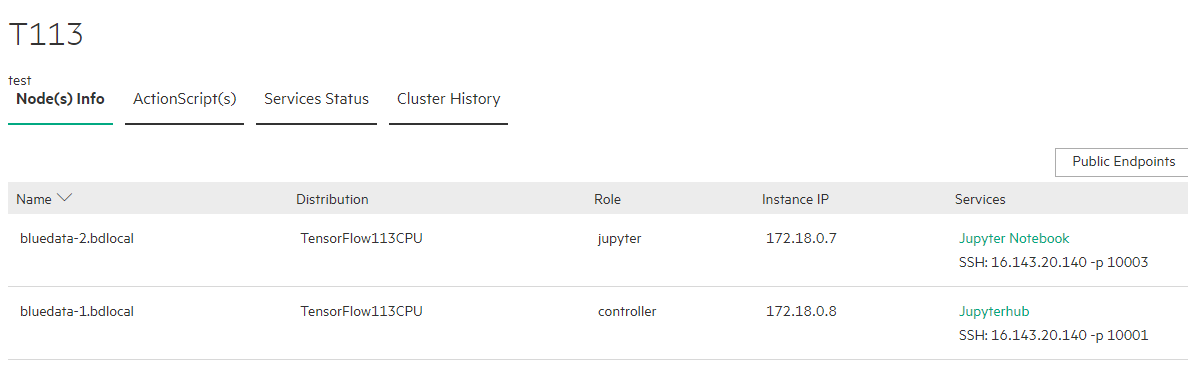
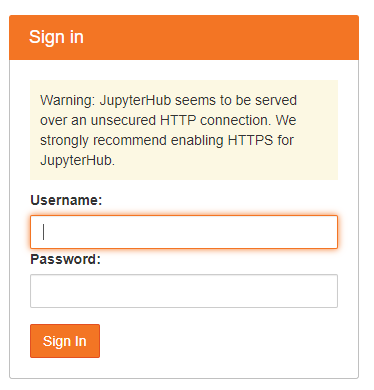
**Tensorflow 1.13 Smoke Test**

**Login to JupyterHub Web UI**

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

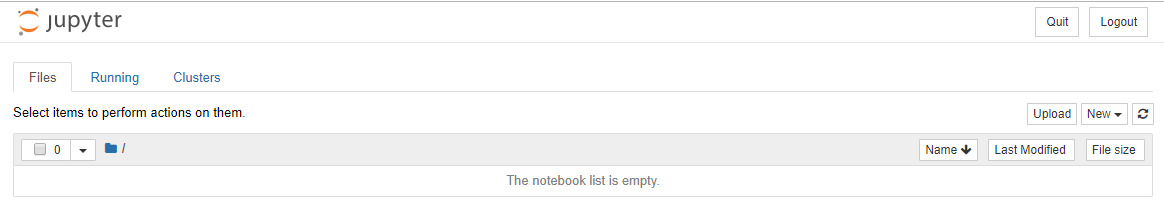


1. It will open-up a new tab with JupyterHub login page, login using your credentials

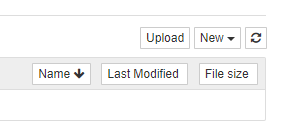


**Create Notebook**

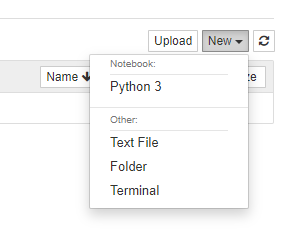
1. You will get JupyterHub dashboard as below:



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



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



1. **Tensorflow in Python3 Testing using JupyterHub:**

**Testing 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

# Training Data

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([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,2.827,3.465,1.65,2.904,2.42,2.94,1.3])

n\_samples = train\_X.shape[0]

# tf Graph Input

X = tf.placeholder("float")

Y = tf.placeholder("float")

# Set model weights

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')

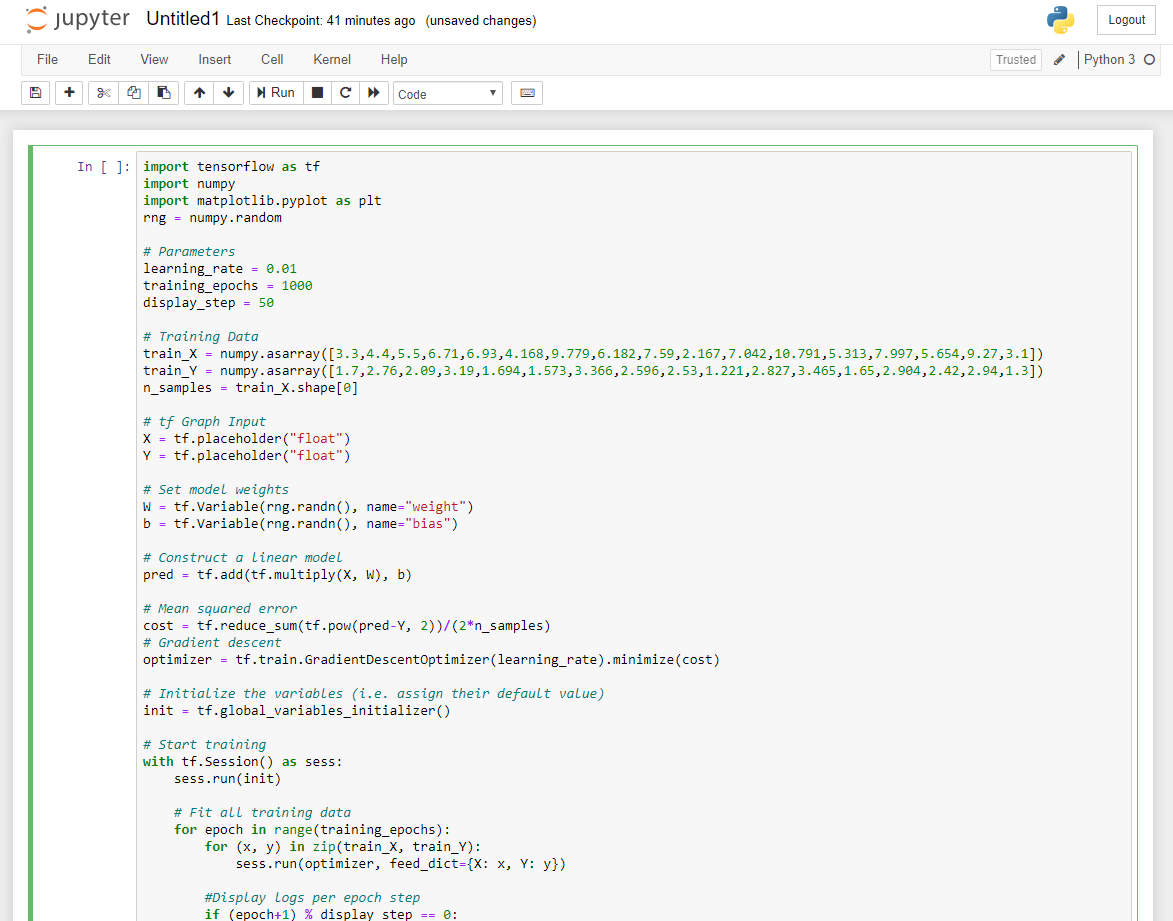
#Graphic display

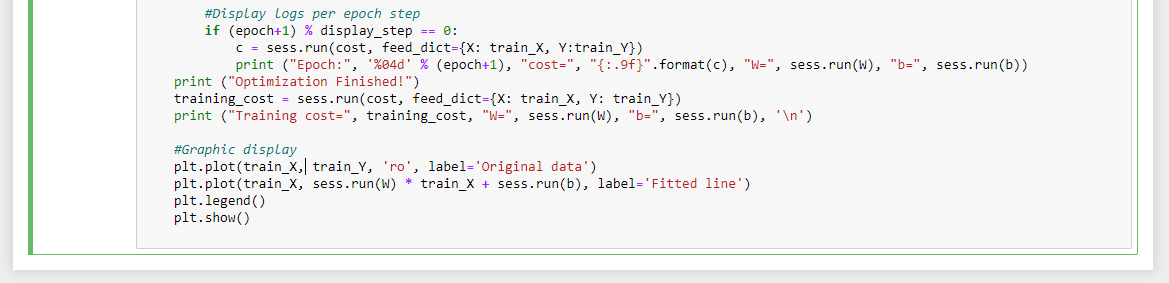
plt.plot(train\_X, train\_Y, 'ro', label='Original data')

plt.plot(train\_X, sess.run(W) \* train\_X + sess.run(b), label='Fitted line')

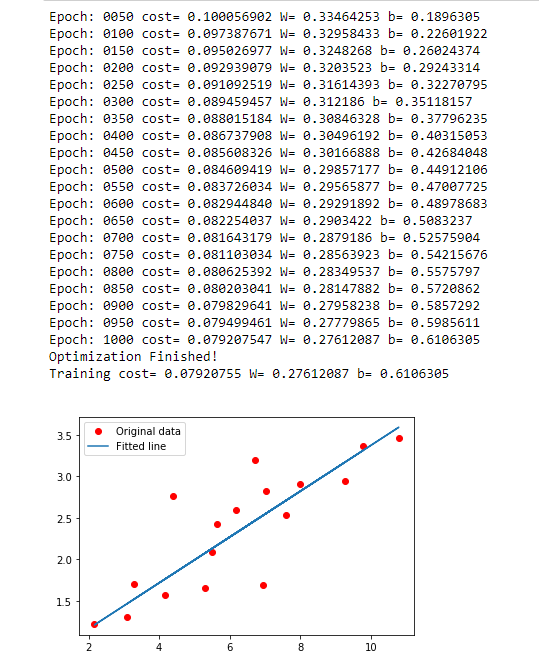
plt.legend()

plt.show()





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



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

**2**. **Tensorflow in Python3 Testing using Jupyter-notebook:**

**Neural Network**

1. Add the below code in the code cell

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: "))

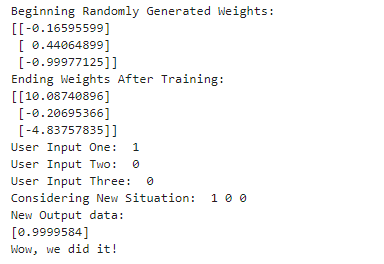
print("Considering New Situation: ", user\_input\_one, user\_input\_two, user\_input\_three)

print("New Output data: ")

print(neural\_network.think(np.array([user\_input\_one, user\_input\_two, user\_input\_three])))

print("Wow, we did it!")

1. Click on **Run**, to execute the code (Code will prompt for input)



**Single Neuron Neural Network**

1. Add the below code in the code cell

# Python program to implement a

# single neuron neural network

# import all necessery libraries

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

# Class to create a neural

# network with single neuron

class NeuralNetwork():

def \_\_init\_\_(self):

# Using seed to make sure it'll

# generate same weights in every run

random.seed(1)

# 3x1 Weight matrix

self.weight\_matrix = 2 \* random.random((3, 1)) - 1

# tanh as activation fucntion

def tanh(self, x):

return tanh(x)

# derivative of tanh function.

# Needed to calculate the gradients.

def tanh\_derivative(self, x):

return 1.0 - tanh(x) \*\* 2

# forward propagation

def forward\_propagation(self, inputs):

return self.tanh(dot(inputs, self.weight\_matrix))

# training the neural network.

def train(self, train\_inputs, train\_outputs,

num\_train\_iterations):

# Number of iterations we want to

# perform for this set of input.

for iteration in range(num\_train\_iterations):

output = self.forward\_propagation(train\_inputs)

# Calculate the error in the output.

error = train\_outputs - output

# multiply the error by input and then

# by gradient of tanh funtion to calculate

# the adjustment needs to be made in weights

adjustment = dot(train\_inputs.T, error \*

self.tanh\_derivative(output))

# Adjust the weight matrix

self.weight\_matrix += adjustment

# Driver Code

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')

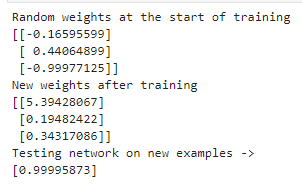
print (neural\_network.weight\_matrix)

# Test the neural network with a new situation.

print ("Testing network on new examples ->")

print (neural\_network.forward\_propagation(array([1, 0, 0])))

1. Click on **Run**, to execute the code



**One Dimensional Vector**

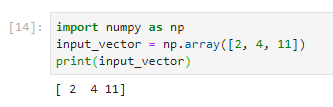
1. In a Python3 notebook, add the below code in the code cell

import numpy as np

input\_vector = np.array([2, 4, 11])

print(input\_vector)

1. To see the output, click on **Run**



**Two Dimensional Array with one column**

1. Add the below code in the code cell

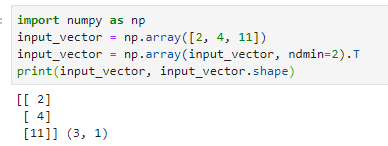
import numpy as np

input\_vector = np.array([2, 4, 11])

input\_vector = np.array(input\_vector, ndmin=2).T

print(input\_vector, input\_vector.shape)

1. Click on **Run**



**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))

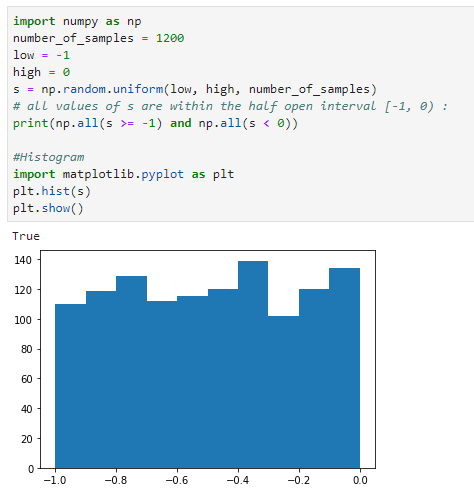
#Histogram

import matplotlib.pyplot as plt

plt.hist(s)

plt.show()

1. Click on **Run**



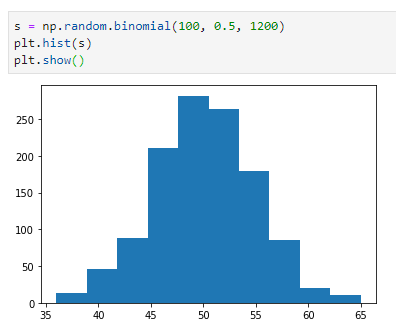
**Note:** Try adding **%matplotlib inline** at the top of the code if you don’t get the graph.

1. To test the binomial function, execute this code

s = np.random.binomial(100, 0.5, 1200)

plt.hist(s)

plt.show()



**Sigmoid Function**

1. Execute the below code

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')

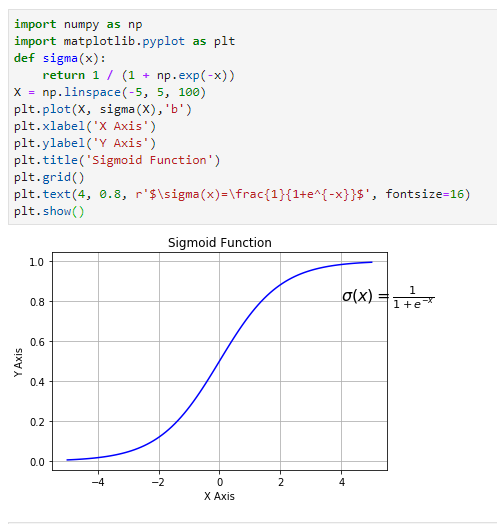
plt.ylabel('Y Axis')

plt.title('Sigmoid Function')

plt.grid()

plt.text(4, 0.8, r'$\sigma(x)=\frac{1}{1+e^{-x}}$', fontsize=16)

plt.show()



**3. CPU Consumption Test in Python3 using JupyterHub:**

1. In a Python3 notebook, add the below code for CPU consumption code

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()

1. Click on **Run**, to execute the code

