```
#!/bin/python3.6
#Ostap Voynarovskiy
#CGML HW2
#Sept 19 2018
#Professor Curro

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
from tqdm import tqdm
#from tensorflow.python import debug as tfdbg

BATCH_SIZE = 200
NUM_ITER = 5000# iterations of training

class Data(object):
        def __init__(self):
                #create spirals
                nPoints = 200
                self.index = np.arange(nPoints)
                self.nPoints = nPoints
                self.featx, self.featy,self.lab  = self.gen_spiral(nPoints)

        def gen_spiral(self,nPoints):

                scale = 1
                offset = 1
                sigma = .2

                t = np.linspace(0,3.5*np.pi,num = nPoints)
                noise0 = sigma*np.random.normal(size=nPoints)
                noise1 = sigma*np.random.normal(size=nPoints)
                noise2 = sigma*np.random.normal(size=nPoints)
                noise3 = sigma*np.random.normal(size=nPoints)

                #add normal noise
                theta0 = -t*scale + noise0
                r0 = (t + offset) + noise1
                theta1= -t*scale + np.pi + noise2       #the addition of pi does
 a 180 degree shift
                r1 = (t + offset) + noise3

                #convert from polar to cartesian
                self.x0 = np.cos(theta0)*(r0)
                self.y0 = np.sin(theta0)*(r0)
                cat0 = [0]*nPoints                      # the categories
                self.x1 = np.cos(theta1)*(r1)
                self.y1 = np.sin(theta1)*(r1)
                cat1 = [1]*nPoints                      # the categories
                return np.concatenate((self.x0,self.x1)),np.concatenate((self.y0
,self.y1)), np.concatenate((cat0,cat1))

        def get_batch(self):
                choices = np.random.choice(self.nPoints*2, size=BATCH_SIZE)
                return list(zip(self.featx[choices],self.featy[choices])), self.
lab[choices]


def f(x): #this is where we decide our tunable parameters and create our percept
ron
        m1 = 74 # first layer nodes = my fav 2 numbers
        m2 = 47 # second layer nodes = my fav 2 numbers but swapped
        m3 = 1  # one so that its a single yes or no

        # These are the initializations of the things we will learn including w’
s b’s and

        # Weight matricies should all be aproximately gaussian distribution sinc
```



```
e we care about
        # diversity but wanna give all features similar chances on average.
        w1 = tf.get_variable(’w1’, [2,m1], tf.float32,tf.random_normal_initiali
zer())
        w2 = tf.get_variable(’w2’, [m1, m2], tf.float32,tf.random_normal_initia
lizer())
        w3 = tf.get_variable(’w3’, [m2, m3], tf.float32,tf.random_normal_initia
lizer())

        # start at 0
        b1 = tf.get_variable(’b1’, [1,m1], tf.float32, tf.random_normal_initiali
zer())  #update
        b2 = tf.get_variable(’b2’, [1,m2], tf.float32, tf.random_normal_initiali
zer())
        b3 = tf.get_variable(’b3’, [1,m3], tf.float32, tf.random_normal_initiali
zer())

        #activation functions
        layer1 = tf.nn.elu(tf.matmul(x,w1)+b1)  # Activation function 1
        layer2 = tf.nn.leaky_relu(tf.matmul(layer1,w2)+b2)              # Activa
tion function 2
        layer3 = (tf.matmul(layer2,w3)+b3)                              # produc
e logits for cross entropy loss

                # to give a clear "is this group 0 or 1"

                # so dont put it through a sigmoid now
        '''The decision to use a leaky relu and an elu was carefully considered. When I first
        selected an activation function, I was not picky and used only sigmoids since they are classic.
        When I got everything working, I realized that it took many iteratios to converge. I proceeded
        to test then the hyperbolic tangent, the relu, elu, and leak relu along with different combinations
        of them. I found that the best results with the least training iterations happened with the
        leaky relu and the elu function.'''

        # This will be left out. We are performing binary classification.
        # We will not be modeling something in multiple dim.
        # mu is the x loc of the gaussian so we use a uniform distribution
        #mu =    tf.get_variable(’mu’, [NUM_PHIS, 1], tf.float32, tf.random_unif
orm_initializer())
        # the sigmas are gonna be approx
        #sig =   tf.get_variable(’sig’, [NUM_PHIS, 1], tf.float32,tf.random_norm
al_initializer())
        # phi = tf.exp(-tf.pow((x-mu)/sig, 2))
        return layer3   # tf.squeeze(layer3)  This is cux the losses.sigmoid_cro
ss_entropy wants
                                                #[batch size, num_classes] im guessing n
um classes is 1

features  = tf.placeholder(tf.float32, [None,2])        # Should get batch size
by 2 array of labels
labels = tf.placeholder(tf.float32, [None])             # Should get batch size
by 1 array ...

                        # we want a binary classification
labels_predicted = f(features)

# which w are we taking the norm of there are 3?
l = 0.002; # l is lambda

loss = tf.losses.sigmoid_cross_entropy(tf.stack([labels, 1-labels], 1),tf.squeez
e(tf.stack([labels_predicted, -labels_predicted], 1))) \
        + l*tf.reduce_sum([tf.nn.l2_loss(tV) for tV in tf.trainable_variables
()])
#loss  = tf.reduce_mean(tf.pow(y-y_hat, 2)/2) #loss funtion = cross entropy + L2
 norm
optim = tf.train.GradientDescentOptimizer(learning_rate=.1).minimize(loss) #this
 does gradient descent
#optim??? = tf.train.momentum #cuz we read about it in the reading
init  = tf.global_variables_initializer()
```

```
sess = tf.Session()
#sess = tfdbg.LocalCLIDebugWrapperSession(sess)
sess.run(init)

data = Data()
for _ in tqdm(range(0, NUM_ITER)):
    x_np, labels_np = data.get_batch()
    loss_np, yhats, _ = sess.run([loss, labels_predicted, optim], feed_dict={fea
tures: x_np, labels: labels_np})
    #print(loss_np)

#rslt=sess.run(tf.stack(labels_predicted), feed_dict={features: list(zip(data.fe
atx,data.featy))})
fig1= plt.figure(1)

xc,yc = np.linspace(-15,15,500),np.linspace(-15,15,500)
xv,yv = np.meshgrid(xc,yc)

feat = np.array(list(zip(xv.flatten(),yv.flatten())))
res  = sess.run(labels_predicted, feed_dict={features: feat })  # lt = sess.run(
what_you_want,    feed_dict={features: what_you_have})
cont = sess.run(tf.sigmoid(res))
plt.contourf(xv,yv,cont.reshape((500,500)),[0,.5,1])
plt.scatter(data.x0,data.y0)
plt.scatter(data.x1,data.y1)

plt.xlabel('x')
plt.ylabel('y')
plt.title("3 Layer Perceptron ")
plt.axis('equal') #make it so that it isnt warped
plt.show()
```