Tensorflow 2.0 does not support the defining of a placeeholder,placeholder, to run this notebook smootly smoothly without errors install tesorflow 1.15 by using the code snippet below.

```
pip install tensorflow==1.15
```

```
Collecting tensorflow==1.15
  Downloading https://files.pythonhosted.org/packages/92/2b/e3af15221da9ff323521565fa332
     |████████████████████████████████| 412.3MB 40kB/s
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python3.7/dist-packages (f
Collecting tensorflow-estimator==1.15.1
  Downloading https://files.pythonhosted.org/packages/de/62/2ee9cd74c9fa2fa450877847ba56
     |████████████████████████████████| 512kB 45.7MB/s
Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: keras-preprocessing>=1.0.5 in /usr/local/lib/python3.7/di
Collecting tensorboard<1.16.0,>=1.15.0
  Downloading https://files.pythonhosted.org/packages/1e/e9/d3d747a97f7188f48aa5eda48690
     |████████████████████████████████| 3.8MB 38.7MB/s
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packag
Collecting gast==0.2.2
  Downloading https://files.pythonhosted.org/packages/4e/35/11749bf99b2d4e3cceb4d55ca225
Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: google-pasta>=0.1.6 in /usr/local/lib/python3.7/dist-pack
Collecting keras-applications>=1.0.8
  Downloading https://files.pythonhosted.org/packages/71/e3/19762fdfc62877ae9102edf6342d
     |████████████████████████████████| 51kB 6.1MB/s
Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (from kera
Requirement already satisfied: importlib-metadata; python_version < "3.8" in /usr/local/
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: typing-extensions>=3.6.4; python_version < "3.8" in /usr/
Building wheels for collected packages: gast
  Building wheel for gast (setup.py) ... done
  Created wheel for gast: filename=gast-0.2.2-cp37-none-any.whl size=7540 sha256=d98b6ea
  Stored in directory: /root/.cache/pip/wheels/5c/2e/7e/a1d4d4fcebe6c381f378ce7743a3ced3
Successfully built gast
ERROR: tensorflow-probability 0.12.1 has requirement gast>=0.3.2, but you'll have gast 0
Installing collected packages: tensorflow-estimator, tensorboard, gast, keras-applicatio
  Found existing installation: tensorflow-estimator 2.4.0
    Uninstalling tensorflow-estimator-2.4.0:
      Successfully uninstalled tensorflow-estimator-2.4.0
  Found existing installation: tensorboard 2.4.1
    Uninstalling tensorboard-2.4.1:
      Successfully uninstalled tensorboard-2.4.1
  Found existing installation: gast 0.3.3
    Uninstalling gast-0.3.3:
      Successfully uninstalled gast-0.3.3
```

```
  Found existing installation: tensorflow 2.4.1
    Uninstalling tensorflow-2.4.1:
      Successfully uninstalled tensorflow-2.4.1
Successfully installed gast-0.2.2 keras-applications-1.0.8 tensorboard-1.15.0 tensorflow
```

Introduction to Keras and Tensorflow with Python

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

print("Training data: {}, {}".format(train_images.shape, train_labels.shape))
# prints Training data: (60000, 28, 28), (60000,)
print("Test data: {}, {}".format(test_images.shape, test_labels.shape))
# prints Test data: (10000, 28, 28), (10000,)
class_labels = np.unique(train_labels)
print("There are {} classes in the dataset. They are: {}".format(len(class_labels), class_lab
# prints There are 10 classes in the dataset. They are: [0 1 2 3 4 5 6 7 8 9]
```

```
Training data: (60000, 28, 28), (60000,)
Test data: (10000, 28, 28), (10000,)
There are 10 classes in the dataset. They are: [0 1 2 3 4 5 6 7 8 9]
```

Visualize the images

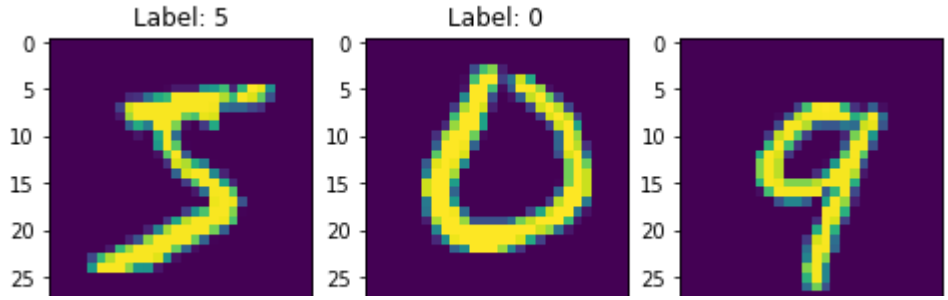
```
plt.figure(figsize=(8, 5))

plt.subplot(1,3,1)
plt.imshow(train_images[0])
plt.title("Label: {}".format(train_labels[0]))

plt.subplot(1,3,2)
plt.imshow(train_images[2500])
plt.title("Label: {}".format(train_labels[2500]))

plt.subplot(1,3,3)
plt.imshow(test_images[12])

plt.show()
```



Scale the Data

```
# As usual, we scale our dataset to range between 0 and 1. In this dataset, the pixel values
# Dividing the data by 255 scales to the required range

train_images = train_images / 255.0
test_images = test_images / 255.0
```

Split training data to training and validation sets

```
x_train = train_images[0:50000]
x_val = train_images[50000:]
y_train = train_labels[0:50000]
y_val = train_labels[50000:]

print("x_train: {}".format(x_train.shape)) # prints x_train: (50000, 28, 28)
print("x_val: {}".format(x_val.shape)) # prints x_val: (10000, 28, 28)
print("y_train: {}".format(y_train.shape)) #prints y_train: (50000,)
print("y_val: {}".format(y_val.shape)) #prints y_val: (10000,)
```

```
x_train: (50000, 28, 28)
x_val: (10000, 28, 28)
y_train: (50000,)
y_val: (10000,)
```

Reshape data from 28 * 28 array to a single array

```
new_dimension = np.prod(train_images.shape[1:])
x_train = x_train.reshape(x_train.shape[0], new_dimension)
x_val = x_val.reshape(x_val.shape[0], new_dimension)
test_images = test_images.reshape(test_images.shape[0], new_dimension)

print("x_train: {}".format(x_train.shape)) #prints  x_train: (50000, 784)
print("x_val: {}".format(x_val.shape)) #prints x_val: (10000, 784)
print("test_images: {}".format(test_images.shape)) #prints test_images: (10000, 784)
```

```
x_train: (50000, 784)
x_val: (10000, 784)
test_images: (10000, 784)
```

Encode labels to categorical variables

```
from tensorflow.keras.utils import to_categorical
no_labels = 10
y_train = to_categorical(y_train, no_labels)
y_val = to_categorical(y_val, no_labels)
y_test = to_categorical(test_labels, no_labels)
```

Activation functions and Neural Networks hyperparameters

```
X = tf.placeholder(tf.float32, [None, new_dimension])
Y = tf.placeholder(tf.float32, [None, no_labels])

# create model architecture
def multilayer_perceptron(x, no_classes, first_layer_neurons=256, second_layer_neurons=128):
  # first layer
  first_weight = tf.Variable(tf.random_uniform([new_dimension, first_layer_neurons]))
  first_bias = tf.Variable(tf.zeros([first_layer_neurons]))
  first_layer_output = tf.nn.relu(tf.add(tf.matmul(x, first_weight), first_bias))

  # second layer
  second_weight = tf.Variable(tf.random_uniform([first_layer_neurons,second_layer_neurons]))
  second_bias = tf.Variable(tf.zeros([second_layer_neurons]))
  second_layer_output = tf.nn.relu(tf.add(tf.matmul(first_layer_output, second_weight),
                                          second_bias))

  # output layer
  final_weight = tf.Variable(tf.random_uniform([second_layer_neurons, no_classes]))
  final_bias = tf.Variable(tf.zeros([no_classes]))
  logits = tf.add(tf.matmul(second_layer_output, final_weight), final_bias)

  return logits
```

Call the multilayer perception function

```
logits = multilayer_perceptron(X, no_labels)

learning_rate = 0.01
#we define the loss and optimiser for the network
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=Y))
optimiser = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimiser.minimize(loss_op)

#initialize the variables
```

```
#initialise the variables
init = tf.global_variables_initializer()

epochs = 20
batch_size = 1000
iteration = len(x_train) // batch_size


#train model
with tf.Session() as session:
  session.run(init)
  for epoch in range(epochs):
    average_cost = 0
    start, end = 0, batch_size

    for i in range(iteration):
      batch_x, batch_y = x_train[start: end], y_train[start: end]
      _, loss = session.run([train_op, loss_op], feed_dict={X: batch_x, Y: batch_y})
      start += batch_size
      end += batch_size
      #average loss
      average_cost += loss/iteration
    print("Epoch========{}".format(epoch))

  #evaluate model
  prediction = tf.nn.softmax(logits)
  ground_truth = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
  accuracy = tf.reduce_mean(tf.cast(ground_truth, "float"))
  print("Accuracy: {}".format(accuracy.eval({X: test_images, Y: y_test})))
```

```
WARNING:tensorflow:From <ipython-input-16-4a582314747b>:5: softmax_cross_entropy_with_lo
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.

Epoch========0
Epoch========1
Epoch========2
Epoch========3
Epoch========4
Epoch========5
Epoch========6
Epoch========7
Epoch========8
Epoch========9
Epoch========10
Epoch========11
Epoch========12
Epoch========13
Epoch========14
Epoch========15
```

```
Epoch========16
Epoch========17
Epoch========18
Epoch========19
Accuracy: 0.9045000076293945
```

✓ 25s completed at 7:28 PM