([x])
    y, _ = gpt_model.predict(x, verbose=0)
    sample_token = sample_from(y[0][sample_index], top_k)
    tokens_generated.append(sample_token)
    start_tokens.append(sample_token)
    num_tokens_generated = len(tokens_generated)
  gen_text = " ".join(
      [vocab[index] for index in start_tokens + tokens_generated]
  )
  return gen_text

# Assemble word to index dictionary
word_to_index = {word: index for index, word in enumerate(vocab)}

# Process each starting sentence into tokens
seed_tokens = [[word_to_index.get(word, 1) for word in sentence.split()]
               for sentence in seed_sentences]

# Iterate through the seed sentences
# Apply the gpt_output_from_prompt function as defined above
# for each starting sentence
### --YOUR CODE HERE-- ###
for idx, seed_token in enumerate(seed_tokens):
  print(f"Sample sentence {idx + 1}:")
  print(gpt_output_from_prompt(seed_token))
  print("\n")
```

Figure 9. Generation and Evaluation

After training the transformer model, it is used to generate text based on a given seed sentence. Similar to the RNN model, the transformer predicts the next characters in the sequence by attending to the input seed sentence. The generated text is then evaluated for coherence and fluency. The performance of the transformer model is also assessed using the perplexity metric. The generated text samples and perplexity scores are compared to those of the RNN model to evaluate which model performs better at text generation.

Results and Analysis

A. Text Generation with RNN Models

A1.

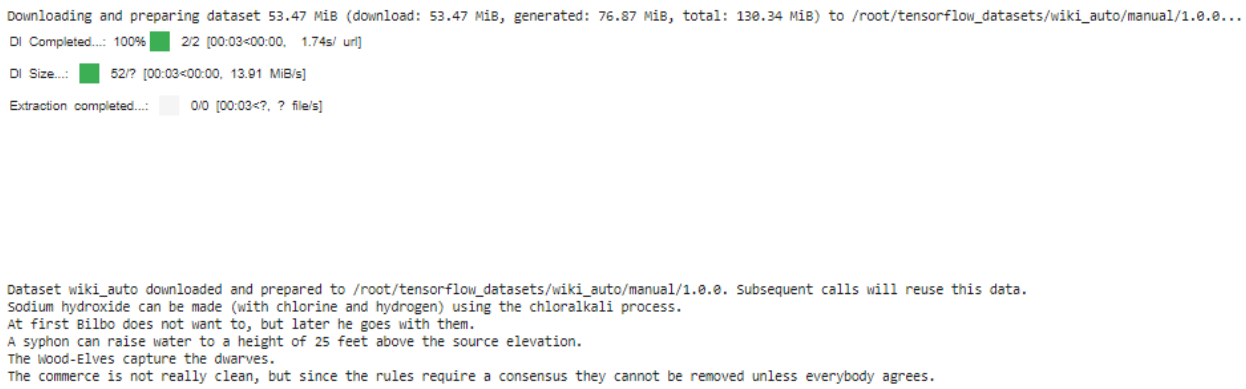

Figure 10. Download status and 5 sentences in the test dataset

The output shown in Figure 10 shows the download status as well as where the data is stored locally. The latter part of the figure shows the 5 sentences of the test dataset, and they are complete, cohesive sentences and follow proper grammar. This gives the students what sentences are to be expected and be the standard.

A2.

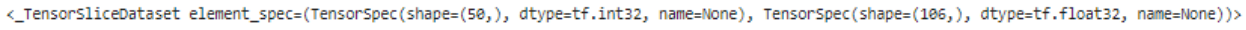

Figure 11. Output text of item A2

The output shown in Figure 11 indicates the shape of the train and test dataset after the preprocessing and tokenization have been made. It shows that it would be using TensorSlice dataset and that it would be slicing the tensors into slices for processing.

A3.

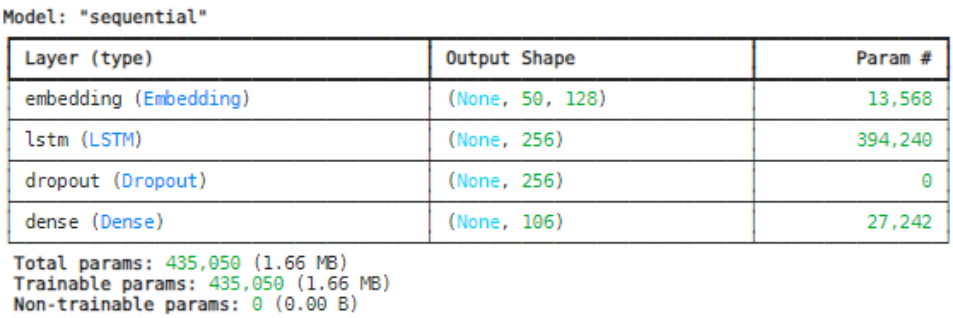

Figure 12. Model summary of RNN Model with LSTM layer

The figure shown in Figure 12 shows the summary of the model built using the Keras sequential function, which conforms to the given architecture in the laboratory manual. It indicates that all of its parameters are trainable. This model will be configured and trained in the next procedure.

A4.

```
Epoch 1/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 63s 9ms/step - loss: 0.0536
Epoch 2/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 65s 10ms/step - loss: 0.0300
Epoch 3/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 59s 9ms/step - loss: 0.0277
Epoch 4/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 62s 9ms/step - loss: 0.0260
Epoch 5/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 59s 9ms/step - loss: 0.0246
Epoch 6/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 60s 9ms/step - loss: 0.0234
Epoch 7/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 59s 9ms/step - loss: 0.0222
Epoch 8/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 58s 9ms/step - loss: 0.0212
Epoch 9/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 83s 9ms/step - loss: 0.0203
Epoch 10/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 59s 9ms/step - loss: 0.0193
```

Figure 13. Training output of the RNN with LSTM

The model is trained for 10 epochs with a batch size of 100. It was observed that the model took considerably longer than those of previous models thus, it was trained for only 10 epochs. The loss per epoch is already low, just with a few epochs trained, but it could be said that if it is increased, it should get even lower.

A5.

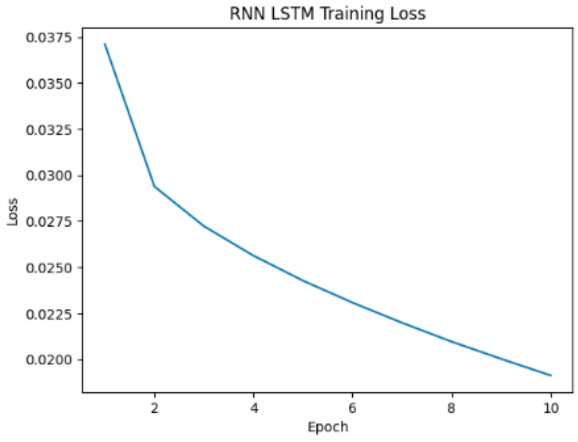

Figure 14. Training loss graph of RNN LSTM Model

The graph shown in Figure 14 is the training loss graph of the built RNN Model with the LSTM layer. It shows that the loss has a gradual decrease, almost linear, which indicates that the model has more room to improve given more epochs as it did not plateau yet, but due to limitations of Google Colab, it has been just set to 10 epochs. However, it already has low losses compared to the previous models trained, which could indicate that it would perform well even with a few epochs.

A6.

```
Sample sentence 1:
They have 21 digits.
In May 2002, Frank Iero joine is and to the dwarves of It was of the propiced into the solice to the solution of a thrist of the

Sample sentence 2:
21 digits.
In May 2002, Frank Iero joined as the reating of 1970 was sent in the police selbor dound were solved in the tomerucing the dwarves and Bil

Sample sentence 3:

In May 2002, Frank Iero joined as the rhythm guit is a fore the dwarves.
Bilbo names the dwarves and Bilbo pave of the somecited in the somecised to

Sample sentence 4:
02, Frank Iero joined as the rhythm guitarist for the solutions in the trace the somecial used to and the liquid in the lace their to the governed and

Sample sentence 5:
Iero joined as the rhythm guitarist for the band.
The conside ald of the ring and consinge to and many tiles.
The dwarves and his gouse of the Lonely

Perplexity for seed sentence 1: 245023.890625
Perplexity for seed sentence 2: 10000074.0
Perplexity for seed sentence 3: 10000055.0
Perplexity for seed sentence 4: 138.68923950195312
Perplexity for seed sentence 5: 551234.9375
```

Figure 15. Output of A6 Cell

The output shown in Figure 15 shows the generated text samples of the model. The generated sentences are observed to be incomplete and incoherent and do not follow grammar rules, which indicates that the model is not trained sufficiently or there is a hyperparameter that needs to be changed, most likely the epochs and batch size. The perplexity scores confirm that the generated sentences are indeed not good, and the model is underperforming in predicting the words as it is in the range of hundreds, and some are even thousands.

A7.

```
Sample sentence 1:
Seed: They have 21 digits.
In May 2002, Frank Iero joine

Temperature: 0.2
They have 21 digits.
In May 2002, Frank Iero joine of the Lonely Mountain, and continue to
the dwarves.
The solice them andalf them the solite the sol


Temperature: 0.5
They have 21 digits.
In May 2002, Frank Iero joine of the was an areacter for their
```

```
intonaling the into the solitical commanites.
Bilbo store the dwar


Temperature: 1.5
They have 21 digits.
In May 2002, Frank Iero joined, hes at laze fack.
Love, liquid comes siorsend Corkedmethin's lakes whean flog the spicbe
of Chori


Temperature: 2.0
They have 21 digits.
In May 2002, Frank Iero joine.
Naw TV. twen remalitict maOH6 of the topingle aid mujo's" by "Iwh Iad
sapcuxide "Prkevem Thr57 Ago

--------------------------------------------------
Sample sentence 2:
Seed: 21 digits.
In May 2002, Frank Iero joined as the r

Temperature: 0.2
21 digits.
In May 2002, Frank Iero joined as the rearing the dwarves.
The solice them andalf them to the dwarves and Bilbo to the dwarved by
the lawer


Temperature: 0.5
21 digits.
In May 2002, Frank Iero joined as the reater of the contron their to the
foring and since the liquid in the solution is a fored a rived and


Temperature: 1.5
21 digits.
In May 2002, Frank Iero joined as the renatever becain, aboun Bl2, ug
the may.
Whigh the was "65, Arghewar Sals evenerd, a wave apperine Pr


Temperature: 2.0
21 digits.
In May 2002, Frank Iero joined as the ract muid, nol on the whack old
paning ann'.
Ip of restrolilact cheety the "Tollovesecs.
Lay aptorins

--------------------------------------------------
Sample sentence 3:
Seed:
In May 2002, Frank Iero joined as the rhythm guit
```

Temperature: 0.2

In May 2002, Frank Iero joined as the rhythm guit is and and seresting are somestianing and solvesting they and the dwarves and Bilbo to the dwarves

Temperature: 0.5

In May 2002, Frank Iero joined as the rhythm guit has a somecise to the armed tray them and the concreasing the police allo maned to the homestory wa

Temperature: 1.5

In May 2002, Frank Iero joined as the rhythm guit be kitht.
The police wing the movemies a to neact Berrin) as an 1Bar elestillso finmits out Micknow

Temperature: 2.0

In May 2002, Frank Iero joined as the rhythm guitly nuaplevisleran uer phecusate, Ijing-cneatutes to other's nes (biciber latin "Heny ol Ket' Is watl

---
Sample sentence 4:
Seed: 02, Frank Iero joined as the rhythm guitarist for

Temperature: 0.2
02, Frank Iero joined as the rhythm guitarist for the solice them andalf the solite of the solution of the solution of the solitioning the street and

Temperature: 0.5
02, Frank Iero joined as the rhythm guitarist for indived the simed in the most andiamed the have the dwarves and Bilbo to the dwarves, and the wizard

Temperature: 1.5
02, Frank Iero joined as the rhythm guitarist for armamethedy of pillokef theac rearciets the whorch part Supinten of Cormint Dwars Her" trisibed is n

Temperature: 2.0
02, Frank Iero joined as the rhythm guitarist for I"Etylamaghan Propederily appyct.

```
Baterw, in elreg if udder diquyly witl profery from Oirsed, 1937,

--------------------------------------------------
Sample sentence 5:
Seed: Iero joined as the rhythm guitarist for the band.


Temperature: 0.2
Iero joined as the rhythm guitarist for the band.
The solice them and and ander the many times.
The solution of the solite in the somecises they and t


Temperature: 0.5
Iero joined as the rhythm guitarist for the band.
He as a group as a solice them and the treasures were and the dwarves
and Bilbo to the dwarves and B


Temperature: 1.5
Iero joined as the rhythm guitarist for the band.
Fit hows s'ipless simsundites, triaged hat of dwarves welitage an 196,
sollite, cinimagagand somenco


Temperature: 2.0
Iero joined as the rhythm guitarist for the band.
Tho flueg capple, and shogut of muastilnm ovect 20 it bayt Cumpur,
LPweicr.
Loling dries: AMC Citemo

--------------------------------------------------
Perplexity for sentence 1: 245023.890625
Perplexity for sentence 2: 10000074.0
Perplexity for sentence 3: 10000055.0
Perplexity for sentence 4: 138.68923950195312
Perplexity for sentence 5: 551234.9375
```

Figure 16. Output of A7 Cell

The output shown in Figure 16 shows what would happen if the temperature of the predictive function is changed from the default of 1.0 to the given temperatures. It is observed that as the temperature gets higher, the generated sentences, specifically the majority of the words, become gibberish to the point that they are not based on an actual word and are just a bunch of letters. This can be attributed to the temperature somehow dictating the variability of the predicted word, thus increasing the odds of making gibberish words as the characters of those words are very different from one another.

A8.

Model: "sequential 14"

Layer (type)	Output Shape	Param #
embedding_gru (Embedding)	(None, 50, 128)	13,568
gru (GRU)	(None, 256)	296,448
dropout_gru (Dropout)	(None, 256)	0
dense_gru (Dense)	(None, 106)	27,242

Total params: 337,258 (1.29 MB)
Trainable params: 337,258 (1.29 MB)
Non-trainable params: 0 (0.00 B)

Figure 17. Model Summary of RNN Model with GRU Layer

The layers and the hyperparameters of the model are the same as the RNN Model with the LSTM layer, with the exception of replacing that LSTM layer with a GRU layer instead, this is also to ensure that there would be a direct comparison with the LSTM results. The GRU layer is observed to have fewer parameters than the LSTM layer, which indicates it has a very different architecture and operation from LSTM layer, particularly in terms of efficiency in training and accuracy on smaller datasets.

```
Epoch 1/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 56s 8ms/step - loss: 0.0210
Epoch 2/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 54s 8ms/step - loss: 0.0188
Epoch 3/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 82s 8ms/step - loss: 0.0171
Epoch 4/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 54s 8ms/step - loss: 0.0158
Epoch 5/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 55s 8ms/step - loss: 0.0148
Epoch 6/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 54s 8ms/step - loss: 0.0140
Epoch 7/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 54s 8ms/step - loss: 0.0134
Epoch 8/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 55s 8ms/step - loss: 0.0129
Epoch 9/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 54s 8ms/step - loss: 0.0124
Epoch 10/10
6633/6633 ━━━━━━━━━━━━━━━━━━━━ 83s 8ms/step - loss: 0.0120
```

Figure 18. Training output and loss graph of RNN with GRU Layer

When compared to the RNN with the LSTM layer, the training is faster with the GRU layer and produces lower losses overall, as shown in Figure 18. This could indicate that the model with the GRU layer would be better at predicting and generating sentences than the one with the LSTM layer. The loss graph shows that it somehow follows an exponential decrease in loss in the 10 epochs which would indicate that the model could still be trained and have better performance given more epochs and changing hyperparameters if necessary. The perplexity metric could not be integrated into the model training due to differences in dimensions and the use of Binary cross-entropy as the loss function which is not compatible with the perplexity metric, which needs sparse or categorical cross-entropy as the loss function, for the sake of direct comparison between the two architectures, Binary Cross entropy loss function was used instead.

```
Sample sentence 1:
They have 21 digits.
In May 2002, Frank Iero joine way of example.
The top of the tube is where the lowest pressure is, so the liquid fill
of the drag

Sample sentence 2:
21 digits.
In May 2002, Frank Iero joined as the reared of govern.
The hobbit Bilbo Baggins lives a quiet and peeped an an and from the
most an ary on

Sample sentence 3:

In May 2002, Frank Iero joined as the rhythm guit on ofem on the
proposala worlds of the treasures work so his with a groupone way then.
And of the r

Sample sentence 4:
02, Frank Iero joined as the rhythm guitarist for ofe of an army
cannongs on the street comeling a light chouint for his relial and
sensers condirican

Sample sentence 5:
Iero joined as the rhythm guitarist for the band.
The dwarves are and unders and by the chart sold torced than the dwarves
and Bilbo into the dwarves.

Perplexity for seed sentence 1: 245023.890625
Perplexity for seed sentence 2: 10000074.0
Perplexity for seed sentence 3: 10000055.0
Perplexity for seed sentence 4: 138.68923950195312
Perplexity for seed sentence 5: 551234.9375
```

Figure 19. Output of Cell A8

The output shown in Figure 19 confirms the previous assumption that the RNN with the GRU layer would perform better than with the LSTM layer. That is evident when looking at the generated sentences, which have better or some sense of coherence, and there are relatively fewer nonexistent words, or they are close to an actual word. The better results can be attributed to the better loss values of the model with GRU and the relatively small training dataset.

B. Text Generation with Transformer Model

tf.Tensor(b"Answering the Community Needs of Our City\nThe Silver City Council recognizes that citizens have certain needs. To better meet your needs, we have made several changes to community facilities in 2004. This chart shows how we have tried to make your life better.\nTransport Three stations for the suburbs have been added to the western train service.20 new buses for the southern line were purchased in January. 50 percent of city bus-stops have been upgraded . Buses to the eastern suburbs will run every15 minutes.\nCommunication Broadband cable is now available to all parts of the city. All of the new Government buildings are ' smart'-wired for better computer service!\nMedical Facilities The new state-of-the-art Nightingale Hospital was opened in June. To overcome a shortage of trained medical staff at Dover Hospital, 10 doctors have been employed from overseas.Some facilities at Station Street Hospital have been upgraded.\nEducation Textbooks will be free to all primary students in 2004 ! Rent for private schools has been reduced. Teachers report that the 'no hat - no play' rule has been successful.\nProtection and Security Extra police now patrol ( ) the tourist areas. 50 new police officers graduated in July and have taken up duties in the city area.\nEntertainment / Recreation The John Street basketball courts have been re-surfaced ! The new Central Community Building opened in May. 5,000 new fiction books were bought for the Silver City Library.", shape=(), dtype=string)

Figure 20. Sample Sentence in Test Dataset

The output shown in Figure 20 shows a sample output of the preprocessed dataset where only the articles from each element in the dataset are taken. One article is then taken from the training dataset and it is printed as a sample output which will be used as the input training data to the model.

Model: "functional_3"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 50)	0
token_and_position_embedding_1 (TokenAndPositionEmbedding)	(None, 50, 256)	2,572,800
transformer_block_1 (TransformerBlock)	(None, 50, 256)	658,688
dense_2 (Dense)	(None, 50, 10000)	2,570,000

Total params: 5,801,488 (22.13 MB)
Trainable params: 5,801,488 (22.13 MB)
Non-trainable params: 0 (0.00 B)

Figure 21. Output Summary of GPT_Model

The output shown in the Figure 21 shows the output summary of the GPT model, where the architecture of the model conforms to the given manual. A total of four layers are shown which are the input layer, token and position embedding layer, transformer block layer, and dense layer. All training parameters are also trainable which will be seen in the next step.

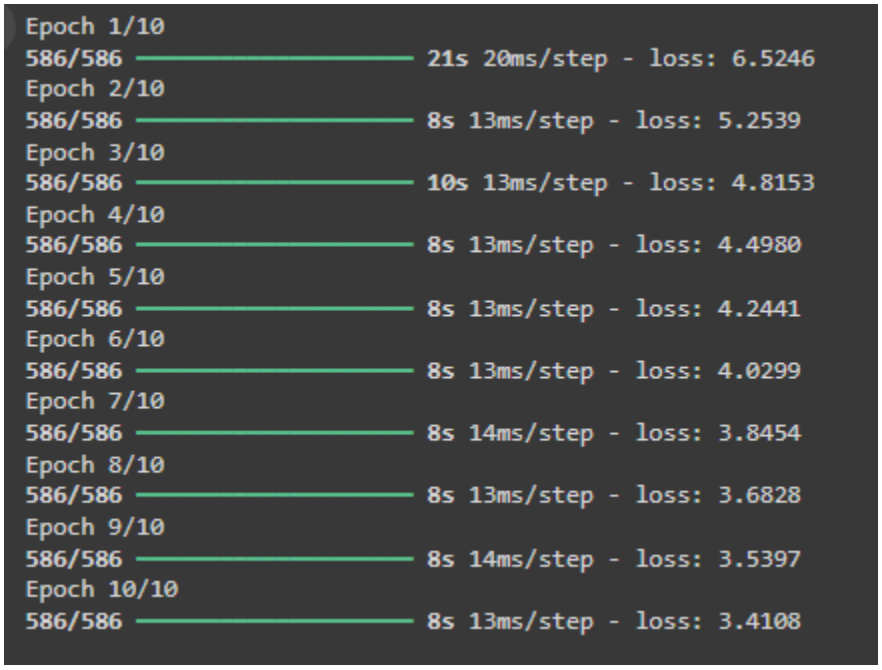

Figure 22. Training output of the GPT Model

The model is trained for 10 epochs with a batch size of 32. It was observed that the model took considerably longer than those of previous models thus it was trained for only 10 epochs. The loss per epoch is already low just with few epochs trained but it could be said that it is increased, it should get even lower.

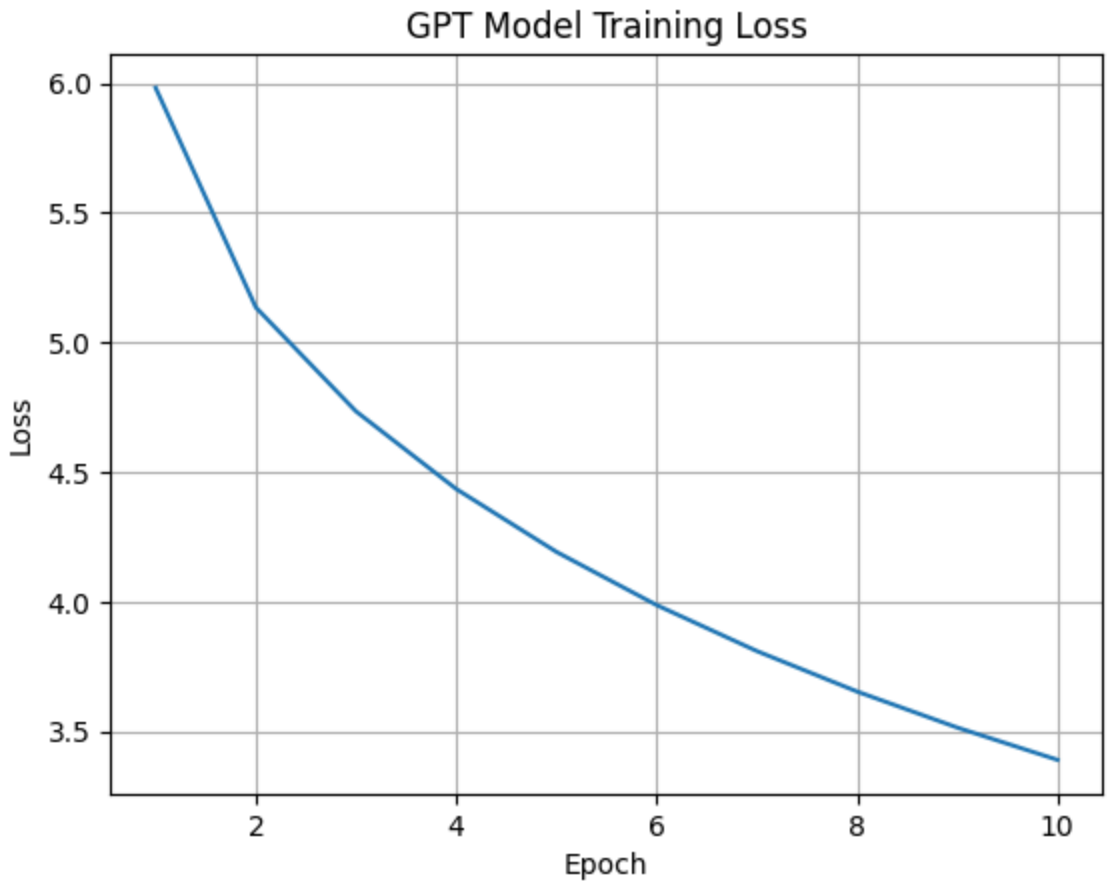

Figure 23. Training loss graph of GPT Model Training Loss

The graph shown in Figure 23 is the training loss graph of the GPT Model It shows that the loss has a gradual decrease. However, the graph has not plateaued yet which is an indication that the model can be further trained to improve the model's performance.

Sample sentence 1:
[UNK] am currently working on a new [UNK] 1400 [UNK] provinces in the [UNK] we are all areas and [UNK] conditions that make them [UNK] and [UNK] werent [UNK] to catch up in the first wooden food [UNK] in space [UNK] [UNK] [UNK] [UNK] [UNK] the [UNK] [UNK] [UNK] [UNK] of was and [UNK] the has is and in 1400 [UNK] provinces in the [UNK] we are all areas and [UNK] conditions that make them [UNK] and [UNK] werent [UNK] to catch up in the first wooden food [UNK] in space [UNK] [UNK] [UNK] [UNK] [UNK] the [UNK] [UNK] [UNK] [UNK] of was and [UNK] the has is and in

Sample sentence 2:
[UNK] is dancing [UNK] on [UNK] 23 percent in [UNK] [UNK] the [UNK] the [UNK] river [UNK] crashed with [UNK] and [UNK] of the [UNK] on the [UNK] the [UNK] of [UNK] was convinced that [UNK] [UNK]

[UNK] [UNK] [UNK] was the [UNK] were [UNK] [UNK] the [UNK] was made from by of mainly to by from 23 percent in [UNK] [UNK] the [UNK] the [UNK] river [UNK] crashed with [UNK] and [UNK] of the [UNK] on the [UNK] the [UNK] of [UNK] was convinced that [UNK] [UNK] [UNK] [UNK] [UNK] was the [UNK] were [UNK] [UNK] the [UNK] was made from by of mainly to by from

Sample sentence 3:
[UNK] are enjoying their vacation in [UNK] [UNK] [UNK] the entire [UNK] [UNK] is [UNK] to [UNK] [UNK] and their shopping [UNK] [UNK] of the [UNK] in [UNK] [UNK] [UNK] the [UNK] [UNK] of [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] is a [UNK] of [UNK] [UNK] [UNK] the [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] will [UNK] was [UNK] [UNK] [UNK] the entire [UNK] [UNK] is [UNK] to [UNK] [UNK] and their shopping [UNK] [UNK] of the [UNK] in [UNK] [UNK] [UNK] the [UNK] [UNK] of [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] is a [UNK] of [UNK] [UNK] [UNK] the [UNK] [UNK] [UNK] [UNK] [UNK] [UNK] will [UNK] was [UNK]

Sample sentence 4:
[UNK] are learning to play the [UNK] and [UNK] [UNK] is [UNK] [UNK] and [UNK] the [UNK] [UNK] are all girls [UNK] are [UNK] [UNK] [UNK] are the most popular with their [UNK] [UNK] backgrounds [UNK] [UNK] the [UNK] is a [UNK] [UNK] and that is not only 4 of [UNK] [UNK] the all them course those those and [UNK] [UNK] is [UNK] [UNK] and [UNK] the [UNK] [UNK] are all girls [UNK] are [UNK] [UNK] [UNK] are the most popular with their [UNK] [UNK] backgrounds [UNK] [UNK] the [UNK] is a [UNK] [UNK] and that is not only 4 of [UNK] [UNK] the all them course those those

Sample sentence 5:
[UNK] is studying for his upcoming [UNK] and [UNK] [UNK] [UNK] and credit cards to go before each student to [UNK] are [UNK] all over the world each other [UNK] [UNK] is a [UNK] [UNK] the [UNK] [UNK] [UNK] each [UNK] passage is sold for children [UNK] is a [UNK] [UNK] sport [UNK] [UNK] sport in on [UNK] and [UNK] [UNK] [UNK] and credit cards to go before each student to [UNK] are [UNK] all over the world each other [UNK] [UNK] is a [UNK] [UNK] the [UNK] [UNK] [UNK] each [UNK] passage is sold for children [UNK] is a [UNK] [UNK] sport [UNK] [UNK] sport in on [UNK]

Figure 24. Sample Generated Output of the GPT Model

The table shown in Figure 24 is the sample-generated sentences of the GPT model. The sample sentences show that the GPT model is able to construct sensible sentences despite the low training time. However, it can be observed that only a few parts of the sentence are sensible, while the majority of the sentence consists of the unknown token. This is an indication that the model can be improved further with longer training time.

```
Epoch 44/50
586/586 ━━━━━━━━━━━━━━━━━━━━ 8s 14ms/step - loss: 1.6565
Epoch 45/50
586/586 ━━━━━━━━━━━━━━━━━━━━ 8s 13ms/step - loss: 1.6510
Epoch 46/50
586/586 ━━━━━━━━━━━━━━━━━━━━ 8s 14ms/step - loss: 1.6465
Epoch 47/50
586/586 ━━━━━━━━━━━━━━━━━━━━ 8s 14ms/step - loss: 1.6405
Epoch 48/50
586/586 ━━━━━━━━━━━━━━━━━━━━ 8s 13ms/step - loss: 1.6371
Epoch 49/50
586/586 ━━━━━━━━━━━━━━━━━━━━ 8s 14ms/step - loss: 1.6352
Epoch 50/50
586/586 ━━━━━━━━━━━━━━━━━━━━ 8s 14ms/step - loss: 1.6313
```

Figure 25. Training output of the GPT Model

The model is retrained for 50 epochs with the same batch size of 32. It was observed that the model loss gradually decreased to a 1.63% loss per epoch, which indicates that the model can still be trained.

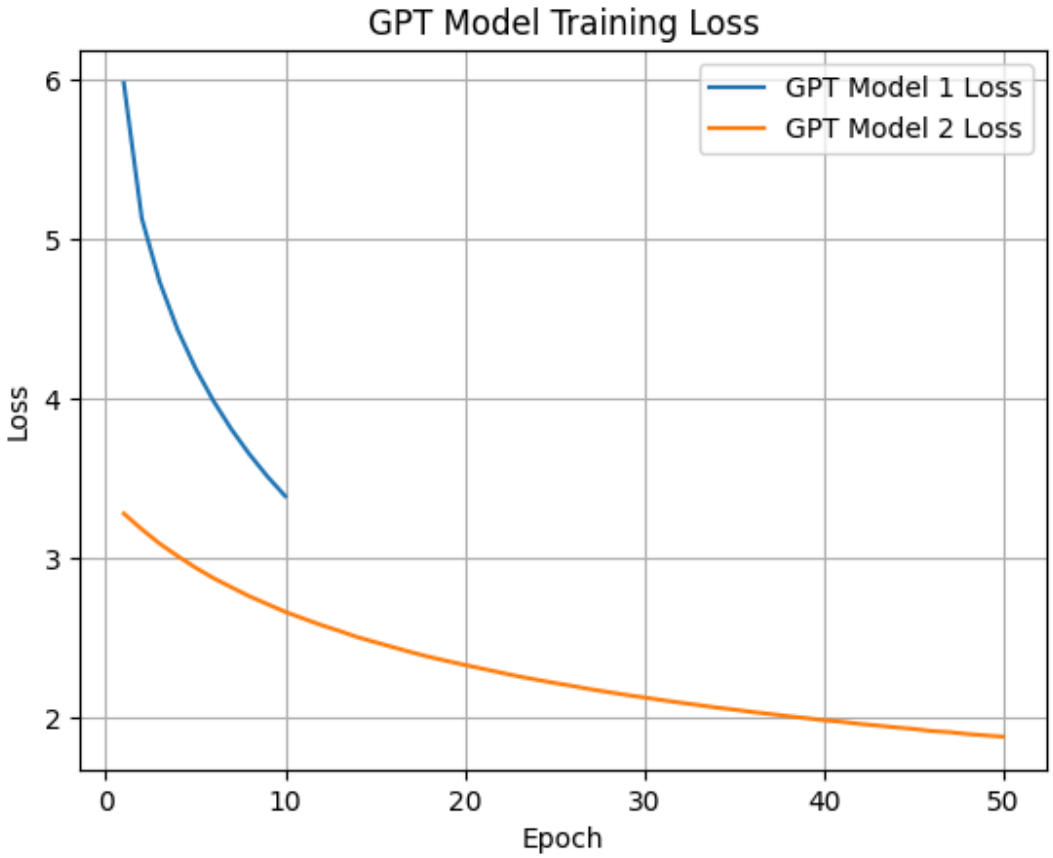

Figure 26. Training Loss of GPT Model 1 and GPT Model 2

Figure 26 shows a graph of the training loss of both GPT model 1 and GPT model 2. The GPT model shows a steep decrease in training loss which started from 6 and decreased to 3.4. The GPT model 2, which was trained with 50 epochs, started with a training loss of 3.2, which is lower than the initial training loss of the first model. The training loss of the 2nd GPT model gradually decreased from 3.2 to 1.6. The training loss after 40 epochs still shows a gradual decrease, which indicates that the model can be trained further.

Sample sentence 1:
[UNK] am currently working on a new [UNK] will be learning to read using new cars washing cars the cars aiming to begin a legal career into a [UNK] [UNK] latest changes that increase greatly over the past 50 years has a fulltime [UNK] times when the latest seed companies around tell tell around tell tell zones tell of will be learning to read using new cars washing cars the cars aiming to begin a legal career into a [UNK] [UNK] latest changes that increase greatly over the past 50 years has a fulltime [UNK] times when the latest seed companies around tell tell around tell tell zones tell of

Sample sentence 2:
[UNK] is dancing [UNK] on [UNK] the early settlers in the rose by the [UNK] took the english language so many forms of english communication skills and express learning was born in [UNK] a formal english country to measure the night of [UNK] [UNK] 16 the french row across a french french [UNK] french french french french the early settlers in the rose by the [UNK] took the english language so many forms of english communication skills and express learning was born in [UNK] a formal english country to measure the night of [UNK] [UNK] 16 the french row across a french french [UNK] french french french french

Sample sentence 3:
[UNK] are enjoying their vacation in [UNK] the uk [UNK] the workers in [UNK] they are using steel as the uk help to help stop the [UNK] the cell phones run on the [UNK] are done by [UNK] the law read before term [UNK] the cell phones on the flight left of while if the to scene the the uk [UNK] the workers in [UNK] they are using steel as the uk help to help stop the [UNK] the cell phones run on the [UNK] are done by [UNK] the law read before term [UNK] the cell phones on the flight left of while if the to scene the

Sample sentence 4:
[UNK] are learning to play the [UNK] of the unwanted [UNK] it and [UNK] is a simple thing for many people in the office workers face challenges facing one [UNK] between experiencing potential effect on their personalities and nutrition even in [UNK] have to study researchers identified the patterns that ran not ran it are it [UNK] the of the unwanted [UNK] it and [UNK] is a simple thing for many people in the office workers face challenges facing one [UNK] between experiencing potential effect on their personalities and nutrition even in [UNK] have to study researchers identified the patterns that ran not ran it are it [UNK] the

Sample sentence 5:
[UNK] is studying for his upcoming [UNK] [UNK] [UNK] [UNK] the important river the [UNK] [UNK] the grand 20 [UNK] the [UNK] has been the [UNK] of the [UNK] in the world around the world the [UNK] [UNK] are useful paintings to help the action [UNK] in the [UNK] [UNK] [UNK] [UNK] grand the cut [UNK] the grand [UNK] [UNK] [UNK] the important river the [UNK] [UNK] the grand 20 [UNK] the [UNK] has been the [UNK] of the [UNK] in the world around the world the [UNK] [UNK] are useful paintings to help the action [UNK] in the [UNK] [UNK] [UNK] [UNK] grand the cut [UNK] the grand

Figure 27. Sample Generated Sentence of GPT Model 2

The table shown in Figure 27 is the sample-generated sentences of the 2nd GPT model. The sample sentences show that further training improved the capabilities of the model to generate sentences. As seen in the table, the sentences contain more structured sentences compared to the 1st GPT model. The amount of unknown tokens also decreased compared to the 1st GPT model. The model still struggles to generate grammatically correct sentences, as it can be observed in the first sample sentence that multple words are repeated consecutively.

Conclusions

The fourth laboratory activity introduces Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) with the use of generative models. These models are used to generate sentences from a certain dataset. The first part of the activity uses the WifiAuto Dataset to generate text with RNN models. The generated results for the first text sample output are incomplete, incoherent, and grammatically incorrect. This suggests insufficient training or the need to adjust hyperparameters like epochs and batch sizes. High perplexity scores also confirm that the model has poor performance. The temperature is modified. As the temperature increases, the generated sentences become increasingly nonsensical, with most words turning into random letters. This happens because higher temperatures increase the variability of predicted words, raising the likelihood of producing gibberish. A new model was then created with a GRU layer. The output of the said model generates sentences that are more coherent, with fewer words that are close to the actual words. This improvement is likely due to better loss values in the GRU model and the relatively small training dataset.

The second part of the activity uses Race Dataset to generate text with the transformer model. After training the first model with only 10 epochs, the output also shows incomplete, incoherent, and grammatically incorrect sentences with unknown tokens that represent [UNK]. This token represents any word that is not included in the vocabulary of the model. Further training is done to improve the model by increasing the epochs to 50. This results in lower losses which improve the model's performance. The output shows less [UNK] but still has grammatically incorrect sentences.

Overall, the laboratory activity has helped the students understand the concepts and the application of LSTM and GRU in generating text using a generative model. Training took less time compared to the previous laboratory experiment since it only dealt with sentences and words. With that, the students were able to complete the activity. This activity will also help the students in future AI application activities.

References

Snyder, W. (n.d.). *Perplexity*. Boston University. Retrieved October 16, 2024, from https://www.cs.bu.edu/fac/snyder/cs505/PerplexityPosts.pdf

Vega, M. de la. (2023, November 4). Understanding openai's "temperature" and "top_p" parameters in language models. *Medium*. https://medium.com/@1511425435311/understanding-openais-temperature-and-top-p-parameters-in-language-models-d2066504684f