Try_5_May_4_2023

May 4, 2023

1 Project 4

Authors: Harshvardhan and Yu Jiang

[4]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import re
import random
```

```
2023-05-03 15:32:55.397399: I tensorflow/core/util/port.cc:110] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2023-05-03 15:32:55.511878: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
```

[5]:
```
print(tf.__version__)
```

```
2.12.0
```

1.1 Task 1

[6]:
```
class TransformerModel():
    def __init__(self, vocab_size, embed_dim=256, num_heads=2, num_blocks=1,␣
 ↪ff_dim=256, maxlen=64, rate=0.1):
        #initailize variables
        #vocab_size: the size of the vocabulary
        #embed_dim: the dimension of the embedding layer
        #num_heads: the number of heads in the multi-head attention layer
        #num_blocks: the number of transformer blocks
        #ff_dim: the dimension of the feed forward layer
        #maxlen: the maximum length of the input sequence
        #rate: the dropout rate
```

```
        self.vocab_size = vocab_size
        self.embed_dim = embed_dim
        self.num_heads = num_heads
        self.num_blocks = num_blocks
        self.ff_dim = ff_dim
        self.maxlen = maxlen
        self.rate = rate

    def TransformerBlock(self, inputs):
    # Create a causal mask for the MultiHeadAttention layer
        seq_len = tf.shape(inputs)[1]
        causal_mask = 1 - tf.linalg.band_part(tf.ones((seq_len, seq_len)), -1,␣
↪0)

        # MultiHeadAttention layer
        attn_layer = tf.keras.layers.MultiHeadAttention(
            num_heads=self.num_heads,
            key_dim=self.embed_dim // self.num_heads,
            use_bias=True
        )
        attn_output = attn_layer(inputs, inputs, attention_mask=causal_mask)
        attn_output = tf.keras.layers.Dropout(self.rate)(attn_output)
        out1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)(inputs +␣
↪attn_output)

        # Feed Forward Network
        ffn_output = tf.keras.layers.Dense(self.ff_dim, activation='relu')(out1)
        ffn_output = tf.keras.layers.Dense(self.embed_dim)(ffn_output)
        ffn_output = tf.keras.layers.Dropout(self.rate)(ffn_output)

        # LayerNormalization
        out2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)(out1 +␣
↪ffn_output)

        return out2

    def EmbeddingLayer(self, inputs):
        #create the embedding layer
        #create (1) an embedding for the tokens and (2) an embedding for the␣
↪positions
        #you can use https://keras.io/api/layers/core_layers/embedding/␣
↪Embedding class
        #you can use tf.range to enocde positions
        #add (1) and (2) and return the layer
        token_embeddings = tf.keras.layers.Embedding(self.vocab_size, self.
↪embed_dim)(inputs)
```

```
        positions = tf.range(start=0, limit=self.maxlen, delta=1)
        position_embeddings = tf.keras.layers.Embedding(self.maxlen, self.
↪embed_dim)(positions)

        mask = tf.cast(tf.math.equal(inputs, 0), tf.float32)
        mask = mask[:, tf.newaxis, tf.newaxis, :]

        embeddings = token_embeddings + position_embeddings
        embeddings = tf.keras.layers.Dropout(self.rate)(embeddings)

        return embeddings, mask

    def create_model(self):
        #combine the EmbeddingLayer and num_blocks TransformerBlocks to create␣
↪the model, use the Keras functional API (https://keras.io/guides/
↪functional_api/)
        #use the SparseCategoricalCrossentropy loss function (https://keras.io/
↪api/losses/probabilistic_losses/#sparsecategoricalcrossentropy-class)
        inputs = tf.keras.layers.Input(shape=(self.maxlen,))
        embeddings, mask = self.EmbeddingLayer(inputs)
        x = embeddings

        for _ in range(self.num_blocks):
            x = self.TransformerBlock(inputs=x)

        # x = tf.keras.layers.GlobalAveragePooling1D()(x)
        x = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(self.
↪vocab_size, activation='softmax'))(x)
        x = tf.keras.layers.Dense(self.vocab_size, activation='softmax')(x)

        model = tf.keras.Model(inputs=inputs, outputs=x)
        model.compile(optimizer='adam',
                      loss=tf.keras.losses.
↪SparseCategoricalCrossentropy(from_logits=False),
                      metrics=['accuracy'])

        return model
```

1.2 Task 2

[7]:
```
class DataSet():
    def __init__(self, filename, len):
        # Load the text from the file
        with open(filename, 'r', encoding='utf-8') as f:
            self.text = f.read()
        self.len = len
```

```
    def prep_text(self):
        # Remove all punctuation, set to lowercase, remove duplicate spaces and␣
↪other whitespace (keep newlines)
        self.text = self.text.lower()
        self.text = re.sub(r"[^a-z0-9,.!?\n]+", " ", self.text)
        self.text = re.sub(r"[\s]+", " ", self.text)
        self.text = self.text.strip()


    def tokenize_text(self):
        # Separate into words, create a vocab and convert the text to a list of␣
↪numbers using the vocab such that each unique word is represented by its own␣
↪number
        words = self.text.split()
        unique_words = np.unique(words)
        self.vocab = {word: idx for idx, word in enumerate(unique_words)}
        self.text = [self.vocab[word] for word in words]


    def create_dataset(self):
        # Split the tokenized data into sequences of length len, return the␣
↪sequences and vocab
        self.prep_text()
        self.tokenize_text()

        x = []
        y = []
        for i in range(len(self.text) - self.len):
            x.append(self.text[i:i + self.len])
            y.append(self.text[i + 1:i + self.len + 1])

        x = np.array(x)
        y = np.array(y)

        return x, y, self.vocab
```

1.3 Task 3

[8]:
```
class GenerateText:
    def __init__(self, model, vocab, sequence_length):
        self.model = model
        self.vocab = vocab
        self.sequence_length = sequence_length
        self.index_to_word = {i: word for i, word in enumerate(vocab)}
        self.word_to_index = {word: i for i, word in enumerate(vocab)}
```

```
    def _sample_from_logits(self, logits):
        logits = tf.squeeze(logits)[-1]  # Use the last element of the logits
        logits = logits.numpy() / 1.0
        return np.random.choice(len(self.vocab), p=np.exp(logits) / np.sum(np.
↪exp(logits)))


    def generate_text(self, start_string, num_generate=100):
        input_eval = [self.word_to_index[s] for s in start_string.split()]
        input_eval = tf.keras.preprocessing.sequence.
↪pad_sequences([input_eval], maxlen=self.sequence_length, padding='post')
        input_eval = tf.reshape(input_eval, (1, self.sequence_length))

        generated_text = []

        self.model.reset_states()
        for _ in range(num_generate):
            predictions = self.model.call(input_eval)
            predicted_id = self._sample_from_logits(predictions)

            input_eval = tf.concat([input_eval[:, 1:], tf.
↪expand_dims([predicted_id], 1)], axis=1)

            generated_text.append(self.index_to_word[predicted_id])

        return ' '.join(generated_text)

    def generate_random_text(self, num_generate=100):
        generated_text = []
        for _ in range(num_generate):
            random_word = random.choice(self.vocab)
            generated_text.append(random_word)

        return ' '.join(generated_text)
```

1.4 Task 4

[9]:
```
def train_model(vocab, train_data, num_epochs, num_heads):
    vocab_size = len(vocab)
    model = TransformerModel(vocab_size, num_heads = num_heads)
    transformer = model.create_model()

    x_train, y_train = train_data

    history = transformer.fit(x_train, y_train, batch_size=32,␣
↪epochs=num_epochs, validation_split=0.1)
```

```
    # Print the training loss for each epoch
    for epoch, loss in enumerate(history.history['loss'], start=1):
        print(f'Epoch {epoch}, Loss: {loss}')

    return transformer, history
```

[6]:
```
# def train_model(vocab, train_data, num_epochs, num_heads):
#     vocab_size = len(vocab)
#     model = TransformerModel(vocab_size = vocab_size, num_heads = num_heads)
#     transformer = model.create_model()

#     x_train, y_train = train_data

#     history = transformer.fit(x_train, y_train, batch_size=32,␣
 ↪epochs=num_epochs, validation_split=0.1)

#     return transformer, history
```

1.5 Loading Data

[10]:
```
# Load and preprocess the dataset
sequence_length = 64
data = DataSet("beatles.txt", sequence_length)
x, y, vocab = data.create_dataset()
```

[11]:
```
def write_output_to_file(output, file_name):
    with open(file_name, 'w') as f:
        f.write(output)
```

1.5.1 Epochs = 1

Number of Attention Heads = 2

[12]:
```
%%capture captured_output
# Train the model
num_epochs = 1
num_heads = 2
trained_model, training_history = train_model(vocab, (x, y), num_epochs,␣
 ↪num_heads = num_heads)
```

```
2023-05-03 15:34:44.742812: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1635] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 43482 MB memory:  -> device:
0, name: NVIDIA A40, pci bus id: 0000:51:00.0, compute capability: 8.6
2023-05-03 15:34:49.665236: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32
will be used for the matrix multiplication. This will only be logged once.
```

```
2023-05-03 15:34:49.701672: I tensorflow/compiler/xla/service/service.cc:169]
XLA service 0x7fa8cc00e070 initialized for platform CUDA (this does not
guarantee that XLA will be used). Devices:
2023-05-03 15:34:49.701721: I tensorflow/compiler/xla/service/service.cc:177]
StreamExecutor device (0): NVIDIA A40, Compute Capability 8.6
2023-05-03 15:34:49.717060: I
tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR
crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
2023-05-03 15:34:50.052779: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN
version 8800
2023-05-03 15:34:50.124687: I tensorflow/tsl/platform/default/subprocess.cc:304]
Start cannot spawn child process: No such file or directory
2023-05-03 15:34:50.189238: I ./tensorflow/compiler/jit/device_compiler.h:180]
Compiled cluster using XLA!  This line is logged at most once for the lifetime
of the process.
```

[13]:
```
output_text = captured_output.stdout
file_name = 'epochs_1.txt'
write_output_to_file(output_text, file_name)
```

[15]:
```
# Generate text using the trained model
generate = GenerateText(trained_model, vocab, sequence_length)
start_string = "daisy"
generated_text = generate.generate_text(start_string, num_generate=10)
print(generated_text)
```

```
stick saw sunday give crime, surely stays matchbox da, stays
```

[16]:
```
start_string = "happy"
generated_text = generate.generate_text(start_string, num_generate=10)
print(generated_text)
```

```
bits pray coming love about. disconnect armen break red spinning
```

[17]:
```
start_string = "country"
generated_text = generate.generate_text(start_string, num_generate=10)
print(generated_text)
```

```
diverted would. boy. comfort grandchildren belonged. pie, ago kitchen overnight.
```

Number of Attention Heads = 3

[18]:
```
%%capture captured_output
# Train the model
num_epochs = 1
num_heads = 3
trained_model, training_history = train_model(vocab, (x, y), num_epochs,␣
 ↪num_heads = num_heads)
```

[19]:
```
output_text = captured_output.stdout
file_name = 'epochs_1_heads_3.txt'
write_output_to_file(output_text, file_name)
```

[20]:
```
# Generate text using the trained model
generate = GenerateText(trained_model, vocab, sequence_length)
start_string = "daisy"
generated_text = generate.generate_text(start_string, num_generate=10)
print(generated_text)
```

```
whoah butted peep meant white, yesterday. sweet, photographs knee. blue,
```

[21]:
```
start_string = "happy"
generated_text = generate.generate_text(start_string, num_generate=10)
print(generated_text)
```

```
review sdaeh specially poop heading return horse beam existence isle
```

[22]:
```
start_string = "country"
generated_text = generate.generate_text(start_string, num_generate=10)
print(generated_text)
```

```
turn shining happened suddenly chains wave three, bus 4, waits
```

1.5.2 Epochs = 50

Number of Heads = 2

[23]:
```
%%capture captured_output
# Train the model
num_epochs = 50
num_heads = 2
trained_model50, training_history50 = train_model(vocab, (x, y), num_epochs,␣
 ↪num_heads = num_heads)
```

[24]:
```
output_text = captured_output.stdout
file_name = 'epochs_50_heads_2.txt'
write_output_to_file(output_text, file_name)
```

[25]:
```
# Generate text using the trained model
generate50 = GenerateText(trained_model50, vocab, sequence_length)
start_string = "daisy"
generated_text50 = generate50.generate_text(start_string, num_generate=10)
print(generated_text50)
```

```
be? fly. to, bell haze. child, think! lizzie you moonlight
```

[26]:
```
start_string = "happy"
generated_text = generate.generate_text(start_string, num_generate=10)
```

```
print(generated_text)
```

```
minds them, people n letter, party. turing customer, buys party.
```

[27]:
```
start_string = "country"
generated_text = generate.generate_text(start_string, num_generate=10)
print(generated_text)
```

```
shirt, magical knickers postcards tuned ties, pam. pass college, few
```

Num of Heads = 3

[28]:
```
%%capture captured_output
# Train the model
num_epochs = 50
num_heads = 3
trained_model50, training_history50 = train_model(vocab, (x, y), num_epochs,␣
 ↪num_heads = num_heads)
```

[29]:
```
output_text = captured_output.stdout
file_name = 'epochs_50_heads_3.txt'
write_output_to_file(output_text, file_name)
```

[30]:
```
# Generate text using the trained model
generate50 = GenerateText(trained_model50, vocab, sequence_length)
start_string = "daisy"
generated_text50 = generate50.generate_text(start_string, num_generate=10)
print(generated_text50)
```

```
million, tuned porters disagree joker natural move hello kind ruins
```

[31]:
```
start_string = "happy"
generated_text = generate.generate_text(start_string, num_generate=10)
print(generated_text)
```

```
of, store mir seen, mack guaranteed lover aaaaahhhhhhhhh… tee band
```

[32]:
```
start_string = "country"
generated_text = generate.generate_text(start_string, num_generate=10)
print(generated_text)
```

```
ob da awoke, meaningless deeper being, thinking poe. dead, jack
```

1.5.3 Epochs = 100

Number of Heads = 2

[ ]:
```
%%capture captured_output
# Train the model
num_epochs = 100
```

```
num_heads = 2
trained_model100, training_history100 = train_model(vocab, (x, y), num_epochs,␣
 ↪num_heads = num_heads)
```

[52]:
```
output_text = captured_output.stdout
file_name = 'epochs_100_heads_2.txt'
write_output_to_file(output_text, file_name)
```

[53]:
```
# Generate text using the trained model
generate100 = GenerateText(trained_model100, vocab, sequence_length)
start_string = "daisy"
generated_text100 = generate100.generate_text(start_string, num_generate=10)
print(generated_text100)
```

```
linger round banks thoughtlessly inside ice kind. friend, michelle. pain,
```

[54]:
```
# Generate text using the trained model
generate100 = GenerateText(trained_model100, vocab, sequence_length)
start_string = "happy"
generated_text100 = generate100.generate_text(start_string, num_generate=10)
print(generated_text100)
```

```
mit would stand pigs near. jude, clothes, easy, whim, wail
```

[55]:
```
# Generate text using the trained model
generate100 = GenerateText(trained_model100, vocab, sequence_length)
start_string = "country"
generated_text100 = generate100.generate_text(start_string, num_generate=10)
print(generated_text100)
```

```
walked bed ah peanuts aware. habit age based rita club
```

Number of Heads = 3

[ ]:
```
%%capture captured_output
# Train the model
num_epochs = 100
num_heads = 3
trained_model100, training_history100 = train_model(vocab, (x, y), num_epochs,␣
 ↪num_heads = num_heads)
```

[56]:
```
output_text = captured_output.stdout
file_name = 'epochs_100_heads_3.txt'
write_output_to_file(output_text, file_name)
```

[57]:
```
# Generate text using the trained model
generate100 = GenerateText(trained_model100, vocab, sequence_length)
start_string = "daisy"
```

```
generated_text100 = generate100.generate_text(start_string, num_generate=10)
print(generated_text100)
```

```
sermon risk di your, ev brother imitate armen skies, knee.
```

[58]:
```
start_string = "happy"
generated_text100 = generate100.generate_text(start_string, num_generate=10)
print(generated_text100)
```

```
place, mmm. row boac cries yer hate. day, upset already
```

[59]:
```
start_string = "country"
generated_text100 = generate100.generate_text(start_string, num_generate=10)
print(generated_text100)
```

```
falling, picking sha 15 returned younger, saw screen buys owww!
```

1.5.4 Epochs = 120

Number of Heads = 2

[ ]:
```
%%capture captured_output
# Train the model
num_epochs = 120
num_heads = 2
trained_model120, training_history120 = train_model(vocab, (x, y), num_epochs,␣
 ↪num_heads = num_heads)
```

[60]:
```
output_text = captured_output.stdout
file_name = 'epochs_120_heads_2.txt'
write_output_to_file(output_text, file_name)
```

[61]:
```
# Generate text using the trained model
generate120 = GenerateText(trained_model120, vocab, sequence_length)
start_string = "daisy"
generated_text120 = generate120.generate_text(start_string, num_generate=10)
print(generated_text120)
```

```
pum bother hi love fields. anyway bag. longer two. bet
```

[62]:
```
# Generate text using the trained model
start_string = "happy"
generated_text120 = generate120.generate_text(start_string, num_generate=10)
print(generated_text120)
```

```
changing hurting mattered already entschuldigst talked liverpool rent? madly
backdoor
```

[63]:
```
# Generate text using the trained model
start_string = "country"
generated_text120 = generate120.generate_text(start_string, num_generate=10)
print(generated_text120)
```

```
harmony beginning on? carry door girlfriend true, away wall home.
```

Number of Heads = 3

[ ]:
```
%%capture captured_output
# Train the model
num_epochs = 120
num_heads = 2
trained_model120, training_history120 = train_model(vocab, (x, y), num_epochs,␣
 ↪num_heads = num_heads)
```

[ ]:
```
output_text = captured_output.stdout
file_name = 'epochs_120_heads_3.txt'
write_output_to_file(output_text, file_name)
```

[64]:
```
# Generate text using the trained model
generate120 = GenerateText(trained_model120, vocab, sequence_length)
start_string = "daisy"
generated_text120 = generate120.generate_text(start_string, num_generate=10)
print(generated_text120)
```

```
didn mine. enough monkey cloud, hey, works rhythm girls, sweat
```

[65]:
```
# Generate text using the trained model
start_string = "happy"
generated_text120 = generate120.generate_text(start_string, num_generate=10)
print(generated_text120)
```

```
postcards presents. roll. stream, middle singer there, anymore. seashell meadows
```

[66]:
```
# Generate text using the trained model
start_string = "country"
generated_text120 = generate120.generate_text(start_string, num_generate=10)
print(generated_text120)
```

```
gave rest. left chance. asleep wishing ease jetzt kids weeks,
```

2 Report

2.1 Introduction

In this project, we aim to develop a token-based Transformer neural network that generates lyrics in the style of the Beatles. The problem is framed as a many-to-many task, where the goal is to predict

a series of words. The dataset consists of lyrics from 246 Beatles songs, which are concatenated and treated as a single long sequence.

The project involves implementing the TransformerModel Class, the DataSet Class, and the GenerateText Class to create, train, and evaluate the model. The model will be trained and qualitatively evaluated using different numbers of epochs, and its performance will be assessed based on the generated text's similarity to the original Beatles lyrics.

The ultimate goal is to create a model capable of generating creative and coherent lyrics that resemble the Beatles' unique style.

2.2 Our Network

We define a TransformerModel class, which creates a Transformer-based neural network for generating Beatles-style lyrics. The model architecture consists of an embedding layer, Transformer blocks, and a dense output layer. The DataSet class loads and preprocesses the lyrics data, converting it into sequences for training. The GenerateText class is responsible for generating new text using the trained model.

The model is created with TensorFlow, and its architecture includes multi-head attention, layer normalization, and dropout layers. The EmbeddingLayer method creates token and positional embeddings and combines them before feeding them into the Transformer blocks. The TransformerBlock method applies multi-head attention and feed-forward layers with skip connections and layer normalization.

The DataSet class reads a text file containing Beatles lyrics, preprocesses it by removing special characters, converting to lowercase, and tokenizing it. It then creates sequences of specified length for training. The GenerateText class uses the trained model to generate new text based on a given start string or randomly selected words from the vocabulary.

Finally, the train_model function trains the model with a specified number of epochs and returns the trained model and its training history. The write_output_to_file function writes the generated output to a file.

2.3 Results

Epochs	Number of Heads	Cat-Cross-Entropy Loss (Final)
1	2	7.1664
1	3	7.1611
50	2	6.0984621
50	3	6.0984640
100	2	6.09845
100	3	6.09840
120	2	6.09851
120	3	6.09847

We do not see huge improvements in the model after running it for many epochs. The lyrics are probably as meaningless/meaingful as running for one epoch. However, we are not huge fans of

Beatles so we may be missing the context of the lyrics. It is possible that the model run through more epochs is actually generating better lyrics.

The loss has nearly settled at 6.098 and further improvements are only in the fourth and fifth decimal places.

For our testing purpose, we tried three starting words: daisy, happy and country. Since the transformer model can only use words that are already in the lyrics, it limits the output. We weren't able to generate full text of songs.

2.4 Conclusion

First, the text from the 'beatles.txt' file is loaded and preprocessed to create a dataset. The preprocessing involves cleaning the text by removing punctuation, setting it to lowercase, and removing duplicate spaces and other whitespace.

The preprocessed text is tokenized, which involves splitting the text into words, creating a vocabulary, and converting the text to a list of numbers using the vocabulary such that each unique word is represented by its own number.

The tokenized text is then split into sequences of length 64.

A Transformer model is created and trained using the sequences from the dataset for different numbers of epochs (1, 50, 100, and 120 and different number of attention heads (2 and 3). For each experiment, the training progress, including loss and accuracy, is captured and written to a file with names **epochs_1_heads_2.txt**, **epochs_1_heads_3.txt**, **epochs_50_heads_2.txt**, **epochs_50_heads_3.txt**, **epochs_100_heads_2.txt**, **epochs_100_heads_3.txt**, **epochs_120_heads_2.txt**, and **epochs_120_heads_3.txt**, respectively.

After training the model, you will generate text using the trained model with varying numbers of epochs. You will start the generated text with the word "daisy" and generate 10 words. The quality of the generated text will likely improve as the number of training epochs increases.First, the text from the 'beatles.txt' file is loaded and preprocessed to create a dataset. The preprocessing involves cleaning the text by removing punctuation, setting it to lowercase, and removing duplicate spaces and other whitespace.

The preprocessed text is tokenized, which involves splitting the text into words, creating a vocabulary, and converting the text to a list of numbers using the vocabulary such that each unique word is represented by its own number.

The tokenized text is then split into sequences of length 64.

A Transformer model is created and trained using the sequences from the dataset for different numbers of epochs (1, 50, 100, and 120 and different number of attention heads (2 and 3). For each experiment, the training progress, including loss and accuracy, is captured and written to a file with names **epochs_1_heads_2.txt**, **epochs_1_heads_3.txt**, **epochs_50_heads_2.txt**, **epochs_50_heads_3.txt**, **epochs_100_heads_2.txt**, **epochs_100_heads_3.txt**, **epochs_120_heads_2.txt**, and **epochs_120_heads_3.txt**, respectively.

After training the model, you will generate text using the trained model with varying numbers of epochs. You will start the generated text with the word "daisy" and generate 10 words. The quality of the generated text will likely improve as the number of training epochs increases.

2.5 How to Run Code

1. Import required libraries in the first few cells and run the TransformerModel, DataSet, and GenerateText classes in the following Jupyter cells.
2. Run the train_model function and the write_output_to_file functions.
3. Load and preprocess the dataset in another cell. Make sure you have the "beatles.txt" file available in the working directory.
4. Run the rest of code with train_model() with appropriate parameters for number of epochs and attention heads.
5. The rest of the codes will give you the generated text using the generate_text function from the GenerateText class.

3 System Configurations

[69]:
```
# Get the number of CPU cores
num_cores = psutil.cpu_count()

# Get the operating system name and version
os_name = f"{psutil.os.name} {psutil.os.uname().release}"

# Get the system uptime
uptime = psutil.boot_time()

# Get the total system memory and swap memory
total_memory = psutil.virtual_memory().total
swap_memory = psutil.swap_memory().total

# Format the output
output = f"""
System Information
------------------
CPU cores\t: {num_cores}
Operating System\t: {os_name}
System uptime\t: {psutil.datetime.datetime.fromtimestamp(uptime).
 ↪strftime("%Y-%m-%d %H:%M:%S")}
Total Memory\t: {total_memory // (1024**3)} GB
Swap Memory\t: {swap_memory // (1024**3)} GB
"""

# Print the output
print(output)
```

```
System Information
------------------
```

```
CPU cores       : 64
Operating System        : posix 5.15.0-69-generic
System uptime   : 2023-04-21 15:22:09
Total Memory    : 251 GB
Swap Memory     : 1 GB
```

[70]:
```
tf.__version__
```

[70]:
```
'2.12.0'
```

[ ]: