```
Start coding or generate with AI.
```

Assignment --- 4 Text and Sequence Data

Loading the IMDB dataset

```
from keras.datasets import imdb

# Load the IMDB dataset
max_words  = 10000
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=max_words)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
from keras.preprocessing.sequence import pad_sequences

# Truncate or pad the reviews to a length of 150 words
maxlen = 150
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Select 5000 samples for testing
test_data = test_data[:5000]
test_labels = test_labels[:5000]

# Select 10,000 samples for validation
val_data = test_data[:10000]
val_labels = test_labels[:10000]
```

Model Building

```
from keras.models import Sequential
from keras.layers import Embedding, Bidirectional, LSTM, Dense, Dropout, BatchNormalization

# Build The RNN model
rnn_model = Sequential()

rnn_model.add(Embedding(10000, 32, input_length=len(train_data[0])))
rnn_model.add(Bidirectional(LSTM(64, return_sequences=True)))
rnn_model.add(Dropout(0.5))
rnn_model.add(BatchNormalization())
rnn_model.add(Bidirectional(LSTM(32)))
rnn_model.add(Dropout(0.5))
rnn_model.add(BatchNormalization())
rnn_model.add(Dense(1, activation='sigmoid'))
rnn_model.compile(loss="binary_crossentropy", optimizer="rmsprop", metrics=["accuracy"])

# Print model summary
print(" ")
print("RNN Model Architecture:")
print(rnn_model.summary())
print(" ")
```

```
RNN Model Architecture:
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
bidirectional (Bidirectional)	?	0 (unbuilt)
dropout (Dropout)	?	0 (unbuilt)
batch_normalization (BatchNormalization)	?	0 (unbuilt)
bidirectional_1 (Bidirectional)	?	0 (unbuilt)
dropout_1 (Dropout)	?	0 (unbuilt)
batch_normalization_1 (BatchNormalization)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
None
```

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip

import numpy as np

# Load GloVe word embeddings
embeddings_index = {}
with open('glove.6B.100d.txt') as f: # The file is assumed to be in the current directory
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

embedding_dim = 100
embedding_matrix = np.zeros((10000, embedding_dim))
```

```
    embedding_matrix = np.zeros((10000, embedding_dim))
    for i, word in enumerate(embeddings_index.keys()):
        if i < 10000:
            embedding_vector = embeddings_index.get(word)
            if embedding_vector is not None:
                embedding_matrix[i] = embedding_vector

    # Define the model with pretrained word embeddings
    rnn_model_pretrained = Sequential()
    rnn_model_pretrained.add(Embedding(10000, embedding_dim, input_length=maxlen, trainable=False))
    rnn_model_pretrained.add(Bidirectional(LSTM(64, return_sequences=True)))
    rnn_model_pretrained.add(Dropout(0.5))
    rnn_model_pretrained.add(BatchNormalization())
    rnn_model_pretrained.add(Bidirectional(LSTM(32)))
    rnn_model_pretrained.add(Dropout(0.5))
    rnn_model_pretrained.add(BatchNormalization())
    rnn_model_pretrained.add(Dense(1, activation='sigmoid'))
    rnn_model_pretrained.compile(loss="binary_crossentropy", optimizer="rmsprop", metrics=["accuracy"])

    # Print model summary
    print(" ")
    print("RNN Model Pre Trained Architecture:")
    print(rnn_model_pretrained.summary())
    print(" ")
```

```
--2024-11-22 02:33:52--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2024-11-22 02:33:52--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2024-11-22 02:33:53--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.05MB/s    in 2m 56s

2024-11-22 02:36:50 (4.67 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]

Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt

RNN Model Pre Trained Architecture:
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	?	0 (unbuilt)
bidirectional_2 (Bidirectional)	?	0 (unbuilt)
dropout_2 (Dropout)	?	0 (unbuilt)
batch_normalization_2 (BatchNormalization)	?	0 (unbuilt)
bidirectional_3 (Bidirectional)	?	0 (unbuilt)
dropout_3 (Dropout)	?	0 (unbuilt)
batch_normalization_3 (BatchNormalization)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)

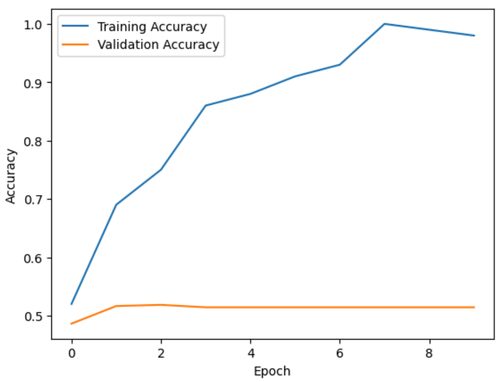
```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
None
```

For 100 Training samples

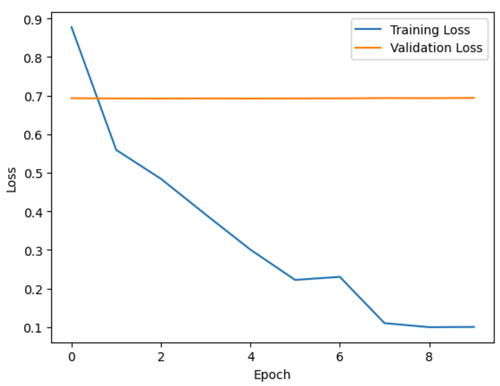
```
# Select the first 100 samples for training
train_data_100 = train_data[:100]
train_labels_100 = train_labels[:100]


# Train the RNN model
rnn_model_100 = rnn_model
rnn_history_100 = rnn_model_100.fit(train_data_100, train_labels_100, epochs=10, batch_size=32, validation_data=(val_data, val_labels))

# Evaluate the model
test_loss_rnn100, test_accuracy_rnn100 = rnn_model_100.evaluate(test_data, test_labels)

print("Test Loss : ", test_loss_rnn100)
print("Test Accuracy : ", test_accuracy_rnn100)


#Model Perfomance Evaluation
import matplotlib.pyplot as plt

print(" ")
print("Perfomance of RNN Model for 100 Training Samples : ")
print(" ")
# Plot training and validation accuracy
print("Accuracy : ")
print(" ")
plt.plot(rnn_history_100.history['accuracy'], label='Training Accuracy')
plt.plot(rnn_history_100.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
print(" ")
print("Loss : ")
print(" ")
plt.plot(rnn_history_100.history['loss'], label='Training Loss')
plt.plot(rnn_history_100.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 31s 7s/step - accuracy: 0.5361 - loss: 0.8859 - val_accuracy: 0.4862 - val_loss: 0.6932
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 22s 7s/step - accuracy: 0.6823 - loss: 0.5554 - val_accuracy: 0.5164 - val_loss: 0.6926
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 41s 7s/step - accuracy: 0.7740 - loss: 0.4619 - val_accuracy: 0.5184 - val_loss: 0.6924
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 4s/step - accuracy: 0.8669 - loss: 0.3621 - val_accuracy: 0.5142 - val_loss: 0.6927
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 14s 5s/step - accuracy: 0.8791 - loss: 0.2929 - val_accuracy: 0.5142 - val_loss: 0.6923
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 5s/step - accuracy: 0.9109 - loss: 0.2277 - val_accuracy: 0.5142 - val_loss: 0.6926
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 5s/step - accuracy: 0.9449 - loss: 0.2298 - val_accuracy: 0.5142 - val_loss: 0.6928
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 21s 7s/step - accuracy: 1.0000 - loss: 0.1029 - val_accuracy: 0.5142 - val_loss: 0.6936
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 31s 4s/step - accuracy: 0.9908 - loss: 0.0862 - val_accuracy: 0.5142 - val_loss: 0.6933
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - accuracy: 0.9847 - loss: 0.0924 - val_accuracy: 0.5142 - val_loss: 0.6940
157/157 ━━━━━━━━━━━━━━━━━━━━ 14s 86ms/step - accuracy: 0.5212 - loss: 0.6920
Test Loss :  0.6940479278564453
Test Accuracy :  0.5141999721527

Perfomance of RNN Model for 100 Training Samples :

Accuracy :
```



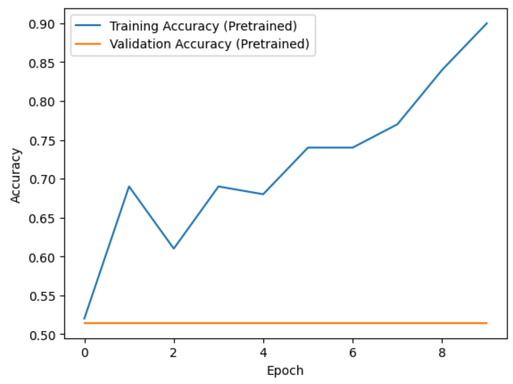

```
Loss :
```



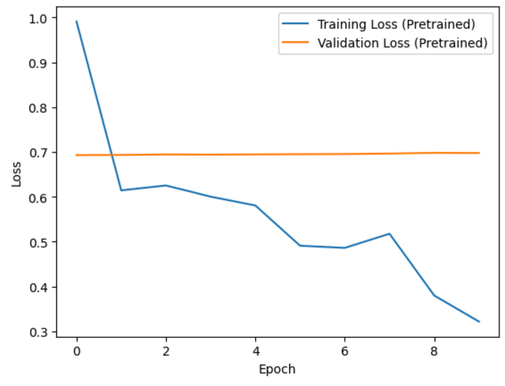

```
# Train the RNN model with pretrained embeddings
rnn_model_pretrained_100 = rnn_model_pretrained
rnn_history_pretrained_100 = rnn_model_pretrained_100.fit(train_data_100, train_labels_100, epochs=10, batch_size=32, validation_data=(val_data, val_labels))

# Evaluate the model on the test data
test_loss_pre_trained_rnn100, test_accuracy_pre_trained_rnn100 = rnn_model_pretrained_100.evaluate(test_data, test_labels)

print("Test Loss : ", test_loss_pre_trained_rnn100)
print("Test Accuracy : ", test_accuracy_pre_trained_rnn100)

# Plot training and validation accuracy
print("Perfomance of Pre Trained RNN Model for 100 Training Samples : ")
print(" ")
```

```
print("Accuracy : ")
print(" ")
plt.plot(rnn_history_pretrained_100.history['accuracy'], label='Training Accuracy (Pretrained)')
plt.plot(rnn_history_pretrained_100.history['val_accuracy'], label='Validation Accuracy (Pretrained)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

print(" ")
print("Loss : ")
print(" ")
# Plot training and validation loss
plt.plot(rnn_history_pretrained_100.history['loss'], label='Training Loss (Pretrained)')
plt.plot(rnn_history_pretrained_100.history['val_loss'], label='Validation Loss (Pretrained)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 25s 6s/step - accuracy: 0.5195 - loss: 1.0335 - val_accuracy: 0.5142 - val_loss: 0.6929
Epoch 2/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 24s 7s/step - accuracy: 0.7041 - loss: 0.5958 - val_accuracy: 0.5142 - val_loss: 0.6934
Epoch 3/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 41s 7s/step - accuracy: 0.5940 - loss: 0.6359 - val_accuracy: 0.5142 - val_loss: 0.6943
Epoch 4/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 42s 7s/step - accuracy: 0.6916 - loss: 0.5978 - val_accuracy: 0.5142 - val_loss: 0.6940
Epoch 5/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 40s 7s/step - accuracy: 0.7220 - loss: 0.5400 - val_accuracy: 0.5142 - val_loss: 0.6944
Epoch 6/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 41s 7s/step - accuracy: 0.7595 - loss: 0.4630 - val_accuracy: 0.5142 - val_loss: 0.6950
Epoch 7/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 41s 7s/step - accuracy: 0.7231 - loss: 0.4977 - val_accuracy: 0.5142 - val_loss: 0.6953
Epoch 8/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 41s 7s/step - accuracy: 0.7788 - loss: 0.5405 - val_accuracy: 0.5142 - val_loss: 0.6964
Epoch 9/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 33s 4s/step - accuracy: 0.8391 - loss: 0.3919 - val_accuracy: 0.5142 - val_loss: 0.6979
Epoch 10/10
4/4 ━━━━━━━━━━━━━━━━━━━━ 28s 7s/step - accuracy: 0.9100 - loss: 0.3168 - val_accuracy: 0.5142 - val_loss: 0.6976
157/157 ━━━━━━━━━━━━━━━━━━━━ 13s 81ms/step - accuracy: 0.5212 - loss: 0.6956
Test Loss :  0.6975911259651184
Test Accuracy :  0.51419997215271
Perfomance of Pre Trained RNN Model for 100 Training Samples :

Accuracy :
```



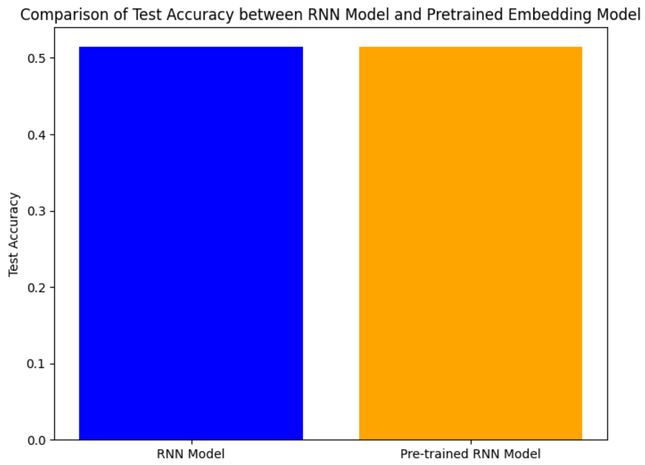

```
Loss :
```




```
# Model names for labeling
model_names = ['RNN Model', 'Pre-trained RNN Model']

# Evaluate the pre-trained RNN model # This was the missing part
test_loss_pre_trained_rnn100, test_accuracy_pre_trained_rnn100 = rnn_model_pretrained.evaluate(test_data, test_labels)

# Plot comparison graph
plt.figure(figsize=(8, 6))
plt.bar(model_names, [test_accuracy_rnn100, test_accuracy_pre_trained_rnn100], color=['blue', 'orange'])
plt.title('Comparison of Test Accuracy between RNN Model and Pretrained Embedding Model')
```

```
plt.title('Comparison of Test Accuracy between RNN Model and Pretrained Embedding Model')
plt.ylabel('Test Accuracy')
plt.show()
```

```
157/157 ━━━━━━━━━━━━━━━━━━━━ 14s 88ms/step - accuracy: 0.5212 - loss: 0.6956
```



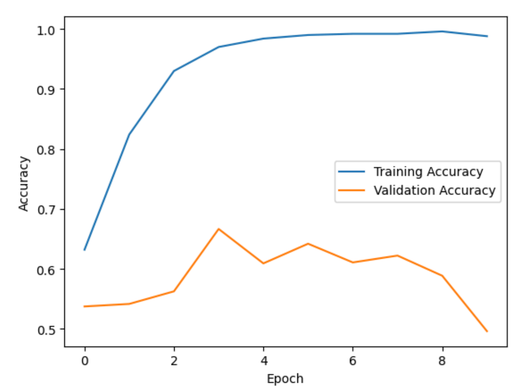

```
# Model names for labeling
model_names = ['RNN Model', 'Pre-trained RNN Model']

# Plot comparison graph
plt.figure(figsize=(8, 6))
plt.bar(model_names, [test_loss_rnn100, test_loss_pre_trained_rnn100], color=['blue', 'orange'])
plt.title('Comparison of Test Loss between RNN Model and Pretrained Embedding Model')
plt.ylabel('Test Loss')
plt.show()
```


For Training Samples 500

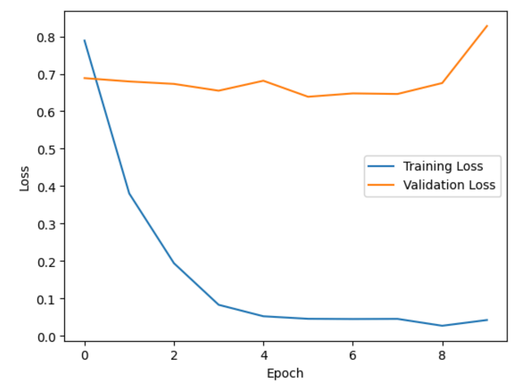
```
# Select the first 500 samples for training
train_data_500 = train_data[:500]
train_labels_500 = train_labels[:500]


# Train the RNN model
rnn_model_500 = rnn_model
rnn_history_500 = rnn_model_500.fit(train_data_500, train_labels_500, epochs=10, batch_size=32, validation_data=(val_data, val_labels))

# Evaluate the model
test_loss_rnn500, test_accuracy_rnn500 = rnn_model_500.evaluate(test_data, test_labels)

print("Test Loss : ", test_loss_rnn500)
print("Test Accuracy : ", test_accuracy_rnn500)


#Model Perfomance Evaluation
print(" ")
print("Perfomance of RNN Model for 500 Training Samples : ")
print(" ")
# Plot training and validation accuracy
print("Accuracy : ")
print(" ")
plt.plot(rnn_history_500.history['accuracy'], label='Training Accuracy')
plt.plot(rnn_history_500.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
```

```
plt.legend()
plt.show()

# Plot training and validation loss
print(" ")
print("Loss : ")
print(" ")
plt.plot(rnn_history_500.history['loss'], label='Training Loss')
plt.plot(rnn_history_500.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 21s 1s/step - accuracy: 0.6707 - loss: 0.7877 - val_accuracy: 0.5374 - val_loss: 0.6885
Epoch 2/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.7932 - loss: 0.4389 - val_accuracy: 0.5416 - val_loss: 0.6796
Epoch 3/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 25s 2s/step - accuracy: 0.9357 - loss: 0.1858 - val_accuracy: 0.5626 - val_loss: 0.6731
Epoch 4/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - accuracy: 0.9544 - loss: 0.1075 - val_accuracy: 0.6666 - val_loss: 0.6550
Epoch 5/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - accuracy: 0.9854 - loss: 0.0422 - val_accuracy: 0.6092 - val_loss: 0.6815
Epoch 6/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - accuracy: 0.9848 - loss: 0.0560 - val_accuracy: 0.6420 - val_loss: 0.6387
Epoch 7/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.9932 - loss: 0.0354 - val_accuracy: 0.6108 - val_loss: 0.6477
Epoch 8/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 21s 1s/step - accuracy: 0.9949 - loss: 0.0337 - val_accuracy: 0.6222 - val_loss: 0.6462
Epoch 9/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 20s 1s/step - accuracy: 0.9969 - loss: 0.0265 - val_accuracy: 0.5886 - val_loss: 0.6754
Epoch 10/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 20s 1s/step - accuracy: 0.9929 - loss: 0.0307 - val_accuracy: 0.4962 - val_loss: 0.8278
157/157 ━━━━━━━━━━━━━━━━━━━━ 13s 81ms/step - accuracy: 0.4903 - loss: 0.8355
Test Loss :  0.8277802467346191
Test Accuracy :  0.49619999527931213

Perfomance of RNN Model for 500 Training Samples :

Accuracy :
```



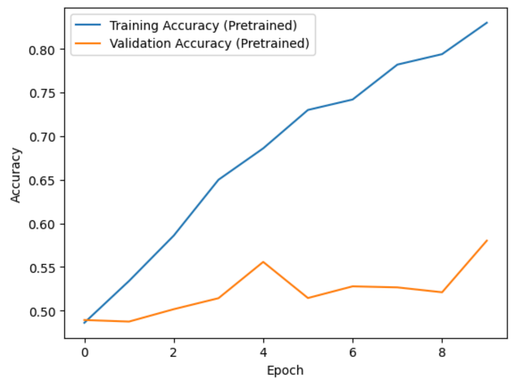

```
Loss :
```



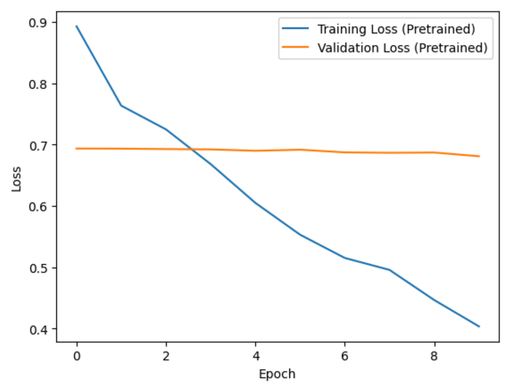

```
# Train the RNN model with pretrained embeddings
rnn_model_pretrained_500 = rnn_model_pretrained
rnn_history_pretrained_500 = rnn_model_pretrained_500.fit(train_data_500, train_labels_500, epochs=10, batch_size=32, validation_data=(val_data, val_labels))

# Evaluate the model on the test data
test_loss_pre_trained_rnn500, test_accuracy_pre_trained_rnn500 = rnn_model_pretrained_500.evaluate(test_data, test_labels)

print("Test Loss : ", test_loss_pre_trained_rnn500)
print("Test Accuracy : ", test_accuracy_pre_trained_rnn500)

# Plot training and validation accuracy
print("Perfomance of Pre Trained RNN Model for 500 Training Samples : ")
print(" ")
print("Accuracy : ")
```

```
print(" ")
plt.plot(rnn_history_pretrained_500.history['accuracy'], label='Training Accuracy (Pretrained)')
plt.plot(rnn_history_pretrained_500.history['val_accuracy'], label='Validation Accuracy (Pretrained)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

print(" ")
print("Loss : ")
print(" ")
# Plot training and validation loss
plt.plot(rnn_history_pretrained_500.history['loss'], label='Training Loss (Pretrained)')
plt.plot(rnn_history_pretrained_500.history['val_loss'], label='Validation Loss (Pretrained)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

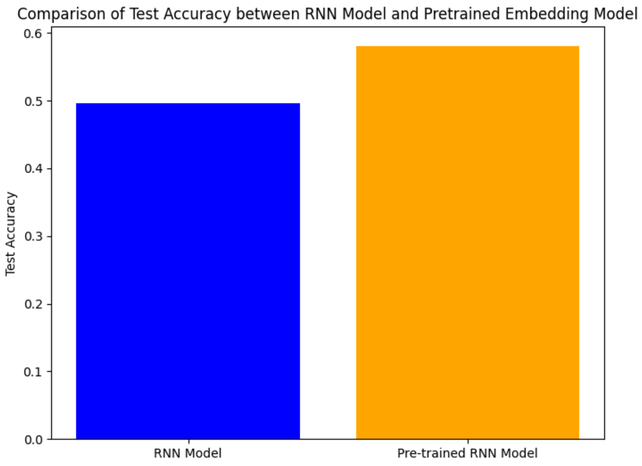
```
Epoch 1/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 28s 1s/step - accuracy: 0.4477 - loss: 0.9398 - val_accuracy: 0.4892 - val_loss: 0.6933
Epoch 2/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.5301 - loss: 0.7838 - val_accuracy: 0.4874 - val_loss: 0.6931
Epoch 3/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 45s 2s/step - accuracy: 0.5764 - loss: 0.7372 - val_accuracy: 0.5016 - val_loss: 0.6925
Epoch 4/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 45s 2s/step - accuracy: 0.6881 - loss: 0.6382 - val_accuracy: 0.5142 - val_loss: 0.6920
Epoch 5/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 31s 1s/step - accuracy: 0.6584 - loss: 0.6340 - val_accuracy: 0.5558 - val_loss: 0.6897
Epoch 6/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 31s 2s/step - accuracy: 0.7166 - loss: 0.5551 - val_accuracy: 0.5144 - val_loss: 0.6914
Epoch 7/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 31s 1s/step - accuracy: 0.7216 - loss: 0.5419 - val_accuracy: 0.5278 - val_loss: 0.6871
Epoch 8/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 21s 1s/step - accuracy: 0.7832 - loss: 0.4741 - val_accuracy: 0.5266 - val_loss: 0.6864
Epoch 9/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.7996 - loss: 0.4369 - val_accuracy: 0.5210 - val_loss: 0.6868
Epoch 10/10
16/16 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.8565 - loss: 0.3536 - val_accuracy: 0.5802 - val_loss: 0.6808
157/157 ━━━━━━━━━━━━━━━━━━━━ 15s 97ms/step - accuracy: 0.5943 - loss: 0.6797
Test Loss :  0.6807900071144104
Test Accuracy :  0.5802000164985657
Perfomance of Pre Trained RNN Model for 500 Training Samples :

Accuracy :
```



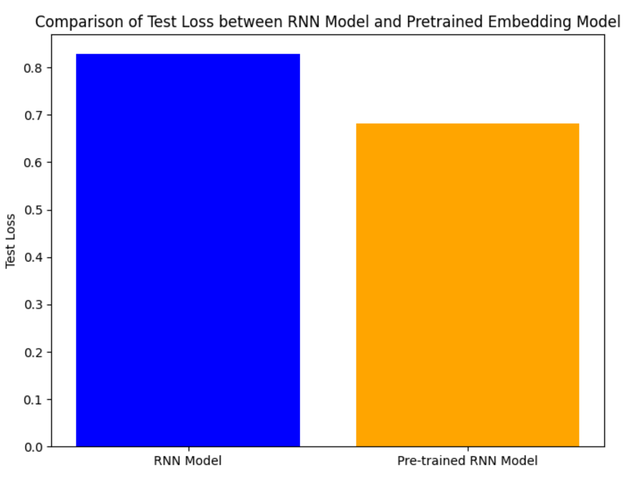

```
Loss :
```




```
# Model names for labeling
model_names = ['RNN Model', 'Pre-trained RNN Model']

# Plot comparison graph
plt.figure(figsize=(8, 6))
plt.bar(model_names, [test_accuracy_rnn500, test_accuracy_pre_trained_rnn500], color=['blue', 'orange'])
plt.title('Comparison of Test Accuracy between RNN Model and Pretrained Embedding Model')
plt.ylabel('Test Accuracy')
plt.show()
```




```
# Model names for labeling
model_names = ['RNN Model', 'Pre-trained RNN Model']

# Plot comparison graph
plt.figure(figsize=(8, 6))
plt.bar(model_names, [test_loss_rnn500, test_loss_pre_trained_rnn500], color=['blue', 'orange'])
plt.title('Comparison of Test Loss between RNN Model and Pretrained Embedding Model')
plt.ylabel('Test Loss')
plt.show()
```

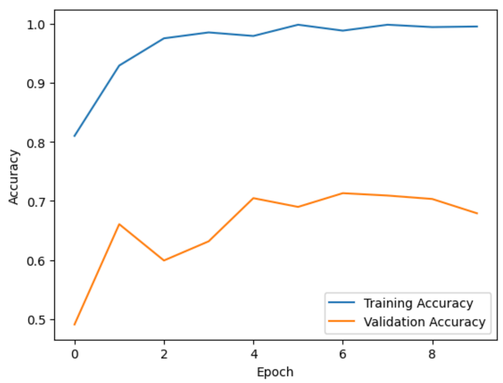


For running 1000 samples

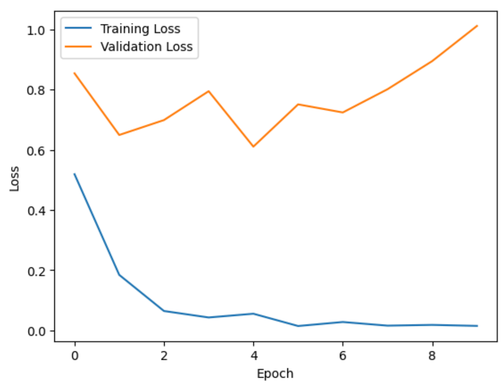
```
# Select the first 1000 samples for training
train_data_1000 = train_data[:1000]
train_labels_1000 = train_labels[:1000]


# Train the RNN model
rnn_model_1000 = rnn_model
rnn_history_1000 = rnn_model_1000.fit(train_data_1000, train_labels_1000, epochs=10, batch_size=32, validation_data=(val_data, val_labels))

# Evaluate the model
test_loss_rnn1000, test_accuracy_rnn1000 = rnn_model_1000.evaluate(test_data, test_labels)

print("Test Loss : ", test_loss_rnn1000)
print("Test Accuracy : ", test_accuracy_rnn1000)


#Model Perfomance Evaluation
print(" ")
print("Perfomance of RNN Model for 1000 Training Samples : ")
print(" ")
# Plot training and validation accuracy
print("Accuracy : ")
print(" ")
plt.plot(rnn_history_1000.history['accuracy'], label='Training Accuracy')
plt.plot(rnn_history_1000.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
```

```
print(" ")
print("Loss : ")
print(" ")
plt.plot(rnn_history_1000.history['loss'], label='Training Loss')
plt.plot(rnn_history_1000.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 25s 793ms/step - accuracy: 0.8017 - loss: 0.5904 - val_accuracy: 0.4912 - val_loss: 0.8538
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 25s 789ms/step - accuracy: 0.9309 - loss: 0.1713 - val_accuracy: 0.6608 - val_loss: 0.6490
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 25s 785ms/step - accuracy: 0.9810 - loss: 0.0571 - val_accuracy: 0.5994 - val_loss: 0.6984
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 28s 894ms/step - accuracy: 0.9848 - loss: 0.0436 - val_accuracy: 0.6318 - val_loss: 0.7943
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 38s 785ms/step - accuracy: 0.9856 - loss: 0.0412 - val_accuracy: 0.7048 - val_loss: 0.6103
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 29s 941ms/step - accuracy: 0.9956 - loss: 0.0169 - val_accuracy: 0.6900 - val_loss: 0.7506
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 28s 877ms/step - accuracy: 0.9837 - loss: 0.0365 - val_accuracy: 0.7132 - val_loss: 0.7237
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 38s 784ms/step - accuracy: 0.9957 - loss: 0.0280 - val_accuracy: 0.7092 - val_loss: 0.8009
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 41s 790ms/step - accuracy: 0.9969 - loss: 0.0112 - val_accuracy: 0.7034 - val_loss: 0.8944
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 47s 986ms/step - accuracy: 0.9902 - loss: 0.0246 - val_accuracy: 0.6792 - val_loss: 1.0111
157/157 ━━━━━━━━━━━━━━━━━━━━ 14s 87ms/step - accuracy: 0.6769 - loss: 0.9991
Test Loss :  1.0110595226287842
Test Accuracy :  0.6791999936103821

Perfomance of RNN Model for 1000 Training Samples :

Accuracy :
```



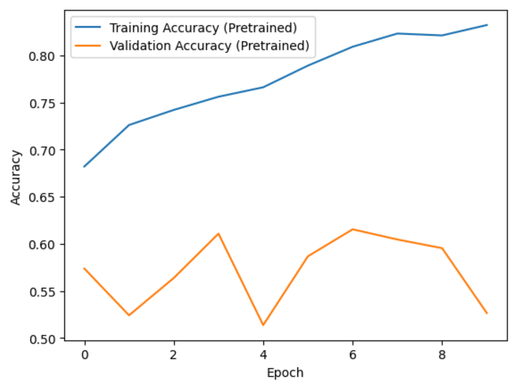

```
Loss :
```



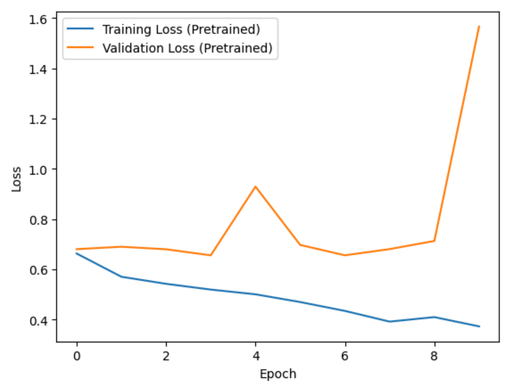

```
# Train the RNN model with pretrained embeddings
rnn_model_pretrained_1000 = rnn_model_pretrained
rnn_history_pretrained_1000 = rnn_model_pretrained_1000.fit(train_data_1000, train_labels_1000, epochs=10, batch_size=32, validation_data=(val_data, val_labels))

# Evaluate the model on the test data
test_loss_pre_trained_rnn1000, test_accuracy_pre_trained_rnn1000 = rnn_model_pretrained_1000.evaluate(test_data, test_labels)

print("Test Loss : ", test_loss_pre_trained_rnn1000)
print("Test Accuracy : ", test_accuracy_pre_trained_rnn1000)

# Plot training and validation accuracy
print("Perfomance of Pre Trained RNN Model for 1000 Training Samples : ")
print(" ")
print("Accuracy : ")
print(" ")
plt.plot(rnn_history_pretrained_1000.history['accuracy'], label='Training Accuracy (Pretrained)')
plt.plot(rnn_history_pretrained_1000.history['val_accuracy'], label='Validation Accuracy (Pretrained)')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')
plt.legend()
plt.show()

print(" ")
print("Loss : ")
print(" ")
# Plot training and validation loss
plt.plot(rnn_history_pretrained_1000.history['loss'], label='Training Loss (Pretrained)')
plt.plot(rnn_history_pretrained_1000.history['val_loss'], label='Validation Loss (Pretrained)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

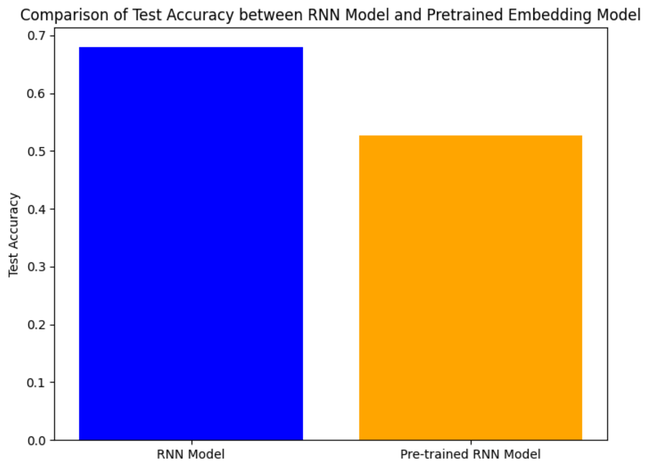
```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 33s 1s/step - accuracy: 0.6926 - loss: 0.6585 - val_accuracy: 0.5738 - val_loss: 0.6807
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - accuracy: 0.7155 - loss: 0.5819 - val_accuracy: 0.5242 - val_loss: 0.6904
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 31s 810ms/step - accuracy: 0.7483 - loss: 0.5281 - val_accuracy: 0.5638 - val_loss: 0.6802
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 29s 936ms/step - accuracy: 0.7810 - loss: 0.4952 - val_accuracy: 0.6108 - val_loss: 0.6562
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 39s 861ms/step - accuracy: 0.7591 - loss: 0.5003 - val_accuracy: 0.5138 - val_loss: 0.9297
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 46s 1s/step - accuracy: 0.8066 - loss: 0.4524 - val_accuracy: 0.5868 - val_loss: 0.6976
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 44s 1s/step - accuracy: 0.8210 - loss: 0.4253 - val_accuracy: 0.6154 - val_loss: 0.6563
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 32s 819ms/step - accuracy: 0.8236 - loss: 0.3856 - val_accuracy: 0.6046 - val_loss: 0.6809
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 26s 832ms/step - accuracy: 0.8251 - loss: 0.4042 - val_accuracy: 0.5954 - val_loss: 0.7134
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 48s 1s/step - accuracy: 0.8407 - loss: 0.3579 - val_accuracy: 0.5266 - val_loss: 1.5664
157/157 ━━━━━━━━━━━━━━━━━━━━ 18s 111ms/step - accuracy: 0.5333 - loss: 1.5379
Test Loss :  1.566427230834961
Test Accuracy :  0.5266000032424927
Perfomance of Pre Trained RNN Model for 1000 Training Samples :

Accuracy :
```



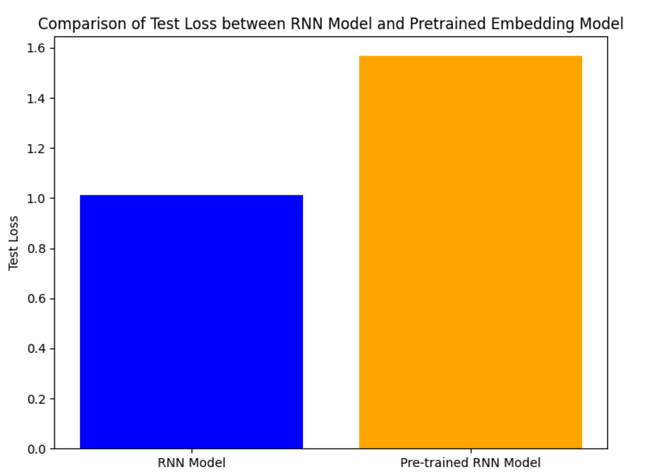

```
Loss :
```




```
# Model names for labeling
model_names = ['RNN Model', 'Pre-trained RNN Model']

# Plot comparison graph
plt.figure(figsize=(8, 6))
plt.bar(model_names, [test_accuracy_rnn1000, test_accuracy_pre_trained_rnn1000], color=['blue', 'orange'])
plt.title('Comparison of Test Accuracy between RNN Model and Pretrained Embedding Model')
plt.ylabel('Test Accuracy')
plt.show()
```




```
# Model names for labeling
model_names = ['RNN Model', 'Pre-trained RNN Model']

# Plot comparison graph
plt.figure(figsize=(8, 6))
plt.bar(model_names, [test_loss_rnn1000, test_loss_pre_trained_rnn1000], color=['blue', 'orange'])
plt.title('Comparison of Test Loss between RNN Model and Pretrained Embedding Model')
plt.ylabel('Test Loss')
plt.show()
```



Double-click (or enter) to edit