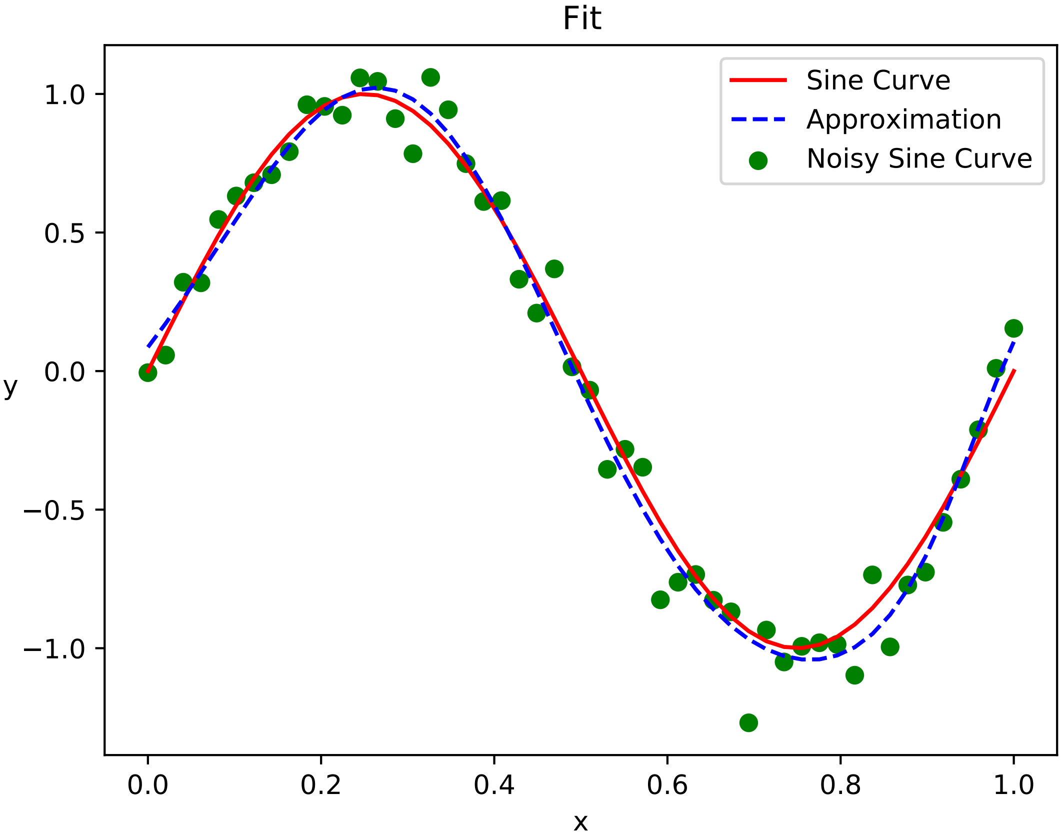
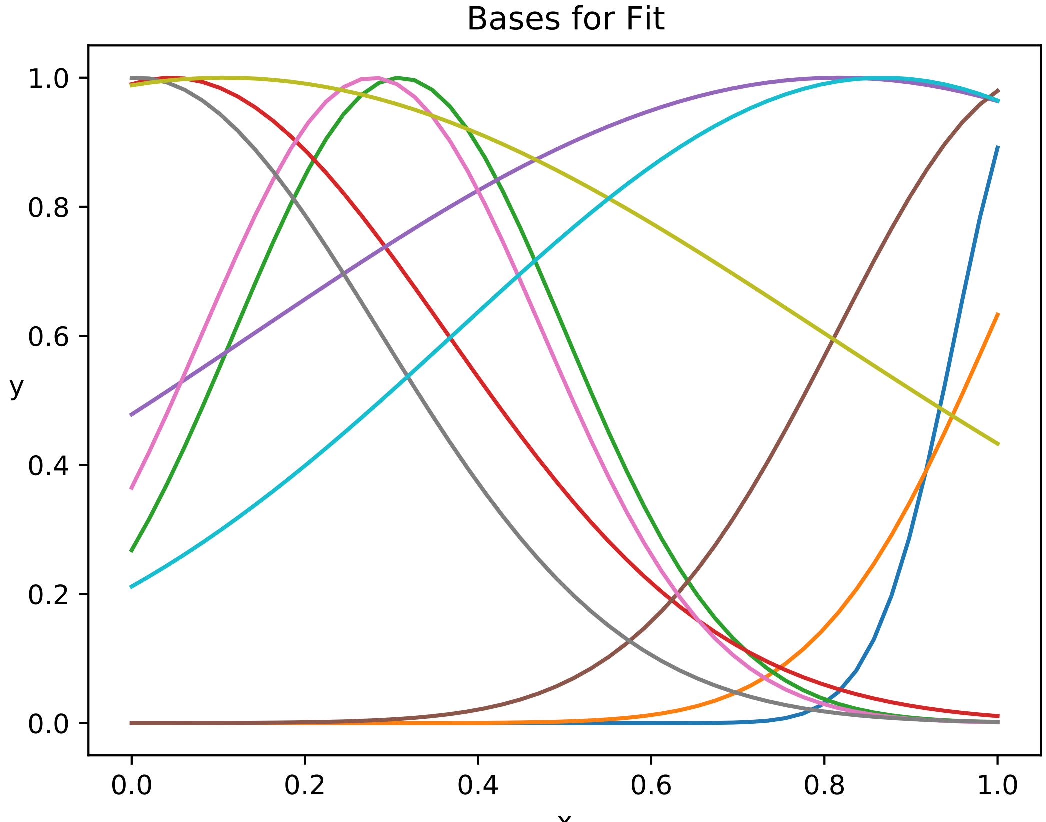
```python
'''
Alexander Koldy
ECE 472 - Deep Learning
Assignment 1
'''

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

'''
Code would not run without this:
'''
tf.enable_eager_execution()

# Known parameters
N = 50 # number of discrete steps in function
sigma_noise = 0.1
epsilon = np.random.normal(0, sigma_noise, N) # noise

# Loop parameters
'''
Editting these parameters may yield better results
'''
M = 10 # number of gaussian basis functions
epochs = 1000 # loop iterations
learning_rate = 0.02 # step size

# Noisy sin wave
x = np.linspace(0, 1, N)
y = np.sin(2*np.pi*x) + epsilon

# Parameters to estimate
w = []
mu = []
sigma = []
b = []
for _ in range(M):
    '''
    Each parameter was established with a random value
    with no range. It may sometimes take more epochs to
    minimize the cost function due to this,
    '''
    w.append(tf.Variable(np.random.rand()))
    mu.append(tf.Variable(np.random.rand()))
    sigma.append(tf.Variable(np.random.rand()))
    b.append(tf.Variable(np.random.rand()))

# Gaussian
def phi(x, mu, sigma):
    return tf.exp(-(x - mu)**2 / sigma**2)

# Approximation of y
def y_hat(x):
    y_hat_i = 0
    for j in range(M):
        y_hat_i = y_hat_i + w[j]*phi(x, mu[j], sigma[j]) + b[j]
    return y_hat_i

# Cost function
```

```
def J(y, y_hat):
    return (1/2)*tf.square(y - y_hat)

# Gradient Descent
'''
The following gradient descent code is editted from (1) to fit
the scope of the assignment.
'''
for epoch in range(epochs):
    with tf.GradientTape() as tape:
        loss = tf.reduce_mean(J(y, y_hat(x)))

    gradients = tape.gradient(loss, [w, mu, sigma, b])

    for j in range(M):
        w[j].assign_sub(gradients[0][j]*learning_rate)
        mu[j].assign_sub(gradients[1][j]*learning_rate)
        sigma[j].assign_sub(gradients[2][j]*learning_rate)
        b[j].assign_sub(gradients[3][j]*learning_rate)

    '''
    Once again, this line is taken directly from (1). It helps visualize the loss as more iterations of the loop go through.
    '''
    print(f"Epoch count {epoch}: Loss value: {loss.numpy()}")

# Plot regression approximation
plt.figure(1)
plt.title("Fit")
plt.xlabel("x")
plt.ylabel("y", rotation="horizontal")
plt.scatter(x, y, label="Noisy Sine Curve", color="green")
plt.plot(x, np.sin(2*np.pi*x), label="Sine Curve", color="red")
plt.plot(x, y_hat(x), '--', label="Approximation", color="blue")
plt.legend()
plt.show()

# Plot basis functions
plt.figure(2)
plt.title("Bases for Fit")
plt.xlabel("x")
plt.ylabel("y", rotation="horizontal")
for j in range(M):
    plt.plot(x, phi(x, mu[j], sigma[j]))
plt.show()


'''
References:
(1) https://www.machinelearningplus.com/deep-learning/linear-regression-tensorflow/
(2) Cooper Union ECE-472: Deep Learning - Learning Materials
'''
```


Fit
Sine Curve
Approximation
Noisy Sine Curve
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 for Fit
y
x
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0