Deep Learning - HW4

Q1.

Doc2Vec Model With RNN, LSTM, GRU, Bi-RNN, Bi-LSTM, Bi-GRU

```
doc2vec_model = Doc2Vec.load("/content/drive/MyDrive/DeepLearning/HW/HW4/doc2vec_model")

# Get document embeddings
doc_embeddings = np.array([doc2vec_model.infer_vector(doc.split()) for doc in df['clean_review']])

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(doc_embeddings, df['target'], test_size=0.2, random_state=42)

# Reshape the input data for all bidirectional models
X_train_reshaped = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test_reshaped = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

models = [
    (SimpleRNN(128), "RNN"),
    (LSTM(128), "LSTM"),
    (GRU(128), "GRU"),
    (Bidirectional(SimpleRNN(128, return_sequences=True), input_shape=(1, doc_embeddings.shape[1])), "Bi-RNN"),
    (Bidirectional(LSTM(128, return_sequences=True), input_shape=(1, doc_embeddings.shape[1])), "Bi-LSTM"),
    (Bidirectional(GRU(128, return_sequences=True), input_shape=(1, doc_embeddings.shape[1])), "Bi-GRU")
]
```

```
Classification Report for RNN with Doc2Vec :
              precision    recall  f1-score   support

           0       0.84      0.85      0.85      4961
           1       0.85      0.84      0.85      5039

    accuracy                           0.85     10000
   macro avg       0.85      0.85      0.85     10000
weighted avg       0.85      0.85      0.85     10000

Confusion Matrix for RNN with Doc2Vec :
[[4219  742]
 [ 801 4238]]
RNN with Doc2Vec embedding has an Accuracy of: 84.57%
```

```
Classification Report for LSTM with Doc2Vec :
              precision    recall  f1-score   support

           0       0.84      0.85      0.84      4961
           1       0.85      0.84      0.84      5039

    accuracy                           0.84     10000
   macro avg       0.84      0.84      0.84     10000
weighted avg       0.84      0.84      0.84     10000

Confusion Matrix for LSTM with Doc2Vec :
[[4198  763]
 [ 816 4223]]
LSTM with Doc2Vec embedding has an Accuracy of: 84.21%
```

Classification Report for GRU with Doc2Vec :

	precision	recall	f1-score	support
0	0.85	0.82	0.83	4961
1	0.83	0.85	0.84	5039
accuracy			0.84	10000
macro avg	0.84	0.84	0.84	10000
weighted avg	0.84	0.84	0.84	10000

Confusion Matrix for GRU with Doc2Vec :
[[4057 904]
[740 4299]]
GRU with Doc2Vec embedding has an Accuracy of: 83.56%

Classification Report for Bi-RNN with Doc2Vec :

	precision	recall	f1-score	support
0	0.84	0.85	0.84	4961
1	0.85	0.84	0.85	5039
accuracy			0.84	10000
macro avg	0.84	0.84	0.84	10000
weighted avg	0.84	0.84	0.84	10000

Confusion Matrix for Bi-RNN with Doc2Vec :
[[4210 751]
[803 4236]]
Bi-RNN with Doc2Vec embedding has an Accuracy of: 84.46%

Classification Report for Bi-LSTM with Doc2Vec :

	precision	recall	f1-score	support
0	0.82	0.84	0.83	4961
1	0.84	0.82	0.83	5039
accuracy			0.83	10000
macro avg	0.83	0.83	0.83	10000
weighted avg	0.83	0.83	0.83	10000

Confusion Matrix for Bi-LSTM with Doc2Vec :
[[4179 782]
[898 4141]]
Bi-LSTM with Doc2Vec embedding has an Accuracy of: 83.20%

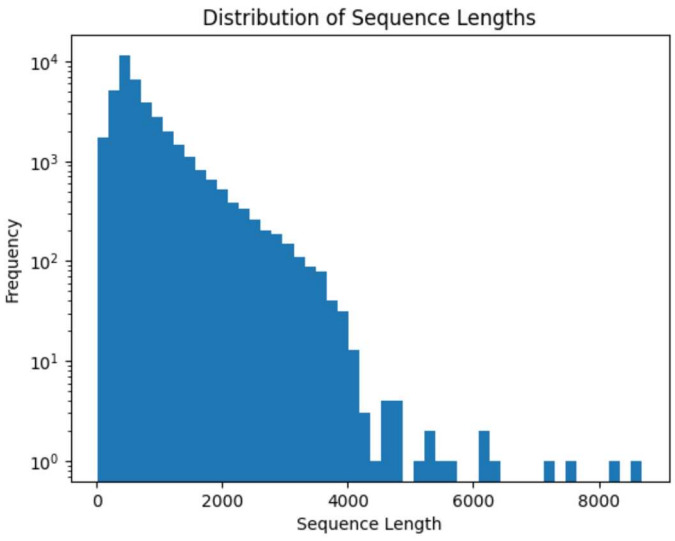
```
Classification Report for Bi-GRU with Doc2Vec :
              precision    recall  f1-score   support

           0       0.84      0.82      0.83      4961
           1       0.83      0.85      0.84      5039

    accuracy                           0.83     10000
   macro avg       0.83      0.83      0.83     10000
weighted avg       0.83      0.83      0.83     10000

Confusion Matrix for Bi-GRU with Doc2Vec :
[[4067  894]
 [ 763 4276]]
Bi-GRU with Doc2Vec embedding has an Accuracy of: 83.43%
```

Q1a. Word2Vec, Fasttext, Glove with RNN, LSTM, GRU, Bi-RNN, Bi-LSTM, Bi-GRU

```
# Calculate the maximum number of words dynamically
max_words = len(set(" ".join(X_train).split()))
print("Maximum number of words:", max_words)
```

```
Maximum number of words: 82241
```

```
# Tokenization
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_train)
X_train_sequence = tokenizer.texts_to_sequences(X_train)
X_test_sequence = tokenizer.texts_to_sequences(X_test)
```



The distribution of sequence I checked to see how much max length should I keep to keep the maximum information intact. I tried with 8000, 4000, and 2000 but it was taking way too long. So, going with 400 because it has highest frequency and also for efficiency.

```
# Load models

word2vec_model = api.load("word2vec-google-news-300")
fasttext_model = FastText.load_fasttext_format("/content/drive/MyDrive/DeepLearning/HW/HW4/wiki.en/wiki.en.bin")
glove_model = KeyedVectors.load_word2vec_format("/content/drive/MyDrive/DeepLearning/HW/HW4/glove.6B.300d.txt", binary=False, no_header=True)
```

```
# Word Embeddings
embeddings = {
    'FastText': fasttext_model,
    'Word2Vec': word2vec_model,
    'GloVe': glove_model,
}
```

```
# Sequential Models
models = {
    'RNN': SimpleRNN,
    'LSTM': LSTM,
    'GRU': GRU,
    'Bi-RNN': lambda units, **kwargs: Bidirectional(SimpleRNN(units, **kwargs)),
    'Bi-LSTM': lambda units, **kwargs: Bidirectional(LSTM(units, **kwargs)),
    'Bi-GRU': lambda units, **kwargs: Bidirectional(GRU(units, **kwargs)),
}
```

```
def load_pretrained_embedding_matrix(embedding_model, word_index, max_words, embedding_dim):
    # Initialize the embedding matrix with zeros
    embedding_matrix = np.zeros((max_words, embedding_dim))

    # Iterate over the words in the tokenizer's word index
    for word, i in word_index.items():
        if i < max_words:
            try:
                # Get the embedding vector for the word from the pre-trained model
                if isinstance(embedding_model, dict):
                    embedding_vector = embedding_model.get(word) # For GloVe
                else:
                    #, FastText, Doc2Vec, Word2Vec
                    embedding_vector = embedding_model[word]

                if embedding_vector is not None:
                    embedding_matrix[i] = embedding_vector
            except KeyError:
                # Handle the case where the word is not in the pre-trained model vocabulary
                pass

    return embedding_matrix
```

```
# Define the model checkpoint callback
checkpoint_filepath = '/content/drive/MyDrive/DeepLearning/HW/HW4/best_model_doc2vec.h5'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_best_only=True,
    monitor='val_accuracy',  # Choose the metric to monitor for saving the best model
    mode='max',  # Use 'max' if the metric should be maximized, 'min' if minimized
    verbose=1
)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=1e-6)
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

```python
model.add(embedding_layer)
model.add(model_type(128))  # Adjust units as needed

model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()

# Train the model
model.fit(X_train_padded, y_train, epochs=10, batch_size=32, validation_split=0.2, verbose=0, callbacks=[model_checkpoint_callback,reduce_lr,early_stop])
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 400, 300)          24672300

 simple_rnn_10 (SimpleRNN)   (None, 128)               54912

 dense_24 (Dense)            (None, 1)                 129

=================================================================
Total params: 24727341 (94.33 MB)
Trainable params: 55041 (215.00 KB)
Non-trainable params: 24672300 (94.12 MB)
_________________________________________________________________
```

Same Architecture for all the models was used.

FastText with RNN

```
model_name RNN with FastText has accuracy of 0.5267000198364258
313/313 [==============================] - 8s 26ms/step
Classification Report for RNN with FastText:
              precision    recall  f1-score   support

           0       0.77      0.07      0.12      4961
           1       0.52      0.98      0.68      5039

    accuracy                           0.53     10000
   macro avg       0.64      0.52      0.40     10000
weighted avg       0.64      0.53      0.40     10000

Confusion Matrix for RNN with FastText:
[[ 323 4638]
 [  95 4944]]
```

FastText with LSTM

```
model_name LSTM with FastText has accuracy of 0.8784000277519226
313/313 [==============================] - 2s 6ms/step
Classification Report for LSTM with FastText:
              precision    recall  f1-score   support

           0       0.88      0.87      0.88      4961
           1       0.87      0.89      0.88      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000

Confusion Matrix for LSTM with FastText:
[[4316  645]
 [ 571 4468]]
```

FastText with GRU

```
model_name GRU with FastText has accuracy of 0.8790000081062317
313/313 [==============================] - 2s 6ms/step
Classification Report for GRU with FastText:
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      4961
           1       0.90      0.86      0.88      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000

Confusion Matrix for GRU with FastText:
[[4455  506]
 [ 704 4335]]
```

FastText with Bi-RNN

```
model_name Bi-RNN with FastText has accuracy of 0.6388000249862671
313/313 [==============================] - 17s 50ms/step
Classification Report for Bi-RNN with FastText:
              precision    recall  f1-score   support

           0       0.63      0.65      0.64      4961
           1       0.64      0.63      0.64      5039

    accuracy                           0.64     10000
   macro avg       0.64      0.64      0.64     10000
weighted avg       0.64      0.64      0.64     10000

Confusion Matrix for Bi-RNN with FastText:
[[3206 1755]
 [1857 3182]]
```

FastText with Bi-LSTM

```
model_name Bi-LSTM with FastText has accuracy of 0.8797000050544739
313/313 [==============================] - 4s 12ms/step
Classification Report for Bi-LSTM with FastText:
              precision    recall  f1-score   support

           0       0.90      0.85      0.87      4961
           1       0.86      0.91      0.88      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000

Confusion Matrix for Bi-LSTM with FastText:
[[4208  753]
 [ 450 4589]]
```

FastText with Bi-GRU

```
model_name Bi-GRU with FastText has accuracy of 0.8772000074386597
313/313 [==============================] - 4s 12ms/step
Classification Report for Bi-GRU with FastText:
              precision    recall  f1-score   support

           0       0.88      0.87      0.87      4961
           1       0.87      0.89      0.88      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000

Confusion Matrix for Bi-GRU with FastText:
[[4293  668]
 [ 560 4479]]
```

Word2Vec with RNN

```
model_name RNN with Word2Vec has accuracy of 0.772599995136261
313/313 [==============================] - 9s 27ms/step
Classification Report for RNN with Word2Vec:
              precision    recall  f1-score   support

           0       0.86      0.64      0.74      4961
           1       0.72      0.90      0.80      5039

    accuracy                           0.77     10000
   macro avg       0.79      0.77      0.77     10000
weighted avg       0.79      0.77      0.77     10000

Confusion Matrix for RNN with Word2Vec:
[[3199 1762]
 [ 512 4527]]
```

Word2Vec with LSTM

```
model_name LSTM with Word2Vec has accuracy of 0.8751999735832214
313/313 [==============================] - 2s 6ms/step
Classification Report for LSTM with Word2Vec:
              precision    recall  f1-score   support

           0       0.89      0.85      0.87      4961
           1       0.86      0.90      0.88      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000

Confusion Matrix for LSTM with Word2Vec:
[[4230  731]
 [ 517 4522]]
```

Word2Vec with GRU

```
model_name GRU with Word2Vec has accuracy of 0.8798999786376953
313/313 [==============================] - 2s 6ms/step
Classification Report for GRU with Word2Vec:
              precision    recall  f1-score   support

           0       0.86      0.91      0.88      4961
           1       0.91      0.85      0.88      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000

Confusion Matrix for GRU with Word2Vec:
[[4521  440]
 [ 761 4278]]
```

Word2Vec with Bi-RNN

```
model_name Bi-RNN with Word2Vec has accuracy of 0.6664999723434448
313/313 [==============================] - 16s 51ms/step
Classification Report for Bi-RNN with Word2Vec:
              precision    recall  f1-score   support

           0       0.69      0.59      0.64      4961
           1       0.65      0.74      0.69      5039

    accuracy                           0.67     10000
   macro avg       0.67      0.67      0.66     10000
weighted avg       0.67      0.67      0.66     10000

Confusion Matrix for Bi-RNN with Word2Vec:
[[2941 2020]
 [1315 3724]]
```

Word2Vec with Bi-LSTM

```
model_name Bi-LSTM with Word2Vec has accuracy of 0.8920000195503235
313/313 [==============================] - 4s 12ms/step
Classification Report for Bi-LSTM with Word2Vec:
              precision    recall  f1-score   support

           0       0.89      0.90      0.89      4961
           1       0.90      0.89      0.89      5039

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000

Confusion Matrix for Bi-LSTM with Word2Vec:
[[4445  516]
 [ 564 4475]]
```

Word2Vec with Bi-GRU

```
model_name Bi-GRU with Word2Vec has accuracy of 0.8805000185966492
313/313 [==============================] - 4s 12ms/step
Classification Report for Bi-GRU with Word2Vec:
              precision    recall  f1-score   support

           0       0.88      0.88      0.88      4961
           1       0.88      0.88      0.88      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000

Confusion Matrix for Bi-GRU with Word2Vec:
[[4365  596]
 [ 599 4440]]
```

Glove with RNN

```
model_name RNN with GloVe has accuracy of 0.7426000237464905
313/313 [==============================] - 8s 26ms/step
Classification Report for RNN with GloVe:
              precision    recall  f1-score   support

           0       0.80      0.64      0.71      4961
           1       0.70      0.84      0.77      5039

    accuracy                           0.74     10000
   macro avg       0.75      0.74      0.74     10000
weighted avg       0.75      0.74      0.74     10000

Confusion Matrix for RNN with GloVe:
[[3169 1792]
 [ 782 4257]]
```

Glove with LSTM

```
model_name LSTM with GloVe has accuracy of 0.8820000290870667
313/313 [==============================] - 2s 6ms/step
Classification Report for LSTM with GloVe:
              precision    recall  f1-score   support

           0       0.89      0.88      0.88      4961
           1       0.88      0.89      0.88      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000

Confusion Matrix for LSTM with GloVe:
[[4345  616]
 [ 564 4475]]
```

Glove with GRU

```
model_name GRU with GloVe has accuracy of 0.8791000247001648
313/313 [==============================] - 2s 6ms/step
Classification Report for GRU with GloVe:
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      4961
           1       0.90      0.86      0.88      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000

Confusion Matrix for GRU with GloVe:
[[4479  482]
 [ 727 4312]]
```

Glove with Bi-RNN

```
model_name Bi-RNN with GloVe has accuracy of 0.642799973487854
313/313 [==============================] - 16s 50ms/step
Classification Report for Bi-RNN with GloVe:
              precision    recall  f1-score   support

           0       0.65      0.61      0.63      4961
           1       0.64      0.68      0.66      5039

    accuracy                           0.64     10000
   macro avg       0.64      0.64      0.64     10000
weighted avg       0.64      0.64      0.64     10000

Confusion Matrix for Bi-RNN with GloVe:
[[3010 1951]
 [1621 3418]]
```

Glove with Bi-LSTM

```
model_name Bi-LSTM with GloVe has accuracy of 0.8765000104904175
313/313 [==============================] - 4s 12ms/step
Classification Report for Bi-LSTM with GloVe:
              precision    recall  f1-score   support

           0       0.89      0.86      0.87      4961
           1       0.87      0.89      0.88      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000

Confusion Matrix for Bi-LSTM with GloVe:
[[4267  694]
 [ 541 4498]]
```

Glove with Bi-GRU

```
model_name Bi-GRU with GloVe has accuracy of 0.8823000192642212
313/313 [==============================] - 4s 12ms/step
Classification Report for Bi-GRU with GloVe:
              precision    recall  f1-score   support

           0       0.88      0.89      0.88      4961
           1       0.89      0.88      0.88      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000

Confusion Matrix for Bi-GRU with GloVe:
[[4395  566]
 [ 611 4428]]
```

	Embedding	Model	Accuracy
19	Word2Vec	Bi-LSTM	89.20
12	GloVe	Bi-GRU	88.23
16	GloVe	LSTM	88.20
18	Word2Vec	Bi-GRU	88.05
21	Word2Vec	GRU	87.99
7	FastText	Bi-LSTM	87.97
15	GloVe	GRU	87.91
9	FastText	GRU	87.90
10	FastText	LSTM	87.84
6	FastText	Bi-GRU	87.72
13	GloVe	Bi-LSTM	87.65
22	Word2Vec	LSTM	87.52
5	Doc2Vec	RNN	84.59
2	Doc2Vec	Bi-RNN	84.56
0	Doc2Vec	Bi-GRU	84.55
4	Doc2Vec	LSTM	84.21
3	Doc2Vec	GRU	84.08
1	Doc2Vec	Bi-LSTM	83.20
23	Word2Vec	RNN	77.26
17	GloVe	RNN	74.26
20	Word2Vec	Bi-RNN	66.65
14	GloVe	Bi-RNN	64.28
8	FastText	Bi-RNN	63.88
11	FastText	RNN	52.67

Adam optimizer is used throughout. Binary_crossentropy is used. Batch_size=32, epochs = 10,20 , units are 128, Learning rate is dynamically reduced on plateau by the factor of 0.2 with the minimum lr = 1e-6 with patience level=3. Early stopping is with patience level of 5.

The Highest Performing Model is Bi-LSTM with Word2Vec embedding with accuracy of 89%

Q1b. Use Keras embedding layer with sequential model and use cosine similarity to find the first five most similar words to "movie"

```
# Load IMDb dataset
df = pd.read_csv('/content/drive/MyDrive/DeepLearning/HW/HW4/IMDB_cleaned.csv')

# Tokenize the entire dataset
max_words = 5000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(df['clean_review'])
sequences = tokenizer.texts_to_sequences(df['clean_review'])

# Pad sequences to a fixed length
max_len = 100
padded_sequences = pad_sequences(sequences, maxlen=max_len)

# Build the embedding model
embedding_dim = 100
model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=embedding_dim, input_length=max_len))
model.add(LSTM(128))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model on the entire dataset
model.fit(padded_sequences, df['target'], epochs=5, batch_size=32, verbose=2)
```

```
# Get the word embeddings
embedding_layer = model.layers[0]
weights = embedding_layer.get_weights()[0]

# Get the word index from the tokenizer
word_index = tokenizer.word_index

# Function to get the vector for a word
def get_vector(word):
    if word in word_index and word_index[word] < max_words:
        return weights[word_index[word]]
    else:
        return None
```

```python
# Function to find the most similar words to a given word using cosine similarity, excluding the word itself
def most_similar_words_exclude(word, exclude_word, top_n=5):
    vector1 = get_vector(word)
    if vector1 is None:
        return []

    similarities = cosine_similarity([vector1], weights)[0]

    # Find the indices of the top N most similar words (excluding the given word)
    top_indices = np.argsort(similarities)[::-1]
    top_words = [(list(word_index.keys())[i], similarities[i]) for i in top_indices if list(word_index.keys())[i] != exclude_word][:top_n]

    return top_words

# Find the most similar words to "movie" excluding "movie"
similar_words_movie_exclude_movie = most_similar_words_exclude("movie", exclude_word="movie", top_n=5)
print(f"The most similar words to 'movie' (excluding 'movie') are: {similar_words_movie_exclude_movie}")
```

Results (4 approaches):

Keras + LSTM => The most similar words to 'movie' (excluding 'movie') are: [('**film**', 1.0), ('**obsession**', 0.73142433), ('**lord**', 0.6873765), ('**festival**', 0.66646934), ('**quest**', 0.663234)]

Keras + GRU => The most similar words to 'movie' (excluding 'movie') are: [('**film**', 1.0), ('**home**', 0.6828351), ('**crime**', 0.67553556), ('**jr**', 0.6702849), ('**convey**', 0.65589595)]

Glove Embeddings => The most similar words to 'movie' (excluding 'movie') are: [('**film**', 0.85887855), ('**tourism**', 0.84934735), ('**approach**', 0.79086804), ('**jail**', 0.67923284), ('**calif.**', 0.6750692)]

Word2Vec embeddings => The most similar words to 'movie' (excluding 'movie') are: [('**film**', 0.8676770329475403), ('**movies**', 0.8013108372688293), ('**films**', 0.7363011837005615), ('**moive**', 0.6830360889434814), ('**Movie**', 0.6693680286407471)]

Q2. One Step Ahead Forecasting - Moby Dick Chapter 4

Problem 2: 1. Perform Text Preprocessing

- Tokenization
- Convert to lower case
- Expand contraction
- Remove punctuation
- Lemmatization

```
# Text preprocessing
def preprocess_text(text):
    # Tokenization
    tokens = word_tokenize(text)

    # Convert to lowercase
    tokens = [word.lower() for word in tokens]

    # Expand contractions
    # Example: {'isn't': 'is not', 'haven't': 'have not', ...}
    tokens = [contractions.fix(word) for word in tokens]

    # Remove punctuation except for newline character
    # Remove punctuation except for newline character
    punctuation_except_newline = string.punctuation.replace('\n', '')
    tokens = [word if word == '\n' or word.strip(punctuation_except_newline + "'") else '\n' for word in tokens]

    # Lemmatization (using WordNet lemmatizer from NLTK)
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    return ' '.join(tokens)

start = time.time()
# Apply preprocessing
tokens = preprocess_text(text)
stop = time.time()
print(f'Cleaning took: {round((stop-start)/60, 3)} minutes')
```

```
Cleaning took: 0.002 minutes
```

Problem 2: Q2. Keras Embedding Next Word Forecaster

```
tokenizer.word_index
```

```
{'the': 1,
 'a': 2,
 'and': 3,
 'of': 4,
 'i': 5,
 'to': 6,
 'in': 7,
 'it': 8,
 'that': 9,
 'his': 10,
 'he': 11,
 'was': 12,
 'but': 13,
 'me': 14,
 'with': 15,
 'as': 16,
 'this': 17,
 'at': 18,
 'you': 19,
 'is': 20,
 'all': 21,
```

```
input_sequences = []
for line in tokens.split('\n'):
  #print(line)
  token_list = tokenizer.texts_to_sequences([line])[0]
  #print(token_list)
  for i in range(1, len(token_list)):
      n_gram_sequence = token_list[:i+1]
      #print(n_gram_sequence)
      input_sequences.append(n_gram_sequence)
      #print(input_sequence)
```

```
input_sequences = []
for line in tokens.split('\n'):
  #print(line)
  token_list = tokenizer.texts_to_sequences([line])[0]
  #print(token_list)
  for i in range(1, len(token_list)):
      n_gram_sequence = token_list[:i+1]
      #print(n_gram_sequence)
      input_sequences.append(n_gram_sequence)
      #print(input_sequence)
```

```
max_sequence_len = max([len(seq) for seq in input_sequences])
input_sequence =  np.array(pad_sequences(input_sequences, maxlen = max_sequence_len, padding='pre' ))
```

```
input_sequence
```

```
array([[   0,    0,    0, ...,    0,  558,   14],
       [   0,    0,    0, ...,  558,   14,  220],
       [   0,    0,    0, ...,   14,  220,   48],
       ...,
       [   0,    0,    0, ...,  241,   51,    2],
       [   0,    0,    0, ...,   51,    2, 2726],
       [   0,    0,    0, ...,    2, 2726, 2727]], dtype=int32)
```

```
X = input_sequence[:, :-1]
y = input_sequence[:,-1]
```

```
X[0]
```

```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0, 558], dtype=int32)
```

```
y[0]
```

```
14
```

```
X[1]
```

```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0, 558,  14], dtype=int32)
```

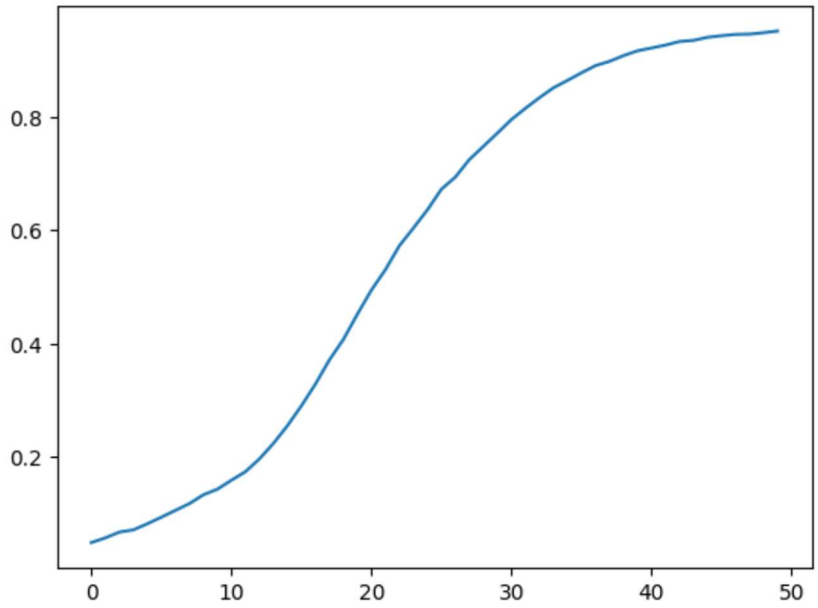
```
y = np.array(tf.keras.utils.to_categorical(y, num_classes= total_words))
y
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 16, 100)           272800

 lstm_2 (LSTM)               (None, 150)               150600

 dense_2 (Dense)             (None, 2728)              411928

=================================================================
Total params: 835328 (3.19 MB)
Trainable params: 835328 (3.19 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
```

```
[<matplotlib.lines.Line2D at 0x7df0cc0bd0f0>]
```



Token List While generating words

```
[28, 36, 20, 87, 1004]
1/1 [==============================] - 0s 370ms/step
[28, 36, 20, 87, 1004, 320]
1/1 [==============================] - 0s 19ms/step
[28, 36, 20, 87, 1004, 320, 4]
1/1 [==============================] - 0s 17ms/step
[28, 36, 20, 87, 1004, 320, 4, 1]
1/1 [==============================] - 0s 18ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005]
1/1 [==============================] - 0s 18ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006]
1/1 [==============================] - 0s 20ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113]
1/1 [==============================] - 0s 18ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40]
1/1 [==============================] - 0s 17ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40, 1]
1/1 [==============================] - 0s 17ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40, 1, 208]
1/1 [==============================] - 0s 18ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40, 1, 208, 3]
1/1 [==============================] - 0s 20ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40, 1, 208, 3, 87]
1/1 [==============================] - 0s 19ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40, 1, 208, 3, 87, 62]
1/1 [==============================] - 0s 18ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40, 1, 208, 3, 87, 62, 557]
1/1 [==============================] - 0s 18ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40, 1, 208, 3, 87, 62, 557, 3]
1/1 [==============================] - 0s 18ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40, 1, 208, 3, 87, 62, 557, 3, 1]
1/1 [==============================] - 0s 19ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40, 1, 208, 3, 87, 62, 557, 3, 1, 698]
1/1 [==============================] - 0s 19ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40, 1, 208, 3, 87, 62, 557, 3, 1, 698, 67]
1/1 [==============================] - 0s 20ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40, 1, 208, 3, 87, 62, 557, 3, 1, 698, 67, 8]
1/1 [==============================] - 0s 18ms/step
[28, 36, 20, 87, 1004, 320, 4, 1, 1005, 1006, 113, 40, 1, 208, 3, 87, 62, 557, 3, 1, 698, 67, 8, 43]
```

```
[43] generated_text
```

```
'There now is your insular city of the manhattoes belted round by the whale and your head civilized and the crossed sea it had a'
```

Seed Text:

There now is your insular

Generated Text- Keras + LSTM :

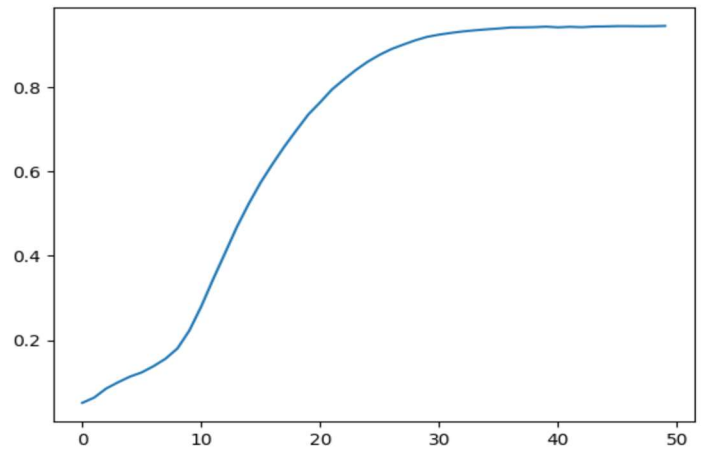
There now is your insular city of the manhattoes belted round by the whale and your head civilized and the crossed sea it had a

Problem 2: 3. Transfer Learning: Text generation model developed using word2vec word embeddings

```
max_sequence_len = max([len(seq) for seq in input_sequences])
input_sequence = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

X = input_sequence[:, :-1]
y = input_sequence[:,-1]

y = np.array(tf.keras.utils.to_categorical(y, num_classes=total_words))

# Create an embedding matrix using Word2Vec embeddings
embedding_dim = 300
embedding_matrix = np.zeros((total_words, embedding_dim))
for word, i in tokenizer.word_index.items():
    if word in word2vec_model:
        embedding_matrix[i] = word2vec_model[word]

# Build the model
model = Sequential()
model.add(Embedding(input_dim=total_words, output_dim=embedding_dim, weights=[embedding_matrix], input_length=max_sequence_len-1, trainable=False))
model.add(LSTM(150))
model.add(Dense(total_words, activation='softmax'))
model.summary()
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
```

```
[<matplotlib.lines.Line2D at 0x7deeecf9cac0>]
```



```
[73] print(generated_text)

     There now is your insular lives of something sort come in airth sort of a land probably sea was this most of guise whether so
```

Seed Text:

There now is your insular

Generated Text- Word2Vec + LSTM :

There now is your insular lives of something sort come in airth sort of a land probably sea was this most of guise whether so