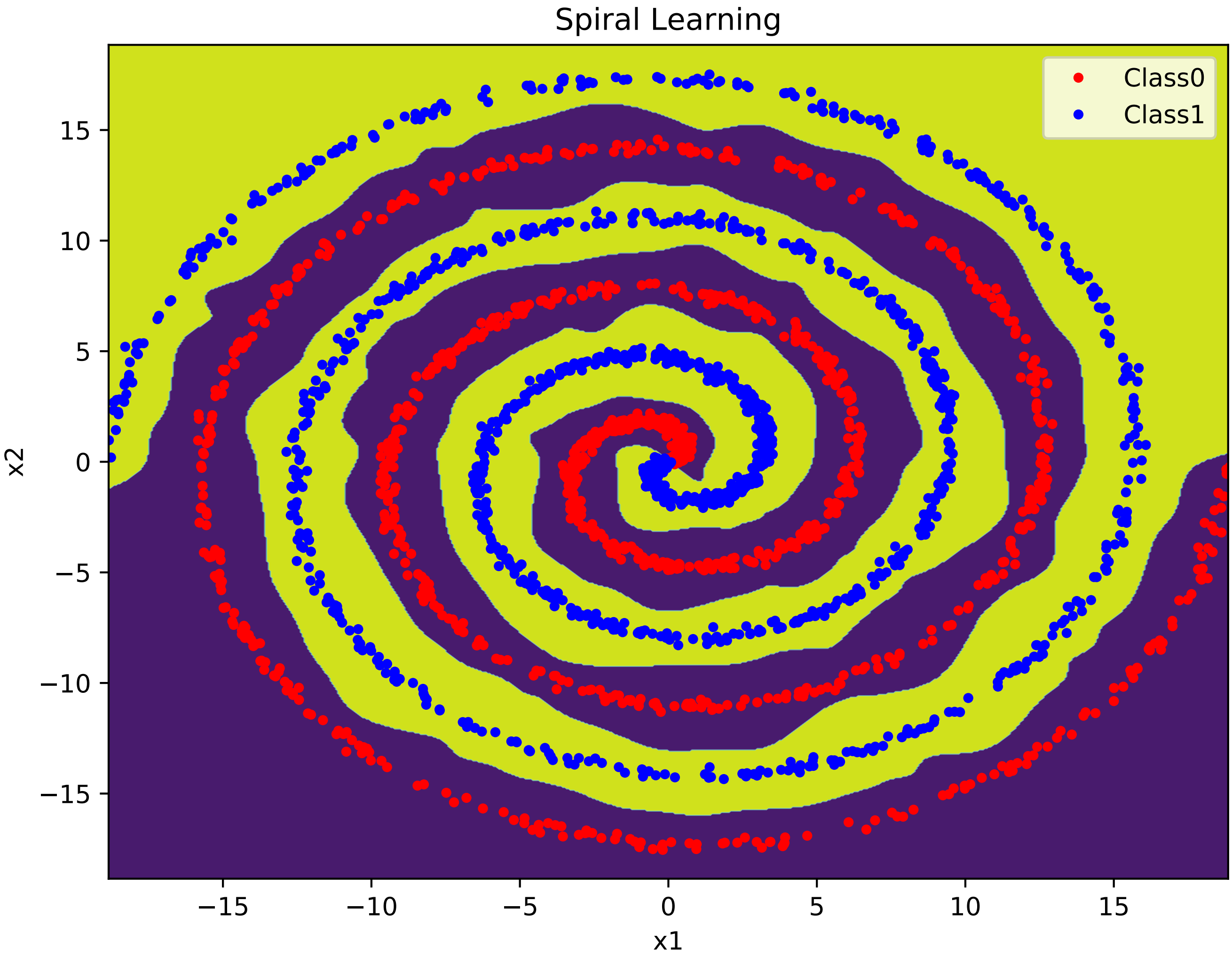
```
# DongKyu Kim
# ECE 471 CGML Assignment 2
# Professor Curro

# library imports
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tqdm import tqdm

# hyper parameters
iteration = 50000
learning_rate = 0.4
lambda_ = 0.0001 # regularization constant
display = 1000
sigma_noise = 0.15
samples = 2500
layers = [2,40,50,40,1] # This is my neural network set up
spiral_difficulty = 3 # This controls how many times the spiral goes around
sd = spiral_difficulty*2 # I don’t have to write spiral_difficulty*2 everytime.

’’’
Layers = [2,30,1]: After 100000 iterations at lr = 0.5, lambda = 0.00001, it has 0.089 loss for spiral_difficulty =2
This layer was too slow to converge, and wasn’t too correct. So I changed my layer to [2,30,3
0,1].
I increased the spiral difficulty from 2 to 3. Then my second layer option wasn’
t sufficient enough.
So I increased both hidden layer and number of nodes, and now it learns the spiral_difficulty = 3 well.
I actually had a minor difficulty of having too few samples, so my boundaries weren’t as pretty as I wante
d, so I increased the sample size as well.
’’’

# Data Generation
class Data(object):
    def __init__(self,N):
        np.random.seed(31415)
        temp = np.random.uniform(0,sd*np.pi,size=(N,1))
        self.label = np.random.randint(2,size=(N,1))
        self.x1 = temp*np.cos(temp+self.label*np.pi)+sigma_noise*np.random.normal(size=(N,1))
        self.x2 = temp*np.sin(temp+self.label*np.pi)+sigma_noise*np.random.normal(size=(N,1))

# Model
class My_Model(object):
    def __init__(self,sess,data,layers,learning_rate,iteration,lambda_):
        self.sess = sess
        self.data = data
        self.layers = layers
        self.learning_rate = learning_rate
        self.iteration = iteration
        self.lambda_ = lambda_
        self.build_model()

    def build_model(self):
        self.x = tf.placeholder(tf.float32,[None,2])
        self.y = tf.placeholder(tf.float32,[None,1])
        self.w = {}
        self.b = {}
        for i in range(0,len(self.layers)−1):
            self.w[i] = tf.get_variable(’w’+str(i),[self.layers[i],self.layers[i+1]],tf.float32,tf.random_normal_initializer())
            tf.add_to_collection(’l2_norm’, tf.reduce_sum(tf.square(self.w[i])))
            self.b[i] = tf.get_variable(’b’+str(i),[self.layers[i+1],1],tf.float32,tf.random_normal_initializer())
            tf.add_to_collection(’l2_norm’, tf.reduce_sum(tf.square(self.b[i])))
        # first layer
        self.y_hat = tf.sigmoid(tf.add(tf.matmul(self.x,self.w[0]),tf.transpose(self.b[0])))
        # subsequent layers
        for i in range(1,len(self.layers)−1):
            self.y_temp = tf.add(tf.matmul(self.y_hat,self.w[i]),tf.transpose(self.b[i]))
            self.y_hat = tf.sigmoid(self.y_temp)
        self.costs = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=self.y_temp,labels=self.y))
        self.l2 = tf.reduce_sum(tf.get_collection('l2'))
        self.loss = self.costs+self.l2*self.lambda_

    def train_init(self):
        self.optim = tf.train.GradientDescentOptimizer(self.learning_rate).minimize(self.loss)
        self.init = tf.global_variables_initializer()
        self.sess.run(self.init)

    def train(self):
        for k in tqdm(range(0,self.iteration)):
            w,loss,optim = self.sess.run([self.w,self.loss,self.optim], feed_dict={self.x: np.concatenate((self.data.x1,self.dat
a.x2),axis = 1),self.y:self.data.label})
            if k % display == 0:
                print('The Current Loss at '+str(k)+'th iteration is '+str(loss))

    def predict(self,x_new):
        temp = self.sess.run(self.y_hat,feed_dict={self.x:x_new})
        return temp > 0.5

# Session Run
sess = tf.Session()
data =Data(samples)
model = My_Model(sess,data,layers,learning_rate,iteration,lambda_)
model.train_init()
model.train()

# Predicting
N_plot = sd*100
xGrid = np.linspace(−sd*np.pi,sd*np.pi,N_plot,dtype=np.float32)
yGrid = xGrid
x_plot,y_plot = np.meshgrid(xGrid,yGrid)
XX = np.reshape(x_plot,(−1,1))
YY = np.reshape(y_plot,(−1,1))
test = np.concatenate((XX,YY),1)
LGrid = model.predict(test)

# Plotting
plt.figure(figsize=(8,6))
plt.contourf(x_plot,y_plot,np.reshape(LGrid,(N_plot,N_plot)))
plt.plot(data.x1[data.label==0],data.x2[data.label==0],'.r',label='Class0')
plt.plot(data.x1[data.label==1],data.x2[data.label==1],'.b',label='Class1')
txt="sigma noise = 0.15, Each background color represents each class"
plt.figtext(0.5, 0.01, txt, wrap=True, horizontalalignment='center', fontsize=12)
plt.xlabel('x1')
plt.ylabel('x2')
plt.axis([−sd*np.pi,sd*np.pi,−sd*np.pi,sd*np.pi])
plt.title('Spiral Learning')
plt.legend()
plt.show()
```



sigma noise = 0.15, Each background color represents each class