Topics in Deep learning Hands-On Unit 1

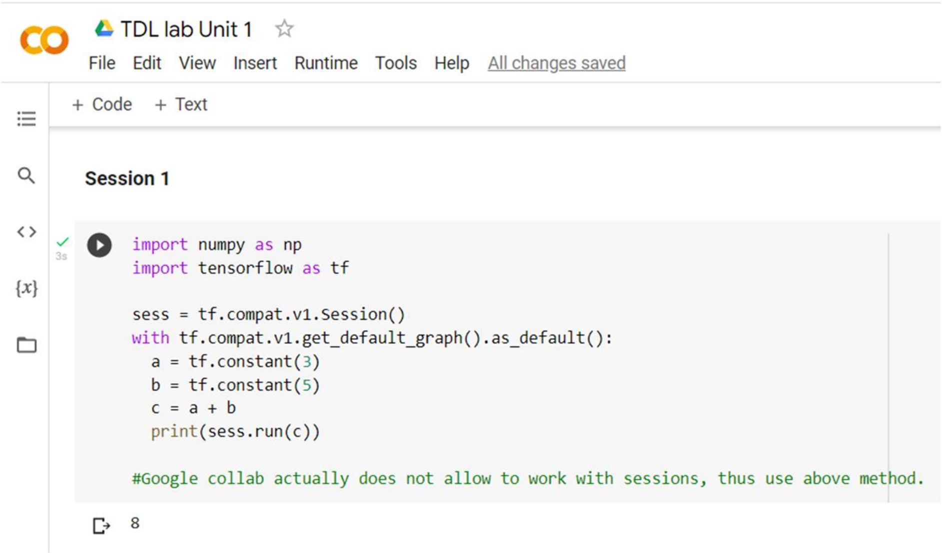
Name – B Pravena Section – B

SRN – PES2UG19CS076

Google collab link -:

https://colab.research.google.com/drive/1Zx4-m4HNDyR9Mi1noYS_pgKrMKPyqayS#scrollTo=VXTs5uqWfTCT

Sessions and Graphs -:

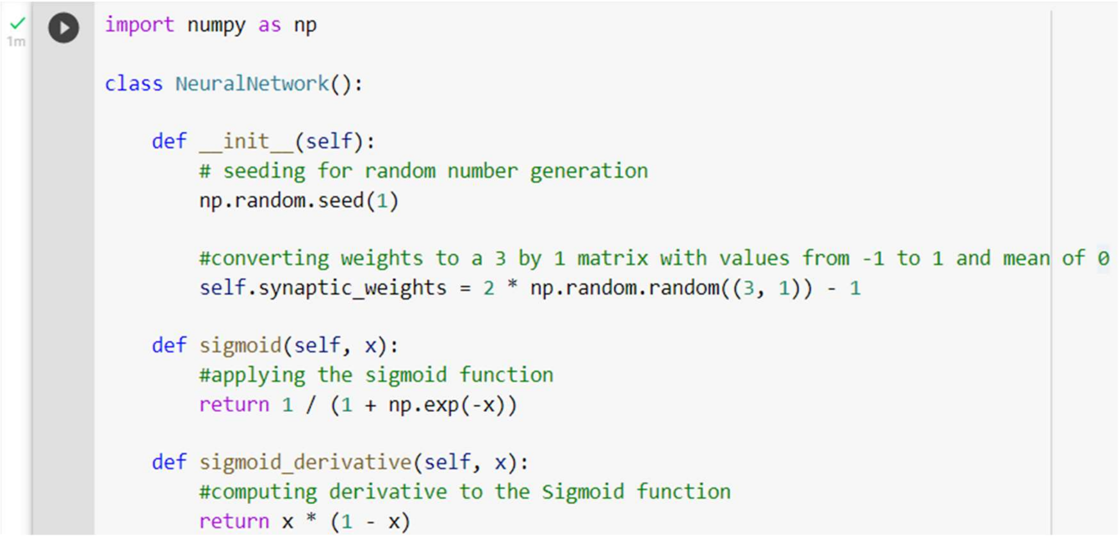


Session 1

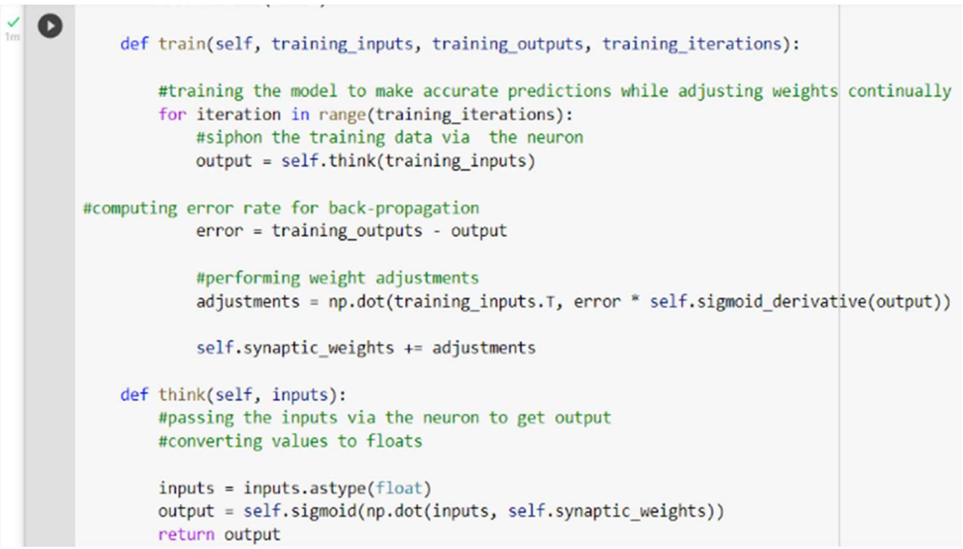
```
import numpy as np
import tensorflow as tf

sess = tf.compat.v1.Session()
with tf.compat.v1.get_default_graph().as_default():
  a = tf.constant(3)
  b = tf.constant(5)
  c = a + b
  print(sess.run(c))

#Google collab actually does not allow to work with sessions, thus use above method.
```

```
8
```

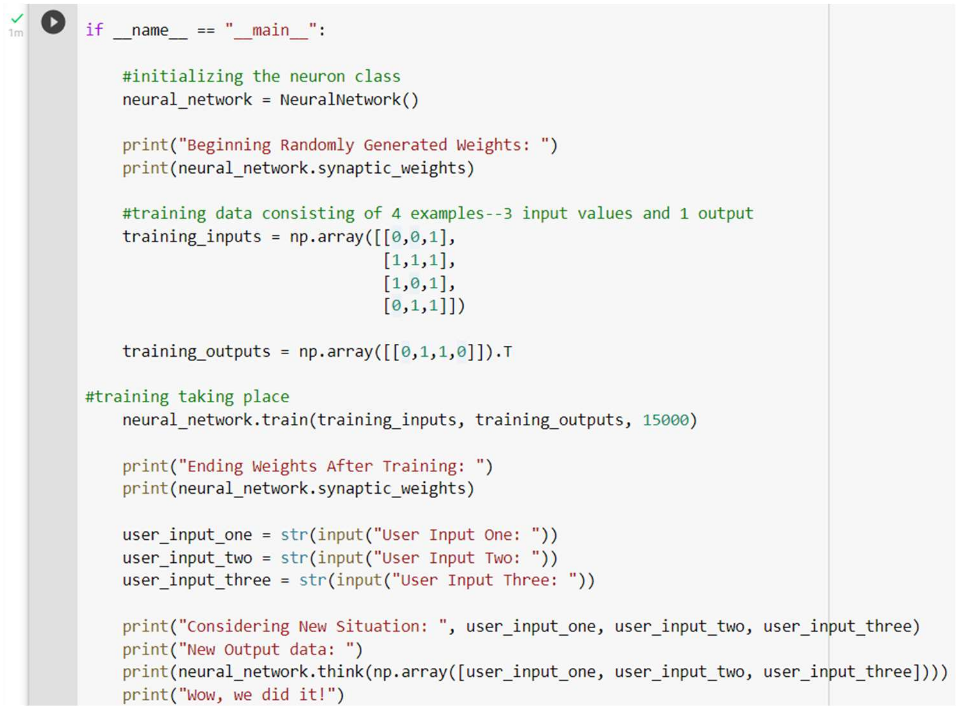
Simple Neural Network -:



```
import numpy as np

class NeuralNetwork():

    def __init__(self):
        # seeding for random number generation
        np.random.seed(1)

        #converting weights to a 3 by 1 matrix with values from -1 to 1 and mean of 0
        self.synaptic_weights = 2 * np.random.random((3, 1)) - 1

    def sigmoid(self, x):
        #applying the sigmoid function
        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):
        #computing derivative to the Sigmoid function
        return x * (1 - x)
```

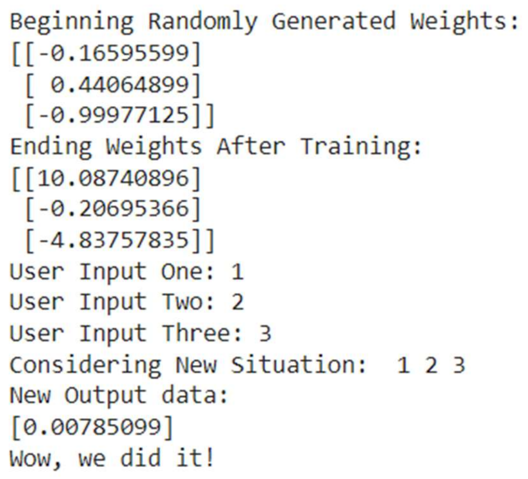


```
    def train(self, training_inputs, training_outputs, training_iterations):

        #training the model to make accurate predictions while adjusting weights continually
        for iteration in range(training_iterations):
            #siphon the training data via  the neuron
            output = self.think(training_inputs)

#computing error rate for back-propagation
            error = training_outputs - output

            #performing weight adjustments
            adjustments = np.dot(training_inputs.T, error * self.sigmoid_derivative(output))

            self.synaptic_weights += adjustments

    def think(self, inputs):
        #passing the inputs via the neuron to get output
        #converting values to floats

        inputs = inputs.astype(float)
        output = self.sigmoid(np.dot(inputs, self.synaptic_weights))
        return output
```



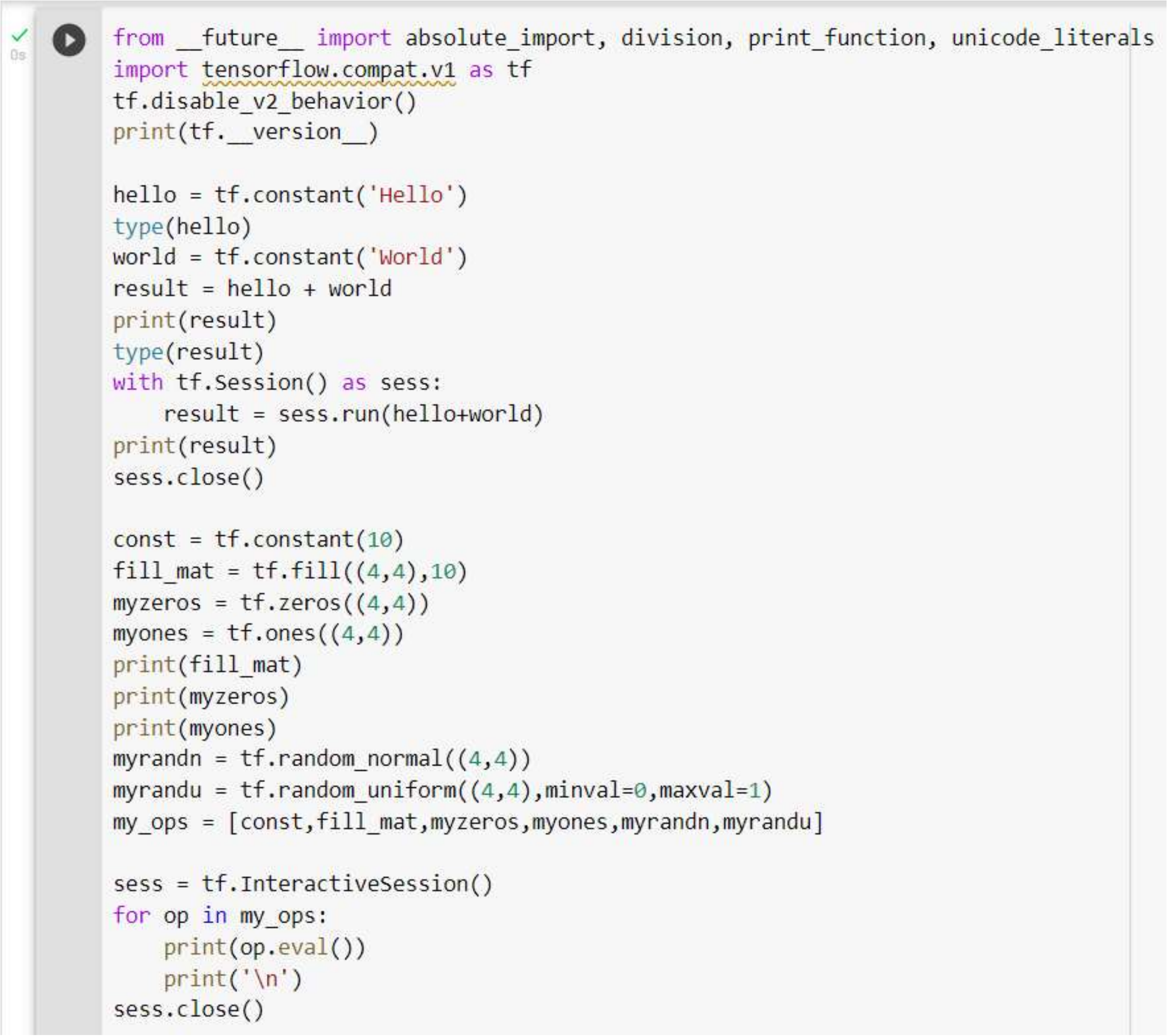
```
if __name__ == "__main__":

    #initializing the neuron class
    neural_network = NeuralNetwork()

    print("Beginning Randomly Generated Weights: ")
    print(neural_network.synaptic_weights)

    #training data consisting of 4 examples--3 input values and 1 output
    training_inputs = np.array([[0,0,1],
                                [1,1,1],
                                [1,0,1],
                                [0,1,1]])

    training_outputs = np.array([[0,1,1,0]]).T

#training taking place
    neural_network.train(training_inputs, training_outputs, 15000)

    print("Ending Weights After Training: ")
    print(neural_network.synaptic_weights)

    user_input_one = str(input("User Input One: "))
    user_input_two = str(input("User Input Two: "))
    user_input_three = str(input("User Input Three: "))

    print("Considering New Situation: ", user_input_one, user_input_two, user_input_three)
    print("New Output data: ")
    print(neural_network.think(np.array([user_input_one, user_input_two, user_input_three])))
    print("Wow, we did it!")
```

Output-:

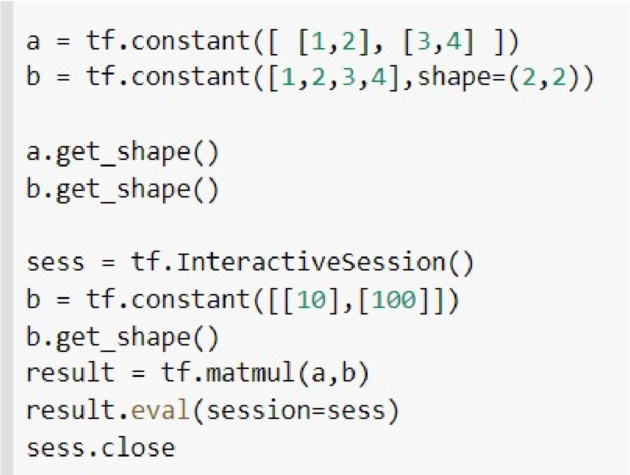


```
Beginning Randomly Generated Weights:
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
Ending Weights After Training:
[[10.08740896]
 [-0.20695366]
 [-4.83757835]]
User Input One: 1
User Input Two: 2
User Input Three: 3
Considering New Situation:  1 2 3
New Output data:
[0.00785099]
Wow, we did it!
```

TensorFlow Basics -:

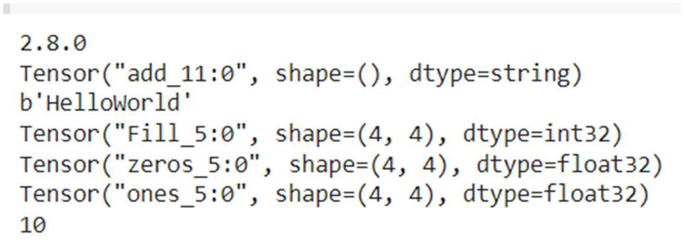


```
from __future__ import absolute_import, division, print_function, unicode_literals
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
print(tf.__version__)

hello = tf.constant('Hello')
type(hello)
world = tf.constant('World')
result = hello + world
print(result)
type(result)
with tf.Session() as sess:
    result = sess.run(hello+world)
print(result)
sess.close()

const = tf.constant(10)
fill_mat = tf.fill((4,4),10)
myzeros = tf.zeros((4,4))
myones = tf.ones((4,4))
print(fill_mat)
print(myzeros)
print(myones)
myrandn = tf.random_normal((4,4))
myrandu = tf.random_uniform((4,4),minval=0,maxval=1)
my_ops = [const,fill_mat,myzeros,myones,myrandn,myrandu]

sess = tf.InteractiveSession()
for op in my_ops:
    print(op.eval())
    print('\n')
sess.close()
```

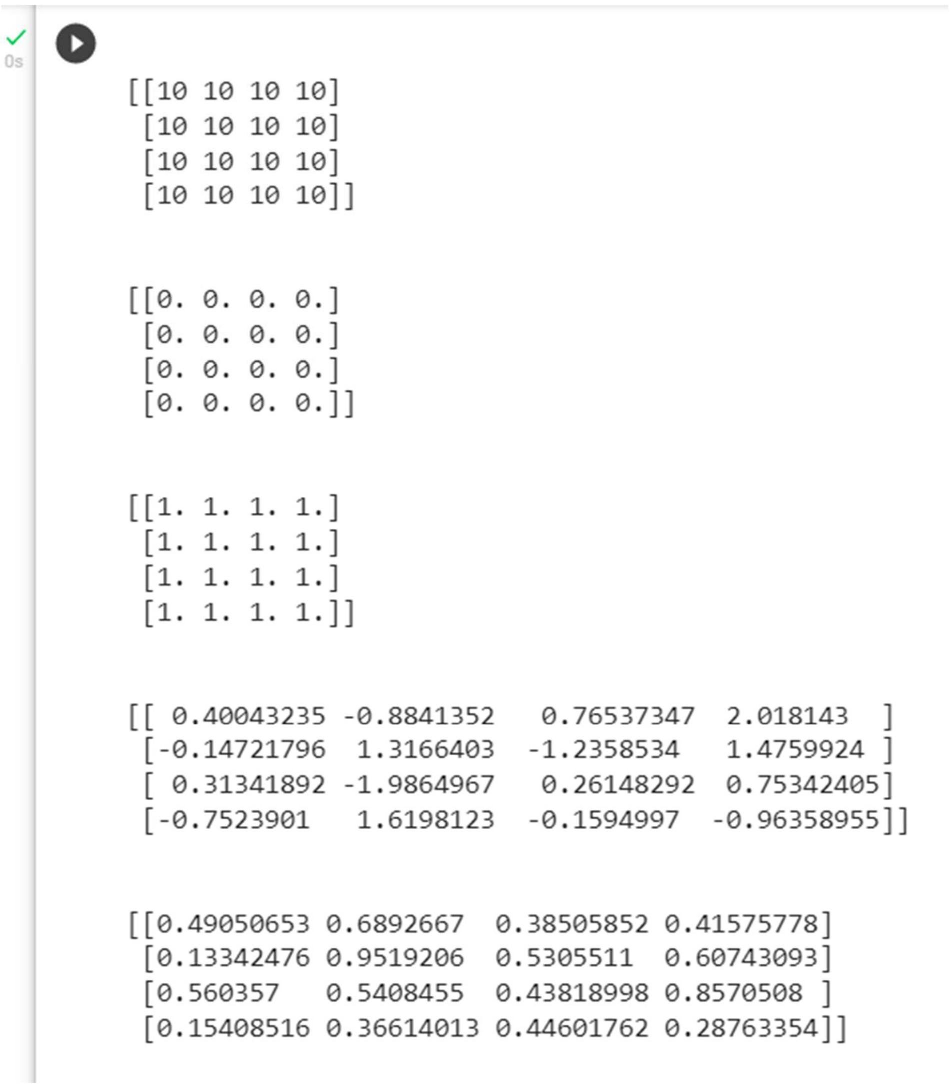


```
a = tf.constant([ [1,2], [3,4] ])
b = tf.constant([1,2,3,4],shape=(2,2))

a.get_shape()
b.get_shape()

sess = tf.InteractiveSession()
b = tf.constant([[10],[100]])
b.get_shape()
result = tf.matmul(a,b)
result.eval(session=sess)
sess.close
```

Output -:

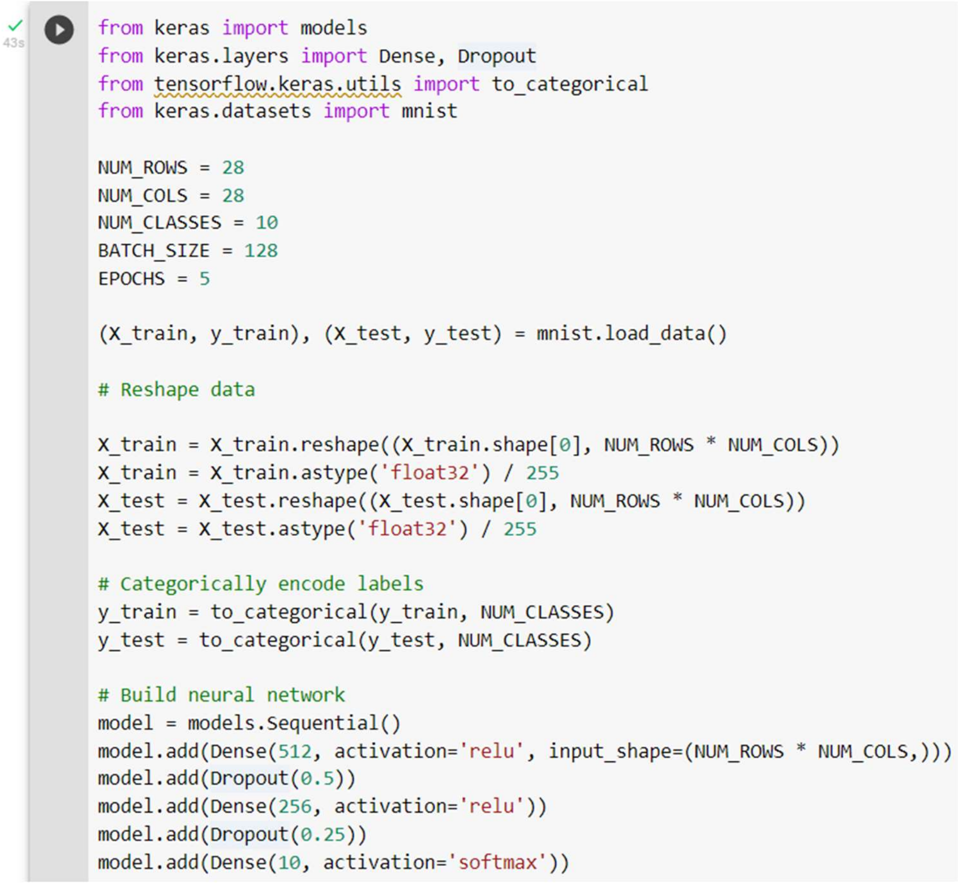


```
2.8.0
Tensor("add_11:0", shape=(), dtype=string)
b'HelloWorld'
Tensor("Fill_5:0", shape=(4, 4), dtype=int32)
Tensor("zeros_5:0", shape=(4, 4), dtype=float32)
Tensor("ones_5:0", shape=(4, 4), dtype=float32)
10
```

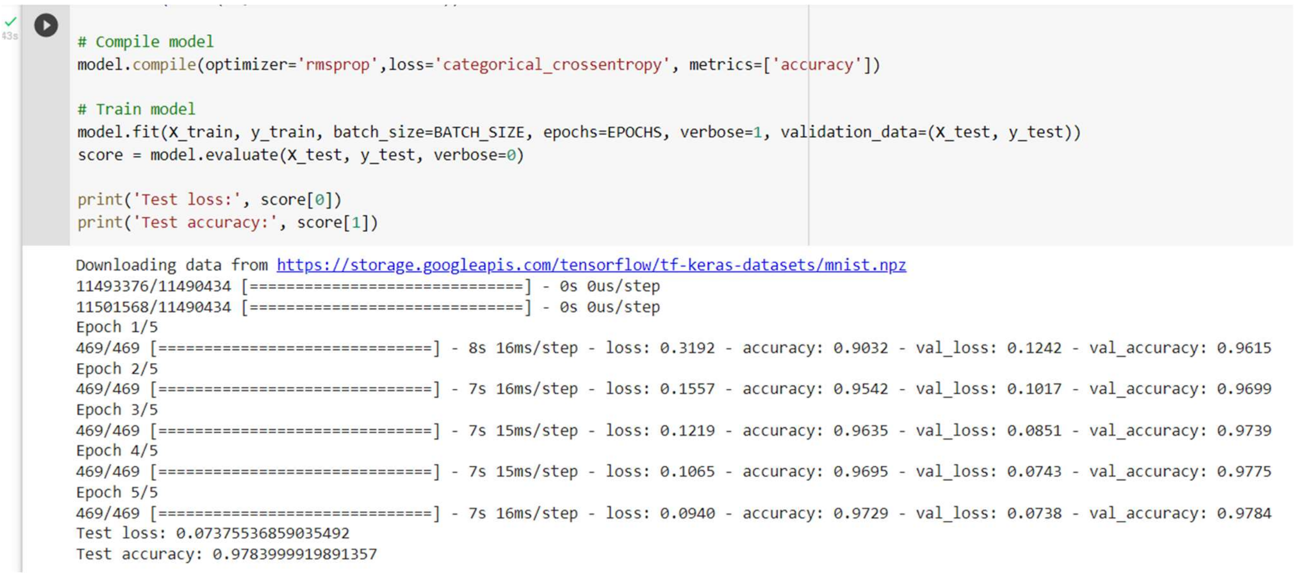


```
[[10 10 10 10]
 [10 10 10 10]
 [10 10 10 10]
 [10 10 10 10]]


[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]


[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]


[[ 0.40043235 -0.8841352   0.76537347  2.018143  ]
 [-0.14721796  1.3166403  -1.2358534   1.4759924 ]
 [ 0.31341892 -1.9864967   0.26148292  0.75342405]
 [-0.7523901   1.6198123  -0.1594997  -0.96358955]]


[[0.49050653 0.6892667  0.38505852 0.41575778]
 [0.13342476 0.9519206  0.5305511  0.60743093]
 [0.560357   0.5408455  0.43818998 0.8570508 ]
 [0.15408516 0.36614013 0.44601762 0.28763354]]
```

Sequential Model -:



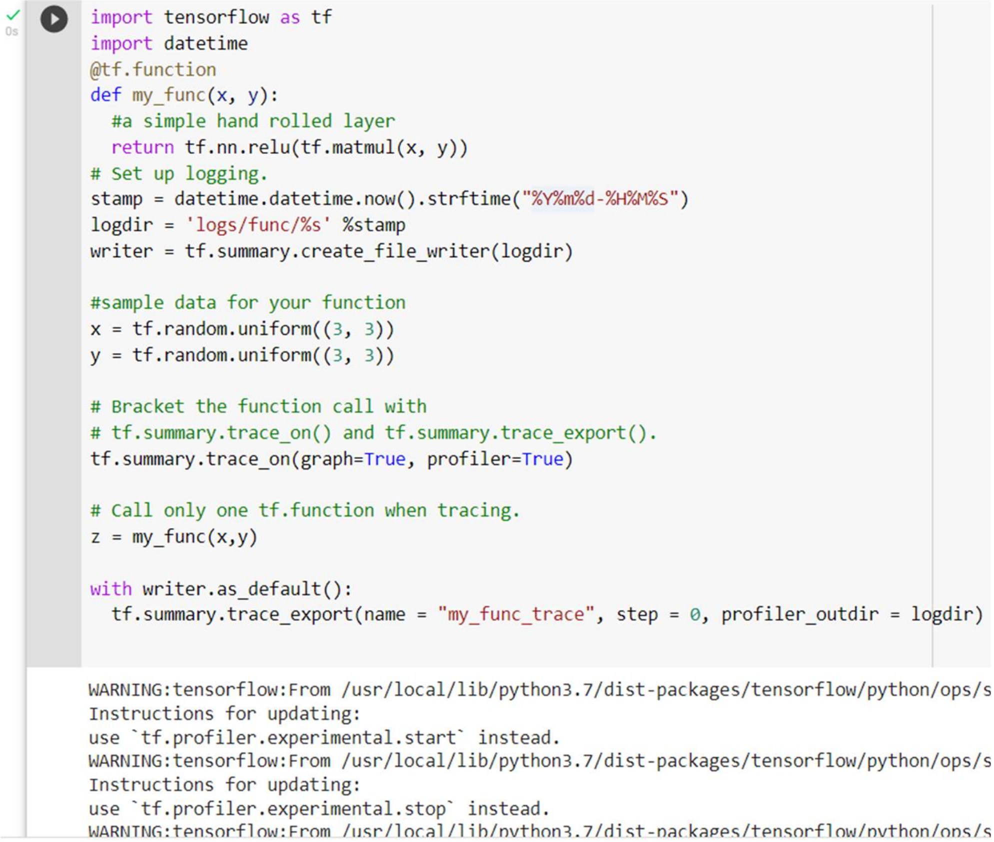
```
from keras import models
from keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical
from keras.datasets import mnist

NUM_ROWS = 28
NUM_COLS = 28
NUM_CLASSES = 10
BATCH_SIZE = 128
EPOCHS = 5

(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Reshape data

X_train = X_train.reshape((X_train.shape[0], NUM_ROWS * NUM_COLS))
X_train = X_train.astype('float32') / 255
X_test = X_test.reshape((X_test.shape[0], NUM_ROWS * NUM_COLS))
X_test = X_test.astype('float32') / 255

# Categorically encode labels
y_train = to_categorical(y_train, NUM_CLASSES)
y_test = to_categorical(y_test, NUM_CLASSES)

# Build neural network
model = models.Sequential()
model.add(Dense(512, activation='relu', input_shape=(NUM_ROWS * NUM_COLS,)))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))
```



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

# Train model
model.fit(X_train, y_train, batch_size=BATCH_SIZE, epochs=EPOCHS, verbose=1, validation_data=(X_test, y_test))
score = model.evaluate(X_test, y_test, verbose=0)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

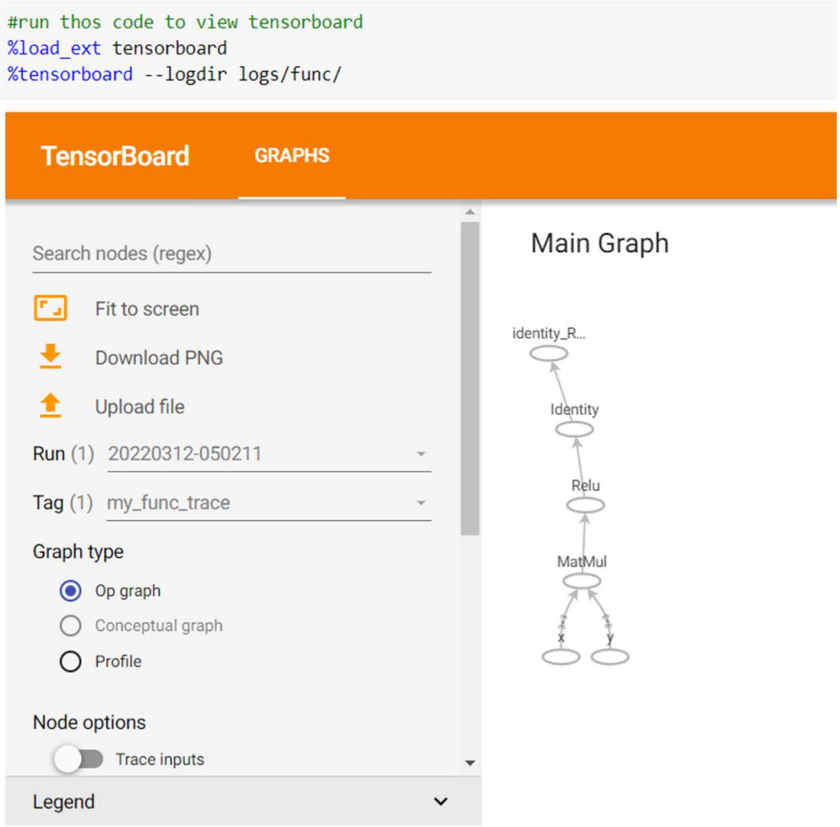
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
Epoch 1/5
469/469 [==============================] - 8s 16ms/step - loss: 0.3192 - accuracy: 0.9032 - val_loss: 0.1242 - val_accuracy: 0.9615
Epoch 2/5
469/469 [==============================] - 7s 16ms/step - loss: 0.1557 - accuracy: 0.9542 - val_loss: 0.1017 - val_accuracy: 0.9699
Epoch 3/5
469/469 [==============================] - 7s 15ms/step - loss: 0.1219 - accuracy: 0.9635 - val_loss: 0.0851 - val_accuracy: 0.9739
Epoch 4/5
469/469 [==============================] - 7s 15ms/step - loss: 0.1065 - accuracy: 0.9695 - val_loss: 0.0743 - val_accuracy: 0.9775
Epoch 5/5
469/469 [==============================] - 7s 16ms/step - loss: 0.0940 - accuracy: 0.9729 - val_loss: 0.0738 - val_accuracy: 0.9784
Test loss: 0.07375536859035492
Test accuracy: 0.9783999919891357
```

Computational Graph Tensor Board -:



```
import tensorflow as tf
import datetime
@tf.function
def my_func(x, y):
  #a simple hand rolled layer
  return tf.nn.relu(tf.matmul(x, y))
# Set up logging.
stamp = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
logdir = 'logs/func/%s' %stamp
writer = tf.summary.create_file_writer(logdir)

#sample data for your function
x = tf.random.uniform((3, 3))
y = tf.random.uniform((3, 3))

# Bracket the function call with
# tf.summary.trace_on() and tf.summary.trace_export().
tf.summary.trace_on(graph=True, profiler=True)

# Call only one tf.function when tracing.
z = my_func(x,y)

with writer.as_default():
  tf.summary.trace_export(name = "my_func_trace", step = 0, profiler_outdir = logdir)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow/python/ops/s
Instructions for updating:
use `tf.profiler.experimental.start` instead.
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow/python/ops/s
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow/python/ops/s
```



```
#run thos code to view tensorboard
%load_ext tensorboard
%tensorboard --logdir logs/func/
```

Run (1) 20220312-050211

Tag (1) my_func_trace

Graph type

● Op graph

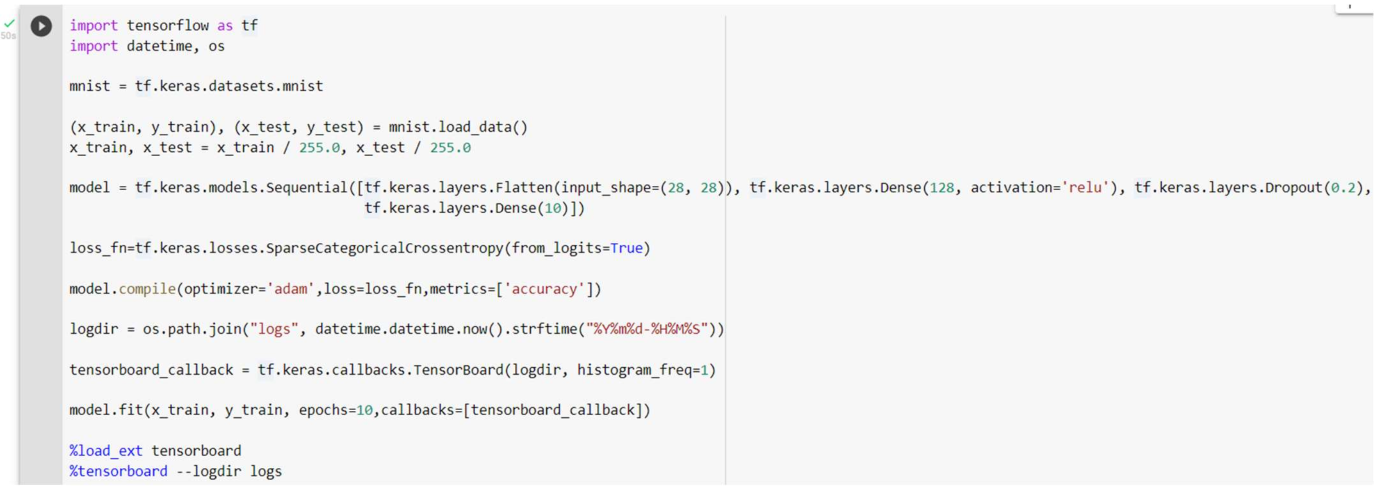
○ Conceptual graph

○ Profile

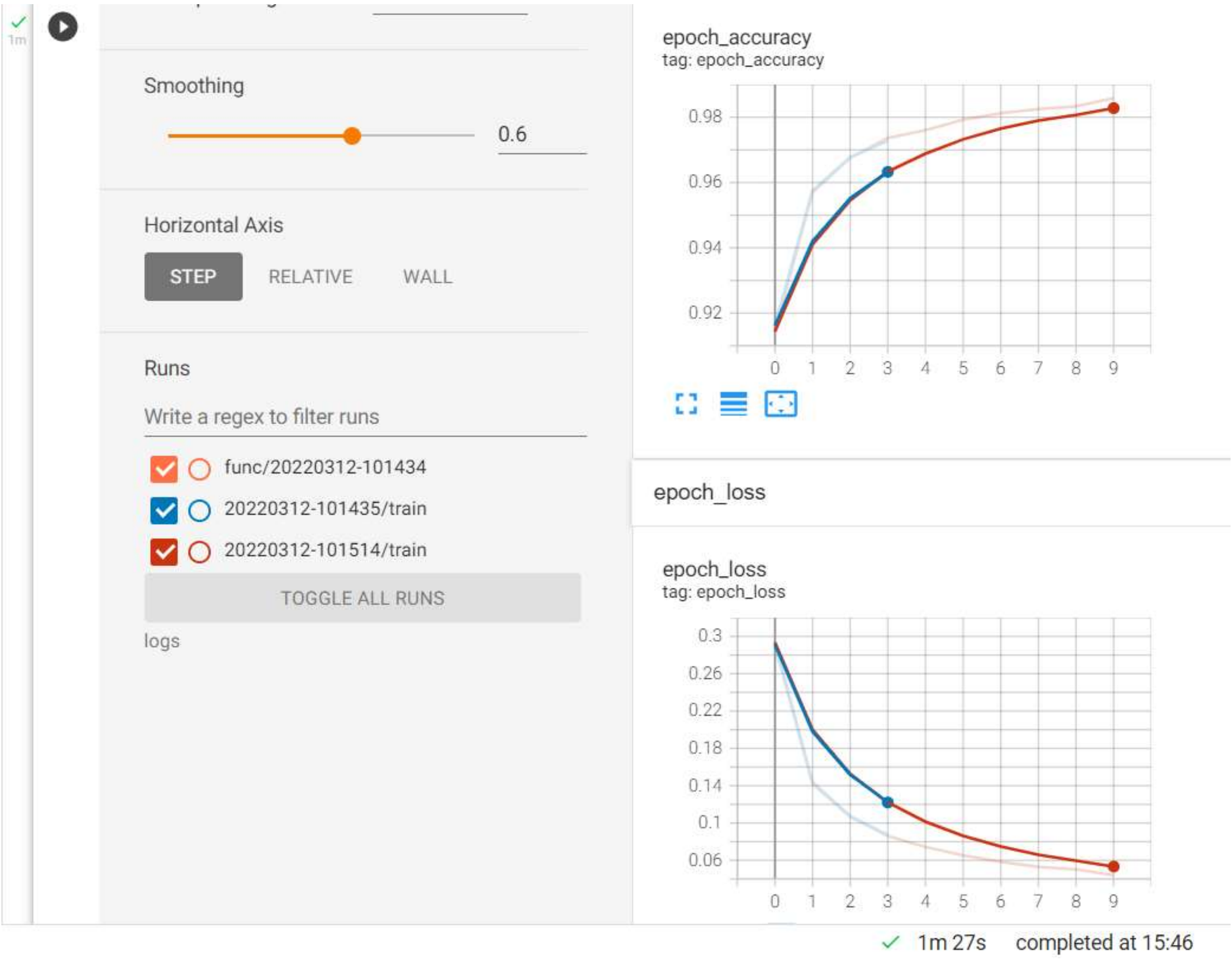
Node options

Trace inputs

Sequential Model Tensor board -:



```
import tensorflow as tf
import datetime, os

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28, 28)), tf.keras.layers.Dense(128, activation='relu'), tf.keras.layers.Dropout(0.2),
                                    tf.keras.layers.Dense(10)])

loss_fn=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

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

logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))

tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir, histogram_freq=1)

model.fit(x_train, y_train, epochs=10,callbacks=[tensorboard_callback])

%load_ext tensorboard
%tensorboard --logdir logs
```



Smoothing

0.6

Horizontal Axis

STEP RELATIVE WALL

Runs

Write a regex to filter runs

☑ ○ func/20220312-101434

☑ ○ 20220312-101435/train

☑ ○ 20220312-101514/train

TOGGLE ALL RUNS

logs

epoch_accuracy

tag: epoch_accuracy

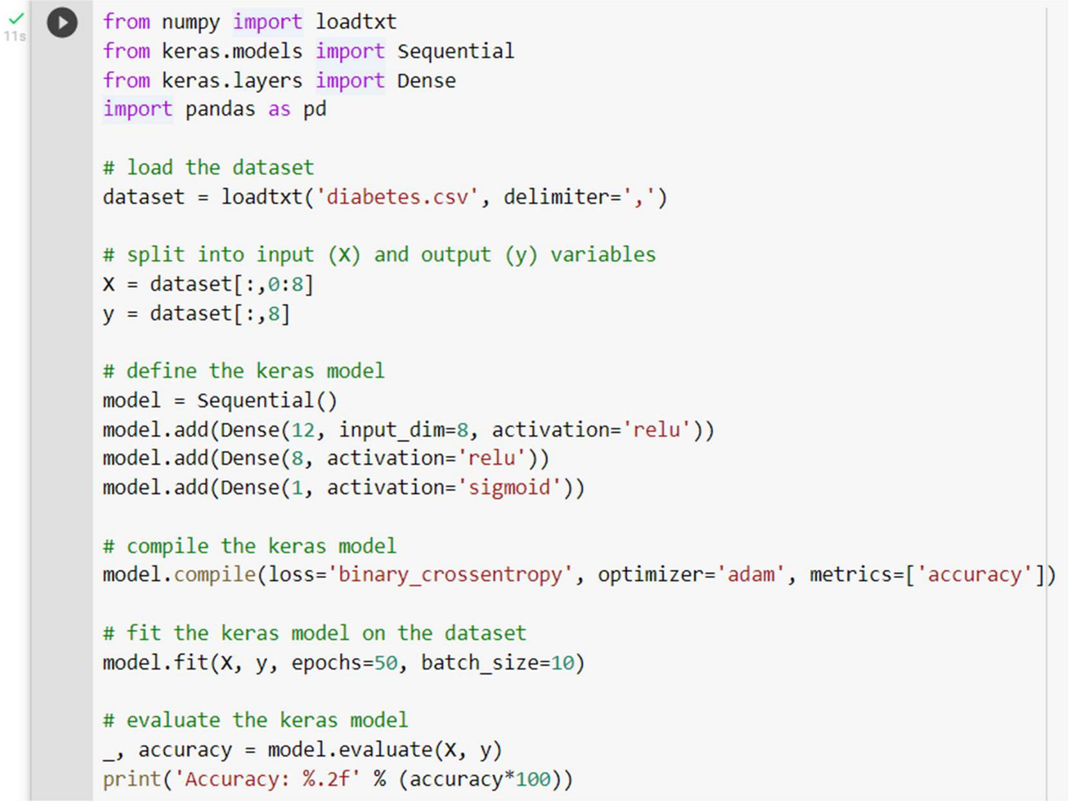
epoch_loss

epoch_loss

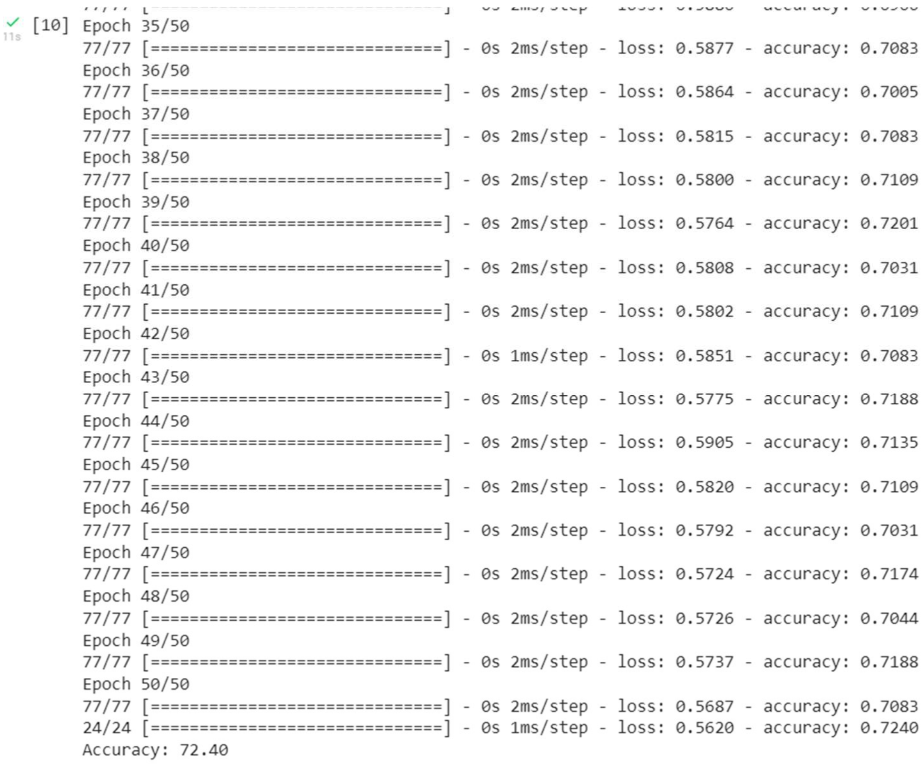
tag: epoch_loss

✓ 1m 27s completed at 15:46

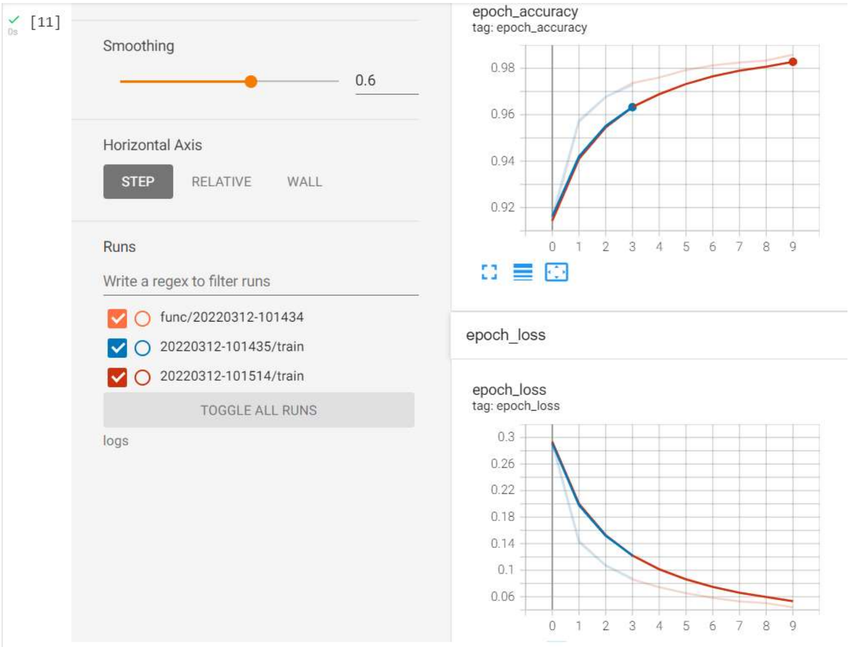
Sequential Model on PIMA Diabetes Dataset -:



```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
import pandas as pd

# load the dataset
dataset = loadtxt('diabetes.csv', delimiter=',')

# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]

# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

# fit the keras model on the dataset
model.fit(X, y, epochs=50, batch_size=10)

# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```



```
Epoch 35/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5877 - accuracy: 0.7083
Epoch 36/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5864 - accuracy: 0.7005
Epoch 37/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5815 - accuracy: 0.7083
Epoch 38/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5800 - accuracy: 0.7109
Epoch 39/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5764 - accuracy: 0.7201
Epoch 40/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5808 - accuracy: 0.7031
Epoch 41/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5802 - accuracy: 0.7109
Epoch 42/50
77/77 [==============================] - 0s 1ms/step - loss: 0.5851 - accuracy: 0.7083
Epoch 43/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5775 - accuracy: 0.7188
Epoch 44/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5905 - accuracy: 0.7135
Epoch 45/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5820 - accuracy: 0.7109
Epoch 46/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5792 - accuracy: 0.7031
Epoch 47/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5724 - accuracy: 0.7174
Epoch 48/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5726 - accuracy: 0.7044
Epoch 49/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5737 - accuracy: 0.7188
Epoch 50/50
77/77 [==============================] - 0s 2ms/step - loss: 0.5687 - accuracy: 0.7083
24/24 [==============================] - 0s 1ms/step - loss: 0.5620 - accuracy: 0.7240
Accuracy: 72.40
```



[11]

Smoothing

0.6

Horizontal Axis

STEP RELATIVE WALL

Runs

Write a regex to filter runs

☑ ○ func/20220312-101434

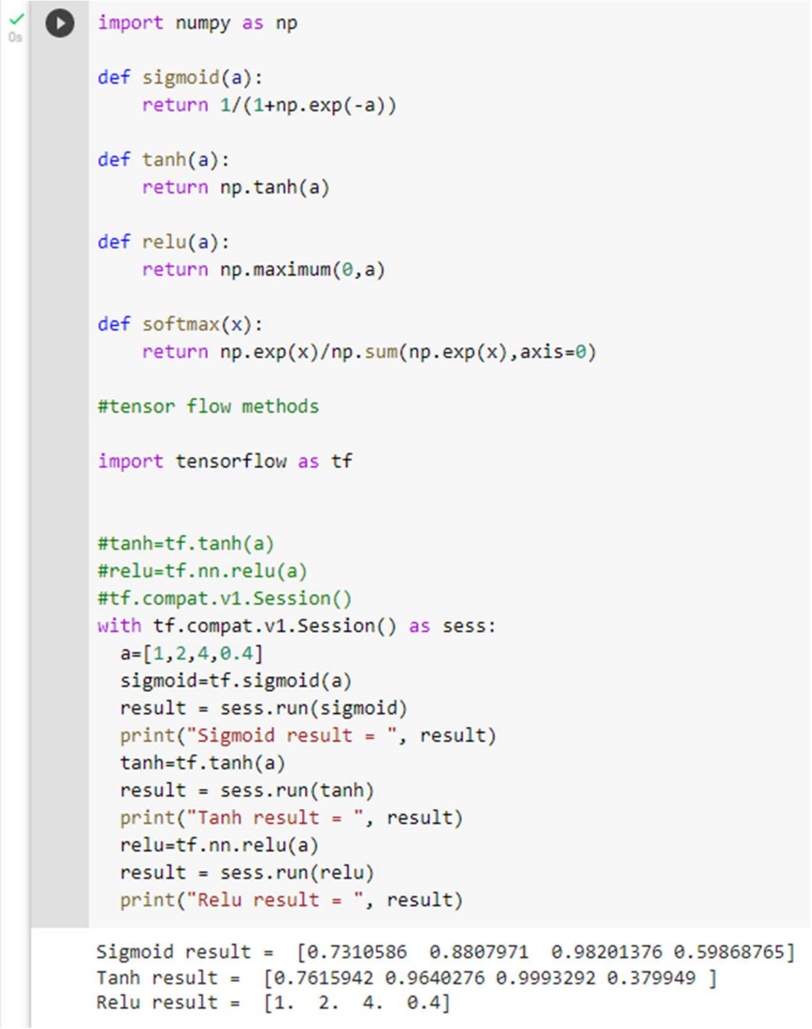
☑ ○ 20220312-101435/train

☑ ○ 20220312-101514/train

TOGGLE ALL RUNS

logs

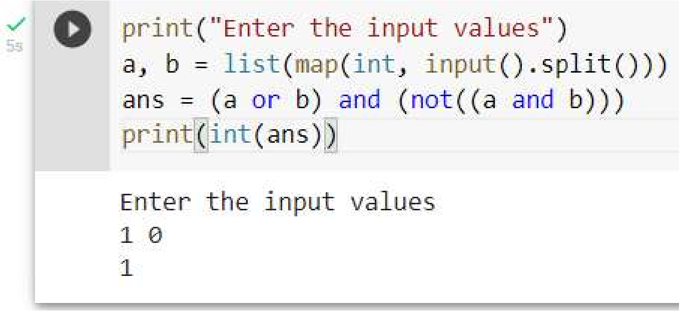
Activation Functions -:



```
import numpy as np

def sigmoid(a):
    return 1/(1+np.exp(-a))

def tanh(a):
    return np.tanh(a)

def relu(a):
    return np.maximum(0,a)

def softmax(x):
    return np.exp(x)/np.sum(np.exp(x),axis=0)

#tensor flow methods

import tensorflow as tf


#tanh=tf.tanh(a)
#relu=tf.nn.relu(a)
#tf.compat.v1.Session()
with tf.compat.v1.Session() as sess:
  a=[1,2,4,0.4]
  sigmoid=tf.sigmoid(a)
  result = sess.run(sigmoid)
  print("Sigmoid result = ", result)
  tanh=tf.tanh(a)
  result = sess.run(tanh)
  print("Tanh result = ", result)
  relu=tf.nn.relu(a)
  result = sess.run(relu)
  print("Relu result = ", result)
```

```
Sigmoid result =  [0.7310586  0.8807971  0.98201376 0.59868765]
Tanh result =  [0.7615942 0.9640276 0.9993292 0.379949 ]
Relu result =  [1.  2.  4.  0.4]
```

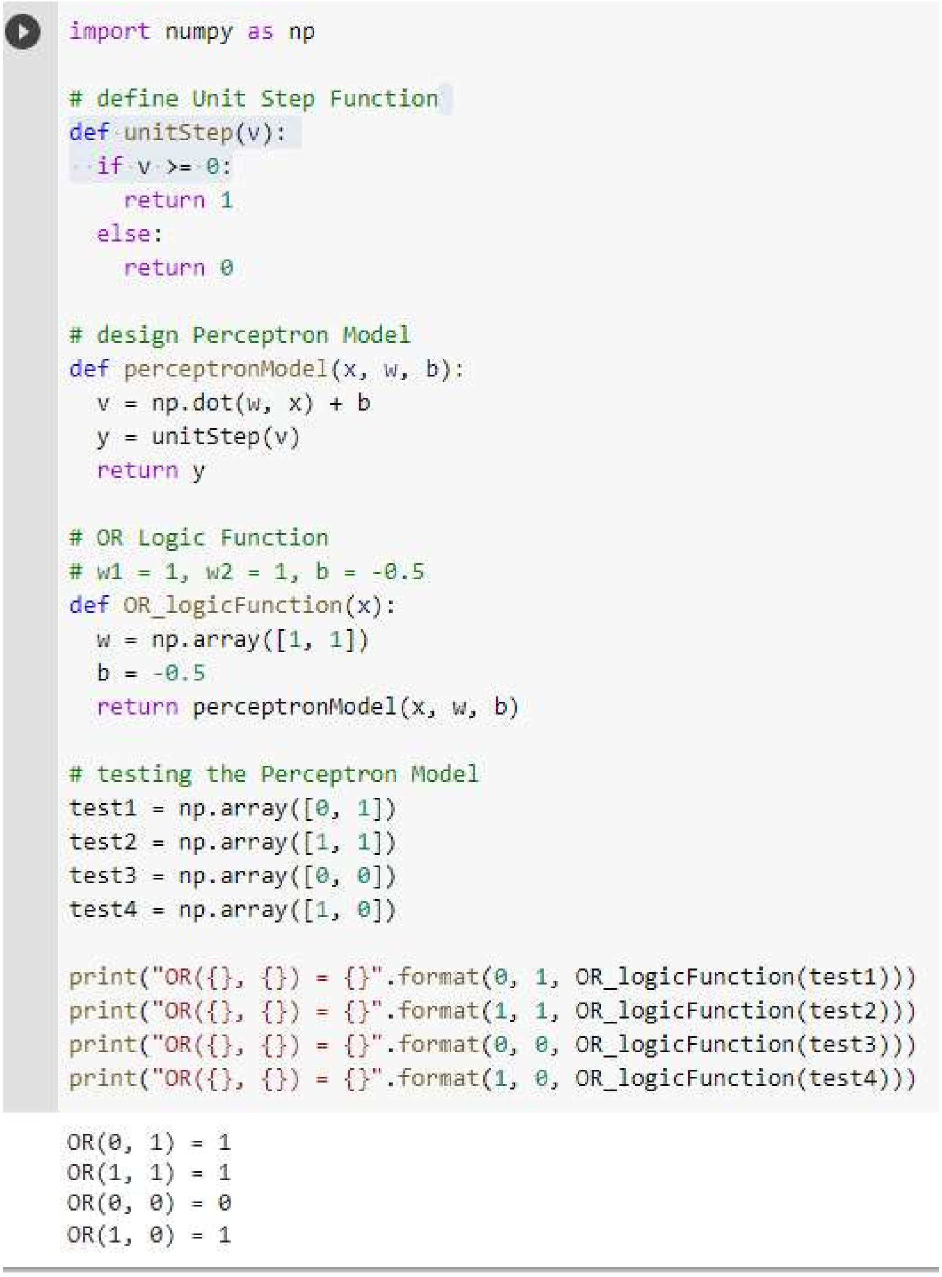
Implementation of XOR using Basic Gates -:



```
print("Enter the input values")
a, b = list(map(int, input().split()))
ans = (a or b) and (not((a and b)))
print(int(ans))
```

```
Enter the input values
1 0
1
```

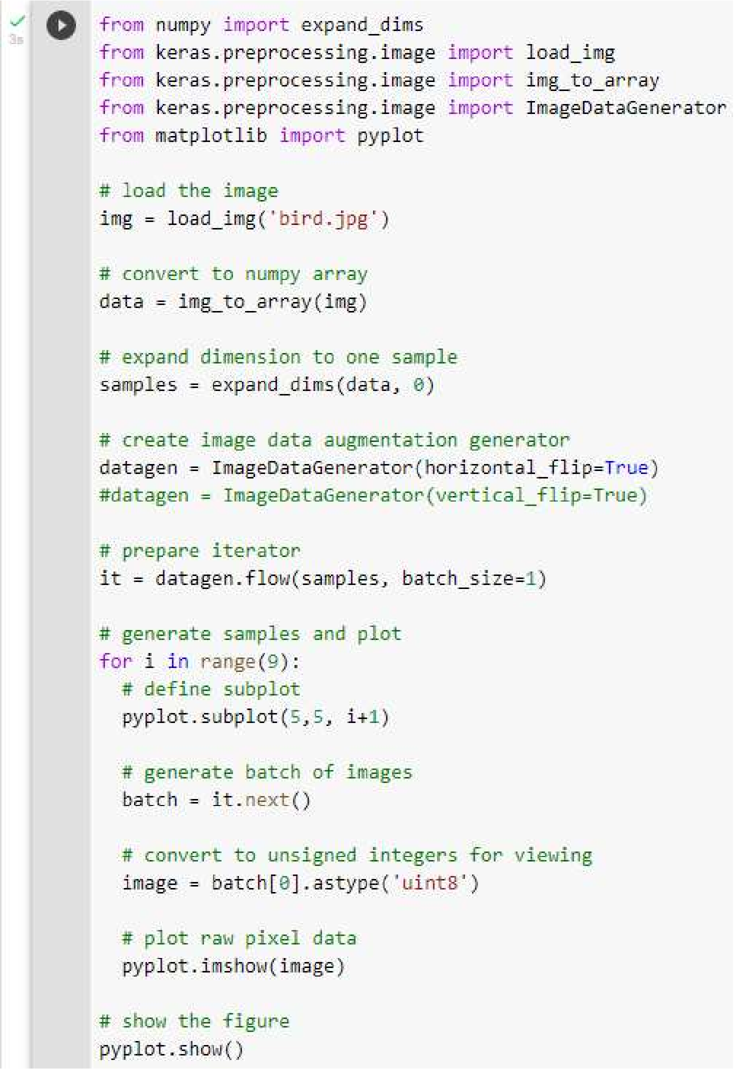
Implementation of OR -:



```
import numpy as np

# define Unit Step Function
def unitStep(v):
  if v >= 0:
    return 1
  else:
    return 0

# design Perceptron Model
def perceptronModel(x, w, b):
  v = np.dot(w, x) + b
  y = unitStep(v)
  return y

# OR Logic Function
# w1 = 1, w2 = 1, b = -0.5
def OR_logicFunction(x):
  w = np.array([1, 1])
  b = -0.5
  return perceptronModel(x, w, b)

# testing the Perceptron Model
test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("OR({}, {}) = {}".format(0, 1, OR_logicFunction(test1)))
print("OR({}, {}) = {}".format(1, 1, OR_logicFunction(test2)))
print("OR({}, {}) = {}".format(0, 0, OR_logicFunction(test3)))
print("OR({}, {}) = {}".format(1, 0, OR_logicFunction(test4)))
```

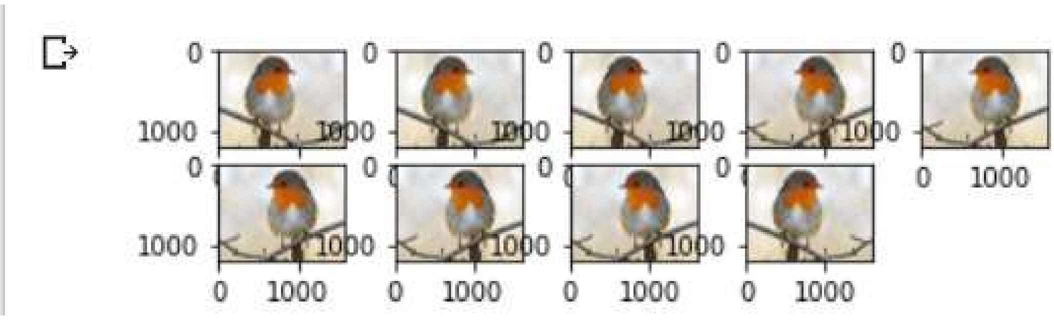
```
OR(0, 1) = 1
OR(1, 1) = 1
OR(0, 0) = 0
OR(1, 0) = 1
```

Data Augmentation -:



```
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot

# load the image
img = load_img('bird.jpg')

# convert to numpy array
data = img_to_array(img)

# expand dimension to one sample
samples = expand_dims(data, 0)

# create image data augmentation generator
datagen = ImageDataGenerator(horizontal_flip=True)
#datagen = ImageDataGenerator(vertical_flip=True)

# prepare iterator
it = datagen.flow(samples, batch_size=1)

# generate samples and plot
for i in range(9):
  # define subplot
  pyplot.subplot(5,5, i+1)

  # generate batch of images
  batch = it.next()

  # convert to unsigned integers for viewing
  image = batch[0].astype('uint8')

  # plot raw pixel data
  pyplot.imshow(image)

# show the figure
pyplot.show()
```

Output -:


0
1000
0
1000