Using RNNs to classify sentiment on IMDB data

In this assignment,you will train three types of RNNs: "vanilla" RNN, LSTM and GRU to predict the sentiment on IMDB reviews.

Keras provides a convenient interface to load the data and immediately encode the words into integers (based on the most common words). This will save you a lot of the drudgery that is usually involved when working with raw text.

The IMDB is data consists of 25000 training sequences and 25000 test sequences. The outcome is binary (positive/negative) and both outcomes are equally represented in both the training and the test set.

Walk through the followinng steps to prepare the data and the building of an RNN model.

In [1]:
```
import tensorflow as tf
from tensorflow import keras
from keras.datasets import imdb

# Load the IMDB data
# Picking up the most common 2000 words

max_features = 2000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
```

In [2]:
```
# Installing the necessary packages
pip install Tensorflow
```

```
Requirement already satisfied: Tensorflow in c:\users\shiva\anaconda3\lib\site-packag
es (2.11.0)
Requirement already satisfied: tensorflow-intel==2.11.0 in c:\users\shiva\anaconda3\l
ib\site-packages (from Tensorflow) (2.11.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\shiva\anaconda3\lib\site-pa
ckages (from tensorflow-intel==2.11.0->Tensorflow) (1.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\shiva\anaconda3\l
ib\site-packages (from tensorflow-intel==2.11.0->Tensorflow) (4.1.1)
Requirement already satisfied: keras<2.12,>=2.11.0 in c:\users\shiva\anaconda3\lib\si
te-packages (from tensorflow-intel==2.11.0->Tensorflow) (2.11.0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in c:\users\shiva\anaconda3\lib\si
te-packages (from tensorflow-intel==2.11.0->Tensorflow) (0.4.0)
Requirement already satisfied: tensorflow-estimator<2.12,>=2.11.0 in c:\users\shiva\a
naconda3\lib\site-packages (from tensorflow-intel==2.11.0->Tensorflow) (2.11.0)
Requirement already satisfied: flatbuffers>=2.0 in c:\users\shiva\anaconda3\lib\site-
packages (from tensorflow-intel==2.11.0->Tensorflow) (23.1.21)
Requirement already satisfied: six>=1.12.0 in c:\users\shiva\anaconda3\lib\site-packa
ges (from tensorflow-intel==2.11.0->Tensorflow) (1.16.0)
Requirement already satisfied: tensorboard<2.12,>=2.11 in c:\users\shiva\anaconda3\li
b\site-packages (from tensorflow-intel==2.11.0->Tensorflow) (2.11.2)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in c:\users\shiva\anaconda3\lib
\site-packages (from tensorflow-intel==2.11.0->Tensorflow) (3.19.1)
Requirement already satisfied: numpy>=1.20 in c:\users\shiva\anaconda3\lib\site-packa
ges (from tensorflow-intel==2.11.0->Tensorflow) (1.21.5)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\users\shiva
\anaconda3\lib\site-packages (from tensorflow-intel==2.11.0->Tensorflow) (0.30.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\shiva\anaconda3\lib\si
te-packages (from tensorflow-intel==2.11.0->Tensorflow) (0.2.0)
Requirement already satisfied: wrapt>=1.11.0 in c:\users\shiva\anaconda3\lib\site-pac
kages (from tensorflow-intel==2.11.0->Tensorflow) (1.12.1)
Requirement already satisfied: packaging in c:\users\shiva\anaconda3\lib\site-package
s (from tensorflow-intel==2.11.0->Tensorflow) (21.3)
Requirement already satisfied: libclang>=13.0.0 in c:\users\shiva\anaconda3\lib\site-
packages (from tensorflow-intel==2.11.0->Tensorflow) (15.0.6.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\shiva\anaconda3\lib\si
te-packages (from tensorflow-intel==2.11.0->Tensorflow) (1.42.0)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\shiva\anaconda3\lib\site
-packages (from tensorflow-intel==2.11.0->Tensorflow) (3.3.0)
Requirement already satisfied: setuptools in c:\users\shiva\anaconda3\lib\site-packag
es (from tensorflow-intel==2.11.0->Tensorflow) (61.2.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\shiva\anaconda3\lib\site-
packages (from tensorflow-intel==2.11.0->Tensorflow) (2.2.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\shiva\anaconda3\lib\site
-packages (from tensorflow-intel==2.11.0->Tensorflow) (1.6.3)
Requirement already satisfied: h5py>=2.9.0 in c:\users\shiva\anaconda3\lib\site-packa
ges (from tensorflow-intel==2.11.0->Tensorflow) (3.6.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\shiva\anaconda3\lib\sit
e-packages (from astunparse>=1.6.0->tensorflow-intel==2.11.0->Tensorflow) (0.37.1)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\shiva\anaconda3\lib\site-p
ackages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->Tensorflow) (2.0.3)
Requirement already satisfied: google-auth<3,>=1.6.3 in c:\users\shiva\anaconda3\lib
\site-packages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->Tensorflow)
(1.33.0)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\shiva\anaconda3\lib\si
te-packages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->Tensorflow) (2.2
7.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in c:\users\shiva\anacon
da3\lib\site-packages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->Tensor
flow) (1.8.1)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in c:\users\shiv
```

```
a\anaconda3\lib\site-packages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0
->Tensorflow) (0.6.1)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in c:\users\shiva\ana
conda3\lib\site-packages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->Ten
sorflow) (0.4.6)
Requirement already satisfied: markdown>=2.6.8 in c:\users\shiva\anaconda3\lib\site-p
ackages (from tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->Tensorflow) (3.3.4)
Requirement already satisfied: rsa<5,>=3.1.4 in c:\users\shiva\anaconda3\lib\site-pac
kages (from google-auth<3,>=1.6.3->tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0-
>Tensorflow) (4.7.2)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in c:\users\shiva\anaconda3\lib
\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.12,>=2.11->tensorflow-intel
==2.11.0->Tensorflow) (4.2.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in c:\users\shiva\anaconda3\lib
\site-packages (from google-auth<3,>=1.6.3->tensorboard<2.12,>=2.11->tensorflow-intel
==2.11.0->Tensorflow) (0.2.8)
Requirement already satisfied: requests-oauthlib>=0.7.0 in c:\users\shiva\anaconda3\l
ib\site-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.12,>=2.11->ten
sorflow-intel==2.11.0->Tensorflow) (1.3.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in c:\users\shiva\anaconda3\lib\s
ite-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.12,>=
2.11->tensorflow-intel==2.11.0->Tensorflow) (0.4.8)
Requirement already satisfied: idna<4,>=2.5 in c:\users\shiva\anaconda3\lib\site-pack
ages (from requests<3,>=2.21.0->tensorboard<2.12,>=2.11->tensorflow-intel==2.11.0->Te
nsorflow) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\shiva\anaconda3\lib\sit
e-packages (from requests<3,>=2.21.0->tensorboard<2.12,>=2.11->tensorflow-intel==2.1
1.0->Tensorflow) (2021.10.8)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\shiva\anaconda3
\lib\site-packages (from requests<3,>=2.21.0->tensorboard<2.12,>=2.11->tensorflow-int
el==2.11.0->Tensorflow) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\shiva\anaconda3\lib
\site-packages (from requests<3,>=2.21.0->tensorboard<2.12,>=2.11->tensorflow-intel==
2.11.0->Tensorflow) (1.26.9)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\shiva\anaconda3\lib\site-p
ackages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard
<2.12,>=2.11->tensorflow-intel==2.11.0->Tensorflow) (3.2.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\shiva\anaconda3\l
ib\site-packages (from packaging->tensorflow-intel==2.11.0->Tensorflow) (3.0.4)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
pip install keras
```

```
Requirement already satisfied: keras in c:\users\shiva\anaconda3\lib\site-packages
(2.11.0)
Note: you may need to restart the kernel to use updated packages.
```

1- Use the `imdb.load_data()` to load in the data

2- Specify the maximum length of a sequence to 30 words and the pick the 2000 most common words.

In [4]:
```
# Check the number of sequences in the train and test sets
print("x_train = %d train sequences" % x_train.shape[0])
print("x_test = %d train sequences" % x_test.shape[0])
```

```
x_train = 25000 train sequences
x_test = 25000 train sequences
```

3- Check that the number of sequences in train and test datasets are equal (default split):

Expected output:

- `x_train = 25000 train sequences`
- `x_test = 25000 test sequences`

In [5]:
```
# Set the maximum sequence length to 30
max_words = 30
x_train = tf.keras.utils.pad_sequences(x_train, maxlen=max_words)
x_test = tf.keras.utils.pad_sequences(x_test, maxlen=max_words)
```

4- Pad (or truncate) the sequences so that they are of the maximum length

In [6]:
```
# Printing the shape of each x_train and x_test
print("x_train shape:", x_train.shape)
print("x_test shape:", x_test.shape)
```

```
x_train shape: (25000, 30)
x_test shape: (25000, 30)
```

5- After padding or truncating, check the dimensionality of x_train and x_test.

Expected output:

- `x_train shape: (25000, 30)`
- `x_test shape: (25000, 30)`

In [7]:
```
from keras.callbacks import TensorBoard
from sklearn.model_selection import StratifiedKFold
%load_ext tensorboard

# Define the callback and specify the log directory
tensorboard = TensorBoard(log_dir='logs_assignment_3/', histogram_freq=0, write_graph=

# Split the data into 10 folds using StratifiedKFold
kfold = StratifiedKFold(n_splits=10, shuffle=True)
```

In the above cell, KFold is a traditional cross-validation technique that divides a dataset into k equal-sized folds, trains the model on k-1 folds, and validates the model on the remaining fold. This is the traditional method but we used "StratifiedKFold" because it aims to ensure that the distribution of the target variable (i.e., the class labels) is roughly the same in each fold. This technique helps our model to deal with balanced data, which can improve its performance.

In [8]:
```
%tensorboard --logdir logs_assignment_3/
```

```
Reusing TensorBoard on port 6006 (pid 35356), started 0:11:56 ago. (Use '!kill 35356'
to kill it.)
```

Data could not be loaded.

The TensorBoard server may be down or inaccessible.

Last reload:

6- For all your models:

- Use tensorboard to run your experiments

- Use cross valdiation with 10 folds

7- Build the RNN with three layers:

- The SimpleRNN layer with 5 neurons and initialize its kernel with stddev=0.001
- The Embedding layer and initialize it by setting the word embedding dimension to 50. This means that this layer takes each integer in the sequence and embeds it in a 50-dimensional vector.
- The output layer has the sigmoid activation function.

In [9]:
```
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN, Dense
from keras.preprocessing import sequence
import time

# Define the model
keras.backend.clear_session()
model = Sequential()
model.add(Embedding(input_dim=max_features, output_dim=50, input_length=max_words))
model.add(SimpleRNN(units=5, kernel_initializer=keras.initializers.RandomNormal(stddev
                    activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
```

The above code snippet defines the Sequential model using Keras, where an Embedding layer is added as the first layer, followed by a SimpleRNN layer with 5 units, and finally a Dense layer with a single neuron and sigmoid activation. The model is used to classify IMDB movie reviews as positive or negative. The code also clears the Keras session before defining the model such that while re-running the code block won't let model to build on another model.

8- How many parameters have the embedding layer?

In [10]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 30, 50)            100000

 simple_rnn (SimpleRNN)      (None, 5)                 280

 dense (Dense)               (None, 1)                 6

=================================================================
Total params: 100,286
Trainable params: 100,286
Non-trainable params: 0
_________________________________________________________________
```

In [49]:
```
# print the summary of the model
model.summary()

# get the embedding layer by name
embedding_layer = model.get_layer('embedding')

# print the number of parameters in the embedding layer
print("Number of parameters in the embedding layer:", embedding_layer.count_params())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 30, 50)            100000

 simple_rnn (SimpleRNN)      (None, 5)                 280

 dense (Dense)               (None, 1)                 6

=================================================================
Total params: 100,286
Trainable params: 100,286
Non-trainable params: 0
_________________________________________________________________
Number of parameters in the embedding layer: 100000
```

9- Train the network with the RMSprop with learning rate of .0001 and epochs=10.

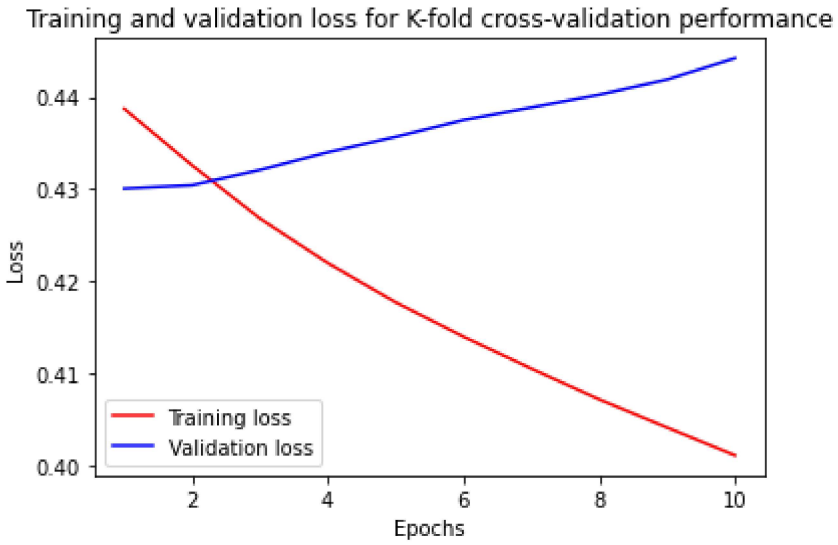
In [11]:
```
# Importing the RMSprop optimizer from keras package
# Intializing the learning rate to be 0.0001
from keras.optimizers import RMSprop
rmsprop = RMSprop(learning_rate=0.0001)
```

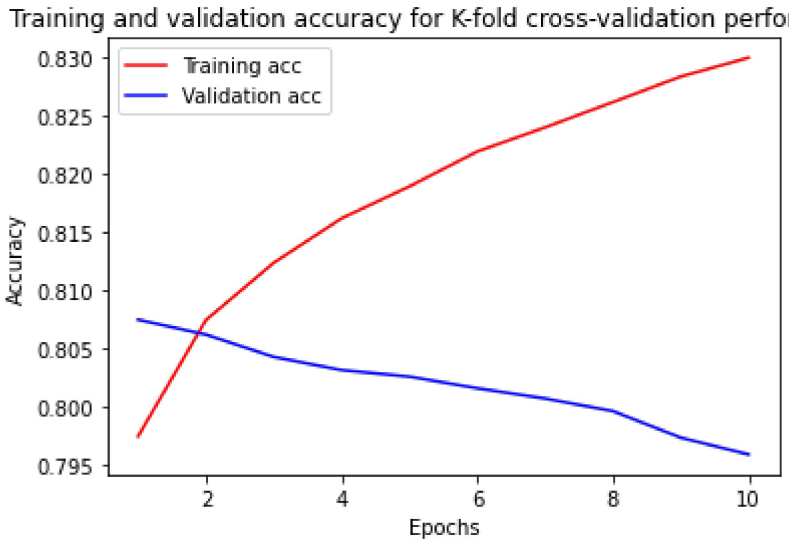
In [12]:
```
# Compile the model
model.compile(loss='binary_crossentropy', optimizer=rmsprop, metrics=['accuracy'])

loss = []
accuracy = []
val_loss = []
val_accuracy = []
test_accuracy = []

for train, test in kfold.split(x_train, y_train):
    # Train the model
    history = model.fit(x_train[train], y_train[train], epochs=10, batch_size=64, cal
              validation_data=(x_train[test], y_train[test]), verbose = 0)
    loss.append(history.history['loss'])
    accuracy.append(history.history['accuracy'])
    val_loss.append(history.history['val_loss'])
    val_accuracy.append(history.history['val_accuracy'])

    # Evaluate the model
    scores = model.evaluate(x_test, y_test, verbose=0)
    # Appending the Evaluated scores for the model on Test Dataset in a array for each
    test_accuracy.append(scores)
    # Printing the Accuracy of the Model while performing on the Test Dataset
    print("Test Accuracy: %.2f%%" % (scores[1]*100))
```

```
Test Accuracy: 72.40%
Test Accuracy: 76.60%
Test Accuracy: 76.80%
Test Accuracy: 76.72%
Test Accuracy: 76.88%
Test Accuracy: 76.84%
Test Accuracy: 76.27%
Test Accuracy: 75.78%
Test Accuracy: 75.41%
Test Accuracy: 74.70%
```

10- PLot the loss and accuracy metrics during the training and interpret the result.

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt

# Plot the average loss and accuracy across all folds over the training epochs
loss = np.mean(loss, axis=0)
accuracy = np.mean(accuracy, axis=0)
val_loss = np.mean(val_loss, axis=0)
val_accuracy = np.mean(val_accuracy, axis=0)

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

plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, accuracy, 'r', label='Training acc')
plt.plot(epochs, val_accuracy, 'b', label='Validation acc')
plt.title('Training and validation accuracy for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





In [14]:
```
# Printing the list of test losses and test accuracies for each folds
test_accuracy
```

Out[14]:
```
[[0.5497504472732544, 0.7240399718284607],
 [0.49029743671417236, 0.7659599781036377],
 [0.48873353004455566, 0.7679600119590759],
 [0.4895753860473633, 0.7672399878501892],
 [0.4881678521633148, 0.7687600255012512],
 [0.48756295442581177, 0.7684400081634521],
 [0.500869870185852, 0.7626799941062927],
 [0.515429675579071, 0.7578399777412415],
 [0.5281108617782593, 0.7541199922561646],
 [0.5438806414604187, 0.7470399737358093]]
```

11- Check the accuracy and the loss of your models on the test dataset.

In [15]:
```
# For our built baseline model we now print the test loss and accuracy for the final f
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test loss for baseline model: ", test_loss)
print("Test accuracy for baseline model: ", test_acc)
```

```
782/782 [==============================] - 2s 2ms/step - loss: 0.5439 - accuracy: 0.7
470
Test loss for baseline model:  0.5438806414604187
Test accuracy for baseline model:  0.7470399737358093
```

Tuning The Vanilla RNN Network

12- Prepare the data to use sequences of length 80 rather than length 30 and retrain your model. Did it improve the performance?

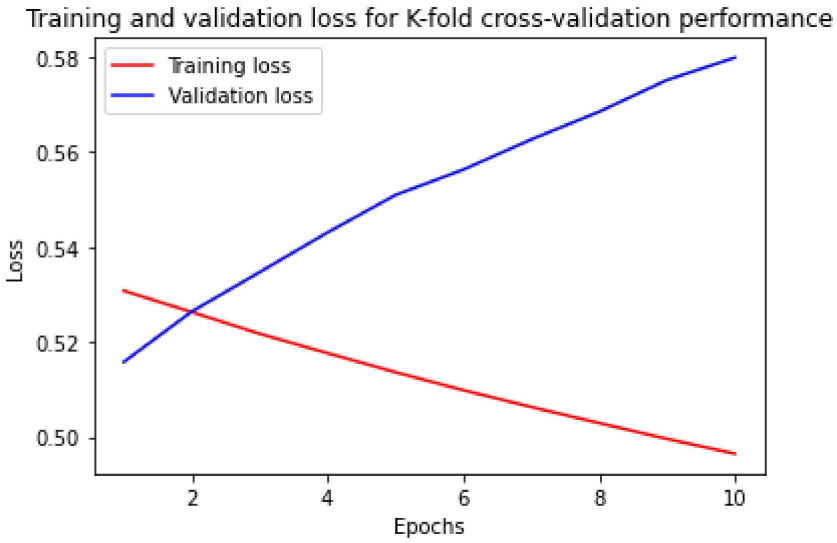
13- Try different values of the maximum length of a sequence ("max_features"). Can you improve the performance?

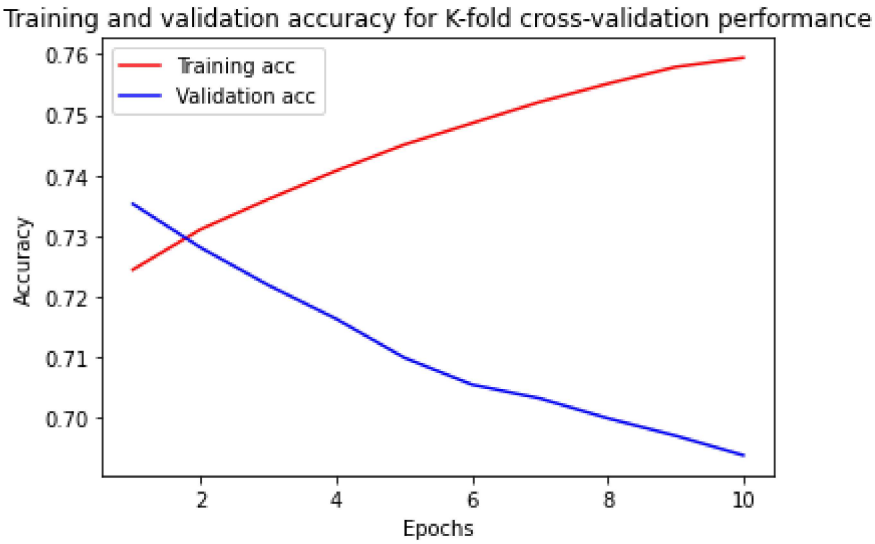
14- Try smaller and larger sizes of the RNN hidden dimension. How does it affect the model performance? How does it affect the run time?

In [16]:
```
maxlen = 80
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = tf.keras.utils.pad_sequences(x_train, maxlen=maxlen)
x_test = tf.keras.utils.pad_sequences(x_test, maxlen=maxlen)

start_time_80 = time.time()
keras.backend.clear_session()
model1 = Sequential()
model1.add(Embedding(input_dim=max_features, output_dim=50, input_length=maxlen))
model1.add(SimpleRNN(units=5, kernel_initializer=keras.initializers.RandomNormal(stdde
                     activation='tanh'))
model1.add(Dense(1, activation='sigmoid'))

# Compile the model
model1.compile(loss='binary_crossentropy', optimizer=rmsprop, metrics=['accuracy'])

loss1 = []
accuracy1 = []
val_loss1 = []
val_accuracy1 = []
test_accuracy1 = []

for train, test in kfold.split(x_train, y_train):
    # Train the model
    history1 = model1.fit(x_train[train], y_train[train], epochs=10, batch_size=64, ca
              validation_data=(x_train[test], y_train[test]), verbose=0)
    loss1.append(history1.history['loss'])
    accuracy1.append(history1.history['accuracy'])
    val_loss1.append(history1.history['val_loss'])
    val_accuracy1.append(history1.history['val_accuracy'])
    # Evaluate the model
    scores1 = model1.evaluate(x_test, y_test, verbose=0)
    test_accuracy1.append(scores)
    print("Test Accuracy: %.2f%%" % (scores[1]*100))

end_time_80 = time.time()
runtime_80 = end_time_80 - start_time_80
print(runtime_80)

# plot the average loss and accuracy across all folds over the training epochs
loss1 = np.mean(loss1, axis=0)
accuracy1 = np.mean(accuracy1, axis=0)
val_loss1 = np.mean(val_loss1, axis=0)
val_accuracy1 = np.mean(val_accuracy1, axis=0)
epochs = range(1, len(loss1) + 1)

plt.figure()
plt.plot(epochs, loss1, 'r', label='Training loss')
plt.plot(epochs, val_loss1, 'b', label='Validation loss')
plt.title('Training and validation loss for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, accuracy1, 'r', label='Training acc')
plt.plot(epochs, val_accuracy1, 'b', label='Validation acc')
plt.title('Training and validation accuracy for K-fold cross-validation performance')
```

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

```
Test Accuracy: 74.70%
Test Accuracy: 74.70%
Test Accuracy: 74.70%
Test Accuracy: 74.70%
Test Accuracy: 74.70%
Test Accuracy: 74.70%
Test Accuracy: 74.70%
Test Accuracy: 74.70%
Test Accuracy: 74.70%
Test Accuracy: 74.70%
580.8257400989532
```





This above code snippet performs Stratified k-fold cross-validation on the IMDB movie review dataset for our First model which is loaded and padded to a maximum length of 80, Remaining all are similar to the Baseline Model. Finally, the code generates plots of the training and validation loss as well as the training and validation accuracy. The runtime of the code is also recorded for further evaluation purposes.From the results, it shows that the model achieves a test accuracy of 74.70% consistently across all folds. The average training and validation losses

are quite high, suggesting that the model might be overfitting to the training data. It provides us with an insight that the further experimentation with different model architectures, hyperparameters, and regularization techniques may help to improve the model's performance. The entire process takes around 581 seconds to run.

Compared to Our Baseline Model, Our First Model does not greatly improve its performance. The test accuracies turned out to be same while the training and validation accuracies has been reduced by a little margin.On a whole, while performing on the test dataset the performance did not change i.e., constant.

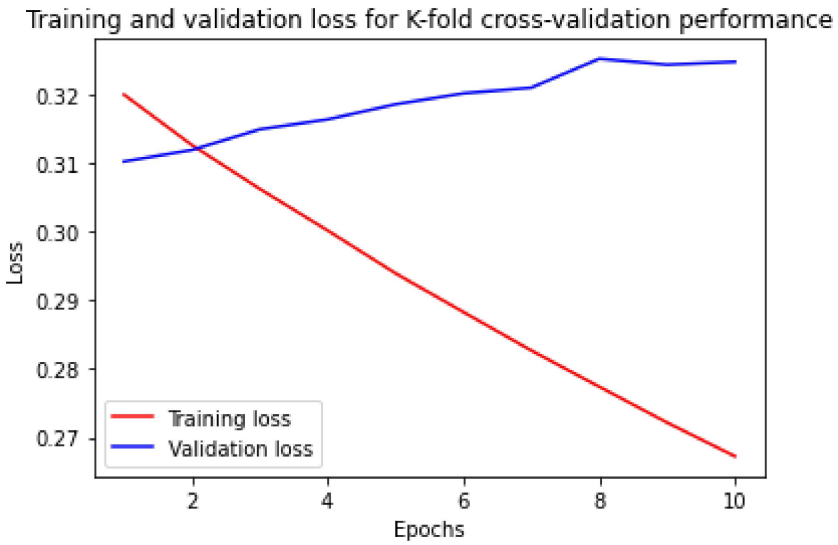
In [17]:
```
# Printing the list of test losses and test accuracies for each folds
test_accuracy1
```

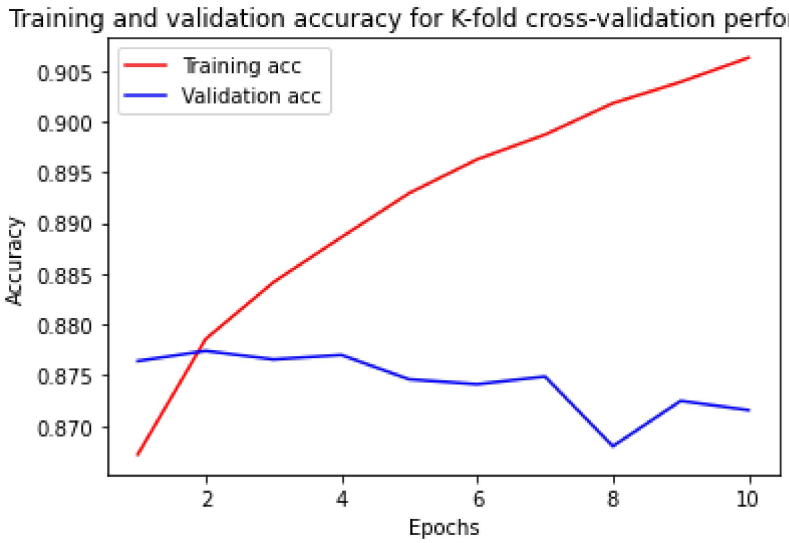
Out[17]:
```
[[0.5438806414604187, 0.7470399737358093],
 [0.5438806414604187, 0.7470399737358093],
 [0.5438806414604187, 0.7470399737358093],
 [0.5438806414604187, 0.7470399737358093],
 [0.5438806414604187, 0.7470399737358093],
 [0.5438806414604187, 0.7470399737358093],
 [0.5438806414604187, 0.7470399737358093],
 [0.5438806414604187, 0.7470399737358093],
 [0.5438806414604187, 0.7470399737358093],
 [0.5438806414604187, 0.7470399737358093]]
```

In [21]:
```
maxlen = 130
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = tf.keras.utils.pad_sequences(x_train, maxlen=maxlen)
x_test = tf.keras.utils.pad_sequences(x_test, maxlen=maxlen)

start_time_130 = time.time()
keras.backend.clear_session()
model2 = Sequential()
model2.add(Embedding(input_dim=max_features, output_dim=50, input_length=maxlen))
model2.add(SimpleRNN(units=5, kernel_initializer=keras.initializers.RandomNormal(stdde
                     activation='tanh'))
model2.add(Dense(1, activation='sigmoid'))

# Compile the model
model2.compile(loss='binary_crossentropy', optimizer=rmsprop, metrics=['accuracy'])

loss2 = []
accuracy2 = []
val_loss2 = []
val_accuracy2 = []
test_accuracy2 = []

for train, test in kfold.split(x_train, y_train):
    # Train the model
    history2 = model2.fit(x_train[train], y_train[train], epochs=10, batch_size=64, ca
              validation_data=(x_train[test], y_train[test]), verbose=0)
    loss2.append(history2.history['loss'])
    accuracy2.append(history2.history['accuracy'])
    val_loss2.append(history2.history['val_loss'])
    val_accuracy2.append(history2.history['val_accuracy'])
    # Evaluate the model
    scores = model2.evaluate(x_test, y_test, verbose=0)
    test_accuracy2.append(scores)
```

```
    print("Test Accuracy: %.2f%%" % (scores[1]*100))

end_time_130 = time.time()
runtime_130 = end_time_130 - start_time_130
print(runtime_130)
# plot the average loss and accuracy across all folds over the training epochs
loss2 = np.mean(loss2, axis=0)
accuracy2 = np.mean(accuracy2, axis=0)
val_loss2 = np.mean(val_loss2, axis=0)
val_accuracy2 = np.mean(val_accuracy2, axis=0)
epochs = range(1, len(loss2) + 1)

plt.figure()
plt.plot(epochs, loss2, 'r', label='Training loss')
plt.plot(epochs, val_loss2, 'b', label='Validation loss')
plt.title('Training and validation loss for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, accuracy2, 'r', label='Training acc')
plt.plot(epochs, val_accuracy2, 'b', label='Validation acc')
plt.title('Training and validation accuracy for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Test Accuracy: 75.16%
Test Accuracy: 81.98%
Test Accuracy: 83.94%
Test Accuracy: 83.97%
Test Accuracy: 83.56%
Test Accuracy: 83.17%
Test Accuracy: 82.90%
Test Accuracy: 82.44%
Test Accuracy: 82.14%
Test Accuracy: 82.18%
1350.3745930194855
```





Yes!, We improved the Performance of model

From the Above code, We are building our second model by varying the maximum length to 130 and padding the loaded imdb data. Similar to our First model, we saved all our losses and accuracies in lists and plotted graphs for them. By changing our maximum length to 130, Our model started to perform really well when compared to the Baseline and First models. The Validation accuracy reached a maximum of around 88%, while the training accuracy went till 90% depicting the model fits the data well. Coming to Testing Accuracy(Accuracy while performing on New unseen Data), It also improved on a great margin and reached around 84%, Which is really a good sign for the model. This is the Best model so far in our analysis. Comparing the Runtime, It is nearly doubled from 580 seconds to 1350 seconds, which is one of the drawback of the model.

Finally, By allowing more words to be included in each review, the model was able to pick up on more complex patterns and correlations in the data, resulting in higher validation and testing accuracies making it the best model built so far in this analysis. However, this improvement came at the cost of a longer runtime, which is an important factor to consider in the practical implementation of the model. Despite this drawback, the increase in maximum length demonstrates the importance of experimenting with different hyperparameters to find the optimal configuration for a given task.

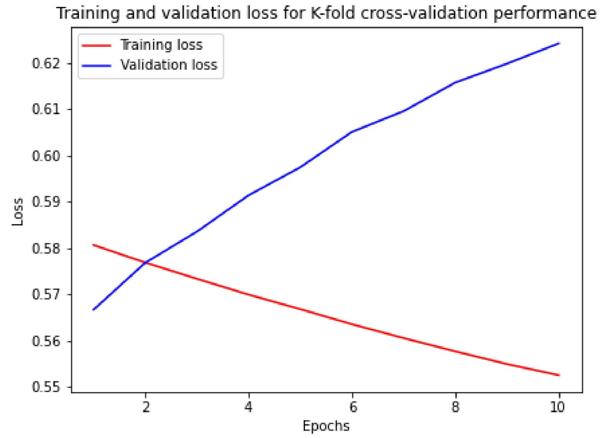
In [36]:
```
# Printing the list of test losses and test accuracies for each folds
test_accuracy2
```

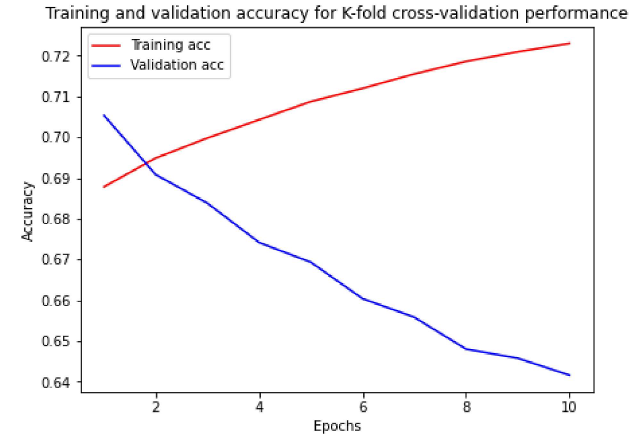
Out[36]:
```
[[0.5764103531837463, 0.7516400218009949],
 [0.44075337052345276, 0.8198400139808655],
 [0.3775266706943512, 0.8393599987030029],
 [0.374681293964386, 0.8397200107574463],
 [0.38884589076042175, 0.835640013217926],
 [0.4028087854385376, 0.8317199945449829],
 [0.4211269021034241, 0.8289999961853027],
 [0.43685901165008545, 0.8244400024414062],
 [0.4510191082954407, 0.8213599920272827],
 [0.4648720622062683, 0.8217599987983704]]
```

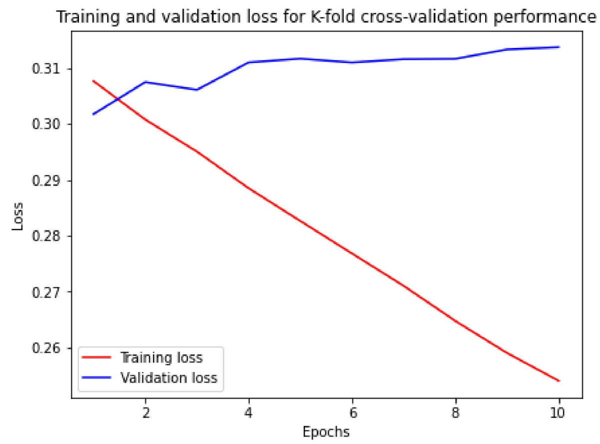
In [40]:
```
from keras.layers import SimpleRNN, Dense
from keras.models import Sequential
import time
maxlen = 130
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = tf.keras.utils.pad_sequences(x_train, maxlen=maxlen)
x_test = tf.keras.utils.pad_sequences(x_test, maxlen=maxlen)

hidden_dimensions = [4, 8, 16, 32, 64, 128]
time_comparison = []

for hidden_dim in hidden_dimensions:
    keras.backend.clear_session()
    start_time = time.time()
    model3 = Sequential()
    model3.add(Embedding(input_dim=max_features, output_dim=50, input_length=maxlen))
    model3.add(SimpleRNN(units=hidden_dim, kernel_initializer=keras.initializers.Rando
                         activation='tanh'))
    model3.add(Dense(1, activation='sigmoid'))
    # Compile the model
    model3.compile(loss='binary_crossentropy', optimizer=rmsprop, metrics=['accuracy']

    kfold = StratifiedKFold(n_splits=10, shuffle=True)

    loss3 = []
    accuracy3 = []
    val_loss3 = []
    val_accuracy3 = []
    test_accuracy3 = []


    for train, test in kfold.split(x_train, y_train):
        # Train the model
        history3 = model3.fit(x_train[train], y_train[train], epochs=10, batch_size=64
                              validation_data=(x_train[test], y_train[test]), verbose=
        loss3.append(history3.history['loss'])
        accuracy3.append(history3.history['accuracy'])
        val_loss3.append(history3.history['val_loss'])
        val_accuracy3.append(history3.history['val_accuracy'])
        # Evaluate the model
        scores = model3.evaluate(x_test, y_test, verbose=0)
        test_accuracy3.append(scores[1]*100)
        print("Test Accuracy: %.2f%%" % (scores[1]*100))

    end_time = time.time()
    runtime = end_time - start_time
    time_comparison.append(runtime)

    # plot the average loss and accuracy across all folds over the training epochs
    fig, axs = plt.subplots(1, 2, figsize=(15, 5))

    loss3 = np.mean(loss3, axis=0)
    val_loss3 = np.mean(val_loss3, axis=0)
    accuracy3 = np.mean(accuracy3, axis=0)
    val_accuracy3 = np.mean(val_accuracy3, axis=0)
    epochs = range(1, len(loss3) + 1)

    axs[0].plot(epochs, loss3, 'r', label='Training loss')
    axs[0].plot(epochs, val_loss3, 'b', label='Validation loss')
```

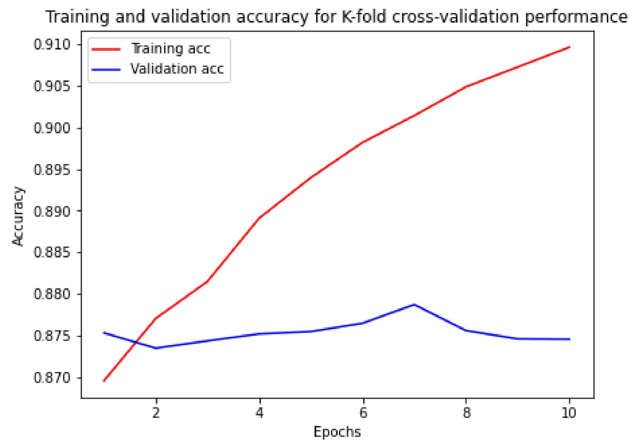
```
    axs[0].set_title('Training and validation loss for K-fold cross-validation perform
    axs[0].set_xlabel('Epochs')
    axs[0].set_ylabel('Loss')
    axs[0].legend()

    axs[1].plot(epochs, accuracy3, 'r', label='Training acc')
    axs[1].plot(epochs, val_accuracy3, 'b', label='Validation acc')
    axs[1].set_title('Training and validation accuracy for K-fold cross-validation per
    axs[1].set_xlabel('Epochs')
    axs[1].set_ylabel('Accuracy')
    axs[1].legend()
```

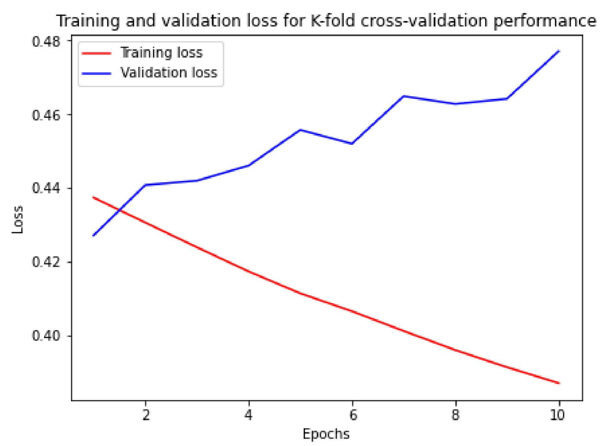
```
Test Accuracy: 51.36%
Test Accuracy: 50.39%
Test Accuracy: 51.15%
Test Accuracy: 51.45%
Test Accuracy: 52.44%
Test Accuracy: 52.96%
Test Accuracy: 53.16%
Test Accuracy: 53.59%
Test Accuracy: 53.37%
Test Accuracy: 53.59%
Test Accuracy: 80.12%
Test Accuracy: 84.70%
Test Accuracy: 84.30%
Test Accuracy: 84.36%
Test Accuracy: 83.91%
Test Accuracy: 83.66%
Test Accuracy: 82.91%
Test Accuracy: 82.98%
Test Accuracy: 82.90%
Test Accuracy: 82.39%
Test Accuracy: 64.65%
Test Accuracy: 65.52%
Test Accuracy: 69.90%
Test Accuracy: 70.30%
Test Accuracy: 72.84%
Test Accuracy: 75.42%
Test Accuracy: 76.22%
Test Accuracy: 76.45%
Test Accuracy: 74.85%
Test Accuracy: 75.61%
Test Accuracy: 81.72%
Test Accuracy: 83.70%
Test Accuracy: 84.87%
Test Accuracy: 84.40%
Test Accuracy: 82.90%
Test Accuracy: 82.64%
Test Accuracy: 82.34%
Test Accuracy: 81.47%
Test Accuracy: 82.02%
Test Accuracy: 81.60%
Test Accuracy: 81.66%
Test Accuracy: 83.88%
Test Accuracy: 83.52%
Test Accuracy: 83.37%
Test Accuracy: 82.62%
Test Accuracy: 82.11%
Test Accuracy: 80.81%
Test Accuracy: 79.34%
Test Accuracy: 79.68%
Test Accuracy: 77.34%
Test Accuracy: 80.59%
Test Accuracy: 81.84%
Test Accuracy: 82.07%
Test Accuracy: 80.16%
Test Accuracy: 79.98%
Test Accuracy: 76.80%
Test Accuracy: 80.72%
Test Accuracy: 75.37%
Test Accuracy: 78.14%
Test Accuracy: 77.76%
```

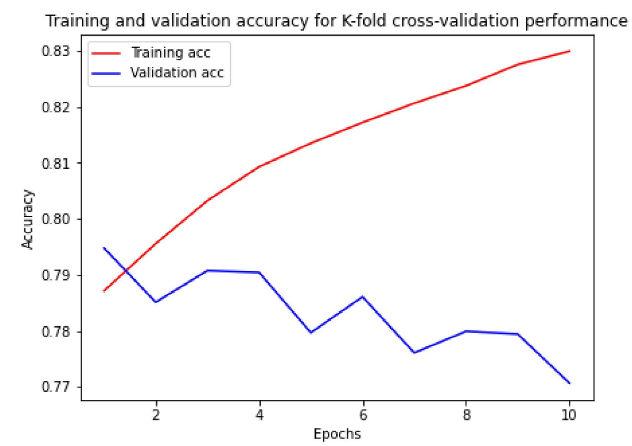

Training and validation loss for K-fold cross-validation performance
Training loss
Validation loss
Loss
Epochs

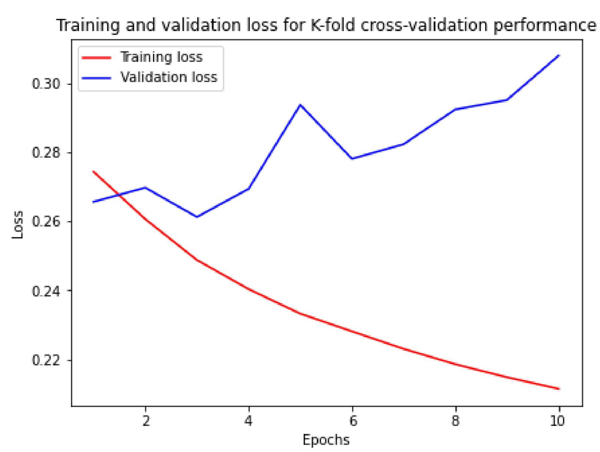

Training and validation accuracy for K-fold cross-validation performance
Training acc
Validation acc
Accuracy
Epochs

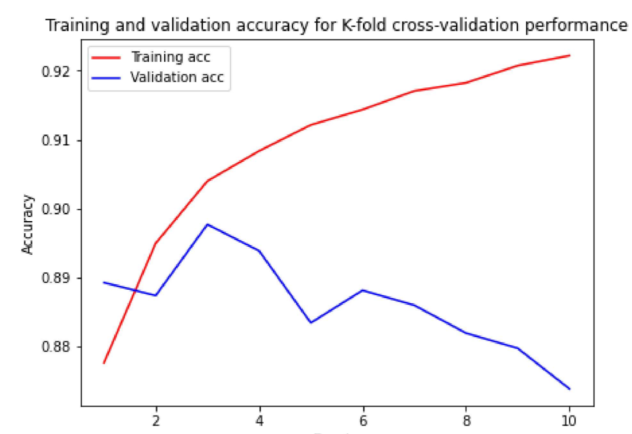

Training and validation loss for K-fold cross-validation performance
Training loss
Validation loss
Loss
Epochs

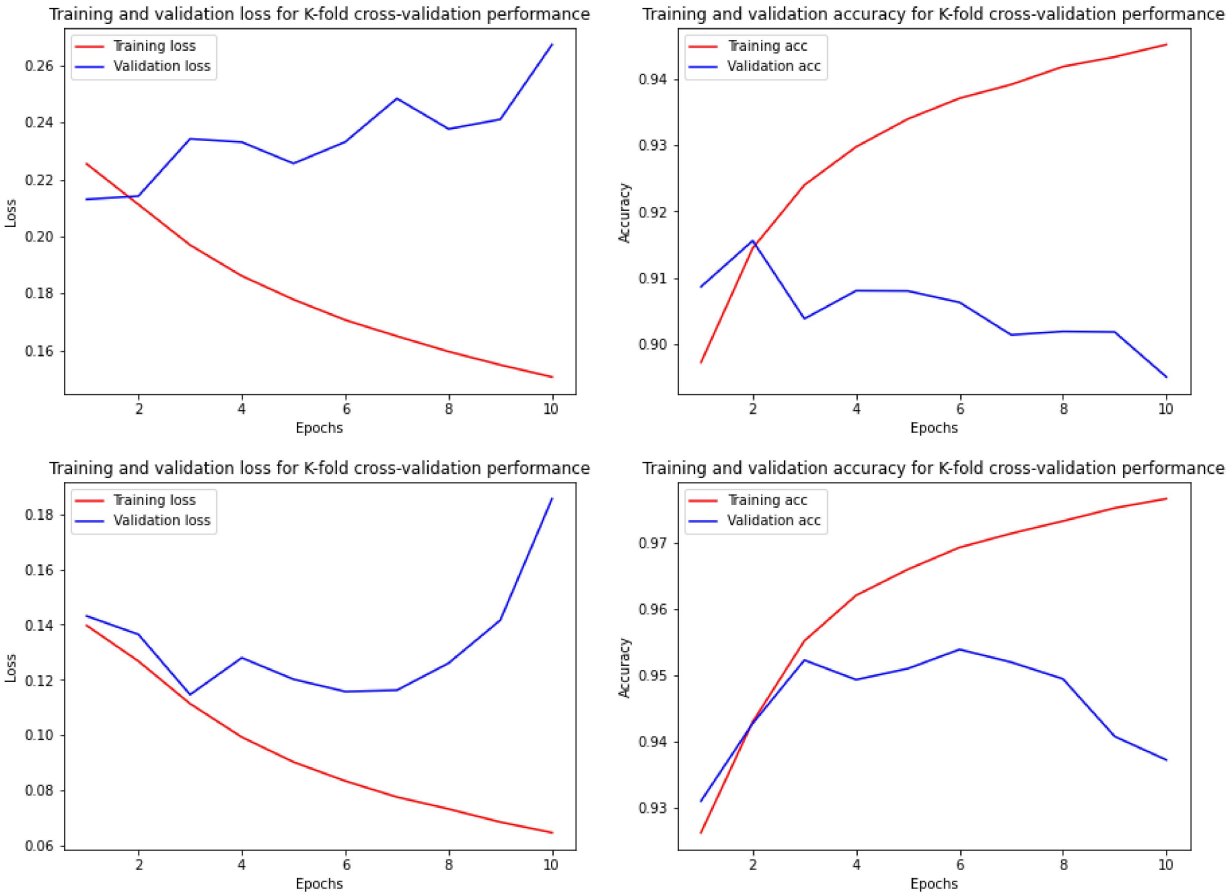

Training and validation accuracy for K-fold cross-validation performance
Training acc
Validation acc
Accuracy
Epochs


Training and validation loss for K-fold cross-validation performance
Training loss
Validation loss
Loss
Epochs


Training and validation accuracy for K-fold cross-validation performance
Training acc
Validation acc
Accuracy
Epochs


Training and validation loss for K-fold cross-validation performance
Training loss
Validation loss
Loss
Epochs


Training and validation accuracy for K-fold cross-validation performance
Training acc
Validation acc
Accuracy
Epochs



In [41]:
```
time_comparison
```

Out[41]:
```
[977.514979839325,
 1015.3498513698578,
 1053.0655736923218,
 1082.5691859722137,
 1063.7134582996368,
 2141.750696659088]
```

Yes, the Model Performance improved and runtime to train the model nearly increased by 55%. In the above Code, As we have tuned our best model using max length of 130, We are looking to play with the number of hidden dimensions in RNN. We looked to tune our best model by varying the hidden dimensions ranging from 4 to extending upto 128. We Performed our analysis and plotted graphs for the losses and accuracies of training and testing data over the k folds, which would make room for comparison between models easy. From the above analysis, Intially the accuracies dropped while doing analysis with 4 hidden dimensions and started performing well while increasing the number of hidden dimensions, but there is a point of diminishing returns. The best test accuracy achieved was 84.87% with 128 hidden dimensions, while the worst was 50.39% with only 8 hidden dimensions. Coming to the compariosn in view of Runtime, As we increase the number of hidden layers the training time also increases but point worth of noting is for the 128 hidden dimensions the model performance increased well but the training time nearly doubled when compared to previous steps, which grabs the attention during the analysis. For the model having 128 hidden dimensions, the training and validation accuracies reached the range of 93% to 97%, which is best so far and the losses ranged in between 0.06 to 0.18, relatively low when compared to other models. With this

analysis, We can state that this model can capture complex patterns hidden in the data and perform well on the new unseen data with a relatively high accuracies.

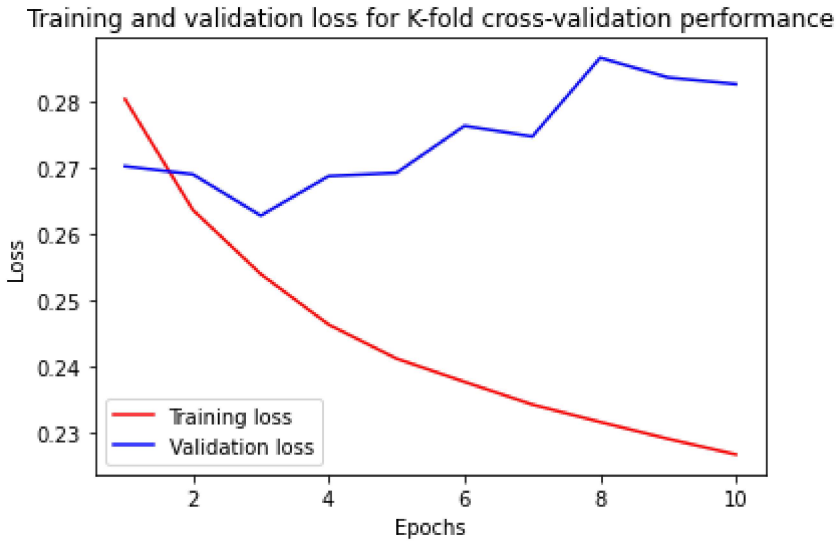
Train LSTM and GRU networks

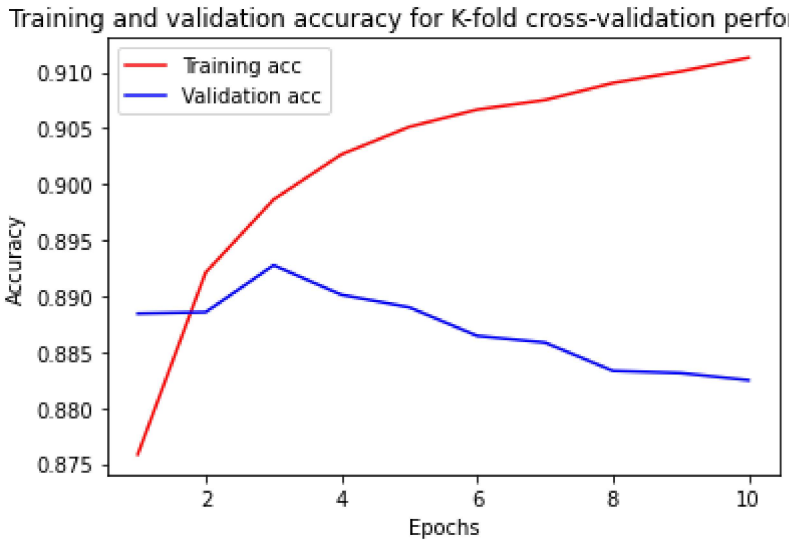
15- Build LSTM and GRU networks and compare their performance (accuracy and execution time) with the SimpleRNN. What is your conclusion?

In [42]:
```
#LSTM Model
from keras.layers import LSTM, GRU, Dense
from keras.models import Sequential

maxlen = 130
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = tf.keras.utils.pad_sequences(x_train, maxlen=maxlen)
x_test = tf.keras.utils.pad_sequences(x_test, maxlen=maxlen)

start_time_lstm = time.time()
keras.backend.clear_session()
model_lstm = Sequential()

model_lstm.add(Embedding(input_dim=max_features, output_dim=50, input_length=maxlen))
model_lstm.add(LSTM(units=128, kernel_initializer=keras.initializers.RandomNormal(stdc
                    activation='tanh'))
model_lstm.add(Dense(1, activation='sigmoid'))

# Compile the model
model_lstm.compile(loss='binary_crossentropy', optimizer=rmsprop, metrics=['accuracy']

loss_lstm = []
accuracy_lstm = []
val_loss_lstm = []
val_accuracy_lstm = []
test_accuracy_lstm = []

for train, test in kfold.split(x_train, y_train):
    # Train the model
    history_lstm = model_lstm.fit(x_train[train], y_train[train], epochs=10, batch_siz
              validation_data=(x_train[test], y_train[test]), verbose = 0)
    loss_lstm.append(history_lstm.history['loss'])
    accuracy_lstm.append(history_lstm.history['accuracy'])
    val_loss_lstm.append(history_lstm.history['val_loss'])
    val_accuracy_lstm.append(history_lstm.history['val_accuracy'])
    # Evaluate the model
    scores = model_lstm.evaluate(x_test, y_test, verbose=0)
    test_accuracy_lstm.append(scores)
    print("Accuracy: %.2f%%" % (scores[1]*100))

end_time_lstm = time.time()
runtime_lstm = end_time_lstm - start_time_lstm
print(runtime_lstm)

# plot the average loss and accuracy across all folds over the training epochs
loss_lstm = np.mean(loss_lstm, axis=0)
accuracy_lstm = np.mean(accuracy_lstm, axis=0)
val_loss_lstm = np.mean(val_loss_lstm, axis=0)
```

```
val_accuracy_lstm = np.mean(val_accuracy_lstm, axis=0)
epochs = range(1, len(loss_lstm) + 1)

plt.figure()
plt.plot(epochs, loss_lstm, 'r', label='Training loss')
plt.plot(epochs, val_loss_lstm, 'b', label='Validation loss')
plt.title('Training and validation loss for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, accuracy_lstm, 'r', label='Training acc')
plt.plot(epochs, val_accuracy_lstm, 'b', label='Validation acc')
plt.title('Training and validation accuracy for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Accuracy: 85.55%
Accuracy: 85.20%
Accuracy: 85.40%
Accuracy: 84.87%
Accuracy: 85.49%
Accuracy: 85.23%
Accuracy: 85.23%
Accuracy: 85.06%
Accuracy: 84.82%
Accuracy: 84.46%
7279.7962148189545
```



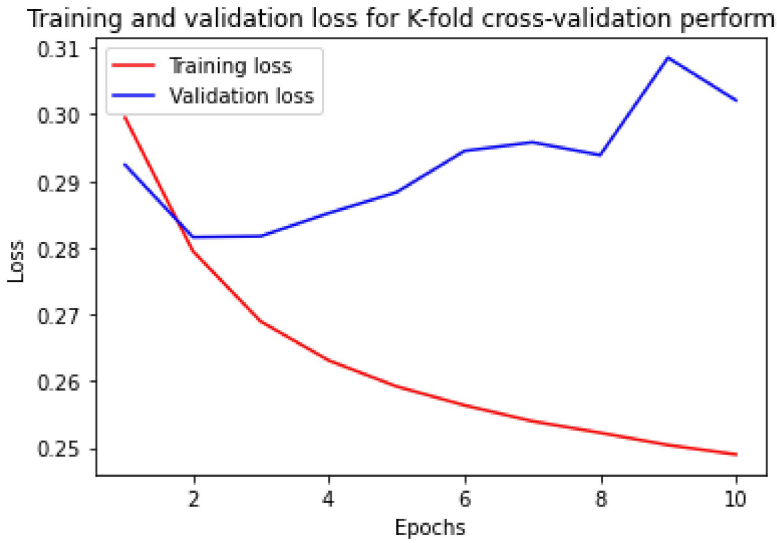


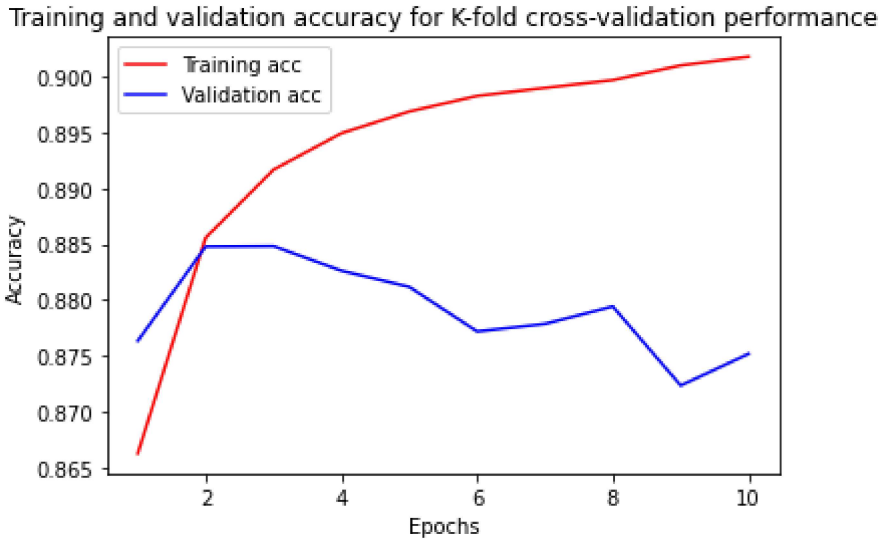
After our analysis with SimpleRNN done, We started to explore different other models namely LSTM and GRU. Firstly, We did our analysis training our model with LSTM and plotting the losses and accuracy of it. The time it took to train is around 7300 seconds, which is more than 2 hours, relatively so much more compared to SimpleRNN which turned out to be major drawback of the LSTM model. While coming down to accuracy and losses, the validation and training accuracy ranged upto 88% - 91% and testing accuracy ranged upto 85%, which is lower than our simpleRNN model. So, We can conclude the by changing to LSTM from SimpleRNN it does not improve the performance much but the training time took so long when compared to other for capturing the same hidden patterns in the data.

In [43]:
```
#GRU Model
from keras.layers import GRU, Dense
from keras.models import Sequential

maxlen = 120
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = tf.keras.utils.pad_sequences(x_train, maxlen=maxlen)
x_test = tf.keras.utils.pad_sequences(x_test, maxlen=maxlen)

start_time_gru = time.time()
keras.backend.clear_session()
model_gru = Sequential()

model_gru.add(Embedding(input_dim=max_features, output_dim=50, input_length=maxlen))
model_gru.add(GRU(units=128, kernel_initializer=keras.initializers.RandomNormal(stddev
                  activation='tanh'))
model_gru.add(Dense(1, activation='sigmoid'))

# Compile the model
model_gru.compile(loss='binary_crossentropy', optimizer=rmsprop, metrics=['accuracy'])

loss_gru = []
accuracy_gru = []
val_loss_gru = []
val_accuracy_gru = []
test_accuracy_gru = []

for train, test in kfold.split(x_train, y_train):
```

```
    # Train the model
    history_gru = model_gru.fit(x_train[train], y_train[train], epochs=10, batch_size=
              validation_data=(x_train[test], y_train[test]), verbose=0 )
    loss_gru.append(history_gru.history['loss'])
    accuracy_gru.append(history_gru.history['accuracy'])
    val_loss_gru.append(history_gru.history['val_loss'])
    val_accuracy_gru.append(history_gru.history['val_accuracy'])
    # Evaluate the model
    scores = model_gru.evaluate(x_test, y_test, verbose=0)
    test_accuracy_gru.append(scores)
    print("Test Accuracy: %.2f%%" % (scores[1]*100))

end_time_gru = time.time()
runtime_gru = end_time_gru - start_time_gru
print(runtime_gru)

# plot the average loss and accuracy across all folds over the training epochs
loss_gru = np.mean(loss_gru, axis=0)
accuracy_gru = np.mean(accuracy_gru, axis=0)
val_loss_gru = np.mean(val_loss_gru, axis=0)
val_accuracy_gru = np.mean(val_accuracy_gru, axis=0)
epochs = range(1, len(loss_gru) + 1)

plt.figure()
plt.plot(epochs, loss_gru, 'r', label='Training loss')
plt.plot(epochs, val_loss_gru, 'b', label='Validation loss')
plt.title('Training and validation loss for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, accuracy_gru, 'r', label='Training acc')
plt.plot(epochs, val_accuracy_gru, 'b', label='Validation acc')
plt.title('Training and validation accuracy for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Test Accuracy: 85.13%
Test Accuracy: 84.98%
Test Accuracy: 84.75%
Test Accuracy: 85.04%
Test Accuracy: 84.91%
Test Accuracy: 85.18%
Test Accuracy: 85.74%
Test Accuracy: 85.60%
Test Accuracy: 85.33%
Test Accuracy: 85.24%
6126.137657165527
```





After our analysis with SimpleRNN, LSTM done, We started to explore other models namely GRU. Firstly, We did our analysis training our model with GRU and plotting the losses and accuracy of it. The time it took to train is around 6200 seconds, which is around than 2 hours, relatively so much more compared to SimpleRNN and relatively low compared to LSTM ,which turned out to be major drawback of the LSTM model. While coming down to accuracy and losses, the validation and training accuracy ranged upto 87% - 90% and testing accuracy ranged upto 85%, which is lower than our simpleRNN model. So, We can conclude the by changing to GRU from SimpleRNN, LSTM it does not improve the performance much but the training time took so long when compared to other for capturing the same hidden patterns in the data.

In [48]:
```
#Saving the best model weights into a binary file.
path = 'C:/Users/shiva/Desktop/Assignment_3_RNN/Best_val_accuracy_model_Weights'
model3.save_weights(path)
```

Our Best Model so far discovered in the analysis is Model with SimpleRNN architecture with 128 hidden dimensions and max length to be padded is 130 which is our "model3".For this model, the training and validation accuracies reached the range of 93% to 97%, which is best so far and the losses ranged in between 0.06 to 0.18, relatively low and best when compared to other models

16- Save the weights of the best model to the local drive.

Limitation :

In our Analysis using jupyter notebook with Tensorboard, There are few issues with TensorBoard, It is not refreshing while changing the directory and if we use the refresh option available in it, it is stuck in the previous directory and not loading the new directory even after modifying the command. So, I went ahead plotting graphs using matplotlibm but i have used the interactive dashboard given by Tensorboard to play with scalars and values, it is a great technique for analysation of variations.

Bonus

17- Instead of word tokenization, tokonize the reviews based on characters and build LSTM and GRU networks, and compare their performance with respect to word based tokenization.

In [45]:
```
from keras.preprocessing.text import Tokenizer

maxlen = 130
max_features = 2000

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.imdb.load_data(num_words=max_

# Tokenize reviews based on characters
tokenizer = Tokenizer(char_level=True)
tokenizer.fit_on_texts([text.lower() for text in tf.keras.datasets.imdb.get_word_index
x_train = tokenizer.texts_to_sequences([' '.join([tokenizer.index_word.get(i - 3, '')
x_test = tokenizer.texts_to_sequences([' '.join([tokenizer.index_word.get(i - 3, '') f
x_train = tf.keras.utils.pad_sequences(x_train, maxlen=maxlen)
x_test = tf.keras.utils.pad_sequences(x_test, maxlen=maxlen)
```

In [47]:
```
#Let's Define the Model
start_time_t_lstm = time.time()
tf.keras.backend.clear_session()
model_t_lstm = Sequential()
model_t_lstm.add(Embedding(input_dim=max_features, output_dim=128, input_length=maxlen
model_t_lstm.add(LSTM(units=128, kernel_initializer=tf.keras.initializers.RandomNormal
model_t_lstm.add(Dense(1, activation='sigmoid'))

# Compile the model
model_t_lstm.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accura

loss_t_lstm = []
accuracy_t_lstm = []
val_loss_t_lstm = []
val_accuracy_t_lstm = []
test_accuracy_t_lstm = []

for train, test in kfold.split(x_train, y_train):
    # Train the model
    history_t_lstm = model_t_lstm.fit(x_train[train], y_train[train], epochs=10, batch
                        validation_data=(x_train[test], y_train[test]), verbose=0)
    loss_t_lstm.append(history.history['loss'])
```

```
    accuracy_t_lstm.append(history.history['accuracy'])
    val_loss_t_lstm.append(history.history['val_loss'])
    val_accuracy_t_lstm.append(history.history['val_accuracy'])
    # Evaluate the model
    scores = model_t_lstm.evaluate(x_test, y_test, verbose=0)
    test_accuracy_t_lstm.append(scores[1]*100)
    print("Test Accuracy: %.2f%%" % (scores[1]*100))

end_time_t_lstm = time.time()
runtime_t_lstm = end_time_t_lstm - start_time_t_lstm
print(runtime_t_lstm)

# plot the average loss and accuracy across all folds over the training epochs
loss_t_lstm = np.mean(loss_t_lstm, axis=0)
accuracy_t_lstm = np.mean(accuracy_t_lstm, axis=0)
val_loss_t_lstm = np.mean(val_loss_t_lstm, axis=0)
val_accuracy_t_lstm = np.mean(val_accuracy_t_lstm, axis=0)
epochs = range(1, len(loss_t_lstm) + 1)

plt.figure()
plt.plot(epochs, loss_t_lstm, 'r', label='Training loss')
plt.plot(epochs, val_loss_t_lstm, 'b', label='Validation loss')
plt.title('Training and validation loss for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, accuracy_t_lstm, 'r', label='Training acc')
plt.plot(epochs, val_accuracy_t_lstm, 'b', label='Validation acc')
plt.title('Training and validation accuracy for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
Test Accuracy: 73.08%
Test Accuracy: 75.23%
Test Accuracy: 75.25%
Test Accuracy: 73.67%
Test Accuracy: 72.40%
Test Accuracy: 71.86%
Test Accuracy: 71.36%
Test Accuracy: 71.85%
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [47], in <cell line: 18>()
     16 test_accuracy_t_lstm = []
     18 for train, test in kfold.split(x_train, y_train):
     19     # Train the model
---> 20     history_t_lstm = model_t_lstm.fit(x_train[train], y_train[train], epochs=
10, batch_size=64, callbacks=[tensorboard],
     21                         validation_data=(x_train[test], y_train[test]), verbo
se=0)
     22     loss_t_lstm.append(history.history['loss'])
     23     accuracy_t_lstm.append(history.history['accuracy'])

File ~\anaconda3\lib\site-packages\keras\utils\traceback_utils.py:65, in filter_trace
back.<locals>.error_handler(*args, **kwargs)
     63 filtered_tb = None
     64 try:
---> 65     return fn(*args, **kwargs)
     66 except Exception as e:
     67     filtered_tb = _process_traceback_frames(e.__traceback__)

File ~\anaconda3\lib\site-packages\keras\engine\training.py:1650, in Model.fit(self,
x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_data, shuf
fle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, v
alidation_batch_size, validation_freq, max_queue_size, workers, use_multiprocessing)
   1642 with tf.profiler.experimental.Trace(
   1643     "train",
   1644     epoch_num=epoch,
   (...)
   1647     _r=1,
   1648 ):
   1649     callbacks.on_train_batch_begin(step)
-> 1650     tmp_logs = self.train_function(iterator)
   1651     if data_handler.should_sync:
   1652         context.async_wait()

File ~\anaconda3\lib\site-packages\tensorflow\python\util\traceback_utils.py:150, in
filter_traceback.<locals>.error_handler(*args, **kwargs)
    148 filtered_tb = None
    149 try:
--> 150   return fn(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)

File ~\anaconda3\lib\site-packages\tensorflow\python\eager\polymorphic_function\polym
orphic_function.py:880, in Function.__call__(self, *args, **kwds)
    877 compiler = "xla" if self._jit_compile else "nonXla"
    879 with OptionalXlaContext(self._jit_compile):
--> 880   result = self._call(*args, **kwds)
    882 new_tracing_count = self.experimental_get_tracing_count()
    883 without_tracing = (tracing_count == new_tracing_count)

File ~\anaconda3\lib\site-packages\tensorflow\python\eager\polymorphic_function\polym
orphic_function.py:912, in Function._call(self, *args, **kwds)
    909   self._lock.release()
    910   # In this case we have created variables on the first call, so we run the
    911   # defunned version which is guaranteed to never create variables.
--> 912   return self._no_variable_creation_fn(*args, **kwds)  # pylint: disable=not-
callable
    913 elif self._variable_creation_fn is not None:
```

```
    914   # Release the lock early so that multiple threads can perform the call
    915   # in parallel.
    916   self._lock.release()

File ~\anaconda3\lib\site-packages\tensorflow\python\eager\polymorphic_function\tracing_compiler.py:134, in TracingCompiler.__call__(self, *args, **kwargs)
    131 with self._lock:
    132   (concrete_function,
    133    filtered_flat_args) = self._maybe_define_function(args, kwargs)
--> 134 return concrete_function._call_flat(
    135     filtered_flat_args, captured_inputs=concrete_function.captured_inputs)

File ~\anaconda3\lib\site-packages\tensorflow\python\eager\polymorphic_function\monomorphic_function.py:1745, in ConcreteFunction._call_flat(self, args, captured_inputs, cancellation_manager)
   1741 possible_gradient_type = gradients_util.PossibleTapeGradientTypes(args)
   1742 if (possible_gradient_type == gradients_util.POSSIBLE_GRADIENT_TYPES_NONE
   1743     and executing_eagerly):
   1744   # No tape is watching; skip to running the function.
-> 1745   return self._build_call_outputs(self._inference_function.call(
   1746       ctx, args, cancellation_manager=cancellation_manager))
   1747 forward_backward = self._select_forward_and_backward_functions(
   1748     args,
   1749     possible_gradient_type,
   1750     executing_eagerly)
   1751 forward_function, args_with_tangents = forward_backward.forward()

File ~\anaconda3\lib\site-packages\tensorflow\python\eager\polymorphic_function\monomorphic_function.py:378, in _EagerDefinedFunction.call(self, ctx, args, cancellation_manager)
    376 with _InterpolateFunctionError(self):
    377   if cancellation_manager is None:
--> 378     outputs = execute.execute(
    379         str(self.signature.name),
    380         num_outputs=self._num_outputs,
    381         inputs=args,
    382         attrs=attrs,
    383         ctx=ctx)
    384   else:
    385     outputs = execute.execute_with_cancellation(
    386         str(self.signature.name),
    387         num_outputs=self._num_outputs,
   (...)
    390         ctx=ctx,
    391         cancellation_manager=cancellation_manager)

File ~\anaconda3\lib\site-packages\tensorflow\python\eager\execute.py:52, in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     50 try:
     51   ctx.ensure_initialized()
---> 52   tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     53                                       inputs, attrs, num_outputs)
     54 except core._NotOkStatusException as e:
     55   if name is not None:

KeyboardInterrupt:
```

In [ ]:

```
#Let's Define the Model
start_time_t_gru = time.time()
tf.keras.backend.clear_session()
```

```
model_t_gru = Sequential()
model_t_gru.add(Embedding(input_dim=max_features, output_dim=128, input_length=maxlen)
model_t_gru.add(LSTM(units=128, kernel_initializer=tf.keras.initializers.RandomNormal(
model_t_gru.add(Dense(1, activation='sigmoid'))

# Compile the model
model_t_gru.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accurac

loss_t_gru = []
accuracy_t_gru = []
val_loss_t_gru = []
val_accuracy_t_gru = []
test_accuracy_t_gru = []

for train, test in kfold.split(x_train, y_train):
    # Train the model
    history_t_gru = model_t_gru.fit(x_train[train], y_train[train], epochs=10, batch_s
                        validation_data=(x_train[test], y_train[test]), verbose=0)
    loss_t_gru.append(history.history['loss'])
    accuracy_t_gru.append(history.history['accuracy'])
    val_loss_t_gru.append(history.history['val_loss'])
    val_accuracy_t_gru.append(history.history['val_accuracy'])
    # Evaluate the model
    scores = model_t_gru.evaluate(x_test, y_test, verbose=0)
    test_accuracy_t_gru.append(scores[1]*100)
    print("Test Accuracy: %.2f%%" % (scores[1]*100))

end_time_t_gru = time.time()
runtime_t_gru = end_time_t_gru - start_time_t_gru
print(runtime_t_gru)

# plot the average loss and accuracy across all folds over the training epochs
loss_t_gru = np.mean(loss_t_gru, axis=0)
accuracy_t_gru = np.mean(accuracy_t_gru, axis=0)
val_loss_t_gru = np.mean(val_loss_t_gru, axis=0)
val_accuracy_t_gru = np.mean(val_accuracy_t_gru, axis=0)
epochs = range(1, len(loss_t_gru) + 1)

plt.figure()
plt.plot(epochs, loss_t_gru, 'r', label='Training loss')
plt.plot(epochs, val_loss_t_gru, 'b', label='Validation loss')
plt.title('Training and validation loss for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, accuracy_t_gru, 'r', label='Training acc')
plt.plot(epochs, val_accuracy_t_gru, 'b', label='Validation acc')
plt.title('Training and validation accuracy for K-fold cross-validation performance')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```