assignment-4-text-and-sequence

May 5, 2024

Assignment 4: Text and Sequence

Shiva Chaitanya Goud Gadila - 811252042

Praneeth simha - 811252718

The Embedding layer requires two essential parameters:

The vocabulary size, represented here as 1000, which includes the maximum word index plus one. The embedding dimensionality, specified as 64, determining the size of the vector space in which words will be embedded.

[ ]:
```
from keras.layers import Embedding
embedding_layer = Embedding(1000, 64)
```

[ ]:
```
from keras.models import Sequential
from keras.layers import Flatten, Dense
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import ModelCheckpoint
from keras.models import Sequential
from keras.layers import Flatten, Dense, Embedding, LSTM,  Conv1D,
↪MaxPooling1D, GlobalMaxPooling1D, Dropout
from keras.models import load_model
from keras.preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
from keras.optimizers import RMSprop
from google.colab import files
import re, os
from keras.datasets import imdb
from keras import preprocessing
from keras.utils import pad_sequences
```

Model 1 From Scratch

```
[ ]: #Specify the number of words considered as features and then Loading a the data␣
     ↪as lists of integers followed by preprocessing.sequence.pad_sequences.
     max_features = 10000
     maxlen = 150
     (x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
     x_train = pad_sequences(x_train, maxlen=maxlen)
     x_test = pad_sequences(x_test, maxlen=maxlen)
```

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

We create a sequential model. Then, we add an Embedding layer with input length, flatten the tensor, add a classifier, and compile the model. Finally, we train the model with training data for 10 epochs.

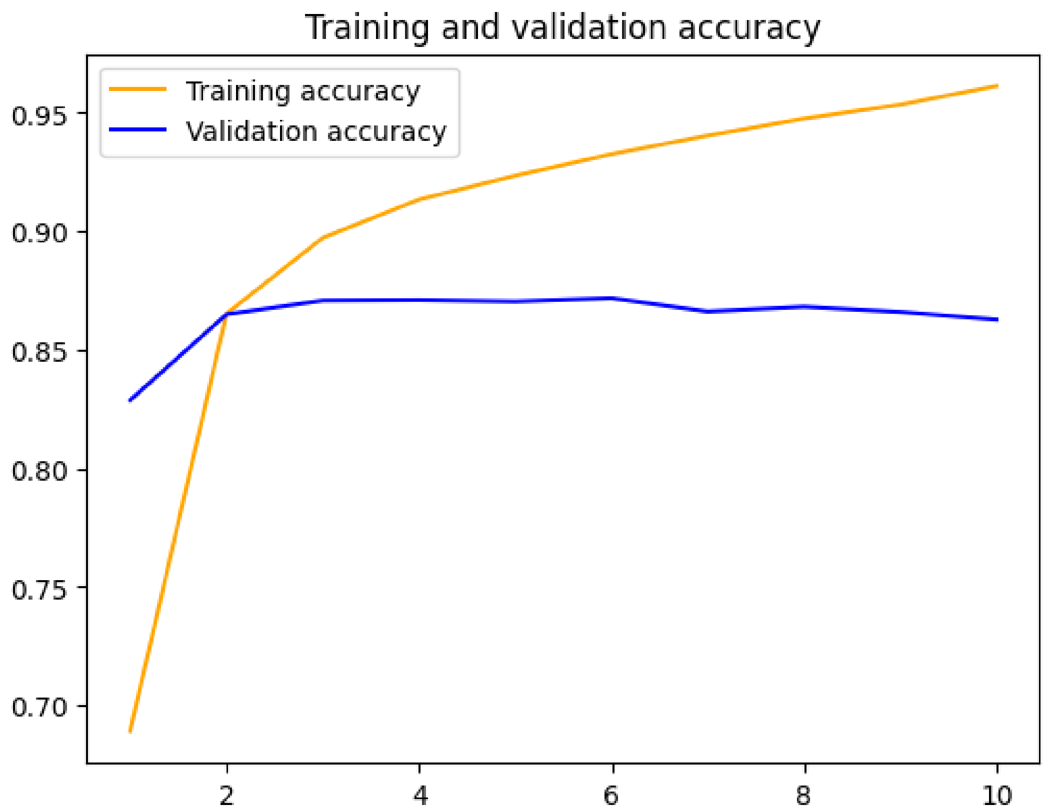
```
[ ]: model = Sequential()

     # Giving the maximum input length to our Embedding layer so that we can later␣
     ↪flatten the embedded inputs

     model.add(Embedding(10000, 8, input_length=maxlen))

     # We flatten the 3D tensor of embeddings into a 2D tensor of shape `(samples,␣
     ↪maxlen * 8)`
     model.add(Flatten())

     # We add the classifier on top and compiling the model
     model.add(Dense(1, activation='sigmoid'))
     model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
     model.summary()

     history_1 = model.fit(x_train, y_train,
                         epochs=10,
                         batch_size=32,
                         validation_split=0.2)
```

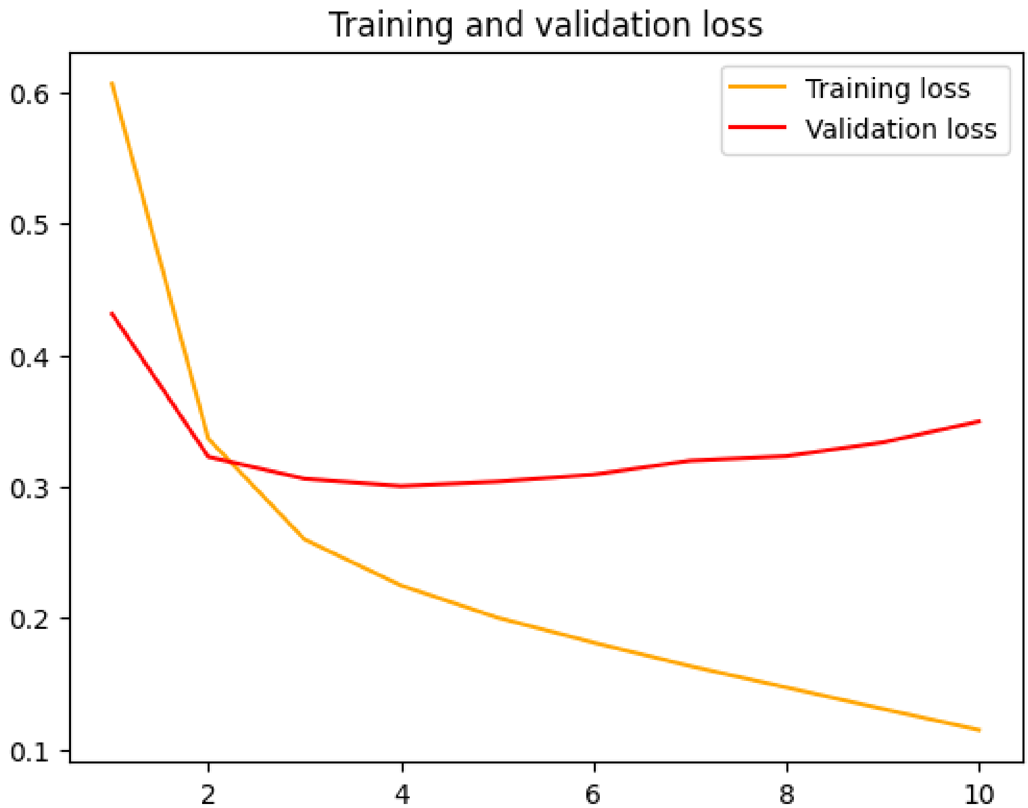
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 150, 8)            80000

 flatten (Flatten)           (None, 1200)              0

 dense (Dense)               (None, 1)                 1201

=================================================================
```

```
Total params: 81201 (317.19 KB)
Trainable params: 81201 (317.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 26s 39ms/step - loss: 0.6064 - acc:
0.6893 - val_loss: 0.4313 - val_acc: 0.8288
Epoch 2/10
625/625 [==============================] - 6s 9ms/step - loss: 0.3363 - acc:
0.8652 - val_loss: 0.3225 - val_acc: 0.8650
Epoch 3/10
625/625 [==============================] - 3s 5ms/step - loss: 0.2596 - acc:
0.8972 - val_loss: 0.3058 - val_acc: 0.8708
Epoch 4/10
625/625 [==============================] - 4s 7ms/step - loss: 0.2247 - acc:
0.9134 - val_loss: 0.3003 - val_acc: 0.8710
Epoch 5/10
625/625 [==============================] - 3s 5ms/step - loss: 0.2002 - acc:
0.9234 - val_loss: 0.3037 - val_acc: 0.8704
Epoch 6/10
625/625 [==============================] - 3s 5ms/step - loss: 0.1812 - acc:
0.9326 - val_loss: 0.3090 - val_acc: 0.8718
Epoch 7/10
625/625 [==============================] - 3s 4ms/step - loss: 0.1634 - acc:
0.9403 - val_loss: 0.3196 - val_acc: 0.8662
Epoch 8/10
625/625 [==============================] - 3s 4ms/step - loss: 0.1471 - acc:
0.9475 - val_loss: 0.3230 - val_acc: 0.8682
Epoch 9/10
625/625 [==============================] - 2s 4ms/step - loss: 0.1308 - acc:
0.9534 - val_loss: 0.3334 - val_acc: 0.8660
Epoch 10/10
625/625 [==============================] - 2s 4ms/step - loss: 0.1148 - acc:
0.9613 - val_loss: 0.3495 - val_acc: 0.8628
```

[ ]:
```
import matplotlib.pyplot as plt

accuracy = history_1.history['acc']
val_accuracy = history_1.history['val_acc']
loss = history_1.history['loss']
val_loss = history_1.history['val_loss']

epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, 'orange', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
```

```
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'orange', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





[ ]:
```
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.3474 - acc:
0.8658
Test loss: 0.3473982810974121
Test accuracy: 0.8657600283622742
```

Model 2 Training - 100 samples

The model consists of an Embedding layer with input dimensions (10000) and output dimensions (8), followed by a Flatten layer and a Dense layer with sigmoid activation. Compiled with RMSprop optimizer, binary crossentropy loss, and accuracy metrics, the model is trained on 100 samples for 10 epochs with a batch size of 32 and a validation split of 0.2.

[ ]:
```
max_features=10000
maxlen=150
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
```

```python
texts = np.concatenate((x_train, x_test), axis=0)
labels = np.concatenate((x_train, x_test), axis=0)

x_train = x_train[:100]
y_train = y_train[:100]
```

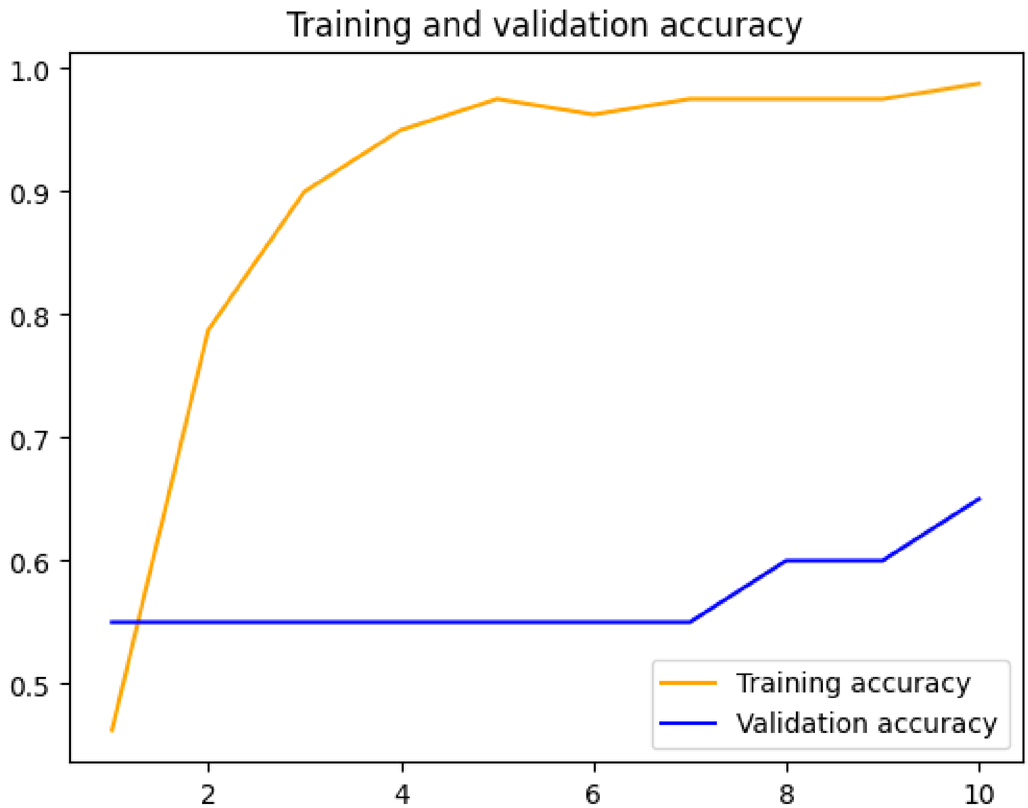
[ ]:
```python
model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()
history_2 = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 150, 8)            80000

 flatten_1 (Flatten)         (None, 1200)              0

 dense_1 (Dense)             (None, 1)                 1201

=================================================================
Total params: 81201 (317.19 KB)
Trainable params: 81201 (317.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
3/3 [==============================] - 1s 180ms/step - loss: 0.6971 - acc:
0.4625 - val_loss: 0.6898 - val_acc: 0.5500
Epoch 2/10
3/3 [==============================] - 0s 98ms/step - loss: 0.6731 - acc: 0.7875
- val_loss: 0.6896 - val_acc: 0.5500
Epoch 3/10
3/3 [==============================] - 0s 97ms/step - loss: 0.6557 - acc: 0.9000
- val_loss: 0.6896 - val_acc: 0.5500
Epoch 4/10
3/3 [==============================] - 0s 70ms/step - loss: 0.6399 - acc: 0.9500
- val_loss: 0.6900 - val_acc: 0.5500
Epoch 5/10
3/3 [==============================] - 0s 104ms/step - loss: 0.6246 - acc:
```

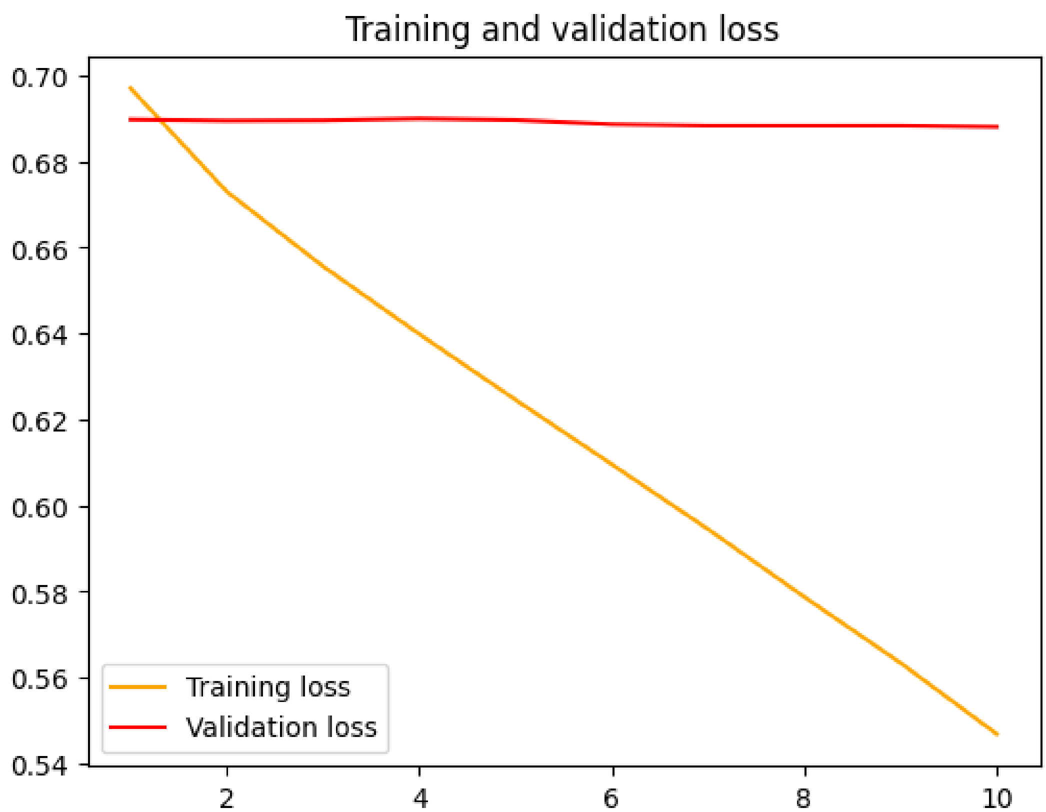
```
0.9750 - val_loss: 0.6897 - val_acc: 0.5500
Epoch 6/10
3/3 [==============================] - 0s 142ms/step - loss: 0.6096 - acc:
0.9625 - val_loss: 0.6887 - val_acc: 0.5500
Epoch 7/10
3/3 [==============================] - 0s 76ms/step - loss: 0.5945 - acc: 0.9750
- val_loss: 0.6884 - val_acc: 0.5500
Epoch 8/10
3/3 [==============================] - 0s 162ms/step - loss: 0.5787 - acc:
0.9750 - val_loss: 0.6884 - val_acc: 0.6000
Epoch 9/10
3/3 [==============================] - 0s 156ms/step - loss: 0.5634 - acc:
0.9750 - val_loss: 0.6884 - val_acc: 0.6000
Epoch 10/10
3/3 [==============================] - 0s 153ms/step - loss: 0.5468 - acc:
0.9875 - val_loss: 0.6881 - val_acc: 0.6500
```

[ ]:

```python
accuracy = history_2.history['acc']
val_accuracy = history_2.history['val_acc']
loss = history_2.history['loss']
val_loss = history_2.history['val_loss']

epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, 'orange', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'orange', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



Training and validation accuracy
1.0
0.9
0.8
0.7
0.6
0.5
2
4
6
8
10
Training accuracy
Validation accuracy



[ ]:
```
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.6951 - acc:
0.5012
Test loss: 0.6951099038124084
Test accuracy: 0.5012000203132629
```

1 Using Pre-Trained word embeddings

Download the IMDB data as raw text

- List item
- List item

Model 3 Pre-Trained model, Training- 100 samples

[ ]:
```
```

[ ]:
```
# Define the variable `content` with the appropriate value
content = "/content/IMDB-Movie-Data.csv"
```

```
[ ]: import os
```

```
[ ]: !curl -O https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
     !tar -xf aclImdb_v1.tar.gz

     !rm -r aclImdb/train/unsup
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 80.2M  100 80.2M    0     0  19.1M      0  0:00:04  0:00:04 --:--:-- 19.1M
```

```
[ ]: imdb_dir = imdb_dir = '/content/aclImdb'
```

```
[ ]: train_dir = os.path.join(imdb_dir, 'train')
```

```
[ ]: labels = []
     texts = []
```

```
[ ]: for label_type in ['neg', 'pos']:
         dir_name = os.path.join(train_dir, label_type)
         for fname in os.listdir(dir_name):
             if fname[-4:] == '.txt':
                 f = open(os.path.join(dir_name, fname))
                 texts.append(f.read())
                 f.close()
                 if label_type == 'neg':
                     labels.append(0)
                 else:
                     labels.append(1)
```

Tokenizing the data

Before splitting the data into training and validation sets, shuffling is essential to ensure randomness, particularly when the samples are ordered. This step helps prevent any bias that might arise from the original ordering, thus ensuring a more representative distribution in both the training and validation sets.

```
[ ]: maxlen = 150  # We will cut reviews after 100 words
     training_samples = 100 # We will be training on 100 samples
     validation_samples = 10000  # We will be validating on 10000 samples
     max_words = 10000  # We will only consider the top 10,000 words in the dataset

     tokenizer = Tokenizer(num_words=max_words)
     tokenizer.fit_on_texts(texts)
     sequences = tokenizer.texts_to_sequences(texts)

     word_index = tokenizer.word_index
     print('Found %s unique tokens.' % len(word_index))
```

```
data = pad_sequences(sequences, maxlen=maxlen)

labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 150)
Shape of label tensor: (25000,)
```

Download the GloVe word embeddings

Pre-Processing the embeddings

[ ]:
```
import numpy as np
import os

# Define the directory containing the GloVe embeddings
glove_file = "/content/glove.6B.100d.txt"
```

This code snippet loads pre-trained GloVe word embeddings from a file (glove.6B.100d.txt). It creates a dictionary where each word is mapped to its corresponding embedding vector. After parsing the file, it prints the total number of word vectors found in the GloVe file.

[ ]:
```
# Define the directory containing the GloVe embeddings
glove_file = "/content/glove.6B.100d.txt"

# Load the pre-trained word embeddings
embeddings_index = {}
with open(glove_file, encoding="utf-8") as f:
    for line in f:
        values = line.split()
        word = values[0]
        try:
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
        except ValueError:
            print(f"Issue with word: {word}. Skipping...")
```

```
            continue

print('Found %s word vectors.' % len(embeddings_index))
```

```
Found 48870 word vectors.
```

[ ]:
```
embedding_dim = 100

embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if i < max_words:
        if embedding_vector is not None:
            # Words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector
```

Building the model

[ ]:
```
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 150, 100)          1000000

 lstm (LSTM)                 (None, 32)                17024

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 1017057 (3.88 MB)
Trainable params: 1017057 (3.88 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Loading the GloVe embeddings in the model

- List item
- List item

```
[ ]: model.layers[0].set_weights([embedding_matrix])
     model.layers[0].trainable = False
```

```
[ ]: print("Training data shape:", y_train.shape)
```

```
Training data shape: (100,)
```

Train and evaluate

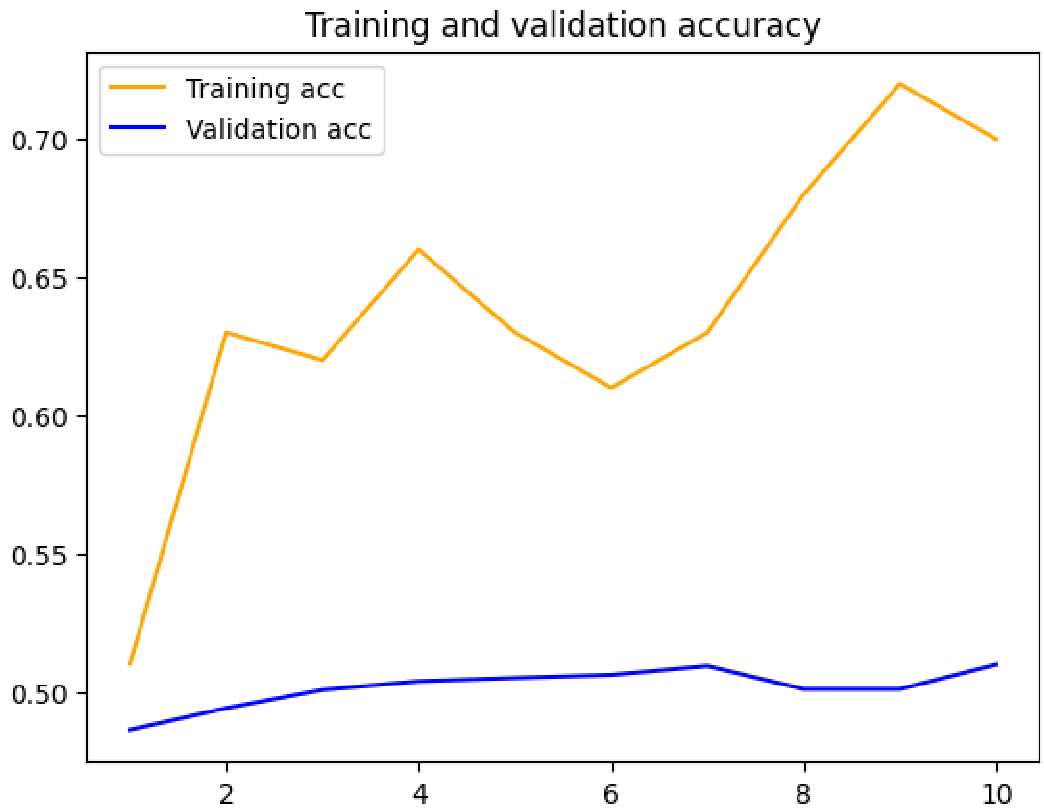
The code compiles and trains the model for 10 epochs using RMSprop optimizer, binary crossentropy loss, and accuracy metric, while validating the performance on validation data and saving the model weights.

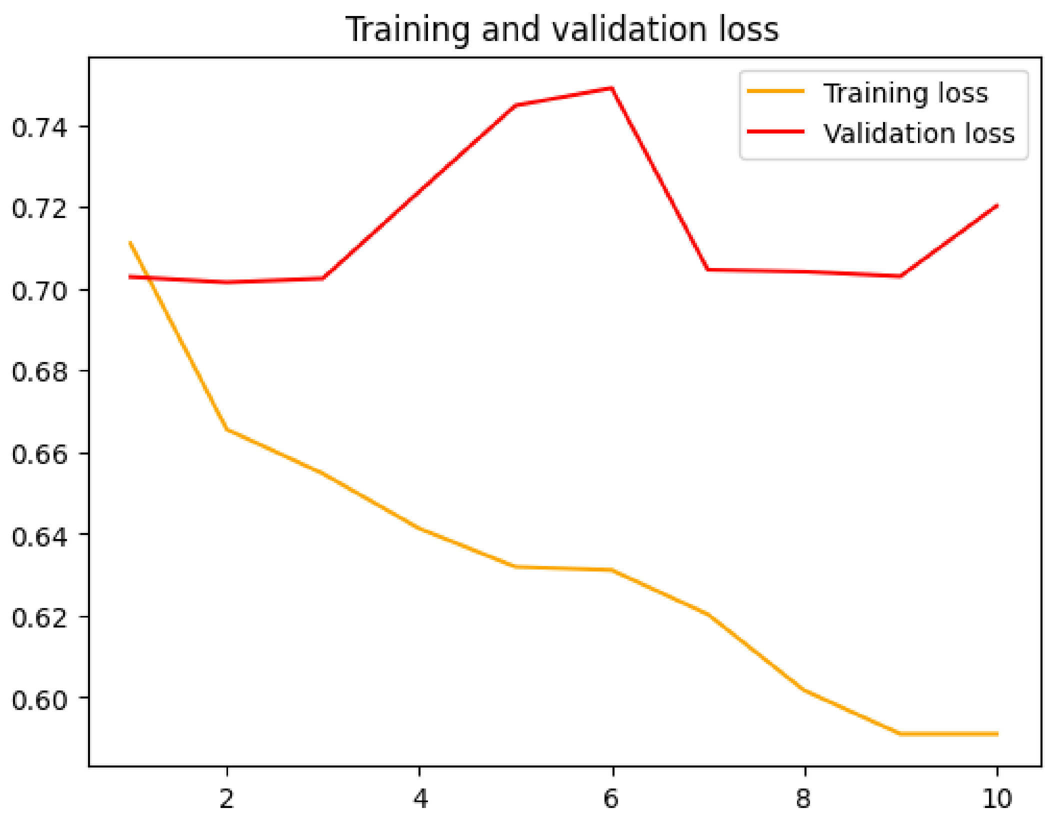
```
[ ]: model.compile(optimizer='rmsprop',
                   loss='binary_crossentropy',
                   metrics=['acc'])
     history_3 = model.fit(x_train, y_train,
                           epochs=10,
                           batch_size=32,
                           validation_data=(x_val, y_val))
     model.save_weights('pre_trained_glove_model.3a')
```

```
Epoch 1/10
4/4 [==============================] - 6s 1s/step - loss: 0.7111 - acc: 0.5100 -
val_loss: 0.7029 - val_acc: 0.4862
Epoch 2/10
4/4 [==============================] - 1s 379ms/step - loss: 0.6656 - acc:
0.6300 - val_loss: 0.7015 - val_acc: 0.4940
Epoch 3/10
4/4 [==============================] - 1s 439ms/step - loss: 0.6547 - acc:
0.6200 - val_loss: 0.7025 - val_acc: 0.5006
Epoch 4/10
4/4 [==============================] - 1s 372ms/step - loss: 0.6413 - acc:
0.6600 - val_loss: 0.7237 - val_acc: 0.5037
Epoch 5/10
4/4 [==============================] - 1s 435ms/step - loss: 0.6319 - acc:
0.6300 - val_loss: 0.7448 - val_acc: 0.5049
Epoch 6/10
4/4 [==============================] - 1s 399ms/step - loss: 0.6311 - acc:
0.6100 - val_loss: 0.7490 - val_acc: 0.5060
Epoch 7/10
4/4 [==============================] - 1s 379ms/step - loss: 0.6203 - acc:
0.6300 - val_loss: 0.7046 - val_acc: 0.5092
Epoch 8/10
4/4 [==============================] - 1s 437ms/step - loss: 0.6017 - acc:
0.6800 - val_loss: 0.7041 - val_acc: 0.5009
Epoch 9/10
4/4 [==============================] - 3s 875ms/step - loss: 0.5911 - acc:
0.7200 - val_loss: 0.7030 - val_acc: 0.5009
```

```
Epoch 10/10
4/4 [==============================] - 3s 862ms/step - loss: 0.5911 - acc:
0.7000 - val_loss: 0.7202 - val_acc: 0.5097
```

[ ]:
```
import matplotlib.pyplot as plt

acc = history_3.history['acc']
val_acc = history_3.history['val_acc']
loss = history_3.history['loss']
val_loss = history_3.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'orange', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'orange', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.70
0.65
0.60
0.55
0.50
2
4
6
8
10



[ ]:
```python
test_dir = os.path.join(imdb_dir, 'test')

labels = []
texts = []

for label_type in ['neg', 'pos']:
    dir_name = os.path.join(test_dir, label_type)
    for fname in sorted(os.listdir(dir_name)):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname))
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)

sequences = tokenizer.texts_to_sequences(texts)
x_test = pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(labels)
```

[ ]:
```
model.load_weights('pre_trained_glove_model.3a')
model.evaluate(x_test, y_test)
```

```
782/782 [==============================] - 3s 4ms/step - loss: 0.7191 - acc:
0.5115
```

[ ]:
```
[0.7191476821899414, 0.5114799737930298]
```

2 Now we change the number of training samples to determine at what point the embedding layer gives better performance

Model 4 training sample size - 1000 using embedding layer

You are loading the IMDb dataset with a vocabulary size of 10,000 and a maximum sequence length of 150. Then, you pad the sequences to ensure uniform length. Finally, you concatenate the training and testing data and select the first 1000 samples for training along with their corresponding labels.

[ ]:
```
max_features=10000
maxlen=150
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

texts = np.concatenate((x_train, x_test), axis=0)
labels = np.concatenate((x_train, x_test), axis=0)

x_train = x_train[:1000]
y_train = y_train[:1000]
```

[ ]:
```
model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()
history_4 = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

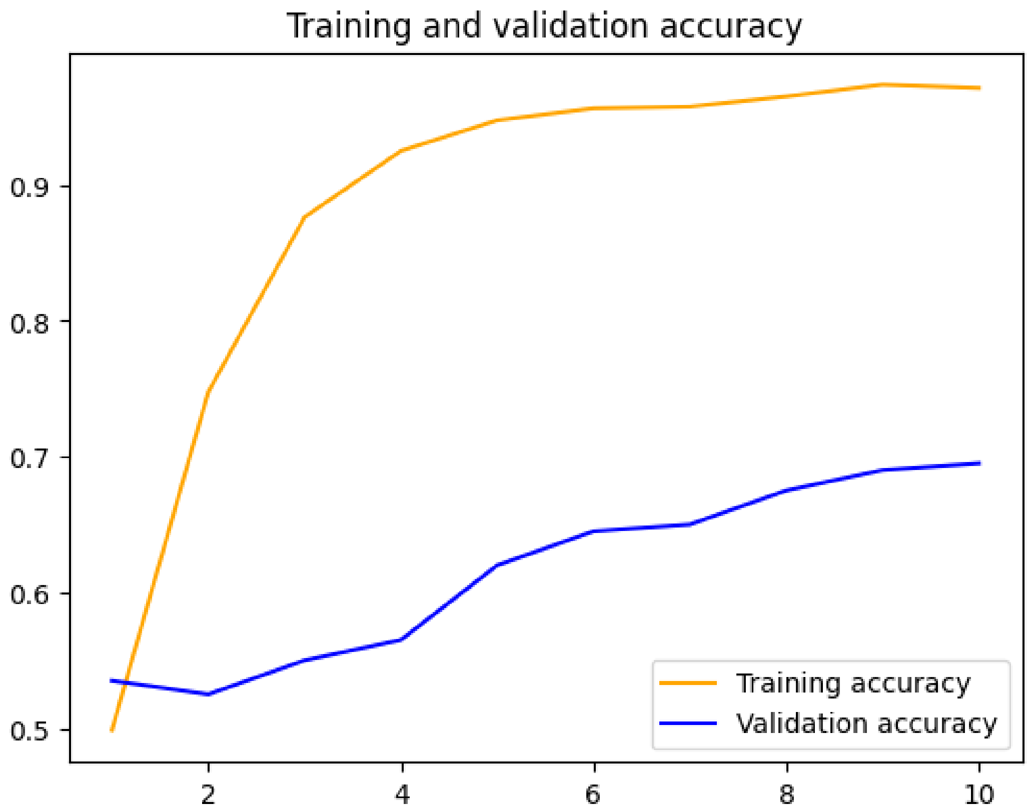
```
Model: "sequential_3"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 150, 8)            80000
```

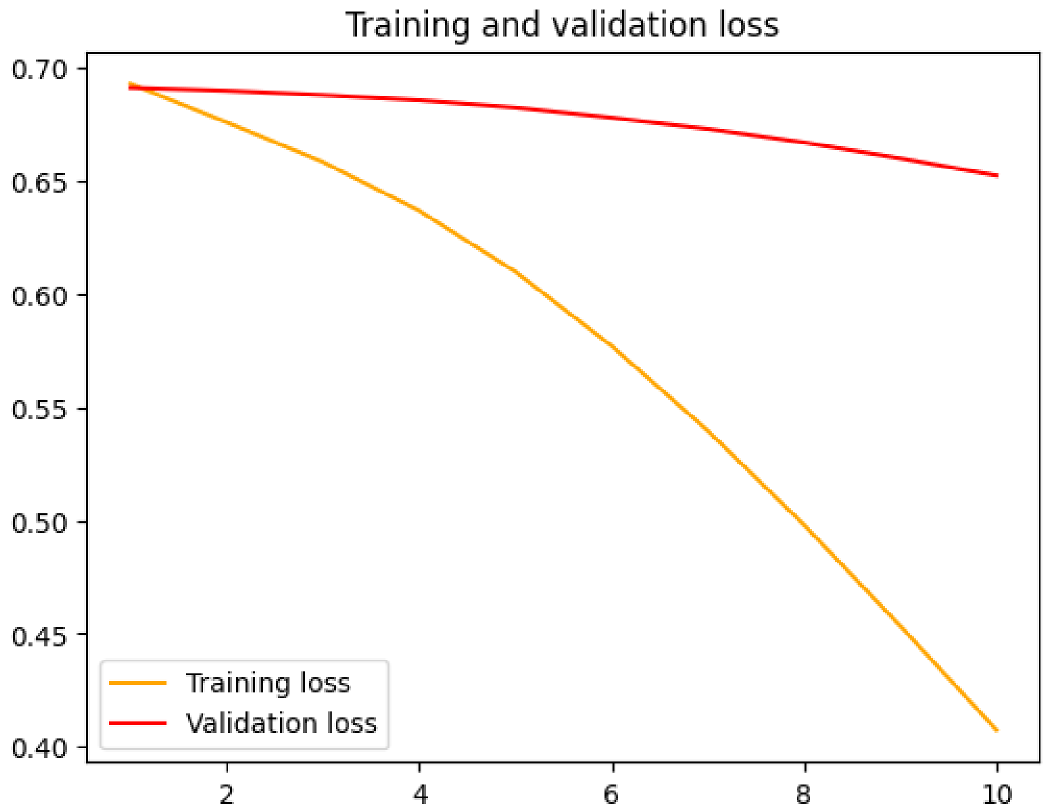
```
 flatten_2 (Flatten)         (None, 1200)              0

 dense_3 (Dense)             (None, 1)                 1201

=================================================================
Total params: 81201 (317.19 KB)
Trainable params: 81201 (317.19 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
25/25 [==============================] - 2s 68ms/step - loss: 0.6930 - acc:
0.4988 - val_loss: 0.6912 - val_acc: 0.5350
Epoch 2/10
25/25 [==============================] - 1s 58ms/step - loss: 0.6760 - acc:
0.7475 - val_loss: 0.6899 - val_acc: 0.5250
Epoch 3/10
25/25 [==============================] - 1s 48ms/step - loss: 0.6584 - acc:
0.8763 - val_loss: 0.6880 - val_acc: 0.5500
Epoch 4/10
25/25 [==============================] - 1s 49ms/step - loss: 0.6370 - acc:
0.9250 - val_loss: 0.6857 - val_acc: 0.5650
Epoch 5/10
25/25 [==============================] - 1s 56ms/step - loss: 0.6100 - acc:
0.9475 - val_loss: 0.6824 - val_acc: 0.6200
Epoch 6/10
25/25 [==============================] - 1s 56ms/step - loss: 0.5772 - acc:
0.9563 - val_loss: 0.6780 - val_acc: 0.6450
Epoch 7/10
25/25 [==============================] - 1s 46ms/step - loss: 0.5397 - acc:
0.9575 - val_loss: 0.6730 - val_acc: 0.6500
Epoch 8/10
25/25 [==============================] - 1s 37ms/step - loss: 0.4981 - acc:
0.9650 - val_loss: 0.6670 - val_acc: 0.6750
Epoch 9/10
25/25 [==============================] - 1s 26ms/step - loss: 0.4535 - acc:
0.9737 - val_loss: 0.6600 - val_acc: 0.6900
Epoch 10/10
25/25 [==============================] - 1s 26ms/step - loss: 0.4075 - acc:
0.9712 - val_loss: 0.6524 - val_acc: 0.6950
```

[ ]:

```
accuracy = history_4.history['acc']
val_accuracy = history_4.history['val_acc']
loss = history_4.history['loss']
val_loss = history_4.history['val_loss']

epochs = range(1, len(accuracy) + 1)
```

```
plt.plot(epochs, accuracy, 'orange', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'orange', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





[ ]:
```
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.6674 - acc:
0.5978
Test loss: 0.6674152612686157
Test accuracy: 0.5978000164031982
```

Model 5 Taining sample - 15000 using both embedding layer and Conv1D

In Model 5, you're training on 15,000 samples using both an Embedding layer and Conv1D layer. This combination allows for learning patterns from text data while considering the sequential nature of the input, enhancing the model's ability to capture complex features within the data.

[ ]:
```
max_features=10000
maxlen=150
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
```

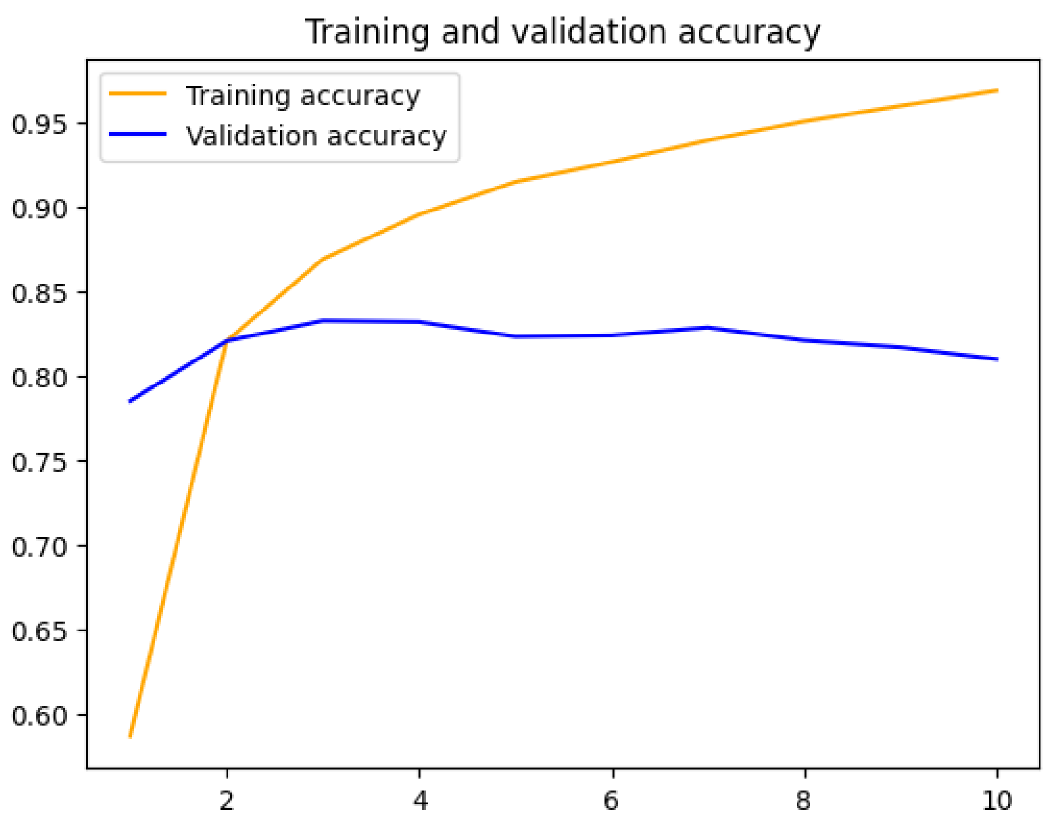
```
texts = np.concatenate((x_train, x_test), axis=0)
labels = np.concatenate((x_train, x_test), axis=0)

x_train = x_train[:15000]
y_train = y_train[:15000]
```

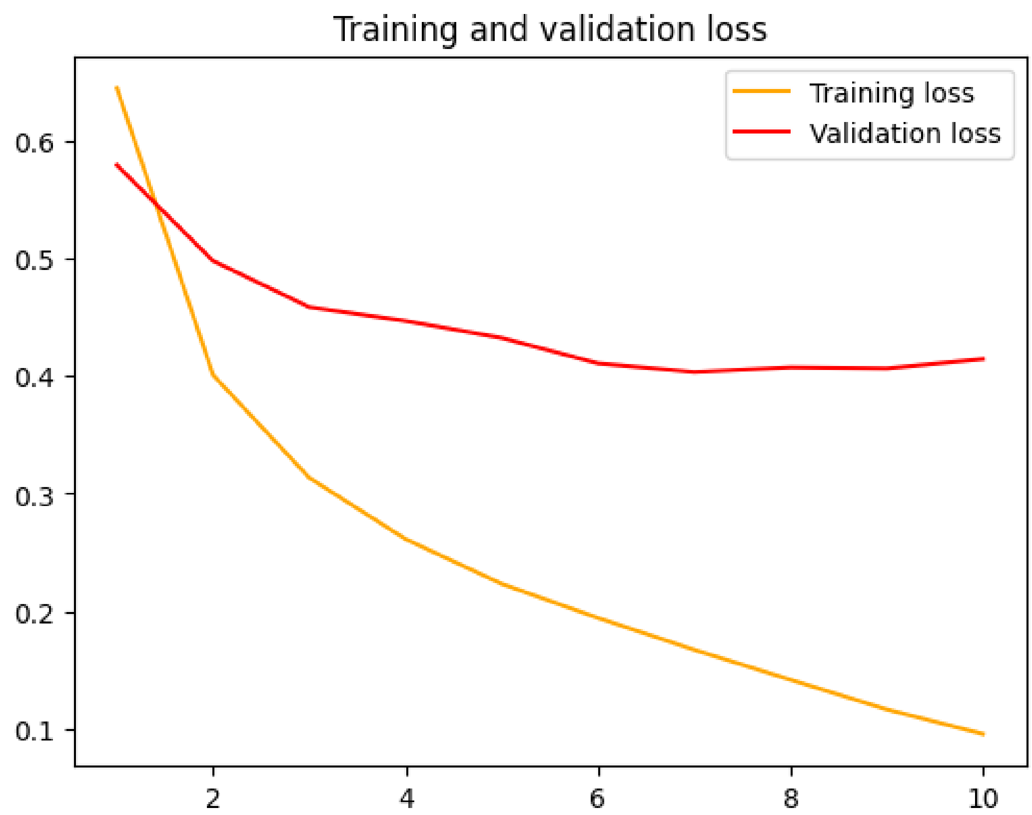
[ ]:
```
model = Sequential()
model.add(Embedding(10000, 10, input_length=maxlen))
model.add(Conv1D(512, 3, activation='relu'))
model.add(MaxPooling1D(3))

model.add(Conv1D(256, 3, activation='relu'))
model.add(MaxPooling1D(3))

model.add(Conv1D(256, 3, activation='relu'))
model.add(Dropout(0.8))
model.add(MaxPooling1D(3))

model.add(GlobalMaxPooling1D())
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()
history_5 = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
Model: "sequential_4"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, 150, 10)           100000

 conv1d (Conv1D)             (None, 148, 512)          15872

 max_pooling1d (MaxPooling1  (None, 49, 512)           0
 D)

 conv1d_1 (Conv1D)           (None, 47, 256)           393472

 max_pooling1d_1 (MaxPoolin  (None, 15, 256)           0
 g1D)

 conv1d_2 (Conv1D)           (None, 13, 256)           196864

 dropout (Dropout)           (None, 13, 256)           0
```

```
 max_pooling1d_2 (MaxPoolin  (None, 4, 256)            0
 g1D)

 global_max_pooling1d (Glob  (None, 256)               0
 alMaxPooling1D)

 flatten_3 (Flatten)         (None, 256)               0

 dense_4 (Dense)             (None, 1)                 257

=================================================================
Total params: 706465 (2.69 MB)
Trainable params: 706465 (2.69 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
375/375 [==============================] - 28s 61ms/step - loss: 0.6445 - acc:
0.5872 - val_loss: 0.5791 - val_acc: 0.7850
Epoch 2/10
375/375 [==============================] - 8s 21ms/step - loss: 0.4006 - acc:
0.8202 - val_loss: 0.4976 - val_acc: 0.8203
Epoch 3/10
375/375 [==============================] - 5s 14ms/step - loss: 0.3133 - acc:
0.8687 - val_loss: 0.4584 - val_acc: 0.8323
Epoch 4/10
375/375 [==============================] - 3s 9ms/step - loss: 0.2612 - acc:
0.8950 - val_loss: 0.4466 - val_acc: 0.8317
Epoch 5/10
375/375 [==============================] - 5s 12ms/step - loss: 0.2231 - acc:
0.9143 - val_loss: 0.4322 - val_acc: 0.8230
Epoch 6/10
375/375 [==============================] - 3s 8ms/step - loss: 0.1941 - acc:
0.9260 - val_loss: 0.4106 - val_acc: 0.8237
Epoch 7/10
375/375 [==============================] - 3s 8ms/step - loss: 0.1671 - acc:
0.9388 - val_loss: 0.4032 - val_acc: 0.8283
Epoch 8/10
375/375 [==============================] - 3s 7ms/step - loss: 0.1416 - acc:
0.9500 - val_loss: 0.4072 - val_acc: 0.8207
Epoch 9/10
375/375 [==============================] - 3s 8ms/step - loss: 0.1164 - acc:
0.9591 - val_loss: 0.4063 - val_acc: 0.8167
Epoch 10/10
375/375 [==============================] - 4s 11ms/step - loss: 0.0956 - acc:
0.9682 - val_loss: 0.4142 - val_acc: 0.8097
```

[ ]:
```python
accuracy = history_5.history['acc']
val_accuracy = history_5.history['val_acc']
loss = history_5.history['loss']
val_loss = history_5.history['val_loss']

epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, 'orange', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'orange', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





[ ]:
```
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
782/782 [==============================] - 3s 4ms/step - loss: 0.4355 - acc:
0.7970
Test loss: 0.43550741672515 87
Test accuracy: 0.7970399856567383
```

As we have seen in the previous model even though we increased the training sample size the accuracy was still low but when we used Con1D along with increased training sample size the accuracy improved to 81%

Model 6 Training sample 30000 using both embedding layers and Conv1D

[ ]:
```
max_features=10000
maxlen=150
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
```

```
texts = np.concatenate((x_train, x_test), axis=0)
labels = np.concatenate((x_train, x_test), axis=0)

x_train = x_train[:30000]
y_train = y_train[:30000]
```

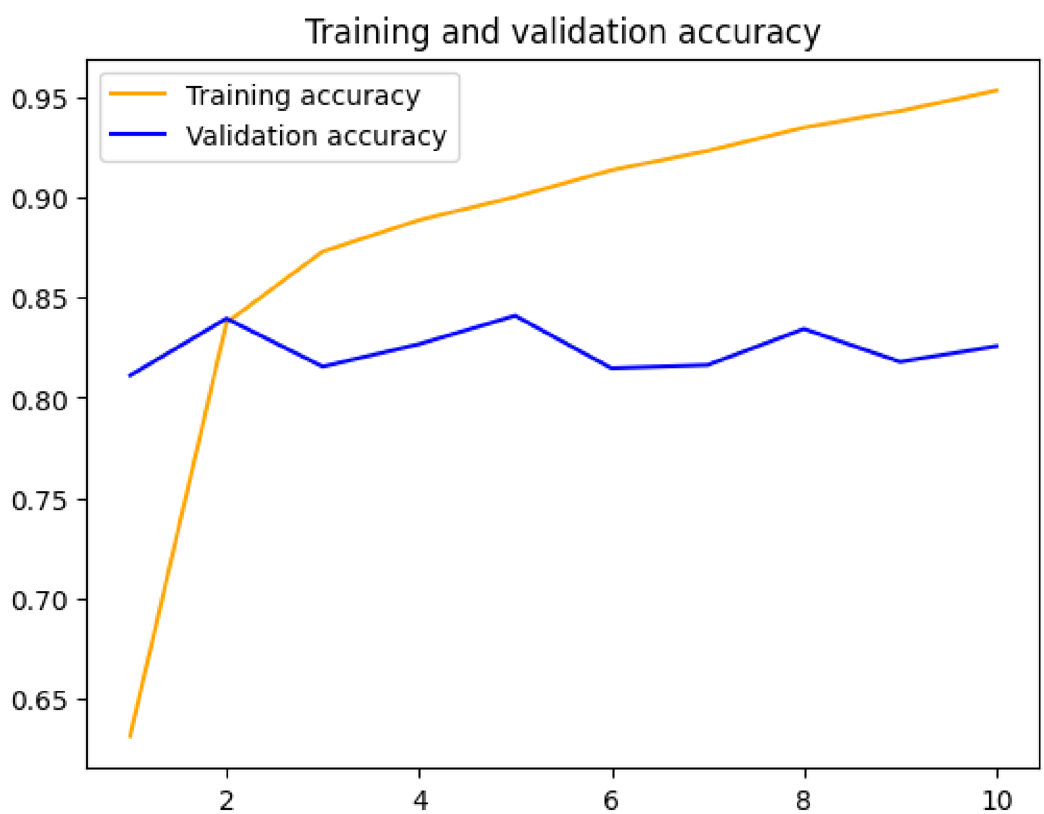
[ ]:
```
model = Sequential()
model.add(Embedding(10000, 12, input_length=maxlen))
model.add(Conv1D(512, 3, activation='relu'))
model.add(MaxPooling1D(3))

model.add(Conv1D(256, 3, activation='relu'))
model.add(MaxPooling1D(3))

model.add(Conv1D(256, 3, activation='relu'))
model.add(Dropout(0.8))
model.add(MaxPooling1D(3))

model.add(GlobalMaxPooling1D())
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()
history_6 = model.fit(x_train, y_train,
                      epochs=10,
                      batch_size=32,
                      validation_split=0.2)
```

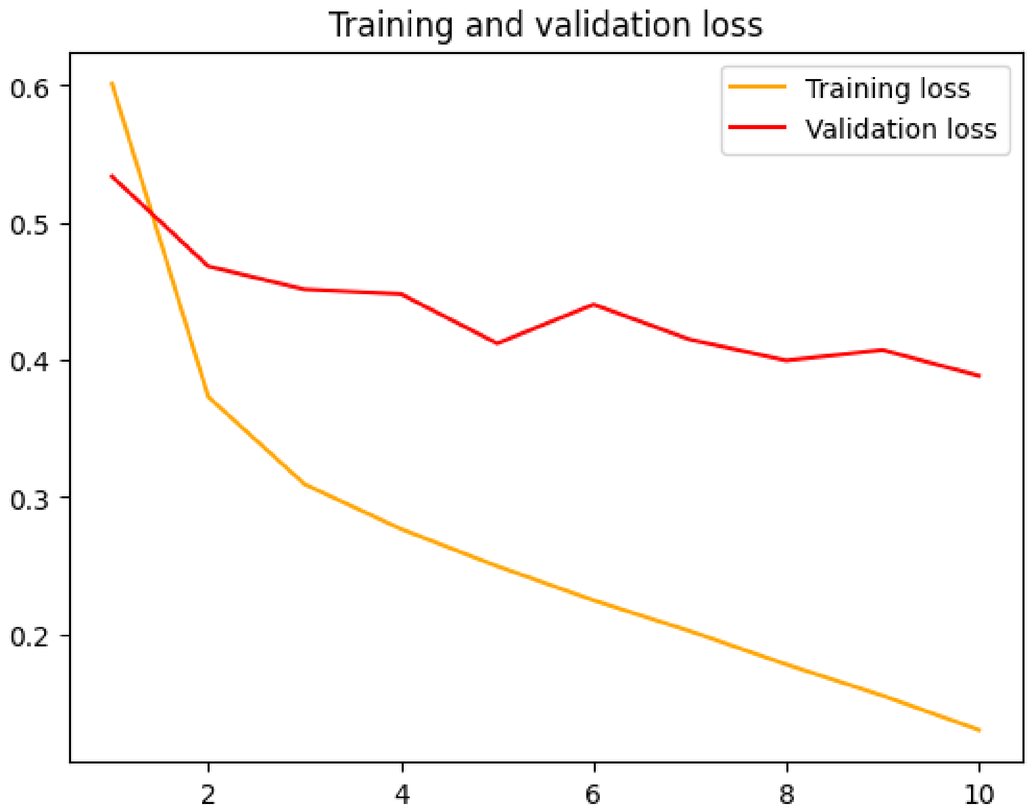
```
Model: "sequential_5"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_7 (Embedding)     (None, 150, 12)           120000

 conv1d_3 (Conv1D)           (None, 148, 512)          18944

 max_pooling1d_3 (MaxPoolin  (None, 49, 512)           0
 g1D)

 conv1d_4 (Conv1D)           (None, 47, 256)           393472

 max_pooling1d_4 (MaxPoolin  (None, 15, 256)           0
 g1D)

 conv1d_5 (Conv1D)           (None, 13, 256)           196864

 dropout_1 (Dropout)         (None, 13, 256)           0
```

```
 max_pooling1d_5 (MaxPoolin  (None, 4, 256)            0
 g1D)

 global_max_pooling1d_1 (Gl  (None, 256)               0
 obalMaxPooling1D)

 flatten_4 (Flatten)         (None, 256)               0

 dense_5 (Dense)             (None, 1)                 257

=================================================================
Total params: 729537 (2.78 MB)
Trainable params: 729537 (2.78 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
625/625 [==============================] - 29s 44ms/step - loss: 0.6013 - acc:
0.6312 - val_loss: 0.5337 - val_acc: 0.8112
Epoch 2/10
625/625 [==============================] - 7s 12ms/step - loss: 0.3730 - acc:
0.8375 - val_loss: 0.4681 - val_acc: 0.8396
Epoch 3/10
625/625 [==============================] - 6s 10ms/step - loss: 0.3093 - acc:
0.8731 - val_loss: 0.4513 - val_acc: 0.8156
Epoch 4/10
625/625 [==============================] - 6s 9ms/step - loss: 0.2767 - acc:
0.8887 - val_loss: 0.4480 - val_acc: 0.8268
Epoch 5/10
625/625 [==============================] - 6s 10ms/step - loss: 0.2496 - acc:
0.9003 - val_loss: 0.4119 - val_acc: 0.8410
Epoch 6/10
625/625 [==============================] - 4s 7ms/step - loss: 0.2247 - acc:
0.9136 - val_loss: 0.4404 - val_acc: 0.8148
Epoch 7/10
625/625 [==============================] - 4s 7ms/step - loss: 0.2022 - acc:
0.9233 - val_loss: 0.4148 - val_acc: 0.8164
Epoch 8/10
625/625 [==============================] - 6s 9ms/step - loss: 0.1780 - acc:
0.9349 - val_loss: 0.3995 - val_acc: 0.8344
Epoch 9/10
625/625 [==============================] - 5s 7ms/step - loss: 0.1552 - acc:
0.9433 - val_loss: 0.4072 - val_acc: 0.8180
Epoch 10/10
625/625 [==============================] - 4s 7ms/step - loss: 0.1303 - acc:
0.9534 - val_loss: 0.3884 - val_acc: 0.8258
```

[ ]:

```
accuracy = history_6.history['acc']
val_accuracy = history_6.history['val_acc']
loss = history_6.history['loss']
val_loss = history_6.history['val_loss']

epochs = range(1, len(accuracy) + 1)

plt.plot(epochs, accuracy, 'orange', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'orange', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```






```
[ ]: test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
782/782 [==============================] - 3s 3ms/step - loss: 0.3921 - acc:
0.8216
Test loss: 0.3921342194080353
Test accuracy: 0.8216400146484375
```

Model 7 pretrained model. Training - 15000 samples

This code snippet prepares the IMDb dataset for training by loading text data and corresponding labels, tokenizing the text, and padding sequences to ensure uniform length. It shuffles the data and splits it into training and validation sets. Finally, it prints the shapes of the data and label tensors.

```
[ ]: import os
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np

# Define the directory containing the IMDb dataset
```

```python
imdb_dir = '/content/aclImdb'

texts = []
labels = []

# Load the IMDb dataset
for label_type in ['neg', 'pos']:
    dir_name = os.path.join(imdb_dir, 'train', label_type)
    for fname in os.listdir(dir_name):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname))
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)

# Define parameters for tokenization and padding
maxlen = 150
training_samples = 15000
validation_samples = 10000
max_words = 10000

# Tokenize the text data
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

# Pad sequences to ensure uniform length
data = pad_sequences(sequences, maxlen=maxlen)

labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

# Shuffle the data
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

# Split the data into training and validation sets
x_train = data[:training_samples]
```

```
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 150)
Shape of label tensor: (25000,)
```

[ ]:
```
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_6"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_8 (Embedding)     (None, 150, 100)          1000000

 lstm_1 (LSTM)               (None, 32)                17024

 dense_6 (Dense)             (None, 1)                 33

=================================================================
Total params: 1017057 (3.88 MB)
Trainable params: 1017057 (3.88 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

[ ]:
```
print("Training data shape:", y_train.shape)
```

```
Training data shape: (15000,)
```

[ ]:
```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history_7 = model.fit(x_train, y_train,
                      epochs=10,
                      batch_size=32,
                      validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.7a')
```

```
Epoch 1/10
469/469 [==============================] - 7s 11ms/step - loss: 0.5919 - acc:
```

```
0.6819 - val_loss: 0.5391 - val_acc: 0.7362
Epoch 2/10
469/469 [==============================] - 7s 14ms/step - loss: 0.4839 - acc:
0.7756 - val_loss: 0.4379 - val_acc: 0.8051
Epoch 3/10
469/469 [==============================] - 5s 10ms/step - loss: 0.4292 - acc:
0.8075 - val_loss: 0.4097 - val_acc: 0.8133
Epoch 4/10
469/469 [==============================] - 5s 10ms/step - loss: 0.3943 - acc:
0.8247 - val_loss: 0.4972 - val_acc: 0.7753
Epoch 5/10
469/469 [==============================] - 6s 12ms/step - loss: 0.3684 - acc:
0.8378 - val_loss: 0.3805 - val_acc: 0.8269
Epoch 6/10
469/469 [==============================] - 5s 10ms/step - loss: 0.3463 - acc:
0.8494 - val_loss: 0.3602 - val_acc: 0.8392
Epoch 7/10
469/469 [==============================] - 6s 13ms/step - loss: 0.3313 - acc:
0.8584 - val_loss: 0.3601 - val_acc: 0.8420
Epoch 8/10
469/469 [==============================] - 5s 11ms/step - loss: 0.3169 - acc:
0.8661 - val_loss: 0.3428 - val_acc: 0.8486
Epoch 9/10
469/469 [==============================] - 5s 10ms/step - loss: 0.3024 - acc:
0.8708 - val_loss: 0.3445 - val_acc: 0.8511
Epoch 10/10
469/469 [==============================] - 6s 12ms/step - loss: 0.2901 - acc:
0.8793 - val_loss: 0.3490 - val_acc: 0.8517
```

[ ]:
```
model.load_weights('pre_trained_glove_model.7a')
model.evaluate(x_test, y_test)
```

```
782/782 [==============================] - 3s 4ms/step - loss: 1.0136 - acc:
0.5071
```

[ ]:
```
[1.013551115989685, 0.5070800185203552]
```

Model pre trained 3

[ ]:
```
maxlen = 150  # We will cut reviews after 100 words
training_samples = 30000 # We will be training on 30000 samples
validation_samples = 10000  # We will be validating on 10000 samples
max_words = 10000  # We will only consider the top 10,000 words in the dataset

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
```

```
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

data = pad_sequences(sequences, maxlen=maxlen)

labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

# Split the data into a training set and a validation set
# But first, shuffle the data, since we started from data
# where sample are ordered (all negative first, then all positive).
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:30000]
y_train = labels[:30000]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]
```

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 150)
Shape of label tensor: (25000,)
```

[ ]:
```
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(LSTM(128))
model.add(Dropout(0.3))

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))

model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

[ ]:
```
model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

[ ]:
```
print("Training data shape:", y_train.shape)
```

```
Training data shape: (25000,)
```

[ ]:
```
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
```

```python
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense
import numpy as np

maxlen = 150  # Cut texts after 150 words
training_samples = 15000  # Train on 15000 samples
validation_samples = 10000  # Validate on 10000 samples
max_words = 10000  # Consider only the top 10,000 words in the dataset

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

data = pad_sequences(sequences, maxlen=maxlen)

labels = np.asarray(labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

# Shuffle data
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_train = data[:training_samples]
y_train = labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = labels[training_samples: training_samples + validation_samples]

# Define the model
model = Sequential()
model.add(Embedding(max_words, 64))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
model.summary()

# Train the model
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
```

```python
                    validation_data=(x_val, y_val))

# Save the model weights
model.save_weights('pre_trained_glove_model.8a')
```

```
Found 88582 unique tokens.
Shape of data tensor: (25000, 150)
Shape of label tensor: (25000,)
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_11 (Embedding)    (None, None, 64)          640000

 lstm_4 (LSTM)               (None, 32)                12416

 dense_10 (Dense)            (None, 1)                 33

=================================================================
Total params: 652449 (2.49 MB)
Trainable params: 652449 (2.49 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
469/469 [==============================] - 21s 40ms/step - loss: 0.6934 - acc:
0.4981 - val_loss: 0.6933 - val_acc: 0.5011
Epoch 2/10
469/469 [==============================] - 10s 22ms/step - loss: 0.6889 - acc:
0.5391 - val_loss: 0.6965 - val_acc: 0.4977
Epoch 3/10
469/469 [==============================] - 6s 14ms/step - loss: 0.6724 - acc:
0.5903 - val_loss: 0.7072 - val_acc: 0.5036
Epoch 4/10
469/469 [==============================] - 8s 17ms/step - loss: 0.6356 - acc:
0.6384 - val_loss: 0.7395 - val_acc: 0.5005
Epoch 5/10
469/469 [==============================] - 5s 11ms/step - loss: 0.5809 - acc:
0.6937 - val_loss: 0.7860 - val_acc: 0.4928
Epoch 6/10
469/469 [==============================] - 8s 18ms/step - loss: 0.5198 - acc:
0.7457 - val_loss: 0.8721 - val_acc: 0.4934
Epoch 7/10
469/469 [==============================] - 5s 11ms/step - loss: 0.4524 - acc:
0.7923 - val_loss: 0.9321 - val_acc: 0.4964
Epoch 8/10
469/469 [==============================] - 5s 11ms/step - loss: 0.3899 - acc:
0.8295 - val_loss: 1.0357 - val_acc: 0.4947
```

```
Epoch 9/10
469/469 [==============================] - 6s 13ms/step - loss: 0.3268 - acc:
0.8646 - val_loss: 1.1690 - val_acc: 0.5002
Epoch 10/10
469/469 [==============================] - 5s 10ms/step - loss: 0.2670 - acc:
0.8916 - val_loss: 1.2158 - val_acc: 0.4978
```

[ ]:
```
model.load_weights('pre_trained_glove_model.8a')
model.evaluate(x_test, y_test)
```

```
782/782 [==============================] - 3s 4ms/step - loss: 1.2765 - acc:
0.4843
```

[ ]:
```
[1.276474118232727, 0.4843200147151947]
```