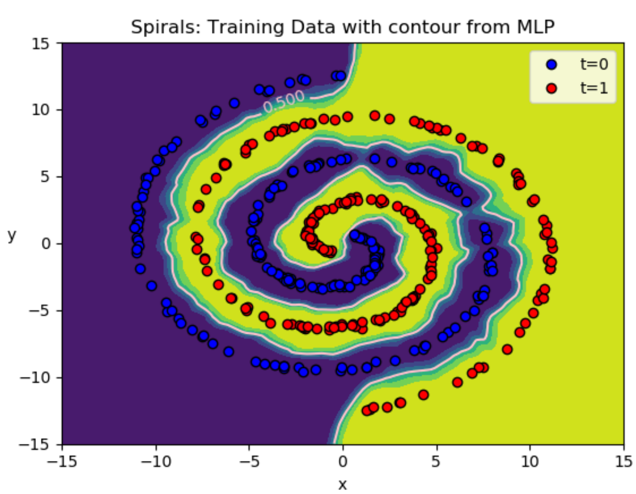
Nithilam Subbaian
9/18/2019
ECE 471

Selected Topics in Machine Learning – Assignment 2

```
#!/bin/python3.6
import numpy as np
import tensorflow as tf
from tqdm import trange
import matplotlib.pyplot as plt

# I credit these resources with good help for this assignment (in addition to tensorflow docs of course!):
# https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/anatomy
# https://www.jessicayung.com/explaining-tensorflow-code-for-a-multilayer-perceptron/
# https://towardsdatascience.com/sigmoid-activation-and-binary-crossentropy-a-less-than-perfect-match-
b801e130e31
# https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/

BATCH_SIZE = 50
NUM_BATCHES = 10000
LEARNING_RATE = 0.1
NUM_SAMP = 200

class Data(object):
    def __init__(self):
        np.random.seed(31415)
        sigma = 0.1

        self.index = np.arange(2*NUM_SAMP)

        # spirals made to replicate given graph
        self.t1 = np.random.uniform(np.pi/4, 4*np.pi, NUM_SAMP)
        self.x1 = np.array([ self.t1*np.sin(self.t1)+ np.random.normal(0, sigma, NUM_SAMP),
                             self.t1*np.cos(self.t1)+ np.random.normal(0, sigma, NUM_SAMP) ]).T

        # rotate the previously generated spiral by 180 degrees to produce second spiral
        self.t2 = np.random.uniform(np.pi/4, 4*np.pi, NUM_SAMP)
        self.x2 = np.array([ self.t2*np.sin(self.t2 + np.pi)+ np.random.normal(0, sigma, NUM_SAMP),
                             self.t2*np.cos(self.t2 + np.pi)+np.random.normal(0, sigma, NUM_SAMP) ]).T

        self.coordinates = np.concatenate((self.x1, self.x2))
        self.labels = np.concatenate((np.zeros(NUM_SAMP), np.ones(NUM_SAMP)))

    def get_batch(self):
        choices = np.random.choice(self.index, size = BATCH_SIZE)
        return self.coordinates[choices,:], self.labels[choices].flatten()


class Model(tf.Module):
    def __init__(self, dimensions_x = 2, dimensions_layer_1 = 40, dimensions_layer_2 = 40):
        self.weights = {
            'w1': tf.Variable(tf.random.normal(shape=[dimensions_x, dimensions_layer_1]), dtype= tf.float32),
            'w2': tf.Variable(tf.random.normal(shape=[dimensions_layer_1, dimensions_layer_2]), dtype=
tf.float32),
            'out': tf.Variable(tf.random.normal(shape=[dimensions_layer_2, 1]), dtype=  tf.float32)
        }
        self.biases = {
            'b1': tf.Variable(tf.random.normal(shape=[dimensions_layer_1]),  dtype= tf.float32),
            'b2': tf.Variable(tf.random.normal(shape=[dimensions_layer_2]),  dtype= tf.float32),
            'out': tf.Variable(tf.zeros(shape=[]), dtype=  tf.float32)
        }

    def f(self, x_coords):
        layer_one = tf.nn.relu6(tf.add( tf.matmul(x_coords, self.weights['w1']), self.biases['b1']))
        layer_two = tf.nn.relu6(tf.add( tf.matmul(layer_one, self.weights['w2']), self.biases['b2'] ))
        output = tf.add( tf.matmul(layer_two, self.weights['out']), self.biases['out'])
        return tf.squeeze(output)

    def loss(self, coords, labs):
        # Sigmoid can be used for interpreting the output of  f : V -> R function as a probability,
        # i.e using it's squashing properties to go R -> [0,1]. You can use the sigmoid to compute the
conditional density.
         # logits are unnormalized probabilities  (i.e they exist in R not [0,1]) and it internally computes
         the sigmoid #for numerical reasons
        loss_p1 = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels = labs, logits =
self.f(coords)) )
```

```
        # standard linear regression problem: y_hat = w^t x + b, loss function = 0.5*(y - w^t x - b)
        # L2 penalty --> modify the loss function to 0.5*(y - w^t x - b) + lambda*norm(w)**2
        #  norm --> the L2 norm
        #  lambda --> hyperparamter  where 0 removes regularization (training is then minimizing loss, risk of
overfitting)
        #l2_loss function -->  output = sum(t ** 2) / 2
        loss_p2 = (.001) * tf.reduce_sum( [tf.nn.l2_loss(var) for var in model.weights.variables] )

        loss = loss_p1 + loss_p2

        return loss


if __name__ == "__main__":

    data = Data()
    model = Model()

    optimizer = tf.optimizers.SGD(learning_rate=LEARNING_RATE) # Stochastic gradient descent and momentum
optimizer

    bar = trange(NUM_BATCHES) # creates bar visual to show progress completion status
    for i in bar:
        with tf.GradientTape() as tape: # GradientTape= Record operations for automatic differentiation.
            coords, labs = data.get_batch()
            loss_fromBatch = model.loss(coords.astype(np.float32), labs.astype(np.float32))
            grads = tape.gradient(loss_fromBatch, model.variables)
            optimizer.apply_gradients(zip(grads, model.variables))

    xAxis = np.linspace(-15, 15, 150)
    yAxis = np.linspace(-15, 15, 150)
    X, Y = np.meshgrid(xAxis, yAxis)
    Z = np.zeros((150,150))

    for i in range(150):
        for j in range(150):
            Zarr = np.array([ X[i,j], Y[i,j] ]).reshape(1,2)
            Zval = model.f(Zarr.astype(np.float32))
            Z[i,j] = tf.nn.sigmoid(Zval)

    plt.figure()

    plt.plot(data.x1[:,0], data.x1[:,1], 'ob', markeredgecolor="black")
    plt.plot(data.x2[:,0], data.x2[:,1], 'or', markeredgecolor="black")

    point_five_line = plt.contour(X, Y, Z, levels = [0.5], colors = "pink")
    plt.clabel(point_five_line, inline = 1, fontsize = 10, colors='pink')
    plt.legend(["t=0", "t=1"])

    plt.contourf(X, Y, Z)

    plt.title("Spirals: Training Data with contour from MLP")
    plt.xlabel('x')
    plt.ylabel('y',rotation=0)
    plt.show()
```



To find the functional form of **f** I did the computations as seen in the method f in class Model. Essentially there are three layers total, in which the coordinates are fed into and a final output is produced. For each layer the output of the previous layer (or the coordinates for the first layer) is multiplied by the weights of that layer and added to the bias of that layer.

Relu6 was a sufficient activation function, and the number of layers used was appropriate for the data we have. If more layers were introduced there would be overfitting.

In the code, a pdf was essentially generated to determine the probability of it being red, and this is what was used to classify the results.

In working on the problem, the lambda was important in determining the quality of the contour produced, it tended to produce a better classification when the lambda was near 0.01.

There was flexibility in the batch size and number of samples, but the number of samples I selected to resemble the sample data.

The number of features chosen for each layer at 40 produced adequate results.

The pink line on the graph demonstrates where the line between equal probability lies.