BAN200 Week 08 Homework

To complete the homework you will need to modify this template by adding Python code and/or text.

Before starting the homework, make sure to save a copy of this template to your personal Google Drive. If you haven't saved your own copy, any changes you make will be lost when you close your browser window.

To submit your homework: go to "File" in the Colab menu bar > select "Download" > select "Download .ipynb". This will download a ".ipynb" file to your computer. You must submit this file.

The homework is to be completed in groups. It is due at the start of next class.

Homework is graded on the following scale:

- *100%* -- The assignment was submitted on time, any code runs without errors, and every question is answered correctly.
- *80%* -- The assignment was submitted on time, any code runs without errors, and every question is answered. Some questions may be incorrect, but the submission demonstrates an average level of effort and average level of understanding of the material.
- *60%* -- The submission demonstrates a below-average level of effort and below-average level of understanding of the material. This is the highest grade that should be given to submissions that are submitted late, have code that throws uncaught errors, or leave some questions unanswered.
- *0%* -- No assignment was submitted, or the submission demonstrates little-to-no effort and little-to-no understanding of the material.

IMDB Dataset

For the homwork, we will be using the Large Movie Review Dataset that we used in class. The code below downloads the data, unzips it, and uses it to create Keras dataset objects following the same procedure used in class.

```
!wget 'https://storage.googleapis.com/wd13/imdb_data.zip'
```

```
--2023-11-07 16:57:52--  https://storage.googleapis.com/wd13/imdb_data.zip
Resolving storage.googleapis.com (storage.googleapis.com)... 142.251.16.207, 172.253.62.207, 172.253.115.207, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|142.251.16.207|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 64200274 (61M) [application/zip]
Saving to: ‘imdb_data.zip’

imdb_data.zip       100%[===================>]  61.23M   126MB/s    in 0.5s

2023-11-07 16:57:52 (126 MB/s) - ‘imdb_data.zip’ saved [64200274/64200274]
```

```
!unzip "imdb_data.zip"
```

```
        inflating: __MACOSX/imdb_data/train/pos/._9974_8.txt
        inflating: imdb_data/train/pos/4641_9.txt
        inflating: __MACOSX/imdb_data/train/pos/._4641_9.txt
        inflating: imdb_data/train/pos/7445_7.txt
        inflating: __MACOSX/imdb_data/train/pos/._7445_7.txt
        inflating: imdb_data/train/pos/4479_8.txt
        inflating: __MACOSX/imdb_data/train/pos/._4479_8.txt
        inflating: imdb_data/train/pos/6665_10.txt
        inflating: __MACOSX/imdb_data/train/pos/._6665_10.txt
        inflating: imdb_data/train/pos/5733_7.txt
        inflating: __MACOSX/imdb_data/train/pos/._5733_7.txt
        inflating: imdb_data/train/pos/7352_10.txt
        inflating: __MACOSX/imdb_data/train/pos/._7352_10.txt
        inflating: imdb_data/train/pos/3169_8.txt
        inflating: __MACOSX/imdb_data/train/pos/._3169_8.txt
        inflating: imdb_data/train/pos/4290_9.txt
        inflating: __MACOSX/imdb_data/train/pos/._4290_9.txt
        inflating: imdb_data/train/pos/3266_7.txt
        inflating: __MACOSX/imdb_data/train/pos/._3266_7.txt
        inflating: imdb_data/train/pos/8561_10.txt
        inflating: __MACOSX/imdb_data/train/pos/._8561_10.txt
        inflating: imdb_data/train/pos/8230_10.txt
        inflating: __MACOSX/imdb_data/train/pos/._8230_10.txt
        inflating: imdb_data/train/pos/9707_10.txt
        inflating: __MACOSX/imdb_data/train/pos/._9707_10.txt
        inflating: imdb_data/train/pos/35_8.txt
        inflating: __MACOSX/imdb_data/train/pos/._35_8.txt
        inflating: imdb_data/train/pos/6034_10.txt
        inflating: __MACOSX/imdb_data/train/pos/._6034_10.txt
        inflating: imdb_data/train/pos/2780_9.txt
        inflating: __MACOSX/imdb_data/train/pos/._2780_9.txt
        inflating: imdb_data/train/pos/437_9.txt
        inflating:   MACOSX/imdb_data/train/pos/._437_9.txt
```

```
import tensorflow as tf
import keras


train_dataset, validation_dataset = tf.keras.utils.text_dataset_from_directory(
    "imdb_data/train",
    validation_split=0.2,
    subset="both",
    seed=13
)
```

```
 Found 25000 files belonging to 2 classes.
 Using 20000 files for training.
 Using 5000 files for validation.
```

```
test_dataset = tf.keras.utils.text_dataset_from_directory(
    "imdb_data/test",
    seed=13
)
```

```
 Found 25000 files belonging to 2 classes.
```

Question 1

Create a Keras `TextVectorization` object called `vectorizer` that represents text as a 5000-word document-term-matrix.

```
# put your answer here
vectorizer = keras.layers.TextVectorization(
    output_mode='multi_hot',
    max_tokens=5000
)
```

Question 2

Adapt `vectorizer` using the training data.

```
# put your answer here
train_text = train_dataset.map(lambda x,y:x)
```

```
vectorizer.adapt(train_text)
```

Question 3

Create a single-layer, logistic regression model to predict whether a movie review is positive or negative. Use the same method used in class, including an early stop.

```
# put your answer here
input_layer = keras.Input(shape=(1,),dtype=tf.string)


vectorize_layer = vectorizer(input_layer)


logistic_regression = keras.layers.Dense(1, activation='sigmoid')(vectorize_layer)


model = keras.Model(inputs=input_layer, outputs=logistic_regression)


model.compile(optimizer='adam', loss ='binary_crossentropy', metrics=["accuracy"])


model.summary()
```

```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 1)]               0

 text_vectorization_1 (Text  (None, 5000)              0
 Vectorization)

 dense_9 (Dense)             (None, 1)                 5001

=================================================================
Total params: 5001 (19.54 KB)
Trainable params: 5001 (19.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = model.fit(
    train_dataset,
    validation_data = validation_dataset,
    epochs=20
)
```

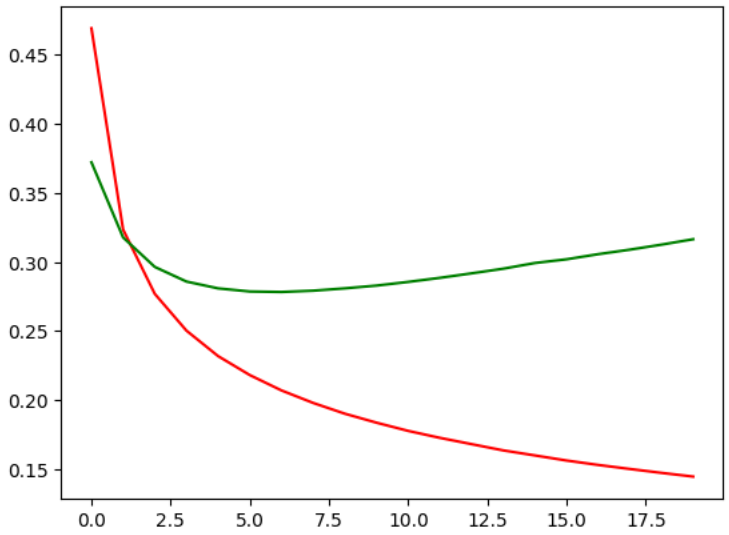
```
Epoch 1/20
625/625 [==============================] - 5s 7ms/step - loss: 0.4692 - accuracy: 0.8250 - val_loss: 0.3723 - val_accuracy:
Epoch 2/20
625/625 [==============================] - 5s 8ms/step - loss: 0.3235 - accuracy: 0.8884 - val_loss: 0.3178 - val_accuracy:
Epoch 3/20
625/625 [==============================] - 6s 10ms/step - loss: 0.2770 - accuracy: 0.9014 - val_loss: 0.2965 - val_accuracy:
Epoch 4/20
625/625 [==============================] - 5s 8ms/step - loss: 0.2504 - accuracy: 0.9117 - val_loss: 0.2859 - val_accuracy:
Epoch 5/20
625/625 [==============================] - 6s 10ms/step - loss: 0.2319 - accuracy: 0.9187 - val_loss: 0.2810 - val_accuracy:
Epoch 6/20
625/625 [==============================] - 5s 8ms/step - loss: 0.2182 - accuracy: 0.9237 - val_loss: 0.2787 - val_accuracy:
Epoch 7/20
625/625 [==============================] - 5s 8ms/step - loss: 0.2071 - accuracy: 0.9277 - val_loss: 0.2784 - val_accuracy:
Epoch 8/20
625/625 [==============================] - 6s 10ms/step - loss: 0.1981 - accuracy: 0.9319 - val_loss: 0.2793 - val_accuracy:
Epoch 9/20
625/625 [==============================] - 6s 10ms/step - loss: 0.1903 - accuracy: 0.9353 - val_loss: 0.2810 - val_accuracy:
Epoch 10/20
625/625 [==============================] - 5s 7ms/step - loss: 0.1838 - accuracy: 0.9374 - val_loss: 0.2830 - val_accuracy:
Epoch 11/20
625/625 [==============================] - 6s 9ms/step - loss: 0.1778 - accuracy: 0.9398 - val_loss: 0.2857 - val_accuracy:
Epoch 12/20
625/625 [==============================] - 5s 8ms/step - loss: 0.1728 - accuracy: 0.9418 - val_loss: 0.2886 - val_accuracy:
Epoch 13/20
625/625 [==============================] - 7s 10ms/step - loss: 0.1682 - accuracy: 0.9434 - val_loss: 0.2919 - val_accuracy:
Epoch 14/20
625/625 [==============================] - 5s 8ms/step - loss: 0.1637 - accuracy: 0.9449 - val_loss: 0.2952 - val_accuracy:
Epoch 15/20
625/625 [==============================] - 6s 9ms/step - loss: 0.1601 - accuracy: 0.9468 - val_loss: 0.2993 - val_accuracy:
```

```
Epoch 16/20
625/625 [==============================] - 5s 9ms/step - loss: 0.1564 - accuracy: 0.9481 - val_loss: 0.3020 - val_accuracy:
Epoch 17/20
625/625 [==============================] - 5s 7ms/step - loss: 0.1532 - accuracy: 0.9495 - val_loss: 0.3057 - val_accuracy:
Epoch 18/20
625/625 [==============================] - 7s 10ms/step - loss: 0.1503 - accuracy: 0.9499 - val_loss: 0.3090 - val_accuracy:
Epoch 19/20
625/625 [==============================] - 6s 10ms/step - loss: 0.1474 - accuracy: 0.9513 - val_loss: 0.3126 - val_accuracy:
Epoch 20/20
625/625 [==============================] - 5s 7ms/step - loss: 0.1447 - accuracy: 0.9525 - val_loss: 0.3166 - val_accuracy:
```

```
import matplotlib.pyplot as plt
```

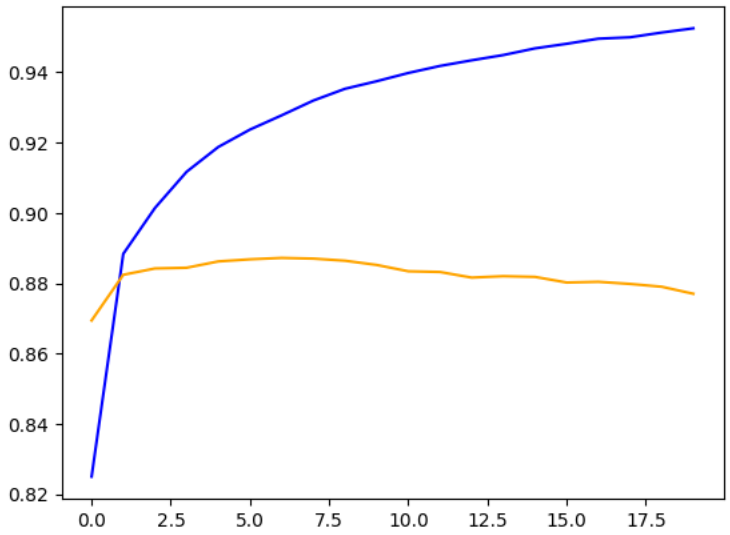
```
plt.plot(history.history['loss'], color='red')
plt.plot(history.history['val_loss'], color = 'green')
```

```
[<matplotlib.lines.Line2D at 0x7adcb5cbb370>]
```




```
#accuracy plot
plt.plot(history.history['accuracy'], color='blue')
plt.plot(history.history['val_accuracy'], color = 'orange')
```

```
[<matplotlib.lines.Line2D at 0x7adca4f5f4c0>]
```




```
#rebuild model with early stopping
input_layer = keras.Input(shape=(1,),dtype=tf.string)
vectorize_layer = vectorizer(input_layer)
logistic_regression = keras.layers.Dense(1,activation='sigmoid')(vectorize_layer)
model = keras.Model(inputs= input_layer, outputs= logistic_regression)
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
history = model.fit(
    train_dataset,
```

```
validation_data = validation_dataset,
epochs=20,
callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',patience=3,restore_best_weights=True)])
```

```
Epoch 1/20
625/625 [==============================] - 6s 8ms/step - loss: 0.4729 - accuracy: 0.8212 - val_loss: 0.3730 - val_accuracy:
Epoch 2/20
625/625 [==============================] - 5s 8ms/step - loss: 0.3246 - accuracy: 0.8862 - val_loss: 0.3178 - val_accuracy:
Epoch 3/20
625/625 [==============================] - 7s 11ms/step - loss: 0.2774 - accuracy: 0.9025 - val_loss: 0.2963 - val_accuracy:
Epoch 4/20
625/625 [==============================] - 5s 8ms/step - loss: 0.2507 - accuracy: 0.9115 - val_loss: 0.2856 - val_accuracy:
Epoch 5/20
625/625 [==============================] - 6s 10ms/step - loss: 0.2324 - accuracy: 0.9184 - val_loss: 0.2806 - val_accuracy:
Epoch 6/20
625/625 [==============================] - 5s 8ms/step - loss: 0.2184 - accuracy: 0.9229 - val_loss: 0.2787 - val_accuracy:
Epoch 7/20
625/625 [==============================] - 6s 10ms/step - loss: 0.2075 - accuracy: 0.9278 - val_loss: 0.2783 - val_accuracy:
Epoch 8/20
625/625 [==============================] - 6s 9ms/step - loss: 0.1982 - accuracy: 0.9316 - val_loss: 0.2793 - val_accuracy:
Epoch 9/20
625/625 [==============================] - 5s 8ms/step - loss: 0.1905 - accuracy: 0.9349 - val_loss: 0.2806 - val_accuracy:
```

Question 4

Use the test data to evaluate the accuracy of the model.

```
# put your answer here
model.evaluate(test_dataset)
```

```
782/782 [==============================] - 5s 6ms/step - loss: 0.2859 - accuracy: 0.8846
[0.2858824133872986, 0.8846399784088135]
```

Question 5

How many parameters does the model have?

The model has 5001 parameters.

Question 6

Now build a model with a 32-unit hidden layer, using the same method used in class (including an early stop).

```
# put your answer here
input_layer = keras.Input(shape=(1,),dtype=tf.string)

vectorize_layer = vectorizer(input_layer)

hidden_layer = keras.layers.Dense(32,activation='relu')(vectorize_layer)

logistic_regression = keras.layers.Dense(1,activation='sigmoid')(hidden_layer)

model = keras.Model(inputs= input_layer, outputs= logistic_regression)

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

model.summary()
```

```
Model: "model_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 1)]               0
```

```
 text_vectorization_1 (Text  (None, 5000)              0
 Vectorization)

 dense_11 (Dense)            (None, 32)                160032

 dense_12 (Dense)            (None, 1)                 33

=================================================================
Total params: 160065 (625.25 KB)
Trainable params: 160065 (625.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = model.fit(
    train_dataset,
    validation_data = validation_dataset,
    epochs=20)
```

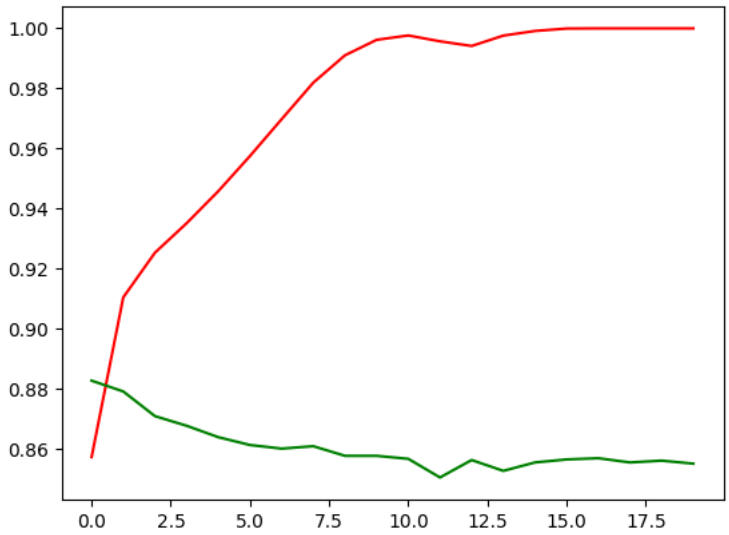
```
Epoch 1/20
625/625 [==============================] - 9s 13ms/step - loss: 0.3437 - accuracy: 0.8572 - val_loss: 0.2863 - val_accuracy:
Epoch 2/20
625/625 [==============================] - 7s 11ms/step - loss: 0.2265 - accuracy: 0.9103 - val_loss: 0.3114 - val_accuracy:
Epoch 3/20
625/625 [==============================] - 7s 11ms/step - loss: 0.1907 - accuracy: 0.9251 - val_loss: 0.3412 - val_accuracy:
Epoch 4/20
625/625 [==============================] - 8s 13ms/step - loss: 0.1658 - accuracy: 0.9349 - val_loss: 0.3719 - val_accuracy:
Epoch 5/20
625/625 [==============================] - 9s 15ms/step - loss: 0.1388 - accuracy: 0.9456 - val_loss: 0.4150 - val_accuracy:
Epoch 6/20
625/625 [==============================] - 7s 11ms/step - loss: 0.1138 - accuracy: 0.9573 - val_loss: 0.4623 - val_accuracy:
Epoch 7/20
625/625 [==============================] - 8s 12ms/step - loss: 0.0854 - accuracy: 0.9696 - val_loss: 0.5218 - val_accuracy:
Epoch 8/20
625/625 [==============================] - 7s 11ms/step - loss: 0.0609 - accuracy: 0.9817 - val_loss: 0.5901 - val_accuracy:
Epoch 9/20
625/625 [==============================] - 8s 13ms/step - loss: 0.0379 - accuracy: 0.9908 - val_loss: 0.6867 - val_accuracy:
Epoch 10/20
625/625 [==============================] - 7s 11ms/step - loss: 0.0240 - accuracy: 0.9959 - val_loss: 0.7640 - val_accuracy:
Epoch 11/20
625/625 [==============================] - 8s 12ms/step - loss: 0.0180 - accuracy: 0.9974 - val_loss: 0.8357 - val_accuracy:
Epoch 12/20
625/625 [==============================] - 8s 12ms/step - loss: 0.0188 - accuracy: 0.9955 - val_loss: 0.9170 - val_accuracy:
Epoch 13/20
625/625 [==============================] - 6s 10ms/step - loss: 0.0206 - accuracy: 0.9939 - val_loss: 0.8781 - val_accuracy:
Epoch 14/20
625/625 [==============================] - 7s 11ms/step - loss: 0.0120 - accuracy: 0.9973 - val_loss: 0.9776 - val_accuracy:
Epoch 15/20
625/625 [==============================] - 9s 15ms/step - loss: 0.0069 - accuracy: 0.9989 - val_loss: 1.0451 - val_accuracy:
Epoch 16/20
625/625 [==============================] - 8s 13ms/step - loss: 0.0034 - accuracy: 0.9997 - val_loss: 1.1221 - val_accuracy:
Epoch 17/20
625/625 [==============================] - 7s 11ms/step - loss: 0.0019 - accuracy: 0.9998 - val_loss: 1.1633 - val_accuracy:
Epoch 18/20
625/625 [==============================] - 8s 13ms/step - loss: 0.0012 - accuracy: 0.9998 - val_loss: 1.2257 - val_accuracy:
Epoch 19/20
625/625 [==============================] - 8s 13ms/step - loss: 8.5904e-04 - accuracy: 0.9998 - val_loss: 1.2656 - val_accur
Epoch 20/20
625/625 [==============================] - 7s 11ms/step - loss: 6.4493e-04 - accuracy: 0.9998 - val_loss: 1.3175 - val_accur
```

```
#plot of loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x7adca44d85e0>]
```


```
#accuracy plot
plt.plot(history.history['accuracy'], color='red')
plt.plot(history.history['val_accuracy'], color = 'green')
```

```
[<matplotlib.lines.Line2D at 0x7adca5f4ce20>]
```




```
# rebuild model with early stopping
input_layer = keras.Input(shape=(1,),dtype=tf.string)
vectorize_layer = vectorizer(input_layer)
hidden_layer = keras.layers.Dense(32,activation='relu')(vectorize_layer)
logistic_regression = keras.layers.Dense(1,activation='sigmoid')(hidden_layer)
model = keras.Model(inputs= input_layer, outputs= logistic_regression)
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
history = model.fit(
    train_dataset,
    validation_data = validation_dataset,
    epochs=20,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',patience=3,restore_best_weights=True)])
```

```
Epoch 1/20
625/625 [==============================] - 9s 12ms/step - loss: 0.3468 - accuracy: 0.8537 - val_loss: 0.2877 - val_accuracy:
Epoch 2/20
625/625 [==============================] - 9s 14ms/step - loss: 0.2257 - accuracy: 0.9115 - val_loss: 0.3116 - val_accuracy:
Epoch 3/20
625/625 [==============================] - 8s 13ms/step - loss: 0.1866 - accuracy: 0.9280 - val_loss: 0.3490 - val_accuracy:
Epoch 4/20
625/625 [==============================] - 9s 14ms/step - loss: 0.1540 - accuracy: 0.9406 - val_loss: 0.3909 - val_accuracy:
```

Question 7

Use the test data to evaluate the accuracy of the model.

```
# put your answer here
model.evaluate(test_dataset)
```

```
782/782 [==============================] - 5s 6ms/step - loss: 0.2960 - accuracy: 0.8784
[0.2960011065006256, 0.8784400224685669]
```

Question 8

How many parameters does the model have?

The model has 160065 parameters

Question 9

Now, try adding a second hidden layer so that your model has the following layers:

- input_layer
- vectorize_layer
- hidden_layer_1
- hidden_layer_2
- logistic regression

The first hidden layer should have 32 units, the second hidden layer should have 16 units. Both should use a relu activation function.

Train using the same method used in class, including an early stop.

```
# put your answer here
input_layer = keras.Input(shape=(1,),dtype=tf.string)


vectorize_layer = vectorizer(input_layer)


hidden_layer_1 = keras.layers.Dense(32,activation='relu')(vectorize_layer)


hidden_layer_2 = keras.layers.Dense(16,activation='relu')(vectorize_layer)


merged_layer = keras.layers.concatenate([hidden_layer_1,hidden_layer_2])


logistic_regression = keras.layers.Dense(1,activation='sigmoid')(merged_layer)


model = keras.Model(inputs= input_layer, outputs= logistic_regression)


model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])


model.summary()
```

```
Model: "model_9"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_10 (InputLayer)       [(None, 1)]               0         []

 text_vectorization_1 (Text  (None, 5000)              0         ['input_10[0][0]']
 Vectorization)

 dense_15 (Dense)            (None, 32)                160032    ['text_vectorization_1[4][0]']

 dense_16 (Dense)            (None, 16)                80016     ['text_vectorization_1[4][0]']

 concatenate_1 (Concatenate  (None, 48)                0         ['dense_15[0][0]',
 )                                                               'dense_16[0][0]']

 dense_17 (Dense)            (None, 1)                 49        ['concatenate_1[0][0]']

==================================================================================================
Total params: 240097 (937.88 KB)
Trainable params: 240097 (937.88 KB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

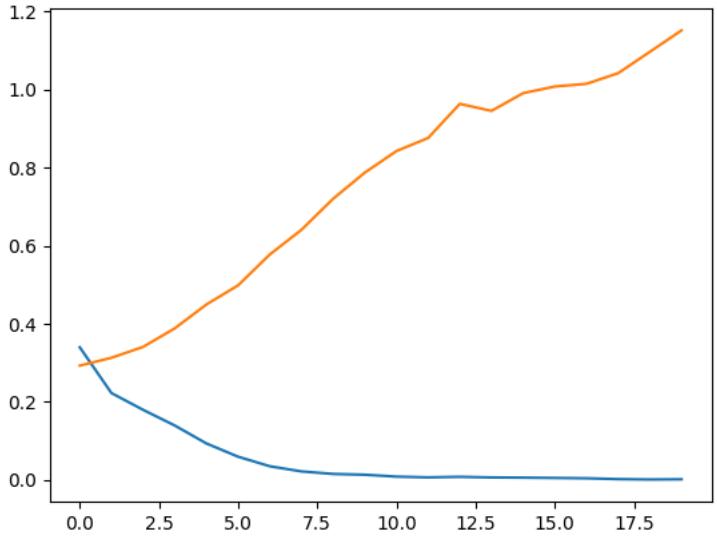
```
history = model.fit(
    train_dataset,
    validation_data = validation_dataset,
    epochs=20)
```

```
Epoch 1/20
625/625 [==============================] - 8s 12ms/step - loss: 0.3401 - accuracy: 0.8586 - val_loss: 0.2930 - val_accuracy:
Epoch 2/20
```

```
625/625 [==============================] - 7s 12ms/step - loss: 0.2228 - accuracy: 0.9120 - val_loss: 0.3131 - val_accuracy:
Epoch 3/20
625/625 [==============================] - 8s 13ms/step - loss: 0.1797 - accuracy: 0.9301 - val_loss: 0.3410 - val_accuracy:
Epoch 4/20
625/625 [==============================] - 9s 14ms/step - loss: 0.1395 - accuracy: 0.9480 - val_loss: 0.3886 - val_accuracy:
Epoch 5/20
625/625 [==============================] - 9s 14ms/step - loss: 0.0935 - accuracy: 0.9685 - val_loss: 0.4498 - val_accuracy:
Epoch 6/20
625/625 [==============================] - 9s 14ms/step - loss: 0.0596 - accuracy: 0.9835 - val_loss: 0.4988 - val_accuracy:
Epoch 7/20
625/625 [==============================] - 9s 15ms/step - loss: 0.0352 - accuracy: 0.9934 - val_loss: 0.5778 - val_accuracy:
Epoch 8/20
625/625 [==============================] - 9s 14ms/step - loss: 0.0220 - accuracy: 0.9968 - val_loss: 0.6411 - val_accuracy:
Epoch 9/20
625/625 [==============================] - 8s 12ms/step - loss: 0.0155 - accuracy: 0.9983 - val_loss: 0.7206 - val_accuracy:
Epoch 10/20
625/625 [==============================] - 9s 14ms/step - loss: 0.0135 - accuracy: 0.9980 - val_loss: 0.7875 - val_accuracy:
Epoch 11/20
625/625 [==============================] - 9s 15ms/step - loss: 0.0086 - accuracy: 0.9990 - val_loss: 0.8428 - val_accuracy:
Epoch 12/20
625/625 [==============================] - 7s 12ms/step - loss: 0.0068 - accuracy: 0.9992 - val_loss: 0.8763 - val_accuracy:
Epoch 13/20
625/625 [==============================] - 9s 14ms/step - loss: 0.0081 - accuracy: 0.9984 - val_loss: 0.9638 - val_accuracy:
Epoch 14/20
625/625 [==============================] - 9s 14ms/step - loss: 0.0065 - accuracy: 0.9987 - val_loss: 0.9457 - val_accuracy:
Epoch 15/20
625/625 [==============================] - 8s 13ms/step - loss: 0.0060 - accuracy: 0.9990 - val_loss: 0.9911 - val_accuracy:
Epoch 16/20
625/625 [==============================] - 10s 16ms/step - loss: 0.0052 - accuracy: 0.9989 - val_loss: 1.0081 - val_accuracy
Epoch 17/20
625/625 [==============================] - 9s 14ms/step - loss: 0.0044 - accuracy: 0.9992 - val_loss: 1.0150 - val_accuracy:
Epoch 18/20
625/625 [==============================] - 8s 13ms/step - loss: 0.0022 - accuracy: 0.9999 - val_loss: 1.0424 - val_accuracy:
Epoch 19/20
625/625 [==============================] - 11s 18ms/step - loss: 0.0013 - accuracy: 0.9999 - val_loss: 1.0970 - val_accuracy
Epoch 20/20
625/625 [==============================] - 11s 17ms/step - loss: 0.0018 - accuracy: 0.9997 - val_loss: 1.1519 - val_accuracy
```

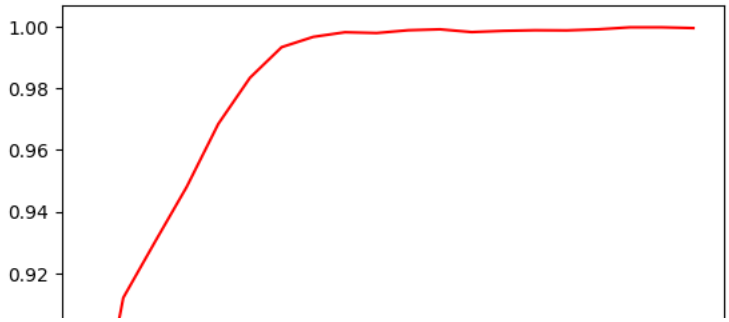
```
#loss plot
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x7adca5cde2f0>]
```




```
#accuracy plot
plt.plot(history.history['accuracy'], color='red')
plt.plot(history.history['val_accuracy'], color = 'green')
```

```
[<matplotlib.lines.Line2D at 0x7adca5352fe0>]
```




```
# rebuild model with early stopping
input_layer = keras.Input(shape=(1,),dtype=tf.string)
vectorize_layer = vectorizer(input_layer)
hidden_layer_1 = keras.layers.Dense(32,activation='relu')(vectorize_layer)
hidden_layer_2 = keras.layers.Dense(16,activation='relu')(vectorize_layer)
merged_layer = keras.layers.concatenate([hidden_layer_1,hidden_layer_2])
logistic_regression = keras.layers.Dense(1,activation='sigmoid')(merged_layer)
model = keras.Model(inputs= input_layer, outputs= logistic_regression)
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
history = model.fit(
    train_dataset,
    validation_data = validation_dataset,
    epochs=20,
    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',patience=3,restore_best_weights=True)])
```

```
Epoch 1/20
625/625 [==============================] - 12s 17ms/step - loss: 0.3400 - accuracy: 0.8570 - val_loss: 0.2929 - val_accuracy
Epoch 2/20
625/625 [==============================] - 8s 13ms/step - loss: 0.2185 - accuracy: 0.9136 - val_loss: 0.3145 - val_accuracy:
Epoch 3/20
625/625 [==============================] - 9s 14ms/step - loss: 0.1710 - accuracy: 0.9358 - val_loss: 0.3517 - val_accuracy:
Epoch 4/20
625/625 [==============================] - 9s 15ms/step - loss: 0.1276 - accuracy: 0.9533 - val_loss: 0.4138 - val_accuracy:
```

Question 10

Use the test data to evaluate the accuracy of the model.

```
# put your answer here
model.evaluate(test_dataset)
```

```
782/782 [==============================] - 6s 8ms/step - loss: 0.3017 - accuracy: 0.8767
[0.3017010986804962, 0.8766800165176392]
```

Question 11

How many parameters does the model have?

The model has 240097 parameters