hw1\hw1.py

```python
import numpy as np
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
plt.style.use("seaborn")

def preprocess(data_path):
    """Returns - input_data: 400x5 np array of inputs (x)
                 output data: 200x5 np array of outputs (y)"""
    input_labels = ["x1", "x2", "x3", "x4", "x5"]
    output_labels = ["y1", "y2"]
    data = pd.read_csv(data_path, names=input_labels+output_labels)

    # create 400x5 numpy array of inputs
    input_data = np.zeros((len(data["x1"]), len(input_labels)))
    for i, label in enumerate(input_labels):
        # input_data[label] = data[label]
        input_data[:,i] = data[label]

    # create 400x2 numpy array for outputs
    output_data = np.zeros((len(data["y1"]), len(output_labels)))
    for i, label in enumerate(output_labels):
        output_data[:,i] = data[label]

    return input_data, output_data

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def d_sigmoid(x):
    return sigmoid(x)*(1-sigmoid(-x))

def forward_pass(x, w1, w2, b1, b2):
    """Inputs - x: 5x1 np array"""
    x.shape = (-1, 1)
    z1 = np.dot(x.T, w1.T) + b1.T
    a1 = sigmoid(z1)

    z2 = np.dot(a1, w2.T) + b2.T
    a2 = sigmoid(z2)
    return z1, z2, a1, a2

def error(y, y_hat):
    err = np.mean(np.square(np.subtract(y_hat, y)))
    return err

def d_error(y, y_hat):
    dE_da1 = 2*np.subtract(y_hat[0], y[0])
    dE_da2 = 2*np.subtract(y_hat[1], y[1])
    ret = np.empty((2,1))
    ret[0] = dE_da1
    ret[1] = dE_da2
```

```
    return ret

def calc_delta(dE_da, a):
    da_dz = np.multiply(a, (1-a))
    delta = np.multiply(dE_da, da_dz)
    return delta

def update_weights(w, weight_gradient, learning_rate):
    new_weights = np.subtract(w, learning_rate*weight_gradient)
    return new_weights

def update_bias(b, bias_gradient, learning_rate):
    new_bias = np.subtract(b, learning_rate*bias_gradient)
    return new_bias

def save_weights(w1, w2, b1, b2):
    np.save("w1.npy", w1)
    np.save("w2.npy", w2)
    np.save("b1.npy", b1)
    np.save("b2.npy", b2)

def train(input_data, output_data, num_hidden, epochs, learning_rate):
    # number of inputs and outputs
    n = len(output_data[0])
    m = len(input_data[0])
    # initialize random weights and biases
    w1 = np.random.rand(num_hidden, m)
    w2 = np.random.rand(n, num_hidden)
    b1 = np.random.rand(num_hidden, 1)
    b2 = np.random.rand(n, 1)

    # initialize vars
    err_list = []
    count_correct_list = []
    learning_rate = learning_rate
    for epoch in tqdm(range(epochs), desc="Iterating..."):

        for i, x in enumerate(input_data):
            # x.shape = (-1, 1)

            y = output_data[i]

            # foraward pass and error
            z1, z2, a1, a2 = forward_pass(x, w1, w2, b1, b2)

            err = error(y, a2)
            err_list.append(err)

            d_err = 2*(np.subtract(a2, y))

            # step 0: calculate the 'delta' term for layer 2 (da2/dz2*dE/da2)
            delta = d_err * (sigmoid(z2)*(1-sigmoid(z2)))

            # step 1: dE/dW and dE/dB for layer 2
                # how much the error depends on w2 and b2
```

```
            weight_gradient_l2 = delta.T*a1
            bias_gradient_l2 = delta

            # step 2: dE/da_1
                # how much the error depends on a1
            activation_gradient_l1 = w2.T@delta.T

            # step 3: dE/dW and dE/dB for layer 1
                # how much the error depends on w1 and b1
            da1_dz1 = (sigmoid(z1)*(1-sigmoid(z1)))
            weight_gradient_l1 = x.T*(np.multiply(da1_dz1.T, activation_gradient_l1))
            bias_gradient_l1 = np.multiply(da1_dz1.T, activation_gradient_l1)

            w1 = update_weights(w1, weight_gradient_l1, learning_rate)
            b1 = update_bias(b1, bias_gradient_l1, learning_rate)
            w2 = update_weights(w2, weight_gradient_l2, learning_rate)
            b2 = update_bias(b2, bias_gradient_l2.T, learning_rate)

        count_correct = 0
        for j, x in enumerate(input_data):
            # foraward pass
            z1, z2, a1, a2 = forward_pass(x, w1, w2, b1, b2)
            y = output_data[j]

            if np.array_equal(np.round(a2), y.reshape((1,2))):
                count_correct += 1

        # print(f"Test accuracy: {count_correct/400}")

        count_correct_list.append(count_correct)
        lr_decay = 1
        learning_rate *= (1. / (1. + lr_decay * epoch))

    save_weights(w1, w2, b1, b2)
    return err_list, count_correct_list

####### flight code

# load training and testing data
pair = 1
train_inputs, train_outputs = preprocess("train"+str(pair)+".csv")
test_inputs, test_outputs = preprocess("test"+str(pair)+".csv")

do_training = False
# train and test the data with different parameters
if do_training == True:
    epochs = 5
    num_hidden = 10
    initial_learning_rate = 0.1

    train_correct = []
    test_correct = []

    # Run for 10 trials
    trials = 10
```

```
    for trial in range(trials):
        # training
        train_err_list, train_correct_list = train(train_inputs, train_outputs, num_hidden=num_hidden, epochs=epochs, learning_rate=initial_learning_rate)
        train_correct.append(train_correct_list)

        # testing
        w1 = np.load("w1.npy")
        w2 = np.load("w2.npy")
        b1 = np.load("b1.npy")
        b2 = np.load("b2.npy")

        num_correct = 0
        for i, x in enumerate(test_inputs):
            _, _, _, a2 = forward_pass(x, w1, w2, b1, b2)
            y = test_outputs[i]

            if np.array_equal(np.round(a2), y.reshape((1,2))):
                num_correct += 1

        test_correct.append(num_correct)

    train_correct_avg = np.mean(train_correct, axis=0)
    test_correct_avg = np.mean(test_correct)
    test_correct_stdv = np.std(test_correct)
    train_iterations = np.linspace(1, epochs, len(train_correct_avg))

    print(f"Average test correct classification percentage: {test_correct_avg/400}")
    print(f"Standard deviation: {test_correct_stdv/400}")

    fig, ax = plt.subplots(1,1)
    y = np.array(train_correct_avg)/400
    ax.plot(train_iterations, y, label="Hidden units: "+str(num_hidden))
    ax.set_title(f"Average Training {pair} Results with {num_hidden} Hidden Units, {initial_learning_rate} Learning Rate")
    ax.set_xlabel("Epoch")
    ax.set_ylabel("Correct Classification Percentage")
    plt.show()

```