Configuration Manual

MSc Research Project
MSc in Data Analytics

Sarthak Bhatnagar
Student ID: X21185352

School of Computing
National College of Ireland

Supervisor: Dr. Giovani Estrada

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sarthak Bhatnagar
Student ID:	X21185352
Programme:	MSc in Data Analytics **Year:** 2022-23
Module:	Research Project
Lecturer:	Dr. Giovani Estrada
Submission Due Date:	18-09-2023
Project Title:	A comprehensive evaluation of stacked autoencoders for text embedding.
Word Count:	935 **Page Count: 12**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Sarthak Bhatnagar

Date: 16-09-2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sarthak Bhatnagar
Student ID: X21185352

1 Introduction-

This Configuration Manual document specifies the system design in terms of software and hardware required to implement the research project, "A comprehensive evaluation of stacked autoencoders for text embedding. Moreover, it lays out the stepwise instructions to execute the project.

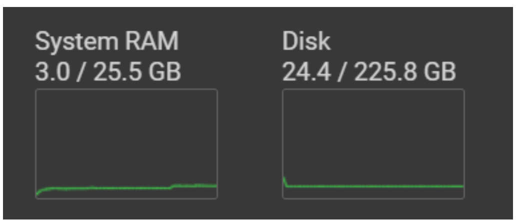
2 System Configuration-

2.1 Software Requirements:

- Google Colab: Google Colab is a free-of-charge product from Google that allows to write and execute Python code. It is an online version of the Jupyter notebook and is mainly suited for machine learning and data analysis-related tasks.
- Python Programming Language: Python is a prevalent programming language and is famous among people who call themselves Pythonistas. The language is well known for developing software, creating websites, and Data Analysis tasks.
- Google Drive: Google Drive is one of the cloud-based free services which allow users to store data and access files online from their devices. To access this service, the user must have a registered Gmail account.
- Jupyter Notebook: A Jupyter Notebook is an open-source application that allows user to execute and run all aspects of data analysis project with the help of a web browser.

2.2 Hardware Requirements:

- Dell Inspiron 14
- Processor: 11th Gen Intel(R) Core(TM) i5-11300H @ 3.10GHz.
- Installed RAM: 16.0 GB (15.7 GB usable).
- System Type: 64-bit operating system, x64-based processor.



. 2.3 Libraries Installed:

These are the necessary libraries that were imported and installed for the completion of the project.

```
[2]  # Importing libraries.
     import nltk
     import random
     import numpy as np
     import pandas as pd
     from nltk.corpus import brown
     import matplotlib.pyplot as plt
     from nltk.probability import FreqDist
     from keras.utils import pad_sequences
     from keras import Input, Model, optimizers
     from keras.preprocessing.text import Tokenizer
     from sklearn.metrics import classification_report, confusion_matrix
     from keras.layers import Bidirectional, GRU, LSTM, Embedding, RepeatVector, Dense
```

The Dataset Brown Corpus is obtained using the nltk('brown') command, which is used to download the Brown Corpus. It is a very well-known corpus widely used in computational linguistics and Natural Language Processing. The corpus was the first 1 million-word textual data set in the English language.
Link- https://www.nltk.org/book/ch02.html

The textual corpus comprised text from 500 sources and categorized by genre such as news, editorial, fiction, etc. It is the most used corpus for linguistic analysis, especially in POS tagging, word-sense disambiguation, and much more.

```
from nltk.corpus import brown
nltk.download('brown')
# Get the categories from the Brown Corpus
categories = brown.categories()

# Print the categories horizontally
categories_horizontal = ' '.join(categories)
print(categories_horizontal)
```

```
adventure belles_lettres editorial fiction government hobbies humor learned lore mystery news religion reviews romance science_fiction
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
```

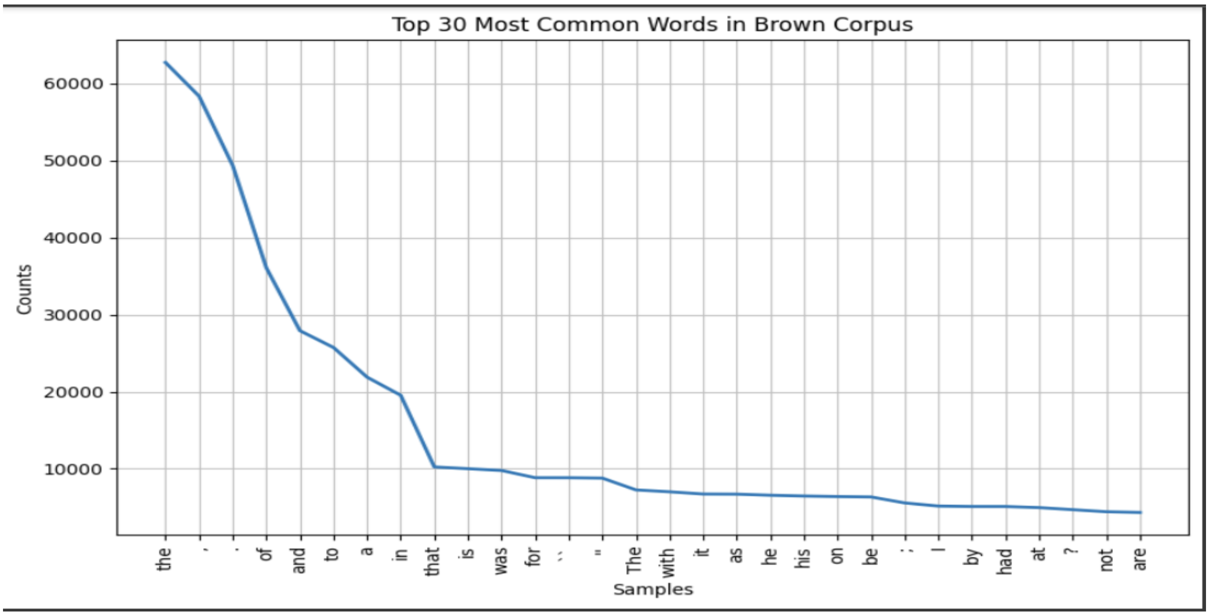
Brown Corpus.

3 Data Preprocessing and Visualization steps.

Data pre-processing is a crucial step in the data analysis process. It involves cleaning and transforming the raw data which is suitable to use for further analysis such as training the model or creating visualizations.

- **Checking the distribution of the top 30 common words in the corpus.**

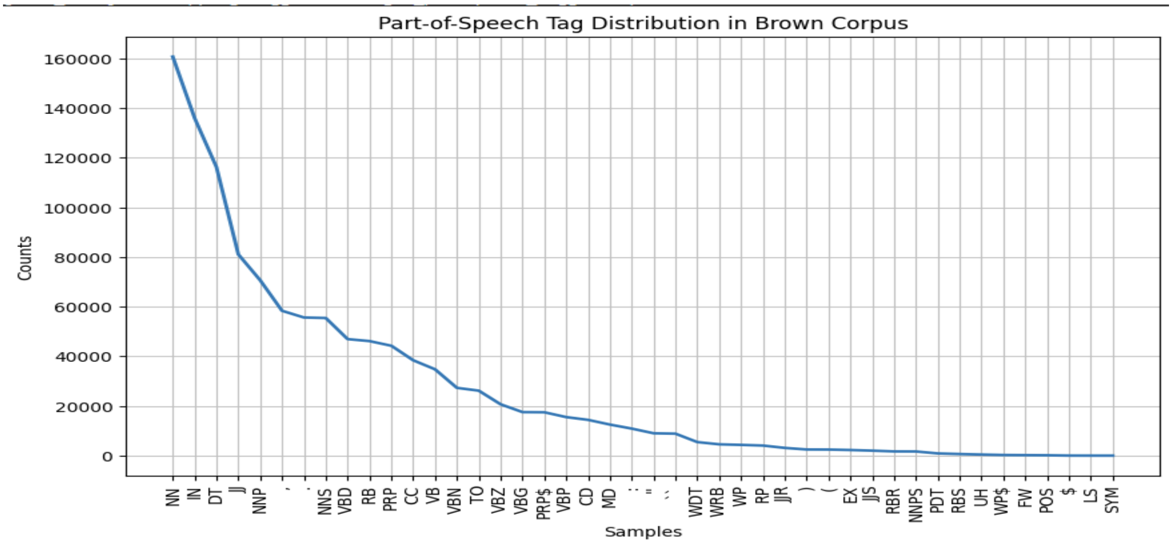
```
[4] # Get words from the corpus
    words = brown.words()

    # Calculate frequency distribution
    fdist = FreqDist(words)

    # Plot the frequency distribution
    plt.figure(figsize=(10, 6))
    fdist.plot(30, title='Top 30 Most Common Words in Brown Corpus')
    plt.show()
```



- **Checking the distributions of POS-tagged words in the corpus.**

POS tagging is one of the important parts of natural processing language which involves assigning a grammatical category to each word in a sentence. It has various applications and is used in Syntactic parsing, semantic understanding information extraction, etc.

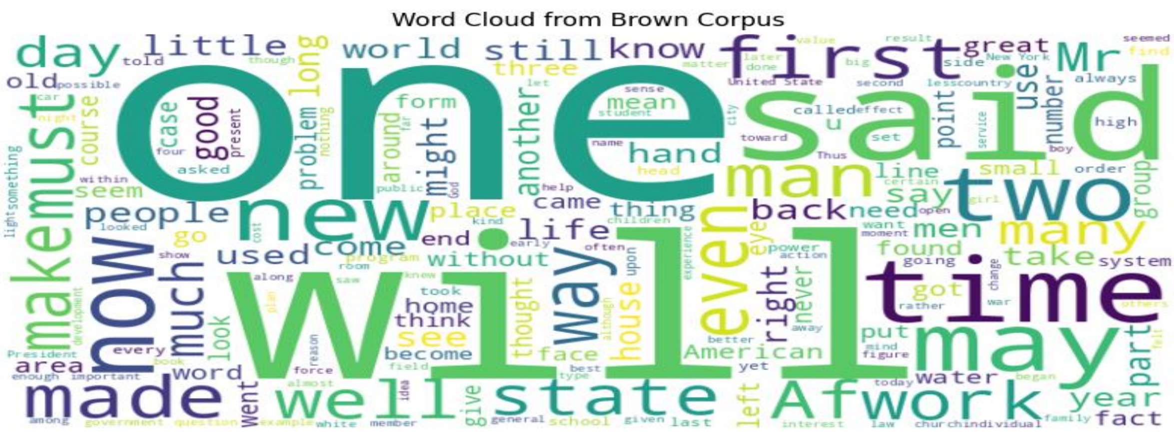
```
from nltk.tag import pos_tag
nltk.download('averaged_perceptron_tagger')

# Get words from the corpus
words = brown.words()

# Get tagged words (word, POS) from the corpus
tagged_words = pos_tag(brown.words())

# Get a list of POS tags
pos_tags = [tag for word, tag in tagged_words]

# Calculate frequency distribution of POS tags
pos_fdist = FreqDist(pos_tags)

# Plot the frequency distribution of POS tags
plt.figure(figsize=(10, 6))
pos_fdist.plot(title='Part-of-Speech Tag Distribution in Brown Corpus')
plt.show()
```



- **Word Cloud for the words in the Corpus.**

Word clouds are used for visualizing text data based on the frequency of words. The words which are more frequently used will be seen in varying sizes and colours.

```
from wordcloud import WordCloud

nltk.download('brown')

# Get words from the corpus
words = brown.words()

# Combine words into a single string
text = ' '.join(words)

# Generate a word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

# Display the word cloud using matplotlib
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud from Brown Corpus')
plt.show()
```

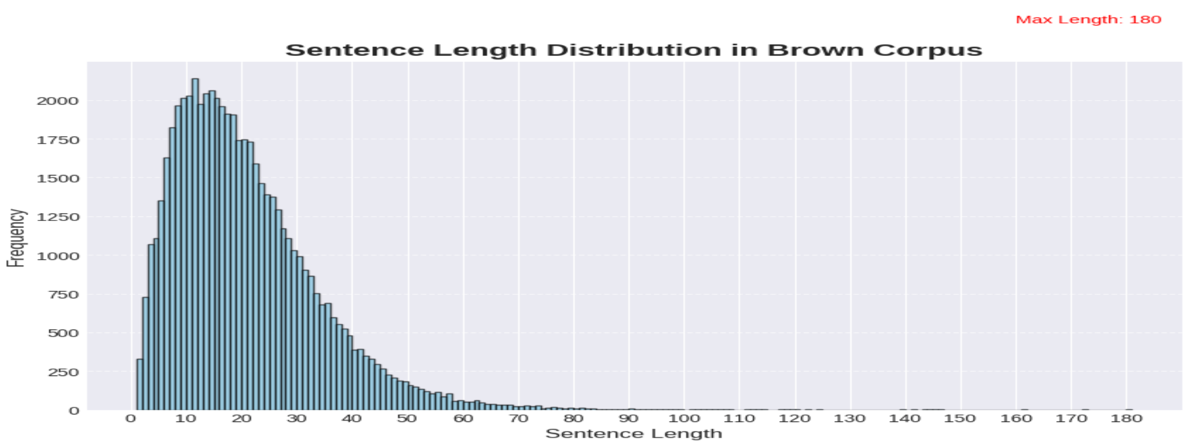


- . **Checking for the count and length of Sentences in the corpus.**

```
[8] nltk.download('brown')
    sents = brown.sents() # Senetences in the corpus # 57340
    len(sents)

[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
57340
```

```
nltk.download('brown')
sents = brown.sents()
maxlen = max([len(s) for s in sents])
print(maxlen)        # Maximum length is sentence is 180.

[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
180
```

```
[ ] # Calculate sentence lengths
    sent_lengths = [len(sent) for sent in sents]

    # Get the maximum sentence length
    maxlen = max(sent_lengths)

    # Customize plot style
    plt.style.use('seaborn-darkgrid')
    plt.figure(figsize=(10, 6))

    # Plot the sentence length distribution
    plt.hist(sent_lengths, bins=range(1, maxlen + 2), edgecolor='black', color='skyblue', alpha=0.7)
    plt.title('Sentence Length Distribution in Brown Corpus', fontsize=16, fontweight='bold')
    plt.xlabel('Sentence Length', fontsize=12)
    plt.ylabel('Frequency', fontsize=12)
    plt.xticks(range(0, maxlen + 1, 10), fontsize=10)
    plt.yticks(fontsize=10)
    plt.grid(axis='y', linestyle='--', alpha=0.7)


    plt.text(maxlen - 20, 2500, 'Max Length: {}'.format(maxlen),
             fontsize=10, color='red', bbox=dict(facecolor='white', alpha=0.7, edgecolor='none'))

    # Show the plot
    plt.show()
```



- **Checking for Vocab size in the Corpus**.

```
vocab = set(brown.words())
num_words = len(vocab)
print(num_words)    # Number of unique words 56057.
print(len(sents))   # Length of senetences 57340.
```

```
56057
57340
```

- **Initializing the parameters used in the model.**

```
# magic numbers
num_words = 10000 #maximum number of words in the vocabulary .
embed_dim = 128  #dimensionality of the word embeddings.
batch_size = 512  #data samples will be processed in each iteration.
maxlen = 60 #maximum length of input sequences.
epochs = 10   #number of times the entire training dataset will be used to train the model.
workers = 16 #it speed up data processing by performing tasks concurrently.
```

- **Creating a tokenizer and padding the sequences up to a fixed length.**

```
# Tokenizing and Padding
tokenizer = Tokenizer(num_words = num_words, split=' ')  #Tokenizer: text into sequence of Tokens.
tokenizer.fit_on_texts(sents) #Fitting the tokenizer on the input text data.
seqs = tokenizer.texts_to_sequences(sents) #converts the list of sentences into sequences of integers.
pad_seqs = pad_sequences(seqs, maxlen) #Pdding the sequeneces upto max Length.
```

- In this section of code, training data is pre-processed by converting the input text data into sequences of integer tokens and padding the sequences to a fixed length.
- Padding the sequences is an essential step that is used in output to produce target sequences of the same length, and it is used to create consistent mini-batches and enable efficient computation.

4 Model Implementation-

4.1(a) Bidirectional GRU Model-

```
# Encoder Model
encoder_inputs = Input(shape=(maxlen,))
emb_layer = Embedding(num_words, embed_dim,input_length = maxlen, mask_zero=False)
x = emb_layer(encoder_inputs)
x = Bidirectional(GRU(embed_dim, activation='relu'))(x)
encoder_model = Model(inputs=encoder_inputs, outputs=x, name='Encoder-Model')
seq2seq_encoder_out = encoder_model(encoder_inputs)

# Decoder Model
decoded = RepeatVector(maxlen)(seq2seq_encoder_out)
x = Bidirectional(GRU(embed_dim, return_sequences=True))(decoded)
decoder_outputs = Dense(num_words, activation='softmax')(x)

# Combining Model and Training
seq2seq_Model = Model(encoder_inputs, decoder_outputs)
seq2seq_Model.compile(optimizer="Adam", \
        loss='sparse_categorical_crossentropy', metrics=['accuracy'])
seq2seq_Model.summary()
```

- Training the bidirectional GRU-based Autoencoder.

```
print('Model fit (epochs=%d)' % epochs)
history = seq2seq_Model.fit(pad_seqs, np.expand_dims(pad_seqs, -1),
          validation_split=1/epochs,
          workers=workers,
          use_multiprocessing=True,
          epochs=epochs)
```

- In this code, the evaluate method is used to check the accuracy which in turn returns how well the autoencoder model is able to reconstruct the data.

```
print('Model fit (epochs=%d)' % epochs)
history = seq2seq_Model.fit(pad_seqs, np.expand_dims(pad_seqs, -1),
          validation_split=0.1,
          workers=workers,
          callbacks=[Early_St1],
          use_multiprocessing=True,
          epochs=epochs)
```

- The codes below are used to generate a plot for Training and Validation accuracy and loss.

```
# Plot training and validation loss curves
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
# Plot training and validation accuracy curves
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

4.1(b) Two-Bidirectional GRU Model.

- Two stacked layer of Bidirectional GRU is used.

```
[15] from keras.layers import TimeDistributed
     encoder_inputs = Input(shape=(maxlen,), name='Encoder-Input')
     emb_layer = Embedding(num_words, embed_dim,input_length = maxlen, name='Body-Word-Embedding', mask_zero=False)
     x = emb_layer(encoder_inputs)
     state_h1 = Bidirectional(GRU(128, activation='relu', name='Encoder-Last-GRU', return_sequences=True))(x)
     state_h2 = Bidirectional(GRU(128, activation='relu', name='Encoder-Last-GRU'))(state_h1)
     encoder_model = Model(inputs=encoder_inputs, outputs=state_h2, name='Encoder-Model')
     seq2seq_encoder_out = encoder_model(encoder_inputs)
```

```
decoded = RepeatVector(maxlen)(seq2seq_encoder_out)
decoder_gru1 = Bidirectional(GRU(128, return_sequences=True, name='Decoder-GRU-before'))
decoder_gru2 = Bidirectional(GRU(128, return_sequences=True, name='Decoder-GRU-before'))
decoder_gru_output = decoder_gru2(decoded)
decoder_dense = Dense(num_words, activation='softmax', name='Final-Output-Dense-before')
decoder_outputs = decoder_dense(decoder_gru_output)
```

```
[18] # Combining Model and Training
     seq2seq_Model = Model(encoder_inputs, decoder_outputs)
     seq2seq_Model.compile(optimizer="Adam", \
             loss='sparse_categorical_crossentropy', metrics=['accuracy'])
     seq2seq_Model.summary()
```

- Early Stopping hyperparameter is used to prevent the model from Overfitting.

```
[19] from keras.callbacks import EarlyStopping
     Early_St1 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2)
```

```
print('Model fit (epochs=%d)' % epochs)
history = seq2seq_Model.fit(pad_seqs, np.expand_dims(pad_seqs, -1),
          validation_split=0.2, #validation_split=1/epochs
          workers=workers,
          callbacks=[Early_St1],
          use_multiprocessing=True,
          epochs=epochs)
```

4.1(c) Three-Bidirectional GRU Model.

```
[26] from keras.layers import TimeDistributed
     encoder_inputs = Input(shape=(maxlen,), name='Encoder-Input')
     emb_layer = Embedding(num_words, embed_dim,input_length = maxlen, name='Body-Word-Embedding', mask_zero=False)
     x = emb_layer(encoder_inputs)
     state_h1 = Bidirectional(GRU(128, activation='relu', name='Encoder-Last-GRU', return_sequences=True))(x)
     state_h2 = Bidirectional(GRU(128, activation='relu', name='Encoder-Last-GRU', return_sequences=True))(state_h1)
     state_h3 = Bidirectional(GRU(128, activation='relu', name='Encoder-Last-GRU'))(state_h2)
     encoder_model = Model(inputs=encoder_inputs, outputs=state_h3, name='Encoder-Model')
     seq2seq_encoder_out = encoder_model(encoder_inputs)
```

```
decoded = RepeatVector(maxlen)(seq2seq_encoder_out)
decoder_gru1 = Bidirectional(LSTM(128, return_sequences=True, name='Decoder-GRU-before'))
decoder_gru2 = Bidirectional(LSTM(128, return_sequences=True, name='Decoder-GRU-before'))
decoder_gru3 = Bidirectional(LSTM(128, return_sequences=True, name='Decoder-GRU-before'))
decoder_gru_output = decoder_gru3(decoded)
decoder_dense = Dense(num_words, activation='softmax', name='Final-Output-Dense-before')
decoder_outputs = decoder_dense(decoder_gru_output)
```

```
[ ] from keras.callbacks import EarlyStopping
    Early_St1 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2)
```

```
print('Model fit (epochs=%d)' % epochs)
history = seq2seq_Model.fit(pad_seqs, np.expand_dims(pad_seqs, -1),
          validation_split=0.2, #validation_split=1/epochs
          workers=workers,
          callbacks=[Early_St1],
          use_multiprocessing=True,
          epochs=epochs)
```

4.1(d) Four-Bidirectional GRU Model.

```
encoder_inputs = Input(shape=(maxlen,), name='Encoder-Input')
emb_layer = Embedding(num_words, embed_dim,input_length = maxlen, name='Body-Word-Embedding', mask_zero=False)
x = emb_layer(encoder_inputs)
state_h1 = Bidirectional(GRU(128, activation='relu', name='Encoder-Last-GRU', return_sequences=True))(x)
state_h2 = Bidirectional(GRU(128, activation='relu', name='Encoder-Last-GRU', return_sequences=True))(state_h1)
state_h3 = Bidirectional(GRU(128, activation='relu', name='Encoder-Last-GRU',return_sequences=True))(state_h2)
state_h4 = Bidirectional(GRU(128, activation='relu', name='Encoder-Last-GRU'))(state_h3)
encoder_model = Model(inputs=encoder_inputs, outputs=state_h4, name='Encoder-Model')
seq2seq_encoder_out = encoder_model(encoder_inputs)
```

```
[17] decoded = RepeatVector(maxlen)(seq2seq_encoder_out)
     decoder_gru = Bidirectional(GRU(128, return_sequences=True, name='Decoder-GRU-before'))
     decoder_gru = Bidirectional(GRU(128, return_sequences=True, name='Decoder-GRU-before'))
     decoder_gru = Bidirectional(GRU(128, return_sequences=True, name='Decoder-GRU-before'))
     decoder_gru = Bidirectional(GRU(128, return_sequences=True, name='Decoder-GRU-before'))
     decoder_gru_output = decoder_gru(decoded)
     decoder_dense = Dense(num_words, activation='softmax', name='Final-Output-Dense-before')
     decoder_outputs = decoder_dense(decoder_gru_output)
```

```
[19] # Combining Model and Training
     seq2seq_Model3 = Model(encoder_inputs, decoder_outputs)
     seq2seq_Model3.compile(optimizer="Adam", \
             loss='sparse_categorical_crossentropy', metrics=['accuracy'])
     seq2seq_Model3.summary()
```

```
[20] from keras.callbacks import EarlyStopping
     Early_St1 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2)
```

```
import time

# Record the start time
start_time = time.time()


print('Model fit (epochs=%d)' % epochs)
history = seq2seq_Model3.fit(pad_seqs, np.expand_dims(pad_seqs, -1),
          validation_split=0.2, #validation_split=1/epochs
          workers=workers,
          callbacks=[Early_St1],
          use_multiprocessing=True,
          epochs=epochs)

# Calculate and display the training time
end_time = time.time()
training_time_seconds = end_time - start_time

# Convert training time to hours and minutes
training_hours = int(training_time_seconds // 3600)
training_minutes = int((training_time_seconds % 3600) // 60)

print('Training time: {} hours and {} minutes'.format(training_hours, training_minutes))
```

4.2(a) One-Bidirectional LSTM Model.

```
[ ] # Encoder Model
    encoder_inputs = Input(shape=(maxlen,))
    emb_layer = Embedding(num_words, embed_dim,input_length = maxlen, mask_zero=False)
    x = emb_layer(encoder_inputs)
    x = Bidirectional(LSTM(embed_dim, activation='relu'))(x)
    seq2seq_encoder_out = Model(inputs=encoder_inputs, outputs=x)(encoder_inputs)

    # Decoder Model
    decoded = RepeatVector(maxlen)(seq2seq_encoder_out)
    x = Bidirectional(LSTM(embed_dim, return_sequences=True))(decoded)
    decoder_outputs = Dense(num_words, activation='softmax')(x)

    # Combining Model and Training
    seq2seq_Model_lstm = Model(encoder_inputs, decoder_outputs)
    seq2seq_Model_lstm.compile(optimizer="Adam", \
            loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    seq2seq_Model_lstm.summary()
```

```
[ ] from keras.callbacks import EarlyStopping
    Early_St1 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2)
```

```
print('Model fit (epochs=%d)' % epochs)
history = seq2seq_Model.fit(pad_seqs, np.expand_dims(pad_seqs, -1),
          validation_split=0.2, #validation_split=1/epochs
          workers=workers,
          callbacks=[Early_St1],
          use_multiprocessing=True,
          epochs=epochs)
```

```
# Plot training and validation loss curves
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
# Plot training and validation accuracy curves
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

4.2(b) Two-Bidirectional LSTM Model.

```
encoder_inputs = Input(shape=(maxlen,), name='Encoder-Input')
emb_layer = Embedding(num_words, embed_dim,input_length = maxlen, name='Body-Word-Embedding', mask_zero=False)
x = emb_layer(encoder_inputs)
state_h1 = Bidirectional(LSTM(128, activation='relu', name='Encoder-Last-LSTM', return_sequences=True))(x)
state_h2 = Bidirectional(LSTM(128, activation='relu', name='Encoder-Last-LSTM'))(state_h1)
encoder_model = Model(inputs=encoder_inputs, outputs=state_h2, name='Encoder-Model')
seq2seq_encoder_out = encoder_model(encoder_inputs)
```

```
[41] decoded = RepeatVector(maxlen)(seq2seq_encoder_out)
decoder_lstm1 = Bidirectional(LSTM(128, return_sequences=True, name='Decoder-LSTM-before'))
decoder_lstm2 = Bidirectional(LSTM(128, return_sequences=True, name='Decoder-LSTM-before'))
decoder_lstm_output = decoder_lstm2(decoded)
decoder_dense = Dense(num_words, activation='softmax', name='Final-Output-Dense-before')
decoder_outputs = decoder_dense(decoder_lstm_output)
```

```
[42] # Combining Model and Training
seq2seq_Model2 = Model(encoder_inputs, decoder_outputs)
seq2seq_Model2.compile(optimizer="Adam", \
        loss='sparse_categorical_crossentropy', metrics=['accuracy'])
seq2seq_Model2.summary()
```

```
[19] from keras.callbacks import EarlyStopping
Early_St1 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2)
```

```
print('Model fit (epochs=%d)' % epochs)
history = seq2seq_Model.fit(pad_seqs, np.expand_dims(pad_seqs, -1),
          validation_split=0.2, #validation_split=1/epochs
          workers=workers,
          callbacks=[Early_St1],
          use_multiprocessing=True,
          epochs=epochs)
```

4.2(c) Three-Bidirectional LSTM Model.

```
encoder_inputs = Input(shape=(maxlen,), name='Encoder-Input')
emb_layer = Embedding(num_words, embed_dim,input_length = maxlen, name='Body-Word-Embedding', mask_zero=False)
x = emb_layer(encoder_inputs)
state_h1 = Bidirectional(LSTM(128, activation='relu', name='Encoder-Last-LSTM', return_sequences=True))(x)
state_h2 = Bidirectional(LSTM(128, activation='relu', name='Encoder-Last-LSTM',return_sequences=True))(state_h1)
state_h3 = Bidirectional(LSTM(128, activation='relu', name='Encoder-Last-LSTM'))(state_h2)
encoder_model = Model(inputs=encoder_inputs, outputs=state_h3, name='Encoder-Model')
seq2seq_encoder_out = encoder_model(encoder_inputs)
```

```
[71] decoded = RepeatVector(maxlen)(seq2seq_encoder_out)
decoder_lstm = Bidirectional(LSTM(128, return_sequences=True, name='Decoder-LSTM-before'))
decoder_lstm = Bidirectional(LSTM(128, return_sequences=True, name='Decoder-LSTM-before'))
decoder_lstm = Bidirectional(LSTM(128, return_sequences=True, name='Decoder-LSTM-before'))
decoder_lstm_output = decoder_lstm(decoded)
decoder_dense = Dense(num_words, activation='softmax', name='Final-Output-Dense-before')
decoder_outputs = decoder_dense(decoder_lstm_output)
```

```
[72] # Combining Model and Training
seq2seq_Model = Model(encoder_inputs, decoder_outputs)
seq2seq_Model.compile(optimizer="Adam", \
        loss='sparse_categorical_crossentropy', metrics=['accuracy'])
seq2seq_Model.summary()
```

```
[19] from keras.callbacks import EarlyStopping
     Early_St1 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2)
```

```
print('Model fit (epochs=%d)' % epochs)
history = seq2seq_Model.fit(pad_seqs, np.expand_dims(pad_seqs, -1),
          validation_split=0.2, #validation_split=1/epochs
          workers=workers,
          callbacks=[Early_St1],
          use_multiprocessing=True,
          epochs=epochs)
```

4.2(d) Four-Bidirectional LSTM Model.

```
[59] encoder_inputs = Input(shape=(maxlen,), name='Encoder-Input')
     emb_layer = Embedding(num_words, embed_dim,input_length = maxlen, name='Body-Word-Embedding', mask_zero=False)
     x = emb_layer(encoder_inputs)
     state_h1 = Bidirectional(LSTM(128, activation='relu', name='Encoder-Last-LSTM', return_sequences=True))(x)
     state_h2 = Bidirectional(LSTM(128, activation='relu', name='Encoder-Last-LSTM', return_sequences=True))(state_h1)
     state_h3 = Bidirectional(LSTM(128, activation='relu', name='Encoder-Last-LSTM',return_sequences=True))(state_h2)
     state_h4 = Bidirectional(LSTM(128, activation='relu', name='Encoder-Last-LSTM'))(state_h3)
     encoder_model = Model(inputs=encoder_inputs, outputs=state_h4, name='Encoder-Model')
     seq2seq_encoder_out = encoder_model(encoder_inputs)
```

```
[60] decoded = RepeatVector(maxlen)(seq2seq_encoder_out)
     decoder_lstm = Bidirectional(LSTM(128, return_sequences=True, name='Decoder-LSTM-before'))
     decoder_lstm = Bidirectional(LSTM(128, return_sequences=True, name='Decoder-LSTM-before'))
     decoder_lstm = Bidirectional(LSTM(128, return_sequences=True, name='Decoder-LSTM-before'))
     decoder_lstm = Bidirectional(LSTM(128, return_sequences=True, name='Decoder-LSTM-before'))
     decoder_lstm_output = decoder_lstm(decoded)
     decoder_dense = Dense(num_words, activation='softmax', name='Final-Output-Dense-before')
     decoder_outputs = decoder_dense(decoder_lstm_output)
```

```
[61] # Combining Model and Training
     seq2seq_Model3 = Model(encoder_inputs, decoder_outputs)
     seq2seq_Model3.compile(optimizer="Adam", \
             loss='sparse_categorical_crossentropy', metrics=['accuracy'])
     seq2seq_Model.summary()
```