```
import tensorflow as tf
import matplotlib.pyplot as plt
from help_functions import readDataset, next_batch, plot_dataset, write_data, read_data, plot_da

# Hyperparameters
learning_rate = 0.1
num_epochs = 200
batch_size = 100
beta = 0.5

display_step = 10

# Logging
num_layers = input('How many hidden layers?\n')
mean_type = input('Single or double mean? [1/2]\n')
filename = 'hidden_' + num_layers + '_' + mean_type + 'mean.txt'
loss = [0]*(int(num_epochs/display_step)+1)
acc = [0]*(int(num_epochs/display_step)+1)
epoch_list = [0]*(int(num_epochs/display_step)+1)

# Read data files
if mean_type == '1':
    train_dataset= readDataset('Dataset/train_dataset_1mean.txt')
    test_dataset = readDataset('Dataset/test_dataset_1mean.txt')
    validation_dataset= readDataset('Dataset/validation_dataset_1mean.txt')
elif mean_type == '2':
    train_dataset= readDataset('Dataset/train_dataset_2mean.txt')
    test_dataset = readDataset('Dataset/test_dataset_2mean.txt')
    validation_dataset= readDataset('Dataset/validation_dataset_2mean.txt')

# Normalize data
#train_dataset = normalize_dataset(train_dataset)

# Network Parameters
num_hidden_1 = 10 # 1st layer number of neurons
num_hidden_2= 10 # 2nd layer number of neurons
num_input = 2
num_classes = 2

# tf Graph input
X = tf.placeholder(tf.float32, [None, num_input])
Y = tf.placeholder(tf.float32,[None, num_classes])

## Store layers weight & biases
if num_layers == '1':
    h1 = tf.Variable(tf.random_normal([num_input, num_hidden_1]))
    w_out = tf.Variable(tf.random_normal([num_hidden_1, num_classes]))

    b1 = tf.Variable(tf.random_normal([num_hidden_1]))
    b_out = tf.Variable(tf.random_normal([num_classes]))
elif num_layers == '2':
    h1 = tf.Variable(tf.random_normal([num_input, num_hidden_1]))
    h2 = tf.Variable(tf.random_normal([num_hidden_1, num_hidden_2]))
    w_out = tf.Variable(tf.random_normal([num_hidden_2, num_classes]))

    b1 = tf.Variable(tf.random_normal([num_hidden_1]))
    b2 = tf.Variable(tf.random_normal([num_hidden_2]))
    b_out = tf.Variable(tf.random_normal([num_classes]))

# Create model
def neural_net_1(x):
    layer_1 = tf.add(tf.matmul(x,h1), b1)
    layer_1 = tf.nn.relu(layer_1)

    out_layer = tf.matmul(layer_1, w_out) + b_out
    out_layer = tf.nn.sigmoid(out_layer)
    return out_layer

def neural_net_2(x):
    layer_1 = tf.add(tf.matmul(x,h1), b1)
    layer_1 = tf.nn.relu(layer_1)

    layer_2 = tf.add(tf.matmul(layer_1,h2), b2)
    layer_2 = tf.nn.relu(layer_2)

    out_layer = tf.matmul(layer_2, w_out) + b_out
    out_layer = tf.nn.sigmoid(out_layer)
    return out_layer

# Construct model
if num_layers == '1':
    logits = neural_net_1(X)
elif num_layers == '2':
    logits = neural_net_2(X)


# Define loss function with regularizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=Y))
regularizer = tf.nn.l2_loss(w_out)
loss_op = tf.reduce_mean(loss_op+beta*regularizer)

# Define optimizer
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)

# Evaluate model
correct_ped = tf.equal(tf.argmax(logits,1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correct_ped, tf.float32))

# Initialize the variables
init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:
    # Run the initializer
    sess.run(init)

    for epoch in range(1, num_epochs):
        batch_x, batch_y = next_batch(train_dataset,batch_size)
        sess.run(train_op, feed_dict={X:batch_x, Y:batch_y})

        if epoch % display_step == 0 or epoch == 1:
            logg_it = 1+int(epoch/display_step)
            # Calculate batch loss and accuracy
            loss[logg_it], acc[logg_it] = sess.run([loss_op, accuracy], feed_dict={X: batch_x, Y
            epoch_list[logg_it] = epoch
            #print("Epoch " + str(epoch) + ", Minibatch Loss= " + "{:.4f}".format(loss[logg_it]
    print("Optimization Finished!")

    # Calculate accuracy
    batch_x, batch_y = next_batch(test_dataset,test_dataset.N)
    print("Testing Accuracy:", \
        sess.run(accuracy, feed_dict={X: batch_x,
                                      Y: batch_y}))
    # Validation accuracy
    batch_x, batch_y = next_batch(validation_dataset,validation_dataset.N)
    print("Validation Accuracy:", \
        sess.run(accuracy, feed_dict={X: batch_x,
                                      Y: batch_y}))

    write_data(loss, acc, epoch_list, filename)
    plot_dataset(train_dataset)
    plot_data()
```