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)**

```
# To get started, import tf.keras as part of your TensorFlow program setup:
import tensorflow as tf
from tensorflow import keras
```

```
from tensorflow.keras.datasets import mnist

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

```
train_images.shape
```

```
(60000, 28, 28)
```

```
len(train_labels)
```

```
60000
```

```
train_labels
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
test_images.shape
```

```
(10000, 28, 28)
```

```
len(test_labels)
```

```
10000
```

```
test_labels
```

```
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

```
from tensorflow.keras import models
from tensorflow.keras import layers

network1 = models.Sequential()
network1.add(layers.Dense(256, activation='tanh', input_shape=(28 * 28,)))
network1.add(layers.Dense(256, activation='tanh'))
network1.add(layers.Dense(10, activation='softmax'))
```

```
network1.compile(optimizer='rmsprop',
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])
```

```
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
```

```
from tensorflow.keras.utils import to_categorical

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

```
network1.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Epoch 1/5
469/469 [==============================] - 3s 5ms/step - loss: 0.3090 - accuracy: 0.9086
Epoch 2/5
469/469 [==============================] - 2s 4ms/step - loss: 0.1415 - accuracy: 0.9565
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 0.0928 - accuracy: 0.9717
Epoch 4/5
469/469 [==============================] - 2s 4ms/step - loss: 0.0665 - accuracy: 0.9797
Epoch 5/5
469/469 [==============================] - 2s 5ms/step - loss: 0.0506 - accuracy: 0.9846
<keras.callbacks.History at 0x7f8402a3af10>
```

```
test_loss, test_acc = network1.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0626 - accuracy: 0.9798
```

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

```
test_acc: 0.9797999858856201
```

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(1234)

tf.random.set_seed(1234)
```

```
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data"
col_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight', 'Acceleration', 'Model Year', 'Origin', 'c
df = pd.read_csv(url, names = col_names, na_values = "?",
                     comment = '\t', sep = '\s+', header = None)
print('The shape of the dataset is: ', df.shape)
```

```
The shape of the dataset is:  (398, 9)
```

```
print('The first 5 rows of the DataFrame are: \n', df.head())
```

```
The first 5 rows of the DataFrame are:
     MPG  Cylinders  Displacement  Horsepower  Weight  Acceleration  \
0  18.0          8         307.0       130.0  3504.0          12.0
1  15.0          8         350.0       165.0  3693.0          11.5
2  18.0          8         318.0       150.0  3436.0          11.0
3  16.0          8         304.0       150.0  3433.0          12.0
4  17.0          8         302.0       140.0  3449.0          10.5

   Model Year  Origin                   car_name
0          70       1  chevrolet chevelle malibu
1          70       1          buick skylark 320
2          70       1         plymouth satellite
3          70       1              amc rebel sst
4          70       1                ford torino
```

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)**

```
df = df.drop(columns = ["car_name"])
df.isna().sum()
df.shape
```

```
(398, 8)
```

```
df = df.dropna()
df.shape
```

```
(392, 8)
```

```
df.head()
```

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)**

```
n_dataset = pd.get_dummies(df, columns = ["Origin"])
n_dataset.shape
```

```
(392, 10)
```

```
n_dataset.head()
```

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

X = n_dataset.drop(columns = 'MPG')
y = n_dataset['MPG']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

```
X_train.describe()
```

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().

```
average = X_train.mean(axis = 0)
X_train = X_train - average
std_val = X_train.std(axis = 0)
X_train = X_train / std_val

X_test = X_test - average
X_test = X_test / std_val
```

```
X_train.describe()
```

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.

```
from keras import models
from keras import layers

model1 = models.Sequential()
model1.add(layers.Dense(32, activation = 'relu', input_shape = (9, )))
model1.add(layers.Dense(32, activation = 'relu'))
model1.add(layers.Dense(1))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                320

 dense_1 (Dense)             (None, 32)                1056

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 1,409
Trainable params: 1,409
Non-trainable params: 0
_________________________________________________________________
```

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

```
model1.compile(optimizer = 'rmsprop', loss = 'mse', metrics = ['mae', 'mse'])
```

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.

```
com_data = model1.fit(X_train, y_train, batch_size = 32, epochs = 100, validation_split = 0.2, verbose = 0)
```

```
com_data_dict = com_data.history
com_data_dict.keys()
```

```
dict_keys(['loss', 'mae', 'mse', 'val_loss', 'val_mae', 'val_mse'])
```

```
mae_val = com_data.history['val_mae']
mse_val = com_data.history['val_mse']
mae = com_data.history['mae']
```

```
mse = com_data.history['mse']

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

plt.plot(epochs, mse, label = 'Training mse')
plt.plot(epochs, mse_val, label = 'Validation mse')
plt.title('TRAINING AND VALIDATION MSE')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

```
plt.clf()
plt.plot(epochs, mae, label = 'Training MAE')
plt.plot(epochs, mae_val, 'b', label = 'Validation MAE')
plt.title('TRAINING AND VALIDATION MAE')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

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

```
model1.evaluate(X_test, y_test)
```

```
3/3 [==============================] - 0s 6ms/step - loss: 9.0672 - mae: 2.1624 - mse: 9.0672
[9.067174911499023, 2.162414312362671, 9.067174911499023]
```

✓ 0s completed at 9:39 PM