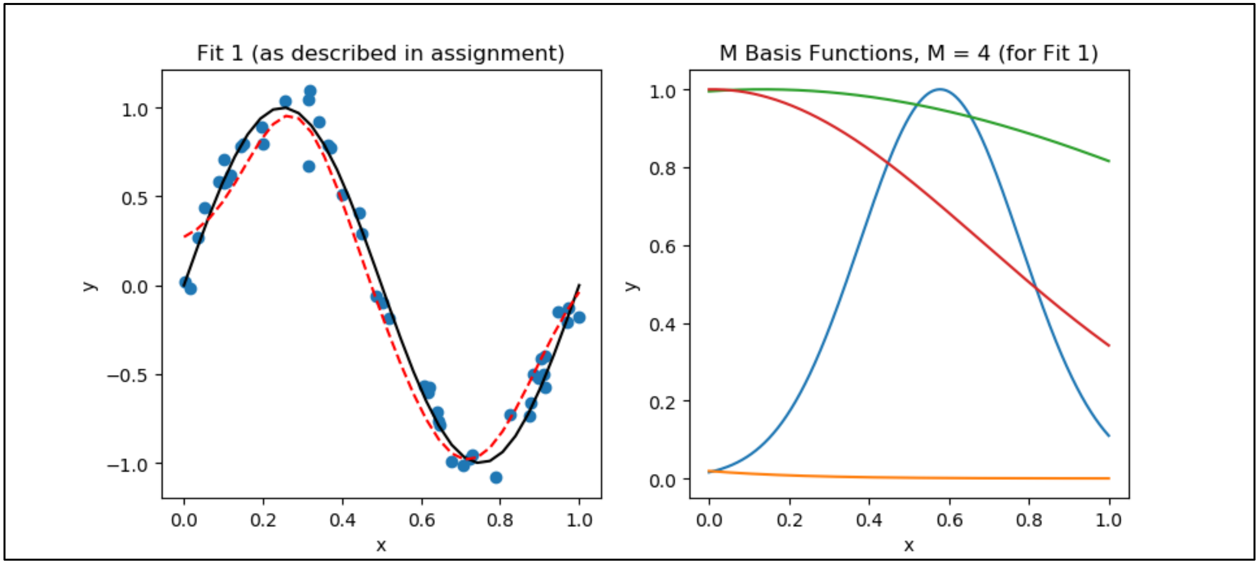
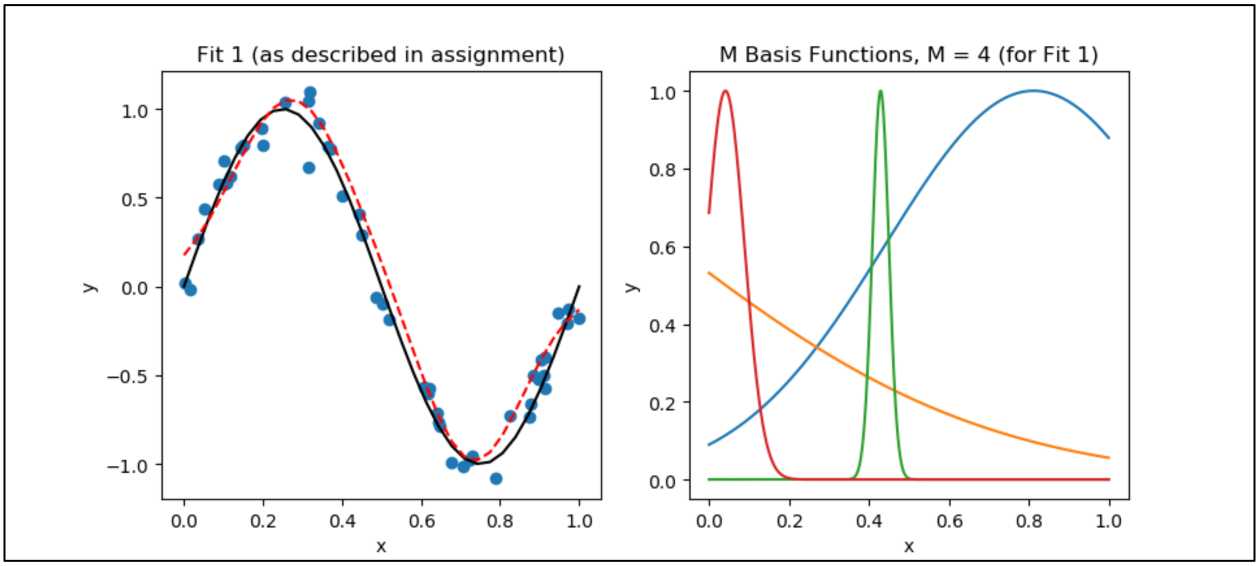
Nithilam Subbaian
Assignment 1

```
#!/bin/python3.6
import numpy as np
import tensorflow as tf
from tqdm import trange
import matplotlib.pyplot as plt

NUM_FEATURES = 4
NUM_SAMP = 50
BATCH_SIZE = 32
NUM_BATCHES = 300
LEARNING_RATE = 0.1

class Data(object):
    def __init__(self, num_features=NUM_FEATURES, num_samp=NUM_SAMP):
        """
        Draw random weights and bias. Project vectors in R^NUM_FEATURES
        onto R with said weights and bias.
        """
        num_samp = NUM_SAMP # this is set to 50
        sigma = 0.1

        np.random.seed(31415) #seed the generator, RandomState uses it. can be called again.
        self.index = np.arange(num_samp) # Return evenly spaced values from 0 to num_samp

        self.x = np.random.uniform(size=(num_samp, 1))
        self.eps = np.random.normal(scale = sigma, size = (num_samp, 1))

        self.y = np.sin(2 * np.pi * self.x) + self.eps # yi written out to obtain data

    def get_batch(self, batch_size=BATCH_SIZE):
        """
        Select random subset of examples for training batch
        """
        choices = np.random.choice(self.index, size=batch_size)

        return self.x[choices].flatten(), self.y[choices].flatten()


class Model(tf.Module):
    def __init__(self, num_features = NUM_FEATURES):

        self.w = tf.Variable(tf.random.normal(shape=[num_features, 1]))
        self.mu = tf.Variable(tf.random.normal(shape=[num_features, 1]))
        self.sigma = tf.Variable(tf.random.normal(shape=[num_features, 1]))
        self.b = tf.Variable(tf.zeros(shape=[1, 1]))

        # plots the M basis functions
        plt.subplot(122) #rows, column, index
        xbasis = np.linspace(0,1,1000)
        mu_matrix = self.mu
        sigma_matrix = self.sigma
        for i in range(NUM_FEATURES):
            y = np.exp(-((xbasis - mu_matrix[i])/sigma_matrix[i])**2)
            plt.plot(xbasis,y)

        plt.xlabel('x')
        plt.ylabel('y')
        plt.title('M Basis Functions, M = 4 (for Fit 1)')

    def __call__(self, x):

        phi = tf.exp(- tf.pow((x - self.mu)/self.sigma, 2))
        # Use of matrix multiplication form
        return tf.squeeze(tf.matmul( tf.transpose(self.w), phi) + self.b)


if __name__ == "__main__":
    data = Data()
    model = Model()
    optimizer = tf.optimizers.SGD(learning_rate=LEARNING_RATE) #Stochastic gradient descent and
momentum optimizer.

    bar = trange(NUM_BATCHES) # creates bar visual to show progress completion status
    for i in bar:
        with tf.GradientTape() as tape: # GradientTape= Record operations for automatic differentiation.
            x, y = data.get_batch()
            y_hat = model(x)
            loss = tf.reduce_mean((y_hat - y) ** 2)

        grads = tape.gradient(loss, model.variables)
        optimizer.apply_gradients(zip(grads, model.variables))

        bar.set_description(f"Loss @ {i} => {loss.numpy():0.6f}")
        bar.refresh()

    plt.figure(1, figsize=[30,30])
    plt.subplot(121) #rows, column, index
    xfit = np.linspace(0,1,BATCH_SIZE)
    plt.plot(xfit, np.sin(2 * np.pi * xfit), 'k')
    plt.plot(xfit, model(xfit), 'r--')
    plt.scatter(data.x, data.y)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Fit 1 (as described in assignment)')

    plt.show()
```

```
NUM_BATCHES = 300
Loss @ 299 => 0.021591:
100%|████████████████████████████████████████████████████████████████████
█| 300/300 [00:01<00:00, 151.33it/s]
```



```
NUM_BATCHES = 3000
Loss @ 2999 => 0.010320:
100%|███████████████████████████████████████████████████████████████████|
3000/3000 [00:17<00:00, 174.85it/s]
```


Fit 1 (as described in assignment)
y
x
M Basis Functions, M = 4 (for Fit 1)
y
x