P1 (40pt): In the following code example explained in Lecture 6, https://colab.research.google.com/drive/13cof4XUULbUqO0s5h-cd17FskWjpyMkm,

make the following changes to the neural network model sequentially in the example:

1. Change the number of neurons on the hidden layer to 256 units. **(10pt)**
2. Use the tanh activation (an activation that was popular in the early days of neural networks) instead of relu for the hidden layer. **(10pt)**
3. Add an additional hidden layer with 256 units and tanh activation function. **(10pt)**

Retrain the newly defined model and evaluate the trained model on the testing dataset to get the accuracy. **(10pt)**

```
# import necessary packages
import tensorflow as tf
from tensorflow import keras


# import data
from tensorflow.keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist
11490434/11490434 [==============================] - 0s 0us/step
```

```
# build network with 256 neurons in the input layer
from tensorflow.keras import models
from tensorflow.keras import layers

network = models.Sequential()
network.add(layers.Dense(256, activation='tanh', input_shape=(28 * 28,)))
network.add(layers.Dense(256, activation='tanh'))
network.add(layers.Dense(10, activation='softmax'))


network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

```
# reshape data
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255



# add labels
from tensorflow.keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)


# fit neural network
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Epoch 1/5
469/469 [==============================] - 4s 3ms/step - loss: 0.3055 - accuracy: 0.908
Epoch 2/5
469/469 [==============================] - 1s 3ms/step - loss: 0.1405 - accuracy: 0.957
Epoch 3/5
469/469 [==============================] - 1s 3ms/step - loss: 0.0929 - accuracy: 0.972
Epoch 4/5
469/469 [==============================] - 1s 3ms/step - loss: 0.0675 - accuracy: 0.979
Epoch 5/5
469/469 [==============================] - 1s 3ms/step - loss: 0.0494 - accuracy: 0.984
<keras.callbacks.History at 0x7f25103f5390>
```

```
# run network on test data
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0788 - accuracy: 0.973
```

```
# accuracy
print('test_acc:', test_acc)
```

```
test_acc: 0.9735000133514404
```

P2 (60pt): Write a Python code in Colab using NumPy, Panda, Scikit-Learn and Keras to complete the following tasks:

1. Import the Auto MPG dataset using pandas.read_csv(), use the attribute names as explained in the dataset description as the column names, view the strings '?' as the missing value, and whitespace (i.e., '\s+') as the column delimiter. Print out the shape and first 5 rows of the DataFrame. **(5pt)**

    a. Dataset source file: http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data

    b. Dataset description: http://archive.ics.uci.edu/ml/datasets/Auto+MPG

```
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
from tensorflow import keras

import numpy as np
np.random.seed(100)

tf.random.set_seed(100)


# read in data
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data"
auto = pd.read_csv(url,
                   delimiter = "\s+",
                   header = None,
                   names = ["mpg", "cylinders", "displacement", "horsepower", "weight", "acce
                   na_values= ["?", " ?", "? ", " ? "])
auto.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
0	18.0	8	307.0	130.0	3504.0	12.0	70	1
1	15.0	8	350.0	165.0	3693.0	11.5	70	1
2	18.0	8	318.0	150.0	3436.0	11.0	70	1
3	16.0	8	304.0	150.0	3433.0	12.0	70	1
4	17.0	8	302.0	140.0	3449.0	10.5	70	1

```
auto.shape
```

```
(398, 9)
```

2. Delete the "car_name" column using .drop() and drop the rows containing NULL value using .dropna(). Print out the shape of the DataFrame. **(5pt)**

```
newauto = auto.drop(columns = ["car name"])
newauto = newauto.dropna()
newauto.shape
```

```
(392, 8)
```

```
newauto.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 392 entries, 0 to 397
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64  
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        392 non-null    float64
 5   acceleration  392 non-null    float64
 6   year          392 non-null    int64  
 7   origin        392 non-null    int64  
dtypes: float64(5), int64(3)
memory usage: 27.6 KB
```

3. For the ‘origin’ column with categorical attribute, replace it with the columns with numerical attributes using one-hot encoding. Print out the shape and first 5 rows of the new DataFrame. **(5pt)**

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np
vec = OneHotEncoder(dtype = "int64")
origin = np.array(newauto['origin'])
origin = origin.reshape(-1, 1)
origin = vec.fit_transform(origin)
origin = pd.DataFrame(origin.toarray(), columns=vec.get_feature_names())
origin
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
  warnings.warn(msg, category=FutureWarning)
```

	x0_1	x0_2	x0_3
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
...	...	...	...

```
newauto2 = newauto.drop(columns = ["origin"])
newauto2 = newauto2.join(origin)
newauto2.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	x0_1	x0_2	x0
0	18.0	8	307.0	130.0	3504.0	12.0	70	1.0	0.0	(
1	15.0	8	350.0	165.0	3693.0	11.5	70	1.0	0.0	(
2	18.0	8	318.0	150.0	3436.0	11.0	70	1.0	0.0	(
3	16.0	8	304.0	150.0	3433.0	12.0	70	1.0	0.0	(
4	17.0	8	302.0	140.0	3449.0	10.5	70	1.0	0.0	(

4. Separate the “mpg” column from other columns and view it as the label vector and others as the feature matrix. Split the data into a training set (80%) and testing set (20%) using train_test_split and print out their shapes. Print out the statistics of your training feature matrix using .describe(). **(5pt)**

```
from sklearn.model_selection import train_test_split

mpg = newauto2['mpg']
autosplit = newauto2.drop(columns = ["mpg"])

X_train, X_test, y_train, y_test = train_test_split(autosplit, mpg,
                                                    test_size=0.20,
                                                    random_state=42)
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (313, 9)
```

```
X_test shape: (79, 9)
y_train shape: (313,)
y_test shape: (79,)
```

```
X_train.describe()
```

	cylinders	displacement	horsepower	weight	acceleration	year	
count	313.000000	313.000000	313.000000	313.000000	313.000000	313.000000	309.0
mean	5.482428	195.517572	104.594249	2986.124601	15.544089	76.207668	0.6
std	1.700446	103.766567	38.283669	841.133957	2.817864	3.630136	0.4
min	3.000000	70.000000	46.000000	1613.000000	8.000000	70.000000	0.0
25%	4.000000	105.000000	76.000000	2234.000000	13.500000	73.000000	0.0
50%	4.000000	151.000000	95.000000	2855.000000	15.500000	76.000000	1.0
75%	8.000000	302.000000	129.000000	3645.000000	17.300000	79.000000	1.0
max	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	1.0

5. Normalize the feature columns in both training and testing datasets so that their means equal to zero and variances equal to one. Note that the testing set can only be scaled by the mean and standard deviation values obtained from the training set. Describe the statistics of your normalized feature matrix of training dataset using .describe() in Pandas. **(5pt)**

- Option 1: You can follow the normalization steps in the code example of “Predicting house prices: a regression example” in Lecture 6.
- Option 2: You can use StandardScaler() in Scikit-Learn as in Homework 2 but you may need to transform a NumPy array back to Pandas DataFrame using pd.DataFrame() before calling .describe().

```
xtrainmean = X_train.mean(axis = 0)
X_train -= xtrainmean
ytrainmean = y_train.mean(axis = 0)
y_train -= ytrainmean

xtrainstd = X_train.std(axis = 0)
X_train /= xtrainstd
ytrainstd = y_train.std(axis = 0)
y_train /= ytrainstd



X_test -= xtrainmean
```

```
y_test -= ytrainmean
X_test /= xtrainstd
y_test /= ytrainstd
```

```
X_train.describe()
```

	cylinders	displacement	horsepower	weight	acceleration	y
count	3.130000e+02	3.130000e+02	3.130000e+02	3.130000e+02	3.130000e+02	3.130000e
mean	-3.688920e-17	2.837631e-17	1.702578e-17	-1.135052e-17	1.418815e-17	-3.405157
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e
min	-1.459869e+00	-1.209615e+00	-1.530529e+00	-1.632468e+00	-2.677237e+00	-1.710037e
25%	-8.717877e-01	-8.723192e-01	-7.469046e-01	-8.941793e-01	-7.254039e-01	-8.836218
50%	-8.717877e-01	-4.290165e-01	-2.506094e-01	-1.558903e-01	-1.564641e-02	-5.720659

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-90-6e837eca3a7a> in <module>
----> 1 y_train.info()

/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py in __getattr__(self, name)
   5485         ):
   5486             return self[name]
-> 5487         return object.__getattribute__(self, name)
   5488 
   5489     def __setattr__(self, name: str, value) -> None:

AttributeError: 'Series' object has no attribute 'info'
```

SEARCH STACK OVERFLOW

6. Build a sequential neural network model in Keras with two densely connected hidden layers (32 neurons and ReLU activation function for each hidden layer), and an output layer that returns a single, continuous value. Print out the model summary using .summary(). **(10pt)**

- Hint: You can follow the “Classifying movie reviews” example in Lecture 6, but need to change input_shape and last layer activation function correctly in the model definition.

```
network = models.Sequential()
network.add(layers.Dense(32, activation='relu'))
network.add(layers.Dense(32, activation='relu'))
network.add(layers.Dense(10, activation = "sigmoid"))
```

7. Define the appropriate loss function, optimizer, and metrics for this specific problem and compile the NN model. **(10pt)**

```
network.compile(optimizer='rmsprop',
                loss='binary_crossentropy',
                metrics=['accuracy'])
```

8. Put aside 20% of the normalized training data as the validation dataset by setting validation_split = 0.2 and set verbose = 0 to compress the model training status in Keras .fit(). Train the NN model for 100 epochs and batch size of 32 and plot the training and validation loss progress with respect to the epoch number. **(10pt)**

- Remember to use GPU for training in Colab. Otherwise, you may find out of memory error or slow execution.
- There is no need to do K-fold cross-validation for this step.

```
# I can't get the model to run, there's some sort of bug that I cannoot figure out but from h
# keeping it from running

NN = network.fit(X_train, y_train,
                 epochs = 100,
                 batch_size = 32,
                 validation_split = 0.2,
                 verbose = 0)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-100-306930f3197c> in <module>
```

```
import matplotlib.pyplot as plt
plt.plot(network.network['acc'])
plt.plot(network.network['val_acc'])
```

1 frames

9. Use the trained NN model to make predictions on the normalized testing dataset and observe the prediction error. **(5pt)**

```
--> 947         return self._stateless_fn(*args, **kwds)  # pylint: disable=not-callable
```

```
test_loss, test_acc = network.evaluate(X_test, y_test)
print('test_acc:', test_acc)
```

```
TypeError: 'NoneType' object is not callable
```

SEARCH STACK OVERFLOW

Colab paid products - Cancel contracts here

0s completed at 8:32 PM