AML-Assignment-4

Text and Sequence

Sushma

 # Import necessary libraries and modules for data manipulation, visualization, and building a neural network using TensorFlow and Keras

```
import os
from operator import itemgetter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
get_ipython().magic(u'matplotlib inline')
plt.style.use('ggplot')

import tensorflow as tf

from keras import models, regularizers, layers, optimizers, losses, metrics
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

The IMDB dataset classifies movie reviews into positive and negative sentiments.

During the dataset preprocessing, each review is transformed into a series of word embeddings, where each word is represented by a fixed-size vector.

```
from keras.layers import Embedding

# The Embedding layer requires a minimum of two inputs:
# The maximum word index plus one, or 1000, is the number of potential tokens.
# and the embeddings' dimensions, in this case 64.
emb_lay = Embedding(1000, 64)
from keras.datasets import imdb
from keras import preprocessing
from keras.utils import pad_sequences
```

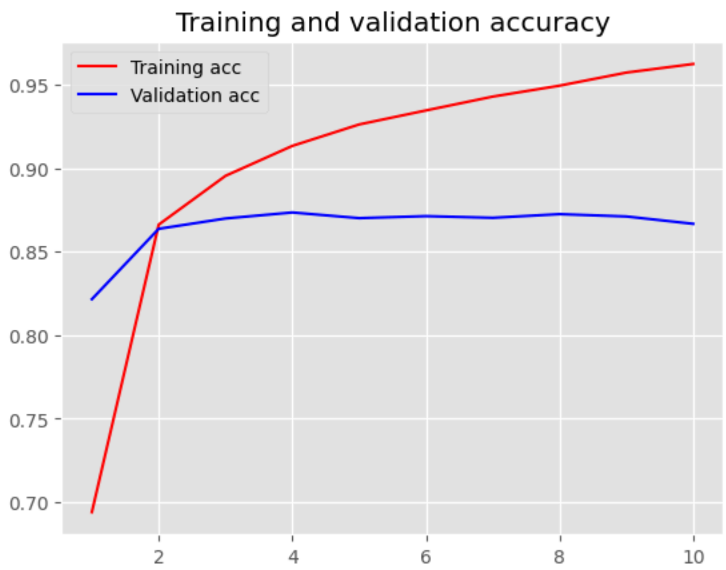
Custom-trained embedding layer using a training sample size of 100

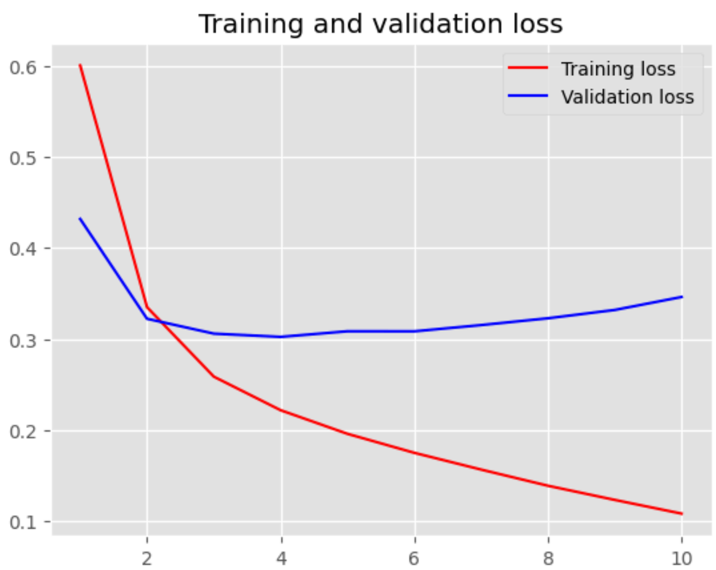
```
# The number of words that should be considered as features
features = 10000
# Remove the text after this number of words(from the top max_features most common words)
length = 150

# Data loading to integers
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=features)

x_tr = x_train[:100]
y_tr = y_train[:100]

# The integer lists are now transformed into a 2D integer tensor with the shape of {(samples, maxlen)}.
x_train = pad_sequences(x_train, maxlen=length)
x_test = pad_sequences(x_test, maxlen=length)
from keras.models import Sequential
from keras.layers import Flatten, Dense

model1 = Sequential()
# In order to finally flatten the embedded inputs, the maximum length of the input to the Embedding layer is provided.
model1.add(Embedding(10000, 8, input_length=length))
# After the Embedding layer, our activations have shape `(samples, maxlen, 8)`.

# We flatten the 3D tensor of embeddings into a 2D tensor of shape
#`(samples, maxlen * 8)`
model1.add(Flatten())

# We add the classifier on top
model1.add(Dense(1, activation='sigmoid'))
model1.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model1.summary()

hist1 = model1.fit(x_train, y_train,
                   epochs=10,
                   batch_size=32,
                   validation_split=0.2)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (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 [==============================] - 33s 49ms/step - loss: 0.6004 - acc: 0.6940 - val_loss: 0.4318 - val_acc: 0.8216
Epoch 2/10
625/625 [==============================] - 7s 11ms/step - loss: 0.3350 - acc: 0.8663 - val_loss: 0.3223 - val_acc: 0.8638
Epoch 3/10
625/625 [==============================] - 5s 7ms/step - loss: 0.2587 - acc: 0.8956 - val_loss: 0.3059 - val_acc: 0.8700
Epoch 4/10
625/625 [==============================] - 3s 5ms/step - loss: 0.2218 - acc: 0.9135 - val_loss: 0.3024 - val_acc: 0.8736
Epoch 5/10
625/625 [==============================] - 3s 5ms/step - loss: 0.1959 - acc: 0.9263 - val_loss: 0.3085 - val_acc: 0.8702
Epoch 6/10
625/625 [==============================] - 3s 4ms/step - loss: 0.1750 - acc: 0.9348 - val_loss: 0.3085 - val_acc: 0.8714
Epoch 7/10
625/625 [==============================] - 3s 6ms/step - loss: 0.1566 - acc: 0.9431 - val_loss: 0.3153 - val_acc: 0.8704
Epoch 8/10
625/625 [==============================] - 3s 4ms/step - loss: 0.1389 - acc: 0.9496 - val_loss: 0.3228 - val_acc: 0.8726
Epoch 9/10
625/625 [==============================] - 2s 4ms/step - loss: 0.1233 - acc: 0.9574 - val_loss: 0.3319 - val_acc: 0.8712
Epoch 10/10
625/625 [==============================] - 2s 4ms/step - loss: 0.1082 - acc: 0.9626 - val_loss: 0.3462 - val_acc: 0.8668
```

Plot

```
import matplotlib.pyplot as plt

# Train accuracy
accuracy = hist1.history["acc"]
# Validation accuracy
val_accuracy = hist1.history["val_acc"]
# Train loss
Train_loss = hist1.history["loss"]
# Validation loss
val_loss = hist1.history["val_loss"]


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

plt.plot(epochs, accuracy, "red", label = "Training acc")
plt.plot(epochs, val_accuracy, "b", label = "Validation acc")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, Train_loss, "red", label = "Training loss")
plt.plot(epochs, val_loss, "b", label = "Validation loss")
plt.title("Training and validation loss")
plt.legend()

plt.show()
```





Validating Test loss and Test Accuracy

```
test_loss, test_acc = model1.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_acc)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.3464 - acc: 0.8656
Test loss: 0.34639772772789
Test accuracy: 0.8656399846076965
```

Custom-trained embedding layer with training sample size = 5000

```
features=10000
length=150
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=features)

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

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

x_tr = x_train[:5000]
y_tr = y_train[:5000]
```

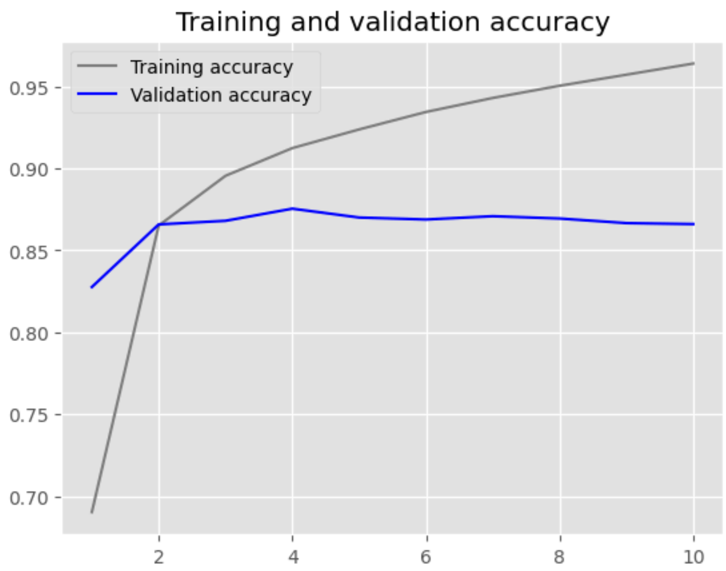
Define and compile a sequential neural network model with an embedding layer, flattening layer, and dense output layer for binary classification, and train the model on the training data with 20% validation split.

```
model2 = Sequential()
model2.add(Embedding(10000, 8, input_length=length))
model2.add(Flatten())
model2.add(Dense(1, activation='sigmoid'))
model2.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model2.summary()
hist2 = model2.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (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
625/625 [==============================] - 17s 27ms/step - loss: 0.6027 - acc: 0.6903 - val_loss: 0.4263 - val_acc: 0.8276
Epoch 2/10
625/625 [==============================] - 5s 7ms/step - loss: 0.3346 - acc: 0.8651 - val_loss: 0.3205 - val_acc: 0.8658
Epoch 3/10
625/625 [==============================] - 4s 6ms/step - loss: 0.2592 - acc: 0.8954 - val_loss: 0.3051 - val_acc: 0.8680
Epoch 4/10
625/625 [==============================] - 4s 6ms/step - loss: 0.2236 - acc: 0.9123 - val_loss: 0.3004 - val_acc: 0.8754
Epoch 5/10
625/625 [==============================] - 3s 4ms/step - loss: 0.1989 - acc: 0.9237 - val_loss: 0.3039 - val_acc: 0.8700
Epoch 6/10
625/625 [==============================] - 2s 4ms/step - loss: 0.1781 - acc: 0.9344 - val_loss: 0.3084 - val_acc: 0.8688
Epoch 7/10
625/625 [==============================] - 2s 4ms/step - loss: 0.1598 - acc: 0.9430 - val_loss: 0.3146 - val_acc: 0.8708
Epoch 8/10
625/625 [==============================] - 3s 4ms/step - loss: 0.1425 - acc: 0.9504 - val_loss: 0.3241 - val_acc: 0.8694
Epoch 9/10
625/625 [==============================] - 4s 6ms/step - loss: 0.1261 - acc: 0.9571 - val_loss: 0.3400 - val_acc: 0.8666
Epoch 10/10
625/625 [==============================] - 2s 4ms/step - loss: 0.1100 - acc: 0.9639 - val_loss: 0.3438 - val_acc: 0.8660
```

Plot

```
accuracy2 = hist2.history['acc']
val_accuracy2 = hist2.history['val_acc']
Train_loss2 = hist2.history['loss']
val_loss2 = hist2.history['val_loss']

epochs = range(1, len(accuracy2) + 1)

plt.plot(epochs, accuracy2, 'grey', label='Training accuracy')
plt.plot(epochs, val_accuracy2, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, Train_loss2, 'grey', label='Training loss')
plt.plot(epochs, val_loss2, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```



Validating Test loss and accuracy

```
test_loss2, test_accuracy2 = model2.evaluate(x_test, y_test)
print('Test loss:', test_loss2)
print('Test accuracy:', test_accuracy2)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.3466 - acc: 0.8664
Test loss: 0.3466218113899231
Test accuracy: 0.8664399981498718
```

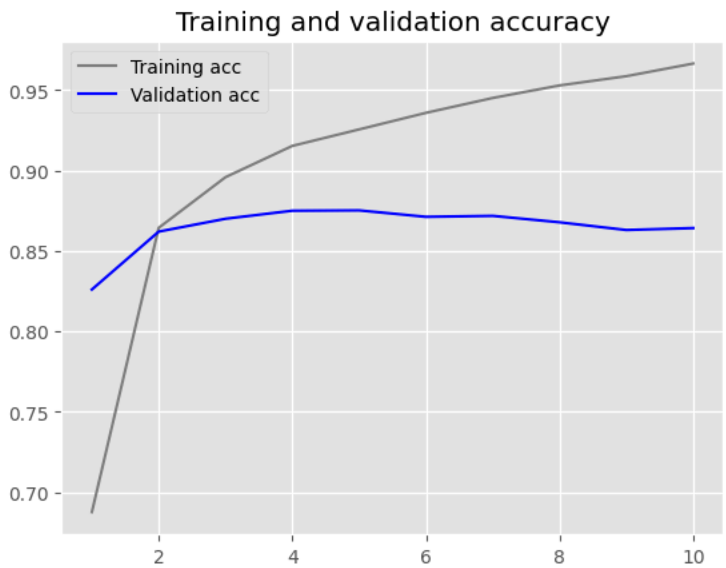
Custom-trained embedding layer with training sample size = 1000

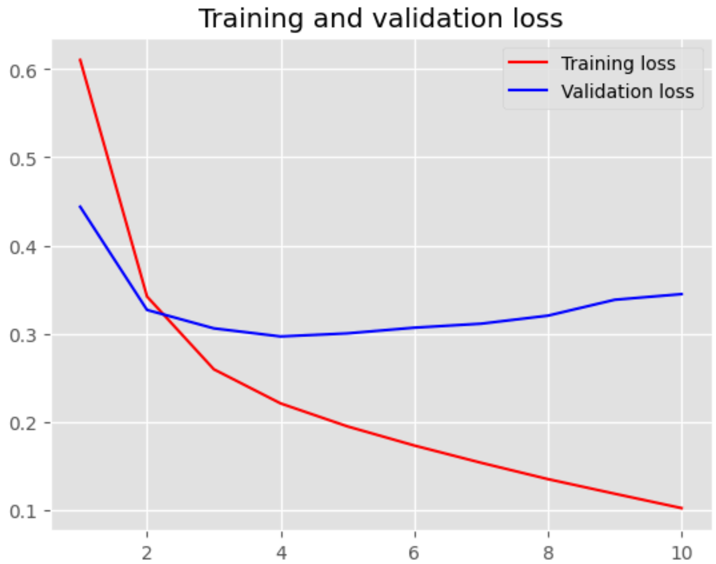
```
features=10000
length=150
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=features)

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

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

x_tr = x_train[:1000]
y_tr = y_train[:1000]


model3 = Sequential()
model3.add(Embedding(10000, 8, input_length=length))
model3.add(Flatten())
model3.add(Dense(1, activation='sigmoid'))
model3.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model3.summary()
hist3 = model3.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 150, 8)            80000

 flatten_2 (Flatten)         (None, 1200)              0

 dense_2 (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 [==============================] - 16s 25ms/step - loss: 0.6104 - acc: 0.6877 - val_loss: 0.4440 - val_acc: 0.8260
Epoch 2/10
625/625 [==============================] - 4s 6ms/step - loss: 0.3422 - acc: 0.8644 - val_loss: 0.3271 - val_acc: 0.8620
Epoch 3/10
625/625 [==============================] - 3s 5ms/step - loss: 0.2598 - acc: 0.8959 - val_loss: 0.3062 - val_acc: 0.8700
Epoch 4/10
625/625 [==============================] - 3s 5ms/step - loss: 0.2210 - acc: 0.9153 - val_loss: 0.2970 - val_acc: 0.8750
Epoch 5/10
625/625 [==============================] - 3s 6ms/step - loss: 0.1950 - acc: 0.9255 - val_loss: 0.3005 - val_acc: 0.8752
Epoch 6/10
625/625 [==============================] - 3s 4ms/step - loss: 0.1734 - acc: 0.9359 - val_loss: 0.3069 - val_acc: 0.8712
Epoch 7/10
625/625 [==============================] - 2s 4ms/step - loss: 0.1538 - acc: 0.9451 - val_loss: 0.3114 - val_acc: 0.8718
Epoch 8/10
625/625 [==============================] - 3s 4ms/step - loss: 0.1353 - acc: 0.9529 - val_loss: 0.3206 - val_acc: 0.8678
Epoch 9/10
625/625 [==============================] - 2s 4ms/step - loss: 0.1187 - acc: 0.9586 - val_loss: 0.3386 - val_acc: 0.8630
Epoch 10/10
625/625 [==============================] - 4s 6ms/step - loss: 0.1024 - acc: 0.9665 - val_loss: 0.3450 - val_acc: 0.8642
```

Plot

```
accuracy3 = hist3.history["acc"]
val_accuracy3 = hist3.history["val_acc"]
Train_loss3 = hist3.history["loss"]
val_loss3 = hist3.history["val_loss"]

epochs = range(1, len(accuracy3) + 1)

plt.plot(epochs, accuracy3, "grey", label = "Training acc")
plt.plot(epochs, val_accuracy3, "b", label = "Validation acc")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, Train_loss3, "red", label = "Training loss")
plt.plot(epochs, val_loss3, "b", label = "Validation loss")
plt.title("Training and validation loss")
plt.legend()

plt.show()
```





Validation of accuracy and loss

```
test_loss3, test_accuracy3 = model3.evaluate(x_test, y_test)
print('Test loss:', test_loss3)
print('Test accuracy:', test_accuracy3)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.3474 - acc: 0.8653
Test loss: 0.34735482931137085
Test accuracy: 0.8652799725532532
```

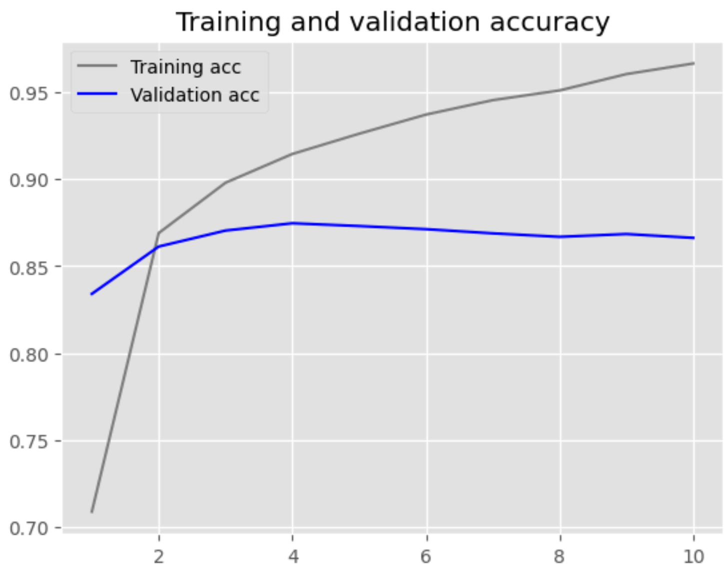
Custom-trained embedding layer with training sample size = 10000

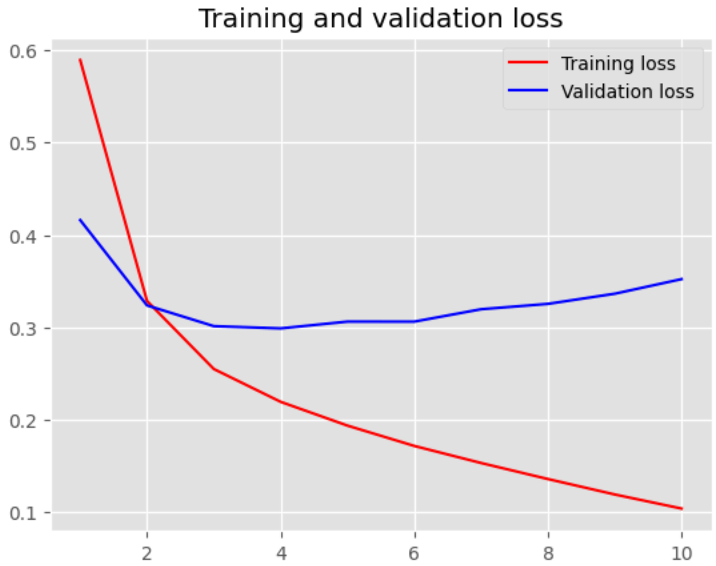
```
features=10000
length=150
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=features)

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

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

x_tr = x_train[:10000]
y_tr = y_train[:10000]
```

```
model4 = Sequential()
model4.add(Embedding(10000, 8, input_length=length))
model4.add(Flatten())
model4.add(Dense(1, activation='sigmoid'))
model4.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model4.summary()
hist4 = model4.fit(x_train, y_train,
                   epochs=10,
                   batch_size=32,
                   validation_split=0.2)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 150, 8)            80000

 flatten_3 (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
625/625 [==============================] - 15s 23ms/step - loss: 0.5894 - acc: 0.7088 - val_loss: 0.4161 - val_acc: 0.8340
Epoch 2/10
625/625 [==============================] - 6s 9ms/step - loss: 0.3285 - acc: 0.8690 - val_loss: 0.3237 - val_acc: 0.8612
Epoch 3/10
625/625 [==============================] - 3s 5ms/step - loss: 0.2549 - acc: 0.8979 - val_loss: 0.3013 - val_acc: 0.8704
Epoch 4/10
625/625 [==============================] - 3s 5ms/step - loss: 0.2193 - acc: 0.9144 - val_loss: 0.2988 - val_acc: 0.8746
Epoch 5/10
625/625 [==============================] - 3s 5ms/step - loss: 0.1936 - acc: 0.9261 - val_loss: 0.3062 - val_acc: 0.8730
Epoch 6/10
625/625 [==============================] - 3s 4ms/step - loss: 0.1716 - acc: 0.9371 - val_loss: 0.3062 - val_acc: 0.8712
Epoch 7/10
625/625 [==============================] - 2s 4ms/step - loss: 0.1531 - acc: 0.9454 - val_loss: 0.3196 - val_acc: 0.8688
Epoch 8/10
625/625 [==============================] - 2s 4ms/step - loss: 0.1357 - acc: 0.9510 - val_loss: 0.3254 - val_acc: 0.8668
Epoch 9/10
625/625 [==============================] - 3s 4ms/step - loss: 0.1190 - acc: 0.9604 - val_loss: 0.3364 - val_acc: 0.8684
Epoch 10/10
625/625 [==============================] - 3s 5ms/step - loss: 0.1038 - acc: 0.9665 - val_loss: 0.3522 - val_acc: 0.8662
```

```
accuracy4 = hist4.history["acc"]
val_accuracy4 = hist4.history["val_acc"]
Train_loss4 = hist4.history["loss"]
val_loss4 = hist4.history["val_loss"]

epochs = range(1, len(accuracy4) + 1)

plt.plot(epochs, accuracy4, "grey", label = "Training acc")
plt.plot(epochs, val_accuracy4, "b", label = "Validation acc")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, Train_loss4, "red", label = "Training loss")
plt.plot(epochs, val_loss4, "b", label = "Validation loss")
plt.title("Training and validation loss")
plt.legend()

plt.show()
```






```
test_loss4, test_accuracy4 = model4.evaluate(x_test, y_test)
print('Test loss:', test_loss4)
print('Test accuracy:', test_accuracy4)
```

```
782/782 [==============================] - 2s 3ms/step - loss: 0.3567 - acc: 0.8641
Test loss: 0.35666316747665405
Test accuracy: 0.864080011844635
```

```
!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  18.0M      0  0:00:04  0:00:04 --:--:-- 18.0M
```

```
import os
import shutil

imdb = 'aclImdb'
training = os.path.join(imdb, 'train')

labels = []
texts = []

for label_type in ['neg', 'pos']:
    dir_name = os.path.join(training, label_type)
    for fname in os.listdir(dir_name):
        if fname[-4:] == '.txt':
            f = open(os.path.join(dir_name, fname), encoding='utf-8')
            texts.append(f.read())
            f.close()
            if label_type == 'neg':
                labels.append(0)
            else:
                labels.append(1)
```

Utilizing Pretrained Word Embeddings: When there is insufficient training data to generate effective word embeddings, use pretrained word embeddings to achieve the desired solution.

Data tokenization

```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

length2 = 150 # cut off review after 150 words
train_data = 100 # Training sample 100
val_data = 10000 # Validation sample 10000
words = 10000 # Considers only the top 10000 words in the dataset

tokenizer1 = Tokenizer(num_words=words)
tokenizer1.fit_on_texts(texts)
sequences = tokenizer1.texts_to_sequences(texts)
word_index = tokenizer1.word_index
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=length2)

labels = np.asarray(labels)
print("Shape of data tensor:", data.shape)
print("Shape of label tensor:", labels.shape)
# Splits data into training and validation set, but shuffles is, since samples are ordered:
# all negatives first, then all positive
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]

x_tr = data[:train_data] # (200, 100)
y_tr = labels[:train_data] # shape (200,)
x_val = data[train_data:train_data+val_data] # shape (10000, 100)
y_val = labels[train_data:train_data+val_data] # shape (10000,)
```

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

GloVe word embedding installation and set up

```
import numpy as np
import requests
from io import BytesIO
import zipfile

glove_url = 'https://nlp.stanford.edu/data/glove.6B.zip'  # URL to download GloVe embeddings
glove_zip = requests.get(glove_url)

# Unzip the contents
with zipfile.ZipFile(BytesIO(glove_zip.content)) as z:
    z.extractall('/content/glove')

# Loading GloVe embeddings into memory
emb_index = {}
with open('/content/glove/glove.6B.100d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        emb_index[word] = coefs

print("Found %s word vectors." % len(emb_index))
```

```
Found 400000 word vectors.
```

We trained the 6B version of the GloVe model, which includes 400,000 words and 6 billion tokens, using data from Gigaword 5 and Wikipedia.

Initializing the GloVe word embeddings matrix

pretrained word embedding layer with training sample size = 100

```
emb_di = 100

emb_matrix = np.zeros((words, emb_di))
for word, i in word_index.items():
    emb_vector = emb_index.get(word)
    if i < words:
        if emb_vector is not None:
            # Words not found in embedding index will be all-zeros.
            emb_matrix[i] = emb_vector
```

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

model = Sequential()
model.add(Embedding(words, emb_di, input_length=length2))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 150, 100)          1000000

 flatten_4 (Flatten)         (None, 15000)             0

 dense_4 (Dense)             (None, 32)                480032

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 1480065 (5.65 MB)
Trainable params: 1480065 (5.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

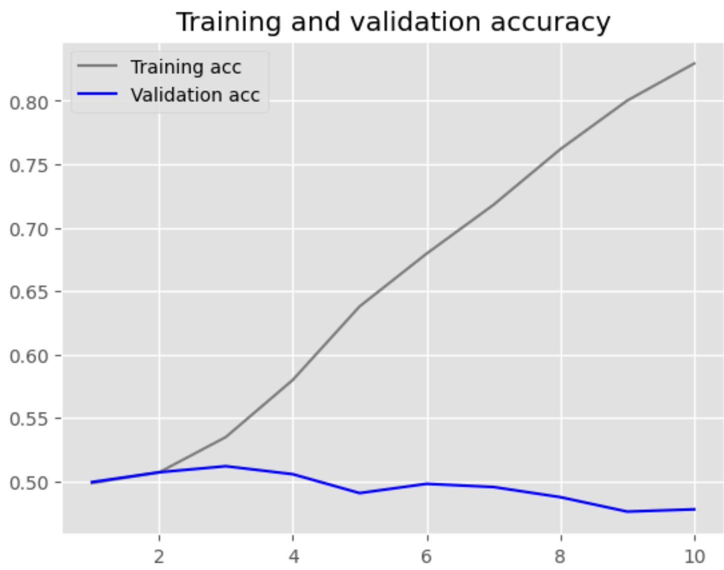
The Embedding layer includes pretrained word embeddings. By setting `trainable` to False before using the Embedding layer, you ensure that these embeddings cannot be modified during training. If you set `trainable` to True, the optimization process can update the word embedding values. To prevent pretrained embeddings from losing their existing knowledge during the training of other parts of the model, it is best to keep them untrainable.

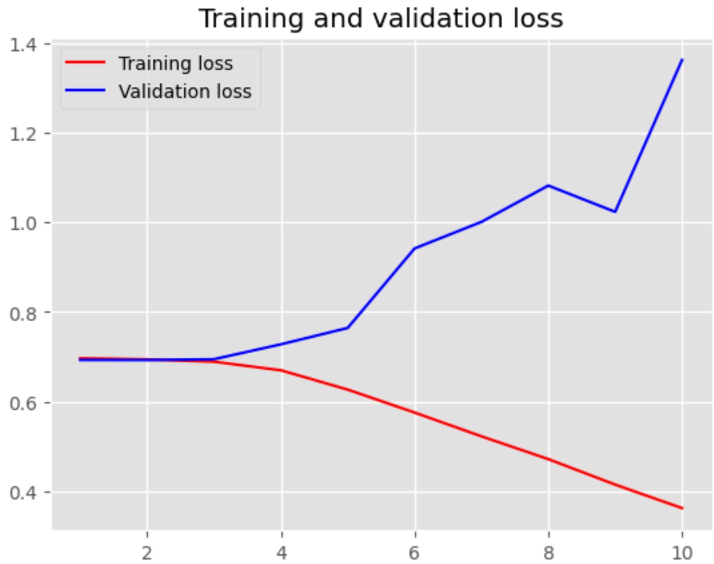
```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
hist = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
model.save_weights('pre_trained_glove_model.h5')
```

```
Epoch 1/10
782/782 [==============================] - 6s 6ms/step - loss: 0.6969 - acc: 0.4987 - val_loss: 0.6931 - val_acc: 0.4995
Epoch 2/10
782/782 [==============================] - 5s 7ms/step - loss: 0.6945 - acc: 0.5070 - val_loss: 0.6929 - val_acc: 0.5072
Epoch 3/10
782/782 [==============================] - 3s 4ms/step - loss: 0.6892 - acc: 0.5350 - val_loss: 0.6948 - val_acc: 0.5120
Epoch 4/10
782/782 [==============================] - 3s 4ms/step - loss: 0.6701 - acc: 0.5800 - val_loss: 0.7280 - val_acc: 0.5057
Epoch 5/10
782/782 [==============================] - 3s 4ms/step - loss: 0.6269 - acc: 0.6380 - val_loss: 0.7645 - val_acc: 0.4908
Epoch 6/10
782/782 [==============================] - 4s 5ms/step - loss: 0.5758 - acc: 0.6795 - val_loss: 0.9419 - val_acc: 0.4981
Epoch 7/10
782/782 [==============================] - 5s 6ms/step - loss: 0.5227 - acc: 0.7180 - val_loss: 1.0012 - val_acc: 0.4956
Epoch 8/10
782/782 [==============================] - 5s 7ms/step - loss: 0.4718 - acc: 0.7619 - val_loss: 1.0823 - val_acc: 0.4876
Epoch 9/10
782/782 [==============================] - 3s 4ms/step - loss: 0.4149 - acc: 0.8002 - val_loss: 1.0237 - val_acc: 0.4762
Epoch 10/10
782/782 [==============================] - 3s 4ms/step - loss: 0.3625 - acc: 0.8294 - val_loss: 1.3625 - val_acc: 0.4780
```

As expected with the limited amount of training data, the model overfits quickly. This also explains the significant fluctuations in validation accuracy.

```
import matplotlib.pyplot as plt

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

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

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

plt.figure()

plt.plot(epochs, train_loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```





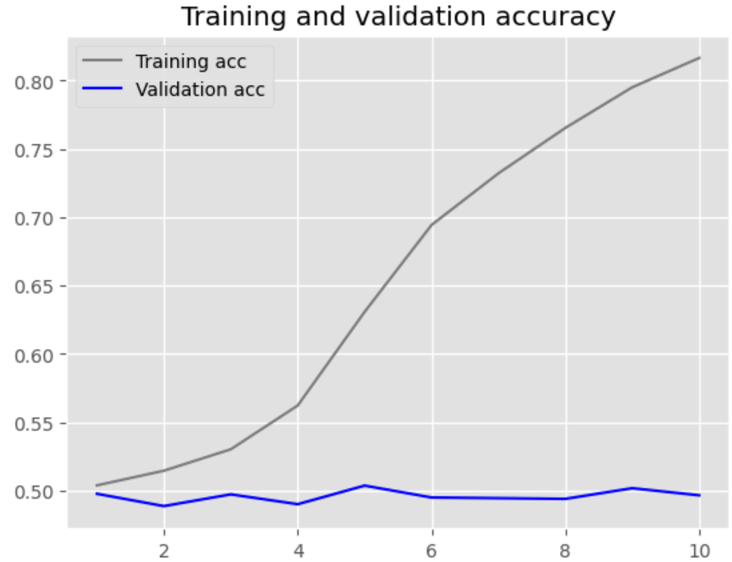

```
test_loss, test_accuracy= model.evaluate(x_test, y_test)
print('Test loss:', test_loss)
print('Test accuracy:', test_accuracy)
```

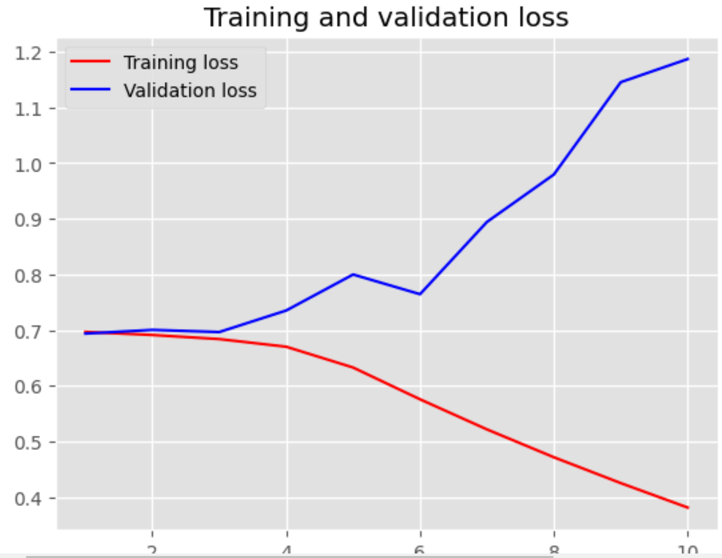
```
782/782 [==============================] - 3s 3ms/step - loss: 1.1396 - acc: 0.5521
Test loss: 1.1395903825759888
Test accuracy: 0.552079975605011
```

Pretrained word embedding layer with training sample size = 5000

```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

length2 = 150
train_data = 5000 # Training sample is 5000
val_data = 10000
words = 10000

tokenizer2 = Tokenizer(num_words=words)
tokenizer2.fit_on_texts(texts)
sequences = tokenizer2.texts_to_sequences(texts)
word_index = tokenizer2.word_index
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=length2)

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_tr = data[:train_data]
y_tr = labels[:train_data]
x_validation = data[train_data:train_data+val_data]
y_validation = labels[train_data:train_data+val_data]
emb_di = 100

emb_matrix = np.zeros((words, emb_di))
for word, i in word_index.items():
    emb_vector = emb_index.get(word)
    if i < words:
        if emb_vector is not None:

            emb_matrix[i] = emb_vector
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model11 = Sequential()
model11.add(Embedding(words, emb_di, input_length=length2))
model11.add(Flatten())
model11.add(Dense(32, activation='relu'))
model11.add(Dense(1, activation='sigmoid'))
model11.summary()

model11.layers[0].set_weights([emb_matrix])
model11.layers[0].trainable = False
model11.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
hist11 = model11.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_validation, y_validation))
model11.save_weights('pre_trained_glove_model.h5')
import matplotlib.pyplot as plt

accuracy11 = hist11.history['acc']
val_acc11 = hist11.history['val_acc']
train_loss11 = hist11.history['loss']
val_loss11 = hist11.history['val_loss']

epochs = range(1, len(accuracy11) + 1)

plt.plot(epochs, accuracy11, 'grey', label='Training acc')
plt.plot(epochs, val_acc11, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, train_loss11, 'red', label='Training loss')
plt.plot(epochs, val_loss11, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
 dense_6 (Dense)             (None, 32)                480032

 dense_7 (Dense)             (None, 1)                 33

=================================================================
Total params: 1480065 (5.65 MB)
Trainable params: 1480065 (5.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
782/782 [==============================] - 6s 6ms/step - loss: 0.6968 - acc: 0.503
Epoch 2/10
782/782 [==============================] - 5s 6ms/step - loss: 0.6913 - acc: 0.514
Epoch 3/10
782/782 [==============================] - 5s 6ms/step - loss: 0.6839 - acc: 0.530
Epoch 4/10
782/782 [==============================] - 4s 5ms/step - loss: 0.6702 - acc: 0.562
Epoch 5/10
782/782 [==============================] - 3s 4ms/step - loss: 0.6328 - acc: 0.630
Epoch 6/10
782/782 [==============================] - 3s 4ms/step - loss: 0.5758 - acc: 0.694
Epoch 7/10
782/782 [==============================] - 4s 5ms/step - loss: 0.5217 - acc: 0.732
Epoch 8/10
782/782 [==============================] - 5s 6ms/step - loss: 0.4716 - acc: 0.765
Epoch 9/10
782/782 [==============================] - 5s 7ms/step - loss: 0.4252 - acc: 0.795
Epoch 10/10
782/782 [==============================] - 4s 5ms/step - loss: 0.3813 - acc: 0.816
```





```
test_loss11, test_accuracy11 = model11.evaluate(x_test, y_test)
print('Test loss:', test_loss11)
print('Test accuracy:', test_accuracy11)
```

```
782/782 [==============================] - 2s 3ms/step - loss: 1.0352 - acc: 0.5764
Test loss: 1.0352002382278442
Test accuracy: 0.5763599872589111
```

Pretrained word embedding layer with training sample size = 1000

```
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import numpy as np

length = 150
train_data = 1000 #Trains on 1000 samples
val_data = 10000
words = 10000

tokenizer3 = Tokenizer(num_words=words)
tokenizer3.fit_on_texts(texts)
sequences = tokenizer3.texts_to_sequences(texts)
word_index = tokenizer3.word_index
print("Found %s unique tokens." % len(word_index))

data = pad_sequences(sequences, maxlen=length)

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_tr = data[:train_data]
y_tr = labels[:train_data]
x_val = data[train_data:train_data+val_data]
y_val = labels[train_data:train_data+val_data]
emb_dim = 100

emb_matrix = np.zeros((words, emb_dim))
for word, i in word_index.items():
    emb_vector = emb_index.get(word)
    if i < words:
        if emb_vector is not None:

            emb_matrix[i] = emb_vector
            from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

model12 = Sequential()
model12.add(Embedding(words, emb_dim, input_length=length))
model12.add(Flatten())
model12.add(Dense(32, activation='relu'))
model12.add(Dense(1, activation='sigmoid'))
model12.summary()

model12.layers[0].set_weights([emb_matrix])
model12.layers[0].trainable = False
model12.compile(optimizer='rmsprop',
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.