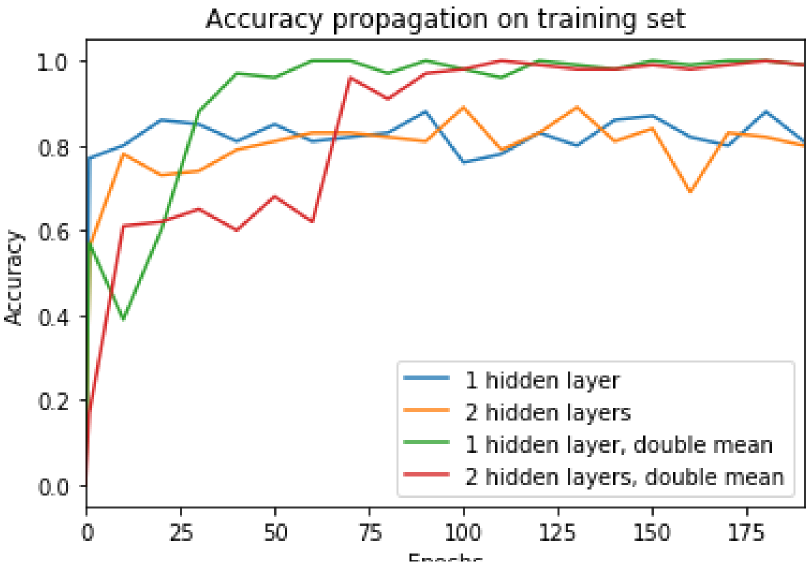
Homework 2 ECE283

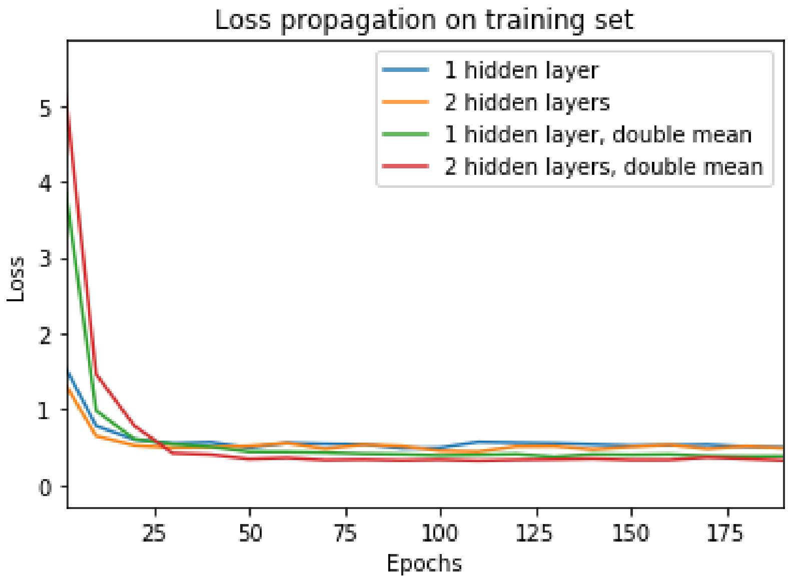
Problem 1
See attached code at the end of report

Problem 2
I started out using N=200 samples distributed on training, testing and validation set split of 70:20:10. With this number of samples, I went through numerous trainings with different number of neurons and different regularization constant. This went on until I had an adequate performance that slightly overfitted the data, with validation accuracy about 0,89. I ended up with the following hyperparameters:

- learning_rate = 0.1
- num_epochs = 200
- batch_size = 100
- beta = 0.1 (Regularization constant)

After this, I increased the number of samples to N = 1000. With this higher number of samples, I got the following results.

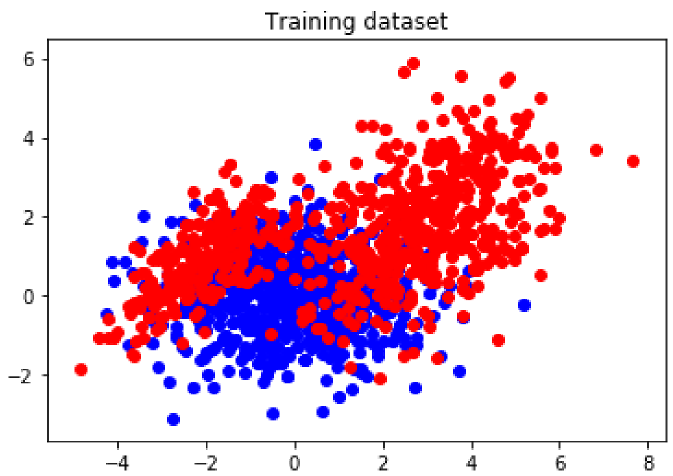


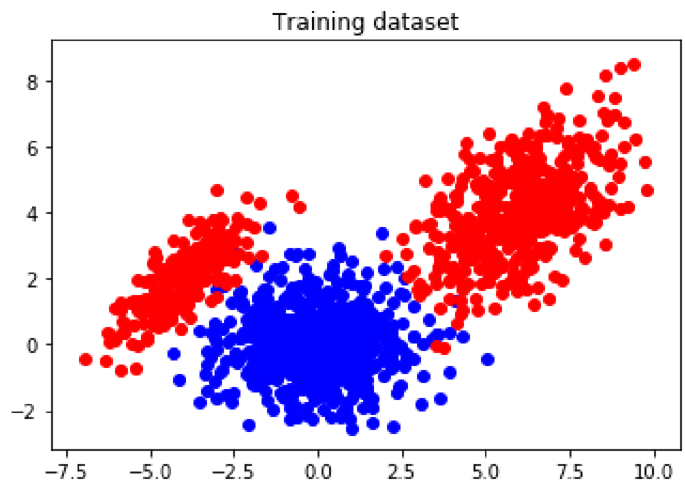


I chose the number of epochs based on how fast the loss – and accuracy propagation evolved. Ended up with 200 epochs. After the training was complete, I got validation accuracy = 0.865 with one hidden layer and 0.845 with two hidden layers. I was expecting the validation accuracy to be greater using two hidden layers, but I think the reason why is due to a slight overfitting using only one layer. When we used the MP rule in homework 1, we got the benchmark of 0.85. So this should be the highest accuracy to aim for without too much overfitting.

When we doubled the mean on the datasets, we got a validation accuracy = 0.995 with one hidden layer as well as with two hidden layers. This makes sense because in these cases, the classes are easier to distinguish.

See figure for dataset with regular mean and double mean on class 1.

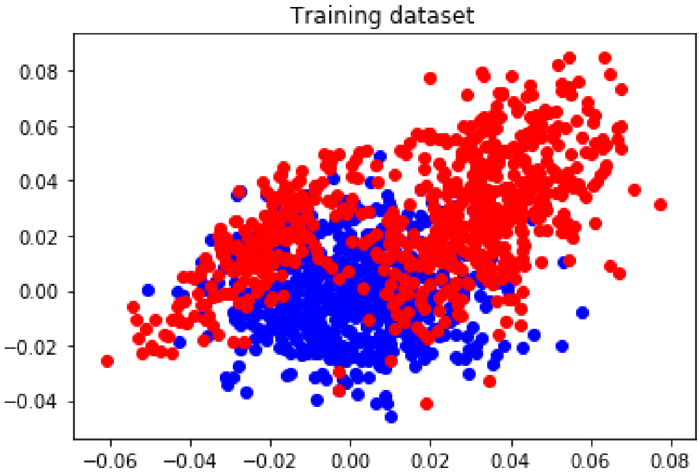


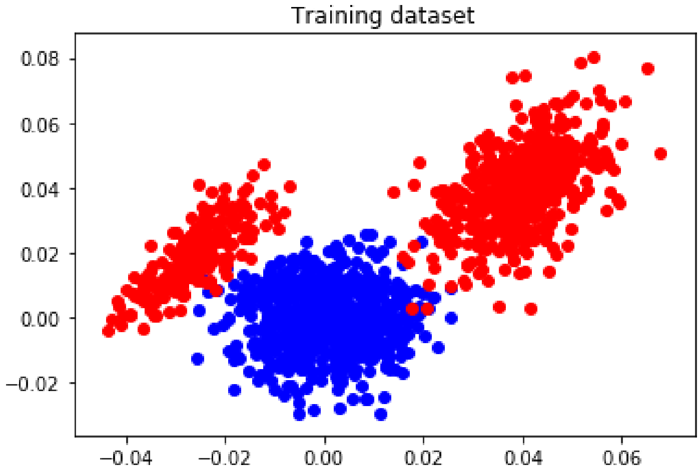


The final classification accuracy on the test set ended up as 0.83 with one layer and 0.82 with two layers.

Problem 3

I added a function that normalized the data to zero mean and unit variance. (Dataset can be seen in figure). For some reason the test accuracy as well as the validation accuracy ended up as 0.5 for both one and two hidden layers. Initially I thought that normalization should be beneficial for the accuracy, but maybe because the neural network isn't trained for this we get such poor performance.





As far as weight initialization goes, all the numbers presented in this report used normalized Gaussian for initializing the weights. In the case where the neural network is big with a lot of dimensions, correct initialization of the weights is crucial for adequate performance. After watching the Stanford lectures on the topic, they convinced the students that Xavier initialization. But since this network was so small, different initializations was not tested.

Problem 4

Comparing the data from the neural network to the methods used in homework 1, the differences are not tremendous. Using the MAP-rule, we achieved an accuracy of 0.85, which is pretty much the same as with the neural network. As stated before, our goal during testing was to get the validation accuracy as close as possible to the MAP-rule accuracy. But as seen in the previous homework, the neural network performed better than the logistic regression models.

```python
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):
```

```python
    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()
```

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

class Dataset_raw:
    def __init__(self,N,x,y,t):
        self.N = N
        self.x = x
        self.y = y
        self.t = t

def readDataset(path):
    d1 = []
    d2 = []
    d3 = []
    with open(path,'r') as file:
        for line in file:
            data = line.split()
            d1.append(float(data[0]))
            d2.append(float(data[1]))
            d3.append(float(data[2]))

    dataset = Dataset_raw(len(d1),[],[],[])
    for i in range(len(d1)):
        dataset.x.append(d1[i])
        dataset.y.append(d2[i])
        dataset.t.append(d3[i])
    return dataset

def next_batch(dataset, batch_size):
    batch_x = np.array([[0 for i in range(2)] for j in range(batch_size)])
    batch_y = np.array([[0 for i in  range(2)]for j in range(batch_size)])

    # Shuffle dataset
    k = np.random.permutation(dataset.N)
    for i in range(batch_size):
        batch_x[i][0] = dataset.x[k[i]]
        batch_x[i][1] = dataset.y[k[i]]
        label = dataset.t[k[i]]
        if label == 0:
            batch_y[i][0] = 1
            batch_y[i][1] = 0
        else:
            batch_y[i][0] = 0
            batch_y[i][1] = 1

    return batch_x, batch_y

def plot_dataset(dataset):
    for i in range(dataset.N):
        if dataset.t[i] == 1.0:
            plt.scatter(dataset.x[i], dataset.y[i], c='r', label='Class 0')
        else:
            plt.scatter(dataset.x[i], dataset.y[i], c='b', label='Class 1')

    plt.title('Training dataset')


def write_data(loss, acc, epoch_list, filename):
    with open('Dataset/loss_' + filename,'w') as f:
        for s in loss:
```

```
            f.write(str(s) + '\n')
    with open('Dataset/acc_' + filename,'w') as f:
        for s in acc:
            f.write(str(s) + '\n')
    with open('Dataset/epoch_list_' + filename,'w') as f:
        for s in epoch_list:
            f.write(str(s) + '\n')

def read_data(filename):
    with open('Dataset/acc_' + filename,'r') as f:
        acc = [float(line.rstrip('\n')) for line in f]

    with open('Dataset/loss_' + filename,'r') as f:
        loss = [float(line.rstrip('\n')) for line in f]

    with open('Dataset/epoch_list_' + filename,'r') as f:
        epoch_list = [float(line.rstrip('\n')) for line in f]

    return acc, loss, epoch_list


def plot_data():
    # Read files
    acc_hidden_1_1mean, loss_hidden_1_1mean, epoch_list_hidden_1_1mean = read_data('hidden_1_1me
    acc_hidden_2_1mean, loss_hidden_2_1mean, epoch_list_hidden_2_1mean = read_data('hidden_2_1me
    acc_hidden_1_2mean, loss_hidden_1_2mean, epoch_list_hidden_1_2mean = read_data('hidden_1_2me
    acc_hidden_2_2mean, loss_hidden_2_2mean, epoch_list_hidden_2_2mean = read_data('hidden_2_2me


    plt.figure(2)
    plt.plot(epoch_list_hidden_1_1mean,loss_hidden_1_1mean, label='1 hidden layer')
    plt.plot(epoch_list_hidden_2_1mean,loss_hidden_2_1mean, label = '2 hidden layers')
    plt.plot(epoch_list_hidden_1_2mean,loss_hidden_1_2mean, label='1 hidden layer, double mean')
    plt.plot(epoch_list_hidden_2_2mean,loss_hidden_2_2mean, label = '2 hidden layers, double mea
    plt.legend()
    plt.xlim([2, epoch_list_hidden_2_2mean[len(epoch_list_hidden_1_1mean)-1]])
    plt.title('Loss propagation on training set')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')

    plt.figure(3)
    plt.plot(epoch_list_hidden_1_1mean, acc_hidden_1_1mean, label='1 hidden layer')
    plt.plot(epoch_list_hidden_2_1mean,acc_hidden_2_1mean, label = '2 hidden layers')
    plt.plot(epoch_list_hidden_1_2mean, acc_hidden_1_2mean, label='1 hidden layer, double mean')
    plt.plot(epoch_list_hidden_2_2mean,acc_hidden_2_2mean, label = '2 hidden layers, double mean
    plt.legend()
    plt.xlim([0, epoch_list_hidden_2_2mean[len(epoch_list_hidden_1_1mean)-1]])
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title('Accuracy propagation on training set')

def normalize_dataset(dataset):
    dataset.x = tf.keras.utils.normalize(x=dataset.x,order=2)
    dataset.x = dataset.x[0]
    dataset.y = tf.keras.utils.normalize(x=dataset.y, order=2)
    dataset.y = dataset.y[0]
    return dataset
```