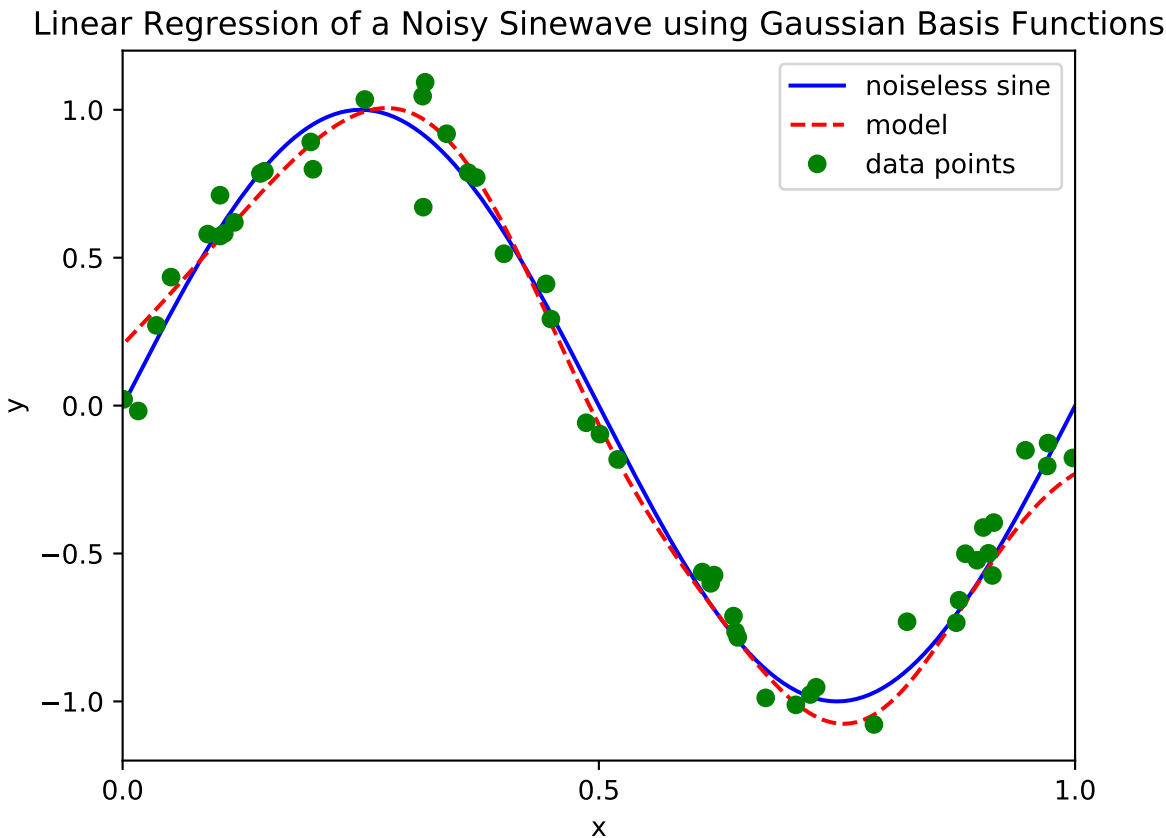
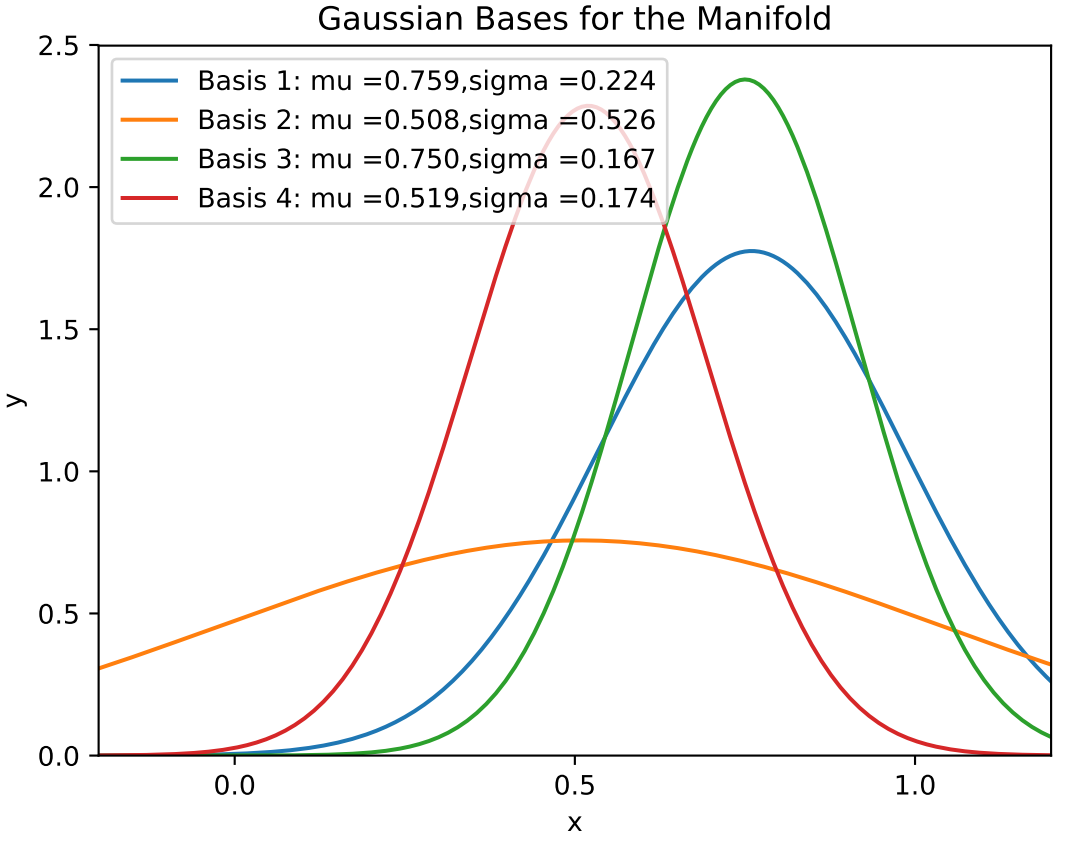
```
# DongKyu Kim
# ECE 471 CGML Assignment 1
# Professor Curro

# library imports
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tqdm import tqdm

# hyper parameters
M = 4
N = 50
iteration = 10000
learning_rate = 0.02
sigma_noise = 0.1

# Data Generation
class Data(object):
    def __init__(self):
        np.random.seed(31415)
        self.x = np.random.uniform(size=(N,1))
        self.y = np.sin(2*np.pi*self.x)+sigma_noise*np.random.normal(size=(N,1))

# Useful Functions
def gaussian_model(x,w,mu,sigma,b):
    phi = tf.exp(-(tf.transpose(x)-mu)**2/sigma**2)
    return tf.transpose(tf.matmul(tf.transpose(w),phi)+b)

def f(x):
    w = tf.get_variable('w',[M,1],tf.float32,tf.random_normal_initializer())
    mu = tf.get_variable('mu',[M,1],tf.float32,tf.random_uniform_initializer())
        # as x is drawn from uniform(0,1) it makes sense that mu is in the
        # range as well
    sigma = tf.get_variable('sigma',[M,1],tf.float32,tf.random_normal_initializer
(),constraint=lambda x:tf.abs(x))
        # since technically sigma should be positive, sigma is forced to be.
    b = tf.get_variable('b',[],tf.float32,tf.zeros_initializer())
    return gaussian_model(x,w,mu,sigma,b)

def norm_dist(x,mu,sigma):
    return 1/(sigma*np.sqrt(2*np.pi))*np.exp(-(x-mu)**2/(2*sigma**2))

# tensor definition
x = tf.placeholder(tf.float32, [N,1])
y = tf.placeholder(tf.float32, [N,1])
y_hat = f(x)
loss = tf.reduce_mean(tf.pow(y_hat - y, 2))
optim = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
init = tf.global_variables_initializer()

# session run
sess = tf.Session()
sess.run(init)
data = Data()
for _ in tqdm(range(0, iteration)):
    sess.run([loss, optim], feed_dict={x: data.x, y: data.y})

# results
print("Parameter estimates:")
storage = []
for var in tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES):
    print(
```

```
        var.name.rstrip(":0"),
        np.array_str(np.array(sess.run(var)).flatten(), precision=3))
    storage.append(sess.run(var))
[w_model,mu_model,sigma_model,b_model] = storage

#plotting
x_plot = np.linspace(-0.2,1.2,300,dtype=np.float32)
y_truth = np.sin(2*np.pi*x_plot)
y_model = gaussian_model(x_plot,w_model,mu_model,sigma_model,b_model)

    #plot 1
plt.figure(1)
plt.subplot()
plt.plot(x_plot,y_truth,'b',label='noiseless sine')
plt.plot(x_plot,sess.run(y_model),'--r',label='model')
plt.plot(data.x,data.y,'go',label='data points')
plt.xlabel('x')
plt.xticks(np.arange(0,1.2,step=0.5))
plt.ylabel('y')
plt.axis([0,1,-1.2,1.2])
plt.title('Linear Regression of a Noisy Sinewave using Gaussian Basis Function')
plt.legend()
plt.show()

    #plot 2
plt.figure(2)
plt.subplot()
for i in range(1,M+1):
    plt.plot(x_plot,norm_dist(x_plot,mu_model[i-1],sigma_model[i-1]),
        label='Basis '+str(i)+': mu ='+str(mu_model[i-1])[1:6]+',sigma ='
        +str(sigma_model[i-1])[1:6])
plt.xlabel('x')
plt.xticks(np.arange(0,1.2,step=0.5))
plt.ylabel('y')
plt.axis([-0.2,1.2,0,plt.axis()[3]])
plt.title('Gaussian Bases for the Manifold')
plt.legend()
plt.show()
```


Linear Regression of a Noisy Sinewave using Gaussian Basis Functions
noiseless sine
model
data points
1.0
0.5
0.0
−0.5
−1.0
y
0.0
0.5
1.0
x


Gaussian Bases for the Manifold
Basis 1: mu =0.759,sigma =0.224
Basis 2: mu =0.508,sigma =0.526
Basis 3: mu =0.750,sigma =0.167
Basis 4: mu =0.519,sigma =0.174
2.5
2.0
1.5
1.0
0.5
0.0
y
0.0
0.5
1.0
x