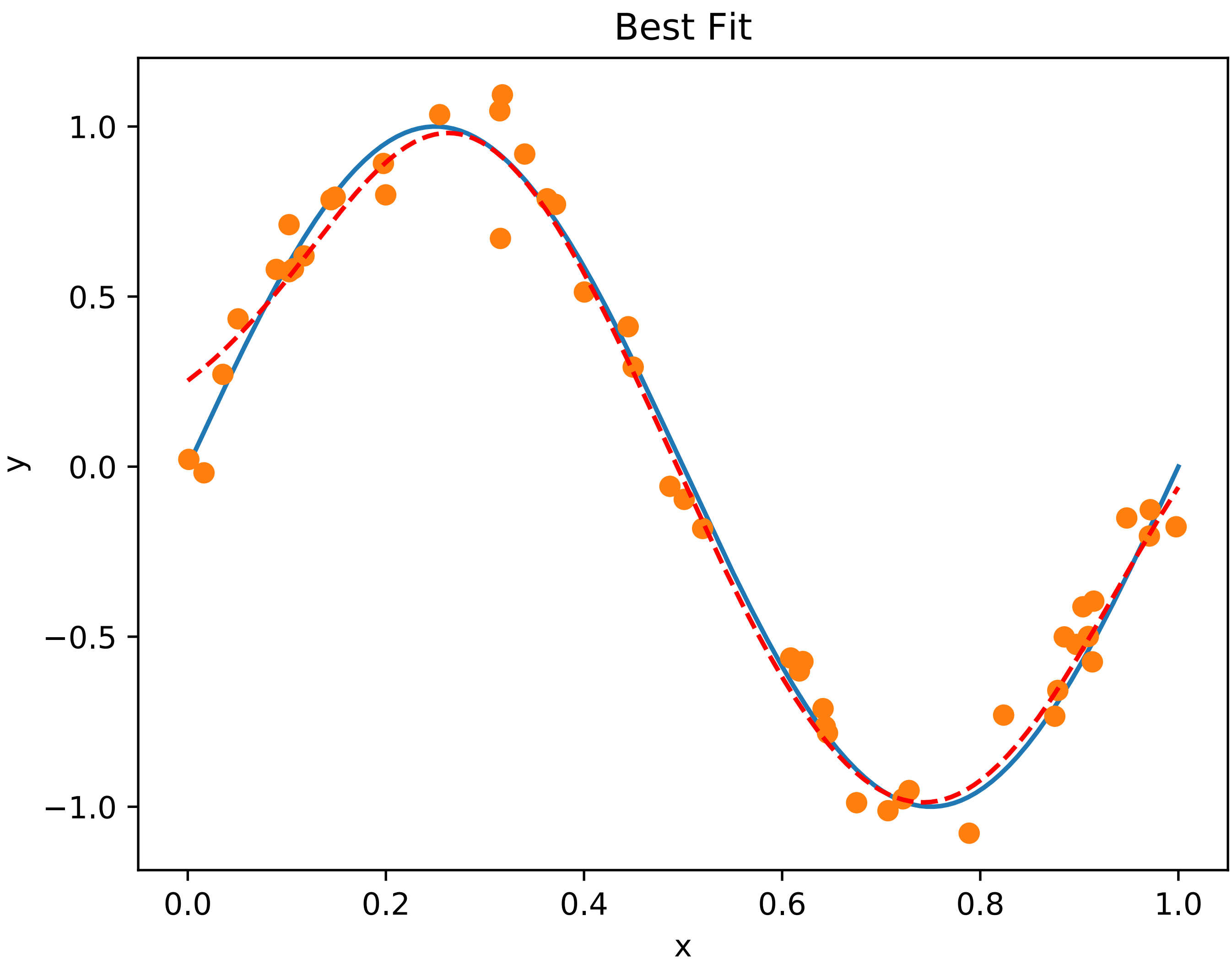
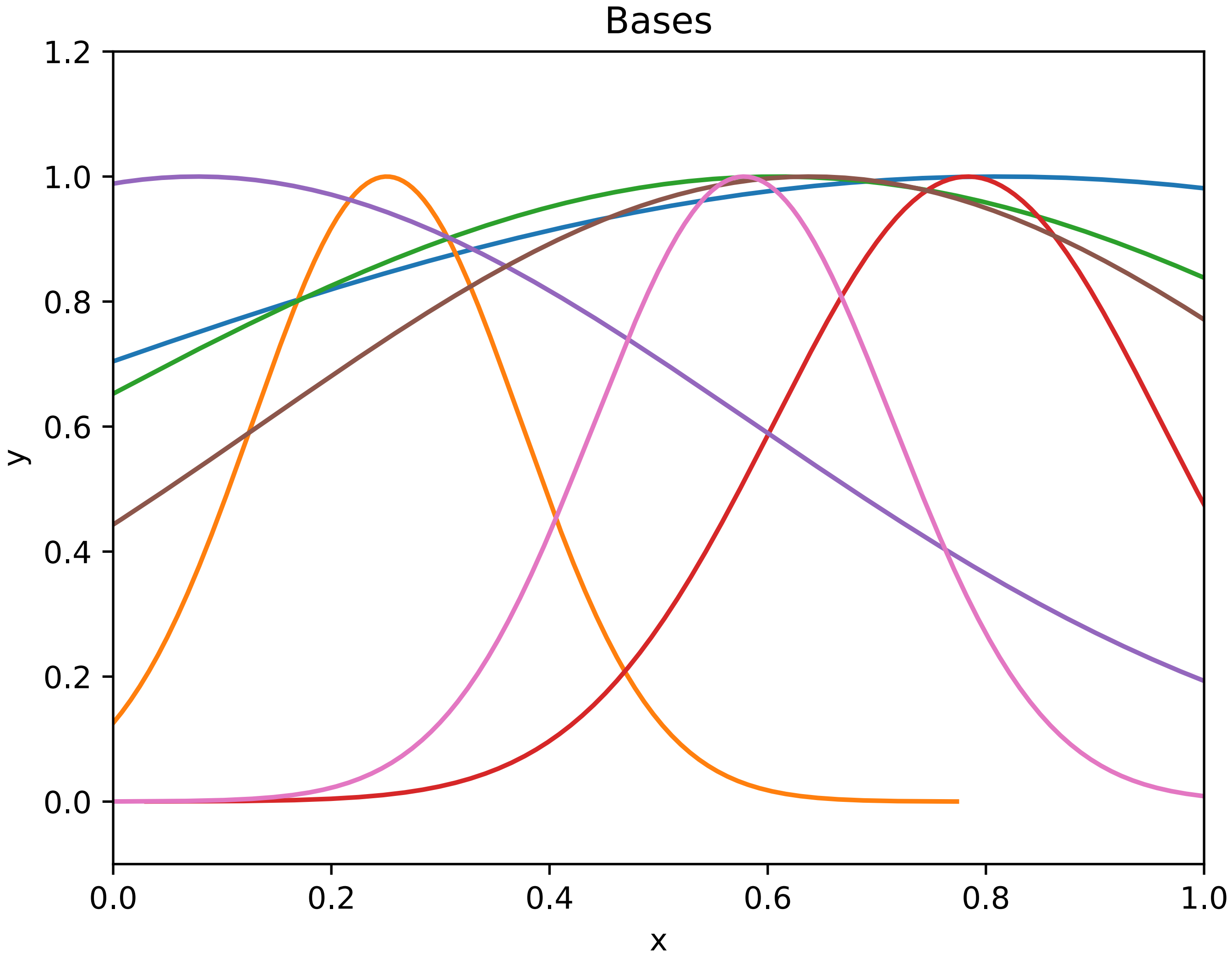
```
#!/bin/python3.6
#Ostap Voynarovskiy
#CGML HW1
#Sept 12 2018

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
from tqdm import tqdm

NUM_PHIS = 7
BATCH_SIZE = 32
NUM_BATCHES = 400

class Data(object):
    def __init__(self):
        num_samp = 50
        sigma = .1
        np.random.seed(31415)
        # We're going to learn these paramters

        #creates your training data
        self.index = np.arange(num_samp)
        self.x = np.random.uniform(size=(num_samp, 1))          # random uniform
 distribution
        self.eps = sigma*np.random.normal(size=(num_samp,1))
        self.y = np.sin(self.x*2*np.pi) + self.eps              # sin wave plus
noise


    def get_batch(self):
        choices = np.random.choice(self.index, size=BATCH_SIZE)
        return self.x[choices].flatten(), self.y[choices].flatten()


def f(x):
    #these are the initializations of the things we will learn
    w =     tf.get_variable('w', [NUM_PHIS, 1], tf.float32,tf.random_normal_ini
tializer())
    b =     tf.get_variable('b', [], tf.float32, tf.zeros_initializer())
    mu =    tf.get_variable('mu', [NUM_PHIS, 1], tf.float32, tf.random_uniform_
initializer())
    sig =   tf.get_variable('sig', [NUM_PHIS, 1], tf.float32,tf.random_normal_ini
tializer())

    phi = tf.exp(-tf.pow((x-mu)/sig, 2))
    return tf.squeeze(tf.matmul(tf.transpose(w), phi) + b)


x = tf.placeholder(tf.float32, [BATCH_SIZE])
y = tf.placeholder(tf.float32, [BATCH_SIZE])
y_hat = f(x)

loss = tf.reduce_mean(tf.pow(y-y_hat, 2)/2) #loss funtion
optim = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(loss) #thi
s does gradient descent
init = tf.global_variables_initializer()

sess = tf.Session()
sess.run(init)
data = Data()

for _ in tqdm(range(0, NUM_BATCHES)):
    x_np, y_np = data.get_batch()
    loss_np, _ = sess.run([loss, optim], feed_dict={x: x_np, y: y_np})

prms = {}
print("Parameter estimates:")
```



```
for var in tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES):
    name = var.name.rstrip(":0")
    value = np.array(sess.run(var)).flatten()
    prms[name] = value
    print(name, np.array_str(value, precision=3))


def rbf(x, mu, sig):
    return np.exp(-np.power((x-mu)/sig,2))

def genManifold(prms, x):
    return np.sum([w * rbf(x, mu, sig) for mu, sig, w in zip(prms['mu'], prms['s
ig'],prms['w'])],0)+prms['b']


#figure 1
fig1= plt.figure(1)

#plot sin wave
NumPoints = 1000
x_v = np.linspace(0, 1, NumPoints)
y_v = np.sin(2*np.pi*x_v)
plt.plot(x_v, y_v)

#plot our noisy training data points
x_p = data.x
y_p = data.y
plt.plot(x_p, y_p, 'o')

#plot our sum of gaussians curve AKA manifold
yplot = genManifold(prms, x_v)
plt.plot(x_v,yplot,'r--')


plt.title("Best Fit")
plt.xlabel('x')
plt.ylabel('y')

fig2 = plt.figure(2)

#loop to plot all phis
for i in range(NUM_PHIS):
    sd= 3
    x_b = np.linspace(prms['mu'][i] - sd*prms['sig'][i], prms['mu'][i] + sd*prms
['sig'][i], NumPoints)# plot 3 standard devs from the mean
    plt.plot(x_b, rbf(x_b, prms['mu'][i], prms['sig'][i]))

plt.title("Bases")
plt.xlabel('x')
plt.ylabel('y')

plt.xlim(-0,1)
plt.ylim(-.1,1.2)

fig1.savefig('fig1.pdf', bbox_inches= 'tight')
fig2.savefig('fig2.pdf', bbox_inches= 'tight')

plt.show()
```


Best Fit
1.0
0.5
0.0
−0.5
−1.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x



Bases
1.2
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x