SMOKE TEST

Optimized Tensorflow 1.15

Date Prepared: April 2020

Document Information

Project Name	**EPIC Accelerator Deployment & Integration Services**		
Project Owner		Document Version No	0.1
Quality Review Method			
Prepared By	Prasad Adireddi	Preparation Date	April 2020
Reviewed By		Review Date	

Table of Contents

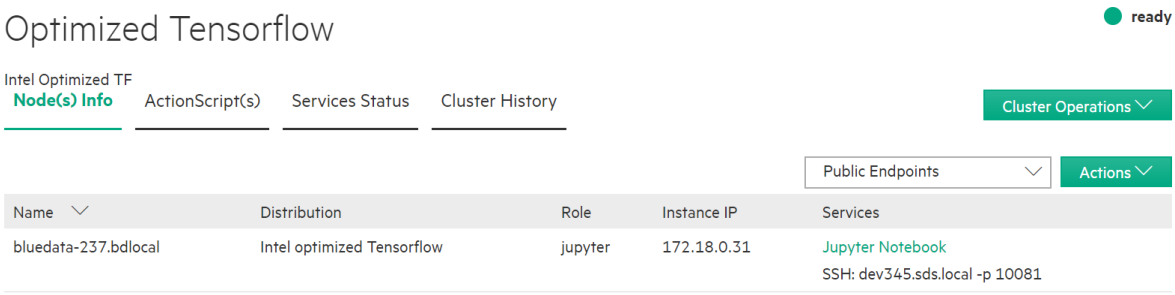
Table of Tables

NO TABLE OF FIGURES ENTRIES FOUND.

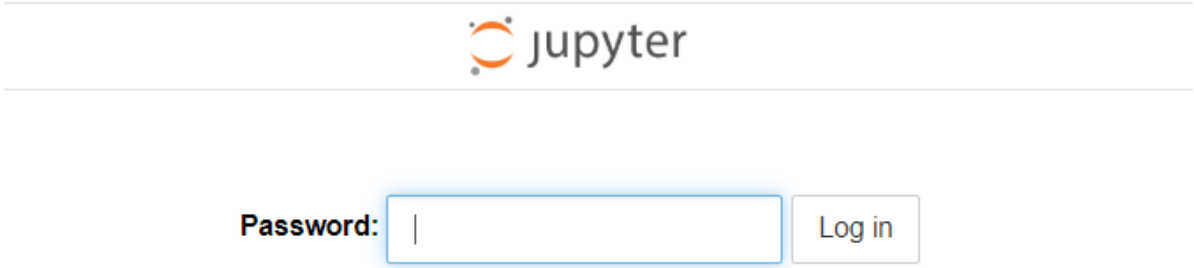
1 LOGIN TO JUPYTERHUB WEB UI

Login to the JupyterHub Web UI, using the following steps:

1. From the Cluster page under Services, click on **Jupyter Notebook**



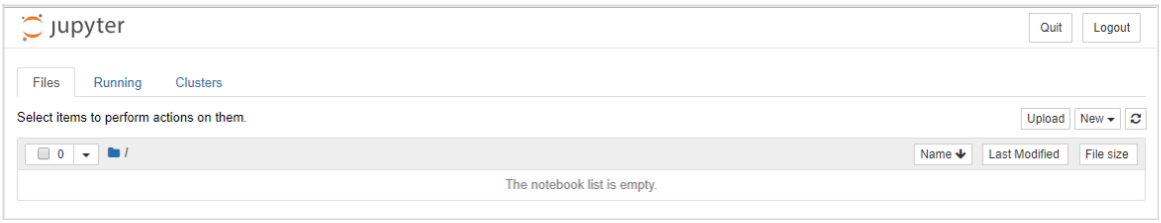
2. It will open-up a new tab with JupyterHub login page, login using your credentials



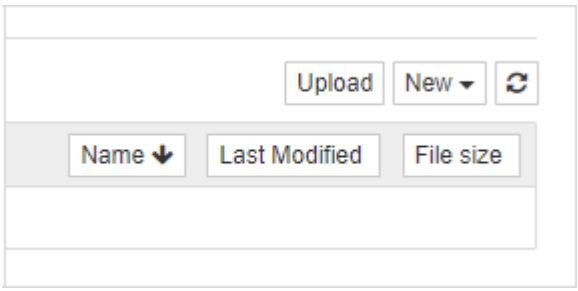
2 CREATE NOTEBOOK

To create a new notebook, in JupyterHub use the following steps:

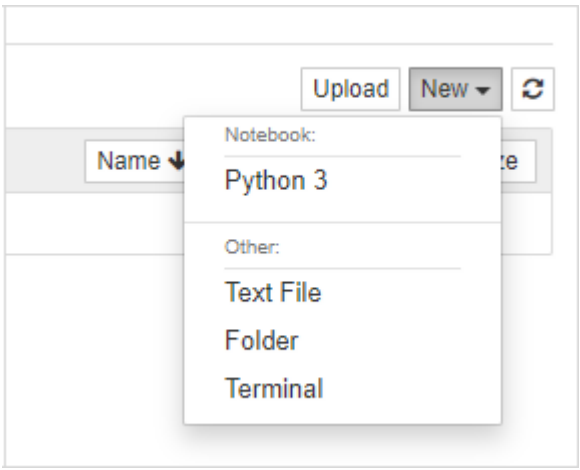
1. Once you log into JupyterHub, you will get the dashboard like below:



2. Click on **New** drop-down button, to create a new notebook



3. Select **Python3**, for upcoming tasks

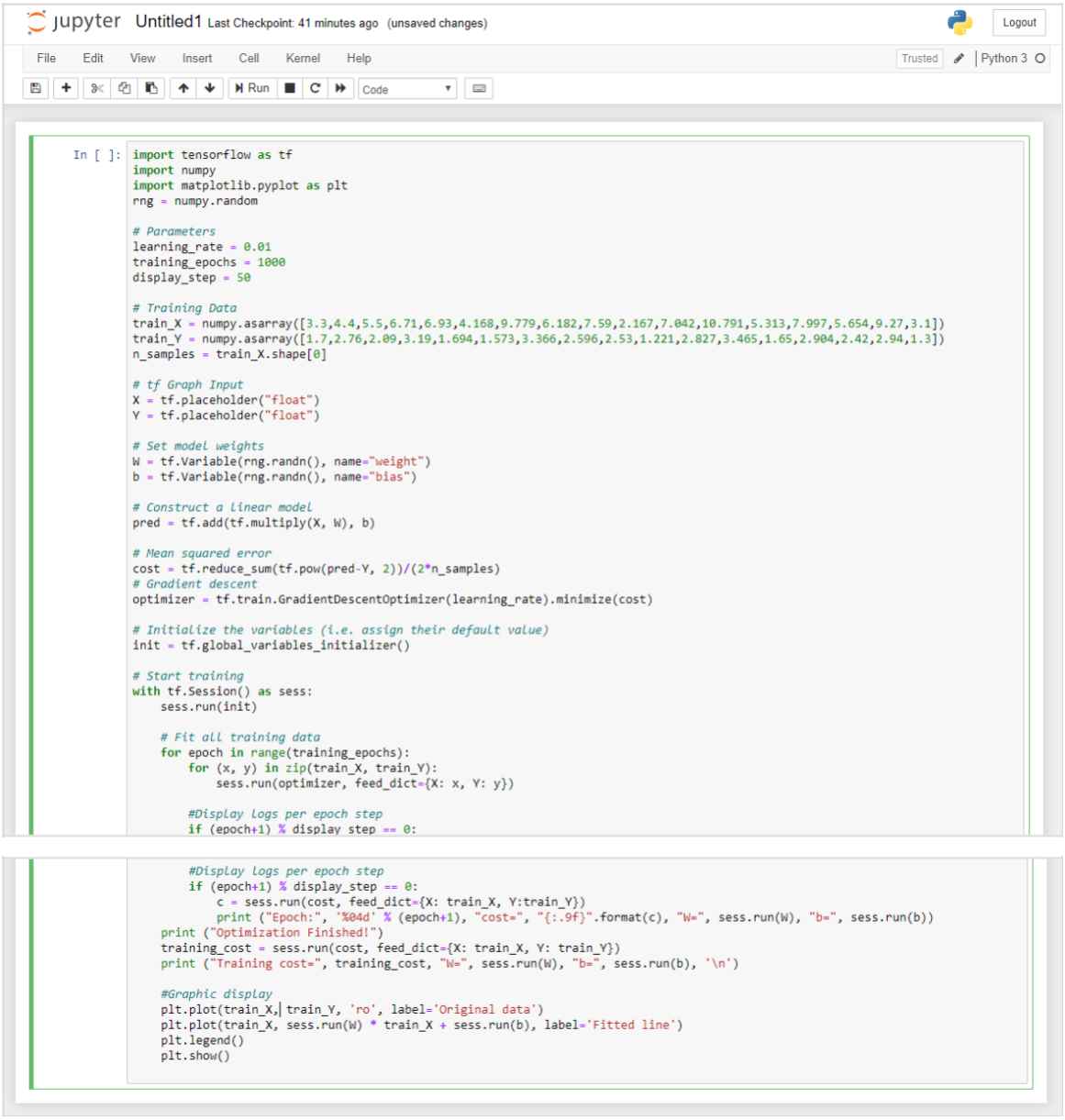


3 LINEAR REGRESSION

In this section, we are testing a Linear Regression example:

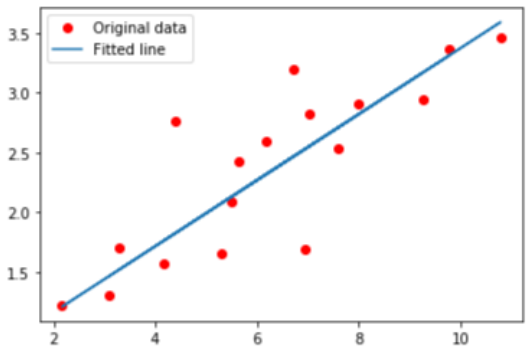
1. In Python3 notebook, add the following code in the code cell:

```
import tensorflow as tf
import numpy
import matplotlib.pyplot as plt
rng = numpy.random

#  Parameters learning_rate = 0.01 training_epochs = 1000 display_step = 50 train_X =



numpy.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,7.042,10.791,5.313 ,7.997,5.654,9.27,3.1])



train_Y =
numpy.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,2.827,3.465,1.6
5,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]
X = tf.placeholder("float") Y
= tf.placeholder("float")
W = tf.Variable(rng.randn(), name="weight") b
= tf.Variable(rng.randn(), name="bias")

# Construct a linear model
pred = tf.add(tf.multiply(X, W), b)
# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y, 2))/(2*n_samples)
# Gradient descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Initialize the variables (i.e. assign their default
value) init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:
    sess.run(init)
    # Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})
        #Display logs per epoch step
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X, Y:train_Y})
            print ("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(c),
"W=", sess.run(W), "b=", sess.run(b))
    print ("Optimization Finished!")
    training_cost = sess.run(cost, feed_dict={X: train_X, Y: train_Y})
    print ("Training cost=", training_cost, "W=", sess.run(W), "b=",
sess.run(b), '\n')

    #Graphic display
    plt.plot(train_X, train_Y, 'ro', label='Original data')
    plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
    plt.legend()
    plt.show()
```



2. Click on **Run**, to view the output

```
Epoch: 0050 cost= 0.100056902 W= 0.33464253 b= 0.1896305
Epoch: 0100 cost= 0.097387671 W= 0.32958433 b= 0.22601922
Epoch: 0150 cost= 0.095026977 W= 0.3248268 b= 0.26024374
Epoch: 0200 cost= 0.092939079 W= 0.3203523 b= 0.29243314
Epoch: 0250 cost= 0.091092519 W= 0.31614393 b= 0.32270795
Epoch: 0300 cost= 0.089459457 W= 0.312186 b= 0.35118157
Epoch: 0350 cost= 0.088015184 W= 0.30846328 b= 0.37796235
Epoch: 0400 cost= 0.086737908 W= 0.30496192 b= 0.40315053
Epoch: 0450 cost= 0.085608326 W= 0.30166888 b= 0.42684048
Epoch: 0500 cost= 0.084609419 W= 0.29857177 b= 0.44912106
Epoch: 0550 cost= 0.083726034 W= 0.29565877 b= 0.47007725
Epoch: 0600 cost= 0.082944840 W= 0.29291892 b= 0.48978683
Epoch: 0650 cost= 0.082254037 W= 0.2903422 b= 0.5083237
Epoch: 0700 cost= 0.081643179 W= 0.2879186 b= 0.52575904
Epoch: 0750 cost= 0.081103034 W= 0.28563923 b= 0.54215676
Epoch: 0800 cost= 0.080625392 W= 0.28349537 b= 0.5575797
Epoch: 0850 cost= 0.080203041 W= 0.28147882 b= 0.5720862
Epoch: 0900 cost= 0.079829641 W= 0.27958238 b= 0.5857292
Epoch: 0950 cost= 0.079499461 W= 0.27779865 b= 0.5985611
Epoch: 1000 cost= 0.079207547 W= 0.27612087 b= 0.6106305
Optimization Finished!
Training cost= 0.07920755 W= 0.27612087 b= 0.6106305
```



Note: Try adding **%matplotlib inline** at the top of the code if you get, <Figure size 640x480 with 1 Axes> instead of graph.

4 NEURAL NETWORK

In this section, we will test some neural network example:

1. In a Python3 notebook, add the below code in the code cell:

```
import numpy as np

class NeuralNetwork():
    def __init__(self):
        np.random.seed(1)
        self.synaptic_weights = 2 * np.random.random((3, 1)) - 1

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

    def sigmoid_derivative(self, x):
        return x * (1 - x)

    def train(self, training_inputs, training_outputs, training_iterations):
        for iteration in range(training_iterations):
            output = self.think(training_inputs)
            error = training_outputs - output
            adjustments = np.dot(training_inputs.T, error *
self.sigmoid_derivative(output))
            self.synaptic_weights += adjustments

    def think(self, inputs):
        inputs = inputs.astype(float)
        output = self.sigmoid(np.dot(inputs, self.synaptic_weights))
        return output


if __name__ == "__main__":
    neural_network = NeuralNetwork()
    print("Beginning Randomly Generated Weights: ")
    print(neural_network.synaptic_weights)
    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
    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!")
```

2. Click on **Run**, to view the 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:  0
User Input Three:  0
Considering New Situation:  1 0 0
New Output data:
[0.9999584]
Wow, we did it!
```

4.1 Single Neuron Neural Network

1. Add the below code in the code cell:

```
from numpy import exp, array, random, dot, tanh

class NeuralNetwork():

    def __init__(self):
        random.seed(1)
        self.weight_matrix = 2 * random.random((3, 1)) - 1

    def tanh(self, x):
        return tanh(x)

    def tanh_derivative(self, x):
        return 1.0 - tanh(x) ** 2

    def forward_propagation(self, inputs):
        return self.tanh(dot(inputs, self.weight_matrix))

    def train(self, train_inputs, train_outputs,
                            num_train_iterations)
                            :
        for iteration in range(num_train_iterations): output
            = self.forward_propagation(train_inputs) error =
            train_outputs - output
            adjustment = dot(train_inputs.T, error *
                             self.tanh_derivative(output)
                             )
            self.weight_matrix += adjustment

if __name__ == "__main__":
    neural_network = NeuralNetwork()
    print ('Random weights at the start of training')
    print (neural_network.weight_matrix)
    train_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])
    train_outputs = array([[0, 1, 1, 0]]).T
    neural_network.train(train_inputs, train_outputs, 10000)
    print ('New weights after training')
    print (neural_network.weight_matrix)
    print ("Testing network on new examples ->")
    print (neural_network.forward_propagation(array([1, 0, 0])))
```

2. Click on **Run**, to view the output

```
Random weights at the start of training
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
New weights after training
[[5.39428067]
 [0.19482422]
 [0.34317086]]
Testing network on new examples ->
[0.99995873]
```

4.2 One Dimensional Vector

1. Add the below code in the code cell:

```
import numpy as np
input_vector = np.array([2, 4, 11])
print(input_vector)
```

2. Click on **Run**, to view the output

```
[14]: import numpy as np
      input_vector = np.array([2, 4, 11])
      print(input_vector)

      [ 2  4 11]
```

4.3 Two Dimensional Array with one column

1. Add the below code in the code cell:

```
import numpy as np
input_vector = np.array([2, 4, 11])
input_vector = np.array(input_vector, ndmin=2).T
print(input_vector, input_vector.shape)
```

2. Click on **Run**, to view the output

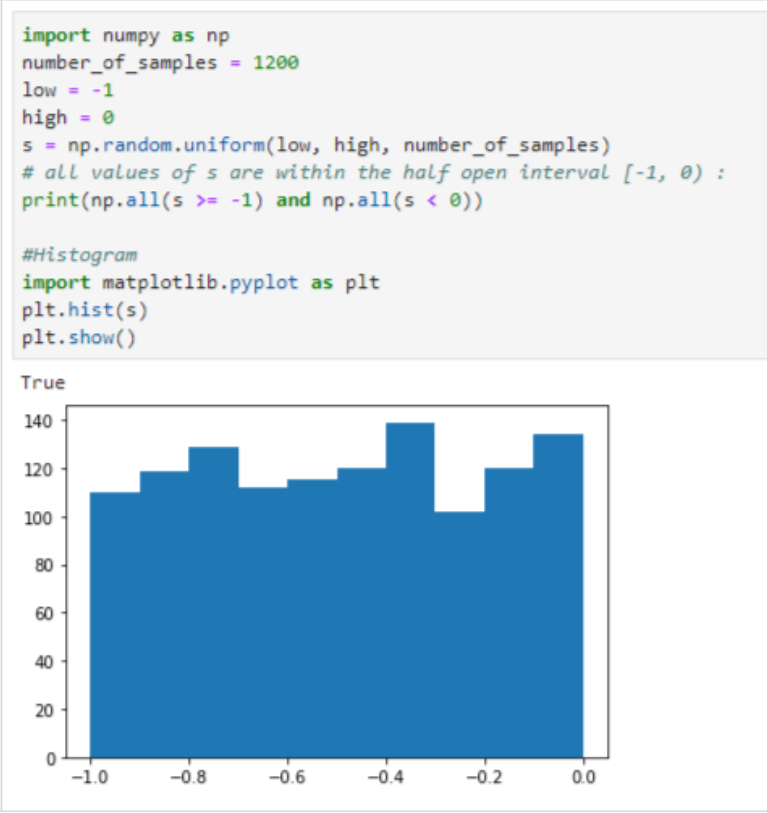
```
import numpy as np
input_vector = np.array([2, 4, 11])
input_vector = np.array(input_vector, ndmin=2).T
print(input_vector, input_vector.shape)

[[ 2]
 [ 4]
 [11]] (3, 1)
```

4.4 Weight Matrices

1. Add the below code in the code cell:

```
import numpy as np
number_of_samples = 1200
low = -1
high = 0
s = np.random.uniform(low, high, number_of_samples)
# all values of s are within the half open interval [-1, 0)
: print(np.all(s >= -1) and np.all(s < 0))

#Histogram
import matplotlib.pyplot as plt
plt.hist(s)
plt.show()
```

2. To execute the code, click on **Run**



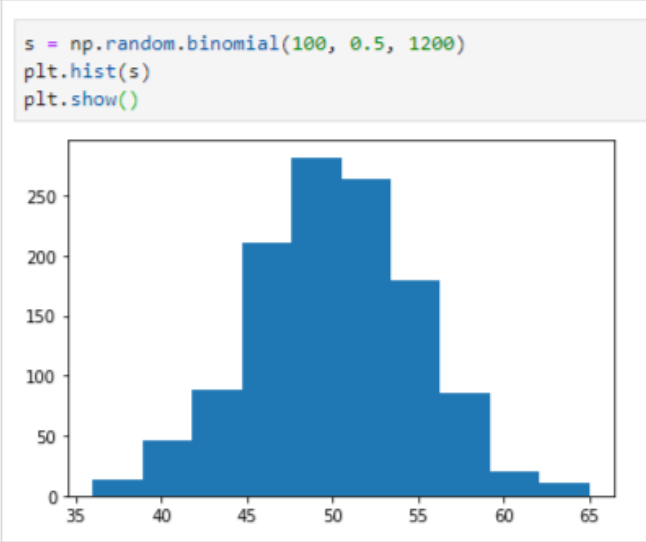
Note: Try adding **%matplotlib inline** at the top of the code if you don't get the graph.

4.5 Binomial Function

1. Add the below code in the code cell:

```
s = np.random.binomial(100, 0.5, 1200)
plt.hist(s)
plt.show()
```

2. To execute click on **Run**



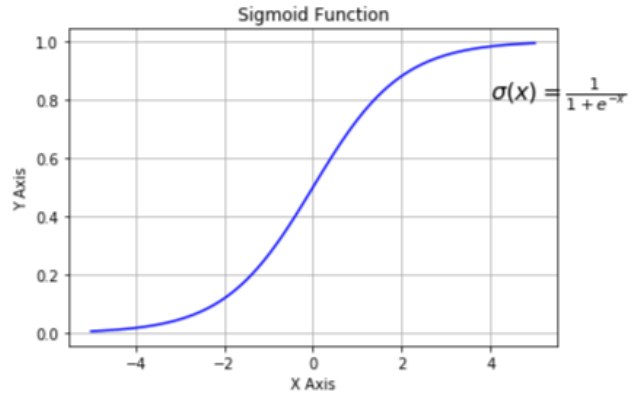
4.6 Sigmoid Function

1. Add the below code in the code cell:

```
import numpy as np
import matplotlib.pyplot as plt
def sigma(x):
    return 1 / (1 + np.exp(-x))
X = np.linspace(-5, 5, 100)
plt.plot(X, sigma(X),'b')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Sigmoid
Function') plt.grid()
plt.text(4, 0.8, r'$\sigma(x)=\frac{1}{1+e^{-x}}$',
fontsize=16) plt.show()
```

2. To execute click on **Run**

```
import numpy as np
import matplotlib.pyplot as plt
def sigma(x):
    return 1 / (1 + np.exp(-x))
X = np.linspace(-5, 5, 100)
plt.plot(X, sigma(X),'b')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Sigmoid Function')
plt.grid()
plt.text(4, 0.8, r'$\sigma(x)=\frac{1}{1+e^{-x}}$', fontsize=16)
plt.show()
```



5 CPU CONSUMPTION

1. In a Python3 notebook, add the below code in the code cell:

```
from __future__ import print_function
import matplotlib
import matplotlib.pyplot as plt
import tensorflow as tf
import time

with tf.Session(config=tf.ConfigProto(log_device_placement=True))
as session:
    start_time = time.time()
    time_taken = time.time() - start_time
    #print(result)

def get_times(maximum_time):

    device_times = {
        "/cpu:0":[]
    }
    matrix_sizes = range(500,50000,50)

    for size in matrix_sizes:
        for device_name in device_times.keys():

            print("####### Calculating on the " + device_name + " #######")

            shape = (size,size)
            data_type = tf.float16
            with tf.device(device_name):
                r1 = tf.random_uniform(shape=shape, minval=0, maxval=1,
dtype=data_type)
                r2 = tf.random_uniform(shape=shape, minval=0, maxval=1,
dtype=data_type)
                dot_operation = tf.matmul(r2, r1)


            with
tf.Session(config=tf.ConfigProto(log_device_placement=True)) as session:
                    start_time = time.time()
                    result = session.run(dot_operation)
                    time_taken = time.time() - start_time
                    print(result)
                    device_times[device_name].append(time_taken)

            print(device_times)

            if time_taken > maximum_time:
                return device_times, matrix_sizes

device_times, matrix_sizes = get_times(1.5)
cpu_times = device_times["/cpu:0"]

plt.plot(matrix_sizes[:len(cpu_times)], cpu_times, 'o-')
plt.ylabel('Time')
plt.xlabel('Matrix size')
plt.show()
```

2. To execute the code, click on **Run**

```
####### Calculating on the /cpu:0 #######
[[122.   126.5  122.4  ... 128.4  122.75 125.56]
 [125.25 124.75 125.9  ... 130.6  125.   127.75]
 [125.2  124.   123.   ... 130.2  122.9  125.7 ]
 ...
 [128.2  128.   126.6  ... 133.8  127.75 129.8 ]
 [117.5  120.9  116.9  ... 125.94 118.2  120.6 ]
 [128.5  128.4  127.94 ... 134.   126.   131.8 ]]
{'/cpu:0': [0.898186445236206]}
####### Calculating on the /cpu:0 #######
[[133.8 136.2 135.8 ... 139.6 142.8 135. ]
 [135.8 135.9 135.2 ... 138.6 143.6 135.1]
 [131.1 132.2 130.8 ... 135.2 139.5 128.4]
 ...
 [136.4 136.5 131.8 ... 136.6 143.9 133.2]
 [137.6 138.8 136.  ... 138.6 146.6 133. ]
 [135.5 132.9 134.6 ... 138.6 138.8 131. ]]
{'/cpu:0': [0.898186445236206, 1.0762956142425537]}
####### Calculating on the /cpu:0 #######
[[151.1 149.8 142.5 ... 154.  149.8 141.5]
 [159.  154.5 145.9 ... 152.  151.1 145.2]
 [153.2 151.5 145.4 ... 153.  150.1 146. ]
 ...
 [147.6 145.2 135.5 ... 149.  145.8 136.8]
 [146.2 151.  140.5 ... 149.9 146.  141.6]
 [147.2 141.5 135.6 ... 145.5 148.1 138.4]]
{'/cpu:0': [0.898186445236206, 1.0762956142425537, 1.306037187576294]}
####### Calculating on the /cpu:0 #######
[[155.9 156.5 161.2 ... 157.1 161.  161.2]
 [164.8 156.5 159.1 ... 158.1 156.8 165.9]
 [159.  156.6 163.1 ... 161.  160.6 164.8]
 ...
 [160.5 160.6 164.9 ... 160.  163.2 162.8]
 [154.5 157.5 156.5 ... 156.1 157.  161.1]
 [159.9 159.6 159.6 ... 156.4 159.9 159.8]]
{'/cpu:0': [0.898186445236206, 1.0762956142425537, 1.306037187576294, 1.6062920093536377]}
```