binary_classification.py


```
#!/bin/python

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

#font = {'family' : 'Adobe Caslon Pro',
#        'size'   : 10}

#matplotlib.rc('font', **font)

def model_variable(shape, minval, maxval):
        variable = tf.Variable(tf.random_uniform(shape=shape, minval=minval, maxval=maxval))
        tf.add_to_collection('model_variables', variable)
        tf.add_to_collection('l2', tf.reduce_sum(tf.pow(variable,2)))
        return variable

class Model():
    def __init__(self, sess, data, nEpochs, learning_rate, lambduh):
        self.sess = sess
        self.data = data
        self.nEpochs = nEpochs
        self.learning_rate = learning_rate
        self.lambduh = lambduh
        self.build_model()

    def build_model(self):
        self.x = tf.placeholder(tf.float32, shape=[1,2])
        self.y = tf.placeholder(tf.float32, shape=[1,1])

        w_hidden_in = model_variable([2,16], -7/2, 7/2)
        w_hidden_out = model_variable([16,1], -0.001/2, 0.001/2)

        self.layer_in = tf.matmul(self.x, w_hidden_in)
        self.layer_in = tf.sin(self.layer_in)

        #Loss function already performs sigmoid
        self.layer_loss = tf.matmul(self.layer_in, w_hidden_out)
        self.layer_out = tf.sigmoid(self.layer_loss)

        self.logloss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=self.layer_loss, targets=self.y))
        self.l2_penalty = tf.reduce_sum(tf.get_collection('l2'))
        self.loss = self.logloss + self.lambduh*self.l2_penalty

    def train_init(self):
        model_variables = tf.get_collection('model_variables')
        self.optim = (
             tf.train.GradientDescentOptimizer(learning_rate = self.learning_rate)
                .minimize(self.loss, var_list=model_variables)
            )
        self.sess.run(tf.initialize_all_variables())
    def train_iter(self, x, y):
        loss, logloss, l2_penalty, _ = self.sess.run([self.loss, self.logloss, self.l2_penalty, self.optim], feed_dict={self.x : x, self.y : y})
        print('loss: {}, logloss: {}, l2_penalty {}'.format(loss, logloss, l2_penalty))
    def train(self):
        for _ in range(self.nEpochs):
            for x, y in self.data():
                self.train_iter(x, y)

    def infer(self, x):
        return self.sess.run(self.layer_out, feed_dict={self.x : x})


def data():
    sigma = 0.1
    np.random.seed(31415)
    iset = np.linspace(0,96,200)
    for i, ii in enumerate(iset):
```

binary_classification.py


```
        theta = (ii/16)*np.pi
        r = ((6.5*(104-ii))/104)+np.random.normal()*sigma
        for j in range(0,2):
            if j == 0:
                x = [r*np.cos(theta),r*np.sin(theta)]
                y = 1
            else:
                x = [(-1)*r*np.cos(theta), (-1)*r*np.sin(theta)]
                y = 0
            x = np.reshape(x, (1,2))
            y = np.reshape(y, (1,1))
            yield x,y

sess = tf.Session()
model = Model(sess, data, nEpochs=50, learning_rate=6e-3, lambduh=0)
model.train_init()
model.train()

N = 100

examples, targets= zip(*list(data()))

spiralx_1 = []
spiraly_1 = []
spiralx_2 = []
spiraly_2 = []
for a,b in zip(examples,targets):
    if (b == 1):
        spiralx_1.append(a[0][0])
        spiraly_1.append(a[0][1])
    else:
        spiralx_2.append(a[0][0])
        spiraly_2.append(a[0][1])

xGrid = np.linspace(-7.5, 7.5, num=N)
yGrid = np.linspace(-7.5, 7.5, num=N)
p = np.zeros((N,N))
for i in range(N):
    for j in range(N):
        p[i,j] = model.infer(np.reshape((xGrid[j],yGrid[i]), (1,2)))

X, Y = np.meshgrid(xGrid, yGrid)
plt.contourf(X,Y,p,20)
plt.plot(np.array(spiralx_1),np.array(spiraly_1),'r.',np.array(spiralx_2),np.array(spiraly_2),'.')

plt.xlabel('x')
plt.ylabel('y')
plt.title('Spiral Dataset - Yash Sharma')

plt.tight_layout()
plt.savefig('plot.pdf', format='pdf', bbox_inches='tight')
```