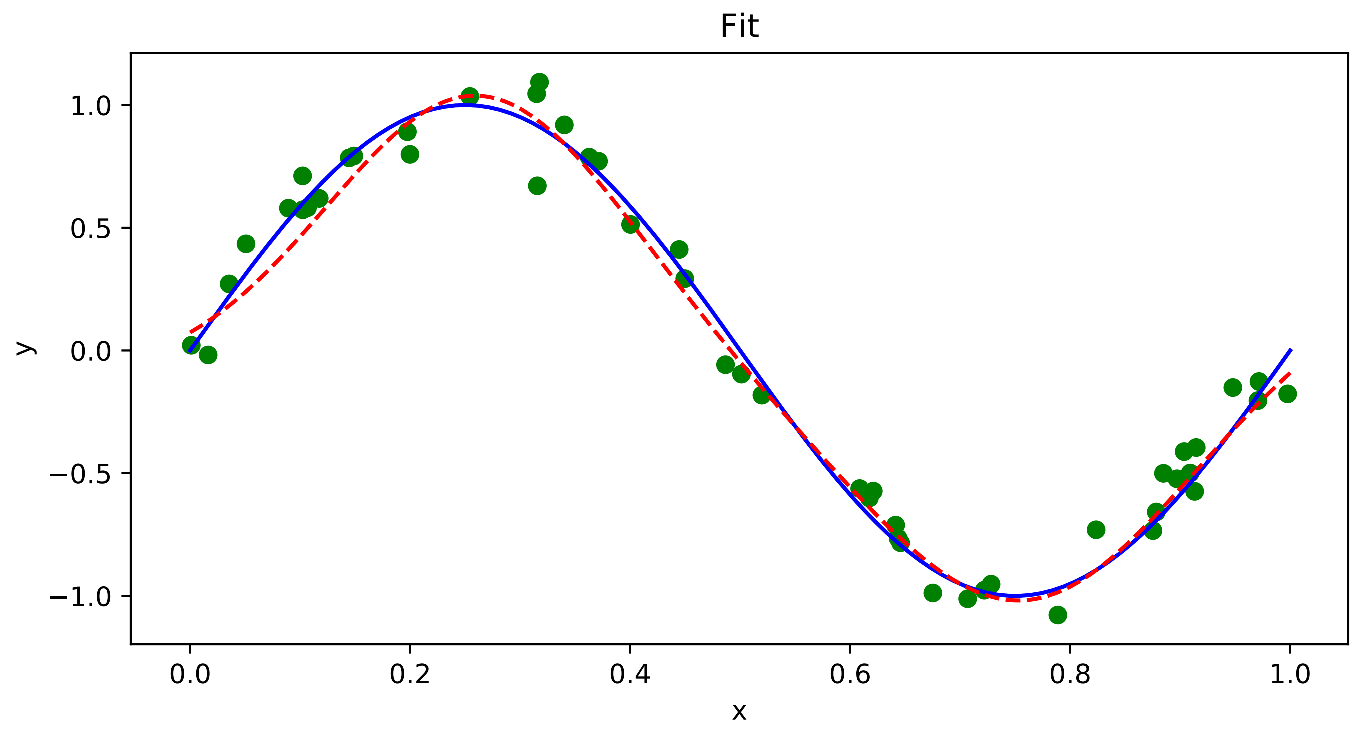
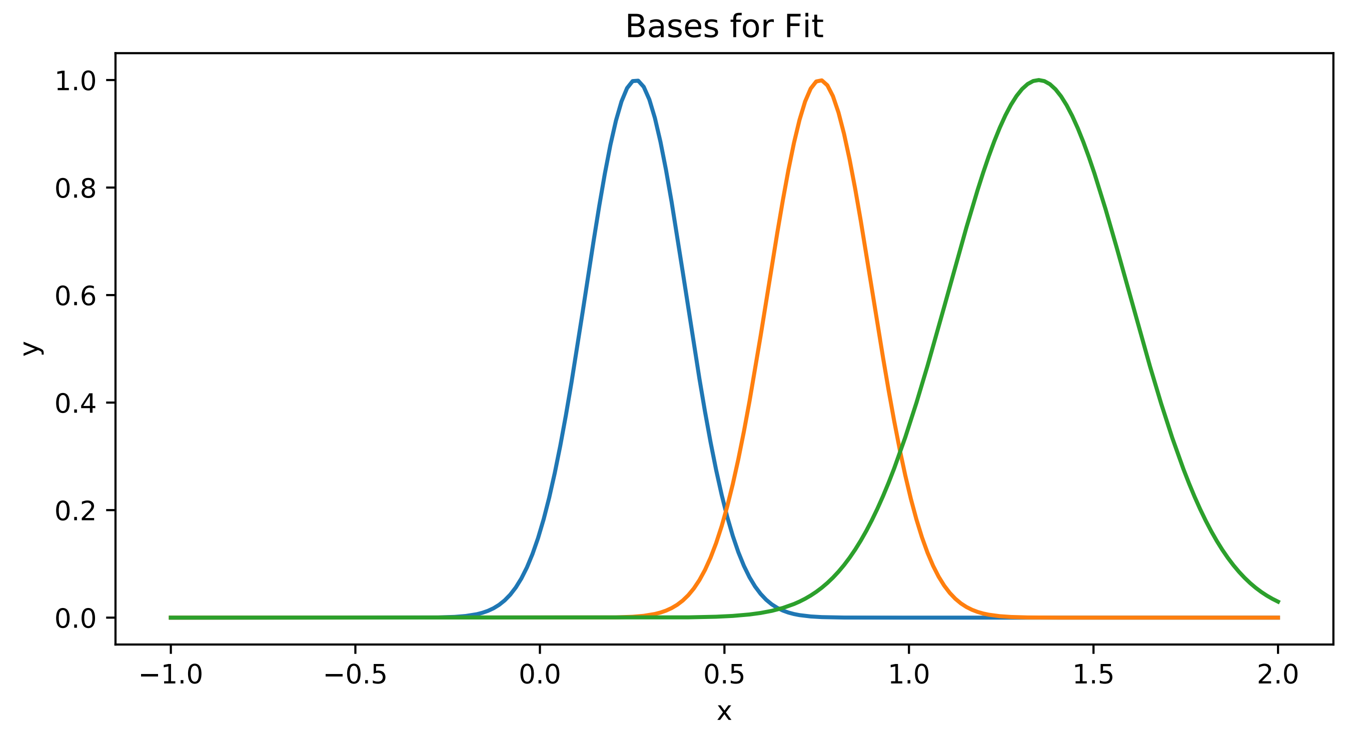
```
'''
ECE 471, Selected Topics in Machine Learning - Assignment 1

Submit by Sept. 12, 10pm

tldr: Perform linear regression of a noisy sinewave using a set of gaussian
basis functions with learned location and scale parameters. Model parameters
are learned with stochastic gradient descent. Use of automatic differentiation
is required. Hint: note your limits!
'''

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

NUM_DATAPOINTS = 50
NUM_FEATURES = 3
BATCH_SIZE = 10  # Batch size must evenly divide NUM_DATAPOINTS... sorry
NUM_EPOCHS = 300

sigma = 0.1
np.random.seed(31415)
x_np = np.random.uniform(size=NUM_DATAPOINTS)
y_np = np.sin(2*np.pi*x_np) \
       + np.random.normal(scale=sigma, size=NUM_DATAPOINTS)

# Split into batches, reshape to column vectors
x_batches = np.split(x_np, np.ceil(NUM_DATAPOINTS/BATCH_SIZE))
y_batches = np.split(y_np, np.ceil(NUM_DATAPOINTS/BATCH_SIZE))
x_batches = [x.reshape(x.shape[0], -1) for x in x_batches]
y_batches = [y.reshape(y.shape[0], -1) for y in y_batches]


def f(x):
    mu_init = (np.linspace(0, 1, NUM_FEATURES, dtype=np.float32)
               .reshape(NUM_FEATURES, -1))
    mu = tf.get_variable('mu', dtype=tf.float32, initializer=mu_init)
    sigma = tf.get_variable('sigma', [NUM_FEATURES, 1], tf.float32,
                            tf.constant_initializer(0.5))
    w = tf.get_variable('w', [NUM_FEATURES, 1], tf.float32,
                        tf.random_normal_initializer())
    b = tf.get_variable('b', [], tf.float32, tf.zeros_initializer())

    # Tensorflow broadcasting!
    phi_x = tf.exp(-tf.div(tf.square(x - tf.transpose(mu)),
                           tf.square(tf.transpose(sigma))))

    return tf.matmul(phi_x, w) + b


x = tf.placeholder(tf.float32, [BATCH_SIZE, 1])
y = tf.placeholder(tf.float32, [BATCH_SIZE, 1])
y_hat = f(x)

loss = tf.reduce_mean(tf.pow(y_hat - y, 2))
optim = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(loss)
init = tf.global_variables_initializer()

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

for _ in tqdm(range(NUM_EPOCHS)):
    for x_batch, y_batch in zip(x_batches, y_batches):
        _, loss_np = sess.run([optim, loss],
                              feed_dict={x: x_batch, y: y_batch})

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

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


# Plotting
fig, axarr = plt.subplots(ncols=2, nrows=1, figsize=[18, 4])


def phi(x, i):
    return np.exp(-(x - params['mu'][i])**2 / params['sigma'][i]**2)


# First plot
x_fit = np.linspace(0, 1, 200)
y_noiseless = np.sin(2*np.pi*x_fit)
y_learned = np.vstack([params['w'][i] * phi(x_fit, i)
                       for i in range(NUM_FEATURES)]).sum(axis=0) + params['b']

axarr[0].scatter(x_np, y_np, color='g')
axarr[0].plot(x_fit, y_noiseless, color='b')
axarr[0].plot(x_fit, y_learned, 'r--')
axarr[0].set_title('Fit')
axarr[0].set_xlabel('x')
axarr[0].set_ylabel('y')

# Second plot
x_bases = np.linspace(-1, 2, 200)
for i in range(NUM_FEATURES):
    axarr[1].plot(x_bases, phi(x_bases, i))
axarr[1].set_title('Bases for Fit')
axarr[1].set_xlabel('x')
axarr[1].set_ylabel('y')

plt.show()
```


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


Bases for Fit
y
x
0.0
0.2
0.4
0.6
0.8
1.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0